

---

# **Blender User Manual**

*Release 2.75*

**Blender Community**

July 22, 2015

<b>1</b>	<b>Getting Started</b>	<b>2</b>
1.1	Getting Started . . . . .	2
<b>2</b>	<b>Sections</b>	<b>113</b>
2.1	Editors . . . . .	113
2.2	Data System . . . . .	159
2.3	Modeling . . . . .	175
2.4	Modifiers . . . . .	450
2.5	Rigging . . . . .	548
2.6	Animation . . . . .	648
2.7	Physics . . . . .	743
2.8	Render . . . . .	854
2.9	Composite Nodes . . . . .	1277
2.10	Motion Tracking . . . . .	1365
2.11	Grease Pencil . . . . .	1376
2.12	Game Engine . . . . .	1387
2.13	Extensions . . . . .	1474
2.14	Preferences . . . . .	1494
2.15	Advanced . . . . .	1510
2.16	Troubleshooting . . . . .	1514
2.17	Glossary . . . . .	1517
2.18	Get Involved . . . . .	1523

Welcome to the Blender Manual!

## GETTING STARTED

### 1.1 Getting Started

#### 1.1.1 About Blender

##### Introduction



Fig. 1.1: Blender 2.5 with a Big Buck Bunny scene open

Welcome to Blender, the free and open source 3D animation suite.

Blender can be used to create 3D visualizations such as still images, video and real-time interactive video games.

Blender is well suited to individuals and small studios who benefit from its unified pipeline and responsive development process.

It is cross-platform and runs equally well on Linux, Windows and Macintosh computers with a small memory and disk footprint. Its interface uses OpenGL to provide a consistent experience across all supported hardware and platforms.

## Key Features

- Blender is a fully integrated 3D content creation suite, offering a broad range of essential tools, including [modeling](#), [UV mapping](#), [texturing](#), [rigging and skinning](#), [animation](#), many types of [simulations](#) (fluid, rigid bodies, etc), [scripting](#), [rendering](#), [compositing](#), [VFX](#), and [game creation](#).
- Cross platform, with an OpenGL GUI that is uniform on all major platforms (and customizable with Python scripts).
- High quality 3D architecture enabling fast and efficient creation work-flow.
- Excellent community support from [forums](#) and [IRC](#).
- Small executable size, optionally portable.

You can download the latest version of Blender [here](#).



Fig. 1.2: A rendered image being post-processed

Blender makes it possible to perform a wide range of tasks, and it may seem daunting when first trying to grasp the basics. However, with a bit of motivation and the right learning material, it is possible to familiarize yourself with Blender after a few hours of practice.

This manual is a good start, though it serves more as a reference. There are also many online video tutorials from specialized websites, and several books and training DVDs available in the [Blender Store](#) and on the [Blender Cloud](#).

Despite everything Blender can do, it remains a tool. Great artists do not create masterpieces by pressing buttons or manipulating brushes, but by learning and practicing subjects such as human anatomy, composition, lighting, animation principles, etc.

3D content creation software such as Blender have the added technical complexity and jargon associated with the underlying technologies. Terms like UV maps, materials, shaders, meshes, and subsurf are the mediums of the digital artist, and understanding them, even broadly, will help you to use Blender to its best.

So keep reading this manual, learn the great tool that Blender is, keep your mind open to other artistic and technological areas, and you too can become a great artist.

## Blender's History

In 1988 Ton Roosendaal co-founded the Dutch animation studio NeoGeo. NeoGeo quickly became the largest 3D animation studio in the Netherlands and one of the leading animation houses in Europe. NeoGeo created award-winning productions (European Corporate Video Awards 1993 and 1995) for large corporate clients such as multi-national electronics company Philips. Within NeoGeo Ton was responsible for both art direction and internal software development. After careful deliberation Ton decided that the current in-house 3D toolset for NeoGeo was too old and cumbersome to maintain, and needed to be rewritten from scratch. In 1995 this rewrite began and was destined to become the 3D software creation we all know as Blender. As NeoGeo continued to refine and improve Blender it became apparent to Ton that Blender could be used as a tool for other artists outside of NeoGeo.

In 1998, Ton decided to found a new company called Not a Number (NaN) as a spin-off of NeoGeo to further market and develop Blender. At the core of NaN was a desire to create and distribute a compact, cross platform 3D application for free. At the time this was a revolutionary concept as most commercial 3D applications cost thousands of dollars. NaN hoped to bring professional level 3D modeling and animation tools within the reach of the general computing public. NaN's business model involved providing commercial products and services around Blender. In 1999 NaN attended its first SIGGRAPH conference in an effort to more widely promote Blender. Blender's first SIGGRAPH convention was a huge success and gathered a tremendous amount of interest from both the press and attendees. Blender was a hit and its huge potential confirmed!

Following the success of the SIGGRAPH conference in early 2000, NaN secured financing of €4.5M from venture capitalists. This large inflow of cash enabled NaN to rapidly expand its operations. Soon NaN boasted as many as fifty employees working around the world trying to improve and promote Blender. In the summer of 2000, Blender 2.0 was released. This version of Blender added the integration of a game engine to the 3D application. By the end of 2000, the number of users registered on the NaN website surpassed 250,000.

Unfortunately, NaN's ambitions and opportunities didn't match the company's capabilities and the market realities of the time. This over-extension resulted in restarting NaN with new investor funding and a smaller company in April 2001. Six months later NaN's first commercial software product, Blender Publisher was launched. This product was targeted at the emerging market of interactive web-based 3D media. Due to disappointing sales and the ongoing difficult economic climate, the new investors decided to shut down all NaN operations. The shutdown also included discontinuing the development of Blender. Although there were clearly shortcomings in the then current version of Blender, such as a complex internal software architecture, unfinished features and a non-standard way of providing the GUI, the enthusiastic support from the user community and customers who had purchased Blender Publisher in the past meant that Ton couldn't justify leaving Blender to fade into insignificance. Since restarting a company with a sufficiently large team of developers wasn't feasible, Ton Roosendaal founded the non-profit organization Blender Foundation in March 2002.

The Blender Foundation's primary goal was to find a way to continue developing and promoting Blender as a community-based [open source](#) project. In July 2002, Ton managed to get the NaN investors to agree to a unique Blender Foundation plan to attempt to release Blender as open source. The "Free Blender" campaign sought to raise €100,000 so that the Foundation could buy the rights to the Blender source code and intellectual property rights from the NaN investors and subsequently release Blender to the open source community. With an enthusiastic group of volunteers, among them several ex-NaN employees, a fund raising campaign was launched to "Free Blender". To everyone's surprise and delight the campaign reached the €100,000 goal in only seven short weeks. On Sunday October 13, 2002, Blender was released to the world under the terms of the [GNU GPL](#). Blender development continues to this day driven by a team of dedicated volunteers from around the world led by Blender's original creator, Ton Roosendaal.

---

## Video: From Blender 1.60 to 2.50

### Version/Revision Milestones

#### The start!

- 1.00 - January 1994: Blender [in development](#) at animation studio NeoGeo.
- 1.23 - January 1998: SGI version published on the web, IrisGL.
- 1.30 - April 1998: Linux and FreeBSD version, port to OpenGL and X11.
- 1.3x - June 1998: NaN founded.
- 1.4x - September 1998: Sun and Linux Alpha version released.
- 1.50 - November 1998: First Manual published.
- 1.60 - April 1999: C-key (new features behind a lock, \$95), Windows version released.
- 1.6x - June 1999: BeOS and PPC version released.
- 1.80 - June 2000: End of C-key, Blender full freeware again.
- 2.00 - August 2000: Interactive 3D and real-time engine.
- 2.10 - December 2000: New engine, physics, and Python.
- 2.20 - August 2001: Character animation system.
- 2.21 - October 2001: Blender Publisher launch.
- 2.2x - December 2001: Mac OSX version.

#### Blender goes Open Source

- **13 October 2002: Blender goes Open Source, 1st Blender Conference.**
- 2.25 - October 2002: [Blender Publisher](#) becomes freely available.
- Tuhopuu1 - Oct 2002: The experimental tree of Blender is created, a coder's playground.
- 2.26 - February 2003: The first true open source Blender release.
- 2.27 - May 2003: The second open source Blender release.
- 2.28x - July 2003: First of the 2.28x series.
- 2.30 - October 2003: Preview release of the 2.3x UI makeover presented at the 2nd Blender Conference.
- 2.31 - December 2003: Upgrade to stable 2.3x UI project.
- 2.32 - January 2004: Major overhaul of internal rendering capabilities.
- 2.33 - April 2004: Game Engine returns, ambient occlusion, new procedural textures.
- 2.34 - August 2004: Particle interactions, LSCM UV mapping, functional YafRay integration, weighted creases in subdivision surfaces, ramp shaders, full OSA, and many many more.
- 2.35 - November 2004: Another version full of improvements: object hooks, curve deforms and curve tapers, particle duplicators and much more.
- 2.36 - December 2004: A stabilization version, much work behind the scene, normal and displacement mapping improvements.
- 2.37 - June 2005: Transformation tools and widgets, softbodies, force fields, deflections, incremental subdivision surfaces, transparent shadows, and multi-threaded rendering.

- 2.40 - December 2005: Full rework of armature system, shape keys, fur with particles, fluids and rigid bodies.
- 2.41 - January 2006: Lots of fixes, and some game engine features.
- 2.42 - July 2006: The nodes release, array modifier, vector blur, new physics engine, rendering, lip sync, and many other features. This was the release following [Project Orange](#).
- 2.43 - February 2007: Multi-resolution meshes, multi-layer UV textures, multi-layer images and multi-pass rendering and baking, sculpting, retopology, multiple additional matte, distort and filter nodes, modeling and animation improvements, better painting with multiple brushes, fluid particles, proxy objects, sequencer rewrite, and post-production UV texturing.
- 2.44 - May 2007: The big news, in addition to two new modifiers and re-awakening the 64-bit OS support, was the addition of subsurface scattering, which simulates light scattering beneath the surface of organic and soft objects.
- 2.45 - September 2007: Serious bug fixes, with some performance issues addressed.
- 2.46 - May 2008: The Peach release was the result of a huge effort of over 70 developers providing enhancements to provide hair and fur, a new particle system, enhanced image browsing, cloth, a seamless and non-intrusive physics cache, rendering improvements in reflections, AO, and render baking, a mesh deform modifier for muscles and such, better animation support via armature tools and drawing, skinning, constraints and a colorful Action Editor, and much more. It was the release following [Project Peach](#).
- 2.47 - August 2008: Bugfix release.
- 2.48 - October 2008: The Apricot release, cool GLSL shaders, lights and GE improvements, snap, sky simulator, shrinkwrap modifier, and Python editing improvements. This was the release following [Project Apricot](#).
- 2.49 - June 2009: Node-based textures, armature sketching (called Etch-a-Ton), boolean mesh operation improvements, JPEG2000 support, projection painting for direct transfer of images to models, and a significant Python script catalogue. GE enhancements included video textures, where you can play movies in-game, upgrades to the Bullet physics engine, dome (fish-eye) rendering, and more API GE calls made available.

### Blender 2.5 - The Recode!

- 2.5x - From 2009 to August 2011: This series released four [pre-version](#) (from Alpha 0 in November 2009 to Beta in July 2010) and three stable versions (from 2.57 - April 2011 - to 2.59 - August 2011). It is one of the most important development projects, with a total refactor of the software with new functions, redesign of the internal window manager and event/tool/data handling system, and new Python API. The final version of this project was Blender 2.59 in August 2011.
- 2.60 - October 2011: Internationalization of the UI, improvements in animation system and the GE, vertex weight groups modifiers, 3D audio and video, bug fixes, and the UI internationalization.
- 2.61 - December 2011: The Cycles renderer was added in trunk, the camera tracker was added, dynamic paint for modifying textures with mesh contact/approximation, the Ocean Sim modifier to simulate ocean and foam, new add-ons, bug fixes, and more extensions added for the Python API.
- 2.62 - February 2012: The [Carve](#) library was added to improve boolean operations, support for object tracking was added, the Remesh modifier was added, many improvements in the GE, matrices and vectors in the Python API were improved, new add-ons, and many bug fixes.
- 2.63 - April 2012: Bmesh was merged to trunk with full support for n-sided polygons, sculpt hiding, a panoramic camera for Cycles, mirror ball environment textures and float precision textures, render layer mask layers, ambient occlusion and viewport display of background images and render layers, new import and export add-ons were added, and 150 bug fixes.
- 2.64 - October 2012: Mask editor, improved motion tracker, OpenColorIO, Cycles improvements, sequencer improvements, better mesh tools (Inset and Bevel were improved), new keying nodes, sculpt masking, Collada improvements, new skin modifier, new compositing nodes backend, and many bugs were fixed.
- 2.65 - December 2012: Fire and smoke improvements, anisotropic shader for Cycles, modifier improvements, bevel tool now includes rounding, new add-ons, and over 200 bug fixes.

- 2.66 - February 2013: Dynamic topology, rigid body simulation, improvements in UI and usability (including retina display support), Cycles now supports hair, the bevel tool now supports individual vertex bevelling, new [Mesh Cache](#) modifier and the new [UV Warp](#) modifier, new SPH particle fluid solver. More than 250 bug fixes.
- 2.67 - May 2013: Freestyle was added, paint system improvements, subsurface scattering for Cycles, Ceres library in the motion tracker, new custom python nodes, new mesh modeling tools, better support for UTF8 text and improvements in text editors, new add-ons for 3D printing, over 260 bug fixes.
- 2.68 - July 2013: New and improved modeling tools, three new Cycles nodes, big improvements in the motion tracker, Python scripts and drivers are disabled by default when loading files for security reasons, and over 280 bug fixes.
- 2.69 - October 2013: Even more modeling tools, Cycles improved in many areas, plane tracking is added to the motion tracker, better support for FBX import/export, and over 270 bugs fixed.
- 2.70 - March 2014: Cycles gets basic volumetric support on the CPU, more improvements to the motion tracker, two new modeling modifiers, some UI consistency improvements, and more than 560 bug fixes.
- 2.71 - June 2014: Deformation motion blur and fire/smoke support is added to Cycles, UI popups are now draggable, performance optimizations for sculpting mode, new interpolation types for animation, many improvements to the GE, and over 400 bug fixes.
- 2.72 - October 2014: Cycles gets volume and SSS support on the GPU, pie menus are added and tooltips greatly improved, the intersection modeling tool is added, new sun beam node for the compositor, Freestyle now works with Cycles, texture painting workflow is improved, and more than 220 bug fixes.

## About Free Software and the GPL



When one hears about “free software”, the first thing that comes to mind might be “no cost”. While this is typically true, the term “free software” as used by the Free Software Foundation (originators of the GNU Project and creators of the GNU General Public License) is intended to mean “free as in freedom” rather than the “no cost” sense (which is usually referred to as “free as in free beer” or *gratis*). Free software in this sense is software which you are free to use, copy, modify, redistribute, with no limit. Contrast this with the licensing of most commercial software packages, where you are allowed to load the software on a single computer, are allowed to make no copies, and never see the source code. Free software allows incredible freedom to the end user. Since the source code is universally available, there are also many more chances for bugs to be caught and fixed.

When a program is licensed under the GNU General Public License (the GPL):

- You have the right to use the program for any purpose.

- You have the right to modify the program, and have access to the source codes.
- You have the right to copy and distribute the program.
- You have the right to improve the program, and release your own versions.

In return for these rights, you have some responsibilities if you distribute a GPL'd program, responsibilities that are designed to protect your freedoms and the freedoms of others:

- You must provide a copy of the GPL with the program, so that recipients are aware of their rights under the license.
- You must include the source code or make the source code freely available.
- If you modify the code and distribute the modified version, you must license your modifications available under the GPL (or a compatible license).
- You may not restrict the licensing of the program beyond the terms of the GPL. (you may not turn a GPL'd program into a proprietary product.)

For more on the GPL, check the its page on the [GNU Project web site](#).

---

**Note:** The GPL only applies to the Blender application and **not** the artwork you create with it; for more info see the [Blender License](#).

---

## The Blender Community

Being freely available from the start, even while closed source, helped considerably in Blender's adoption. A large, stable, and active community of users has gathered around Blender since 1998. The community showed its support for Blender in 2002 when they helped raise €100,000 in seven weeks to enable Blender to go Open Source under the [GNU GPL](#).

### Who uses Blender?

Blender has a wide variety of tools making it suitable for almost any sort of media production. People and studios around the world use it for hobby projects, commercials, feature films, games and other interactive applications like kiosks, games and scientific research.

Check out the [User Stories](#) page on the Blender website for more examples.

### Independent Sites

There are [several independent websites](#) such as forums, blogs, news and tutorial sites dedicated to Blender.

One of the largest community forums is [Blender Artists](#), where Blender users gather to show off their creations, get feedback, ask and offer help and in general discuss Blender.

### Support

Blender's community is one of its greatest features, so apart from this user manual there are many different ways to get support from other users, such as [IRC](#) and [Stack Exchange](#).

There are also more official sources of support, such as [Certified Trainers](#) and the [Blender Cloud](#). If you think you have found an issue with Blender, you can easily [report a bug](#).

More details about support can be found on the [support page](#).

## Development

Being open source, some of Blender's development is done by volunteers. Communication between developers is done mostly through three platforms:

- The [developer.blender.org](https://developer.blender.org) system
- Various [mailing lists](#)
- The [#blendercoders](#) IRC channel (see below)

If you are interested in helping develop Blender, see the [Get Involved](#) page.

## IRC Channels

For real-time discussion, there are some Blender IRC channels on the Freenode network. You can join these with your favorite IRC client:

- [#blender](#) Community support channel
- [#blenderchat](#) For general discussion or offtopic chat
- [#blendercoders](#) For developers to discuss Blender development
- [#blenderpython](#) For support for developers using the Python API
- [#gameblender](#) For discussion on issues related to game creation with the GE
- [#blenderwiki](#) For discussion related to Blender's documentation

---

**Note:** If you do not have an IRC client, you can access IRC using [webchat](#).

---

There also several more Blender-related channels not listed here (e.g. channels for speakers of a particular language). We recommend you search Freenode to see them all.

## Other Useful Links

- [Blender FAQ](#) (Can I use Blender commercially? What is GPL/GNU? ...)
- [Demo and benchmark files](#)
- [Developers Ask Us Anything!](#)

## About this manual

In this manual aims to be a complete and concise source of information to help you to become familiar with the application. You can find links to the particular areas of interest in the navigation bar on the left.

## Conventions used

The mouse buttons are referred to as:

**LMB** Left Mouse Button

**RMB** Right Mouse Button

If your mouse has a wheel

**MMB** Middle Mouse Button

**Wheel** Scrolling the wheel.

Hotkey letters are shown in this manual like they appear on a keyboard; for example,

**G** refers to the lowercase `g`.

**Shift, Ctrl, Alt** are specified as modifier keys.

**Ctrl-W, Shift-Alt-A, ...** indicates that these keys should be pressed simultaneously

**Numpad0 to Numpad9, NumpadPlus** refer to the keys on the separate numeric keypad.

Other keys are referred to by their names, such as `Esc`, `Tab`, `F1` to `F12`. Of special note are the arrow keys, `Left`, `Right` and so on.

**Get Involved** If you would like to contribute to this manual, see [About this Manual](#), check for [open tasks](#), or join the [mailing list](#) and [#blenderwiki](#) channel on [IRC](#).

## 1.1.2 Installing Blender

### Getting Blender

Blender is available for download for Windows, Mac and Linux.

#### Minimum Requirements

Check if your system meets the [minimum or recommended requirements](#).

Always check that the graphics drivers are up to date, and that OpenGL is well supported. Other Blender dependencies are already included in the binary, and you do not need to worry about those unless installing from source.

Support for other hardware such as 3D mice and graphic tablets is covered later in [Supported Hardware](#).

#### Download Blender

The Blender Foundation distributes Blender in 3 different ways that you can choose from, to better suit your needs.

The options comprise binary packages for all the supported platforms and the source code. Within the binary packages, you can choose from a stable release or a daily build. The first has the benefit of being more reliable, the latter provides the newest features, as they are developed. Blender is released approximately every 3 months. You can keep up to date with the newest changes through the [release notes](#).

**Latest Stable Release** This is a binary distribution of the latest version of Blender. It is considered stable and without regressions.

**Daily Builds** This is a binary distribution of Blender that is updated daily to include the newest changes in development. These versions are not as thoroughly tested as the stable release, and might break, although they are official and generally not highly experimental.

#### Build from Source

---

**Note:** This is included for completeness, but it is **not** expected that regular users should have to compile their own Blender builds.

---

Blender's source is available for reference and installation, with the following advantages:

- Blender is always up to date,
- it allows access to any version or branch where a feature is being developed,
- it can be freely customized.

Building Blender from source is not trivial, as there are many steps involved. Here are the [instructions](#).

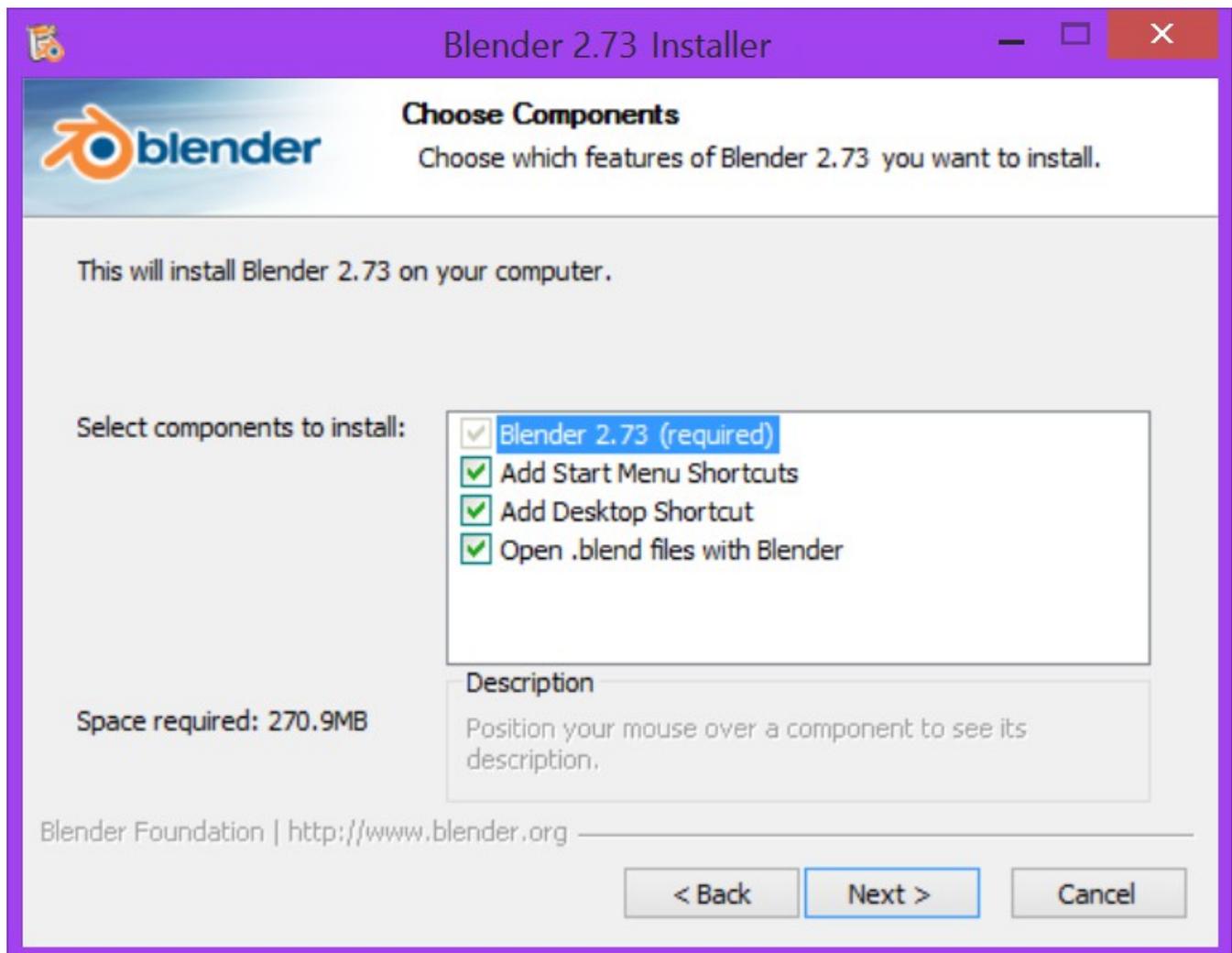
## Install Blender

The procedure for installing a binary, either the last stable release or a daily build, is the same. Follow the steps for your operative system:

**Installing on Windows** Check the [minimum requirements](#) and [where to get Blender](#), if you haven't done so yet.

Download the `.zip` or `.exe` for your architecture (64bit is preferable if your machine supports it).

The `.exe` will run an installer to choose where to place Blender and to configure Windows to have an entry on the menu and to open `.blend` files with Blender. Administrator rights are needed to install Blender on your system.



With `.zip` you have to manually copy and extract Blender to the desired folder, where you can then double-click the executable to run Blender. There is no installer to place Blender on the menu, but there is also no need for Administrator rights. With this option it is possible to have multiple versions of Blender without conflicting, as they are not actually installed on the system.

**Installing on OSX** Check the [minimum requirements and where to get Blender](#), if you haven't done so yet.

After downloading Blender for the Mac, uncompress the file and drag `blender.app` onto the Applications folder. Alternatively, you can launch Blender directly from the uncompressed folder.

**Running from the terminal** To run Blender from the terminal without needing to be in the same directory and navigate through the app bundle, add the following alias to your profile:

```
echo "alias blender=/Applications/Blender/blender.app/Contents/MacOS/blender" >> ~/.profile
```

**Menus** Because *Blender* doesn't use the standard OS menu system, if you are using a Mac, you likely have a redundant menubar at the top. To remove it see [this post](#) on Macworld, but beware that it is somewhat complex. As an alternative: simply make *Blender* full screen with the last button in the info window header (most times at the top of the screen layout).

**Installing on Linux** Check the [minimum requirements and where to get Blender](#), if you haven't done so yet.

**Specific packages for distributions** Some Linux distributions may have on their repositories a specific package for Blender. Installing Blender via the distribution's native mechanisms ensures consistency with other packages on the system and may provide other features (given by the package manager), such as listing of packages, update notifications and automatic menu configuration. Be aware, though, that the package may be outdated comparing to the latest official release, or not include some features of Blender. For example, some distributions do not build Blender with CUDA support for licensing reasons.

If there is a specific package for your distribution, you may choose what is preferable and most convenient, otherwise there is nothing wrong with the official binary on [blender.org](#).

**Download from blender.org** Download the Linux version for your architecture and uncompress the file to the desired location (eg. `~/software` or `/usr/local`).

Blender can now be launched by double-clicking the executable.

For easy access, you can configure your system by adding a menu entry or shortcut for Blender and associate and open `.blend` files with Blender when opening from the file browser. These settings typically belong to the Window Manager (KDE, Gnome, Unity).

**Running from the terminal** To run Blender from the terminal without needing to be in the executable directory, add the extracted folder to the environment `PATH`.

Add the following command to `.bash_rc` or `.bash_profile` with Blender's binary:

```
export PATH=$/path/to/blender-VERSION-linux-glibcVERSION-ARCH:$PATH
```

**Tip:** If you use daily builds and update Blender frequently, you can link or always rename your folder to 'blender' and use this name for the `PATH` environment variable and for keeping the window manager menu up to date.

**Avoiding Alt+Mouse Conflict** Many Window Managers default to `Alt-LMB` for moving windows, which is a shortcut that Blender uses to simulate a 3 button mouse. You can either have this feature disabled *User Preferences* → *Input* → *Emulate 3 Button Mouse* or you can change the Window Manager settings to use the Meta key instead (also called Super or Windows key):

- **KDE:** System Settings > Window Behavior > Window Behavior > Window Actions , Switch ‘Alt’ for ‘Meta’ key
- **Unity/Gnome:** enter the following in a command line (effective at next login):

```
gsettings set org.gnome.desktop.wm.preferences mouse-button-modifier '<Super>'
```

## Configuring Peripherals

### Multi-Monitor Setup

### Graphic Tablets

### 3D Mice

### Configuration

Here are some quick preferences that you may wish to set as quickly as possible. The full list and explanation of the preferences is in the section [User Preferences](#).

### Language

At *File* → *User Preferences* → *System*, enable `International Fonts` to choose the Language and what to translate from `Interface`, `Tooltips` and `New Data`. See more at [Internationalization](#)

### Input

If you have a compact keyboard without a separate number pad enable *File* → *User Preferences* → *Emulate Numpad*.

If you don't have a middle mouse button you can enable *File* → *User Preferences* → *Emulate 3 Button Mouse*.

### File and Paths

At *File* → *User Preferences* → *File* you can set options such as what external `Image Editor` to use, such as GIMP or Krita, and the `Animation Player`.

The `Temp` directory sets where to store files such as temporary renders and autosaves.

---

**Tip:** `//` at the start of a path in Blender means the directory of the currently opened `.blend` file, used to reference relative-paths.

---

If you trust the source of your `.blend` files, you can enable `Auto Run Python Scripts`. This option is meant to protect you from malicious Python scripts that someone can include inside a Blender file. This would not happen by accident, and most users leave this option on to automatically run scripts such as `Rigify` that controls the skeleton of a human rig.

## Configuration and Data Paths

There are 3 different directories Blender may use, their exact locations are operating system dependent.

**LOCAL** Location of configuration and runtime data (for self contained bundle)

**USER** Location of configuration files (normally in the user's home directory).

**SYSTEM** Location of runtime data for system wide installation (may be read-only).

For system installations both **SYSTEM** and **USER** directories are needed.

For locally extracted Blender distributions, the user configuration and data runtime data are kept in the same sub-directory, allowing multiple Blender versions to run without conflict, ignoring the **USER** and **SYSTEM** files.

---

**Note:** You may need to have the “show hidden files” option checked in your file browser settings.

---

Here are the default locations for each system:

### Linux

#### LOCAL

`./2.75/`

#### USER

`$HOME/.config/blender/2.75/`

#### SYSTEM

`/usr/share/blender/2.75/`

---

**Note:** The path `./2.75/` is relative to the Blender Executable & used for self contained bundles distributed by official blender.org builds.

---

**Note:** The **USER** path will use `$XDG_CONFIG_HOME` if its set:

`$XDG_CONFIG_HOME/blender/2.75/`

---

### OSX

#### LOCAL

`./2.75/`

#### USER

`/Users/{user}/Library/Application Support/Blender/2.75/`

#### SYSTEM

`/Library/Application Support/Blender/2.75/`

---

**Note:** OSX stores the Blender binary in `./blender.app/Contents/MacOS/blender`, so the local path to data & config is:

`./blender.app/Contents/MacOS/2.75/`

---

## Windows

### LOCAL

.\2.75\.

### USER

C:\Documents and Settings\{username}\AppData\Roaming\Blender Foundation\Blender\2.75\

### SYSTEM

C:\Documents and Settings\All Users\AppData\Roaming\Blender Foundation\Blender\2.75\

## Path Layout

This is the path layout which is used within the directories described above.

Where `./config/startup.blend` could be `~/blender/2.75/config/startup.blend` for example.

`./autosave/` ... Autosave blend file location. *Windows only, temp directory used for other systems.*

Search order: LOCAL, USER.

`./config/` ... Defaults & session info.

Search order: LOCAL, USER.

`./config/startup.blend` Default file to load on startup.

`./config/userpref.blend` Default preferences to load on startup.

`./config/bookmarks.txt` File selector bookmarks.

`./config/recent-files.txt` Recent file menu list.

`./datafiles/` ... Runtime files.

Search order: LOCAL, USER, SYSTEM

`./datafiles/locale/{language}/` Static precompiled language files for UI translation.

`./datafiles/icons/*.png` Icon themes for Blenders user interface. *Not currently selectable in the theme preferences.*

`./datafiles/brushicons/*.png` Images for each brush.

`./scripts/` ... Python scripts for the user interface and tools.

Search order: LOCAL, USER, SYSTEM.

`./scripts/addons/*.py` Python add-ons which may be enabled in the user preferences, includes import/export format support, render engine integration and many handy utilities.

`./scripts/addons/modules/*.py` Modules for add-ons to use (added to Python's `sys.path`).

`./scripts/addons_contrib/*.py` Another add-ons directory which is used for community maintained add-ons (must be manually created).

`./scripts/addons_contrib/modules/*.py` Modules for `addons_contrib` to use (added to Python's `sys.path`).

`./scripts/modules/*.py` Python modules containing our core API and utility functions for other scripts to import (added to Python's `sys.path`).

`./scripts/startup/*.py` Scripts which are automatically imported on startup.

`./scripts/presets/{preset}/*.py` Presets used for storing user defined settings for cloth, render formats etc.

`./scripts/templates/*.py` Example scripts which can be accessed from: Text Space's Header → Text → Script Templates.

`./python/ . . .` Bundled Python distribution, only necessary when the system Python installation is absent or incompatible.

Search order: LOCAL, SYSTEM.

When starting Blender, the splash screen appears. On the left side are links to official web pages, and on the right are your most recently opened projects.

To close the splash screen start a new project, you can press `ESC` or click anywhere inside the Blender Window (except on the splash screen).

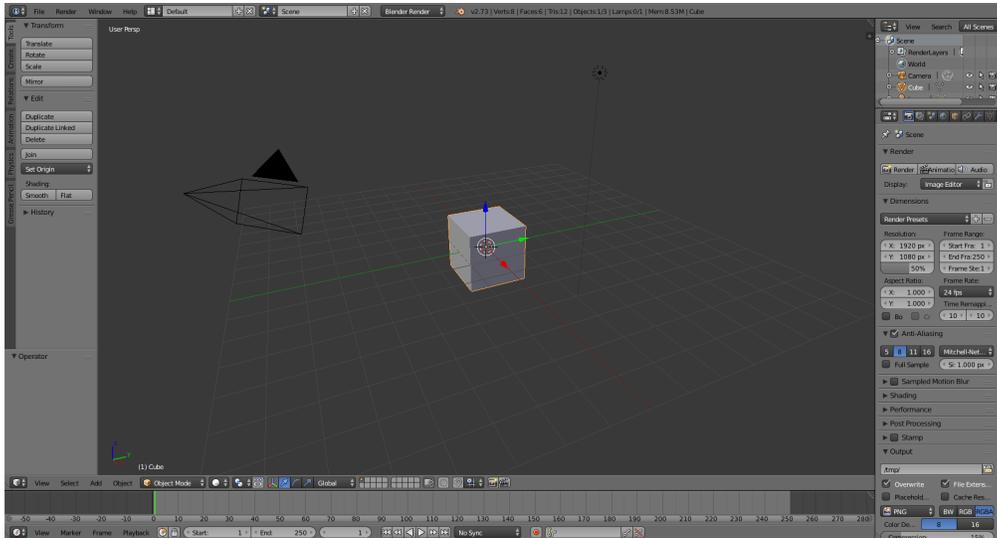


Fig. 1.3: After closing splash screen, this is what the default Blender window looks like.

## 1.1.3 Basics

### Interface

#### Overview

**Introduction** Blender's user interface is consistent across all platforms. The interface can be customized to match specific tasks using Screen Layouts, which can then be named and saved for later use.

Blender also makes heavy use of keyboard shortcuts to speed up work and allows customization of the [keymap](#).

#### User Interface Principles

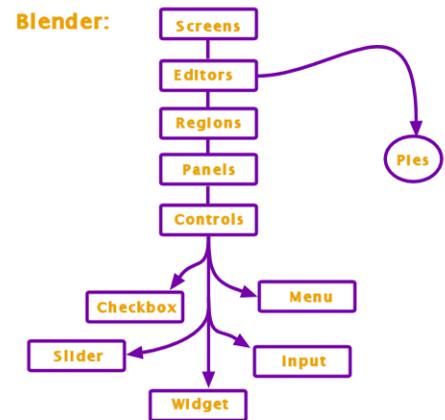
**Non Overlapping** The UI is designed to allow you to view all relevant options and tools at a glance without pushing or dragging editors around.

**Non Blocking** Tools and interface options do not block the user from any other parts of Blender. Blender typically doesn't use pop-up boxes (requiring users to fill in data before running an operation).

**Non Modal Tools** Tools can be accessed efficiently without taking time to select between different tools. Many tools use consistent and predictable, mouse and keyboard actions for interaction.



Fig. 1.4: This is an example of Blender's multiple window support.



**Screen Elements** The Blender window is organized into one or more *Areas* with each area containing an *Editor*. Editors are divided into a *Header* and one or more *Regions*. Regions can have smaller structuring elements like *panels* with buttons, controls and widgets placed within them.

The composition of various *Areas* with predefined *Editors* in them is called a *Screen Layout*. By default Blender starts up with a layout of 5 *Editors* as shown in the image below.

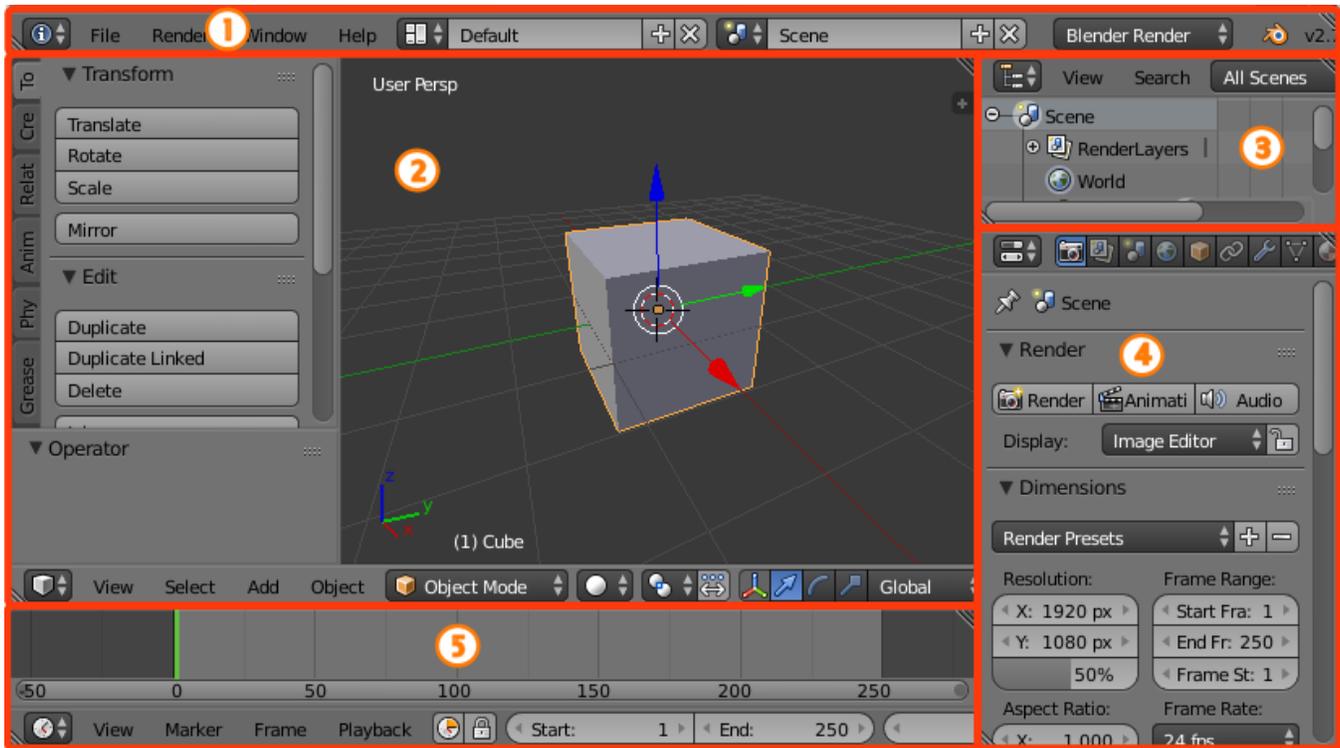


Fig. 1.5: Blender's default Screen Layout with 5 Editors: Info (1), 3D View (2), Outliner (3), Properties (4) and Timeline (5)

**Components of an Editor** In general an editor provides a way to view and modify your work through a specific part of Blender.

The image below shows the 3D View as an example of an editor.

Editors are consistently organized into following parts:

**Regions** At least one region of an editor is always visible. It's called the main region and is the most prominent part of the editor. In the 3D View above this is marked with a green frame.

Aside from that there can be more regions available. In the 3D View above these are the *Toolshef* (toggle visibility with T) on the left side and the *Properties* (toggle visibility with N) on the right side. They're marked with red frames. Additional regions mostly show context-sensitive content.

Each editor has a specific purpose, so the main region and the availability of additional regions are different between editors. See specific documentation about each editor in the [Editors](#) chapter.

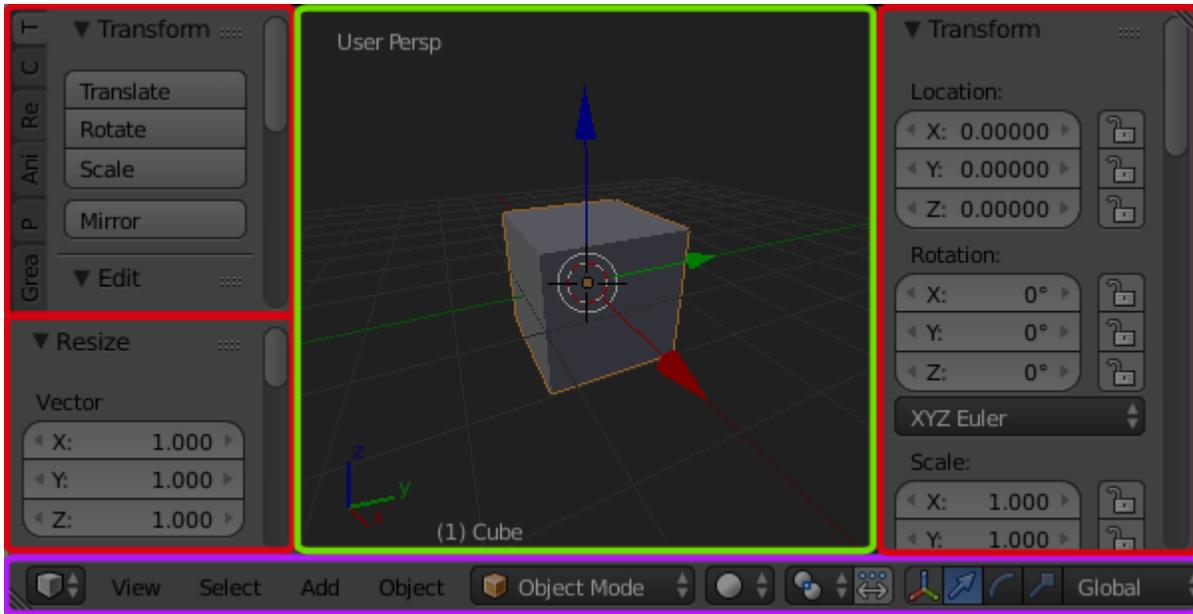


Fig. 1.6: The 3D View

Table 1.1: Useful Hotkeys

T	Toggle visibility of Toolshelf Region
N	Toggle visibility of Properties Region
F5	Flip the Region under the mouse pointer to the opposite side

**Header** A header is a small horizontal part of an editor and sits either at the top or bottom of the area. It acts as a container for menus and commonly used tools. Much like additional regions the header can be hidden.

The 3D View above the header is marked with a purple frame.

Table 1.2: Useful Hotkeys

F5	Move Header from Top to Bottom (mouse pointer must be over it)
----	--

See: [Headers](#) for details.

**Panels** The smallest organizational unit in the user interface is a panel, which can be collapsed to hide its contents by clicking on its header. This is where the buttons, menus, checkboxes, etc. are located.

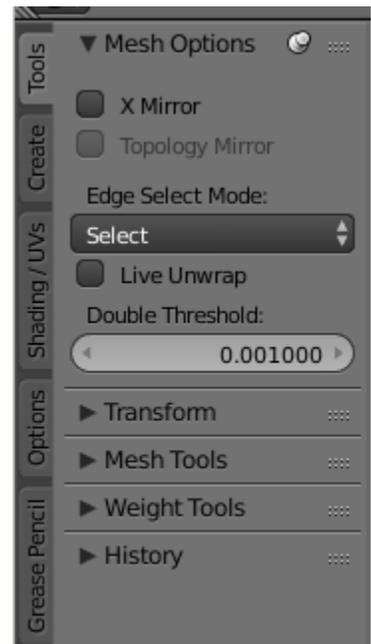
Panels are usually found in the side regions of an editor, but also make up most of the [Properties Editor](#)'s main region.

In the image on the right there are 3 panels: **Transform**, **Edit** and **History**. The edit panel is expanded and the other 2 panels are collapsed. Note that you can change the order of panels by clicking on the handle in the upper right corner of a panel's title.

See: [panels](#) for details.

**Tabs** The Toolshelf has been further structured into a set of context sensitive vertical tabs.

In the image to the right you can see the tabs: **Tools**, **Create**, etc. The **Tools** tab is currently selected, showing a set of panels containing various tools.



**Pinning** Often it is desirable to view panels from different tabs at the same time. This has been solved by making panels pinnable.

A pinned panel remains visible regardless of which tab has been selected. You can pin a panel by `Shift` clicking its header, or by right clicking on the header and choosing *Pin*.

Shown in the image above is an example of the *Mesh Options* pinned in the tools tab.

**Input Devices** Blender supports various types of input devices:

- Keyboard (recommended: keyboard with numeric keypad, english layout works best)
- Mouse (recommended: 3 button mouse with scroll wheel)
- NDOF Devices (also known as *3D Mouse*)
- Graphic Tablets

**Usage of Mouse Buttons** In Blender the `RMB` (Right Mouse Button) is generally used for Selection and the `LMB` (Left Mouse Button) initiates or confirms actions.

The mouse usage summarized:

<code>RMB</code>	To select an item
<code>Shift-RMB</code>	To add more items to the selection
<code>LMB</code>	to perform an action on the selection

Video: [Learn more about Blender's Mouse Button usage](#)

**Exceptions from the Rule** There are a few corner cases where `LMB` is used for selection. For example, the *File Selector*.

**Non English Keyboard** If you use a keyboard with a non-english keyboard layout, you still may benefit from switching your computer to the UK or US layout as long as you work with Blender.

---

**Note:** You can also change the default keymap and default hotkeys from the [User Preferences](#), however this manual assumes you are using the default keymap. [Read more about Blender configuration](#)

---

## Window Controls

### Window System

**The Window System** When you start Blender you should see a screen similar to this (the splash screen in the center will change with new versions):

In the center of the window is the splash screen. This gives quick and easy access to recently opened Blender files. If you want to start work on a new file just click outside of the splash screen. The splash screen will disappear revealing the default layout and cube.

Every window you see can be further broken down into separate areas (as described in the section on [arranging frames](#)). The default scene is described below.

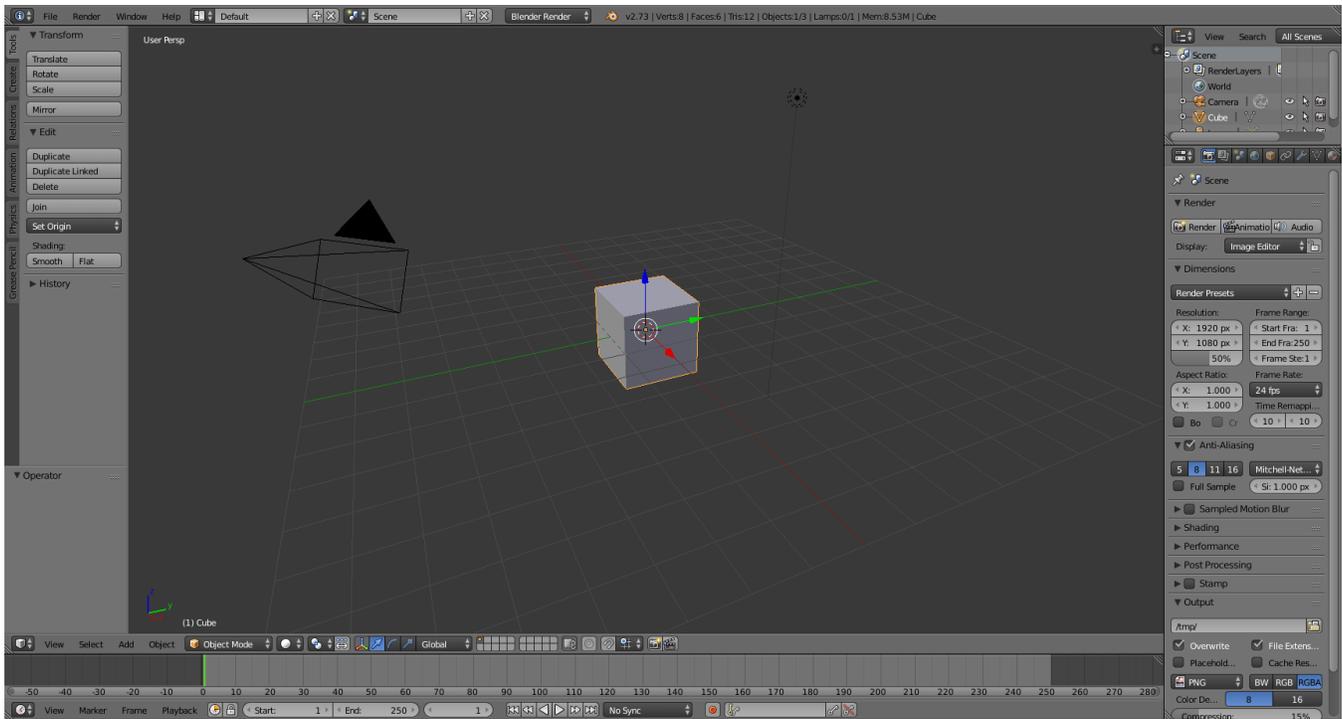


Fig. 1.7: Initial Blender screen

**The default scene** The default scene is separated into five windows and is loaded each time you start Blender or a new file. The five windows are:

- The Info window (shaded red) at the top. The Info window is comprised solely of a header.
- A large 3D window (3D View) (shaded green).
- A Timeline window at the bottom (shaded purple).
- An Outliner window at the top right (shaded yellow).
- A Properties window (Buttons window) at the bottom right (shaded blue).

As an introduction we will cover a few of the basic elements.

**Arranging Frames** Blender uses a novel screen-splitting approach to arrange window frames. The application window is always a rectangle on your desktop. It divides it up into a number of re-sizable window frames. A window frame contains the workspace for a particular type of window, like a 3D View window, or an Outliner. The idea is that you split up that big application window into any number of smaller (but still rectangular) non-overlapping window frames. That way, each window is always fully visible, and it is very easy to work in one window and hop over to work in another.

**Maximizing a Window** You can maximize a window frame to fill the whole application window with the *View → Toggle Full Screen* menu entry. To return to normal size, use again *View → Toggle Full Screen*. A quicker way to achieve this is to use **Shift-Spacebar**, **Ctrl-Down** or **Ctrl-Up** to toggle between maximized and framed windows. NOTE: The window your mouse is currently hovering over is the one that will be maximized using the keyboard shortcuts.

**Splitting a Window** In the upper right and lower left corners of a window are the window splitter widgets, and they look like a little ridged thumb grip. It both splits and combines window panes. When you hover over it, your cursor will change to a cross. LMB and drag it to the left to split the window pane vertically, or downward to split it horizontally.

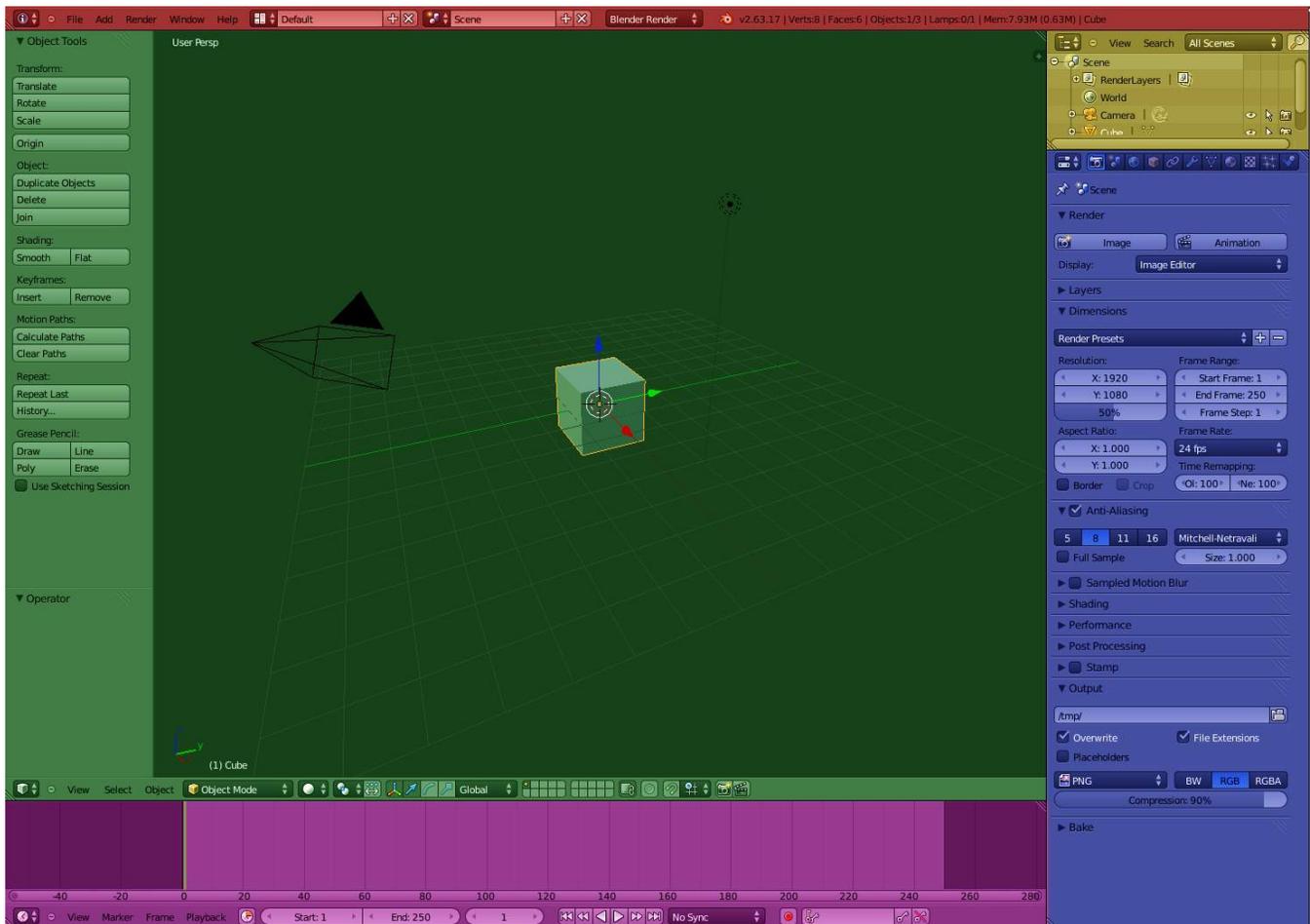
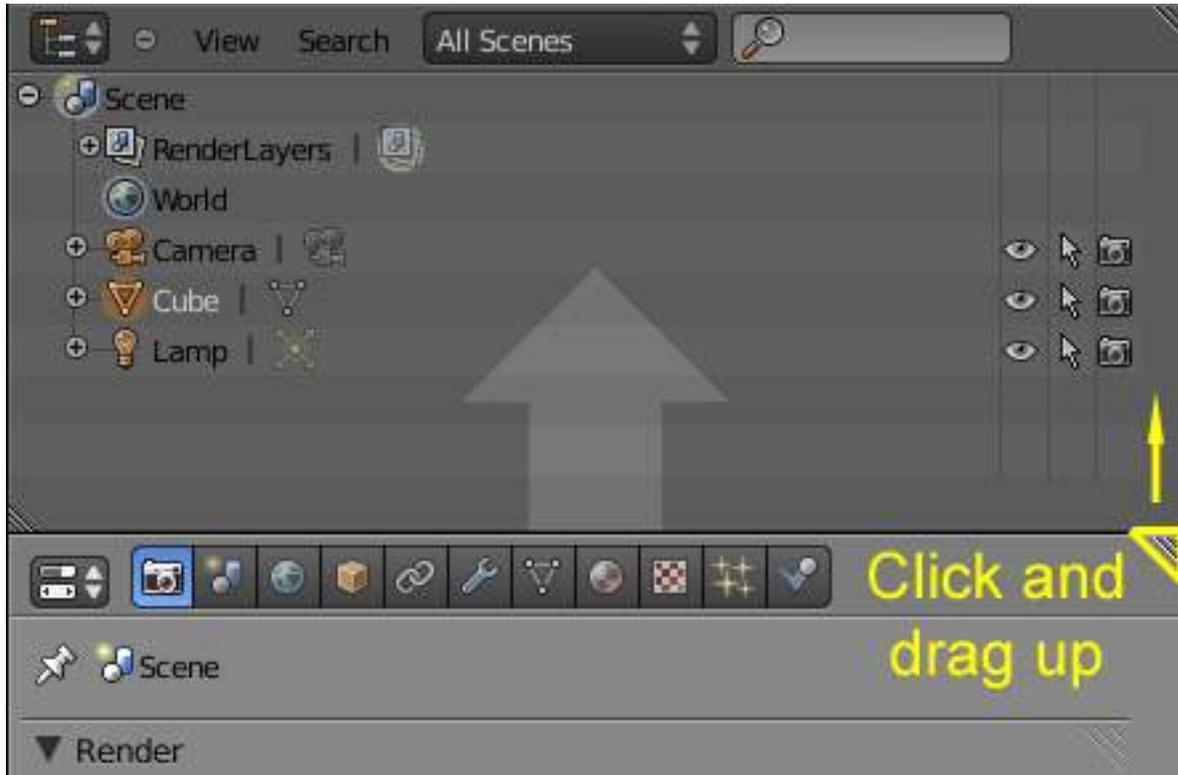


Fig. 1.8: Default Blender scene and Window arrangement



**Joining Two Frames** In order to merge two window frames, they must be the same dimension in the direction you wish to merge. For example, if you want to combine two window frames that are side-by-side, they must be the same height. If the one on the left is not the same as the one on the right, you will not be able to combine them horizontally. This is so that the combined window space results in a rectangle. The same rule holds for joining two window frames that are stacked on top of one another; they must both have the same width. If the one above is split vertically, you must first merge those two, and then join the bottom one up to the upper one.



To merge the current window with the one above it (in the picture the properties window is being merged “over” the Outliner), hover the mouse pointer over the window splitter. When the pointer changes to a cross, LMB click and drag up to begin the process of combining. The upper window will get a little darker, overlaid with an arrow pointing up. This indicates that the lower (current) frame will “take over” that darkened frame space. Let go of the LMB to merge. If you want the reverse to occur, move your mouse cursor back into the original (lower) frame, and it will instead get the arrow overlay.

In the same way, windows may be merged left to right or vice versa.

If you press `Esc` before releasing the mouse, the operation will be aborted.

**Changing Window Size** You can resize window frames by dragging their borders with LMB. Simply move your mouse cursor over the border between two frames until it changes to a double-headed arrow, and then click and drag.

**Swapping Contents** You can swap the contents between two frames with `Ctrl-LMB` on one of the splitters of the initial frame, dragging towards the target frame, and releasing the mouse there. Those two frames don’t need to be side by side, though they must be inside the same window.

**Opening New Windows** You may wish to have a new full window containing Blender frames. This can be useful, for instance, if you have multiple monitors and want them to show different information on the same instance of Blender.





**Header position** To move a header from top to bottom or the other way round, simply RMB on it and select the appropriate item from the pop-up menu. If the header is at the top, the item text will read “Flip to Bottom”, and if the header is at the bottom the item text will read “Flip to Top”.

---

**Tip:** Theme colors

Blender allows for most of its interface color settings to be changed to suit the needs of the user. If you find that the colors you see on screen do not match those mentioned in the Manual then it could be that your default theme has been altered. Creating a new theme or selecting/altering a pre-existing one can be done by selecting the [User Preferences](#) window and clicking on the *Themes* tab of the window.

---

**Window type button** LMB clicking on the first icon at the left side of a header allows changing the window type. Every window frame in Blender may contain any type of window, allowing you to customize your window layout to your own work flows.

**Menus and buttons** Most Window Headers, located immediately next to this first “Window Type” Menu button, exhibit a set of menus which can be hidden - again with a little minus sign. So if you cannot find a menu that was mentioned somewhere, try looking for a little plus sign (once again) next to the “Window Type” button. By clicking LMB on it, the menu will come back.

Menus allow you to directly access many features and commands, so just look through them to see what’s there. All Menu entries show the relevant shortcut keys, if any.

Menus and buttons will change with *Window Type* and the selected object and mode. They only show the actions that can be performed.

**Collapsing Menus** Sometimes its helpful to gain some extra horizontal space in the header by collapsing menus, this can be accessed from the header context menu, simply right click on the header and enable set it to collapsed.

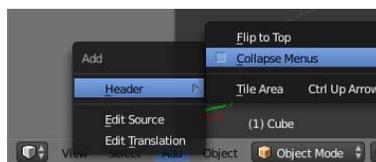


Fig. 1.9: Right click to access the header menu

**The Console Window** The *Console Window* is an operating system text window that displays messages about Blender operations, status, and internal errors. If Blender crashes on you, the *Console Window* may be able to indicate the cause or error.

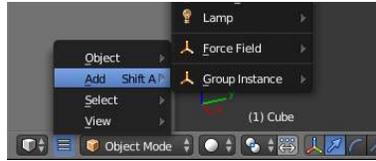


Fig. 1.10: Access the menu from the collapsed icon



Fig. 1.11: The Blender Console Window on Windows XP and subsequent messages.

**Microsoft Windows** When Blender is started on a Windows operating system, the *Console Window* is first created as a separate window on the desktop. The main Blender window will also appear and the *Console Window* will then be toggled off. To display the console again, go to *Window* → *Toggle System Console*.

The screenshot shows the Blender *Console Window* on Windows XP directly after starting Blender and then a short while later after opening a file along with the relevant messages.

**Tip:** Closing the Blender Console Window

The Blender *Console Window* must remain open while Blender is running. Closing the *Console Window* will also close Blender, losing any unsaved work. To turn off the console without closing Blender, toggle the console state to off via re-selecting *Toggle System Console* option from the drop-down menu *Window* → *Toggle System Console*.

Note the Blender *Console Window* can look very similar to MS-DOS, so make sure that you are closing the correct window if an instance of MS-DOS is open.



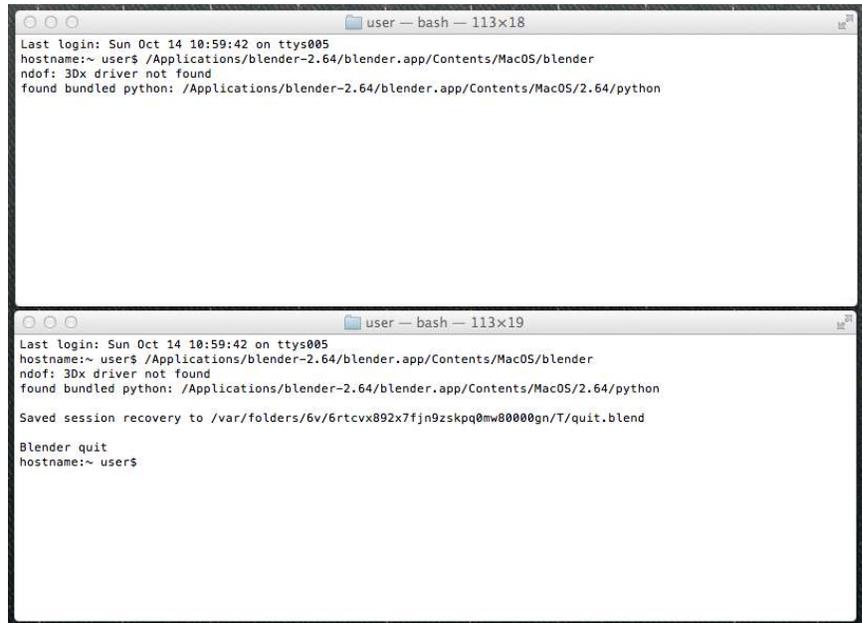
Fig. 1.12: Starting Blender from a Linux console window and subsequent messages.

**Linux** The Blender *Console Window* in Linux will generally only be visible on the desktop if Blender is started from a terminal, as Blender outputs to the *Console Window* it is started from.

Depending on your desktop environment setup, a Blender icon may appear on your desktop or an entry for Blender added to your menu after you install Blender. When you start Blender using a desktop icon or menu entry rather than a Terminal window, the Blender *Console Window* text will most likely be hidden on the Terminal that your *XWindows* server was started from.

This screenshot shows Blender started from a Linux Terminal and the resulting console text being printed to it. This example

shows that when Blender was started it was unable to access a library related to the Pulseaudio sound server. When Blender closed, it saved the recovery file to `/tmp/quit.blend`.



```

user — bash — 113x18
Last login: Sun Oct 14 10:59:42 on ttys005
hostname:~ user$ /Applications/blender-2.64/blender.app/Contents/MacOS/blender
ndof: 3Dx driver not found
found bundled python: /Applications/blender-2.64/blender.app/Contents/MacOS/2.64/python

user — bash — 113x19
Last login: Sun Oct 14 10:59:42 on ttys005
hostname:~ user$ /Applications/blender-2.64/blender.app/Contents/MacOS/blender
ndof: 3Dx driver not found
found bundled python: /Applications/blender-2.64/blender.app/Contents/MacOS/2.64/python

Saved session recovery to /var/folders/6v/6rtcvx892x7fjn9zskpq0mw80000gn/T/quit.blend

Blender quit
hostname:~ user$

```

Fig. 1.13: Starting Blender from a Mac OS X console window and subsequent messages.

**MacOS** MacOS uses “files” with the `.app` extension called *applications*. These files are actually folders that appear as files in Finder. In order to run Blender you will have to specify that path to the Blender executable inside this folder, to get all output printed to the terminal. You can start a terminal from Applications → Utilities. The path to the executable in the `.app` folder is `./blender.app/Contents/MacOS/blender`.

If you have Blender installed in the Applications folder, the following command could be used/adapted to the particular Blender version: `/Applications/blender-2.75/blender.app/Contents/MacOS/blender`

**Console Window Status and Error Messages** The *Blender Console Window* can display many different types of Status and Error Messages. Some messages simply inform the user what Blender is doing, but have no real impact on Blender’s ability to function. Other messages can indicate serious errors that will most likely prevent Blender carrying out a particular task and may even make Blender non-responsive or shut down completely. The *Blender Console Window* messages can also originate internally from within the Blender code or from external sources such as [Python scripts](#).

### Common messages

- found bundled python: (FOLDER)

This message indicates that Blender was able to find the [Python](#) library for the Python interpreter embedded within Blender. If this folder is missing or unable to be found, it is likely that an error will occur, and this message will not appear.

- malloc returns nil()

When Blender carries out operations that require extra memory (RAM), it calls a function called malloc (short for memory allocate) which tries to allocate a requested amount of memory for Blender. If this cannot be satisfied, malloc will return nil/null/0 to indicate that it failed to carry out the request. If this happens Blender will not be able to carry out the operation requested by the user. This will most likely result in Blender

operating very slowly or shutting down. If you want to avoid running out of memory you can install more memory in your system, reduce the amount of detail in your Blender models, or shut down other programs and services which may be taking up memory that Blender could use.

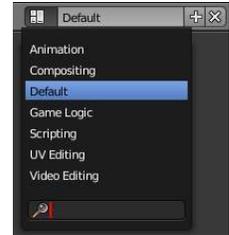


Fig. 1.14: Layout dropdown

**Screens** Blender’s flexibility with windows lets you create customized working environments for different tasks such as modeling, animating, and scripting. It is often useful to quickly switch between different environments within the same file.

To do each of these major creative steps, Blender has a set of pre-defined *screens*, that show you the types of windows you need to get the job done quickly and efficiently. *Screens* are essentially pre-defined window layouts. If you are having trouble finding a particular screen, you can use the search function at the bottom of the list (pictured right).

#### Default Screens available

**3D View Full** A full screen 3D view, used to preview your scene.

**Animation** Making actors and other objects move about, change shape or color, etc.

**Compositing** Combining different parts of a scene (e.g. background, actors, special effects) and filter them (e.g. color correction).

**Default** The default layout used by Blender for new files. Useful for modeling new objects.

**Game Logic** Planning and programming of games within Blender.

**Motion Tracking** Used for motion tracking with the movie clip editor.

**Scripting** Documenting your work and/or writing custom scripts to automate Blender.

**UV Editing** Flattening a projection of an object mesh in 2D to control how a texture maps to the surface.

**Video Editing** Cutting and editing of animation sequences.

Screens can be selected in the *Info Window* header that is at the top of the layout for preset screens. This is often confused for a menu bar by those new to Blender; however it is simply a window showing only its *header*.

To cycle between screens use `Ctrl-Right` and `Ctrl-Left`.



Fig. 1.15: Screen and Scene selectors

By default, each screen layout ‘remembers’ the last *scene* it was used on. Selecting a different layout will switch to the layout **and** jump to that scene.

All changes to windows, as described in [Editor types](#), are saved within one screen. If you change your windows in one screen, other screens won’t be affected.

#### Configuring your Screens

**Adding a new Screen Type** Click on the “Add” button (⊕) and a new frame layout will be created based on your current layout.

**Deleting a Screen** You can delete a screen by using the *Delete datablock* button (⊗). See *Screen and Scene selectors* above.

**Rearranging a Screen** Use the [window controls](#) to move frame borders, split and consolidate windows. When you have a layout that you like, press `Ctrl-U` to update your User defaults. Be aware that all of the current scenes become part of those defaults, so consider customizing your layouts with only a single, simple scene.

The properties window has a special option: pressing RMB on its background will allow you to arrange its panels horizontally or vertically. Of the two, vertically-arranged panels have greater support.

**Overriding Defaults** When you save a .blend file, the screen layouts are also saved in it. When you open a file, enabling the *Load UI* checkbox in the file browser indicates that Blender should use the file’s screen layouts (overriding your defaults in the process). Leaving the *Load UI* checkbox disabled tells Blender to use the current layout.

**Additional Layouts** As you become more experienced with Blender, consider adding some other screen layouts to suit your workflow as this will help increase your productivity. Some examples could include:

**1-Model** Four 3D windows (top, front, side and perspective), Properties window for Editing

**2-Lighting** 3D windows for moving lights, UV/Image Window for displaying Render Result, Properties window for rendering and lamp properties and controls

**3-Material** Properties window for Material settings, 3D window for selecting objects, Outliner, Library script (if used), Node Editor (if using [Node based materials](#))

**4-UV Layout** UV/Image Editor Window, 3D Window for seaming and unwrapping mesh

**5-Painting** UV/Image Editor for texture painting image, 3D window for painting directly on object in UV Face Select mode, three mini-3D windows down the side that have background reference pictures set to full strength, Properties window

**6-Animation** Graph Editor, 3D Window for posing armature, NLA Window

**7-Node** Big Node Editor window for noodles, UV/Image window linked to Render Result

**8-Sequence** Graph Editor, video sequence editor in Image Preview mode, video sequence editor in timeline mode, a Timeline window, and the good old Properties window.

**9-Notes/Scripting** Outliner, Text Editor (Scripts) window

---

**Note:** Reuse your Layouts

If you create a new window layout and would like to use it for future .blend files, simply save it as the User default by pressing `Ctrl-U` (don’t forget: all screens and scenes themselves will be saved as default too).

---

**Scenes** Scenes are a very useful tool for managing your projects. The Cube model in empty space you see when you open Blender for the first time is the default scene. You can imagine scenes to be similar to tabs in your web browser. For example, your web browser can have many tabs open at once. The tabs could be empty, showing identical views of the same web page, showing different views of the same page or show entirely different pages altogether. Blender’s scenes work in much the same way. You can have an empty scene, a complete independent copy of your scene or a new copy that links to your original scene in a number of ways.

You can select and create scenes with the *Scene selector* in the Info window header (the bar at the top of most Blender layouts, see *Screen and Scene selectors*).



Fig. 1.16: Screen and Scene selectors

## Scene configuration

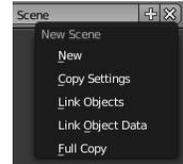


Fig. 1.17: Add Scene menu

**Adding a new Scene** You can add a new scene by clicking the *Add* button (⊕) in the scene selector option. When you create a new scene, you can choose between a number of options to control its contents.

To choose between these options, it's useful to understand the difference between *Objects* and *Object Data*. See [Duplication](#).

The choices for adding a scene, therefore determine just how much of this information will be *copied from* the active scene to the new one, and how much will be *shared* (linked).

**New** Creates an empty scene with default values.

**Copy Settings** Creates an empty scene but also copies the settings from the active scene into the new one.

**Link Objects** This option creates a new scene with the same settings and contents as the active scene. However, instead of copying the objects, the new scene contains links to the objects in the old scene. Therefore, changes to objects in the new scene will result in the same changes to the original scene, because the objects used are literally the same. The reverse is also true.

**Link Object Data** Creates new, duplicate copies of all of the objects in the currently selected scene, but each one of those duplicate objects will have *links to* the object-data (meshes, materials and so on) of the corresponding objects in the original scene.

This means that you can change the position, orientation and size of the objects in the new scene without affecting other scenes, but any modifications to the object-data (meshes, materials *etc*) will also affect other scenes. This is because a *single instance of* the “object-data” is now being shared by all of the objects in all of the scenes that link to it.

More information at the [Window Type](#) page. This has the effect of making a new independent copy of the object-data.

**Full Copy** Using this option, nothing is shared. This option creates a fully independent scene with copies of the active scenes contents. Every object in the original scene is duplicated, and a duplicate, private copy of its object-data is made as well.

To better understand the way Blender works with data, read through [Blender's Library and Data System](#).

---

### Note: A Brief Example

Consider a bar scene in a film. You initially create the bar as a clean version, with everything unbroken and in its proper place. You then decide to create the action in a separate scene. The action in your scene will indicate which type of linking (if any) would suit your scene best.

**Link Objects** Every object will be linked to the original scene. If you correct the placement of a wall, it will move in every scene that uses the bar as a setting.

**Link Object Data** Will be useful when the positions of objects need to change, but their shape and material settings will remain constant. For example, chairs might stand on the floor in the “crowded bar” scene and up on the tables in the “we are closing” scene. Since the chairs don't change form, there is no need to waste memory on exact mesh-copies.

**Full Copy** A glass shattering on the floor will need its own copy because the mesh will change shape.

It is not possible to do all of the above in the same scene, but it might help in understanding why you would link different objects in different ways.

**Deleting a Scene** You can delete a scene by using the *Delete datablock* button (☒) from the scene selector option (see *Screen and Scene selectors*).

## Interface Controls

**Panels** Panels are collapsible sections within regions to help organise the interface. They are heavily used in the *Properties Editor* but also appear elsewhere (For example: in the *Tool Shelf* or the *Properties Shelf*, available in some editors).

The image below shows panels in different regions in their expanded and collapsed state.

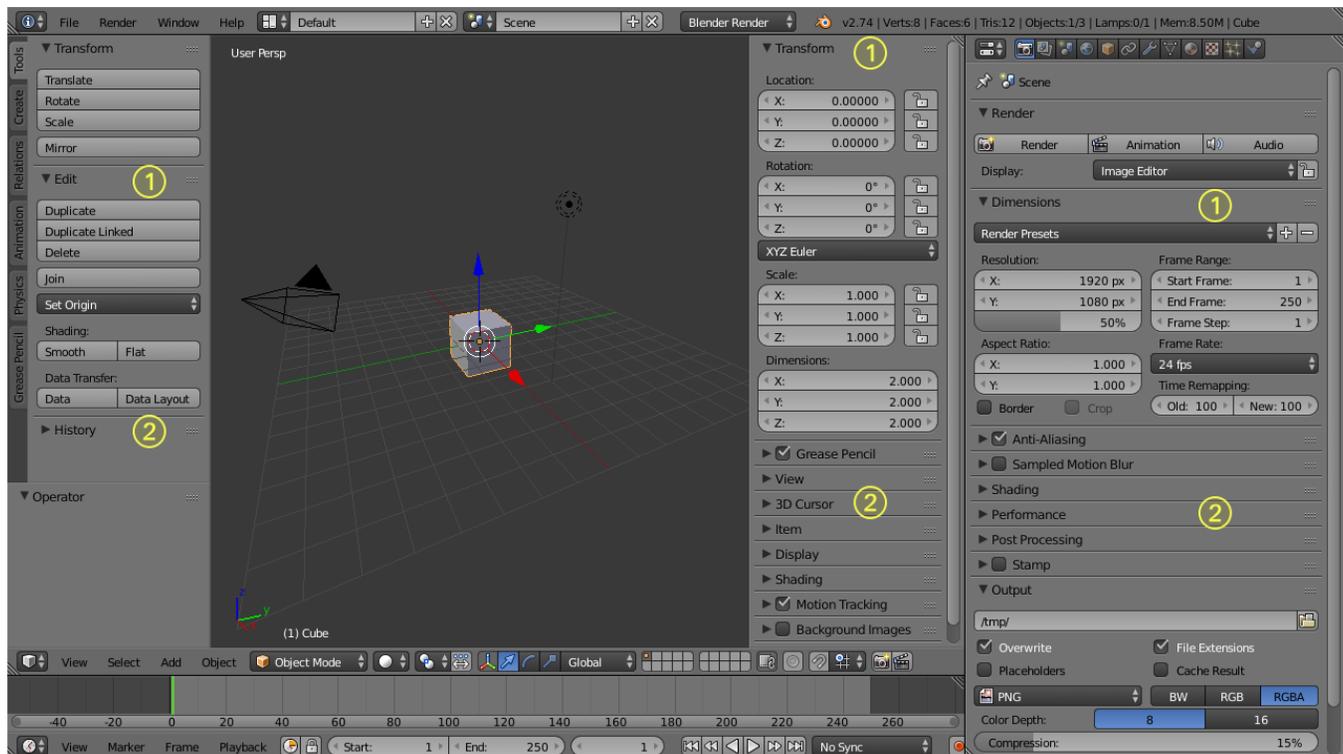


Fig. 1.18: Expanded (1) and collapsed (2) Panels in the Properties Editor (right area) and in the additional Regions of the 3D View Editor (left area)

- A click with the LMB on the title area of a panel expands or collapses it.
- A LMB drag motion over the title area will expand or collapse many at once.
- A **Ctrl**+LMB click on the title area of a specific panel will collapse all other panels and make this the only expanded one.

Some panels only show in certain contexts. So for instance the *Tool Shelf* will show different panels depending on the objects mode.

There are some options available to customize panels to your preference:

- You can change the position of a panel within its region by clicking and dragging it with the LMB on the little widget in the upper right corner.
- The zoom factor of a whole region with panels can be changed by `Ctrl+MMB` clicking and moving the mouse anywhere within that region.
- The alignment of the panels in the *Properties Editor* can be changed between vertical and horizontal. To do this click with RMB somewhere within the main region of the *Properties Editor* and choose either *Horizontal* or *Vertical* from the appearing menu. Keep in mind though that the panels are optimized for vertical alignment.

**Buttons and Controls** Buttons and other controls can be found in almost every [Window](#) of the Blender interface. The different types of controls are described below.

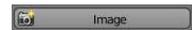


Fig. 1.19: Operation button

**Operation Buttons** These are buttons that perform an operation when clicked with LMB. They can be identified by their gray color in the default color scheme.

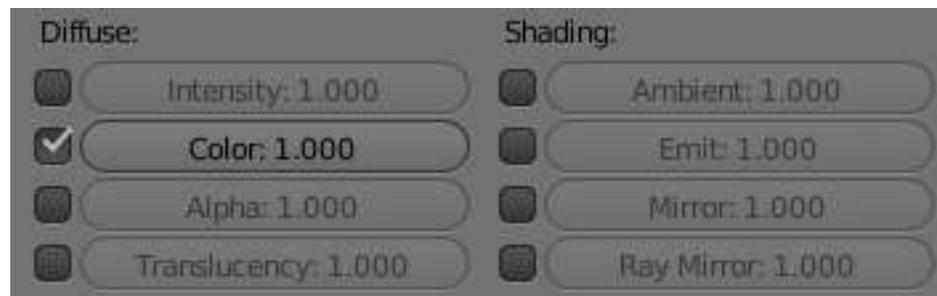


Fig. 1.20: Toggle buttons

**Toggle Buttons** Toggle buttons are typically displayed as check boxes. Clicking this type of button will toggle a state but will not perform any operation.

**Toggle Drag** To change many toggle buttons at once, you can LMB drag over multiple buttons. This works for check-boxes, toggles in the outliner and layer buttons.

---

**Note:** For layer buttons (a type of toggle button) it is often useful to hold `Shift` at the same time, to set or clear many layers at once.

---

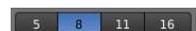


Fig. 1.21: Radio buttons

**Radio Buttons** Radio buttons are used to choose from a small selection of “mutually exclusive” options.

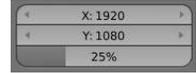


Fig. 1.22: Number buttons

**Number Buttons** Number buttons can be identified by their labels, which in most cases contains the name and a colon followed by a number. Number buttons can be edited in several ways:

**Incremental Steps** To change the value in steps, click LMB on the small triangles on the sides of the button.

**Dragging** To change the value in a wider range, hold down LMB and drag the mouse to the left or right.

**Text Input** Press LMB or Return to edit the value as a text field.

When entering values by hand, this button works like any other text button.

- Press Return to apply the change.
- Press Esc will cancel the value.

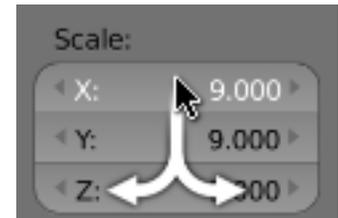


Fig. 1.23: Multi-value-editing

**Multi-Value Editing** It's often useful to edit multiple values at once (object scale or render resolution for example).

This can be done by clicking on the button and dragging vertically to include buttons above/below.

After the vertical motion you can drag from side to side, or release the LMB to type in a value.

**Expressions** You can also enter expressions such as  $3*2$  instead of 6. or  $5/10+3$ . Even constants like  $\pi$  (3.142) or functions like  $\text{sqrt}(2)$  (square root of 2) may be used.

*These expressions are evaluated by Python; for all available math expressions see: [math module reference](#)*

**Units** As well as expressions, you can mix units with numbers; for this to work, units need to be set in the scene settings (Metric or Imperial).

Examples of valid units include:

- 1cm
- 1m 3mm
- 1m, 3mm
- 2ft
- 3ft/0.5km
- 2.2mm + 5' / 3" - 2yards

*Note that the commas are optional. Notice how you can mix between metric and imperial even though the display can only show one at a time.*

**Unit Names** Unit names have can be used with both long and short forms, here are listed recognized unit names you can use. Plurals of the names are recognized too, so `meter` and `meters` can both be used.

Table 1.3: Imperial Units

Full Name	Short Name(s)	Scale of a Meter
thou	mil	0.0000254
inch	" , in	0.0254
foot, feet	' , ft	0.0254
yard	yd	0.9144
chain	ch	20.1168
furlong	fur	201.168
mile	mi, m	1609.344

Table 1.4: Metric Units

Full Name	Short Name(s)	Scale of a Meter
micrometer	um	0.000001
millimeter	mm	0.001
centimeter	cm	0.01
decimeter	dm	0.1
meter	m	1.0
dekameter	dam	10.0
hectometer	hm	100.0
kilometer	km	1000.0

**Menu Buttons** Blender uses a variety of different menus for accessing options, tools and selecting data-blocks.

### Menu Shortcuts

- Arrow keys can be used to navigate.
- Each menu item has an underlined character which can be pressed to activate it.
- Number keys or num-pad can be used to access menu items. (Where 1 is the first menu item, 2 the second... etc).
- Press `Return` to activate the selected manu item.
- Press `Esc` to cancel the menu.

**Header Menus** Header menus are used to configure the editor and access tools.

**Pop-Up Menus** Pop-up menus are displayed as regular buttons, showing a range of options. For example, the *Add Modifier* button will produce a menu with all of the available modifiers.

**Data-Block Menus** Menu buttons are used link data-blocks to each other. data-blocks are items like *Meshes*, *Objects*, *Materials*, *Textures*, and so on.

These menu's may show a preview and allow you to search by name since its common all items wont fit in the list.

Sometimes there is a list of applied data-blocks (such as a list of materials used on the object). See *data-block link buttons* above.

For details on the behavior of linking data see [data-block](#).

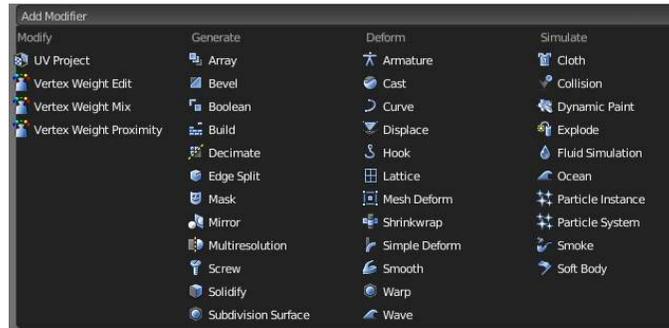


Fig. 1.24: Modifier options

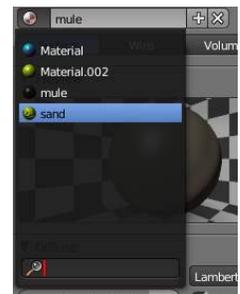


Fig. 1.25: Datablock link menu with search

- The first button (with an icon of the data-block type) opens up a menu to select an item by clicking LMB.
- The second button displays the name of the linked data-block which can be edited as a regular text field.
- The “+” button duplicates the current data-block and applies it.
- The “X” button clears the link.

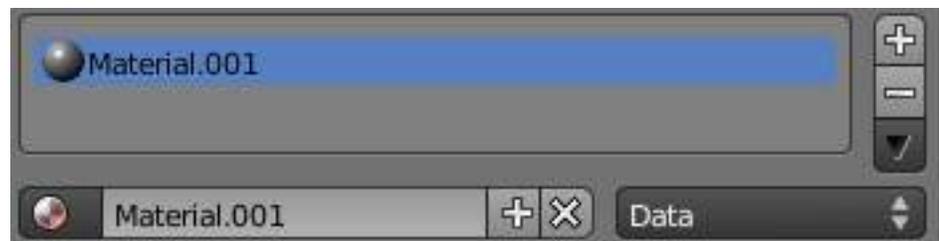


Fig. 1.26: Datablock link buttons

- To select a datablock, click LMB on it.
- To add a new section (e.g. material, or particle system), click LMB on the “+” button to the right of the list.
- To remove a section, click LMB on the “-” to the right of the list.

**Common Shortcuts** There are shortcuts shared between many button types.

**While Hovering** *When the cursor is held over a button*

- `Ctrl-C` - copy the value of the button.

---

**Note:** Pressing `Ctrl-C` over any *Operation Buttons* copies their Python command into the clipboard which can be used in the Python console or in the text editor when writing scripts.

---

- `Ctrl-V` - paste the value of the button.
- `RMB` - open the context menu.
- `Backspace` - clear the value (sets to zero or clears a text field).
- `Minus` - negate number values (multiply by  $-1.0$ ).
- `Ctrl-Wheel` - changes the value incremental steps.

For pop-up option menus buttons this cycles the value.

Animation:

- `I` - insert a keyframe.
- `Alt-I` - clear the keyframe.
- `Alt-Shift-I` - clear all keyframes.
- `D` - assign a driver.
- `Alt-D` - clear the driver.
- `K` - add a keying set.
- `Alt-K` - clear the keying-set.

**While Dragging Numbers**

- `Ctrl` - while dragging snap the discrete steps.
- `Shift` - gives finer control over the value.

**While Editing Text**

- `Home` - go to the start.
- `End` - go to the end.
- `Left`, `Right` - move the cursor a single character.
- `Ctrl-Left`, `Ctrl-Right` - move the cursor an entire word.
- `Backspace`, `Delete` - delete characters.
- `Ctrl-Backspace`, `Ctrl>Delete` - delete words.
- **Holding Shift** - while moving the cursor selects.
- `Ctrl-C` - copy the selected text.
- `Ctrl-V` - paste text at the cursor location.
- `Ctrl-A` - selects all text.

**All Modes**

- `Esc`, `RMB` - cancels.
- `Return`, `LMB` - confirms.

**Extended Controls** This page documents some of the more involved interface controls.

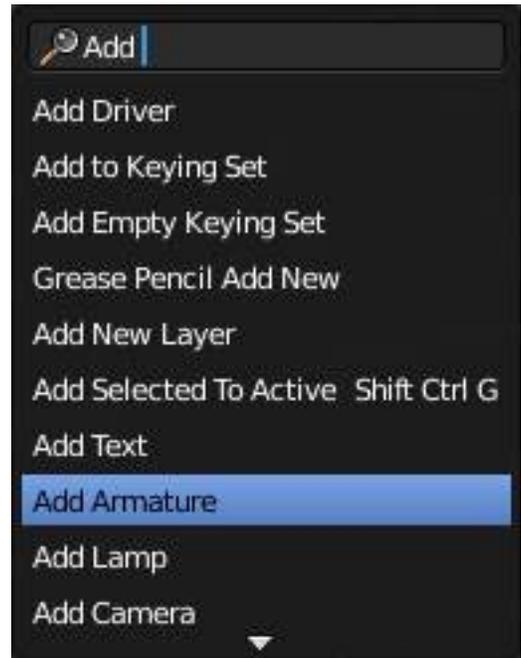


Fig. 1.27: The [Space]-menu

**Operator Search Menu** A menu with access to all *Blender* commands is available by pressing `Spacebar`. Simply start typing the name of the command you want to refine the list. When the list is sufficiently narrowed, `LMB` on the desired command or navigate with `Down` and `Up`, activate it by pressing `Return`.

**Color Picker** All of the Color picker types have the common *RGB*, *HSV* and *Hex* options to show values.

Blender uses `0 - 1.0` values to express colors for *RGB* and *HSV* values.

Some colors also define an alpha value (*A*), below the color sliders.

You can optionally use hexadecimal (*Hex*) values, expressed as `(RRGGBB)`, a common way to represent colors for HTML.

For more information about how to select the type of color picker, see: [System preferences page](#).

---

**Note:** Blender corrects Gamma by default

for more information about how to disable Gamma correction in Blender, see: [Color Management and Exposure page](#).

---

- Use `Wheel` to change overall brightness.
- Color sliders don't have a default value; the last value before any changes is used instead.

**Eye Dropper** The eye dropper allows you to sample from anywhere in the Blender window.

The eyedropper can be use to select different kinds of data.

**Color** This is the most common usage.

**Objects / Object-Data** This is used with object buttons such as parent, constraints or modifiers to select an object from the 3D view.



Fig. 1.28: Fig. 2 - Color Picker - Circle

**Circle (Default)** A full gamut of colors ranging from center to the borders is always shown; center is a mix of the colors. Brightness is adjusted with the right bar, from top to bottom. For operations that are capable of using Alpha, another slider is added at the bottom of the color picker. See Fig. 2 - Color Picker - Circle



Fig. 1.29: Fig. 3 - Color Picker Square (HS + V)

**Square (HS + V)** Hue, Saturation plus Value → A full gamut of colors is always shown. Brightness is subtracted from the base color chosen on the square of the color picker moving the slider to the left. For operations that are capable of using Alpha, another slider is added at the bottom of the color picker. See Fig. 3 - Color Picker - Square (HS + V)



Fig. 1.30: Fig. 4 - Color Picker Square (SV + H)

**Square (SV + H)** Saturation, Value plus Hue → A full Gamut of colors is always shown at the bar in the middle of the color picker. Colors are adjusted using the a range of brightness of the base color chosen at the color bar in the middle of the picker. For operations that are capable of using Alpha, another slider is added at the bottom of the color picker. See Fig. 4 - Color Picker - Square (SV + H)



Fig. 1.31: Fig. 5 - Color Picker Square (HV + S)

**Square (HV + S)** Hue, Value and Saturation →\* A full gamut of colors is always shown at the square of the color picker. Brightness is added to the base color chosen on the square of the color picker moving the slider to the left. For operations that are capable of using Alpha, another slider is added at the bottom of the color picker. See Fig. 5 - Color Picker - Square (HV + S)

**Camera Depth** Number buttons effecting distance can also use the eye-dropper, this is most useful for camera depth of field.

- E will activate the eye-dropper while hovering over a button.
- LMB dragging will mix the colors you drag over which can help when sampling noisy imagery.
- Spacebar resets and starts mixing the colors again.

### Color Ramp

*Color Ramps* enables the user to specify a range of colors based on color stops. Color stops are similar to a mark indicating where the exact chosen color should be. The interval from each of the color stops added to the ramp is a result of the color interpolation and chosen interpolation method. The available options for Color Ramps are:

**Add (Button)** Clicking on this button will add a stop to your custom weight paint map. The stops are added from the last selected stop to the next one, from left to right and they will be placed in the middle of both stops.

**Delete (Button)** Deletes the selected color stop from the list.

**'F' (Button)** Flips the color band, inverting the values of the custom weight paint range.

**Numeric Field** Whenever the user adds a color stop to the custom weight paint range, the color stop will receive an index. This field shows the indexes added (clicking in the arrows until the counter stops), and allows the user to select the color stop from the list. The selected color stop will be shown with a dashed line.

**Interpolation Options** Enables the user to choose from **4** types of calculations for the color interpolation for each color stop. Available options are:

**B-Spline** Uses a *B-Spline* Interpolation for the color stops.

**Cardinal** Uses a *Cardinal* Interpolation for the color stops.

**Linear** Uses a *Linear* Interpolation for the color stops.

**Ease** Uses a *Ease* Interpolation for the color stops.

**Constant** Uses a *Constant* Interpolation for the color stops.

**Position** This slider controls the positioning of the selected color stop in the range.

**Color Bar** Opens a color Picker for the user to specify color and Alpha for the selected color stop. When a color is using Alpha, the Color Bar is then divided in two, with the left side showing the base color and the right side showing the color with the alpha value.

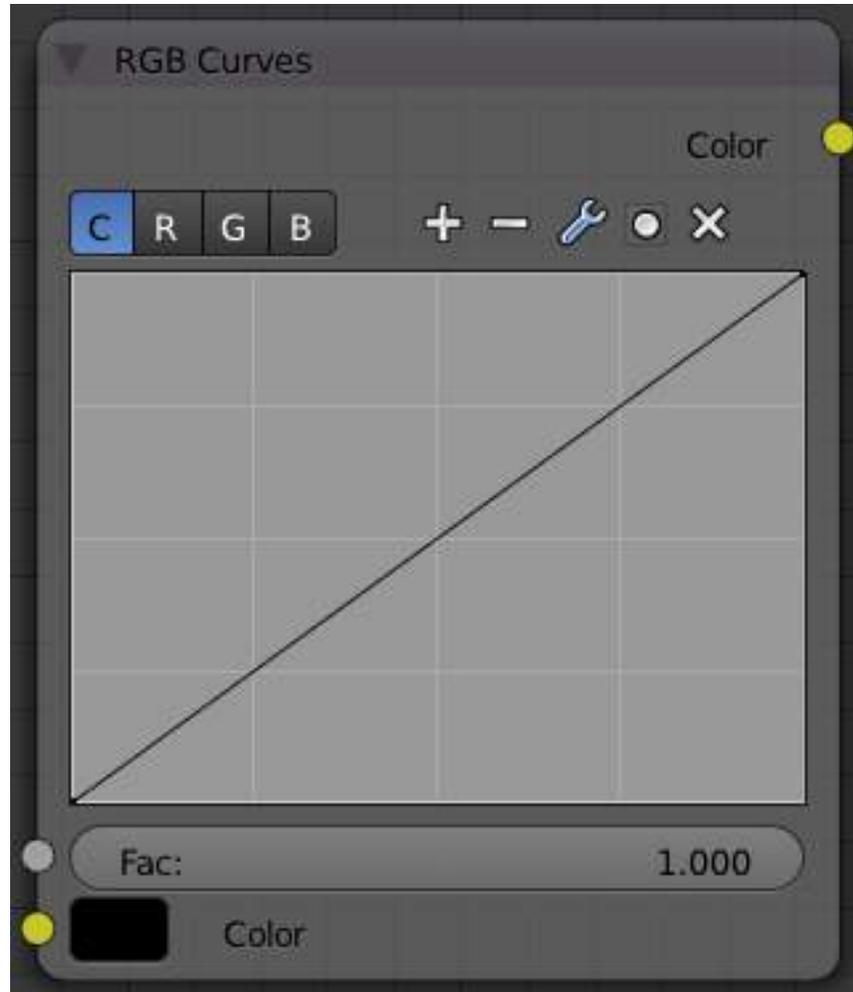


Fig. 1.36: RGB Curves node

**Curve Widget** The *Curve Widget* is found in several places throughout Blender, such as:

- RGB Curves node
- Vector Curves node
- Paint/Sculpt brush falloff
- Color Management curves

The purpose of the *Curve Widget* is to allow the user to modify an input (such as an image) in an intuitive manner by smoothly adjusting the values up and down using the curve.

The input values are mapped to the X-axis of the graph, and the Y-axis is mapped to the output values.

**Control Points** Like all curves in Blender, the curve of the *Curve Widget* is controlled using *control points*.

By default there are two control points: one at  $0.0, 0.0$  and one at  $1.0, 1.0$ , meaning the input is mapped directly to the output (unchanged).

**To move a control point** Simply click and drag it around.

**To add a new control point** Click anywhere on the curve where there is not already a control point.

**To remove a control point** select it and click the  button at the top right.

**Controls** Above the curve graph is a row of controls. These are:



Fig. 1.37: Node curve controls

**Channel selector** Allows to select appropriate curve channel.



Fig. 1.38: Curve channel selector

**Zoom In** Zoom into the center of the graph to show more details and provide more accurate control. To navigate around the curve while zoomed in, click and drag in an empty part of the graph.

**Zoom Out** Zoom out of the graph to show less details and view the graph as a whole. You cannot zoom out further than the clipping borders (see *Clipping* below).

**Tools**



Fig. 1.39: Advanced tools for curve

**Reset View** Resets view of the curve.

**Vector Handle** Vector type of curve point's handle.

**Auto Handle** Automatic type of curve point's handle.

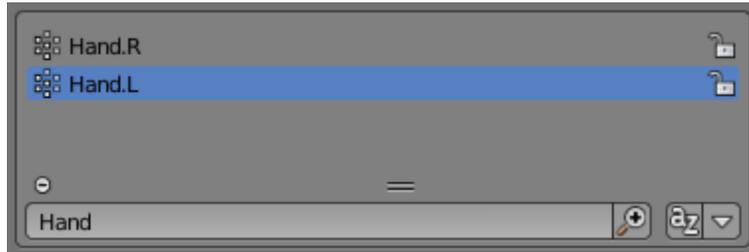
**Extend Horizontal** Extends the curve horizontal.

**Extend Extrapolated** Extends the curve extrapolated.

**Reset Curve** Resets the curve in default (removes all added curve's points).

**Clipping** Enable/disable clipping and set the values to clip to.

**Delete** Remove the selected control point.



**List View** At the bottom of a list view (like the ones found in the object data properties) there are controls for filtering and sorting and resizing.

**Rename** By pressing (Ctrl, LMB) over an item's name, you can edit the text-field. This can also be achieved by double clicking.

**Resize** The list view can be resized to show more or less items. Hover the mouse over the handle then click and drag the handle to expand or shrink the list.

**Filter** Click the *Show filtering options* button to toggle filter option buttons.

Type part of a list item's name in the filter text box to filter items by part of their name.

**Filter Include** When the magnifying glass icon has a + sign then only items that match the text will be displayed.

**Filter Exclude** When the magnifying glass icon has a – sign then only items that do not match text will be displayed.

**Sort** Sort list items.

**Alphabetical** This button switches between alphabetical and non-alphabetical ordering.

**Inverse** Sort objects in ascending or descending order. This also applies to alphabetical sorting, if selected.

## Navigating

### Introduction

The *3D View* is where you perform most of the object modeling and scene creation. Blender has a wide array of tools and options to support you in efficiently working with your mouse, keyboard and numpad.

**3D Window Header** The *3D View* window is comprised of a workspace and a header. The header is shown at the bottom or top of the workspace, and can be hidden if desired. The header shows you a menu and the current mode, as explained below.

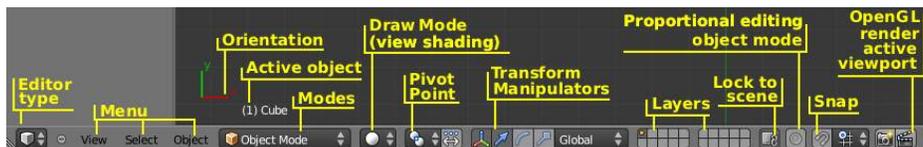


Fig. 1.40: 3D View header.

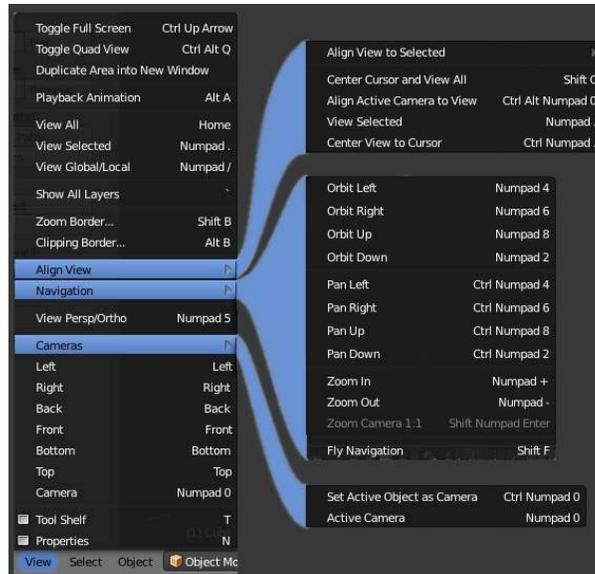


Fig. 1.41: The View menu.

## View Menu

**Properties Panel** Toggles the *Properties* side panel (N), which allows you to tweak many 3D view settings:

- Transform
- Grease Pencil
- View
- Item
- Display
- Background Images
- Transform Orientations

**Tool Shelf** Toggles the *Tool Shelf* (T), which appears on the left side of the 3d view, and allows you to perform various operations, depending on the type of object selected, and the mode you are in.

**Camera (Numpad0)** Switches the view to the current camera view.

**Viewing angles:** These commands change the view to the default Top/Bottom, Front/Back, or Left/Right views.

- Top (Numpad7)
- Bottom (Ctrl-Numpad7)
- Front (Numpad1)
- Back (Ctrl-Numpad1)
- Right (Numpad3)
- Left (Ctrl-Numpad3)

**Cameras Menu:** *Set Active object as camera* *Active camera*

**Perspective/Orthographic View (Numpad5)** These commands change the projection of the 3D view

**Navigation Menu** This sub-menu contains commands for rotating and panning the view. Using these commands through the menu is not that efficient. However, like all Blender menus, the much more convenient keyboard shortcuts are listed next to the commands.

**Align View** This submenu allows you to align the 3D view in certain ways.

- *Align to selected*
- *Center cursor and view all*
- *Align active camera to view*
- *View Selected*
- *Center View to cursor*

**Clipping Border...** (**Alt-B**) Allows you to define a clipping border to limit the 3D view display to a portion of 3D space.

**Zoom Border...** (**Shift-B**) Allows you to define the area you want to zoom into.

**Show all Layers** (**~**) Makes all of the display layers visible.

**Global View/Local View** (**NumpadSlash**) Global view shows all of the 3D objects in the scene. Local view only displays the selected objects. This helps if there are many objects in the scene, that may be in the way. Accidentally pressing **NumpadSlash** can happen rather often if you're new to Blender, so if a bunch of the objects in your scene seem to have mysteriously vanished, try turning off local view.

**View Selected** (**NumpadPeriod**)

**Zooms the 3D view to encompass all the selected objects.** Read more about [Zooming the 3D View](#)

**View All** (**Home**) Zooms the 3D view to encompass *all* the objects in the current scene.

**Play Back Animation** (**Alt-A**) Plays back the animation from the current frame.

**Duplicate area in new window** Clones the current 3D view in a new window

**Quad View** Toggles a four pane 3D view, each showing a different angle of the scene.

**Toggle Full Screen** (**Ctrl-Up**) Maximizes the *3D View* window to fill the full screen area.

**Select Menu** This menu contains tools for selecting objects.

[Read more about Selecting](#)

**Object Menu** This menu appears when in Object Mode. In edit mode, it will change to the appropriate menu with editing tools.

[Read more about Objects](#)

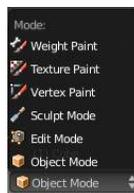


Fig. 1.42: The Mode drop-down list.

**Mode List** Blender has several modes of operation.

**Object Mode** mode allows you to work with objects as a whole.

**Edit Mode** Allows you to modify the shape of the object.

**Sculpt mode** `</modeling/meshes/editing/sculpt_mode>` In this mode your cursor becomes a tool to shape the object

The cursor becomes a brush in:

- [Vertex Paint](#) mode
- [Weight Paint](#) mode
- [Texture Paint](#) mode.

**ViewPort Shading List** Allows you to change the way 3D objects are displayed in the viewport.

- Bounding Box
- Wireframe
- Solid
- Texture
- Material
- Rendered

Read more about 3D view options

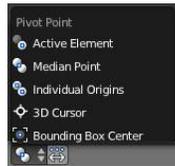


Fig. 1.43: Pivot point selector.

**Pivot Point Selector** When rotating or scaling an object or group of vertices/edges/faces, you may want to shift the pivot point (the transformation center) in 3D space. Using this selector, you can change the pivot point to the location of the:

- Active Element
- Median Point *the average center spot of the selected items*
- Individual Origins
- 3D Cursor
- Bounding Box Center

Use the *Object Center* to switch between transforming the entire objects, or just the position of the objects

[Read more about Pivot Points](#)

**Transform (Manipulator) Selectors** These handy selectors allow you to rotate or move objects by grabbing (clicking with your mouse) their controls and moving your mouse in the axis.

[Read more about Transform Manipulators](#)

**Layer Selector** Layers are well documented in the [Layers](#) page. Toggling layer visibility is covered in the section on [viewing layers](#) and [moving objects between layers](#) is also discussed in this page.

**Lock to Scene** By default, the “lock” button to the right of the layer buttons is enabled. This means that in this view, the active layers and camera are those of the whole scene (and those used at render time). Hence, all 3D views locked this way will share the same active layers and camera - when you change them in one view, all locked others will immediately reflect these changes.

But if you disable this “lock” button, you then can specify different active layers and camera, specific to this view. This might be useful if you don’t want to have your working areas (views) cluttered with the whole scene, and still have an ancillary complete view (which is unlocked with e.g. all layers shown). Or to have several views with different active cameras. Remember that you can use (Ctrl-Numpad0) to make the active object the active camera.

[Read more about Scenes](#)

**Snap to Mesh** This “magnet” button controls the snapping tools that help with transforming and modeling objects.

[Read more about Snapping](#)

**Render Buttons** The Render Buttons render an OpenGL version of the 3D view.

The first button renders a still image of the Objects in the 3D view without displaying the grid, axes, etc. It uses the same *Draw* mode as the 3D view, so it’s rather useful if someone asks to see the wireframe of an Object you’re working on.

The second button will render an animation of the 3D View, making it useful for making preview renders of animations. The animation will be saved in the folder and format specified in the *Output* panel of the *Render* context.

## 3D View

To be able to work in the three dimensional space that Blender uses, you must be able to change your viewpoint as well as the viewing direction of the scene. While we will describe the *3D View* window, most of the other windows have similar functions. For example, it is possible to translate and zoom a *Buttons* window and its panels.

---

**Tip:** Mouse Buttons and Numpad

If you have a mouse with less than three buttons or a keyboard without numpad, see the [Keyboard and Mouse](#) page of the manual to learn how to use them with Blender.

---

## Perspective and Orthographic Views Reference

Mode: All modes

Menu: *View* → *Perspective* / *View* → *Orthographic*

Hotkey: Numpad5

---

**Description** Each 3D viewport supports two different types of projection. These are demonstrated in the *Orthographic (left) and perspective (right) projections* image below.

Our eye is used to perspective viewing because distant objects appear smaller. Orthographic projection often seems a bit odd at first, because objects stay the same size regardless of their distance. It is like viewing the scene from an infinitely distant

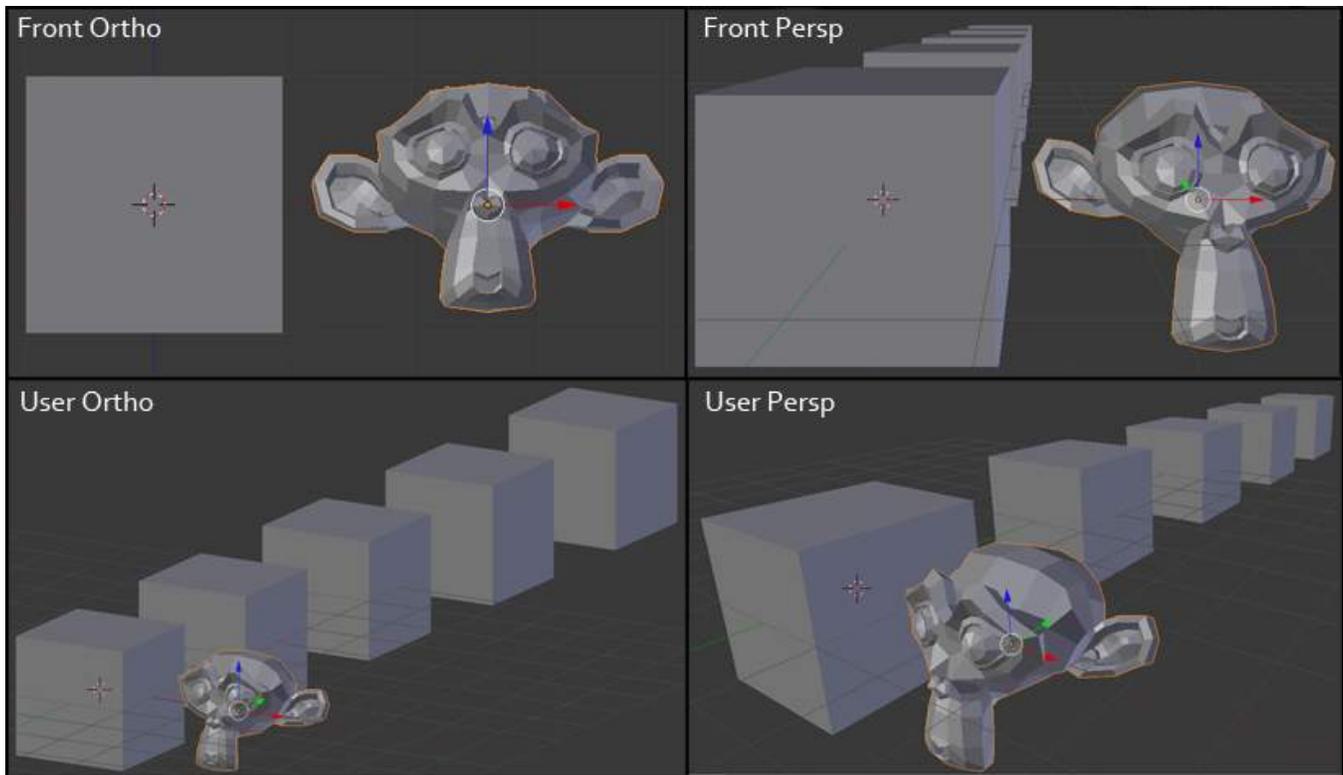


Fig. 1.44: Orthographic (left) and perspective (right) projections.

point. Nevertheless, orthographic viewing is very useful (it is the default in Blender and most other 3D applications), because it provides a more “technical” insight into the scene, making it easier to draw and judge proportions.

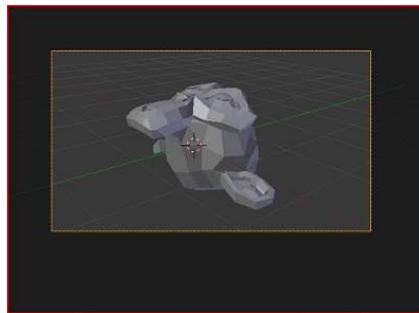


Fig. 1.45: Demonstration of camera view.

**Options** To change the projection for a 3D view, choose the *View* → *Orthographic* or the *View* → *Perspective* menu entry. The Numpad5 shortcut toggles between the two modes. Changing the projection for a 3D view does not affect the way the scene will be rendered. Rendering is in perspective by default. If you need to create an orthographic rendering, select the camera, go to the *Object Data* context and press the *Orthographic* button in the *Lens* panel.

The *View* → *Camera* menu entry sets the 3D view to camera mode (Numpad0). The scene is then displayed as it will be rendered later (see *Demonstration of camera view*). The rendered image will contain everything within the orange dotted line. Zooming in and out is possible in this view, but to change the viewpoint, you have to move or rotate the camera.

If you have a large scene, viewing it through Camera View may not display all of the Objects in the scene. One possibility may

be that the `clipping` distance of the camera is too low. The camera will only show objects that fall within the clipping range.

[Read more about Render perspectives](#)

[Read more about Camera View](#)

[Read more about Camera clipping](#)

## Technical Details

**Perspective definition** A *perspective* view is geometrically constructed by taking a scene in 3D and placing an observer at point  $O$ . The 2D perspective scene is built by placing a plane (e.g. a sheet of paper) where the 2D scene is to be drawn in front of point  $O$ , perpendicular to the viewing direction. For each point  $P$  in the 3D scene a  $PO$  line is drawn, passing by  $O$  and  $P$ . The intersection point  $S$  between this  $PO$  line and the plane is the perspective projection of that point. By projecting all points  $P$  of the scene you get a perspective view.

**Orthographic definition** In an *orthographic* projection, you have a viewing direction but not a viewing point  $O$ . The line is then drawn through point  $P$  so that it is parallel to the viewing direction. The intersection  $S$  between the line and the plane is the orthographic projection of the point  $P$ . By projecting all points  $P$  of the scene you get the orthographic view.

## Rotating the View

### Reference

Mode: All modes

Menu: *View* → *Navigation*

Hotkey: MMB / Numpad2 / Numpad4 / Numpad6 / Numpad8 / Ctrl-Alt-Wheel

---

**Description** Blender provides four default viewing directions: *Side*, *Front*, *Top* and *Camera* view. Blender uses a right-angled “Cartesian” coordinate system with the Z axis pointing upwards. “Side” corresponds to looking along the X axis, in the negative direction, “Front” along the Y axis, and “top” along the Z axis. The *Camera* view shows the current scene as seen from the camera view point.

**Options** You can select the viewing direction for a 3D viewport with the *View* menu entries, or by pressing the hotkeys Numpad3 for “side”, Numpad1 for “front”, Numpad7 for “top”. You can select the opposite directions if you hold `Ctrl` while using the same numpad shortcuts. Finally Numpad0 gives access to the “camera” viewpoint.

Apart from these four default directions, the view can be rotated to any angle you wish. Click and drag MMB on the viewport’s area. If you start in the middle of the window and move up and down or left and right, the view is rotated around the middle of the window. Alternatively, if the *Emulate 3 button mouse* option is select in the *User Preferences* you can press and hold `Alt` while dragging LMB in the viewport’s area.

To change the viewing angle in discrete steps, use Numpad8 and Numpad2 (which correspond to vertical MMB dragging, from any viewpoint), or use Numpad4 and Numpad6 (or `Ctrl-Alt-Wheel`) to rotate the scene around the Z global axis from your current point of view.

### Note: Hotkeys

Remember that most hotkeys affect **the active window** (the one that has focus), so check that the mouse cursor is in the area you want to work in before your use the hotkeys.

---

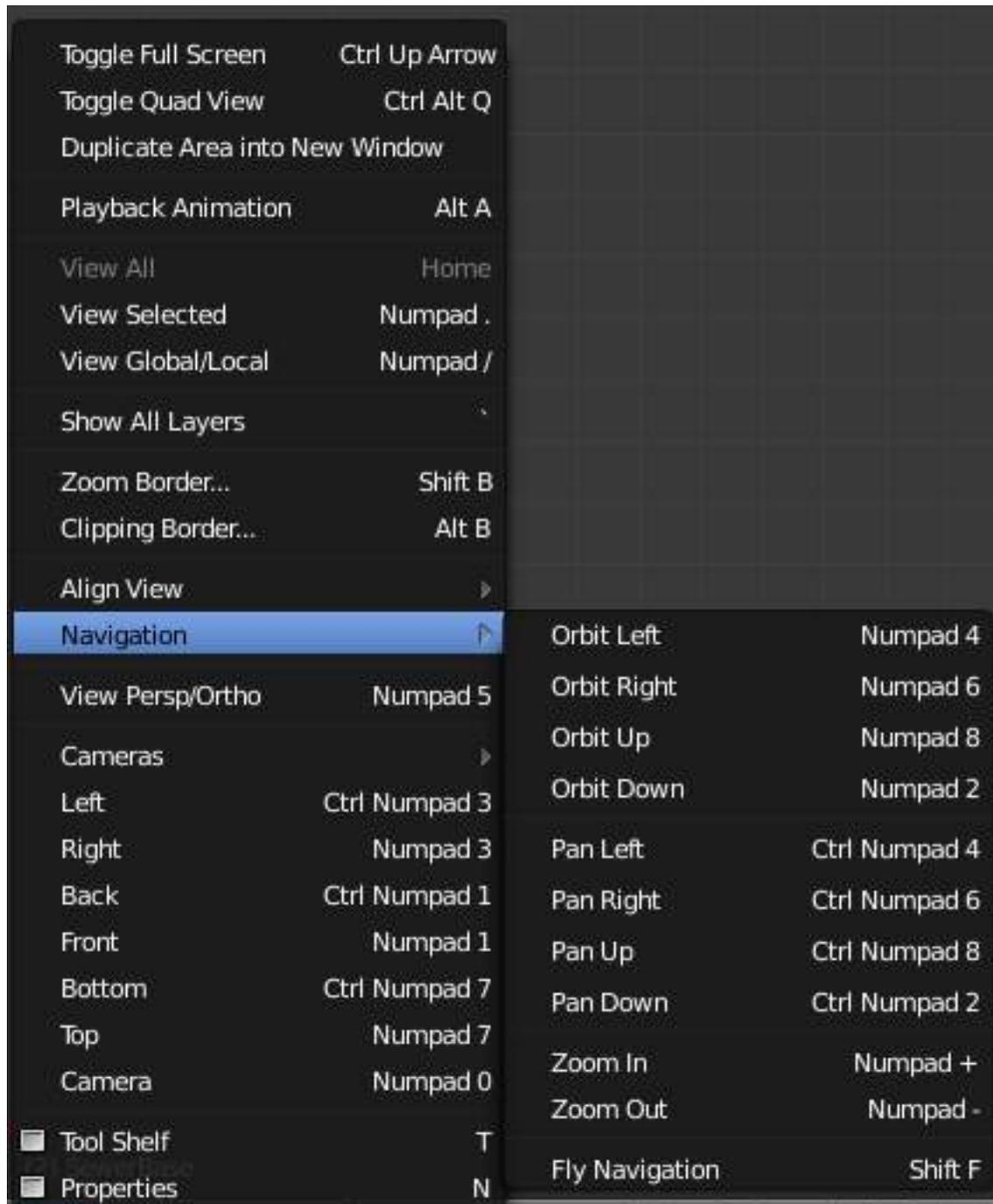


Fig. 1.46: A 3D viewport's View menu.

**TrackBall/Turntable** By default, when you rotate the view as described above, you are using the *turntable* method. For some users this is intuitive and for others it is not. If you feel you are having difficulties with this style of 3D window rotation you can switch to the *trackball* style. With the trackball style you are rotating the scene as though you are rolling your hand across a *trackball*.

The *turntable* style is fashioned more like a record player where you have two axes of rotation available, and the world seems to have a better definition of what is “Up” and “Down” in it. The downside to using the *Turntable* style is that you lose some flexibility when working with your objects. However, you gain the sense of “Up” and “Down” which can help if you are feeling disoriented. Of course you can always switch between the styles depending on what you are working on.



Fig. 1.47: View rotation.

To change the rotation “style”, use the [User Preferences window](#). Click on the *Input* button and you will see an option for choosing the Orbit style. There are two additional checkboxes for controlling the display in the 3D window in the *Interface* tab in the *User Preferences*. *Auto Perspective* will automatically switch to perspective whenever the view is rotated using MMB. *Rotate Around Selection* will rotate the view around the center of the current selection. If there is no selection at that moment (e.g. if you used A to deselect everything), the last selection will be used anyway.

---

## Panning the View

### Reference

Mode: All modes

Menu: *View* → *Navigation*

Hotkey:

Shift-MMB / Ctrl-Numpad2 / Ctrl-Numpad4 /

Ctrl-Numpad6 / Ctrl-Numpad8 / Shift-Alt-LMB

---

**Description** To pan the view, hold down Shift and drag MMB in the 3D Viewport. For discrete steps, use the hotkeys Ctrl-Numpad8, Ctrl-Numpad2, Ctrl-Numpad4 and Ctrl-Numpad6 as with rotating (note: you can replace Ctrl by Shift). For those without a middle mouse button, you can hold Shift Alt while dragging with LMB.

---

## Zooming the View

### Reference

Mode: All modes

Menu: *View* → *Navigation*

Hotkey: Ctrl-MMB / Wheel / NumpadPlus / NumpadMinus

**Description** You can zoom in and out by holding down `Ctrl` and dragging `MMB`. The hotkeys are `NumpadPlus` and `NumpadMinus`. The *View* → *Navigation* sub-menu holds these functions too as well. Refer to the 3D viewport's *View* menu image above for more information.

If you have a wheel mouse, you can perform all of the actions in the 3D viewport that you would do with `NumpadPlus` and `NumpadMinus` by rotating the `Wheel`. To zoom a *Buttons* window, hold `Ctrl-MMB` and move your mouse up and down.

---

**Note:** If You Get Lost

If you get lost in 3D space, which is not uncommon, two hotkeys will help you: `Home` changes the view so that you can see all objects (*View* → *View All* menu entry), while `NumpadPeriod` zooms the view to the currently selected objects when in perspective mode (*View* → *View Selected* menu entry).

---

**Zoom Border** The *Zoom Border* tool allows you to specify a rectangular region and zoom in so that the region fills the 3d view.

You can access this through the *View* menu, or the shortcut `Shift-B`, then `LMB` click and drag a rectangle to zoom into.

Alternatively you can zoom out using the `MMB`.

---

**Dolly the View**  
**Reference**

Mode: All modes

Hotkey: `Ctrl-Shift-MMB`

---

**Description** In most cases its sufficient to zoom the view to get a closer look at something, however you may notice that at a certain point you cannot zoom any closer.

This is because Blender stores a view-point thats used for orbiting and zooming, This works well in many cases but sometimes you want to move the view-point to a different place - This is what Dolly supports, allowing you to transport the view from one place to another.

You can dolly back and fourth by holding down `Ctrl-Shift` and dragging `MMB`.

**Aligning the View**

**Align View** These options allow you to align and orient the view in different ways. They are found in the *View Menu*

**Align View to Selected menu** These options align your view with specified local axes of the selected object, bone or in *Edit* mode, with the normal of the selected face.

Hold down `Shift` while using the numpad to set the view axis.

**Center Cursor and View All (`Shift-C`)** moves the cursor back to the origin **and** zooms in/out so that you can see everything in your scene.

**Align Active Camera to View, `Ctrl-Alt-Numpad0`** Gives your active camera the current viewpoint

**View selected, `NumpadPeriod`** Focuses view on currently selected object/s by centering them in the viewport, and zooming in until they fill the screen.

**Center view to cursor, `Alt-Home`** Centers view to 3D-cursor

**View Selected** See above

**View All Home** Frames all the objects in the scene, so they are visible in the viewport.

**Local and Global View** You can toggle between *Local* and *Global* view by selecting the option from the *View Menu* or using the shortcut `NumpadSlash`. Local view isolates the selected object or objects, so that they are the only ones visible in the viewport. This is useful for working on objects that are obscured by other ones, or have heavy geometry. Press `NumpadSlash` to return to *Global View*.

## Quad View Reference

Mode: All modes

Menu: *View* → *Toggle Quad View*

Hotkey: `Ctrl-Alt-Q`

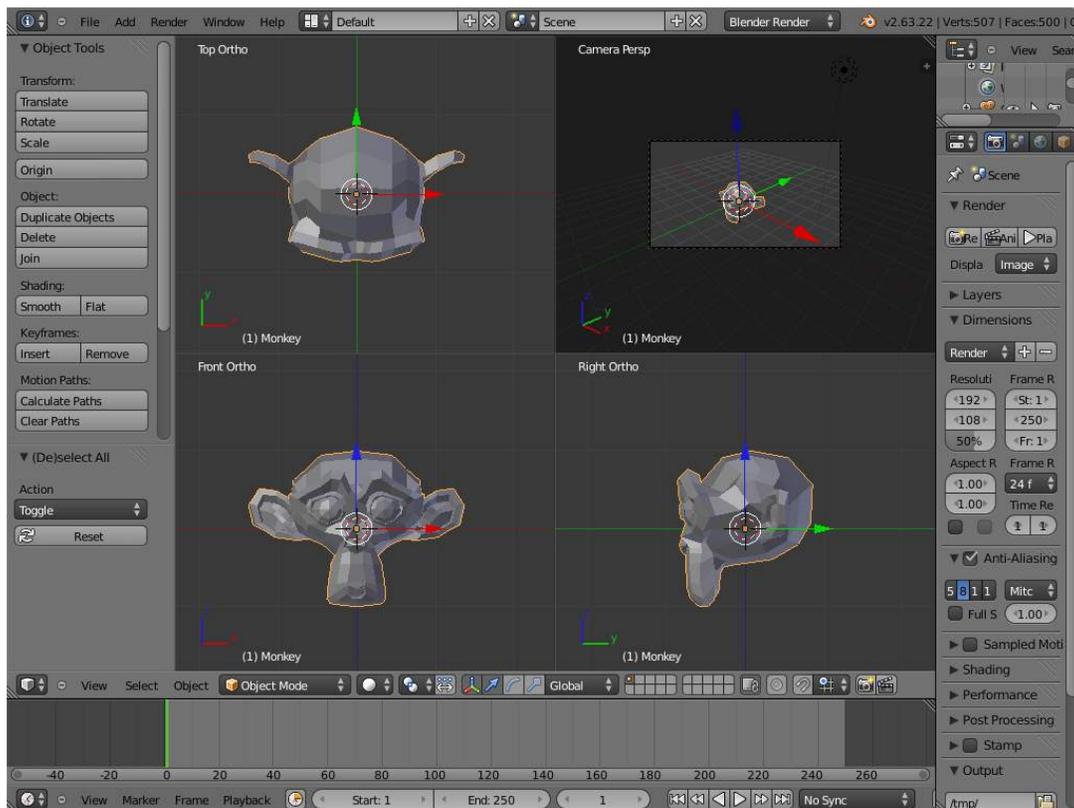


Fig. 1.48: Quad View

Toggling Quad View will split the 3D window into 4 views: 3 *Ortho* views and a *Camera / User View*. This view will allow you to instantly see your model from a number of view points. In this arrangement, you can zoom and pan each view independently but you cannot rotate the view. Note that this is different from splitting the windows and aligning the view manually. In Quad View, the four views are still part of a single 3D window. So they share the same draw options and layers.

If you want to be able to rotate each view, you can un-check the *Locked* option.

However in sometimes its preferable to split the view, so each can have its own configuration.

[Read more about splitting windows](#)

## View Clipping Border

### Reference

Mode: All modes

Menu: *View* → *Set Clipping Border*

Hotkey: `Alt-B`

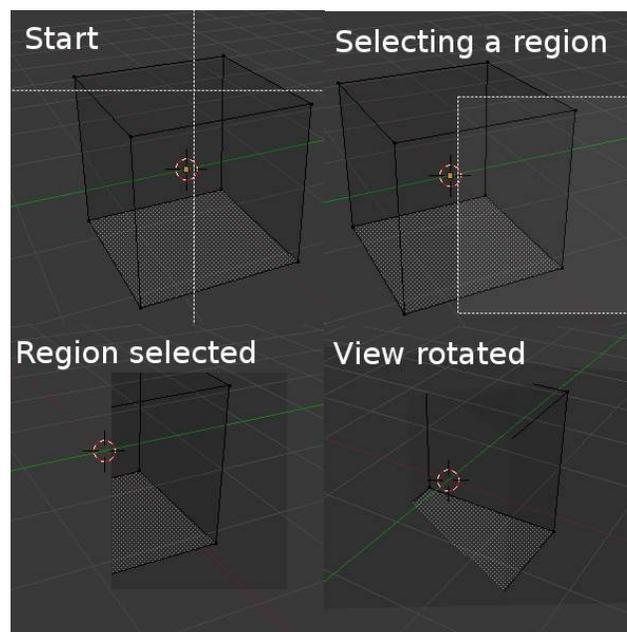


Fig. 1.49: Region/Volume clipping.

**Description** To assist in the process of working with complex models and scenes, you can set the view clipping to visually isolate what you're working on.

Once clipping is used, you will only see what's inside a volume you've defined. Tools such as paint, sculpt, selection, transform-snapping etc. will also ignore geometry outside the clipping bounds.

Once activated with `Alt-B`, you have to draw a rectangle with the mouse, in the wanted 3D view. The created clipping volume will then be:

- A right-angled **parallelepiped** (of infinite length) if your view is orthographic.
- A rectangular-based pyramid (of infinite height) if your view is in perspective.

To delete this clipping, press `Alt-B` again.

**Example** The *Region/Volume clipping* image shows an example of using the clipping tool with a cube. Start by activating the tool with `Alt-B` (upper left of the image). This will generate a dashed cross-hair cursor. Click with the LMB and drag out a rectangular region shown in the upper right. Now a region is defined and clipping is applied against that region in 3D space.

Notice that part of the cube is now invisible or clipped. Use the `MMB` to rotate the view and you will see that only what is inside the pyramidal volume is visible. All the editing tools still function as normal but only within the pyramidal clipping volume.

The dark gray area is the clipping volume itself. Once clipping is deactivated with another `Alt-B`, all of 3D space will become visible again.

---

## View Navigation

### Reference

Mode: All modes

Hotkey: `Shift-F`

---

**Description** When you have to place the view, normally you do as described above.

However, there are cases in which you really prefer to just navigate your model, especially if it's very large, like environments or some architectural model. In these cases viewing the model in perspective mode has limitations, for example after zooming a lot of panning is extremely uncomfortable and difficult, or you apparently cannot move the camera any nearer. As an example, try to navigate to a very distant object in the view with traditional methods (explained above) and see what you can get.

With `Walk` mode and `Fly` mode you move, pan and tilt, and dolly the camera around without any of those limitations.

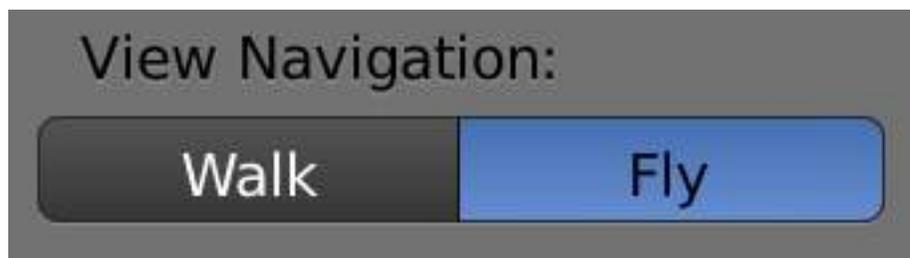


Fig. 1.50: View Navigation.

In the [User Preferences window](#) select the navigation mode you want to use as default when invoking the View Navigation operator. Alternatively you can call the individual modes from the View Navigation menu.

## Camera View

---

### Reference

Mode: All modes

Menu: *View* → *Camera* → *Active Camera*

Hotkey: `Numpad0`

---

Cameras View can be used to virtually compose shots and preview how the scene will look when rendered. Pressing `Numpad0` will show the scene as viewed from the currently active camera. In this view you can also set the *Render Border* which defines the portion of the camera view to be rendered.

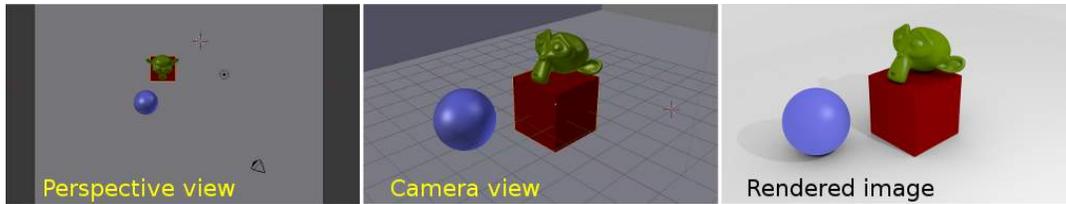


Fig. 1.51: Camera view provides a preview for the final rendered image.

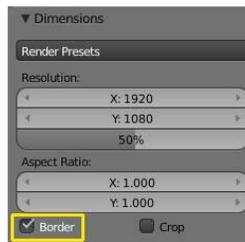


Fig. 1.52: Render Border toggle

**Render Border** While in camera view, you can define a *Render Border* by pressing `Ctrl-B`. This will allow you to draw out a dotted orange rectangle within the camera view. Your renders will now be limited to the part of scene visible within the render border. This can be very useful for reducing render times. The border can be disabled by disabling the *Border* option in the *Dimensions* panel of the *Render* context or by using `Ctrl-B` to set a *Render Border* larger than the camera view.

**Note:** Anti-Aliasing and blur options with borders

Note that when Render Borders are activated, Full Sampling Anti-Aliasing will be disabled while Sampled Motion Blur will become available.

[Read more about Anti-Aliasing](#) [Read more about Motion Blur](#)

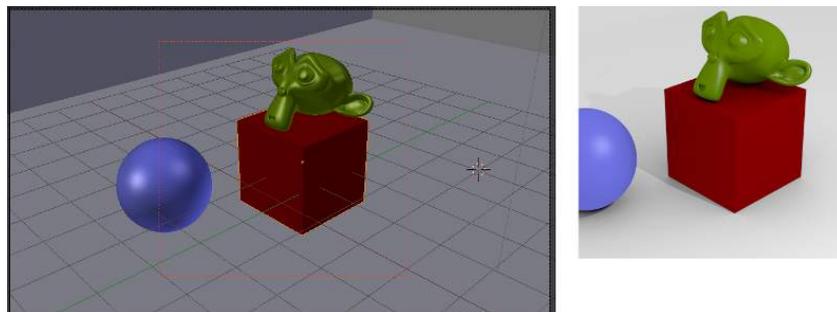


Fig. 1.53: Render border and associated render.

[Read more about Render Output options](#)

[Read more about Cameras](#)

## Transformations

### Introduction

Transformations refer to a number of operations that can be performed on a selected Object or Mesh that alters its position or characteristics.

Basic transformations include:

- Grabbing (moving)
- Rotating
- Scaling

More advanced transformations such as mirroring, shearing and warping can be found in the Advanced Transformations section.

### Grab/Move

#### Reference

Mode: *Object Mode*, *Edit Mode*, and *Pose Mode* for the 3D View; *UV/Image Editor Tools*, *Sequence Editor*, *Dopesheet*, and *Graph Editor* for other specific types of Grab/Move operation.

Menu: Context Sensitive, Object Based → *Transform* → *Grab/Move*

Hotkey: G or combinations for specific Axis constraint

In Object Mode, the grab/move option lets you translate (move) objects. It also lets you translate any elements that make up the object within the 3D space of the active 3D viewport. Grab/Move works similarly here as it does in the Node Editor, Graph Editor, UV Editor, Sequencer, etc.

Options and other details will be discussed in their respective sections.

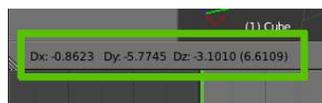


Fig. 1.54: Translation Display

While Grab/Move is active, the amount of change in the X, Y and Z co-ordinates is displayed at the bottom left corner of the 3D View window.

**3D View** There are 2 ways to Grab/Move in 3D View:

- Using shortcuts and combinations of shortcuts.
- Using the *Transform Widget* helper. This can be toggled from the *Translation Widget* in the header of the 3DView.

**Transform Widget** In the default installation of Blender, this is the *Transform Widget*. It is active by default. You can use the widget by holding LMB over it and dragging in the 3D view.

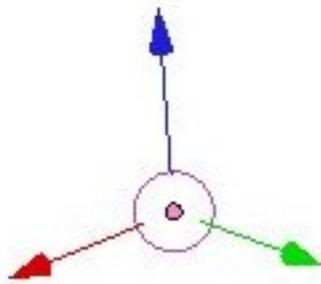


Fig. 1.55: Translation Widget

**Shortcuts in the 3D View** A quicker way to move objects in 3D space is with the **G** hotkey. Pressing **G** activates “Grab/Move” transformation mode. The selected object or data then moves freely according to the mouse pointer’s location and camera. Using this shortcut in combination with specific shortcuts which specify a chosen axis gives you full control over your transformation.

**LMB** Confirm the move, and leave the object or data at its current location on the screen.

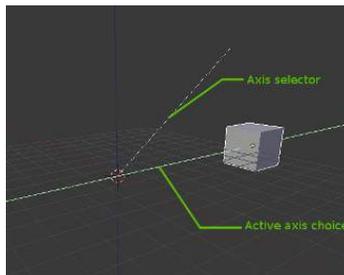


Fig. 1.56: Axis-Constraint in action

**MMB** Constrain the move to the X, Y or Z axis according to the position of the mouse pointer in the 3D View. After pressing the **G** key, if the **MMB** is pressed, a visual option to constrain the translation will be available, showing the three axes in the 3D View space. The axis of choice to confirm the operation will depend on the axis about which the **MMB** is released. At any point during the operation, the chosen axis can be changed by pressing X, Y, Z on the keyboard.

**RMB or Esc** Cancel the move, and return the object or data to its original location.

**Shift + X/Y/Z** This modifying hotkey locks the translation axis, allowing the object to move freely on the two axes that aren’t locked. For example, **Shift + X** means the object will translate on the Y and Z axes while remaining at the same point on the X axis.

**Alt + G** clears any previous transformation on the object and sets its origin back to the center. This only works in Object Mode.

You can also move an object by clicking and holding **RMB** on the object to move it. To confirm the action, press **LMB**.

**Note:** This behavior can be changed using *Release Confirms* in the [User Preferences](#), so that a single **RMB** drag can be used to move and confirm.

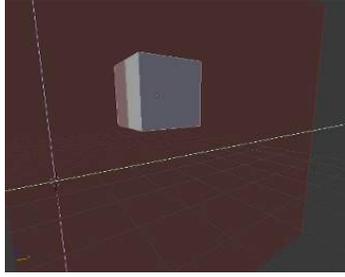


Fig. 1.57: Shift+X in action

**Controlling Grab/Move Precision** In addition to the Axis constraint options listed above, Blender offers options to limit the amount of the transformation in small or predefined steps.

**Shift** Slow translation mode. While still in the grab mode i.e. after `G` is pressed, holding down `Shift` reduces how quickly the object moves and allows extra precision.

**Ctrl** This activates [snapping](#) based on the snapping constraint which has been already set. You may not be able to enable every snapping option in all cases.

**Ctrl + Shift** Precise snap. This option will move the object with high precision along with the snapping constraint.

**X/Y/Z + decimal number** This option limits the transformation to the specified axis and the decimal number specified will be the magnitude of the translation along that axis. This decimal number is displayed at the bottom left corner of the 3D view window as it is entered.

- Hitting `Backspace` during number entry and deleting the number removes the numerical specification option but the object will remain constrained to the same axis.
- Hitting `/` during number entry switches the number being entered to its reciprocal, e.g. `2 /` results in `0.5 (1/2)`, `2 / 0` results in `0.05 (1/20)`.
- The axis of movement can be changed at any time during translation by typing `X/Y/Z`.

**Orientations** There are 5 standard orientation references for all transformations. You can find out more about transform orientations [here](#).

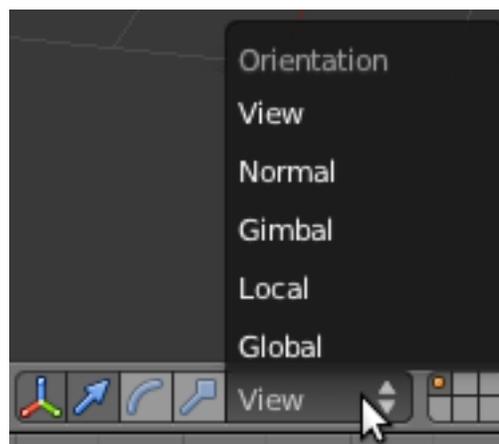


Fig. 1.58: Orientation choice menu

- Global (the default)

- Local
- Normal
- Gimbal
- View

Each mode is a co-ordinate system in which transformations can be carried out. These orientations can be chosen from the pop-up menu to the side of the controls which toggle and select the transformation manipulator widgets.

If you have changed the orientation to something other than Global, you can hotkey your chosen axis of orientation by hitting the relevant axis modifying hotkey **twice** instead of just once. Hitting the axis modifying hotkey three times reverts back to Global orientation.

- The G hotkey followed by `xx` or `yy` or `zz` allows you to translate the object in the object's Local axis by default, or on an axis of the selected orientation if the transform orientation is not set to Global. This modifying hotkey combination can be followed with numbers as described in the previous section.
- The G hotkey followed by `Shift` and `xx` or `yy` or `zz` will lock the object's translation on a single Local axis by default, or on an axis of the selected orientation if the transform orientation is not set to Global. Locking one axis means the selected object moves freely on the other two axes.

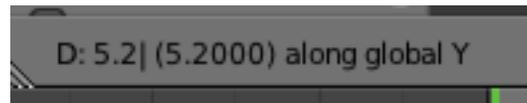


Fig. 1.59: Numerical Entry Display

**Other Editor Windows** In other editors such as the UV/Image Editor, Sequence Editor, Dopesheet and Graph Editor, the Grab/Move Operations are used to move objects or elements - the difference from 3D View is that only two axes are used - usually X and Y. You can use many of the same Grab/Move hotkeys after G (such as `Shift` or X) in other editor windows and they will work much the same way as they do in 3D View. Rotating and scaling also work in certain editors as well.

**Python Scripting** You can use Python Scripting in Blender to Grab/Move Objects or elements to a specific location, either using the Python interactive console or running a Python script in the Text Editor Window.

Getting the location vector for current object `bpy.context.scene.objects.active.location` returns you the location vector for the active object in the scene. You can assign a different value to the location vector to change the position of the object.

Operator for translating active object and its syntax:

```
bpy.ops.transform.translate(value=(<DX>, <DY>, <DZ>), constraint_axis=(<bool>,
<bool>, <bool>), constraint_orientation='<ORIENTATION NAME>', mirror=<bool>,
proportional='<ENABLE?DISABLE>', proportional_edit_falloff='<FALLOFF TYPE>',
proportional_size=<INT>, snap=<bool>, snap_target='<SNAP TARGET>', snap_point=<x,y,z>,
snap_align=<bool>, snap_normal=<x,y,z>, texture_space=<bool>, release_confirm=<bool>)
```

## Hints

- Moving an object in Object mode changes the object's origin. Moving the object's vertices/edges/faces in Edit Mode doesn't change the object's origin.

---

## Rotate

---

### Reference

Mode: *Object* and *Edit* modes

Menu: *Object/Mesh/Curve/Surface* → *Transform* → *Rotate*

Hotkey: R

---

**Description** Rotation is also known as a spin, twist, orbit, pivot, revolve, or roll and involves changing the orientation of elements (vertices, edge, face, Object etc) around one or more axes or the element's Pivot Point. There are multiple ways to rotate an element which include:

- The keyboard shortcut (R)
- The 3D manipulator widget
- The Properties menu (N)

Basic rotation usage and common options are described below. For additional information, you may wish to read the [Transform Control](#) and [Orientation](#) pages which provide more information about options such as Precision, Axis Locking, Numeric Input, Snapping and the different types of Pivot Point.

Read more about [Transform Control](#) [Read more about Transform Orientations](#)

---

## Usage

### Rotation using the keyboard shortcut

- Use RMB to select the elements you want to rotate.
- Tap R once to enter rotation mode.
- Rotate the elements by moving the mouse. The closer the mouse is to the elements's center, the higher the rotation influence.
- LMB click to accept changes.

The amount of rotation will be displayed in the bottom left hand corner of the 3D window.



Fig. 1.60: Rotation values

**Constraining the rotation axis (axis locking)** Rotation can be constrained to a particular axis or axes through the use of [Axis Locking](#). To constrain rotation, the following shortcuts can be used:

- R, X: Rotate only along the **X Axis**
- R, Y: Rotate only along the **Y Axis**
- R, Z: Rotate only along the **Z Axis**

Axis locking can also be enabled by pressing the MMB after enabling rotation and moving the mouse in the desired direction e.g.

- R, move the mouse along the X axis, MMB: Rotate only along the **X Axis**

[Read more about Axis Locking](#)

**Fine Tuning The Rotation** Precise control can be had over rotation through the use of the Shift and Ctrl keys to limit rotation to discrete amounts. You can also enter a numerical value in degrees to specify the amount of rotation after after initiating a rotation transformation.

- Hold Ctrl down while performing a rotation to rotate the selected element in 5 degree increments.
- Hold Shift down while performing a rotation to rotate the selected element in 0.01 degree increments.
- Hold Shift-Ctrl down while performing a rotation to rotate the selected element in 1 degree increments.
- Press R, type in a number and press Return to confirm.
- Press R, R to enable Trackball rotation.

---

**Tip:** Orientation dependant rotations

By default, all rotations happen around a Global Orientation. You can change the rotation orientation by pressing the axis key twice. For example, pressing R, X, X will by default set rotation to occur around the local orientation.

---

[Read more about Precision Control](#) [Read more about Numerical Transformations](#) [Read more about Transform Orientations](#)

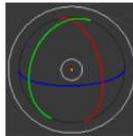


Fig. 1.61: Rotation Transform Manipulator

**Rotation with the 3D Transform Manipulator** In the 3D View header, ensure that the Transform Manipulator is enabled (the red, green, and blue triad is selected). Set the manipulator type to rotation (the highlighted arc icon shown below).



- Select your element with RMB.
- Use LMB and drag any of the three colored axes on the rotation manipulator to rotate your object along that axis. You can also use Shift, Ctrl or numeric input with the 3D manipulator widget for further control.
- Your changes will be applied when you release LMB or press Spacebar or Return. Your changes will be cancelled if you press RMB or Esc.

[Read more about the 3D Transform Manipulator](#)

**Rotation with the Properties Panel** Rotation values can also be specified in the Properties panel (N) by altering the degree value in the rotation slider of the Transform panel. Rotation along particular axes can be enabled or disabled by toggling the padlock icon. The rotation mode (Euler, Axis Angle, Quaternion) can also be set in this panel from the drop down box.

[Read more about Panels](#)

[Read more about rotation modes](#)

[Additional detail about rotation modes](#)

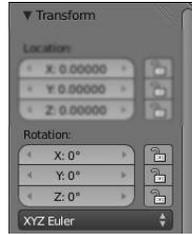


Fig. 1.62: Rotation transform properties panel.

## Scale

### Reference

Mode: *Object* and *Edit* modes

Menu: *Object/Mesh/Curve/Surface* → *Transform* → *Scale*

Hotkey: S

**Description** Pressing S will enter the *Scale* transformation mode where the selected element is scaled inward or outward according to the mouse pointer's location. The element's scale will increase as the mouse pointer is moved away from the Pivot Point and decrease as the pointer is moved towards it. If the mouse pointer crosses from the original side of the Pivot Point to the opposite side, the scale will continue in the negative direction and flip the element.

[Read more about Pivot Points](#)

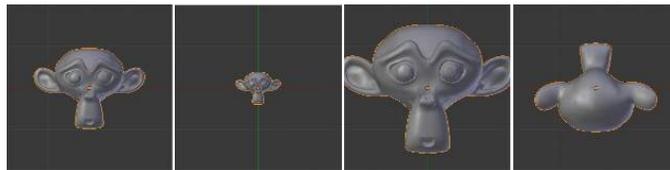


Fig. 1.63: Basic scale usage. From left to right, the panels show: the original Object, a scaled down Object, a scaled up Object and a scale-flipped Object.

There are multiple ways to scale an element which include:

- The keyboard shortcut (S)
- The 3D manipulator widget
- The Properties menu (N)

Basic scale usage and common options are described below. For additional information, you may wish to read the Transform Control and Orientation pages which provide more information about options such as Precision, Axis Locking, Numeric Input, Snapping and the different types of Pivot Point.

[Read more about Transform Control](#)

[Read more about Transform Orientations](#)

## Usage

### Scaling using the keyboard shortcut

- Use **RMB** to select the elements you want to scale.
- Tap **S** once to enter scale mode.
- Scale the elements by moving the mouse.
- **LMB** click to accept changes.

The amount of scaling will be displayed in the bottom left hand corner of the 3D window.

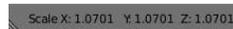


Fig. 1.64: Scale values

**Constraining the scaling axis (axis locking)** Scaling can be constrained to a particular axis or axes through the use of [Axis Locking](#). To constrain scaling, the following shortcuts can be used:

- **S, X**: Scale only along the **X Axis**
- **S, Y**: Scale only along the **Y Axis**
- **S, Z**: Scale only along the **Z Axis**

Axis locking can also be enabled by pressing the **MMB** after enabling scaling and moving the mouse in the desired direction e.g.

- **S**, move the mouse along the X axis, **MMB**: Scale only along the **X Axis**

[Read more about Axis Locking](#)

**Fine Tuning The Scaling** [Precise control](#) can be had over scaling through the use of the **Shift** and **Ctrl** keys to limit scaling to discrete amounts. You can also enter a [numerical value](#) in Blender Units (BU) to specify the amount of scaling after initiating a scale transformation.

- Hold **Ctrl** down while scaling to scale the selected element in degree 0.1 BU increments.
- Hold **Shift** down while scaling to scale the selected element in very fine increments.
- Hold **Shift-Ctrl** down while scaling to scale the selected element in 0.01 BU increments.
- Press **S**, type in a number and press **Return** to confirm.

---

**Tip:** Orientation dependent scaling

By default, all scaling happens around a Global Orientation. You can change the scaling orientation by pressing the axis key twice. For example, pressing **S, X, X** will by default set scaling to occur around the local orientation.

---

[Read more about Precision Control](#)

[Read more about Numerical Transformations](#)

[Read more about Transform Orientations](#)

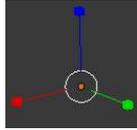


Fig. 1.65: Scaling Transform Manipulator



**Scaling with the 3D Transform Manipulator** In the 3D View header, ensure that the Transform Manipulator is enabled (the red, green, and blue triad is selected). Set the manipulator type to scale (the highlighted square icon shown below).

- Select your element with RMB.
- Use LMB and drag any of the three colored axes on the scaling manipulator to scale your object along that axis. You can also use *Shift*, *Ctrl* or numeric input with the 3D manipulator widget for further control.
- Your changes will be applied when you release LMB or press *Spacebar* or *Return*. Your changes will be cancelled if you press RMB or *Esc*.

[Read more about the 3D Transform Manipulator](#)

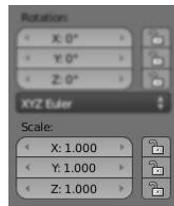


Fig. 1.66: Scale transform properties panel.

**Scaling with the Properties Panel** Scale values can also be specified in the Properties panel (N) by altering the amount value in the scaling slider of the Transform panel. Scaling along particular axes can be enabled or disabled by toggling the padlock icon.

[Read more about Panels](#)

[Read more about scaling modes](#)

## Transform Control

Transform controls can be used to modify and control the effects of the available transformations.

The following pages detail the available control options:

### Precision Reference

Mode: *Object* and *Edit* modes

Hotkey: *Ctrl* and/or *Shift*

**Description** Holding `Ctrl` or `Shift` during a transformation operation (such as grab/move, rotate or scale) will allow you to perform the transformation in either fixed amounts, very small amounts or both. The magnitude of the transformation can be viewed in the 3D window header in the bottom left hand corner. Releasing `Ctrl` or `Shift` during the transformation will cause the movement to revert back to its normal mode of operation.

[Read more about Window Headers](#)

## Usage

**With hotkeys** Press `G`, `R` or `S` and then hold either `Ctrl`, `Shift` or `Ctrl-Shift`.

**With the Transform Manipulator** Hold `Ctrl`, `Shift` or `Ctrl-Shift` and click on the appropriate manipulator handle. Then move the mouse in the desired direction. The reverse action will also work i.e. clicking the manipulator handle and then holding the shortcut key for precision control.

[Read more about the Transform Manipulator](#)

---

### Tip: Combining with other controls

All of the precision controls detailed on the page can be combined with the [Axis Locking](#) controls and used with the different [Pivot Points](#).

---

## Holding CTRL

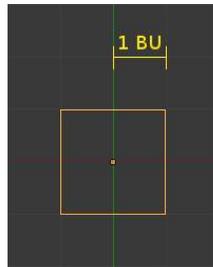


Fig. 1.67: 1 Blender Unit - shown at the default zoom level.

**Grab/move transformations** For grab/move operations at the default zoom level, holding `Ctrl` will cause your selection to move by increments of 1 Blender Unit (1 BU) (i.e. between the two light grey lines). Zooming in enough to see the next set of grey lines will now cause `Ctrl` movements to occur by 1/10 of a BU. Zooming in further until the next set of grey lines becomes visible will cause movement to happen by 1/100 of a BU and so on until the zoom limit is reached. Zooming out will have the opposite effect and cause movement to happen by increments of 10, 100 etc BU.

[Read more about Zooming](#)

**Rotation transformations** Holding `Ctrl` will cause rotations of 5 degrees.

**Scale transformations** Holding `Ctrl` will cause size changes in increments of 0.1 BU.

---

### Note: Snapping modes

Note that if you have a `Snap Element` option enabled, holding `Ctrl` will cause the selection to snap to the nearest element.

---

[Read more about Snapping](#)

**Holding SHIFT** Holding `Shift` during transformations allows for very fine control that does not rely on fixed increments. Rather, large movements of the mouse across the screen only result in small transformations of the selection.

### Holding CTRL and SHIFT

**Grab/move transformations** For grab/move operations at the default zoom level, holding `Ctrl-Shift` will cause your selection to move by increments of 1/10 Blender Units. Holding `Ctrl-Shift` at any zoom level will cause the transformation increments to always be 1/10 of the increment if you were only holding `Ctrl`.

**Rotation transformations** Holding `Ctrl-Shift` will cause rotations of 1 degree.

**Scale transformations** Holding `Ctrl-Shift` will cause size changes in 0.01 BU increments.

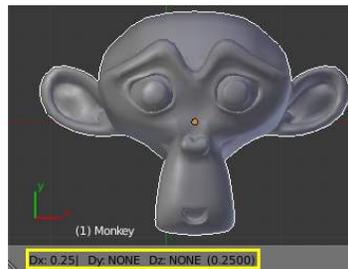


Fig. 1.68: Numeric input in the 3D window header

**Numeric input** Using the mouse for transformations is convenient, but if you require more precise control, you can also enter numeric values. After pressing `G`, `R` or `S`, type a number to indicate the magnitude of the transformation.

You can see the numbers you enter in the bottom left hand corner of the 3D window header. Negative numbers and decimals can be entered by pressing the minus (`Minus`) and period (`.`) keys respectively.

**Translation** To move Objects, vertices, faces or edges select the element, press `G` and then type a number. By default and with no other key presses, movement will occur along the `X`-axis. To confirm the movement, press `Return` or `LMB`. To cancel the movement, press `Esc` or `RMB`. If you mistype the value, press `Backspace` to cancel the current entry and retype a new value.

To enter numeric values for multiple axes, use the `Tab` key after entering a value for the axis. e.g. To move an Object, one (1) Blender unit on all three axes press: `G`, `1`, `Tab`, `1`, `Tab`, `1`. This will move the element one unit along the `X`-axis, followed by the `Y`-axis and then the `Z`-axis.

You can also combine numeric input with axis locking to limit movement to a particular axis. To do so, press `G` followed by `X`, `Y` or `Z` to indicate the axis. Then type in the transform amount using (0 - 9) followed by `Return` to confirm. Pressing `X`, `Y` or `Z` will initially constrain movement to the *Global* axis. Pressing `X`, `Y` or `Z` again will constrain movement to the orientation set in the *Transform Orientation* setting of the 3D window header.

[Read more about Transform Orientations](#)

[Read more about Axis Locking](#)

**Rotation** To specify a value for clockwise rotation, press `R`, (0 - 9), then `Return` to confirm. To specify counter-clockwise rotation press `R`, `Minus`, (kbd:0 - 9), then `Return` to confirm. Note that 270 degrees of clockwise rotation is equivalent to -90 degrees of counter-clockwise rotation.

**Scaling** Objects, faces and edges can be scaled by pressing `S`, (0 - 9), then `Return` to confirm., Scaling transformations can also be constrained to an axis by pressing `X`, `Y` or `Z` after pressing `S`. Essentially, scaling with numeric values works in almost identical fashion to translation. The primary difference is that by default, scaling applies equally to all three axes. e.g. pressing `S`, `0.5`, `Return` will scale an Object by 0.5 on all three axes.

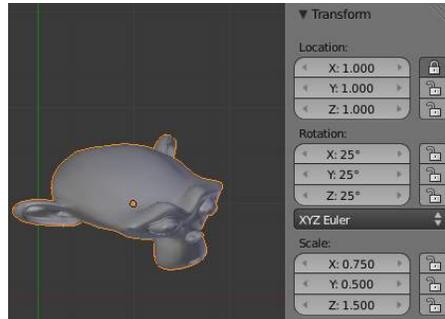


Fig. 1.69: Transformations can also be entered through the Transform panel on the Properties shelf.

**Numeric input via the Properties shelf** It is also possible to enter numeric values for each transformation using the *Transform* panel found on the Properties shelf (`N`). The *Transform* panel can also be used to prevent transformations along particular axes by clicking the lock icon.

**Transform Properties** Each object stores its position, orientation, and scale values. These may need to be manipulated numerically, reset, or applied.

## Transform Properties Panel Reference

Mode: *Edit* and *Object* modes

Menu: *Object* → *Transform Properties*

Hotkey: `N`

The *Transform Properties* section in the *View Properties* panel allows you to view and manually/numerically control the position, rotation, and other properties of an object, in *Object* mode. In *Edit* mode, it mainly allows you to enter precise coordinates for a vertex, or median position for a group of vertices (including an edge/face). As each type of object has a different set of options in its *Transform Properties* panel in *Edit* mode, see their respective descriptions in the [Modeling chapter](#).

### Options in Object mode

**Location** The object's center location in global coordinates.

**Rotation** The object's orientation, relative to the global axes and its own center.

**Scale** The object's scale, relative to its center, in local coordinates (i.e. the *Scale X* value represents the scale along the local X-axis). Each object (cube, sphere, etc.), when created, has a scale of one blender unit in each local direction. To make the object bigger or smaller, you scale it in the desired dimension.

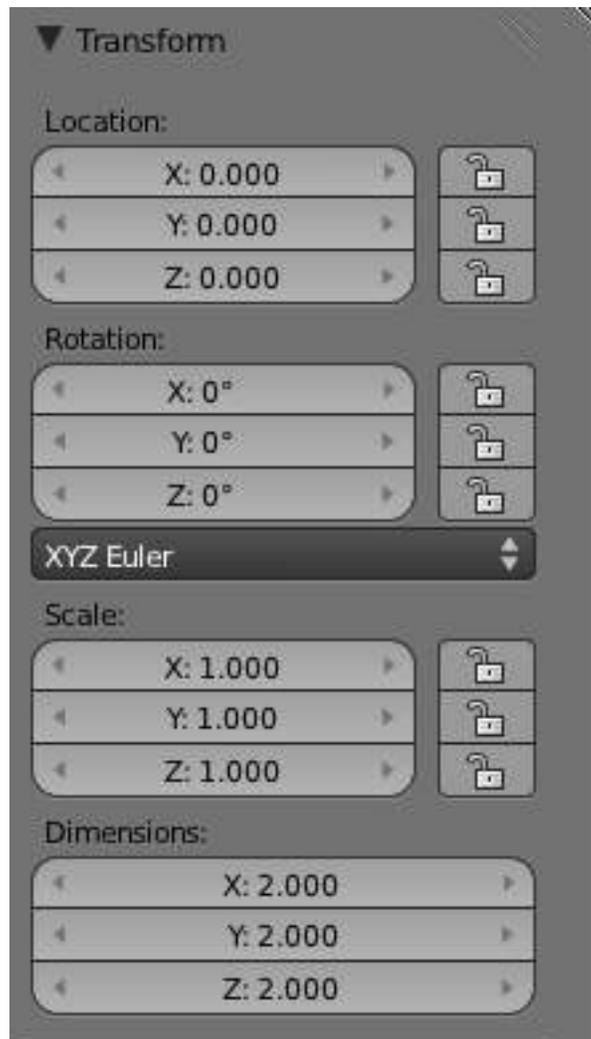


Fig. 1.70: Transform Properties panel in Object mode.

**Dimensions** The object's basic dimensions (in blender units) from one outside edge to another, as if measured with a ruler. For multi-faceted surfaces, these fields give the dimensions of the bounding box (aligned with the local axes - think of a cardboard box just big enough to hold the object).

Use this panel to either edit or display the object's transform properties such as position, rotation and/or scaling. These fields change the object's center and then affects the aspect of all of its *vertices* and faces.

---

**Note:** center

---

Some fields have extra functionality or features, such as scroll regions. When attempting to edit these types of fields it is easier to use `Shift-LMB` instead of just `LMB`. After you have edited a field click outside of the field's edit area or press `Return` to confirm the changes. Changes will be reflected in the display window immediately. To cancel, press `Esc`. For further descriptions of the other features of an edit field see [The Interface](#) section.

**Transform Properties Locking** The locking feature of the Location, Rotation and Scale fields allows you to control a transform property solely from the properties panel. Once a lock has been activated any other methods used for transformation are blocked. For example, if you locked the *Location X* field then you can't use the mouse to translate the object along the global X axis. However, you can still translate it using the *Location X* edit field. Consider the locking feature as a rigid constraint only changeable from the panel.

To lock a field, click the padlock icon next to the field. The field is unlocked if the icon appears as (



), and it is locked if the icon appears as (



).

---

## Clear Object transformations

### Reference

Mode: *Object* mode

Menu: *Object* → *Clear* → *Clear Location/Clear Scale/Clear Rotation/Clear Origin*

Hotkey: `Alt-G`, `Alt-S`, `Alt-R`, `Alt-O`

---

**Description** Clearing transforms simply resets the transform values. The objects location and rotation values return to 0, and the scale returns to 1.

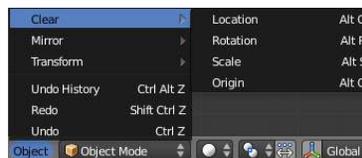


Fig. 1.71: Clear Transformation menu

## Clear Options

**Clear Location `Alt-G`** Clear (reset) the location of the selection. This will move the selection back to the coordinates 0,0,0.

**Clear Scale `Alt-S`** Clear (reset) the scale of the selection. This will resize the selection back to the size it was when created.

**Clear Rotation `Alt-R`** Clear (reset) the rotation of the selection. This will set the rotation of the selection to 0 degrees in each plane.

**Clear Origin `Alt-O`** Clear (reset) the origin of the Child objects. This will cause Child objects to move to the coordinates of the parent.

## Apply Object transformations

### Reference

Mode: *Object* mode

Menu: *Object* → *Apply* →

Hotkey: `Ctrl-A`

Applying transform values essentially resets the values of object's position, rotation, or scale, but does not actually do anything to the object. The center point is moved to the origin and the transform values are set to zero. In terms of scale, the scale values return to 1.

To apply a transform select the *Apply* sub-menu from the *Object menu* or use the shortcut `Ctrl-A` and select the appropriate transform to apply

*Make Duplicates Real* unlinks linked duplicates so each duplicate now has its own datablock.

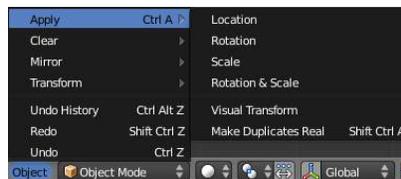


Fig. 1.72: Apply Transformation menu

### Apply Options

**Apply Location `Ctrl-A`** Apply (set) the location of the selection. This will make Blender consider the current location to be equivalent to 0 in each plane i.e. the selection will not move, the current location will be considered to be the “default location”. The Object Center will be set to actual 0,0,0 (where the colored axis lines intersect in each view).

**Apply Rotation `Ctrl-A`** Apply (set) the rotation of the selection. This will make Blender consider the current rotation to be equivalent to 0 degrees in each plane i.e. the selection will not rotated, the current rotation will be considered to be the “default rotation”.

**Apply Scale `Ctrl-A`** Apply (set) the scale of the selection. This will make Blender consider the current scale to be equivalent to 0 in each plane i.e. the selection will not scaled, the current scale will be considered to be the “default scale”.

**Apply Rotation and Scale `Ctrl-A`** Apply (set) the rotation and scale of the selection. Do the above two applications simultaneously.

**Apply Visual Transform `Ctrl-A`** Apply (set) the result of a constraint and apply this back to the Object's location, rotation and scale. See the following post for more detailed discussion: [Apply visual transform](#).

**Make Duplicate Real `Shift-Ctrl-A`** Make any duplicates attached to this Object real so that they can be edited.

**Proportional Edit** Proportional Edit is a way of transforming selected elements (such as vertices) while having that transformation affect other nearby elements. For example, having the movement of a single vertex cause the movement of unselected vertices within a given range. Unselected vertices that are closer to the selected vertex will move more than those farther from it (i.e. they will move proportionally relative to the location of the selected element). Since proportional editing affects the nearby geometry, it is very useful when you need to smoothly deform the surface of a dense mesh.

**Note:** Sculpting

Blender also has [Sculpt Mode](#) that contains brushes and tools for proportionally editing a mesh without seeing the individual vertices.

---

## Object mode

### Reference

Mode: *Object mode*

Menu: Via the icon in the header indicated by the yellow square in the below image.



Hotkey: `O`

---

Proportional editing is typically used in *Edit mode*, however, it can also be used in *Object mode*. In *Object mode* the tool works on entire objects rather than individual mesh components. In the image below, the green cube is the active Object, while the red and blue cubes are located within the proportional edit tool's radius of influence. When the green cube is moved to the right, the other two cubes follow the movement.

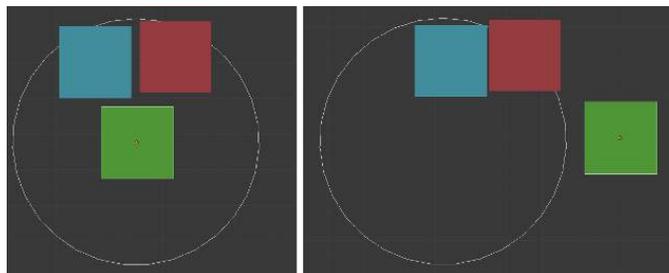


Fig. 1.73: Proportional editing in Object mode.

---

## Edit mode

### Reference

Mode: *Edit mode*

Menu: *Mesh* → *Proportional Editing* and via the highlighted icon in the below image



Hotkey: **O** / **Alt-O** / **Shift-O**

When working with dense geometry, it can become difficult to make subtle adjustments to the vertices without causing visible lumps and creases in the model's surface. When you face situations like this the proportional editing tool can be used to smoothly deform the surface of the model. This is done by the tool's automatic modification of unselected vertices within a given range.

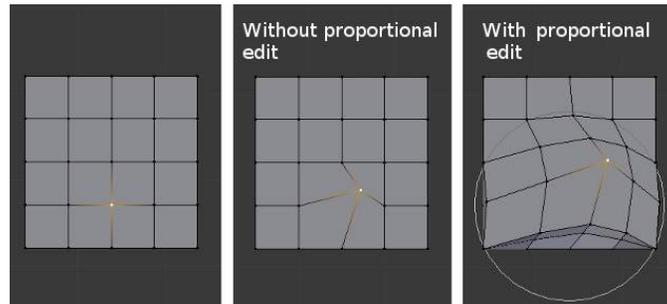


Fig. 1.74: Proportional editing in Edit mode.

**Influence** You can increase or decrease the radius of the proportional editing influence with the mouse wheel **WheelUp** / **WheelDown** or **PageUp** / **PageDown** respectively. As you change the radius, the points surrounding your selection will adjust their positions accordingly.

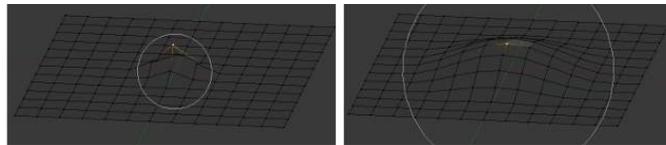


Fig. 1.75: Influence circle.

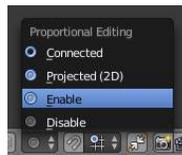


Fig. 1.76: Proportional Editing tool.

**Options** The *Proportional Editing* mode menu is on the *3D View* header.

**Disable** (**O** or **Alt-O**) Proportional Editing is Off, only selected vertices will be affected.

**Enable** (**O** or **Alt-O**) Vertices other than the selected vertex are affected, within a defined radius.

**Projected (2D)** Depth along the view is ignored when applying the radius.

**Connected** (**Alt-O**) Rather than using a radius only, the proportional falloff spreads via connected geometry. This means that you can proportionally edit the vertices in a finger of a hand without affecting the other fingers. While the other vertices are physically close (in 3D space), they are far away following the topological edge connections of the mesh. The icon will have a grey center when *Connected* is active. This mode is only available in *Edit mode*.

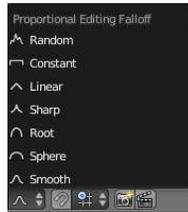


Fig. 1.77: Falloff menu.

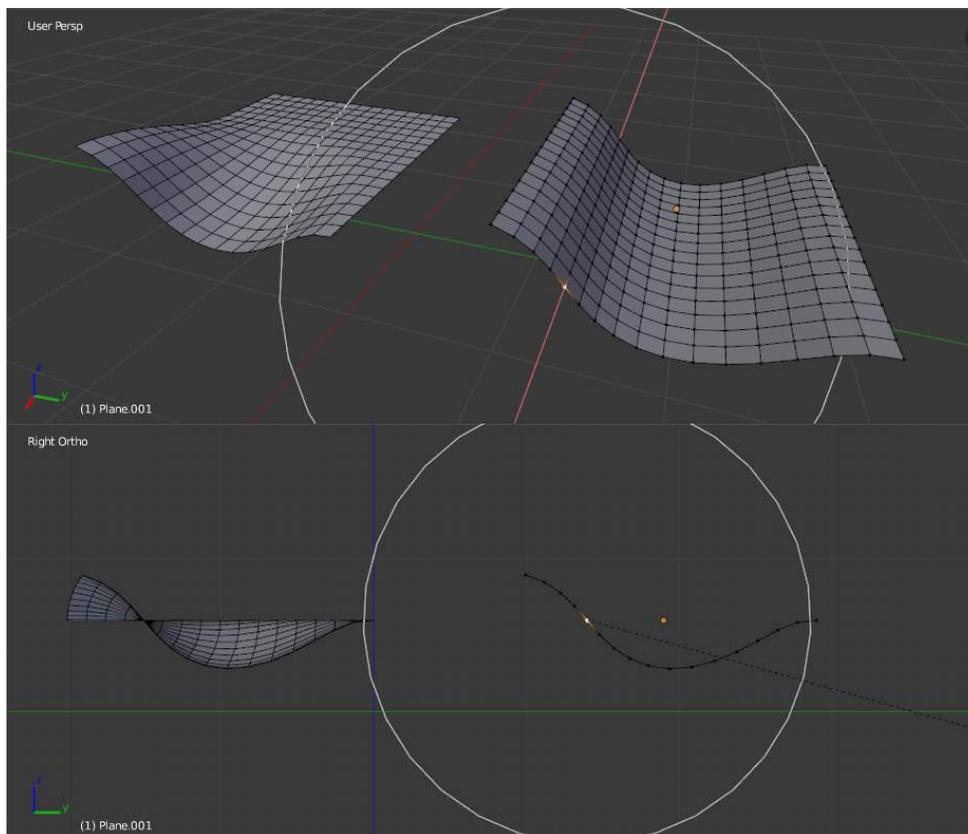
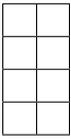


Fig. 1.78: The difference between regular and Projected (2D) proportional option (right).

**Falloff** While you are editing, you can change the curve profile used by either using the *Mesh* → *Proportional Falloff* submenu, using the toolbar icon (*Falloff menu*), or by pressing `Shift-O` to toggle between the various options.



**Examples** Switch to a front view (`Numpad1`) and activate the grab tool with `G`. As you drag the point upwards, notice how nearby vertices are dragged along with it. When you are satisfied with the placement, click `LMB` to fix the position. If you are not satisfied, cancel the operation and revert your mesh to the way it looked before with `RMB` (or `ESC`).

You can use the proportional editing tool to produce great effects with the scaling (`S`) and rotation (`R`) tools, as *A landscape obtained via proportional editing* shows.

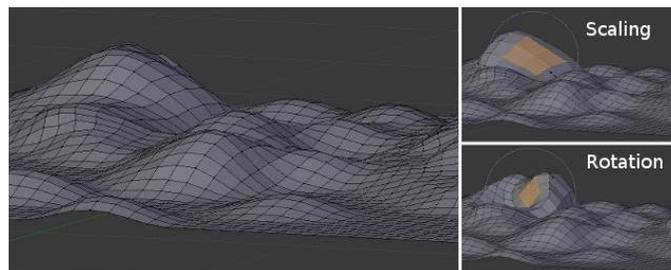


Fig. 1.93: A landscape obtained via proportional editing.

Combine these techniques with vertex painting to create fantastic landscapes. The *final rendered landscape* image below shows the results of proportional editing after the application of textures and lighting.



Fig. 1.94: Final rendered landscape.

## Manipulators

### Reference

Mode: *Object* and *Edit* modes

Hotkey: `Ctrl-Spacebar`

In combination with [axis locking](#), the normal Transform commands (`G` for Grab, `R` for Rotation, `S` for Scale), can be used to manipulate objects along any axis. However, there may be times when these options are not adequate. For example, when you

want to translate a single face on a randomly rotated object in a direction perpendicular to the face's normal. In instances like this, *Transform Manipulators* may be useful.



Fig. 1.95: Manipulator options in the Window Header.

Transform manipulators provide a visual representation of the transform options and allow movement, rotation and scaling along any axis, mode and orientation of the 3D view. The manipulator can be enabled by clicking on the axis icon from the manipulator options portion of the window header or via the shortcut key `Ctrl-Spacebar`.

There is a separate manipulator for each Transform Command. Each manipulator can be used separately or in combination with the others. Clicking with `Shift-LMB` on multiple manipulator icons (arrow, arc, box) will combine manipulator options.

Manipulators can be accessed in the header of the *3D View* window:

- Axis: Enable/disable the manipulators.
- Arrow: Translation.
- Arc: Rotation.
- Box: Scale.
- Transform Orientation menu: choice of the transformation orientation.

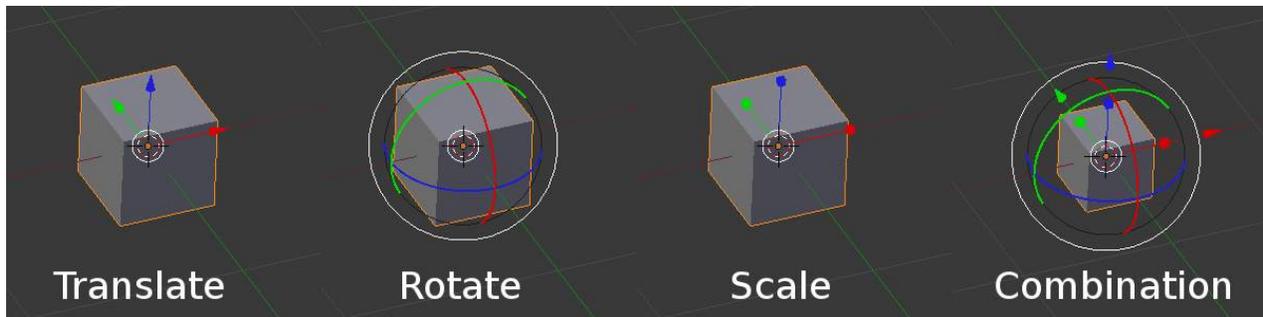


Fig. 1.96: Manipulator Options

### Manipulator controls

- Holding down `Ctrl` constrains the action to set increments. Holding down `Shift` **after** you `LMB` the manipulator handle will constrain the action to smaller increments.
- Holding down `Shift` **before** you `LMB` click on one of the handles will cause the manipulator action to be performed relative to the other two axes (you can let go of `Shift` once you have clicked). For example, if you `Shift` then `LMB` the `Z` axis handle of the translate manipulator, movement will occur in the `X` and `Y` planes.
- When in rotate mode, `LMB` on the white circle (largest circle around the rotation manipulator) will be equivalent to pressing `R`.
- When in rotate mode, `LMB` on the grey circle (small inner circle at the center of the rotation manipulator) will be equivalent to pressing `R` twice. This will start *trackball* rotation.

[Read more about constraining transformations](#) [Read more about axis locking](#) [Read more about trackball rotation](#)



Fig. 1.97: Manipulator preferences.

**Manipulator Preferences** The settings of the manipulator (e.g. its size) can be found in the *Interface* section of the *User Preferences* window.

**Size** Diameter of the manipulator.

**Handle Size** Size of manipulator handles, as a percentage of the manipulator radius ( $\text{size} / 2$ ).

**Hotspot** Hotspot size (in pixels) for clicking the manipulator handles.

---

### Choosing the Transform Orientation Reference

Mode: *Object* and *Edit* modes

Hotkey: Alt-Spacebar

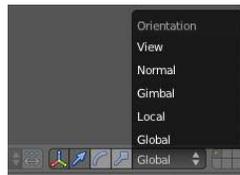


Fig. 1.98: Transform Orientation options.

You can also change the [orientation of the Transform Manipulator](#) to global, local, gimbal, normal or view from the Transform options menu. The image below shows a cube with the rotation manipulator active in multiple transform orientations. Notice how the manipulator changes depending on the orientation selected (compare A with F).

Similarly, notice how when normal orientation (F and G) is selected the manipulator changes between *Object mode* and *Edit mode*. The normal orientation manipulator will also change depending on what is selected in *Edit mode* i.e. the orientation is based on the normal of the selection which will change depending on how many and which faces, edges or vertices are selected.

---

### Transform Orientations Reference

Mode: *Object* and *Edit* modes

Hotkey: Alt-Spacebar

---

Orientations affect the behavior of Transformations: Location, Rotation, and Scale. You will see an effect on the 3D Manipulator (the widget in the center of the selection), as well as on transformation constraints (like [axis locking](#)). This means that, when you press G-X, it will constrain to the *global* x-axis, but if you press G-X-X it will constrain to your *Transform Orientation* s x-axis.

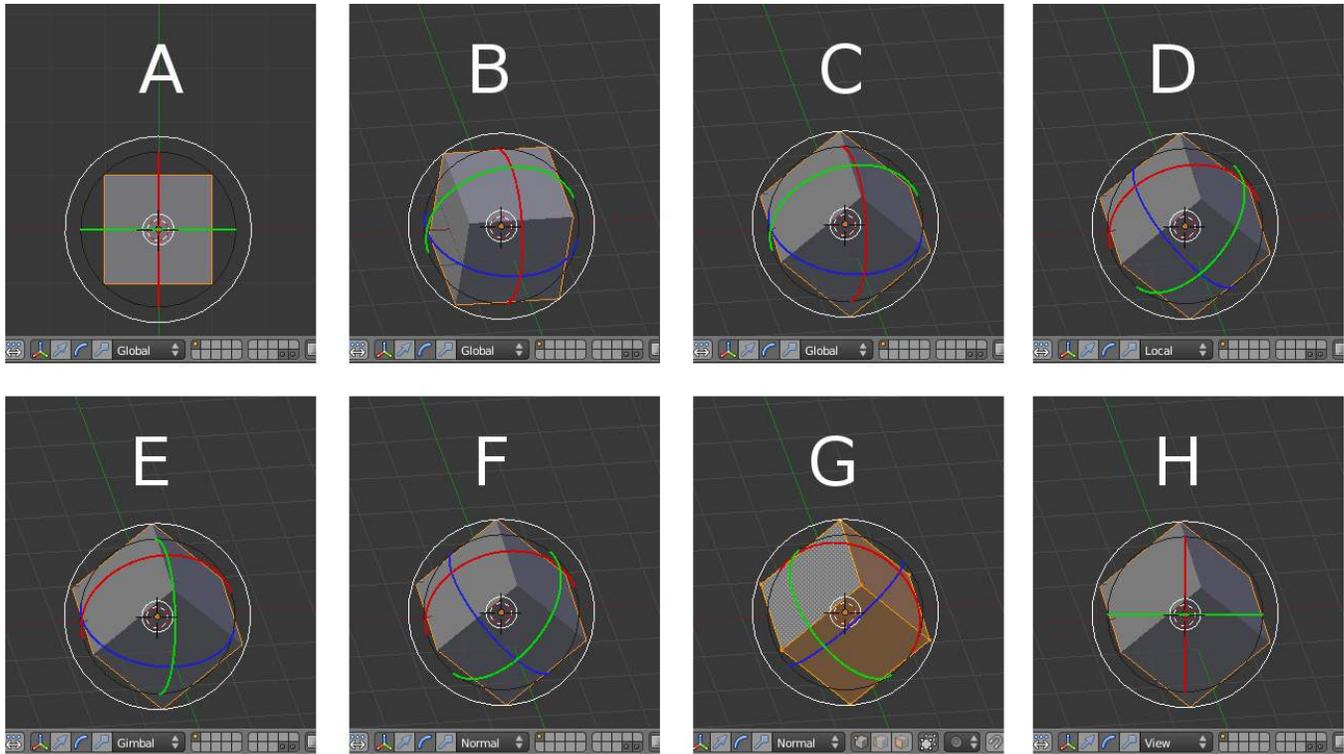


Fig. 1.99: Transform manipulator orientation options.

- 1. Standard cube in default top view with *global* orientation selected
- 2. Standard cube with view rotated and *global* orientation selected
- 3. Randomly rotated cube with view rotated and *global* orientation selected
- 4. Randomly rotated cube with *local* orientation selected
- 5. Randomly rotated cube with *gimbal* orientation selected
- 6. Randomly rotated cube with *normal* orientation selected
- 7. Randomly rotated cube, vertices selected with *normal* orientation selected
- 8. Randomly rotated cube with *view* orientation selected

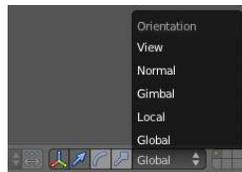


Fig. 1.100: Transform orientations selection menu.

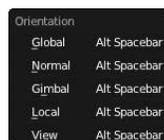


Fig. 1.101: Alt+Space Menu.

The Orientations options can be set on the 3D View's header (or “footer”, since it is at the bottom of the view by default), or with `Alt-Spacebar` or through the *Orientation* menu in a 3D view header.

In addition to the four built-in options, you can define your own custom orientation (see *Custom Orientations* below).

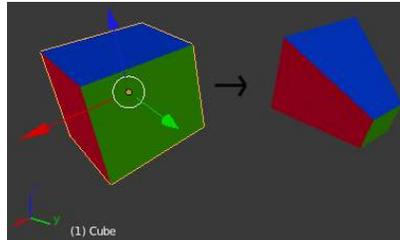


Fig. 1.102: To demonstrate the various behaviors, we add some colors to the default cube, rotate it  $-15^\circ$  along its local z- and x-axes, and we scale its “y” face down.

**Our Demo Cube** Please note two things:

- The “Mini-axis” in the lower-left corner, which represents the Global x/y/z orientation.
- The “Object Manipulator” widget emanating from the selection, which represents the current Transform Orientation.
  - If you click on one of the axes of the Manipulator with LMB, it will allow you to constrain movement to only this direction. An example of a keyboard equivalent is `G, Z, Z`.
  - If you `Shift-LMB` click, it will lock the axis you clicked on and allow you to move in the plane of the two remaining axes. The keyboard analogue is `G, Shift-Z, Shift-Z`.

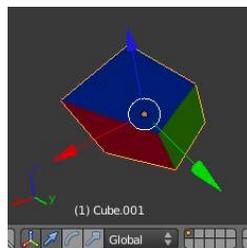


Fig. 1.103: Global.

## Orientations

**Global** The manipulator matches the global axis. When using the Global orientation, the orientation's x,y,z matches world's x,y,z axis. When this mode is selected, the local coordinates of the object are subjected to the Global coordinates. This is good to place objects in the scene. To constrain an axis, press `G` and the desired axis. To constrain to a local axis, press the desired axis two times. The difference between Global and Local, is more noticeable when you have an object in which the origin is not located at the exact center of the object, and doesn't match the Global coordinates.

**Local** The manipulator matches the object axis. Notice that, here, the Manipulator is at a slight tilt (it is most visible on the object's y-axis, the green arrow). This is due to our  $15^\circ$  rotation of the object. This demonstrates the difference between local coordinates and global coordinates. If we had rotated the object  $90^\circ$  along its x-axis, we would see that the object's “Up” is the world's “Forward” – or the object's z-axis would now be the world's y-axis. This orientation has an effect on many parts of the interface, so it is important to understand the distinction.

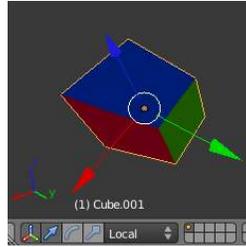


Fig. 1.104: Local.

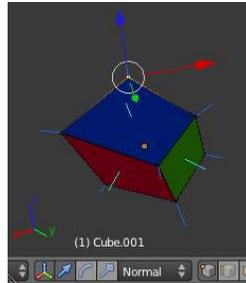


Fig. 1.105: Normal.

**Normal** The z-axis of the manipulator will match the normal vector of the selected object. In Object Mode, this is equivalent to Local Orientation, but in Edit Mode, it becomes more interesting.

As you see, the light blue lines indicate the faces' normals, and the darker blue lines indicate the vertex normals (these were turned on in the **N** Properties Panel under *Mesh Display* → *Normals* → *Face* and *Vertex*). Selecting any given face will cause our Manipulator's z-axis to align with that normal. The same goes for Vertex Select Mode. Edge Select is different—A selected Edge has the z-axis aligned with it (so you will have to look at the Manipulator widget to determine the direction of x and y). If you select several elements, it will orient towards the average of those normals.

A great example of how this is useful is in Vertex Select Mode: Pick a vertex and then do **G**, **Z**, **Z** to tug it away from the mesh and shove it into the mesh. To make this even more useful, select a nearby vertex and press **Shift-R** to repeat the same movement—except along that second vertex's normal instead.

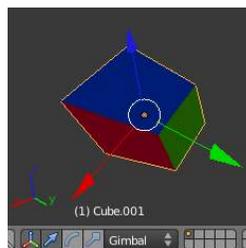


Fig. 1.106: Gimbal.

**Gimbal** Gimbal's behavior highly depends on the **Rotation Mode** that you are in (accessible in the **N** Properties Panel in the *3D View*, in top section, *Transform*).

**XYZ Euler** the default rotation mode, the object Manipulator's z-axis will always point to the global z-axis, where the other two will remain perpendicular to each other. In the other *Euler* rotation modes, the last axis applied will be the one for which the Manipulator stays fixed. So, for *YZX Euler*, the x-axis of the Manipulator will be the same as the global x-axis.

**Axis Angle** The x, y, and z coordinates define a point relative to the object origin through which an imaginary “skewer” passes. The w value is the rotation of this skewer. Here, the Manipulator’s z-axis stays aligned with this skewer.

**Quaternion** Though Quaternion rotation is very different from the Euler and Axis Angle rotation modes, the Manipulator behaves the same as in *Local* mode.

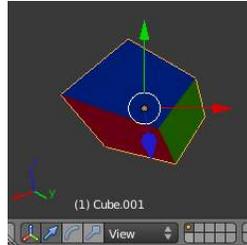


Fig. 1.107: View.

**View** The manipulator will match the 3D view, Y → Up/Down, X → Left/Right, Z → Towards/Away from you.

This way you can constrain movement to one View axis with G-X-X.

## Custom Orientations

### Reference

Mode: *Object* and *Edit* modes

Hotkey: Ctrl-Alt-Spacebar



Fig. 1.108: custom orientation

You can define custom transform orientations, using object or mesh elements. Custom transform orientations defined from objects use the local orientation of the object whereas those defined from selected mesh elements (vertices, edges, faces) use the normal orientation of the selection.

The *Transform Orientations* panel, found in the Properties Panel, can be used to manage transform orientations: selecting the active orientation, adding and deleting custom orientations.

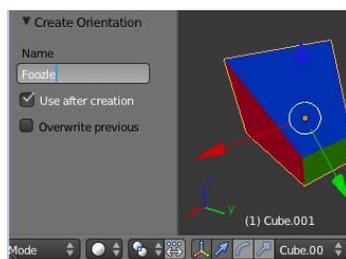


Fig. 1.109: Renaming a Custom Orientation

The default name for these orientations comes from whatever you have selected. If an edge, it will be titled, “Edge,” if an object, it will take that object’s name, etc. The Toolshelf (T in the 3D View) allows you to rename the custom orientation after you press `Ctrl-Alt-Spacebar`.

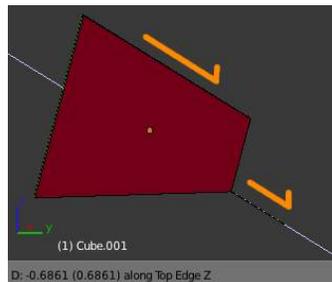


Fig. 1.110: Figure 1.

The technique of creating custom orientations can become important in creating precise meshes. In *Figure 1*, to achieve this effect:

- Select the object’s sloping top edge
- Create a Custom Orientation with `Ctrl-Alt-Spacebar` and rename it “Top Edge”.
- Select the object’s bottom, right edge.
- Extrude with `E`.
- Cancel the extrusion’s default movement by pressing `RMB` or `Esc`.
- Hit `G` to reinitiate movement.
- Hit `Z-Z` to constrain to the “Top Edge” orientation.

### Axis Locking

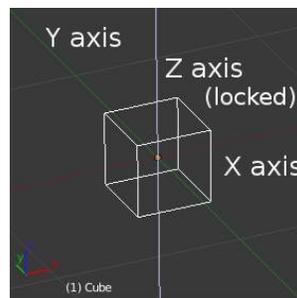


Fig. 1.111: Axis locking

**Description** Transformations (translation/scale/rotation) in *Object* and *Edit* mode, as well as extrusion in *Edit* mode) can be locked to particular axis relative to the current transform orientation. By locking a transformation to a particular axis you are restricting transformations to a single dimension.

**Usage** A locked axis will display in a brighter color than an unlocked axis. For example in the image to the right, the Z axis is drawn in light blue as movement is constrained to this axis. This example can be achieved in two ways:

- Press `G` to enable translation, press `Z` to constrain movement to the Z-axis.

- Press **G** to enable translation, move the mouse in the **Z** direction, then press **MMB**.

### Axis locking types

#### Axis locking Reference

Mode: *Object* and *Edit* modes (translate, rotate, scale, extrude)

Hotkey: **X**, **Y**, **Z** or **MMB** after moving the mouse in the desired direction.

Axis locking limits the transformation to a single axis (or forbids transformations along two axes). An object, face, vertex or other selectable item will only be able to move, scale or rotate in a single dimension.

#### Plane locking Reference

Mode: *Object* and *Edit* modes (translate, scale)

Hotkey: **Shift-X**, **Shift-Y**, **Shift-Z** or **Shift-MMB**  
after moving the mouse in the desired direction.

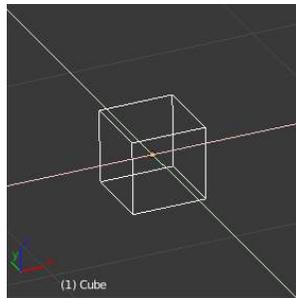


Fig. 1.112: Plane locking

Plane locking locks the transformation to *two* axes (or forbids transformations along one axis), thus creating a plane in which the element can be moved or scaled freely. Plane locking only affects translation and scaling.

Note that for rotation, both axis and plane locking have the same effect because a rotation is always constrained around one axis. *Trackball* type rotations **R-R** cannot be locked at all.

**Axis locking modes** A single key press constrains movement to the corresponding *Global* axis. A second key press of the *same* key constrains movement to the current transform orientation selection (except if it is set to *Global*, in which case the *Local* orientation is used). Finally, a third key press of the same key removes constraints.

For example, if the current transform orientation is set to *Normal*, pressing **G** to start translation, followed by **Z** will lock translation in the **Z** direction relative to the *Global* orientation, pressing **Z** again will lock translation to the **Z** axis relative to the *Normal* orientation. Pressing **Z** again will remove all constraints. The current mode will be displayed in the left hand side of the *3D window header*.

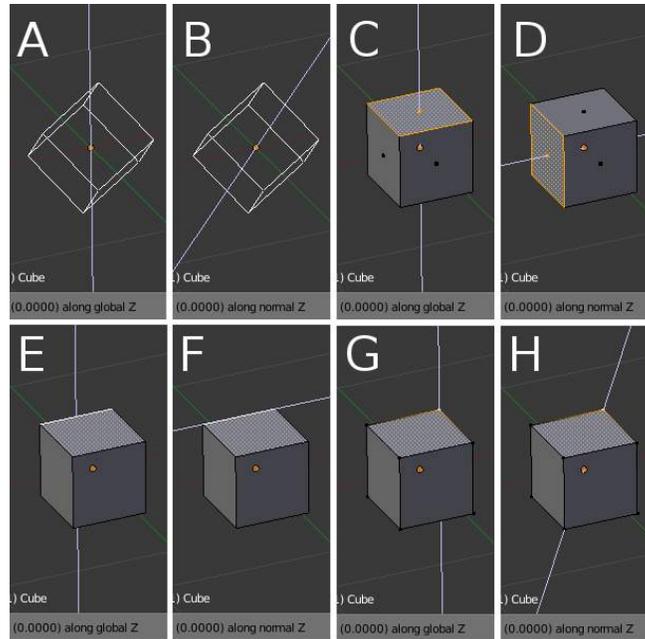


Fig. 1.113: Axis locking modes

As can be seen in the *Axis locking modes* image, the direction of the transform also takes into account the selection. Sections A and B show Z axis locking in *Global* and *Normal* orientations respectively. C and D show the same situation with face selection, E and F with edge selection and G and H with vertex selection.

Note that using a locked axis does not prevent you from using the keyboard to enter [numeric transformation values](#).

**Snapping** There are two types of snap operations that you can use in Blender. The first type snaps your selection or cursor to a given point while the second type is used during transformations (translate, rotate, scale) and snaps your selection to elements within the scene.

Snap

---

## Reference

Mode: *Object* and *Edit* modes

Hotkey: `Shift-S`

---

The *Snap* menu (also available from the 3D header in both *Object* and *Edit* mode (*Object* → *Snap* and *Mesh* → *Snap*)). This menu provides a number of options to move the cursor or your selection to a defined point (the cursor, selection or the grid).

**Selection to Grid** Snaps the currently selected object(s) to the nearest grid point.

**Selection to Cursor** Snaps the currently selected object(s) to the cursor location.

**Cursor to Selected** Moves the cursor to the center of the selected object(s).

**Cursor to Center** Moves the cursor to the center of the grid.

**Cursor to Grid** Moves the cursor to the nearest grid point.

**Cursor to Active** Moves the cursor to the center of the active object.

**Transform Snapping** The ability to snap Objects and Mesh element to various types of scene elements during a transformation is available by toggling the magnet icon (which will turn red) in the 3D view's header buttons.



Fig. 1.114: Magnet icon in the 3D view header (red when enabled).



Fig. 1.115: Snap Element menu

### Snap Element

**Volume** Snaps to regions within the volume of the first Object found below the mouse cursor. Unlike the other options, this one controls the depth (i.e. Z-coordinates in current view space) of the transformed element. By toggling the button that appears to the right of the snap target menu (see below), target objects will be considered as a whole when determining the volume center.

**Face** Snap to the surfaces of faces in mesh objects. Useful for retopologizing.

**Edge** Snap to edges of mesh objects.

**Vertex** Snap to vertices of mesh objects.

**Increment** Snap to grid points. When in Orthographic view, the snapping increment changes depending on zoom level.

---

**Note:** In this context the grid does not mean the visual grid cue displayed. Snapping will use the resolution of the displayed grid, but all transformations are relative to the initial position (before the snap operation).

---

**Snap Target** Snap target options become active when either *Vertex*, *Edge*, *Face*, or *Volume* is selected as the snap element. These determine what part of the selection snaps to the target objects.

**Active** move the active element (vertex in Edit mode, object in Object mode) to the target.

**Median** move the median of the selection to the target.

**Center** move the current transformation center to the target. Can be used with 3D cursor to snap with an offset.

**Closest** move the closest point of the selection to the target.



**Additional snap options**

As seen by the red highlighted areas in the image above, additional controls are available to alter snap behaviour. These options vary between mode (Object and Edit) as well as Snap Element. The four options available are:

- 
-



Fig. 1.126: Align rotation with the snapping target.



Fig. 1.127: Project individual elements on the surface of other objects.

- 
- 

**Multiple Snap Targets** Once transforming a selection with Snapping on (not just with the Ctrl key held), you can press A to mark the current snapping point, then proceed to mark as many other snapping points as you wish and the selection will be snapped to the average location of all the marked points.

Marking a point more than once will give it more weight in the averaged location.

---

### Pivot Point

#### Reference

Mode: *Object mode* and *Edit mode*

Menu: Droplist in the header of the 3D view

---

The pivot point is the point in space around which all rotations, scalings and mirror transformations are centered. You can choose one of five *Pivot Points* from a drop-down list in the header of any 3D area, as seen here in (*Pivot Point modes*). The pages linked below describe each *Pivot Point* mode in more detail.

---

### Active Element as Pivot

#### Reference

Mode: *Object mode* and *Edit mode*

Menu: Select from the following icon in the 3D window header

Hotkey: Alt-. .

---

The *active* element can be an Object, vertex, edge or a face. The active element is the last one to be selected and will be shown in a lighter orange color when in *Object mode* and white when in *Edit mode*. With *Active element as Pivot* set to active, all transformations will occur relative to the active element.

[Read more about selecting different pivot points](#)



Fig. 1.128: Snaps elements to its own mesh.



Fig. 1.129: Consider Objects as whole when finding volume center.

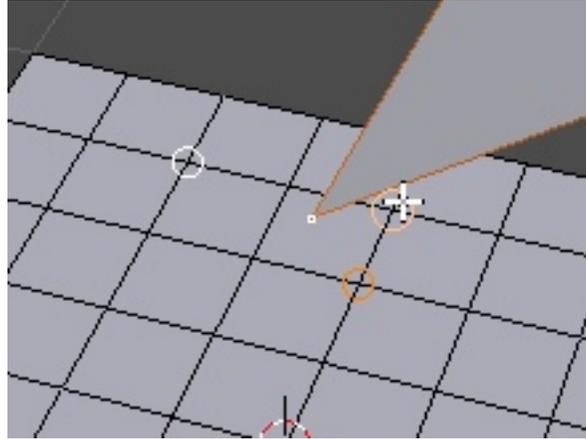


Fig. 1.130: Multiple snapping targets.

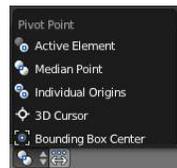


Fig. 1.131: Pivot Point modes.

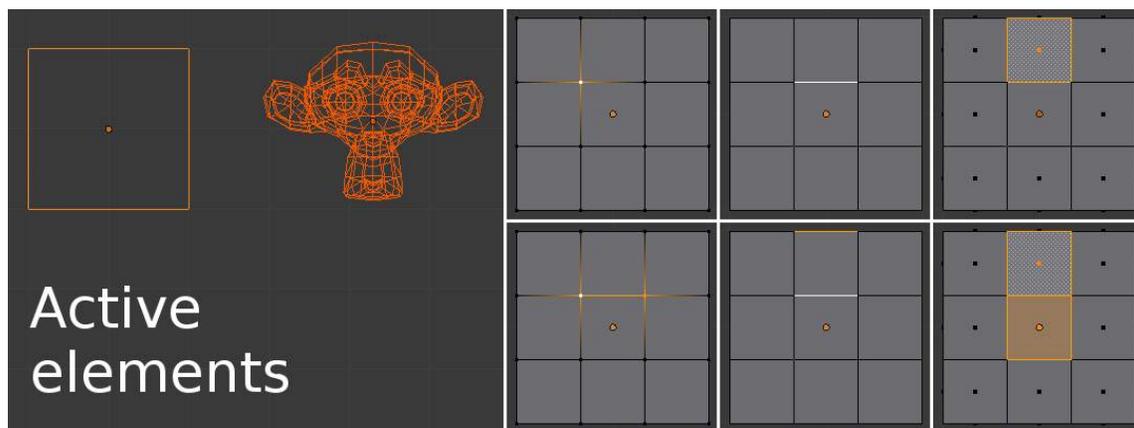


Fig. 1.132: Display of active elements in Object mode is shown on the left of the image where the active element (cube) is a lighter orange. Active elements for vertices, edges and faces in Edit mode are displayed in white and are shown on the right.

**In Object mode** When in *Object mode*, rotation and scaling happen around the active Object's center. This is shown by the figure to the below where the active Object (the cube) remains in the same location (note its position relative to the 3D cursor) while the other Objects rotate and scale in relation to the active element.

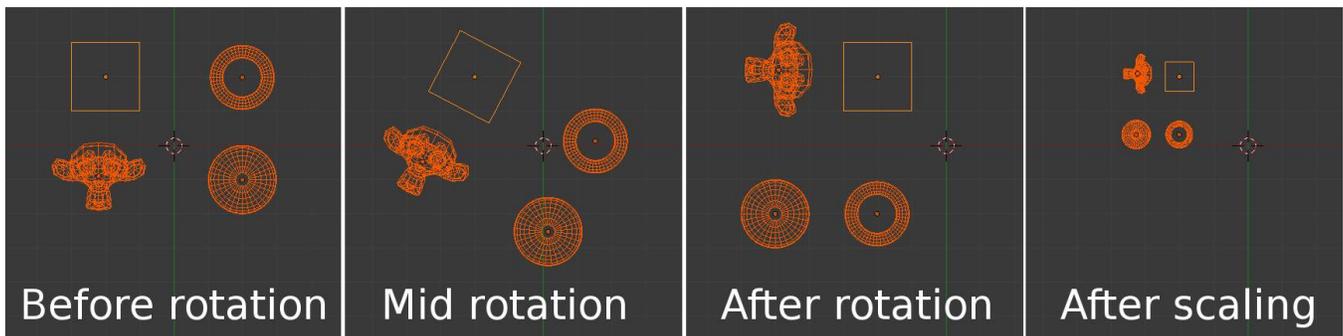


Fig. 1.133: Rotation and scaling with the cube as the active element.

**In Edit mode** Using the active element as a pivot point in *Edit mode* may seem complex but all the possible transformations follow a few rules:

- The pivot point is always at the median of the active element(s).
- The transformations occur by transformation of the **vertices** of the selected element(s). If an unselected element shares one or more vertices with a selected element then the unselected one will get some degree of transformation also.

Let's examine the following examples: in each case we will see that the two rules apply.

**Single selection** When one single element is selected it becomes automatically active. In the image below, you can see that when it is transformed its vertices move, with the consequence that any adjacent element which shares one or more vertices with the active element is also transformed.

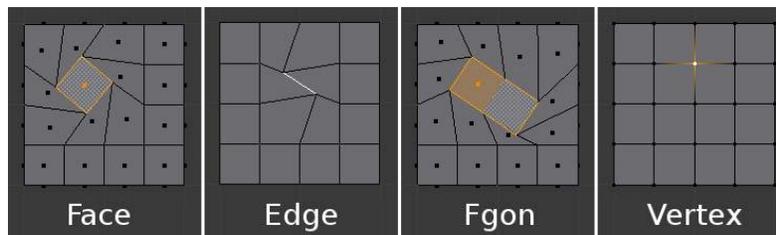


Fig. 1.134: Edit mode and only one element selected.

Let's review each case:

- *Faces* have their pivot point where their selection dot appears, which is where the median of their vertices is.
- *Edges* have their pivot point on their middle since this is always where the median of an edge is.
- *Fgons* behave the same as faces.
- A single *Vertex* has no dimensions at all so it can't show any transformation (except translation, which is not affected by the pivot point).

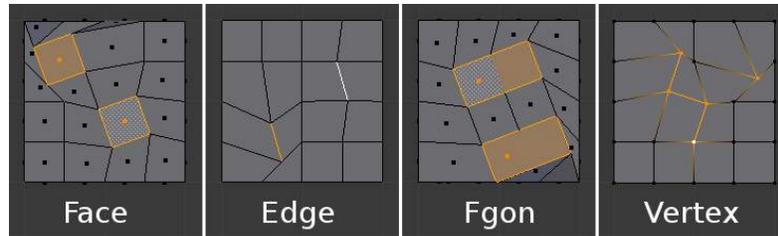


Fig. 1.135: Edit mode and multiple selections.

**Multiple selection** When multiple elements are selected they all transform. The pivot points stay in the same place as what we've seen above, with only one exception for Fgons. In the image below, the selected elements have been rotated.

- For *Faces* the transformation occurs around the selection dot of the active face.
- *Edges* also keep the same behavior with their pivot point at their median.
- *Fgons* behave exactly like faces.
- There is a case for *Vertices* this time: the active Vertex is where the pivot point resides. All other vertices are transformed relative to it.

### Median Point as Pivot Reference

Mode: *Object mode* and *Edit mode*

Menu: Select from the following icon in the 3D window header



Hotkey: `Ctrl-,`

The *Median Point* can be considered to be broadly similar to the concept of Center of Gravity (COG). If we assume that every element (Object, face, vertex etc) of the selection has the same mass, the median point would sit at the point of equilibrium for the selection (the COG).

**In Object Mode** In Object Mode, Blender only considers the Object centers when determining the median point. This can lead to some counterintuitive results. In the Object Mode median points image below, you can see that the median point is between the Object centers and can be nowhere near the Objects' mesh.

**In Edit Mode** In Edit Mode, the median point is determined via the part of the selection that has the most elements. For example, in the *Median points in Edit Mode* image, when there are two cubes with an equal number of vertices, the median point lies directly between the two cubes. However, if we subdivide one cube multiple times so that it has many more vertices, you can see that the median point has shifted to the region with the most vertices.

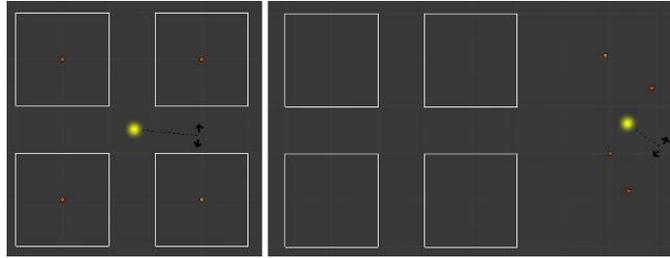


Fig. 1.136: Median points in Object Mode. The Median point is indicated by the yellow dot.

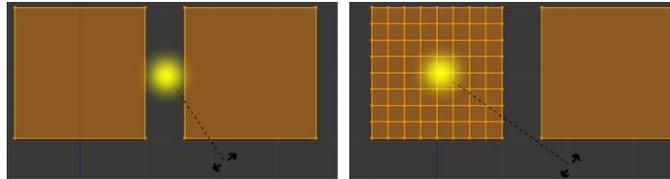


Fig. 1.137: Median points in Edit Mode. The Median point is indicated by the yellow dot.

### Individual Origins as Pivot Reference

Mode: *Object mode* and *Edit mode*

Menu: Select from the following icon in the 3D window header



Hotkey: `Ctrl-.`

**In Object mode** The Origin of an Object is shown in the 3D view by a small orange circle. This is highlighted in the image to the right by the red arrow. The origin tells Blender *where that Object is in 3D space*. What you see in the 3D view (vertices, edges etc) is what makes up the Object.

While the Origin is equivalent to the center of the *Object*, it does not have to be located in the center of the *Mesh*. This means that an Object can have its center located on one end of the mesh or even completely outside the mesh. For example, the orange rectangle in the image has its Origin located on the far left of the mesh.

Now let's examine *Rotation around the individual origins*.

- The blue rectangle has its Origin located in the center of the mesh, while the orange rectangle has its Origin located on the left hand side.
- When the Pivot Point is set to *Individual Origins*, the center of each Object (indicated by the red arrow) remains in place while the Object rotates around it in the path shown by the black arrow.

**In Edit mode** In Edit mode, setting the Pivot Point to Individual Origins produces different results when the selection mode is set to Vertex, Edge or Face. For example, Vertex mode produces results similar to setting the pivot point to median and Edge mode often produces distorted results. Using Individual Origins in Face mode produces the most predictable results.

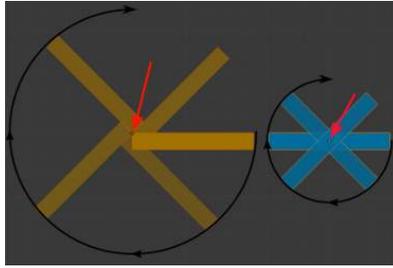


Fig. 1.138: Rotation around individual origins.



As can be seen in the images above, faces that touch each other will deform when rotated when the pivot point is set to Individual Origins. Faces that do not touch will rotate around their Individual Origins (their center).



Groups of faces and Fgons can be scaled without their outside perimeter being deformed. However, the individual faces inside will not be scaled uniformly.



Fig. 1.147: Modeling with faces and individual origins as the pivot point.

Once you are aware of its limitations and pitfalls, this tool can save a lot of time and lead to unique shapes. This “anemone” was modeled from a 12 sided cylinder in about 10 minutes by repeatedly using this workflow: extrusions of individual faces, scaling with *median as a pivot point*, and scaling and rotations of those faces with *Individual Origins as pivot points*.

### 3D Cursor as Pivot Reference

Mode: *Object mode* and *Edit mode*

Hotkey: .

The 3D cursor is the most intuitive of the pivot points. With the 3D cursor selected as the active pivot point (from either the *Window Header* or via the . hotkey), simply position the 3D cursor and then do the required transformation. All rotation and scaling transformations will now be done relative to the location of the 3D cursor. The image below shows the difference when rotating an Object from its starting position (first panel) 90 degrees around the median point (second panel) and 90 degrees around the 3D cursor (third panel).

[Read more about selecting different pivot points](#)

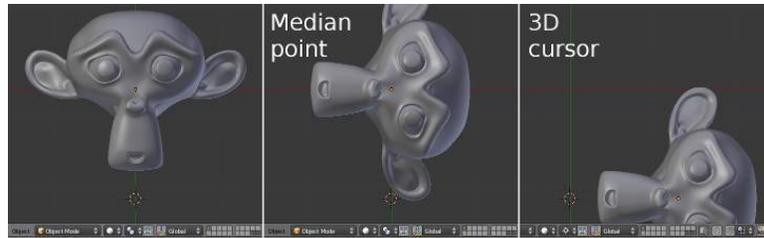


Fig. 1.148: Rotation around the 3D cursor compared to the median point.

## Bounding Box Center as Pivot Reference

Mode: *Object mode* and *Edit mode*

Menu: Select from the following icon in the 3D window header



Hotkey: ,

The bounding box is a rectangular box that is wrapped as tightly as possible around the selection. It is oriented parallel to the world axes. In this mode the pivot point lies at the center of the bounding box. You can set the pivot point to bounding box with the , hotkey or via the menu in the *Window Header*. The image below shows how the Object's Bounding Box size is determined by the size of the Object.

[Read more about selecting different pivot points](#)

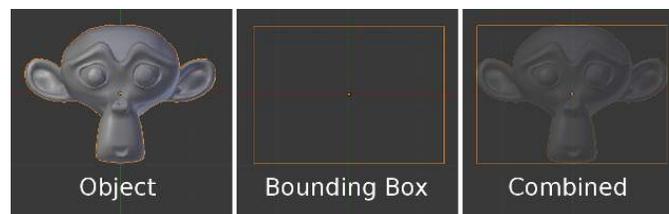


Fig. 1.149: Relationship between an Object and its Bounding Box.

**In Object mode** In *Object mode*, the bounding box is wrapped around the Object and transformation takes place relative to the location of the Object center (indicated by the yellow circle). The image below shows the results of using the Bounding Box as the pivot point in a number of situations.

For example, images A (before rotation) and B show rotation when the Object center is in its default position, while images C (before rotation) and D shows the result when the Object center has been moved. Image E shows that when multiple Objects are selected, the pivot point is calculated based on the Bounding Box of all the selected Objects.

**In Edit mode** This time it is the ObData that is enclosed in the bounding box. The bounding box in *Edit mode* takes no account of the Object(s) centers, only the center of the selected vertices.

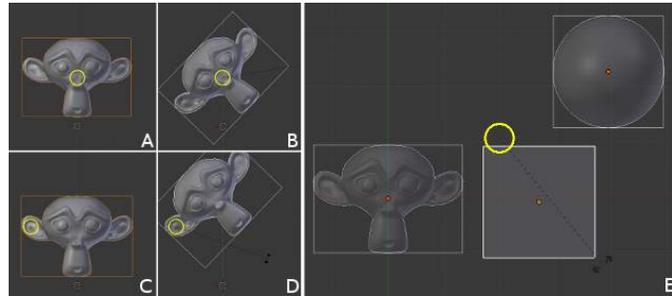


Fig. 1.150: The grid of four images on the left (ABCD) shows the results of Object rotation when the pivot point is set to Bounding Box. The image to the right (E) shows the location of the Bounding Box pivot point when multiple Objects are selected. The pivot point is shown by a yellow circle.

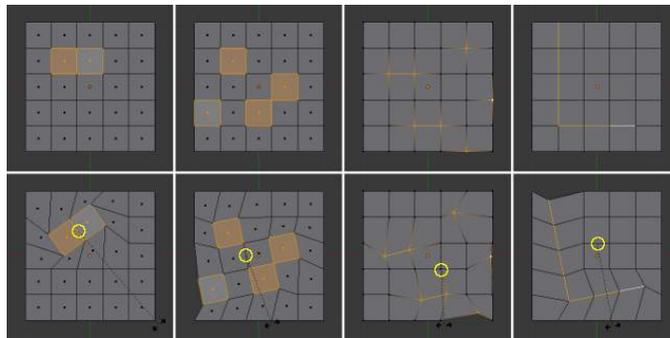


Fig. 1.151: The effects of rotation in different mesh selection modes when the bounding box is set as the pivot point. The pivot point is shown by a yellow circle.

Note that even if the above examples use meshes, the same rules apply for other types (curves, surfaces...) as well.

## Quick Rendering

### What is rendering?

Rendering is the process of creating a 2D image. Blender creates this image by taking into account your model and all of your materials, textures, lighting and compositing.

- There are two main types of rendering engines built inside Blender, one for *Full render*, and other for *OpenGL render*. This page shows you basic information about rendering Images. For a deeper understanding about the *Full Render* Engine built inside Blender, called *Blender Internal*, consult the section about [Rendering with Blender Internal](#).
- There is also a section in this wiki manual dedicated to the new [Cycles](#) Render Engine.

### Rendering an image using “Full Render” - Blender Internal

#### Reference

Mode: All modes

Hotkey: F12

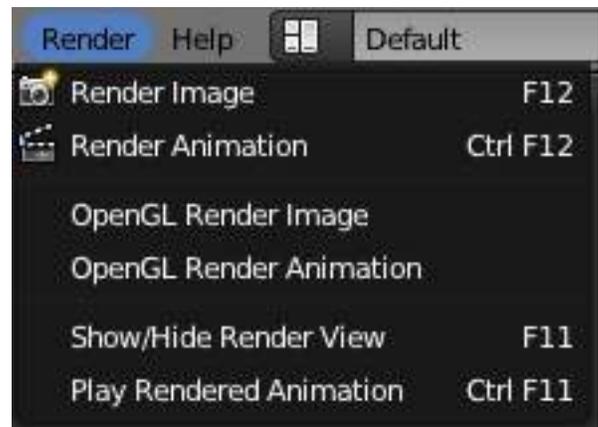


Fig. 1.152: Header of the Info Window

To start a *Full render* using *Blender Internal* you can use any of the following options:

- Press F12
- Go to *Properties Window* → *Render context* → *Render panel* and press the *Image* button
- Go to *Render* → *Render Image* from the header of the *Info Window* (see: *Header of the Info Window*)
- Using Blender Search: press *Spacebar*, type *Render* and click on *Render*.

To abort or quit the render, press *Esc*.

## Rendering an image using “OpenGL Render”

### Reference

Mode: All modes

Hotkey: Undefined -You can add one for your [Keymap](#)

To start an *OpenGL render* you can use any of the following options:

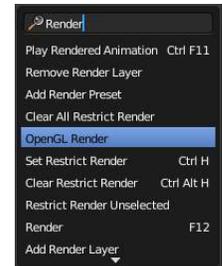


Fig. 1.153: Search functionality

- Click on *OpenGL Render Active Viewport*, in the header of the 3D Window, using the small button showing a *Camera* (together with a small image showing a *slate*) in the header of the 3D View
- Go to *Render* → *OpenGL Render Image* from the header of the *Info Window* (see: *Header of the Info Window Image*)
- Using Blender Search: press `Spacebar`, type *Render* and click on *OpenGL Render*.

To abort or quit the render, press `Esc`.

### Adjusting the resolution

The *Dimensions panel* of the *Render context* allows you to change the resolution. The default installation of Blender is set initially to **50%** of **1920 x 1080**, resulting in a **960 x 540** Image. (Highlighted in yellow, in *Dimensions Panel Image*.) Higher resolutions and high percentage scales will show more detail, but will also take longer to render.

### Output format and output file

You can also choose an output format and the output location for your rendered image or animation. By default they are saved in a temporary folder (*/tmp*), using an absolute path. You can set up your file paths using instructions in the [File setup chapter](#); however you can change this to a different folder by clicking the folder icon in the *Output panel*. You can also choose the type of image or movie format for your work from the Menu Button.

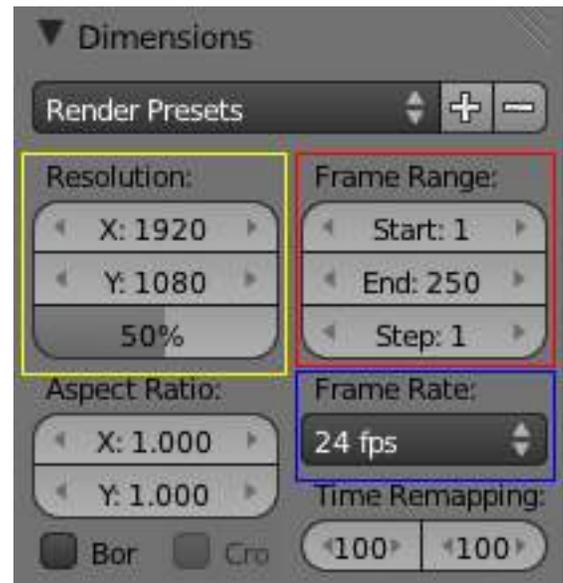


Fig. 1.154: Dimensions panel



Fig. 1.155: Output panel

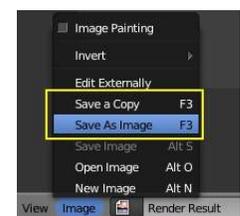


Fig. 1.156: Save as dialog

## Saving your image

Blender does not save your image automatically. To save your image, you can either press `F3` or click *Save As Image* from the *Image* menu of the UV/Image editor window's header. This action will open the Blender Internal File Browser, and then you can search for folders to place your Render.

## Rendering an animation using "Full Render" - Blender Internal

### Reference

Mode: All modes

Hotkey: `Ctrl-F12`

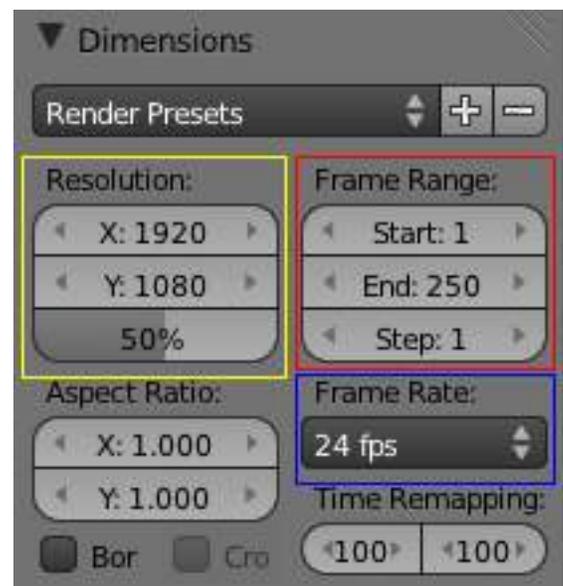


Fig. 1.157: Dimensions panel

Rendering an animation is simple; the *Frame Range* (Highlighted in red, in Dimensions Panel Image) in the Output Panel is used to define the **number of frames** your animation will render. The **time** is defined by the *Frames Per Second*, defined in the *Frame Rate* (Highlighted in blue, in Dimensions Panel Image) drop-down list. The default is set to **24 FPS** and **250** frames.

A quick example to understand those numbers:

- The Panel shows that the animation will start at frame **1** and end at frame **250**, and the FPS setting is set to **24**, so, the standard Blender installation will give you approximately **10** (ten) seconds of animation ( $250 / 24 = 10.41$  sec).

To render an animation using *Full Render* with the *Blender Internal* Engine, you can use any of the following options:

- Press `Ctrl-F12`
- Go to *Properties Window* → *Render context* → *Render panel* and press the *Animation* button or
- Go to *Render* → *Render animation* from the header of the *Info Window* (see: *Header of the Info Window* Image)

To abort or quit rendering the animation, press `Esc`.

---

## Rendering an animation using ‘OpenGL Render’

---

### Reference

Mode: All modes

Hotkey: Undefined - You can add one for your [Keymap](#)

---

To Render an animation using *OpenGL Render*, you can use any of the following options:



- Click on the small button showing a *slate* (together with a small image showing a *camera*) in the header of the 3D View
- Go to *Render* → *OpenGL Render animation* from the header of the *Info Window* (see: *Header of the Info Window Image*)

To abort or quit rendering the animation, press `Esc`.

## Showing Only Rendered Objects

---

### Reference

Mode: All modes

Hotkey: Undefined - You can add one for your [Keymap](#)

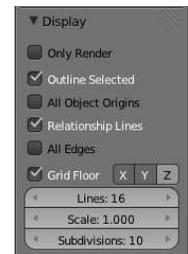


Fig. 1.158: Transform Panel - Display Tab.

At render time (either Full or OpenGL), there are some Objects in the scene that won't be rendered, either because of their type (Bones, Empties, Cameras, etc.), because they are void or have no visible geometry (Mesh without any vertex, curves not extruded, etc.), or simply because they are set as not renderable.

Blender has an option to only show Objects in the Scene that will be rendered.

To access this option, put your Mouse in a 3D View (focusing on it), use shortcut `N` or click in the `+` sign in the upper right side, to show the *Transform Panel*. Rolling through the options, you will find the *Display* tab, whose options are for controlling how Objects are displayed in the 3D View.

Just enable the *Only Render* option - now, only Objects that will be rendered will be shown (see Fig: Transform Panel - Display Tab). This option also works when generating Images using OpenGL Render. Note that all of the other options for selective displaying will be disabled.

### The purposes of OpenGL Rendering

OpenGL rendering allows you to quickly inspect your animatic (for things like object movements, alternate angles, etc.), by giving you a draft quality rendering of the current viewport.

Because it is only rendered using OpenGL, it is much faster to generate, even if it only looks as good as what you see in the 3D viewport.

This allows you to preview your animation with fluid playback, which you would otherwise not be able to do in real time due to scene complexity (i.e., pressing `Alt-A` results in too low of a *Frames Per Second* to get a good feel for the animation).

This is an example of an OpenGL rendered image:

And then here is the *Full Render* using Blender Internal render engine:

You can use OpenGL to render both images and animations, and change dimensions using the same instructions explained above. As with a normal render, you can abort it with `Esc`.

### Recovering from mistakes or problems

Blender provides a number of ways for the user to recover from mistakes, and reduce the chance of losing their work in the event of operation errors, computer failures, or power outages. There are two ways for you to recover from mistakes or problems:

At the *User Level* (Relating to *Actions*)

- For your actions, there are options like *Undo*, *Redo* and an *Undo History*, used to roll back from mistakes under normal operation, or return back to a specific action.
- Blender also has new features like *Repeat* and *Repeat History*, and the new *Redo Last* which you can use in conjunction with the options listed.

At the *System Level* (Relating to *Files*)

- There are options to save your files like *Auto Save* that saves your file automatically over time, and *Save on Quit*, which saves your Blender file automatically when you exit Blender.

---

**Note:** In addition to these functions being enabled by default, the *Save on Quit* functionality cannot be disabled.

---

### Options for Actions (User Level)

The commands listed below will let you roll back an accidental action, redo your last action, or let you choose to recover to a specific point, by picking from a list of recent actions recorded by Blender.

#### Undo

#### Reference

---

Mode: All modes

Hotkey: `Ctrl-Z`

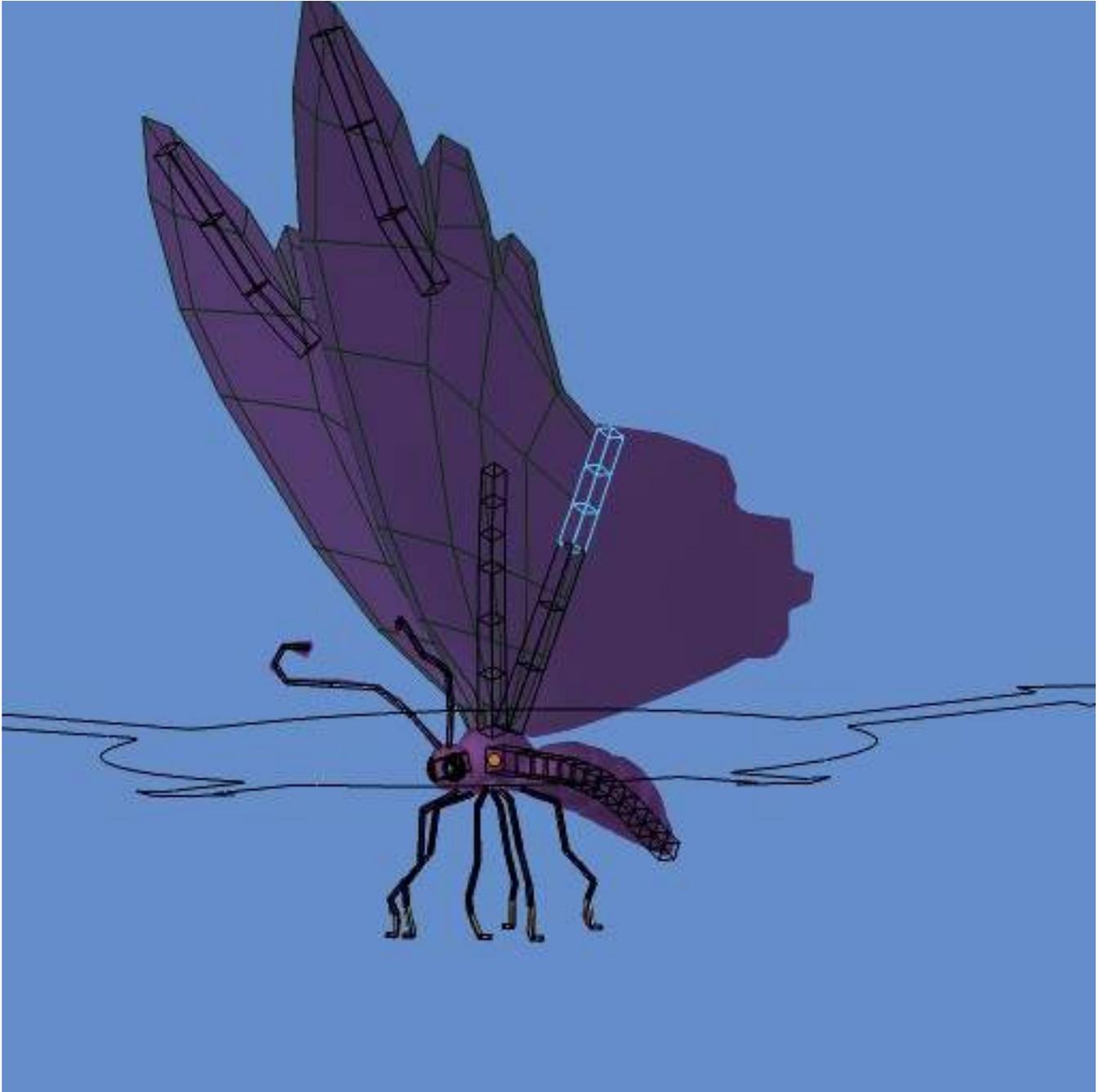


Fig. 1.159: OpenGL Render



Fig. 1.160: Full Render

---

If you want to undo your last action, just press `Ctrl-Z`

See [Editing Preferences](#) section on undo to change defaults.

---

## Redo Reference

Mode: All modes

Hotkey: `Ctrl-Shift-Z`

---

To roll back your Undo action, press `Ctrl-Shift-Z`

---

## Redo Last Reference

Mode: All modes

Hotkey: `F6`

---

*Redo Last* is short for *Redo(ing your) Last (Action)*. Hitting `F6` after an action will present you a context-sensitive Pop-Up Window based on your last action taken and the *Mode* and *Window* in which Blender is being used.

For example, if your last action was a rotation in *Object Mode*, the Window will show you the last value changed for the angle (see Fig:Redo Last - Rotation), where you can change your action back completely by typing **0** (zero) in the numeric field. There are other useful options, based on your action context, and you can not only Undo actions, but change them completely using the available options.

If you are in *Edit Mode*, the Window will also change its contents based on your last action taken. In our second example (at the right), the last action taken was a Vertex Move; we did a *Scale* on a Face, and, as you can see, the contents of the Pop-Up Window are different, because of your context (*Edit Mode*). (See Fig:Redo Last - Scale)

<b>Left Image:</b> Redo Last - Rotation ( Object Mode, 60 degrees )
<b>Right Right:</b> Redo Last - Scale ( Edit Mode, Resize face )

**Tip:** Operations using Redo Last

Some operations produce particularly useful results if you tweak their parameters with the `F6` Menu. Take, for example, adding a Circle. If you reduce the Vertex count to 3, you get a perfect equilateral triangle.

---

## Undo History Reference

Mode: All modes

Hotkey: `Ctrl-Alt-Z`

---

There is also a Undo History of your actions, recorded by Blender. You can access the history with `Ctrl-Alt-Z`.

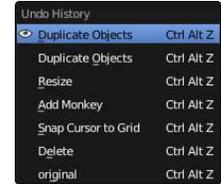


Fig. 1.161: The Undo History menu, which appears upon `Ctrl-Alt-Z` press.

Rolling back actions using the *Undo History* feature will take you back to the action you choose. Much like how you can alternate between going backward in time with `Ctrl-Z` and then forward with `Ctrl-Shift-Z`, you can hop around on the Undo timeline as much as you want as long as you do not make a new change. Once you do make a new change, the Undo History is truncated at that point.

---

### Repeat Last Reference

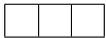
Mode: All modes

Hotkey: `Shift-R`

---

The Repeat Last feature will Repeat your last action when you press `Shift-R`.

In the example Images below, we duplicated a *Monkey Mesh*, and then we moved the Object a bit. Using repeat `Shift-R`, the *Monkey* was also duplicated and moved.




---

### Repeat History Reference

Mode: All modes

Hotkey: `F3`

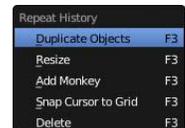


Fig. 1.168: The Repeat menu, which appears upon `F3` press.

The *Repeat History* feature will present you a list of the last repeated actions, and you can choose the actions you want to repeat. It works in the same way as the Undo History, explained above, but the list contains only repeated actions. To access Repeat History, use `F3`.

---

**Note:** There are two separate Histories for Blender

Blender uses two separate Histories, one dedicated for the *Edit Mode*, and one dedicated for the *Object Mode*.

---

## Blender Search Reference

Mode: All modes

Hotkey: Spacebar

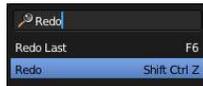


Fig. 1.169: Spacebar search for Redo Last

You can always access all of the explained options for user actions, using Blender Search Spacebar.

**Important:** When you quit Blender, the complete list of user actions will be lost, even if you save your file before quitting.

## Options for Files (System Level)

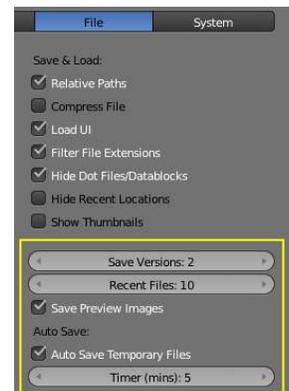


Fig. 1.170: Auto Save options

**Save and Auto Save** Computer crashes, power outages or simply forgetting to save can result in the loss or corruption of your work. To reduce the chance of losing files when those events occur, Blender can use an *Autosave* function. The *File* tab of the *User Preferences* window allows you to configure the two ways that Blender provides for you to regress to a previous version of your work.

**Save on Quit** The function *Save on Quit* is enabled by default in Blender. Blender will always save your files when you quit the application under normal operation.

**Save Versions** This option tells Blender to keep the indicated number of saved versions of your file in your current working directory when you manually save a file. These files will have the extension: `.blend1`, `.blend2`, etc., with the number increasing to the number of versions you specify. Older files will be named with a higher number. e.g. With the default setting of **2**, you will have three versions of your file: `*.blend` (your last save), `*.blend1` (your second last save) and `*.blend2` (your third last save).

**Auto Save Temporary Files** Checking this box tells Blender to *automatically* save a backup copy of your work-in-progress to the Temp directory (refer to the *File* panel in the *User Preferences* window for its location). This will also enable the *Timer (mins)* control which specifies the number of minutes between each Auto Save. The default value of the Blender

installation is **5** (5 minutes). The minimum is **1**, and the Maximum is **60** (Save at every one hour). The Auto Saved files are named using a random number and have a `.blend` extension.

---

**Tip:** Compress Files

The option to Compress files will try to compact your files whenever Blender is saving them. Large Scenes, dense Meshes, big Textures or lots of elements in your Scene will result in a big `.blend` being created. This option could slow down Blender when you quit, or under normal operation when Blender is saving your backup files. In fact, using this option you will trade processor time for file space.

---

### Recovering Auto Saves

**Recover Last Session** *File* → *Recover Last Session* will open the quit `.blend` that is saved into the *Temp* directory when you exit Blender. Note that files in your *Temp* directory are deleted when you reboot.



Fig. 1.171: Blender File Browser

---

**Tip:** When recovering files, you will navigate to your temporary folder. It is important, when browsing, to enable the detailed list view. Otherwise, you will not be able to figure out the dates of the auto-saved `.blends`. (See Figure: Blender File Browser)

---

**Recover Auto Save** *File* → *Recover Auto Save...* allows you to open the Auto Saved file. After loading the Auto Saved version, you may save it over the current file in your working directory as a normal `.blend` file.

---

**Important:** When recovering an Auto Saved file, you will lose any changes made since the last *Auto Save* was performed. Only **one** Auto Saved file exists for each project (i.e. Blender does not keep older versions - hence you won't be able to go back more than a few minutes with this tool).

---

### Other options

**Recent Files** This setting controls how many recent files are listed in the *File* → *Open Recent* sub-menu.

**Save Preview Images** Previews of images and materials in the *File Browser* window are created on demand. To save these previews into your `.blend` file, enable this option (at the cost of increasing the size of your `.blend` file).

### Setting the Startup File

---

#### Reference

Mode: All modes

Menu: *File* → *Save Startup File*

Hotkey: `Ctrl-U`

---

When you start Blender or start a new project with the menu entry *File* → *New*, a new scene is created from the default scene included with Blender.

This default scene can instead be your own customized setup.

To change the default scene, make all of the desired changes to the current scene or current file and *File* → *Save Startup File*.

## Restoring to Factory Settings

### Reference

Mode: All modes

Menu: *File* → *Load factory Settings*

Hotkey: Undefined, you can add one for your [Keymap](#)

To restore the startup file to the factory settings, open *File* → *Load Factory Settings*. This will restore the *Startup File* and *User Preferences* back to the original factory settings. After this you can save the startup file to overwrite any customizations permanently.

**Note:** User Preferences Window

For more information about the Editor Window for User Preferences or how to clean your preferences manually, see [User Preferences](#)

## Blender Screenshots

### Reference

Mode: All modes

Hotkey: `Ctrl-F3`



Fig. 1.172: Save Screenshot Option

`Ctrl-F3` will take a screenshot of your Blender window and then open the Blender *File Browser* window, allowing you to specify the name and location of the screenshot. In the example image at the right, the PNG format will be the output of the screenshot taken (settings are the same as the ones available to save render results). When the Blender *File Browser* window opens for you, at the left, there is a tab called *Save Screenshot* where you can find format settings and a checkbox with the option *Full Screen*.

- Check the Option to save the entire Blender window (full width and height of the Blender window you are using when you call the command).

- Uncheck the box to save only your active window (where your mouse is located when you call the command).

## Blender Screencasts

### Reference

Mode: All modes

Hotkey: Alt-F3

The shortcut Alt-F3 starts the screencast function. Screencasts will record your actions over time either as a video or sequence of image files. The type and location of the output is determined by the settings in the [Output panel](#) of the `Render` context window. The default settings will generate a screencast consisting of a series of PNG images captured every 50 ms and stored in the `/tmp` folder. If you want to record a video, set the *Output* to one of the *Movie File Formats* supported by your system listed in the *Output panel* format menu. If you are unsure what video codecs your system supports, select *AVI JPEG*.

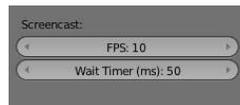


Fig. 1.173: Options in the User Preferences Editor

The FPS for video Screencasts and time between each Screenshot for an image series Screencast can be set from the [System panel](#) of the [User Preferences](#) window.

(See Fig: Options in the User Preferences Editor)

#### **Note:** Audio support

Blender Screencast doesn't support audio recordings, so you will have to do it manually using other software, e.g. [Audacity](#), in conjunction with Blender.

When you start Blender Screencasts, the header of the *Info Window* will change, and it will show you a button for stopping your capture.

Below, we show the normal header of the *Info Window*, when in normal Blender operation (See Fig: Info Window - Header - Normal Operation), and with the Stop button for the Screencast, when in Screencast Mode. (See Fig: Info Window - Header - Capture Stop Button).

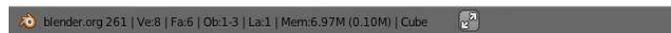


Fig. 1.174: Info Window - Header - Normal Operation

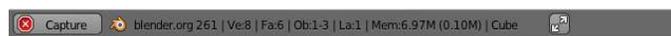


Fig. 1.175: Info Window - Header - Capture Stop Button

**Note:** The only way to stop the Screencast

Pressing the Stop button in the header of the Info Window is the only way to stop the Screencast capture. If you press `ESC`, the shortcut will only work for operations performed in the Blender *User Interface*, (it will stop animations, playbacks and so on...), but will not work to stop *Screencasts*.

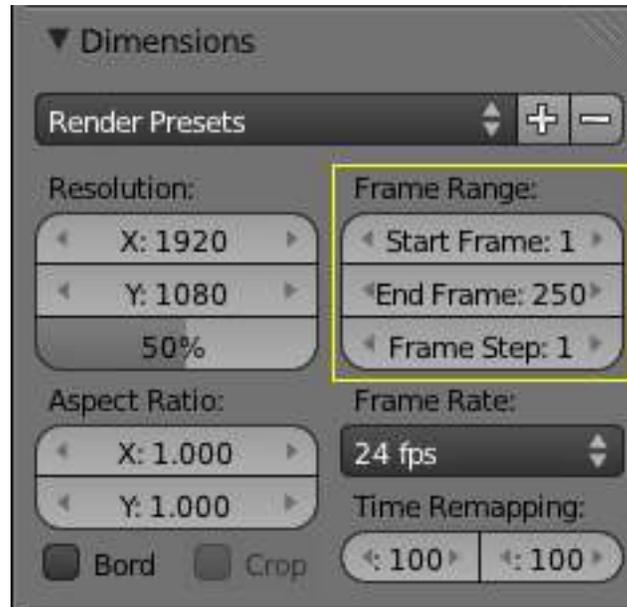


Fig. 1.176: Dimensions Panel - Frame Range

The frames are stored using a suffix added to their file name, where the suffix is composed of the numbers present in the fields for *start* and *end frames*, defined in the Frame Range of the Dimensions panel, Render context. (See Fig: Dimensions Panel - Frame Range - highlighted in yellow)

**Note:** The configuration of the End frame, present in the Frame Range of the Dimensions Panel, **will not** stop your capture automatically. You will always have to stop the Screencast manually, using the Stop button.

The Videos are generated internally in the same manner as the *Screenshots*, using the width and height of the Window you are working in. If you choose to capture to a Video file, Blender will have to pass those frames to a Video codec.

**Warning:** Some codecs limit the output width/height or the video quality.

- When you save your *Screencast* in an Image format, the Images will be saved using the entire Blender Window, with full width and height, and the quality of the Image will be defined by its type (i.e. JPG, PNG, and so on) and configuration (i.e. Slider *quality* of the .JPG format).
- When you save your *Screencast* in a Video format, it will be sent to a codec. Depending on the codec limitations, the resulting output Video could be scaled down. Furthermore, some combinations of Window width and height cannot be processed by certain codecs. In these cases, the *Screencast* will try to start, but will immediately stop. In order to solve this, choose another Window format and/or another codec.

### Blender Window Dimension

There is a way to match the Blender Window dimensions with the Output Video File, achieving standard dimensions for the output of the Blender Screencast. (I.e. NTSC, HD, Full HD, etc). You can control the width and height of your Blender Window, starting Blender from a Command Line. To learn more about starting Blender from a command line, see the page about [Blender Console Window](#).

## Help system

### Reference

Mode: All modes

Menu: *Help*

Blender has a range of built-in and web-based Help options.

### General Web-based Help Options

#### **Tip:** Browser and Internet Connection

Some forms of Help start up your web browser and access the Blender Foundation’s web servers. In order to do this, you must have configured a default web browser for your Operating System, and have a connection to the Internet.

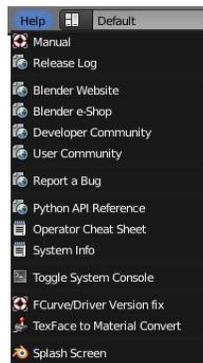


Fig. 1.177: Help menu

- **This is a link for the Official Blender Manual**, which you are now reading.
- **Release Log** - The release notes on the Web for the current Blender version.
- **Blender Website** - The *blender.org* home page.
- **Blender e-Shop** - The Blender e-Store, where you can buy Training DVD’s, books, t-shirts and other products.
- **Developer Community** - The *blender.org* “Get Involved” page. This is the launch page for Blender software development, bug tracking, patches and scripts, education and training, documentation development and functionality research.
- **User Community** - Lists of many different support venues here.
- **Report a Bug** - The Blender Bug Tracker page.

**Important:** in order to Report a Bug, you must register at the website.

### Programming Options

- **Python API Reference** - Python application programming interface (API) that Blender and Python use to communicate with each other. Useful for the Blender Game Engine, Customizing, and other scripting.

Access from *Help* → *Operator Cheat Sheet*

Creates the `OperatorList.txt` text-block, which you can access in the *Text Editor*. You can also use Blender Search to generate the file. The text will list the available Python operators.

While Blender is generating this list, the *Info Window* will change, showing a message for the operation (See Fig: Info Window - Operator Cheat Sheet ). To read the Text, switch to the Blender *Text Editor* Window, using the [Editor type Selector](#), and then, clicking on the button *Browse Text to be Linked* of the Text Editor, your text block will be shown in the Editor. The file will be in your list of Text-block, named as `OperatorsList.txt`, if the file is already generated, Blender will add a numeric suffix for the subsequent ones.

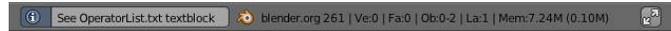


Fig. 1.178: Info Window - Operator Cheat Sheet

## Diagnostics Options

Access *Help* → *System Info*

Creates a `system-info.txt` text block, which you can access in the Blender *Text Editor*. The text lists various key properties of your system and Blender, which can be useful in diagnosing problems.

To read the Text, switch to the Blender *Text Editor* Window, using the [Editor type Selector](#), and then, clicking on the button *Browse Text to be Linked* of the Text Editor, your text block will be shown in the Editor. The `system-info.txt` will be in your list of Text-blocks.

The text file contains sections:

**Blender** This section of the info.txt shows you the Blender version, flags used when Blender was compiled, day and time when Blender was compiled, build system, and the path in which Blender is running.

**Python** The Python version you are using, showing the paths of the Python programming language paths.

**Directories** The Blender directories setup for scripts, user scripts, datafiles, config, scripts (internal), autosave directory and temp dir. Those directories are configured using the [User Preferences Editor Window](#).

**OpenGL** This section will show you the version of OpenGL that you are using for Blender, the name of the manufacturer, version, vendor and a list with your card capabilities or OpenGL software capabilities.



Fig. 1.179: Info Window - Info.txt

- *Toggle System Console* - Reveals the command window that contains Blender's *stdout* messages. Can be very useful for figuring out how the UI works, or what is going wrong if you encounter a problem. Even more information is available here, if you invoke Blender as *blender -d*. This menu item only shows up on Windows.
  - In all Operating Systems, to see this information, simply run *blender* from the command-line.
  - On Linux, if you ran Blender from the GUI, you can see the output in `~/.xsession-errors`
  - On Mac OS X, you can open `Console.app` (in the Utilities folder in Applications) and check the Log there.
- *Info Window Log* - This is not exactly a Help menu, but it is related. If you mouseover the line between the Info window and the 3D then click and drag the Info window down a bit, you can see the stream of Python calls that the UI is making when you work. This can be useful in creating scripts.

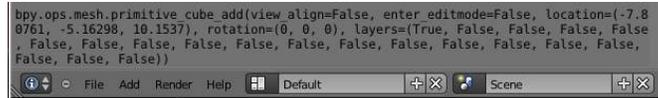


Fig. 1.180: The Info Window Log after adding a Cube

## Legacy Version Support

**FCurve/Driver fix** Sometimes, when you load .blend's made from older versions of Blender (2.56 and previous), the Function Curves and Shapekey Drivers will not function correctly due to updates in the animation system. Selecting this option updates the FCurve/Driver data paths.

**TexFace to Material Convert** Convert old Texface settings into material. It may create new materials if needed.

## Splash Screen

Access this by clicking on the Blender icon in the Info Window's header.

This displays the image where you can identify package and version. At the top-right corner, you can see the Version and SVN (Subversion) revision (See Fig: Blender Splash Screen). For example, in our Splash Screen, you can see the version **2.66.0** and the revision number **r54697**. This can be useful to give to support personnel when diagnosing a problem.

There are some Internet Based Help options that are also present in the Blender *Splash Screen*. They are presented as the same links you will find at the *Help* Menu.



Fig. 1.181: Blender Splash Screen, Blender Version 2.66

**Other Help Options** Here we explain the two new features added for Blender, *Blender Search* and the recoded *Tooltips*.

## Blender Search

---

## Reference

Mode: All modes

Hotkey: Spacebar



Fig. 1.182: Blender Search - Render

The Blender Search feature, called the *Search Menu*, Activate by pressing `Spacebar`, Blender will present you with a search pop-up, no matter at which Blender Editor your Mouse pointer is located (except the *Text Editor* Window and *Python console*), and a field for you to type in. Just type what you need and Blender will present you a list of available options. You can click on the appropriate function for you, or search through them using your keyboard, type `Return` to accept, or `Esc` to leave. Clicking outside of the Blender Search Window or taking the Mouse pointer away, will also leave Blender Search.

The Image at the right shows Blender Search when we type the word *Render* inside the field. If you continue typing, your search keywords will refine your search and if no named operator can be found, the small Pop Up Window for the Blender Search will stay blank.

### How it works

Every Blender Internal Operator can use a defined name, some of them are predefined names for the user. For example, the *Render* command is a named Python call, the appropriate Operator is `Python: bpy.ops.render.render()`, but for the user, it is called *Render*. All of those *user* names that were previously attributed for Python operators can searched for using *Blender Search*.

### Tooltips

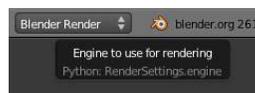


Fig. 1.183: The Mouse pointer was Stopped for a while over the Render Engines List in the Info Window. The normal Tooltip is in white and the Python operator is displayed in grey

The *Tooltips* in Blender were completely recoded, and every time you hover your Mouse over a Button, a Command, Numeric Fields or things that are related to Operators, staying for a while, it will show you not only the normal Tooltip, but also the specific related operator. Those operators are useful for lots of tasks, from Python Scripts to Keymaps. In the example Image at the right, we pointed our Mouse over the Info Window, specifically over the list of the Render engines available, waited for a while, and the Tooltip with the appropriate operator was shown. In our example, it shows the Tooltip *Engine to Use for Rendering* in white, and `Python: RenderSettings.engine` in grey, which is the Operator associated with the function.

## 2.1 Editors

Blender provides a number of different Editor types for displaying and modifying different aspects of data.

It is also possible to open the same Editor type multiple times and thus show different views on the same data in parallel. For example you can open the 3D View Editor 4 times to show the current scene from the top, front, side and Camera perspective at the same time.

[Read more about arranging frames](#)

### 2.1.1 Info

The Info Editor is found at the top of the Default Scene and has the following components:

**Editor Type Selector** The red shaded area allows you to change the [Editor Type](#). This region is found on every Window.

**Menu options** The dark blue shaded area provides access to the main menu options.

**Current Screen (default is Default)** The green shaded area allows you to select different [Screens](#). By default, Blender comes with several pre-configured *Screen* s for you to choose from. If you need custom screen layouts, you can create and name them.

**Current Scene** The yellow shaded area allows you to select different [Scenes](#). Having multiple Scenes allows you to work with separate virtual environments, with completely separate data, or with objects and/or mesh data linked between them. (In some 3D packages, each file contains one scene, while in Blender, one `.blend` file may contain several scenes.)

**Current Engine** The purple shaded area gives a list of available rendering and game engines.

**Resource Information** The aqua shaded area gives you information about Blender and system resources in use. This region will tell you how much memory is being consumed based on the number of vertices, faces and objects in the selected scene, as well as totals of what resources are currently selected. This can help identify when you are reaching the limits of your hardware.

### 2.1.2 Outliner

The *Outliner* is a list that organizes data in your scene. In the outliner, you can:

- View the data in the scene.
- Select and deselect objects in the scene.
- Hide or show an object in the scene.
- Enable or disable selection (to make an object “unselectable” in the 3D View).

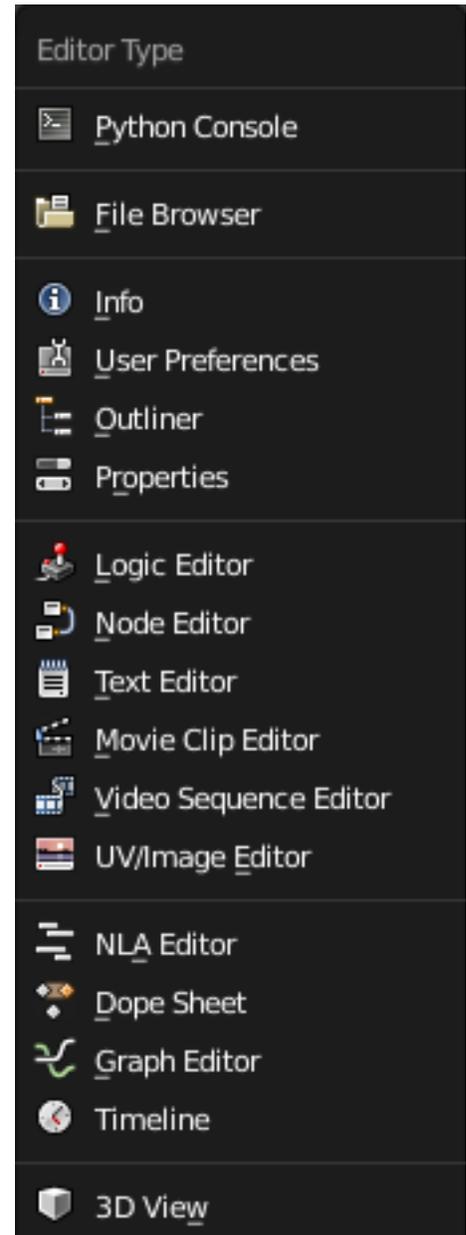


Fig. 2.1: The Editor selection menu.

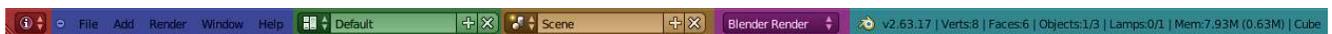


Fig. 2.2: Info Window

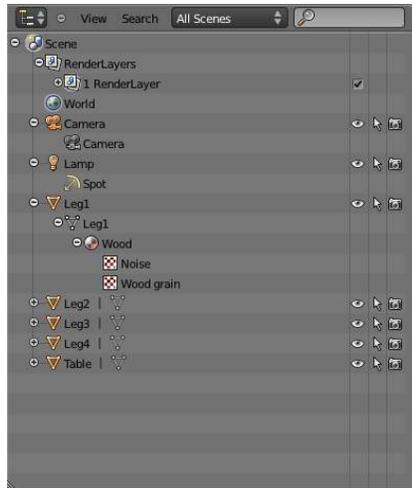


Fig. 2.3: The Outliner window.

- Enable or disable the rendering of an object.
- Delete objects from the scene.
- Unlink data (equivalent to pressing the X button next to the name of a datablock).
- Easily select which render layer to render.
- Easily select which render pass to render (for example, you can choose to render just the *Specular* pass).

## Using the Outliner

Each row in the *Outliner* shows a datablock. You can click the plus-sign to the left of a name to expand the current datablock and see what other datablocks it contains.

You can select datablocks in the *Outliner*, but this won't necessarily select the datablock in the scene. To select the datablock in the scene, you have to activate it.

## Selecting and Activating

Single selection doesn't require any pre-selection: just work directly with LMB (and/or RMB - contextual menu, see below) *inside* the name/icon area.

When you select an object in the list this way, it is selected and becomes the active object in all other 3D Views. Use this feature to find objects in your 3D View, select them in the *Outliner*, then zoom to them with *View* → *Show Active* or `NumpadPeriod`.

**Activating a datablock** *Activate* the datablock with LMB on the *icon* of the datablock. Activating the datablock will automatically switch to the relevant mode. For example, activating the mesh data of the cube will select the cube and enter *Edit mode* while activating the object data of the cube will select the cube and enter *Object mode* (see right).

**Toggle pre-selection of a group of datablocks** Useful when you want to select/deselect a whole bunch of datablocks. For this you must prepare the selection using, to your liking:

- RMB or LMB,
- Shift-RMB or Shift-LMB,
- RMB and drag or LMB and drag,

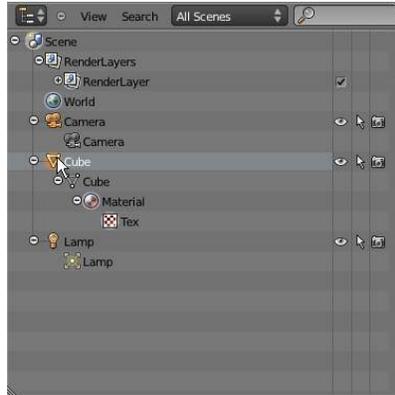


Fig. 2.4: Click LMB on the mesh data of the cube to activate Edit mode.

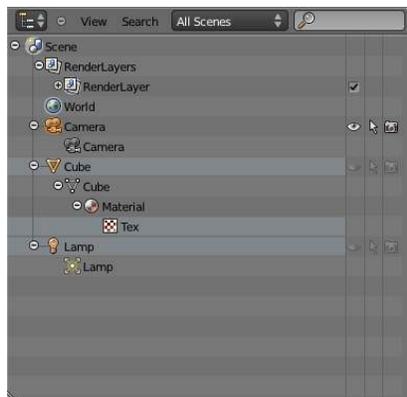


Fig. 2.5: Toggling pre-selection of a datablock.

all *outside* the name/icon area. Those pre-selected have their line in a lighter color. You then can (de)select them with a RMB *on* the name/icon area, which brings on a context menu (see below).

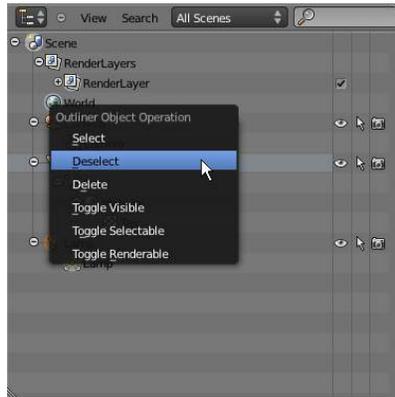


Fig. 2.6: Context menu for the Cube object.

**Context menu** Show the context menu for a datablock with RMB on the icon or name. Depending on the type of the pre-selected datablock(s), you will have all or part of the following options:

- *Select*.
- *Deselect*.
- *Delete*.
- *Unlink* - To unlink a datablock from its “owner” (e.g., a material from its mesh).
- *Make Local* - To create a “local” duplicate of this datablock.

---

**Note:** Some datablock types will not have a context menu at all!

---

**Deleting a datablock** Use X to delete the selected datablock(s).

**Expanding one level** Use NumpadPlus to expand one level down in the tree-list.

**Collapsing one level** Use NumpadMinus to collapse one level up in the tree-list.

**Expanding/collapsing everything** Use A to expand/collapse all levels of the tree-list.

### Toggle object-level restrictions

The three following options, in the right side of the *Outliner* window, are only available for objects:

**Visibility (*eye icon*)** Toggles the visibility of the object in the 3D View. V will toggle this property for any objects that are selected in the *Outliner*.

**Selectability (*mouse cursor icon*)** This is useful for if you have placed something in the scene and don’t want to accidentally select it when working on something else. S will toggle this property for any objects that are selected in the *Outliner*.

**Rendering (*camera icon*)** This will still keep the object visible in the scene, but it will be ignored by the renderer. R will toggle this property for any objects that are selected in the *Outliner*.

### Searching

You can search the file for datablocks, either by using the *Search* menu in the header of the *Outliner*, or by using one of the following hotkeys:

- F - Find.
- Ctrl-F - Find (case sensitive).
- Alt-F - Find complete.
- Ctrl-Alt-F - Find complete (case sensitive).
- Shift-F - Find again.

Matching datablocks will be automatically selected.

## Filtering the display

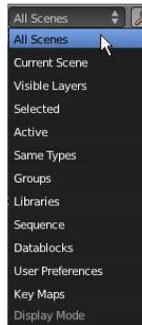


Fig. 2.7: Outliner Display dropdown.

The window header has a field to let you select what the outliner should show to help you narrow the list of objects so that you can find things quickly and easily.

**All Scenes** Shows *everything* the outliner can display (in all scenes, all layers, etc.)

**Current Scene** Shows everything in the current scene.

**Visible Layers** Shows everything on the visible (currently selected) layers in the current scene. Use the [layer](#) buttons to make objects on a layer visible in the 3D window.

**Selected** Lists only the object(s) currently selected in the 3D window. You can select multiple objects by `Shift-RMB` - clicking.

**Active** Lists only the active (often last selected) object.

**Same Types** Lists only those objects in the current scene that are of the same types as those selected in the 3d window.

**Groups** Lists only [Groups](#) and their members.

**Libraries** TODO

**Sequence** TODO

**Data Blocks** TODO

**User Preferences** TODO

**Key Maps** TODO

### 2.1.3 Properties

TODO - see: <https://developer.blender.org/T42899>

## Contexts

The *Properties (or Buttons) Window* shows several *Contexts*, which can be chosen via the icon row in the header (see *Context button example*).



Fig. 2.8: Context button example

The number and type of buttons changes with the selected Context so that only useful buttons show up. The order of these buttons follows a hierarchy which is detailed below:

-  **Render:** Everything related to render output (dimensions, anti-aliasing, performance etc).
-  **Scene:** Gravity in the scene, units and other general information.
-  **World:** Environmental lighting, sky, mist and Ambient Occlusion.
-  **Object:** Transformations, display options, visibility settings (via layers) duplication settings and animation information (regarding Object position).
-  **Constraints:** Used to control an Object's transform (position, rotation, scale), tracking and relationship properties.
-  **Modifiers:** Operations that can non-destructively affect Objects by changing how they are rendered and displayed without altering their geometry (e.g. mirror and smoothing).
-  **Object Data:** Contains all Object specific data (color of a lamp, focal length of a camera, vertex groups etc). The icon differs with the type of Object (the one shown here is for a mesh object).
-  **Materials:** Information about a surface (color, specularility, transparency, etc).
-  **Textures:** Used by materials to provide additional details (e.g. color, transparency, fake 3-dimensional depth).
-  **Particles:** Add variable amounts of (usually small) objects such as lights or mesh Objects that can be manipulated by Force Fields and other settings.
-  **Physics:** Properties relating to Cloth, Force Fields, Collision, Fluid and Smoke Simulation.

The **Buttons** in each context are grouped into **Panels**.

## 2.1.4 Node Editor

### Introduction

TODO - see: <https://developer.blender.org/T43570>

## 2.1.5 Video Sequence Editor

### Introduction

In addition to modeling and animation, Blender has a fully functional Video Sequence Editor (VSE) as well as an advanced node-based editor that also manipulates a video stream. [Compositing Nodes](#) operate equally well on images or video streams, and can apply detailed image manipulation on the stream.

Operating at a higher conceptual level, and used later in the video production process, Blender's legacy VSE operates on a set of entire strips at a time, as a chunk of footage. The many parts of Blender work together in typical work flow fashion:

- Model to construct the objects
- Assign Materials and introduce Lighting to color the objects
- Animate your objects to make them move
- Render layers of video using cameras
- Use Compositing nodes to: - Enhance the images by adjusting colors, adding in-scene special effects - Layer the images into a composite image sequence (strip)
- Assemble the video strips together to make a movie using the VSE.

The VSE within Blender is a complete video editing system that allows you to combine multiple video channels and add effects to them. Its functionality has been inside Blender since the beginning. Even though it has a limited number of operations, you can use these to create powerful video edits (especially when you combine it with the animation power of Blender!) Furthermore, it is extensible via a plugin system to perform an unlimited number of image manipulations.

Using the VSE, you load multiple video clips and lay them end-to-end (or in some cases, overlay them), inserting fades and transitions to link the end of one clip to the beginning of another. Finally, add an audio track so you can synchronize the timing of the video sequence to match it. The result of using the VSE is your finished movie.

---

**Note:** FFMPEG Support

Support for exporting an avi/quicktime movie using FFMPEG does work, currently (since 2.44) only within the Linux and Windows builds. With FFMPEG support, you are able to save the audio track with your video.

---

### Overview of the Sequence Editor

Blender's Video Sequence Editor is a flexible workbench for editing your video footage. It is used to review your footage, and glue many sequences of your movie together. It offers a number of built-in and plug-in effects to transition from sequence to sequence, providing advanced hollywood-style effects for a professional looking video.

The Video Sequence Editor has a header (where the menu and view modes are shown) and a workspace, and works in one of several view modes. The Marker menu allows you to add markers in the VSE. Markers are shared across animation editors. See [Markers](#)

The sequencer workspace is horizontally striped into channels and each video strip will go in a horizontal channel. Each channel is numbered on the left-hand side, starting from 0 (you can't put anything thing in this special one!) and going up. Stripes toward the bottom are more dominant, which we'll get to in a minute. In the x direction, seconds of animation or frames

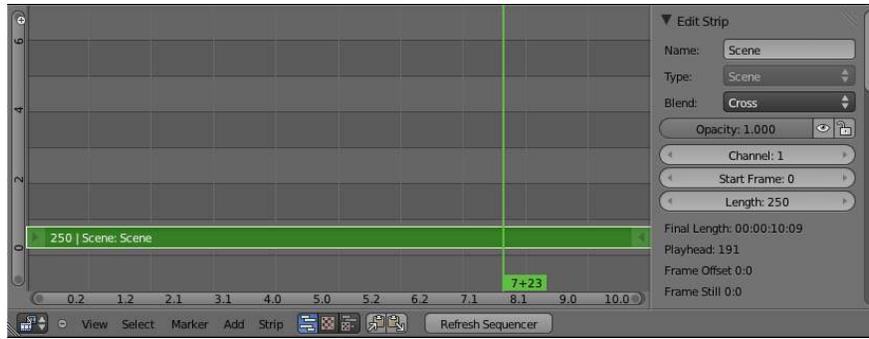


Fig. 2.9: Video Sequence Editor in Sequence display mode

of animation (`Ctrl-T` to choose) are used as the measure of time (seconds 1 through 7 are shown). You can scale the time using the zoom keys or mouse actions (see the Reference for more info).



Fig. 2.10: Sequence Output Enabled

When you click Render or Anim to generate an image or video, Blender has a choice of what image to compose for the current frame/scrub range:

- Current Scene layer result
- Sequence Editor channel 0 result
- Composition Node Editor renderlayer result

The video sequencer is enabled by default.

When you go to the render panel where ordinary renderings take place and you click animation or image with the VSE open, it will render the clips for the VSE instead of the scene.

## Using the Sequence Editor

**Adding Strips** The Add menu is the main menu you will be using to add content to the VSE. In general, you load up your strips, create strips of special transition effects, and then animate out your sequence by selecting “Do Sequence” and clicking the Anim button. You can use the Add menu in the header, or hover your mouse cursor over the Sequence workspace and press `Shift-A`.

**Note:** Clips can be Huge

A three minute quicktime .mov file can be 140Mega. Loading it, even over a high-speed LAN can take some time. Don't assume your computer or Blender has locked up if nothing happens for awhile.

First, let's add a clip:

- A movie clip in the Audio-Video Interleaved format (\*.avi file)
- A movie clip in the Apple QuickTime format (\*.mov)

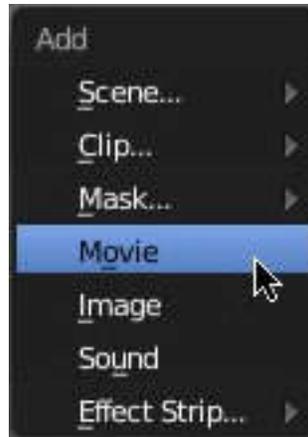


Fig. 2.11: the add menu

- A single still image to be repeated for a number of frames (\*.jpg, \*.png, etc.)
- A numbered sequence of images (\*-0001.jpg, \*-0002.jpg, \*-0003.jpg, etc, of any image format)
- One or more images from a directory
- A Scene in your .blend file.

Blender does not care which of these you use; you can freely mix and match any of them. They all become a color-coded strip in the VSE:

- Blue is used for Avi/mov codec strips
- Grey is a single image that is repeated/copied
- Purple is an image sequences or group of images played one after the other
- Green is an Audio track

When you choose to add one of these, the VSE window will switch to a file browser for you to select what you want to add. Supported files have a little rectangle next to their name (blue for images, green for clips) as a visual cue that you can pick them successfully:

**Add Movies or Images** When adding a Movie or Movie+Audio LMB to put the name of the file into the text box at the top; this selects a **single** file (like a movie)

In the case of (numbered) image **sequences**, you have a choice:

**Directory** RMB right-click on a directory name, and all files in that directory will be brought in as part of the image, in sort order, one image per frame

**Range** Navigate into the directory and right-click and drag over a range of names to highlight multiple files. You can page down and continue right-click-dragging to add more to the selection

**Batch** Shift-right-click selected non-related stills for batch processing; each image will be one frame, in sort order, and can be a mix of file types (jpg, png, exr, etc.)

**All** Press A to select/deselect All files in the directory.

When you click the *Select <whatever>* button, the window pane will switch back to VSE, and the strip will be rubber-banded to your mouse. You cannot load multiple movies at the same time by right-clicking them; no movies load if you right click them. Right-clicking only works for images.

In order to add items to the VSE, left-click for movies, left-click for single images, or right-click and drag for image sequences. Move your mouse to the frame/time and stripe you want, and click to break the rubberband and drop the strip in place (in a channel and starting at a frame).

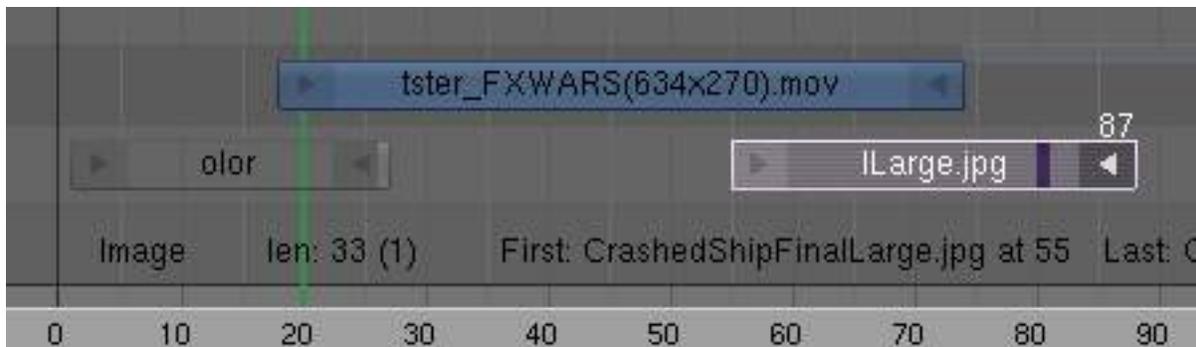
When you add an image, Blender makes it into a 50-frame strip, which means that image will be in your video for two seconds (at 25 fps - PAL). Aside from re-positioning it, you will want to scale it by RMB -clicking on either the start or end arrow, and dragging left or right. As you move, the frame number updates to say where the arrow is. Click LMB to validate, or RMB to cancel the modification.

---

**Tip:** Dealing with Different Sizes

Dealing with different sized images and different sized outputs is tricky. Think like a pixel. If you have a mis-match between the size of the input image and the render output size, the VSE does try to auto-scale the image to fit it entirely in the output. This may result in clipping. If you do not want that, use Crop and/or Offset in the Input panel to move and select a region of the image within the output. When you use Crop or Offset, the auto-scaling will be disabled and you can manually re-scale by adding the Transform effect.

---



If you scroll up the workspace, you will see an information channel (at vertical location channel 0) that gives you some helpful hints about the active strip. The example above shows a color strip from frames 1 to 25, then a mov file, and then an image strip. The info channel shows handy information about the image strip, whose name has been scrunched in the strip display, but is clearly spelled out in the information strip.

**Add Scene** You can add the virtual image output of a Scene in your current .blend file as well. Select the scene from the pop-up list, and a strip will be added and rubberbanded to your mouse just like a movie or image. The strip length will be determined based on the animation settings in that scene (not the current scene, unless the VSE is operating in the same scene).

When adding a Scene strip, please note that, in order to show you the strip in the VSE Image preview mode, Blender must render the scene. This may take awhile if the scene is complex, so there may be a delay between the time you select the scene and the time the strip appears. To reduce the delay, simplify the scene rendering by selecting fewer layers to render.

If the extra overhead of rendering the scene becomes burdensome (for either preview or for multiple test renders) and you have enough disk space consider rendering the scene to a sequence of PNGs and using an Image Sequence strip instead of a scene. This is very popular for static graphic overlays like title cards which are often little more than a static image with animated opacity.

**Add Audio** The VSE can incorporate an audio channel which you can hear as you scrub. Add an audio track when you are trying to time your video/animation to an audio track, or vice versa. Please refer to [the Audio Sequences section](#) for more information.

**Adding Effects** Blender offers two categories of effects: Built-in and Plug-in. The built-in effects are listed to the right. They are built-in to Blender and everyone has them. The plug-in effects are separate files in a sequence-plugin directory on your

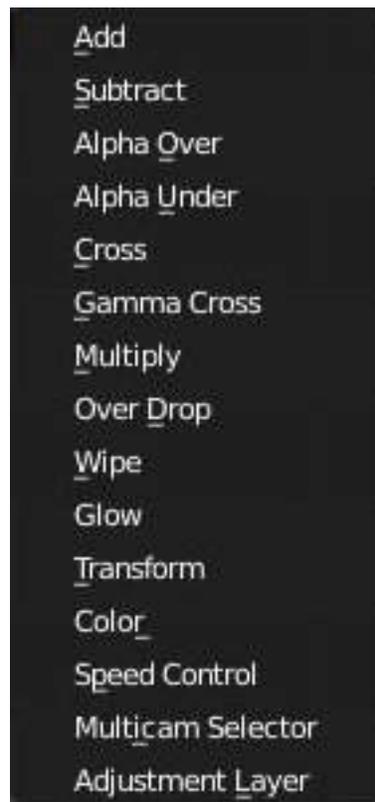


Fig. 2.12: Available Built-in Effects

PC that are loaded as they are needed. While a standard set of plugins are distributed when you installed Blender, everyone's computer may have a different set.

Every Built-in effect is explained in the next page individually, but they all are added and controlled in the same way. To add an effect strip, select one base strip (image, movie, or scene) by **RMB** clicking on it. For some effects, like the Cross transition effect, you will need to **Shift-RMB** a second overlapping strip (it depends on the effect you want). Then select **Add -> Effect** and pick the effect you want from the pop-up menu. When you do, the Effect strip will be shown above the source strips. If it is an independent effect, like the color generator (described later), it will be rubberbanded to your mouse; click to drop the strip.

Since most Effects strips depend on one or two source strips, their frame location and duration depends on their source strips. Thus, you may not be able to move it; you have to move the source strips in order to affect the effect strip.

To use an effect that combines or makes a transition between (or composites) two strips, you must **Box** select or **shift-right-click** two of them. When you add the effect strip, it will be placed in a channel above the two in **Grab** mode (click to drop it on a channel). Its duration will be the overlap between the two strips as a maximum.

With some effects, like the AlphaOver, the order in which you select the strips is important. You can also use one effect strip as the input or source strip with another strip, thus layering effects on top of one another.

---

**Note:** The only exception is the Color Generator effect. It does not depend on a base strip; you can add and position it independent of any other strip. Change the length as you would any strip.

---

## Reference

Mode: Sequence, Effects Strip Selected

Menu: Strip -> Change Effect

Hotkey: C

---

If you picked the wrong effect from the menu, you can always change it by selecting the strip (**RMB**) and using the **Strip->Change Effect** selection. Or, you can press **C** hange to switch effects on a selected Effects strip.

**Strip Properties** The properties for the strip are examined and set in the properties panel, shortcut **N**.

- Edit Strip - change properties of the strip
- Strip Input - where to pull images from
- Effect - Settings for effects strips
- Filter - Image pre-processing
- Proxy - Use representatives of the real image, for low-powered PCs
- Scene - Settings for when a scene strip is selected
- Sound - Settings for a sound clip

The panels for each of these sets of options and controls are shown to the right

## Edit Strip Panel

**Name** You can name or rename your strips here.

**Type** Displays the type of strip selected.

**Blend Mode** By default, a strip Replaces the output image of any lower-level strips. However, many other blending modes are available based on the strip type. For example, Alpha-Over automatically overlays the image on top of a lower level strip. Autoblending modes remove the need for separate effect strips. Blend percent controls how much of an effect the strip exerts, even over time.

**Opacity** Set the opacity of the strip.

**Mute** Hides the strip so that it does not participate in the final image computation

**Lock** Prevents the strip from being moved.

**Channel** Changes the channel number, or row, of the strip.

**Start Frame** Changes the starting frame number of the strip, which is the same as grabbing and moving the strip. Tip: when you add a strip, I like to just drop it and then use this field to place it at the frame I want, rather than trying to drag and drop in exactly the right place.

**Length** Specify the number of frames to use for the strip.

Use the *Convert to Premul* button if a strip has an Alpha (transparency) channel. Use *FilterY* if the strip is from broadcast video and has even or odd interlacing fields. Enhance the color saturation through the *Mul* multiply field. Play a strip backwards by enabling *Reverse Frames*. Tell Blender to display every nth frame by entering a *Strobe* value. Finally, when using MPEG video (VCD, DVD, XVID, DivX, ...), an image is built up over the course of a few frames; use the *Preseek* field to tell Blender to look backward and compose the image based on the n previous frames (e.g. **15** for Mpeg2 DVD).

**Effect Strip** For all effects, use the Strip Properties panel to control the effects strip; each effect has different controls, but they can all be set in the Properties panel. Control the length of the strip to vary the speed with which the transform happens. Regardless of whether they are built-in or plug-in, all effect strips do some special image manipulation, usually by operating on another strip or two in a different channel. The effect strip is shown in some channel, but its resultant effect shows up as Channel 0.

**Strip Input** Controls the source of the strip. Fields include file path, file name, image offset, crop settings.

This is here you can edit/update the path of the file used by a strip. Very useful when you moved it one way or the other - this avoids you deleting and re-creating the strip!

You have two text fields for path, the first being the path of the parent directory (*Path*), and the second the file name itself.

**Filter** Enables you to quickly set common image pre-processing options. *Strobe*

**Flip** X flips (reverses) the image left-to-right, Y reverses top-to-bottom.

**Backwards** Reverses strip image sequence

**De-Interlace** Removes fields in a video file.

**Saturation** Increase or decrease the saturation of an image.

**Multiply** Multiplies the colors by this value.

**Premultiply** Premultiply the Alpha channel.

**Convert Float** Converts input to float data.

**Use Color Balance** Provides three filters to adjust coloration: Lift, Gamma, and Gain. Each pass can have a positive, or inverted effect by clicking the appropriate button. Set the amount of the effect by setting the color swatch; white (RGB 1,1,1) has no effect.

**Proxy Strip Properties Panel** A proxy is a smaller image (faster to load) that stands in for the main image. When you *Rebuild proxy* Blender computes small images (like thumbnails) for the big images and may take some time. After computing them, though, editing functions like scrubbing and scrolling and compositing functions like cross using these proxies is much faster but gives a low-res result. Disable proxies before final rendering.

In order to actually *use* the proxies, the proper “Proxy Render Size” dropdown value must be selected in the Properties panel of the Sequencer View (where the edit plays back).

**Sound** This panel appears when a sound file is selected.

Here you can specify the Sound Strip’s file path and file name.

**Pack** Packs the sound file into the current .blend file.

**Caching** The sound file is decoded and loaded into RAM.

**Volume** Set the volume of the Sound file.

**Attenuation/dB** Attenuation in decibels

**Trim Duration Start/End** Offset the start and end of a sound strip.

**Scene** Specify the scene to be linked to the current scene strip.

**Sequencer** Process the render (and composited) result through the video sequence editor pipeline, if sequencer strips exist. This is the same function as in the render settings.

**Camera Override** Change the camera that will be used.

**Adjusting the View** Use these shortcuts to adjust the sequence area of the VSE: Pan `MMB` Zoom `Wheel` Vertical Scroll use `Shift-Wheel`, or drag on the left scroll bar. Horizontal Scroll use `Ctrl-Wheel`, or drag on the lower scroll bar. Scale View Vertically, drag on the circles on the vertical scroll bar. Scale View Horizontally, drag on the circles on the horizontal scroll bar.

As usual, the View Menu controls what and how you view in the workspace.

**Properties Panel** The Properties Panel contains options for the way the preview is displayed.

**View all Sequences Home** Zooms (out) the display to show all strips.

**Fit preview in Window Home** Resizes preview so that it fits in the window.

**Show Preview 1 : 1 Numpad1** Resizes preview to a 1 : 1 scale (actual size).

**View Selected NumpadPeriod** Zooms in the display to fit only the selected strips

Use this when working arranging a lot of strips and you want to use all of your screen to work.

---

## Reference

Mode: Sequence

Menu: View -> Show Frames, View -> Show Seconds

Hotkey: T

---

**Draw Frames** Displays the frame number instead of the time, in the Frame Number Indicator.

**Show Frame Number Indicator** Toggles the units of measure across the bottom of the workspace between seconds or frames.

**Safe Margin** Displays an overlay on the preview, marking where title safe region is.

**Separate Colors** When using Luma Waveform view, this separates R,G, and B into separate graphs.

**Transform Markers** Transform Markers as well as Strips.

**Scrubbing** To move back and forth through your movie, use the Timeline window. LMB click and drag left/right in the timeline window, moving the vertical bar which indicates the current frame. As you do, the image for that frame is displayed in the VSE window.

Real-time scrubbing and image display is possible on reasonable computers when viewing an image sequence or movie (avi/mov) file. Scene images have to be rendered individually, which may take some time.

**View Modes** The icons in the header allow to change the view of the VSE. By default, only the sequencer is displayed. The second button displays only the Preview window, and the third button displays both the Sequencer and the Preview.

When the preview is enabled, you have several options to change what type of preview to display. They are explained in the [Display Modes Page](#).

**Scene Preview** When using a Scene Strip in the sequencer, these settings in the Properties Panel determine how they are shown in the preview window.

**Open GL Preview** If you have Open GL, enable this setting to use Open GL for the scene preview renders. The drop down menu allows you to change how the Scene is displayed (Bounding Box, Wireframe, Solid, Textured).

**View Settings** The View Settings section in the properties panel contains addition display options.

**Show Overexposed** Increasing this number to 1 or greater displays a striped overlay to the preview image, showing where it is overexposed. A higher number gives a higher threshold for marking overexposure.

**Safe Margin** Displays an overlay on the preview, marking where title safe region is.

**Proxy Render Size** Draws preview using full resolution or different proxy resolutions. Render resolution is determined in the render settings panel. Using a smaller preview size will increase speed.

**Refresh View** Certain operations, like moving an object in 3D View, may not force the Sequencer to call for a refresh of the rendered image (since the movement may not affect the rendered image). If an image or video, used as a strip, is changed by some application outside of Blender, Blender has no real way of being notified from your operating system. To force Blender to re-read in files, and to force a re-render of the 3D View, click the Refresh button to force Blender to update and synchronize all cached images and compute the current frame.

**Selecting Strips** The Select Menu helps you select strips in different ways.

**Strips to the Left** Select all strips to the left of the currently selected strip.

**Strips to the Right** Select all strips to the right of the currently selected strip.

**Select Surrounding Handles `Alt-Ctrl-RMB`** Select both handles of the strip, plus the neighboring handles on the immediately adjoining strips. Select with this method to move a strip that is between to others without affecting the selected strip's length.

**Left Handle `Alt-RMB`** Select the left handle of the currently selected strip.

**Right Handle `Ctrl-RMB`** Select the right handle of the currently selected strip.

**Linked** Select all strips linked to the currently selected strip

**Select All `A`** Selects all the strips loaded.

**Select Inverse** Inverts the current selection.

**Border Select B** Begins the *Box* mode select process. Click and drag a rectangular lasso around a region of strips in your Sequence workspace. When you release the mouse button, the additional strips will be selected.

**Moving and Modifying Strips** **G** Moves the selected strip(s) in time or in channels. Move your mouse horizontally (left/right) to change the strip's position in time. Move vertically (up/down) to change channels.

- To snap while dragging hold **Ctrl**
- To 'ripple edit' (Make room for strips you drag) hold **Alt** when placing a strip.

If you have added a strip by mistake or no longer want it, delete it by pressing **X** or using this menu option.

*Duplicate* a strip to make an unlinked copy; drag it to a time and channel, and drop it by **LMB** click.

The Strip Menu contains additional tools for working with strips: *Grab/Move*

*Grab/Extend from Frame*

*Cut (hard) at frame*

*Cut (soft) at frame*

*Separate Images Deinterlace Movies*

*Duplicate Strips*

*Erase Strips*

*Set Render Size*

*Make Meta Strip*

*UnMeta Strip*

*Reload Strips*

*Reassign Inputs*

*Swap Inputs*

*Lock Strips*

*UnLock Strips*

*Mute Strips*

*Un-Mute Strips*

*Mute Deselected Strips*

*Snap Strips*

*Swap Strips*

**Snap to Frame** **Shift-S** Position your cursor (vertical green line) to the time you want. Snap to current frame to start a strip exactly at the beginning of the frame. If your Time display is in seconds, you can get to fractional parts of a second by zooming the display; you can get all the way down to an individual frame.

**Separate Images to Strips** **Y** Converts the strip into multiple strips, one strip for each frame. Very useful for slide shows and other cases where you want to bring in a set on non-continuous images.

## Editing Strips

- RMB in the middle of the strip selects the **entire** strip; holding it down (or pressing `G` grab) and then moving the mouse drags a strip around.
- RMB on the left arrow of the strip selects the **start** frame offset for that strip; holding it down (or pressing `G` grab and then moving the mouse left/right changes the start frame within the strip by the number of frames you move it:
  - If you have a 20-image sequence strip, and drag the left arrow to the right by 10 frames, the strip will start at image 11 (images 1 to 10 will be skipped). Use this to clip off a rollup or useless lead-in.
  - Dragging the left arrow left will create a lead-in (copies) of the first frame for as many frames as you drag it. Use this when you want some frames for transitions to the this clip.
- RMB on the right arrow of the strip selects the **end** frame of the strip; holding it down (or pressing `G` grab) and then moving the mouse changes the ending frame within the strip:
  - Dragging the right arrow to the left shortens the clip; any original images at the tail are ignored. Use this to quickly clip off a rolldown.
  - Dragging the right arrow right extends the clip. For movies and images sequences, more of the animation is used until exhausted. Extending a clip beyond its end results in Blender making a copy of the last image. Use this for transitions out of this clip.

---

### Note: Multiple selection

You can select several (handles of) strips by `Shift`-RMB clicking: when you press `G`, everything that's selected will move with your mouse - this means that, for example, you can at the same time move a strip, shorten two others, and extend a fourth one.

- **STRIP EXTEND.** With a number of Image strips selected, pressing `E` enters EXTEND mode. All selected strip handles to the “mouse side” of the current frame indicator will transform together, allowing you to essentially extend the strips that fall exactly on the current frame marker and having all others adjust to compensate.

While splicing two strips happens just by placing them finish-to-start, cut a strip by pressing `K` to cut. At the selected frame for the selected strips, `K` cuts them in two. Use Cut to trim off roll-ups or lead-ins, or roll-downs or extra film shot (“C” was already taken for Change).

---

### Note: Note on the ‘cut’

When you ‘cut’ a strip, you don’t really make a cut like it was with the ‘old editing’ on real film. In fact, you make a copy of the strip: the end of the original one is ‘winded’ to the cut point, as with the beginning of the new copy.

For example, imagine that you have a strip of **50** frames, and that you want to delete the first ten ones. You have to go to the 11<sup>th</sup> frame, and press `K`; the cut ‘divides’ your strip in two parts. You now can select the first small part (frames 1 to 10), and delete it press `X`.

You might think that you have really erased the frames **1** to **10**, but there are still there, ‘winded’, as in a film reel, under your frame **11**: you just have deleted one of the two copies of your strip created by the ‘cut’. And you can at any time get your ‘lost’ frames back (just RMB -click on the left arrow of the strip, then `G` grab it to the left to display the desired number of frames again (or to the right to ‘hide’ more frames - this is another way to remove frames at the beginning/end of a strip!).

This is at the heart of nearly every editor solution, and that’s quite handy!

---

### Note: Action Stops

When extending the start beyond the beginning or end after the ending, keep in mind that only the last image copies, so when viewed, action will stop on that frame. Start your transition (fade, cross) a little early while action is still happening so that the stop action is not that noticeable (unless, of course, you want it to be, like the 80’s drama sitcoms).

Change the length of an effect strip by changing the start/end frame of the origin strips.

**Copy and Paste** You can copy a clip and paste it using the two header buttons.

**Meta Strips** A Meta-Strip is a group of strips. Select all the strips you want to group, and Ctrl-g to group them into one meta. The meta spans from the beginning of the first strip to the end of the last one, and condenses all channels into a single strip, just like doing a mixdown in audio software. Separating (ungrouping) them restores them to their relative positions and channels.

The default blend mode for a meta strip is Replace. There are many cases where this alters the results of the animation so be sure to check the results and adjust the blend mode if necessary.

One convenient use for meta strips is when you want to apply the same effect to multiple strips. For example: scaling a loop. Until blender gets a Loop effect, the only way to loop a clip is to duplicate it several times. If the clip needs any transforms (like scaling or translating an animated watermark or source material in a different aspect ratio) it is much more convenient to apply a single set of transforms to a meta strip built from the repeated duplicates than apply copies of those transforms to each instance in the loop.

It is possible to edit the contents of a meta strip by selecting it and pressing Tab. You can press Tab again to finish editing that strip. Since meta strips can be nested, to pop out one level of meta strip make sure you do not have a meta strip as the active strip when you press Tab.

## Sequence Display Modes

By default, the VSE only displays the sequencer. Several options in the header bar allow you change the editor to display the sequence in real time, and in various ways.

The second button will change the editor to display only the preview, and the third button displays both the sequencer and the preview

the VSE workspace can show you different aspects of the composite result, for the current frame:

- Image/Sequence: Colors (what you see)
- Chroma: Color hue and saturation
- Luma: Brightness/contrast
- Histogram: Levels of red, green, and blue

In the Chroma, Luma, and Image modes, a channel selector appears; channel 0 is the result of compositing the strips with their special effects strips. Channel 1 is what the current frame's image from the strip in channel 1 looks like (channel 1 is at the bottom of the heap). The display of these modes is either the composite (channel 0) or the frame from the strip (channels 1 through n).

Zoom the view of any of these workspaces by scrolling your middle mouse wheel.

## Image Preview

In the upper window pane of the Sequence screen layout is another VSE window, this one set to Image Preview mode. It shows you what the resulting video will look like when saved. This is the main working mode for adding strips and moving them around, cutting, grouping (making meta) and splicing them through special effects.

## Luma Waveform

For the selected channel, brightness, or luminosity, is mapped with this display.

A luma waveform allows you to judge the quality of the luminance distribution across your video signal, you can view a luma-waveform instead of the usual output display on every control monitor.

The display plots for every scanline the luminance value. The lines are all drawn on top of each other. The points get brighter if the lines cross (which is very likely with several hundred scanlines). You will understand the picture most easily if you plug an oscilloscope to the Luma-video-output of your television set. It will basically look the same.

In this mode, the vertical axis represents the luminosity: 0 at the bottom, 1 at the top; the horizontal axis is a mapping from the horizontal axis of the frame. There are as many curves as scanlines in the frame: each one of these curves represents the luminosity of the pixels of one line. Moreover, the color of a pixel in this mode represents the number of pixels from the matching column of the frame sharing the same luminosity - i.e. the number of curves that cross at this point (black/transparent, for no pixel, white/opaque for at least 3 pixels).

This mode is good for:

- If the waveform does not fill the whole picture you might want to play with the “setup” and “gain” master-sliders in the “gamma”-plugin until it fills the whole picture (contrast autostretch).
- With the more advanced gamma-plugin you can decide where you have to desaturate (especially in dark regions).
- You can judge if you want to dump the whole thing since it is completely distorted and clips at the top or the bottom.

	Examples of <i>VSE Luma</i> Previews. Note that the pictures (first green frame, at the top) are only 50px high, to limit the number of curves displayed in the <i>Luma waveform</i> !
--	--

Use this display to check for appropriate contrast and luminosity across all frames in the channel. When spots in the film that should have even illumination don't, it looks like a flashbulb went off or an extra light was suddenly turned on. This can happen if two strips were rendered or shot under different lighting conditions but are supposed to be contiguous.

## Chroma Vectorscope

Use this mode judge the quality of the color-distribution and saturation, you can also view a U/V scatter-plot.

The picture is converted to YUV-format. The U- and V-values represent the angle of the color. For pixel of the picture, one point is plotted in the display at the U and V-value-position. If several pixels happen to have the same U/V-value the pixel in the plot gets brighter.

To help you understand what color is meant, a hexagram marking the extreme positions (red, magenta, blue, cyan, green, yellow) is drawn and a red cross to mark the origin.

In other words, for the selected channel, this display shows the color space of the image inside a hexagon. Each point of the hexagon is a primary color: red, magenta, blue, cyan, green, and yellow. Black is at the center, and overall saturation is scaled as dots closer to the outside. The example to the right shows that the image has a lot of red (50% saturation) and small amount of blue, with no green.

Always: remember to activate an additional control monitor of the end result. Color calibration is a matter of taste and depends on what you want.

Use this display to check for too much color saturation. While over-saturated images look great for op-art and computer displays, they stink when shown on the big screen TV. Use the *Alt-A* animation key to scrub the video; this display will update with a new/revised map for each frame. Just like watching the Image preview to see what it looks like, watch the Chroma Vectorscope to watch for color use.

This mode is good for:

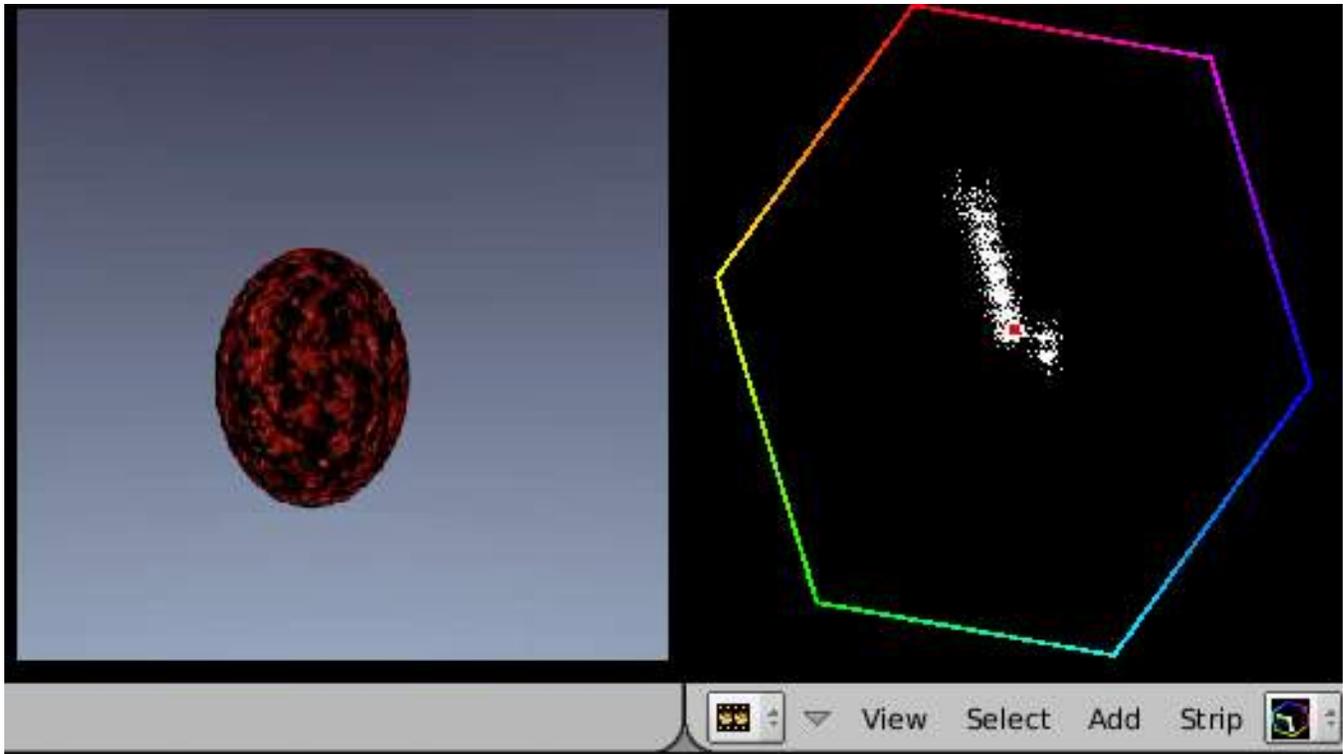


Fig. 2.17: Example VSE Chroma Preview

- If your picture looks very moody or desaturated you might want to take a look at the U/V-plot. You will most likely see all pixels building a crowd at the origin. If you add saturation using the “gamma”-plugin you can see in the U/V-plot if you distort the color.
- If you do color-matching on a by hand basis you can match the angle you see of different channels monitors.

### Histogram

This mode displays a graph showing the distribution of color information in the pixels of the currently displayed image. The X-axis represents values of pixel, from 0 to 1 (or 0 to 255), while the Y-axis represents the number of pixels in that tonal range. A predominantly dark image would have most of its information toward the left side of the graph.

Use this mode to balance out the tonal range in an image. A well balanced image should have a nice smooth distribution of color values.

### Sequencer Screen Layout

#### Sequencer Effects

Blender offers 16 built effects that are built into Blender, and are therefore universal. Some operate on two strips; some on one, and some create a new strip. Each effect enhances your content in some way or allows professional-quality transitions.



Fig. 2.18: Default Video Editing screen lay out

Fig. 2.19: Can you hear the thunder?

## Add

The Add effect adds two colors together. Red and Cyan (Green and Blue) make White. Red and Blue make “Magenta” (i.e. Purple!). Red and Green make Yellow.

The Add Effect adds the colors of two strips together, Use this effect with a base image strip, and a modifier strip. The modifier strip is either a solid color or a black-and-white mask, or another image entirely. The example to the right shows what happens when you add gray to an image, and animate the effect over time. The image gets bright because we are adding gray (R:.5, G:.5, B:.5) to say, a blue color (R:1, G:.1, B:.5) resulting in (R:.6, G:.6, B:1.0) which retains the original hue (relationship between the colors) but is much brighter (has a higher value). When applied to the whole image like this, the whole image seems to flash.

You can use this effect to increase the brightness of an image, or if you use a BW mask, selectively increase the brightness of certain areas of the image. The Mix node, in Add mode, does exactly the same thing as the Add sfx strip here, and is controlled the same way by feeding the Factor input.

## Subtract Effect

This effect takes away one strip’s color from the second. Make a negative of an image using this effect, or switch the order of the strips and just darken the strip. Subtracting a hue of blue from a white image will make it yellow, since red and green make yellow.

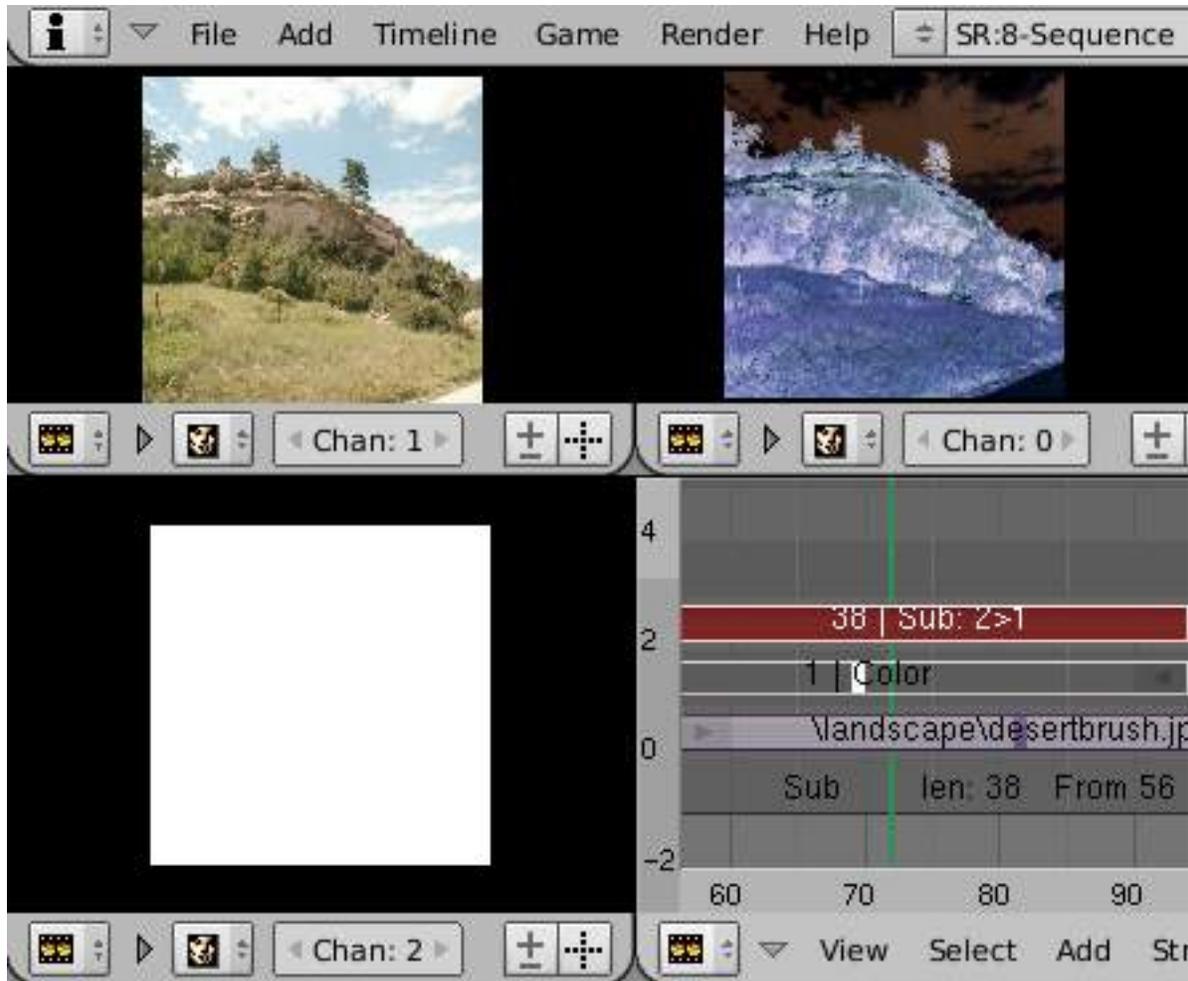
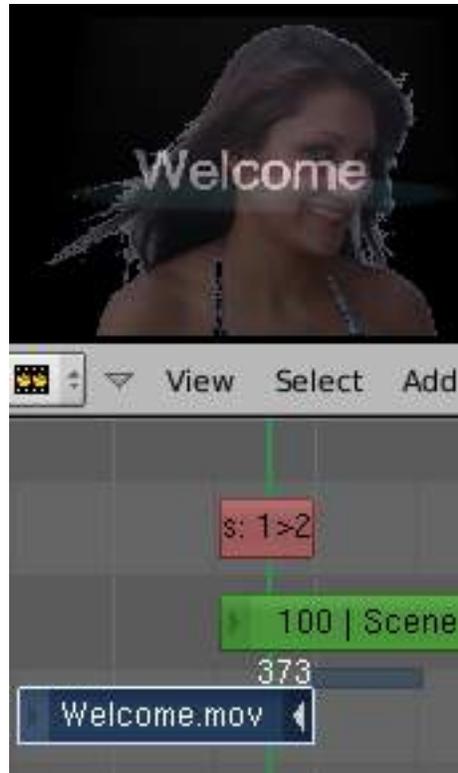


Fig. 2.20: Subtract Effect



### Cross and Gamma Cross

This effect fades from one strip to another, based on how many frames the two strips overlap. This is a very useful strip that blends the whole image from one to the other.

Gamma Cross uses color correction in doing the fade, resulting in a smooth transition that is easier on the eye.

**Fade to Black** Many scenes fade to black, and then fade in from black, rather than directly from one to the other.

The strip setup to do this is shown to the right. The two strips are on Channel 1, and you Add→Color Generator strip to Channel 2, straddling the two main strips. Change the color to black, and add two Cross Effects; the first from Channel 1 to Channel 2 (black), and the second from Channel 2 to Channel 1. The first strip will fade to black, and then the second will fade in from black. Of course, you can use any transition color you want. Black is a relaxing intermediary; red is alarming. Use the dominant color in the second strip to introduce the second strip.

### Multiply

The *Multiply* effect multiplies two colors. Blender uses values between **0.0** and **1.0** for the colors, he doesn't have to normalise this operation, the multiplication of two terms between **0.0** and **1.0** always gives a result between **0.0** and **1.0** (with the 'traditional' representation with three bytes - like RGB(**124**, **255**, **56**) -, the multiplications give far too high results - like RGB(**7316**, **46410**, **1848**) -, that have to be 'brought back', normalised - just by dividing them by **256** ! - to 'go back' to range of **0** to **255** ...). **256** ! - to 'go back' to range of **0** to **255** ...).

This effect has two main usages:

**With a mask** A mask is a B&W picture witch, after multiplication with a 'normal' image, only show this one in the white areas of the mask (everything else is black). The opening title sequence to James Bond movies, where the camera is looking down the barrel of a gun at James, is a good example of this effect.

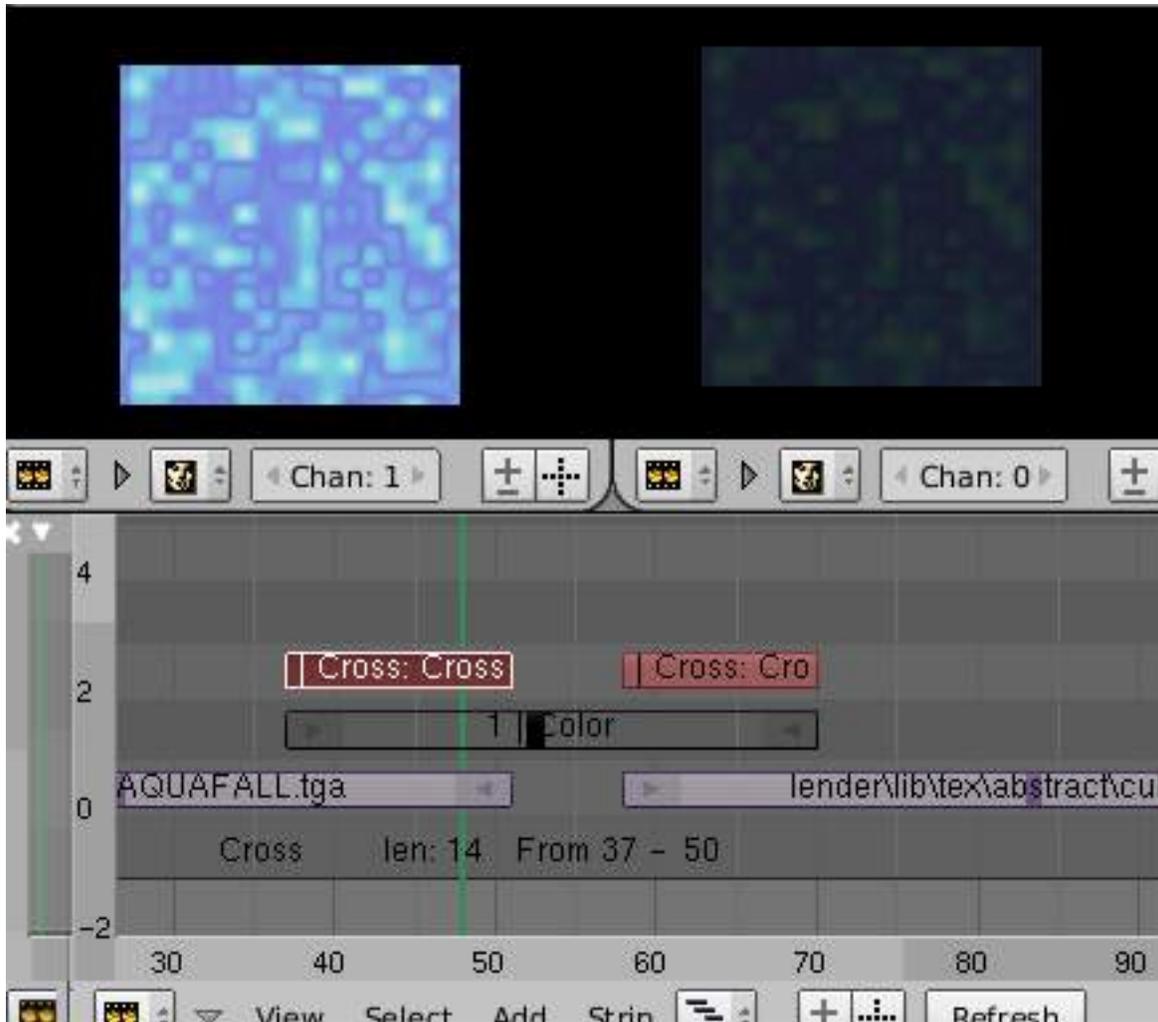


Fig. 2.21: Cross-Fade between Black

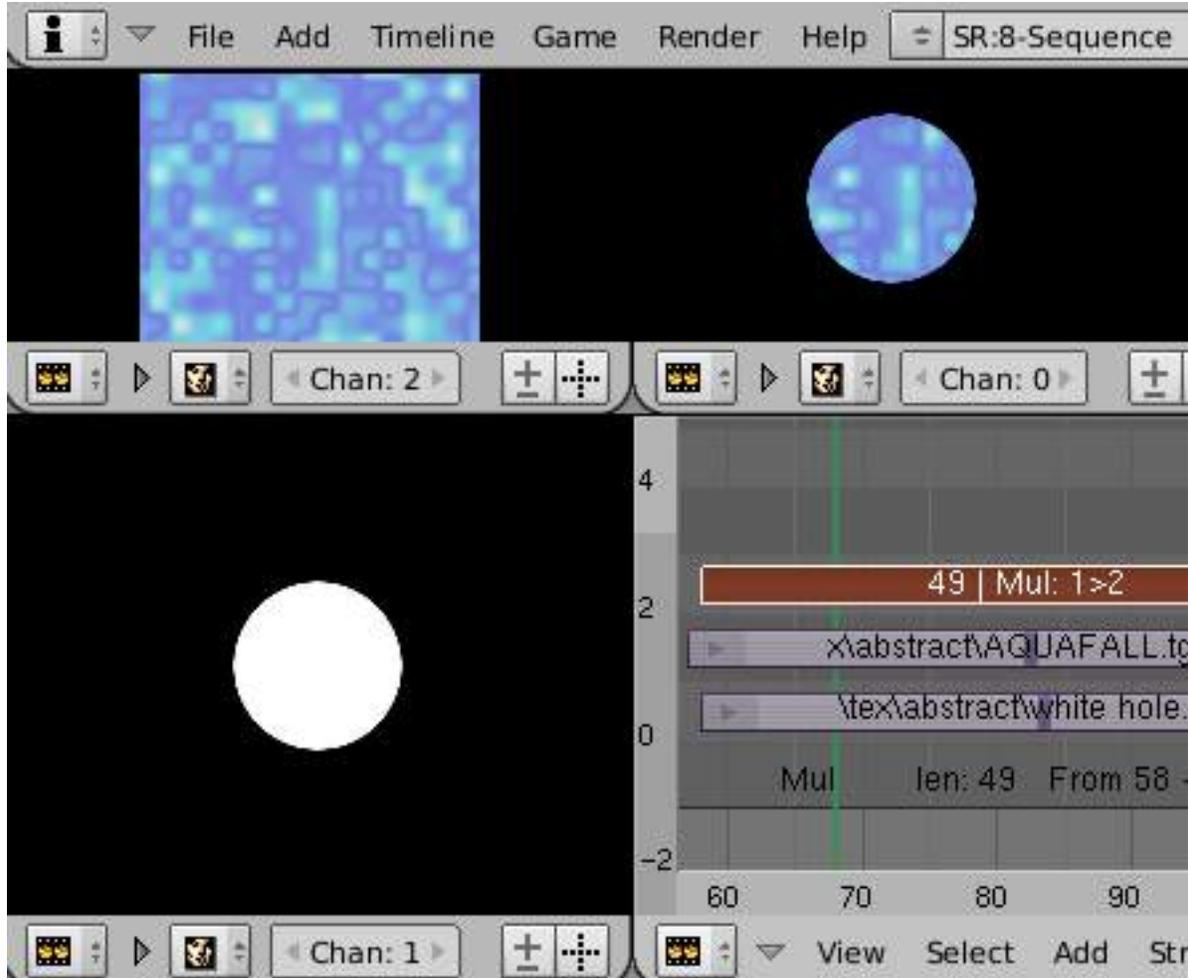


Fig. 2.22: Multiply Effect.

**With uniform colors** Multiplying a color with a ‘normal’ image allows you to soften some hues of this one (and so - symmetrically - to enhance the others). For example, if you have a brown pixel RGB(0.50, 0.29, 0.05), and you multiply it with a cyan filter (uniform color RGB(0.0, 1.0, 1.0), you’ll get a color RGB(0.0, 0.29, 0.5). Visually, the result is to kill the reds and bring up (by ‘symmetry’ - the real values remain unchanged!) the blues and greens. Physically, it is the same effect as shining a cyan light onto a chocolate bar. Emotionally, vegetation becomes more lush, water becomes more Caribbean and inviting, skies become friendlier.

**Note:** This effect reduces the global luminosity of the picture (the result will always be smaller than the smallest operand). If one of the image is all white, the result is the other picture; if one of the image is all black, the result is all black!

### Alpha Over, Under, and Over Drop

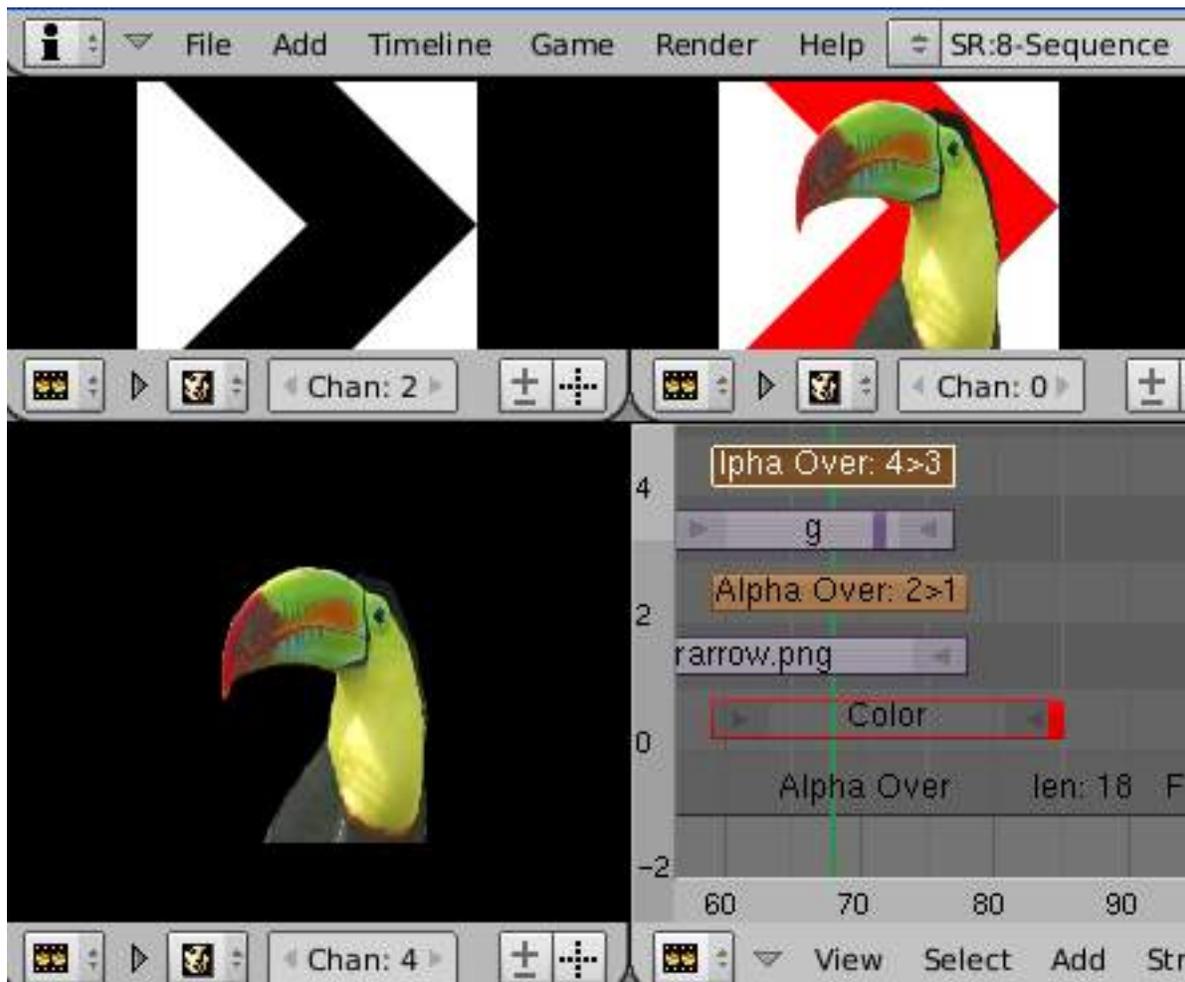


Fig. 2.23: AlphaOver Effect

Using the alpha (transparency channel), this effect composites a result based on transparent areas of the dominant image. If you use a Scene strip, the areas of the image where there isn’t anything solid are transparent; they have an alpha value of 0. If you use a movie strip, that movie has an alpha value of 1 (completely opaque).

So, you can use the *Alpha Over / Alpha Under* effect to composite the CGI Scene on top of your movie. The result is your model doing whatever as if it was part of the movie. The Factor curve controls how much the foreground is mixed over the

background, fading in the foreground on top of the background. The colors of transparent foreground image areas is ignored and does not change the color of the background.

Select two strips (`Shift-RMB`):

- With *Alpha Over*, the strips are layered up in the order selected; the first strip selected is the background, and the second one goes *over* the first one selected. The *Fac* tor controls *the transparency of the foreground*, i.e. a *Fac* of **0.0** will only show the background, and a *Fac* of **1.0** will completely override the background with the foreground (except in the transparent areas of this one, of course!)
- With *Alpha Under*, this is the contrary: the first strip selected is the foreground, and the second one, the background. Moreover, the *Fac* tor controls *the transparency of the background*, i.e. a *Fac* of **0.0** will only show the foreground (the background is completely transparent), and a *Fac* of **1.0** will give the same results as with *Alpha Over*.
- *Alpha Over Drop* is between the two others: as with *Alpha Under*, the first strip selected will be the foreground, but as with *Alpha Over*, the *Fac* tor controls the transparency of this foreground.

The example shows layering of AlphaOver effects. The very bottom channel is red, and an arrow is on top of that. Those two are AlphaOver to Channel 3. My favorite toucan is Channel 4, and Channel 5 alphaovers the toucan on top of the composited red arrow. The last effect added is tied to Channel 0 which will be rendered.

By clicking the PreMult Alpha button in the properties panel of the foreground strip, the Alpha values of the two strips are not multiplied or added together. Use this effect when adding a foreground strip that has a variable alpha channel (some opaque areas, some transparent, some in between) over a strip that has a flat opaque (Alpha=1.0 or greater) channel. If you notice a glow around your foreground objects, or strange transparent areas of your foreground object when using AlphaOver, enable PreMultiply. The AlphaOver Drop effect is much like the Cross, but puts preference to the top or second image, giving more of a gradual overlay effect than a blend like the Cross does. Of course, all of the Alpha effects respect the alpha (transparency) channel, whereas Cross does not.

The degree of Alpha applied, and thus color mixing, can be controlled by an F-curve. Creating a Sine wave could have the effect of the foreground fading in and out.

**Wipe** Wipe transitions from one strip to another. This very flexible effect has four transition types:

- Clock: like the hands of an analog clock, it sweeps clockwise or (if Wipe In is enabled) counterclockwise from the 9:00 position. As it sweeps, it reveals the next strip.
- Iris: like the iris of a camera or eye, it reveals the next strip through an expanding (or contracting) circle. You can blur the transition, so it looks like ink bleeding through a paper.
- Double Wipe: Starts in the middle and wipes outward, revealing the next strip. It can also Wipe In, which means it starts at the outside and works its way toward the middle. You can angle and blur the wipe direction as well.
- Single Wipe: Reveals the next strip by uncovering it. Controls include an angle control so you can start at a corner or side, and blur the transition.

The wipe will have no effect if created from a single strip instead of two strips. The duration of the wipe is the intersection of the two source strips and can not be adjusted. To adjust the start and end of the wipe you must adjust the temporal bounds of the source strips in a way that alters their intersection.

**Glow** This effect makes parts of an image glow brighter by working on the luminance channel of an image. The *Glow* is the superposition of the base image and a modified version, where some areas (brighter than the *Threshold*:) are blurred. With the *Glow* strip properties, you control this *Threshold*:, the maximum luminosity that can be added (*Clamp*:), a *Boost factor*: for it, the size of the blur (*Blur distance*:), and its *Quality*:. The *Only boost* button allows you to only show/use the ‘modified’ version of the image, without the base one. To “animate” the glow effect, mix it with the base image using the Gamma Cross effect, crossing from the base image to the glowing one.

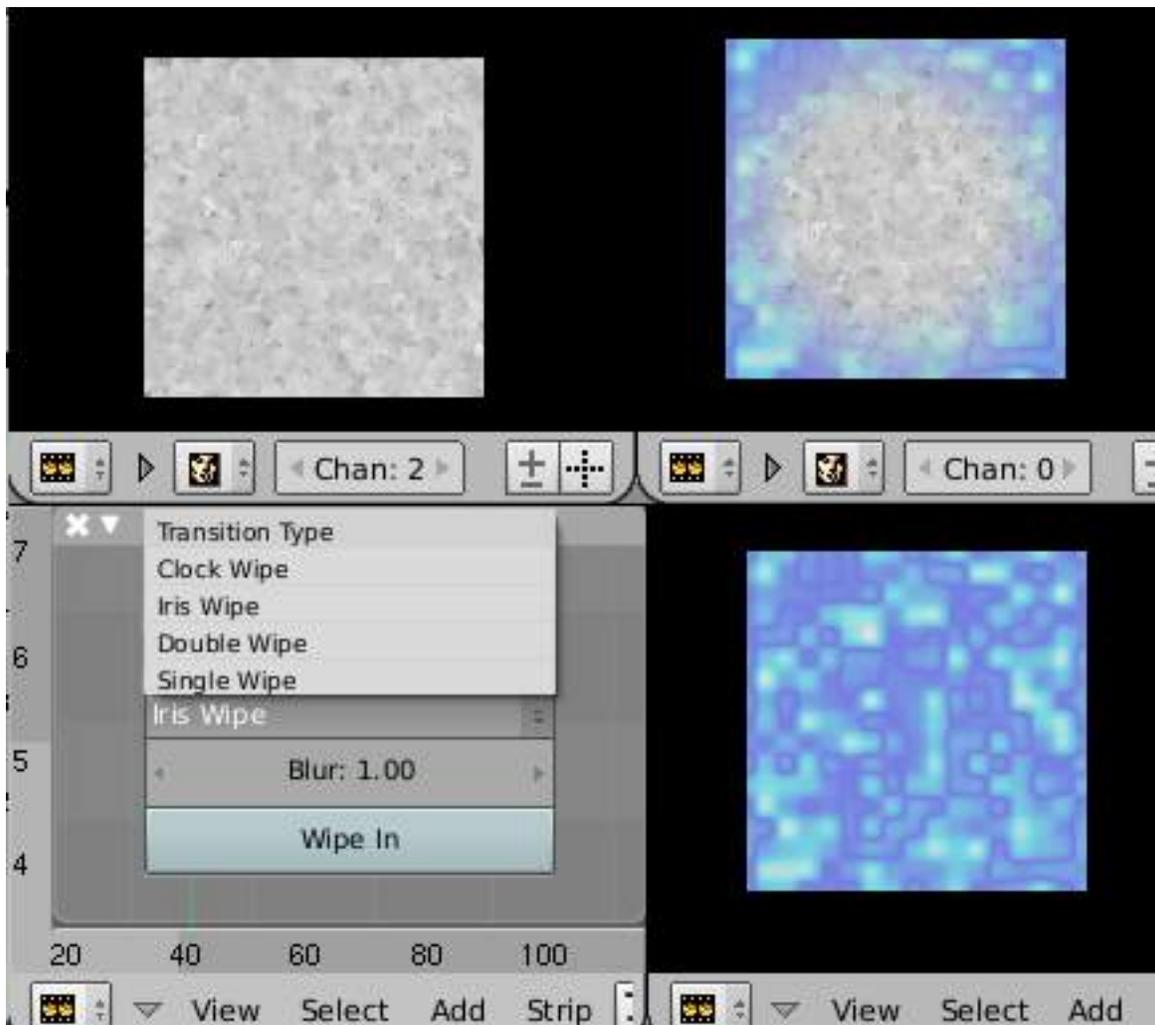


Fig. 2.24: VSE Wipe Built-in Effect

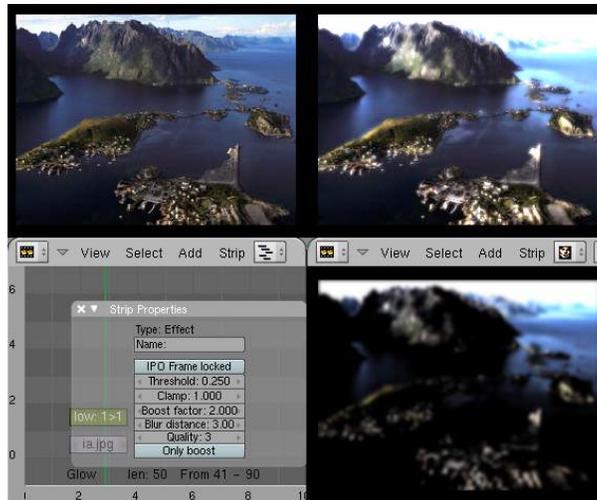
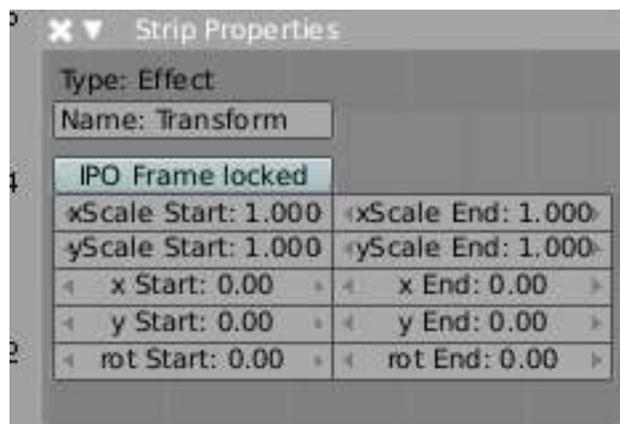


Fig. 2.25: Example of a Glow effect applied to a picture. Top left: base picture (Lofoten Islands, Norway - source: wikipedia.fr); Top right: result of the effect; Bottom left: effect settings; Bottom right: result with the Only boost button activated.

## Transform

Transform is a swiss-army knife of image manipulation. It scales, shifts, and rotates the images within a strip. The example to the right shows what can be done with a single image. To make a smooth transition to the final effect, enable the *Frame locked* button and define a curve in the Ipo Window (Sequence mode).



With the *Transform* strip selected, uses the properties panel to adjust the settings of this effect:

**(x,y)Scale (Start,End):** To adjust the scale (size). *xScale Start* defines the start width, *xScale End* the end width, *yScale Start* the start height, and *yScale End* the end height. The values higher than **1.0** will scale up the picture, while values lower than **1.0** will scale it down.

**(x,y) (Start,End):** To adjust the position (shifting). *x Start* defines the horizontal start position, *x End*, the end one; positive values shift the image to the right, negative values, to the left. *y Start* defines the vertical start position, *y End*, the end one; positive values shift the picture to the top, negative values, to the bottom.

**rot (Start,End):** The rotation is in degrees (**360** for a full turn) and is counter-clockwise. To make an image spin clockwise, make the end value lower than the start one (e.g. start it at 360 and go down from there).

## Color

This effect works by itself to create a color strip. By default, when it is created, it is 50 frames long, but you can extend it by grabbing and moving one of the ends. Click on the color swatch in the Effect panel under Sequencer buttons, which is under the Scene tab, to pick a different color (by default, it is gray). Use this strip crossed with your main movie to provide a fade-in or fade-out.

## Speed Control

Speed Control time-warps the strip, making it play faster or slower than it normally would. A Global Speed less than 1.0 makes the strip play slower; greater than 1.0 makes it play faster. Playing faster means that some frames are skipped, and the strip will run out of frames before the end frame. When the strip runs out of frames to display, it will just keep repeating the last one; action will appear to freeze. To avoid this, position the next strip under the original at a point where you want motion to continue.

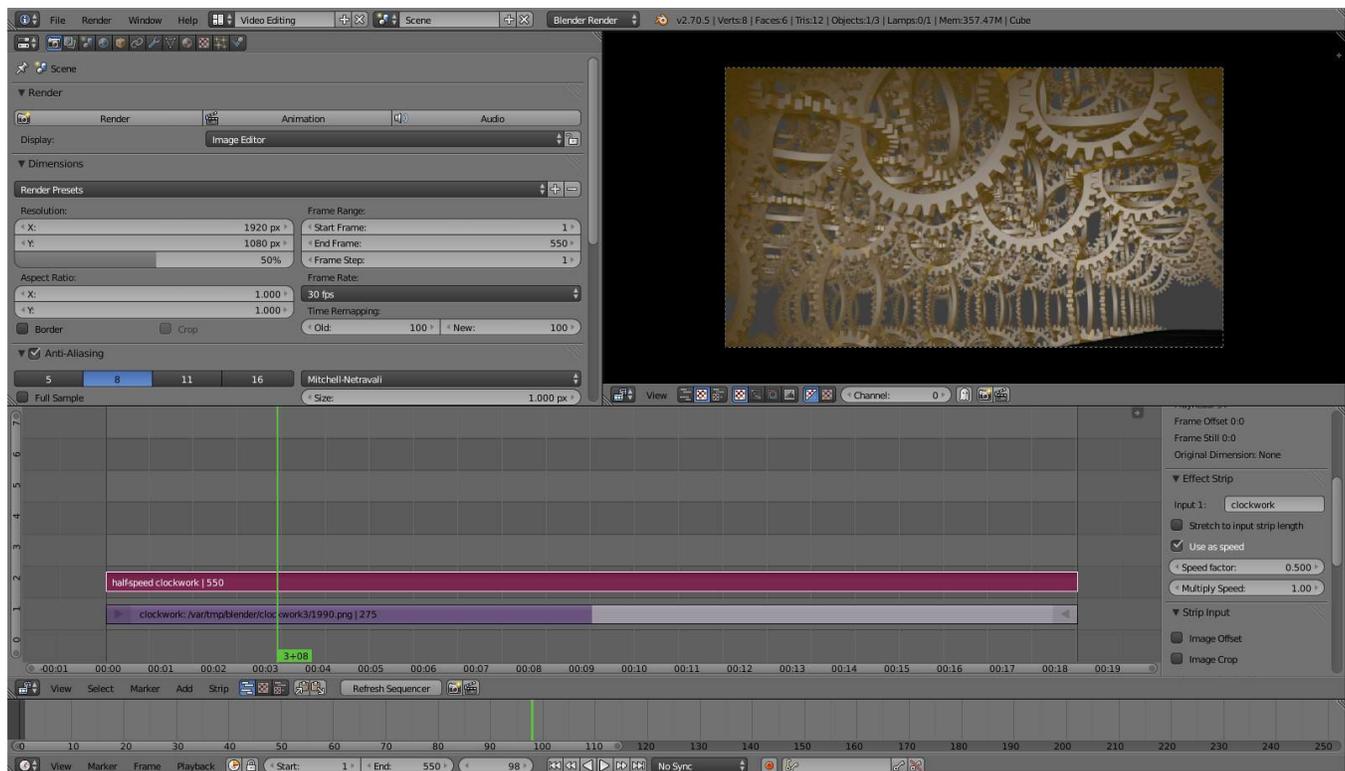


Fig. 2.26: 50% Slow motion using Speed Control

**Creating a Slow-Motion Effect** Suppose you want to slow your strip down. You need to affect the speed of the video clip without affecting the overall frame rate. Select the clip and Add→Effect→Speed Control effect strip. Click to drop it and press N to get the Properties. Uncheck the *Stretch to input strip length* option in the Effect Strip section. Set the Speed factor to be the factor by which you want to adjust the speed. To cut the displayed speed by 50%, enter 0.5. Now, a 275-frame clip will play at half speed, and thus display only the first 137 frames.

If you want the remaining frames to show in slo-mo after the first set is displayed, double the Length of the source strip (since effects strip bounds are controlled by their source strips). If you're using a speed factor other than 0.5 then use the formula

$$\text{new\_length} = \text{true\_length} / \text{Speed\_factor}$$

That's it! Set your render to animate (in this example) all 550 frames.

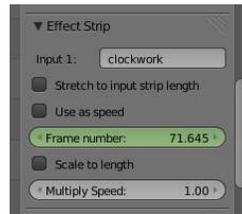


Fig. 2.27: keyframing the Frame number

**Keyframing the Speed Control** To get even finer control over your clip timing, you can use curves! While it is possible to keyframe the Speed factor, usually you want to keyframe the Frame number directly.

Uncheck *Stretch to input strip length* and uncheck *Use as speed*. You now have a Frame number field which you can keyframe. If you want the strip to animate **at all** you will have to insert some keyframes, otherwise it will look like a still. In most cases you will want to use the Graph editor view to set the curve interpolation to Linear since the default Bezier will rarely be what you want.

If you do choose to keyframe the Speed factor instead, remember to click the Refresh Sequencer button in the header of the Video Sequence Editor's strip view or your changes will not take effect.

**Changing Video Frame Rates** You can use the speed control to change the frames per second (fps), or framerate, of a video. If you are rendering your video to a sequence set, you can effectively increase or decrease the number of individual image files created, by using a Global Speed value less than or greater than one, respectively. For example, if you captured a five-minute video at 30 fps and wanted to transfer that to film, which runs at 24 fps, you would enter a Global Speed of 30/24, or 1.25 (and Enable Frame Blending to give that film blur feel). Instead of producing  $5 \times 60 \times 30 = 9000$  frames, Blender would produce  $9000 / 1.25 = 7200 = 5 \times 60 \times 24$  frames. In this case, you set a Start:1 and End:7200, set your Format output to Jpeg, 30fps, and image files 0001.jpg through 7200.jpg would be rendered out, but those images 'cover' the entire 9000 frames. The image file 7200.jpg is the same as a frame 9000. When you read those images back into your film .blend at 24 fps, the strip will last exactly 5 minutes.

## Multicam Selector

Ever wanted to do multicam editing with Blender? Now you can and it is mindbogglingly easy:

- Add your input strips on channels say 1 to 4 (you can use as many you like, interface gets a little bit clunky if you have more than 10, see below).
- Sync the strips up. There is no automatic sync feature in Blender, but you can open two viewer windows, choose one camera as the master channel and sync the other against them just by looking at the movement of legs or light flashes (depending on the show, you want to edit). We might add automatic sync feature based on global brightness of the video frames in the future. (Syncing based on the audio tracks, like most commercial applications do, isn't very clever, since the speed of sound is only around 340 metres per second and if you have one of your camera 30 meters away, which isn't uncommon, you are already 2-3 frames off. Which *is* noticeable...)
- Build small resolution proxies (25%) on all your input video strips.
- Use meta strips, so that every input camera fits in exactly one channel.
- Add a viewer window for every input channel and put it into 25% proxy display mode (I suggest to line them up on the left side on top of each other, but just do, whatever pleases your personal habits)
- Add a large viewer window for the final output and let it run on full resolution.

- Add a multicam selector effect strip *above* all the channel tracks
- Enlarge it, so that it covers the whole running time of your show (just change it's length or drag the right handle, the former is probably easier, since you can just type in a very large number and you are done)
- Cross you fingers :) (that's important :) )
- Select the multicam strip, if you take a look at the strip options (N-key), you will notice, that multicam is a rather simple effect strip: it just takes a selected channel as it's input. That's all. The magic comes with the convenient keyboard layout: when you select multicam, the keys 1-0 are mapped to a python handler, that does a cut on the multicam and changes it's input.
- So: you select the multicam strip, you start playback and press the keys 1-4 while watching your show.
- You'll end up with a small multicam selector strip for every cut.

In reality, it boils down to: watch a few seconds to see, what's coming, watch it again and do a rough cut using the number keys, do some fine tuning by selecting the outer handles of two neighboring multicam for A/B rolling.

### Adjustment Layer

The adjustment layer strip works like a regular input file strip except for the fact, that it considers all strips below it as it's input.

Real world use cases: you want to add some last finishing color correction on top of parts of your final sequencer timeline without messing with metastrips around. Just add an adjustment layer on top and activate the color balance.

Or: you can stack a primary color correction and several secondary color correction on top of each other (probably using the new mask input for area selection).

### Sound Editing

Blender contains a multi-track Audio sequencing toolbox. You can add WAV, Mp3 files from your hard disk as a file, or as encoded within a movie, and mix them using an F-Curve as a volume control.

### Options

Audio-RAM loads a file into memory and plays it from there. You can only load stand-alone WAV files. Audio-HD plays the sound back from the hard disk and thus does not take up memory. With Audio HD, you can load stand-alone WAV files, but also audio tracks from movies.

For either, a green audio strip will be created. With Audio RAM, a waveform is created that shows you the waveform inside the green strip, scaled to the height of the green strip. Since Audio-RAM files are read into memory, changing the audio file will not affect playback, and you will have to re-open the file so that Blender re-reads the file.

---

**Note:** Hiss, Crackle and Pop

Some audiophile users report that Hiss is introduced sometimes if Audio RAM is used. There must be some decoding or sampling going on, that does not occur when Audio HD is used, that introduces some playback noise. If you hear pops and crackles, usually that is a sign that your hardware cannot keep up in real-time playback. They will not be present in your final rendered animation output (but they may show up in Game mode).

Also, static hiss seems to occur whenever two or more audio strips are overlapping in the timeline...

---

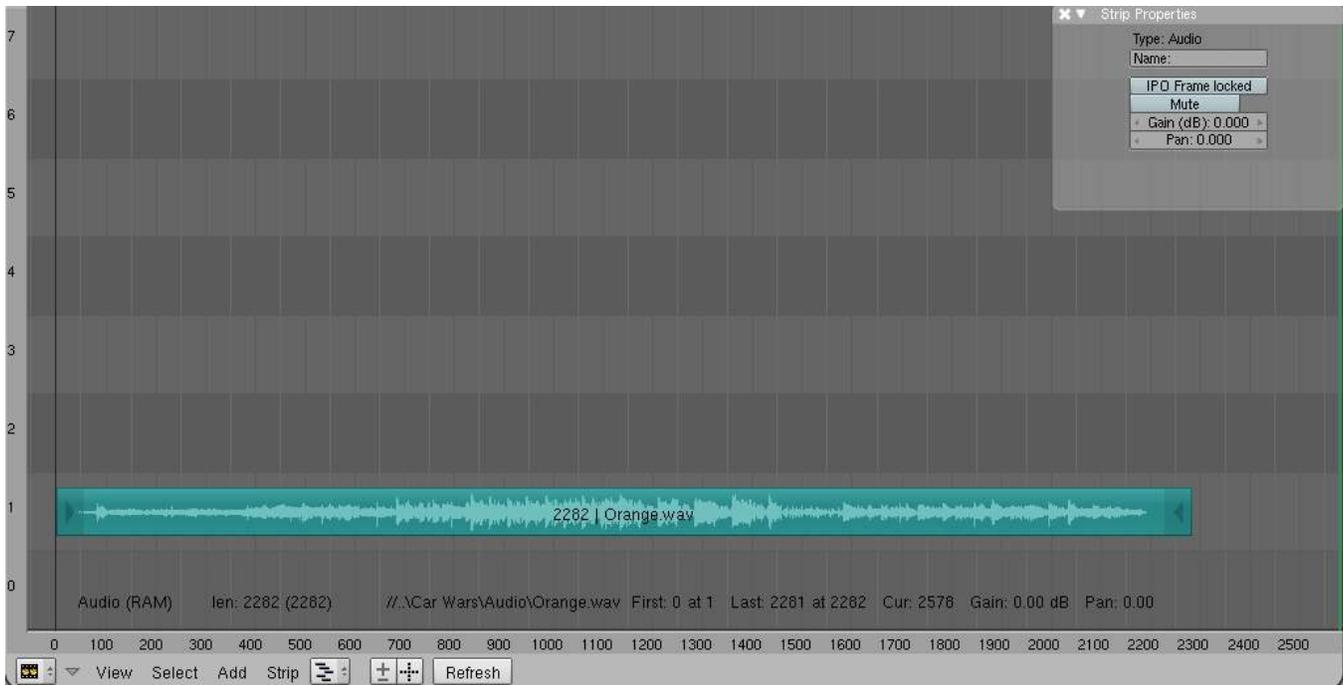


Fig. 2.28: A sound strip in the sequence editor.

### Audio Mixing in the VSE

You can have as many Audio strips as you wish and the result will be the mixing of all of them. You can give each strip its own name and Gain (in dB) via the N menu. This also let you set a strip to mute or ‘Pan’ it; -1 is hard left, +1 is hard right, with percentages in-between.

Overlapping strips are automatically mixed down during ANIM processing. For example, you can have the announcer on channel 5, background music on channel 6, and foley sound effects on channel 7.

### Working with Audio Tracks

An audio track (strip) is just like any other strip in the VSE. You can grab and move it, adjust its starting offset using RMB over the arrow end handles, and K cut it into pieces. A useful example is cutting out the “um’s” and dead voice time.

### Animating Audio Track Properties

You want to set a value somewhere between 0.0 and 1.0, and the volume becomes that percent; 0.6 is 60%. You can add a gain to the volume through the strip properties (N). You can make a curve by having multiple points, to vary the volume over its length. Press Tab to edit the curve, just like any old bezier F-curve.

In the Y direction, 1.0 is full volume, 0.0 is completely silent. Only the FFMPEG-output system is currently able to mix audio and video into one output stream. Use Ctrl-LMB to add control points, and Tab to edit a curve.

Animating an audio strip affects the volume of the strip in the resulting composite. Use animation on an audio strip to fade in/out background music or to adjust volume levels. Layered/crossed audio strips are added together; the lower channel does not override and cut out higher channels. This makes Blender an audio mixer. By adding audio tracks and using the curves to adjust each tracks’ sound level, you have an automated dynamic multi-track audio mixer!

**Output** The output is therefore a video file if the *ANIMATION* button in the *Render* Panel of the Scene Context/Render Sub-context is used as described before. An audio file may be created via the *MIXDOWN* button in the *Sequencer* button of the Scene Context, Sound Sub-context. This WAV file contains the full audio sequence and is created in the same directory of the video file and with the *same name* but with a *.wav* extension. You can mix Video and Audio later on with an external program or by adding it to, for example, an image sequence strip as described above.

The advantage of using Blender's sequence editor lies in the easier synchronization attainable by sequencing frames and sound in the same application.

To enable audio synchronisation after importing an audio track, select the *Scene* button in the buttons window then choose the *Sound Block* Button (small blue sine wave). In here you'll see the *Sync* and *Scrub* tools.

**Sync** lets Blender drop image frames to keep up with realtime audio when you play an animation in the 3D window. This gives you a rough overview of the timing of your animation.

**Scrub** allows you to drag your frame-marker or change frames in any window and it will play a clip of audio for that point in time.

Dragging the frame-marker over a range of frames in the Action Editor will allow you to hear roughly where specific sounds occur so that you can key poses or shapes on this frame.

## 2.1.6 Timeline



Fig. 2.29: Timeline Header

This Editor gives a timeline, through which you can scrub with the LMB.

## 2.1.7 3D View

### Introduction

The 3D View is used to interact with the 3D scene for a variety of purposes, such as modeling, animation, texture painting, etc. Navigating in the 3D space is done with the use of both mouse movement and keyboard shortcuts.

**Orbit (MMB)** Rotate the view around the point of interest.

**Pan (Shift+MMB)** Move the view up, down, left and right

**Zoom (Ctrl+MMB/Scroll)** Move the camera forwards and backwards

[Read more about navigation.](#)

### Modes

Blender has a number of *Modes* used for editing different kinds of data:

- Object Mode
- Edit Mode
- Pose Mode
- Sculpt Mode
- Vertex Paint

- Weight Paint
- Texture Paint
- Particle Edit

The mode can be changed using the menu in the 3D View header, or using the hotkey associated with that mode.

[Read more about modes.](#)

## Regions of the 3D View

**Toolshelf** The Toolshelf is a context-sensitive region containing tools depending on the current mode (for example, modeling tools in *Edit Mode*, brush tools in *Sculpt Mode*...).

For more information on specific tools available, see:

- Transformations
- History
- Creating Objects
- Groups and Parenting
- *Animation*
- *Rigid Body*
- *Grease Pencil*
- *Modeling*
- Sculpting
- Vertex Painting
- Weight Painting
- Texture Painting

**Properties Region** The Properties Region contains properties of the active object and selected objects (such as their locations), as well as properties of the editor itself (such as [Display](#) settings and [background images](#)).

**Header** Contains various menus, buttons and options based on the current *mode*, such as:

- Shading mode
- Pivot options
- Transform manipulator
- Proportional Edit
- Snapping
- *OpenGL render*

## Local View

### Reference

Editor: *3D View*

Menu: *View* → *View Global/Local*

Hotkey: NumpadSlash

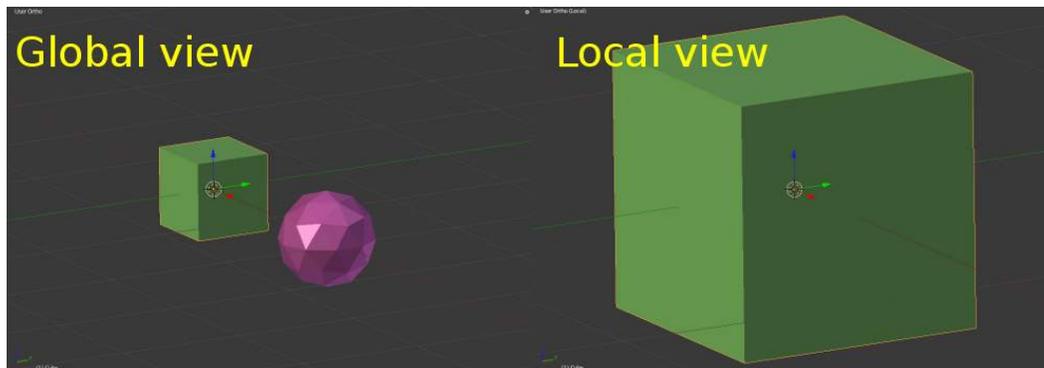


Fig. 2.30: Global and Local view

When entering *Local View*, the selected objects are isolated and all other objects are temporarily hidden from view. This only affects the current 3D View editor, and not the render.

This can be used to speed up viewport performance in heavy scenes, or allow you to focus on a specific object without others getting in your way.

## Modes

*Modes* are a Blender-level object-oriented feature, which means that *the whole Blender application* is always in *one and only one mode*, and that the available modes vary depending on the selected active object's type - most of them only enable the default *Object* mode (like cameras, lamps, etc.). Each mode is designed to edit an aspect of the selected object. See the *Blender's Modes* table below for details.

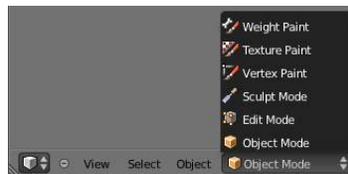


Fig. 2.31: Mode selection example (mesh object).

You set the current mode in the *Mode* drop-down list of *3D View* header (see *Mode selection example (mesh object)*).

**Warning:** You can only select objects in *Object* mode. In all others, the current object selection is “locked” (except, to some extent, with an armature's *Pose* mode).

Modes might affect many things in Blender:

- They can modify the panels and/or controls available in some *Buttons* windows' contexts.
- They can modify the behavior of whole windows, like e.g. the *UV/Image Editor* window (and obviously, *3D Views*!).
- They can modify the available header tools (menus and/or menu entries, as well as other controls...). For example, in the *3D View* window, the *Object* menu in *Object* mode changes to a *Mesh* menu in *Edit* mode (with an active mesh object!), and a *Paint* menu in *Vertex Paint* mode...

Table 2.1: Blender's Modes

Icon	Name	Short-cut	Details
	<i>Object</i> mode	<i>None</i> <sup>1</sup>	The default mode, available for all object types, as it is dedicated to <i>Object</i> datablock editing (i.e. position/rotation/size).
	<i>Edit</i> mode	Tab <sup>1</sup>	A mode available for all renderable object types, as it is dedicated to their "shape" <i>ObData</i> datablock editing (i.e. vertices/edges/faces for meshes, control points for curves/surfaces, etc.)
	<i>Sculpt</i> mode	<i>None</i> <sup>1</sup>	A mesh-only mode, that enables Blender's mesh 3D-sculpting tool.
	<i>Vertex Paint</i> mode	<i>None</i> <sup>1</sup>	A mesh-only mode, that allows you to set your mesh's vertices colors (i.e. to "paint" them).
	<i>Texture Paint</i> mode	<i>None</i> <sup>1</sup>	A mesh-only mode, that allows you to paint your mesh's texture directly on the model, in the 3D views.
	<i>Weight Paint</i> mode	Ctrl-Tab <sup>2</sup>	A mesh-only mode, dedicated to vertex group weighting.
	<i>Particle</i> mode	<i>None</i> <sup>1</sup>	A mesh-only mode, dedicated to particle systems, useful with editable systems (hair).
	<i>Pose</i> mode	Ctrl-Tab <sup>2</sup>	An armature-only mode, dedicated to armature posing.

Notes about modes shortcuts:

- Tab toggles *Edit* mode.
- Ctrl-Tab switches between the *Weight Paint* (meshes) / *Pose* (armatures) modes, and the other current one (by default, the *Object* mode). However, the same shortcut has other, internal meanings in some modes (e.g. in *Sculpt* mode, it is used to select the current brush)...

As you can see, using shortcuts to switch between modes can become quite tricky, especially with meshes...

We won't detail further more modes' usages here. Most of them are tackled in the [modeling chapter](#), as they are mainly related to this topic. The *Particle* mode is discussed in the [particle section](#), and the *Pose* and *Edit* modes for armatures, in the [rigging one](#).

---

**Note:** If you are reading this manual and some button or menu option is referenced that does not appear on your screen, it may be that you are not in the proper mode for that option to be valid.

---

## Shading

### Shading Modes

Shading refers to the way objects are drawn and lit in the 3D View.

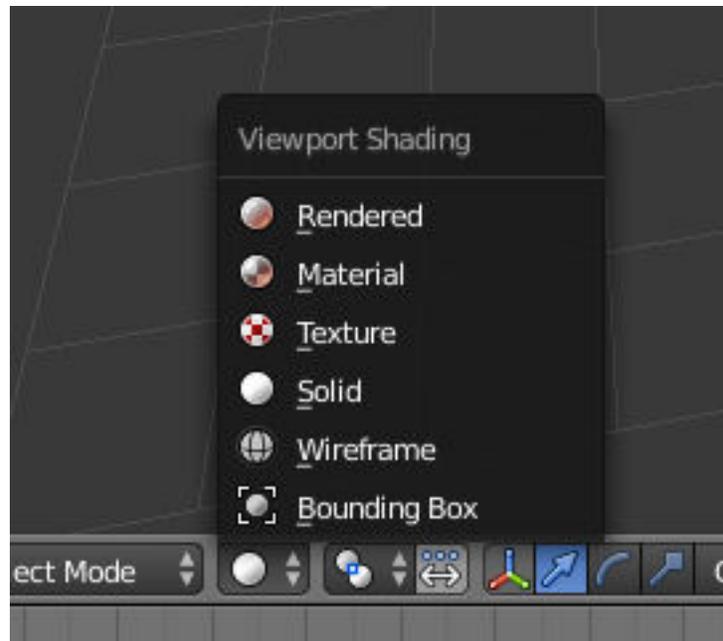


Fig. 2.32: 3D viewport shade button.

**Rendered** An accurate representation using the selected *Render Engine* and lit with the visible scene lights.

**Material** A fast approximation of the applied material. Some effects, such as procedural textures may not be shown.

**Textured** Shows meshes with an image applied using the mesh's active UV Map. For Cycles materials, the image is the last one selected in the Node Editor. For other render engine's, the UV Map's applied face texture will be shown.

**Solid** The default drawing mode using solid colored surfaces and simple lighting.

**Wireframe** Objects appear as a mesh of lines representing the edges of faces and surfaces.

**Bounding Box** Shows only the rectangular boxes that outlines an object's size and shape.

Except for *Rendered*, these shading modes are not dependent on light sources in the scene. Instead they use a simple default lighting adjusted by the *Solid OpenGL Lights* controls on the *System* tab of the [User Preferences](#) window.

The viewport shading controls the appearance of all objects in a scene, but this can be overridden for individual objects using the Display panel in their Object Properties.

Keyboard Shortcuts

Switches between <i>Wireframe</i> and <i>Solid</i> draw modes	Z
Switches between <i>Wireframe</i> and <i>Rendered</i> draw modes.	Shift-Z
Switches between <i>Solid</i> and <i>Textured</i> draw modes.	Alt-Z

## Shading Panel

The shading panel in the Properties Region provides additional control over the way objects in the 3D view appear.

**Textured Solid** Display assigned *face textures* in the *Solid* shading mode. (*Not available in the Cycles Render Engine*)

**Matcap** A selection of preset shader effects, (overriding regular materials) which can help visualize your models while editing or sculpting, without having to set up complex materials first.

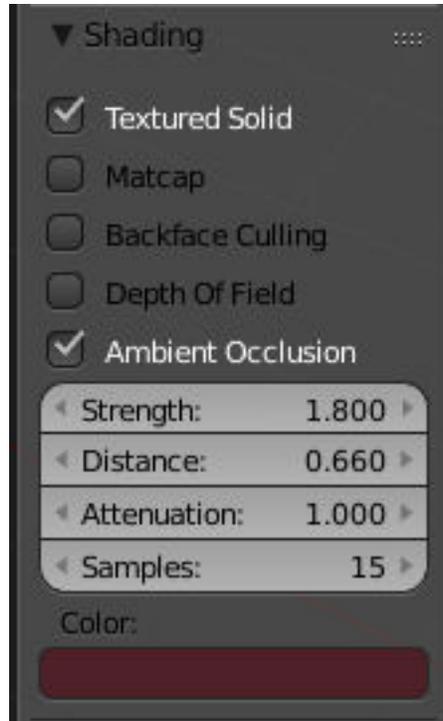


Fig. 2.33: 3D Viewport Shading Panel.

**Backface Culling** Only show the front side of faces. Use this to find faces flipped the wrong way, especially when exporting to programs that use single sided drawing.

**Depth of Field** Simulates a camera's focal blur effect in the 3D viewport. *Only visible in a camera view.* Control the effect using these options in the *Properties Tab* of the active camera: Focal Length, Sensor Size, Focus Object or Focus Distance, and Viewport F-stop.

**Ambient Occlusion** Improves the realism of the viewport image by simulating the darkening effect that occurs in crevices and corners. Typically such effects are rendered at higher quality, but this is a quick real-time preview which can help when modelling or sculpting.

These settings control the AO effect.

**Strength** A higher number makes the corners darker.

**Distance** How far out of the corners does the effect extend.

**Attenuation** How strongly the effect attenuates with distance. Increasing this makes far away surfaces contribute less to the effect. Use this to get rid of some banding artifacts.

**Samples** The number of samples used for the effect. Low numbers produce a grainy effect, but the actual number used is squared so use high numbers with caution.

**Color** Color of the effect, can be modified to give a different feel, from ambient lighting to dirt/rust.

## Display

### Display Panel

**Only Render** Displays only items that will be rendered.

This can be useful to preview how animations look without being distracted by rigs and empties.

**Outline Selected** If disabled, the pink outline around your selected objects in *Solid / Shaded / Textured* draw types will no longer be displayed.

**All Object Origins** If enabled, the center dot of objects will always be visible, even for non-selected ones (by default, unselected centers might be hidden by geometry in solid/shaded/textured shadings).

**Relationship Lines** Controls whether the dashed parenting, constraining, hooking, etc., lines are drawn.

**Grid Floor** If disabled, you have no grid in other views than the orthographic top/front/side ones.

**X Axis, Y Axis, Z Axis** Control which axes are shown in other views than the orthographic top/front/side ones.

**Lines** Controls the number of lines that make the grid in non-top/front/side orthographic views, in both directions.

**Scale** Control the scale of the grid floor

**Subdivisions** Controls the number of sub-lines that appear in each cell of the grid when you zoom in, so it is a setting specific to top/front/side orthographic views.

**Toggle Quad View** Toggles the four pane 3D view. [Read more about arranging frames](#)

## View Panel

The *View Properties* panel lets you set other settings regarding the 3D view. You can show it with the *View* → *View Properties...* menu entry.

**Lens** Control the focal length of the 3d view camera in millimeters, unlike a [rendering camera](#)

**Lock to Object** By entering the name of an object in the *Object* field, you lock your view to this object, i.e. it will always be at the center of the view (the only exception is the active camera view, NumPad0). If the locked object is an armature, you can further center the view on one of its bones by entering its name in the *Bone* field.

**Lock to Cursor** Lock the center of the view to the position of the 3D cursor

**Lock Camera to View** When in camera view, use this option to move the camera in 3D space, while continuing to remain in camera view.

**Clip Start and Clip End** Adjust the minimum and maximum distances to be visible for the view-port.

---

**Note:** A large clipping range will allow you to see both near and far objects, but reduces the depth precision.

To avoid this:

- increase the near clipping when working on large scenes.
- decrease the far clipping when objects are not viewed at a distance.

When perspective is disabled only the far Clip-End is used, very high values can still give artifacts.

*This is not specific to blender, all OpenGL/DirectX graphics applications have these same limitations.*

Examples:

---

**Local Camera** Active camera used in this view

**Render Border** Use a Render Border when not looking through a camera. Using `Ctrl-B` to draw a border region will automatically enable this option.

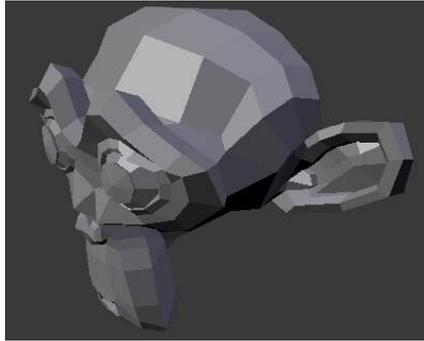


Fig. 2.34: Model with no clipping artifacts.

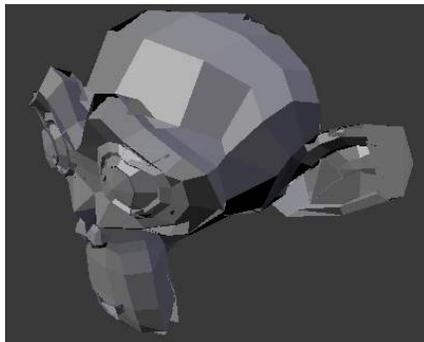


Fig. 2.35: Model with clipping artifacts.

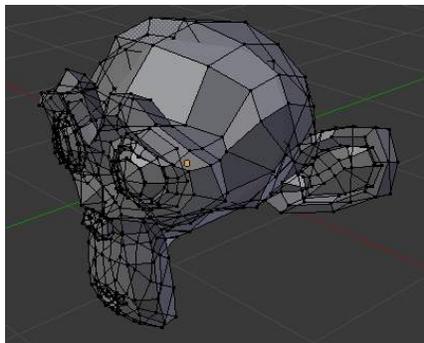


Fig. 2.36: Mesh with artifacts in edit-mode.

## Layers

### Reference

Mode: *Object* mode

Panel: *Relations* (*Object* context)

Menu: *Object* → *Move to Layer...*

Hotkey: M

3D scenes often become exponentially more confusing as they grow more complex. Sometimes the artist also needs precise control over how individual objects are lit, and does not want lights for one object to affect nearby objects. For this and other reasons below, objects can be placed into one or more “layers”. Using object layers, you can:

- Selectively display objects from certain layers in your 3D view, by selecting those layers in the *3D View* header bar. This allows you to speed up interface redrawing, reduce virtual-world clutter, and help improve your workflow.
- Control [which lights illuminate an object](#), by making a light illuminate only the objects on its own layer(s).
- Control which forces affect which [particle systems](#), since particles are only affected by forces and effects on the same layer.
- Control which layers are rendered (and hence, which objects), and which properties/channels are made available for compositing by using [render layers](#).

Armatures can also become very complex, with different types of bones, controllers, solvers, custom shapes, and so on. Since armatures are usually located close together, this can quickly become cluttered. Therefore, Blender also provides layers just for armatures. Armature layers are very similar to object layers, in that you can divide up an armature (rig) across layers and only display those layers you wish to work on.

Read more about [armature layers](#)

### Working with Layers

3D layers differ from the layers you may know from 2D graphics applications as they have no influence on the drawing order and are there (except for the special functions listed above) mainly to allow you to organize your scene.

When rendering, Blender only renders the selected layers. If all your lights are on a layer that is *not selected*, you won’t see anything in your render except for objects lit by ambient lighting.

[Groups and Parenting](#) are other ways to logically group related sets of objects. Please refer to the relevant sections for more information.

**Viewing layers** Blender provides twenty layers whose visibility can be toggled with the small unlabeled buttons in the header (see *3D Viewport layer buttons*). To select a single layer, click the appropriate button with LMB; to select more than one, use `Shift-LMB` - doing this on an already active layer will deselect it.



Fig. 2.37: 3D Viewport layer buttons.

To select layers via the keyboard, press 1 to 0 (on the main area of the keyboard) for layers 1 through 10 (the top row of buttons), and `Alt-1` to `Alt-0` for layers 11 through 20 (the bottom row). The `Shift` key for multiple (de)selection works for these shortcuts too.

You can select or deselect all Scene Layer buttons at once by pressing the ` key.

**Locking to the scene** By default, the lock button directly to the right of the layer buttons is enabled. This means that changes to the viewed layers affect all other 3D Views locked to the scene - see the [navigating the 3D view options page](#) for more information.

**Multiple Layers** An object can exist on multiple layers. For example, a lamp that only lights objects on a shared layer could “be” on layers 1, 2, and 3. An object on layers 3 and 4 would be lit, whereas an object on layers 4 and 5 would not. There are many places where layer-specific effects come into play, especially lights and particles.



Fig. 2.38: Layer selection.

**Moving objects between layers** To move selected objects to a different layer, press M and then select the layer you want from the pop-up dialog. Objects can also be on more than one layer at a time. To have an object on multiple layers, hold Shift while clicking.



Fig. 2.39: Object context selection.

Another way to view or change a selected object layer is via the *Relations* panel, in the *Object* context.

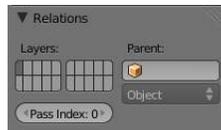


Fig. 2.40: Layers in Object context, Relations panel.

You will then see the layer buttons in the *Relations* panel - as before the object can be displayed on more than one layer by clicking Shift-LMB.

## 3D Cursor

The 3D Cursor is simply a point in 3D space which can be used for a number of purposes

### Placement

There are a few methods to position the 3D cursor.

**Direct Placement with the Mouse** Using LMB in the 3D area will place the 3D cursor directly under your mouse pointer.

For accuracy you should use two perpendicular orthogonal 3D views, i.e. any combination of top (Numpad7), front (Numpad1) and side (Numpad3). That way you can control the positioning along two axes in one view and determine depth in the second view.

To place the 3D Cursor on the surface of geometry, enable *Cursor Depth* in the [User Preferences](#)



Fig. 2.41: Positioning the 3D cursor with two orthogonal views.

**Using the Snap Menu** The *Snap* menu (`Shift-S` or *Object/Mesh* → *Snap*) will allow you to snap the cursor in the following ways:

**Cursor to Selected** Snaps the cursor to the center of the current selection.

**Cursor to Center** Snaps the cursor to the origin of the scene (location 0,0,0).

**Cursor to Grid** Snaps the cursor to the nearest *visible* grid lines.

**Cursor to Active** Snaps the cursor to the *active* (last selected) object, edge, face or vertex.

The *Cursor to Selected* option is also affected by the current *Pivot Point*. For example:

- With the *Bounding Box Center* pivot point active, the *Cursor to Selected* option will snap the 3D cursor to the center of the bounding box surrounding the objects' centers.
- When the *Median Point* pivot point is selected, *Cursor to Selected* will snap the 3D cursor to the **median** of the object centers.

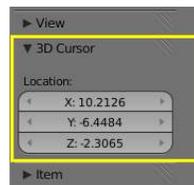


Fig. 2.42: The 3D Cursor panel of the Properties shelf.

**Numeric Input** The 3D cursor can also be positioned by entering Numeric location values into the *3D cursor* panel of the *Properties* shelf (N).

## Background Images

---

### Reference

Editor: *3D View*

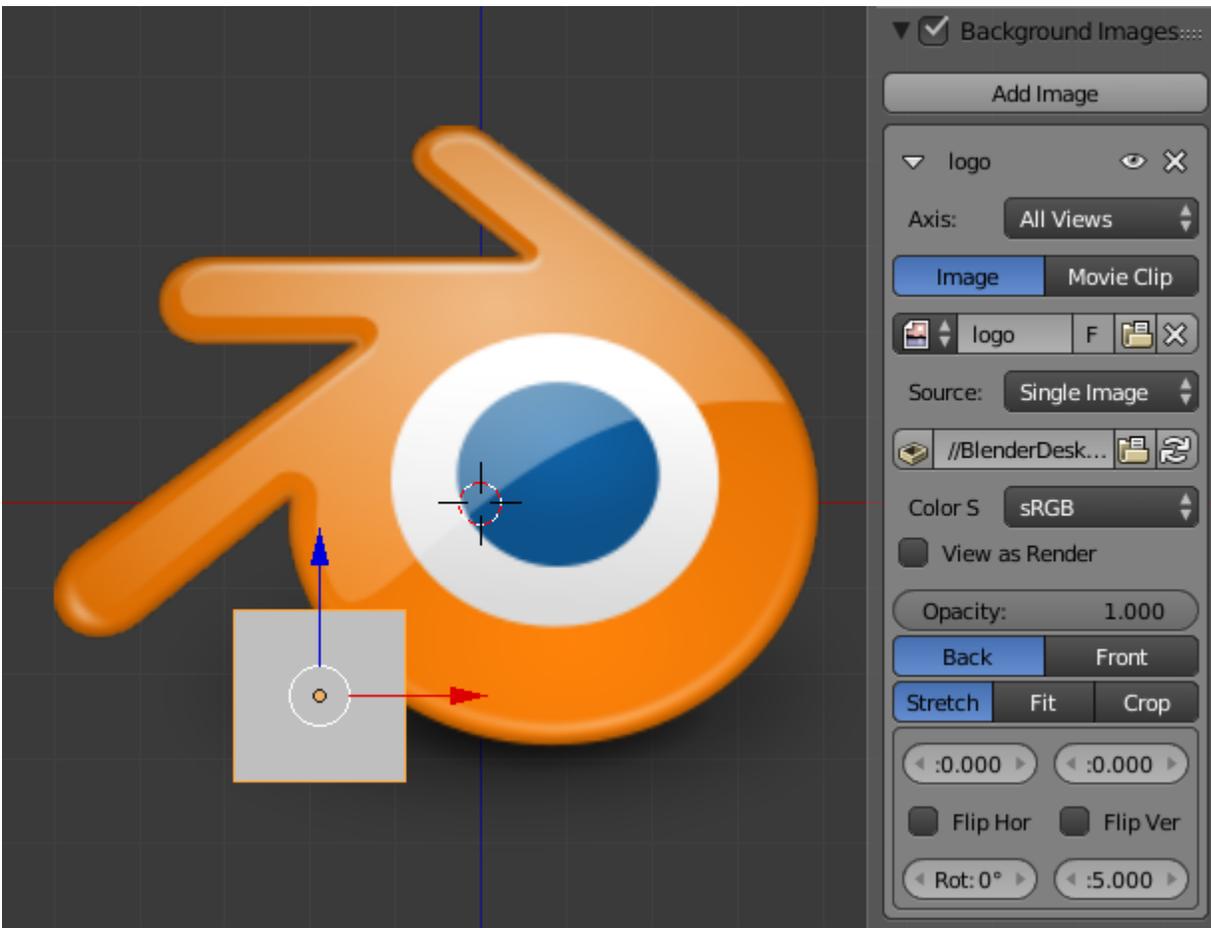
Panel: *Background Image*

---

A background picture in your 3D view is very helpful in many situations: modeling is obviously one, but it is also useful when painting (e.g. you can have reference pictures of faces when painting textures directly on your model...), or animation (when using a video as background), etc.

**Note:** Background images are only available for orthographic views.

## Settings



**Axis** Choose which views the image is visible from. This is helpful when you have several reference images from different views (e.g. top, front and side).

**Data Source** The source of the background image.

**Image** Use an external image, image sequence, video file or generated texture.

**Movie Clip** Use one of the Movie Clip datablocks.

**Opacity** Controls the transparency of the background image.

**Front/Back** Choose whether the image is shown behind all objects, or in front of everything.

**Stretch/Fit/Crop** Controls how the image is placed in the camera view.

**Stretch** Forces the image dimensions to match the camera bounds (may alter the aspect ratio).

**Fit** Scales the image down to fit inside the camera view without altering the aspect ratio.

**Crop** Scales the image up so that it fills the entire camera view, but without altering the aspect ratio (some of the image will be cropped)

**X/Y** Position the background image using these offsets.

In orthographic views, this is measured in the normal scene units. In the camera view, this is measured relative to the camera bounds (0.1 will offset it by 10% of the view width/height)

**Flip Horizontally** Swap the image around, such that the left side is now on the right, and the right now on the left.

**Flip Vertically** Swap the image around, such that the top side is now on the bottom, and the bottom now on the top.

**Rotation** Rotate the image around its center.

**Size** Scale the image up or down from its center.

## 2.2 Data System

### 2.2.1 Introduction

Each `.blend` file contains a database. This database contains all scenes, objects, meshes, textures, etc. that are in the file.

A file can contain multiple scenes and each scene can contain multiple objects. Objects can contain multiple materials which can contain many textures. It is also possible to create links between different objects.

#### Outliner

You can easily inspect the contents of your file by using the *Outliner* editor, which displays all of the data in your `.blend` file.

The *Outliner* allows you to do simple operations on objects, such as selecting, renaming, deleting, linking and parenting.

[Read more about the Outliner](#)

#### Pack and Unpack Data

Blender has the ability to encapsulate (incorporate) various kinds of data within the `.blend` file that is normally saved outside of the `.blend` file. For example, an image texture that is an external `.jpg` file can be put “inside” the `.blend` file via *File* → *External Data* → *Pack into .blend file*. When the `.blend` file is saved, a copy of that `.jpg` file is put inside the `.blend` file. The `.blend` file can then be copied or emailed anywhere, and the image texture moves with it.

You know that an image texture is packed because you will see a little “Christmas present gift box” displayed in the header.

#### Unpack Data

When you have received a packed file, you can *File* → *External Data* → *Unpack into Files...* You will be presented with the option to create the original directory structure or put the file in the `//` (directory where the `.blend` file is). Use “original locations” if you will be modifying the textures and re-packing and exchanging `.blend` files, so that when you send it back and the originator unpacks, his copies of the textures will be updated.

### 2.2.2 Datablocks

The base unit for any Blender project is the data-block. Examples of data-blocks include: meshes, objects, materials, textures, node-trees, scenes, texts, brushes and even screens.

For clarity, bones, sequence strips and vertex groups are **not** data-blocks, they belong to armature, scene and mesh types respectively.

Some common characteristics:

- They're the primary contents of the `.blend` file.
- They can link to each other, for reuse and instancing. (*child/parent, object/object-data, with modifiers and constraints too*).
- Their names are unique.
- They can be added/removed/edited/duplicated.
- They can be linked between files (*only enabled for a limited set of data-blocks*)
- They can have their own animation data.
- They can have custom properties.

When doing more complex projects managing data-blocks becomes more important, especially when inter-linking `.blend` files.

## Users (Garbage Collection)

It's good to be aware of how Blender, handles data-blocks life-time, when they are freed and why.

Blender follows the general rule that data which isn't used, will be freed.

Its common to add and remove a lot of data while working, this has the advantage of not having to manually manage every single data-block.

This works by skipping zero user data-blocks when writing `.blend` files.

While this is acceptable as default behavior, there are times when you want to save a data-block even when its unused (*typically for re-usable asset libraries*). see *Fake User*.

### Fake User

Since zero user data-blocks aren't saved. There are times when you want to force the data to be kept irrespective of its users.

If you're building a `.blend` file to serve as a library of things that you intend to link-to from *other* files, you'll need to make sure that they don't accidentally get deleted from the library file.

Do this by giving the data-blocks a *Fake User*, by pressing the *F* button next to the name of the data-block. This prevents the user count from ever becoming zero: therefore, the data-block won't be deleted. (since Blender doesn't keep track of how many other files link to this one.)

## Users (Sharing)

Many data-blocks can be shared among other data-blocks,

Examples where sharing data is common.

- Sharing textures among materials.
- Sharing meshes between objects (instances).
- Sharing animated actions between objects, for example to make all the lights dim together.

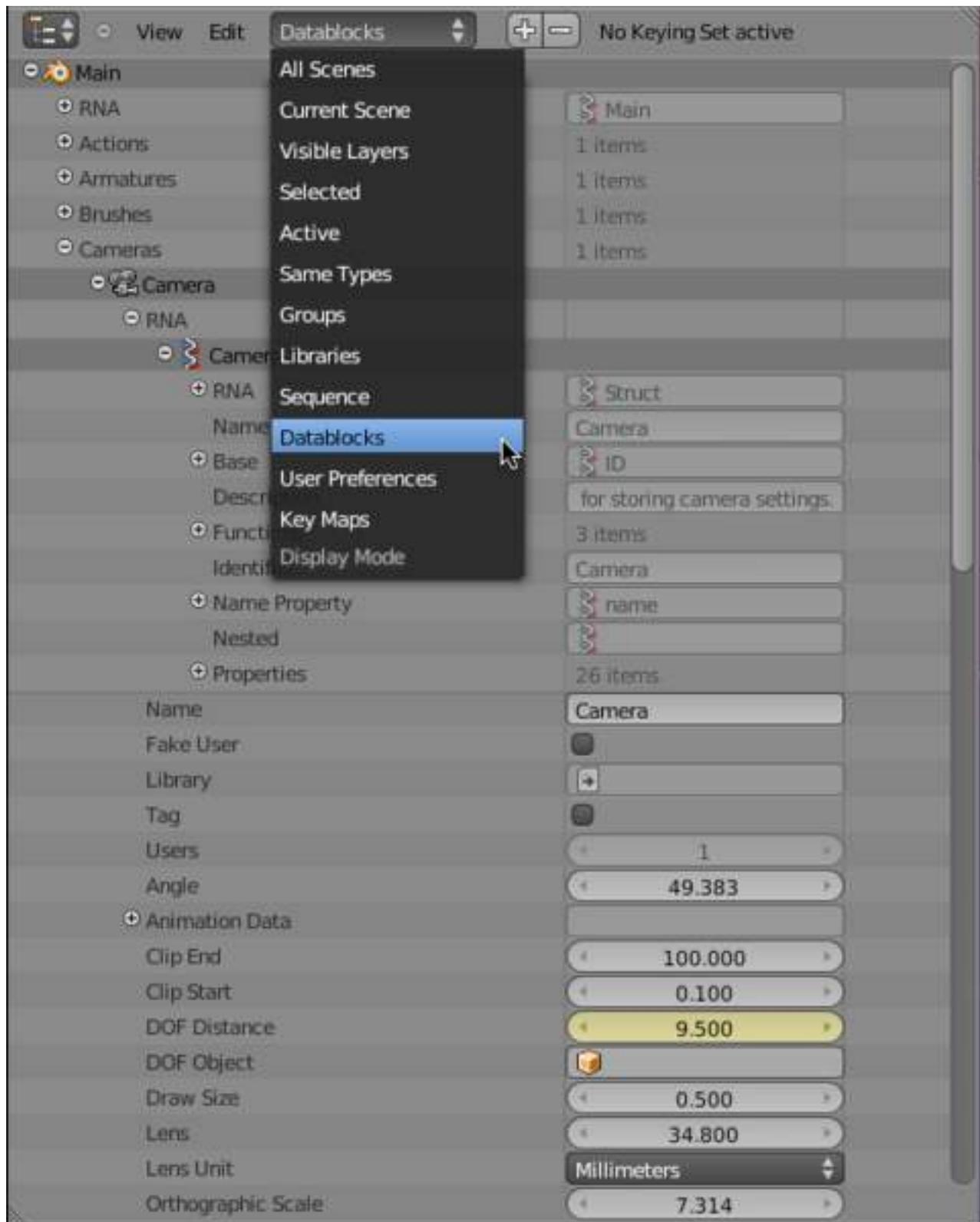


Fig. 2.43: Datablocks view

## Copying and Linking Objects Between Scenes

Sometimes you may want to link or copy objects between scenes. This is possible by first selecting objects you want to link or copy and then using the *Make Links* and *Make Single User* items found in *Object* menu in the 3D viewport header. Use *Make Links* to make links between scenes. To make a plain copy, you first make a link and then use *Make Single User* to make a stand-alone copy of the object in your current scene. Further information on working with scenes can be found [here](#).

## Appending or Linking Across Files

The content of one `.blend` file is easily accessed and put into your current file by using *File* → *Append* or *File* → *Link To* find out more about how to copy or link objects across `.blend` files, see [linked libraries](#).

## Copying and Linking

It is possible to copy or link most of Blender's data-block.

See:

- [Adding scenes](#)
- Object duplication

When an *ObData* data-block is used (linked) by more than one object, a small button with its number of linked objects (users) shows up next to its name (*also visible for materials, textures, images*). If you click on it, you create a single-user copy of this data-block for the current object.

## Removing Datablocks

As covered in *Users (Garbage Collection)*, data-blocks are typically removed when they're no longer used.

There are some exceptions to this however.

Scenes, text, can be removed directly.

Other data-blocks such as groups and actions can be *Unlinked* from the *Outliner* context menu.

---

**Tip:** Some data (images especially) is hard to keep track of, especially since image views are counted as users.

For data-blocks that can be unlinked - hold `Shift` while pressing on the `X` button, This force-clears the user-count, so the data-block will be removed on reload.

---

### 2.2.3 Scenes

Scenes are a way to organize your work. Each `.blend` file can contain multiple scenes which share other data such as objects and materials

Scene management and library appending/linking is based on Blender's [Library and Data System](#), so it is a good idea to read that manual page first if you're not familiar with the basics of that system.

#### Adding a Scene

To add a scene, click on the scene list button, and select *Add New*. While you are adding a new scene, you have these options:

**Empty** Create a completely empty scene.



Fig. 2.44: Add scene pop-up menu.

**Link Objects** All objects are linked to the new scene. The layer and selection flags of the objects can be configured differently for each scene.

**Link ObData** Duplicates objects only. ObData linked to the objects, e.g. mesh and curve, are not duplicated.

**Full Copy** Everything is duplicated.

The new scene can be renamed by clicking in the text field and typing a name.

### Linking to a Scene

You can link any object from one scene to another. Just open the scene where these objects are, use `Ctrl-L -> Objects to Scene...` and choose the scene where you want your objects to appear. The selected objects will be added to that scene but remain linked to the original objects.

To make them single user (independent and unlinked) in a given scene go to that scene, select them and press `U`. You will be presented with a few options that allow you to free up the datablocks (Object, Material, Texture...) that you want.

### Removing a scene from the file

You can delete the current scene by clicking the *X* next to the name in the Info Editor.

## 2.2.4 Files

### Introduction

The options to manage files are:

**New** Clears the current scene and loads startup.blend

**Open** Open a blend file

**Open Recent** Displays a list of recently saved .blend files to open

**Recover last session** This will load the quit.blend file Blender automatically saves just before exiting. So this option enables you to recover your last work session, e.g. if you closed Blender by accident

**Recover Auto Save** This will open an automatically saved file to recover it.

**Save** Save the current blend file.

**Save As** Opens file browser to specify file name and location of save.

**Save Copy** Saves a copy of the current file.

**User Preferences** Opens the user preferences dialog.

**Save User Settings** Saves the current scene and preferences to startup.blend.

**Load Factory Settings** Restore the default scene to the factory settings.

**Link or Append** You don't have to load a complete file, you can load in only selected parts from another file if you wish.

**Import** Blender can use information stored in a variety of other format files which are created by other graphics programs.

**Export** Normally you save your work in a .blend file, but you can export some or all of your work to a format that can be processed by other graphics programs.

### External Data

**Pack into .blend** Pack all used external files into the .blend

**Unpack into Files** Unpack all files packed into this .blend to external ones

**Make all paths Relative** Make all paths to external files relative to current .blend

**Make all paths Absolute** Make all paths to external files absolute

**Report Missing Files** Report all missing external files

**Find Missing Files** Try to find missing external files

## Opening Files



## Reference

Menu: *File* → *Open*

Hotkey: *Ctrl-O* or *F1*

To load a Blender file from disk, press *Ctrl-O* or *F1*. The *File Browser* window, as shown above, will open.

The upper text box displays the current directory path, and the lower text box contains the selected filename.

The + and - buttons to the right of the file name allow you to cycle through numbered files by increasing or decreasing the number at the end of the file name.

Pressing *P* or clicking the up-arrow icon above the list of files will move you up to the parent directory.

Click on a folder to go inside of it, Click on a file then click the *Open Blender File* button or press *Return* to open it

Clicking *Cancel* will close the file browser window and return to the program.

**Warning:** Blender expects that you know what you are doing! When you load a file, you are **not** asked to save unsaved changes to the scene you were previously working on, completing the file load dialog is regarded as being enough confirmation that you didn't do this by accident.

## Sidebar

The left sidebar displays different ways to find files and several options.

**System** The system menu contains a list of drives that are available to navigate through to find files. Click on one to jump to that drive.

**Bookmarks** These are folders that you want to be able to access often without having to navigate to them in the file browser. To add a directory to the bookmark menu, navigate to that folder, then click the *Add* button. To remove a folder from the list, simply click the *X* icon next to it.

**Recent** This is a list of recently accessed folders. You can control how many folders appear in this list by going to the *File* tab of the [User Preferences](#), in the box labeled *Recent Files*.

## Options

**Load UI** Inside each .blend file, Blender saves the user interface arrangement. By default, this saved UI is loaded, overriding any user defaults or current screen layouts that you have. If you want to work on the blend file using your own defaults, start a fresh Blender, then open the file browser and turn off the *Load UI* button, and then open the file.

**Trusted Source** When enabled, python scripts and drivers that may be included in the file will be run automatically. Enable this only if you created the file yourself, or you trust that the person who gave it to you did not include any malicious code with it. See [Security](#) below.

## Header

The Header contains several tools for navigation of files. The four arrow icons allow you to:

- *Move to previous folder*
- *Move to next folder*
- *Move up to parent directory*
- *Refresh current folder*

Create a new folder inside the current one by clicking the *Create New Directory* button.

The other icons allow you to control what files are visible and how they are displayed. You can:

- *Display files as a short list*
- *Display files as a detailed list*
- *Display files as thumbnails*

You can sort files:

- *Alphabetically*
- *By file type*
- *By Date of last edit*
- *By file size*

Click the funnel icon to toggle which file types are shown:

- *Folders*
- *Blend files*
- *Images*
- *Movie files*
- *Scripts*
- *Font files*
- *Music files*
- *Text files*

### Other File Open Options

From the *File* menu, you can also open files with the following tools:

**Open Recent** Lists recently used files. Click on one to load it in.

**Recover Last Session** This will load the `quit.blend` file Blender automatically saved just before exiting. This option enables you to recover your last work session if, for example, you closed Blender by accident.

**Recover Auto Save** This will allow you to open an automatically saved file to recover it.

**See also:**

*Auto Saves*

### Security

**Warning:** Always be very careful when downloading `.blend` files and tools from un-trustworthy sources!

Blender is aimed at production level use and relies heavily on Python, a powerful scripting language used to create new tools, importers drive animation rigs, etc.

Part of Python's power comes from having full access to your system, however this power can also be misused in the wrong hands. It is possible for dishonest people to distribute `.blend` files containing scripts that may damage your system. These scripts can be attached as part of animation rigs, so that they will be run when such a `.blend` file is opened.

### Saving Files

---

#### Reference

Editor: Info

Menu: File

---

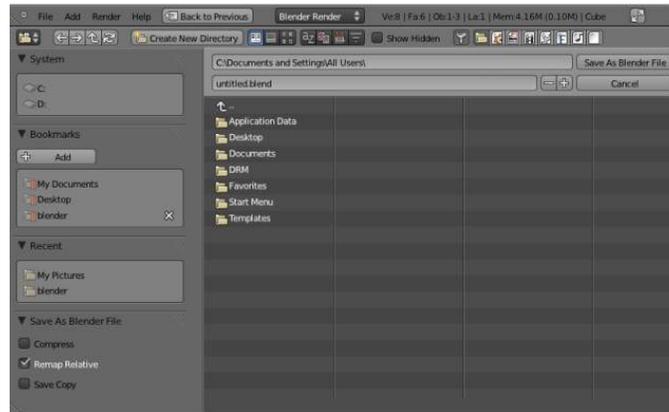
There are a number of slightly different methods you can use to save your blend file to your hard drive:

**Save (Ctrl-S, Ctrl-W)** Save an existing blend file over itself.

**Save As (Ctrl-Shift-S, F2)** Choose a file to save the blend to.

**Save Copy (Ctrl-Alt-S)** Choose a file to save the blend to, but return to editing the original file upon completion. This can be used to save backups of the current working state without modifying the original file.

If the file name doesn't end with `.blend`, the extension is automatically appended. If a file with the same given name already exists, the text field will turn red as a warning.



## Save Versions

Depending on the number of Save Versions you have set, all existing files with the same name will be rotated to a `.blend#` file extension, where # is 1, 2, 3, etc.

So, if you were working on `MyWork.blend`, and saved it, the existing `MyWork.blend` is renamed to `MyWork.blend1`, and a new `MyWork.blend` is saved. This way, you have hot backups of old saved versions in case you need to massively undo changes, or accidentally saved over the wrong file.

---

**Tip:** Use the **plus/minus** buttons to the right of the file name, or `NumpadPlus/NumpadMinus` to increase/decrease a number at the end of the file name (e.g. changing `file_01.blend` to `file_02.blend`).

---

## Options

The save options appear at the bottom of the sidebar.

**Compress File** When enabled, the saved file will be smaller, but take longer to save and load.

**Remap Relative** This option remaps [relative paths](#) (such as linked libraries and images) when saving a file in a new location.

**Save Copy** This option saves a copy of the actual working state, but does not make the saved file active.

**Legacy Mesh Format** Save the blend file, but ignore faces with more than 4 vertices (“ngons”) so that older versions of Blender (before 2.63) can open it.

## Importing and Exporting Files

Blender supports import and export to and from other file formats (e.g. OBJ, FBX, 3DS, PLY... etc).

These formats can be accessed from the menus: *File* → *Import* and *File* → *Export*.

Popular formats are enabled by default, other formats are also supported and distributed with Blender, these can be enabled in the user-preferences through the use of [Add-ons](#).

A list of these add-ons can be found on the [Blender Add-ons Catalog](#)

## Relative Paths

Many Blender files reference external images or other linked .blend files. A path tells Blender where to look for these files. If the external files are moved, the blend file that references them won't look right.

When you specify one of these external files, the default option is to make the path relative. Blender stores a partial path evaluated relative to the directory location of the referencing blend file. This choice helps when you need to reorganize folders or move your files.

With a relative path you can move the .blend file to a new location provided the externally linked files are moved along with it. For example you could send someone a folder that contains a .blend file and a sub-folder of external images that it references.

Most file selection windows provide a *Relative Path* check box, or when you type in a path into a text field, use a double slash prefix (//) to make it so.

Relative paths is the default but this can be changed in the [File Preferences Tab](#) of the User Preferences Editor.

---

**Note:** You can't enter relative paths into a new *untitled* blend file. Save it before linking to external files.

---

**Hint:** If it's necessary to relocate a blend file relative to its linked resources, use Blender's File [Save As](#) function which has an option to *Remap Relative* file links.

---

## Supported Graphics Formats

### Image Formats

This is the list of image file formats supported internally by Blender:

Format	<i>Channel Depth</i>	Alpha	<i>Metadata</i>	DPI	Extensions
BMP	8bit				.bmp
Iris	8bit				.sgi .rgb .bw
PNG	8, 16bit				.png
JPEG	8bit				.jpg .jpeg
JPEG 2000	8, 12, 16bit				.jp2 .j2k .j2c
Targa	8bit				.tga
<i>Cineon &amp; DPX</i>	8, 10, 12, 16bit				.cin .dpx
<i>OpenEXR</i>	float 16, 32bit				.exr
<i>Radiance HDR</i>	float				.hdr
TIFF	8, 16bit				.tif .tiff

**Hint:** If you aren't interested in technical details, a good rule of thumb for selecting an output formats for your project is:

Use **OpenEXR** if you intend to do compositing or color-grading on these images.

Use **PNG** if you intend on-screen output or encoding into multiple video formats.

Use **JPEG** for on-screen output where file size is a concern and quality loss is acceptable.

*All these formats support compression which can be important when rendering out animations.*

---

**Note:** Quicktime

On OSX, Quicktime can be used to access file formats not natively supported (such as GIF).

---

**Channel Depth** Image file formats support a varying number of bits per pixel. This effects the color quality and file-size.

Commonly used depths:

**8 bit (256 levels)** Most common for on-screen graphics and video

**10,12,16 bit (1024,4096,65536 levels)** Used for some formats focusing on photography and digital film formats (such as DPX and JPEG 2000).

**16 bit half float** Since full 32bit float is often more than enough precision, half float can save on disk-space while providing high dynamic range.

**32 bit float** Highest quality color depth.

Internally Blender’s image system supports either:

- 8 bit per channel (4 x 8 bits).
- 32 bit float per channel (4 x 32 bits) - *using 4x as much memory.*

Images higher than 8 bits per channel will be converted into float on loading into Blender.

---

**Note:** Floating point is often used for *HDRI*,

When an image has float colors, all imaging functions in Blender default to use that. This includes the Video Sequence Editor, texture mapping, background images, and the Compositor.

---

**Metadata** Blender can save details such as render-time, marker, camera... etc, into the file. See: [Render Metadata](#).

Only some files support this however.

### Format Details

**Cineon & DPX** Cineon is Kodak’s standard for film scanning, 10 bits/channel and logarithmic. DPX has been derived from Cineon as the ANSI/SMPTE industry standard. DPX supports 16 bits color/channel, linear as well as logarithmic. DPX is currently a widely adopted standard used in the film hardware/software industry.

DPX as well as Cineon only stores and converts the “visible” color range of values between 0.0 and 1.0 (as result of rendering or composite).

---

**OpenEXR** ILM’s [OpenEXR](#) has become a software industry standard for HDR image files, especially because of its flexible and expandable structure.

An OpenEXR file can store multiple layers and passes. This means OpenEXR images can be loaded into a compositor keeping render layers, passes intact.

**Output Options** Available options for OpenEXR render output are:

**Half** Saves images in a custom 16 bits per channel floating point format. This reduces the actual “bit depth” to 10 bits, with a 5 bits power value and 1 bit sign.

**Zbuf** Save the depth information. In Blender this now is written in floats too, denoting the exact distance from the camera in “Blender unit” values.

**Preview** On rendering animations (or single frames via command line), Blender saves the same image also as a JPEG, for quick preview or download.

**Compression** *This button is below the Image menu button, default set to “None”*

**PIZ** lossless wavelet compression. Compresses images with grain well.

**ZIP** standard lossless compression using zlib.

**RLE** runlength encoded, lossless, works well when scanlines have same values.

**PXR24** lossy algorithm from Pixar, converting 32 bits floats to 24 bits floats.

**Radiance HDR** Radiance is a suite of tools for lighting simulation. Since Radiance had the first (and for a long time the only) HDR image format, this format is supported by many other software packages.

Radiance (.hdr) files store colors still in 8 bits per component, but with an additional (shared) 8 bits exponent value, making it 32 bits per pixel.

## Supported Video Formats

### Video Formats

These formats are primarily used for compressing rendered sequences into a playable movie (they can also be used to make plain audio files).

A codec is a little routine that compresses the video so that it will fit on a DVD, or be able to be streamed out over the Internet, over a cable, or just be a reasonable file size. Codecs compress the channels of a video down to save space and enable continuous playback. *Lossy* codecs make smaller files at the expense of image quality. Some codecs, like H.264, are great for larger images. Codecs are used to encode and decode the movie, and so must be present on both the encoding machine (Blender) and the target machine. The results of the encoding are stored in a container file.

There are dozens, if not hundreds, of codecs, including XviD, H.264, DivX, Microsoft, and so on. Each has advantages and disadvantages and compatibility with different players on different operating systems.

Most codecs can only compress the RGB or YUV color space, but some support the Alpha channel as well. Codecs that support RGBA include:

- animation (quicktime)
- PNG TIFF Pixlet - not loss-less, and may be only available on Apple Mac.
- [Lagarith Loss-less Video Codec](#)

**AVI Codec** AVI codec compression. Available codecs are operating-system dependent. When an AVI codec is initially chosen, the codec dialog is automatically launched. The codec can be changed directly using the *Set Codec* button which appears (*AVI Codec settings*).

**AVI Jpeg** AVI but with Jpeg compression. Lossy, smaller files but not as small as you can get with a Codec compression algorithm. Jpeg compression is also the one used in the DV format used in digital camcorders.

**AVI Raw** Audio-Video Interlaced (AVI) uncompressed frames.

**Frameserver** Blender puts out [frames upon request](#) as part of a render farm. The port number is specified in the OpenGL User Preferences panel.

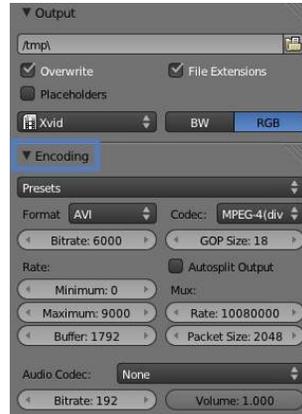
**H.264** Encodes movies with the H.264 codec. See [Advanced Encoding](#).

**MPEG** Encodes movies with the MPEG codec. See [Advanced Encoding](#).

**Ogg Theora** Encodes movies with the Theora codec as Ogg files. See [Advanced Encoding](#).

**QuickTime** Apple's Quicktime `.mov` file. The Quicktime codec dialog is available when this codec is installed on OSX. See *Quicktime* in *Video Formats*.

**Xvid** Encodes movies with the Xvid codec. See *Advanced Encoding*.



**Advanced Encoding** If the *H.264*, *MPEG*, *Ogg Theora*, or *Xvid* codecs are chosen, an *Encoding* panel becomes available. This has settings for encoding these file types, and other formats using FFmpeg.

**FFmpeg**, short for Fast Forward Moving Pictures Expert Group, is a collection of free and open source software libraries that can record, convert and stream digital audio and video in numerous formats. It includes libavcodec, an audio/video codec library used by several other projects, and libavformat, an audio/video container mux and demux library.

**Video Settings** Here you choose which video codec you want to use, and compression settings. With all of these compression choices, there is a tradeoff between file size, compatibility across platforms, and playback quality.

When you view the *System Console*, you can see some of the output of the encoding process. You will see even more output if you execute Blender as `blender -d`.

You can use the presets, DV, SVCD, DVD, etc. which choose optimum settings for you for that type of output, or you can manually select the format (MPEG-1, MPEG-2, MPEG-4, AVI, Quicktime (if installed), DV, H.264, or Xvid (if installed)). You must have the proper codec installed on your computer for Blender to be able to call it and use it to compress the video stream.

## Video Containers

**MPEG-1: .mpg, .mpeg** A standard for lossy compression of video and audio. It is designed to compress VHS-quality raw digital video and CD audio down to 1.5 Mbit/s.

**MPEG-2: .dvd, .vob, .mpg, .mpeg** A standard for “the generic coding of moving pictures and associated audio information”. It describes a combination of lossy video compression and lossy audio data compression methods which permit storage and transmission of movies using currently available storage media and transmission bandwidth.

**MPEG-4(DivX): .mp4, .mpg, .mpeg** Absorbs many of the features of MPEG-1 and MPEG-2 and other related standards, and adds new features.

**AVI: .avi** A derivative of the Resource Interchange File Format (RIFF), which divides a file's data into blocks, or “chunks.”

**Quicktime: .mov** A multi-tracked format. QuickTime and MP4 container formats can use the same MPEG-4 formats; they are mostly interchangeable in a QuickTime-only environment. MP4, being an international standard, has more support.

**DV: .dv** An intraframe video compression scheme, which uses the discrete cosine transform (DCT) to compress video on a frame-by-frame basis. Audio is stored uncompressed.

**H.264: .avi for now.** A standard for video compression, and is currently one of the most commonly used formats for the recording, compression, and distribution of high definition video.

**Xvid: .avi for now** A video codec library following the MPEG-4 standard. It uses ASP features such as b-frames, global and quarter pixel motion compensation, lumi masking, trellis quantization, and H.263, MPEG and custom quantization matrices. Xvid is a primary competitor of the DivX Pro Codec.

**Ogg: .ogg, .ogv** A free lossy video compression format. It is developed by the Xiph.Org Foundation and distributed without licensing fees.

**Matroska: .mkv** An open standard free container format, a file format that can hold an unlimited number of video, audio, picture or subtitle tracks in one file.

**Flash: .flv** A container file format used to deliver video over the Internet using Adobe Flash Player.

**Wav: .wav** An uncompressed (or lightly compressed) Microsoft and IBM audio file format.

**Mp3: .mp3** A highly-compressed, patented digital audio encoding format using a form of lossy data compression. It is a common audio format for consumer audio storage, as well as a de facto standard of digital audio compression for the transfer and playback of music on digital audio players.

### Video Codecs

**None** For audio-only encoding.

**MPEG-1** See *Video Formats*.

**MPEG-2** See *Video Formats*.

**MPEG-4(DivX)** See *Video Formats*.

**HuffYUV** Loss-less video codec created by Ben Rudiak-Gould which is meant to replace uncompressed YCbCr as a video capture format.

**DV** See *Video Formats*.

**H.264** See *Video Formats*.

**Xvid** See *Video Formats*.

**Theora** See Ogg in *Video Formats*.

**Flash Video** See *Video Formats*.

**FFmpeg video codec #1** A.K.A. FFV1, a loss-less intra-frame video codec. It can use either variable length coding or arithmetic coding for entropy coding. The encoder and decoder are part of the free, open-source library libavcodec in FFMpeg.

### Options

**Bitrate** Set the average **bitrate** (quality), which is the count of binary digits per frame. See also: `ffmpeg -b:v`

**Rate** The bitrate control also includes a *Minimum* and a *Maximum*.

**Buffer** The **decoder bitstream buffer** size.

**GOP Size** The number of pictures per **Group of Pictures**. Set to 0 for “intra\_only”, which disables **inter-frame** video. From ffmpeg docs: “For streaming at very low bitrate application, use a low frame rate and a small GOP size. This is especially true for RealVideo where the Linux player does not seem to be very fast, so it can miss frames”

**Autosplit Output** If your video is HUGE and exceeds 2Gig, enable Autosplit Output. The main control over output filesize is the GOP, or keyframe interlace. A higher number generally leads to a smaller file, but needs a higher-powered device to replay it.

**Mux** **Multiplexing** settings.

**Rate** Maximum bit rate of the multiplexed stream.

**Packet Size** (Undocumented in ffmpeg)

---

**Note:** Standards

Codecs cannot encode off-the-wall video sizes, so stick to the XY sizes used in the presets for standard TV sizes.

---

**Audio Settings** Audio is encoded using the codec you choose.

Audio Codecs

**MP2** A lossy audio compression format defined by ISO/IEC 11172-3.

**MP3** See MP3 in *Video Formats* above.)

**AC3** Audio Codec 3, an audio compression technology developed by Dolby Laboratories.

**AAC** Advanced Audio Codec,” a standardized, lossy compression and encoding scheme for digital audio. Designed to be the successor of the MP3 format, AAC generally achieves better sound quality than MP3 at similar bit rates.

**Vorbis** An open-standard, highly-compressed format comparable to MP3 or AAC. Had been shown to perform significantly better than many other lossy audio formats in the past in that it produced smaller files at equivalent or higher quality while retaining computational complexity comparable to other MDCT formats such as AAC or Windows Media Audio.

**FLAC** Free Loss-less Audio Codec. Digital audio compressed by FLAC’s algorithm can typically be reduced to 50-60% of its original size, and decompressed into an identical copy of the original audio data.

**PCM** Pulse Code Modulation, a method used to digitally represent sampled analog signals. It is the standard form for digital audio in computers and various Blu-ray, Compact Disc and DVD formats, as well as other uses such as digital telephone systems

**Bitrate** For each codec, you can to control the bitrate (quality) of the sound in the movie. This example shows MP3 encoding at 128kbps. Higher bitrates are bigger files that stream worse but sound better. Stick to powers of 2 for compatibility.

**Samplerate** Samplerate controls the number of samples per second of the audio. The default, 44100, is standard for many file types, including CD audio, and produces a high quality sound.

**Volume** Set the output volume of the audio.

Tips

---

Choosing which format to use depends on what you are going to do with the image.

If you are animating a movie and are not going to do any post-processing or special effects on it, use either **AVI-JPEG** or **AVI Codec** and choose the XviD open codec. If you want to output your movie with sound that you have loaded into the VSE, use **FFMPEG**.

If you are going to do post-processing on your movie, it is best to use a frame set rendered as **OpenEXR** images; if you only want one file, then choose **AVI Raw**. While AVI Raw is huge, it preserves the exact quality of output for post-processing. After post-processing (compositing and/or sequencing), you can compress it down. You don’t want to post-process a compressed file, because the compression artifacts might throw off what you are trying to accomplish with the post-processing.

Note that you might not want to render directly to a video format. If a problem occurs while rendering, you have to re-render all frames from the beginning. If you first render out a set of static images (such as the default PNG, or the higher-quality OpenEXR), you can stitch them together with an Image Strip in the [Video Sequence Editor \(VSE\)](#). This way, you can easily:

- Restart the rendering from the place (the frame) where the problem occurred.
- Try out different video options in seconds, rather than minutes or hours.
- Enjoy the rest of the features of the VSE, such as adding Image Strips from previous renders, audio, video clips, etc.

## 2.2.5 Append and Link

These functions help you reuse materials, objects and other [datablocks](#) loaded from an external source .blend file. You can build libraries of common content and share them across multiple referencing files.

*Link* creates a reference to the data in the source file such that changes made there will be reflected in the referencing file the next time it is reloaded.

Where as *Append* makes a full copy of the data into your .blend. You can make further edits to your local copy of the data, but changes in the external source file will not be reflected in the referencing file.

---

### Reference

Mode: All Modes

Menu: *File* → *Append or Link*

Hotkey: Shift-F1 or Ctrl-Alt-O

---

In the *File Browser* window navigate to the external source .blend file and select the datablock you want to reuse.

Options:

**Relative Path** Available only when linking, see [relative paths](#).

**Select** Makes the object *Active* after it is loaded.

**Active Layer** Enabled by default, the object is assigned to the visible layers in your scene. Otherwise, it is assigned to the same layers it resides on in the source file.

**Instance Groups** This option links the Group to an object, adding it to the active scene.

When you select an Object type, it will be placed in your scene at the cursor. Many other data types - cameras, curves, and materials for example - must be linked to an object before they become visible. Newly added Group types are available in *Add* → *Group Instances* in 3D View, or for NodeTree groups, the same menu in the Node Editor.

Look in the Outliner, with display mode set to *Blender File*, to see all your linked and appended datablocks. Ctrl-LMB on a file name allows you to redirect a link to another file.

---

**Hint:** You cannot move a linked object. Its position is defined in its source file. Use *Object* → *Make Local* → *Selected Objects* to make the position editable.

---

### Proxy Objects

Used with rigged models, proxy objects, allow specified bone layers to be linked back to the source file while the remainder of the object and its skeleton are edited locally.

Ctrl-Alt-P makes the active linked object into a local proxy, appending *\_proxy* to its name.

Set the *Protected Layers* in the source file using using the Skeleton panel of the Armatures tab. See [Armature Layers](#). The bones in protected layers will have their position restored from the source file when the referencing file is reloaded.

## 2.3 Modeling

### 2.3.1 Introduction

The creation of a 3D scene needs at least three key components: Models, Materials and Lights. In this part, we will delve deeper into the creation of the first of these: modeling. Modeling is the art and science of creating a surface that either mimics the shape of a real-world object or expresses your imagination of abstract objects.

There are three primary types of modeling - mesh modeling, curve/surface modeling, and meta modeling.

#### *Mesh Modeling*

Mesh modeling typically begins with a primitive shape (e.g. circle, cube, cylinder...). This **Mesh Primitive** is defined by an array of points in 3D space called vertices (singular form is *Vertex*). From there you might begin extruding faces and moving vertices to create a larger, more complex shape.

#### *Curve and Surface Modeling*

**Curve modeling** uses control points to define the shape of the curve.

**Surface modeling** is similar to curve modeling, but instead of being limited to simple linear paths, they allow the creation of three dimensional surfaces, potentially with volume.

#### *Meta Object (Metaball) Modeling*

Metaball modeling begins similarly to mesh modeling, with a base shape like a cube or sphere, but instead of extruding these base shapes, these objects are clumped together to form a larger object. In order to accomplish this, the metaballs have a liquid-like quality, when two or more are brought together they merge by smoothly rounding out the point of connection, appearing as one unified object.

This is one of the quickest ways to get started roughly modeling an object. The resulting figure can then be converted into a mesh for further detailing using `Alt-C`.

#### *Text Modeling*

Inserting text is quite common for the creation of logos, and can be seen as a special case of neither curve nor mesh modeling. You may define the text, font, bevel, extruded width and several other parameters that control generated object.

#### *Scripted Modeling*

Since Blender functionality is extensible via Python scripting, there are a number of very useful scripts that assist you in modeling. They may give you new mesh primitives to work with, or apply some fancy manipulation of the meshes that you are already working with.

Modeling scripts are generally more advanced, but also less frequently used programmatic effects that can be a huge time saver in certain cases.

The included `spin` and `screw` functions are examples of a modeling scripts that might otherwise take significantly more work to replicate by hand through mesh or curve modeling.

## 2.3.2 Objects

### Object Mode

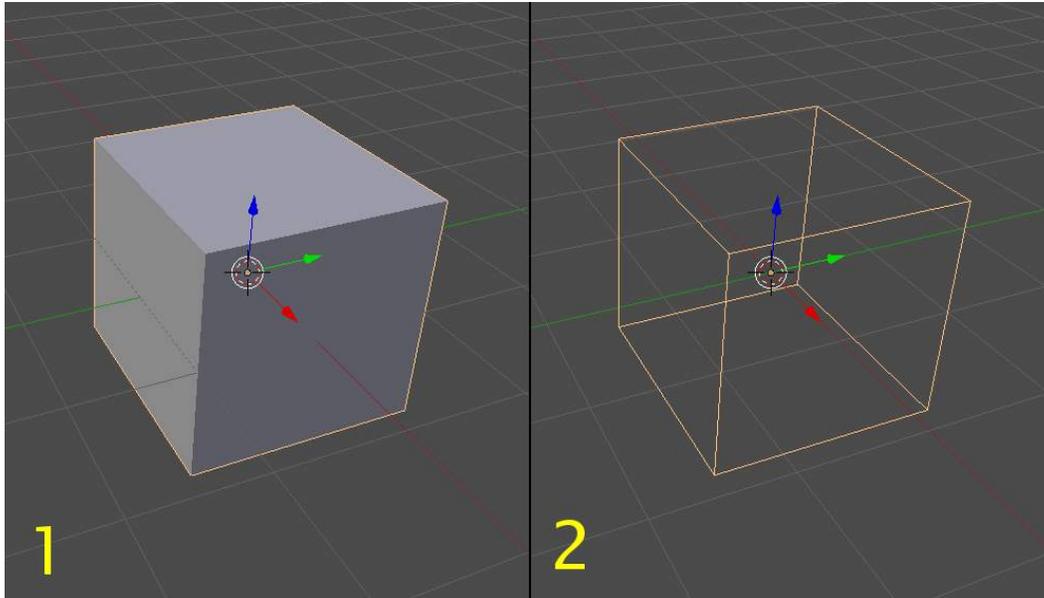


Fig. 2.45: Selected object in 3D View: (1) Solid shading, (2) Wireframe shading.

The geometry of a scene is constructed from one or more Objects. These objects can range from lamps to light your scene, basic 2D and 3D shapes to fill it with models, armatures to animate those models, to cameras to take pictures or video of it all.

### Types of Objects

**Meshes** Meshes are objects composed of Polygonal Faces, Edges and/or Vertices, and can be edited extensively with Blender's mesh editing tools. The default scene features a cube, which is one of the many included basic building-block shapes called [Mesh Primitives](#)

**Curves** Curves are mathematically defined objects which can be manipulated with control handles or control points (instead of vertices), to manage their length and curvature.

**Surfaces** Surfaces are patches that are also manipulated with control points. These are useful for simple rounded forms and organic landscapes.

**Meta Objects** Meta Objects (or Metaballs) are objects formed by a mathematical function (with no control points or vertices) defining the 3D volume in which the object exists. Meta Objects have a liquid-like quality, where when two or more Metaballs are brought together, they merge by smoothly rounding out the connection, appearing as one unified object.

**Text** Text objects create a two dimensional representation of a string of characters.

**Armatures** Armatures are used for [rigging](#) 3D models in order to make them poseable and animateable.

**Lattice** Lattices are non-renderable wireframes, commonly used for taking additional control over other objects with help of the [Lattice Modifier](#).

**Empty** Empties are null objects that are simple visual transform nodes that do not render. They are useful for controlling the position or movement of other objects.

**Speaker** Brings to scene source of sound.

**Cameras** This is the virtual camera that is used to determine what appears in the render.

**Lamps** These are used to place light sources in the scene.

**Force Fields** Force fields are used in physical simulations. They give simulations external forces, creating movement, and are represented in 3d editor by small control objects.



Fig. 2.46: Object Mode button.

Each object can be moved, rotated and scaled in *Object Mode* (see picture). However, not all of these transformations have an effect on all objects. For example, scaling a force field will not increase its effect.



Fig. 2.47: Edit Mode button.

For making other changes to the geometry of editable objects, you should use *Edit mode* (see picture).

Once you've added a basic object, you remain in *Object Mode*. In earlier versions of Blender, you were automatically switched into *Edit mode* if the Object was a Mesh, a Curve or a Surface.

You can switch between *Object Mode* and *Edit Mode* by pressing `Tab`.

The object's wireframe should now appear orange. This means that the object is now selected and active (see picture *Selected object*).

The (*Selected object*) image shows both the solid view and wireframe view of the default cube. To switch between wireframe and solid view, press `Z`.

## Object Centers

Each object has an origin point. The location of this point determines where the object is located in 3D space. When an object is selected, a small circle appears, denoting the origin point. The location of the origin point is important when translating, rotating or scaling an object. See [Pivot Points](#) for more.

**Moving Object Centers** Object Centers can be moved to different positions through *3D View window* → *Transform* → *Origin* (press `T` to open panel):

**Geometry to Origin** Move model to origin and this way origin of the object will also be at the center of the object.

**Origin to Geometry** Move origin to the center of the object and this way origin of the object will also be at the center of the object.

**Origin to 3D Cursor** Move origin of the model to the place of the 3D cursor.

**Origin to Center of Mass** Move origin to calculated center of mass of model.

## Selecting

### Introduction

Selection determines which elements will be the target of our actions. Blender has advanced selection methods. Both in *Object mode* and in *Edit mode*.

## Selections and the Active Object

Blender distinguishes between two different states of selection:

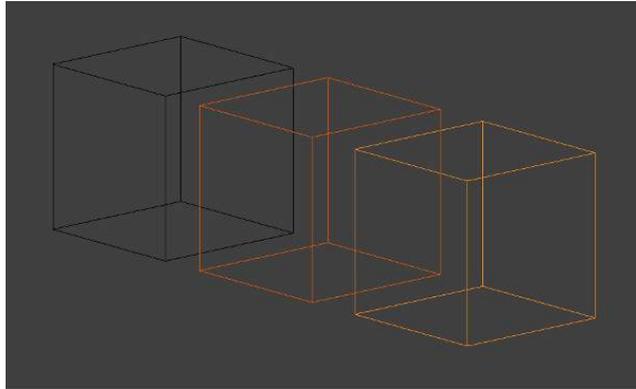


Fig. 2.48: Unselected object in black, selected object in orange, and active object in yellow

- In *Object mode* the last (de)selected item is called the “Active Object” and is outlined in yellow (the others are orange). There is exactly one active object at any time (even when nothing is selected).  
Many actions in Blender use the active object as a reference (for example linking operations). If you already have a selection and need to make a different object the active one, simply re-select it with `Shift-RMB`.
- All other selected objects are just selected. You can select any number of objects.

### Point Selection

The simplest form of object *selection* consists of using `RMB` on it.

To *add to the selection*, use `Shift-RMB` on more objects.

If the *objects are overlapping* in the view, you can use `Alt-RMB` to cycle through possible choices.

If you want to *add to a selection* this way then the shortcut becomes `Shift-Alt-RMB`.

To *activate an object* that is already selected, click `Shift-RMB` on it.

To *deselect* an active object, click `Shift-RMB` one time - and hence two clicks if the object isn't active. Note that this only works if there are no other objects under the mouse. Otherwise it just adds those to the selection. There appears to be no workaround for this bug.

### Rectangular or Border Select

---

#### Reference

Mode: *Object mode* and *Edit mode*

Menu: *Select -> Border Select*

Hotkey: `B`

---

**Description** With *Border Select* you draw a rectangle while holding down `LMB`. Any object that lies even partially within this rectangle becomes selected.

For deselecting objects, use `MMB` or *Border Select* again with holding `Shift`.

To cancel the selection use `RMB`.

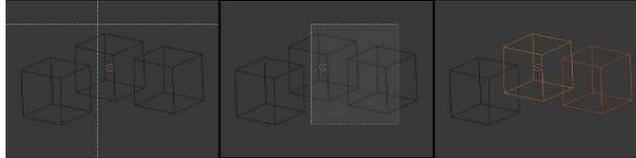


Fig. 2.49: Border selecting in three steps

**Example** *Border Select* has been activated in the first image and is indicated by showing a dotted cross-hair cursor. In the second image, the *selection region* is being chosen by drawing a rectangle with the `LMB`. The rectangle is only covering two cubes. Finally, in the third image, the selection is completed by releasing `LMB`.

Notice in the third image, the bright color of left-most selected cube. This means it is the “active object”, the last selected object prior to using the *Border Select* tool.

**Hints** *Border Select* adds to the previous selection, so in order to select only the contents of the rectangle, deselect all with `A` first.

### Lasso Select

---

#### Reference

Mode: *Object mode* and *Edit mode*

Menu: no entry in the menu

Hotkey: `Ctrl-LMB`

---

**Description** Lasso select is used by drawing a dotted line around the pivot point of the objects, in *Object mode*.

**Usage** While holding `Ctrl` down, you simply have to draw around the pivot point of each object you want to select with `LMB`.

Lasso select adds to the previous selection. For deselection, use `Ctrl-Shift-LMB`.

### Circle Select

---

#### Reference

Mode: *Object mode* and *Edit mode*

Menu: *Select* → *Circle Select*

---

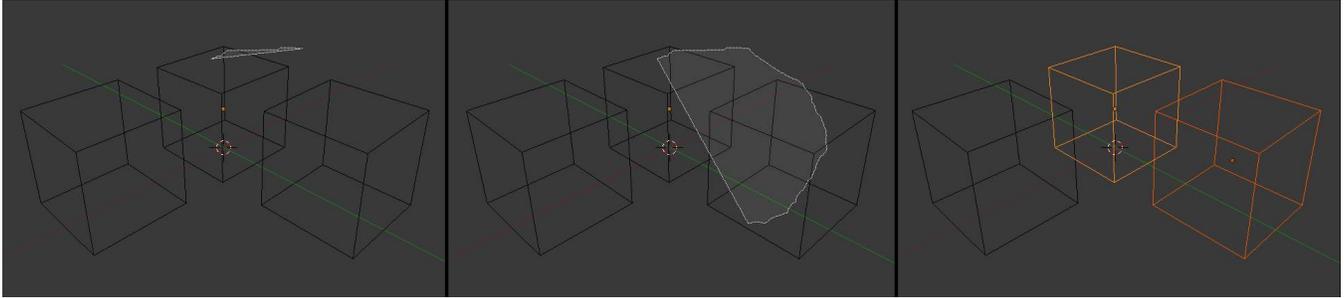


Fig. 2.50: Lasso selection example

Hotkey: C

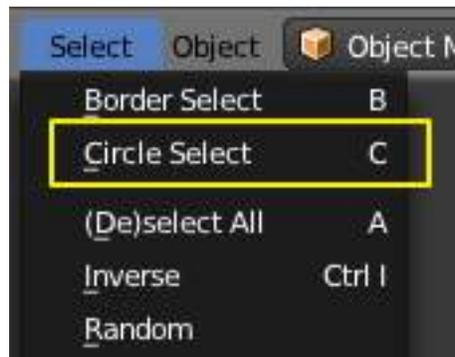


Fig. 2.51: Main selection menu

**Description** *Circle Select* is used by moving with dotted circle through objects with LMB. You can select any object by touching of circle area. It is possible to dynamically change the diameter of circle by scrolling MMB as seen in pictures below. Deselection is under the same principle - MMB. To cancel the selection use RMB or key `ESC`,



### Menu Selection

The selection methods described above are the most common. There are also many more options accessible through the *Select* menu of the 3D view.

Each is more adapted to certain operations.

### Select Grouped

#### Reference

Mode: *Object mode*

Menu: *Select* → *Grouped*

Hotkey: `Shift-G`

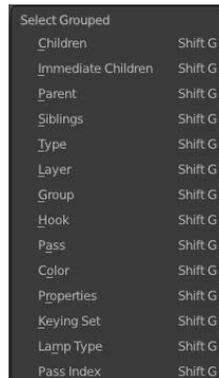


Fig. 2.56: Select Grouped menu

**Description** There are two ways to organize the objects in relation to one another. The first one is *parenting*, and the second is simple *grouping*. We can use these relationships to our advantage by selecting members of respective families or groups.

**Options** *Select → Grouped* in *Object mode* uses the active object as a basis to select all others.

Available options are:

**Children** Selects all children of the active object recursively.

**Immediate Children** Selects all direct children of the active object.

**Parent** Selects the parent of this object if it has one.

**Siblings** Select objects that have the same parent as the active object. This can also be used to select all root level objects (objects with no parents).

**Type** Select objects that are the same type as the active one.

**Layer** Objects that have at least one shared layer.

**Group** Objects that are part of a group (rendered green with the default theme) will be selected if they are in one of the groups that the active object is in.

**Object Hooks** Every hook that belongs to the active object.

**Pass** Select objects assigned to the same render pass. Render passes are set in *Properties → Object → Relations* and can be used in the *Node Compositor (Add → Convertor → ID Mask.)*

**Color** Select objects with same *Object Color*. Object colors are set in *Properties → Object → Display → Object Color.*)

**Properties** Select objects with same *Game Engine Properties*.

**Keying Set** Select objects included in active Keying Set.

**Lamp Type** Select matching lamp types.

**Pass Index** Select matching object pass index.

## Select linked Reference

Mode: *Object mode*

Menu: *Select → Linked*

Hotkey: Shift-L

---

**Description** Selects all objects which share a common datablock with the active object.

**Options** *Select → Linked* in *Object mode* uses the active object as a basis to select all others.

Available options are:

**Object Data** Selects every object that is linked to the same Object Data, i.e. the datablock that specifies the type (mesh, curve, etc.) and the build (constitutive elements like vertices, control vertices, and where they are in space) of the object.

**Material** Selects every object that is linked to the same material datablock.

**Texture** Selects every object that is linked to the same texture datablock.

**Dupligroup** Selects all objects that use the same **Group** for duplication.

**Particle System** Selects all objects that use the same **Particle System**

**Library** Selects all objects that are in the same **Library Library (Object Data)**

---

**Select All by Type**

**Reference**

Mode: *Object mode*

Menu: *Select → Select All by Type*

Hotkey: None

---

**Description** The types are *Mesh, Curve, Surface, Meta, Font, Armature, Lattice, Empty, Camera, Lamp, Speaker*.

With this tool it becomes possible to select every **visible** object of a certain type in one go.

**Options** *Select All by Type* in *Object mode* offers an option for every type of object that can be described by the *ObData* datablock.

Just take your pick.

---

**Select All by Layer**

**Reference**

Mode: *Object mode*

Menu: *Select → Select All by Layer*

Hotkey: None

---

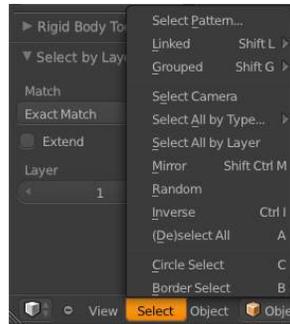


Fig. 2.57: All by Layer selection menu

**Description** Layers are another means to regroup your objects to suit your purpose.

This option allows the selection of every single object that belongs to a given layer, visible or not, in one single command.

**Options** In the *Tool Shelf* → *Select by Layer* the following options are available:

**Match** The match type for selection.

**Extend** Enable to add objects to current selection rather than replacing the current selection.

**Layer** The layer on which the objects are.

---

**Tip:** Selection of Objects

Rather than using the *Select All by Layer* option, it might be more efficient to make the needed layers visible and use A on them. This method also allows objects to be deselected.

---

**Other Menu Options** Available options on the first level of the menu are:

**Select Pattern...** Selects all objects whose name matches a given pattern. Supported wildcards: \* matches everything, ? matches any single character, [abc] matches characters in “abc”, and [!abc] match any character not in “abc”. The matching can be chosen to be case sensitive or not. As an example *house* matches any name that contains “house”, while *floor\** matches any name starting with “floor”.

**Select Camera** Select the active camera.

**Mirror (Shift-Ctrl-M)** Select the Mirror objects of the selected object eg. L.sword → R.sword.

**Random** Randomly selects unselected objects based on percentage probability on currently active layers. On selecting the command a numerical selection box becomes available in the *Tool Shelf*. It’s important to note that the percentage represents the likelihood of an unselected object being selected and not the percentage amount of objects that will be selected.

**Inverse (Ctrl-I)** Selects all objects that were not selected while deselecting all those which were.

**(De)select All (A)** If anything was selected it is first deselected. Otherwise it toggles between selecting and deselecting every visible object.

## Introduction

In this section will be described tools for editing objects in *Object Mode*.

Information about some additional possibilities are described in [Manipulation in 3D](#).

## Object Mode



Fig. 2.58: Object Mode button

By default new files opens with enabled *Object Mode*. To enable it you may in *3D View window* → *Header* click *Object Mode button* (see picture *Object Mode button*)

All edition tools works only with selected objects. See [Selecting Objects](#) for more information.

All commands described below can be found in *Object* menu and/or in *Object tools* panel.

**Creation and deletion** The most basic edition includes manipulation with existence of objects. Below are listed different types of creation and deletion tools.

### Add

---

### Reference

Menu: *Main* → *Add*

Hotkey: *Shift-A*

---

We may add one of those primitives:

**Mesh** Plane, Cube, Circle, UV Sphere, Icosphere, Cylinder, Cone, Grid, Monkey, Torus.

**Curve** Bezier, Circle, NURBS Curve, NURBS Circle, Path.

**Surface** NURBS Curve, NURBS Circle, NURBS Surface, NURBS Cylinder, NURBS Sphere, NURBS Torus.

**Metaball** Ball, Capsule, Plane, Ellipsoid, Cube.

**Text** TODO

**Armature** Single Bone.

**Lattice** TODO

**Empty** Plane Axis, Arrows, Single Arrow, Circle, Cube, Sphere, Cone, Image.

**Speaker** TODO

**Camera** TODO

**Lamp** Point, Sun, Spot, Hemi, Area.

**Force Field** Force, Wind, Vortex, Magnetic, Harmonic, Charge, Lennard-Jones, Texture, Curve Guide, Boid, Turbulence, Drag, Smoke Flow.

**Group Instance** (user defined groups of objects).

## Duplicate

---

### Reference

Menu: *Object* → *Duplicate Objects*

Hotkey: Shift-D

---

### Reference

Menu: *Object* → *Duplicate Linked*

Hotkey: Alt-D

---

Duplication makes exact copy of objects. May be linkage of some attributes depending on specific tool. See [Duplication](#) for more information.

## Join

---

### Reference

Mode: *Object* mode

Menu: *Object* → *Join*

Hotkey: Ctrl-J

---

Joining makes one single object from all selected objects. Objects must be of the same type. Origin point is obtained from the previously *active* object. Performing a join is equivalent to adding new objects while in *Edit mode*. The non-active objects are deleted and their meshes added to the active object, so that only the active object remains. This only works with editable objects containing meshes and curves.

## Delete / Erase

---

### Reference

Mode: *Edit* or *Object* mode

Menu: *Object* → *Delete*

Hotkey: X or Delete

---

Erases or deletes selected objects.

---

**Transformation tools** Objects can be transformed in a variety of ways. Below are listed different types of transformation.

### Translate

---

#### Reference

Menu: *Object -> Transform -> Grab/Move*

Hotkey: G

---

Translation means changing location of objects. This changes X, Y and/or Z coordinates of object's *Origin point* relative to center of coordinates.

### Rotate

---

#### Reference

Menu: *Object -> Transform -> Rotate*

Hotkey: R

---

Rotation means changing angles of objects. This changes rotation angles around X, Y and/or Z axes of object's coordinate system relative to current coordinate system. No parts of each object are changing their position relative to other parts of the same object.

### Scale

---

#### Reference

Menu: *Object -> Transform -> Scale*

Hotkey: S

---

Scaling means changing proportions of objects. This proportionally stretches object along X, Y and/or Z axes of object's coordinate system.

## Grouping And Parenting Objects

There can be many objects in a scene: A typical stage scene consists of furniture, props, lights, and backdrops. Blender helps you keep everything organized by allowing you to group like objects together.

When modeling a complex object, such as a watch, you may choose to model the different parts as separate objects. However, all of the parts may be attached to each other. In these cases, you want to designate one object as the parent of all the children. Movement, rotation or scaling of the parent also affects the children.

## Parenting objects

To parent objects, select at least two objects (select the *Child Objects* first, and select the *Parent Object* last), and press `Ctrl-P`. The *Set Parent To* dialog will pop up allowing you to select from one of several possible different parenting types. Selecting one of the entries in *Set Parent To* confirms, and the child/children to parent relationship is created.

The last object selected will be the *Active Object* (outlined in light orange), and will also be the *Parent Object*. If you selected multiple objects before selecting the parent, they will all be children of the parent and will be at the same level of the hierarchy (they are “siblings”).

The *Set Parent To* pop-up dialog is context sensitive, which means the number of entries it displays can change depending on what objects are selected when the `Ctrl-P` shortcut is used.

For non-inverse-mode, press `Shift-Ctrl-P` instead. This creates an alternative parent-child-relationship where child-objects exist entirely in the parent’s coordinate system. This is the better choice for CAD purposes, for example.

Moving, rotating or scaling the parent will also usually move/rotate/scale the child/children. However moving/rotating/scaling the child/children of the parent will not result in the parent moving/rotating/scaling. In other words, the direction of influence is from parent to child and not child to parent.

In general when using the `Ctrl-P` or [3D View Editor Header > Object Menu > Parent Menu] entires to parent objects, the *Child Objects* can only have one *Parent Object*. If a *Child Object* already has a *Parent Object* and you give it another parent then Blender will automatically remove the previous parent relationship.

Blender supports many different types of parenting, listed below:

- Object
- Armature Deform
- Bone
- Curve Deform
- Path Constraint
- Lattice Deform
- Vertex
- Vertex (Triangle)

**Object Parent** *Object Parent* is the most general form of parenting that Blender supports. It will take selected objects and make the last selected object the *Parent Object*, while all other selected objects will be *Child Objects*.

**Object (Keep Transform) Parent** *Object (Keep Transform) Parent* works in a very similar way to *Object Parent* the major difference is in whether the *Child Objects* will remember any previous transformations applied to them from the previous *Parent Object*.

Since explaining this in an easy to understand technical way is hard, lets instead use an example to demonstrate.

Assume that we have a scene consisting of 3 objects, those being 2 Empty Objects named “EmptyA” and “EmptyB”, and a Monkey object. See figure 1.

Figure 1 shows the 3 objects with no parenting relationships active on them.

If you select the Monkey object by RMB click and then `Shift-RMB` click “EmptyA” object and press `Ctrl-P` and then select *Object* from the *Set Parent To* pop-up dialog box. This will result in “EmptyA” object being the *Parent Object* of the Monkey object. With only “EmptyA” selected rotating/scaling/moving it will result in the Monkey object being altered respectively.

Scale the “EmptyA” object, so that the Monkey becomes smaller and moves to the left a little. See figure 2.



Fig. 2.59: Figure 1 - Scene with 2 Empties and a Monkey, no parenting currently active.



Fig. 2.60: Figure 2 - Scene with Monkey object being the Child Object of “EmptyA”. “EmptyA” has been scaled resulting in the Monkey also being scaled and moved to the left.

If you select only the Monkey object by RMB click and then Shift-RMB click “EmptyB” object and press Ctrl-P and select *Object* from the *Set Parent To* pop-up dialog box. This will result in “EmptyB” object being the *Parent Object* of the Monkey object. Notice that when you change the parent of the Monkey the scale of the Monkey changed. See figure 3.



Fig. 2.61: Figure 3 - Scene with Monkey object having its parent changed from “EmptyA” to “EmptyB” and the resulting change in scale.

This happens because the Monkey object never had its scale altered directly, the change came about because it was the child of “EmptyA” which had its scale altered. Changing the Monkey’s parent to “EmptyB” resulted in those indirect changes in scale being removed, because “EmptyB” has not had its scale altered.

This is often the required behaviour, but it is also sometimes useful that if you change your *Parent Object* that the *Child Object* keep any previous transformations it got from the old *Parent Object*; If instead when changing the *Parent Object* of the Monkey from “EmptyA” to “EmptyB” we had chosen parenting type *Object (Keep Transform)*, the Monkey would keep its scale information it obtained from the old parent “EmptyA” when it is assigned to the new parent “EmptyB”; See Figure 4.

If you want to follow along with the above description here is the blend file used to describe *Object (Keep Transform)* parenting

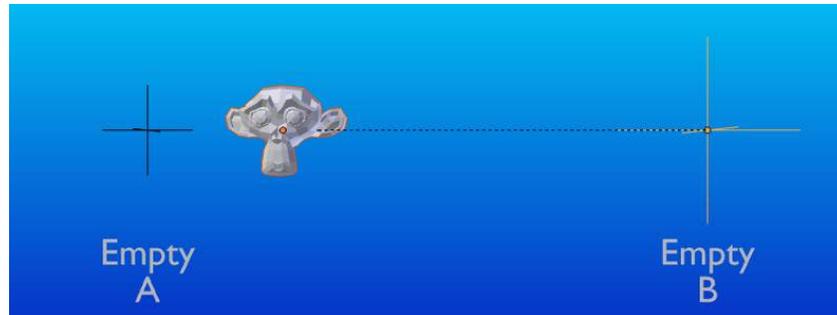


Fig. 2.62: Figure 4 - Scene with Monkey object having its a parent changed from “EmptyA” to “EmptyB” using ‘Object (Keep Transform)’ parent method.

method:

File:Parent\_-\_Object\_(Keep\_Transform)\_(Demo\_File).blend

**Armature Deform Parent** An Armature in Blender can be thought of as similar to the armature of a real skeleton, and just like a real skeleton an Armature can consist of many bones. These bones can be moved around and anything that they are attached to or associated with will move and deform in a similar way.

In Blender Armature Object Types are usually used to associate certain bones of an Armature to certain parts of a Mesh Object Types Mesh Geometry. You are then able to move the Armature Bones and the Mesh Object will deform. See figure 5.

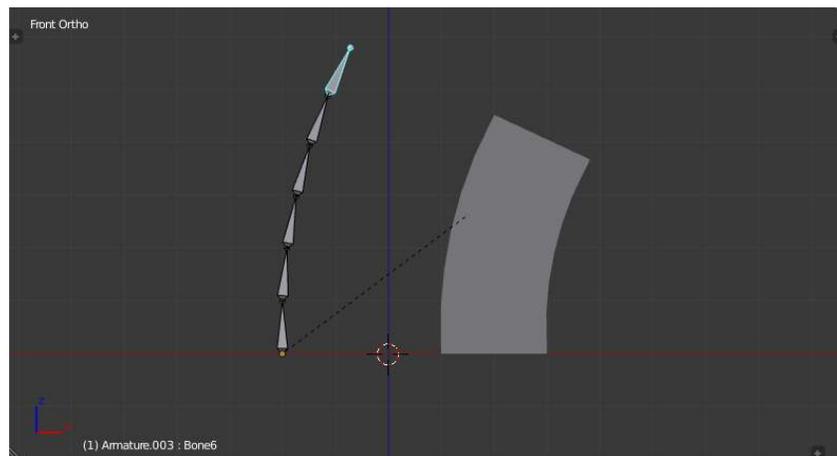


Fig. 2.63: Figure 5 - Armature Object Bone associated with another Mesh Object, as the bone move the Mesh deforms similarly.

Armature Deform Parenting is one of the most flexible ways of associating Bones in an Armature to another Object, it gives a lot of freedom but that comes at the price of a little complexity, as there are multiple steps involved in setting up Armature Deform Parenting such that deformations are actually carried out.

Blender has several different ways of Parenting an Armature to an object, most of them can automate several of the steps involved, but all of them ultimately do all the steps we describe for Armature Deform Parenting.

Using the Armature Deform Parenting operator is the first step in setting up the relationship between an Armature Object and it’s Child Objects.

To use Armature Deform Parenting you must first select all the Child Objects that will be influenced by the Armature and then lastly select the Armature Object itself. Once all the Child Objects and the Armature Object are selected press `Ctrl-P` and

select Armature Deform in the Set Parent To pop-up dialog. See figure 6.

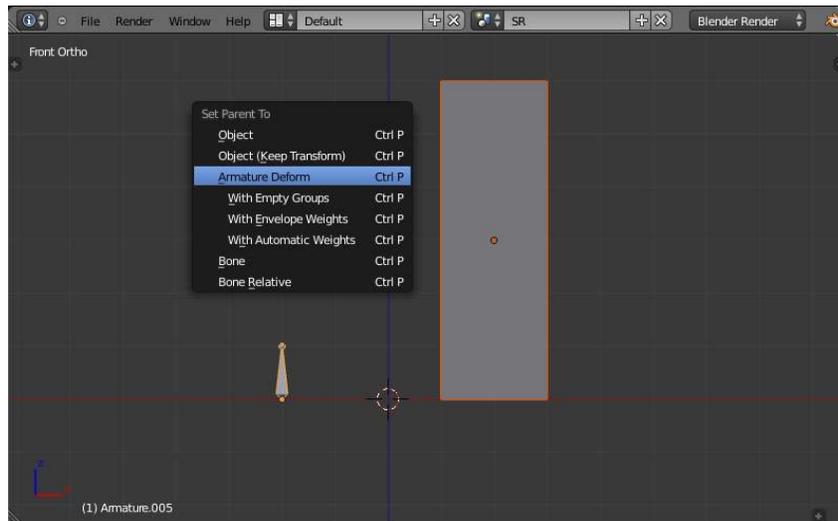


Fig. 2.64: Figure 6 - Set Parent To dialog with Armature Deform Parenting option highlighted.

Once this is done the Armature Object will be the Parent Object of all the other Child Objects, also we have informed Blender that the Bones of the Armature Object can be associated with specific parts of the Child Objects so that they can be directly manipulated by the Bones.

At this point however all Blender knows is that the Bones of the Armature could be used to alter the Child Objects, we haven't yet told Blender which Bones can alter which Child Objects or by how much.

To do that we must individually select each Child Object individually and toggle into Edit Mode on that Child Object. Once in Edit Mode we can then select the vertices we want to be influenced by the Bones in the Armature. Then with the vertices still selected navigate to [Properties Editor > Object Data Context > Vertex Groups Panel] and create a new Vertex Group with the same name as the Bone that you want the selected vertices to be influenced by.

Once the Vertex Group has been created we then assign the selected vertices to the Vertex Group by clicking the Assign Button. By default when selected vertices are assigned to a Vertex Group they will have an Influence Weight of 1.0 This means that they are fully influenced when a Bone they are associated with is moved, if the Influence Weight had been 0.5 then when the bone moves the vertices would only move half as much. See figure 7.

Once all these steps have been carried out, the Bones of the Armature Object should be associated with the Vertex Groups with the same names as the Bones. You can then select the Armature Object and switch to Pose Mode in the [3D View Editor Header > Mode Select Button]. See figure 8.

Once in Pose Mode transforming one of the Bones of the Armature that has been associated with vertices of an object will result in those vertices also being transformed.

**Armature Deform Parent With Empty Groups** The Armature Deform With Empty Groups parenting method works in almost the same way as Armature Deform parenting with one difference. That difference is that when you parent a Child Object to an Armature Object the names of the bones in the armature are copied to the Child Objects in the form of newly created Vertex Groups, one for each different deforming armature bone name. The newly created Vertex Groups will be empty this means they will not have any vertices assigned to those Vertex Groups. You still must manually select the vertices and assign them to a particular Vertex Group of your choosing to have bones in the armature influence them.

For example if you have an Armature Object which consists of 3 bones named BoneA, BoneB and BoneC and Cube Mesh Object type called Cube. If you parent the Cube Child Object to the Armature Parent Object the Cube will get 3 new Vertex Groups created on it called BoneA, BoneB and BoneC. Notice that each Vertex Group is empty. See figure 21.

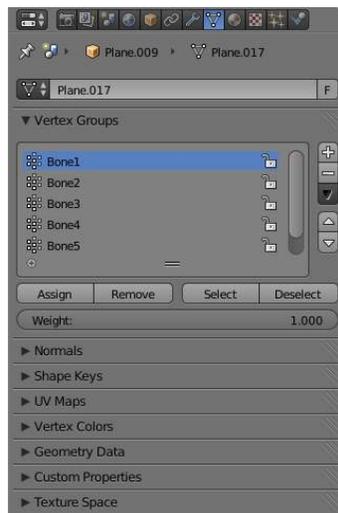


Fig. 2.65: Figure 7 - Properties Editor > Object Data Context > Vertex Groups Panel with Assign Button and influence Weight Slider highlighted.

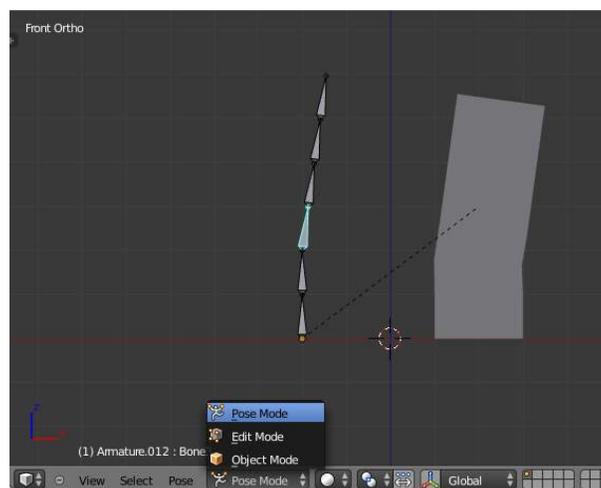


Fig. 2.66: Figure 8 - 3D View Editor Header > Mode Select Button] set to Pose Mode, with Armature Bone highlighted in Cyan and effecting the Mesh Object

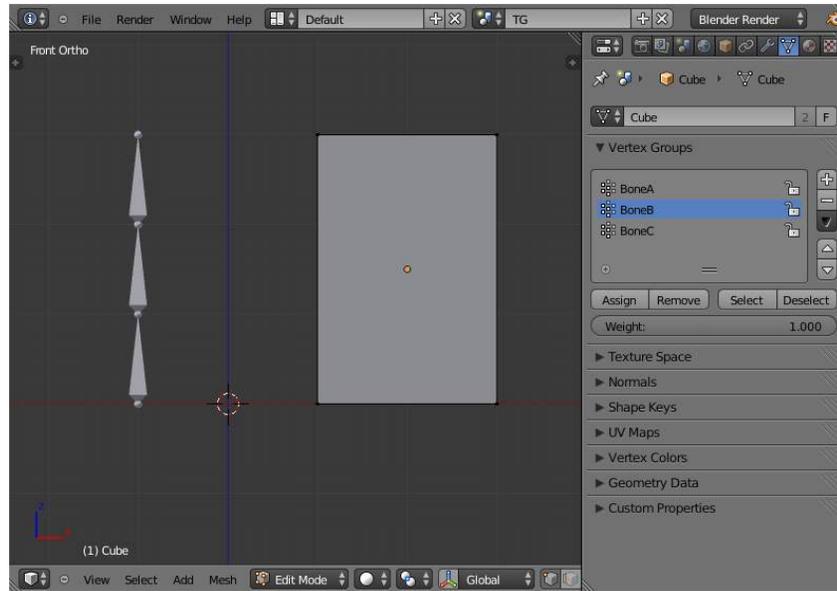


Fig. 2.67: Figure 21 - Cube in Edit Mode showing the 3 created Vertex Groups after it was parented using Armature Deform With Empty Groups to an Armature with 3 Bones named BoneA, BoneB and BoneC with the Vertex Group Panel shown. All the Vertex Groups are empty.

Bones in an Armature can be generally classified into 2 different types:

- Deforming Bones
- Control Bones

**Deforming Bones** - Are bones which when transformed will result in vertices associated with them also transforming in a similar way. Deforming Bones are directly involved in altering the positions of vertices associated with their bones.

**Control Bones** - Are Bones which act in a similar way to switches, in that they control how other bones or objects react when they are transformed. A Control Bone could for example act as a sliding switch control, when the bone is in one position to the left it could indicate to other bones that they react in a particular way when transformed, when the Control Bone is positioned to the right, transforming other bones or objects could do something completely different. Control Bones are not directly used to alter the positions of vertices, in fact Control Bones often have no vertices directly associated with themselves.

When using the Armature Deform With Empty Groups parenting method Vertex Groups on the Child Object will only be created for Armature Bones which are setup as Deforming Bone types. If a Bone is a Control Bone no Vertex Group will be created on the Child Object for that bone.

To check weather a particular bone in an armature is a Deforming Bone simply switch to Pose Mode or Edit Mode on the armature and select the bone you are interested in by RMB it. Once the bone of interest is selected navigate to [Properties Editor > Bone Context > Deform Panel] and check if the Deform tickable option is ticked or not. If it is the selected bone is a Deforming Bone, otherwise it is a Control Bone. See figure 22.

**Armature Deform With Automatic Weights** Armature Deform With Automatic Weights parenting feature does everything Armature Deform With Empty Groups does with one extra thing. That extra thing is that unlike Armature Deform With Empty Groups which leaves the automatically created Vertex Groups empty with no vertices assigned to them; Armature Deform With Automatic Weight will try to calculate how much Influence Weight a particular Armature Bone would have on a certain collection of vertices based on the distance from those vertices to a particular Armature Bone.

Once Blender has calculated the Influence Weight vertices should have it will assign that Influence Weight to the Vertex Groups that were previously created automatically by Blender on the Child Object when Armature Deform With Automatic Weights parenting command was carried out.

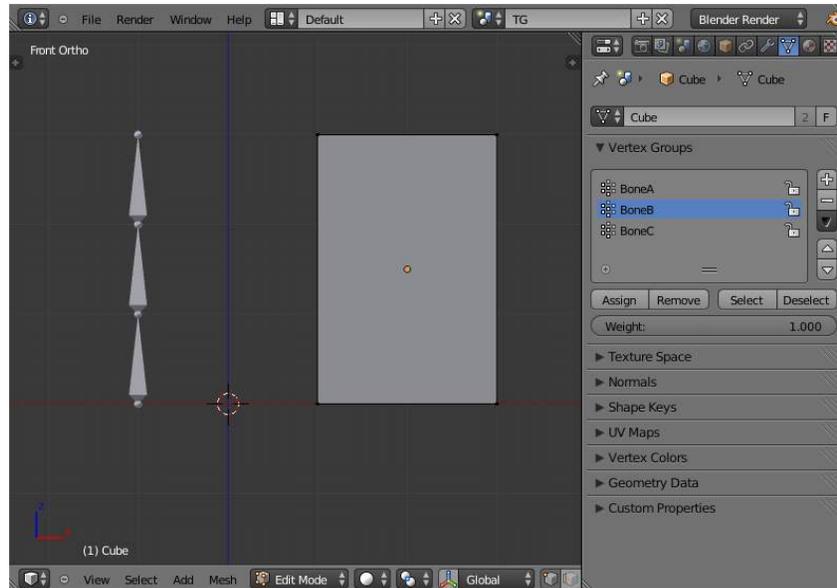


Fig. 2.68: Figure 22 - 3 Bone Armature in Edit Mode with 2nd bone selected with [Properties Editor > Bone Context > Deform Panel] displayed an ticked, indicating the bone is a Deforming Bone.

If all went well it should be possible to select the Armature Object switch it into Pose Mode and transform the bones of the Armature and the Child Object should deform in response. Unlike Armature Deform parenting you won't have to create Vertex Groups on the Child Object, neither will you have to assign Influence Weights to those Vertex Groups, Blender will try to do it for you.

To activate Armature Deform With Automatic Weights you must be in Object Mode or Pose Mode, then select all the Child Objects (usually Mesh Object Types) and lastly select the Armature Object; Once done press `Ctrl-P` and select the Armature Deform With Automatic Weights from the Set Parent To pop-up dialog.

This method of parenting is certainly easier setup but it can often lead to Armatures which do not deform Child Objects in ways you would want, as Blender can get a little confused when it comes to determining which Bones should influence certain vertices when calculating Influence Weights for more complex armatures and Child Objects. Symptoms of this confusion are that when transforming the Armature Object in Pose Mode parts of the Child Objects don't deform as you expect; If Blender does not give you the results you require you will have to manually alter the Influence Weights of vertices in relation to the Vertex Groups they belong to and have influence in.

**Armature Deform With Envelope Weights** Works in a similar way to Armature Deform With Automatic Weights in that it will create Vertex Groups on the Child Objects that have names matching those of the Parent Object Armature Bones. The created Vertex Groups will then be assigned Influence Weights. The major difference is in the way those Influence Weights are calculated.

Influence Weights that are calculated when using Armature Deform With Envelope Weights parenting are calculated entirely visually using Bone Envelopes. See figure 28.

Figure 28 shows a single Armature Bone in Edit Mode with Envelope Weight activated. The gray semi-transparent volume around the bone is the Bone Envelope.

Any Child Object that has vertices inside the volume of the Bone Envelope will be influenced by the Parent Object Armature when the Armature Deform With Envelope Weights operator is used. Any vertices outside the Bone Envelope volume will not be influenced. See figure 29.

The default size of the Bone Envelope volume does not extend very far from the surface of a bone; You can alter the size of the Bone Envelope volume by clicking on the body of the bone you want to alter, switch to Edit Mode or Pose Mode and then

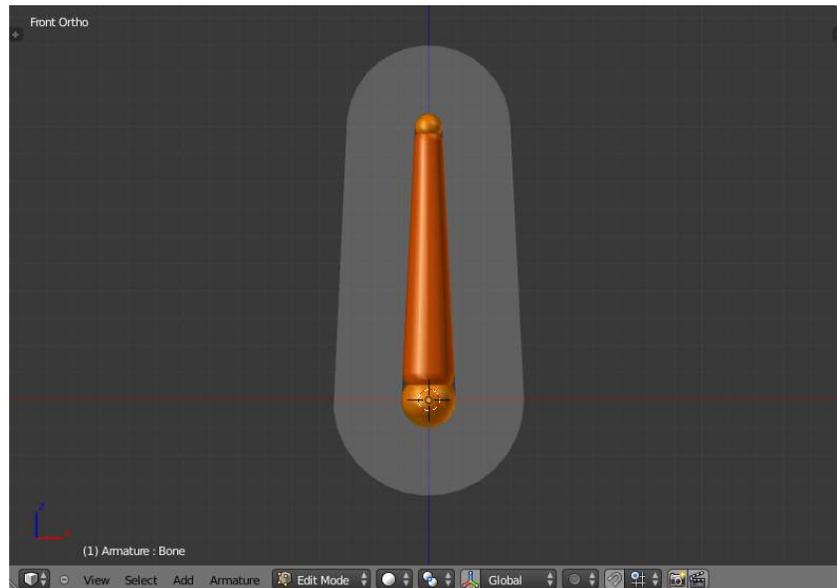


Fig. 2.69: Figure 28 - Single Armature Bone in Edit Mode with Envelope Weight display enabled. The gray volume around the bone is the Bone Envelope.

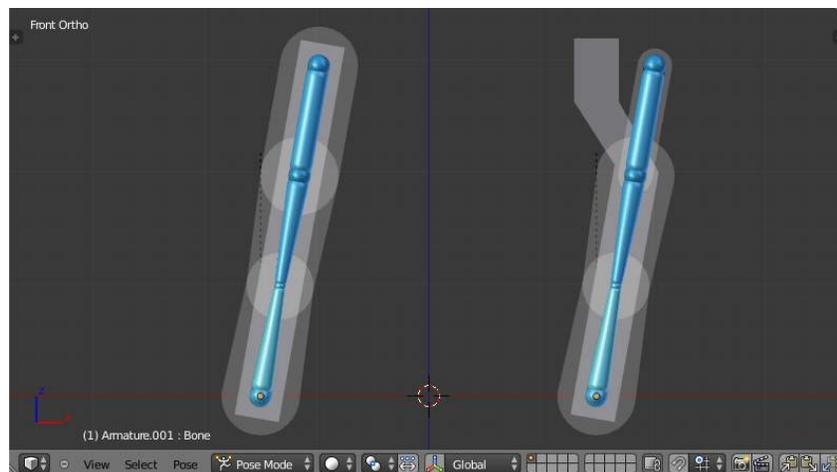


Fig. 2.70: Figure 29 - 2 sets of Armatures each with 3 bones, the first set has all vertices inside the Bone Envelope, the second did not. When the bones are transformed in Pose Mode the results are very different.

pressing `Ctrl-Alt-S` then drag your mouse left or right and the Bone Envelope volume will alter accordingly. See figure 30.

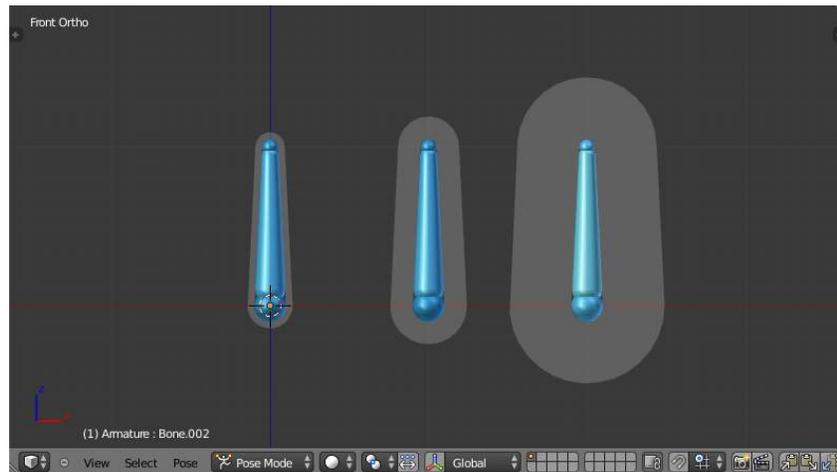


Fig. 2.71: Figure 30 - Single Armature Bone with various different Bone Envelope sizes.

You can also alter the Bone Envelope volume by selecting the Bone you wish to alter and switching to Edit Mode or Pose Mode, then navigate to [Properties Editor > Bone Context > Deform Panel > Envelope Section > Distance field] then enter a new value into it. See figure 31.

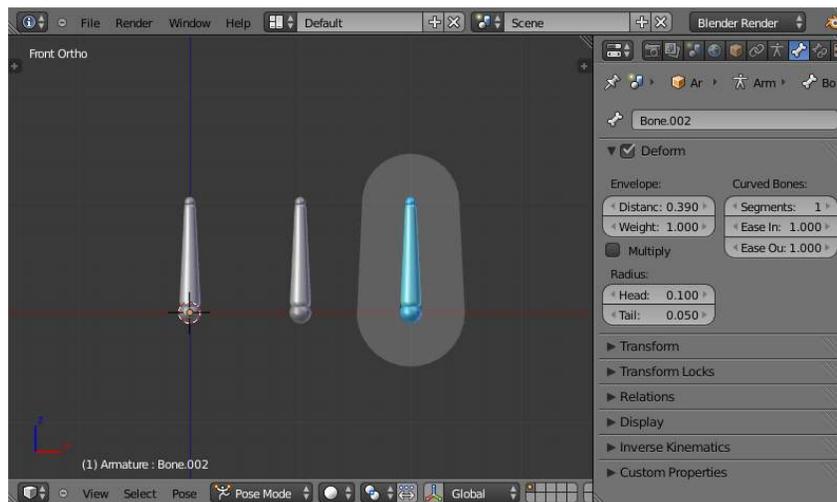


Fig. 2.72: Figure 31 - [Properties Editor > Bone Context > Deform Panel > Envelope Section > Distance field] highlighted.

Altering the Bone Envelope volume does not alter the size of the Armature Bone just the range within which it can influence vertices of Child Objects.

You can alter the radius that a bone has by selecting the head, body or tail parts of a bone while in Edit Mode, and then press `Alt-S` and move the mouse left or right. This will make the selected bone fatter or thinner without altering the thickness of the Bone Envelope volume. See figure 32.

You can also alter the bone radius by selecting the tail or head of the bone you wish to alter and switching to Edit Mode, then navigate to [Properties Editor > Bone Context > Deform Panel > Radius Section] and entering new values for the Tail and Head fields. See figure 33.

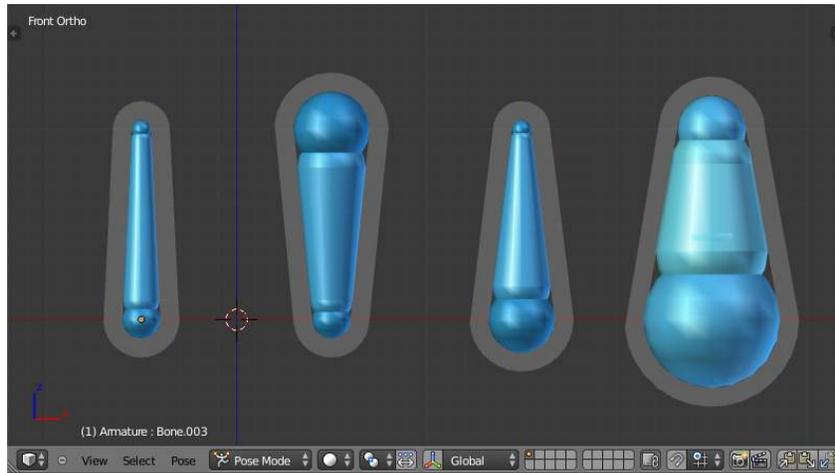


Fig. 2.73: Figure 32 - 4 Armature Bones all using Envelope Weight. The 1st with a default radius value, the 3 others with differing Tail, Head and Body radius values.

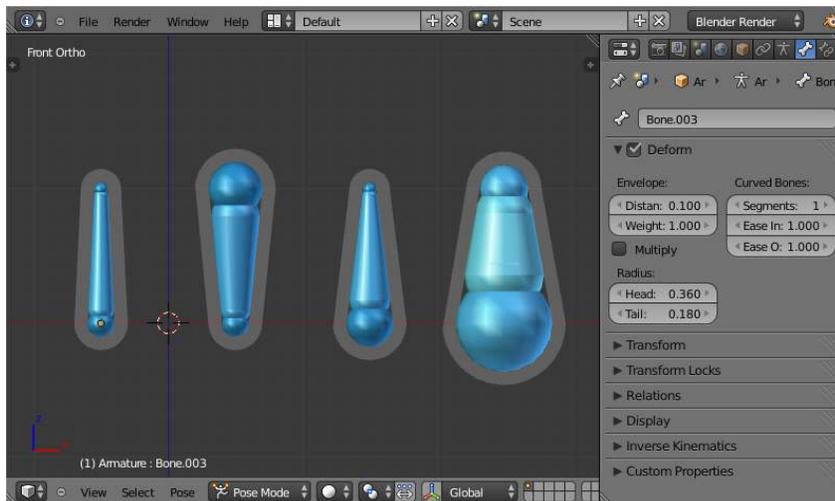


Fig. 2.74: Figure 33 - [Properties Editor > Bone Context > Deform Panel > Radius Section] head and tail fields highlighted.

**Note:** If you alter the Bone Envelope volume of a bone so that you can have it include/exclude certain vertices after you have already used Armature Deform With Envelope Weights, by default the newly included/excluded vertices won't be effected by the change. When using Armature Deform With Envelope Weights it only calculates which vertices will be affected by the Bone Envelope volume at the time of parenting, at which point it creates the required named Vertex Groups and assigns vertices to them as required. If you want any vertices to take account of the new Bone Envelope volume size you will have carry out the Armature Deform With Envelope Weights parenting again; In fact all parenting used in the Set Parent To pop-up dialog which tries to automatically assign vertices to Vertex Groups works like this.

**Bone Parent** Bone parenting allows you to make a certain bone in an armature the Parent Object of another object. This means that when transforming an armature the Child Object will only move if the specific bone it is the Child Object of moves. See figure 34.

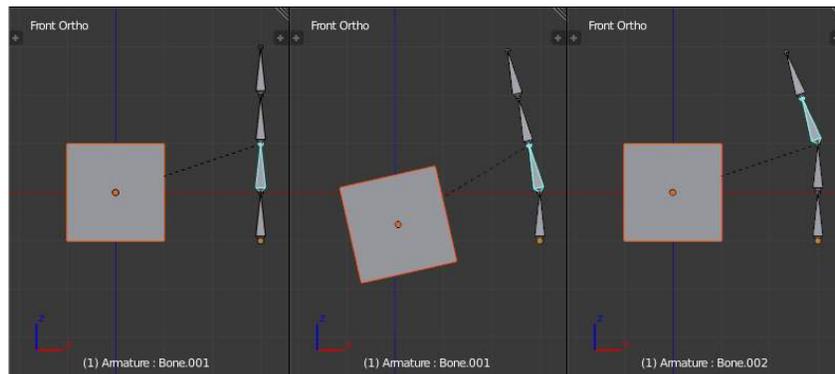


Fig. 2.75: Figure 34 - 3 pictures of Armatures with 4 Bones, with the 2nd bone being the Bone Parent of the Child Object Cube. The Cube is only transformed if the 1st or 2nd bones are. Notice altering the 3rd and 4th bones has no effect on the Cone.

To use Bone Parenting, you must first select all the Child Objects you wish to parent to a specific Armature Bone, then `Shift-RMB` select the Armature Object and switch it into Pose Mode and then select the specific bone you wish to be the Parent Bone by `RMB` selecting it. Once done press `Ctrl-P` and select Bone from the Set Parent To pop-up dialog.

Now transforming that bone in Pose Mode will result in the Child Objects also transforming.

**Bone Relative Parenting** Bone Relative parenting works in the same way as Bone parenting with one difference.

With Bone parenting if you have parented a bone to some Child Objects and you select that bone and switch it into Edit Mode and then translate that bone; When you switch back into Pose Mode on that bone, the Child Object which is parented to that bone will snap back to the location of the bone in Pose Mode. See figure 37.

Bone Relative parenting works differently; If you move a Parent Bone in Edit Mode, when you switch back to Pose Mode, the Child Objects will not move to the new location of the Pose Bone. See figure 38.

**Vertex Parent** You can parent an object to a single vertex or a group of three vertices as well; that way the child/children will move when the parent mesh is deformed, like a mosquito on a pulsing artery.

**Vertex Parent from Edit Mode** In *Object* mode, select the child/children and then the parent object. Tab into *Edit mode* and on the parent object select either one vertex that defines a single point, or select three vertices that define an area (the three vertices do not have to form a complete face; they can be any three vertices of the parent object), and then press `Ctrl-P` and confirm.

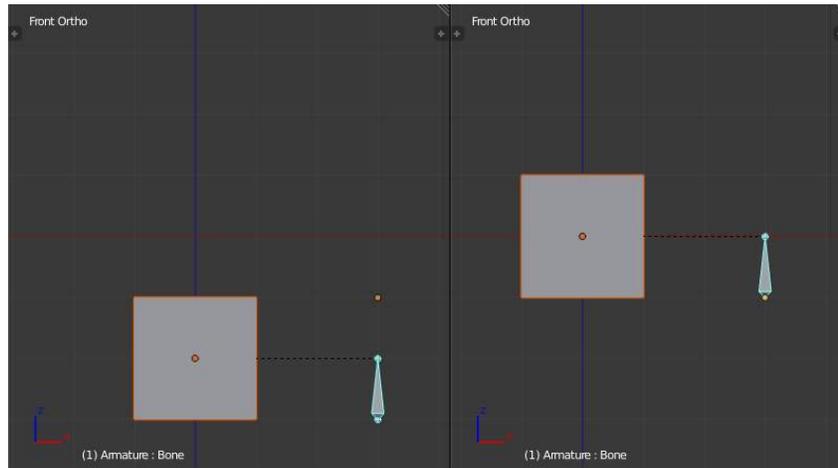


Fig. 2.76: Figure 37 - [Single Armature Bone which has a Child Object cube parented to it using Bone parenting. 1st picture shows the position of the cube and armature before the bone is moved in Edit Mode. 2nd picture shows the position of the cube and armature after the bone was selected in Edit Mode, moved and switched back into Pose Mode. Notice that the Child Object moves to the new location of the Pose Bone.

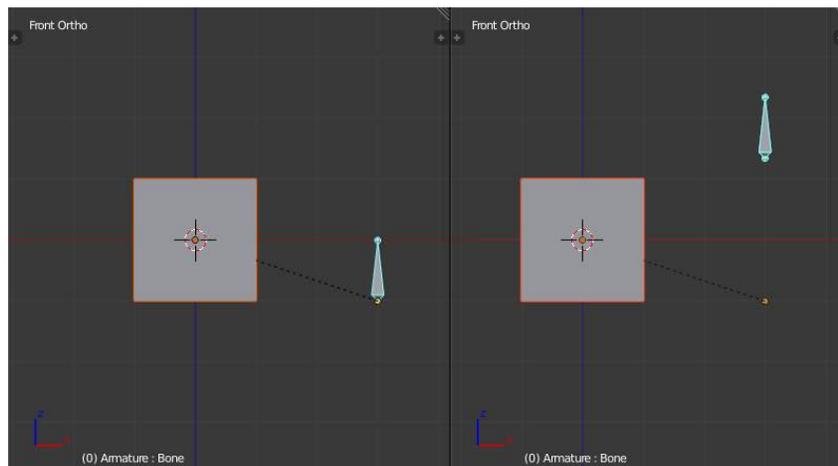


Fig. 2.77: Figure 38 - [Single Armature Bone which has a Child Object cube parented to it using Bone Relative parenting. 1st picture shows the position of the cube and armature before the bone is moved in Edit Mode. 2nd picture shows the position of the cube and armature after the bone was selected in Edit Mode, moved and switched back into Pose Mode. Notice that the Child Object does not move to the new location of the Pose Bone.

At this point, if a single vertex was selected, a relationship/parenting line will be drawn from the vertex to the child/children. If three vertices were selected then a relationship/parenting line is drawn from the averaged center of the three points (of the parent object) to the child/children. Now, as the parent mesh deforms and the chosen parent vertex/vertices move, the child/children will move as well.

**Vertex Parent from Object Mode** Vertex parenting can be performed from object mode, This is done like regular object parenting, Press `Ctrl-P` in object mode and select *Vertex* or *Vertex (Triangle)*.

The nearest vertices will be used from each object which is typically what you would want.

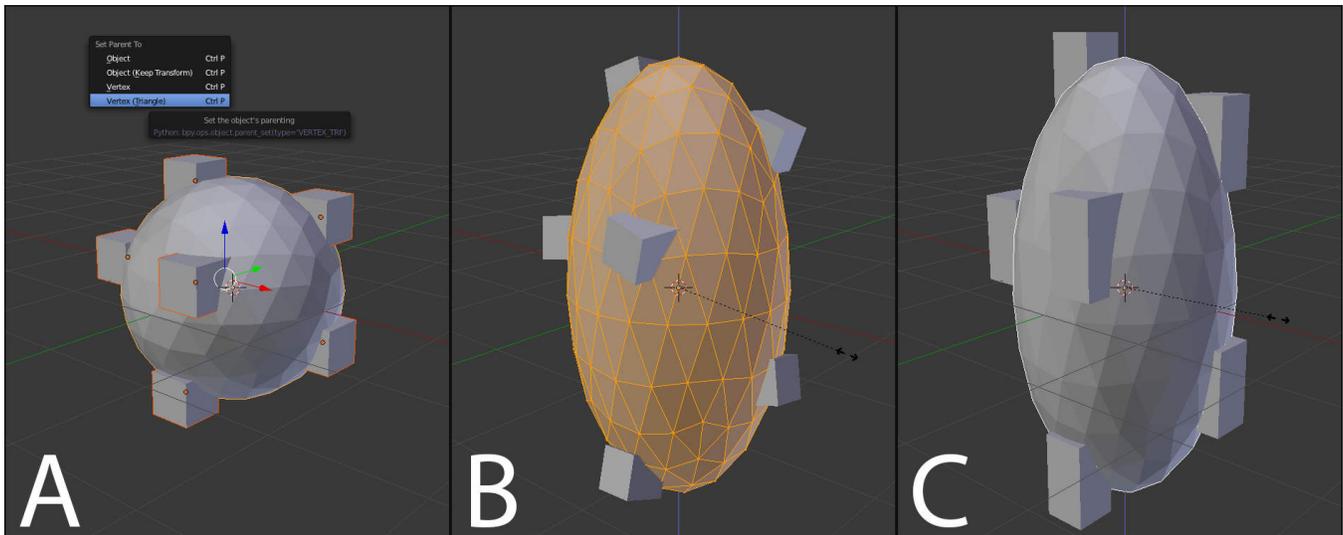


Fig. 2.78: See:

1. The small cubes can each be automatically parented to a triad of nearby vertices on the icosphere using the “Vertex (Triangle)” in the set parent context menu.
2. Reshaping the object in edit mode then means each of the cubes follows their vertex triad parent separately.
3. Re-scaling the parent icosphere in object mode means the child cubes are also rescaled as expected.

The parent context menu item means users can rapidly set up a large number of vertex parent relationships, and avoid the tedious effort of establishing each parent-child vertex relationship separately.

---

**Note:** It is in fact a sort of “reversed” [hook](#)

---

## Options

**Move child** You can *move* a child object to its parent by clearing its origin. The relationship between the parent and child isn’t affected. Select the child object and press `Alt-O`. By confirming the dialog the child object will snap to the parent’s location. Use the *Outliner* view to verify that the child object is still parented.

**Remove relationship/Clear Parent** You can *remove* a parent-child relationship via `Alt-P`

The menu contains:

**Clear Parent** If the parent in the group is selected nothing is done. If a child or children are selected they are disassociated from the parent, or freed, and they return to their *original* location, rotation, and size.

**Clear and Keep Transformation** Frees the children from the parent, and *keeps* the location, rotation, and size given to them by the parent.

**Clear Parent Inverse** Places the children with respect to the parent as if they were placed in the Global reference. This effectively clears the parent’s transformation from the children. For example, if the parent is moved 10 units along the X axis and *Clear Parent Inverse* is invoked, any selected children are freed and moved -10 units back along the X axis. The “Inverse” only uses the last transformation; if the parent moved twice, 10 units each time for a total of 20 units, then the “Inverse” will only move the child back 10 units, not 20.



Fig. 2.79: Outliner view

**Hints** There is another way to see the parent-child relationship in groups and that is to use the *Outliner* view of the *Outliner window*. Image (*Outliner view*) is an example of what the *Outliner* view looks like for the (*Parenting Example*). Cube A’s object name is `Cube_Parent` and cube B is `Cube_Child`.

### Separating Objects

At some point, you’ll come to a time when you need to cut parts away from a mesh to be separate. Well, the operation is easy.

To separate an object, the vertices (or faces) must be selected and then separated, though there are several different ways to do this. In Edit Mode, press `P` then select one of the following.



Fig. 2.80: Suzanne dissected neatly

### Options

**Selected** This option separates the selection to a new object.

**All Loose Parts** Separates the mesh in its unconnected parts.

**By Material** Creates separate mesh objects for each material.

### Grouping objects

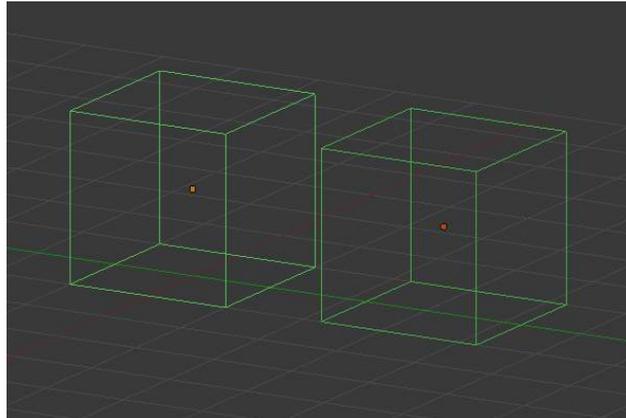


Fig. 2.81: Grouped objects

Group objects together without any kind of transformation relationship. Use groups to just logically organize your scene, or to facilitate one-step appending or linking between files or across scenes. Objects that are part of a group always shows as light green when selected; see image (*Grouped objects*).

### Options

**Creating a Group** `Ctrl-G` creates a new group and adds the selected object(s) to it.

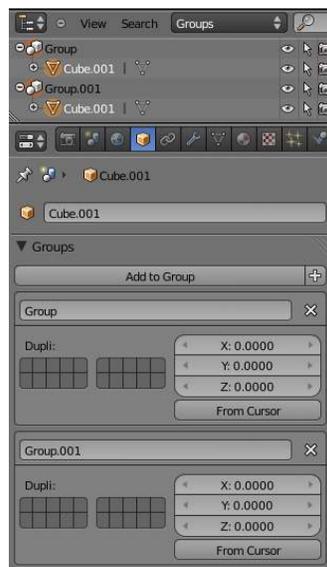


Fig. 2.82: Naming a Group

**Naming a Group** All groups that an object has been assigned to are listed in the *Object Properties Panel* 's *Relations* panel. To rename a group, simply click in the groups name field. To name groups in the *Outliner* window, select *Groups* as the outliner display from the header combo box, and `Ctrl-LMB` click on the group name. The name will change to an editable field; make your changes and press `Return`.

**Restricting Group Contents via Layers** The cluster of layer buttons attached to each group determines from which layers the group objects will be included when duplicated. If your group contains objects on layers 10, 11 and 12, but you disable the layer 12 button in the group controls, duplicates of that group (using the [Dupligroup](#) feature) will only show the portions of the group that reside in layers 10 and 11.

**Appending or Linking Groups** To append a group from another `.blend` file, consult [this page](#). In summary, *File* → *Link / Append Link* Select a `.blend` file and, and then the group.

**Removing Groups** To remove a object from a group, under the object context button, open the “Groups” pane. Find the name of the group from which you wish to remove the object, and click the x to the right of the group name.

---

## Select Grouped Reference

Mode: *Object mode*

Menu: *Select* → *Grouped*

Hotkey: `Shift-G`

---

Select objects by parenting and grouping characteristics. See [Select Grouped](#) for more information.

## Randomize Transform

---

### Reference

Mode: *Object mode*

Menu: *Object* → *Transform* → *Randomize Transform*

---

The randomize transform tool allows you to apply random translate, rotate, and scale values to an object or multiple objects. When applied on multiple objects, each object gets its own seed value, and will get different transform results from the rest.

### Options

**Random Seed** The random seed is an offset to the random transformation. A different seed will produce a new result.

**Transform Delta** Randomize Delta Transform values instead of regular transform. See [Delta Transforms](#).

**Randomize Location** Randomize Location vales

**Location** The maximum distances the objects can move along each axis.

**Randomize Rotation** Randomize rotation values.

**Rotation** The maximum angle the objects can rotate on each axis

**Randomize Scale** Randomize scale values.

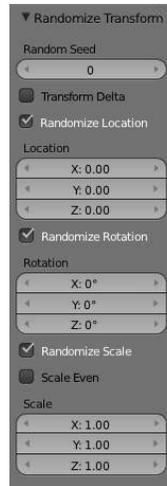


Fig. 2.83: Randomize transform options

**Scale Even** Use the same scale for each axis.

**Scale** The maximum scale randomization over each axis.

## Duplication

### Duplication

There are two types of duplication, being *Duplicate* and *Linked Duplicates* which supports instancing.

**Instancing** Each Blender object type (mesh, lamp, curve, camera *etc.*) is composed from two parts: an *Object* and *Object Data* (sometimes abbreviated to *ObData*):

**Object** Holds information about the position, rotation and size of a particular element.

**Object Data** Holds everything else. For example:

**Meshes** Store geometry, material list, vertex groups... *etc.*

**Cameras** Store focal length, depth of field, sensor size... *etc.*

Each object has a link to its associated object data, and a single object data may be shared by many objects.

### Duplicate Reference

Mode: *Edit* and *Object* modes

Menu: *Object* -> *Duplicate*

Hotkey: Shift-D

This will create a visually-identical copy of the selected object(s). The copy is created at the same position as the original object and you are automatically placed in *Grab* mode. Reference (*Duplicate Example*) for the discussions below.

This copy is a new object, which **shares** some datablocks with the original object (by default, all the Materials, Textures, and Ipos), but which has copied others, like the mesh, for example. This is why this form of duplication is sometimes called “shallow link”, because not all datablocks are shared; some of them are “hard copied”!

Note that you can choose which types of datablock will be linked or copied when duplicating: in the *User Preferences* ‘ (available in the *File* menu) *Editing* “tab”, activate those types of datablocks you want to really copy in the *Duplicate Data* list - the others will just be linked.

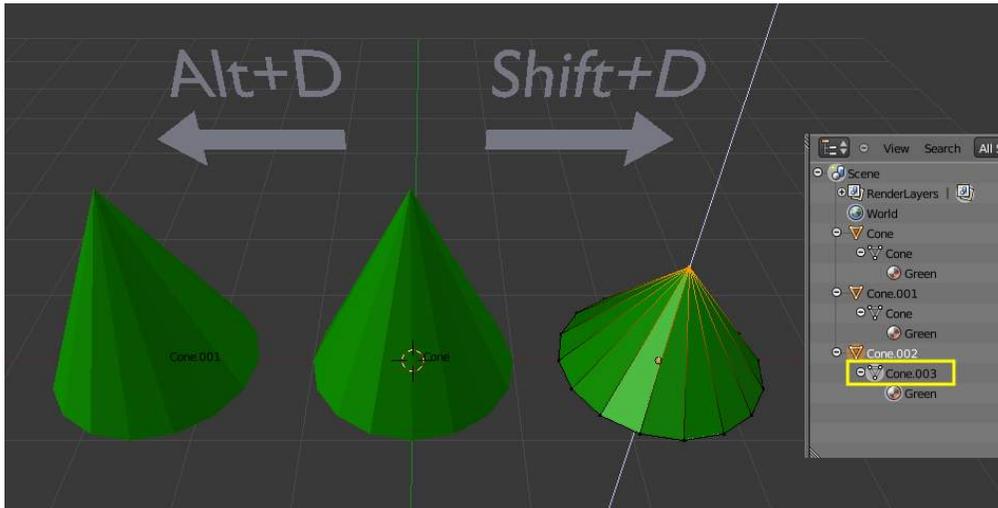


Fig. 2.84: The mesh `Cone.006` of object `Cone.002` is being edited. The mesh’s Unique datablock ID name is highlighted in the Outliner.

**Examples** The cone in the middle has been (1) link duplicated to the left and (2) duplicated to the right.

- The duplicated right cone is being edited, the original cone in the middle remains unchanged. The mesh data has been copied, not linked.
- Likewise, if the right cone is edited in object mode, the original cone remains unchanged. The new object’s transform properties or datablock is a copy, not linked.
- When the right cone was duplicated, it inherited the material of the middle cone. The material properties were linked, not copied.

See above if you want separate copies of the datablocks normally linked.

## Linked Duplicates

### Reference

Mode: *Object* mode

Menu: *Object* → *Duplicate Linked*

Hotkey: `Alt-D`

You also have the choice of creating a *Linked Duplicate* rather than a *Duplicate*; this is called a deep link. This will create a new object with **all** of its data linked to the original object. If you modify one of the linked objects in *Edit mode*, all linked copies are modified. Transform properties (object datablocks) still remain copies, not links, so you still can rotate, scale, and move freely without affecting the other copy. Reference (*Duplicate Example*) for the discussions below.

**Hint:** If you want to make changes to an object in the new linked duplicate independently of the original object, you will have to manually make the object a “single-user” copy by LMB the number in the *Object Data* panel of the Properties Window.

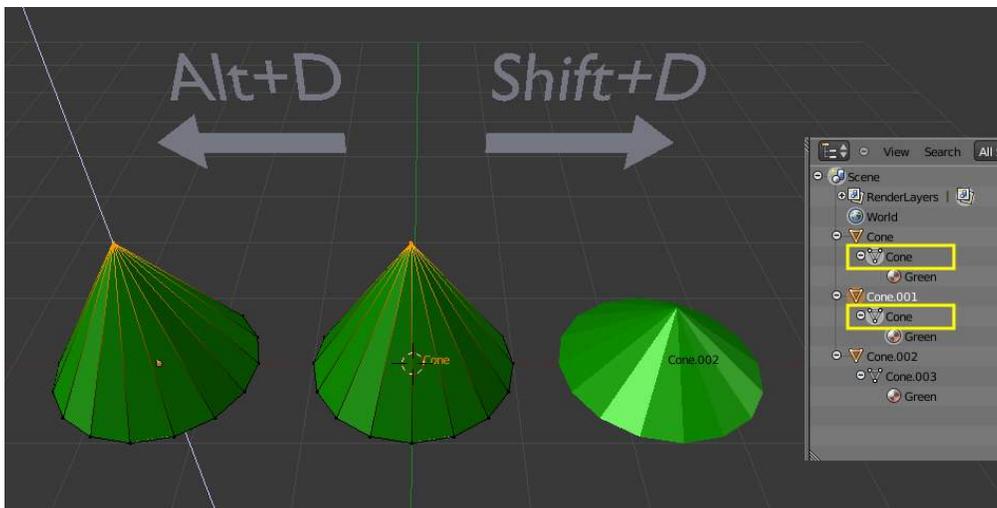
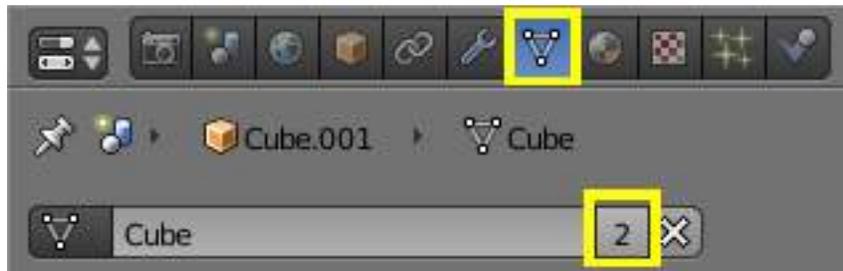


Fig. 2.85: The object `Cone.001` was linked duplicated. Though both these cones are separate objects with unique names, the single mesh named `Cone`, highlighted in the Outliner, is shared by both.

**Examples** The left cone is a *Linked Duplicate* of the middle cone (using `Alt-D`).

- As a vertex is moved in *Edit mode* in one object, the same vertex is moved in the original cone as well. The mesh data are links, not copies.
- In contrast, if one of these two cones is rotated or rescaled in object mode, the other remains unchanged. The transform properties are copied, not linked.
- As in the previous example, the newly created cone has inherited the material of the original cone. The material properties are linked, not copied.

A common table has a top and four legs. Model one leg, and then make linked duplicates three times for each of the remaining legs. If you later make a change to the mesh, all the legs will still match. Linked duplicates also apply to a set of drinking glasses, wheels on a car... anywhere there is repetition or symmetry.

### Procedural Duplication Reference

Mode: *Object mode* and *Edit mode*

Panel: *Object settings*

---

There are currently four ways in Blender to procedurally duplicate objects. These options are located in the *Object* menu.

**Verts** :This creates an instance of all children of this object on each vertex (for mesh objects only).

**Faces** :This creates an instance of all children of this object on each face (for mesh objects only).

**Group** :This creates an instance of the group with the transformation of the object. Group duplicators can be animated using actions, or can get a `Proxy`.

**Frames** :For animated objects, this creates an instance on every frame. As you'll see on this topic's subpage, this is also a *very* powerful technique for arranging objects and for modeling them.

---

## Linked Library Duplication Reference

Menu: *File* → *Link Append*

Hotkey: `Shift-F1`

---

**Linked Libraries** :Linked Libraries are also a form of duplication. Any object or datablock in other `.blend` files can be reused in the current file.

### Hints

- If you want transform properties (i.e. object datablocks) to be “linked”, see the page on [parenting](#).
- Material Transparency will not display when instancing dupli-groups; this is a known limitation of Blender's view-port.

## DupliVerts

---

### Reference

Mode: *Object* mode

Panel: *Object* → *Duplication*

---

*Duplication Vertices* or *DupliVerts* is the duplication of a base object at the location of the vertices of a mesh. In other words, when using *DupliVerts* on a mesh, an instance of the base object is placed on every vertex of the mesh.

There are actually two approaches to modeling using *DupliVerts*. They can be used as an arranging tool, allowing us to model geometrical arrangements of objects (e.g. the columns of a Greek temple, the trees in a garden, an army of robot soldiers, the desks in a classroom). The object can be of any object type which Blender supports. The second approach is to use them to model an object starting from a single part of it (e.g. the spikes in a club, the thorns of a sea-urchin, the tiles in a wall, the petals in a flower).

---

**Note:** Download example `.blend` file

You can download a file with the examples described on this page. In [this .blend](#), the first example, a monkey parented to a circle is on layer 1; while a tentacle parented to an icosphere is on layer 2.

## DupliVerts as an Arranging Tool

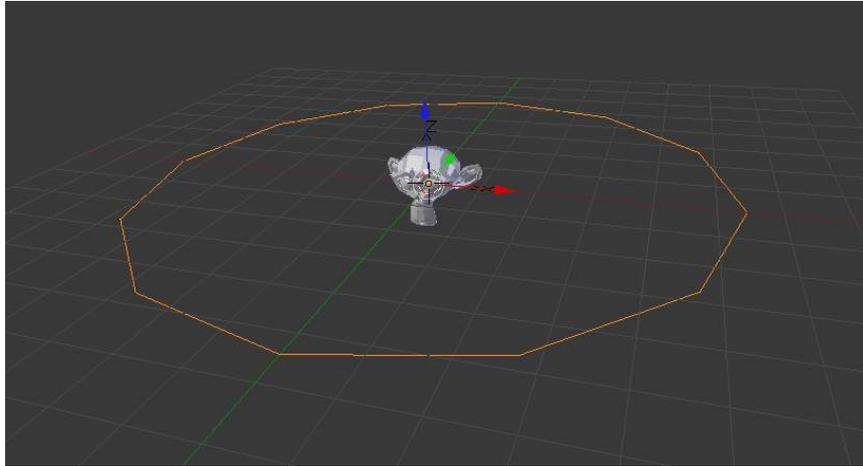


Fig. 2.86: A monkey head and a circle

**Setup** All you need is a base object (e.g. the *tree* or the *column*) and a pattern mesh with its vertices following the pattern you have in mind. In this section, we will use a simple scene for the following part. We'll be using a monkey head located at the origin of the coordinate system as our base object and a circle at the same location as our parent mesh.

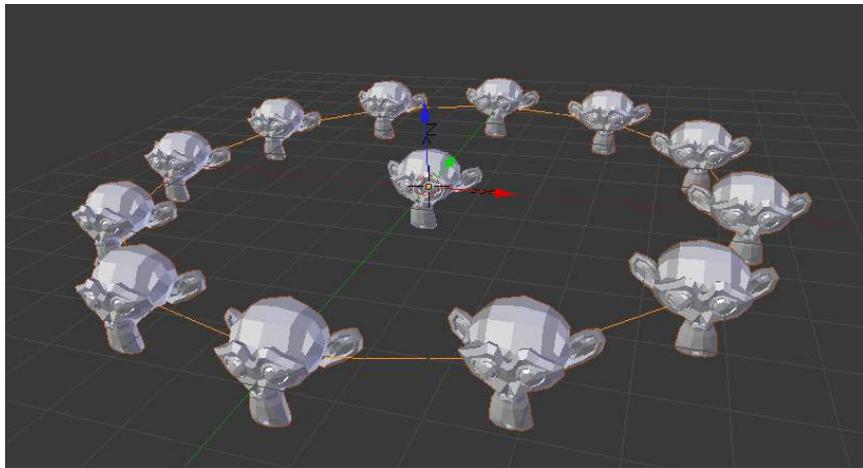


Fig. 2.87: Dupliverged monkeys

First, in *Object mode*, select the base object and **Shift-RMB** to add the circle to the selection (order is very important here), and **Ctrl-P** to parent the base object to the circle. Now, the circle is the parent of the monkey; if you move the circle, the monkey will follow it.

With only the circle selected, enable *Duplication vertices* in the *Object* panel → *Duplication* → *Verts*. A monkey head should be placed at every vertex of the circle.

The original monkey head at the center and the parent mesh are still shown in the 3D view but neither will be rendered. If the placement and rotation of your monkey head is odd, you might need to clear its rotation (**Alt-R**), scale **Alt-S**, location **Alt-G**, and origin **Alt-O**.

**Rearranging** If you now select the base object and modify it in either object or edit mode, all changes will also affect the shape of all duplicate objects. You can also select the parent mesh to modify the arrangement of the duplicates; adding vertices will also add more base objects. Note that the base objects will inherit changes made to the parent mesh in object mode, but not in edit mode - so scaling the circle up in object mode will enlarge the monkey head, while scaling the circle up in edit mode will only increase the distance between the base objects.

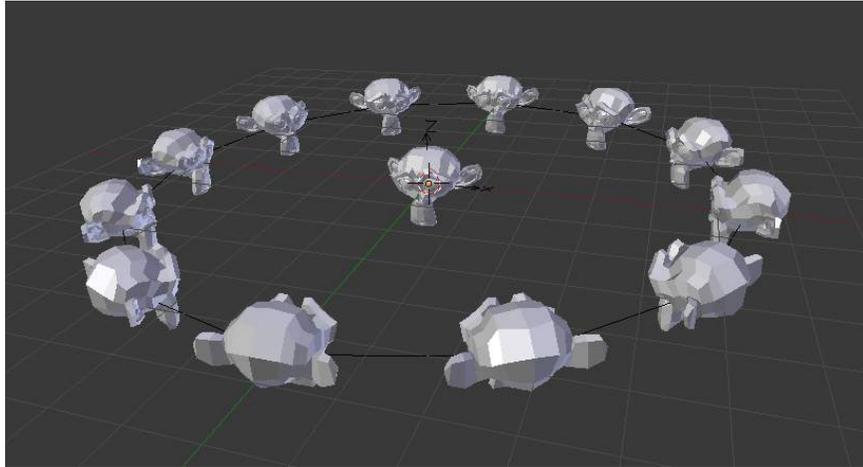


Fig. 2.88: Orientation enabled, orientation +Y

**Orientation** The orientation of the base objects can be controlled by enabling *Rotation* in the *Duplication* panel. This will rotate all base objects according to the vertex normals of the parent mesh.

To change the orientation of the duplicated objects, select the base object and in the *Object* → *Relations extras* panel change the *Tracking Axes*.

Output of various orientations:

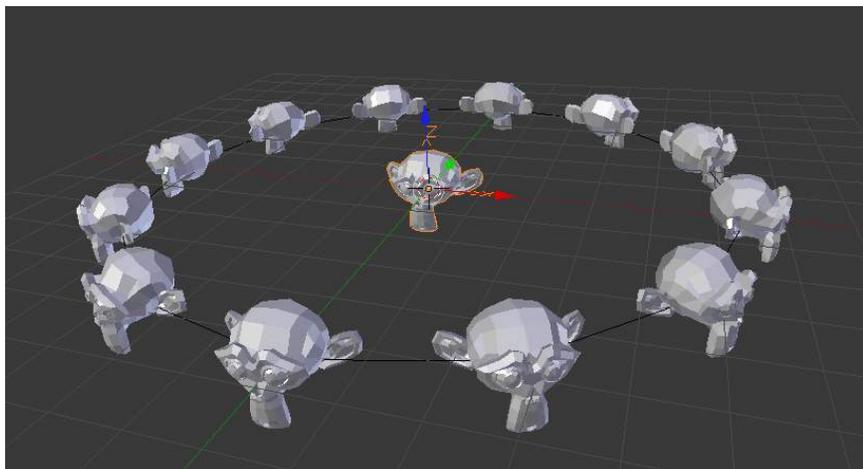


Fig. 2.89: Negative Y

**Note:** The axes of an object can be made visible in the *Object* → *Display* panel. To display the vertex normals of the parent mesh, tab into edit mode and enable this function in *Properties* (N) → *Display* panel where you can also resize the displayed normals as necessary.

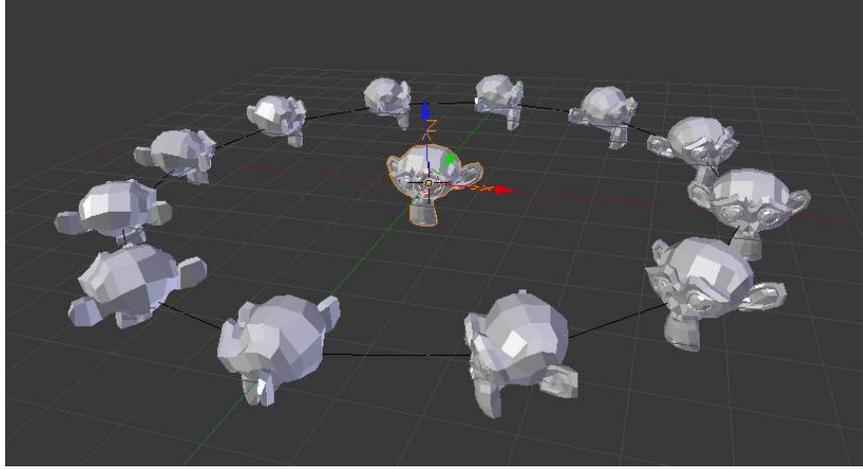


Fig. 2.90: Positive X

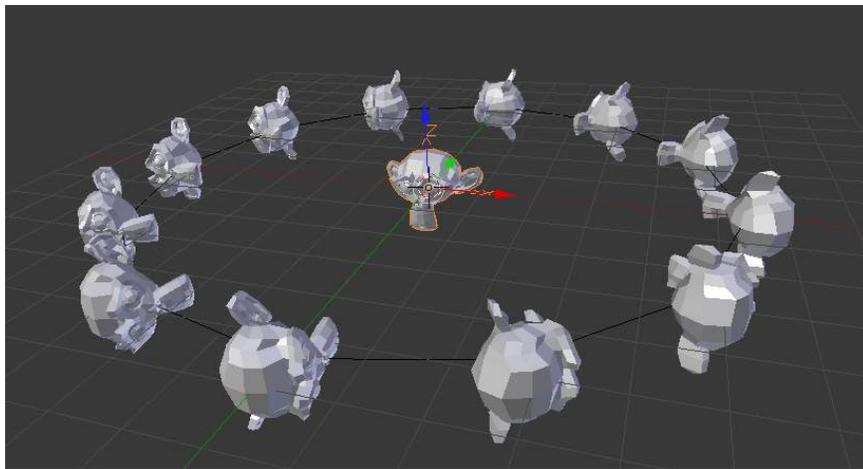


Fig. 2.91: Positive Z, up X

**DupliVerts as a Modeling Tool** Very interesting models can be made using DupliVerts and a standard primitive. In this example, a simple tentacle was made by extruding a cube a couple of times. The tentacle object was then parented to an icosphere. With dupli *Rotation* enabled for the parent mesh (the icosphere), the orientation of the base object (the tentacle) was adapted to the vertex normals of the parent mesh

(in this case the tentacle was rotated -90- about the X axis in edit mode).

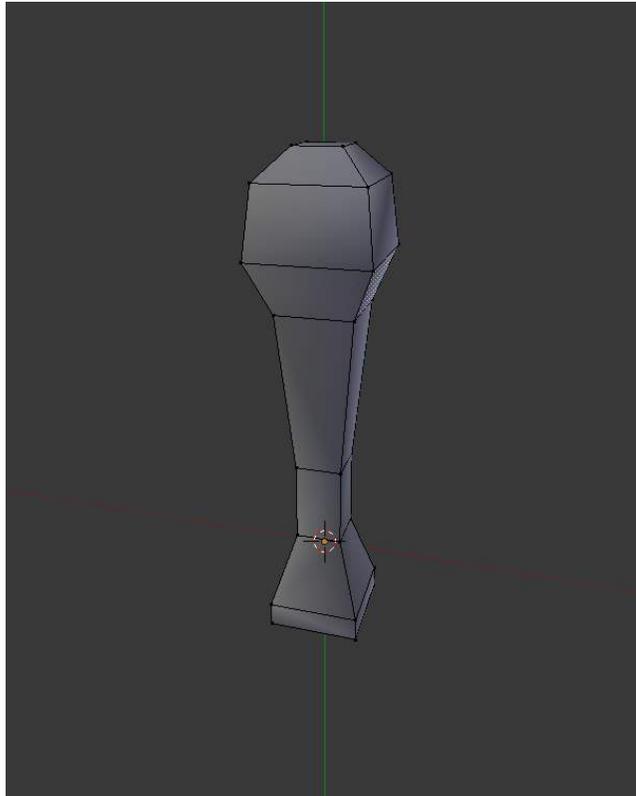


Fig. 2.92: A simple tentacle set to smooth

As in the previous example, the shape and proportions of the arrangement can now be tweaked.

To turn all duplicates into real objects, simply select the icosphere and *Object -> Apply -> Make Duplicates Real* (Ctrl-Shift-A). To make the icosphere and the tentacle a single object, make sure they are all selected and go to *Object -> Join* (Ctrl-J).

**See also** Other duplication methods are listed [here](#).

## DupliFaces

---

### Reference

Mode: *Object* mode

Panel: *Object -> Duplication*

---

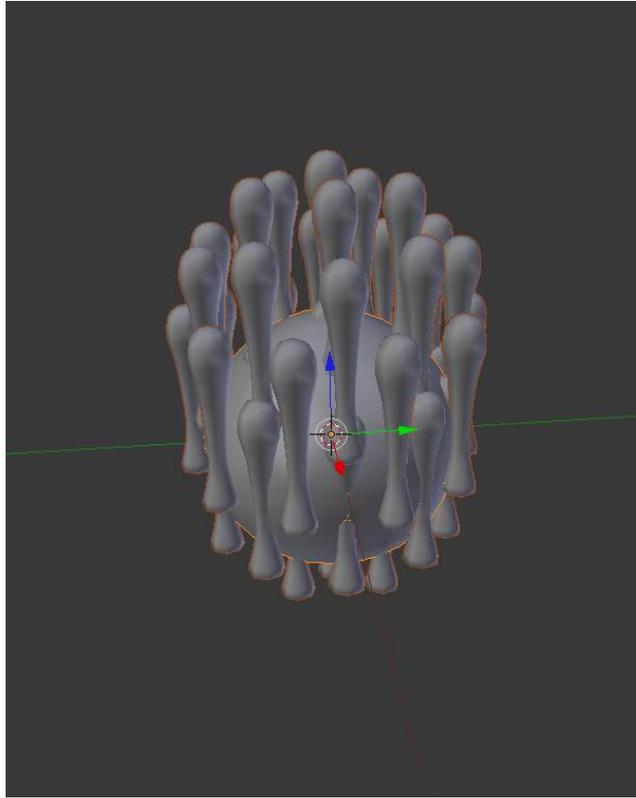


Fig. 2.93: Tentacle dupliverterted onto the parent mesh

*Duplication Faces* or *DupliFaces* is the capability to replicate an object on each face of a parent object. One of the best ways to explain this is through an example illustration.

---

**Note:** Example .blend file

Download the .blend file used for the examples on this page [here](#)

---

**Basic usage** In this example we will use a UV sphere with an extruded “north pole” as our base object and cube as our parent mesh. To parent the sphere to the cube, in *Object mode*, first RMB select the sphere, then *Shift*-RMB select the cube (order is very important here), and finally *Ctrl*-P to parent.

Next, in the *Object* context’s *Duplication* panel, enable *Faces*. The sphere is duplicated one for each face of the cube.

---

**Note:** Inherited properties

The location, orientation, and scale of the duplicated child(ren) matches that of the faces of the parent. So, if several objects are parented to the cube, they will all be duplicated once for each face on the cube. If the cube is subdivided (in *Edit Mode* *W*), every child will be duplicated for each face on the cube.

---

Both the parent object and original are displayed as editable “templates” in 3D view, but neither is rendered.

**Scale** By enabling *Scale* for the parent object, the scale of the child objects will be adapted to the size of each face in the parent object.

Thus, by rescaling the face of the parent object, the size of the duplicated object will change accordingly.

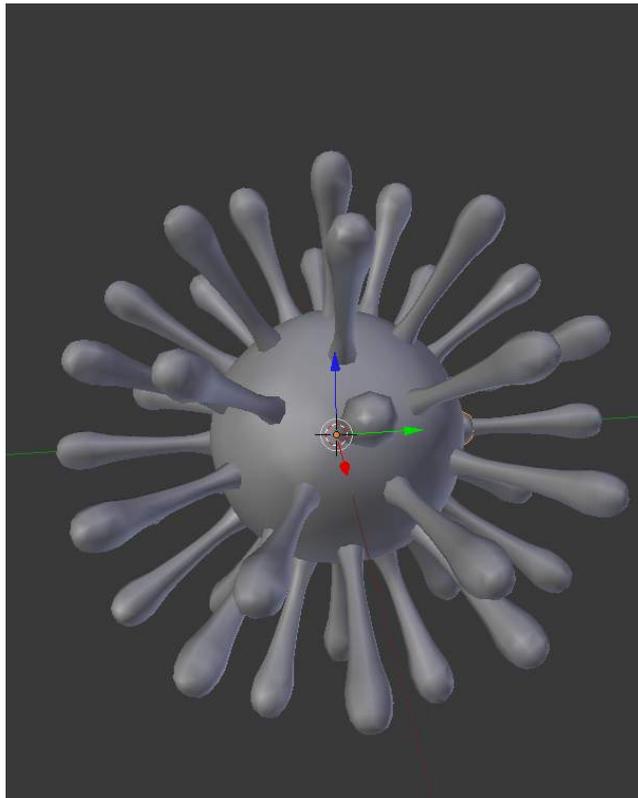


Fig. 2.94: Rotation enabled to align duplicates

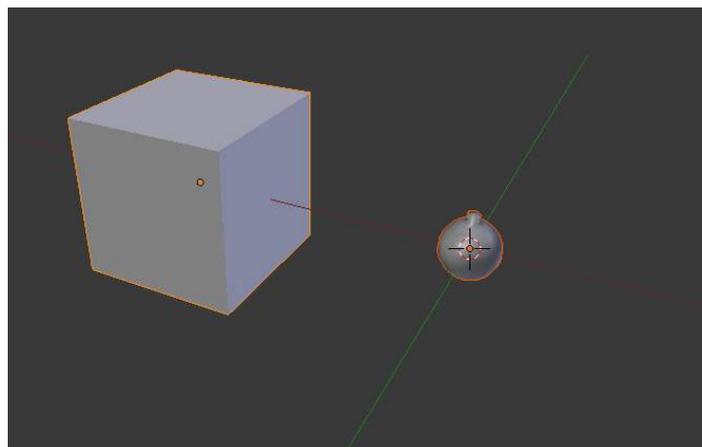


Fig. 2.95: A cube and a sphere

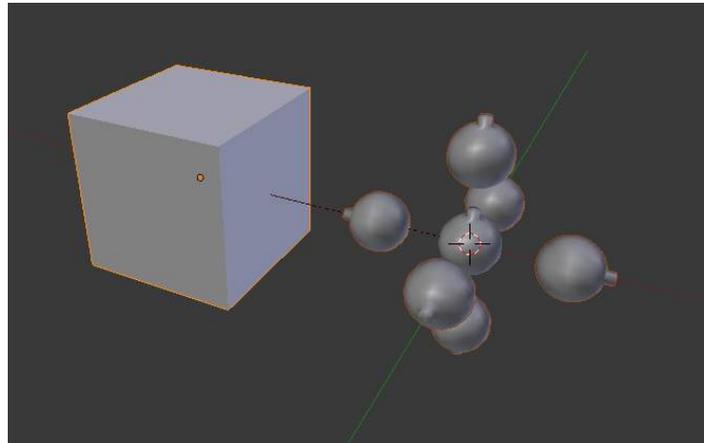


Fig. 2.96: Duplication Faces applied to the cube

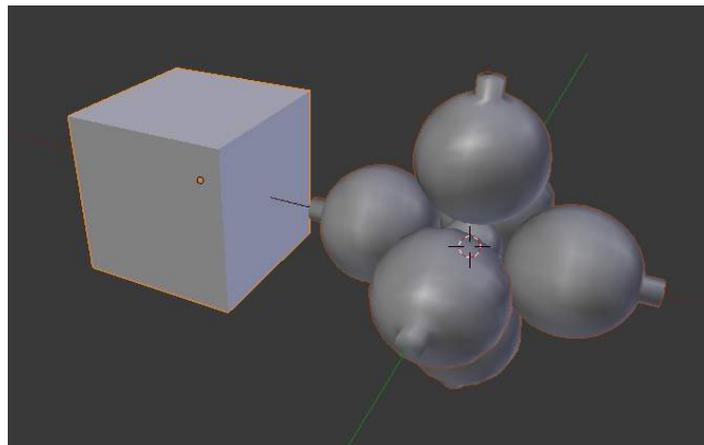


Fig. 2.97: Scale enabled

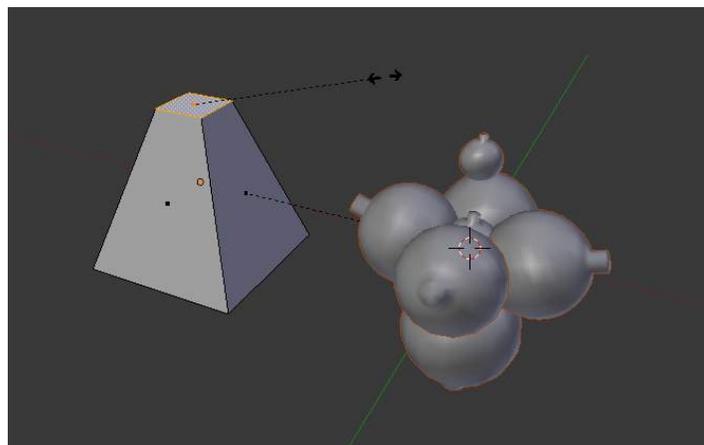


Fig. 2.98: Top face of cube scaled down

**Limitations / Considerations** The positioning of the duplicated geometry relative to the face is dependent upon the position of the child objects relative to the duplicator's origin. This can lead to some visual artifacts in the editor as the geometry of the original objects overlaps or intersects with the duplicates. One workaround is to move the origin of the duplicator mesh off of the plane of the faces.

If the geometry of the children is not symmetrical then the orientation of the face (as determined by the order of its vertices) could matter. As of 2.70 blender does not have tools which allow you to adjust the ordering of the vertices on a face.

However, there is a workflow that lets you control for this. Make a single square and enable the Duplication / Faces so you can see the duplicated geometry in your editor window. If the orientation is not what you want, rotate the face until it is how you want. Typically you want to do the rotation in Edit mode, not Object mode, but this is not a hard rule.

Once you have the orientation correct, Duplicate the face and move the duplicate where you want it. Repeat this process until you have enough faces. Since it is common for these faces to butt up against one another, your geometry will have lots of duplicate vertices. Use the Remove Doubles button in the Tools panel.

[A short video illustrating this workflow.](#)

## DupliFrames

DupliFrames is a tool to duplicate objects at frames distributed along a path. This is a useful tool to quickly arrange objects.

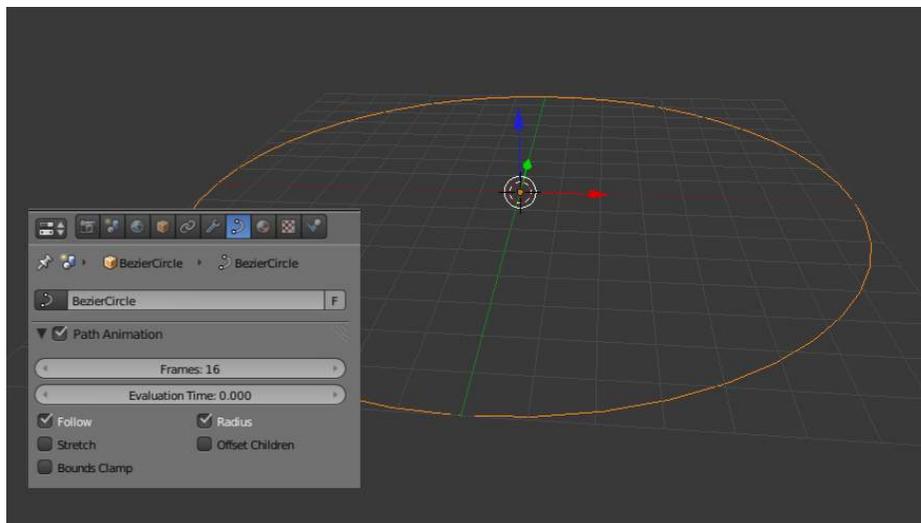


Fig. 2.99: Settings for the curve

**Examples** `Shift-A` to add a *Bezier Circle* and scale it up. In the *Curve* menu under *Path Animation* enable *Follow* and set *Frames* to something more reasonable than 100 (say 16).

Add a *Monkey*. In the *Object* menu under *Duplication* enable *Frames* and disable *Speed*.

---

### Note: Speed

The *Speed* option is used when the parent-child relationship is set to *Follow Path* (see below). In this example, the monkey will then travel along the circle over 16 frames.

---

To parent the monkey to the Bezier circle, first select the monkey then the curve (so that the curve is the active object) and `Ctrl-P`. Select the monkey and `Alt-O` to reset its origin.

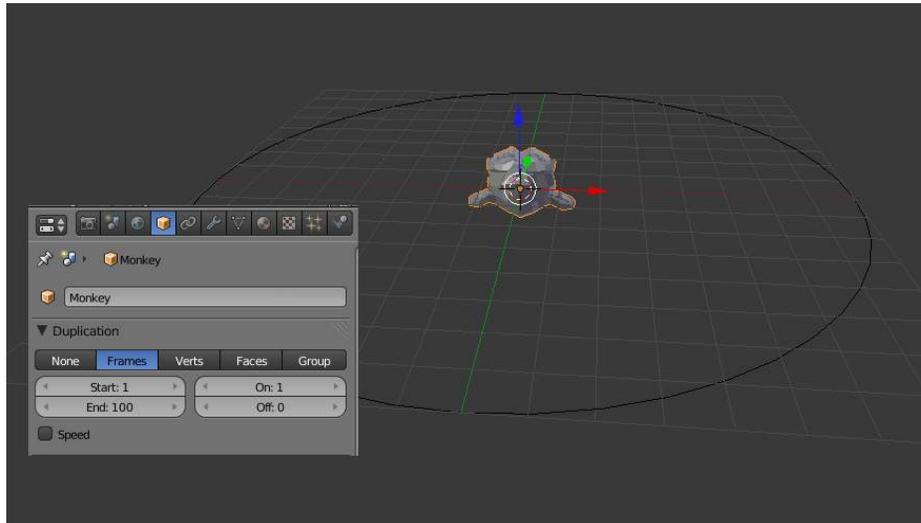


Fig. 2.100: Settings for the object

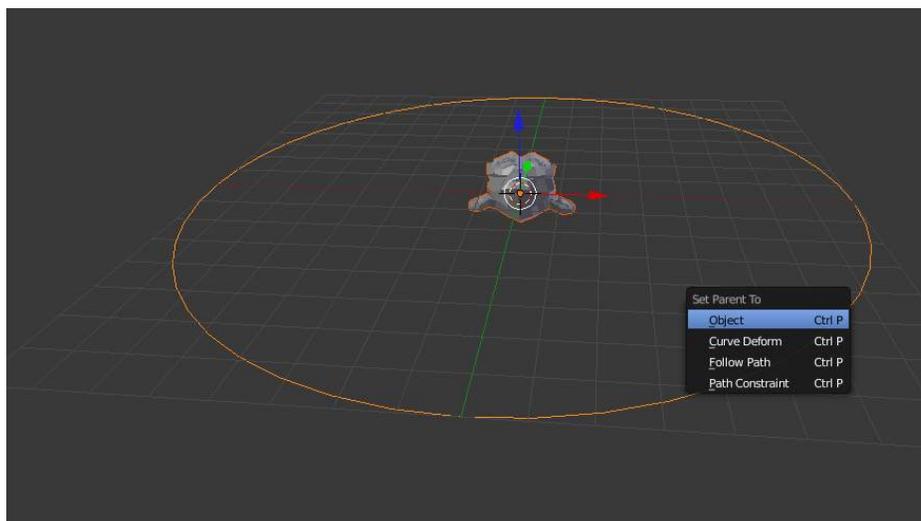


Fig. 2.101: Parenting

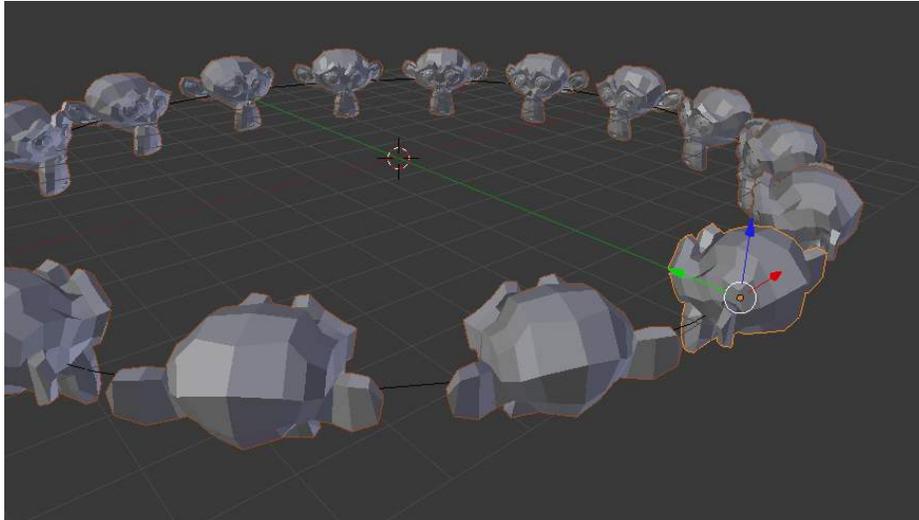


Fig. 2.102: Orientation tweaks

You can now change the orientation of the monkey by either rotating it (either in *Edit mode* or *Object mode*) or by changing the *Tracking Axes* under *Animation Hacks* (with the monkey selected). The arrangement of monkeys can, of course, be further enhanced by editing the curve.

To transform all monkeys into real objects, first `Ctrl-Shift-A` to *Make Duplicates Real*. All monkeys are now real objects, but still linked copies. To change this, *Object* → *Make Single User* → *ObjectData* then choose *All*.

---

**Note:** There are many alternatives to Dupliframes. Which tool to use depends on context.

- To use a small curve as a profile and a larger curve as a path, simply use the former as a *Bevel Object* to the latter.
  - To arrange objects along a curve, combining an *Array Modifier* and a *Curve Modifier* is often useful.
  - Dupliverts can be used to arrange objects, for example, along a circle or across a subdivided plane.
- 

### External links

- [Blender Artists: Dupliframes in 2.5](#)

### DupliGroup

---

### Reference

Mode: *Object mode*

Panel: *Object* → *Duplication* → *Group*

---

*Duplication Group* or *DupliGroup* allows you to create an instance of a group for each instance of another object.

### Basic Usage

- **Create a number of objects and group them by**

- selecting them all,
  - `Ctrl-G`, and
  - eventually rename your group in *Object -> Groups*
- **Create a DupliGroup by**
    - adding another object (`Shift-A`), say an *Empty*,
    - in *Object -> Duplication* enable *Group*, and
    - select the name of your newly created group in the selection box that appears.

**DupliGroup and Dynamic Linking** See [Appending and Linking](#) to understand how to dynamically link data from another `.blend` file into the current file. You can dynamically link groups from one blend file to another. When you do so, the linked group does not appear anywhere in your scene until you create an object controlling where the group instance appears.

### Example

- Link a group from another file into your scene, as described in [Appending and Linking](#).

From here, you can use the easy way or the hard way:

- **The easy way:**
  - Select *Add -> Group Instance -> [name of group you just linked]*.
- **The hard way:**
  - Select *Add -> Empty*, and select the empty that you added.
  - Switch to the *Object* context, and in the *Duplication* panel, click *Group*.
  - In the dropdown box that appears next to *Group:*, pick the group that you linked.

At this point, an instance of the group will appear. You can duplicate the empty, and the DupliGroup settings will be preserved for each empty. This way, you can get multiple copies of linked data very easily.

**Making a DupliGroup Object Real** Say you want to make further edits on an DupliGroup instance:

Simply select your DupliGroup and press `Ctrl-Shift-A` to convert the DupliGroup into regular objects that can be transformed and animated normally.

---

**Note:** Note that if the DupliGroup was linked from an external file the Object Data (mesh, materials, textures, transforms) will also still be linked from the original group. However, the various object's parent-child relationships do not carry over.

---

## 2.3.3 Meshes

### Edit Mode

#### Entering Edit Mode

You can work with geometric objects in two modes.

**Object mode** Operations in *Object Mode* affect the whole object. **Object mode** has the following header in the 3D view:

**Edit mode** Operations in *Edit mode* affect only the geometry of an object, but not global properties such as location or rotation.

**Edit Mode** has the following header in the 3D view:

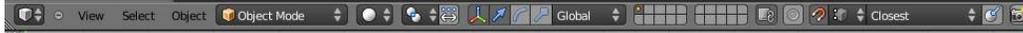


Fig. 2.103: Object Mode header



Fig. 2.104: Edit Mode header

Tools and modes in the 3D view header are (left to right):

- View, Select, and Mesh menus
- Blender Mode
- Display method for 3D view
- Pivot center
- 3D manipulator widget
- Selection mode
- Depth buffer clipping (hide)
- Proportional editing
- Snap
- OpenGL render

You can switch between the Object and Edit modes with the `Tab` key. You can change to any mode by selecting the desired *Mode* in the menu in the 3d view header.

After creating an object you may be immediately placed in *Edit mode* - depending on whether the *Switch to Edit Mode* button is toggled in the *User Preferences Editing* tab. *Edit mode* only applies to one object at a time, the *active*, or most recently selected, object.

### Visualization

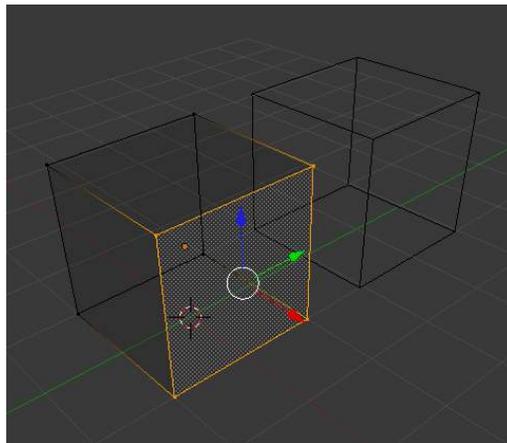


Fig. 2.105: One cube selected

By default, Blender highlights selected geometry in orange in both *Object mode* and *Edit mode*.

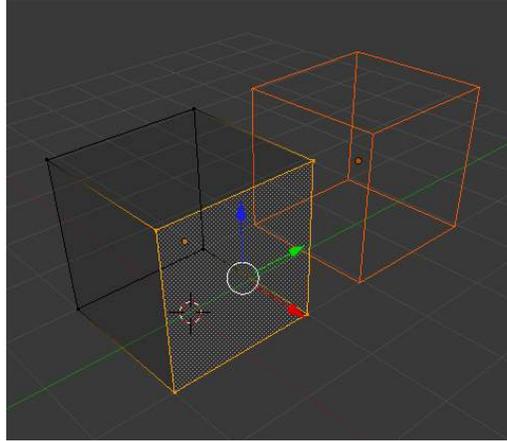


Fig. 2.106: Two cubes selected before entering Edit mode

In *Object mode* with *Wireframe* shading enabled (Z), objects are displayed in black when unselected and in orange when selected. If more than one object is selected, all selected object except the active object, typically the object last selected, is displayed in a darker orange color. Similarly, in *Edit mode*, unselected geometry is drawn in black while selected faces, edges, or vertices are drawn in orange. The active face is highlighted in white.

In *Edit mode*, only one mesh can be edited at the time. However, several objects can be joined into a single mesh (Ctrl-J in *Object mode*) and then separated again (P in *Edit mode*). If multiple objects are selected before entering *Edit mode*, all the selected objects remain highlighted in orange indicating that they are part of the active selection set.

If two vertices joined by an edge are selected in *Vertex selection mode*, the edge between them is highlighted too. Similarly, if enough vertices or edges are selected to define a face, that face is also highlighted.

## Tool Shelf



Fig. 2.107: The Tool Shelf panel in edit mode (panel split in two parts for layout reasons)

Open/close the *Mesh Tools* panel using T. When entering *Edit mode*, several mesh tools become available.

Most of these tools are also available as shortcuts (displayed in the *Tooltips* for each tool) and/or in the *Specials* menu ( $\bar{w}$ ), the *Edge* menu ( $\text{Ctrl-E}$ ), and *Face* menu ( $\text{Ctrl-F}$ ). For each tool a context-dependent menu is opened at the bottom of the *Tool Shelf*.

Even more mesh editing tools can be enabled in the *User Preferences* 'Add-ons' section. The development of new tools is regularly announced on Blender-related sites and forums.

For further information on panels see the `Reference` `panels` section.

## Properties Shelf

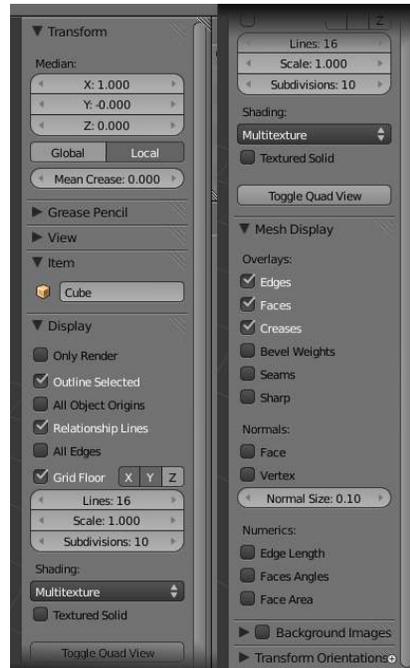


Fig. 2.108: The Properties Shelf panel in edit mode (panel split in two parts for layout reasons)

Open/close the *Properties Shelf* using  $N$ .

In the *Properties Shelf*, panels directly related to mesh editing are the *Transform* panel, where numeric values can be entered, and the *Mesh Display* panel, where for example normals and numeric values for distances, angles, and areas can be turned on.

Other useful tools are found in the *Properties Editor* under the *Object* 's and *Object Data* 's *Context buttons*, including display options and *Vertex groups*.

### Mesh Display TODO...

- Overlays
- Normals
- Edge/Face Info

## Vertices, Edges and Faces

In basic meshes, everything is built from three basic structures: *Vertices*, *Edges* and *Faces* (we're not talking about curves, NURBS, and so forth here). But there is no need to be disappointed: this simplicity still provides us with a wealth of possibilities

that will be the foundation for all our models.

## Vertices

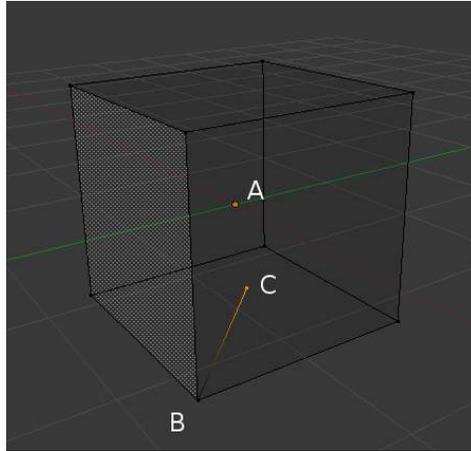


Fig. 2.109: Vertex example

A vertex is primarily a single point or position in 3D space. It is usually invisible in rendering and in *Object mode*. Don't mistake the center point of an object for a vertex. It looks similar, but it's bigger and you can't select it. (*Vertex example*) shows the center point labeled as A. B and C are vertices.

A simple way to create a new vertex is to click `Ctrl-LMB` in *Edit mode*. Of course, as a computer screen is two-dimensional, Blender can't determine all three vertex coordinates from a single mouse click, so the new vertex is placed at the depth of the 3D cursor. Using the method described above, any vertices selected previously are automatically connected to the new ones by an edge. In the image above, the vertex labeled C is a new vertex added to the cube with a new edge added between B and C.

## Edges

An edge always connects two vertices by a straight line. The edges are the “wires” you see when you look at a mesh in wireframe view. They are usually invisible on the rendered image. They are used to construct faces. Create an edge by selecting two vertices and pressing `F`.

## Faces

A face is the highest level structure in a mesh. Faces are used to build the actual surface of the object. They are what you see when you render the mesh. A face is defined as the area between either three (triangles) or four (quadrangles) vertices, with an edge on every side. Triangles are always flat and therefore easy to calculate. On the other hand, quadrangles “deform well” and are therefore preferred for subdivision modeling.

Take care when using four-sided faces (quads), because internally they are simply divided into two triangles each. Four-sided faces only work well if the face is pretty much flat (all points lie within one imaginary plane) and convex (the angle at no corner is greater than or equal to 180 degrees). This is the case with the faces of a cube, for example. That's why you can't see any diagonal in its wireframe model, because they would divide each square face into two triangles.

While you could build a cube with triangular faces, it would just look more confusing in *Edit mode*. An area between three or four vertices, outlined by edges, doesn't have to be a face. If this area does not contain a face, it will simply be transparent or non-existent in the rendered image. To create a face, select three or four suitable vertices and press `F`.

## Loops

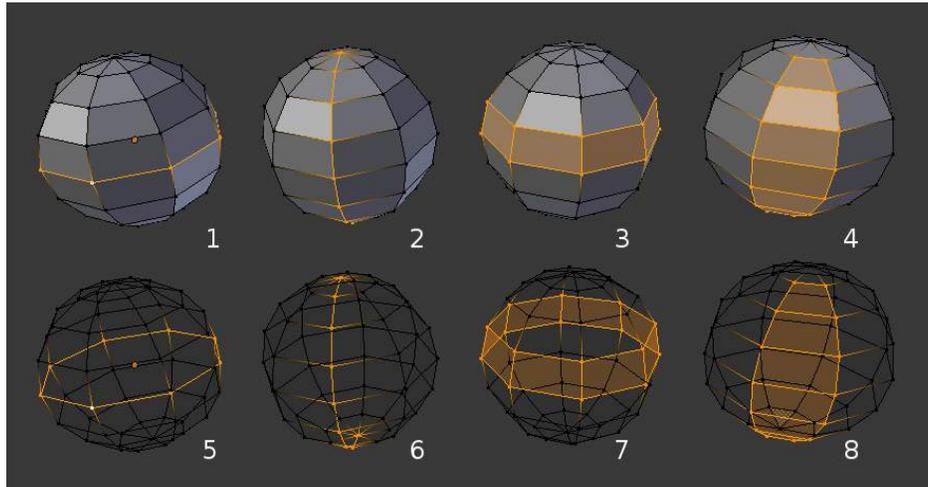


Fig. 2.110: Edge and Face Loops

*Edge* and *Face Loops* are sets of faces or edges that form continuous “loops” as shown in (*Edge and Face Loops*). The top row (1 - 4) shows a solid view, the bottom row (5 - 8) a wireframe view of the same loops.

Note that loops 2 and 4 do not go around the whole model. Loops stop at so called poles because there is no unique way to continue a loop from a pole. Poles are vertices that are connected to either three, five, or more edges. Accordingly, vertices connected to exactly one, two or four edges are not poles.

In the image above, loops that do not end in poles are cyclic (1 and 3). They start and end at the same vertex and divide the model into two partitions. Loops can be a quick and powerful tool to work with specific, continuous regions of a mesh and are a prerequisite for organic character animation. For a detailed description of how to work with loops in Blender, see: [Edge and Face Tools](#).

### Edge Loops

Loops 1 and 2 in (*Edge and Face Loops*) are edge Loops. They connect vertices so that each one on the loop has exactly two neighbors that are not on the loop and placed on both sides of the loop (except the start and end vertex in case of poles).

Edge Loops are an important concept especially in organic (subsurface) modeling and character animation. When used correctly, they allow you to build models with relatively few vertices that look very natural when used as subdivision surfaces and deform very well in animation.

Take (*Edge Loops in organic modeling*) as an example: the edge loops follow the natural contours and deformation lines of the skin and the underlying muscles and are more dense in areas that deform more when the character moves, for example at the shoulders or knees.

Further details on working with Edge Loops can be found in [Edge Loop Selection](#).

### Face Loops

These are a logical extension of Edge Loops in that they consist of the faces between two Edge Loops, as shown in loops 3 and 4 in (*Edge and Face Loops*). Note that for non-circular loops (4) the faces containing the poles are not included in a Face Loop.

Further details on working with Face Loops can be found in [Face Loop Selection](#).

---

## Mesh Primitives

---

### Reference

Mode: *Object* mode

Menu: *Add* → *Mesh*

Hotkey: *Shift-A*

---

A common object type used in a 3D scene is a mesh. Blender comes with a number of “primitive” mesh shapes that you can start modeling from.

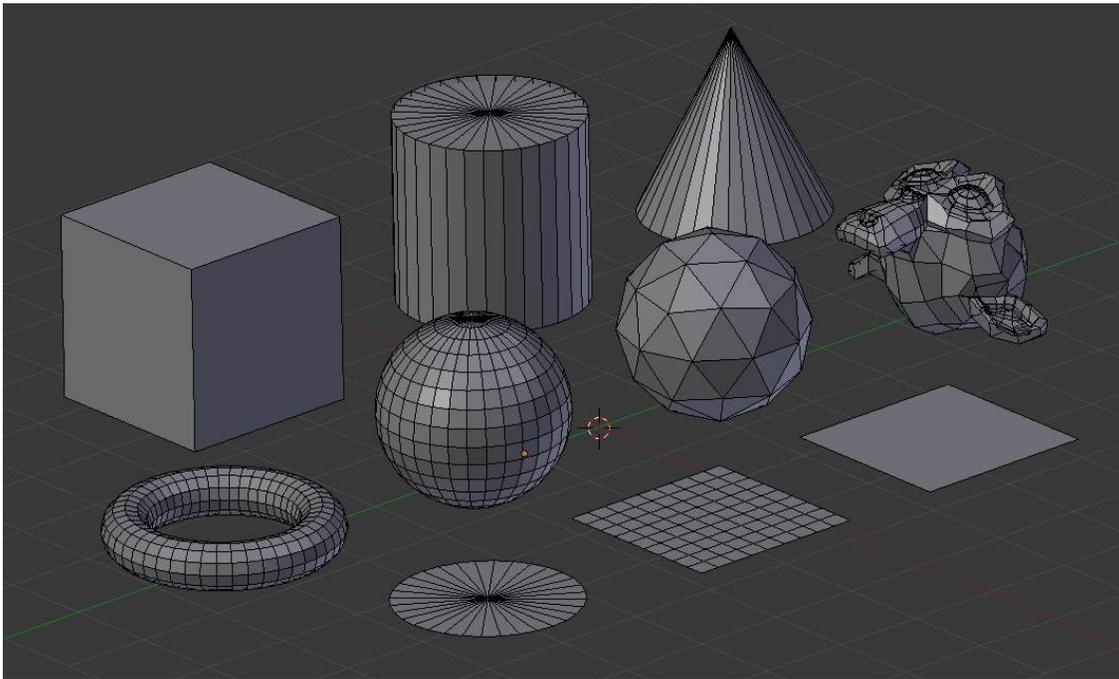


Fig. 2.111: Blender’s ten standard primitives

Options included in more than one primitive are:

**Radius** Sets the starting size for *Circle*, *Cylinder*, *Cone*, *UVSphere* and *IcoSphere*.

**Depth** Sets the starting length for *Cylinder* and *Cone*.

---

**Note:** Note about planar primitives

You can make a planar mesh three-dimensional by moving one or more of the vertices out of its plane (applies to *Plane*, *Circle* and *Grid*). A simple circle is actually often used as a starting point to create even the most complex of meshes.

---

### Plane

A standard plane contains four vertices, four edges, and one face. It is like a piece of paper lying on a table; it is not a real three-dimensional object because it is flat and has no thickness. Objects that can be created with planes include floors, tabletops,

or mirrors.

### Cube

A standard cube contains eight vertices, twelve edges, and six faces, and is a real three-dimensional object. Objects that can be created out of cubes include dice, boxes, or crates.

### Circle

A standard circle is comprised of  $n$  vertices. The number of vertices and radius can be specified in the context panel in the *Tool Shelf* which appears when the circle is created.

**Vertices** The number of vertices that define the circle. The more vertices the circle contains, the smoother its contour will be; see ("*Circles*" *obtained with various settings*). In contrast, a circle with only 3 vertices is actually a triangle - the circle is actually the standard way of adding polygons such as triangles, pentagons, et cetera.

**Radius** Sets the radius of the circle.

**Fill Type** Set how the circle will be filled

**Triangle Fan** Fill with triangular faces which share a vertex in the middle.

**Ngon** fill with a single ngon

**Nothing** Do not fill. Creates only the outer ring of vertices

### UV Sphere

A standard UV sphere is made out of  $n$  segments and  $m$  rings. The level of detail and radius can be specified in the context panel in the *Tool Shelf* which appears when the UV sphere is created. Increasing the number of segments and rings makes the surface of the UV sphere smoother.

**Segments** Number of vertical segments. Like Earth's meridians, going pole to pole and

**Rings** Number of horizontal segments. These are like Earth's parallels.

---

**Note:** If you specify a six segment, six ring UVsphere you'll get something which, in top view, is a hexagon (six segments), with five rings plus two points at the poles. Thus, one ring fewer than expected, or one more, if you count the poles as rings of radius 0.

---

### Icosphere

An icosphere is a polyhedra sphere made up of triangles. The number of subdivisions and radius can be specified in the context panel in the *Tool Shelf* after the Icosphere is created. Icospheres are normally used to achieve a more isotropical and economical layout of vertices than a UV sphere.

**Subdivisions** How many recursions are used to define the sphere. Increasing the number of subdivisions makes the surface of the Icosphere smoother. At level 1 the Icosphere is an icosahedron, a solid with 20 equilateral triangular faces. Any increasing level of subdivision splits each triangular face into four triangles, resulting in a more spherical appearance.

**Size** The radius of the sphere.

---

**Note:** It is possible to add an icosphere subdivided 500 times. Adding such a dense mesh is a sure way to cause a program crash. An icosphere subdivided 10 times would have 5,242,880 triangles, so be very careful about this!

---

## Cylinder

A standard cylinder is made out of  $n$  vertices. The number of vertices in the circular cross-section can be specified in the context panel in the *Tool Shelf* that appears when the object is created; the higher the number of vertices, the smoother the circular cross-section becomes. Objects that can be created out of cylinders include handles or rods.

**Vertices** Then number of vertical edge loops used to define the cylinder.

**Radius** Sets the radius of the cylinder.

**Depth** Sets the height of the cylinder.

**Cap Fill Type** Similar to circle (see above). When set to none, the created object will be a tube. Objects that can be created out of tubes include pipes or drinking glasses (the basic difference between a cylinder and a tube is that the former has closed ends).

## Cone

A standard cone is made out of  $n$  vertices. The number of vertices in the circular base, dimensions and option to close the base of the cone can be specified in the context panel in the *Tool Shelf* that appears when the object is created; the higher the number of vertices, the smoother the circular base becomes. Objects that can be created out of cones include spikes or pointed hats.

**Vertices** The number of vertical edge loops used to define the cone.

**Radius 1** Sets the radius of the base of the cone.

**Radius 2** Sets the radius of the tip of the cone. A value of 0 will produce a standard cone shape.

**Depth** Sets the height of the cylinder.

**Base Fill Type** Similar to circle (see above).

## Torus

A doughnut-shaped primitive created by rotating a circle around an axis. The overall dimensions are defined by the *Major* and *Minor Radius*. The number of vertices (in segments) can be different for the circles and is specified in the context panel in the *Tool Shelf* with both radii (*Major Segments* and *Minor Segments*).

**Major Radius** Radius from the origin to the center of the cross sections

**Minor Radius** Radius of the torus's cross section

**Major Segments** Number of segments for the main ring of the torus. If you think of a torus as a “spin” operation around an axis, this is how many steps in the spin.

**Minor segments** Number of segments for the minor ring of the torus. This is the number of vertices of each circular segment.

**Use Int+Ext Controls** Change the way the torus is defined:

**Exterior Radius** When *Use Int+Ext Controls* is active, if viewed along the major axis, this is the radius from the center to the outer edge.

**Interior Radius** When *Use Int+Ext Controls* is active, if viewed along the major axis, this is the radius of the hole in the center.

## Grid

A standard grid is made out of  $n$  by  $m$  vertices. The resolution of the x-axis and y-axis can be specified in the context panel in the *Tool Shelf* which appears when the object is created; the higher the resolution, the more vertices are created. Example

objects that can be created out of grids include landscapes (with the proportional editing tool or *Displace* modifier) and other organic surfaces. You can also obtain a grid when you create a plane and then use a subdivided modifier in *Edit mode*. However, there is a *Landscape* add-on available in the *User Preferences*.

**X Subdivisions** The number of spans in the x direction. Minimum of 3, creating two face loops.

**Y Subdivisions** The number of spans in the y direction.

**Size** The length of the sides of the grid.

## Monkey

This is a gift from old NaN to the community and is seen as a programmer's joke or "Easter Egg". It creates a monkey's head once you press the *Monkey* button. The Monkey's name is "Suzanne" and is Blender's mascot. Suzanne is very useful as a standard test mesh, much like the *Utah Tea Pot* or the *Stanford Bunny*.

## Add-ons

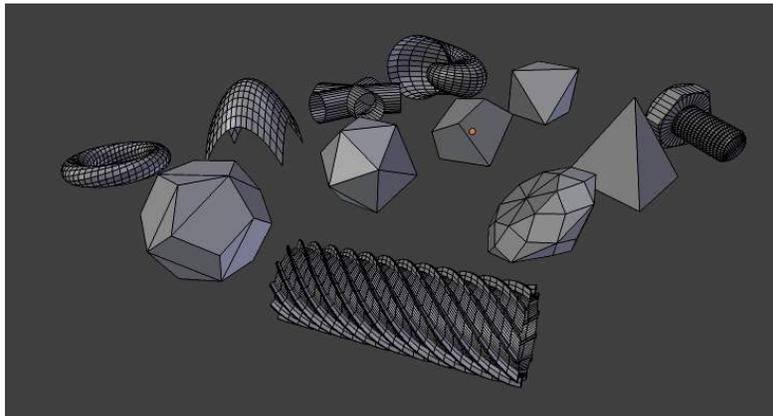


Fig. 2.112: A few of the mesh primitives available as add-ons.

In addition to the basic geometric primitives, Blender has a constantly increasing number of script generated meshes to offer as pre-installed add-ons. These become available when enabled in the *User Preferences* ' *Add-ons* section (filter by *Add Mesh*). Only a few are mentioned here:

**Landscape** Adds a landscape primitive. Many parameters and filters appear in the *Tool Shelf*.

**Pipe Joints** Adds one of five different pipe joint primitives. Radius, angle, and other parameters can be changed in the *Tool Shelf*.

**Gears** Adds a gear or a *worm* with many parameters to control the shape in the *Tool Shelf*.

## Mesh Analysis

Mesh analysis is useful for displaying attributes of the mesh that may impact certain use cases.

The mesh analysis works in editmode and shows areas with a high value in red, and areas with a low value in blue. Geometry outside the range is displayed grey.

Currently the different modes target 3d-printing as their primary use.

## Overhang

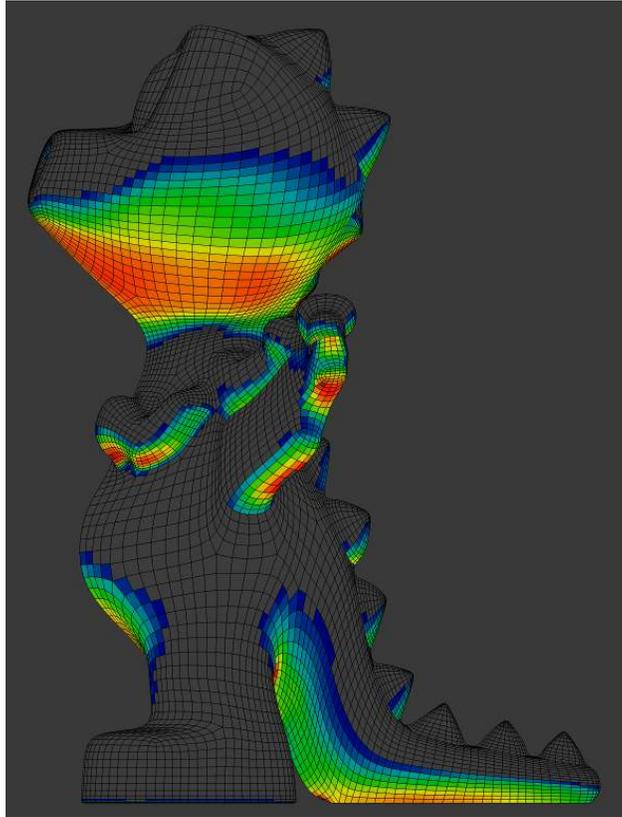


Fig. 2.113: Overhang

Extrusion 3D printers have a physical limit to the overhang that can be printed, this display mode shows the overhang with angle range and axis selection.

## Thickness

Printers have a limited *wall-thickness* where very thin areas can't be printed, this test uses ray casting and a distance range to the thickness of the geometry.

## Intersections

Another common cause of problems for printing are intersections between surfaces, where the inside/outside of a model can't be reliably detected.

Unlike other display modes, intersections have no variance and are either on or off.

## Distortion

Distorted geometry can cause problems since the triangulation of a distorted ngon is undefined.

Distortion is measured by faces which are not flat, with parts of the face pointing in different directions.

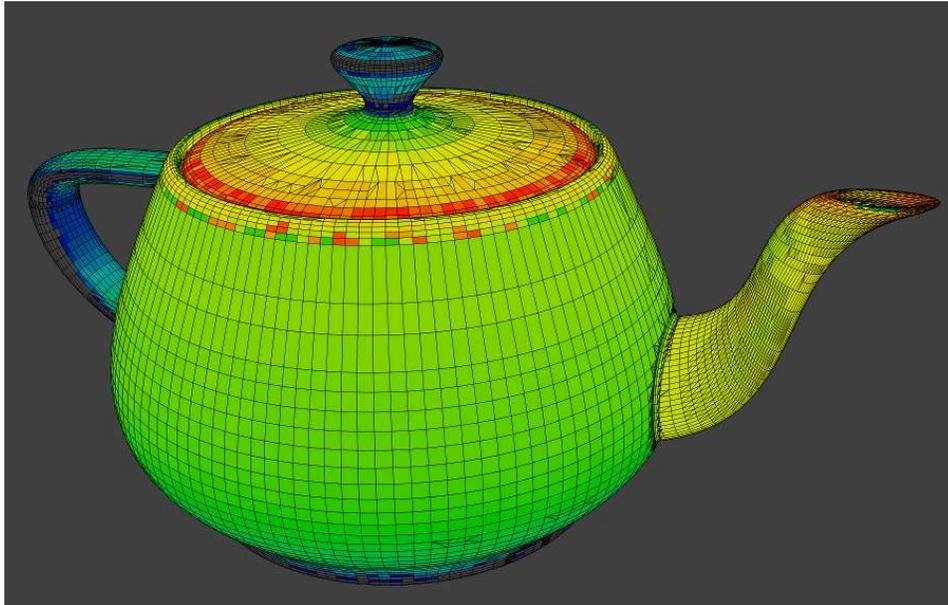


Fig. 2.114: Thickness

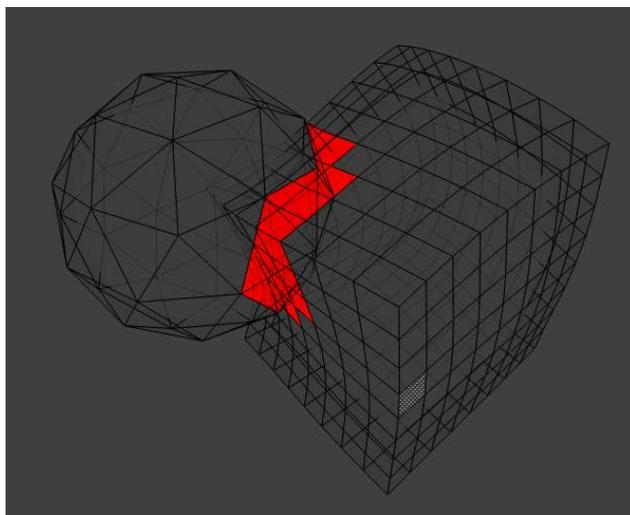


Fig. 2.115: Intersecting faces

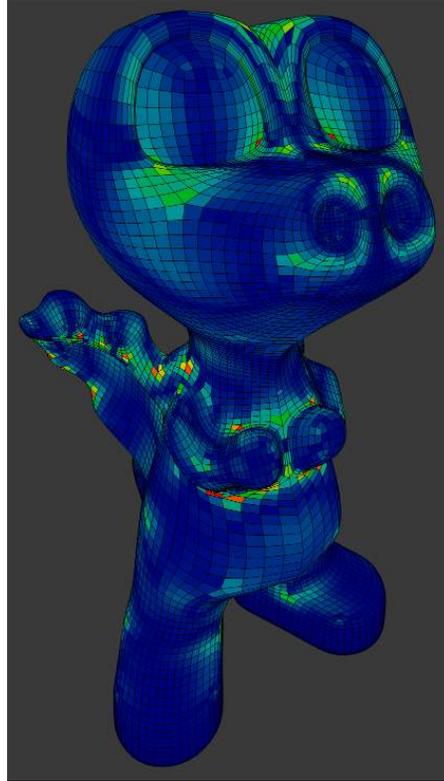


Fig. 2.116: Distorted Faces

## Sharp Edges

Similar to wall-thickness, sharp edges can form shapes that are too thin to be able to print.

**Warning:** There are some known limitations with mesh analysis

- Currently only displayed with deform modifiers.
- For high-poly meshes is slow to use while editing the mesh.

## Selecting

### Selecting Mesh Components

There are many ways to select elements, and it depends on what *Mesh Select Mode* you are in as to what selection tools are available. First we will go through these modes and after that a look is taken at basic selection tools.

### Selection Modes

**Select Mode Header Widgets** In *Edit mode* there are three different selection modes. You can enter the different modes by selecting one of the three buttons in the toolbar.

Using the buttons you can also use more than one selection mode at a time by `Shift-LMB` clicking the buttons.

**Vertices** Selected vertices are drawn in orange, unselected vertices in black, and the active or last selected vertex in white.

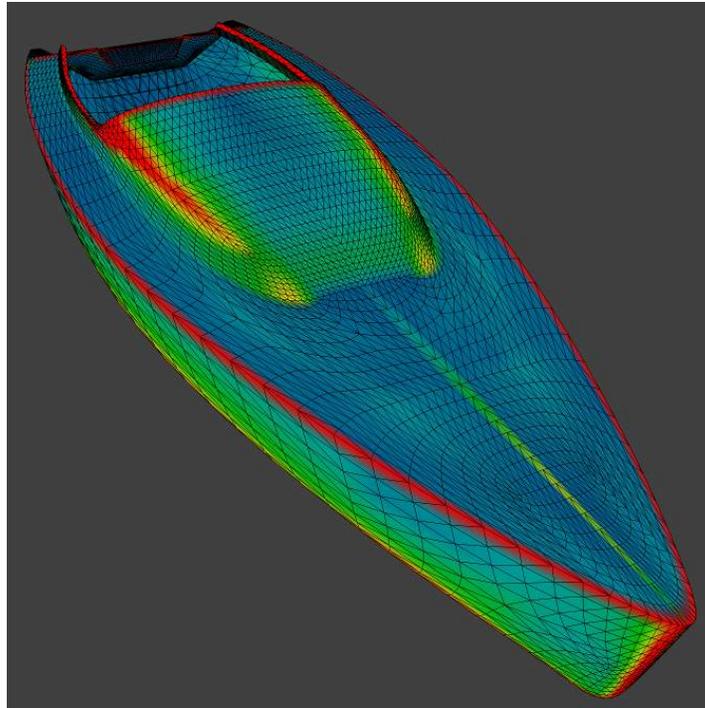


Fig. 2.117: Sharp edges

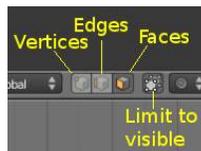


Fig. 2.118: Edit mode selection buttons

**Edges** In this mode the vertices are not drawn. Instead the selected edges are drawn in orange, unselected edges black, and the active or last selected edge in white.

**Faces** In this mode the faces are drawn with a selection point in the middle which is used for selecting a face. Selected faces and their selection point are drawn in orange, unselected faces are drawn in black, and the active or last selected face is highlighted in white.

Almost all modification tools are available in all three mesh selection modes. So you can *Rotate*, *Scale*, *Extrude*, etc. in all modes. Of course rotating and scaling a *single* vertex will not do anything useful without setting the pivot point to another location, so some tools are more or less applicable in some modes.

---

**Note:** The three selection mode buttons are only visible in *Edit mode*. The colors of selected, unselected and active geometry depend entirely on the current theme. Black, orange and white are from the default theme.

---

## Select Mode Pop-up

### Reference

Mode: *Edit mode*

Hotkey: `Ctrl-Tab`



Fig. 2.119: Mesh Select Mode menu

You can also choose a selection mode with the pop-up menu

**Select Mode** → **Vertices** Press `Ctrl-Tab` and select *Vertices* from the pop-up menu, or press `Ctrl-Tab1`.

**Select Mode** → **Edges** Press `Ctrl-Tab` and select *Edges* from the pop-up menu, or press `Ctrl-Tab2`.

**Select Mode** → **Faces** Press `Ctrl-Tab` and select *Faces* from the pop-up menu, or press `Ctrl-Tab3`.

**Switching select mode** When switching modes in an “ascendant” way (i.e. from simpler to more complex), from *Vertices* to *Edges* and from *Edges* to *Faces*, the selected parts will still be selected if they form a complete element in the new mode.

For example, if all four edges in a face are selected, switching from *Edges* mode to *Faces* mode will keep the face selected. All selected parts that do not form a complete set in the new mode will be unselected.

Hence, switching in a “descendant” way (i.e. from more complex to simpler), all elements defining the “high-level” element (like a face) will be selected (the four vertices or edges of a quadrangle, for example).

By holding `Ctrl` when selecting a higher selection mode, all elements touching the current selection will be added, even if the selection does not form a complete higher element.

See (*Vertices mode example*), (*Edges mode example*), (*Faces mode example*) and (*Mixed mode example*) for examples of the different modes.



**Selection Tools** The select menu in edit mode contains tools for selecting components. These are described in more detail in the following pages.

**Border Select** Enables a rectangular region for selection

**Circle Select** Enables a circular shaped region for selection

**(De)select All A** Select all or none of the mesh components.

**Invert Selection Ctrl-I** Selects all geometry that are not selected, and deselect currently selected components.

**Select Random** Selects a random group of vertices, edges, or faces, based on a percentage value.

**Checker Deselect** Deselect alternating faces, to create a checker like pattern.

**Select Sharp Edges** This option will select all edges that are between two faces forming an angle less than a given value, which is asked you *via* a small pop-up dialog. The lower is this angle limit, the sharper will be the selected edges. At 180, all *manifold* edges will be selected.

**Linked Flat Faces (Ctrl-Shift-Alt-F)** Select connected faces based on a threshold of the angle between them. This is useful for selecting faces that are planar.

**Interior Faces** Select faces where all edges have more than 2 faces.

**Side of Active** Selects all data on the mesh in a single axis

**Select Faces by Sides** Selects all faces that have a specified number of edges.

**Non Manifold (Ctrl-Shift-Alt-M)** Selects *non-manifold* geometry. See [Mesh Advanced Selection](#).

**Loose** Select all vertices or edges that do not form part of a face.

**Similar** Select geometry based on how similar certain properties are to it.

**More Ctrl-NumpadPlus** Propagates selection by adding geometry that are adjacent to selected elements.

**Less Ctrl-NumpadMinus** Deselects geometry that form the bounds of the current selection

**Mirror** Select mesh items at the mirrored location.

**Pick Linked L** Selects all geometry connected to the geometry under the cursor.

**Linked Ctrl-L** Selects all geometry that are connected to the current selection.

**Vertex Path** Selects a vertex path between two selected vertices

**Edge Loop** Selects a loop of edges from a selected edge

**Edge Ring** Selects edges parallel to a selected edge in the same ring of faces

**Loop Inner-Region** Converts a closed selection of edges to the region of faces it encloses

**Boundary Loop** Converts a selection of faces to the ring of edges enclosing it

### Selectable Elements

As we have seen in the [mesh structure](#) page, meshes are made of different element types (even though they are all inter-related: in a way, they are different “views”, “representations”, of the same basic data...), “vertices”, “edges” and “faces”.

Hence, you can select different parts of a mesh using one of these three types. There is one key point to understand here: *when you select a type of element (e.g. some edges), you implicitly select the other types of corresponding elements (e.g. all vertices defining those edges, as well as faces fully defined by these same edges)*. This is very important, as some tools only work on vertices, edges and/or faces: if you use a “face” tool with a selection of vertices, only the faces defined by these vertices will be affected.

In general, you will only select one type of element at a time, depending on the “select mode” you are using. However, you can successively add different elements to a same selection, switching between these select modes (see *Selected elements after switching select mode* for what is selected after switching select mode), or even use a “combined” select mode, also described below.

**Select Modes** You have two ways to switch between select modes:

### Select Mode pop-up Reference

---

Mode: *Edit* mode

Hotkey: Ctrl-Tab

In *Edit* mode there are three different select modes for meshes; see (*Select Mode menu*).



Fig. 2.128: Select Mode menu.

**Select Mode** → **Vertices** Press Ctrl-Tab and select *Vertices* from the pop-up menu, or press Ctrl-TabNumpad1. The selected vertices are drawn in yellow and unselected vertices are drawn in a pink color.

**Select Mode** → **Edges** Press Ctrl-Tab and select *Edges* from the pop-up menu, or press Ctrl-TabNumpad2. In this mode the vertices are not drawn. Instead the selected edges are drawn in yellow and unselected edges are drawn in a black color.

**Select Mode** → **Faces** Press Ctrl-Tab and select *Faces* from the pop-up menu, or press Ctrl-TabNumpad3. In this mode the faces are drawn with a selection point in the middle which is used for selecting a face. Selected faces are drawn in yellow with the selection point in orange, unselected faces are drawn in black.

Almost all modification tools are available in all three modes. So you can *Rotate*, *Scale*, *Extrude*, etc. in all modes. Of course rotating and scaling a *single* vertex will not do anything useful, so some tools are more or less applicable in some modes.

### Select Mode header widgets Reference

---

Mode: *Edit* mode

Panel: Header of the *3D View*

You can also enter the different modes by selecting one of the three buttons in the toolbar; see (*Edit mode select buttons*).

Using the buttons you can also enter **mixed** or “combined” mode by Shift-LMB clicking the buttons. This will allow you to select vertices, edges and/or faces at the same time!

**Note:** The “Mode Selection” buttons are only visible for meshes in *Edit* mode.

---



Fig. 2.129: Edit mode select mode buttons.

**Selected elements after switching select mode** When switching modes in an “ascendant” way (i.e. from simpler to more complex), from *Vertices* to *Edges* and from *Edges* to *Faces*, the selected parts will still be selected if they form a complete set in the new mode. For example, if all four edges in a face are selected, switching from *Edges* mode to *Faces* mode will keep the face selected. All selected parts that do not form a complete set in the new mode will be unselected.

Hence, switching in a “descendant” way (i.e. from more complex to simpler), all elements defining the “high-level” element (like a face) will be selected (the four vertices or edges of a quadrangle, for example).

See (*Vertices mode example*), (*Edges mode example*), (*Faces mode example*) and (*Mixed mode example*) for examples of the different modes.



### Basic Selection

---

#### Reference

Mode: *Edit* mode

Hotkey: RMB and Shift-RMB

---

The most common way to select an element is to RMB on that item; this will replace the existing selection with the new item.

**Adding to a Selection** To add to the existing selection, hold down *Shift* while right clicking. Clicking again on a selected item will deselect it.

As in *Object* mode, there is a unique *active* element, displayed in a lighter shade (in general, the last element selected). Depending on the tools used, this element might be very important!

Note that there is no option to choose what element to select between overlapping ones (like the *Alt*-RMB click in *Object* mode). However, if you are in solid, shaded, or textured viewport shading mode (not bounding box or wireframe), you will have a fourth button in the header that looks like a cube, just right of the select mode ones.

When enabled, this limits your ability to select based on visible elements (as if the object was solid), and prevents you from accidentally selecting, moving, deleting or otherwise working on backside or hidden items.

### Selecting Elements in a Region

---

#### Reference

Mode: *Edit* mode

Hotkey: B, C, and Ctrl-LMB click and drag

Region selection allows you to select groups of elements within a 2D region in your 3D view. The region can be either a circle or rectangle. The circular region is only available in *Edit mode*. The rectangular region, or *Border Select*, is available in both *Edit mode* and *Object mode*.

**Note:** What is selected using both these tools is affected by the *Limit Selection to visible* feature (available under the 3D viewport) in *Solid Viewport Shading Mode*.

For example,

- in solid shading mode and face selection mode, all faces *within* the selection area will be selected;
- whilst in the wireframe shading mode and face selection mode, only faces whose handle are within the selection area will be selected.

**Rectangular region (Border select)** *Border Select* is available in either *Edit mode* or *Object mode*. To activate the tool use the B. Use *Border Select* to select a group of objects by drawing a rectangle while holding down LMB. In doing this you will select all objects that lie within or touch this rectangle. If any object that was last active appears in the group it will become selected *and* active.



In (*Start*), *Border Select* has been activated and is indicated by showing a dotted cross-hair cursor. In (*Selecting*), the *selection region* is being chosen by drawing a rectangle with the LMB. The selection area is only covering the selection handles of three faces. Finally, by releasing LMB the selection is complete; see (*Complete*).

**Note:** Border select adds to the previous selection, so in order to select only the contents of the rectangle, deselect all with A first. In addition, you can use MMB while you draw the border to deselect all objects within the rectangle.

**Circular region** This selection tool is only available in *Edit mode* and can be activated with C. Once in this mode the cursor changes to a dashed cross-hair with a 2D circle surrounding it. The tool will operate on whatever the current select mode is. Clicking or dragging with the LMB, causing elements to be inside the circle will cause those elements to be selected.

You can enlarge or shrink the circle region using NumpadPlus and NumpadMinus, or the Wheel.

Table

2.2: After



(*Circle Region Select*) is an example of selecting edges while in *Edge Select Mode*. As soon as an edge intersects the circle the edge becomes selected. The tool is interactive such that edges are selected while the circle region is being dragged with the LMB.

If you want to deselect elements, either hold MMB or Alt-LMB and begin clicking or dragging again.

For *Faces* select mode, the circle must intersect the face indicators usually represented by small pixel squares; one at the center of each face.

To exit from this tool, click RMB, or press the Esc key.

**Lasso region** *Lasso select* is similar to *Border select* in that you select objects based on a region, except *Lasso* is a hand-drawn region that generally forms a circular/round-shaped form; kind of like a lasso.

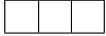
*Lasso* is available in either *Edit Mode* or *Object Mode*. To activate the tool use the Ctrl-LMB while dragging. The one difference between *Lasso* and *Border select* is that in *Object mode*, *Lasso* only selects objects where the lasso region intersects the objects' center.

To deselect, use `Ctrl-Shift-LMB` while dragging.

Table

2.3:

Complete



(*Lasso selection*) is an example of using the *Lasso* select tool in *Vertex Select Mode*.

**Additional Selection Tools** The select menu in edit mode contains additional tool for selecting components:

**(De)select All A** Select all or none of the mesh components.

**Invert Selection Ctrl-I** Selects all components that are not selected, and deselect currently selected components.

**More Ctrl-NumpadPlus** Propagates selection by adding components that are adjacent to selected elements.

**Less Ctrl-NumpadMinus** Deselects components that form the bounds of the current selection

### Advanced Selection

The select menu in edit mode contains additional tool for selecting components:

**Mirror** Select mesh items at the mirrored location.

**Linked** Selects all components that are connected to the current selection. (see [Select Linked](#))

**Random** Selects a random group of vertices, edges, or faces, based on a percentage value.

**Checker Deselect** Deselect alternating faces, to create a checker like pattern.

**Select Every N Number of Vertices** Selects vertices that are multiples of N.

**Sharp Edges** This tool selects all edges between two faces forming an angle greater than the angle option, Where an increasing angle selects sharper edges.

**Linked Flat Faces (Ctrl-Shift-Alt-F)** Select connected faces based on a threshold of the angle between them. This is useful for selecting faces that are planar.

**Non Manifold (Ctrl-Shift-Alt-M)** Selects the *non-manifold* geometry of a mesh. This entry is available when editing a mesh, in Vertex and Edge selection modes only. The *redo* panel provides several selection options:

**Extend** Lets you extend the current selection.

**Wire** Selects all the edges that don't belong to any face.

**Boundaries** Selects edges in boundaries and holes.

**Multiple Faces** Selects edges that belong to 3 or more faces.

**Non Contiguous** Selects edges that belong to exactly 2 faces with opposite normals.

**Vertices** Selects vertices that belong to *wire* and *multiple face* edges, isolated vertices, and vertices that belong to non adjoining faces.

**Interior Faces** Select faces where all edges have more than 2 faces.

**Side of Active** Selects all data on the mesh in a single axis

**Select Faces by Sides** Selects all faces that have a specified number of edges.

**Loose Geometry** Select all vertices or edges that do not form part of a face.

---

## Select Linked Reference

Mode: *Edit* mode

Menu: *Select* → *Linked*

Hotkey: `Ctrl-L`

---

Select parts of a mesh connected to already selected elements. This is often useful when a mesh has disconnected, overlapping parts, where isolating it any other way would be tedious.

To give more control, you can also enable delimiters so the selection is constrained by seams, sharp-edges, materials or UV islands.

---

**Hint:** You can also select linked data directly under the cursor, using the `L` shortcut to select or `Shift-L` to deselect linked. This works differently in that it uses the geometry under the cursor instead of the existing selection.

---

## Select Similar Reference

Mode: *Edit* mode

Menu: *Select* → *Similar...*

Hotkey: `Shift-G`

---

Select components that have similar attributes to the ones selected, based on a threshold that can be set in tool properties after activating the tool. Tool options change depending on the selection mode:

### Vertex Selection Mode:

**Normal** Selects all vertices that have normals pointing in similar directions to those currently selected.

**Amount of Adjacent Faces** Selects all vertices that have the same number of faces connected to them.

**Vertex Groups** Selects all vertices in the same [vertex group](#).

**Amount of connecting edges** Selects all vertices that have the same number of edges connected to them.

### Edge Selection Mode:

**Length** Selects all edges that have a similar length as those already selected.

**Direction** Selects all edges that have a similar direction (angle) as those already selected.

**Amount of Faces Around an Edge** Selects all edges that belong to the same number of faces.

**Face Angles** Selects all edges that are between two faces forming a similar angle, as with those already selected.

**Crease** Selects all edges that have a similar *Crease* value as those already selected. The *Crease* value is a setting used by the [Subsurf Modifier](#).

**Bevel** Selects all edges that have the same *Bevel Weight* as those already selected.

**Seam** Selects all edges that have the same *Seam* state as those already selected. *Seam* is a true/false setting used in [UV-texturing](#).

**Sharpness** Selects all edges that have the same *Sharp* state as those already selected. *Sharp* is a true/false setting (a flag) used by the [EdgeSplit Modifier](#).

#### Face Selection Mode:

**Material** Selects all faces that use the same material as those already selected.

**Image** Selects all faces that use the same UV-texture as those already selected (see [UV-texturing](#) pages).

**Area** Selects all faces that have a similar area as those already selected.

**Polygon Sides** Selects all faces that have the same number of edges.

**Perimeter** Selects all faces that have a similar perimeter as those already selected.

**Normal** Selects all faces that have a similar normal as those selected. This is a way to select faces that have the same orientation (angle).

**Co-planar** Selects all faces that are (nearly) in the same plane as those selected.

**Selecting Loops** You can easily select loops of components:

#### Edge Loops and Vertex Loops

##### Reference

Mode: *Edit* mode → *Vertex* or *Edge* select mode

Menu: *Select* → *Edge Loop* or *Mesh* → *Edges* → *Edge Loop*

Hotkey: `Alt-RMB` or `Ctrl-E` → *Edge Loop*

Holding `Alt` while selecting an edge selects a loop of edges that are connected in a line end to end, passing through the edge under the mouse pointer. Holding `Alt-Shift` while clicking adds to the current selection.

Edge loops can also be selected based on an existing edge selection, using either *Select* → *Edge Loop*, or the *Edge Loop Select* option of the *Edge Specials* menu (`Ctrl-E`).

**Note:** *Vertex* mode

In *Vertex* select mode, you can also select edge loops, by using the same hotkeys, *and clicking on the edges* (not on the vertices).

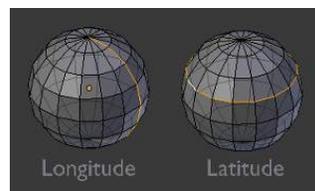


Fig. 2.154: Longitudinal and latitudinal edge loops.

The left sphere shows an edge that was selected longitudinally. Notice how the loop is open. This is because the algorithm hit the vertices at the poles and terminated because the vertices at the pole connect to more than four edges. However, the right sphere shows an edge that was selected latitudinally and has formed a closed loop. This is because the algorithm hit the first edge that it started with.

## Face Loops

### Reference

Mode: *Edit mode* → *Face* or *Vertex* select modes

Hotkey: *Alt-RMB*

In face select mode, holding *Alt* while selecting an **edge** selects a loop of faces that are connected in a line end to end, along their opposite edges.

In vertex select mode, the same can be accomplished by using *Ctrl-Alt* to select an edge, which selects the face loop implicitly.

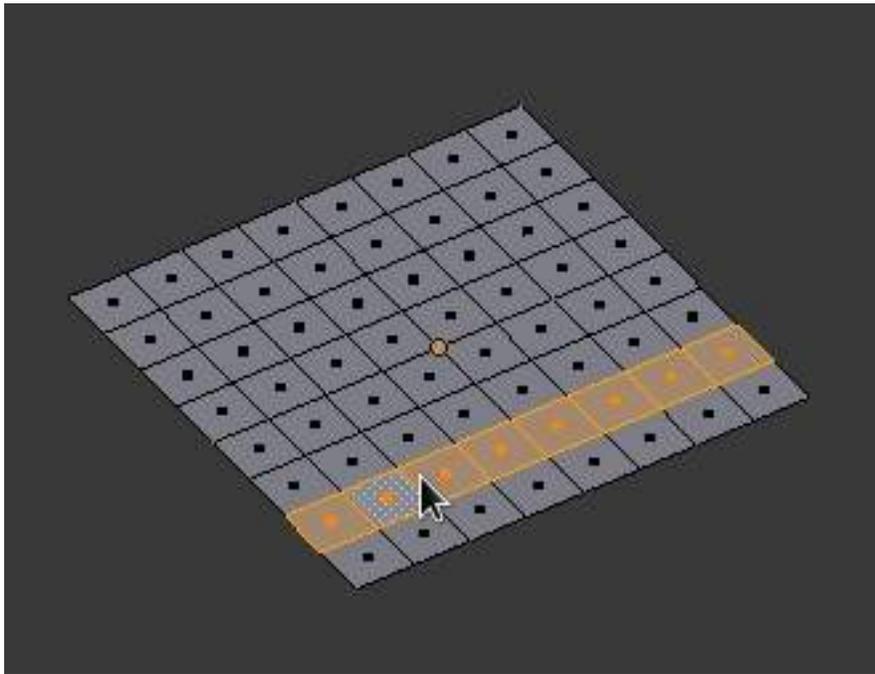


Fig. 2.155: Face loop selection.

This face loop was selected by clicking with *Alt-RMB* on an edge, in *face* select mode. The loop extends perpendicular from the edge that was selected.

A face loop can also be selected in *Vertex* select mode. Technically *Ctrl-Alt-RMB* will select an *Edge Ring*, however in *Vertex* select mode, selecting an *Edge Ring* implicitly selects a *Face Loop* since selecting opposite edges of a face implicitly selects the entire face.

## Edge Ring

### Reference

Mode: *Edit mode* → *Edge* select mode

Menu: *Select* → *Edge Ring* or *Mesh* → *Edges* → *Edge Ring*

Hotkey: *Ctrl-Alt-RMB* or *Ctrl-E* → *Select* → *Edge Ring*

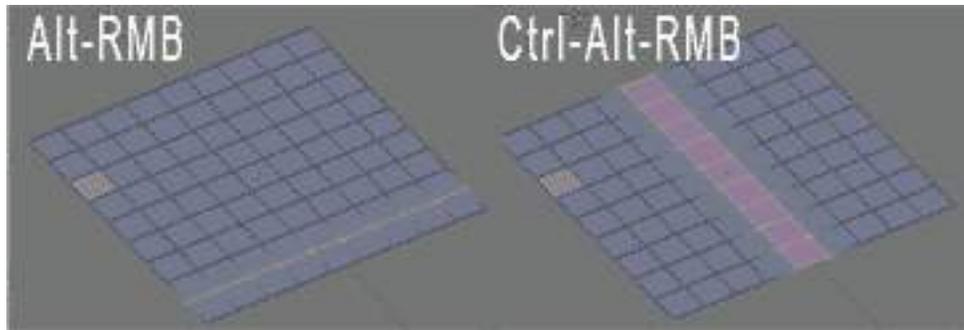


Fig. 2.156: Alt versus Ctrl-Alt in vertex select mode.

In *Edge* select mode, holding `Ctrl-Alt` while selecting an edge selects a sequence of edges that are not connected, but on opposite sides to each other continuing along a *face loop*.

As with edge loops, you can also select edge rings based on current selection, using either *Select → Edge Ring*, or the *Edge Ring Select* option of the *Edge Specials* menu (`Ctrl-E`).

**Note:** *Vertex* mode

In *Vertex* select mode, you can use the same hotkeys when *clicking on the edges* (not on the vertices), but this will directly select the corresponding face loop...

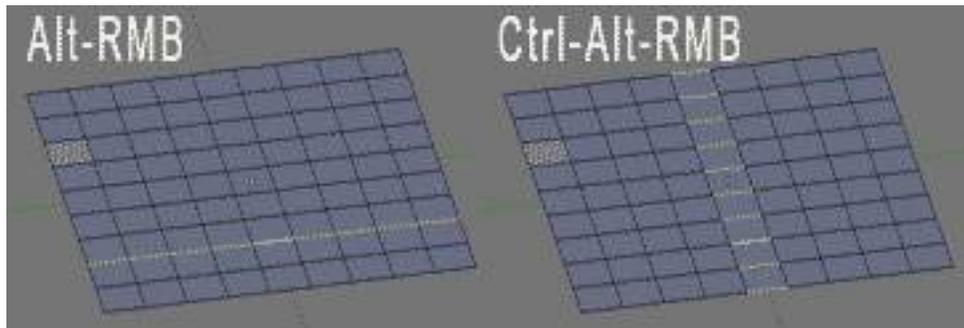


Fig. 2.157: A selected edge loop, and a selected edge ring.

In (*A selected edge loop, and a selected edge ring*), the same edge was clicked on but two different “groups of edges” were selected, based on the different commands. One is based on edges during computation and the other is based on faces.

### Path Selection

#### Reference

Mode: *Edit* mode

Hotkey: `Ctrl-RMB` and the menu item *Select → Shortest Path*

Selects all geometry along the shortest path from the active vertex/edge/face to the one which was selected.

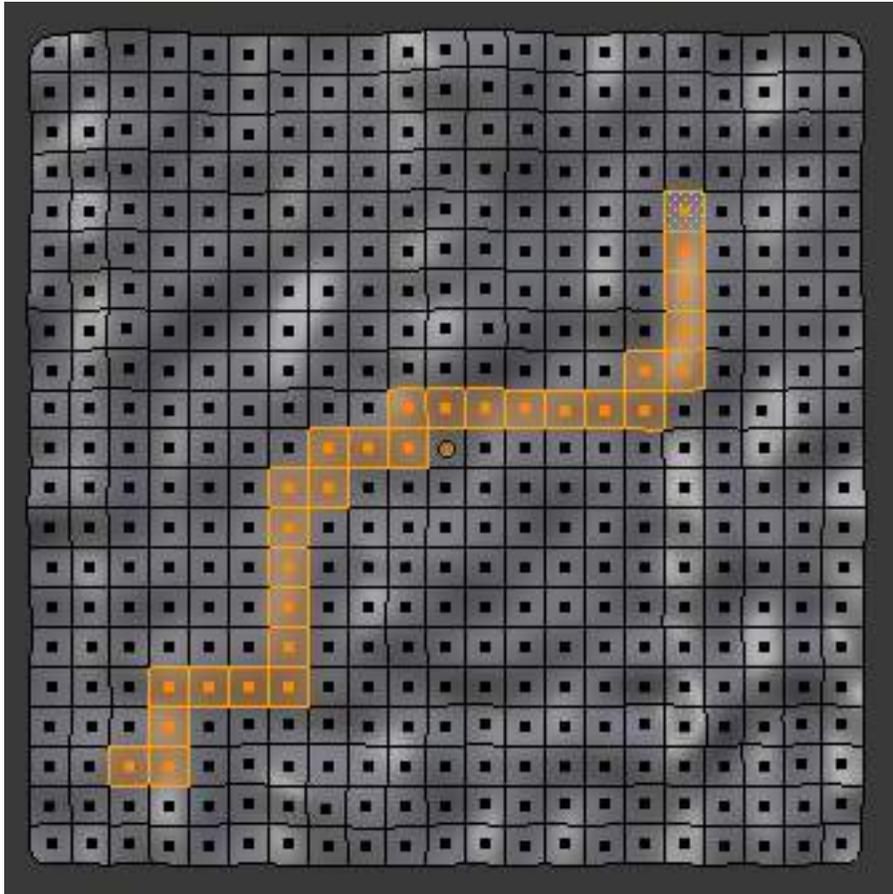


Fig. 2.158: Select a face or vertex path with `Ctrl-RMB`

---

## Loop Inner-Region Reference

Mode: *Edit mode* → *Edge select mode*

Menu: *Select* → *Select Loop Inner-Region* or *Mesh* → *Edges* → *Select Loop Inner-Region*

Hotkey: *Ctrl-E* → *Select Loop Inner-Region*

---

*Select Loop Inner-Region* selects all edges that are inside a closed loop of edges. While it is possible to use this operator in *Vertex* and *Face* selection modes, results may be unexpected. Note that if the selected loop of edges is not closed, then all connected edges on the mesh will be considered inside the loop.

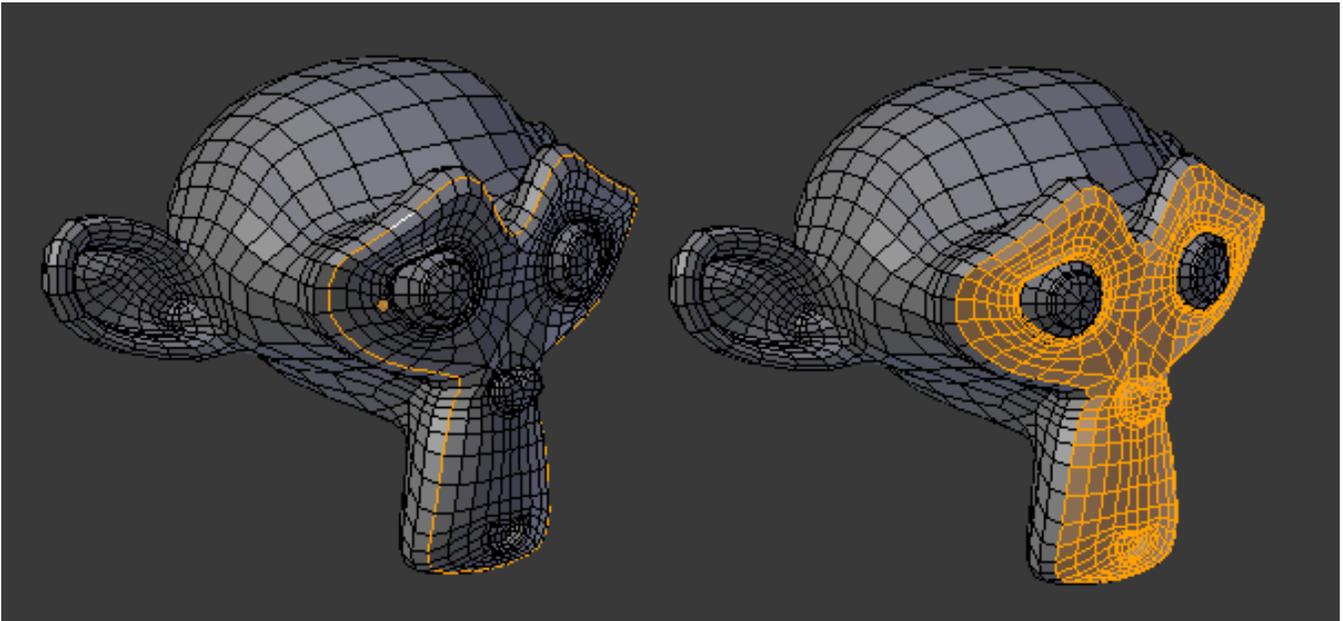


Fig. 2.159: Loop to Region.

---

## Boundary Loop Reference

Mode: *Edit mode* → *Edge select mode*

Menu: *Select* → *Select Boundary Loop* or *Mesh* → *Edges* → *Select Boundary Loop*

Hotkey: *Ctrl-E* → *Select Boundary Loop*

---

*Select Boundary Loop* does the opposite of *Select Loop Inner-Region*, based on all regions currently selected, it selects only the edges at the border of these regions. It can operate in any select mode, but will always switch to *Edge select mode* when run.

All this is much more simple to illustrates with examples:

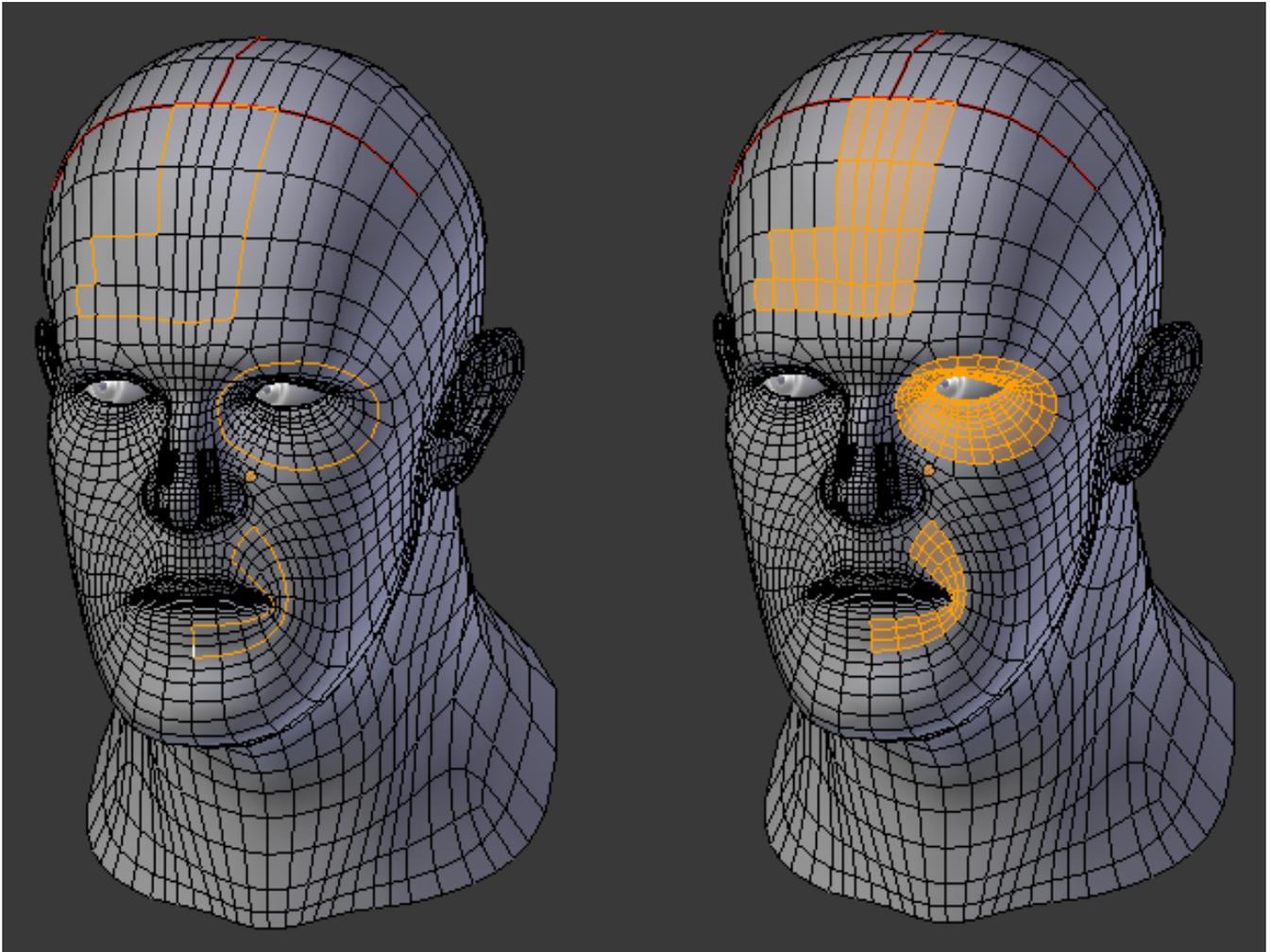


Fig. 2.160: This tool handles multiple loops fine, as you can see.

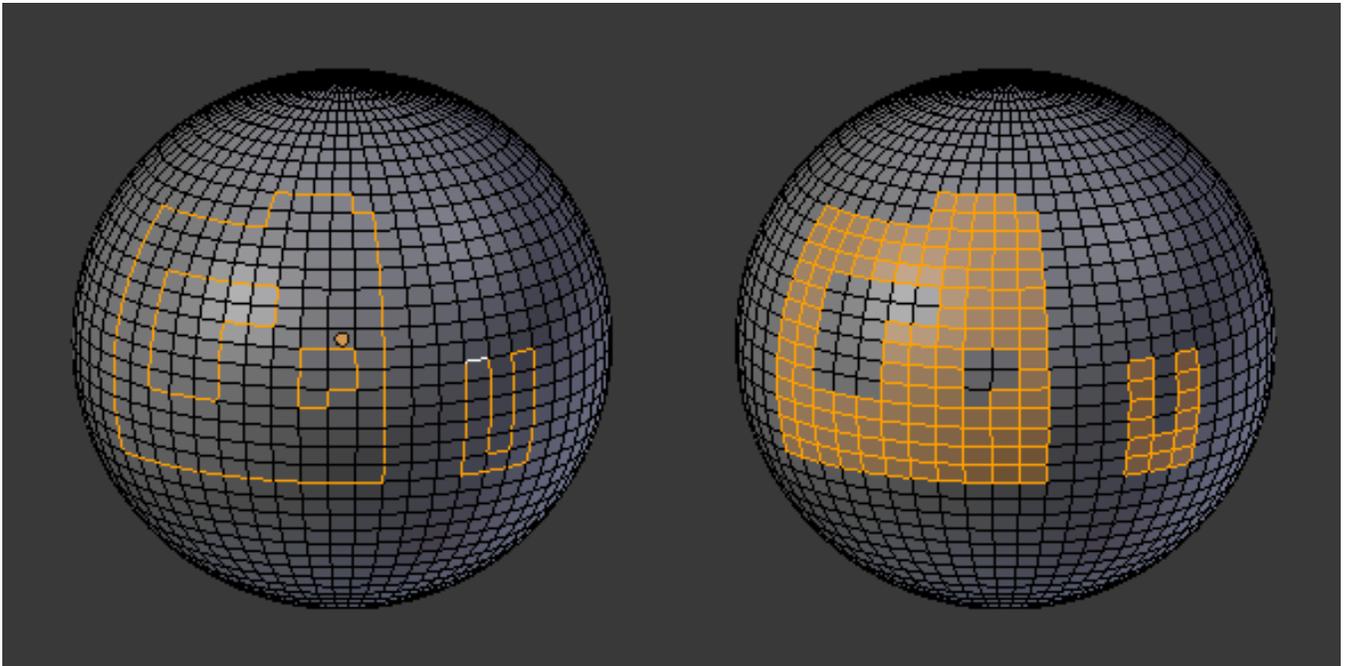


Fig. 2.161: This tool handles “holes” just fine as well.

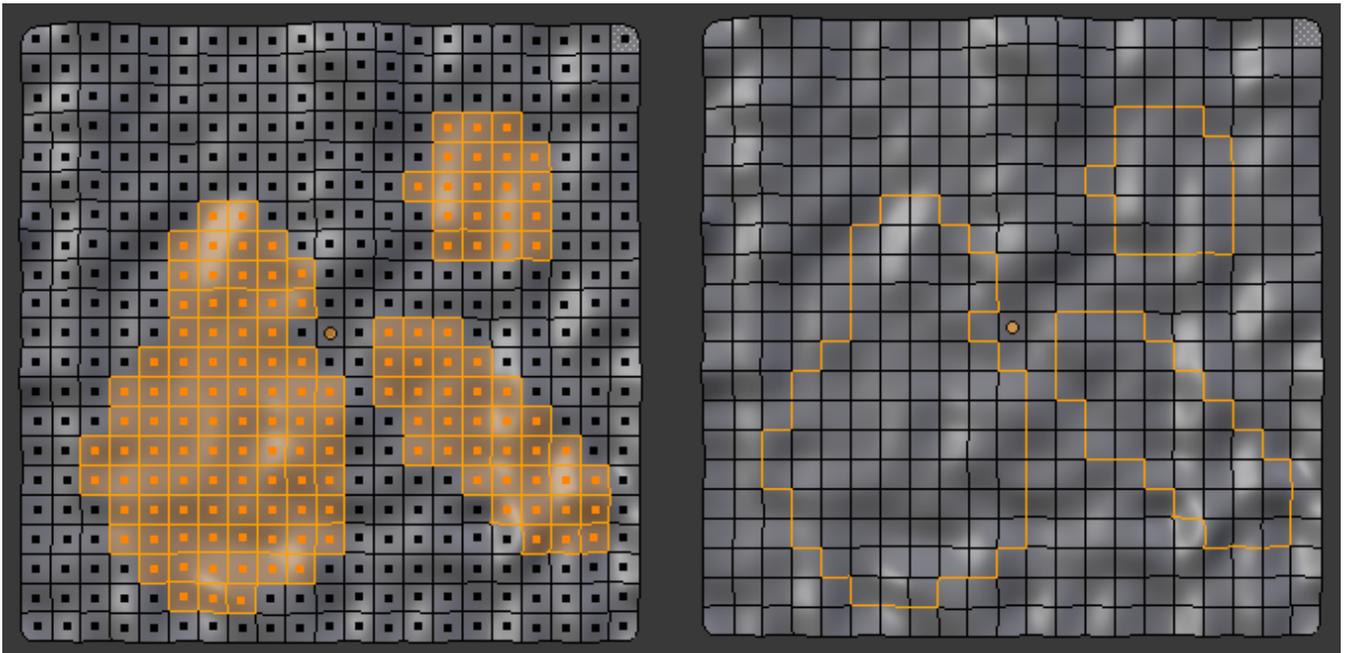


Fig. 2.162: Select Boundary Loop does the opposite and forces into Edge Select Mode



Fig. 2.163: Buttons for the selection modes

## Selecting Edges

Edges can be selected in much the same way as vertices and faces - by right-clicking them while Edge Select Mode is activated. Pressing `Shift` while clicking will add/subtract to the existing selection.

### Edge Loops

#### Reference

Mode: Edit Mode (Mesh)

Menu: *Select* → *Edge Loop*

Hotkey: `Alt-RMB` - or `Shift-Alt-RMB` for modifying existing selection

---

Edge loops can be selected by first selecting an edge (vertex or edge selection mode), and then going to *Select* → *Edge Loop*. The shortcut `Alt-RMB` on an edge (either vertex or edge select mode) is a quicker and more powerful way of doing so. More powerful, because you can add/remove loops from an existing selection if you press `Shift` too.

Note, that if you want to select a loop while being in vertex select mode, you still have to perform the shortcut on an edge - while you, for just selecting vertices, would rightclick on a vertex.

---

**Note:** `Alt` on Linux

`Alt` is on some Linux distros caught by the windows manager. If you see the above shortcut not working, make sure that blender can properly recognize the usage of `Alt`.

---

### Edge Rings

#### Reference

Mode: Edit Mode (Mesh)

Menu: *Select* → *Edge Ring*

Hotkey: `Alt-Ctrl-RMB` - or `Shift-Alt-Ctrl-RMB` for modifying existing selection

---

Edge Rings are selected similarly. Based on the selection of an edge go to *Select* → *Edge Ring*. Or use `Alt-Ctrl-RMB` on an edge.

---

**Note:** Convert selection to whole faces

If the edge ring selection happened in Edge Select Mode, switching to Face Select Mode will erase the selection.

This is because none of those faces had all its (four) edges selected, just two of them.

Instead of selecting the missing edges manually or by using `Shift-Alt-RMB` twice, it is easier to first switch to Vertex Select Mode, which will kind of “flood” the selection. A subsequent switch to Face Select Mode will then properly select the faces.

---

## Selecting Faces

To select parts of a mesh face-wise, you have to switch to Face Select Mode. Do this by clicking the button shown above, or press `Ctrl-Tab` to spawn a menu. The selection works as usual with `RMB`; to add/remove to an existing selection, additionally press `Shift`

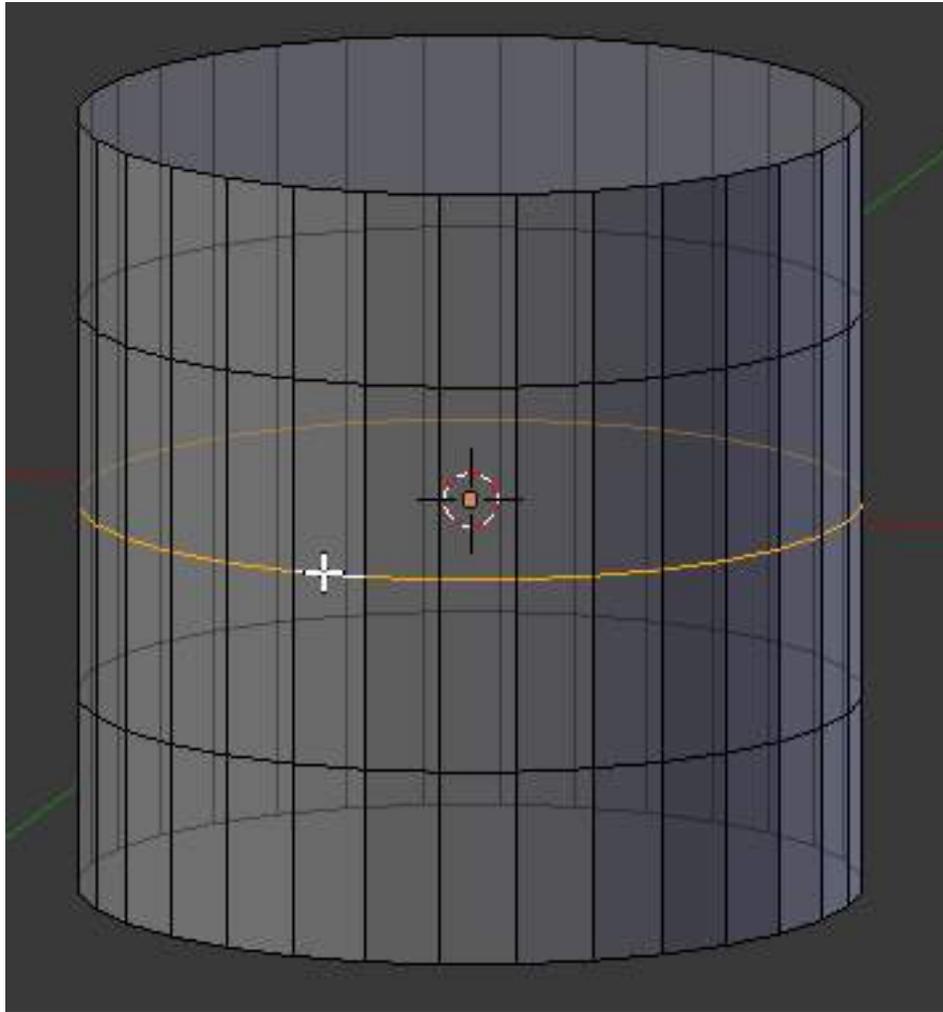


Fig. 2.164: An edge loop

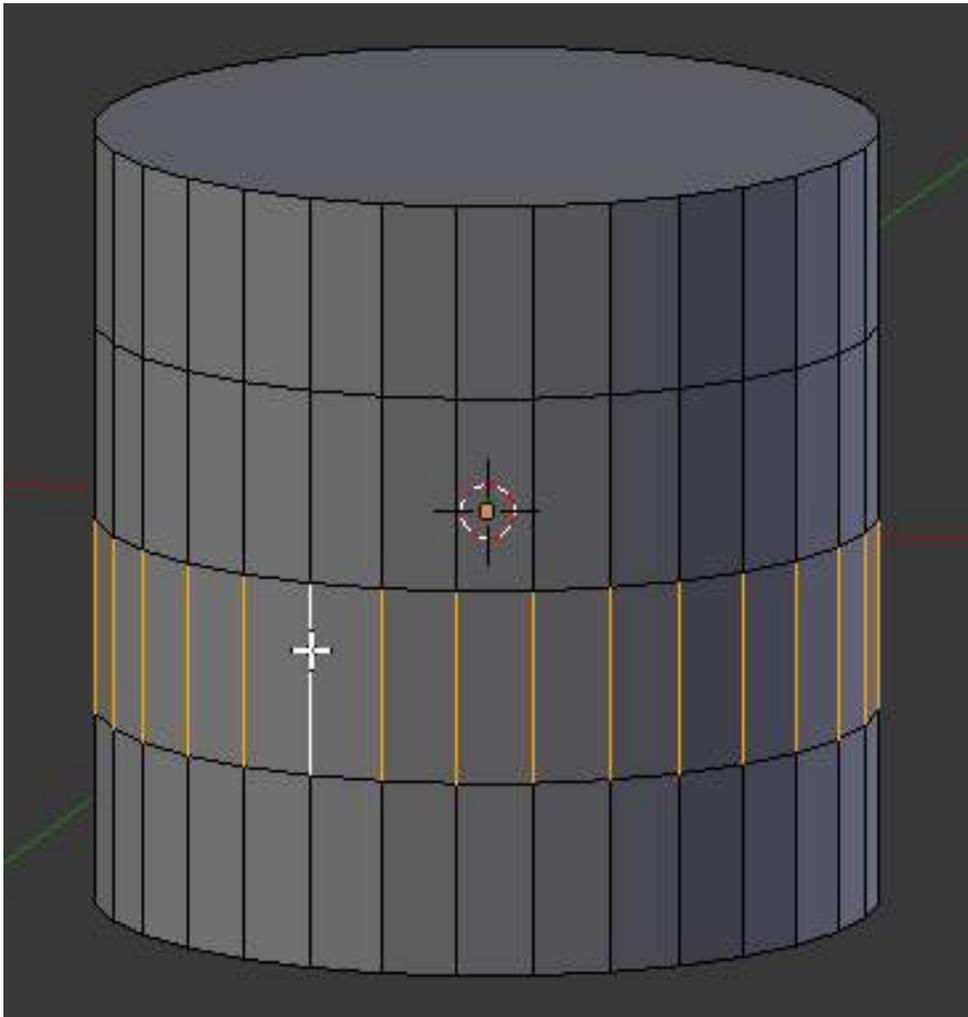


Fig. 2.165: An Edge Ring



Fig. 2.166: Activated the Face Select Mode

## Face Loops Reference

Mode: Edit Mode (Mesh)

Hotkey: `Alt-RMB` - or `Shift-Alt-RMB` for modifying existing selection

Face Loops are pretty much the same as Edge Rings. If you want to select a Face Loop, there is no menu entry that works based on a selected face. Using `Select → Edge Ring` would select a “cross” with the prior selected face as the middle. If you want to avoid switching to Edge Select Mode to select a Face Loop, use the `Alt-RMB` shortcut.

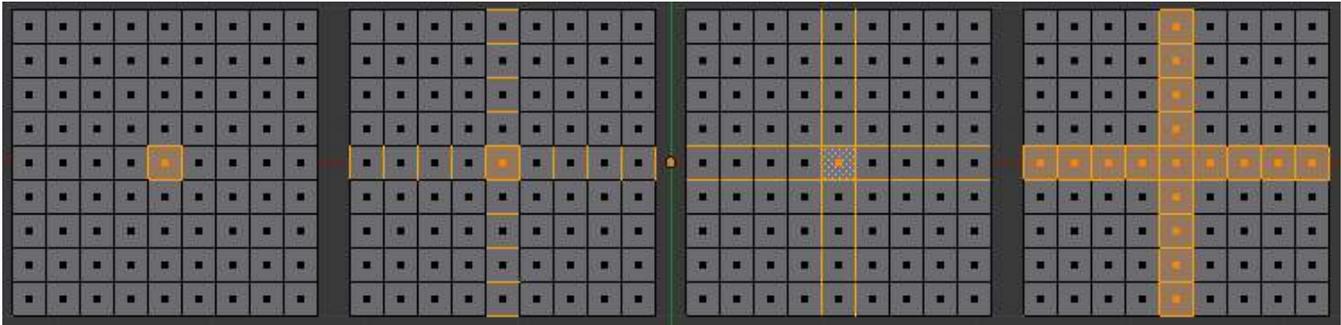


Fig. 2.167: Different Loopselect Operations on a grid in Face Select Mode

- Just the selected face.
- Select the face, then `Select → Edge Ring`. See, how Blender selects edges, even if being in Face Select Mode. If these edges are desired and you want to work on them, switch in Edge Select Mode. Switching to Vertex Select Mode would flood the selection and leave you with the 4th image as result, after going back to Face Select Mode.
- Select the face, the `Select → Edge Loop`. As in the example above, Blender pretends to be in Edge Select Mode and takes the four edges of the selected face as base for the selection operation.
- This selection was created by `Alt-RMB` on the left edge of the center face, followed by twice `Shift-Alt-RMB` on the top edge of the center face. Two times, because the first click will remove the selected face loop (in this case, just the original selected face), while the second click will add the whole vertical running loop to the selection, creating the cross.

**Ngons in Face Select Mode** As already known, faces are marked with a little square dot in the middle of the face. With ngons that can lead in certain cases to a confusing display. The example shows the center dot of the U-shaped ngon being inside of the oblong face inside the “U”. It is not easy to say which dot belongs to which face (the orange dot in the image is the object center). Luckily, you don’t need to care much - because to select a face, you don’t have to click the center dot, but the face itself.

**Tip:** Face selection

*To select a face:* Click the face, not the dot!

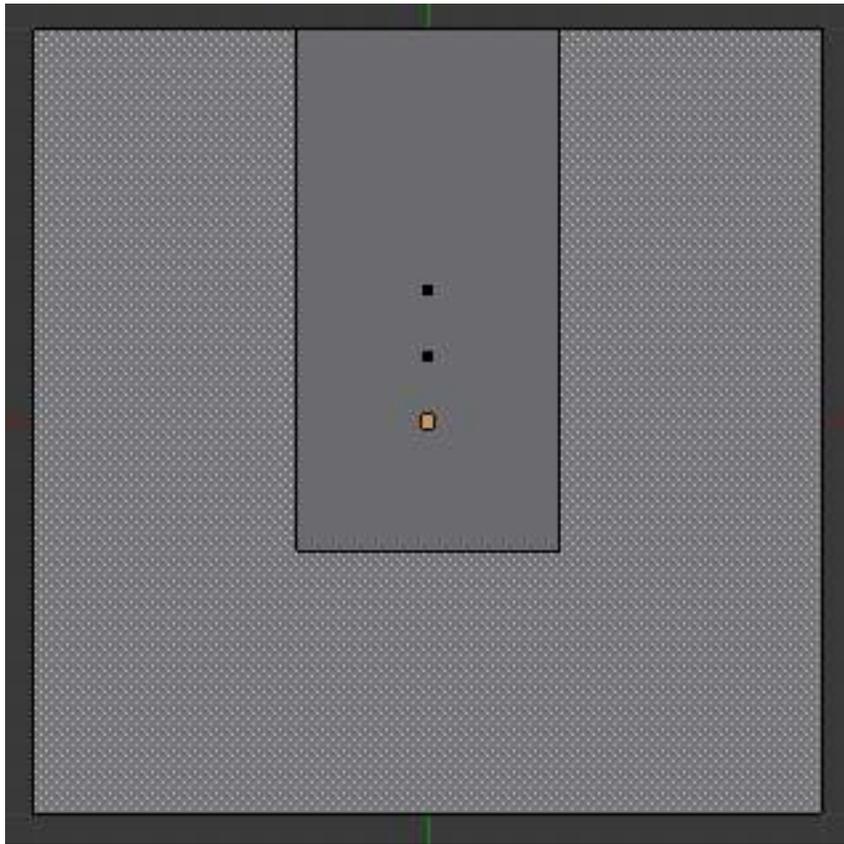


Fig. 2.168: Ngon-Face having its center dot inside another face

## Editing

### Mesh Editing

Blender provides a variety of tools for editing meshes. These are available through the *Mesh Tools* palette, the Mesh menu in the 3d view header, and context menus in the 3d view, as well as individual shortcut keys.

Note that all the “transform precision/snap” keys (Ctrl and/or Shift) work also for all these advanced operations... However, most of them do not have [axis locking](#) possibilities, and some of them do not take into account [pivot point](#) and/or [transform orientation](#) either.

These transform tools are available in the *Transform* section of the *Mesh* menu in the menu bar. Note that some of these can also be used on other editable objects, like curves, surfaces, and lattices.

**Types of Tools** The mesh tools are found in various places, and available through shortcuts as well.

<p><b>Transform and Deform tools:</b></p> <ul style="list-style-type: none"> <li>• Translate</li> <li>• Rotate</li> <li>• Scale</li> <li>• Mirror</li> <li>• Shrink/Flatten/Along Normal</li> <li>• Push/Pull</li> <li>• To Sphere</li> <li>• Shear</li> <li>• Warp</li> <li>• Edge Slide</li> <li>• Vertex Slide</li> <li>• Noise</li> <li>• Smooth Vertex</li> <li>• Rotate Edge</li> </ul> <p><b>Merge and Remove tools:</b></p> <ul style="list-style-type: none"> <li>• Delete</li> <li>• Dissolve</li> <li>• Merge</li> <li>• Auto-Merge</li> <li>• Remove Doubles</li> <li>• Tris to Quads</li> <li>• Unsubdivide</li> </ul>	<p><b>Add and Divide tools:</b></p> <ul style="list-style-type: none"> <li>• Make Edge/Face</li> <li>• Fill</li> <li>• Beauty Fill</li> <li>• Solidify</li> <li>• Quads to Tris</li> <li>• Extrude Region</li> <li>• Extrude Individual</li> <li>• Subdivide</li> <li>• Loop Cut/Slide</li> <li>• Knife tool</li> <li>• Vertex connect</li> <li>• Duplicate</li> <li>• Spin</li> <li>• Screw</li> <li>• Symmetrize</li> <li>• Inset</li> <li>• Bevel</li> <li>• Wireframe</li> </ul> <p><b>Separate tools:</b></p> <ul style="list-style-type: none"> <li>• Rip</li> <li>• Rip fill</li> <li>• Split</li> <li>• Separate</li> <li>• Edge Split</li> </ul>
---	---

### Accessing Mesh Tools

**Mesh Tools Palette** When you select a mesh and Tab into edit mode, the *Tool Shelf* changes from *Object Tools* to *Mesh Tools*. These are only some of the mesh editing tools.

**Menus** The *Mesh* is located in the Header bar. Some of the menus can be accessed with shortcuts: Ctrl-F brings up the Face tool menu Ctrl-E brings up the Edge tool menu Ctrl-V brings up the Vertex tool menu

## Basics

**Basic Mesh Editing** In this section we explain how to do basic editing on a mesh.

- Translation, Rotation, Scale
- Adding Elements
- Deleting Elements
- Creating Faces and Edges
- Mirror editing

---

## Translation, Rotation, Scale Reference

Mode: *Edit* mode

Panel: *Mesh Tools* (*Editing* context)

Menu: *Mesh* → *Transform* → *Grab/Move, Rotate, Scale, ...*

Hotkey: *G / R / S*

---

Once you have a selection of one or more elements, you can grab/move (G), rotate (R) or scale (S) them, like many other things in Blender, as described in the [Manipulation in 3D Space](#) section.

To move, rotate and scale selected components, either use the *Translate*, *Rotate*, and *Scale* buttons, the [transform manipulators](#), or the shortcuts:

G, R, and S respectively. After moving a selection, the options in the Tool Shelf allow you to fine-tune your changes, limit the effect to certain axes, turn proportional editing on and off, etc.

Of course, when you move an element of a given type (e.g. an edge), you also modify the implicitly related elements of other kinds (e.g. vertices and faces).

You also have in *Edit* mode an extra option when using these basic manipulations: the [proportional editing](#).

This page is being developed right now, follow this link to see the page while it's being constructed.

**Adding** Object mode Adding object - menu, manual selection - is under last added object... or shortcut *Shift-A*.

Edit mode: adding object → addin mesh, e.g. final merging into one object.

center for adding: X 3D curson. If you need to center cursor. just press *Shift-C*, or edit in the N tools menu cursor coordinates x,y,z.

**Deleting and Merging** These tools can be used to remove components.

### Delete

**Delete (X or Delete)** Deletes selected vertices, edges, or faces. This operation can also be limited to:

**Vertices** Delete all vertices in current selection, removing any faces or edges they are connected to.

**Edges** Deletes any edges in the current selection. Removes any faces that the edge shares with it.

**Faces** Removes any faces in current selection.

**Only Edges & Faces** Limits the operation to only selected edges and adjacent faces.

**Only faces** Removes faces, but edges within face selection are retained.

**Edge Collapse** Collapses edges into single vertices. This can be used to remove a loop of faces.

**Edge Loop** Deletes an edge loop. If the current selection is not an edge loop, this operation does nothing.

**Dissolve** Dissolve operations are also accessed from the delete menu. Instead of removing the geometry, which may leave holes that you have to fill in again, dissolve will remove the geometry and fill in the surrounding geometry.

**Dissolve** Removes selected geometry, but keeps surface closed, effectively turning the selection into a single n-gon. Dissolve works slightly different based on if you have edges, faces or vertices selected. You can add detail where you need it, or quickly remove it where you don't.

**Limited Dissolve** Limited Dissolve reduces detail on planar faces and linear edges with an adjustable angle threshold.

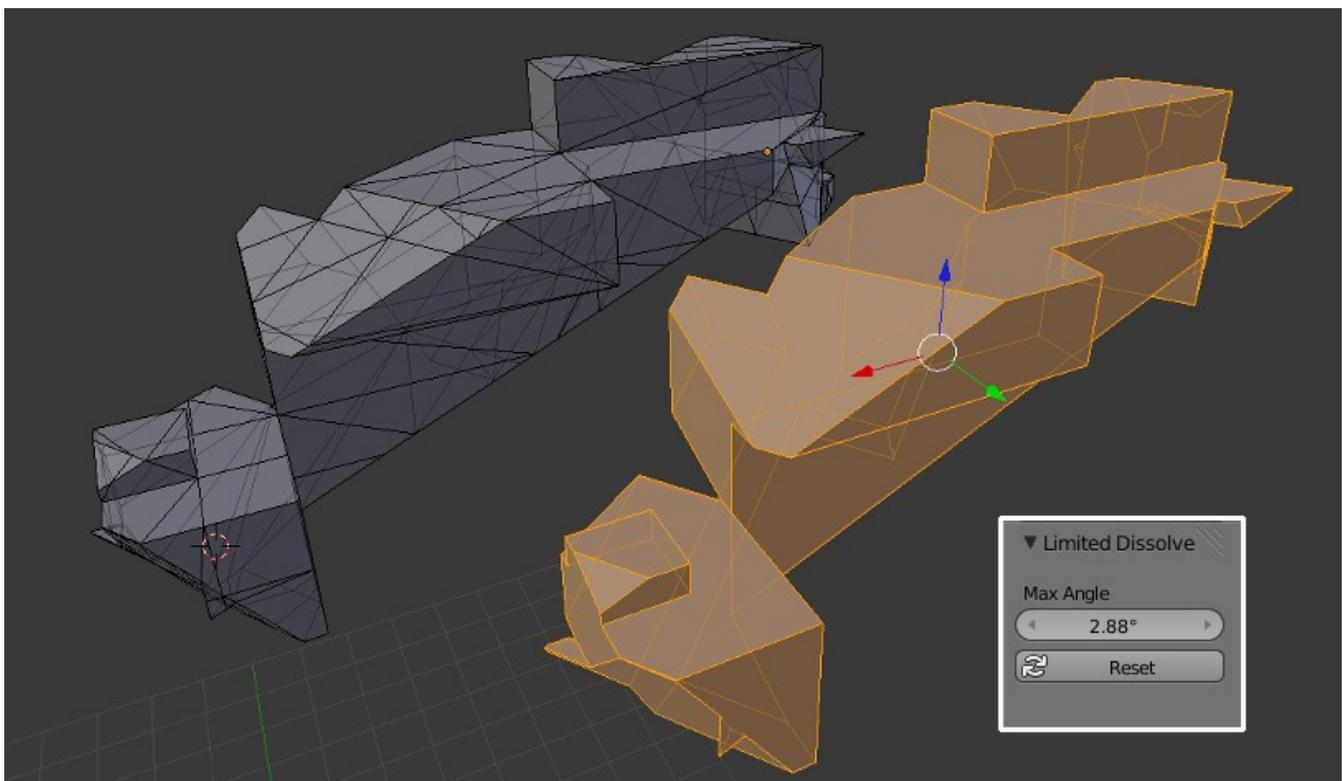


Fig. 2.169: Example showing the how Limited Dissolve can be used.

**Face Split - dissolve option.** When dissolving vertices into surrounding faces, you can often end up with very large, uneven ngons. The face split option limits dissolve to only use the corners of the faces connected to the vertex.

**Convert Triangles to Quads** *Tris to Quads* **Alt-J** This takes adjacent tris and removes the shared edge to create a quad. This tool can be performed on a selection of multiple triangles.

This same action can be done on a selection of just 2 tris, by selecting them and using the shortcut **F**, to create a face.

**Unsubdivide**

**Reference**

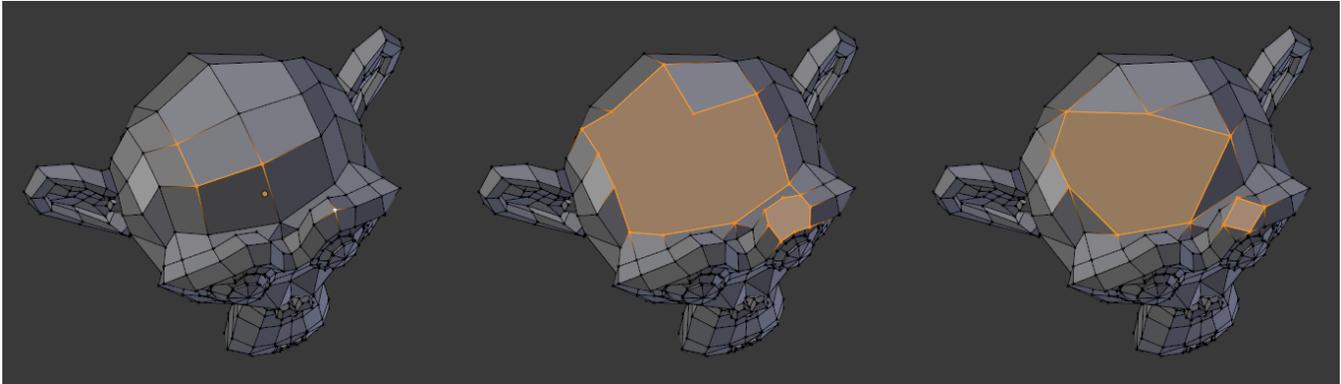


Fig. 2.170: Dissolve Face Split option. Left - the input, middle - regular dissolve, right - Face Split enabled

Mode: *Edit* mode

Menu: *Mesh* → *Edges* → *Unsubdivide*

Unsubdivide functions as the reverse of subdivide by attempting to remove edges that were the result of a subdivide operation. If additional editing has been done after the subdivide operation, unexpected results may occur.

**Iterations** How many subdivisions to remove.

## Merging

### Merging Vertices

#### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Vertices* → *Merge...*, *Specials* → *Merge* or *Vertex Specials* → *Merge*

Hotkey: `Alt-M`

This tool allows you to merge all selected vertices into an unique one, deleting all others. You can choose the location of the surviving vertex in the menu this tool pops up before executing:

**At First** Only available in *Vertex* select mode, it will place the remaining vertex at the location of the first one selected.

**At Last** Only available in *Vertex* select mode, it will place the remaining vertex at the location of the last one selected (the active one).

**At Center** Available in all select modes, it will place the remaining vertex at the center of the selection.

**At Cursor** Available in all select modes, it will place the remaining vertex at the 3D Cursor.

**Collapse** This is a special option, as it might let “live” more than one vertex. In fact, you will have as many remaining vertices as you had “islands” of selection (i.e. groups of linked selected vertices). The remaining vertices will be positioned at the center of their respective “islands”. It is also available *via* the *Mesh* → *Edges* → *Collapse* menu option...

Merging vertices of course also deletes some edges and faces. But Blender will do everything it can to preserve edges and faces only partly involved in the reunion.

---

## AutoMerge Editing Reference

Mode: *Edit* mode

Menu: *Mesh* → *AutoMerge Editing*

---

The *Mesh* menu as a related toggle option: *AutoMerge Editing*. When enabled, as soon as a vertex moves closer to another one than the *Limit* setting (*Mesh Tools* panel, see below), they are automatically merged.

---

## Remove Doubles Reference

Mode: *Edit* mode

Panel: *Editing* context → *Mesh Tools*

Menu: *Mesh* → *Vertices* → *Remove Doubles*, *Specials* → *Remove Doubles* or *Vertex Specials* → *Remove Doubles*

Hotkey:  $\bar{w}$ , *Remove Doubles*

---

Remove Doubles is a useful tool to simplify a mesh by merging vertices that are closer than a specified distance to each other. An alternate way to simplify a mesh is to use the [Decimate modifier](#).

**Merge Distance** Sets the distance threshold for merging vertices, in Blender units.

**Unselected** Allows vertices in a selection to be merged with unselected vertices. When disabled, selected vertices will only be merged with other selected ones.

---

## Make Edge/Face Reference

Mode: *Edit* mode

Menu: *Mesh* → *Faces* → *Make Face/Edge*

Hotkey:  $\bar{F}$

---

This is a context sensitive tool which creates geometry by filling in the selection. When only 2 vertices are selected it will create an edge, otherwise it will create faces.

The typical use case is to select vertices and press  $\bar{F}$ , however Blender also supports creating faces from different selections to help quickly build up geometry.

The following methods are used automatically depending on the context.

---

Isolated vertices.




---

Isolated edges



N-gon from edges: *When there are many edges Blender will make an ngon, note that this doesn't support holes, to support holes you need to use the [Fill Faces](#) tool.*



Mixed vertices/edges: *existing edges are used to make the face as well as an extra vertex.*



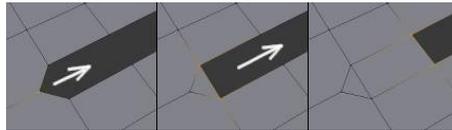
Edge-Net: *sometimes you may have many connected edges without interior faces.*



Point Cloud: *when there are many isolated vertices, Blender will calculate the edges for an n-gon.*



Single Vertex Selection: *with a single vertex selected on a boundary, the face will be created along the boundary, this saves manually selecting the other 2 vertices. Notice this tool can run multiple times to continue creating faces.'*



**Further Reading** For other ways to create faces see:

- [Fill](#)
- [Grid Fill](#)
- [Bridge Edge Loops](#)

## Mirror Editing

### X-Mirror Reference

---

Mode: *Edit mode*

Panel: *Mesh Options* → *X-mirror*

---

The *X-mirror* option of the *Mesh Options* panel allows you edit both “sides” of your mesh in a single action. When you transform an element (vertex, edge or face), if there is its *exact X-mirrored counterpart* (in local space), it will be transformed accordingly, *through a symmetry along the local X axis*.

**Topology Mirror** The *Topology Mirror* option is available in the *3D View Editor* → *Toolshelf Region* → *Mesh Options Panel* whilst in *Edit Mode*

For *Topology Mirror* to work the *X Mirror* option must be enabled.

When using the *X Mirror* option to work on mirrored Mesh Geometry the vertices that are mirrored must be perfectly placed. If they are not exactly positioned in their mirror locations then *X Mirror* will not treat those vertices as mirrored. This can be annoying because often the out of position vertices are only very slightly out of position.

*Topology Mirror* tries to solve this problem by determining which vertices are mirrored vertices not only by using their positions but also by looking at how those vertices are related to others in the Mesh Geometry. It looks at the overall Mesh Geometry topology to determine if particular vertices will be treated as mirrored. The effect of this is that mirrored vertices can be non-symmetrical and yet still be treated as mirrored when *X Mirror* and *Topology Mirror* are both active.

Note that *Topology Mirror* functionality will work more reliably on Mesh Geometry which is more detailed. If you use very simple Mesh Geometry such as a Cube or UV Sphere for example the *Topology Mirror* option will often not work.

For an example of how to use *Topology Mirror* open up a new Blender scene, then delete Blender's default cube and add a Monkey Object to the 3D Viewport.

Press the TAB Key to put the Monkey Object into *Edit Mode*.

With the *X Mirror* option disabled move one of the Monkey Object's vertices slightly.

Then Turn *X Mirror* option on again but leave *Topology Mirror* disabled

If you now move that vertice again *X Mirror* will not work and the mirrored vertices will not be altered.

If you then enable *Topology Mirror* and move the same vertices again, then *X Mirror* should still mirror the other vertice, even though they are not perfectly positioned.

**Mirror Modifier** The conditions for X-mirror to work are quite strict, which can make it difficult to use. To have an exact mirrored version of a (half) mesh, its easier and simpler to use the [Mirror modifier](#)

## Snap to Symmetry

### Reference

Mode: *Edit mode*

Menu: *Mesh* → *Snap to Symmetry*

The *Snap to Symmetry* tool works on meshes which are mostly symmetrical but have vertices which have been moved enough that Blender does not detect them as mirrored (when x-mirror option is enable for example).

This can be caused by accident when editing without x-mirror enabled. Sometimes models imported from other applications are asymmetrical enough that mirror fails too.

**Direction** Specify the axis and direction to snap. Can be any of the 3 axes, and either positive to negative, or negative to positive.

**Threshold** Specify the search radius to use when finding matching vertices.

**Factor** Support for blending mirrored locations from one side to the other (0.5 is an equal mix of both).

**Center** Snap vertices in the center axis to zero.

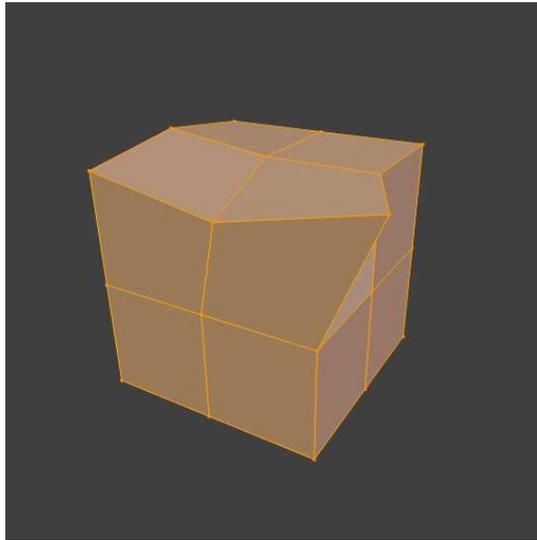


Fig. 2.195: Before Snap to Symmetry

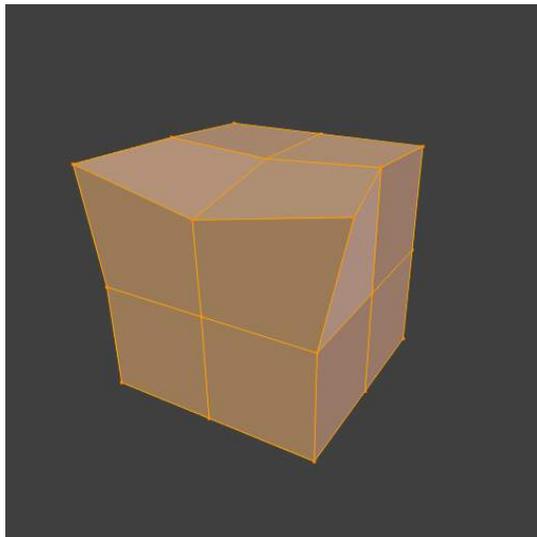


Fig. 2.196: After Snap to Symmetry

## Symmetrize Mesh Reference

Mode: *Edit* mode

Menu: *Mesh* → *Symmetrize*

The *Symmetrize* tool is a quick way to make a mesh symmetrical. *Symmetrize* works by cutting the mesh at the pivot point of the object, and mirroring over the geometry in the specified axis, and merges the two halves together (if they are connected)

**Direction** Specify the axis and direction of the effect. Can be any of the 3 axes, and either positive to negative, or negative to positive.

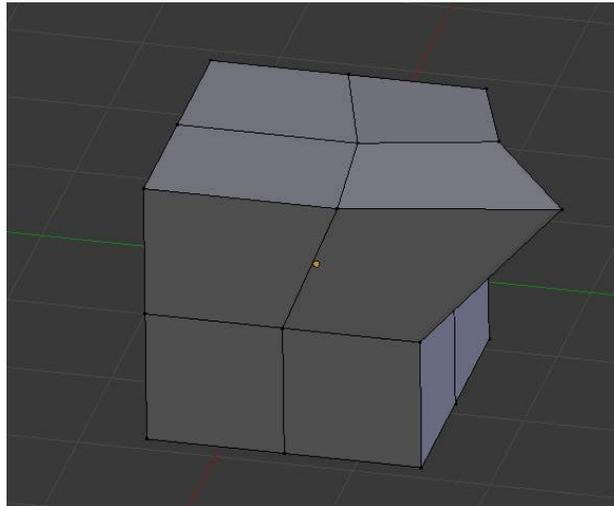


Fig. 2.197: Mesh before Symmetrize

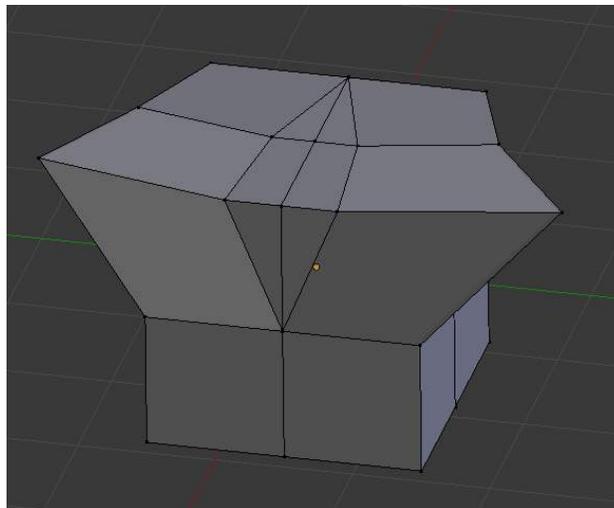


Fig. 2.198: Mesh after Symmetrize

**Mirroring Geometry** See [Mirror](#) for information on mirroring, which allows you to flip geometry across an axis

---

## Vertex Tools

This page covers many of the tools in the *Mesh* → *Vertices* menu. These are tools that work primarily on vertex selections, however, some also work with edge or face selections.

### Merging

---

#### Merging Vertices Reference

Mode: *Edit* mode

Menu: *Mesh* → *Vertices* → *Merge...*, *Specials* → *Merge* or *Vertex Specials* → *Merge*

Hotkey: Alt-M

---

This tool allows you to merge all selected vertices to an unique one, deleting all others. You can choose the location of the surviving vertex in the menu this tool pops up before executing:

**At First** Only available in *Vertex* select mode, it will place the remaining vertex at the location of the first one selected.

**At Last** Only available in *Vertex* select mode, it will place the remaining vertex at the location of the last one selected (the active one).

**At Center** Available in all select modes, it will place the remaining vertex at the center of the selection.

**At Cursor** Available in all select modes, it will place the remaining vertex at the 3D Cursor.

**Collapse** This is a special option, as it might let “live” more than one vertex. In fact, you will have as much remaining vertices as you had “islands” of selection (i.e. groups of linked selected vertices). The remaining vertices will be positioned at the center of their respective “islands”. It is also available *via* the *Mesh* → *Edges* → *Collapse* menu option...

Merging vertices of course also deletes some edges and faces. But Blender will do everything it can to preserve edges and faces only partly involved in the reunion.

---

#### AutoMerge Editing Reference

Mode: *Edit* mode

Menu: *Mesh* → *AutoMerge Editing*

---

The *Mesh* menu as a related toggle option: *AutoMerge Editing*. When enabled, as soon as a vertex moves closer to another one than the *Limit* setting (*Mesh Tools* panel, see below), they are automatically merged.

---

#### Remove Doubles Reference

Mode: *Edit* mode

Panel: *Editing* context → *Mesh Tools*

Menu: *Mesh* → *Vertices* → *Remove Doubles*, *Specials* → *Remove Doubles* or *Vertex Specials* → *Remove Doubles*

---

Hotkey: *[W]* → *Remove Doubles* or *Mesh* → *Vertices* → *Remove doubles*

Remove Doubles is a useful tool to simplify a mesh by merging vertices that are closer than a specified distance to each other. An alternate way to simplify a mesh is to use the [Decimate modifier](#).

**Merge Distance** Sets the distance threshold for merging vertices, in Blender units.

**Unselected** Allows vertices in selection to be merged with unselected vertices. When disabled, selected vertices will only be merged with other selected ones.

## Separating

### Rip

#### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Vertices* → *Rip*

Hotkey: *∇*

Rip creates a “hole” into a mesh by making a copy of selected vertices and edges, still linked to the neighbor non-selected vertices, so that the new edges are borders of the faces on one side, and the old ones, borders of the faces of the other side of the rip.

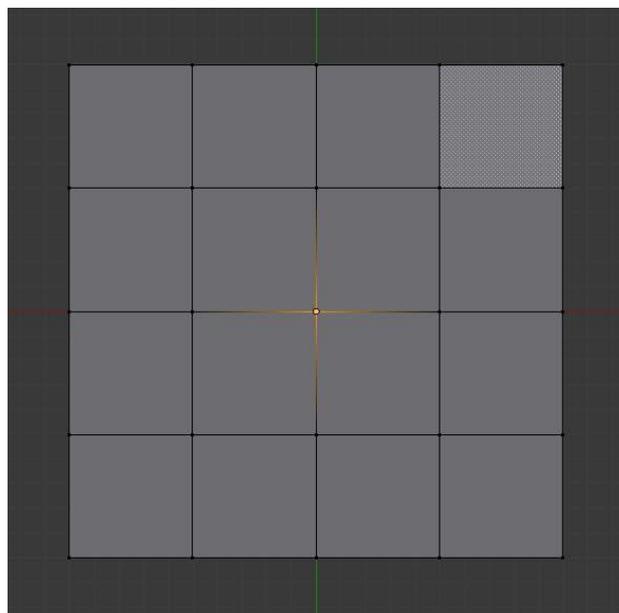


Fig. 2.199: selected vertex

## Examples

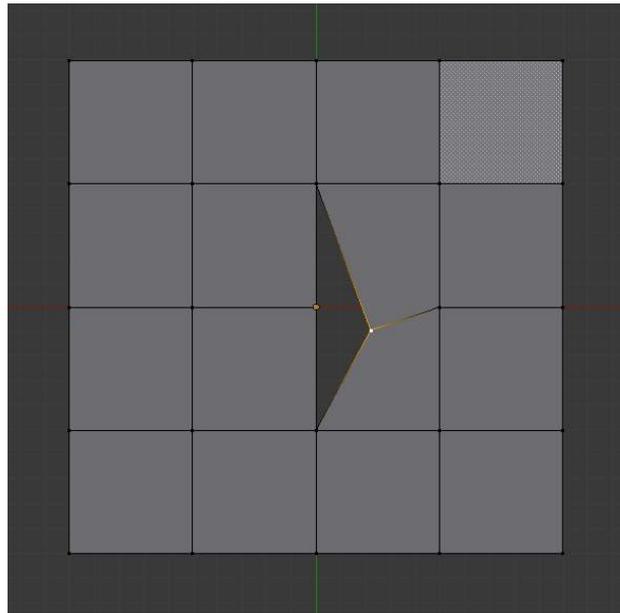


Fig. 2.200: Hole created after using rip on vertex

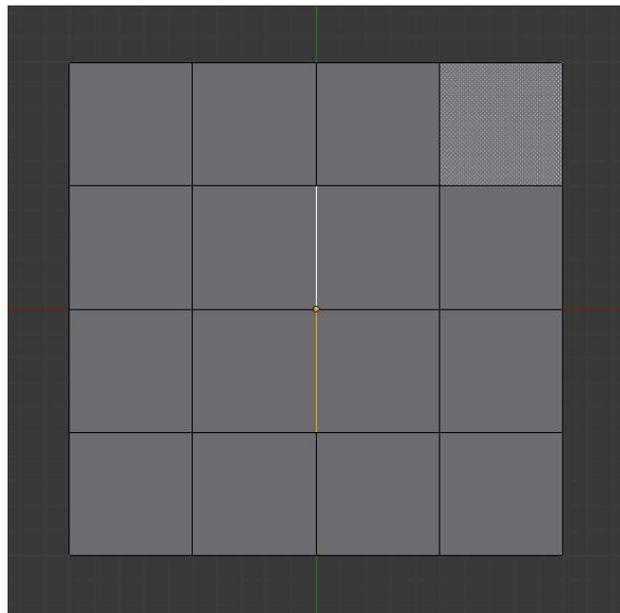


Fig. 2.201: Edges selected

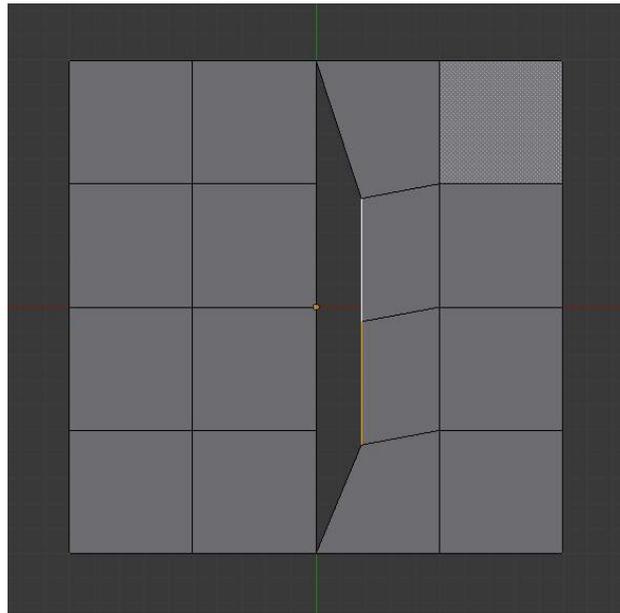


Fig. 2.202: Result of rip with edge selection

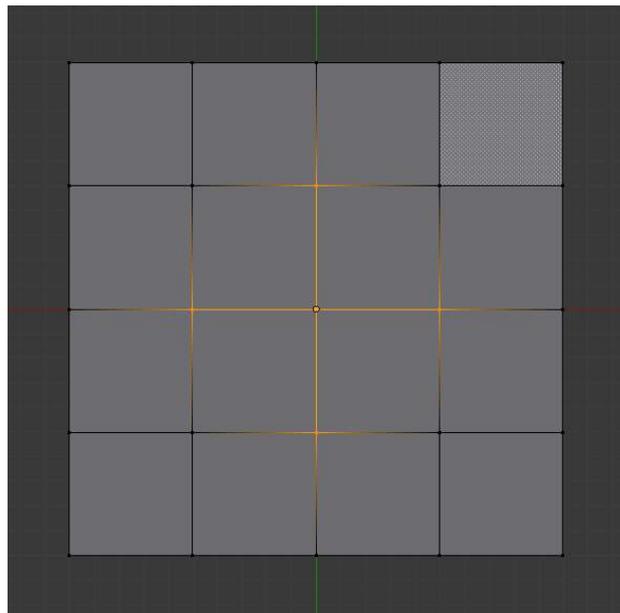


Fig. 2.203: A complex selection of vertices

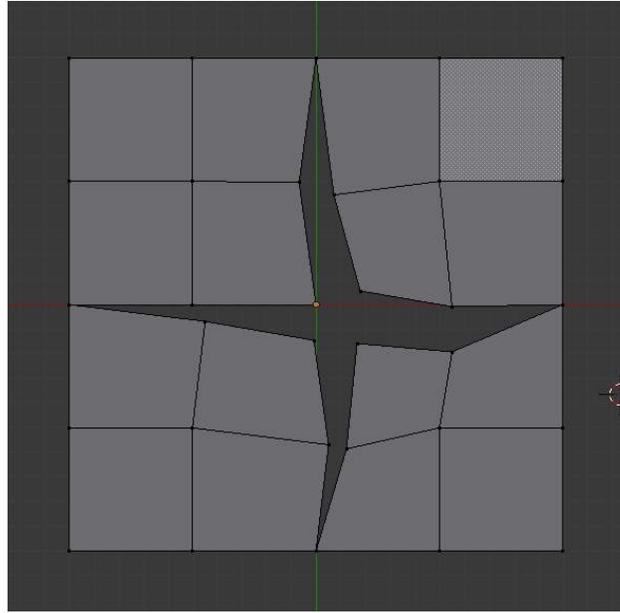


Fig. 2.204: Result of rip operation

**Limitations** Rip will only work when edges and/or vertices are selected. Using the tool when a face is selected (explicitly or implicitly), will return an error message “*Can’t perform ripping with faces selected this way*” If your selection includes some edges or vertices that are not “between” two faces (*manifold*), it will also fail with message “*No proper selection or faces include*”.

---

### Rip Fill

#### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Vertices* → *Rip Fill*

Hotkey: `Alt-V`

---

Rip fill works the same as the Rip tool above, but instead of leaving a hole, it fills in the gap with geometry.

---

### Split

#### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Vertices* → *Split*

Hotkey: `Y`

---

A quite specific tool, it makes a sort of copy of the selection, removing the original data *if it is not used by any non-selected element*. This means that if you split an edge from a mesh, the original edge will still remain unless it is not linked to anything else. If you split a face, the original face itself will be deleted, but its edges and vertices remain unchanged. And so on.

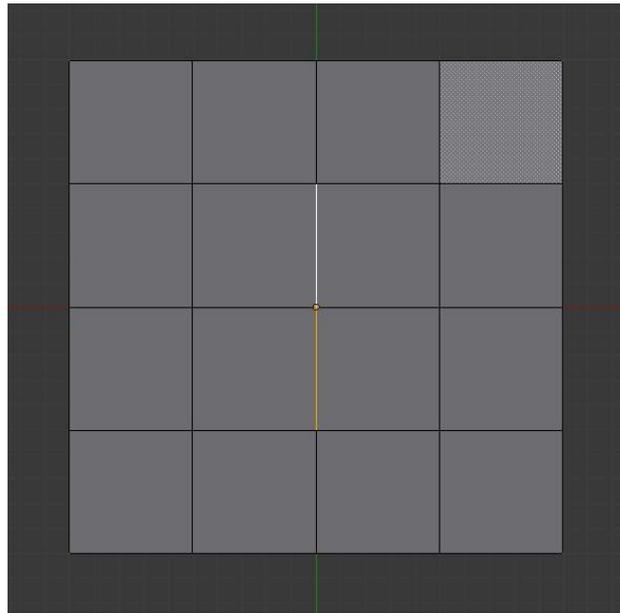


Fig. 2.205: Edges selected

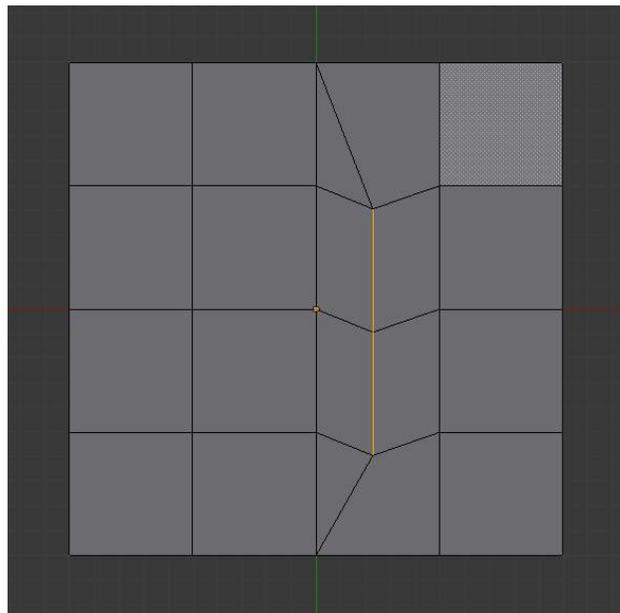


Fig. 2.206: Result of rip fill

Note that the “copy” is left exactly at the same position as the original, so you must move it (G) to see it clearly...

---

**Separate**  
**Reference**

Mode: *Edit mode*

Menu: *Mesh → Vertices → Separate*

Hotkey: P

---

This will separate the selection in another mesh object, as described [here](#).

---

**Connect Vertex Path**  
**Reference**

Mode: *Edit mode*

Menu: *Mesh → Vertices → Connect Vertex Path*

Hotkey: J

---

This tool connects vertices in the order they’re selected, splitting the faces between them.

Runnign a second time will connect the first/last endpoints.

Vertices not connected to any faces will create edges, so this can be used as a way to quickly connect isolated vertices too.

---

**Connect Vertices**  
**Reference**

Mode: *Edit mode*

Menu: *Mesh → Vertices → Connect Vertices*

---

This tool connects selected vertices by creating edges between them and splitting the face.

This tool can be used on many faces at once.

---

**Vertex Slide**  
**Reference**

Mode: *Edit mode*

Panel: *Editing context → Mesh Tools*

Menu: *Mesh → Vertices → Vertex Slide*

Hotkey: Shift-V

---

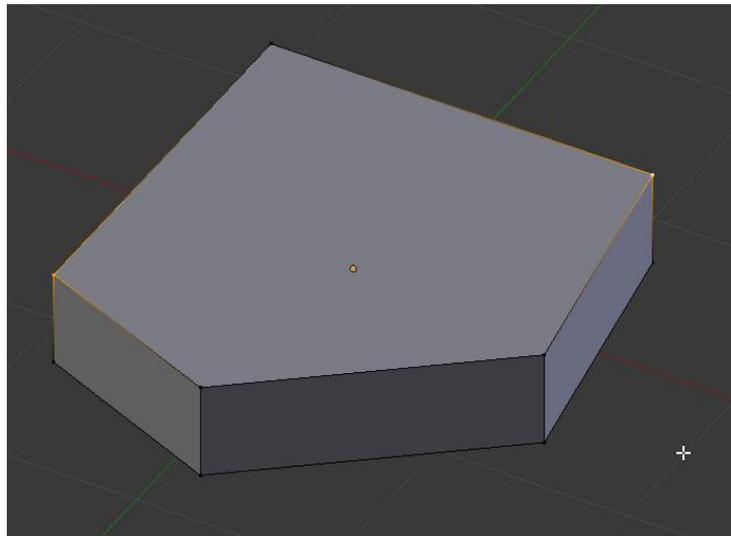


Fig. 2.207: Selected vertices before connecting

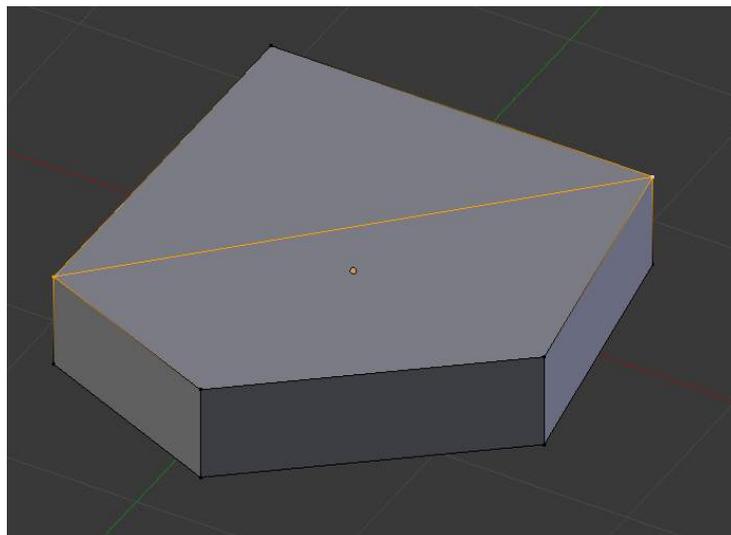


Fig. 2.208: After connecting vertices

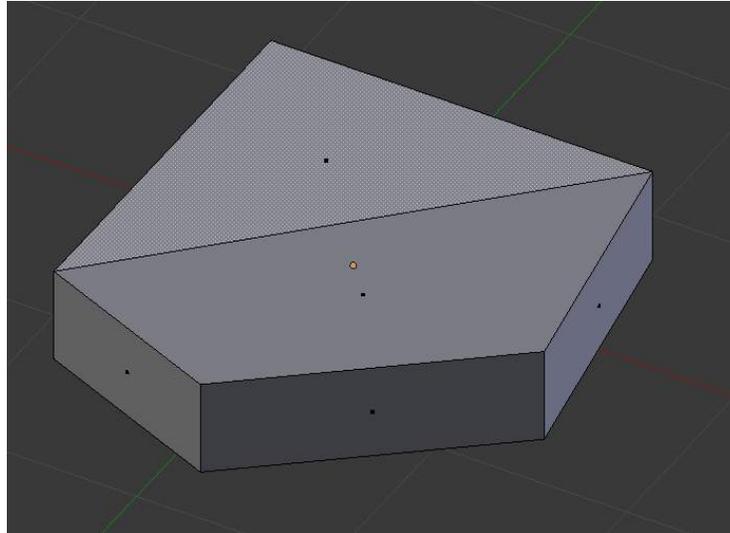


Fig. 2.209: Two faces created from vertex connect operation

Vertex Slide will transform a vertex along one of its adjacent edges. Use `Shift-V` to enter tool. Highlight the desired edge by moving the mouse, then confirm with `LMB`. Drag the cursor to specify the position along the line formed by the edge, then `LMB` again to move the vertex.

**Shift** Higher precision control.

**Ctrl** Snap to value (useful to combine with auto merge)

**LMB** confirms the tool

**RMB or Esc** Cancels.

**Alt or C** Toggle clamping the slide within the edge extents.

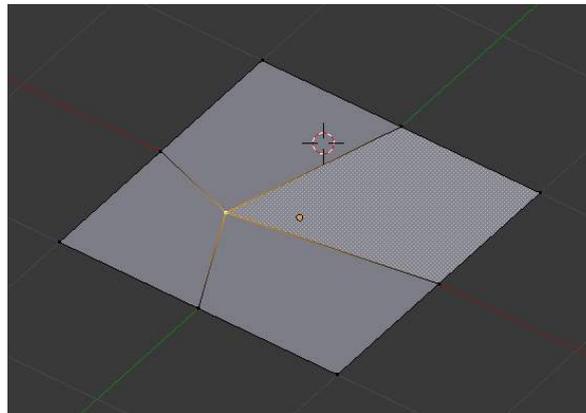


Fig. 2.210: Selected vertex

---

## Smooth Reference

Mode: *Edit mode*

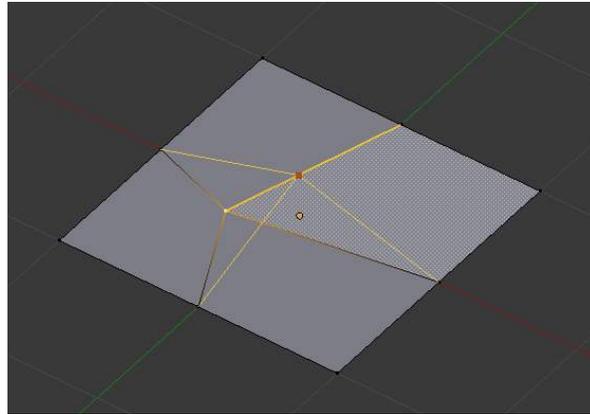


Fig. 2.211: Positioning vertex interactively

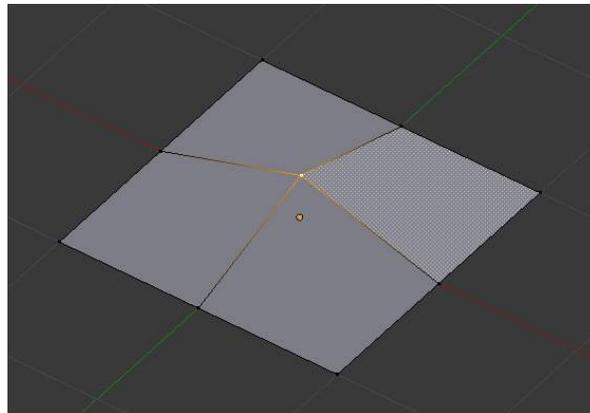


Fig. 2.212: Repositioned vertex

Panel: *Editing* context → *Mesh Tools*

Menu: *Mesh* → *Vertices* → *Smooth, Specials* → *Smooth* or *Vertex Specials* → *Smooth*

Hotkey: *Mesh* → *Vertices* → *Smooth vertex*

---

This will apply once the [Smooth Tool](#).

---

### **Make Vertex Parent**

#### **Reference**

Mode: *Edit* mode

Menu: *Mesh* → *Vertices* → *Make Vertex Parent*

Hotkey: `Ctrl-P`

---

This will parent the other selected object(s) to the vertices/edges/faces selected, as described [here](#).

---

### **Add Hook**

#### **Reference**

Mode: *Edit* mode

Menu: *Mesh* → *Vertices* → *Add Hook*

Hotkey: `Ctrl-H`

---

Adds a [Hook Modifier](#) (using either a new empty, or the current selected object) linked to the selection. Note that even if it appears in the history menu, this action cannot be undone in *Edit* mode - probably because it involves other objects...

---

### **Blend From Shape, Propagate Shapes**

#### **Reference**

Mode: *Edit* mode

Menu: (*Vertex*) *Specials* → *Blend From Shape* and *Mesh* → *Vertices* → *Shape Propagate*

---

These are options regarding [shape keys](#).

## **Edges**

---

### **Make Edge/Face**

#### **Reference**

Mode: *Edit* mode

Menu: *Mesh* → *Edges* → *Make Edge/Face*

Hotkey: `F`

---

It will create an edge or some faces, depending on your selection. We have already discussed this tool in the [editing basics](#) page.

**Set Edge Attributes** Edges can have several different attributes that affect how certain other tools affect the mesh.

---

### Mark Seam and Clear Seam

#### Reference

Mode: *Edit* mode (*Vertex* or *Edge* select modes)

Menu: *Mesh* → *Edges* → *Mark Seam/Clear Seam* (or the same options in *Edge Specials* menu)

Hotkey: `Ctrl-E-Numpad1` and `Ctrl-E-Numpad2`

---

Seams are a way to create separations, “islands”, in UV maps. See the [UVTexturing](#) section for more details. These commands set or unset this flag for selected edges.

---

### Mark Sharp and Clear Sharp

#### Reference

Mode: *Edit* mode (*Vertex* or *Edge* select modes)

Menu: *Mesh* → *Edges* → *Mark Seam/Clear Seam* (or the same options in *Edge Specials* menu)

Hotkey: `Ctrl-E-Numpad1` and `Ctrl-E-Numpad2`

---

The *Sharp* flag is used by the [EdgeSplit](#) modifier, which is part of the smoothing technics. As seams, it is a property of edges, and these commands set or unset it for selected ones.

---

### Adjust Bevel Weight

#### Reference

Mode: *Edit* mode (*Vertex* or *Edge* select modes)

Menu: *Mesh* → *Edges* → *Adjust Bevel Weight*

Hotkey: `Ctrl-Shift-E`

---

This edge property (a value between **0.0** and **1.0**) is used by the [Bevel](#) modifier to control the bevel intensity of the edges. This command enters an interactive mode (a bit like transform tools), where by moving the mouse (or typing a value with the keyboard) you can set the (average) bevel weight of selected edges.

---

### Crease SubSurf

#### Reference

Mode: *Edit* mode (*Vertex* or *Edge* select modes)

Menu: *Mesh* → *Edges* → *Crease SubSurf*

---

Hotkey: Shift-E

This edge property (a value between **0.0** and **1.0**) is used by the [Subsurf modifier](#) to control the sharpness of the edges in the subdivided mesh. This command enters an interactive mode (a bit like transform tools), where by moving the mouse (or typing a value with the keyboard) you can set the (average) crease value of selected edges. To clear the crease edge property, enter a value of **-1**.

## Edge Slide Reference

Mode: *Edit* mode (*Vertex* or *Edge* select modes)

Menu: *Mesh* → *Edges* → *Slide Edge* (or the same option in *Edge Specials* menu)

Hotkey: G, G

Slides one or more edges across adjacent faces with a few restrictions involving the selection of edges (*i.e. the selection must define a valid loop, see below.*)

**Shift** Higher precision control.

**Ctrl** Snap to value (useful to combine with auto merge)

**LMB** confirms the tool

**RMB** or **Esc** Cancels.

**Even E** Forces the edge loop to match the shape of the adjacent edge loop. You can flip to the opposite vertex using **F**. Use **Alt-Wheel** to change the control edge.

**Flip F** When Even mode is active, this flips between the two adjacent edge loops the active edge loop will match

**Alt** or **C** Toggle clamping the slide within the edge extents.

This tool has a factor, which is displayed in the 3D View footer and in the *Tool Shelf* (after confirmation). A numerical value between  $-1$  and  $1$  can be entered for precision.

In *Proportional* mode, **Wheel**, or **Left** and **Right** changes the selected edge for calculating a proportion. Unlike *Percentage* mode, *Proportional*

Holding **Ctrl** or **Shift** control the precision of the sliding. **Ctrl** snaps movement to 10% steps per move and **Shift** snaps movement to 1% steps. The default is 5% steps per move.

**Usage** By default, the position of vertices on the edge loop move as a percentage of the distance between their original position and the adjacent edge loop, regardless of the edges' lengths.

**Even mode** *Even* mode keeps the shape of the selected edge loop the same as one of the edge loops adjacent to it, rather than sliding a percentage along each perpendicular edge.

In *Even* mode, the tool shows the position along the length of the currently selected edge which is marked in yellow, from the vertex that as an enlarged red marker. Movement of the sliding edge loop is restricted to this length. As you move the mouse the length indicator in the header changes showing where along the length of the edge you are.

To change the control edge that determines the position of the edge loop, use the **Alt-Wheel** to scroll to a different edge.

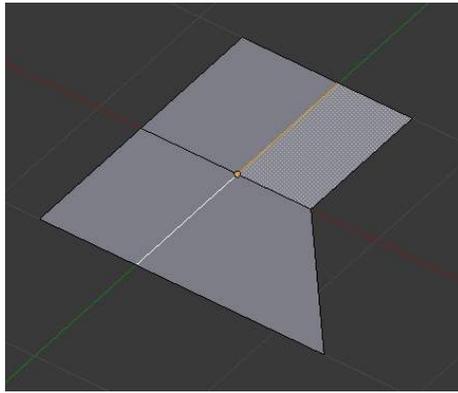


Fig. 2.213: selected edge loop

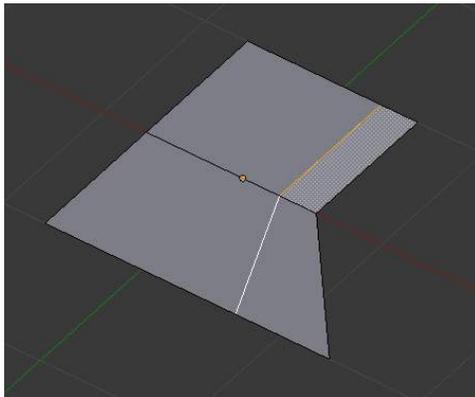


Fig. 2.214: Repositioned edge loop

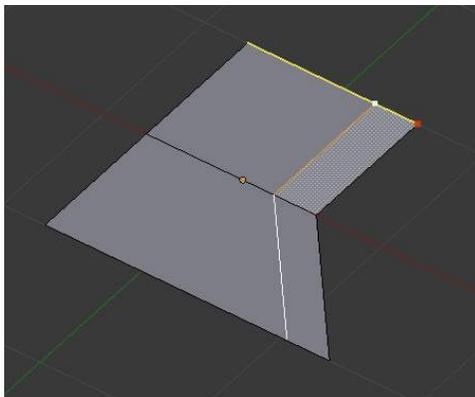


Fig. 2.215: Even mode enabled

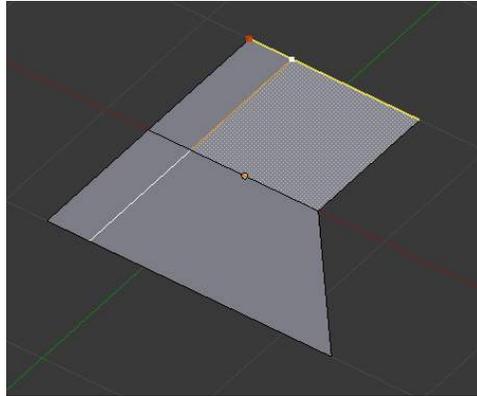


Fig. 2.216: Even mode with flip enabled

Moving the mouse moves the selected edge loop towards or away from the start vertex, but the loop line will only move as far as the length of the currently selected edge, conforming to the shape of one of the bounding edge loops.

**Limitations & Workarounds** There are restrictions on the type of edge selections that can be operated upon. Invalid selections are:

**Loop crosses itself** This means that the tool could not find any suitable faces that were adjacent to the selected edge(s). (*Loop crosses*) is an example that shows this by selecting two edges that share the same face. A face cannot be adjacent to itself.

**Multiple edge loops** The selected edges are not in the same edge loop, which means they don't have a common edge. You can minimize this error by always selecting edges end to end or in a "Chain". If you select multiple edges just make sure they are connected. This will decrease the possibility of getting looping errors.

**Border Edge** When a single edge was selected in a single sided object. An edge loop can not be found because there is only one face. Remember, edge loops are loops that span two or more faces.

A general rule of thumb is that if multiple edges are selected they should be connected end to end such that they form a continuous chain. This is *literally* a general rule because you can still select edges in a chain that are invalid because some of the edges in the chain are in different edge loops.

## Rotate Edge Reference

Mode: *Edit mode (Vertex or Edge select modes)*

Menu: *Mesh → Edges → Rotate Edge CW / Rotate Edge CCW*

Hotkey: *[ctrl][E] → Rotate Edge CW* and *[ctrl][E] → Rotate Edge CCW*

Rotating an edge clockwise or counter-clockwise spins an edge between two faces around their vertices. This is very useful for restructuring a mesh's topology. The tool can operate on one explicitly selected edge, or on two selected vertices or two selected faces that implicitly share an edge between them.

**Using Face Selection** To rotate an edge based on faces you must select two faces, (*Adjacent selected faces*), otherwise Blender notifies you with an error message, "*ERROR: Could not find any select edges that can be rotated*". Using either *Rotate Edge CW* or *Rotate Edge CCW* will produce exactly the same results as if you had selected the common edge shown in (*Selected edge rotated CW and CCW*).

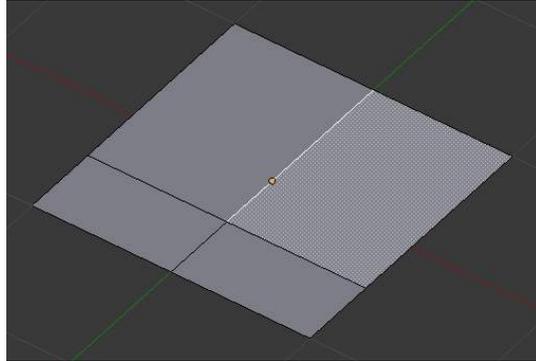


Fig. 2.217: selected edge

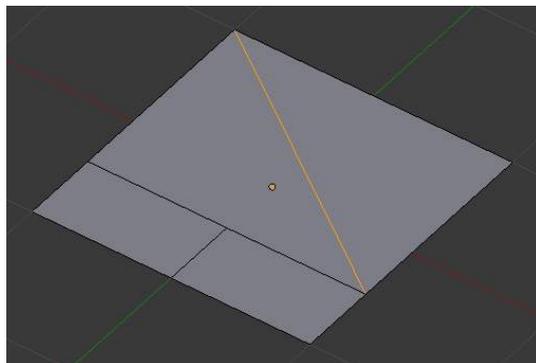


Fig. 2.218: Edge, rotated CW

---

## Delete Edge Loop

### Reference

Mode: *Edit* mode (*Vertex* or *Edge* select modes)

Menu: *Mesh* → *Delete* → *Edge Loop*

Hotkey: *[X]/[Del]* → *[g]*

---

*Delete Edge Loop* allows you to delete a selected edge loop if it is between two other edge loops. This will create one face-loop where two previously existed.

---

**Note:** The *Edge Loop* option is very different to the *Edges* option, even if you use it on edges that look like an edge loop. Deleting an edge loop merges the surrounding faces together to preserve the surface of the mesh. By deleting a chain of edges, the edges are removed, deleting the surrounding faces as well. This will leave holes in the mesh where the faces once were.

---

**Example** The selected edge loop on the UV Sphere has been deleted and the faces have been merged with the surrounding edges. If the edges had been deleted by choosing *Edges* from the (*Erase Menu*) there would be an empty band of deleted faces all the way around the sphere instead.

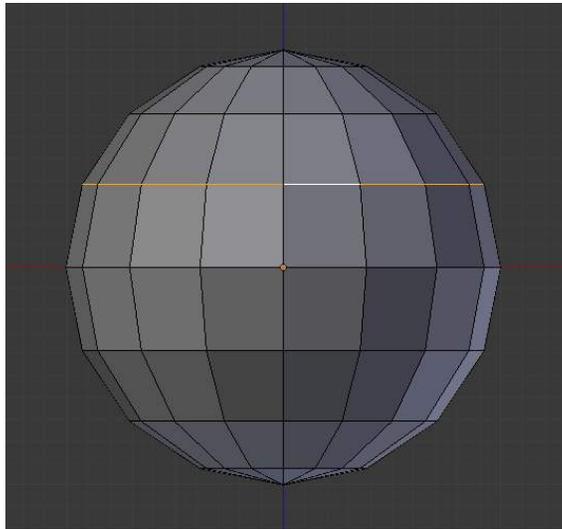


Fig. 2.219: Selected edge loop

---

## Collapse

### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Delete* → *Edge Collapse*

Hotkey: *[alt][M]* → *[pad3]*

---

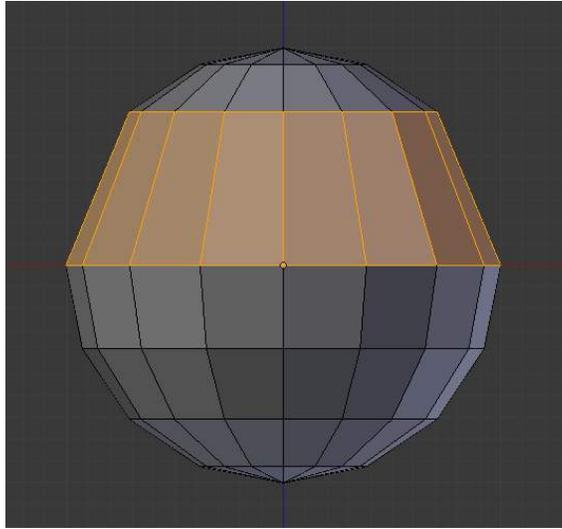


Fig. 2.220: Edge loop deleted

This takes a selection of edges and for each edge, merges its two vertices together. This is useful for taking a ring of edges and collapsing it, removing the face loop it ran through.

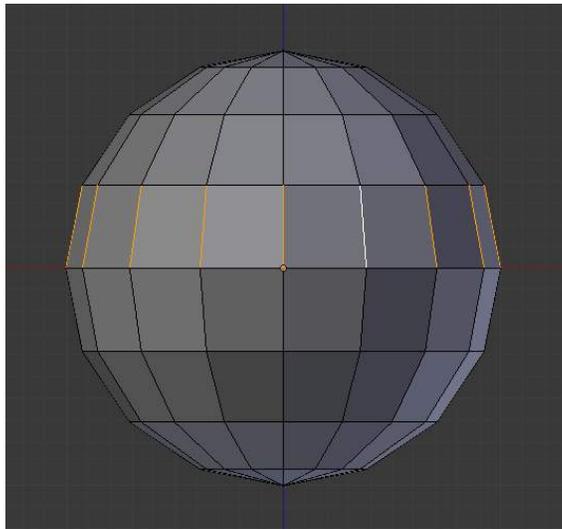


Fig. 2.221: Selected edge ring

---

### Edge Split Reference

Mode: *Edit* mode

Menu: *Mesh* → *Edges* → *Edge Split*

Hotkey: *[Ctrl][E]* → *Edge Split*

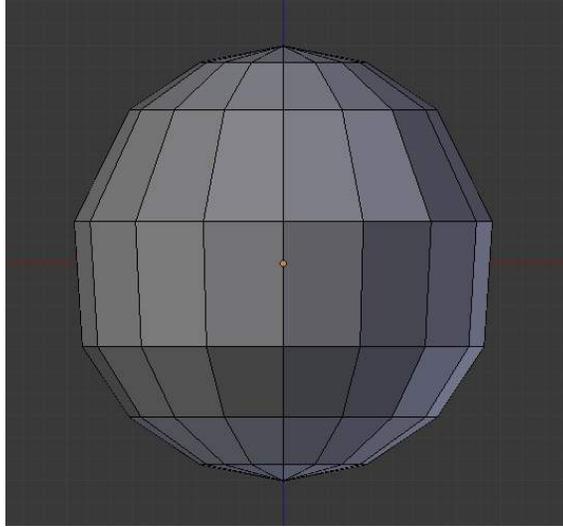


Fig. 2.222: Edge ring collapsed

*Edge split* is similar to the rip tool. When two or more touching interior edges, or a border edge is selected when using *Edge split*, a hole will be created, and the selected edges are duplicated to form the border of the hole

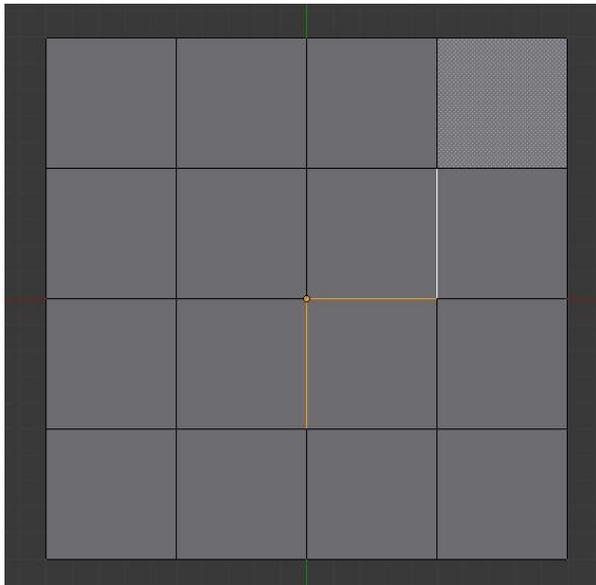


Fig. 2.223: Selected edges

### Bridge Edge Loops Reference

Mode: *Edit mode*

Menu: *Mesh* → *Edges* → *Bridge Edge Loops*

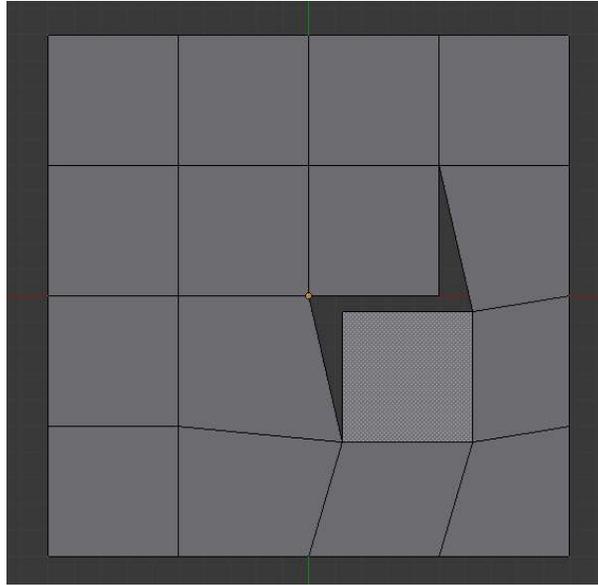


Fig. 2.224: Adjacent face moved to reveal hole left by split

*Bridge Edge Loops* connects multiple edge loops with faces.

Simple example showing 2 closed edge loops.

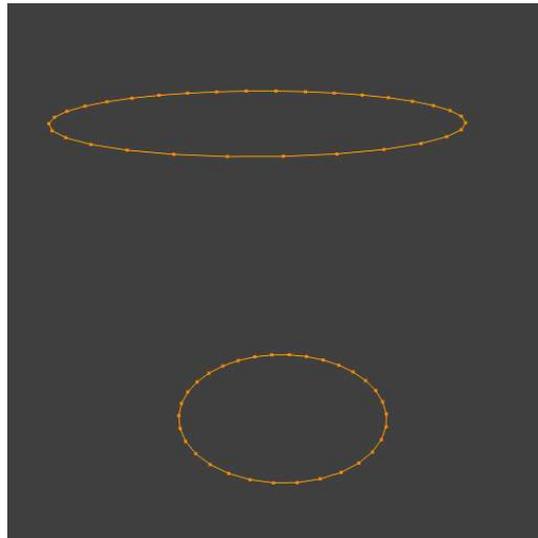


Fig. 2.225: Input

Example of bridge tool between edge loops with different numbers of vertices.

Example using the bridge tool to punch holes in face selections and connect them.

Example showing how bridge tool can detect multiple loops and loft them in one step.

Example of the subdivision option and surface blending with UV's.



Fig. 2.226: Bridge result

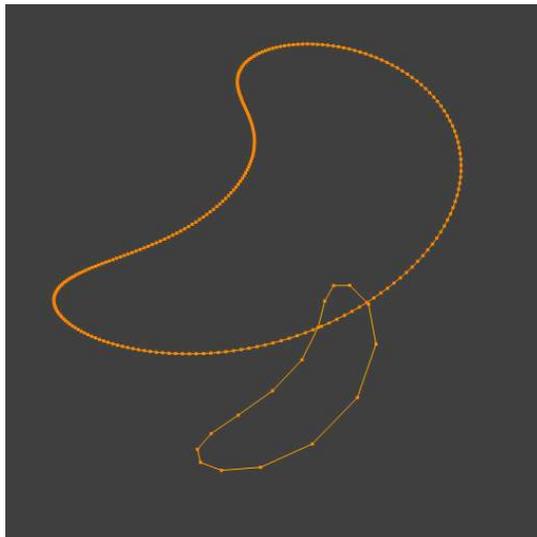


Fig. 2.227: Input

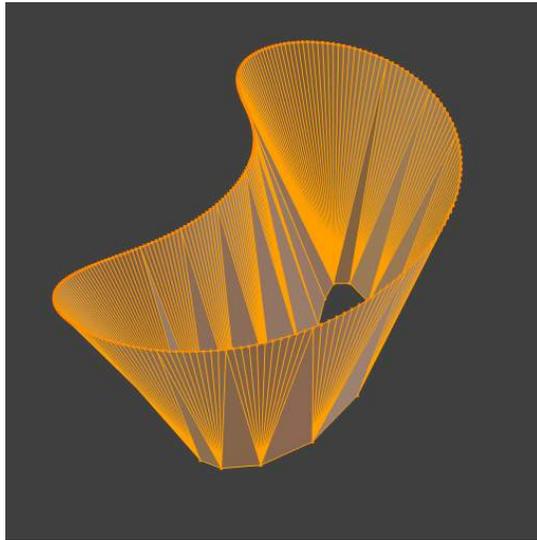


Fig. 2.228: Bridge result

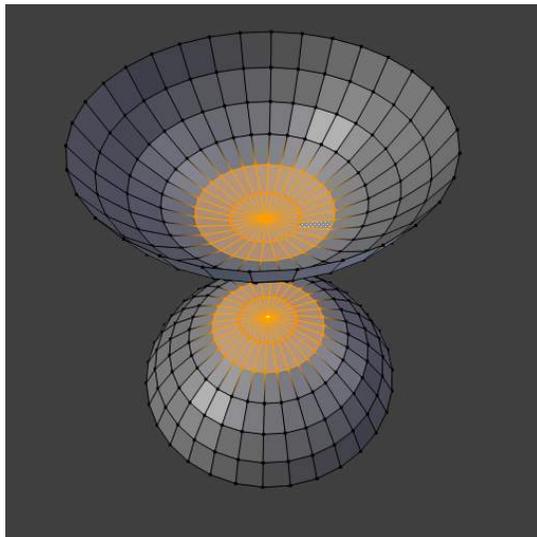


Fig. 2.229: Input

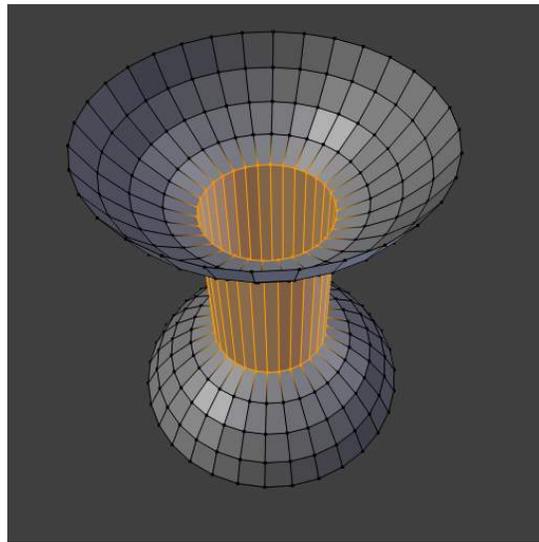


Fig. 2.230: Bridge result

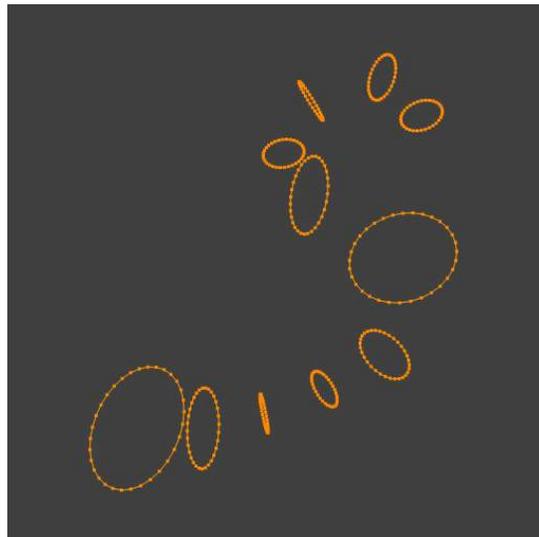


Fig. 2.231: Input

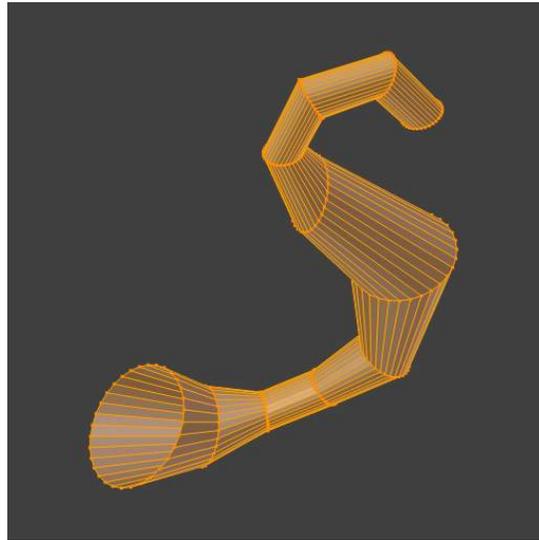


Fig. 2.232: Bridge result

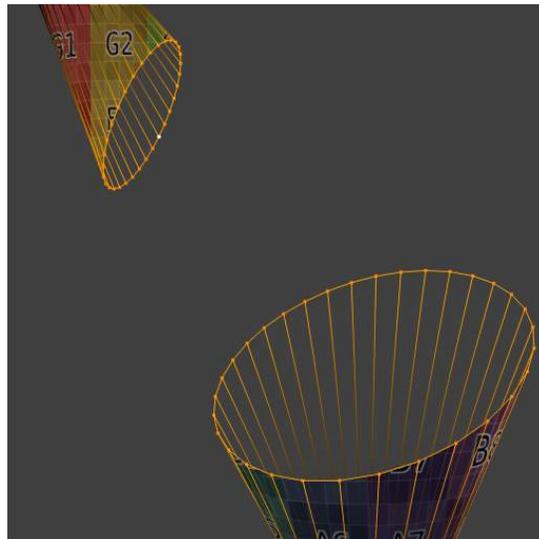


Fig. 2.233: Input

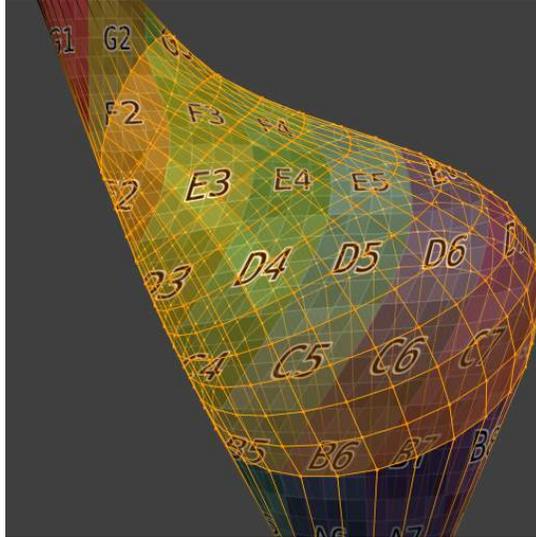


Fig. 2.234: Bridge result

## Face Tools

These are tools that manipulate faces.

### Creating Faces

#### Make Edge/Face Reference

Mode: *Edit mode*

Menu: *Mesh* → *Faces* → *Make Edge/Face*

Hotkey: *F*

This will create an edge or some faces, depending on your selection. It is detailed in the [Basic Editing](#) page.

#### Fill Reference

Mode: *Edit mode*

Menu: *Mesh* → *Faces* → *Fill/Beautify Fill*

Hotkey: *Alt-F*

The *Fill* option will create *triangular* faces from any group of selected edges or vertices, *as long as they form one or more complete perimeters*.

note, unlike creating n-gons, fill supports holes.

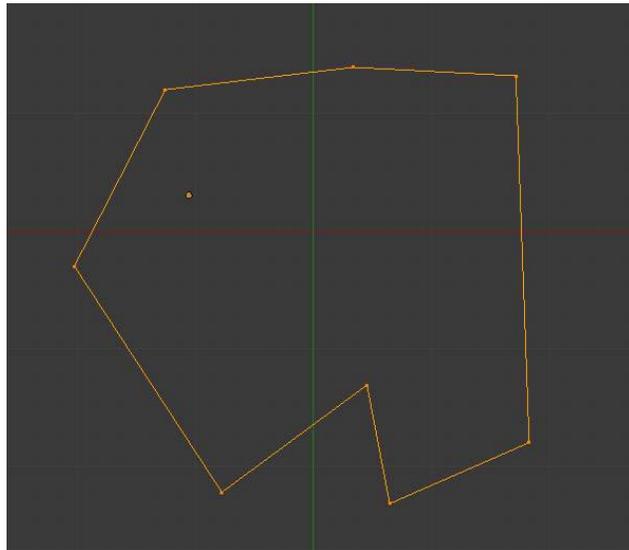


Fig. 2.235: A closed perimeter of edges

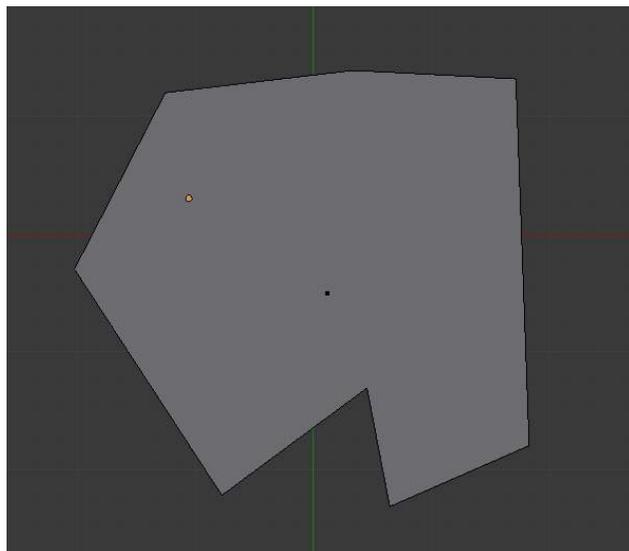


Fig. 2.236: Filled using shortcut [F]. Created an n-gon

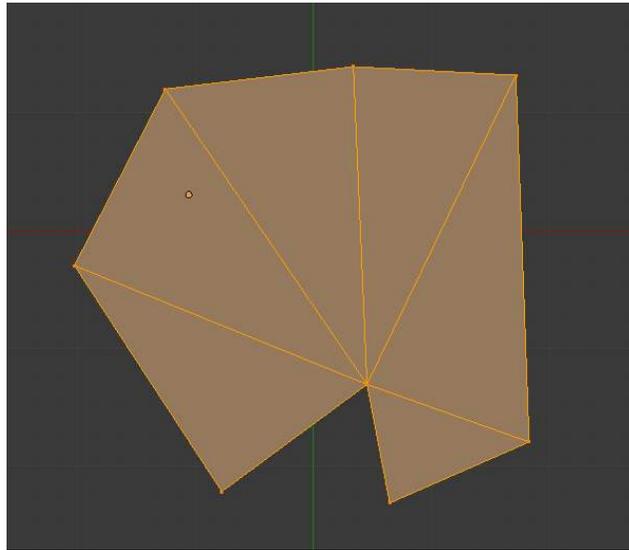


Fig. 2.237: Filled using fill[Alt][F]

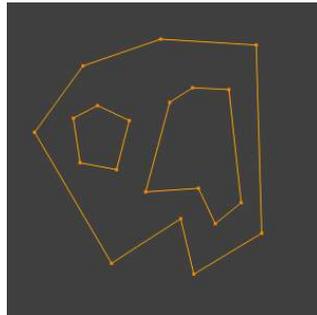


Fig. 2.238: A closed perimeter of edges with holes

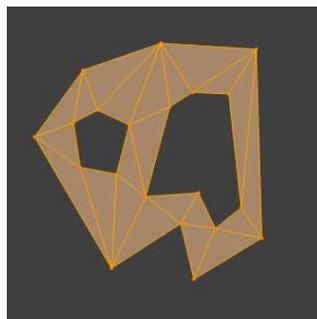


Fig. 2.239: Filled using fill[Alt][F]

---

**Beauty Fill**  
**Reference**

Mode: *Edit* mode

Menu: *Mesh* → *Faces* → *Fill/Beauty Fill*

Hotkey: *Alt-Shift-F*

---

*Beauty Fill* works only on selected existing faces. It rearrange selected triangles to obtain more “balanced” ones (i.e. less long thin triangles).



Fig. 2.240: Text converted to a mesh



Fig. 2.241: Result of Beauty Fill, *Alt-Shift-F*

---

**Grid Fill**  
**Reference**

Mode: *Edit* mode

Menu: *Mesh* → *Faces* → *Fill/Grid Fill*

---

*Grid Fill* uses a pair of connected edge-loops to fill in a grid that follows the surrounding geometry.

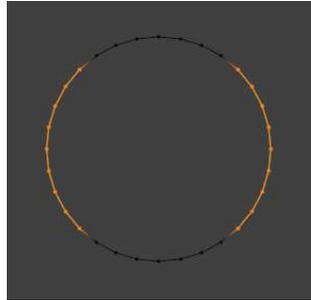


Fig. 2.242: Input

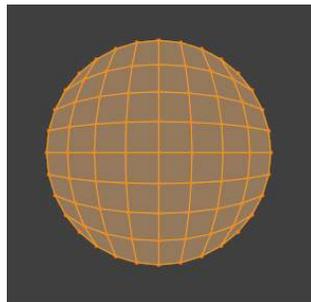


Fig. 2.243: Grid fill result

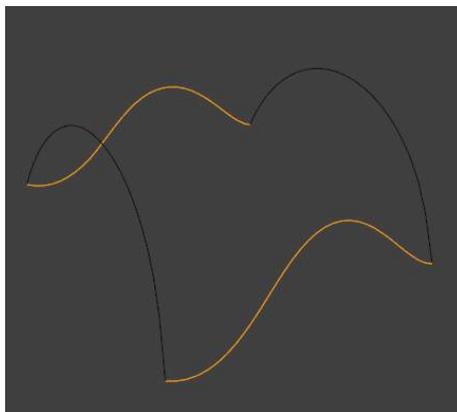


Fig. 2.244: Input

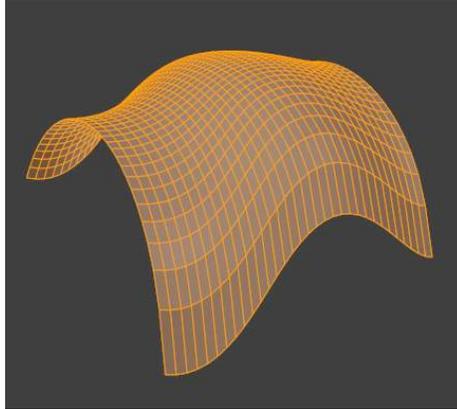


Fig. 2.245: Grid fill result

---

### Convert Quads to Triangles Reference

Mode: *Edit* mode

Menu: *Mesh* → *Faces* → *Convert Quads to Triangles* or *Face Specials* → *Triangulate*

Hotkey: `Ctrl-T`

---

As its name intimates, this tool converts each selected quadrangle into two triangles. Remember that quads are just a set of two triangles.

---

### Convert Triangles to Quads Reference

Mode: *Edit* mode

Panel: *Mesh Tools* (*Editing* context)

Menu: *Mesh* → *Faces* → *Convert Triangles to Quads*

Hotkey: `Alt-J`

---

This tool converts the selected triangles into quads by taking adjacent tris and removes the shared edge to create a quad, based on a threshold. This tool can be performed on a selection of multiple triangles.

This same action can be done on a selection of 2 tris, by selecting them and using the shortcut `F`, to create a face, or by selecting the shared edge and dissolving it with the shortcut `[X]` → *Dissolve*.

To create a quad, this tool needs at least two adjacent triangles. If you have an even number of selected triangles, it is also possible not to obtain only quads. In fact, this tool tries to create “squareshest” quads as possible from the given triangles, which means some triangles could remain.

All the menu entries and hotkey use the settings defined in the *Mesh Tools* panel:

**Max Angle** This values (between **0** and **180**) controls the threshold for this tool to work on adjacent triangles. With a threshold of **0.0**, it will only join adjacent triangles that form a perfect rectangle (i.e. right-angled triangles sharing their hy-

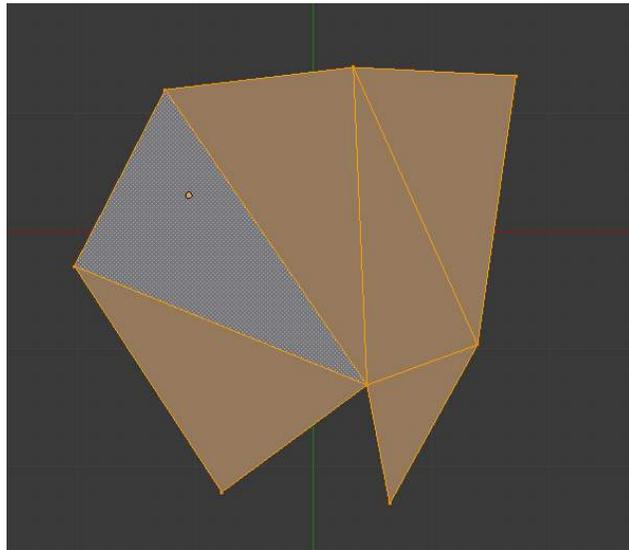


Fig. 2.246: Before converting tris to quads

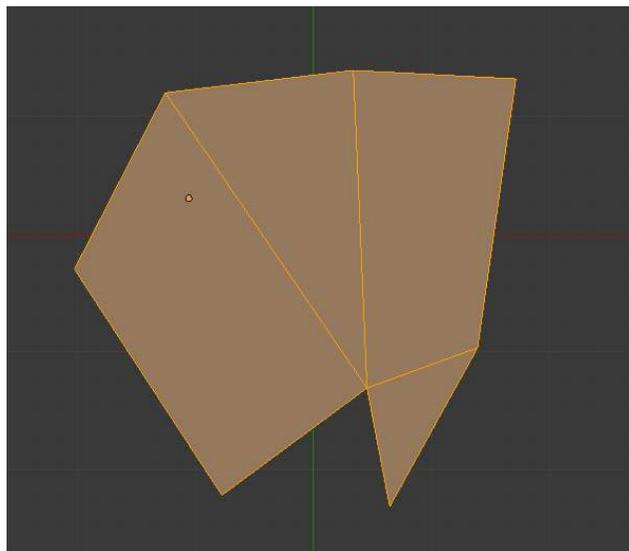


Fig. 2.247: After converting tris to quads, with a max angle of 30

potenuses). Larger values are required for triangles with a shared edge that is small, relative to the size of the other edges of the triangles.

**Compare UVs** When enabled, it will prevent union of triangles that are not also adjacent in the active UV map. Note that this seems to be the only option working...

**Compare Vcol** When enabled, it will prevent union of triangles that have no matching vertex color. I'm not sure how this option works - or even if it really works...

**Compare Sharp** When enabled, it will prevent union of triangles that share a "sharp" edge. I'm not sure either if this option works, and what is the "sharp" criteria - neither the *Sharp* flag nor the angle between triangles seem to have an influence here...

**Compare Materials** When enabled, it will prevent union of triangles that do not use the same material index. This option does not seem to work neither...

---

## Solidify

### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Faces* → *Solidify*

Hotkey: *[ctrl][F]* → *Solidify*

---

This takes a selection of faces and solidifies them by extruding them uniformly to give volume to a *non-manifold* surface. This is also available as a **Modifier**. After using the tool, you can set the offset distance in the Tool Palette.

**Thickness** Amount to offset the newly created surface. Positive values offset the surface inward relative to the normals. Negative values offset outward.

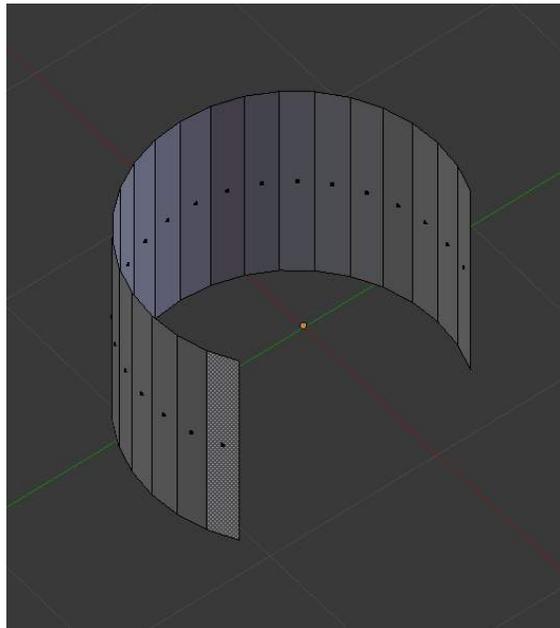


Fig. 2.248: Mesh before solidify operation

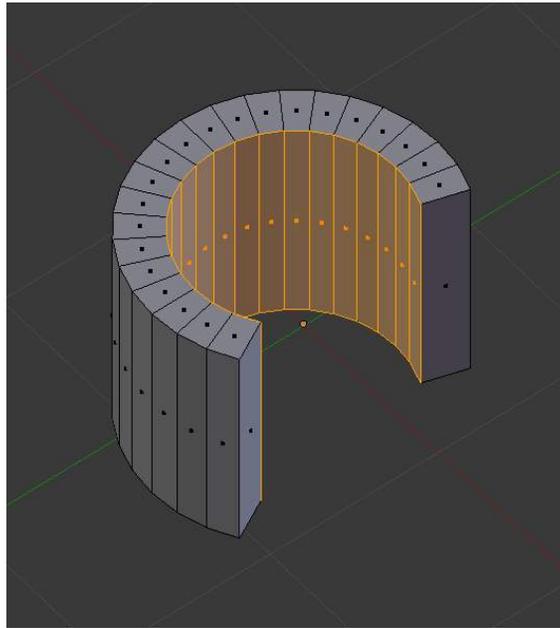


Fig. 2.249: Solidify with a positive thickness

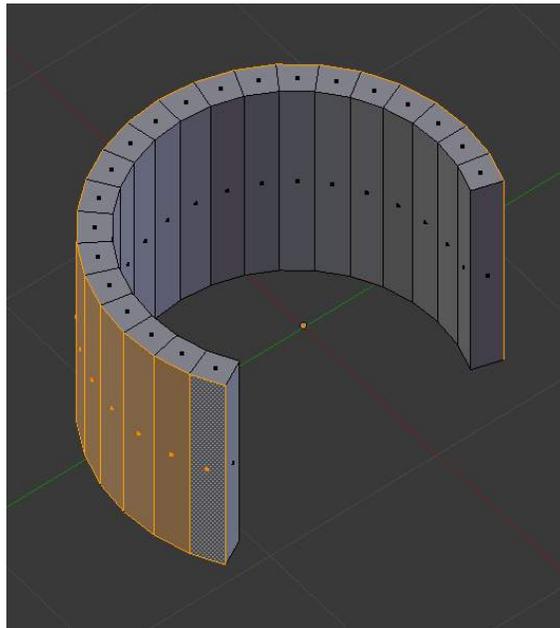


Fig. 2.250: Solidify with a negative thickness

---

**Rotate Edges**  
**Reference**

Mode: *Edit* mode

Menu: *Mesh* → *Faces* → *Rotate Edge CW*

---

This command functions the same edge rotation in edge mode.

It works on the shared edge between two faces and rotates that edge if the edge was selected.

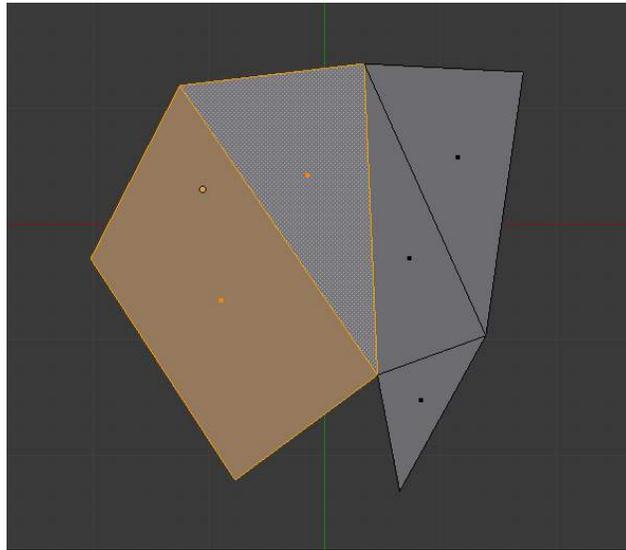


Fig. 2.251: Two faces selected

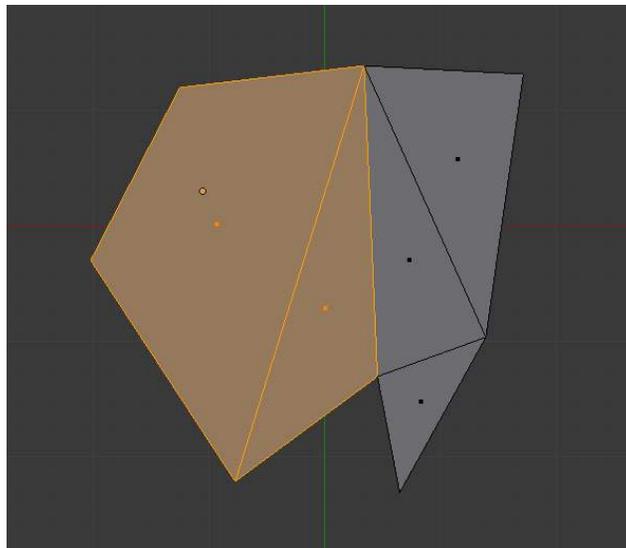


Fig. 2.252: After rotating edge

See `Rotate Edge CW / Rotate Edge CCW` for more information.

**Normals** As normals are mainly a face “sub-product”, we describe their few options here also.

See [Smoothing](#) for additional information on working with face normals.

---

### Flip Direction Reference

Mode: *Edit mode*

Menu: *Mesh → Normals → Flip* or *Specials → Flip Normals*

Hotkey: *[W] → Flip Normals }*

---

Well, it will just reverse the normals direction of all selected faces. Note that this allows you to precisely control the direction (**not the orientation**, which is always perpendicular to the face) of your normals, as only selected ones are flipped.

---

### Recalculate Normals Reference

Mode: *Edit mode*

Menu: *Mesh → Normals → Recalculate Outside* and *Mesh → Normals → Recalculate Inside*

Hotkey: *Ctrl-N* and *ctrl*

---

These commands will recalculate the normals of selected faces so that they point outside (respectively inside) the volume that the face belongs to. This volume do not need to be closed. In fact, this means that the face of interest must be adjacent with at least one non-coplanar other face. For example, with a *Grid* primitive, neither *Recalculate Outside* nor *Recalculate Inside* will never modify its normals...

## Deforming

---

### Mirror Reference

Mode: *Edit mode*

Menu: *Mesh → Mirror → Desired Axis*

Hotkey: *Ctrl-M*

---

The mirror tool mirrors a selection across a selected axis.

The mirror tool in *Edit mode* is similar to [Mirroring in Object mode](#). It is exactly equivalent to scaling by -1 vertices, edges or faces around one chosen pivot point and in the direction of one chosen axis, only it is faster/handier.

After this tool becomes active, select an axis to mirror the selection on entering x,y, or z.

You can also interactively mirror the geometry by holding the *MMB* and dragging in the direction of the desired mirror direction.

**Axis of symmetry** For each transformation orientation, you can choose one of its axes along which the mirroring will occur. As you can see, the possibilities are infinite and the freedom complete: you can position the pivot point at any location around which we want the mirroring to occur, choose one transformation orientation and then one axis on it.

**Pivot point** **Pivot points** must be set first. Pivot points will become the center of symmetry. If the widget is turned on it will always show where the pivot point is.

On (*Mirror around the Individual Centers ...*) the pivot point default to **median point of the selection of vertices** in *Edit* mode. This is a special case of the *Edit* mode as explained on the [pivot point](#) page.

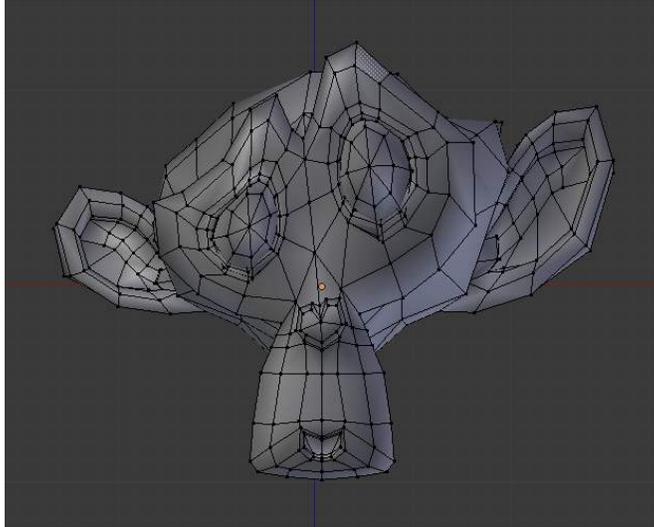


Fig. 2.253: Mesh before mirror.

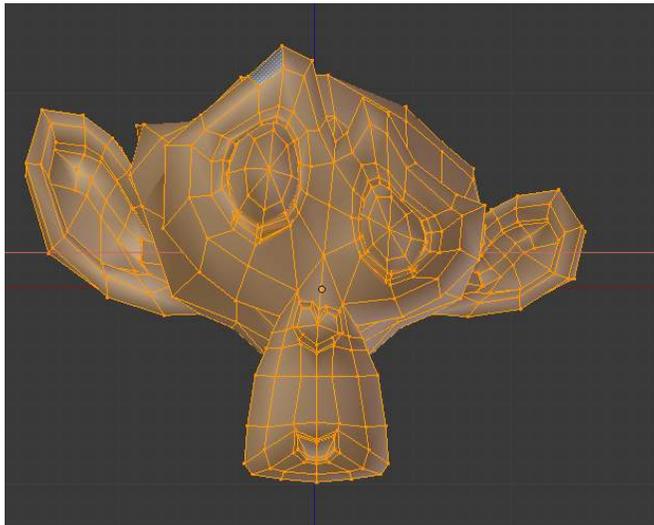


Fig. 2.254: Mesh after mirrored along X axis

On (*Mirror around the 3D Cursor ...*) the pivot point is the *3D Cursor*, the transformation orientation is *Local*, a.k.a. the Object space, and the axis of transformation is X.

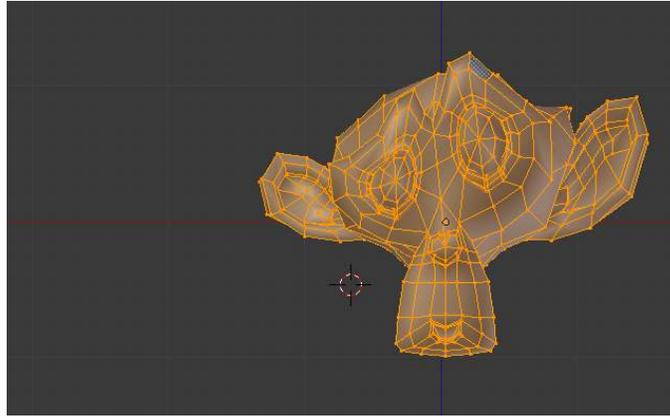


Fig. 2.255: Mesh before mirror.

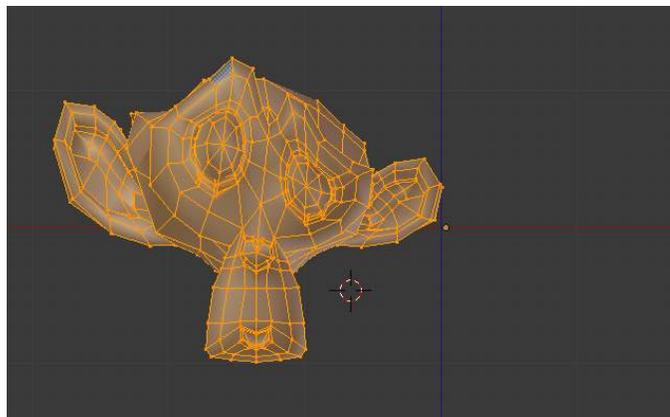


Fig. 2.256: Mesh after mirrored along X axis using the 3d cursor as a pivot point

**Transformation orientation** [Transformation Orientations](#) are found on the 3D area header, next to the *Widget* buttons. They decide which coordinate system will rule the mirroring.

---

### Shrink/Fatten Along Normals

#### Reference

Mode: *Edit* mode

Panel: *Mesh Tools* (*Editing* context)

Menu: *Mesh* → *Transform* → *Shrink/Fatten Along Normals*

Hotkey: `Alt-S`

---

This tool translates selected vertices/edges/faces along their own normal (perpendicular to the face), which, on “standard normal meshes”, will shrink/fatten them.

This transform tool does not take into account the pivot point or transform orientation.

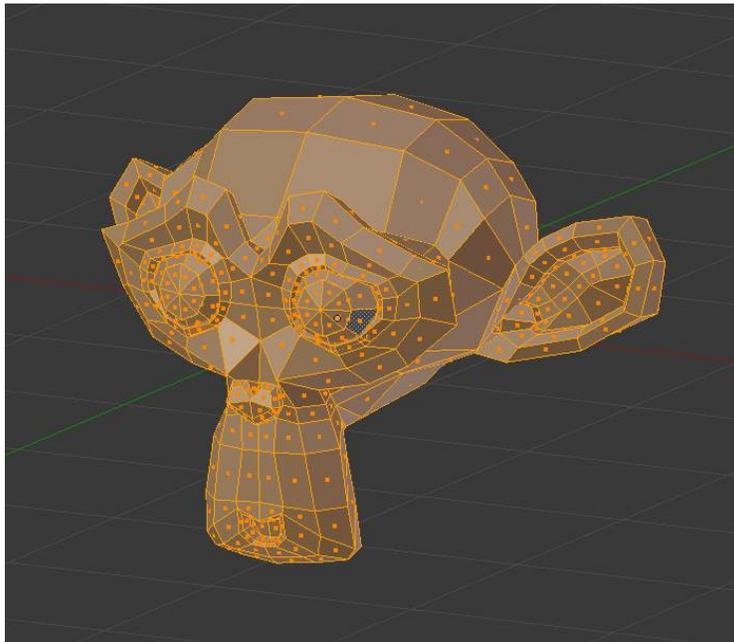


Fig. 2.257: mesh before shrink/flatten

---

### Smooth

#### Reference

Mode: *Edit* mode

Panel: *Mesh Tools* (*Editing* context)

Menu: *Mesh* → *Vertices* → *Smooth vertex*

Hotkey: `[ctrl][V]` → *Smooth vertex*

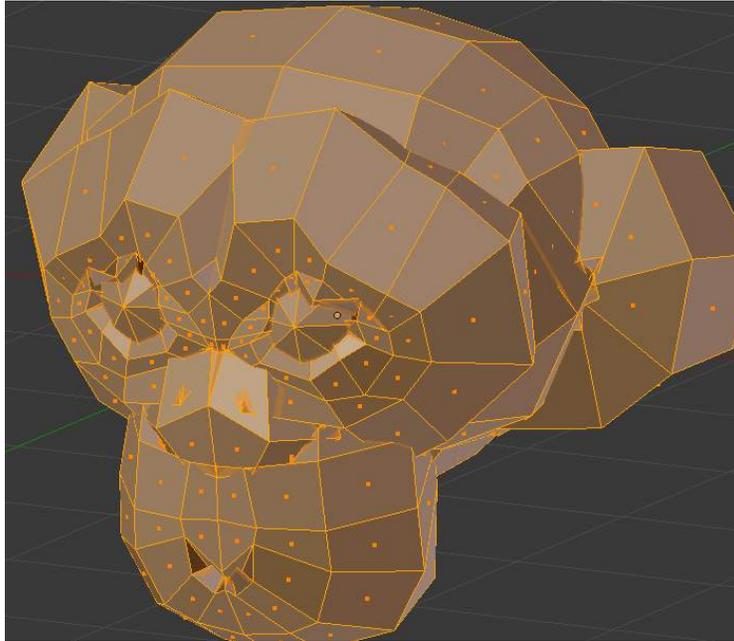


Fig. 2.258: Inflated using a positive value

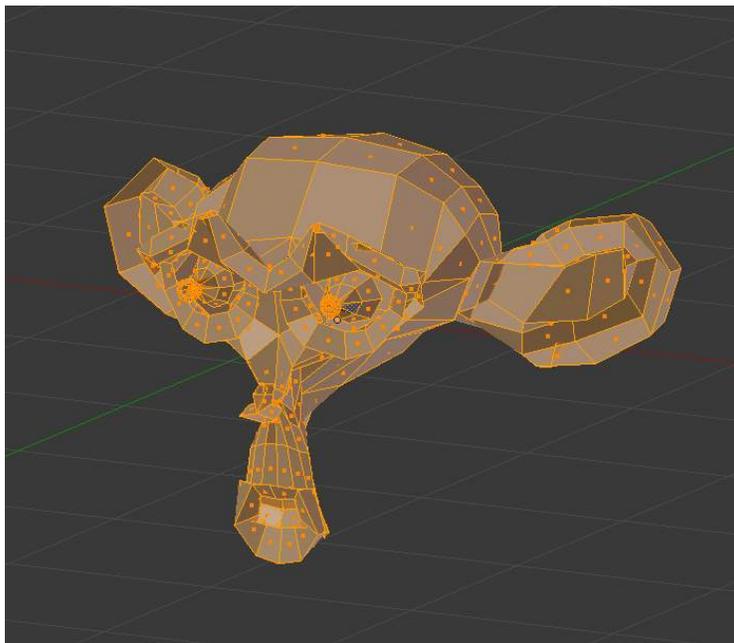


Fig. 2.259: Shrunk using a negative value

This tool smooths the selected components by averaging the angles between faces. After using the tool, options appear in the *Tool Shelf*:

**Number of times to smooth** The number of smoothing iterations

**Axes** Limit the effect to certain axes.

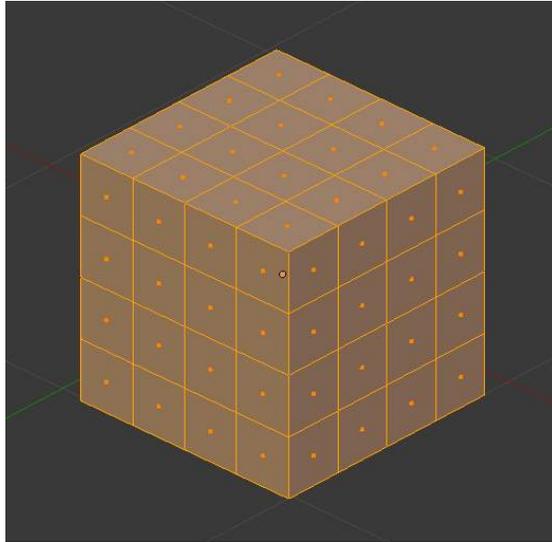


Fig. 2.260: mesh before smoothing

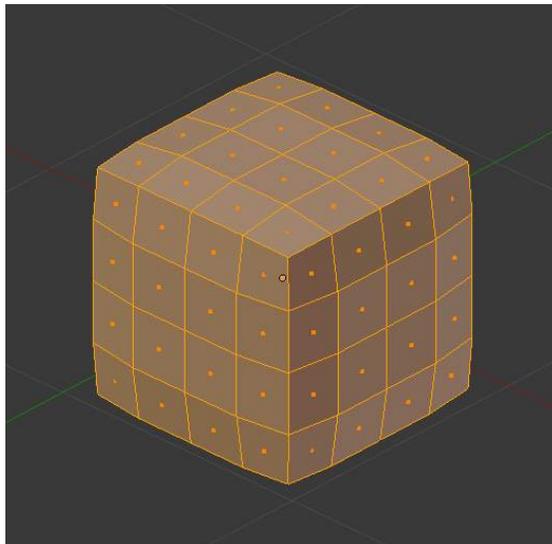


Fig. 2.261: mesh after 1 smoothing iteration

## Laplacian Smooth Reference

Mode: *Edit* mode

Hotkey: *[W]* → *Laplacian Smooth*

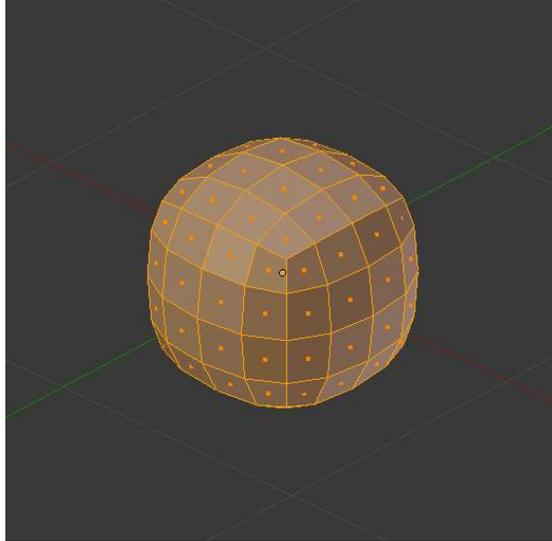


Fig. 2.262: mesh after 10 smoothing iterations

See the [Laplacian Smooth Modifier](#) for details.

Laplacian smooth uses an alternative smoothing algorithm that better preserves the overall mesh shape. Laplacian smooth exists as a mesh operation and as a non-destructive modifier.

**Note:** The [Smooth modifier](#), which can be limited to a *Vertex Group*, is a non-destructive alternative to the smooth tool.

**Note:** Real Smoothing versus Shading Smoothing

Do not mistake this tool with the shading smoothing options described at [this page](#), they do not work the same! This tool modifies the mesh itself, to reduce its sharpness, whereas *Set Smooth / AutoSmooth* and co. only control the way the mesh is shaded, creating an *illusion* of softness - but without modifying the mesh at all...

## Noise

### Reference

Mode: *Edit* mode

Panel: Mesh Tools (*Editing* context)

**Note:** *Noise* is an old feature. The [Displace Modifier](#) is a non-destructive alternative to the Noise tool and is a more flexible way to realize these sort of effects. The key advantages of the modifier are that it can be canceled at any moment, you can precisely control how much and in which direction the displacement is applied, and much more.... See also the [ANT Landscape add-on](#).

The *Noise* function allows you to displace vertices in a mesh based on the grey values of the first texture slot of the material applied to the mesh.

The mesh must have a material and a texture assigned to it for this tool to work. To avoid having the texture affect the material's properties, it can be disabled in the texture menu.

The *Noise* function displaces vertices along the object's  $\pm Z$ -Axis only.

*Noise* permanently modifies your mesh according to the material texture. Each click adds onto the current mesh. For a temporary effect, map the texture to Displacement for a render-time effect. In *Object / Edit* mode, your object will appear normal, but will render deformed.

The deformation can be controlled by modifying the *Mapping* panel and/or the texture's own panel (e.g. *Clouds*, *Marble*, etc.).

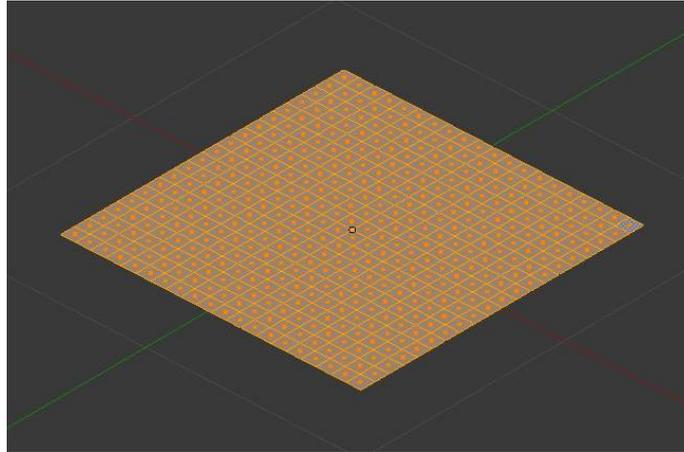


Fig. 2.263: mesh before noise is added

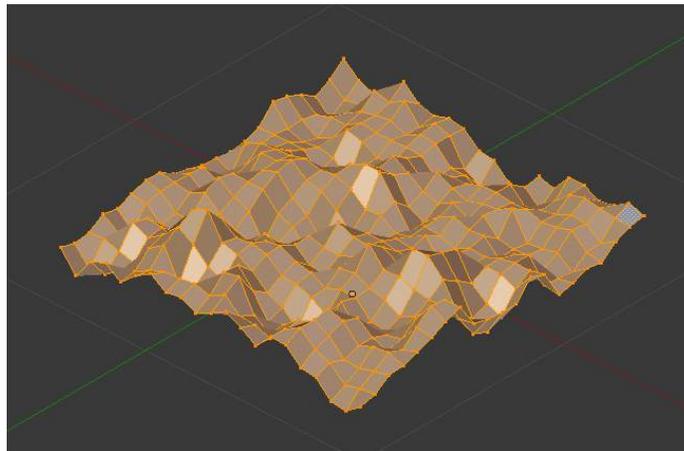


Fig. 2.264: mesh after noise is added, using basic cloud texture

---

### Push/Pull Reference

Mode: *Object* and *Edit* modes

Menu: *Object/Mesh* → *Transform* → *Push Pull*

---



Fig. 2.265: Push/Pull distance.

**Description** *Push/Pull* will move the selected elements (Objects, vertices, edges or faces) closer together (Push) or further apart (Pull). Specifically, each element is moved towards or away from the center by the same distance. This distance is controlled by moving the mouse up (Push) or down (Pull), numeric input or through slider control.

**Usage** Select the elements you want to operate on and activate the Push/Pull transform function. The Push/Pull option can be invoked from the *Object/Mesh* → *Transform* → *Push/Pull* menu option or by pressing `Spacebar` and using the search menu to search for *Push* or *Pull*. The amount of movement given to the selection can be determined interactively by moving the mouse or by typing a number. Pressing `Return` will confirm the transformation. The confirmed transformation can be further edited by pressing `F6` or by going into the Toolshelf (T) and altering the Distance slider provided that no other actions take place between the *Push/Pull* transform confirmation and accessing the slider.

Note that the result of the *Push/Pull* transform is also dependant on the number and type of selected elements (Objects, vertices, faces etc). See below for the result of using *Push/Pull* on a number of different elements.

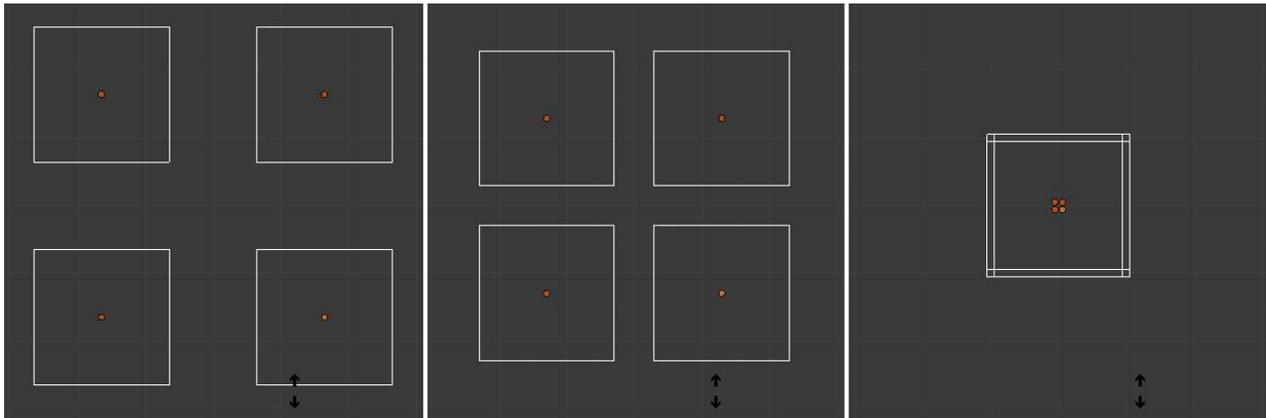


Fig. 2.266: Equidistant Objects being pushed together.

## Shear

---

### Reference

Mode: *Object* and *Edit* modes

Menu: *Object/Mesh/Curve/Surface* → *Transform* → *Shear*

Hotkey: `Shift-Ctrl-Alt-S`

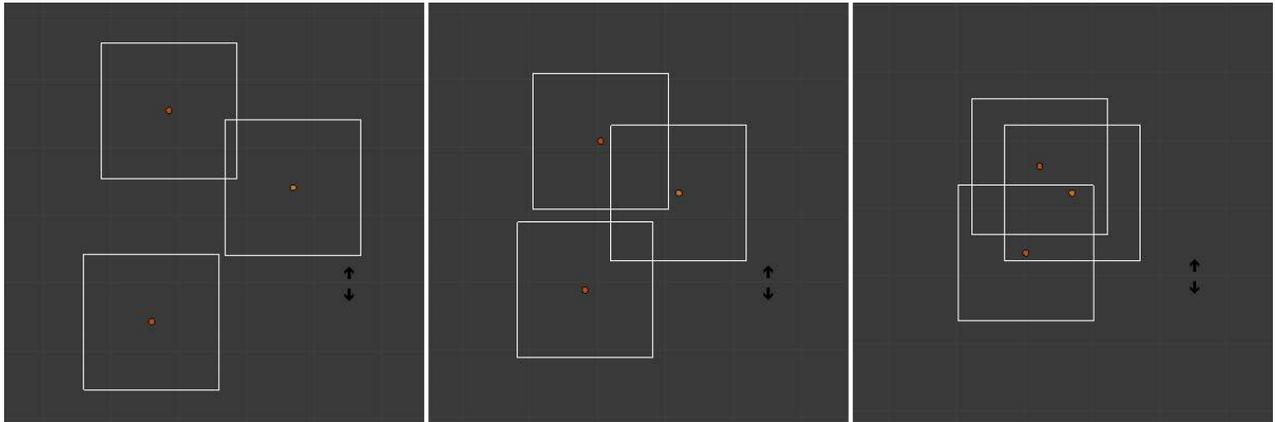


Fig. 2.267: Random Objects being pushed together.

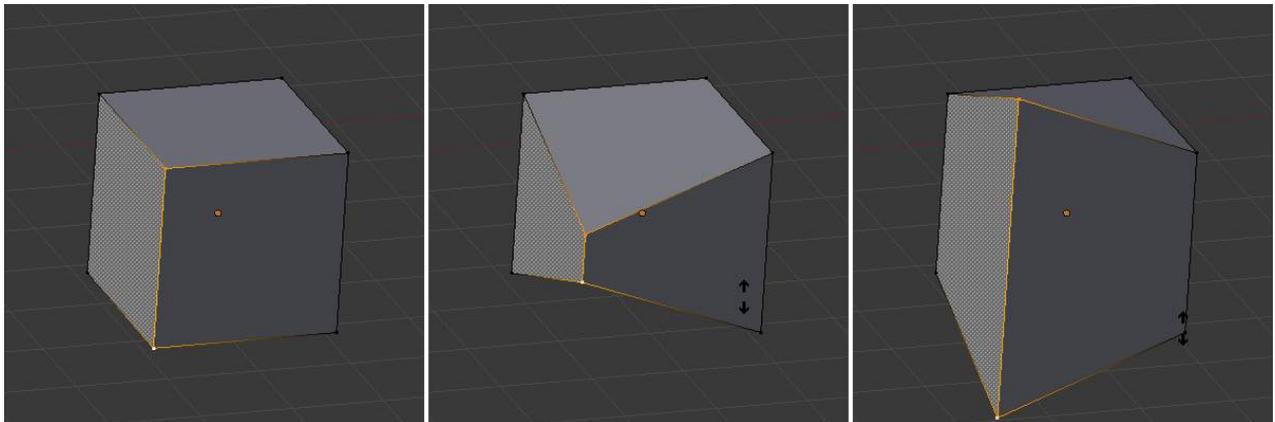


Fig. 2.268: Vertices being pushed together, then pulled apart.

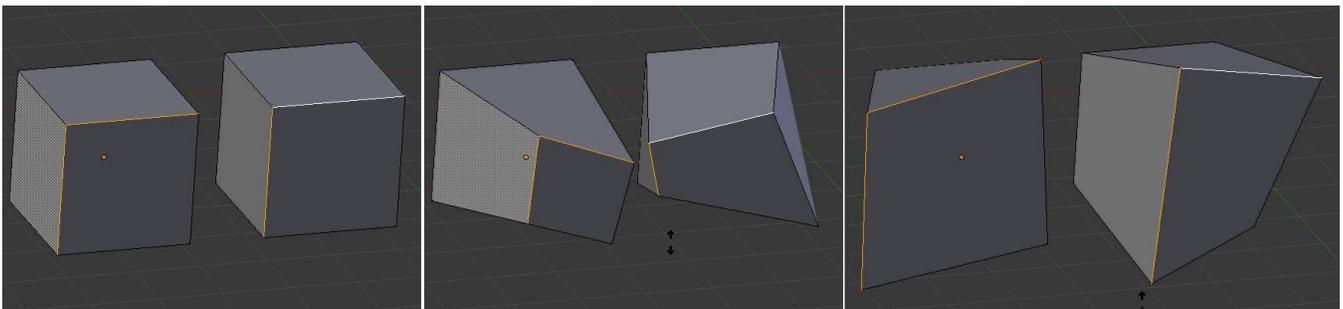


Fig. 2.269: Edges on separate meshes being pushed together, then pulled apart.

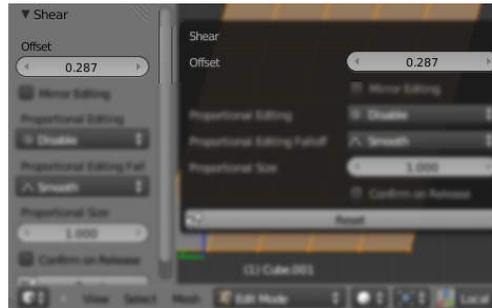


Fig. 2.270: Shear Offset Factor.

**Description** Shearing is a form of movement where parallel surfaces move past one another. During this transform, movement of the selected elements will occur along the horizontal axis of the current view. The axis location will be defined by the *Pivot Point*. Everything that is “above” this axis will move (Shear) in the same direction as your mouse pointer (but always parallel to the horizontal axis). Everything that is “below” the horizontal axis will move in the opposite direction.

[Read more about Pivot Points](#)

**Usage** Select the elements you want to operate on and activate the *Shear* transform function. The *Shear* option can be invoked from the *Object/Mesh/Curve/Surface* → *Transform* → *Shear* menu option or by pressing `Shift-Ctrl-Alt-S`. The amount of movement given to the selection can be determined interactively by moving the mouse or by typing a number. Pressing `Return` will confirm the transformation. The confirmed transformation can be further edited by pressing `F6` or by going into the Toolshelf (T) and altering the Offset slider provided that no other actions take place between the *Shear* transform confirmation and accessing the slider.

Note that the result of the *Shear* transform is also dependant on the number and type of selected elements (Objects, vertices, faces etc). See below for the result of using *Shear* on a number of different elements.

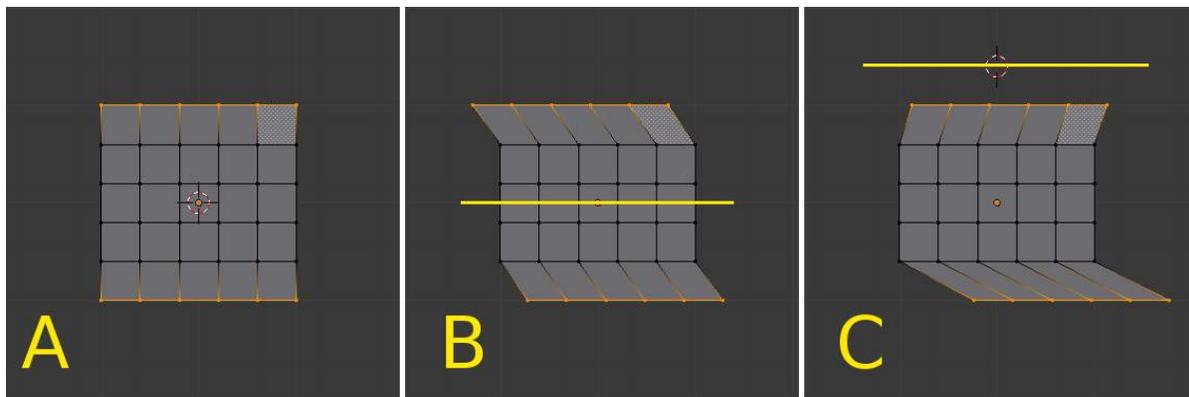


Fig. 2.271: The effects of a Shear transform with different Pivot Points. See the text below for additional information.

The three frames of the image above show the effects of shearing on the selected vertices when the pivot point is altered. In frame B, the *Pivot Point* is set to *Median Point* (indicated by the yellow line) and the mouse was moved to the left during the transform. In frame C, the *Pivot Point* is set to the 3D cursor which is located above the mesh (indicated again by the yellow line). When the mouse is moved to the left during a *Shear* transform the selected vertices are moved to the right as they are below the horizontal axis.

**Tip:** Shear transform magnitude

The magnitude of the *Shear* transform applied to the selected elements is directly proportional to the distance from the horizontal axis. i.e. the further from the axis, the greater the movement.

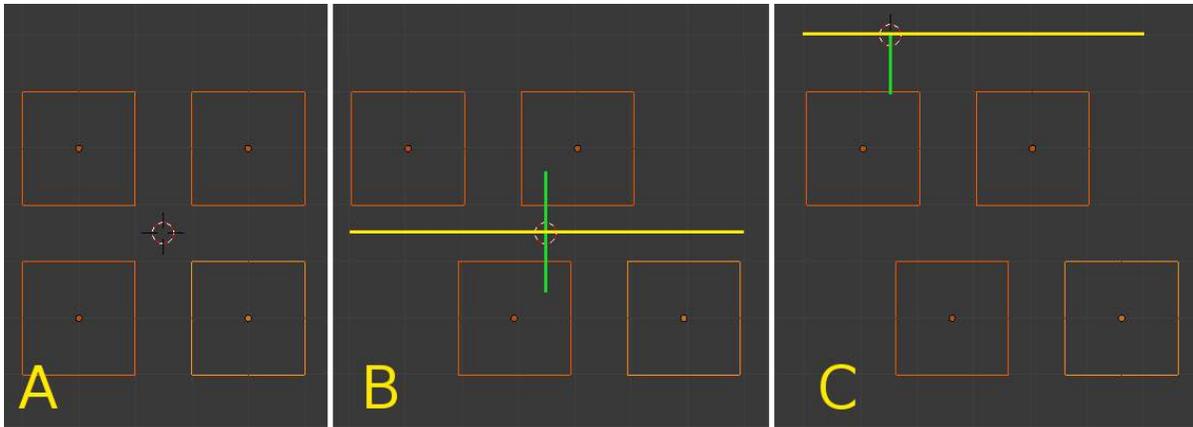


Fig. 2.272: The effects of a Shear transform on Objects with different Pivot Points. See the text below for additional information.

The three frames of the image above show the effects of shearing on the selected Objects when the *Pivot Point* is altered. In frame B, the *Pivot Point* is set to *Median Point* (indicated by the yellow line) and the mouse was moved to the left during the transform. In frame C, the *Pivot Point* is set to the 3D cursor which is located above the Objects (indicated again by the yellow line). When the mouse is moved to the left during a *Shear* transform all of the selected Objects are moved to the right as they are below the horizontal axis. Again, note that the magnitude of the transform is proportional to the distance from the horizontal axis. In this case, the lower Objects move further than the upper ones.

### To Sphere Reference

Mode: *Edit* mode

Menu: *Mesh* → *Transform* → *To Sphere*

Hotkey: *Shift-Alt-S*

**Description** The *To Sphere* transformation will give the selection spherical qualities. The *Suzanne with increasing sphericity* image below shows the results of applying the *To Sphere* transformation to the *Suzanne* mesh.

**Usage** Select the elements you want to operate on and activate the *To Sphere* transform function. The *To Sphere* option can be invoked from the *Mesh* → *Transform* → *To Sphere* menu option or by pressing *Shift-Alt-S*. The amount of sphericity given to the selection can be determined interactively by moving the mouse or by typing a number between 0 and 1. Pressing *Return* will confirm the transformation. The confirmed transformation can be further edited by pressing *F6* or by going into the *Toolshelf* (T) and altering the *Factor* slider provided that no other actions take place between the *To Sphere* transform confirmation and accessing the slider.

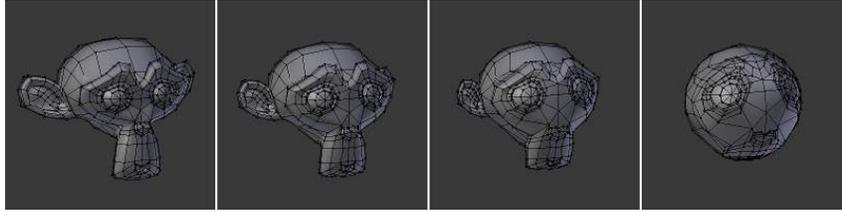


Fig. 2.273: Suzanne with increasing sphericity. The sequence above shows a Suzanne mesh with a 0, 0.25 (25%), 0.5 (50%) and 1 (100%) To Sphere transform applied.

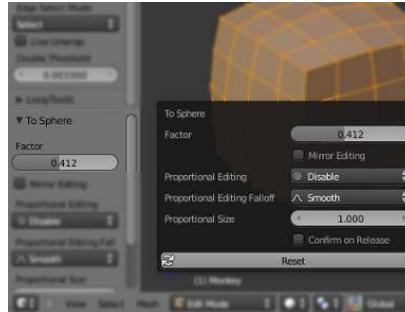


Fig. 2.274: To Sphere Factor.

Note that the result of the *To Sphere* transform is also dependant on the number of selected mesh elements (vertices, faces etc). As can be seen in the below image, the result will be smoother and more spherical when there are more mesh elements available to work with.

The *To Sphere* transform will generate different results depending on the number and arrangement of elements that were selected (as shown by the below image).

## Warp Reference

Mode: *Object* and *Edit* modes

Menu: *Object/Mesh/Curve/Surface* → *Transform* → *Warp*

In *Edit mode*, the *Warp* transformation takes selected elements and warps them around the 3D cursor by a certain angle. Note that this transformation is always dependent on the location of the 3D cursor. The Pivot Point is not taken into account. The results of the *Warp* transformation are also view dependent.

In *Object mode*, the *Warp* transformation takes the selected Objects and causes them to move in an orbit-like fashion around the 3D cursor. Similar to *Edit mode*, the Pivot Point is not taken into account and the results are view dependent.

**Usage** Select the elements you want to operate on and activate the *Warp* transform function. The *Warp* option can be invoked from the *Object/Mesh/Curve/Surface* → *Transform* → *Warp* menu option. The amount of warping given to the selection can be determined interactively by moving the mouse or by typing a number. Pressing `Return` will confirm the transformation. The confirmed transformation can be further edited by pressing `F6` or by going into the Toolshelf (T) and altering the Angle slider provided that no other actions take place between the *Warp* transform confirmation and accessing the slider.

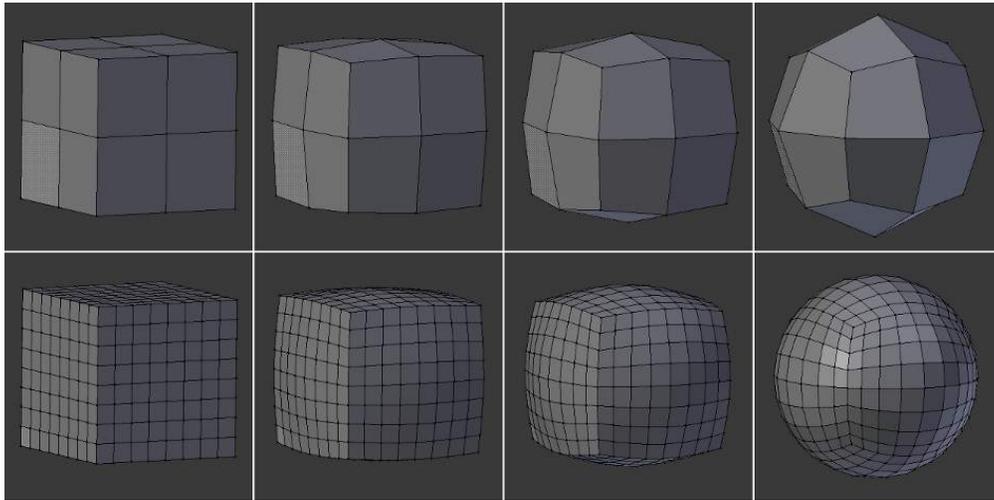


Fig. 2.275: To Sphere applied to cubes with different subdivision levels. In this image sequence, To Sphere was applied to the entire cube at levels of 0, 0.25 (25%), 0.5 (50%) and 1 (100%) respectively.

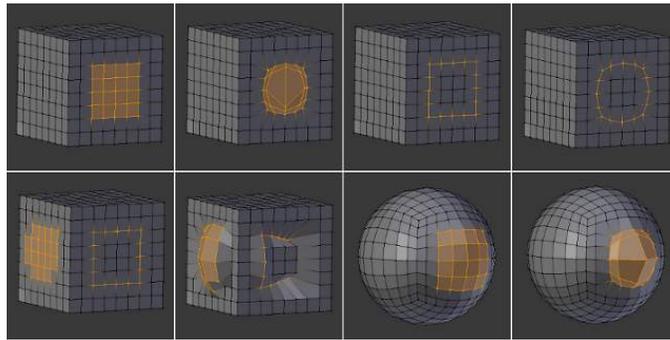


Fig. 2.276: To Sphere applied to different selections.



Fig. 2.277: warp tool options

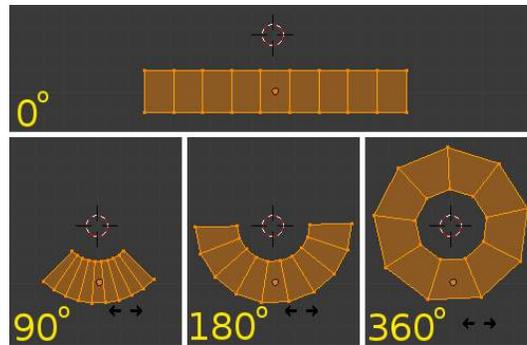


Fig. 2.278: In this example, a plane is warped around the 3D cursor by the indicated number of degrees.

**Cursor position and view** The location of the 3D cursor can be used to alter the results of the *Warp* transformation. As can be seen from the example in this section, the *Warp* radius is dependent on the distance of the cursor from the selected elements. The greater the distance, the greater the radius.

The result of the *Warp* transform is also influenced by your current view. The example in this section shows the results of a 180 degree *Warp* transform applied to the same Suzanne mesh when in different views. A 3D render is also provided for comparison.

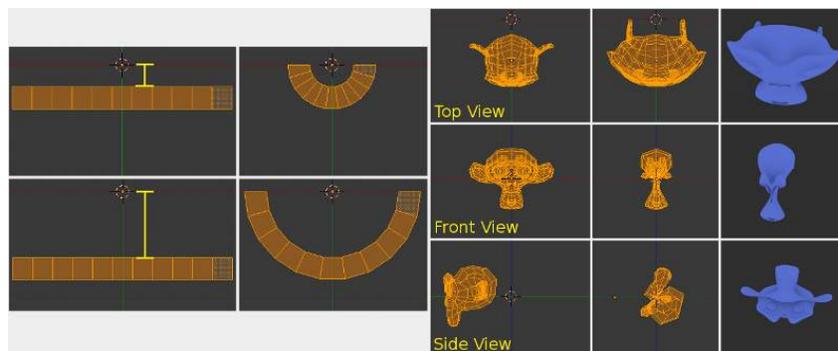


Fig. 2.279: The left side of this image shows how the Warp transform is influenced by the location of the cursor. The right hand side shows the influence of the current view.

---

**Note:** Warping text

If you want to warp text, you will need to convert it from a Text Object to Mesh by pressing `Alt-C` and selecting the *Mesh from Curve/Meta/Surf/Text* option.

---

**Example**

**Bend**

**Reference**

---

Mode: *Object* and *Edit* modes

Menu: *Object/Mesh/Curve/Surface* → *Transform* → *Bend*

Hotkey: `Shift-W`



Fig. 2.280: Text wrapped around logo. This was made by creating the Blender logo and text as separate Objects. The text was converted to a mesh and then warped around the Blender logo.

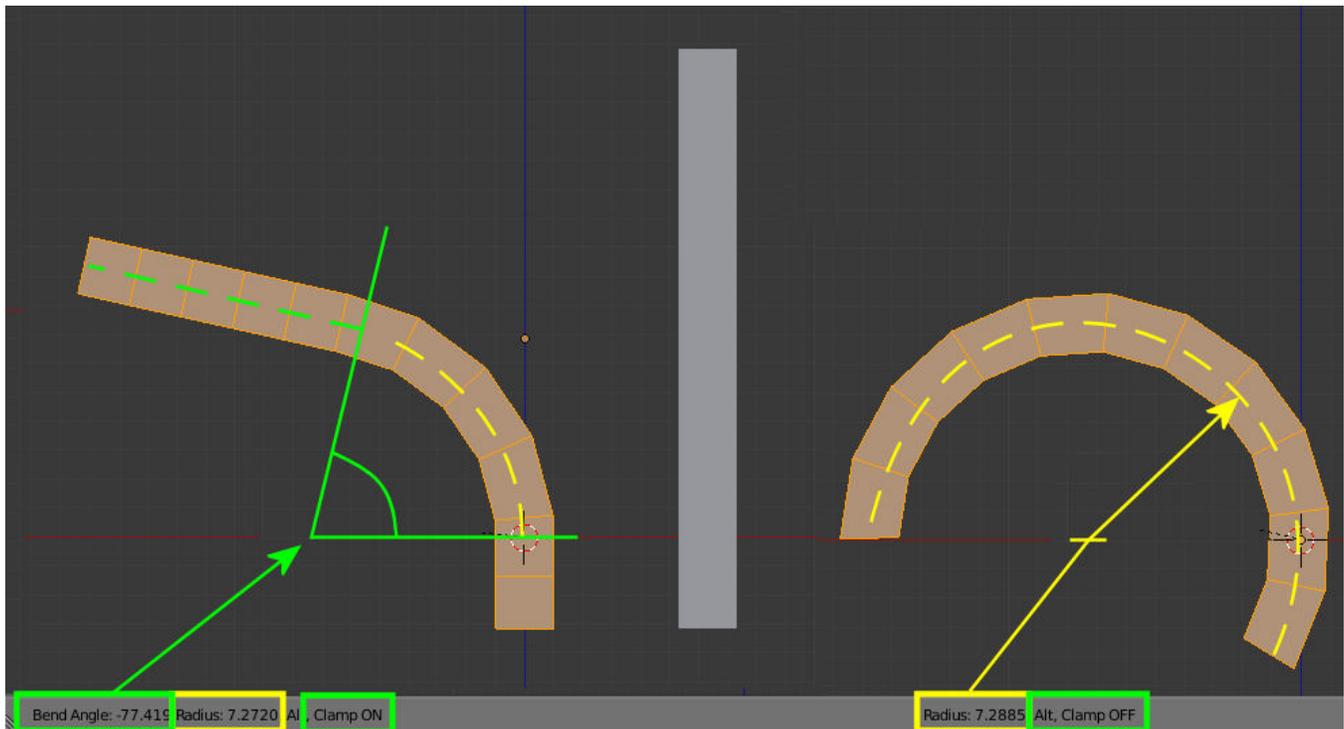


Fig. 2.281: Bend Transform with Clamp ON and OFF

This tool rotates a line of selected elements forming an arc between the mouse-cursor and the 3D-cursor.

**Usage** The bend tool can be used in any case where you might want to bend a shape in two with a gradual transition between both sides.

This may take a little getting used to, the basics are listed below controls are noted here.

- The initial position of the cursors define the axis to bend on.
- The distance of the mouse-cursor to the 3d-cursor controls how sharp the bend will be.
- The relative angle of the mouse-cursor to the initial axis defines the bend angle.

If this seems overly complicated, it's probably best to try the tool where it becomes quickly apparent how the tool reacts to your input.

**Bend Angle** The amount of rotation.

**Radius** The sharpness of the bend.

**Clamp** Normally the arc turns through a clamped rotation angle with the selected elements extended along a tangent line beyond that (see above left). When the clamp is OFF, the arc continues around aligning the selected elements into a circle (right).

When OFF (**Alt**) all selected elements follow a circle, even when outside the segment between the 3d cursor and the mouse.

**Note:** Unlike most other transform modes *Bend* isn't effected by *Pivot Point* or *Transform Orientation*, always using the View Plane instead.

**Hint:** You can turn the bend angle through multiple rotations potentially forming a spiral shape.

---

## Duplicating

**Mesh Duplicating Tools** This section covers mesh editing tools that add additional geometry by duplicating existing geometry in some way.

- Duplicate Geometry.
  - Extrusion.
  - Spin.
  - Screw.
- 

**Note:** Multiple Viewports

When you use one of the duplication tools in the *Mesh Tools* panel, Blender cannot guess which view you want to work in - if you have more than one opened, of course... As the view is often important for these tools, once you have activated one, your cursor turns into a sort of question mark - click with it inside the window you want to use.

---

## Duplicate Reference

---

Mode: *Edit* mode

Menu: *Mesh* → *Duplicate*

Hotkey: Shift-D

---

This tool simply duplicates the selected elements, without creating any links with the rest of the mesh (unlike extrude, for example), and places the duplicate at the location of the original. Once the duplication is done, *only the new duplicated elements are selected*, and you are automatically placed in grab/move mode, so you can translate your copy elsewhere...

In the *Tool Shelf* are settings for *Vector* offset, *Proportional Editing*, *Duplication Mode* (non-functional?), and *Axis Constraints*.

Note that duplicated elements belong to the same [vertex groups](#) as the “original” ones. The same goes for the [material indices](#), the edge’s *Sharp* and *Seam* flags, and probably for the other vertex/edge/face properties...

## Extrude

### Extrude Region Reference

---

Mode: *Edit* mode

Panel: *Mesh Tools* → *Extrude*

Menu: *Mesh* → *Extrude Region*

Hotkey: E or Alt-E

---

One tool of paramount importance for working with meshes is the *Extrude* tool. It allows you to create parallelepipeds from rectangles and cylinders from circles, as well as easily create such things as tree limbs. *Extrude* is one of the most frequently used modeling tools in Blender. It's simple, straightforward, and easy to use, yet very powerful.

The selection is extruded along the common normal of selected faces. In every other case the extrusion can be limited to a single axis by specifying an axis (e.g. `X` to limit to the X axis or `Shift-X` to the YZ plane. When extruding along the face normal, limiting movement to the global Z axis requires pressing `Z` twice, once to disable the face normal Z axis limit, and once to enable the global Z axis limit.



Although the process is quite intuitive, the principles behind *Extrude* are fairly elaborate as discussed below:

- First, the algorithm determines the outside edge-loop of the extrude; that is, which among the selected edges will be changed into faces. By default (see below), the algorithm considers edges belonging to two or more selected faces as internal, and hence not part of the loop.
- The edges in the edge-loop are then changed into faces.
- If the edges in the edge-loop belong to only one face in the complete mesh, then all of the selected faces are duplicated and linked to the newly created faces. For example, rectangles will result in parallelepipeds during this stage.
- In other cases, the selected faces are linked to the newly created faces but not duplicated. This prevents undesired faces from being retained “inside” the resulting mesh. This distinction is extremely important since it ensures the construction of consistently coherent, closed volumes at all times when using *Extrude*.
- When extruding completely closed volumes (like e.g. a cube with all its six faces), extrusion results merely in a duplication, as the volume is duplicated, without any link to the original one.
- Edges not belonging to selected faces, which form an “open” edge-loop, are duplicated and a new face is created between the new edge and the original one.
- Single selected vertices which do not belong to selected edges are duplicated and a new edge is created between the two.

---

### Extrude Individual Reference

Mode: *Edit* mode

Panel: *Mesh Tools* → *Extrude Individual*

Menu: *Mesh* → *Extrude Individual*

Hotkey: `Alt-E`

---

*Extrude Individual* allows you to extrude a selection of multiple faces as individuals, instead of as a region. The faces are extruded along their own normals, rather than their average. This has several consequences: first, “internal” edges (i.e. edges between two selected faces) are no longer deleted (the original faces are).




---

### Extrude Edges and Vertices Only Reference

Mode: *Edit* mode, Vertex and Edge

Hotkey: `Alt-E`

If vertices are selected while doing an extrude, but they do not form an edge or face, they will extrude as expected, forming a *non-manifold* edge. Similarly, if edges are selected that do not form a face, they will extrude to form a face.

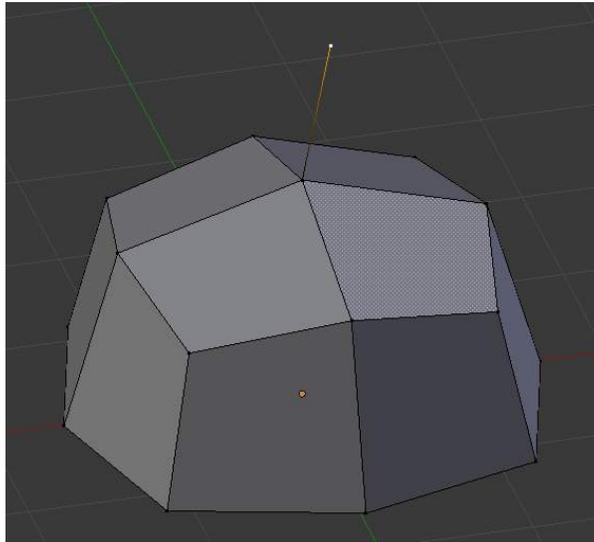


Fig. 2.294: Single vertex extruded

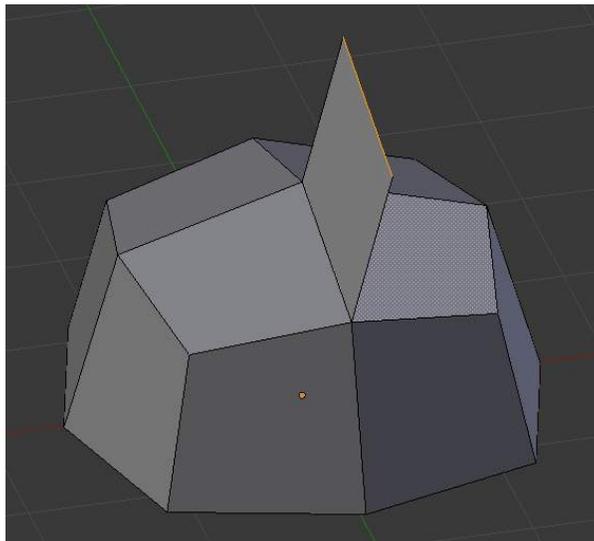


Fig. 2.295: Single edge extruded

When a selection of vertices forms an edge or face, it will extrude as if the edge was selected. Likewise for edges that form a face.

To force a vertex or edge selection to extrude as a vertex or edge, respectively, use `Alt-E` to access the Extrude *Edges Only* and *Vertices Only*.

**Inset**  
**Reference**

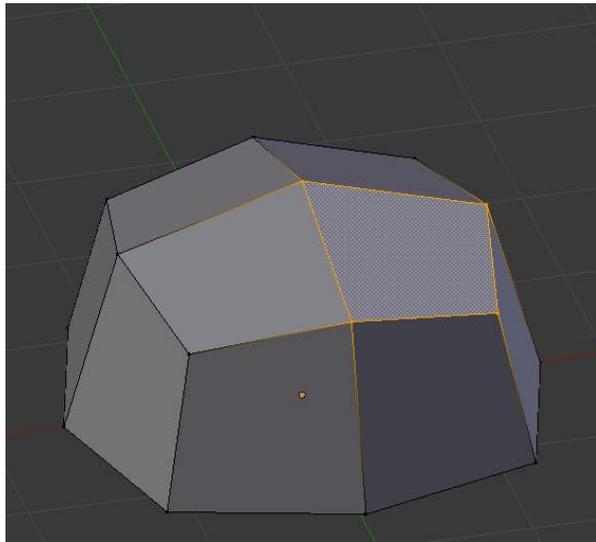


Fig. 2.296: Vertex selected

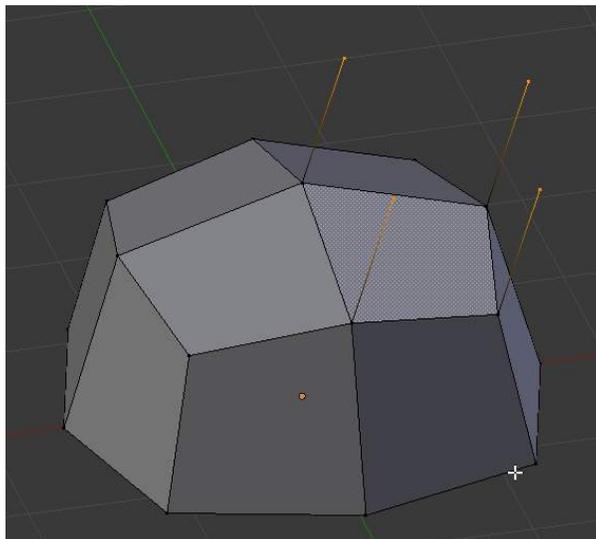


Fig. 2.297: Vertices Only extrude

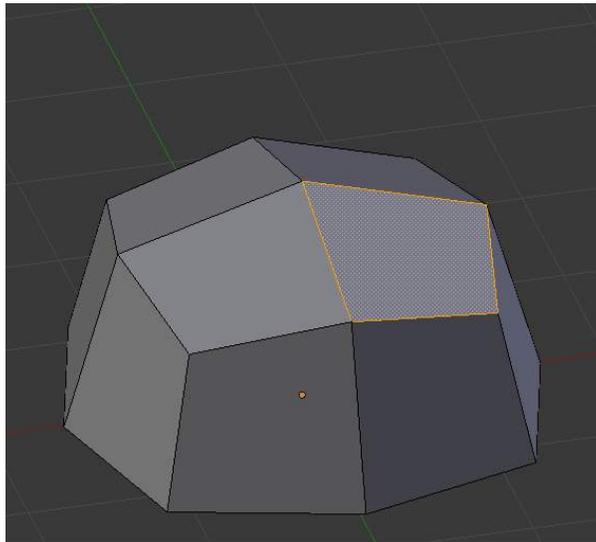


Fig. 2.298: Edge selected

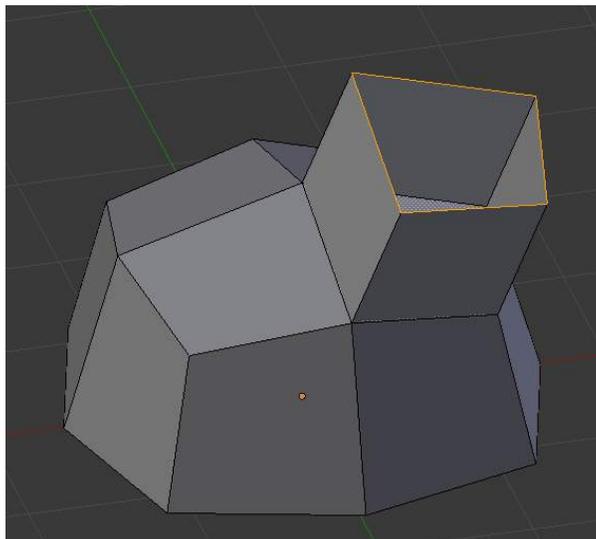


Fig. 2.299: Edge Only extrude

Mode: *Edit* mode

Menu: *Mesh* → *Faces* → *Inset* or *[ctrl][F]* → *Inset*

Hotkey: *I*

This tool takes the currently selected faces and creates an inset of them, with adjustable thickness and depth. The tool is modal, such that when you activate it, you may adjust the thickness with your mouse position. You may also adjust the depth of the inset during the modal operation by holding *Ctrl*.

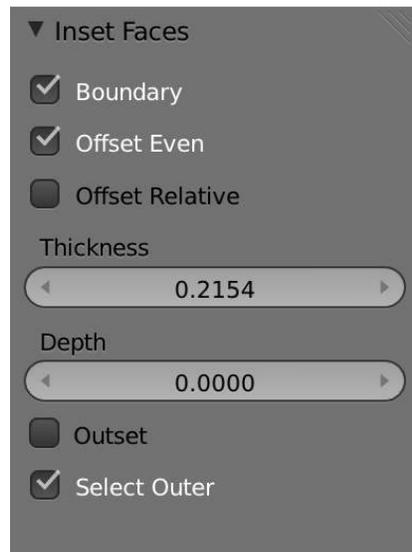


Fig. 2.304: Inset Operator Settings

### Options

**Boundary** Determines whether open edges will be inset or not.

**Offset Even** Scale the offset to give more even thickness.

**Offset Relative** Scale the offset by surrounding geometry.

**Thickness** Set the size of the inset.

**Depth** Raise or lower the newly inset faces to add depth.

**Outset** Create an outset rather than an inset.

**Select Outer** Toggle which side of the inset is selected after operation.

### Mirror

#### Reference

Mode: *Object* and *Edit* modes

Menu: *Object/Mesh* → *Mirror*

Hotkey: *Ctrl-M*

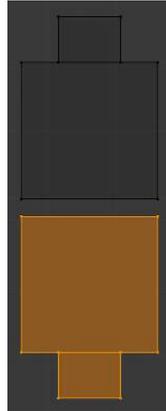


Fig. 2.305: Mirroring a selection.

**Description** Mirroring an Object or Mesh selection will create a reversed version of the selection. The position of the mirrored version of the selection is determined by the *Pivot Point*. A common use of mirroring is to model half an object, duplicate it and then use the mirror transform to create a reversed version to complete the model. Note that mirrored duplicates can also be created with a *Mirror modifier*.

[Read more about the Pivot Point](#)

[Read more about the Mirror Modifier](#)

**Usage** To mirror a selection along a particular global axis press:

Ctrl-M, followed by X, Y or Z.

The image *Mirroring a selection* shows the results of this action after a mesh element has been duplicated.

In Mesh mode, you can mirror the selection on the currently selected Transform Orientation by pressing the appropriate axis key a second time. For example, if the Transform Orientation is set to *Normal*, pressing:

Ctrl-M, followed by X and then X again

will mirror the selection along the X-axis of the *Normal Orientation*.

[Read more about Transform Orientations](#)

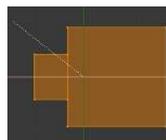


Fig. 2.306: Interactive mirror.

You can alternatively hold the MMB to interactively mirror the object by moving the mouse in the direction of the mirror axis.

## Spin

### Reference

Mode: *Edit* mode

Panel: *Mesh Tools* (*Editing* context)

Use the *Spin* tool to create the sort of objects that you would produce on a lathe (this tool is often called a “lathe”-tool or a “sweep”-tool in the literature, for this reason). In fact, it does a sort of circular extrusion of your selected elements, centered on the 3D cursor, and around the axis perpendicular to the working view...

- The point of view will determine around which axis the extrusion spins...
- The position of the 3D cursor will be the center of the rotation.

Here are its settings:

**Steps** Specifies how many copies will be extruded along the “sweep”.

**Dupli** When enabled, will keep the original selected elements as separated islands in the mesh (i.e. unlinked to the result of the spin extrusion).

**Angle** specifies the angle “swept” by this tool, in degrees (e.g. set it to 180 for half a turn).

**Center** Specifies the center of the spin. By default it uses the cursor position.

**Axis** Specify the spin axis as a vector. By default it uses the view axis.

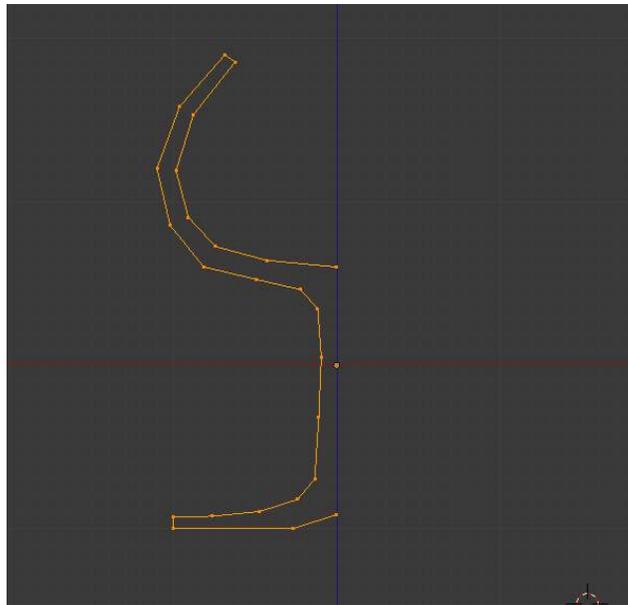


Fig. 2.307: Glass profile.

**Example** First, create a mesh representing the profile of your object. If you are modeling a hollow object, it is a good idea to thicken the outline. (*Glass profile*) shows the profile for a wine glass we will model as a demonstration.

Go to the *Edit* mode and select all the vertices of the Profile with **A**.

We will be rotating the object around the cursor in the top view, so switch to the top view with **Numpad7**.

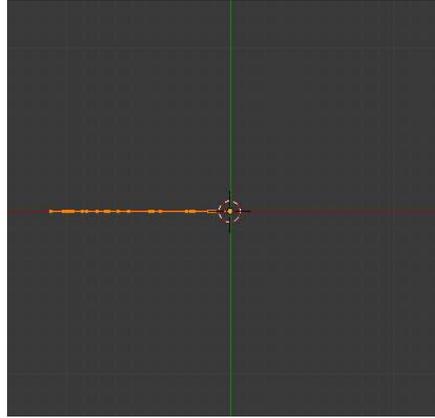


Fig. 2.308: Glass profile, top view in Edit mode, just before spinning.

Place the cursor along the center of the profile by selecting one of the vertices along the center, and snapping the 3D cursor to that location with *Mesh* → *Cursor* → *Selection*. (*Glass profile, top view in Edit mode, just before spinning*) shows the wine glass profile from top view, with the cursor correctly positioned.

Click the *Spin* button. If you have more than one 3D view open, the cursor will change to an arrow with a question mark and you will have to click in the window containing the top view before continuing. If you have only one 3D view open, the spin will happen immediately. (*Spun profile*) shows the result of a successful spin.

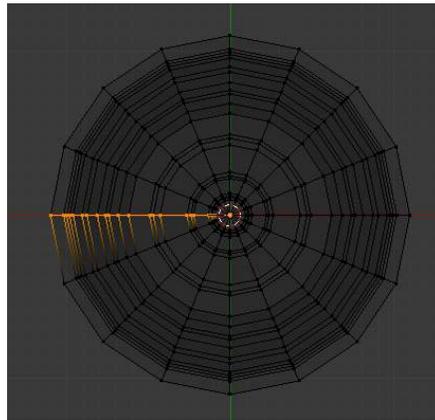


Fig. 2.309: Spun profile using an angle of 360

### Angle

### Dupli

**Merge Duplicates** The spin operation leaves duplicate vertices along the profile. You can select all vertices at the seam with Box select (B) shown in (*Seam vertex selection*) and perform a *Remove Doubles* operation.

Notice the selected vertex count before and after the *Remove Doubles* operation (*Vertex count after removing doubles*). If all goes well, the final vertex count (38 in this example) should match the number of the original profile noted in (*Mesh data - Vertex and face numbers*). If not, some vertices were missed and you will need to weld them manually. Or, worse, too many vertices will have been merged.

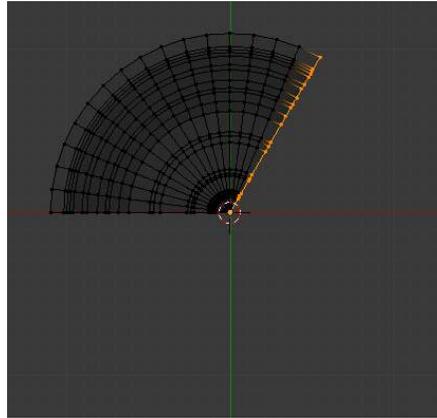


Fig. 2.310: Spun profile using an angle of 120

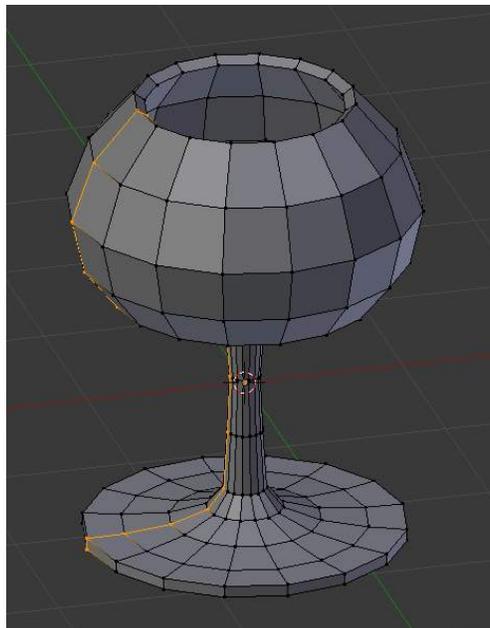


Fig. 2.311: Result of spin operation

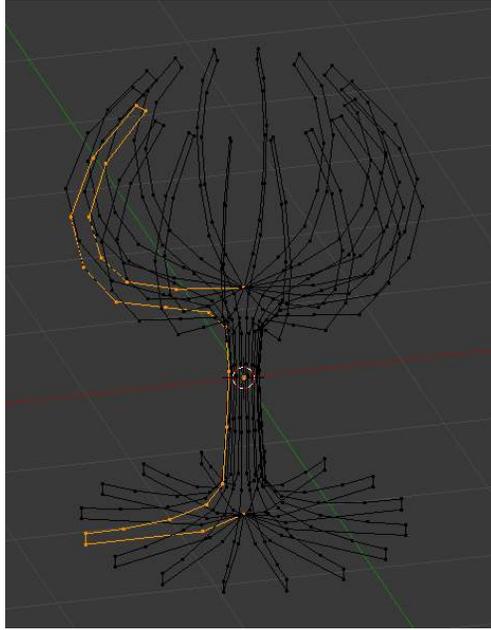


Fig. 2.312: Result of Dupli enabled

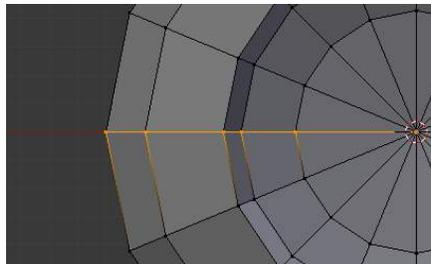


Fig. 2.313: Duplicate vertices

**Note:** Merging two vertices in one

To merge (weld) two vertices together, select both of them by `Shift-RMB` clicking on them. Press `S` to start scaling and hold down `Ctrl` while scaling to scale the points down to 0 units in the X, Y and Z axis. `LMB` to complete the scaling operation and click the *Remove Doubles* button in the *Buttons* window, *Editing* context (also available with `[W]` → *Remove Doubles*).

Alternatively, you can use `[W]` → *Merge* from the same *Specials* menu (or `Alt-M`). Then, in the new pop-up menu, choose whether the merged vertex will be at the center of the selected vertices or at the 3D cursor. The first choice is better in our case!

**Recalculate Normals** All that remains now is to recalculate the normals to the outside by selecting all vertices, pressing `Ctrl-N` and validating *Recalc Normals Outside* in the pop-up menu.

**Screw Tool****Reference**

Mode: *Edit Mode*

Panel: *Edit Mode* → *Mesh Tools* (shortcut `T`) → *Add* → *Screw Button*

**Introduction** The *Screw Tool* is an effective way to revolve a profile, giving similar results to what you would expect from a lathe, with the option to offset the operation to give a screw effect.

You can see some examples of Meshes generated with the *Screw* tool in Fig. 1 - Wood Screw tip done with the screw tool and Fig. 2 - Spring done with the screw tool.

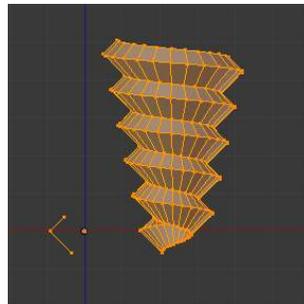


Fig. 2.314: Fig. 1 - Wood Screw tip done with the screw tool

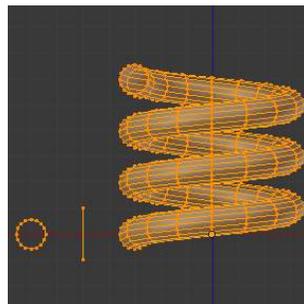


Fig. 2.315: Fig. 2- Spring done with the screw tool

**Description** The *Screw* tool combines a repetitive *Spin* with a translation, to generate a screw-like, or spiral-shaped, object. Use this tool to create screws, springs, or shell-shaped structures (Sea shells, Wood Screw Tips, Special profiles, etc).

The main difference between the Screw Tool and the [Screw Modifier](#) is that the Screw Tool can calculate the angular progressions using the basic profile angle automatically or adjusting the Axis angular vector without using a second modifier (for example, using the Screw Modifier with a Bevel Modifier, Curve Modifier, etc...), resulting in a much cleaner approach for vertex distribution and usage.

This tool works using open or closed profiles, as well as profiles closed with faces. You can use profiles like an open-edge part that is a part of a complete piece, as well as a closed circle or a half-cut sphere, which will also close the profile end.

## Usage

- This tool works only with Meshes.
- In *Edit Mode*, the button for the *Screw* tool operation is located in the *Mesh Tools* Panel, (shortcut T) → Add → Screw Button.
- To use this tool, you need to create at least one open profile or line to be used as a vector for the height, angular vector and to give Blender a direction.
- The *Screw* function uses two points given by the open line to create an initial vector to calculate the height and basic angle of the translation vector that is added to the “Spin” for each full rotation (see examples below). If the vector is created with only two vertices at the same **X**, **Y** and **Z** location (which won’t give Blender a vector value for height), this will create a normal “Spin”.
- Having at least one vector line, you can add other closed support profiles that will follow this vector during the extrusions (See limitations).
- The direction of the extrusions is calculated by two determinant factors, your point of view in Global Space and the position of your cursor in the 3DView Space using Global coordinates.
- The profile and the vector must be fully selected in *Edit Mode* before you click the *Screw Button* (See Limitations.)
- When you have the vector for the open profile and the other closed profiles selected, click the *Screw Button*.

**Limitations** There are strict conditions about your profile selection when you want to use this tool. You must have at least one open line or open profile, giving Blender the starting Vector for extrusion, angular vector and height. (e.g. a simple edge, a half circle, etc...). You need only to ensure that at least one reference line has two “free” ends. If two open Lines are given, Blender won’t determine which of them is the vector, and will then show you an error message, “*You have to select a string of connected vertices too*”. You need to select all of the profile vertices that will participate in the *Screw Tool* operation; if they are not properly selected, Blender will also show you the same message.

Note that the open line is always extruded, so if you only use it to “guide” the screw, you will have to delete it after the tool completion (use linked-selection, `CTRL-L`, to select the whole extrusion of the open line).

If there is any problem with the selection or profiles, the tool will warn you with the error message: “*You have to select a string of connected vertices too*” as seen in Fig. 3 and 4, both in the info Window and at the place where you clicked to start performing the operation (when you click the Screw Button).



Fig. 2.316: Fig. 3 - Screw Error message in the Header of the Info Window

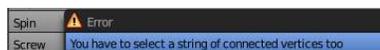


Fig. 2.317: Fig. 4 - Error message when clicking in the Screw Tool with an incorrect or bad selection

You may have as many profiles as you like (like circles, squares, and so on) - Note that not all vertices in a profile need to be in the same plane, even if this is the most common case. You may also have other, more complex, selected closed islands, but they have to be closed profiles because Blender will seek for only one open profile for the translation, height and angular vector. Some closed meshes that overlap themselves may not screw correctly (for example: Half UVsphere = OK, more than half = could cause the Screw Tool to have wrong behavior or errors), and profiles that are closed with faces (like a cone or half sphere) will be closed automatically at their ends, like if you were extruding a region.

**Tip:** Simple way to not result in error

Only one open Profile, all of the others can be closed, avoid volumes and some profiles closed with faces...

**Options** This tool is an interactive and modal tool, and only works in the *Edit Mode*.

Once you click in the *Screw* tool in the Mesh Tools Panel, Blender will enter in the *Screw* interactive mode, and the Operator Panel at the end of the Mesh Tools Panel will be replaced so you can adjust the values explained below. To show the Mesh Tools Panel, use the shortcut **T** in the Edit Mode of the 3D View Window.

Once you perform any other operation, Blender leaves the interactive mode and accepts all of the values. Because it's modal, you can't return to the interactive mode after completing/leaving the operation or changing from *Edit Mode* to *Object Mode*. If you want to restart the operation from its beginning, you can press **Ctrl-Z** at any time in *Edit Mode*.

- The basic location of the cursor at the point of view (using Global coordinates) will determine around which axis the selection is extruded and spun at first (See Fig. 6 - Cursor Basic Location - Transform Panel). Blender will copy your cursor location coordinates to the values present in the *Center* values of the *Screw* interactive Panel. Depending on the Global View position, Blender will automatically add a value of **1** to one of the Axis Vectors, giving the profiles a starting direction for the Screw Operation and also giving a direction for the extrusions. (See examples below.)
- The position of the 3D cursor will be the starting center of the rotation. Subsequent operations (e.g. pressing the Screw button again), will start from the last selected element. Continuous operations without changing the selection will repeat the operation continuously from the last point.

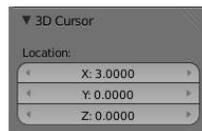


Fig. 2.318: Fig. 6 - Cursor Basic Location - Transform Panel

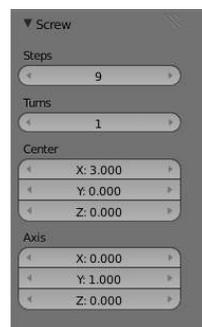


Fig. 2.319: Fig. 7 - Screw Interactive Panel - Mesh Tools Panel (Edit Mode)

**Center** These numeric fields specify the center of the spin. When the tool is called for the first time, it will copy the X, Y and Z location (Global Coordinates) of the cursor presently in the 3D View to start the operation. You can specify the cursor coordinates using the Transform Panel in 3D View, using shortcut **T** to toggle the Panel, and typing in the 3D Cursor

Location coordinates. You can adjust these coordinates interactively and specify another place for the spin center during the interactive session. (See Fig. 7 - Screw Interactive Panel - Mesh Tools Panel (Edit Mode))

**Steps** This numeric field specifies how many extrusion(s) will be done for each 360 turn. The steps are evenly distributed by dividing 360 by the number of steps given. The minimum value is 3; the maximum is 256 (See Fig. 7)

**Turns:** This numeric field specifies how many turns will be executed. Blender will add a new full 360 turn for each incremental number specified here. The minimum value is 1; the maximum is 256. (See Fig. 7)

**Axis** These 3 numeric fields vary from  $-1.0$  to  $1.0$  and are clamped above those limits. These values correspond to angular vectors from  $-90$  to  $90$  degrees. Depending on the position where you started your cursor location and Object operation in the viewport and its axis positions in Global View space and coordinates, Blender will give the proper Axis vector a value of 1, giving the angular vector of the profile a starting direction and giving the extrusions a starting direction based on your view. Blender will let you adjust your axis angular vectors and you can tweak your object such that you can revert the direction of the screw operation (by reverting the angular vector of the height), meaning you can revert the clockwise and counterclockwise direction of some operations, and also adjust the angular vectors of your profile, bending it accordingly. (See Fig. 7)

## Examples

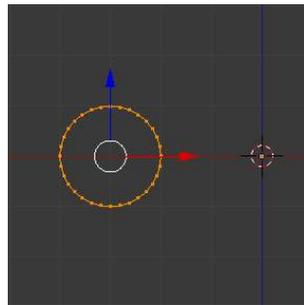


Fig. 2.320: Fig. 8 - Circle placed at X -3,0,0

### The Spring example

- Open Blender and delete the default Cube.
- Change from perspective to orthographic view using shortcut `Numpad5`.
- Change your view from *User Ortho* to *Front Ortho*, using the shortcut `Numpad1`. You will see the X (red) and Z (blue) coordinate lines.
- In case you have moved your cursor by clicking anywhere in the screen, again place your cursor at the Center, using the shortcut `Shift-S` choosing *Cursor to Center* or the Transform Panel, placing your cursor at  $(0, 0, 0)$  typing directly into the Cursor 3D Location.
- Add a circle using shortcut `Shift-A` and choosing `-> Mesh -> Circle`.
- Rotate this circle using the shortcut `R-X` and typing `90` and `Return`.
- Apply the Rotation using `Ctrl-A` and choosing *Rotation*
- Grab and move this circle to the left 3 Blender Units on the X Axis; you can use the shortcut `Ctrl` while grabbing with the mouse using the standard transform widgets (clicking on the red arrow shown with the object and grabbing while using shortcut `Ctrl` until the down left info in the 3D View marks `D. -3.0000 (3.0000) Global`), or press the shortcut `G-X` and typing `-3` and `Return`. You can use the Transform Panel (toggled with the shortcut `T`, and type `-3` and `Return` in the Location too. (See the Fig. 8 - Circle placed at X -3,0,0).
- You will have to scale your circle using the shortcut `S` and typing `.5`, then `Return`.

- Now enter *Edit Mode* using shortcut `Tab`.
- De-select all vertices using the shortcut `A`.

Now we will create a height vector for Blender:

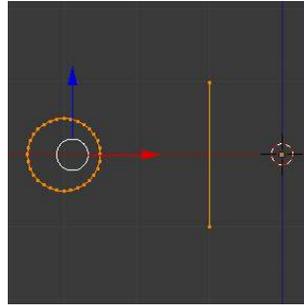


Fig. 2.321: Fig. 9 - Profile and vector created

- Press `Ctrl` and Left click `LMB` near the circle, in more or less at the light grey line of the square above the circle, and, while still pressing `Ctrl`, Left Click `LMB` again in the grey line below the circle. You have created two vertices and an Edge, which Blender will use as the first height and angle vector.
- Now, in the Transform Panel, in the median, clicking in the Global coordinates, for the **X**, **Y**, and **Z** coordinates, put **(-2, 0, -1)**.
- Right Click `RMB` in the other vertex, and again, type its coordinates for **X**, **Y** and **Z** to **(-2, 0, 1)**. This will create a straight vertical line with 2 Blender units of Height.
- De-select and select everything again with the shortcut `A`. (See Fig. 9 - Profile and vector created)
- Place again your cursor at the center. (Repeat step 2)
- At this point, we will save this Blender file to recycle the Spring for another exercise; click with `LMB` in *File*, it is placed at the header of the Info Window, (At the top left side), and choose *Save as*. Our suggestion is to name it *Screw Spring Example.blend* and click in *Save as Blender file*. You can also use the shortcut `Shift-Ctrl-S` to open the File Browser Window in order to save your Blender file.
- Click Screw and adjust the Steps and Turns as you like and we have a nice spring, but now here comes the interesting part!

**Clockwise and Counterclockwise using the Spring Example** Still in the interactive session of the *Screw Tool*, you will see that the **Z** Axis Value of the *Screw* Panel is set to **1.000**. Left click `LMB` in the middle of the Value and set this value to **-1.000**. At first, the Spring was being constructed in a Counterclockwise direction, and you reverted the operation **180** degrees in the **Z** Axis. This is because you have changed the angular vector of the height you have given to Blender to the opposite direction (remember, **-90** to **90** = **180** degrees ?). See Fig. 10 - Counterclockwise direction and Fig. 11 - Flipped to Clockwise direction.

It's also important to note that this vector is related to the same height vector axis used for the extrusion and we have created a parallel line with the **Z** Axis, so, the sensibility of this vector is in practical sense reactive only to negative and positive values because it's aligned with the extrusion axis. Blender will clamp the positive and negative to its maximum values to make the extrusion follow a direction, even if the profile starts reverted. The same rule applies to other Global axes when creating the Object for the *Screw Tool*; this means if you create your Object using the Top View (Shortcut `Numpad7` with a straight parallel line following another axis (for the Top View, the **Y** Axis), the vector that gives the height for extrusion will also change abruptly from negative to positive and vice versa to give the extrusion a direction, and you will have to tweak the corresponding Axis accordingly to achieve the Clockwise and Counterclockwise effect.

---

**Note:** Vectors that aren't parallel with Blender Axis

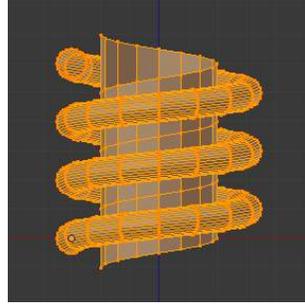


Fig. 2.322: Fig. 10 - Counterclockwise direction

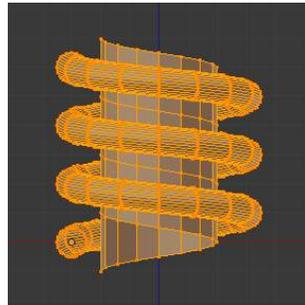


Fig. 2.323: Fig. 11 - Flipped to Clockwise direction.

The high sensibility for the vector doesn't apply to vectors that give the Screw Tool a starting angle (Ex: any non-parallel vector), meaning Blender won't need to clamp the values to stabilize a direction for the extrusion, as the inclination of the vector will be clear for Blender and you will have the full degree of freedom to change the vectors. Our example is important because it only changes the direction of the profile without the tilt and/or bending effect, as there is only one direction for the extrusion, parallel to one of the Blender Axes

**Bending the Profiles using the Spring Example** Still using the Spring Example, we can change the remaining vector for the angles that aren't related to the extrusion Axis of our Spring, thus bending our spring with the remaining vectors and creating a profile that will also open and/or close because of the change in starting angular vector values. What we are really doing is changing the starting angle of the profile prior to the extrusions. It means that Blender will connect each of the circles inclined with the vector you have given. Below we show two bent Meshes using the Axis vectors and the Spring example. See Fig. 12 and Fig. 13. These two Meshes generated with the *Screw* tool were created using the Top Ortho View.

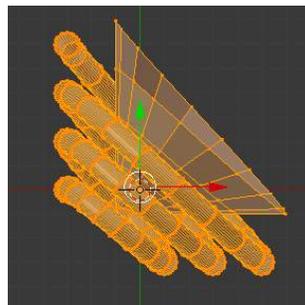


Fig. 2.324: Fig. 12 - Bended Mesh, Example 1 - The Axis will give the profile a starting vector angle

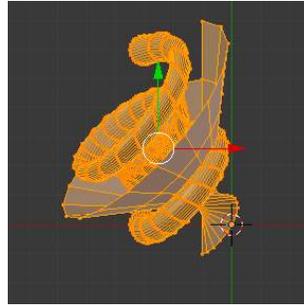


Fig. 2.325: Fig. 13 - Bended Mesh Example 2 - The vector angle is maintained along the extrusions

**Creating perfect Screw Spindles** Using the Spring Example, it's easy to create perfect Screw Spindles (like the ones present in normal screws that we can buy in hardware stores). Perfect Screw Spindles use a profile with the same height as its vector, and the beginning and ending vertex of the profile are placed at a straight parallel line with the axis of extrusion. The easiest way of achieving this effect is to create a simple profile where the beginning and ending vertices create a straight parallel line. Blender won't take into account any of the vertices present in the middle but those two to take its angular vector, so the spindles of the screw (which are defined by the turns value) will assembly perfectly with each other.

- Open Blender and click in *File* located at the header of the Info Window again, choose *Open Recent* and the file we saved for this exercise. All of the things will be placed exactly the way you saved before. Choose the last saved Blender file; in the last exercise, we gave it the name *Screw Spring Example.blend*.
- Press the shortcut **A** to de-select all vertices.
- Press the shortcut **B**, and Blender will change the cursor; you're now in border selection mode.
- Open a box that selects all of the circle vertices except the two vertices we used to create the height of the extrusions in the last example.
- Use the shortcut **X** to delete them.
- Press the shortcut **A** to select the remaining vertices.
- Press the shortcut **W** for the *Specials Menu*, and select *Subdivide*
- Now, click with the Right Mouse button at the middle vertex.
- Grab this vertex using the shortcut **G-X**, type **-1** and **Return**. See Fig. 14 - Profile for a perfect screw spindle.
- At this point, we will save this Blender file to recycle the generated Screw for another exercise; click with **LMB** in *File* – it is in the header of the Info Window (at the top left side), and choose *Save as*. Our suggestion is to name it *Screw Hardware Example.blend* and click in *Save as Blender file*. You can also use the shortcut **Shift-Ctrl-S** to open the File Browser Window in order to save your Blender file.
- Press shortcut **A** twice to de-select and select all vertices again.
- Now press **Screw**.
- Change **Steps** and **Turns** as you like. Fig. 15 - Generated Mesh - Shows you an example of the results.

Here, in Fig. 16 and Fig. 17, we show you an example using a different profile, but maintaining the beginning and ending vertices at the same position. The generated mesh looks like a medieval ramp!

As you can see, the Screw spindles are perfectly assembled with each other, and they follow a straight line from top to bottom. You can also change the Clockwise and Counterclockwise direction using this example, to create right and left screw spindles. At this point, you can give the screw another dimension, changing the Center of the Spin Extrusion, making it more suitable to your needs or calculating a perfect screw and merging its vertices with a cylinder, modeling its head, etc.

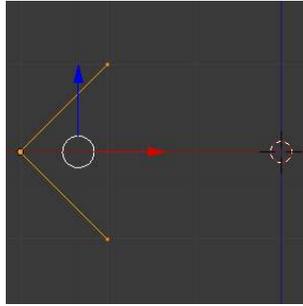


Fig. 2.326: Fig. 14 - Profile for a perfect screw spindle. The starting and ending vertices are forming a parallel line with the Blender Axis

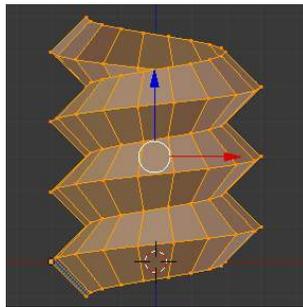


Fig. 2.327: Fig. 15 - Generated Mesh. You can use this technique to perform normal screw modeling.

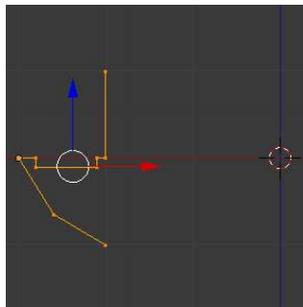


Fig. 2.328: Fig. 16 - Profile with starting and ending vertices forming a parallel line with the Blender Axis

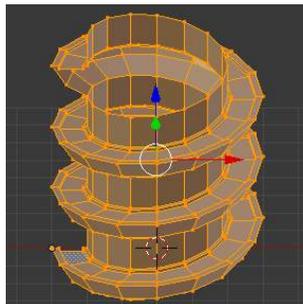


Fig. 2.329: Fig. 17 - Generated Mesh with the profile at the left. We have inclined the visualization a bit.

**A Screw Tip** As we have explained before, the *Screw* tool generates clean and simple meshes to deal with; they are light, well-connected and are created with very predictable results. This is due to the Blender calculations taking into account not only the height of the vector, but also its starting angle. It means that Blender will connect the vertices with each other in a way that they follow a continuous cycle along the extruded generated profile.

In this example, you will learn how to create a simple Screw Tip (like the ones we use for wood; we have shown an example at the beginning of this page). To make this new example as short as possible, we will recycle our last example (again).

- Open Blender and click in *File* located in the header of the Info Window again; choose *Open Recent* and the file we saved for this exercise. All of the things will be placed exactly the way you saved before. Choose the last saved Blender file; in the last exercise, we gave it the name *Screw Hardware Example.blend*.
- Grab the upper vertex and move a bit to the left, but no more than you have moved your last vertex. (See Fig. 18 - Profile With Starting Vector Angle)
- Press the shortcut `A` twice to de-select and select all.
- Press the shortcut `Shift-S` and select *Cursor to Center*
- Press *Screw*.

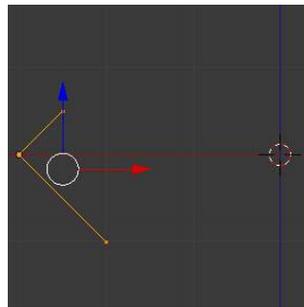


Fig. 2.330: Fig. 18 - Profile With Starting Vector Angle

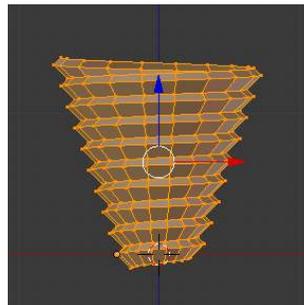


Fig. 2.331: Fig. 19 - Generated Mesh with the Profile

As you can see in Fig. 19, Blender follows the basic angular vector of the profile, and the profile basic angle determines whether the extruded subsequent configured turns will open or close the resulting mesh following this angle. The vector of the extrusion angle is determined by the starting and ending Vertex of the profile.

## Subdividing

**Mesh Subdividing Tools** Subdividing adds resolution by cutting existing faces and edges into smaller pieces. There are several tools that allow you to do this:

**Subdivide** Divide a face or edge into smaller units, adding resolution.

- Loop Subdivide** Insert a loop of edges between existing ones
- Vertex Connect** Connects selected vertices with edges that split faces.
- Knife Subdivide** Cut edges and faces interactively
- Bevel** Subdivides edges or vertices, making them faceted or rounded

## Subdivide

---

### Reference

Mode: *Edit mode*

Panel: *Mesh Tools (Editing context)*

Menu: *Mesh → Edges → Subdivide, Specials → Subdivide/Subdivide Smooth*

Hotkey: *[W] → [pad1]/[pad2]*

---

Subdividing splits selected edges and faces by cutting them in half or more, adding necessary vertices, and subdividing accordingly the faces involved, following a few rules, depending on the settings:

- When only one edge of a face is selected (Tri mode), triangles are subdivided into two triangles, and quads, into three triangles.
- When two edges of a face are selected:
  - If the face is a triangle, a new edge is created between the two new vertices, subdividing the triangle in a triangle and a quad.
  - If the face is a quad, and the edges are neighbors, we have **three** possible behaviors, depending on the setting of *Corner Cut Type* (the drop-down menu next to the *Subdivide* button, in *Mesh Tools* panel) See below for details.
  - If the face is a quad, and the edges are opposite, the quad is just subdivided in two quads by the edge linking the two new vertices.
- When three edges of a face are selected:
  - If the face is a triangle, this means the whole face is selected - it is then sub-divided in four smaller triangles.
  - If the face is a quad, first the two opposite edges are subdivided as described above. Then, the “middle” edge is subdivided, affecting its new “sub-quad” as described above for only one edge.
- When four edges of a face (a quad) are selected, the face is subdivided into four smaller quads.

**Options** These options are available in the *Tool Panel* after running the tool;

**Number of Cuts** Specifies the number of cuts per edge to make. By default this is 1, cutting edges in half. A value of 2 will cut it into thirds, and so on.

**Smoothness** Displaces subdivisions to maintain approximate curvature, The effect is similar to the way the subdivision modifier might deform the mesh.



**Quad/Tri Mode** Forces subdivide to create triangles instead of ngons, simulating old behavior (see examples below)

**Corner Cut Type** This drop-down menu controls the way quads with only two adjacent selected edges are subdivided

**Fan** the quad is sub-divided in a fan of four triangles, the common vertex being the one opposite to the selected edges.

**Innervert** (i.e. “inner vertex”), The selected edges are sub-divided, then an edge is created between the two new vertices, creating a small triangle. This edge is also sub-divided, and the “inner vertex” thus created is linked by another edge to the one opposite to the original selected edges. All this results in a quad sub-divided in a triangle and two quad.

**Path** First an edge is created between the two opposite ends of the selected edges, dividing the quad in two triangles. Then, the same goes for the involved triangle as described above.

**Straight Cut** Currently non functioning...



**Fractal** Displaces the vertices in random directions after the mesh is subdivided



**Along Normal** Causes the vertices to move along the their normals, instead of random directions

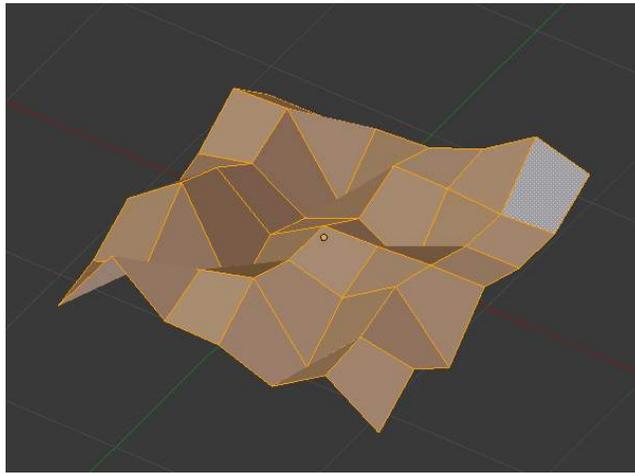


Fig. 2.350: Along normal set to 1

**Random Seed** Changes the random seed of the noise function, producing a different result for each seed value.

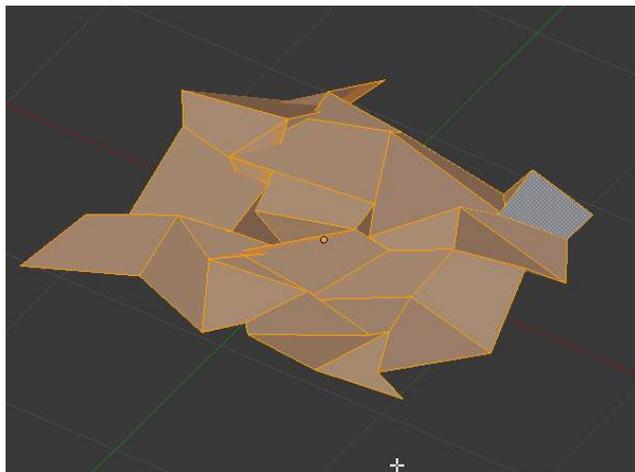


Fig. 2.351: Same mesh with a different seed value

**Examples** Below are several examples illustrating the various possibilities of the *Subdivide* and *Subdivide Multi* tools. Note the selection after subdivision.

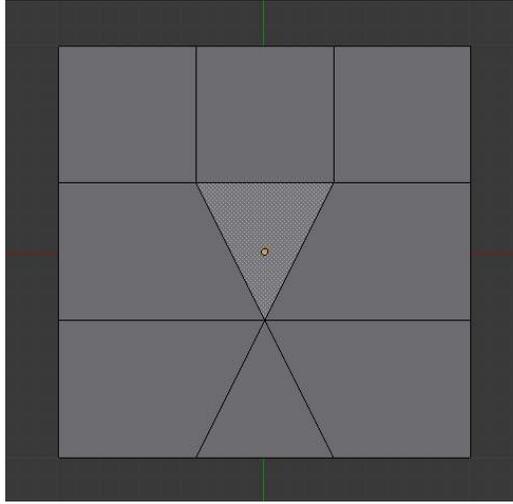


Fig. 2.352: The sample mesh.

One Edge

Two Tri Edges

Two Opposite Quad Edges

Two Adjacent Quad Edges



Three Edges

Tri

Quad/Four Edges

Multicut

Loop Subdivide  
Reference

Mode: *Edit* mode

Panel: *Editing* context → *Mesh Tools*

Hotkey: Ctrl-R

*Loop Cut* splits a loop of faces by inserting a new edge loop intersecting the chosen edge. The tool is interactive and has two steps:

### Usage

**Pre-visualizing the cut** After the tool is activated, move the cursor over a desired edge. The cut to be made is marked with a magenta colored line as you move the mouse over the various edges. The to-be-created edge loop stops at the poles (tris and ngons) where the existing face loop terminates.

**Sliding the new edge loop** Once an edge is chosen via LMB, you can move the mouse along the edge to determine where the new edge loop will be placed. This is identical to the *Edge Slide* tool. Clicking LMB again confirms and makes the cut at the pre-visualized location, or clicking RMB forces the cut to exactly 50%. This step is skipped when using multiple edge loops (see below)

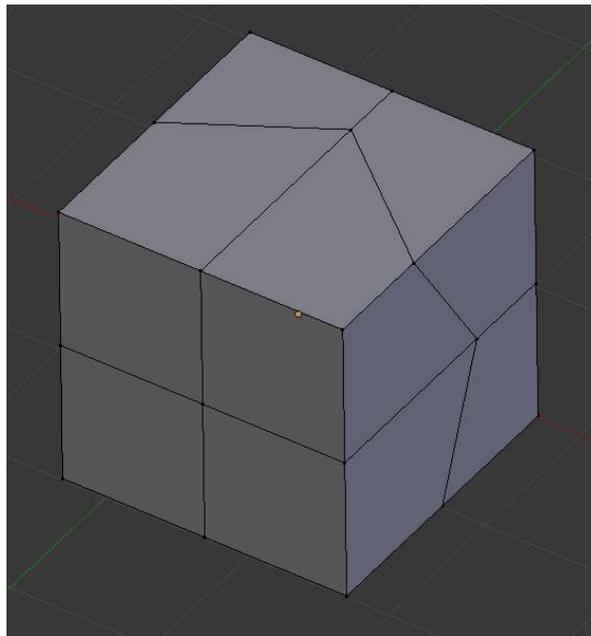


Fig. 2.383: mesh before inserting edge loop

**Options** Options are only available while the tool is in use, and are displayed in the 3d view header

**Even E** Only available for single edge loops. This matches the shape of the edge loop to one of the adjacent edge loops. (See *Edge Slide* tool for details)

**Flip F** When Even is enabled, this flips the target edge loop to match. (See *Edge Slide* tool for details)

**Number of Cuts Wheel or NumpadPlus / NumpadMinus** After activating the tool, but before confirming initial loop location, you can increase and decrease the number of cuts to create, by entering a number with the keyboard, scrolling Wheel or using NumpadPlus and NumpadMinus. Note that when creating multiple loops, these cuts are uniformly distributed in the original face loop, and *you will not be able to control their positions*.

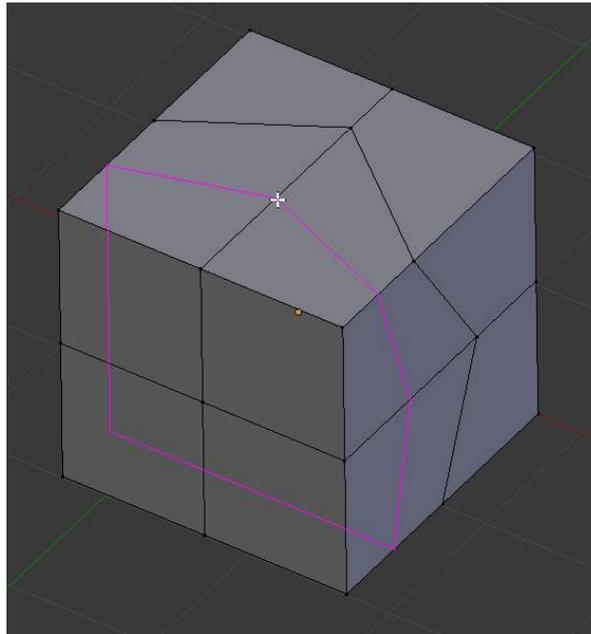


Fig. 2.384: Preview of edge loop location

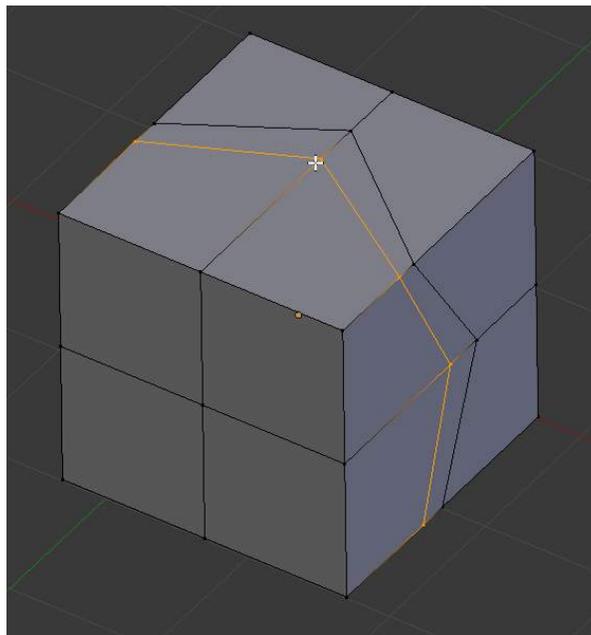


Fig. 2.385: Interactive placement of edge loop between adjacent loops

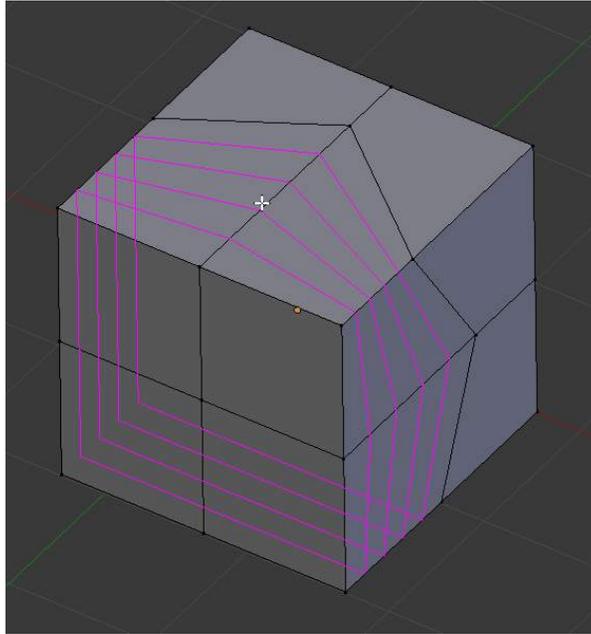


Fig. 2.386: Preview of multiple edge loops

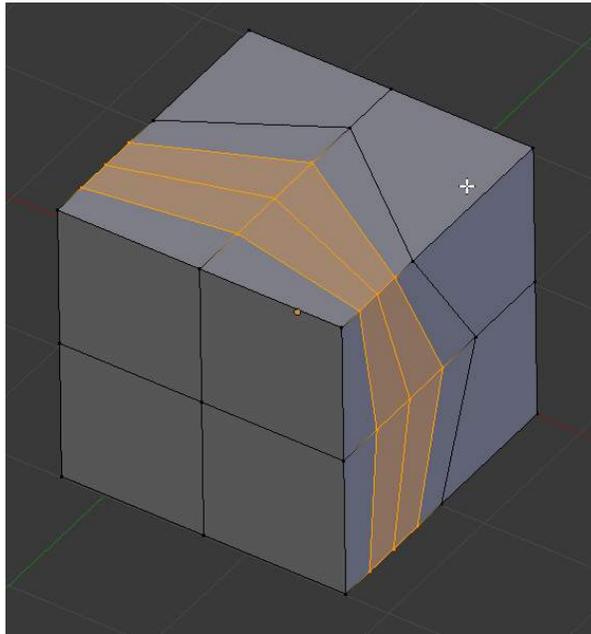


Fig. 2.387: Result of using multiple cuts

**Smoothing `Alt-Wheel`** Smoothing causes edge loops to be placed in an interpolated position, relative to the face it is added to, causing them to be shifted outwards or inwards by a given percentage, similar to the *Subdivide Smooth* command. When not using smoothing, new vertices for the new edge loop are placed exactly on the pre-existing edges. This keeps subdivided faces flat, but can distort geometry, particularly when using *Subdivision Surfaces*. Smoothing can help maintain the curvature of a surface once it is subdivided.

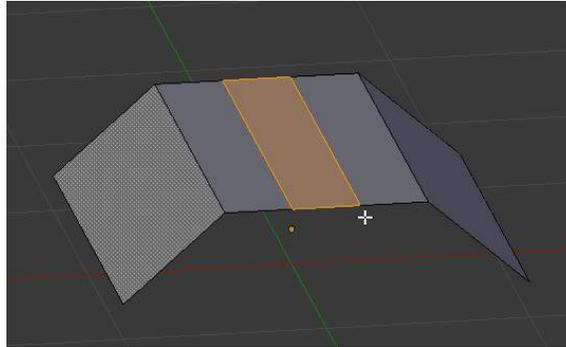


Fig. 2.388: Added edge loops without smoothing

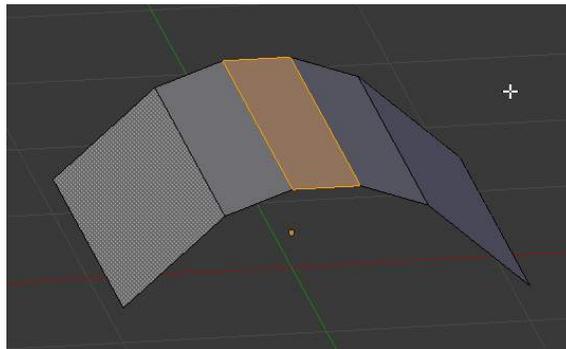


Fig. 2.389: Same edge loops, but with smoothing value

## Knife Tool Reference

Mode: *Edit* mode

Panel: *Mesh Tools* (*Editing* context)

Hotkey: `K` or `Shift-K`

The knife tool can be used to interactively cut up geometry by drawing lines or closed loops to create holes.

**Usage** When you press `K` (or `Shift-K`), the Knife tool becomes active.

**Drawing the cut line** When using *Knife*, the cursor changes to an icon of a scalpel and the header changes to display options for the tool. You can draw connected straight lines by clicking `LMB`.

## Options

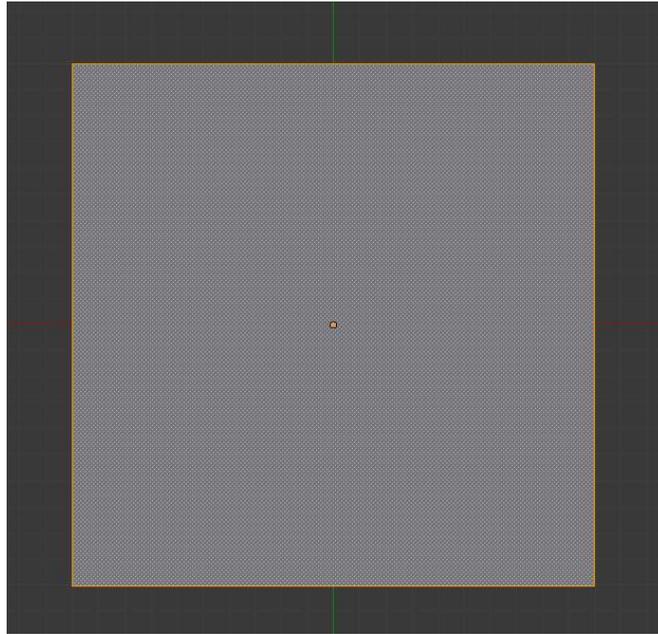


Fig. 2.390: Mesh before knife cut

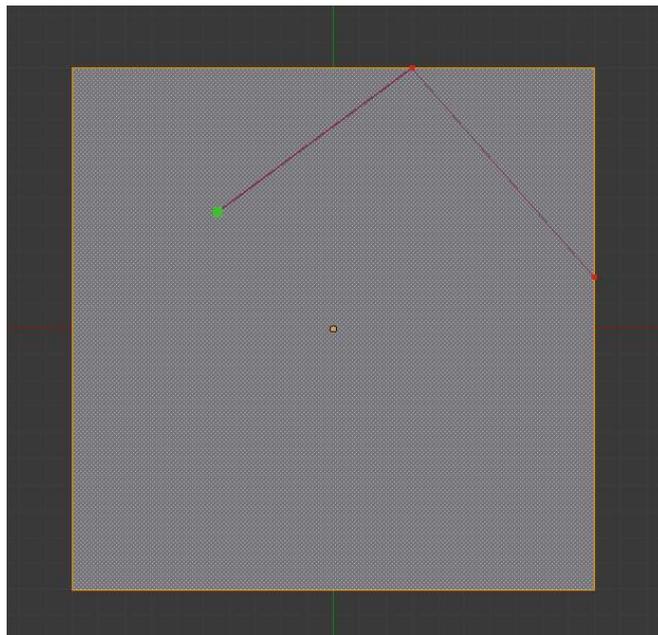


Fig. 2.391: Knife cut active

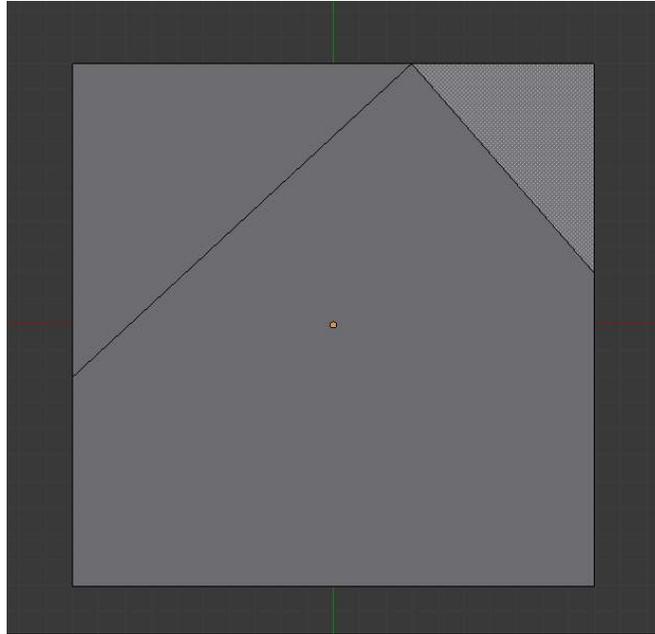


Fig. 2.392: After confirming knife cut

**Knife selection `Shift-K`** Activates the knife so only selected faces are cut.

**New cut `E`** Begins a new cut. This allows you to define multiple distinct cut lines. If multiple cuts have been defined, they are recognized as new snapping points.

**Midpoint snap `Ctrl`** Hold to snap the cursor to the midpoint of edges

**Ignore snap `Shift`** Hold to make the tool ignore snapping.

**Cut through: `Z`** Allow the cut tool to cut through to obscured faces, instead of only the visible ones.

**Angle constrain `C`** Constrains the cut to 45 degree increments.

**Close loop: Double click `LMB`** This is a quick way to close the loop you're currently cutting.

**Draw a continuous line: `LMB drag`.** So you can draw a freehand line over a surface, points will be created at edge intersections.

**Confirming and selection** Pressing `Esc` or `RMB` at any time cancels the tool, and pressing `LMB` or `Return` confirms the cut, with the following options:

`Return` will leave selected every edge except the new edges created from the cut.

**Limitations** Cuts that begin or end in the middle of a face, will be ignored. This is a limitation of the current geometry that can be modeled in Blender.

**Knife Project** Knife projection is a non-interactive tool where you can use objects to cookie-cut into the mesh rather than hand drawing the line.

This works by using the outlines of other selected objects in edit-mode to cut into the mesh, resulting geometry inside the cutters outline will be selected.

Outlines can be wire or boundary edges.

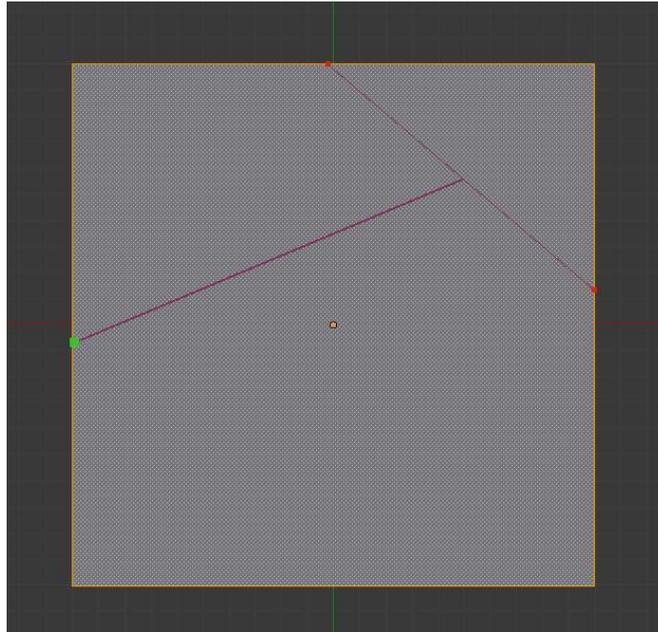


Fig. 2.393: Creating multiple cuts

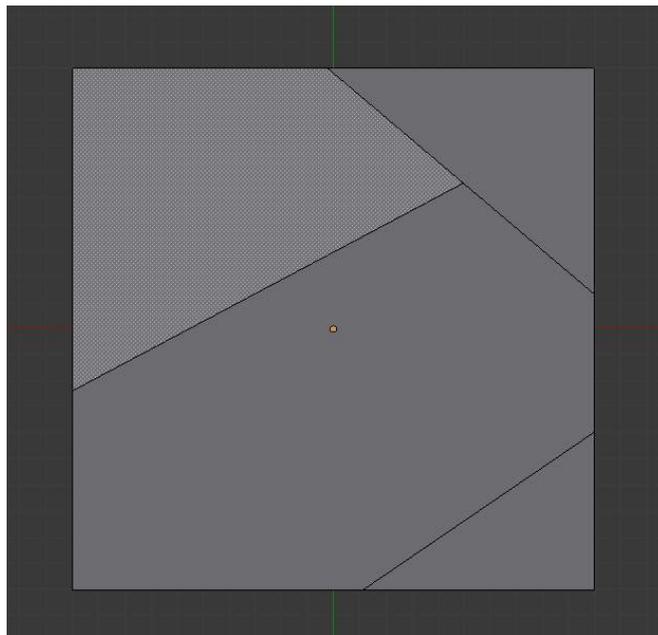


Fig. 2.394: Result of starting new cuts while in the tool

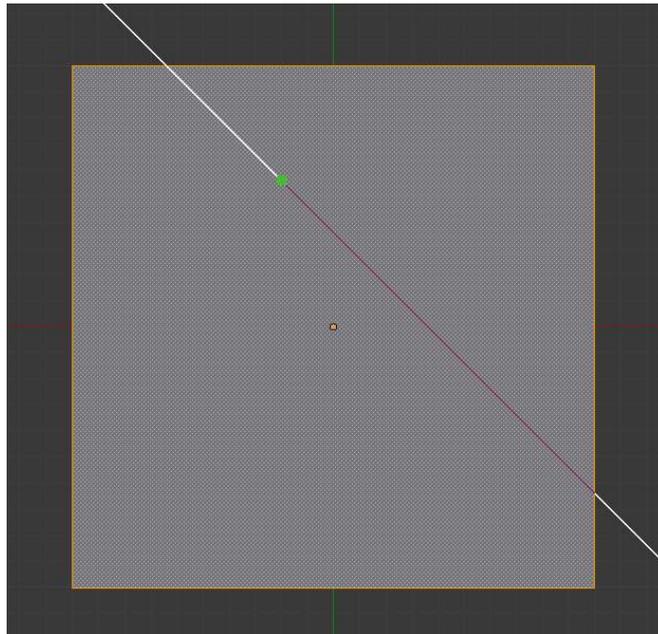


Fig. 2.395: Constraining cut angle

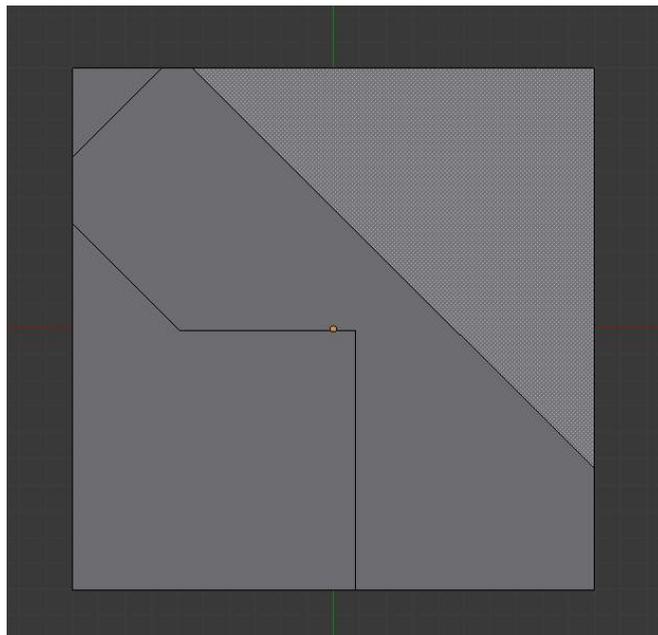


Fig. 2.396: Result of constraining cut angle

To use Knife Project, in 'object' mode select the "cutting object" first then shift select the "object to be cut". Now tab into edit mode and press "knife project".

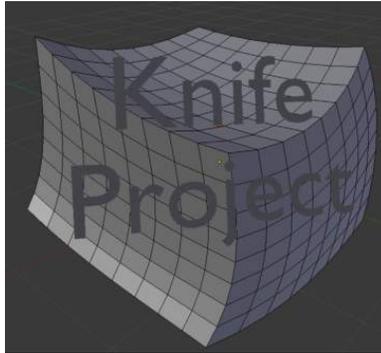


Fig. 2.397: Before projecting from a text object

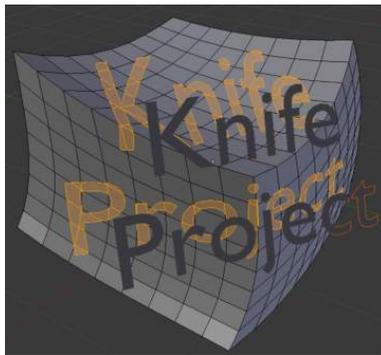


Fig. 2.398: Resulting knife projection

## Examples

**Known Issues** Cutting holes into single faces may fail, this is the same limitation as with the regular knife tool but more noticeable for text, this can be avoided by projecting onto more highly subdivided geometry.

## Mesh Bisect

### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Bisect*

The bisect tool is a quick way to cut a mesh in-two along a custom plane.

There are three important differences between this and the knife tool.

- The plane can be numerically adjusted in the operator panel for precise values.
- Cuts may remove geometry on one side.
- Cuts can optionally fill in the holes created, with materials and UV's & vertex-colors based on the surrounding geometry.

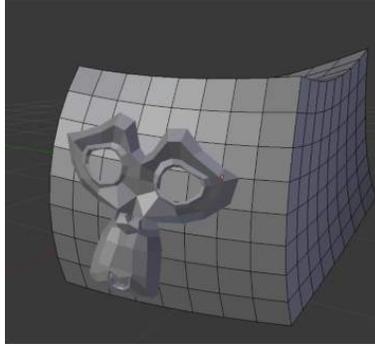


Fig. 2.399: Before projecting from a mesh object

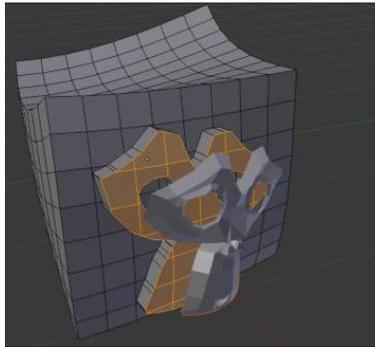


Fig. 2.400: Resulting knife projection (extruded after)

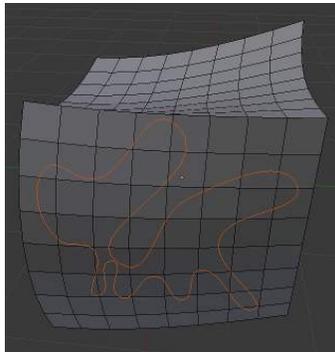


Fig. 2.401: Before projecting from a 3D curve object

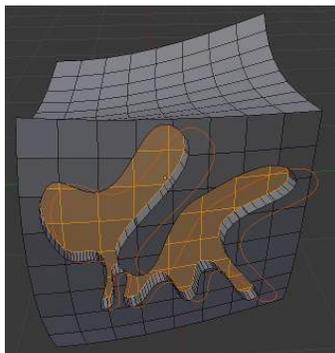


Fig. 2.402: Resulting knife projection (extruded after)

This means the bisect tool can cut off parts of a mesh without creating any holes.

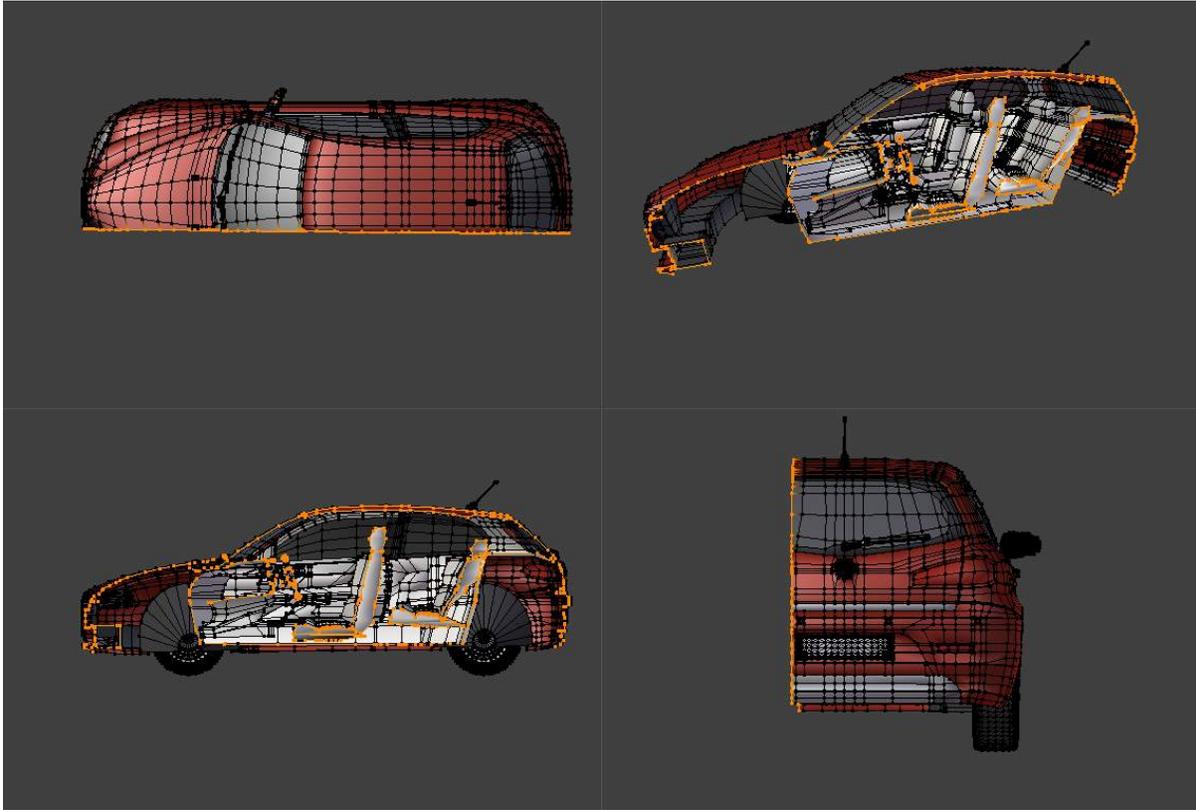


Fig. 2.403: Example of a common use of bisect

## Vertex Connect

### Reference

Mode: *Edit* mode

Menu: Mesh -> Vertices -> Connect

Hotkey:  $\bar{J}$

This tool joins selected vertices by edges, The main difference between this and creating edges is that faces are split by the newly joined vertices.

When many vertices are selected, faces will be split by their selected vertices.



When there are only 2 vertices selected, a cut will be made across unselected faces, a little like the knife tool; however this is limited to straight cuts across connected faces.



## Bevel

### Reference

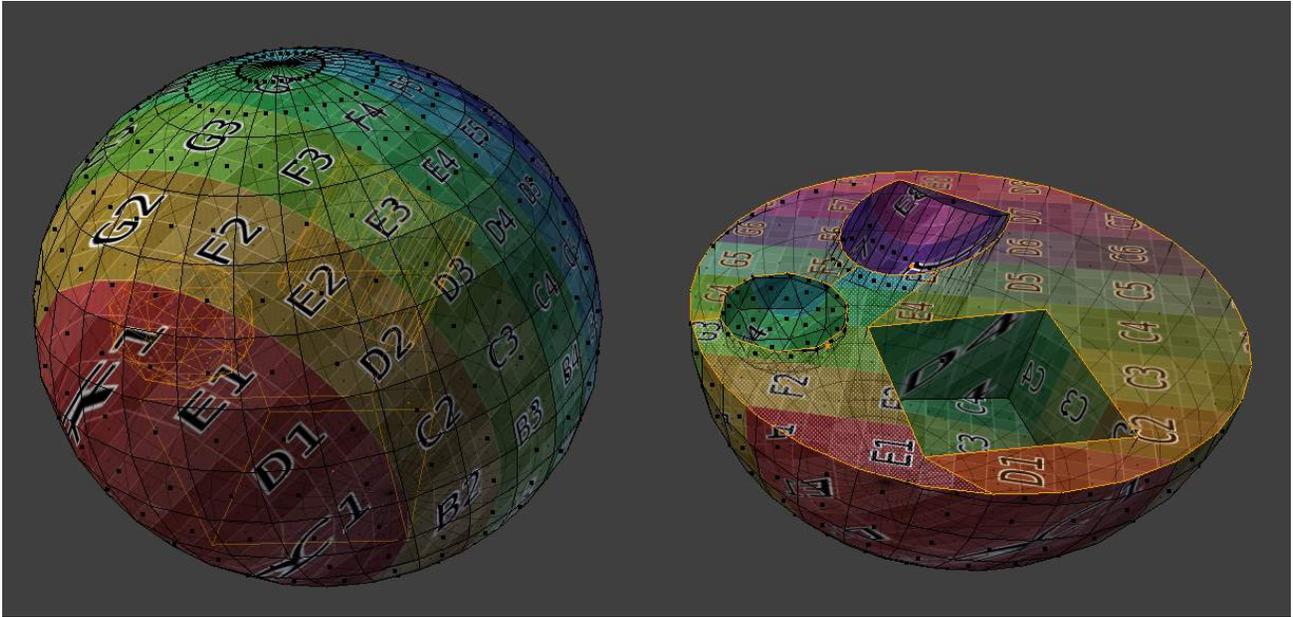


Fig. 2.404: Example of bisect with fill option

Mode: *Edit mode*

Menu: *Mesh* → *Edges* → *Bevel* or *[Ctrl][E]* → *Bevel*

Hotkey: *Ctrl-B* or *[W]* → *Bevel*

Menu (vertex-only): *Mesh* → *Vertices* → *Bevel* or *[Ctrl][V]* → *Bevel*

Hotkey (vertex-only): *Shift-Ctrl-B*



Fig. 2.413: With bevel and without bevel.

The bevel tool allows you to create chamfered or rounded corners to geometry. A bevel is an effect that smooths out edges and corners. True world edges are very seldom exactly sharp. Not even a knife blade edge can be considered perfectly sharp. Most edges are intentionally beveled for mechanical and practical reasons.

Bevels are also useful for giving realism to non-organic models. In the real world, the blunt edges on objects catch the light and change the shading around the edges. This gives a solid, realistic look, as opposed to un-beveled objects which can look too perfect.

**Bevel Modifier** The [Bevel Modifier](#) is a non destructive alternative to the bevel tool. It gives you the same options, with additional goodies, like the bevel width controlled by the vertices weight, and all the modifiers general enhancements (non-destructive operations, ...).

**Usage** The *Bevel* tool works only on selected edges. It will recognize any edges included in a vertex or face selection as well, and perform the bevel the same as if those edges were explicitly selected. In “vertex only” mode, the *Bevel* tool works on selected vertices instead of edges. The *Bevel* tool smooths the edges and/or “corners” (vertices) by replacing them with faces making smooth profiles with a specified number of *segments* (see the options below for details about the bevel algorithm).

Use `Ctrl-B` or a method listed above to run the tool. Move the mouse to interactively specify the bevel offset, and scroll the `Wheel` to increase or decrease the number of segments. (see below)

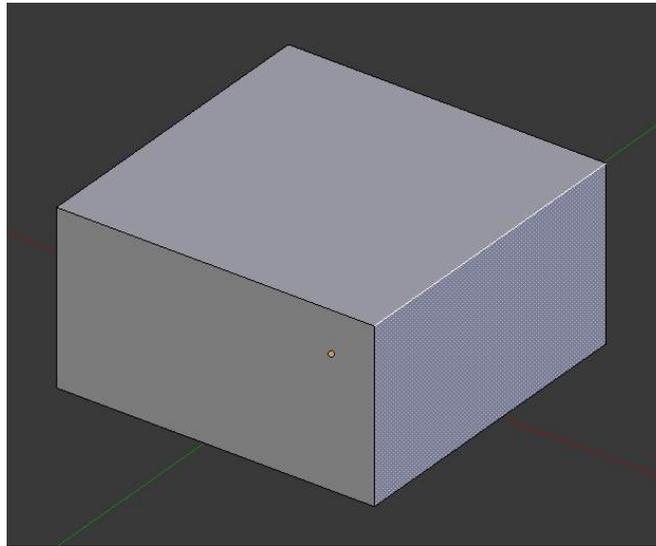


Fig. 2.414: Selected edge before beveling

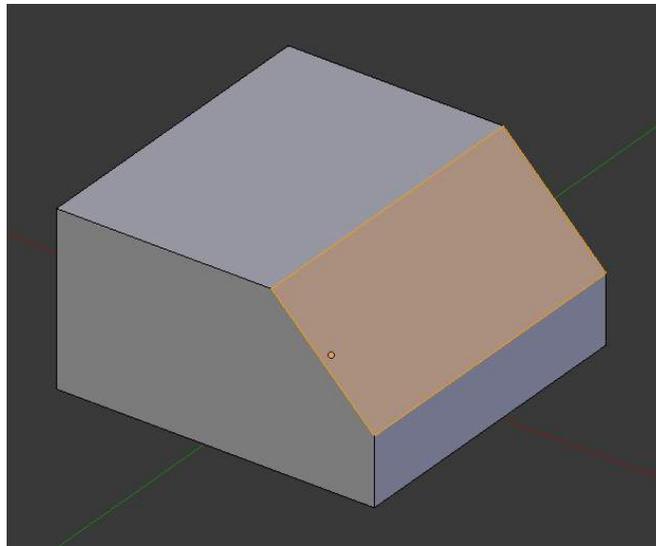


Fig. 2.415: Result of bevel (one segment)

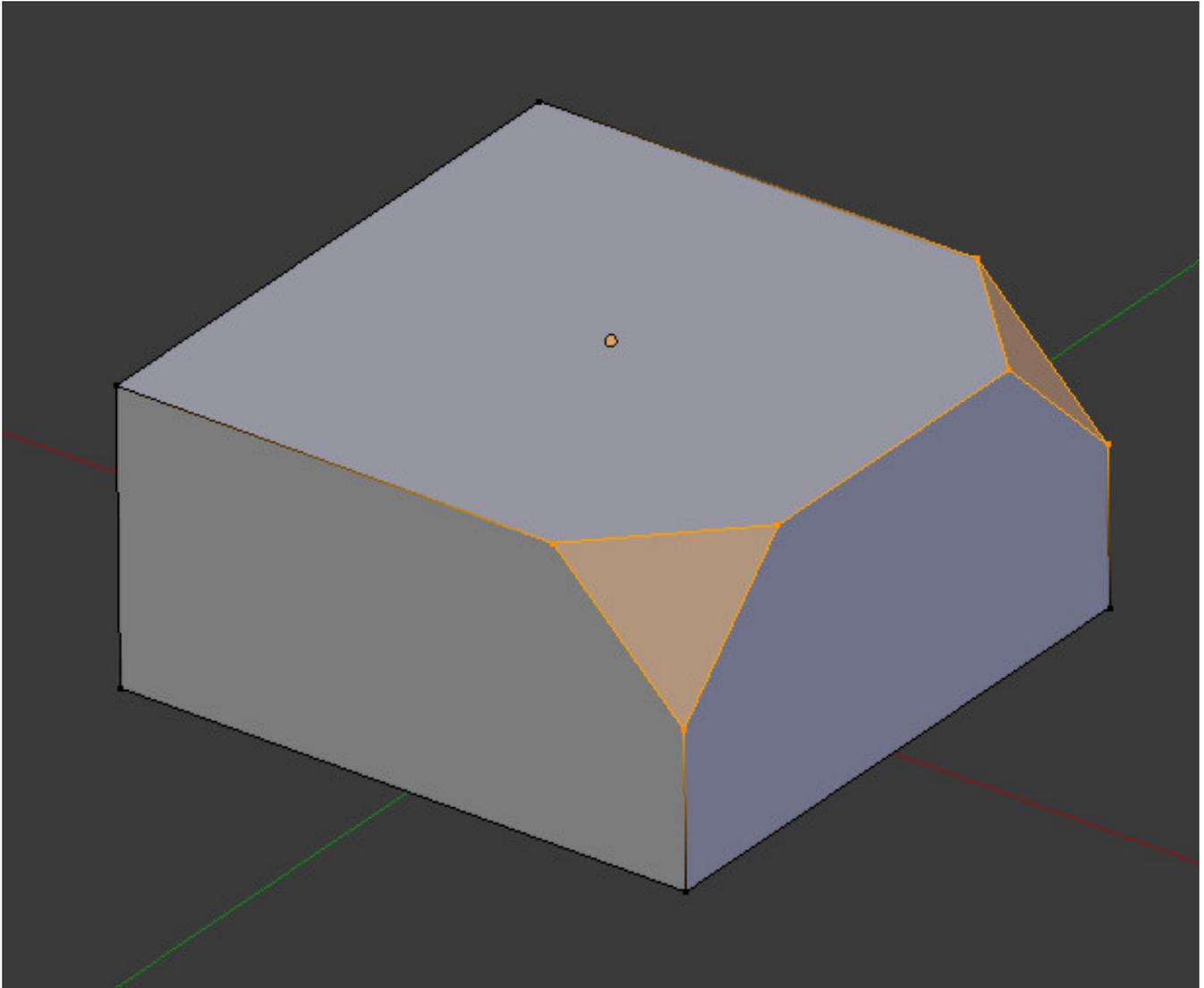
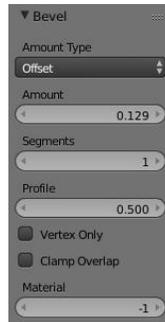


Fig. 2.416: Result of bevel (vertex only)

**Note:** Normal (edge) beveling only works on edges that have exactly two faces attached to them. Vertex bevel has no such restriction.



### Options

**Amount** You can change the bevel amount by moving the mouse towards and away from the object, a bit like with transform tools. The exact meaning of the value depends on the *Amount Type* option (see below). As usual, the scaling can be controlled to a finer degree by holding `Shift` to scale in 0.001 steps. LMB finalizes the operation, RMB or `Esc` aborts the action.

**Amount Type** Selects how the *Amount* value controls the size of the bevel. According to the selection, the amount is: - **Offset** - the distance of a new edge from the original - **Width** - the width of the bevel face - **Depth** - the perpendicular distance from the original edge to the bevel face - **Percent** - the percentage of the length of adjacent edges that the new edges slide

**Segments** The number of segments in the bevel can be defined by scrolling the mouse `Wheel` to increase or decrease this value. The greater the number of segments, the smoother the bevel.

Alternatively, you can manually enter a segment number value while using the tool, or in the Mesh Tool options panel after using the tool.

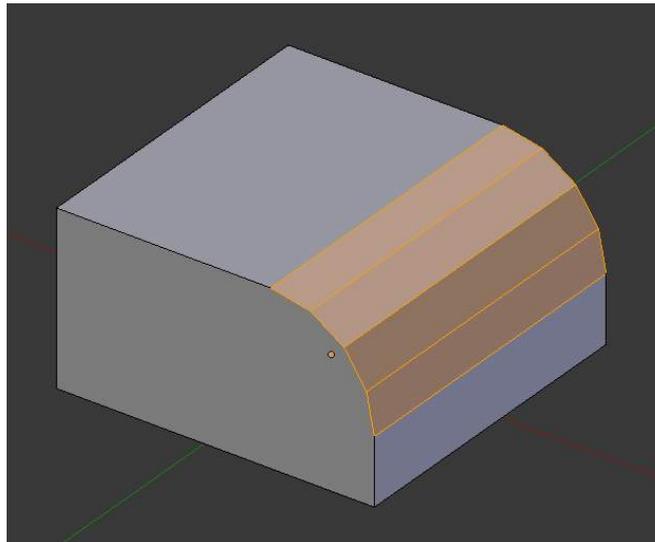


Fig. 2.417: Bevel with 4 segments

**Profile** This is a number between 0 and 1 that controls the shape of the profile (side view of a beveled edge). The default value, 0.5, gives a circular arc (if the faces meet at right angles). Values less than that give a flatter profile, with 0.25 being exactly flat, and values less than that giving a concave bevel. Values more than 0.5 give a more “bulged-out” profile.

**Vertex Only** When selected, the tool is in “vertex only” mode, and only vertices will be beveled.

**Clamp Overlap** When selected, the bevel amount is not allowed to go larger than an amount that causes overlapping collisions with other geometry.

**Material** The *Material* number specifies which material should be assigned to the new faces created by the *Bevel* tool. With the default, -1, the material is inherited from the closest existing face (“closest” can be a bit ambiguous). Otherwise, the number is the slot index of the material to use for all newly created faces.

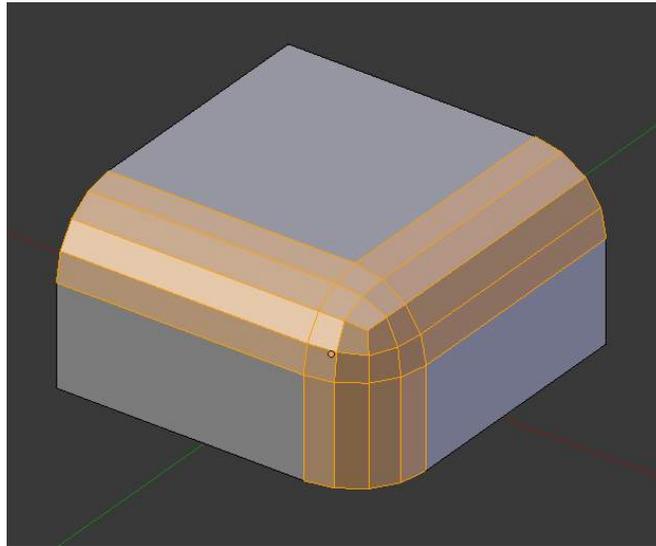


Fig. 2.418: Result of beveling multiple edges

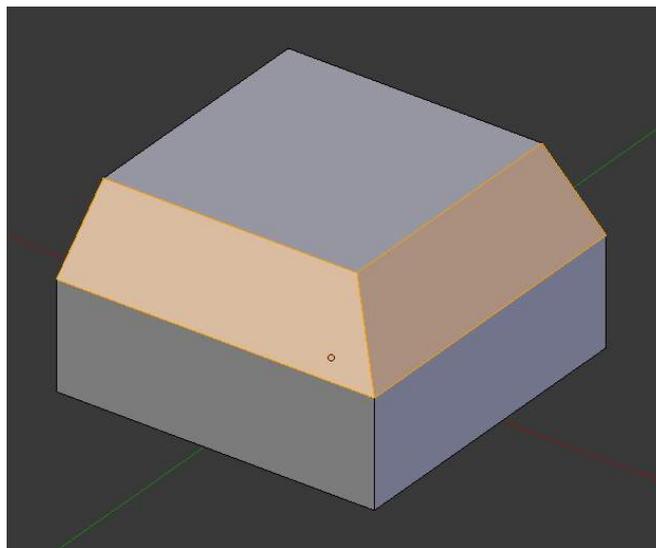


Fig. 2.419: Another example of beveling multiple edges

## Examples

### Miscellaneous Editing Tools

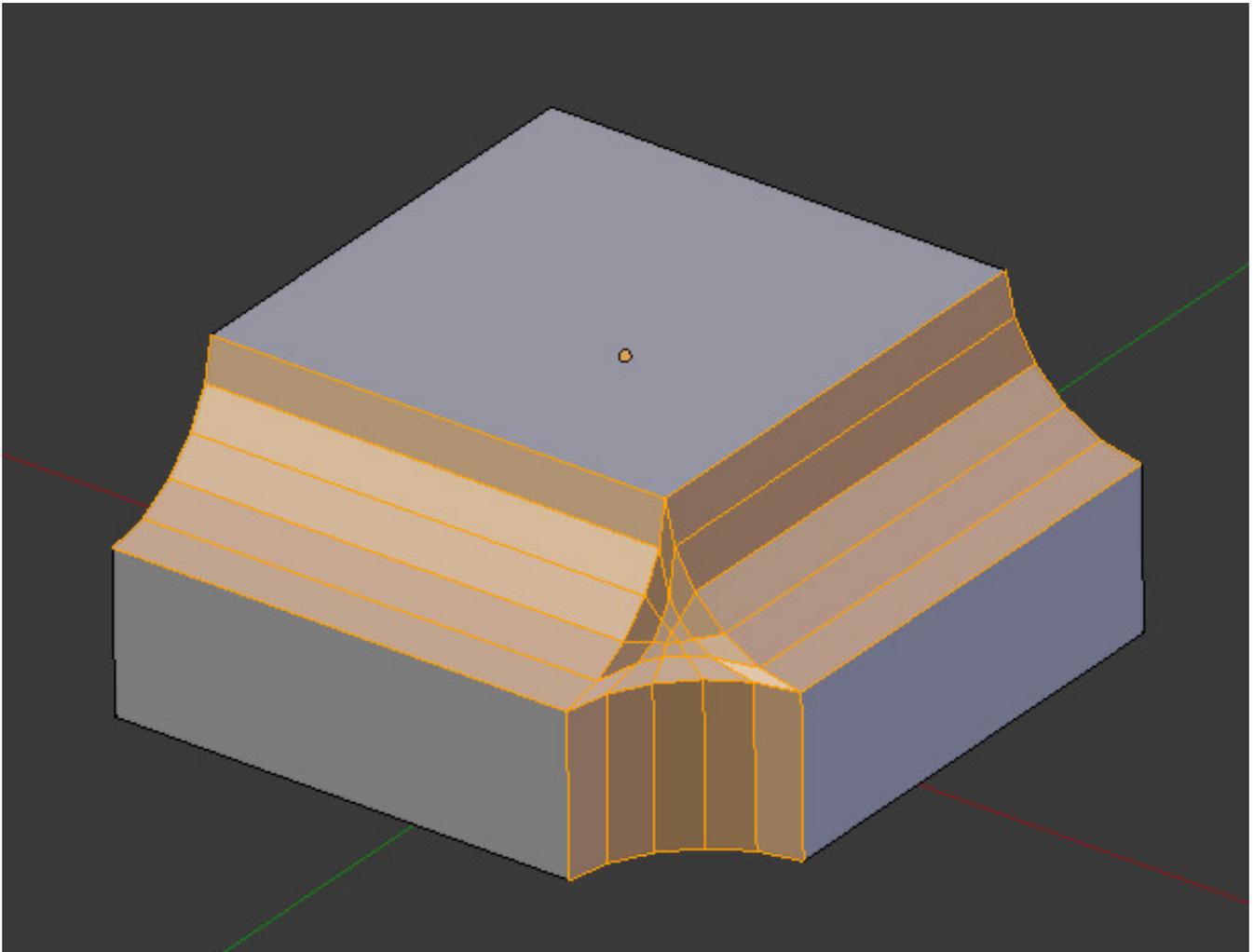


Fig. 2.420: An example using Profile=0.150

**Sort Elements** This tool (available from the *Specials*, *Vertices*, *Edges* and *Faces* menus) allows you to reorder the matching selected mesh elements, following various methods. Note that when called from the *Specials* menu, the affected element types are the same as the active select modes.

**View Z Axis** Sort along the active view's Z axis, from farthest to nearest by default (use *Reverse* if you want it the other way).

**View X Axis** Sort along the active view's X axis, from left to right by default (again, there's the *Reverse* option).

**Cursor Distance** Sort from nearest to farthest away from the 3D cursor position (*Reverse* also available).

**Material Faces only!** Sort faces from those having the lowest material's index to those having the highest. Order of faces inside each of those "material groups" remains unchanged. Note that the *Reverse* option only reverses the order of the materials, *not* the order of the faces inside them.

**Selected** Move all selected elements to the beginning (or end, if *Reverse* enabled), without affecting their relative orders.

**Warning: this option will also affect unselected elements' indices!**

**Randomize** Randomizes indices of selected elements (*without* affecting those of unselected ones). The seed option allows you to get another randomization - the same seed over the same mesh/set of selected elements will always give the same result!

**Reverse** Simply reverse the order of the selected elements.

## Retopologizing

Retopology is a common part of modeling workflows. Often times, a model is created with emphasis on form and detail, however, its topology, or edge flow is not ideal, or the mesh is very dense, and not efficient. Modelers can create a new lower resolution mesh that matches the form of the original mesh.

**Mesh Snapping** By enabling snapping, and setting the snap element to Face, mesh vertices will be projected onto the closest surface in the viewport, in the view's Z-axis.

This allows you to model freely, without concern for form, and to focus on topology

See [Snapping](#)

**Shrinkwrap Modifier** The [Shrinkwrap Modifier](#) is useful in conjunction with face snapping. If edits to the new mesh have been made with snapping disabled, the shrinkwrap modifier will allow you to stick the new mesh to the old mesh, as if you were shrinkwrapping it.

## Sculpt Mode

**Overview** *Sculpt Mode* is similar to *Edit Mode* in that it is used to alter the shape of a model, but *Sculpt Mode* uses a very different workflow: instead of dealing with individual elements (vertices, edges, and faces), an area of the model is altered using a brush. In other words, instead of selecting a group of vertices, *Sculpt Mode* automatically selects vertices based on where the brush is, and modifies them accordingly.

**Sculpt Mode** *Sculpt mode* is selected from the mode menu of the *3D View* header.

Once *sculpt mode* is activated the *Toolbar* of the *3D View* will change to *sculpt mode* specific panels. The panels in the toolbar will be *Brush*, *Texture*, *Tool*, *Symmetry*, *Stroke*, *Curve*, *Appearance*, and *Options*. Also a red circle will appear that follows the location of the cursor in the 3d view.

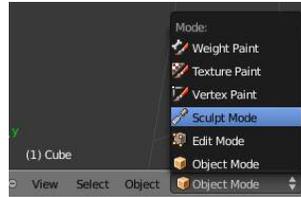


Fig. 2.421: Sculpt Mode Dropdown.



Fig. 2.422: The cursor in Sculpt Mode.

**Sculpt Brushes** Brushes are brush presets. They are a combination of a ‘tool’, along with stroke, texture, and options.

*Sculpt* Mode has sixteen brushes, each of which operates on the model in a unique way. Many can be toggled to have an additive or subtractive effect. They can be selected in the *Tool* menu.

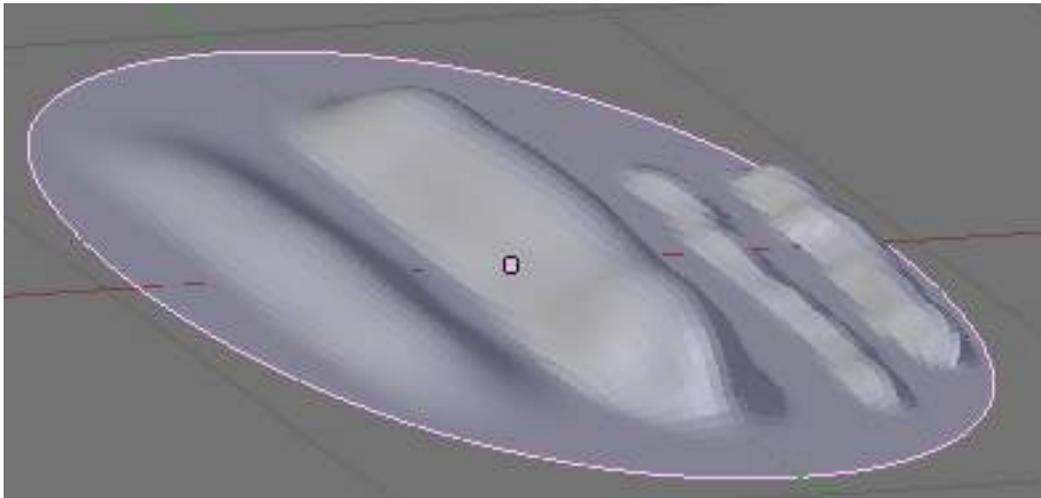


Fig. 2.423: Drawing in various sizes and strengths.

**Blob** Pushes mesh outward or inward into a spherical shape with settings to control the amount of pinching at the edge of the sphere.

**Clay (C)** Similar to the *Draw* brush, but includes settings to adjust the plane on which the brush acts.

**Clay Strips** Similar to the *Clay* brush, but it uses a cube test to define the brush area of influence rather than a sphere.

**Crease** Creates sharp indents or ridges by pushing or pulling the mesh, while pinching the vertices together.

**Draw (D)** Moves vertices inward or outward, based the average normal of the vertices contained within the drawn brush stroke.

**Fill** The *Fill* brush works like the *Flatten* brush, but only brings vertices below the brush plane upwards. The inverse of the *Scrape* brush is to *Deepen* by pushing vertices above the plane downward.

**Flatten (T)** The *Flatten* brush finds an ‘area plane’ located by default at the average height above/below the vertices within the brush area. The vertices are then pulled towards this plane. The inverse of the *Flatten* brush is the *Contrast* brush which pushes vertices up or down away from the brush plane.

**Grab (G)** *Grab* is used to drag a group of points around. Unlike the other brushes, *Grab* does not modify different points as the brush is dragged across the model. Instead, *Grab* selects a group of vertices on mousedown, and pulls them to follow the mouse. The effect is similar to moving a group of vertices in *Edit mode* with proportional-editing enabled, except that *Grab* can make use of other Sculpt Mode options (like textures and symmetry).

**Inflate (I)** Similar to *Draw*, except that vertices in *Inflate* mode are displaced in the direction of their own normals.

**Layer (L)** This brush is similar to *Draw*, except that the height of the displacement layer is capped. This creates the appearance of a solid layer being drawn. This brush does not draw on top of itself; a brush stroke intersects itself. Releasing the mouse button and starting a new stroke will reset the depth and paint on top of the previous stroke.

**Nudge** Moves vertices in the direction of the brush stroke.

**Pinch (P)** *Pinch* pulls vertices towards the center of the brush. The inverse setting is *Magnify*, in which vertices are pushed away from the center of the brush.

**Rotate** Rotates vertices within the brush in the direction the cursor is moved.

**Scrape** The *Scrape* brush works like the Flatten brush, but only brings vertices above the plane downwards. The inverse of the Scrape brush is to *Peak* by pushing vertices above the plane up away from the plane.

**Smooth (S)** As the name suggests, eliminates irregularities in the area of the mesh within the brush's influence by smoothing the positions of the vertices.

**Snake Hook** Pulls vertices along with the movement of the brush to create long, snake-like forms.

**Thumb** Similar to the *Nudge* brush, this one flattens the mesh in the brush area, while moving it in the direction of the brush stroke.

### Sculpting with the Multires Modifier ...

**Sculpt Properties Panel** This panel appears in the tool palette on the left side of the 3D viewport.

### Brush Menu

**Radius** This option controls the radius of the brush, measured in pixels. **F** in the 3D view allows you to change the brush size interactively by dragging the mouse and then left clicking (the texture of the brush should be visible inside the circle). Typing a number then enter while in **F** sizing allows you to enter the size numerically. Brush size can be affected by enabling the pressure sensitivity icon, if a supported tablet is being used.

**Strength** *Strength* controls how much each application of the brush affects the model. For example, higher values cause the *Draw* brush to add depth to the model more quickly, and cause the *Smooth* brush to smooth the model more quickly. This setting is not available for *Grab*, *Snake Hook*, or *Rotate*.

If the range of strengths doesn't seem to fit the model (for example, if even the lowest strength setting still makes too large of a change on the model) then you can scale the model (in *Edit Mode*, not *Object Mode*). Larger sizes will make the brush's effect smaller, and vice versa. You can change the brush strength interactively by pressing **Shift-F** in the 3D view and then moving the brush and then left clicking. You can enter the size numerically also while in **Shift-F** sizing. Brush strength can be affected by enabling the pressure sensitivity icon, if a supported tablet is being used.

**Autosmooth** Sets the amount of smoothing to be applied to each stroke.

**Sculpt Plane** Use this menu to set the plane in which the sculpting takes place.

**Plane Offset** Adjusts the plane on which the brush acts toward or away from the viewer.

**Trim** Enables trimming of the sculpt plane, determined by the *Distance* setting.

**Front Faces Only** When enabled, the brush only affects vertices that are facing the viewer.

**Accumulate** Causes stroke dabs to accumulate on top of each other.

## Stroke Menu

**Stroke Method** Defines the way brush strokes are applied to the mesh:

**Dots** Standard brush stroke.

**Drag Dot** Creates a single displacement in the brush shape. Click then drag on mesh to desired location, then release.

**Space** Creates brush stroke as a series of dots, whose spacing is determined by the *Spacing* setting. *Spacing* represents the percentage of the brush diameter.

**Anchored** Creates a single displacement at the brush location. Clicking and dragging will resize the brush diameter. When *Edge to Edge* the brush location and orientation is determined by a two point circle, where the first click is one point, and dragging places the second point, opposite from the first.

**Airbrush** Flow of the brush continues as long as the mouse click is held, determined by the *Rate* setting. If disabled, the brush only modifies the model when the brush changes its location. This option is not available for the *Grab* brush.

The following parameters are available for the *Dots*, *Space*, and *Airbrush* strokes:

**Smooth stroke** Brush lags behind mouse and follows a smoother path. When enabled, the following become active:

**Radius** Sets the minimum distance from the last point before stroke continues.

**Factor** Sets the amount of smoothing

**Jitter** Jitters the position of the brush while painting.

**Curve Menu** The *Curve* section allows you to use a curve control to the right to modify the intensity of the brush from its centre (left part of the curve) towards its borders (right part of the curve).

**See also:**

- Read more about using the *Curve Widget*.

**Texture Menu** A texture can be used to determine the strength of brush effects as well. Select an existing texture from the texture box, or create a new one by selecting the *New* button

**Brush Mapping** Sets the way the texture is mapped to the brush stroke:

**Fixed** If *Fixed* is enabled, the texture follows the mouse, so it appears that the texture is being dragged across the model.

**Tiled** The *Tile* option tiles the texture across the screen, so moving the brush appears to move separately from the texture. The *Tile* option is most useful with tileable images, rather than procedural textures.

**3D** The *3D* option allows the brush to take full advantage of procedural textures. This mode uses vertex coordinates rather than the brush location to determine what area of the texture to use.

**Angle** This is the rotation angle of the texture brush. It can be changed interactively via `Ctrl-F` in the 3D view. While in the interactive rotation you can enter a value numerically as well. Can be set to:

**User** Directly input the angle value.

**Rake** Angle follows the direction of the brush stroke. Not available with *3D* textures.

**Random** Angle is randomized.

**Offset** Fine tunes the texture map placement in the x, y, and z axes.

**Size** This setting allows you to modify the scaling factor of the texture. Not available for *Drag* textures.

**Sample Bias** Value added to texture samples.

**Overlay** When enabled, the texture is shown in the viewport, as determined by the; *Alpha* value.

**Symmetry Menu** Mirror the brush strokes across the selected local axes. Note that if you want to alter the directions the axes point in, you must rotate the model in *Edit Mode*, not *Object Mode*.

**Feather** Reduces the strength of the stroke where it overlaps the planes of symmetry.

**Radial** These settings allow for radial symmetry in the desired axes. The number determines how many times the stroke will be repeated within 360 degrees around the central axes.

### Options Menu

**Threaded Sculpt** Takes advantage of multiple CPU processors to improve sculpting performance.

**Fast Navigation** For *Multires* models, show low resolution while navigation the viewport.

**Show Brush** Shows the brush shape in the viewport.

### Unified Settings:

**Size** Forces the brush size to be shared across brushes.

**Strength** Forces the brush strength to be shared across brushes.

**Lock** These three buttons allow you to block any modification/deformation of your model along selected local axes, while you are sculpting it.

**Appearance Menu** You can set the color of the brush depending on if it is in additive or subtractive mode.

You can also set the brush icon from an image file.

**Tool Menu** Here you can select the type of brush preset to use. *Reset Brush* will return the settings of a brush to its defaults. You can also set Blender to use the current brush for *Vertex Paint* mode, *Weight Paint* mode, and *Texture Paint* mode using the toggle buttons.

**Hiding and Revealing Mesh** It is sometimes useful to isolate parts of a mesh to sculpt on. To hide a part of a mesh, press **H** then click & drag around the part you want to hide. To reveal a hidden part of a mesh, press **Shift-H** then click & drag around the part you want to reveal. To reveal all hidden parts, just press **Alt-H**.

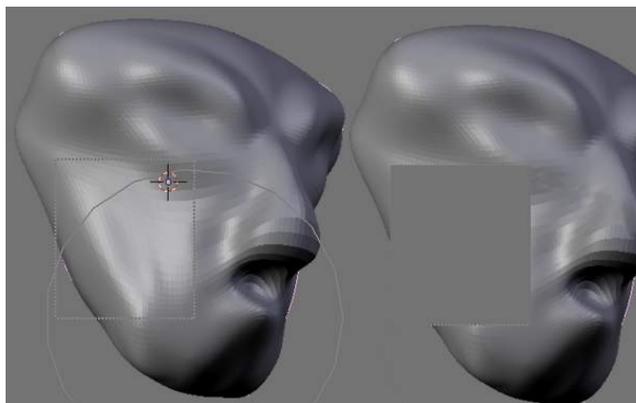


Fig. 2.424: Before and after Hiding.

**Keyboard Shortcuts** These shortcuts may be customized under **File > User preferences > Input > 3D View > Sculpt Mode**.

Table 2.4: Action -&gt; Shortcut table:

Hide mesh inside selection	H then click & drag
Reveal mesh inside selection	Shift-H then click & drag
Show entire mesh	Alt-H
Interactively set brush size	F
Increase/decrease brush size	[ and ]
Interactively set brush strength	Shift-F
Interactively rotate brush texture	Ctrl-F
Brush direction toggle ( <i>Add / Sub</i> )	Ctrl pressed while sculpting
Set stroke method (airbrush, anchored, ..)	A
Smooth stroke toggle	Shift-S
<i>Draw</i> brush	D
<i>Smooth</i> brush	S
<i>Pinch/Magnify</i> brush	P
<i>Inflate/Deflate</i> brush	I
<i>Grab</i> brush	G
<i>Layer</i> brush	L
<i>Flatten/Contrast</i> brush	Shift-T
<i>Clay</i> brush	C
<i>Crease</i> brush	Shift-C
<i>Snake Hook</i> brush	K
<i>Mask</i> brush	M
Mask clear	Alt-M
Mask invert	Ctrl-I
Set brush by number	0 - 9 and Shift-0 to Shift-9
Sculpt options panel toggle	T
Step up one multires level	PageUp
Step down one multires level	PageDown
Set multires level	Ctrl-0 to Ctrl-5
Dynamic topology toggle	Ctrl-D
Set texture angle type	R
Translate/scale/rotate stencil texture	RMB, Shift-RMB, Ctrl-RMB
Translate/scale/rotate stencil mask	Alt-RMB, Alt-Shift-RMB, Alt-Ctrl-RMB

## Multires

FIXME(Template Unsupported: oldfeature; {{oldfeature|2.5|This feature is no longer available in Blender 2.5, see the [[Doc:2.6/Manual/Modifiers/Generate/Multiresolution|Multires modifier]].}} )

## Vertex Groups

### Vertex Groups

Vertex Groups are mainly used to tag the vertices belonging to parts of a Mesh Object or *Lattice*. Think of the legs of a chair or the hinges of a door, or hands, arms, limbs, head, feet, etc. of a character. In addition you can assign different *weight values* (in the range [ 0.0, 1.0 ] ) to the vertices within a Vertex Group. Hence Vertex Groups are sometimes also named *Weight Groups*.

Vertex Groups are most commonly used for Armatures (See also *Skinning Mesh Objects*). But they are also used in many other areas of Blender, like for example:

- Shape keys



Fig. 2.425: The Vertex Group Panel

- Modifiers
- Particle Generators
- Physics Simulations

Many more usage scenarios are possible. Actually you can use Vertex Groups for whatever makes sense to you. In some contexts Vertex Groups can also be automatically generated (i.e. for rigged objects). However in this section we will focus on manually created (user-defined) Vertex Groups.

---

**Note:** Vertex groups only apply to Mesh and Lattice Objects

Any other Object type has no vertices, hence it can not have Vertex Groups.

---

### Typical usage scenarios for Vertex groups

**Skinning an armature** If you want to animate your mesh and make it move, you will define an armature which consists of a bunch of bones. Vertex Groups are used to associate parts of the Mesh to Bones of the Armature, where you can specify an influence *weight* in the range [0.0 ... 1.0] for each vertex in the Vertex Group.

**Working with Modifiers** Many modifiers contain the ability to control the modifier influence on each vertex separately. This is also done via Vertex Groups and the weight values associated to the vertices.

**Quickly select/edit/hide parts of a mesh** By defining mesh regions with Vertex Groups you can easily select entire parts of your mesh with 3 clicks and work on them in isolation without having to create separate objects. With the hide function you can even remove a vertex group from the view (for later unhide).

**Cull out and duplicate parts of a mesh** Consider modeling a Lego block. The most simple brick consists of a base and a stud (the bump to connect the bricks together). To create a four-stud block, you would want to be able to easily select the stud vertices, and, still in *Edit mode*, duplicate them and position them where you want them.

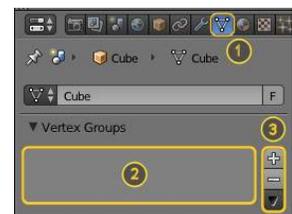


Fig. 2.426: Empty Vertex Group Panel

**Creating Vertex Groups** Vertex Groups are maintained within the *Object Data Properties* window (1), and there in the *Vertex Groups* panel. As long as no Vertex groups are defined (the default for new Mesh Objects), the Panel is empty (2).

You create a vertex group by LMB + on the right Panel border (3). Initially the group is named *Group* (or *Group.nnn* when the name already exists) and gets displayed in the Panel (2) (see next image).

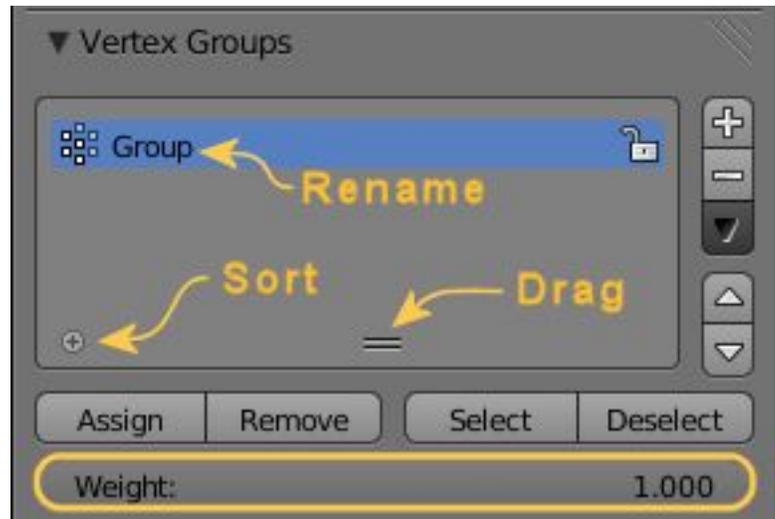


Fig. 2.427: One Vertex Group

**Vertex Groups Panel Controls** Once a new Vertex Group has been added, the new Group appears in the vertex Groups panel. There you find 3 clickable elements:

**Group Name** The Groupname can be changed by double clicking LMB on the name itself. Then you can edit the name as you like.

**Plus Icon** When the little icon in the left lower corner can be clicked, a new row opens up where you can enter a search term. This becomes handy when the number of vertex groups gets big.

**Drag Handle** If you have a large number of vertex groups and you want to see more then a few Groups, you can LMB on the small drag handle to tear the vertex groups list larger or smaller.

**Active Group** When a Vertex Group is created, then it is also automatically marked as the *Active Group*. This is indicated by setting the background of the panel entry to a light blue color. If you have two or more groups in the list, then you can change the active group by LMB on the corresponding entry in the Vertex Group panel.



Fig. 2.428: Delete a Vertex Group

**Deleting vertex Groups** You delete a Vertex Group by first making it the active group (select it in the panel) and then LMB the - button at the right Panel border.

Deleting a Vertex Group only deletes the vertex assignments to the Group. The vertices themselves are not deleted.



Fig. 2.429: Lock a Vertex Group

**Locking Vertex Groups** Right after creation of a Vertex Group, an open lock icon shows up on the right side of the Vertex Group List entry. This icon indicates that the Vertex Group can be edited. You can add vertex assignments to the group or remove assignments from the group. And you can change it with the weight paint brushes, etc.

When you click on the icon, it changes to a closed lock icon and all vertex group modifications get disabled. You can only rename or delete the group, and unlock it again. No other operations are allowed on locked Vertex Groups, thus all corresponding function buttons become disabled for locked Vertex Groups.

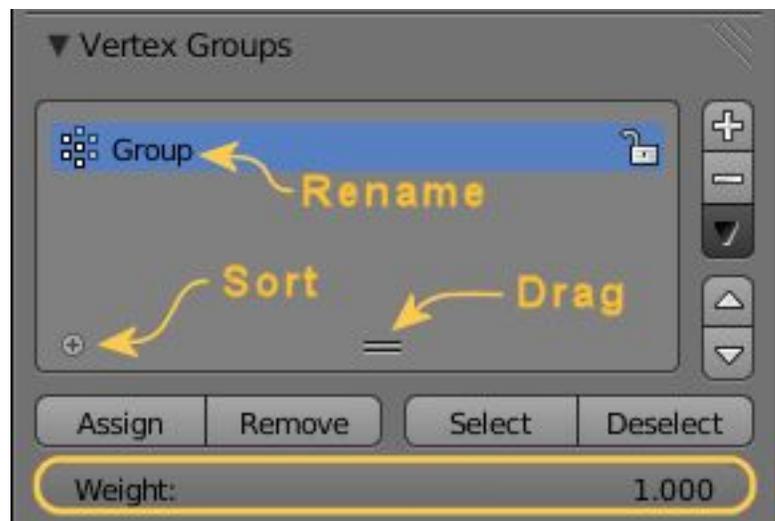


Fig. 2.430: Vertex Group Panel in Edit Mode

**Working with Content of Vertex Groups** When you switch either to *Edit-Mode* or to *Weight-Paint* Vertex Selection Mode, then the Vertex Group panel expands and displays 2 more rows:

The first row contains 4 buttons for maintaining the Assign- and Select- status of vertices of the active Vertex Group:

**Assign** To assign the Selected vertices to the Group with the weight as defined in the “Weight:” input field (see below)

**Remove** To Remove the selected vertices from the Group (and thus also delete their weight values)

**Select** To Select all vertices contained in the Group

**Deselect** To deselect all verts contained in the group

Below this row of buttons you see a numeric “Weight:” input field where you specify the weight value that gets assigned to the selected verts when you press the Assign Button.

**Assigning verts to a Group** You add vertices to a group as follows:

- Select the group from the group list, thus make it the Active Group (1).
- From the 3D Viewport select *Shift-RMB* all vertices that you want to add to the group.

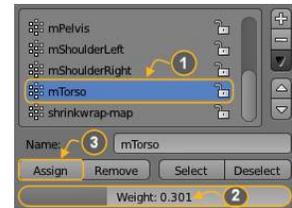


Fig. 2.431: Assign weights to active group

- Set the weight value that shall be assigned to all selected verts (2).
- LMB the *Assign* button to assign the selected verts to the active group using the given weight (3).

Note that weight Assignment is not available for locked Vertex Groups. The Assign button is grayed out in that case.

**Note:** Assign is additive

The *Assign* button only adds the currently selected vertices to the active group. Vertices already assigned to the group are not removed from the group.

Also keep in mind that a vertex can be assigned to multiple groups.

**Checking Assignments** To be sure the selected verts are in the desired Vertex Group, you can try press the deselect button. If the vertices remain selected then they're not yet in the current Vertex Group.

At this point you may assign then, but take care since all selected vertices will have their weight set to the value in the *Weight*: field.

**Removing assignments from a Group** You remove vertices from a group as follows:

- Select the group from the group list (make it the active group).
- Select all vertices that you want to remove from the group.
- Press the *Remove* button.

Note that Removing weight Assignments is not available for locked Vertex Groups. The Remove button is grayed out in that case.

**Using groups for Selecting/Deselecting** You can quickly select all assigned vertices of a group:

- (optionally) press **A** once or twice to unselect all vertices.
- Select the group from the group list (make it the active group).
- When you now LMB click the *Select* button, then the vertices assigned to the active group will be selected and highlighted in the 3D Viewport.
- When you LMB click the *Deselect* button instead, then the vertices assigned to the active group will be deselected in the 3D Viewport.

**Note:** Selecting/Deselecting is additive

If you already have verts selected in the 3D View, then selecting the verts of a group will add the verts but also keep the already-selected verts selected. Vice versa, deselecting the verts of a vertex group will only deselect the verts assigned to the group and keep all other verts selected.

**Finding ungrouped verts** You can find ungrouped vertices as follows:

- Press A once or twice to unselect all vertices.
- In the footer of the 3D Viewport: Navigate to Select → Ungrouped verts

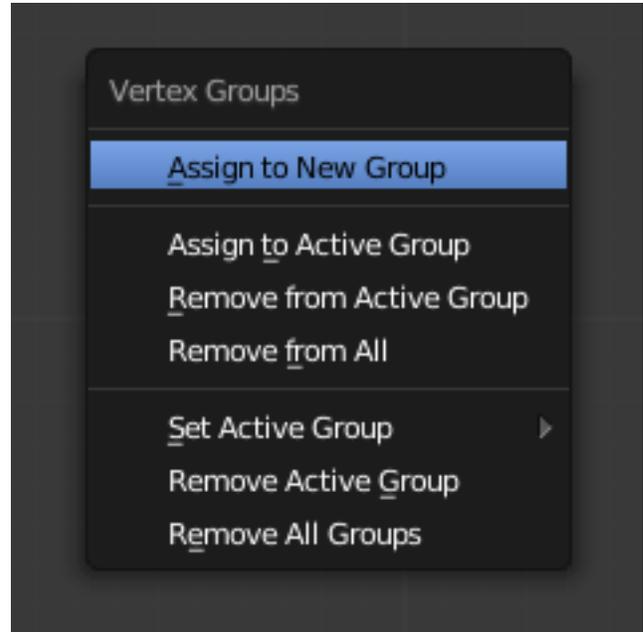


Fig. 2.432: Vertex Groups pop-up menu

**Keyboard Shortcuts** In Edit Mode you can press `Ctrl-G` to a shortcut Menu for adding/removing verts to/from groups. The pop-up menu provides the following functions with obvious functionality: (also available via *Mesh* → *Vertices* → *Vertex Groups*)

- Assign to New Group
- Assign to Active Group
- Remove from Active Group
- Remove from All

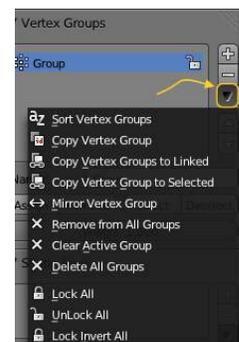


Fig. 2.433: Vertex groups panel's dropdown menu

**Vertex Group Management** Vertex Groups provide a more complex set of functions inside a Pull down menu. This menu is accessible from the Vertex Group Panel by clicking on the dark gray *arrow down* icon on the right panel border.

The following functions of the Pulldown Menu operate on the assigned vertices:

**Sort Vertex Groups:** Sorts Vertex Groups Alphabetically

**Copy Vertex Group:** Add a Copy of the active Vertex Group as a new Group. The new group will be named like the original group with “\_copy” appended at the end of its name. And it will contain associations to exactly the same verts with the exact same weights as in the source vertex group.

**Copy Vertex Groups to Linked:** Copy Vertex Groups of this Mesh to all linked Objects which use the same mesh data (all users of the data).

**Copy Vertex Group to Selected:** Copy all Vertex Groups to other Selected Objects provided they have matching indices (typically this is true for copies of the mesh which are only deformed and not otherwise edited).

**Mirror Vertex Group:** Mirror all Vertex Groups, flip weights and/or names, editing only selected vertices, flipping when both sides are selected; otherwise copy from unselected. Note this function will be reworked (and fully documented) in a future release.

**Remove from All Groups:** (not available for locked groups) Unassigns the selected Vertices from all groups. After this operation has been performed, the verts will no longer be contained in any vertex group.

**Clear Active group (not available for locked groups):** Remove all assigned vertices from the active Group. The group is made empty. Note that the vertices may still be assigned to other Vertex Groups of the Object.

**Delete All Groups:** Remove all Vertex Groups from the Object.

The following functions operate only on the lock state settings:

**Lock All** Lock all groups

**Unlock All** Unlock all groups

**Lock\_Invert All** Invert Group Locks

## Hints

- Multiple objects sharing the same mesh data have the peculiar property that the group names are stored on the object, but the weights in the mesh. This allows you to name groups differently on each object, but take care because removing a vertex group will remove the group from all objects sharing this mesh.

## Weight Editing

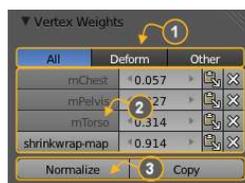


Fig. 2.434: Vertex Weights Panel

As mentioned before in [Vertex Groups](#) each entry in a Vertex Group also contains a weight value in the range of [0.0,1.0]. Blender provides a *Vertex Weights* panel from where you can get (and edit) information about the weight values of each Vertex of a mesh. That is: to which Vertex Groups the vertex is assigned with which weight value.

The Vertex Weights panel can be found in the right property sidebar of the 3D Viewport. It is available in Edit mode and in Weight Paint mode (when Vertex Selection masking is enabled as well). The panel is separated into the sections

- Vertex Group Categories (1)
- Weight Table (2)
- function bar (3)

**Vertex Group Categories** Actually we do not have any strict categories of Vertex Groups in Blender. Technically they all behave the same way. However we can identify 2 implicit categories of Vertex Groups:

**The Deform Groups** These Vertex groups are sometimes also named *Weight Groups*. They are used for defining the weight tables of Armature bones. All Deform Groups of an Object are strictly related to each other via their weight values.

Strictly speaking, the sum of all deform weights for any vertex of a mesh should be exactly 1. 0. In Blender this constraint is a bit relaxed (see below). Nevertheless, Deform Groups should always be seen as related to each other. Hence we have provided a filter that allows restricting the Vertex Weight panel to display only the Deform bones of an Object.

**The Other Groups** All other usages of Vertex Groups are summarized into the *Other* category. These vertex groups can be found within Shape keys, Modifiers, etc... There is really no good name for this category, so we kept it simple and named it *Other*.

**The Weight Table** The Weight Table shows all weights associated to the *active vertex*. Note that a vertex does not necessarily have to be associated to any vertex groups. In that case the Vertex Weights Panel is not displayed.

---

**Tip:** The active Vertex

That is the most recently selected vertex. This vertex is always highlighted so that you can see it easily in the mesh. If the active Vertex does not have weights, or there is no active vertex selected at the moment, then the Vertex Weights Panel disappears.

---

Each row in the Weight table contains 4 active elements:

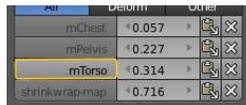


Fig. 2.435: Change Active Group

**Set the Active Group** As soon as you select any of the Vertex Group Names in the Weight table, the referenced Vertex Group becomes the new Active group.

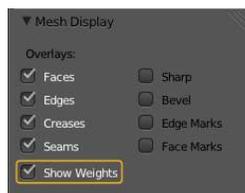


Fig. 2.436: Enable display of Weights in Edit Mode

**Display Weights in Edit Mode** When you are in edit mode, you can make the Weights of the active Group visible on the mesh:

Search the *Mesh Display* panel in the Properties sidebar. And there enable the *Show Weights* option. Now you can see the weights of the active Vertex Group displayed on the mesh surface.

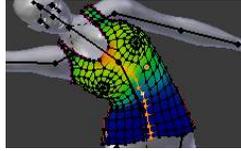


Fig. 2.437: Weights in Edit Mode

**Edit Weights in Edit Mode** It is now very easy to work with weightmaps in Edit mode. All edit options of the mesh are available and you have direct visual control over how your Weights change when you edit the weight values.

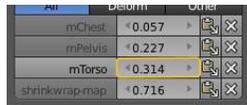


Fig. 2.438: Change Weight Value

**Change a weight** You can either enter a new weight value manually (click on the number and edit the value), or you can change the weight by LMB and while holding down the mouse button, drag right or left to increase/decrease the weight value. You also can use the right/left arrows displayed around the weight value to change the weight in steps.

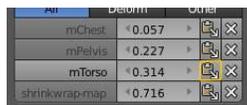


Fig. 2.439: Paste weights

**Paste a weight to other verts** LMB the Paste Icon allows you to forward a single weight of the active Vertex to all selected vertices. But note that weights are only pasted to verts which already have a weight value in the affected Vertex Group.

**Delete a weight from a Group** LMB the Delete Icon will instantly remove the weight from the active vertex. Thus the entire row disappears when you click on the delete icon.

**The Function bar** The function bar contains 2 functions:

**Normalize** Normalizes the weights of the active Vertex. That is all weights of the active vertex are recalculated such that their relative weight is maintained and the weight sum is 1.0.

**Copy** Copies all weights defined for the active Vertex to all selected Verts. Thus all previously defined weights are overwritten.

**Tip:** The filter setting is respected

Note that both functions only work on the Vertex Groups currently displayed in the Weights Table. So if for example only the *Deform weights* are displayed, then Normalize and Copy only affect the Deform bones.

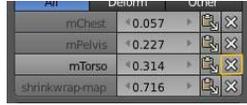


Fig. 2.440: Delete weights

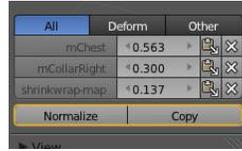


Fig. 2.441: Vertex Weights panel Function Bar

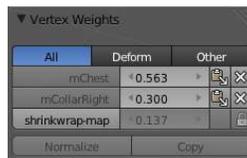


Fig. 2.442: Vertex Weights panel Locked

**About locked Vertex Groups** Whenever a Weight Group is locked, all data changing functions get disabled:

- Normalize the vertex Weights.
- Copy the Vertex weights.
- Change the Weight of the active vert.
- Paste to selected verts.

---

**Tip:** The filter setting is respected

If you have for example all deform weight groups unlocked and all other vertex groups locked, then you can safely select *Deform* from the Filter row and use all available functions from the Weight table again.

---

## Weight Paint Mode

Vertex Groups can potentially have a very large number of associated vertices and thus a large number of weights (one weight per assigned vertex). *Weight Painting* is a method to maintain large amounts of weight information in a very intuitive way. It is primarily used for rigging meshes, where the vertex groups are used to define the relative bone influences on the mesh. But we use it also for controlling particle emission, hair density, many modifiers, shape keys, etc.

The basic principle of the method is: the weight information is literally *painted* on top of the Mesh body by using a set of Weight brushes. And since painting is always associated with color, we also need to define ...

### Weight Paint in a nutshell

- You enter *Weight Paint* mode from the Mode Menu (Ctrl-Tab). The selected Mesh Object is displayed slightly shaded with a rainbow color spectrum.
- The color visualizes the weights associated to each vertex in the active Vertex Group. Blue means unweighted; Red means fully weighted.

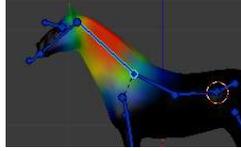


Fig. 2.443: Weight Painted Vertex Group

- You can customize the colors in the weight gradient by enabling *Custom Weight Paint Range* in the *System* tab of the *User Preferences*.
- You assign weights to the vertices of the Object by painting on it with weight brushes. Starting to paint on a mesh automatically adds weights to the active Vertex Group (a new Vertex Group is created if needed).

---

**Tip:** Useful Keyboard Shortcuts

The shortcuts can speed up your weight painting:

**Weight color picker** `Ctrl-LMB` change current weight value to the weight value of clicked vertex

**Resize the brush** `F` then drag to new brush size

**Create linear gradient** `Alt-LMB` then drag

**Create radial gradient** `Alt-Ctrl-LMB` then drag

**Draw a Clipping Border** `Alt-B` then drag the clipping border to select the part of the 3D window which shall be kept visible. You can then draw only in this part. Press `Alt-B` again to remove the *clipping border*.

---

**The weighting Color Code** Weights are visualized by using a cold/hot color system, such that areas of low influence (with weights close to 0.0) are drawn in blue (cold) and areas of high influence (with weights close to 1.0) are drawn in red (hot). And all in-between influences are drawn in rainbow colors, depending on their value (blue, green, yellow, orange, red)

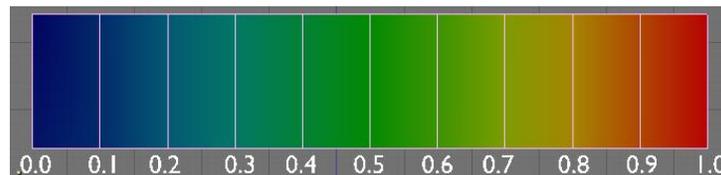


Fig. 2.444: Image 3: The color spectrum and their respective weights.

In addition to the above described color code, Blender has added (as an option) a special visual notation for unreferenced vertices: They are drawn in black. Thus you can see the referenced areas (drawn in cold/hot colors) and the unreferenced areas (in black) at the same time. This is most practical when you look for weighting errors (we will get back to this later).

**Brushes** Painting needs paint brushes and Blender provides a Brush Panel within the Tool Shelf when it operates in *Weight Paint Mode*. You find predefined Brush Presets when you click on the large Brush Icon at the top of the brush Panel. And you can make your own presets as needed. See below for the available brush presets and to create custom presets.

**The main brush properties** The most important and frequently modified properties are:

**Weight** The weight (color) to be used by the brush. However, the weight value is applied to the Vertex Group in different ways depending on the selected Brush Blending mode (see below).

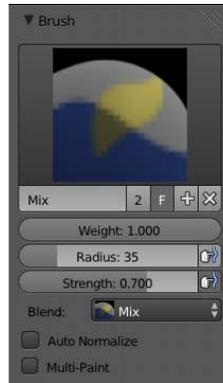


Fig. 2.445: The Brush panel in the Tool Shelf

**Strength** This is the amount of paint to be applied per brush stroke. What that means exactly also depends on the Brush Blending mode.

**Radius** The radius defines the area of influence of the brush.

---

**Note:** You can also change the Brush radius with a keyboard shortcut while painting. Just press `F` at any time, then drag the mouse to increase/reduce the brush radius. Finally click `LMB` to use the new setting. Or press the `ESC` key at any time to return to the current settings.

---

**Blend mode** The brush Blending mode defines in which way the weight value is applied to the Vertex Group while painting.

**Mix** In this Blend mode the Weight value defines the *target weight* that will eventually be reached when you paint long enough on the same location of the mesh. And the strength determines how many strokes you need to arrive at the target weight. Note that for strength = 1.0 the target weight is painted immediately, and for Weight = 0.0 the brush just does nothing.

**Add** In this blend mode the specified weight value is *added* to the vertex weights. The strength determines which fraction of the weight gets added per stroke. However, the brush will not paint weight values above 1.0.

**Subtract** In this blend mode the specified weight is *subtracted* from the vertex weights. The strength determines which fraction of the weight gets removed per stroke. However the brush will not paint weight values below 0.0.

**Lighten** In this blend mode the specified weight value is interpreted as the target weight very similar to the Mix Blend mode. But only weights below the target weight are affected. Weights above the target weight remain unchanged.

**Darken** This Blend mode is very similar to the Lighten Blend mode. But only weights above the target weight are affected. Weights below the target weight remain unchanged.

**Multiply** Multiplies the vertex weights with the specified weight value. This is somewhat like subtract, but the amount of removed weight is now dependent on the Weight value itself.

**Blur** tries to smooth out the weighting of adjacent vertices. In this mode the Weight Value is ignored. The strength defines how effectively the blur is applied.

**Normalize Options** Blender also provides Options regarding the automatic normalizing of all affected Vertex groups:

**Auto Normalize** Ensures that all deforming vertex groups add up to 1 while painting. When this option is turned off, then all weights of a vertex can have any value between 0.0 and 1.0. However, when Vertex Groups are used as Deform Groups for character animation then Blender always interprets the weight values relative to each other. That is, Blender always does a normalization over all deform bones. Hence in practice it is not necessary to maintain a strict normalization and further normalizing weights should not affect animation at all.

**Multi-Paint** Paint on all selected Vertex Groups simultaneously. This option is only useful in the context of Armatures, where you can select multiple Vertex Groups by selecting multiple Pose bones.

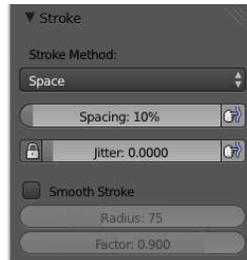


Fig. 2.446: Stroke Panel

### The Brush stroke definition

#### Stroke Method

**Airbrush** Keep applying paint effect while holding mouse down (spray)

**Space** Limit brush application to the distance specified by spacing (see below)

**Dots** Apply paint on each mouse move step

**Rate (only for Airbrush)** Interval between paints for airbrush

**Spacing (only for Space)** Limit brush application to the distance specified by spacing

**Jitter** Jitter the position of the brush while painting

**Smooth Stroke** Brush lags behind mouse and follows a smoother path

**Radius** Minimum distance from last point before stroke continues

**Factor** Higher values give a smoother stroke

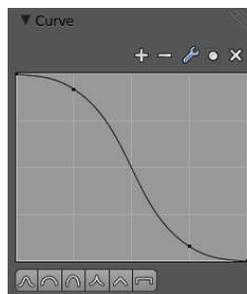


Fig. 2.447: Curve Panel

**The brush Falloff curve** The brush falloff editor allows you to specify the characteristics of your brushes to a large extent. The usage should be obvious and intuitive.

### The brush appearance

**Show Brush** makes the brush visible as a circle (on by default)

**Color setter** To define the color of the brush circle

**Custom icon** Allows definition of a custom brush icon

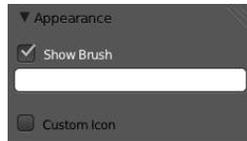


Fig. 2.448: Brush appearance

**Brush presets** Blender provides several Brush presets:

- **Mix, Draw, Brush** : uses the Mix Blending mode to draw the brush weight with varying strength and brush falloff
- **Add** : uses the Add Blending mode
- **Subtract** : uses the Subtract Blending mode
- **Lighten** : uses the Lighten Blending mode
- **Darken** : uses the Darken Blending mode
- **Multiply** : uses the Multiply Blending mode
- **Blur** : uses the Blur Blending mode

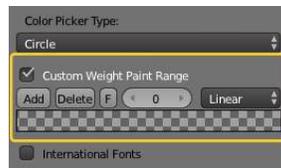


Fig. 2.449: Customizing the Color Band

**Customizing brush color space** Blender allows customization of the color range used for the Weight Paint colors. You can define the color band as you like; for example, you can make it purely black/white (similar to maya Weight painting), and you can even use Alpha values here.

You find the customizer in the User Properties section, in the System Tab.

**Selection Masking** If you have a complex mesh, it is sometimes not easy to paint on all vertices in Weight Paint mode. Suppose you only want to paint on a small area of the Mesh and keep the rest untouched. This is where *selection masking* comes into play. When this mode is enabled, a brush will only paint on the selected verts or faces. The option is available from the footer menu bar of the 3D viewport (see icons surrounded by the yellow frame):



You can choose between *Face Selection masking* (left icon) and *Vertex selection masking* (right icon).

*Select* mode has some advantages over the default *Weight Paint* mode:

- The original mesh edges are drawn, even when modifiers are active.
- You can select faces to restrict painting to the vertices of the selected faces.
- Selecting tools include:

**Details about selecting** The following standard selection operations are supported:

- RMB - Single faces. Use `Shift-RMB` to select multiple.
- A - All faces, also to de-select.
- B - Block/Box selection.
- C - Select with brush.
- L - Pick linked (under the mouse cursor).
- `Ctrl-L` - Select linked.
- `Ctrl-I` - Invert selection (*Inverse*).

---

**Tip:** Selecting Deform Groups

When you are doing weight painting for deform bones (with an Armature), you can select a deform group by selecting the corresponding bone. However, this Vertex Group selection mode is disabled when Selection Masking is active!

---

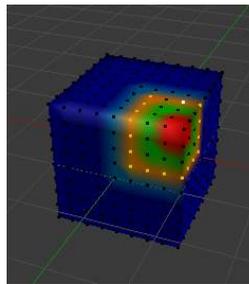


Fig. 2.450: Vertex Selection masking

**Vertex Selection Masking** In this mode you can select one or more vertices and then paint only on the selection. All unselected vertices are protected from unintentional changes.

---

**Note:** This option can also be toggled with the `V` key:

---

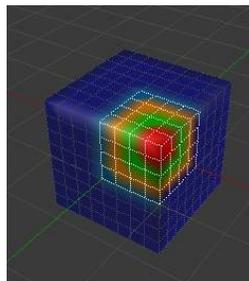


Fig. 2.451: Face Selection masking

**Face Selection Masking** The *Face Selection masking* allows you to select faces and limit the weight paint tool to those faces, very similar to Vertex selection masking.

**Hide/Unhide Faces** You also can hide selected faces as in Edit Mode with the keyboard Shortcut `H`, then paint on the remaining visible faces and finally unhide the hidden faces again by using `Alt-H`

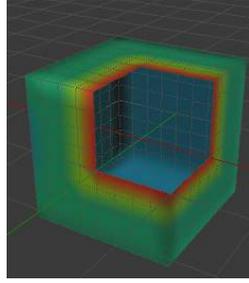


Fig. 2.452: hidden faces

**Hide/Unhide Vertices** You cannot directly hide selected faces in vertex mask selection mode. However you can use a trick:

- First go to Face selection mask mode
- Select the areas you want to hide and then hide the faces (as explained above)
- Switch back to Vertex Selection mask mode

Now the verts belonging to the hidden Faces will remain hidden.

**The Clipping Border** To constrain the paint area further you can use the *Clipping Border*. Press `Alt+B` and `LMB` -drag a rectangular area. The selected area will be “cut out” as the area of interest. The rest of the 3D window gets hidden.

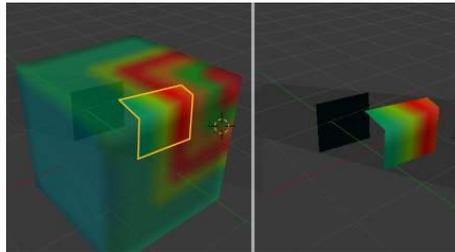


Fig. 2.453: The Clipping Border is used to select interesting parts for local painting

You make the entire mesh visible again by pressing `Alt+B` a second time.

All weight paint tools that use the view respect this clipping, including border select, weight gradient and of course brush strokes.

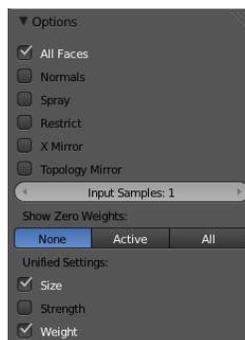


Fig. 2.454: Weight Paint Options

**Weight Paint Options** The Weight Paint Options modify the overall brush behavior:

**Normals** The vertex normal (helps) determine the extent of painting. This causes an effect as if painting with light.

**Spray** This option accumulates weights on every mouse move.

**Restrict** This option limits the influence of painting to vertices belonging (even with weight 0) to the selected vertex group.

**X-mirror** Use the X-mirror option for mirrored painting on groups that have symmetrical names, like with extension `.R / .L`, or `_R / _L`. If a group has no mirrored counterpart, it will paint symmetrically on the active group itself. You can read more about the naming convention in [Editing Armatures: Naming conventions](#). The convention for armatures/bones apply here as well.

**Topology Mirror** Use topology-based mirroring, for when both side of a mesh have matching mirrored topology.

**Input Samples** not so sure

**Show Zero Weights**

- None
- Active
- All

**Unified Settings:** The *Size*, *Strength* and *Weight* of the brush can be set to be shared across different brushes, as opposed to per-brush.

- Spray: to constantly draw (opposed to drawing one stroke per mouse click).
- Restrict: to only paint on vertices which already are weighted in the active weight group. (No new weights are created; only existing weights are modified.)
- x-mirror: to draw symmetrically. Note the this only works when the character symmetry plane is z-y (character looks into y direction).
- Show Zero weights: To display unreferenced and zero weighted areas in black (by default).



Fig. 2.455: Weight Paint Tools

**Weight Paint Tools** Blender provides a set of helper tools for Weight Painting. The tools are located in the weight tools panel.

The weight paint tools are full described in the [Weight Paint Tools](#) page

**Weight Painting for Bones** This is probably the most often used application of weight painting. When a bone moves, vertices around the joint should move as well, but just a little, to mimic the stretching of the skin around the joint. Use a “light” weight (10-40%) paint on the vertices around the joint so that they move a little when the bone rotates. While there are ways to automatically assign weights to an armature (see the [Armature section](#)), you can do this manually. To do this from scratch, refer to the process below. To modify automatically assigned weights, jump into the middle of the process where noted:

- Create an armature.
- Create a mesh that will be deformed when the armature’s bone(s) move.
- With the mesh selected, create an *Armature* modifier for your mesh (located in the *Editing* context, *Modifiers* panel). Enter the name of the armature.

*Pick up here for modifying automatically assigned weights.*

- Select the armature in 3D View, and bring the armature to *Pose mode* (Ctrl-Tab, or the 3D View window header mode selector).
- Select a desired bone in the armature.
- Select your mesh (using RMB) and change immediately to *Weight Paint* mode. The mesh will be colored according to the weight (degree) that the selected bone movement affects the mesh. Initially, it will be all blue (no effect).
- Weight paint to your heart’s content. The mesh around the bone itself should be red (generally) and fade out through the rainbow to blue for vertices farther away from the bone.

You may select a different bone with RMB while weight painting, provided the armature was left in *Pose* mode as described above. This will activate the vertex group sharing the name with the selected bone, and display related weights. If the mesh skins the bones, you will not be able to see the bones because the mesh is painted. If so, turn on *X-Ray* view (*Buttons* window, *Editing* context, *Armature* panel). While there on that panel, you can also change how the bones are displayed (*Octahedron*, *Stick*, *B-Bone*, or *Envelope*) and enable *Draw Names* to ensure the name of the selected bone matches up to the vertex group.

If you paint on the mesh, a vertex group is created for the bone. If you paint on vertices outside the group, the painted vertices are automatically added to the vertex group.

If you have a symmetrical mesh and a symmetrical armature you can use the option *X-Mirror*. Then the mirrored groups with the mirrored weights are automatically created.

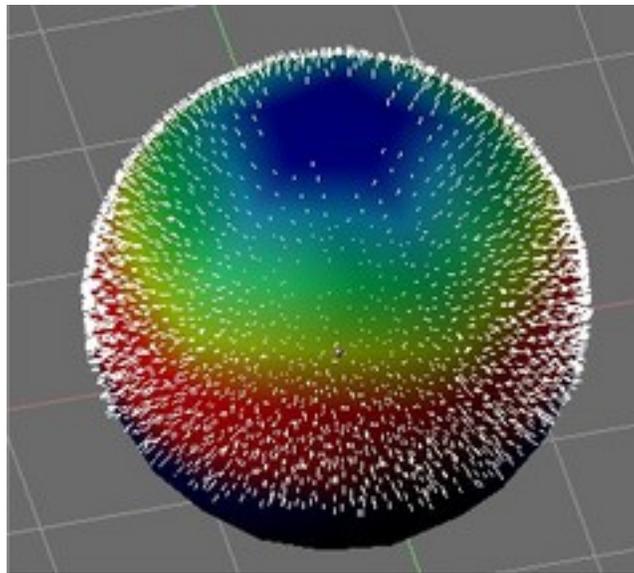


Fig. 2.456: Weight painted particle emission.

**Weight Painting for Particles** Faces or vertices with zero weight generate no particles. A weight of 0.1 will result in 10% of the amounts of particles. This option “conserves” the total indicated number of particles, adjusting the distributions so that the proper weights are achieved while using the actual number of particles called for. Use this to make portions of your mesh hairier than others by weight painting a vertex group, and then calling out the name of the vertex group in the *VGroup:* field (*Particles* panel, *Object* context).

## Weight Tools



Fig. 2.457: Weight Paint Tools

Blender provides a set of helper tools for Weight Painting. The tools are accessible from the Tool Shelf in Weight Paint mode. And they are located in the weight tools panel.

**The Subset Option** Some of the tools also provide a Subset parameter (in the Operator panel, displayed after the tool is called) with following options:

- Active Group
- Selected Pose Bones
- Deform pose Bones
- All Groups

All tools also work with Vertex Selection Masking and Face Selection masking. In these modes the tools operate only on selected verts or faces.

---

**Tip:** About the Blend tool

The Blend tool only works when “Vertex selection masking for painting” is enabled. Otherwise the tool button is grayed out.

---

**Normalize All** For each vertex, this tool makes sure that the sum of the weights across all Vertex Groups is equal to 1. This tool normalizes all of the vertex groups, except for locked groups, which keep their weight values untouched.

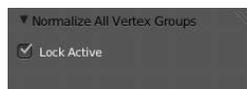


Fig. 2.458: Normalize All Options

### Operator Parameters

**Lock Active** Keep the values of the active group while normalizing all the others.

---

**Note:** Currently this tool normalizes ALL vertex groups except the locked vertex groups.

---

**Normalize** This tool only works on the active Vertex Group. All vertices keep their relative weights, but the entire set of weights is scaled up such that the highest weight value is 1.0

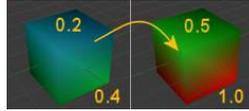


Fig. 2.459: Normalize All Options

**Operator Parameters** None

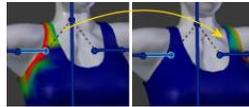


Fig. 2.460: Normalize All Options

**Mirror** This tool mirrors the weights from one side of the mesh to the opposite side (only mirroring along x-axis is supported). But note, the weights are not transferred to the corresponding opposite bone weight group. The mirror only takes place within the selected Vertex Group.

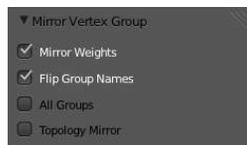


Fig. 2.461: Mirror Options

**Operator Parameters**

**Mirror Weights** Mirrors the weights of the active group to the other side. Note, this only affects the active weight group.

**Flip Group Names** Exchange the names of left and right side. This option only renames the groups.

**All Groups** Operate on all selected bones.

**Topology Mirror** Mirror for meshes which are not 100% symmetric (approximate mirror).

---

**Tip:** Mirror to opposite bone

If you want to create a mirrored weight group for the opposite bone (of a symmetric character), then you can do this:

- Delete the target Vertex Group (where the mirrored weights will be placed)
  - Create a copy of the source bone Vertex Group (the group containing the weights which you want to copy)
  - Rename the new Vertex Group to the name of the target Vertex Group (the group you deleted above)
  - Select the Target Vertex Group and call the Mirror tool (use only the Mirror weights option and optionally Topology Mirror if your mesh is not symmetric)
- 

**Invert** Replaces each Weight of the selected weight group by  $1.0 - \text{weight}$ .

Examples:

- original 1.0 converts to 0.0
- original 0.5 remains 0.5

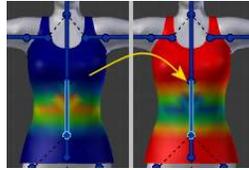


Fig. 2.462: Invert

- original 0.0 converts to 1.0



Fig. 2.463: Mirror Options

### Operator Parameters

**Subset** Restrict the tool to a subset. See above (*The Subset Option*) about how subsets are defined.

**Add Weights** Add verts that have no weight before inverting (these weights will all be set to 1.0)

**Remove Weights** Remove verts from the Vertex Group if they are 0.0 after inverting.

---

**Note:** Locked vertex Groups are not affected.

---

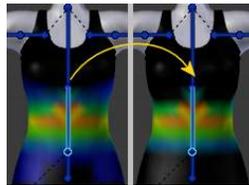


Fig. 2.464: Invert

**Clean** Removes weights below a given threshold. This tool is useful for clearing your weight groups of very low (or zero-) weights.

In the example shown, I used a cutoff value of 0.139 (see operator options below) so all blue parts (left side) are cleaned out (right side).

Note, the images use the *Show Zero weights =Active* option so that unreferenced Weights are shown in Black.

### Operator Parameters

**Subset** Restrict the tool to a subset. See above (*The Subset Option*) for how subsets are defined.

**Limit** This is the minimum weight value that will be kept in the Group. Weights below this value will be removed from the group.

**Keep Single** Ensure that the Clean tool will not create completely unreferenced verts (verts which are not assigned to any Vertex Group), so each vertex will keep at least one weight, even if it is below the limit value!



Fig. 2.465: Mirror Options

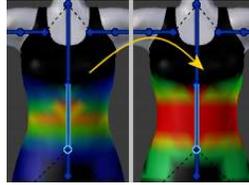


Fig. 2.466: Invert

**Levels** Adds an offset and a scale to all weights of the selected Weight Groups. with this tool you can raise or lower the overall “heat” of the weight group.

**Note:** No weight will ever be set to values above 1.0 or below 0.0 regardless of the settings.



Fig. 2.467: Mirror Options

### Operator Parameters

**Subset** Restrict the tool to a subset. See above (*The Subset Option*) for how subsets are defined.

**Offset** A value from the range [-1.0,1.0]) to be added to all weights in the Vertex Group.

**Gain** All weights in the Subset are multiplied with the gain. The drag sliders of this value allow only a range of [-10.0, 10.0]. However, you can enter any factor you like here by typing from the keyboard.

**Note:** Whichever Gain and Offset you choose, in all cases the final value of each weight will be clamped to the range [0.0, 1.0]. So you will never get negative weights or overheated areas (weight > 1.0) with this tool.

**Blend** Blends the weights of selected vertices with adjacent unselected vertices. This tool only works in vertex select mode.

To understand what the tool really does, let’s take a look at a simple example. The selected vertex is connected to 4 adjacent vertices (marked with a gray circle in the image). All adjacent vertices are unselected. Now the tool calculates the average weight of all connected **and** unselected verts. In the example this is:

$$(1 + 0 + 0 + 0) / 4 = 0.25$$

This value is multiplied by the factor given in the Operator parameters (see below).

- If the factor is 0.0 then actually nothing happens at all and the vertex just keeps its value.

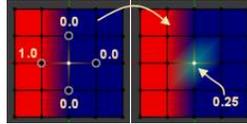


Fig. 2.468: Blending

- If the factor is 1.0 then the calculated average weight is taken (0.25 here).
- Dragging the factor from 0 to 1 gradually changes from the old value to the calculated average.

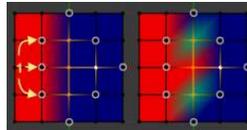


Fig. 2.469: Blending

Now let's see what happens when we select all but one of the neighbors of the selected vert as well. Again all connected and unselected verts are marked with a gray circle. When we call the Blend tool now and set the Factor to 1.0, then we see different results for each of the selected verts:

- The topmost and bottommost selected verts:  
are surrounded by 3 unselected verts, with an average weight of  $(1 + 0 + 0) / 3 = 0.333$  So their color has changed to light green.
- The middle vertex:  
is connected to one unselected vert with  $\text{weight} = 1$ . So the average weight is 1.0 in this case, thus the selected vert color has changed to red.
- The right vert:  
is surrounded by 3 unselected verts with average weight  $= (0 + 0 + 0) / 3 = 0.0$  So the average weight is 0, thus the selected vert color has not changed at all (it was already blue before blend was applied).

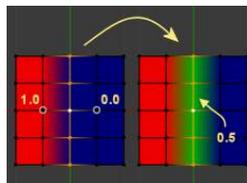


Fig. 2.470: Blending

Finally let's look at a practical example (and explain why this tool is named Blend). In this example I have selected the middle edge loop. And I want to use this edge loop for blending the left side to the right side of the area.

- All selected vertices have 2 unselected adjacent verts.
- The average weight of the unselected verts is  $(1 + 0) / 2 = 0.5$
- Thus when the Blend Factor is set to 1.0 then the edge loop turns to green and finally does blend the cold side (right) to the hot side (left).

## Operator Parameters



Fig. 2.471: Blend Options

**Factor** The effective amount of blending (range [0.0, 1.0]). When Factor is set to 0.0 then the Blend tool does not do anything. For Factor > 0 the weights of the affected vertices gradually shift from their original value towards the average weight of all connected **and** unselected verts (see examples above).

**Transfer Weights** Copy weights from other objects to the vertex groups of the active Object. By default this tool copies all vertex groups contained in the selected objects to the target object. However you can change the tool's behavior in the operator redo panel (see below).

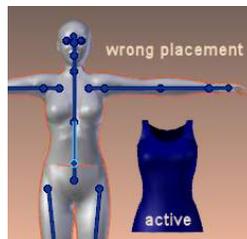


Fig. 2.472: Blending



Fig. 2.473: Blending

**Prepare the copy** You first select all source objects, and finally the target object (the target object must be the active object).

It is important that the source objects and the target object are at the same location. If they are placed side by side, then the weight transfer won't work. You can place the objects on different layers, but you have to ensure that all objects are visible when you call the tool.

Now ensure that the Target Object is in Weight Paint mode.

**Call the tool** Open the Tool Shelf and locate the Weight Tools panel. From there call the "Transfer weights" tool. The tool will initially copy all vertex groups from the source objects. However the tool also has an operator redo panel (which appears at the bottom of the tool shelf). From the redo panel you can change the parameters to meet your needs. (The available Operator parameters are documented below.)

**Redo Panel Confusion** You may notice that the Operator Redo Panel (see below) stays available after the weight transfer is done. The panel only disappears when you call another Operator that has its own redo Panel. This can lead to confusion when you use Transfer weights repeatedly after you changed your vertex groups. If you then use the still-visible redo panel, then Blender will reset your work to its state right before you initially called the Transfer Weights tool.

**Workaround** When you want to call the Transfer Weights tool again after you made some changes to your vertex groups, then always use the “Transfer Weights” Button, even if the operator panel is still available. Unless you really want to reset your changes to the initial call of the tool.

**Operator Parameters** Defaults are marked in boldface:



Fig. 2.474: Blend Options

Group:

**Active** Only copy to the Active Group in the active Object. This option only works when the active Object has an active Vertex Group set. Otherwise the Weight transfer will not do anything.

**All** Copy all Vertex groups from the selected objects to the Active Object.

Method:

- **Nearest vertex In face** : TODO
- Nearest Face: TODO
- Nearest vertex: TODO
- Vertex Index (verbatim copy, works only for meshes with identical index count)

Replace

- **Empty**: Only copy a weight to the active object if the vertex has not yet had a weight set in the group.
- **All** : delete all previous content of the target vertex group before copying the group from the source object.

---

**Note:** If a vertex group is contained in 2 or more of the selected objects, then the result depends on the order in which the selected objects are processed. However, the order of processing cannot be influenced.

---

**Limit Total** Reduce the number of weight groups per vertex to the specified Limit. The tool removes lowest weights first until the limit is reached.

Hint: The tool can only work reasonably when more than one weight group is selected.

### Operator Parameters

**Subset** Restrict the tool to a subset. See above (*The Subset Option*) for how subsets are defined.

**Limit** Maximum number of weights allowed on each vertex (default:4)

**Weight Gradient** This is an interactive tool for applying a linear/radial weight gradient; this is useful at times when painting gradual changes in weight becomes difficult.

The gradient tool can be accessed from the Toolbar as a key shortcut:

- Linear: Alt-LMB and drag

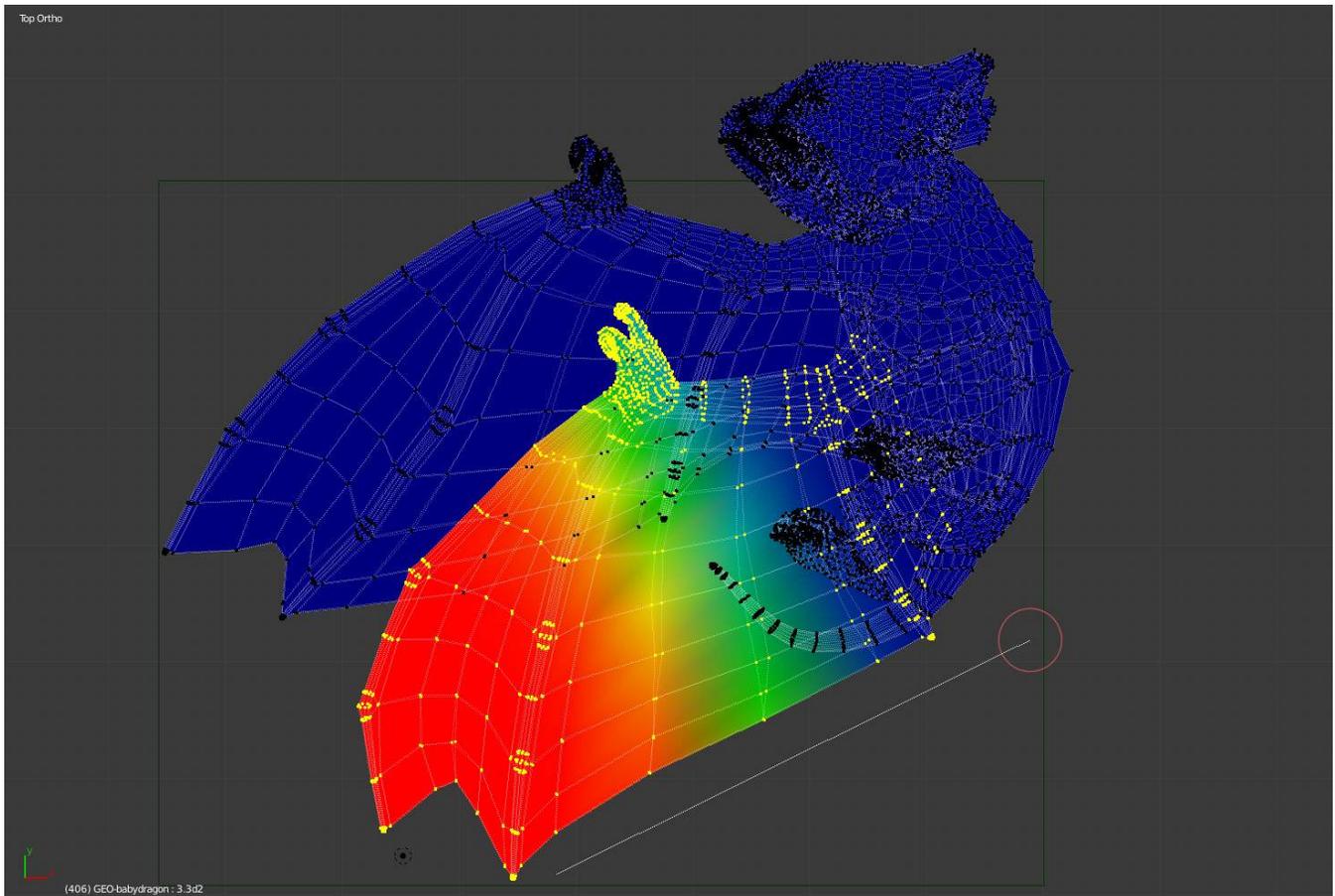


Fig. 2.475: example of the gradient tool being used with selected vertices.

- Radial: `Alt-Ctrl-LMB` and drag

The following weight paint options are used to control the gradient:

- Weight - The gradient starts at the current selected weight value, blending out to nothing.
- Strength - Lower values can be used so the gradient mixes in with the existing weights (just like with the brush).
- Curve - The brush falloff curve applies to the gradient too, so you can use this to adjust the blending.

Blends the weights of selected vertices with unselected vertices.

---

**Hint:** This tool only works in vertex select mode.

---

**Operator Parameters** Type:

- Linear
- Radial

X Start: X End: Y Start: Y End:

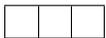
## Smoothing

### Mesh Shading

Table

2.5:

Example  
mesh ren-  
dered flat,  
smoothed  
using  
edge  
split, and  
using  
Subdi-  
vision  
Surface.  
Note how  
edges are  
rendered  
differ-  
ently.  
Sample  
.blend



As seen in the previous sections, polygons are central to Blender. Most objects are represented by polygons and truly curved objects are often approximated by polygon meshes. When rendering images, you may notice that these polygons appear as a series of small, flat faces.

Sometimes this is a desirable effect, but usually we want our objects to look nice and smooth. This section shows you how to visually smooth an object, and how to apply the *Auto Smooth* filter to quickly and easily combine smooth and faceted polygons in the same object.

The last section on this page shows possibilities for smoothing a mesh's geometry, not only its appearance.

---

### Smooth shading Reference

Mode: *Edit* and *Object* mode

Panel: *Mesh Tools* (*Editing* context)

Menu: *Mesh* → *Faces* → *Shade Smooth / Shade Flat*

Hotkey: *[ctrl][F]* → *Shade Smooth / Shade Flat*

---

Table

2.6:

Same

mesh

smooth

shaded



The easiest way is to set an entire object as smooth or faceted by selecting a mesh object, and in *Object* mode, click *Smooth* in the *Tool Shelf*. This button does not stay pressed; it forces the assignment of the “smoothing” attribute to each face in the mesh, including when you add or delete geometry.

Notice that the outline of the object is still strongly faceted. Activating the smoothing features doesn't actually modify the object's geometry; it changes the way the shading is calculated across the surfaces, giving the illusion of a smooth surface. Click the *Flat* button in the *Tool Shelf*'s *Shading panel* to revert the shading back to that shown in the first image above.

**Smoothing parts of a mesh** Alternatively, you can choose which edges to smooth by entering *Edit mode*, then selecting some faces and clicking the *Smooth* button. The selected edges are marked in yellow.

When the mesh is in *Edit mode*, only the selected edges will receive the “smoothing” attribute. You can set edges as flat (removing the “smoothing” attribute) in the same way by selecting edges and clicking the *Flat* button.

---

### Auto Smooth Reference

Panel: *Properties* (*Object Data* context)

---

Table

2.7:

Example

mesh

with Auto

Smooth

enabled

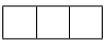


It can be difficult to create certain combinations of smooth and solid faces using the above techniques alone. Though there are workarounds (such as splitting off sets of faces by selecting them and pressing  $\Upsilon$ ), there is an easier way to combine smooth and solid faces, by using *Auto Smooth*.

Auto smoothing can be enabled in the mesh's panel in the *Properties* window. Angles on the model that are smaller than the angle specified in the *Angle* button will be smoothed during rendering (i.e. not in the 3D view) when that part of the mesh is set to smooth. Higher values will produce smoother faces, while the lowest setting will look identical to a mesh that has been set completely solid.

Note that a mesh, or any faces that have been set as *Flat*, will not change their shading when *Auto Smooth* is activated: this allows you extra control over which faces will be smoothed and which ones won't by overriding the decisions made by the *Auto Smooth* algorithm.

**Edge Split Modifier** With the [Edge Split Modifier](#) we get a result similar to *Auto Smooth* with the ability to choose which edges should be split, based on angle - those marked as sharp.



**Smoothing the mesh geometry** The above techniques do not alter the mesh itself, only the way it is displayed and rendered. Instead of just making the mesh look like a smooth surface, you can also physically smooth the geometry of the mesh with these tools:

**Mesh editing tools** You can apply one of the following in *Edit mode*:

**Smooth** This relaxes selected components, resulting in a smoother mesh.

**Laplacian Smooth** Smooths geometry by offers controls for better preserving larger details.

**Subdivide Smooth** Adjusting the *smooth* parameter after using the *subdivide* tool results in a more organic shape. This is similar to using the *subdivide* modifier.

**Bevel** This Bevels selected edged, causing sharp edges to be flattened.

**Modifiers** Alternatively, you can smooth the mesh non-destructively with one or several of the following modifiers:

**Smooth Modifier** Works like the *Smooth* tool in *Edit mode*; can be applied to specific parts of the mesh using vertex groups.

**Laplacian Smooth Modifier** Works like the *Laplacian Smooth* tool in *Edit mode*; can be applied to specific parts of the mesh using vertex groups.

**Bevel Modifier** Works like the *Bevel* tool in *Edit mode*; Bevel can be set to work on an angle threshold, or on edge weight values.

**Subdivision Surface Modifier** Catmull-Clark subdivision produces smooth results. Sharp edges can be defined with `subdivision creases` or by setting certain edges to “sharp” and adding an [EdgeSplit modifier](#) (set to *From Marked As Sharp*) before the *Subsurf* modifier.

## Mesh Clean-up

These tools are to help cleanup degenerate geometry and fill in missing areas of a mesh.

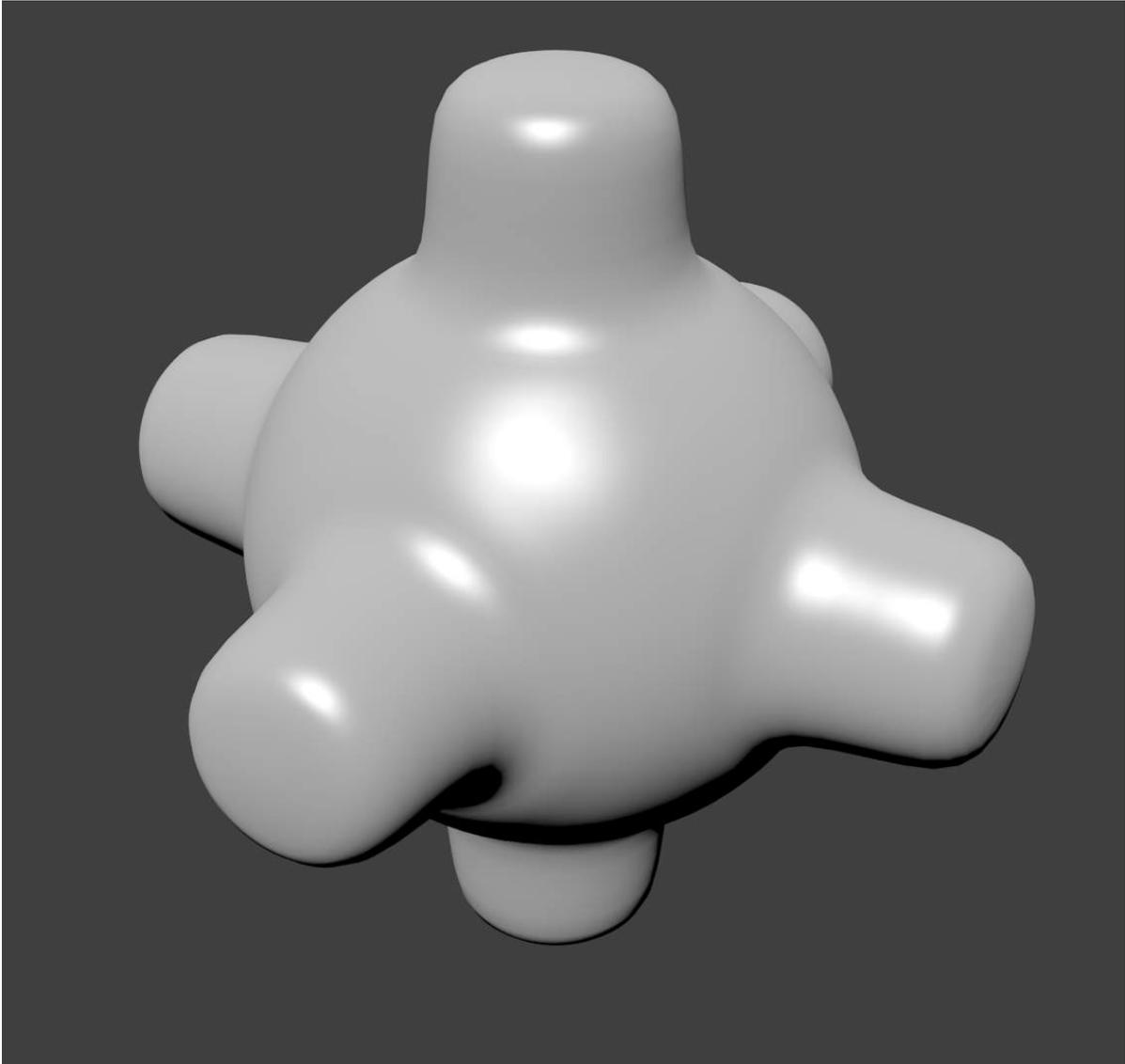


Fig. 2.482: Subsurf

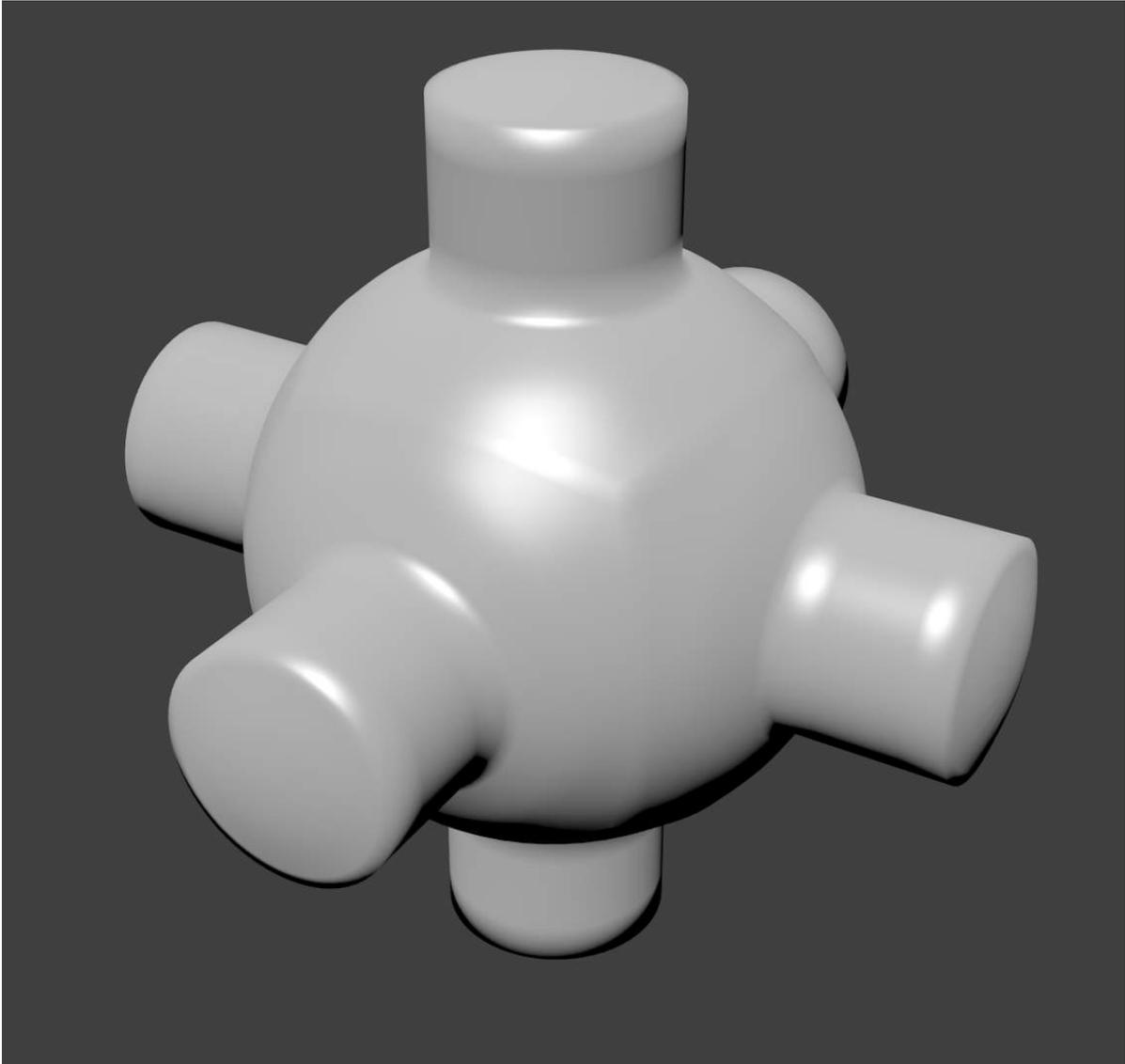


Fig. 2.483: Using creased edges, and resulting subsurf artifacts

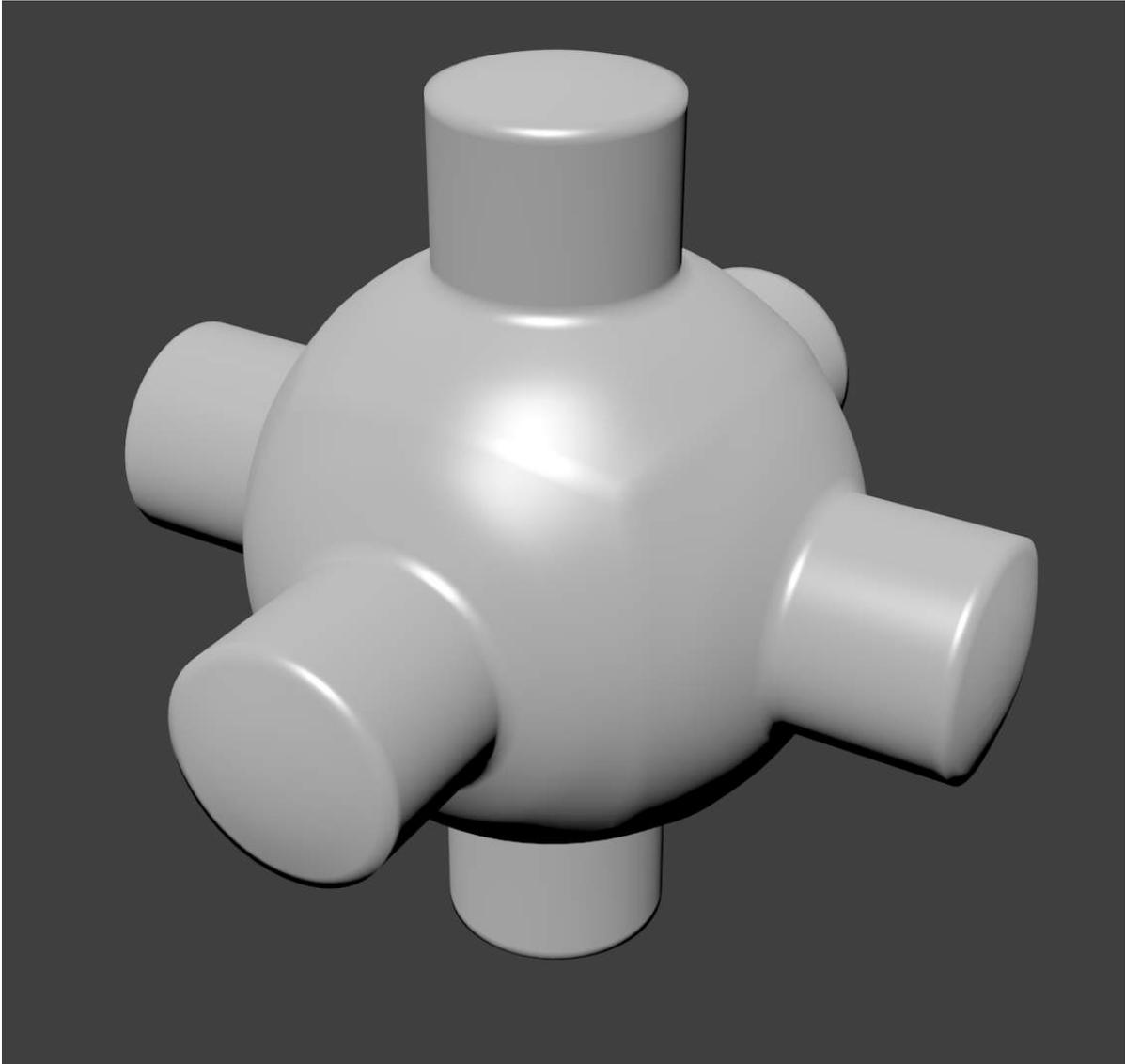


Fig. 2.484: Extra edge loops added

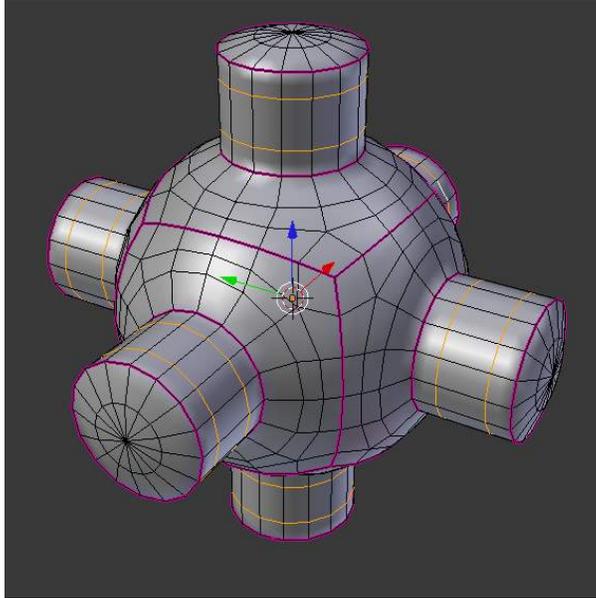


Fig. 2.485: 3D view showing creased edges (pink) and added edges loops (yellow)

### Fill Holes

---

#### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Clean up* → *Fill Holes*

---

This tool can take a large selection and detect the holes in the mesh, filling them in.

This is different from the face creation operator in three important respects.

- holes are detected, so there is no need to manually find and select the edges around the holes.
- holes can have a limit for the number of sides (so only quads or tris are filled in for example).
- mesh data is copied from surrounding geometry (UV's, vertex-colors, multi-res, all layers), since manually creating this data is very time consuming.

### Split Non-Planar Faces

---

#### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Clean up* → *Split Non-Planar Faces*

---

This tool avoids ambiguous areas of geometry by splitting non-flat faces when they are bent beyond a given limit.

---

## Delete Loose Geometry

---

### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Clean up* → *Delete Loose*

---

This tool removes disconnected vertices and edges (optionally faces - off by default).

---

## Degenerate Dissolve

---

### Reference

Mode: *Edit* mode

Menu: *Mesh* → *Clean up* → *Degenerate Dissolve*

---

This tool collapses / removes geometry which you typically won't want.

- Edges with no length.
- Faces with no areas (faces on a point or thin faces).
- Face corners with no area.

## 2.3.4 Curves

### Curves

Curves and [Surfaces](#) are particular types of Blender Objects. They are expressed by mathematical functions rather than a series of points.

Blender offers both *Bezier Curves* and *Non-Uniform Rational B-Splines (NURBS)*. Both Bezier curves and NURBS curves and surfaces are defined in terms of a set of “control points” (or “control vertices”) which define a “control polygon”.

Both bezier and NURBS curves are named after their mathematical definitions, and choosing between them is often more a matter of how they are computed behind the scenes than how they appear from a modeler's perspective. Bezier curves are generally more intuitive because they start and end at the control points that you set, but NURBS curves are more efficient for the computer to calculate when there are many twists and turns in a curve.

The main advantage to using curves instead of polygonal meshes is that curves are defined by less data and so can produce results using less memory and storage space at modeling time. However, this procedural approach to surfaces can increase demands at render time.

Certain modeling techniques, such as extruding a profile along a path, are possible only using curves. On the other hand, when using curves, vertex-level control is more difficult and if fine control is necessary, [mesh editing](#) may be a better modeling option.

Bezier curves are the most commonly used curves for designing letters or logos. They are also widely used in animation, both as [paths](#) for objects to move along and as [F-curves](#) to change the properties of objects as a function of time.



Fig. 2.486: Bird logo made from Bezier curves.

## Tutorials

Create the bird logo with Bezier Curves

Skinning: Making a surface with two or more curves

## Curve Primitives

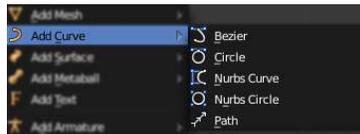


Fig. 2.487: Add Curve menu.

In Object mode, the *Add Curve* menu, Blender provides five different curve primitives:

**Bezier Curve** Adds an open 2D Bezier curve with two control points.

**Bezier Circle** Adds a closed, circle-shaped 2D Bezier curve (made of four control points).

**NURBS Curve** Adds an open 2D NURBS curve, with four control points, with *Uniform* knots.

**NURBS Circle** Adds a closed, circle-shaped 2D NURBS curve (made of eight control points).

**Path** Adds a NURBS open 3D curve made of five aligned control points, with *Endpoint* knots and the *CurvePath* setting enabled.

## Bezier Curves

The main elements used in editing Bezier Curves are the Control Points and Handles. A Segment (the actual Curve) is found between two Control Points. In the image below, the Control Points can be found in the middle of the pink line while the Handles comprise the extensions from the Control Point. By default the arrows on the Segment represents the direction and **relative** speed and direction of movement Objects will have when moving along the curve. This can be altered by defining a custom *Speed Ipo*.

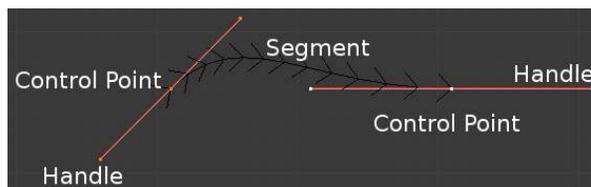


Fig. 2.488: Bezier Curve in Edit mode.

**Editing Bezier Curves** A Bezier curve can be edited by moving the locations of the Control Points and Handles.

- Add a Curve by `Shift-A` to bring up the *Add* menu, followed by *Curve* → *Bezier*.
- Press `Tab` to enter *Edit mode*.
- Select one of the Control Points and move it around. Use `LMB` to confirm the new location of the Control Point, or use `RMB` to cancel.
- Now select one of the Handles and move it around. Notice how this changes the curvature of the curve.

To add more Control Points

- Select at least two adjacent Control Points.
- Press **W** and select *Subdivide*.
- Optionally, you can press **F6** immediately after the subdivision to modify the number of subdivisions.

Note that while in *Edit mode* you cannot directly select a Segment. To do so, select all of the Control Points that make up the Segment you want to move.

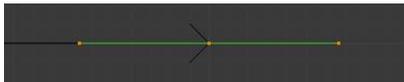
There are four Bezier curve handle types. They can be accessed by pressing **V** and selecting from the list that appears, or by pressing the appropriate hotkey combination. Handles can be rotated, moved, scaled and shrunk/fattened like any vertex in a mesh.

### Bezier Curve Handle Types

**Automatic V-A** This handle has a completely automatic length and direction which is set by Blender to ensure the smoothest result. These handles convert to *Aligned* handles when moved.



**Vector V-V** Both parts of a handle always point to the previous handle or the next handle which allows you to create curves or sections thereof made of straight lines or with sharp corners. Vector handles convert to *Free* handles when moved.



**Aligned V-L** These handles always lie in a straight line, and give a continuous curve without sharp angles.



**Free V-F** The handles are independent of each other.

Additionally, the **V-T** shortcut can be used to toggle between Free and Aligned handle types.

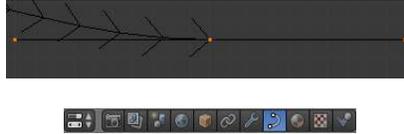
### Curve Properties

Curve Properties can be set from the *Object Data* option in the *Properties Header* (shown below in blue).

#### Shape

**2D and 3D Curves** By default, new curves are set to be 3D, which means that Control Points can be placed anywhere in 3D space. Curves can also be set to 2D which constrain the Control Points to the Curve's local XY axis.

**Resolution** The *resolution* property defines the number of points that are computed between every pair of Control Points. Curves can be made more or less smooth by increasing and decreasing the resolution respectively. The *Preview U* setting determines the resolution in the 3D viewport while the *Render U* setting determines the Curve's render resolution. If *Render U* is set to zero (0), then the *Preview U* setting is used for both the 3D viewport and render resolution.



**Twisting** A 3D Curve has Control Points that are not located on the Curve's local XY plane. This gives the Curve a twist which can affect the Curve normals. You can alter how the twist of the Curve is calculated by choosing from *Minimum*, *Tangent* and *Z-Up* options from the drop-down menu.

**Fill** Fill determines the way a Curve is displayed when it is Beveled (see below for details on Beveling). When set to *Half* (the default) the Curve is displayed as half a cylinder. The *Fill Deformed* option allows you to indicate whether the Curve should be filled before or after (default) applying any Shape Keys or Modifiers.

**Path/Curve-Deform** These options are primarily utilized when using a Curve as a Path or when using the Curve Deform property. The *Radius*, *Stretch* and *Bounds Clamp* options control how Objects use the Curve and are dealt with in more detail in the appropriate links below.

[Read more about Basic Curve Editing](#) [Read more about Paths](#) [Read more about Curve Deform](#)

## Geometry

### Modification

**Offset** By default, text Objects are treated as curves. The Offset option will alter the space between letters.

**Extrude** Will extrude the curve along both the positive and negative local Z axes.

### Bevel

**Depth** Changes the size of the bevel

**Taper Object** Tapering a Curve causes it to get thinner towards one end. You can also alter the proportions of the Taper throughout the tapered object by moving/scaling/rotating the Control Points of the Taper Object. The Taper Object can only be another Curve. Editing the Handles and Control Points of the Taper Object will cause the original Object to change shape.

**Bevel Object** Beveling a Bezier Curve with a Bezier Curve as the Bevel Object generally gives it the appearance of a plane, while using a Bezier Circle as the Bevel Object will give it the appearance of a cylinder. The Bevel Object can only be another Curve. Editing the Handles and Control Points of the Bevel Object will cause the original Object to change shape. Given the options available, it is best to experiment and see the results of this operation.

**Fill Caps** Seals the ends of a beveled Curve.

**Map Taper** For Curves using a Taper Object and with modifications to the *Start/End Bevel Factor* the *Map Taper* option will apply the taper to the beveled part of the Curve (not the whole Curve).

**Start Bevel Factor and End Bevel Factor** These options determine where to start the Bevel operation on the Curve being beveled. Increasing the *Start Bevel Factor* to 0.5 will start beveling the Curve 50% of the distance from the start of the

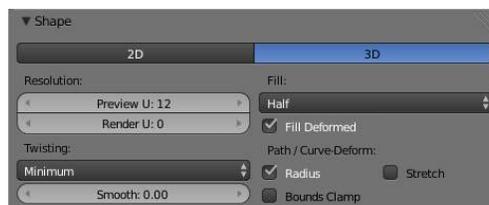


Fig. 2.489: Curves Shape panel.



Fig. 2.490: Curves with a resolution of 3 (left) and 12 (right).

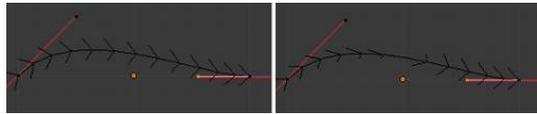


Fig. 2.491: Curves with a twist of minimum (left) and tangent (right).



Fig. 2.492: Curves with a fill of half (left) and full (right).

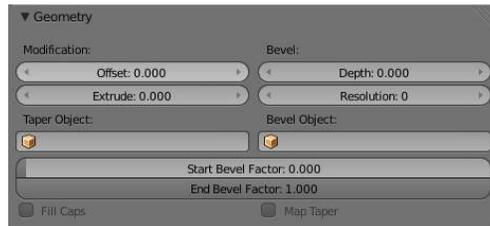


Fig. 2.493: Curves Geometry panel.

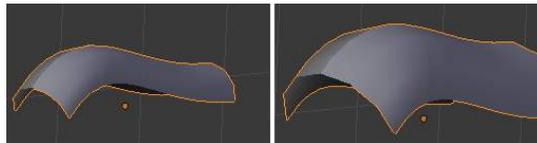


Fig. 2.494: A Curve with different Bevel depths applied.

**Resolution** Alters the smoothness of the bevel

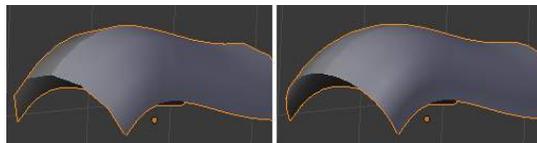


Fig. 2.495: A Curve with different resolutions applied.

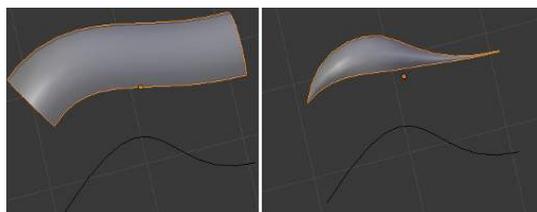


Fig. 2.496: A Curve before (left) and after (right) a Bezier Curve Taper Object was applied.

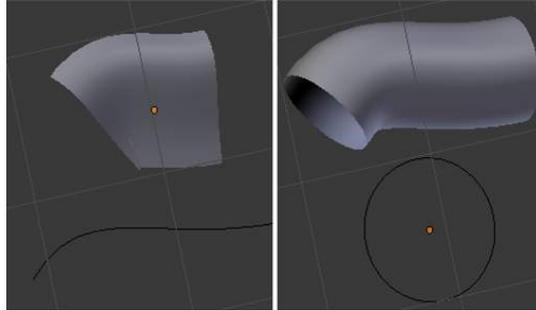


Fig. 2.497: A Curve with the Bevel Object as a Bezier Curve (left) and as a Bezier Circle (right).



Fig. 2.498: A Curve without (left) and with (right) Map Taper applied.

Curve (in effect shortening the Curve). Decreasing the *End Bevel Factor* by 0.25 will start beveling the Curve 25% of the distance from the end of the Curve (again, shortening the Curve).

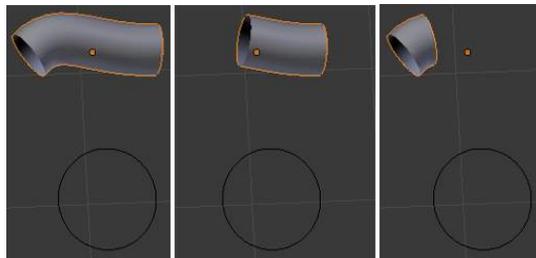


Fig. 2.499: A Curve with no Bevel factor applied (left), with a 50% Start Bevel Factor (middle) and with a 25% End Bevel Factor (right).

[Read more about Advanced Curve Editing](#)

**Path Animation** The Path Animation settings can be used to determine how Objects move along a certain path. See the link below for further information.

[Read more about utilizing Curves for paths during animation](#)

**Active Spline** The *Active Spline* panel becomes available during *Edit mode*.

**Cyclic** Closes the Curve.

**Resolution** Alters the smoothness of each segment by changing the number of subdivisions.

### Interpolation

**Tilt** Alters how the tilt of a segment is calculated.

**Radius** Alters how the radius of a Beveled Curve is calculated. The effects are easier to see after Shrinking/Fattening a control point `Alt-S`.

**Smooth** Smooths the normals of the Curve

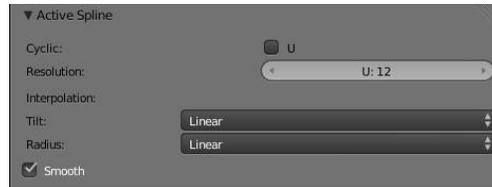


Fig. 2.500: Curves Active Spline panel.

### Non-Uniform Rational B-Splines (NURBS)

One of the major differences between Bezier Objects and NURBS Objects is that Bezier Curves are approximations. For example, a Bezier circle is an *approximation* of a circle, whereas a NURBS circle is an *exact* circle. NURBS theory can be a *very* complicated topic. For an introduction, please consult the [Wikipedia page](#). In practice, many of the Bezier curve operations discussed above apply to NURBS curves in the same manner. The following text will concentrate only on those aspects that are unique to NURBS curves.

**Editing NURBS Curves** A NURBS Curve is edited by moving the location of the Control Points.

- Place a Curve by `Shift-A` to bring up the Add menu, followed by `Curve` → `NURBS curve`.
- Press `Tab` to enter *Edit mode*.
- Select one of the Control Points and move it around. Use `LMB` to confirm the new location of the Control Point, or use `RMB` to cancel.
- If you want to add additional Control Points, select both of them, press `W` and select `Subdivide`. Press `F6` immediately after to determine how many subdivisions to make.

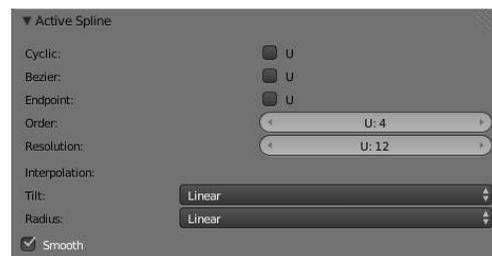


Fig. 2.501: NURBS Active Spline panel.

**Active Spline** One of the characteristics of a NURBS object is the *knot vector*. This is a sequence of numbers used to determine the influence of the control points on the curve. While you cannot edit the knot vectors directly, you can influence them through the *Endpoint* and *Bezier* options in the Active Spline panel. Note that the *Endpoint* and *Bezier* settings only apply to open NURBS curves.

**Cyclic** Makes the NURBS curve cyclic.



Fig. 2.502: A NURBS curve with Cyclic applied.

**Bezier** Makes the NURBS curve act like a Bezier curve.

**Endpoint** Makes the curve contact the end control points. Cyclic must be disabled for this option to work.

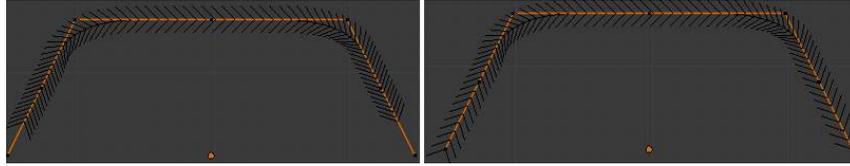


Fig. 2.503: A NURBS curve with Endpoint enabled.

**Order** The order of the NURBS curve determines the area of influence of the control points over the curve. Higher order values means that a single control point has a greater influence over a greater relative proportion of the curve. The valid range of *Order* values is 2-6 depending on the number of control points present in the curve.

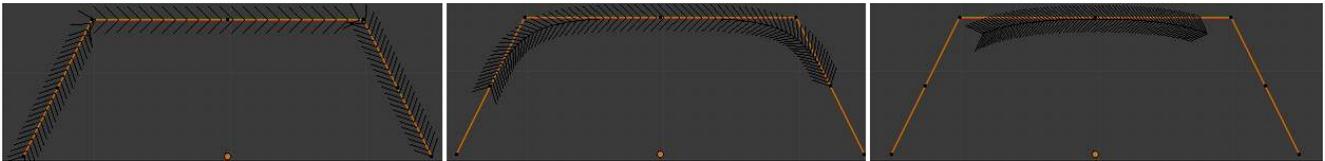


Fig. 2.504: NURBS curves with orders of 2 (left), 4 (middle) and 6 (right).

## Path

As mentioned above, Curves are often used as [paths](#). Any curve can be used as a Path if the *Path Animation* option is selected.

The Path option available from the *Add Curve* menu is identical to a 3D NURBS curve, except that you do not have access to the *Active Spline* panel.

## Curve Selection

Curve selection in *Edit* mode is much less complex than with meshes! Mainly this is because you have only one selectable element type, the control points (no select mode needed here...). These points are a bit more complex than simple vertices, however, especially for Béziers, as there is the central vertex, and its two handles...

The basic tools are the same as with [meshes](#), so you can select a simple control point with a LMB -click, add to current selection with Shift-LMB -clicks, B order-select, and so on.

One word about the Bézier control points: when you select the main central vertex, the two handles are automatically selected too, so you can grab it as a whole, without creating an angle in the curve. However, when you select a handle, only this vertex is selected, allowing you to modify this control vector...

L (or Ctrl-L) will add to the selection the cursor's nearest control point, and all the linked ones, i.e. all points belonging to the same curve. Note that for Bézier, using L with a handle selected will select the whole control point and all the linked ones.

## Select Menu

With curves, all “advanced” selection options are regrouped in the *Select* menu of the 3D views header. Let's detail them.

*Random... Inverse Select/Deselect All*

**Border Select** All these options have the same meaning and behavior as in [Object mode](#) (and the specifics of *Border Select* in *Edit* mode have already been discussed [here](#)).

## Every Nth

---

### Reference

Mode: *Edit* mode

Menu: *Select* → *Every Nth*

Hotkey: None

---

This only works if you already have at least one control point selected. Using the current selection, it will add to it every *n*th control point, before and after the initial selection. The “selection step” is specified in the *N* pop-up numeric field shown during the tool start.

## Select/Deselect First/Last

---

### Reference

Mode: *Edit* mode

Menu: *Select* → *Select/Deselect First*, *Select* → *Select/Deselect Last*

Hotkey: None

---

These commands will toggle the selection of the first or last control point(s) of the curve(s) in the object. This is useful to quickly find the start of a curve (e.g. when using it as path...).

## Select Next/Previous

---

### Reference

Mode: *Edit* mode

Menu: *Select* → *Select Next*, *Select* → *Select Previous*

Hotkey: None

---

These commands will select the next or previous control point(s), based on the current selection (i.e. the control points following or preceding the selected ones along the curve).

## More and Less

---

### Reference

Mode: *Edit* mode

---

Menu: *Select* → *More/Less*

Hotkey: `Ctrl-NumpadPlus` / `Ctrl-NumpadMinus`

These two options are complementary and similar to those for meshes. Their purpose, based on the currently selected control points, is to reduce or enlarge this selection.

The algorithm is the same as with meshes, but results are more easy to understand:

**More** for each selected control point, select **all** its linked points (i.e. one or two...).

**Less** for each selected control point, if **all** points linked to this point are selected, keep this one selected. Otherwise, de-select it.

This implies two points:

- First, when **all** control points of a curve are selected, nothing will happen (as for *Less*, all linked points are always selected, and of course, *More* can't add any). Conversely, the same goes when no control points are selected.
- Second, these tools will never “go outside” of a curve (they will never “jump” to another curve in the same object).

## Editing

### Curve Editing

This page covers the basics of curve editing. Curve basics, selecting and advanced editing are covered in the following pages:

- [Curve basics](#)
- [Curve Selecting](#)
- [Advanced Curve Editing](#)

### Curve Display

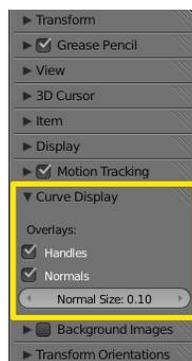


Fig. 2.505: Curve Display panel

**Display Options** When in Edit mode, the Properties Shelf (N) contains options in the *Curve Display* panel for how curves are displayed in the 3D viewport.

**Handles** Toggles the display of Bezier handles while in edit mode. This does not affect the appearance of the curve itself.

**Normals** Toggles the display of Curve Normals.

**Normal Size** Sets the display scale of curve normals.

**Hiding Elements** When in *Edit* mode, you can hide and reveal elements from the display. This can be useful in complex models with many elements on the Screen.

**Hide Selected elements** Use **H**, or the *Curve* → *Show/Hide* → *Hide Selected* menu option from the 3D window header.

**Show Hidden elements** Use **Alt-H**, or the *Curve* → *Show/Hide* → *Show Hidden* menu option from the 3D window header.

**Hide Unselected elements** Use **Shift-H**, or the *Curve* → *Show/Hide* → *Hide Unselected* menu option from the 3D window header.

---

## Basic Curve Editing (translation, rotation, scale)

### Reference

Mode: *Edit* mode

Menu: *Curve* → *Transform* → *Grab/Move, Rotate, Scale, ...*

Hotkey: **G / R / S**

---

Like other elements in Blender, Curve control points can be grabbed/moved (**G**), rotated (**R**) or scaled (**S**) as described in the [Basic Transformations](#) section. When in *Edit* mode, [proportional editing](#) is also available for transformation actions.

## Snapping

### Reference

Mode: *Edit* mode

Panel: *Curve Tools* (*Editing* context)

---

[Mesh snapping](#) also works with curve components. Both control points and their handles will be affected by snapping, except for within itself (other components of the active curve). Snapping works with 2D curves but points will be constrained to the local XY axes.

## Deforming Tools

### Reference

Mode: *Edit* mode

Menu: *Curve* → *Transform*

---

The *To Sphere*, *Shear*, *Wrap* and *Push/Pull* transform tools are described in the [Transformations](#) sections. The two other tools, *Tilt* and *Shrink/Fatten Radius* are related to [Curve Extrusion](#).

---

## Smoothing

---

### Reference

Mode: *Edit* mode

Hotkey: `[W][ ]` → *smooth*

---

Curve smoothing is available through the specials menu. For Bézier curves, this smoothing operation currently only smooths the positions of control points and not their tangents. End points are also constrained when smoothing.

## Mirror

---

### Reference

Mode: *Edit* mode

Menu: *Curve* → *Mirror*

Hotkey: `Ctrl-M`

---

The *Mirror* tool is also available, behaving exactly as with [mesh vertices](#),

---

## Set Bézier Handle Type

---

### Reference

Mode: *Edit* mode

Panel: *Curve Tools* → *Handles*

Menu: *Curve* → *Control Points* → *Set Handle Type*

Hotkey: `∇`

---

Handle types are a property of [Bézier curves](#). and can be used to alter features of the curve. For example, switching to *Vector handles* can be used to create curves with sharp corners. Read the [Bézier curves](#) page for more details.

## Extending Curves

---

### Reference

Mode: *Edit* mode

Menu: *Curve* → *Extrude*

Hotkey: `Ctrl-LMB` or `E`

---

Once a curve is created you can add new segments (in fact, new control points defining new segments), either by extruding, or placing new handles with `Ctrl-LMB` clicks. Each new segment is added to one end of the curve. A new segment will only be added if a single vertex, or handle, at one end of the curve is selected. If two or more control points are selected, a new Bézier closed curve is started.

---

## Subdivision Reference

Mode: *Edit* mode

Panel: *Curve Tools* (*Editing* context)

Menu: *Curve* → *Segments* → *Subdivide*

Hotkey:  $\bar{w}$

---

Curve subdivision simply subdivides all selected segments by adding one or more control points between the selected segments. To control the number of cuts, press  $\bar{w}$  to make a single subdivision. Then press  $F6$  to bring up the *Number of Cuts* menu.

---

## Duplication Reference

Mode: *Edit* mode

Menu: *Curve* → *Duplicate*

Hotkey:  $\text{Shift-D}$

---

This command duplicates the selected control points, along with the curve segments implicitly selected (if any). The copy is selected and placed in *Grab* mode, so you can move it to another place.

---

## Joining Curve Segments Reference

Mode: *Edit* mode

Menu: *Curve* → *Make Segment*

Hotkey:  $F$

---

Two open curves can be combined into one by creating a segment between the two curves. To join two separated curves, select one end control point from each curve then press  $F$ . The two curves are joined by a segment to become a single curve.

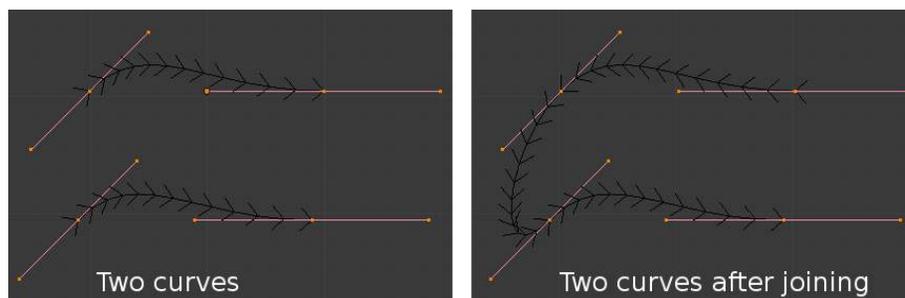


Fig. 2.506: Curves before and after joining

Additionally, you can close a curve by joining the endpoints but note that you can only join curves of the same type (i.e. Bézier with Bézier, NURBS with NURBS)

---

### Separating Curves

#### Reference

Mode: *Edit mode*

Menu: *Curve → Separate*

Hotkey: *P*

---

Curve objects that are made of multiple distinct curves can be separated into their own objects by selecting the desired segments and pressing *P*. Note, if there is only one curve in a Curve object, pressing *P* will create a new Curve object with no control points.

---

### Deleting Elements

#### Reference

Mode: *Edit mode*

Menu: *Curve → Delete...*

Hotkey: *X* or *Delete*

---

The *Erase* pop-up menu of curves offers you three options:

**Selected** This will delete the selected control points, *without* breaking the curve (i.e. the adjacent points will be directly linked, joined, once the intermediary ones are deleted). Remember that NURBS order cannot be higher than its number of control points, so it might decrease when you delete some control point. Of course, when only one point remains, there is no more visible curve, and when all points are deleted, the curve itself is deleted.

**Segment** This option is somewhat the opposite to the preceding one, as it will cut the curve, without removing any control points, by erasing one selected segment. This option always removes *only one segment* (the last “selected” one), even when several are in the selection. So to delete all segments in your selection, you’ll have to repetitively use the same erase option...

**All** As with meshes, this deletes everything in the object!




---

### Opening and Closing a Curve

#### Reference

Mode: *Edit mode*

Menu: *Curve → Toggle Cyclic*

Hotkey: *Alt-C*

---

This toggles between an open curve and closed curve (Cyclic). Only curves with at least one selected control point will be closed/open. The shape of the closing segment is based on the start and end handles for Bézier curves, and as usual on adjacent

control points for NURBS. The only time a handle is adjusted after closing is if the handle is an *Auto* one. (*Open curve*) and (*Closed curve*) is the same Bézier curve open and closed.

This action only works on the original starting control-point or the last control-point added. Deleting a segment(s) doesn't change how the action applies; it still operates only on the starting and last control-points. This means that `Alt-C` may actually join two curves instead of closing a single curve! Remember that when a 2D curve is closed, it creates a renderable flat face.

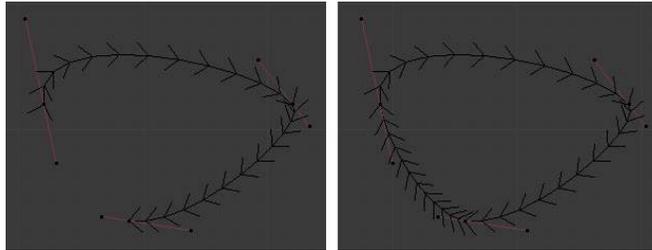


Fig. 2.511: Open and Closed curves.

---

## Switch Direction Reference

Mode: *Edit* mode

Menu: *Curve* → *Segments* → *Switch Direction*, *Specials* → *Switch Direction*

Hotkey: `[W]` → `[pad2]`

---

This command will “reverse” the direction of any curve with at least one selected element (i. e. the start point will become the end one, and *vice versa*). This is mainly useful when using a curve as path, or using the bevel and taper options.

---

## Converting Tools

---

### Converting Curve Type Reference

Mode: *Edit* mode

Panel: *Curve Tools* → *Set Spline type*

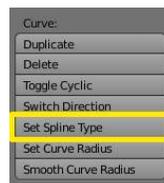


Fig. 2.512: Set Spline Type button

You can convert splines in a curve object between Bézier, NURBS, and Poly curves. Press **T** to bring up the Toolshelf. Clicking on the *Set Spline Type* button will allow you to select the Spline type (Poly, Bézier or NURBS).

Note, this is not a “smart” conversion, i.e. Blender does not try to keep the same shape, nor the same number of control points. For example, when converting a NURBS to a Bézier, each group of three NURBS control points become a unique Bézier one (center point and two handles).

---

### Convert Curve to Mesh

#### Reference

Mode: *Object* mode

Menu: *Object* → *Convert to*

Hotkey: **Alt-C**

---

There is also an “external” conversion, from curve to mesh, that only works in *Object* mode. It transforms a *Curve* object in a *Mesh* one, using the curve resolution to create edges and vertices. Note that it also keeps the faces and volumes created by closed and extruded curves.

---

### Convert Mesh to Curve

#### Reference

Mode: *Object* mode

Menu: *Object* → *Convert to*

Hotkey: **Alt-C**

---

Mesh objects that consist of a series of connected vertices can be converted into curve objects. The resulting curve will be a Poly curve type, but can be converted to have smooth segments as described above.

---

### Curve Parenting

#### Reference

Mode: *Edit* mode

Hotkey: **Ctrl-P**

---

You can make other selected objects children of one or three control points **Ctrl-P**, as with mesh objects.

Select either 1 or 3 control points, then **Ctrl-RMB** another object and use **Ctrl-P** to make a vertex parent.

---

### Hooks

#### Reference

Mode: *Edit* mode

Menu: *Curve* → *control points* → *hooks*

Hotkey: **Ctrl-H**

---

[Hooks](#) can be added to control one or more points with other objects.

### Set Goal Weight Reference

Mode: *Edit* mode

Menu: *W* → *Set Goal Weight*

**Set Goal Weight** This sets the “goal weight” of selected control points, which is used when a curve has Soft Body physics, forcing the curve to “stick” to their original positions, based on the weight.

### Advanced

**Curve Deform** *Curve Deform* provides a simple but efficient method of defining a deformation on a mesh. By parenting a mesh object to a curve, you can deform the mesh up or down the curve by moving the mesh along, or orthogonal to, the dominant axis. This is a most useful tool to make an object follow a complex path, like e.g. a sheet of paper inside a printer, a film inside a camera, the water of a canal...

The *Curve Deform* works on a (global) dominant axis, X, Y, or Z. This means that when you move your mesh in the dominant direction, the mesh will traverse along the curve. Moving the mesh in an orthogonal direction will move the mesh object closer or further away from the curve. The default settings in Blender map the Y axis to the dominant axis. When you move the object beyond the curve endings the object will continue to deform based on the direction vector of the curve endings.

If the “curve path” is 3D, the *Tilt* value of its control points will be used (see the *Extrusion* section above) to twist the “curved” object around it. Unfortunately, the other *Radius* property is not used (it would have been possible, for example, to make it control the size of the “curved” object...).

**Tip:** Try to position your object over the curve immediately after you have added it, before adding the curve deform. This gives the best control over how the deformation works.

**Note:** Use modifiers!

The *Curve Deform* relationship is now also a modifier, called [Curve](#). The *Curve* modifier function acts the same as its counterpart, except that when the modifier is used, the “dominant axis” is set inside its properties - and the *Track X / Y / Z* buttons no longer have an effect on it. And you have some goodies, like the possibility, if “curving” a mesh, to only curve one of its vertex groups...

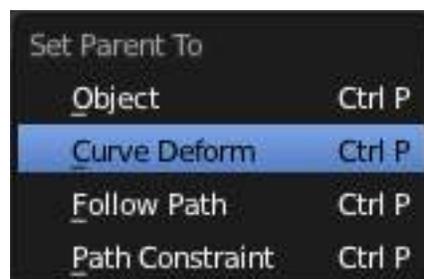


Fig. 2.513: Make Parent menu.

**Interface** When parenting an object (mesh, curve, meta, ...) to a curve (Ctrl-P), you will be presented with a menu (*Make Parent menu*).

By selecting *Curve Deform*, you enable the curve deform function on the mesh object.

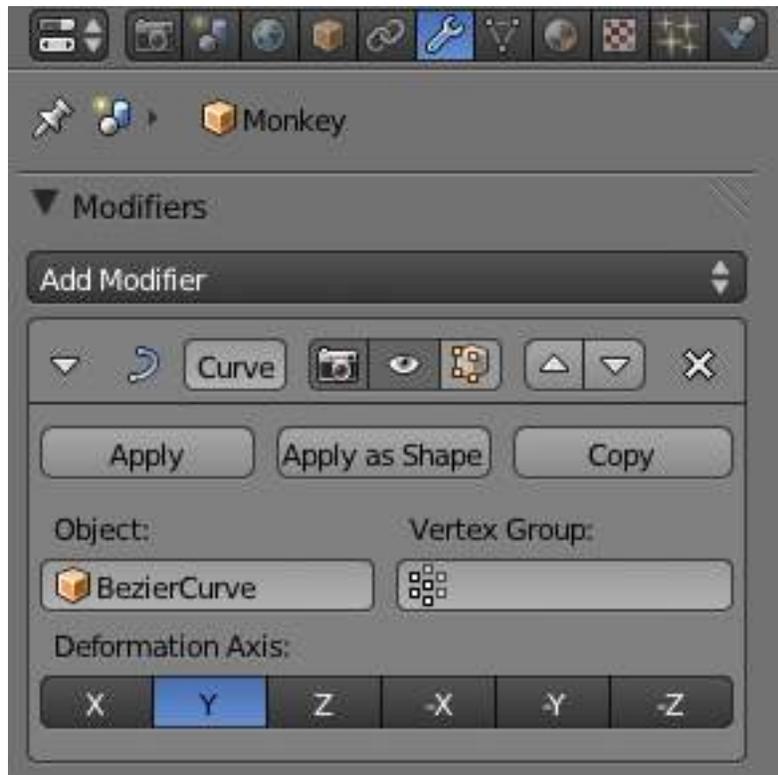


Fig. 2.514: Anim settings panel.

The dominant axis setting is set on the mesh object. By default the dominant axis in Blender is *Y*. This can be changed by selecting one of the *Track X*, *Y* or *Z* buttons in the *Anim Panel*, (*Anim settings panel*), in *Object* context.

Cyclic (or closed) curves work as expected where the object deformations traverse along the path in cycles. Note however that when you have more than one curve in the “parent” object, its “children” will only follow the first one.

The *Stretch* curve option allows you to let the mesh object stretch, or squeeze, over the entire curve. This option is in *Object Data* properties, for the “parent” curve. See (*Curve and Surface panel*).

**Example** Let’s make a simple example:

- Remove default cube object from scene and add a Monkey (*Add* → *Mesh* → *Monkey*, *Add a Monkey!*)!
- Press *Tab* to exit *Edit* mode.
- Now add a curve (*Add* → *Curve* → *Bezier Curve*, *Add a Curve*).
- While in *Edit* mode, move the control points of the curve as shown in (*Edit Curve*), then exit *Edit* mode (*Tab*).
- Now, you can use the new, modern, modifier way of “curving” the Monkey:
  - Select the Monkey (*RMB*).
  - In the *Object Modifiers* properties, *Modifiers* panel, add a *Curve* modifier.

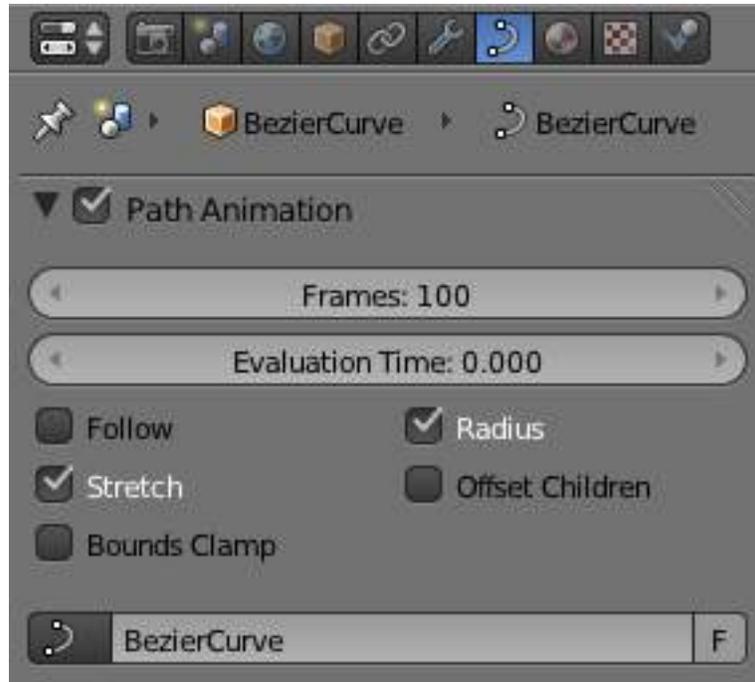


Fig. 2.515: Curve and Surface panel.

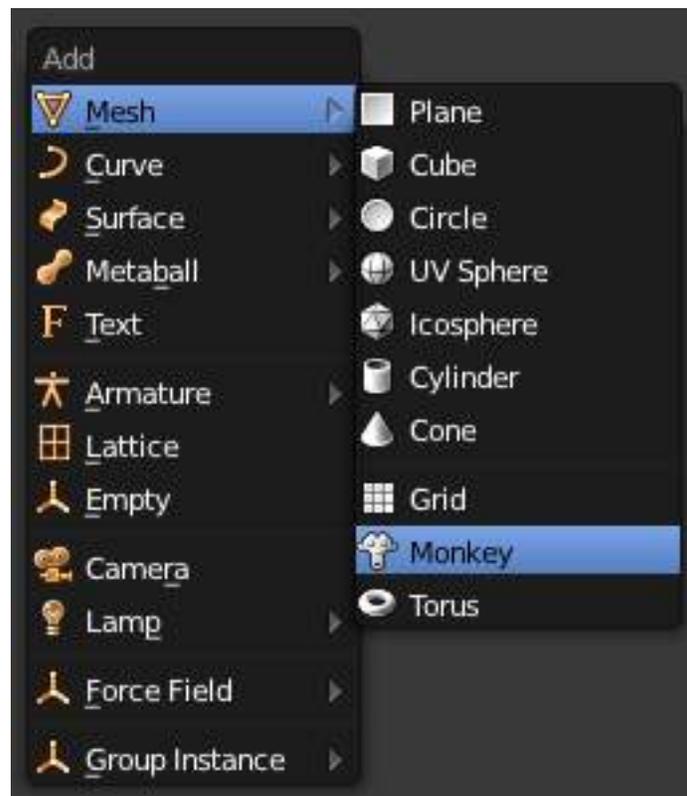


Fig. 2.516: Add a Monkey!

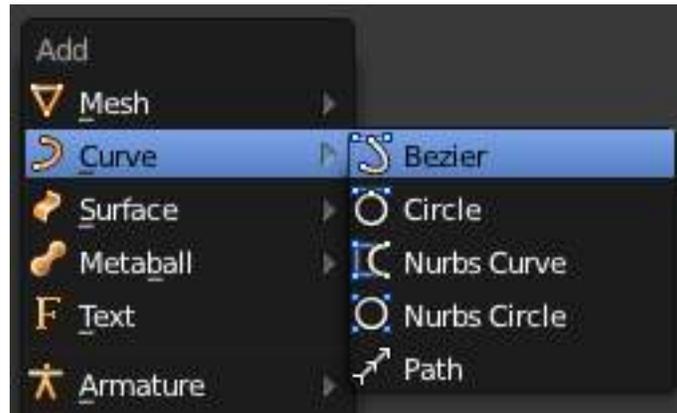


Fig. 2.517: Add a Curve.

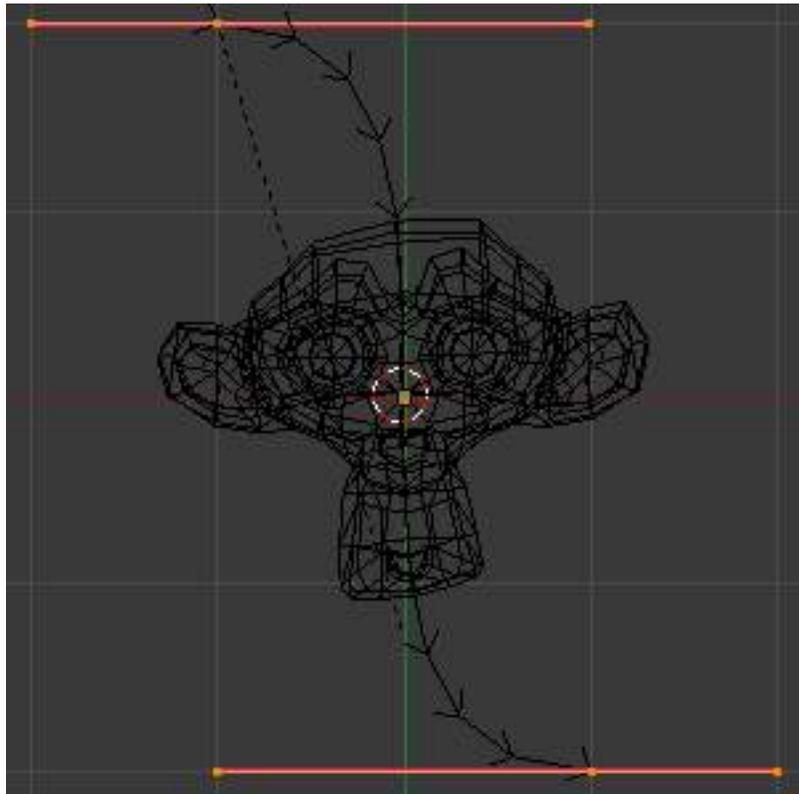


Fig. 2.518: Edit Curve.

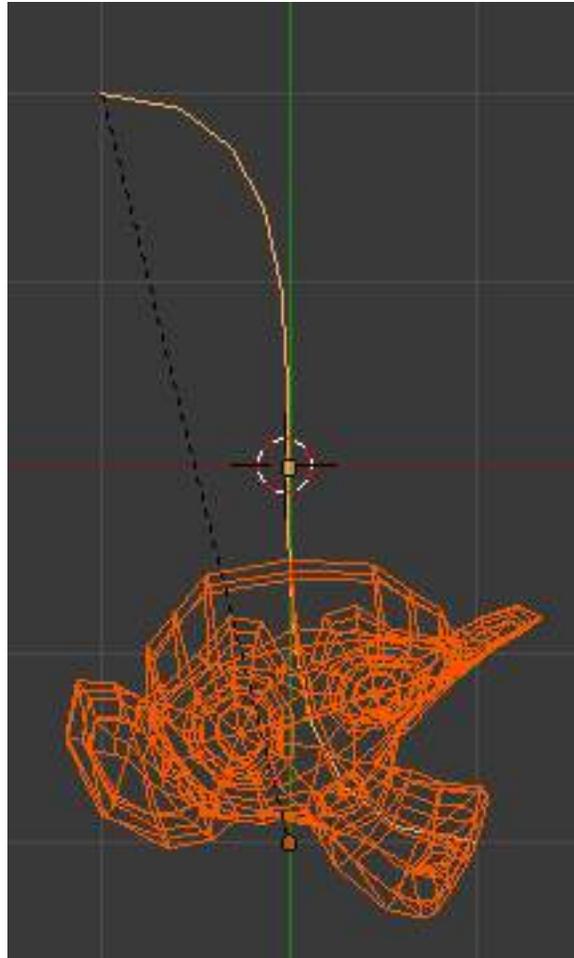


Fig. 2.519: Monkey on a Curve.

- Type the name of the curve (should be `Curve`) in the *Ob* field of the modifier, and optionally change the dominant axis to *Y*.
- Or you can choose the old, deprecated method (note that it creates a “virtual” modifier...):
  - Select the Monkey (RMB), and then shift select the curve (Shift-RMB).
  - Press `Ctrl-P` to open up the *Make Parent* menu.
  - Select *Curve Deform (Make Parent menu)*.
- The Monkey should be positioned on the curve, as in (*Monkey on a Curve*).
- Now if you select the Monkey (RMB), and move it (G), in the *Y*-direction (the dominant axis by default), the monkey will deform nicely along the curve.

---

**Tip:** If you press `MMB` (or one of the *X / Y / Z* keys) while moving the Monkey you will constrain the movement to one axis only.

---

- In (*Monkey deformations*), you can see the Monkey at different positions along the curve. To get a cleaner view over the deformation I have activated *SubSurf* with *Subdiv* to **2**, and *Set Smooth* on the Monkey mesh.

---

**Tip:** Moving the Monkey in directions other than the dominant axis will create some odd deformations. Sometimes this is what you want to achieve, so you’ll need to experiment and try it out!

---



Fig. 2.520: Monkey deformations.

**Curve Extrusion** This section covers methods for extruding curves, or giving them thickness, and how to control the thickness along the path.

---

## Extrusion Reference

Mode: *Object* or *Edit* mode

Panel: *Curve and Surface*

---

Extrusion can be especially with the bevel/taper/Tilt/Radius options. Note that this isn't related to *Extrude* used in mesh edit-mode.

We will see the different settings, depending on their scope of action:

**Width** This controls the position of the extruded “border” of the curve, relative to the curve itself. With closed 2D curves (see below), it is quite simple to understand - with a *Width* greater than **1.0**, the extruded volume is wider, with a *Width* of **1.0**, the border tightly follows the curve, and with a *Width* lower than **1.0**, the volume is narrower? The same principle remains for open 2D and 3D curves, but the way the “outside” and “inside” of the curve is determined seems a bit odd?

It has the same effect with extruded “bevel” objects...

**Tilt** This setting - unfortunately, you can never see its value anywhere in Blender - controls the “twisting angle” around the curve for each point - so it is only relevant with 3D curves! You set it using the *Tilt* transform tool (T, or *Curve* → *Transform* → *Tilt*), and you can reset it to its default value (i.e. perpendicular to the original curve plane) with **Alt+T** (or *Curve* → *Control Points* → *Clear Tilt*). With NURBS, the tilt is always smoothly interpolated. However, with Bézier, you can choose the interpolation algorithm to use in the *Tilt Interpolation* drop-down list of the *Curve Tools* panel (you will find the classical *Linear*, *Cardinal*, *B Spline* and *Ease* options...).

**Simple Extrusion** Let's first see the “simple” extrusion of curves, without additional bevel/taper objects.

**Extrude** This controls the width (or height) of the extrusion. The real size is of course dependent on the scale of the underlying object, but with a scale of one, an *Extrusion* of **1.0** will extrude the curve one BU in both directions, along the axis perpendicular to the curve's plane (see below for specifics of 3D curves?).

If set to **0.0**, there is no “simple” extrusion!

**Bevel Depth** This will add a bevel to the extrusion. See below for its effects... Note that the bevel makes the extrusion wider and higher. If set to **0.0**, there is no bevel (max value: **2.0**).

**Bev Resol** Controls the resolution of the bevel created by a *Bevel Depth* higher than zero. If set the **0** (the default), the bevel is a simple “flat” surface. Higher values will smooth, round off the bevel, similar to the resolution settings of the curve itself...

We have three sub-classes of results, depending on whether the curve is open or closed or 3D:

**Open 2D Curve** The extrusion will create a “wall” or “ribbon” following the curve shape. If using a *Bevel Depth*, the wall becomes a sort of slide or gutter. Note the direction of this bevel is sometimes strange and unpredictable, often the reverse of what you would get with the same curve closed? You can inverse this direction by switching the direction of the curve.

This allows you, e.g., to quickly simulate a marble rolling down a complex slide, by combining an extruded beveled curve, and a sphere with a *Follow Path* constraint set against this curve?

**Closed 2D Curve** This is probably the most useful situation, as it will quickly create a volume, with (by default) two flat and parallel surfaces filling the two sides of the extruded “wall”. You can remove one or both of these faces by disabling the *Back* and/or *Front* toggle buttons next to the *3D* one.

The optional bevel will always be “right-oriented” here, allowing you to smooth out the “edges” of the volume.

**3D Curve** Here the fact that the curve is closed or not has no importance - you will never get a volume with an extruded 3D curve, only a wall or ribbon, like with open 2D curves.

However, there is one more feature with 3D curves: the *Tilt* of the control points (see above). It will make the ribbon twist around the curve ? to create a M?bius strip, for example!

**Advanced Extrusion** These extrusions use one or two additional curve objects, to create very complex organic shapes.

To enable this type of extrusion, you have to type a valid curve object name in the *BevOb* field of the curve you are going to use as the “spinal column” of your extrusion. The “bevel” curve will control the cross section of the extruded object. Whether the *BevOb* curve is 2D or 3D has no importance, but if it is closed, it will create a “tube-like” extrusion; otherwise you will get a sort of gutter or slide object...

The object is extruded along the whole length of all internal curves. By default, the width of the extrusion is constant, but you have two ways to control it, the *Radius* property of control points, and the “taper” object.

The *Radius* of the points is set using the *Shrink/Fatten Radius* transform tool (`Alt-S`, or *Curve* → *Transform* → *Shrink/Fatten Radius*), or with the *Set Radius* entry in the *Specials* menu (`W`). Here again, you unfortunately cannot visualize anywhere the *Radius* of a given control point...

The *Radius* allows you to directly control the width of the extrusion along the “spinal” curve. As for *Tilt* (see above), you can choose the interpolation algorithm used for Bézier curves, in the *Radius Interpolation* drop-down list of the *Curve Tools* panel.

But you have another, more precise option: the “taper” object. As for the “bevel” one, you set its name in the *TaperOb* field of the main curve - it must be an *open curve*. The taper curve is evaluated along *the local X axis*, using *the local Y axis* for width control. Note also that:

- The taper is applied independently to all curves of the extruded object.
- Only the first curve in a *TaperOb* is evaluated, even if you have several separated segments.
- The scaling starts at the first control-point on the left and moves along the curve to the last control-point on the right.
- Negative scaling, (negative local Y on the taper curve) is possible as well. However, rendering artifacts may appear.
- It scales the width of normal extrusions based on evaluating the taper curve, which means sharp corners on the taper curve will not be easily visible. You’ll have to heavily level up the resolution (*DefResolu*) of the base curve.
- With closed curves, the taper curve in *TaperOb* acts along the whole curve (perimeter of the object), not just the length of the object, and varies the extrusion depth. In these cases, you want the relative height of the *TaperOb* Taper curve at both ends to be the same, so that the cyclic point (the place where the endpoint of the curve connects to the beginning) is a smooth transition.

Last but not least, with 3D “spinal” curves, the *Tilt* of the control points can control the twisting of the extruded “bevel” along the curve!

**Examples** Let’s taper a simple curve circle extruded object using a taper curve. Add a curve, then exit *Edit* mode. Add another one (a closed one, like a circle); call it *BevelCurve*, and enter its name in the *BevOb* field of the first curve (*Editing* context *Curve and Surface* panel). We now have a pipe. Add a third curve while in *Object* mode and call it *TaperCurve*. Adjust the left control-point by raising it up about 5 units.

Now return to the *Editing* context, and edit the first curve’s *TaperOb* field in the *Curve and Surface* panel to reference the new taper curve which we called *TaperCurve*. When you hit enter the taper curve is applied immediately, with the results shown in (*Taper extruded curve*).



You can see the **taper curve** being applied to the **extruded object**. Notice how the pipe’s volume shrinks to nothing as the taper curve goes from left to right. If the taper curve went below the local Y axis the pipe’s inside would become the outside, which would lead to rendering artifacts. Of course as an artist that may be what you are looking for!

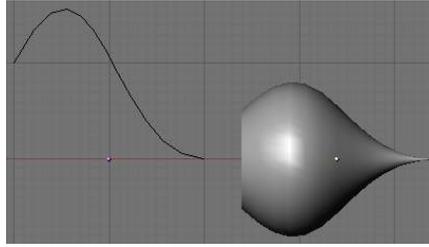


Fig. 2.525: Taper example 1.

In (*Taper example 1*) you can clearly see the effect the left taper curve has on the right curve object. Here the left taper curve is closer to the object center and that results in a smaller curve object to the right.

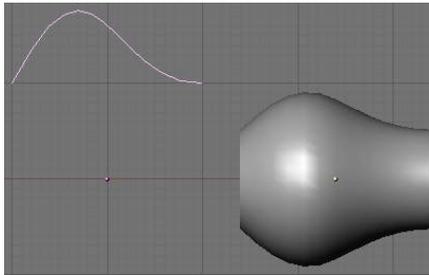


Fig. 2.526: Taper example 2.

In (*Taper example 2*) a control point in the taper curve to the left is moved away from the center and that gives a wider result to the curve object on the right.

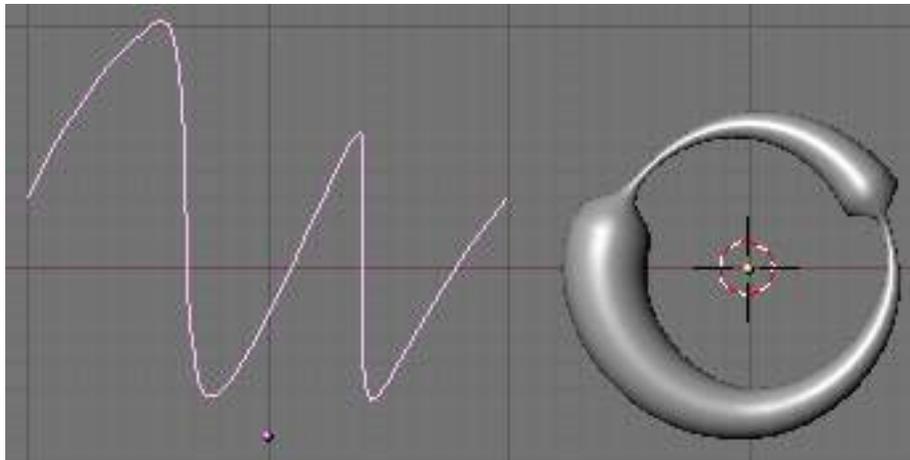


Fig. 2.527: Taper example 3.

In (*Taper example 3*), we see the use of a more irregular taper curve applied to a curve circle.

TODO: add some “bevel” extrusion with *Tilt* examples.

## 2.3.5 Surfaces

### Surfaces

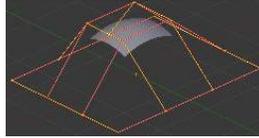


Fig. 2.528: Surface.

Curves are 2D objects, and surfaces are their 3D extension. Note however that in Blender, you only have NURBS surfaces, no Bézier (you have the *Bezier* knot type, though; see below), nor polygonal (but for these, you have meshes!). Even though curves and surfaces share the same object type (with texts also...), they are not the same thing; for example, you cannot have in the same object both curves and surfaces.

As surfaces are 2D, they have two interpolation axes, U (as for curves) and V. It is important to understand that *you can control the interpolation rules (knot, order, resolution) independently for each of these two dimensions* (the U and V fields for all these settings, of course).

You may ask yourself “but the surface appears to be 3D, why is it only 2D?”. In order to be 3D, the object needs to have “Volume,” and a surface, even when it is closed, doesn’t have volume; it is infinitely thin. If it had a volume the surface would have a thickness (its third dimension). Hence, it’s only a 2D object, and has only two interpolation dimensions or axes or coordinates (if you know a bit of math, think of non-euclidean geometry - well, surfaces are just non-euclidean 2D planes...). To take a more “real life” example, you can roll a sheet of paper to create a cylinder; well, even if it “draws” a volume, the sheet itself will remain a (nearly...) 2D object!

In fact, surfaces are very similar to the results you get when [extruding a curve](#) (by the way, I think it should be possible to convert an extruded curve to a surface, at least when only made of NURBS - but Blender cannot do it currently...).

### Finding Surface Tools



Fig. 2.529: Surface Tools.

The panels of the *Editing* context are the same as for [curves](#), just with fewer options... And as usual, you have the *Select* and *Surface* menus in the 3D view headers, and the *Specials* (W) pop-up one.

### Visualization

There is nearly no difference from NURBS curves, except that the U direction is indicated by yellow grid lines, and the V one is materialized by pink grid lines, as you can see in (*Surface*).

You can hide and reveal control points just as with curves, and you have the same draw options in the `Curve Tools` panel.

## Surface Structure

Many of the concepts from [curves](#), especially NURBS ones, carry directly over to NURBS surfaces, such as control points, *Order*, *Weight*, *Resolution*, etc. Here we will just talk about the differences.

It is very important to understand the difference between NURBS curves and NURBS surfaces: the first one has one dimension, the latter has two. Blender internally treats NURBS surfaces and NURBS curves completely differently. There are several attributes that separate them but the most important is that a NURBS curve has a single interpolation axis (U) and a NURBS surface has two interpolation axes (U and V).

However, you can have “2D” surfaces made of curves (using the [extrusion tools](#), or, to a lesser extent, the filling of closed 2D curves. And you can have “1D” curves made of surfaces, like a NURBS surface with only one row (either in U or V direction) of control points produces only a curve...

Visually you can tell which is which by entering *Edit* mode and looking at the 3D window’s header: either the header shows *Surface* or *Curve* as one of the menu choices. Also, you can [extrude](#) a whole NURBS surface curve to create a surface, but you can’t with a simple NURBS curve (we talk here about the “standard” *Extrude* tool, the one activated with the *E* shortcut, not the quite-specific curve extrusion tools - yes, I know, it’s not easy to follow...).

**Control Points, Rows and Grid** Control points for NURBS surfaces are the same as for NURBS curves. However, their layout is quite constraining. The concept of “segment” disappears, replaced by “rows” and the overall “grid”.

A “row” is a set of control points forming one “line” in one interpolation direction (a bit similar to `edge loops` for meshes). So you have “U-rows” and “V-rows” in a NURBS surface. The key point is that *all rows of a given type (U or V) have the same number of control points*. Each control point belongs to exactly one U-row and one V-row.

All this forms a “grid”, or “cage”, the shape of which controls the shape of the NURBS surface. A bit like a [lattice](#) ...

This is very important to grasp: you cannot add a single control point to a NURBS surface; you have to add a whole U- or V-row at once (in practice, you will usually use the *Extrude* tool, or perhaps the *Duplicate* one, to add those...), containing exactly the same number of points as the others. This also means that you will only be able to “merge” different pieces of surfaces if at least one of their rows match together.

**Surface Resolution** Just like NURBS [curves](#), *Resolution* controls the detail of the surface. The higher the *Resolution* the more detailed and smoother the surface is. The lower the *Resolution* the rougher the surface. However, here you have two resolution settings, one for each interpolation axis (U and V). Note that unlike with curves, you have only one resolution (the *Resol U* and *V* fields, in the *Curve Tools* panel)...



(*Resolution 1x1*) is an example of a surface resolution of 3 for both U and V. (*Resolution 3x3 surface*) is an example of a surface resolution of 12 for both U and V.



Fig. 2.534: Resolution panel.

You can adjust the resolution separately for both preview and render, to not slow things down in the viewport, but still get good render results.

**Closed and Open Surfaces** Like curves, surfaces can be closed (cyclical) or open, independently in both directions, allowing you to easily create a tube, donut or sphere shape, and they can be drawn as “solids” in *Edit* mode. This makes working with surfaces quite easy.

**Knots** Just like with NURBS curves, NURBS surfaces have two knot vectors, one for each U and V axis. Here again, they can be one of *Uniform*, *Endpoint*, or *Bezier*, with the same properties as for curves. And as with curves, only open surfaces (in the relevant direction) are affected by this setting...

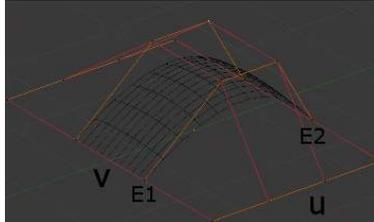


Fig. 2.535: Endpoint U.

In (*Endpoint U*), the U interpolation axis is labeled as U and the V interpolation axis is labeled as V. The U ‘s interpolation axis has been set to *Endpoint* and as such the surface now extends to the outer edges from E1 to E2 along the U interpolation axis.

To cause the surface to extend to all edges you would set the V ‘s axis to *Endpoint* as well.

**Order** One more time, this property is the same as with NURBS Curves; it specifies how much the control points are taken into account for calculating the curve of the surface shape. For high *Orders*, (1), the surface pulls away from the control points, creating a smoother surface - assuming that the *Surface Resolution* is high enough. For lowest *Orders*, (2), the surface follows the control points, creating a surface that tends to follow the grid cage.

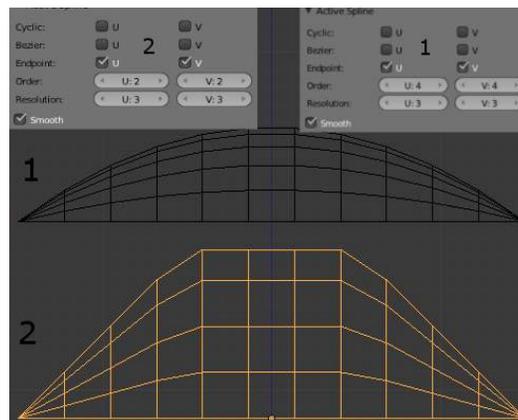


Fig. 2.536: Order 2 and order 4 surface.

For illustration purposes, in both (*Order 4 surface*) and (*Order 2 surface*), the knot vectors were set to *Endpoint*, causing the surface to extend to all edges.

You can set independently the order for each interpolation axis, and like curves, it cannot be lower than 2, and higher than 6 or the number of control points on the relevant axis.

**Weight** Guess what? Yes, it works exactly like NURBS Curves ! *Weight* specifies how much each control point “pulls” on the curve.

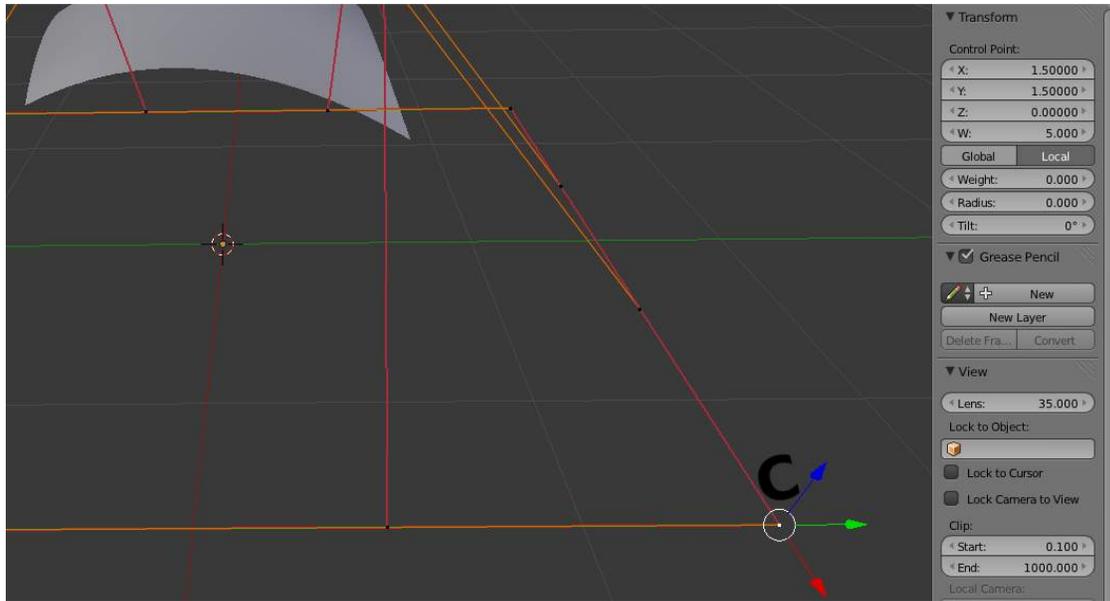


Fig. 2.537: Surface Weight 5.

In (*Surface Weight 5*), a single control point, labeled C, has had its *Weight* set to **5.0** while all others are at their default of **1.0**. As you can see, that control point *pulls* the surface towards it.

If all the control points have the same *Weight* then each effectively cancels each other out. It is the difference in the weights that cause the surface to move towards or away from a control point.

The *Weight* of any particular control point is visible in the Transform Properties panel (N), *in the W field* (and not the *Weight* field...).

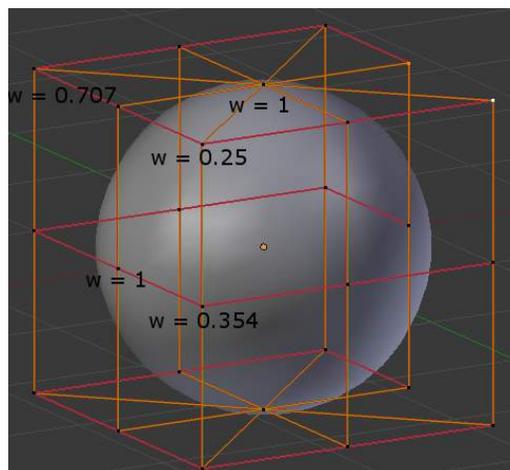


Fig. 2.538: A sphere surface.

**Preset Weights** NURBS can create pure shapes such as circles, cylinders, and spheres (note that a Bézier circle is not a pure circle). To create pure circles, globes, or cylinders, you must set to specific values the weights of the control points - some of which are provided as presets in the *Curve Tools* panel (lower right corner). This is not intuitive, and you should read more on NURBS before trying this.

We saw with 1D NURBS curves how to create a circle; let's see how to create a sphere with 2D surfaces. It is the same principle - you'll note that the four different weights needed for creating a sphere (**1.0**, **0.707** =  $\sqrt{0.5}$ , **0.354** =  $\sqrt{2}/4$ , and **0.25**) are the four presets available in the `Curve Tools` panel...

**Primitives** To help get started in creating surfaces there are four preset NURBS surfaces, found in the `Add` → `Surface` menu: *NURBS Surface*, *NURBS Tube*, *NURBS Sphere* and *NURBS Torus*.

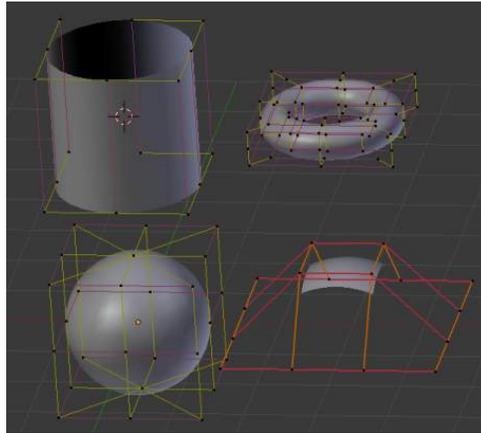


Fig. 2.539: NURBS surface primitives.

There are also two preset NURBS surface curves (with only one control point on each V-row): *NURBS Curve* and *NURBS Circle*.

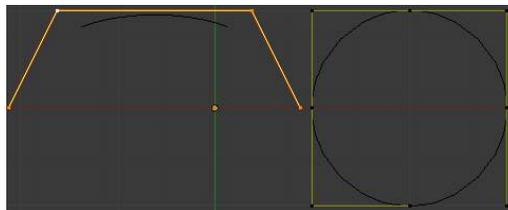


Fig. 2.540: NURBS curve primitives.

Note how a circle NURBS surface is never filled, unlike its “real” curve counterpart...

## Surface Selection

Surface selection in *Edit* mode is very similar to [NURBS curve selection](#). The basic tools are the same as with [meshes](#), so you can select a simple control point with a `LMB` -click, add to current selection with `Shift-LMB` -clicks, `B` order-select, and so on.

`L` (or `Ctrl-L`) will add to the selection the mouse cursor's nearest control point, and all the linked ones, i.e. all points belonging to the same surface.

## Select Menu

The *Select* menu (3D view headers) is even simpler than for curves...

All these options have the same meaning and behavior as in *Object mode* (and the specificities of *Border Select* in *Edit mode* have already been discussed [here](#)).

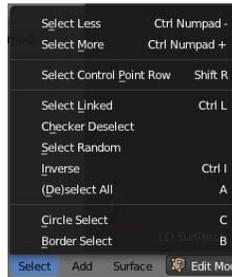


Fig. 2.541: frame[left].

---

### Every Nth Reference

Mode: *Edit mode*

Menu: *Select* → *Every Nth*

Hotkey: None

---

This is the same option as for *curve selection*. However, the behavior of the *N* (“selection step”) parameter in the 2D of a NURBS surface “cage” seems quite difficult to understand...

---

### Control Point Row Reference

Mode: *Edit mode*

Menu: *Select* → *Control Point Row*

Hotkey: Shift-R

---

This option works a bit like *edge loop selection* for meshes, inasmuch it selects a whole *row* of control points, based on the active (the last selected) one. The first time you press Shift-R, the V-row passing through (containing) the active point will be *added to the current selection*. If you use again this shortcut, you will toggle between the U- and V-row of this point, *removing everything else from the selection*.

---

### More and Less Reference

Mode: *Edit mode*

Menu: *Select* → *More/Less*

Hotkey: Ctrl-NumpadPlus / Ctrl-NumpadMinus

These two options are complementary and very similar to [those for meshes](#). Their purpose, based on current selected control points, is to reduce or enlarge this selection.

The algorithm is the same as with meshes:

**More** for each selected control point, select **all** its linked points (i.e. two, three or four).

**Less** for each selected control point, if **all** points linked to this point are selected, keep it selected. For all other selected control points, de-select them.

This implies two points:

- First, when **all** control points of a surface are selected, nothing will happen (as for *Less*, all linked points are always selected, and of course, *More* can't add any). Conversely, the same goes when no control point is selected.
- Second, these tools will never “go outside” of a surface (they will never “jump” to another surface in the same object).

## Surface Editing

Surface editing has even fewer tools and options than its curve counterpart - and has many common points with it... So this page covers (or tries to cover) all the subjects, from the basics of surface editing to more advanced topics, like retopology.

### Basic Surface Editing (translation, rotation, scale)

---

#### Reference

Mode: *Edit* mode

Menu: *Surface* → *Transform* → *Grab/Move, Rotate, Scale, ...*

Hotkey: *G / R / S*

---

Once you have a selection of one or more control points, you can grab/move (G), rotate (R) or scale (S) them, like many other things in Blender, as described in the [Manipulation in 3D Space](#) section.

You also have in *Edit* mode an extra option when using these basic manipulations: the [proportional editing](#).

### Advanced Transform Tools

---

#### Reference

Mode: *Edit* mode

Menu: *Surface* → *Transform*

---

The *To Sphere*, *Shear*, *Wrap* and *Push/Pull* transform tools are described in the [Mesh Editing](#) chapter. Surfaces have no specific transform tools.

### NURBS Control Points Settings

---

#### Reference

Mode: *Edit* mode

Panel: *Curve Tools* (*Editing* context), and *Transform Properties*

---

We saw in a [previous page](#) that NURBS control points have a weight, which is the influence of this point on the surface. You set it either using the big *Set Weight* button in the *Curve Tools* panel (after having defined the weight in the numeric field to the right), or by directly typing a value in the *W* numeric field of the *Transform Properties* panel.

## Adding or Extruding Reference

Mode: *Edit mode*

Menu: *Surface* → *Extrude*

Hotkey: E (or `Ctrl-LMB`)

Unlike meshes or curves, you cannot generally directly add new control points to a surface (with `Ctrl-LMB` clicks), as you can only extend a surface by adding a whole U- or V-row at once. The only exception is when working on a NURBS surface curve, i.e. a surface with only one control point on each U- or V-row. In this special case, all works exactly as with *curves*.

Most of the time, only extrusion is available. As usual, once the tool is activated the extrusion happens immediately and you are placed into *Grab mode*, ready to drag the new extruded surface to its destination.

There are two things very important to understand:

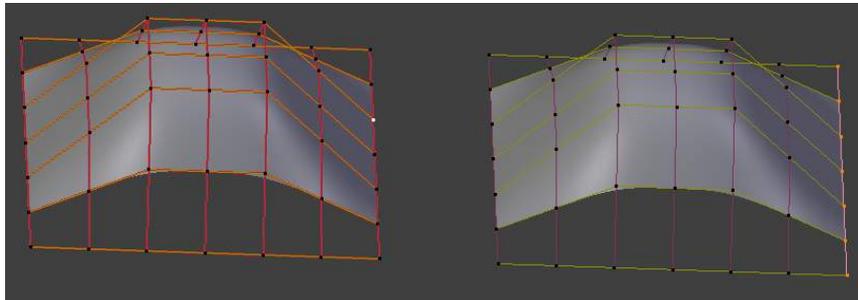
- Surfaces are **2D** objects - so you can't extrude anything *inside* a surface (e.g. "inner" row); it wouldn't make any sense!
- The control "grid" *must remain "squarish"*, which means that you can only extrude a whole row, not parts of rows here and there...

To summarize, the *Extrude* tool will only work when one and only one whole border row is selected - otherwise nothing happens.

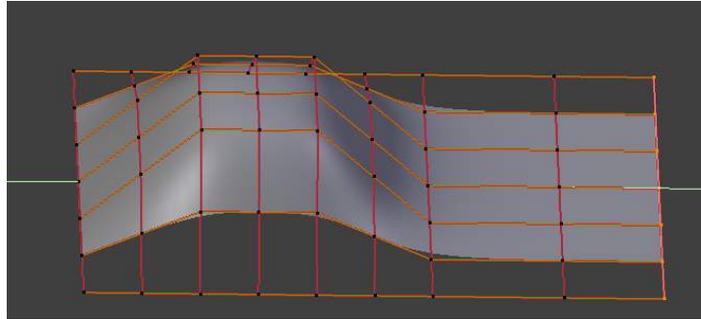
As for curves, you cannot create a new surface in your object out of nowhere, by just `Ctrl-LMB` -clicking with nothing selected. However, unlike for curves, there is no "cut" option allowing you to separate a surface into several parts, so you only can create a new surface by copying (*Duplication*) an existing one (`Shift-D`), or adding a new one (*Add menu*...).

**Examples** Images (*Selecting control-point*) to (*Complete*) show a typical extrusion along the side of a surface.

In (*Selecting control-point*) and (`Shift-R`), a border row of control points were highlighted by selecting a single control point, labeled C, and then using the handy row select tool (`Shift-R`) to select the rest of the control points.



The edge is then extruded using E as shown in (*Extruding*). Notice how the mesh has bunched up next to the highlighted edge; the area in question is highlighted in a light-gray circular area. That is because the *new* extruded surface section is bunched up there as well.



By moving the new section away from the area, the surface begins to “unbunch”. The direction of movement is marked with a white arrow, labeled E, and the new section is labeled S.

You can continue this process of extruding - or adding - new surface sections until you have reached the final shape for your model.

---

### Opening or Closing a Surface Reference

Mode: *Edit* mode

Menu: *Surface* → *Toggle Cyclic*

Hotkey: C

---

As in *curves*, surfaces can be closed (cyclic) or open. However, as surfaces are 2D, you can control this property independently along the U and V axes.

To toggle the cyclic property of a surface along one axis, use C and choose either *cyclic U* or *cyclic V* from the [Toggle pop-up menu](#). The corresponding surface’s outer edges will join together to form a “closed” surface.

---

#### **Note:** Inner and Outer

Surfaces have an “inner” and “outer” face, the first being black whereas the latter is correctly shaded - there does not seem to be any “double sided” shading option for surfaces...). When you close a surface in one or two directions, you might get an entirely black object! In this case, just *Switch Direction* of your surface...

---

### Duplication Reference

Mode: *Edit* mode

Menu: *Curve* → *Duplicate*

Hotkey: Shift-D

---

Well, as with meshes and curves, this command just duplicates the selection. As usual, the copy is selected and placed in *Grab* mode, so you can move it to another place.

However, with surfaces there are some selections that can’t be duplicated, in which case they will just be placed in *Grab* mode... In fact, only selections forming a *single valid sub-grid* are copyable; let’s see this in practice:

- You can copy a single control point. From it, you will be able to “extrude” a “surface curve” along the U axis, and then extrude this unique U-row along the V axis to create a real new surface.
- You can copy a single continuous part of a row (or a whole row, of course). This will give you a new **U-row**, even if you selected (part of) a V-row!
- You can copy a single whole sub-grid.

Note that trying to duplicate several valid “sub-grids” (even being single points) at once won’t work; you’ll have to do it one after the other...

## Deleting Elements

### Reference

Mode: *Edit mode*

Menu: *Curve* → *Delete...*

Hotkey: *X* or *Delete*

The *Erase* pop-up menu of surfaces offers you two options:

**Selected** This will delete the selected rows, *without* breaking the surface (i.e. the adjacent rows will be directly linked, joined, once the intermediary ones are deleted). The selection must abide by the following rules:

- Whole rows, and only whole rows must be selected.
- Only rows along the same axis must be selected (i.e. you can’t delete both U- and V-rows at the same time).

Also remember that NURBS order cannot be higher than its number of control points in a given axis, so it might decrease when you delete some control points... Of course, when only one row remains, the surface becomes a “surface curve”; when only one point remains, there is no more visible surface; and when all points are deleted, the surface itself is deleted.

**All** As with meshes or curves, this deletes everything in the object!

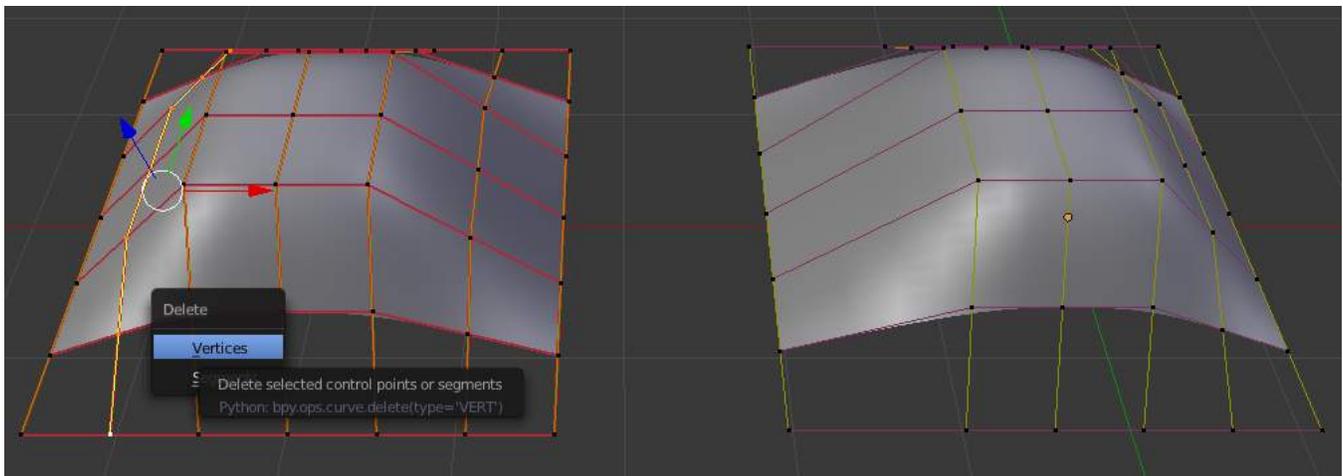


Fig. 2.542: Before and after

**Example** In (*Before*) a row of control points has been selected by initially selecting the control point labeled *A* and using *Shift-R* to select the remaining control points. Then, using the *Erase* menu (*X*), the *selected* row of control points is erased, resulting in (*After*).

## Joining or Merging Surfaces

### Reference

Mode: *Edit* mode

Menu: *Surface* → *Make Segment*

Hotkey: F

Just like *curves*, merging two surfaces requires that a single edge, a border row of control points, from two separate surfaces are selected. This means that the surfaces must be part of the same object. For example, you can't join two surfaces while in *Object* mode - but you can of course, as with any objects of the same type, join two or more *Surface* objects into one object (*Ctrl-J*) - they just won't be "linked" or merged in a single one... Yes, it's a bit confusing!

This command is equivalent to creating edges or *F* *aces* for meshes (hence its shortcut), and so it only works in *Edit* mode. The selection must contain only border rows of the same resolution (with the same number of control points), else Blender will try to do its best to guess what to merge with what, or the merge will fail (either silently, or stating that *Resolution* doesn't match if rows with different number of points are selected, or that there is *Too few selections to merge* if you only selected points in one surface...).

So to avoid problems, you should always only select border rows with the same number of points... Note that you can join a border U-row of one surface with a border V-row of another one, Blender will automatically "invert" the axis of one surface for them to match correctly.

NURBS surface curves are often used to create objects like hulls, as they define cross sections all along the object, and you just have to "skin" them as described above to get a nice, smooth and harmonious shape. See [this tutorial](#) for a detailed workflow.

**Examples** (*Joining ready*) is an example of two NURBS surface curves, **not** NURBS curves, in *Edit* mode, ready to be joined. (*Joining complete*) is the result of joining the two curves.

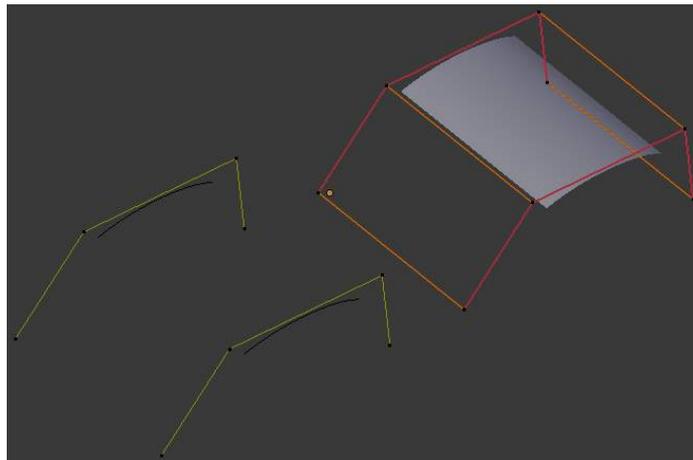


Fig. 2.543: Joining ready.

## Subdivision

### Reference

Mode: *Edit* mode

Panel: *Curve Tools1* (*Editing* context)

Menu: *Surface* → *Segments* → *Subdivide*, *Specials* → *Subdivide*

Hotkey: *[W]* → *[pad1]*

---

Surface subdivision is most simple: using either the *Subdivide* entry in the *Specials* menu (*W*), or the *Subdivide* button of the *Curve Tools1* panel, you will subdivide once all *completely selected grids* by subdividing each “quad” into four smaller ones.

If you apply it to a 1D surface (a “surface curve”), this tool works exactly as with *curves*.

---

## Spin

### Reference

Mode: *Edit* mode

Panel: *Curve Tools1* (*Editing* context)

---

This tool is a bit similar to its *mesh* counterpart - but with less control and options (in fact, there’s none!).

It only works on selected “surfaces” made of *one U-row* (and not with one *V-row*), so-called “surface curves”, by “extruding” this “cross section” in a square pattern, automatically adjusting the weights of control points to get a perfect circular extrusion (this also implies closing the surface along the *V* axis), following exactly the same principle as for the *NURBS Tube* or *NURBS Donut* primitives.

---

## Switch Direction

### Reference

Mode: *Edit* mode

Menu: *Surface* → *Segments* → *Switch Direction*, *Specials* → *Switch Direction*

Hotkey: *[W]* → *[pad2]*

---

This command will “reverse” the direction of any curve with at least one selected element (i. e. the start point will become the end one, and *vice versa*). Mainly useful when using a curve as path, or the bevel and taper options...

---

## Other Specials Options

### Reference

Mode: *Edit* mode

Menu: *Specials*

Hotkey: *W*

---

The *Specials* menu contains exactly the same additional options as for *curves* - but I suppose *Set Radius* and *Smooth Radius* have nothing to do here...

**Conversion** As there are only NURBS surfaces, there is no “internal” conversion here.

However, there is an “external” conversion available, from surface to mesh, that only works in *Object* mode. It transforms a *Surface* object into a *Mesh* one, using the surface resolutions in both directions to create faces, edges and vertices.

**Retopology** Snapping surface components is the same as is with meshes and curves. See [Retopology](#) for more information.

**Misc Editing** You have some of the same options as with meshes, or in *Object* mode. You can `separate` a given surface (`P`), make other selected objects `children` of one or three control points (`Ctrl-P` - note however that parenting to three control points has a strange behavior with curves...), or `add hooks` to control some points with other objects.

The *Mirror* tool is also available, behaving exactly as with `mesh vertices`.

## 2.3.6 Text

### Text Objects

---

#### Reference

Mode: *Edit mode (Text)*

Panel: *Curve and Surface, Font and Char (Editing context)*

Menu: *Add → Text*

---

*Text* objects are exactly what they sound like: they contain some text. They share the same object type as curves and surfaces, as modern fonts (OpenType, TrueType, etc.) are vectorial, made of curves (generally Béziers).

Blender uses a “Font System” to manage mapping “letter codes → objects representing them in 3D views”. This implies that not only does the font system have its own *built-in* font, but it can use external fonts too, including *PostScript Type 1*, *OpenType* and *TrueType* fonts. And last but not least, it can use any objects existing in the current `.blend` file as letters...

Texts in Blender allow you to create/render 2D or 3D text, shaded as you want, with various advanced layout options (like justifying and frames), as we will see below. By default, letters are just flat filled surfaces, exactly like any closed 2D curve. But you can of course extrude them... And texts can follow other curves.

Of course, once you are happy with the shape of your text, you can convert it (with `Alt-C`, in *Object* mode), either to a curve, or directly to a mesh, allowing you to use all the powerful features of these types of objects on it...

(*Text Examples*) shows some examples of various fonts in action, including the “blue” font that has been applied to a curve path.

---

**Note:** A maximum of 50000 characters is allowed per text object; however, be forewarned that the more characters a single text object has, the slower the object will respond interactively.

As you can see when you switch between *Object* and *Edit* modes, the *Font* panel remains the same. This means that its settings can be applied equally in both modes ... and this implies that you cannot apply them to just a part of the mesh. So font, size, and so on, are common to all letters in a *Text* object. There is just one exception: the *Bold / Italic* buttons control properties specific to each letter (this is a way to use up to four different fonts in a text).

For optimum resource usage, only characters that are being used consume memory (rather than the entire character set).

---



The image displays five lines of text in different fonts, all in a yellow color with a slight 3D effect. The fonts are: 'Blender font' (a bold, rounded sans-serif), 'Agate font' (a classic serif), 'Becker font' (a decorative, slightly irregular serif), 'Blippo font' (a very bold, rounded sans-serif), and 'Slogan font' (a cursive script). To the right of these examples, a blue arc of text curves around them, containing the sentence 'This is a text on a curve. driving me in circles'.

Blender font  
Agate font  
Becker font  
Blippo font  
Slogan font

This is a text on a curve. driving me in circles

Fig. 2.544: Text Examples.

---

## Editing Text

---

### Reference

Mode: *Edit* mode

Hotkey: see below

---

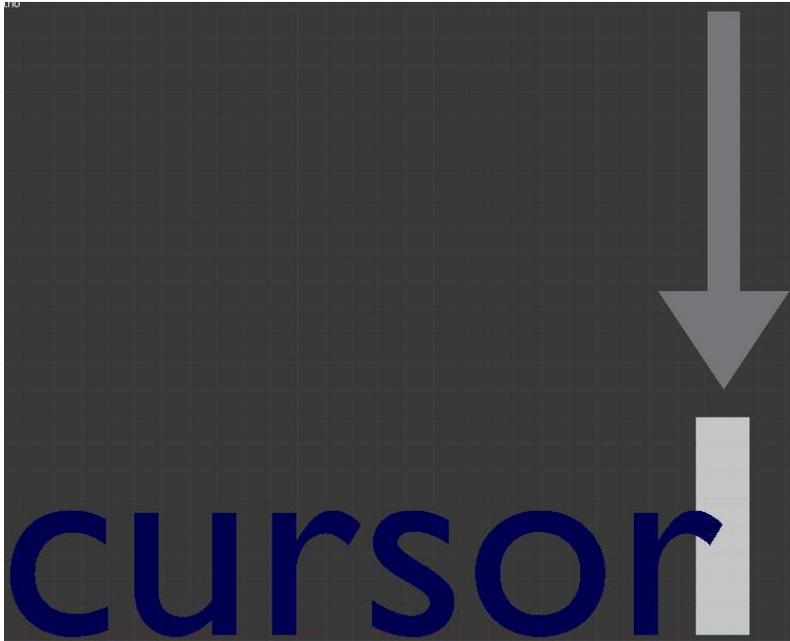


Fig. 2.545: Text in Edit mode.

Editing text is quite different from other object types in Blender, and happens mainly in two areas. First, the 3D view, of course, where you type your text, and have a few shortcuts, e.g. for applying styles (see *Character*) - note however that most Blender hotkeys you know in *Edit* mode do not exist for texts! The second place is the *Button* window (*Editing* context), especially the *Font* panel.

The menu of the 3D view header has nearly no use, and there is no *Specials* menu... You have no transform nor mirror tools, and so on. However, you can apply to texts the same modifiers as for curves.

Editing *Text* is similar to using a standard text editor but is not as full-featured and has some differences:

**Exit *Edit* mode** `Tab` doesn't insert a tab character in the text, but rather enters and exits *Edit* mode, as with other object types.

**Copy** To copy text to the buffer, use `Ctrl-C` or the *Copy* button in the tool shelf.

**Cut and Copy** To cut and copy text to the buffer, use `Ctrl-X` or the *Cut* button in the tool shelf.

**Paste** To paste text from the buffer, use `Ctrl-V` or the *Paste* button in the tool shelf.

**Delete all text** To completely erase or delete all text, use `Ctrl-Backspace`.

**Home/End** `Home` and `End` move the cursor to the beginning and end of a line respectively.

**Next/Previous word** To move the cursor on a word's boundary, use `Ctrl-Left` or `Ctrl-Right`.

The text buffer does not communicate with the desktop. It only works within Blender. To insert text from outside Blender, see [Inserting Text](#) below.

**Inserting Text** You can insert text in three different ways: from the internal text buffer ([Editing Text](#)), or from a text file.

To load text from a text file, use the *Text* → *Paste File* tool. This will bring up a *File Browser* window for navigating to a valid UTF-8 file. As usual, be careful that the file doesn't have too many characters, as interactive response will slow down.

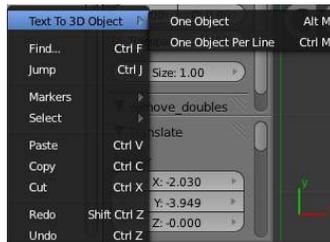
## Special Characters Reference

Mode: *Edit mode*

Menu: *Text* → *Special Characters*

If you need special characters (such as accented chars, which aren't on your keyboard) you can produce many of them using a combination of two other characters. To do so, type the main char, press *Alt-Backspace*, and then press the desired "modifier" to produce the special character. Some examples are given below:

A, <i>Alt-Backspace</i> , ~	ã	A, <i>Alt-Backspace</i> , ´	á	A, <i>Alt-Backspace</i> , `	à
A, <i>Alt-Backspace</i> , O	å	E, <i>Alt-Backspace</i> , "	ë	O, <i>Alt-Backspace</i> , /	ø



**Convert text to text object** An easy way to get text into Blender is to type it in [The Text Editor](#). It is suggested to do this with a *split window* as you will be able to see the 3D view port and text editor at the same time. In the *Text Editor* select *Text* > *Create Text Block*. Then begin typing. When finished, select *Edit* >> *Text to 3D Object* >> *One Object* or *One Object per Line* depending on your needs. It is also possible to load a text file via *Text* >> *Open Text Block* which can be useful for importing large amounts of text at once.

**3D Mesh** It is possible to convert a Text Object to a 3D Mesh object. This can be useful so that you may edit the vertices in *Edit Mode*, but you will lose the ability to edit the text itself. To do this, go to *Object Mode* and select your Text Object. Press *Alt-C* and select *Mesh From Curve/Meta/Surf/Text*. Now you can return to *Edit Mode* and manually edit the vertices. They are usually a bit messy, so it may be useful to use a *Limited Dissolve* deletion or *Remesh Object Modifier* at a low threshold to clean up your mesh.

**Text Selection** In *Edit mode*, your text has a white cursor, and as in any text editor, it determines where new chars will be inserted! You move this cursor with the arrow keys or *PageUp* / *PageDown* and *Home* / *End* keys.

Hold *Shift* while using the arrow keys to select a part of the text. You can use it to specify different materials, the normal/bold/italic state, and not much more...

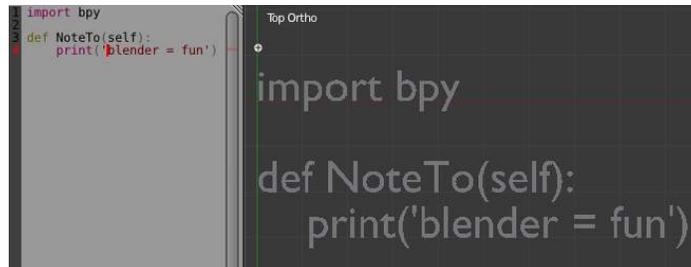


Fig. 2.546: left normal text, right the made text object.

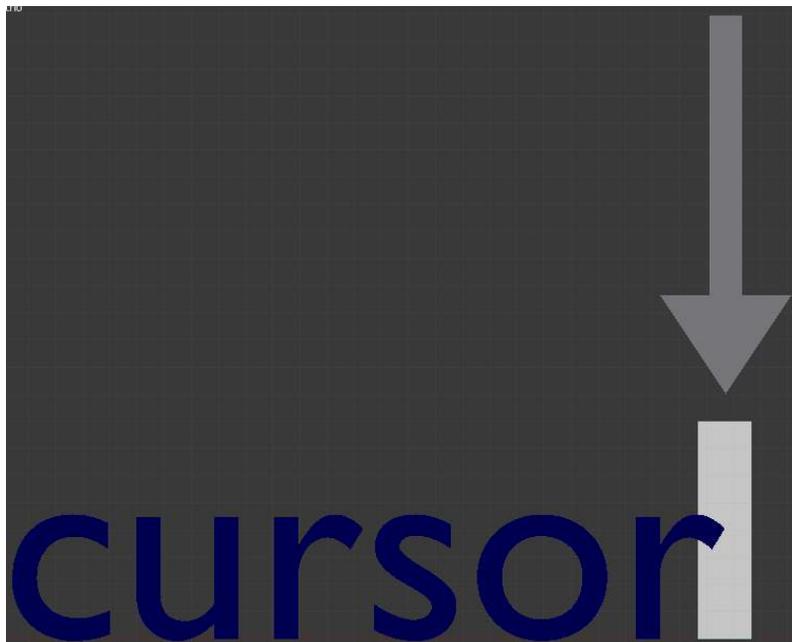


Fig. 2.547: Text in Edit mode.

## Formatting Text

### Fonts

#### Reference

Mode: *Edit* mode

Panel: *Font* (*Editing* context)

The *Font* panel has several options for changing the look of characters.

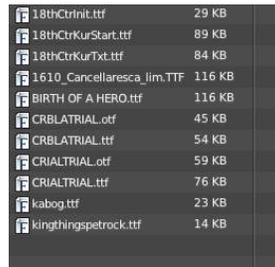


Fig. 2.548: Loading a Type 1 font file.

**Loading and Changing Fonts** Blender comes with a *built-in* font by default and is displayed in each of the four font style choosers. The *built-in* font is always present and shows in this list as `Bfont`. The first icon contains a drop-down list displaying currently loaded fonts. Select one for each font style.

To load a different *Font*, click one of the *Load* buttons in the *Font* panel and navigate to a *valid* font. The *File Browser* window will give all valid fonts a capital F icon, as seen in *Loading a Type 1 font file*.

**Note:** Unix note

Fonts are typically located under `/usr/lib/fonts`, or some variant like `/usr/lib/X11/fonts`, but not always. They may be in other locations as well, such as `/usr/share/local` or `/usr/local/share`, and possibly related sub-trees.

If you select a font that Blender can't understand, you will get the error `Not a valid font`.

Remember the same font will be applied to all chars with same style in a text, but that a separate font is required for each style. For example, you will need to load an *Italics* font in order to make characters or words italic. Once the font is loaded you can apply that font "Style" to the selected characters or the whole object. In all, you would need to load a minimum of four different types of fonts to represent each style (**Normal**, **Italics**, **Bold**, **Bold-Italics**).

It is important to understand that Blender does not care what font you load for "normal", "bold", etc., styles. This is how you can have up to four different fonts in use in the same text - but you have to choose between different styles of a same font, or different fonts. Blender has a number of typographic controls for changing the style and layout of text, found in the *Font* panel.

### Size and Shear

**Size** Controls the size of the whole text (no way to control each char size independently). Note however that chars with different fonts (different styles, see below) might have different visible sizes.

**Shear** Controls the inclination of the whole text. Even if this seems similar to italics style, *this is not the same thing* !



Fig. 2.549: shear: ‘blender’ has a shear value of 1, ‘2.59’ a shear value of 0

**Objects as Fonts** You can also “create” your own “font” inside Blender! This is quite a complex process, so let’s detail it:

- First, you must create your chars. Each char is an object *of any type* (mesh, curve, meta...). They all must have a name following the schema: `common prefix` followed by the `char name` (e.g. `ft.a`, `ft.b`, etc.).
- Then, for the *Text* object, you must enable the *Dupli Verts* button (*Object* context - *Anim Settings* panel).
- Back in *Editing* context, in the *Font* panel, fill the *Ob Family* field with the *common prefix* of your “font” objects.

Now, each time a char in your text matches the *suffix part* of a “font” object’s name, this object is duplicated on this char. *The original chars remain visible*. The objects are duplicated so that their center is positioned at the *lower right corner* of the corresponding chars.

**Text on Curve** With the [curve modifier](#) you can let text follow a curve.

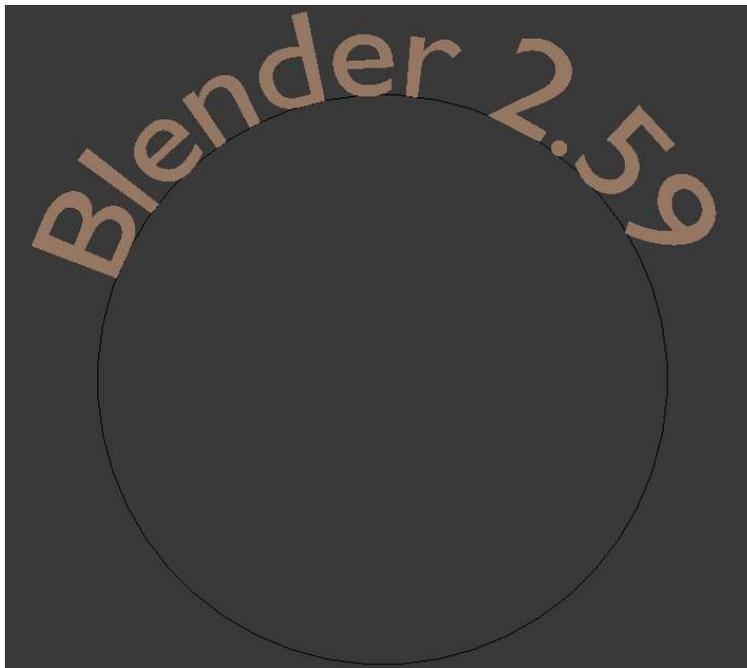


Fig. 2.550: Text on curve.

In (*Text on curve*) you can see a text deformed by a curve (a 2D Bézier circle).

To apply the curve modifier, the text object first has to be converted to a mesh, using `Alt-C` and click mesh.

---

**Note:** There is also a Text on Curve feature, but the curve modifier offers more options.

---

### Underline

**Underline** Toggled with the *Underline* button before typing. Text can also be set to Underlined by selecting it then using the *Underline* button in the Tool Shelf.

**Position** This allows you to shift vertically the position of the underline.

**Thickness** This controls the thickness of the underline.

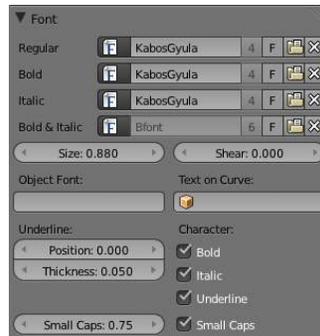


Fig. 2.551: check a character option to, for example, type bold text

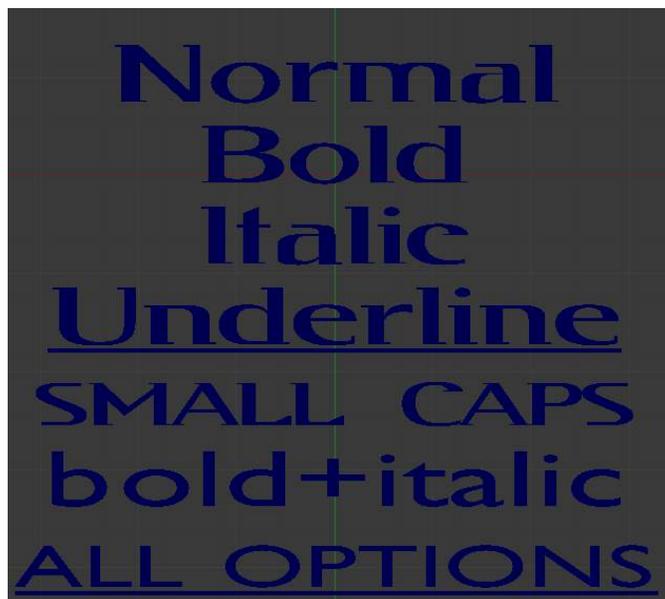


Fig. 2.552: Bold text.

### Character

**Bold** Toggled with the *Bold* button before typing. Text can also be set to Bold by selecting it then using the *Bold* button in the Tool Shelf.

**Italics** Toggled with the *Italic* button before typing. Text can also be set to Italic by selecting it then using the *Italic* button in the Tool Shelf.

**Underline** Enables underlining, as controlled by the Underline settings above.

**Small Caps** type small capital text.

Blender's *Bold* and *Italic* buttons don't work the same way as other applications, as they also serve as placeholders for you to load up other fonts manually, which get applied when you define the corresponding style; see *Fonts*.

To apply the Bold/Italics/Underline attribute to a set of characters, you either turn on *Bold / Italics / Underline* prior to typing characters, or highlight (select) first and then toggle Bold/Italics/Underline.

**Setting Case** You can change the text case by selecting it then clicking the *To Upper* or *To Lower* in the tool shelf.

Enable the *Small Caps* option to type characters as small caps.

The size of the *Small Caps* can be changed with the *Small Caps Scale* setting. Note that the *Small Caps Scale* is applied the same to all *Small Caps* formatted characters.

**Paragraph** The *Paragraph* Panel has settings for the alignment and spacing of text.

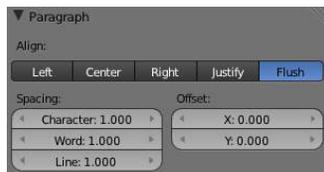


Fig. 2.553: the paragraph tab

### Align

**Left** Aligns text to left of frames when using them, else uses the center point of the *Text* object as the starting point of the text (which grows to the right).

**Center** Centers text in the frames when using them, else uses the center point of the *Text* object as the mid-point of the text (which grows equally to the left and right).

**Right** Aligns text to right of frames when using them, else uses the center point of the *Text* object as the ending point of the text (which grows to the left).

**Justify** Only flushes a line when it is **terminated** by a wordwrap (**not** by Return), it uses *whitespace* instead of *character spacing* (kerning) to fill lines.

**Flush** **Always** flushes the line, even when it's still being entered; it uses character spacing (kerning) to fill lines.

Both *Justify* and *Flush* only work within frames.

### Spacing

**Character** A factor by which space between each character is scaled in width

**Word** A factor by which whitespace between words is scaled in width. You can also control it by pressing Alt-Left or Alt-Right to decrease/increase spacing by steps of 0.1.

**Line** A factor by which the vertical space between lines is scaled.

## Offset

**X offset and Y offset** Well, these settings control the X and Y offset of the text, regarding its “normal” positioning. Note that with frames (see [Text Boxes](#)), it applies to all frames’ content...

## Shape

### Reference

Mode: *Object* or *Edit* modes

Panel: *Curve and Surface* (*Editing* context)

As you can see in the *Curve and Surface* panel, texts have most of the same options as curves.

### Resolution

**Preview** the resolution in the viewport.

**Render** the resolution on the render.

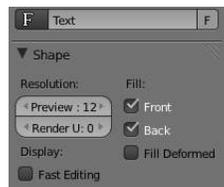


Fig. 2.554: the shape settings

**Fast Editing** disables curve filling while in edit mode.

**Fill** The fill options control how the text curves are filled in when text is *Extruded* or *Beveled* in the *Geometry* Panel.

**Front** Fills in the front side of the surface.

**Back** Fills in the back side of the surface.

**Fill Deformed** Fills the curves after applying shape keys and modifiers.



Fig. 2.555: Texture Settings

## Textures

**Use UV for Mapping** Use UV values as generated texture coordinates.

**Auto Texture Space** Adjusts the active object’s texture space automatically when transforming object.

## Geometry

Text objects have all the `curves` `extrusion` features.

## Text Editing

### Text Boxes

---

## Reference

Mode: *Object* or *Edit* modes

Panel: *Font* (*Editing* context)

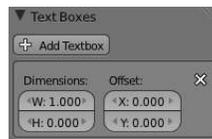


Fig. 2.556: Text frame.

Text “Boxes” allow you to distribute the text amongst rectangular areas within a single text object. An arbitrary number of freely positionable and re-sizable text frames are allowed per text object.

Text flows continuously from the lowest-numbered frame to the highest-numbered frame with text inside each frame word-wrapped. Text flows between frames when a lower-numbered frame can’t fit any more text. If the last frame is reached, text overflows out of it.

Text frames are very similar to the concept of *frames* from a desktop publishing application, like Scribus. You use frames to control the placement and flow of text.

Frames are controlled in the *Text Boxes* panel.

**Frame size** By default the first frame for a new text object, and any additional frames, has a size of **zero** for both *Width* and *Height*, which means the frame is initially not visible.

Frames with a width of **0.0** are ignored completely during text flow (no wordwrap happens), and frames with a height of **0.0** flow forever (no flowing to the next text frame).

In order for the frame to become visible, the frame’s *Width* must be greater than **0.0**.

---

**Note:** Technically the height is never actually **0.0** because the font itself always contributes height.

---

(*Frame width*) is a text object with a width of `5.0`. And because the frame width is greater than `0.0` it is now visible and is drawn in the active theme color as a dashed rectangle. The text has overflowed because the text has reached the end of the last frame, the default frame.

**Adding/Deleting a Frame** To add a frame click the *Add Textbox* button on the *Text Boxes* panel. A new frame is inserted just after (in text flow order) the current one, with its attributes (position and size). Be sure to modify the offset for the new frame in the *X* and/or *Y* fields. Just an *X* modification will create a new column.

To delete the current frame, click the `Delete` button. Any text in higher frames will be re-flowed downward into lower frames.



Fig. 2.557: Frame width.



Fig. 2.558: wrapping

**Example: Text Flow** With two or more frames you can organize text to a finer degree. For example, create a text object and enter `Blender is super duper`. This text object has a frame; it just isn't visible because its *Width* is **0.0**.

Set the width to **5.0**. The frame is now visible and text is wrapping according to the new width, as shown in (*Text 2*). Notice that the text has overflowed out of the frame. This is because the text has reached the end of the last frame, which just happens to be the default/initial frame.



Fig. 2.559: text flowing from box 1 to box 2

When we add another frame and set its width and height, the text will flow into the new frame.

**Example: Multiple columns** To create two columns of text just create a text object and adjust the initial frame's *Width* and *Height* to your requirements, then insert a new frame. The new frame will have the same size as the initial frame. Set the *X* position to something greater or less than the width of the initial frame; see (*Text 5*).

### Assigning Materials

---

### Reference

Mode: *Edit* mode

Panel: *Link and Materials* (*Editing* context)

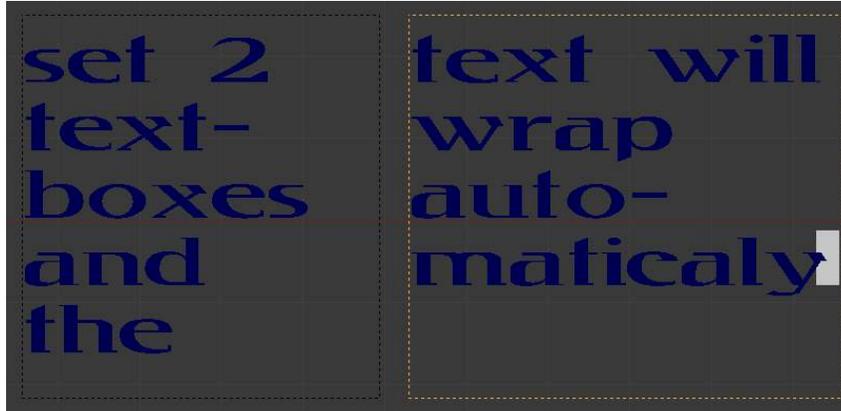


Fig. 2.560: Text 5.

Each character can have a different *Material index* in order to have different materials on different characters.

You can assign indices either as you type, or after by selecting blocks of text and clicking on the *Assign* button in the Materials panel.



Fig. 2.561: Red Green Blue.

For example, to create (*Red Green Blue*) you would need to create three separate materials and three separate material indices. Each word would be assigned a *Material index* by selecting the characters for each word and clicking the *Assign* button. (*Red Green Blue*) is still one single *Text* object.

## 2.3.7 Metas

### Meta Objects

#### Reference

Mode: *Object* or *Edit* modes

Menu: *Add* → *Meta*

Hotkey: *Shift-A*

Meta objects are *implicit surfaces*, meaning that they are *not explicitly* defined by vertices (as meshes are) or control points (as surfaces are): they exist *procedurally*. Meta objects are literally mathematical formulas that are calculated on-the-fly by Blender.

A very distinct visual characteristic of metas is that they are fluid *mercurial*, or *clay-like* forms that have a “rounded” shape. Furthermore, when two meta objects get close to one another, they begin to interact with one another. They “blend” or “merge”, as water droplets do, especially in zero-g (which, by the way, makes them very handy for modeling streams of water when you don’t want to do a fluid simulation). If they subsequently move away from one another, they restore their original shape.

Each of these is defined by its own underlying mathematical structure (*Technical Details*), and you can at any time switch between them using the *Active Element* panel.

Typically *Meta* objects are used for special effects or as a basis for modeling. For example, you could use a collection of metas to form the initial shape of your model and then convert it to another object type (well, only meshes are available...) for further modeling. Meta objects are also very efficient for ray-tracing.

Note that *Meta* objects have a slightly different behavior in *Object* mode.

### Primitives

There are five predefined meta “primitives” (or configurations) available in the *Add* → *Meta* sub-menu:

**Meta Ball** adds a meta with a point underlying structure.

**Meta Tube** adds a meta with a line segment underlying structure.

**Meta Plane** adds a meta with a planar underlying structure.

**Meta Ellipsoid** adds a meta with an ellipsoidal underlying structure.

**Meta Cube** adds a meta with a volumetric cubic underlying structure.

**Visualization** In *Object* mode, the calculated mesh is shown, along with a black “selection ring” (becoming pink when selected).

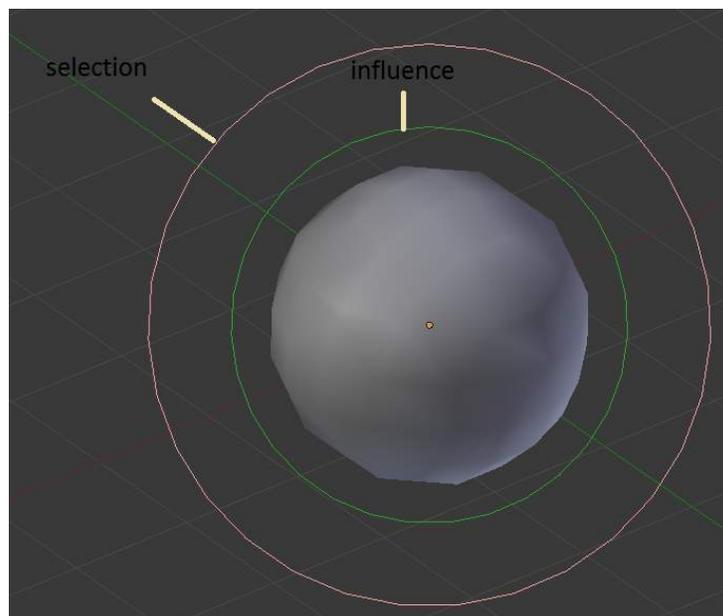


Fig. 2.562: Meta Ball example.

In *Edit* mode (*Meta Ball example*), a meta is drawn as a mesh (either shaded or as black wireframe, but without any vertex of course), with two colored circles: a red one for selection (pink when selected), and a green one for a direct control of the

meta's stiffness (light green when active). Note that except for the *Scale* (S) transformation, having the green circle highlighted is equivalent to having the red one.

**Meta Ball Options** All Meta objects in a scene interact with each other. The settings in the *MetaBall* section apply to all meta objects. In *Edit* mode, the *Active Element* panel appears for editing individual meta elements.



## Resolution

The *Resolution* controls the resolution of the resultant mesh as generated by the

**Meta** object.

**View** The 3D View resolution of the generated mesh. The range is from **0.05** (finest) to **1.0** (coarsest).

**Render** The rendered resolution of the generated mesh. The range is from **0.05** (finest) to **1.0** (coarsest).

One way to see the underlying mathematical structure is to lower the *Resolution*, increase the *Threshold* and set the *Stiffness* (see below) a fraction above the *Threshold*. (*Underlying structure*) is a (*Meta cube*) with the above mentioned configuration applied as follows: *Resolution* of **0.410**, *Threshold* of **5.0** and *Stiffness* a fraction above at **5.01**.

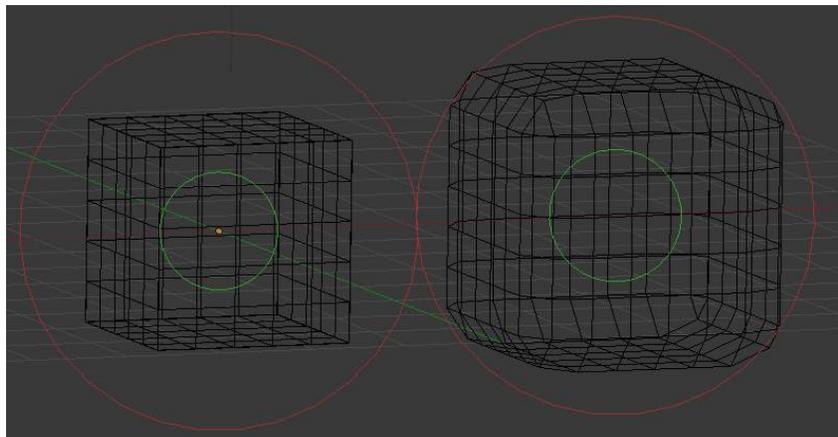


Fig. 2.567: Left: Underlying structure, Right: the shape.

You can clearly see the underlying cubic structure that gives the meta cube its shape.

## Threshold (Influence)

### Reference

Mode: *Object* or *Edit* modes

Panel: *MetaBall* (*Editing* context)

*Threshold* defines how much a meta's surface "influences" other metas. It controls the *field level* at which the surface is computed. The setting is global to a group of *Meta* objects. As the threshold increases, the influence that each meta has on each other increases.

There are two types of influence: **positive** or **negative**. The type can be toggled on the *Active Element* panel while in *Edit* mode, using the *Negative* button. You could think of **positive** as attraction and **negative** as repulsion of meshes. A negative meta will push away or repel the meshes of positive *Meta* objects.

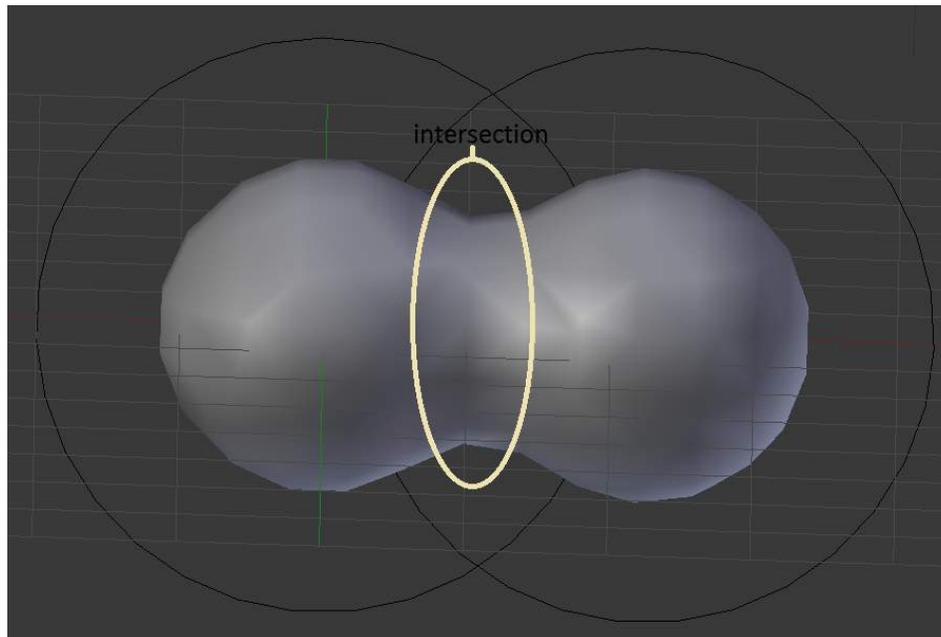


Fig. 2.568: Positive.

A *positive* influence is defined as an attraction, meaning the meshes will stretch towards each other as the *rings of influence* intersect. (*Positive*) shows two meta balls' *rings of influence* intersecting with a *positive* influence.

Notice how the meshes have pulled towards one another. The area circled in white shows the green *influence* rings intersecting.

### Update

While transforming metas (grab/move, scale, etc.), you have four “modes” of visualization, located in the *Update* buttons group of the *MetaBall* panel:

**Always** fully draw the meta during transformations.

**Half Res** During transformations, draw the meta at half its *Wiresize* resolution.

**Fast** Do not show meta mesh during transformations.

**Never** Never show meta mesh (not a very recommended option, as the meta is only visible at render time!).

This should help you if you experience difficulties (metas are quite compute-intensive...), but with modern computers, this shouldn't happen, unless you use many metas, or very high resolutions...

### Meta Structure

#### Technical Details

A more formal definition of a meta object can be given as a *directing structure* which can be seen as the source of a static field. The field can be either positive or negative and hence the field generated by neighboring directing structures can attract or repel.

The implicit surface is defined as the surface where the 3D field generated by all the directing structures assume a given value. For example a meta ball, whose directing structure is a point, generates an isotropic (i.e. identical in all directions) field around it and the surfaces at constant field value are spheres centered at the directing point.

*Meta* objects are nothing more than mathematical formulae that perform logical operations on one another (AND, OR), and that can be added and subtracted from each other. This method is also called **Constructive Solid Geometry** (CSG). Because of its mathematical nature, CSG uses little memory, but requires lots of processing power to compute.

## Underlying Structure

---

### Reference

Mode: *Edit* mode

Panel: *MetaBall tools* (*Editing* context), *Transform Properties*

---

Blender has five types of metas, each determined by its underlying (or directing) structure. In *Edit* mode, you can change this structure, either using the relevant buttons in the *MetaBall tools* panel, or the drop-down list in the *Transform Properties* panel (N). Depending on the structure, you might have additional parameters, located in both *Transform Properties* and *MetaBall tools* panels.

**Ball (point, zero-dimensional structure)** This is the simplest meta, without any additional setting. As it is just a point, it generates an isotropic field, yielding a spherical surface (this is why it is called *Meta Ball* or *Ball* in Blender).

**Tube (straight line, uni-dimensional structure)** This is a meta which surface is generated by the field produced by a straight line of a given length. This gives a cylindrical surface, with rounded closed ends. It has one additional parameter:

**dx** The length of the line (and hence of the tube - defaults to **1.0**).

**Plane (rectangular plane, bi-dimensional structure)** This is a meta which surface is generated by the field produced by a rectangular plane. This gives a parallelepipedal surface, with a fixed thickness, and rounded borders. It has two additional parameters:

**dx, dy** The length, width of the rectangle (defaults to **1.0**).

Note that by default, the plane is a square.

**Ellipsoid (ellipsoidal volume, tri-dimensional structure)** This is a meta which surface is generated by the field produced by an ellipsoidal volume. This gives an ellipsoidal surface. It has three additional parameters:

**dx, dy, dz** The length, width, height of the ellipsoid (defaults to **1.0**).

Note that by default, the volume is a sphere, producing a spherical meta, as the *Ball* option...

**Cube (parallelepipedal volume, tri-dimensional structure)** This is a meta which surface is generated by the field produced by a parallelepipedal volume. This gives a parallelepipedal surface, with rounded edges. As you might have guessed, it has three additional parameters:

**dx, dy, dz** The length, width, height of the parallelepiped (defaults to **1.0**).

Note that by default, the volume is a cube.

## Editing Metas

When in *Edit* mode, the *Active Element* panel appears. These settings apply only to the selected meta element.

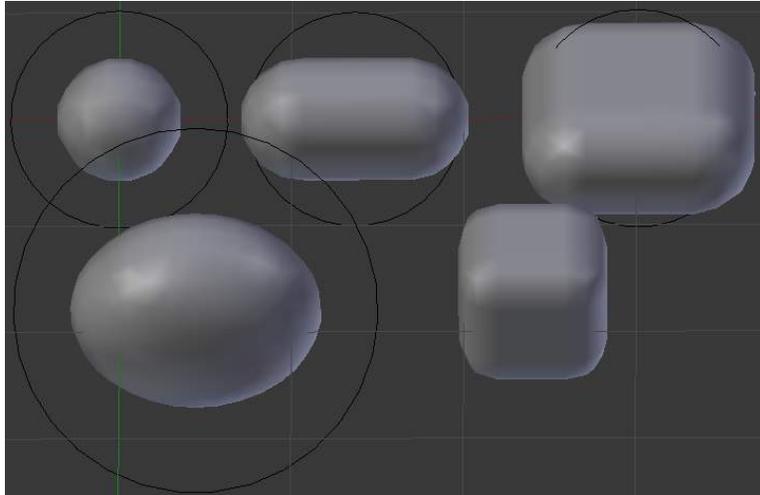


Fig. 2.569: the 5 meta primitives.



Fig. 2.570: the active element panel.

## Meta Shape

The *Type* menu lets you change the shape of the meta object, as explained above.

## Stiffness

Together with *Threshold*, *Stiffness* controls the influencing range. While the threshold is common to all metas in the same object (or even the same *Object Families*), the stiffness is specific to each meta.

Scaling the inner green circle changes the *Stiffness* value. Stiffness defines how much the meta object is filled. This essentially defines how sensitive a meta is to being affected by other metas. With a low stiffness, the meta will begin to deform from further away. A higher value means the meta needs to be close to another one to begin merging.

When a *Meta* object comes within “range” of another meta, the two will begin to interact with each other. They don’t necessarily need to intersect, and depending on the *Threshold* and *Stiffness* settings, they most likely won’t need to. *Stiffness* is materialized by the *green ring*

The range is from **0.0** to **10.0**. But to be visible, the *Stiffness* must be slightly larger than the *Threshold* value. You can also visually adjust the *Stiffness* ring by using the RMB to select it and activate *Scale* mode with *S*.

In (*Stiffness*), the meta ball labeled A, has a smaller *Stiffness* value than the one labeled B. As you can see, the *green ring* radius is different for each of them.

## Negative Influence

The opposite effect of a *positive* influence would be a *negative* influence: the objects repel each other. (*Negative*) shows a meta ball and a meta plane where the first is negative and the second, positive. Notice how the negative meta is not visible: only the

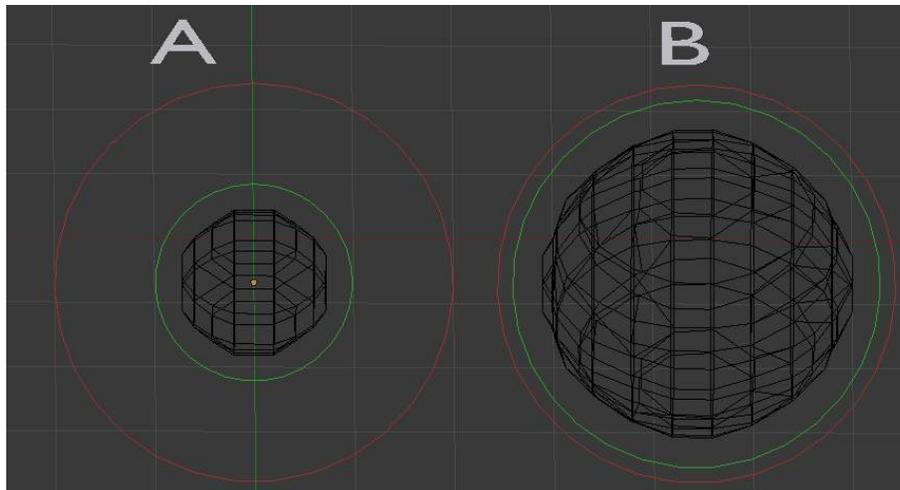


Fig. 2.571: Stiffness.

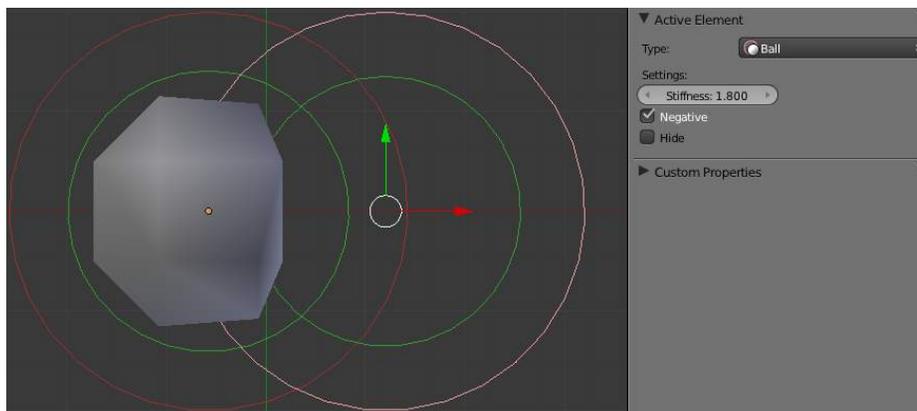


Fig. 2.572: Negative.

surrounding circles appear. This is how Blender indicates that the object is negative.

Moving the sphere to the plane causes the plane’s mesh to “cave in” or collapse inward. If you move the plane away from the sphere, the plane’s mesh will restore itself.

To make a meta *negative*, just select the meta in edit mode, and check *negative* in the *active element* panel.

### Hiding Elements

As in *Object* mode, you can hide the selected meta(s), and then reveal what was hidden. This is very handy for cleaning your views up a bit... Note that the two red and green rings always remain visible in *Edit* mode, as well as the select circle (in *Object* mode...).

To hide the current selection, use **H**, the *Hide* toggle button in the *MetaBall tools*, or the *Metaball* → *Hide MetaElems* → *Hide Selected* menu option.

To hide everything but the current selection, press **Shift-H** or use *Metaball* → *Hide MetaElems* → *Hide Deselected*.

To reveal what was hidden, use **Alt-H**, or the relevant option in the same *Metaball* → *Hide MetaElems* menu. You can also un-toggle the *Hide* button in the (*MetaBall tools* panel).

### Deleting Elements

There is no *Erase* menu for metas, just a confirmation pop-up asking you if you want to delete the selected metas. Clear and simple!

### Conversion

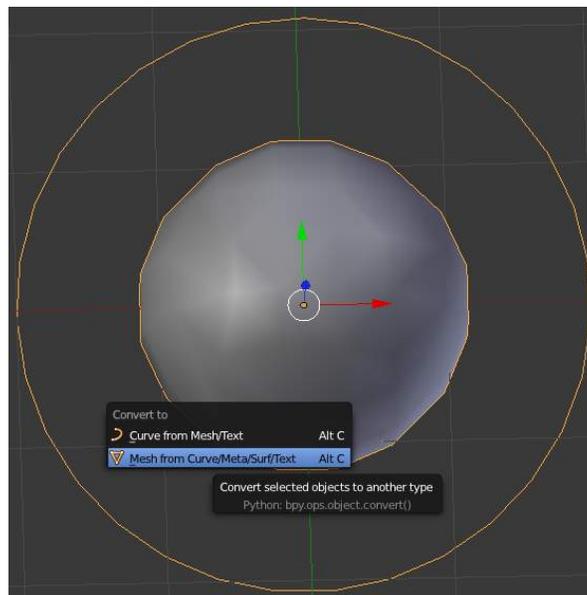


Fig. 2.573: the convert menu

You can only convert metas to meshes, but here you have the option to keep the original *Meta* object (i.e. create a new *Mesh* one, instead of a “real” conversion...). Note that the resolution used for the new mesh is the *Wiresize* one, not the *Renderize* one.

To convert the meta, press **Alt-C** in *Object* mode, and select *mesh*

## Object Families

*Meta* objects have different behavior in *Object* mode than other object types - they can be “regrouped” into so-called “families”.

A “family” is a way to regroup several meta objects, producing something very similar to having several metas inside the same object.

A family is defined by the left part of an object’s name (the one before the dot). Remember, an object’s name is the one in the *OB* field, in most panels, **not** the *MB* field, which is the meta datablock’s name... For example, the *family* part of `MetaPlane.001` is `MetaPlane`. Each meta object in the same “family” is associated with one another as discussed below.

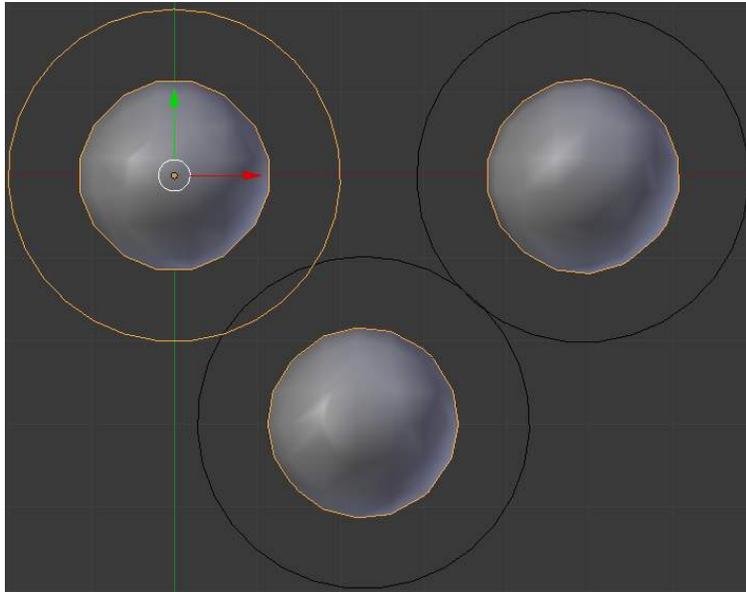


Fig. 2.574: Meta ball base.

Families of metas are controlled by a *base Meta* object which is identified by an *Object* name **without** a right part. For example, if we have five metas called `MetaThing` ", `MetaThing.001`, `MetaThing.002`, `MetaThing.003` and `MetaThing.004`, the *base Meta* object would be `MetaThing`.

The *base Meta* object determines the basis, the resolution, the threshold, *and* the transformations. It also has the material and texture area. The *base meta* is effectively the parent of (or perhaps a better word to use is “the owner of”) the other metas in the group (i.e. it is as if the other metas were “included” or joined into the base one).

## Examples

(*Meta ball base*) shows the *base meta* labeled `B`. The other two *Meta* objects are *children*. Children’s selection rings are always black, while the group’s mesh is orange. Because the metas are grouped, they form a unified mesh which can always be selected by selecting the mesh of any meta in the group. For example, in the example (*Meta ball base*), only the lower sphere (the parent) has been selected, and you see that both the parent’s mesh *and* all of the children’s meshes are now highlighted.

The *base Meta* object controls the **polygonalization** (mesh structure) for the group, and as such, also controls the polygonalization for the children (*non-base*) metas. If we transform the *base meta*, the children’s polygonalization changes. However, if we transform the children, the polygonalization remains unchanged.

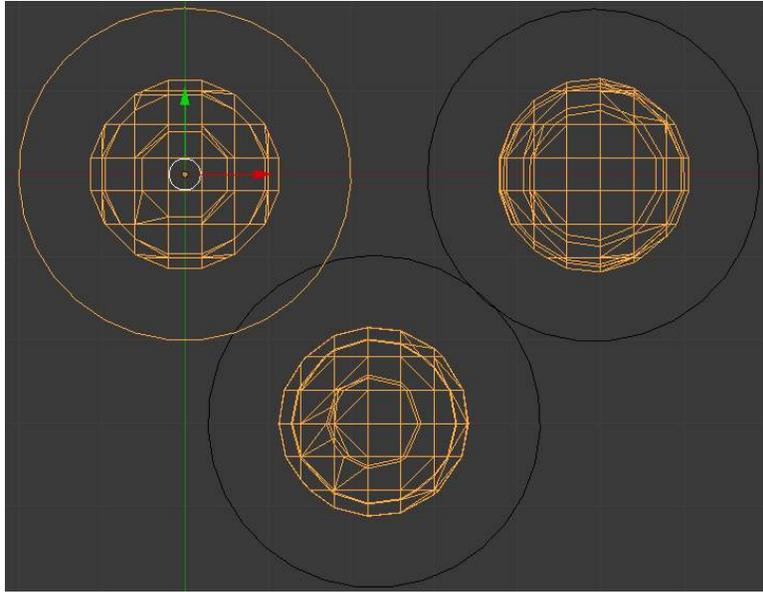


Fig. 2.575: Scaling the “base”.

### Hints

This discussion of “polygonization” *doesn't* mean that the various meshes don't deform towards or away from each other (meta objects always influence one another in the usual way, whether or not they are members of the same family). Rather, it means that the underlying mesh structure changes only when the *base* object transforms. For example, if you scale the *base*, the children's mesh structure changes. In (*Scaling the “base”*), the *base* has been scaled down, which has the effect of scaling the mesh structure of each of the children. As you can see, the children's mesh resolution has increased, while the *base* decreased. *The children did not change size!*

## 2.3.8 Empties

The “Empty” is a null object. It contains no real Geometry, but can be used as a handle for many purposes.

### Settings

**Plain Axes** Draws as six lines, initially with one pointing in each of the +X,-X,+Y,-Y,+Z,and -Z axis directions.

**Arrows** Draws as arrows, initially pointing in the positive X,Y, and Z axis directions, each with a label.

**Single Arrow** Draws as a single arrow, initially pointing in the +Z axis direction.

**Circle** Draws as a circle initially in the XZ plane.

**Cube** Draws as a cube, initially aligned to the XYZ axes.

**Sphere** Draws as an implied sphere defined by 3 circles. Initially, the circles are aligned, one each, to the X, Y, and Z axes.

**Cone** Draws as a cone, initially pointing in the +Y axis direction.

**Image** Empties can display images. This can be used to create reference images, including blueprints or character sheets to model from, instead of using background images. The image is displayed regardless of the 3D display mode. The settings are the same as in [Background Image Settings](#)

**Note:** While alpha-images can be used, there is a known limitation with object draw order, where alphas won't always draw on top of other objects when unselected.

**Size** Controls the local size of the empty. This does not change its scale, but simply resizes the shape.

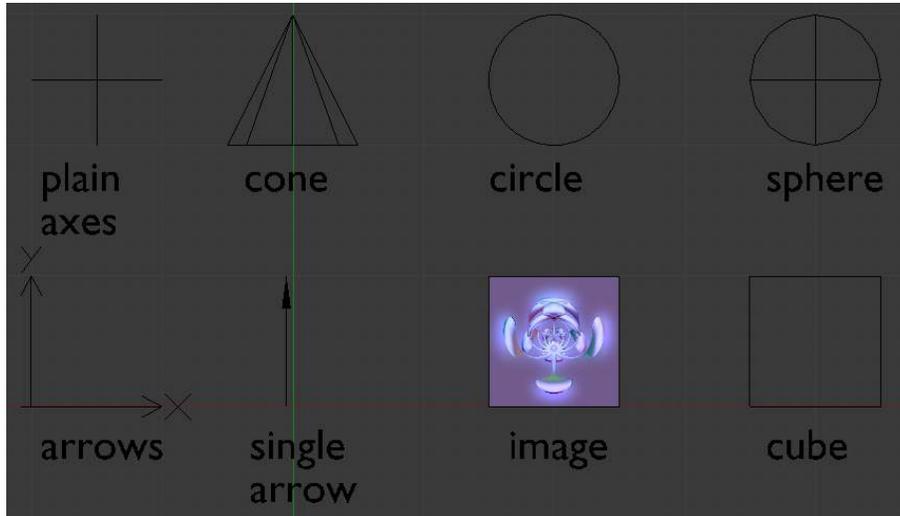


Fig. 2.576: The eight different empty draw types as seen from the top view

### Usage and functions

Empties can serve as transform handles which cannot be edited and do not render. Empties are important and useful objects. Some examples of ways to use them include: *Parent object for a group of objects*

- An Empty can be parented to any number of other objects - This gives the user the ability to control a group of objects easily, and without affecting a render.

#### *Target for constraints*

- An empty can also be used as a target for normal, or bone constraints.
- This gives the user far more control; for instance, a rig can easily be set up to enable a camera to point towards an empty using the **Track to** constraint

#### *Array offset*

- An empty can be used to offset an array modifier, meaning complex deformations can be achieved by only moving a single object.



Fig. 2.577: An example of an empty being used to control an array

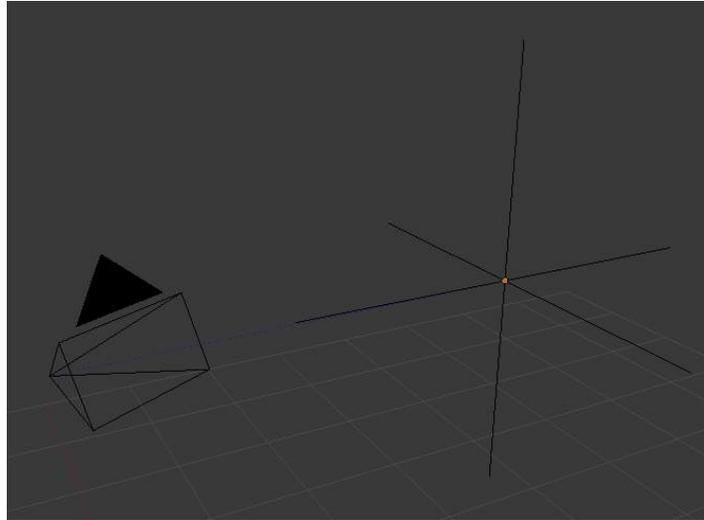


Fig. 2.578: An example of an empty being used to control the track to constraint

*Other common uses.*

- Placeholders
- Rigging controls
- DOF distances
- Reference Images

### 2.3.9 Group Instances

Groups are covered fully here: [Grouping and Parenting](#).

If a group already exists in the scene, an instance of one of those groups can be created from the *Add* menu under *add* → *group instance* → *group*.

These group proxies are controlled/owned by an additional empty object, and hence are not editable (i.e. have no *Edit* mode) - in fact, they behave a bit as if all object copies of the group were children of the empty). So you have all the editing options of objects (you can move/scale/rotate them, etc.).

---

**Note:** Groups and Metas

There seems to be a bug with *Meta* objects in group proxies: their meshes are invisible; only the circles are drawn...

---

#### Visualization

Group “proxies” are drawn in black (unless *Solid* or *Shaded* draw mode, of course). The selection of “real” members of the group is reflected in their “proxies”. When the proxy itself is selected, the empty turns pink, and the other parts, purple.

The only options of these “group proxies” are the same as the ones for `empties` visualization.

## 2.3.10 Scripts

# 2.4 Modifiers

## 2.4.1 Introduction

### Reference

Modifiers are added from the *Modifiers* context of the Properties Editor.

### Modifiers

Modifiers are automatic operations that affect an object in a non-destructive way. With modifiers, you can perform many effects automatically that would otherwise be too tedious to do manually (such as subdivision surfaces) and without affecting the base geometry of your object.

They work by changing how an object is displayed and rendered, but not the geometry which you can edit directly. You can add several modifiers to a single object to form a [Modifier Stack](#) and *Apply* a modifier if you wish to make its changes permanent.



Fig. 2.579: Modifiers menu

There are four types of modifiers:

### Modify

The *Modify* group of modifiers are tools similar to the *Deform Modifiers* (see below), but which do not directly affect the shape of the object; rather they affect some other data, such as vertex groups.

**Mesh Cache** Apply animated mesh data (from external file) to a mesh.

**UV Project** Project UV coordinates on your mesh.

**UV Warp** Dynamically edit the UV coordinates on your mesh.

**Vertex Weight** Edit a vertex group of your mesh, in various ways.

## Generate

The *Generate* group of modifiers are constructive tools that either change the general appearance of or automatically add new geometry to an object.

**Array** Create an array out of your basic mesh and similar (repeating) shapes.

**Bevel** Create a bevel on a selected mesh object.

**Boolean** Combine/subtract/intersect your mesh with another one.

**Build** Assemble your mesh step by step when animating.

**Decimate** Reduce the polygon count of your mesh.

**Edge Split** Add sharp edges to your mesh.

**Mask** Allows you to hide some parts of your mesh.

**Mirror** Mirror an object about one of its own axes, so that the resultant mesh is symmetrical.

**Multiresolution** Sculpt your mesh at several levels of resolution.

**Remesh** Can fix heavily triangulated meshes, and other issues, with careful Threshold adjustments.

**Screw** Generate geometry in a helix-pattern from a simple profile. Similar to the **Screw Tool** in edit mode.

**Skin** Automatically generate topology.

**Solidify** Give depth to mesh faces.

**Subdivision Surface** Subdivides your mesh using Catmull-Clark or Simple algorithms.

**Triangulate** Converts all faces to Triangles.

**Wireframe** Converts all faces into a wireframe.

## Deform

The *Deform* group of modifiers only change the shape of an object without adding new geometry, and are available for meshes, and often texts, curves, surfaces and/or lattices.

**Armature** Use bones to deform and animate your object.

**Cast** Shift the shape of a mesh, surface or lattice to a sphere, cylinder or cuboid.

**Curve** Bend your object using a curve as guide.

**Displace** Deform your object using a texture.

**Hook** Add a hook to your vertice(s) (or control point(s)) to manipulate them from the outside.

**Laplacian Smooth** Allows you to reduce noise on a mesh's surface with minimal changes to its shape.

**Laplacian Deform** allows you to pose a mesh while preserving geometric details of the surface.

**Lattice** Use a Lattice object to deform your object.

**Mesh Deform** Allows you to deform your object by modifying the shape of another mesh, used as a “Mesh Deform Cage” (like when using a lattice).

**Shrinkwrap** Allows you to shrink/wrap your object to/around the surface of a target mesh object.

**Simple Deform** Applies some advanced deformations to your object.

**Smooth** Smooth the geometry of a mesh. Similar to the *Smooth* tool in the mesh editing context.

**Warp** Warp a mesh by specifying two points the mesh stretches between.

**Wave** Deform your object to form (animated) waves.

## Simulate

The *Simulate* group of modifiers activate simulations. In most cases, these modifiers are automatically added to the modifiers stack whenever a *Particle System* or *Physics* simulation is enabled. Their only role is to define the place in the modifier stack used as base data by the tool they represent. Generally, the attributes of these modifiers are accessible in separate panels.

**Cloth** Simulates the properties of a piece of cloth. It is inserted in the modifier stack when you designate a mesh as Cloth.

**Collision** Simulates a collision between objects.

**Dynamic Paint** Makes an object or a particle system paint a material onto another object.

**Explode** Blows up your mesh using a particle system.

**Fluid** The object is part of a fluid simulation... The modifier added when you designate a mesh as Fluid.

**Particle Instance** Makes an object act similar to a particle but using the mesh shape instead.

**Particle System** Represents a particle system in the stack, so it is inserted when you add a particle system to the object.

**Smoke** Simulates realistic smoke.

**Soft Body** The object is soft, elastic... Modifier added when you designate a mesh as Softbody.

**Ocean** Quickly creates a realistic, animated ocean.

## Interface

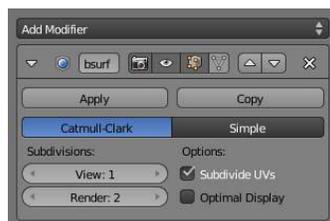


Fig. 2.580: Panel Layout (Subsurf as an example)

Each modifier has been brought in from a different part of Blender, so each has its own unique settings and special considerations. However, each modifier’s interface has the same basic components, see (*Panel Layout (Subsurf as an example)*).

At the top is the *panel header*. The icons each represent different settings for the modifier (left to right):

**Arrow** Collapse modifier to show only the header and not its options.

**Icon** A quick visual reference of the modifier’s type.

**Name** Every modifier has a unique name per object. Two modifiers on one object must have unique names, but two modifiers on different objects can have the same name. The default name is based off the modifier type.

**Camera** Toggles visibility of the modifier effect in the render.

**Eye** Toggles visibility of the modifier effect in the 3D view.

**Box** Displays the modified geometry in edit mode, as well as the original geometry which you can edit.

**Triangle** When enabled, the final modified geometry will be shown in edit mode and can be edited directly.

**Up arrow** Moves modifier up in the stack.

**Down arrow** Moves modifier down in the stack.

**Cross** Deletes the modifier.

---

**Note:** The *Box* and *Triangle* icons may not be available depending on the type of modifier.

---

Below the header are two buttons:

**Apply** Makes the modifier “real” - converts the object’s geometry to match the applied modifier, and deletes the modifier.

**Copy** Creates a duplicate of the modifier at the bottom of the stack.

**Warning:** Applying a modifier that is not first in the stack will ignore the stack order and could produce undesired results.

Below this header, all of the options unique to each modifier will be displayed.

## 2.4.2 The Stack

Modifiers are a series of non-destructive operations which can be applied on top of an objects geometry. They can be applied in just about any order the users chooses.

This kind of functionality is often referred to as a “modifier stack” and is also found in several other 3D applications.

In a modifier stack the order in which modifiers are applied has an effect on the result. Fortunately modifiers can be rearranged easily by clicking the convenient up and down arrow icons. For example, the image below shows [SubSurf](#) and [Mirror](#) modifiers that have switched places.



On the left, the *Mirror* modifier is the last item in the stack and the result looks like two surfaces. On the right, the *Subsurf* modifier is the last item in the stack and the result is a single merged surface.

Modifiers are calculated from top to bottom in the stack. In this example, the desired result (on right) is achieved by first mirroring the object, and then calculating the subdivision surface.

### Example

[Download example file.](#)

## 2.4.3 Modify

### Data Transfer Modifier

The **Data Transfer** modifier transfers several types of data from one mesh to another. Data types include vertex groups, UV layers, vertex colors, custom normals. . .

Transfer works by generating a mapping between source mesh’s items (vertices, edges, etc.) and destination ones, either on a one-to-one basis, or mapping several source items to a single destination one - interpolated mapping.

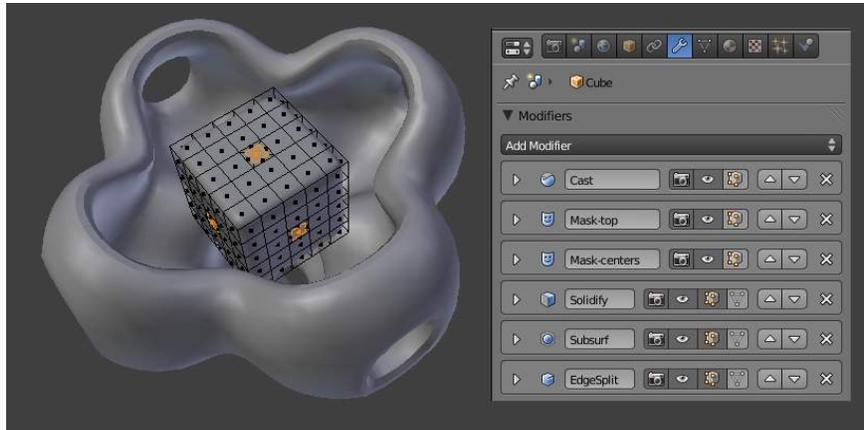


Fig. 2.581: In this example a simple subdivided cube has been transformed into a rather complex object using a stack of modifiers.

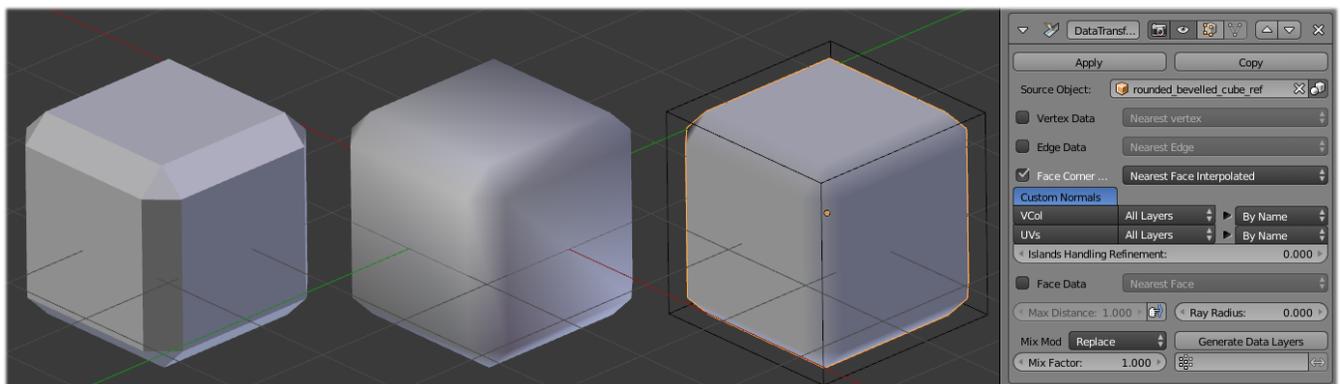


Fig. 2.582: From left to right, a flat-shaded beveled cube, a smooth-shaded beveled cube, and an autosmooth-shaded beveled cube copying its normals from the reference, flat-shaded cube shown as wire here, to achieve the ‘fake round corners’ effect.

## Options

**Source Object** Mesh object to copy data from.

If the button to the right of the field is unset, source and destination geometries are considered in global space when generating the mapping, otherwise they are evaluated in local space (i.e. as if both object's centers were at the same place).

**Max Distance** When the icon “finger” button to the right is enabled, this is the maximum distance between source and destination to get a successful mapping. If a destination item cannot find a source one within that range, then it will get no transferred data.

This allows to transfer a small sub-detailed mesh onto a more complete one (e.g. from a “hand” mesh towards a “full body” one).

**Ray Radius** For ray-casting-based mapping methods, the radius of the cast rays. Especially important for 1D and 2D items (i.e. vertices and edges), without some width there would be nearly no ray-casting matches...

**Mix Mode** Controls how destination data are affected:

**All** Replaces everything in destination (note that *Mix Factor* is still used).

**Above Threshold** Only replaces destination value if it's above given threshold (*Mix Factor*). How that threshold is interpreted depends on data type, note that for boolean values this option fakes a logical AND.

**Below Threshold** Only replaces destination value if it's below given threshold (*Mix Factor*). How that threshold is interpreted depends on data type, note that for boolean values this option fakes a logical OR.

**Mix, Add, Subtract, Multiply** Apply that operation, using mix factor to control how much of source or destination value to use. Only available for a few types (vertex groups, vertex colors).

**Mix Factor** How much of the transferred data gets mixed into existing one (not supported by all data types).

**Vertex Group** Allows per-item fine control of the mix factor. Vertex group influence can be reverted using the small “arrow” button to the right.

**Generate Data Layers** This modifier cannot generate needed data layers itself. Once the set of source data to transfer is selected, this button shall be used to generate matching destination layers.

**Selection of Data to Transfer** To keep the size of the modifier reasonable, the kind of items to be affected must be selected first (vertices, edges, face corners and/or faces).

**Mapping Type** How is generated the mapping between those source and destination items. Each type has its own options, see *Geometry Mapping* below for details.

**Data Types** The left column of toggle buttons, to select which data types to transfer.

**Multi-layers Data Types Options** In those cases (vertex groups, vertex colors, UVs), one can select which source layers to transfer (usually, either all of them, or a single specified one), and how to affect destination (either by matching names, matching order/position, or, if a single source is selected, by specifying manually destination layer).

**Islands Handling Refinement** This setting only affects UV transfer currently. It allows to avoid a given destination face to get UV coordinates from different source UV islands. Keeping it at 0.0 means no island handling at all. Typically, small values like 0.02 are enough to get good results, but if you are mapping from a very high poly source towards a very low poly destination, you may have to raise it quite significantly.

## Usage

First key thing to keep in mind when using this modifier is that **it will not create destination data layers**. *Generate Data Layers* button shall always be used for this purpose, once set of source data to transfer is selected. It should also be well



Fig. 2.583: Data Transfer modifier.

understood that creating those data layers on destination mesh is **not** part of the modifier stack, which means e.g. that they will remain even once the modifier is deleted, or if source data selection is modified.

**Geometry Mapping** Geometry mapping is the process by which a given destination vertex/edge/... knows **which part** of the source mesh to get its data from. It is crucial to understand this topic well to get good results with this modifier.

**Topology** The simplest option, expects both meshes to have identical number of items, and match them by order (indices). Useful e.g. between meshes that were identical copies, and got deformed differently.

**One-To-One Mappings** Those always select only one source item for each destination one, often based on shortest distance.

#### Vertices

**Nearest Vertex** Uses source's nearest vertex.

**Nearest Edge Vertex** Uses source's nearest vertex of source's nearest edge.

**Nearest Face Vertex** Uses source's nearest vertex of source's nearest face.

#### Edges

**Nearest Vertices** Uses source's edge which vertices are nearest from destination edge's vertices.

**Nearest Edge** Uses source's nearest edge (using edge's midpoints).

**Nearest Face Edge** Uses source's nearest edge of source's nearest face (using edge's midpoints).

**Face Corners** A face corner is not a real item by itself, it's some kind of split vertex attached to a specific face. Hence both vertex (location) and face (normal, ...) aspects are used to match them together.

**Nearest Corner and Best Matching Normal** Uses source's corner having the most similar **split** normal with destination one, from those sharing the nearest source's vertex.

**Nearest Corner and Best Matching Face Normal** Uses source's corner having the most similar **face** normal with destination one, from those sharing the nearest source's vertex.

**Nearest Corner of Nearest Face** Uses source's nearest corner of source's nearest face.

#### Faces

**Nearest Face** Uses source's nearest face.

**Best Normal-Matching:** Uses source's face which normal is most similar with destination one.

**Interpolated Mappings** Those use several source items for each destination one, interpolating their data during the transfer.

#### Vertices

**Nearest Edge Interpolated** Uses nearest point on nearest source's edge, interpolates data from both source edge's vertices.

**Nearest Face Interpolated** Uses nearest point on nearest source's face, interpolates data from all that source face's vertices.

**Projected Face Interpolated** Uses point of face on source hit by projection of destination vertex along its own normal, interpolates data from all that source face's vertices.

#### Edges

**Projected Edge Interpolated** This is a sampling process. Several rays are cast from along the destination's edge (interpolating both edge's vertex normals), and if enough of them hit a source's edge, all hit source edges' data are interpolated into destination one.

**Face Corners** A face corner is not a real item by itself, it's some kind of split vertex attached to a specific face. Hence both vertex (location) and face (normal, ...) aspects are used to match them together.

**Nearest Face Interpolated** Uses nearest point of nearest source's face, interpolates data from all that source face's corners.

**Projected Face Interpolated** Uses point of face on source hit by projection of destination corner along its own normal, interpolates data from all that source face's corners.

### Faces

**Projected Face Interpolated** This is a sampling process. Several rays are cast from the whole destination's face (along its own normal), and if enough of them hit a source's face, all hit source faces' data are interpolated into destination one.

## Mesh Cache Modifier

The **Mesh Cache** modifier is used so animated mesh data can be applied to a mesh and played back, deforming the mesh.

This works in a similar way to shape-keys but is aimed at playing back external files and is often used for interchange between applications.

When using this modifier, the vertex locations are overwritten.

### Options

**Format** The input file format (currently **MDD** and **PC2** are supported).

**File Path** Path to the cache file.

### Evaluation:

**Influence** Factor to adjust the influence of the modifiers deformation, useful for blending in/out from the cache data.

**Deform Mode** This setting defaults to 'Overwrite' which will replace the vertex locations with those in the cache file. However you may want to use shape-keys, for example, and mix them with the mesh-cache. In this case you can select the 'Deform' option which integrates deformations with the mesh-cache result.

---

**Note:** This feature is limited to making smaller, isolated edits and won't work for larger changes such as re-posing limbs

---

**Interpolation** None or Linear which will blend between frames; use linear when the frames in the cache file don't match up exactly with the frames in the blend file.

### Time Mapping:

**Time Mode** Select how time is calculated.

**Frame** Allows you to control the frames, which will ignore timing data in the file but is often useful since it gives simple control.

**Time** Evaluates time in seconds, taking into account timing information from the file (offset and frame-times).

**Factor** Evaluates the entire animation as a value from [0 - 1].

**Play Mode** Select how playback operates.

**Scene** Use the current frame from the scene to control playback.

**Frame Start** Play the cache starting from this frame.

**Frame Scale** Scale time by this factor (applied after the start value).

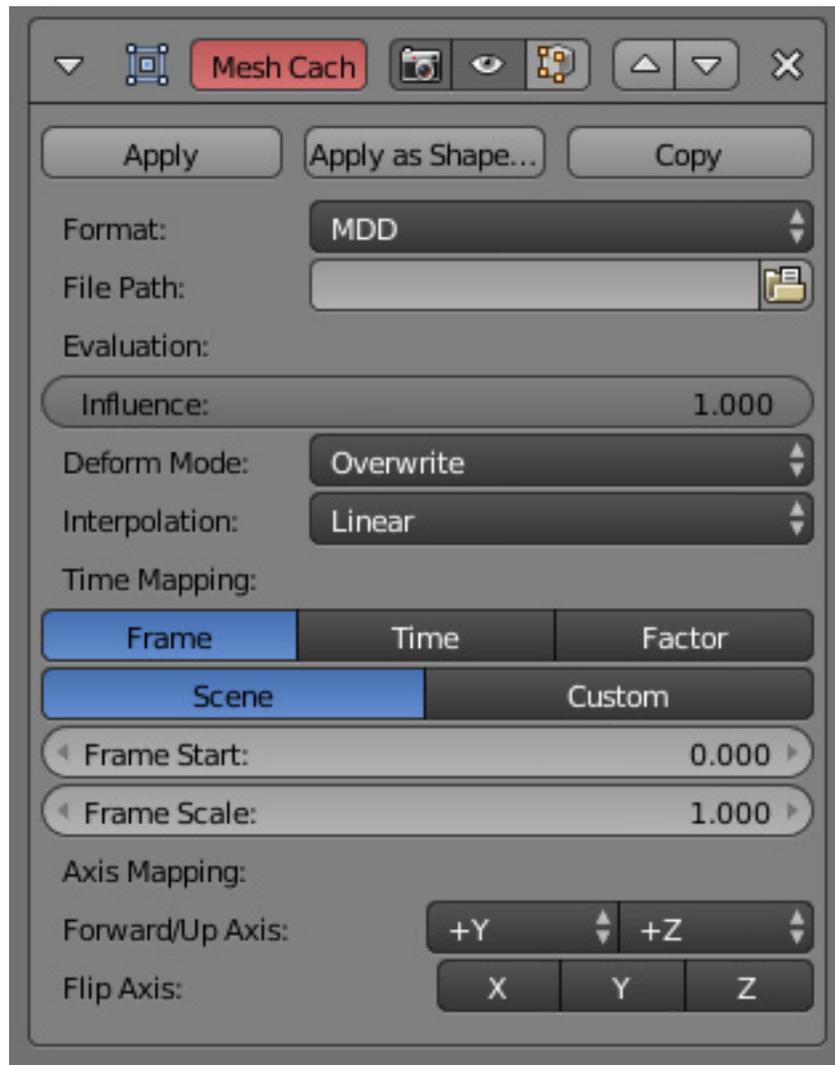


Fig. 2.584: Mesh Cache modifier

**Custom** Control animation timing manually.

**Evaluation Value** Property used for animation time, this gives more control of timing - typically this value will be animated.

### Axis Mapping:

**Forward/Up Axis** The axis for forward and up used in the source file. *Often different applications have different axis defaults for up/down front/back, so it's common to have to switch these on import.*

**Flip Axis** In rare cases you may also need to flip the coordinates on an axis.

### Hints

- Both MDD and PC2 depend on the vertex order on the mesh remaining unchanged; this is a limitation with the method used so take care not to add/remove vertices once this modifier is used.

### Normal Edit Modifier

The **Normal Edit** modifier affects (or generates) custom normals. It uses a few simple parametric methods to compute normals (quite useful in game development and architecture areas), and mixes back those generated normals with existing ones.

### Options

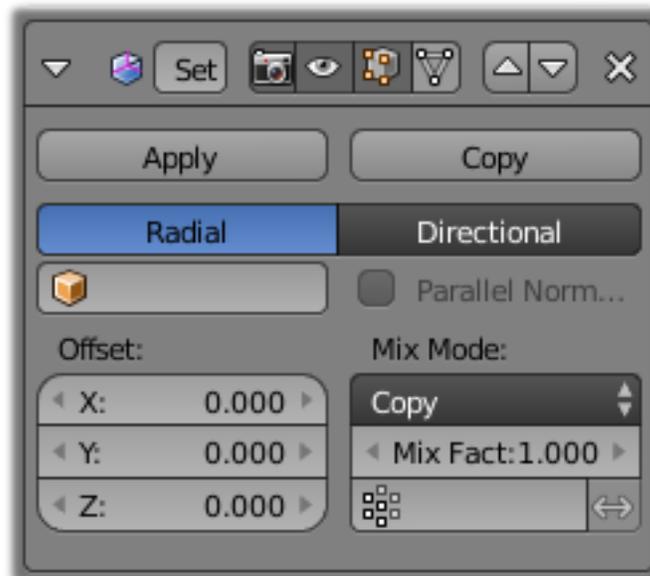


Fig. 2.585: Normal Edit modifier.

**Radial/Directional** The two modes currently available to generate normals.

Radial aligns normals with the (origin, vertex coordinates) vector, in other words all normals seems to radiate from the given center point, as if they were emitted from an ellipsoid surface.

Directional makes all normals point (converge) towards a given target object.

**Target Object** Uses this object’s center as reference point when generating normals.

Optional in *Radial* mode, mandatory in *Directional* one.

**Parallel Normals** Makes all normals parallel to the line between both objects’ centers, instead of converging towards target’s center.

Only relevant in *Directional* mode

**Offset** Gives modified object’s center an offset before using it to generate normals.

Only relevant in *Radial* mode if no *Target Object* is set, and in *Directional* mode when *Parallel Normals* is set.

**Mix Mode** How to affect existing normals with newly generated ones.

Note the *Multiply* option is **not** a cross product, but a mere component-by-component multiplication.

**Mix Factor** How much of the generated normals get mixed into existing ones.

**Vertex Group** Allows per-item fine control of the mix factor. Vertex group influence can be reverted using the small “arrow” button to the right.

## Usage

This modifier can be used to quickly generate radial normals for low-poly tree foliage, or “fix” shading of toon-like rendering by partially bending default normals. . .

The only mandatory prerequisite to use it is to enable *Auto Smooth* option in Mesh properties, *Normals* panel.

---

**Tip:** More complex normal manipulations can be achieved by copying normals from one mesh to another, see the [Data Transfer](#) modifier.

---

## UV Project Modifier

The **UV Project** Modifier acts like a slide projector. It emits a UV map from the negative Z-axis of a controller object (such as an *empty*), and applies it to the object as the “light” hits it. It can optionally override the objects face texture.

[Download an example](#)

## Options

**UV layer** Which UV layer to modify. Defaults to the active rendering layer.

**Image** The image associated with this modifier. Not required; you can just project a UV for use elsewhere. *Override Image*, below, defines how the image is used.

### Override Image

- When true, the Face Texture of all vertices on the mesh is replaced with the Image. This will cause the image to repeat, which is usually undesirable.
- When false, the modifier is limited to faces with the Image as their Face Texture.

**Projectors** Up to ten projector objects are supported. Each face will choose the closest and aligned projector with it’s surface normal. Projections emit from the negative Z-axis (i.e. straight down a camera or lamp). If the projector is a camera, the projection will adhere to its perspective/orthographic setting.

**Objects** Specify the projector Object

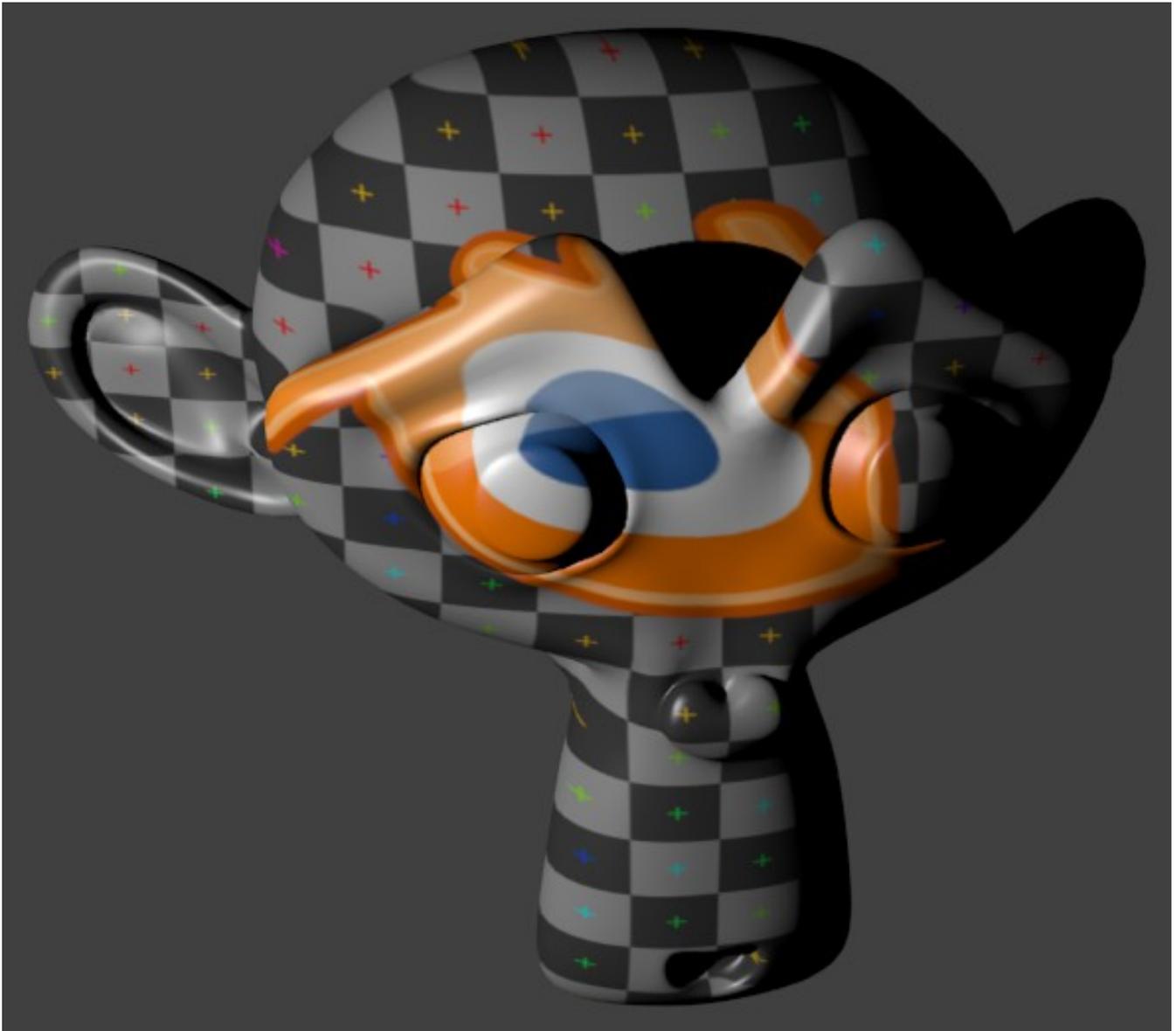
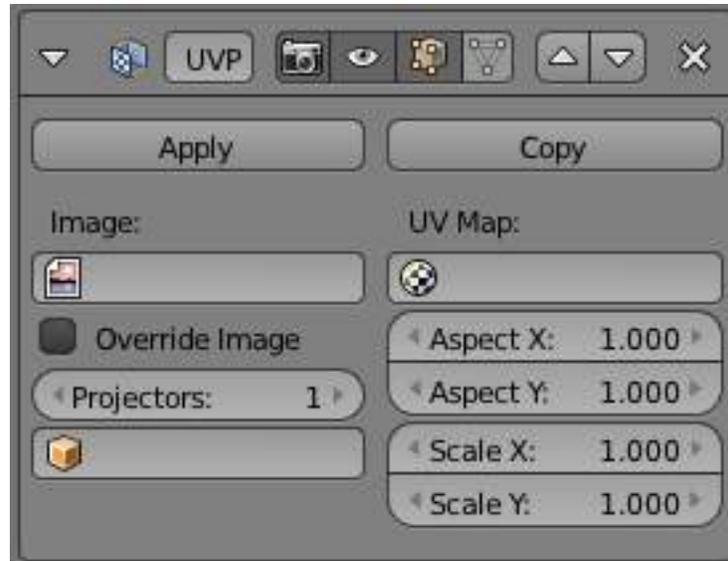


Fig. 2.586: Projecting the Blender logo onto Suzanne.



**Aspect X/Y and Scale X/Y** These allow simple manipulation of the image. Only apply when a camera is used as projector Object.

### Usage

**General** UV Project is great for making spotlights more diverse, and also for creating decals to break up repetition.

The modifier's Image property is not generally used - instead, a texture mapped to the UV layer that the modifier targets is added to the object's Material. This allows you to prevent the image from repeating by setting *Texture* → *Image Mapping* → *Extension* to *Clip*.

**Perspective Cameras** When using perspective cameras or spot lamps, you will likely want to enable the **UV Project** Material Option (available in the materials panel), This uses a different UV interpolation to prevent distortion.

---

**Note:** This option is not yet available for Cycles

---

### UV Warp Modifier

The **UV Warp** modifier uses two objects to define a transformation which is applied to the chosen UV coordinates.

Its purpose is to give you direct control over the object's UVs in the 3D View, allowing you to directly translate, rotate and scale existing UV coordinates using controller objects or bones.

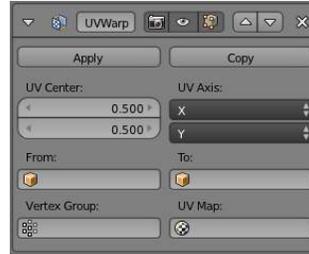
[Download an example](#)

### Options

**UV Center** The center point of the **UV map** to use when applying scale or rotation. With (0, 0) at the bottom left and (1, 1) at the top right. Defaults to (0.5, 0.5).

**UV Axis** The axes to use when mapping the 3D coordinates into 2D.

**From/To** The two objects used to define the transformation. See *Usage* below.



**Vertex Group** The vertex group can be used to scale the influence of the transformation per-vertex.

**UV Map** Which [UV map](#) to modify. Defaults to the active rendering layer.

### Usage

How the UVs are warped is determined by the difference between the transforms (location, rotation and scale) of the *from* and *to* objects.

If the *to* object has the same transforms as the *from* object, the UVs will not be changed.

Assuming the *UV Axis* of the modifier is X/Y and the scale of the objects are (1, 1, 1), if the *to* object is one unit away from the *from* object on the X-axis, the UVs will be transformed on the U-axis (horizontally) by one full UV space (the entire width of the image)

### Vertex Weight Modifiers

The Vertex Weight modifiers work on a vertex group of the affected object, by modifying its weights and/or which vertices belong to the vertex group.

**Warning:** These modifiers do implicit clamping of weight values in the standard  $[0.0, 1.0]$  range. All values below  $0.0$  will be set to  $0.0$ , and all values above  $1.0$  will be set to  $1.0$ .

There are currently three Vertex Weight modifiers:

- *Vertex Weight Edit Modifier*
- *Vertex Weight Mix Modifier*
- *Vertex Weight Proximity Modifier*

### Common Settings

The three Vertex Weight modifiers share a few settings, controlling their influence on the affected vertex group.

#### Global Influence

The overall influence of the modifier ( $0.0$  will leave the vertex group's weights untouched,  $1.0$  is standard influence).

**Warning:** Influence only affects weights, adding/removing of vertices to/from vertex group is not prevented by setting this value to  $0.0$ .

**Vertex Group Mask** An additional vertex group, the weights of which will be multiplied with the global influence value for each vertex. If a vertex is not in the masking vertex group, its weight will be not be affected.

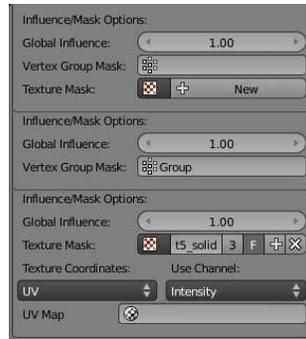


Fig. 2.587: The influence/masking part of Vertex Weight modifiers.

**Texture** An additional texture, the values of which will be multiplied with the global influence value for each vertex.

This is a standard texture [datablock](#) control. When set, it reveals other settings:

**Texture Coordinates** How the texture is mapped to the mesh.

**Local** Use local vertex coordinates.

**Global** Use vertex coordinates in global space.

**Object** Use vertex coordinates in another object's space.

**UV** Use a UV layer's coordinates.

**Use Channel** Which channel to use as weight factor source/

**Red/Green/Blue/Alpha** One of the color channels' values.

**Intensity** The average of the RGB channels (If RGB = 1.0, 0.0, 0.0, value is 0.33)

**Value** The highest value of the RGB channels (If RGB = 1.0, 0.0, 0.0, value is 1.0)

**Hue** Uses the hue value from the standard color wheel (e.g. blue has a higher hue value than yellow)

**Saturation** Uses the saturation value (e.g. pure red's value is 1.0, gray is 0.0)

---

**Note:** All of the channels above are gamma corrected, except for *Intensity*.

---

**Object** The object to be used as reference for *Object* mapping.

**UV Layer** The UV layer to be used for *UV* mapping.

**Viewing Modified Weights** You can view the modified weights in *Weight Paint* mode. This also implies that you'll have to disable the Vertex Weight modifiers if you want to see the original weights of the vertex group you are editing.

### Vertex Weight Edit Modifier

This modifier is intended to edit the weights of one vertex group.

The general process is the following, for each vertex:

- [Optional] It does the mapping, either through one of the predefined functions, or a custom mapping curve.
- It applies the influence factor, and optionally the vertex group or texture mask (0.0 means original weight, 1.0 means fully mapped weight).

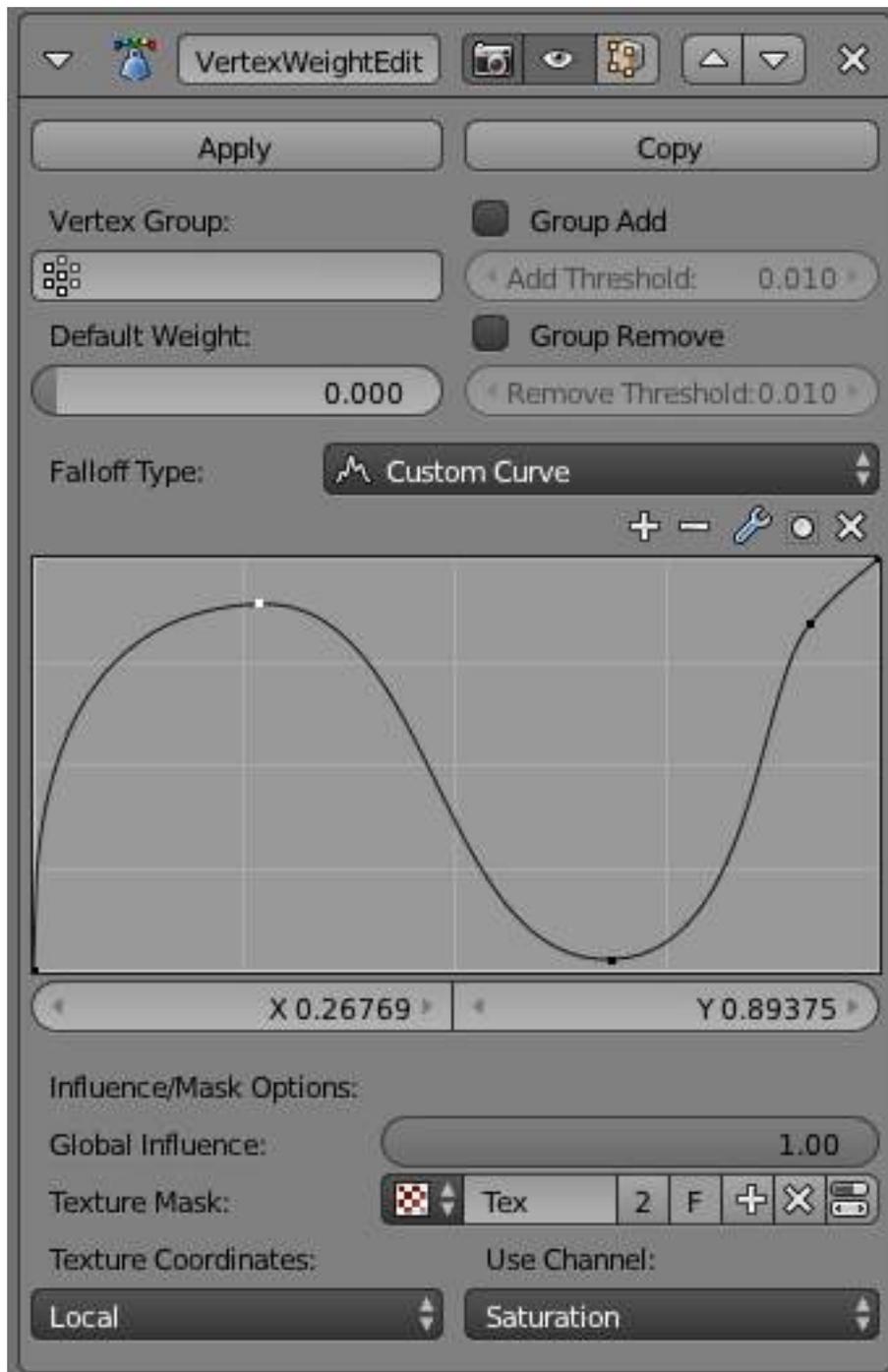


Fig. 2.588: The Vertex Weight Edit modifier panel.

- It applies back the weight to the vertex, and/or it might optionally remove the vertex from the group if its weight is below a given threshold, or add it if it's above a given threshold.

### Options

**Vertex Group** The vertex group to affect.

**Default Weight** The default weight to assign to all vertices not in the given vertex group.

**Falloff Type** Type of mapping:

**Linear** No mapping.

**Custom Curve** Allows the user to manually define the mapping using a curve.

**Sharp, Smooth, Root and Sphere** These are classical mapping functions, from spikiest to roundest.

**Random** Uses a random value for each vertex.

**Median Step** Creates binary weights (0 . 0 or 1 . 0), with 0 . 5 as cutting value.

**Group Add** Adds vertices with a final weight over *Add Threshold* to the vertex group.

**Group Remove** Removes vertices with a final weight below *Remove Threshold* from the vertex group.

### Vertex Weight Mix Modifier

This modifier mixes a second vertex group (or a simple value) into the affected vertex group, using different operations.

### Options

**Vertex Group A** The vertex group to affect.

**Default Weight A** The default weight to assign to all vertices not in the given vertex group.

**Vertex Group B** The second vertex group to mix into the affected one. Leave it empty if you only want to mix in a simple value.

**Default Weight B** The default weight to assign to all vertices not in the given second vertex group.

**Mix Mode** How the vertex group weights are affected by the other vertex group's weights.

**Replace weights** Replaces affected weights with the second group's weights.

**Add to weights** Adds the values of *Group B* to *Group A*.

**Subtract from weights** Subtracts the values of *Group B* from *Group A*.

**Multiply weights** Multiplies the values of *Group B* with *Group A*.

**Divide weights** Divides the values of *Group A* by *Group B*.

**Difference** Subtracts the smaller of the two values from the larger.

**Average** Adds the values together, then divides by 2.

**Mix Set** Choose which vertices will be affected.

**All vertices** Affects all vertices, disregarding the vertex groups content.

**Vertices from group A** Affects only vertices belonging to the affected vertex group.

**Vertices from group B** Affects only vertices belonging to the second vertex group.

**Vertices from one group** Affects only vertices belonging to at least one of the vertex groups.



Fig. 2.589: The Vertex Weight Mix modifier panel.

**Vertices from both groups** Affects only vertices belonging to both vertex groups.

**Warning:** When using *All vertices*, *Vertices from group B* or *Vertices from one group*, vertices might be added to the affected vertex group.

### Vertex Weight Proximity Modifier

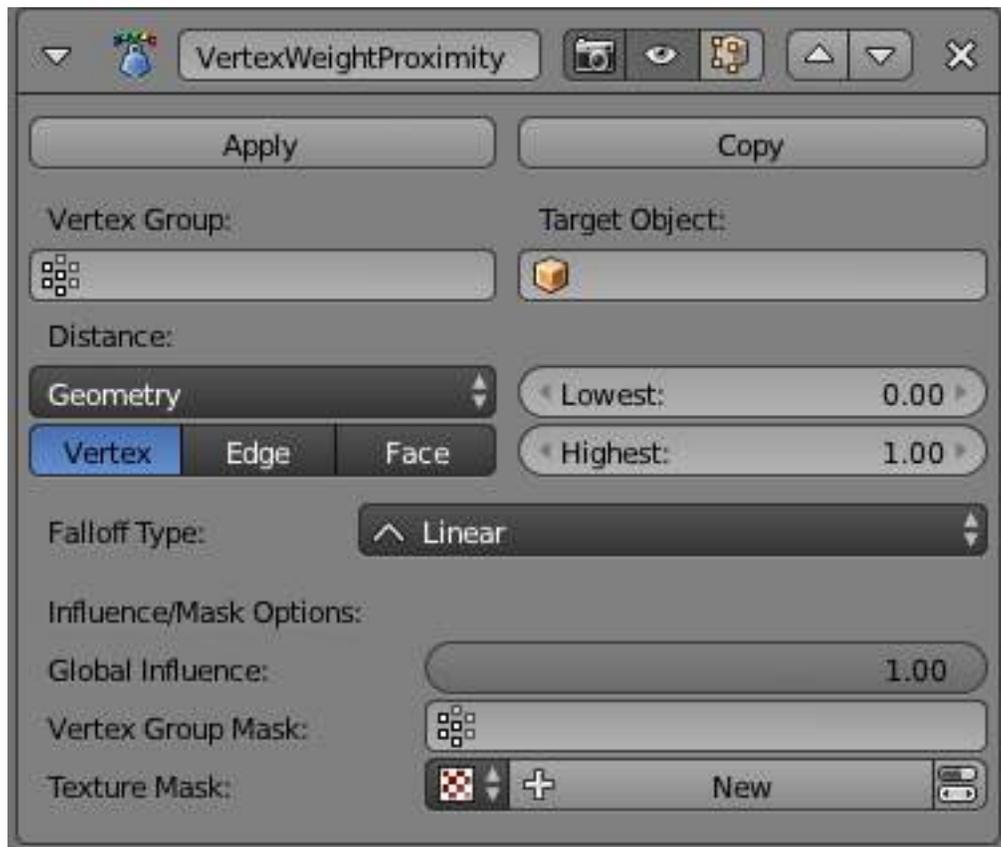


Fig. 2.590: The Vertex Weight Proximity modifier panel.

This modifier sets the weights of the given vertex group, based on the distance between the object (or its vertices), and another target object (or its geometry).

#### Options

**Vertex Group** The vertex group to affect.

**Target Object** The object from which to compute distances.

#### Proximity mode

**Object Distance** Use the distance between the modified mesh object and the target object as weight for all vertices in the affected vertex group.

**Geometry Distance** Use the distance between each vertex and the target object, or its geometry.

The *Geometry Distance* mode has three additional options, *Vertex*, *Edge* and *Face*. If you enable more than one of them, the shortest distance will be used. If the target object has no geometry (e.g. an empty or camera), it will use the location of the object itself.

**Vertex** This will set each vertex's weight from its distance to the nearest vertex of the target object.

**Edge** This will set each vertex's weight from its distance to the nearest edge of the target object.

**Face** This will set each vertex's weight from its distance to the nearest face of the target object.

**Lowest** Distance mapping to 0.0 weight.

**Highest** Distance mapping to 1.0 weight.

**Falloff Type** Some predefined mapping functions, see *Vertex Weight Edit Modifier*.

---

**Tip:** *Lowest* can be set above *Highest* to reverse the mapping.

---

## Examples

**Using Distance from a Target Object** As a first example, let's dynamically control a *Wave* modifier with a modified vertex group.

Add a *Grid* mesh, with many vertices (e.g. a **100×100** vertices), and 10 BU side-length. Switch to *Edit* mode (Tab), and in the *Object Data* properties, *Vertex Groups* panel, add a vertex group. Assign to it all your mesh's vertices (with e.g. a 1.0 weight). Go back to *Object* mode.

Then, go to the *Modifiers* properties, and add a *Vertex Weight Proximity* modifier. Set the mode to *Object Distance*. Select your vertex group, and the target object you want (here I used the lamp).

You will likely have to adjust the linear mapping of the weights produced by the *Vertex Weight Proximity* modifier. To do so, edit *Lowest Dist* and *Highest Dist* so that the first corresponds to the distance between your target object and the vertices you want to have lowest weight, and similarly with the second and highest weight...

Now add a *Wave* modifier, set it to your liking, and use the same vertex group to control it.

Animate your target object, making it move over the grid. As you can see, the waves are only visible around the reference object! Note that you can insert a *Vertex Weight Edit* modifier before the *Wave* one, and use its *Custom Curve* mapping to get larger/narrower "wave influence's slopes".

The Blender file, TEST\_1 scene.

**Using Distance from a Target Object's Geometry** We're going to illustrate this with a *Displace* modifier.

Add a **10×10 BU 100×100** vertices grid, and in *Edit* mode, add to it a vertex group containing all of its vertices, as above. You can even further sub-divide it with a first *Subsurf* modifier.

Now add a curve circle, and place it 0.25 BU above the grid. Scale it up a bit (e.g. 4.0).

Back to the grid object, add to it a *Vertex Weight Proximity* modifier, in *Geometry Distance* mode. Enable *Edge* (if you use *Vertex* only, and your curve has a low U definition, you would get wavy patterns, see (*Wavy patterns*)).

Table  
2.8:  
Distance  
from  
vertices.



Set the *Lowest Dist* to 0.2, and the *Highest Dist* to 2.0, to map back the computed distances into the regular weight range.

Add a third *Displace* modifier and affect it the texture you like. Now, we want the vertices of the grid nearest to the curve circle to remain undisplaced. As they will get weights near zero, this means that you have to set the *Midlevel* of the displace to 0.0. Make it use our affected vertex group, and that's it! Your nice mountains just shrink to a flat plane near the curve circle.

As in the previous example, you can insert a *Vertex Weight Edit* modifier before the *Displace* one, and play with the *Custom Curve* mapping to get a larger/narrower "valley"...

Table  
2.9:  
Convex-  
type  
mapping  
curve.

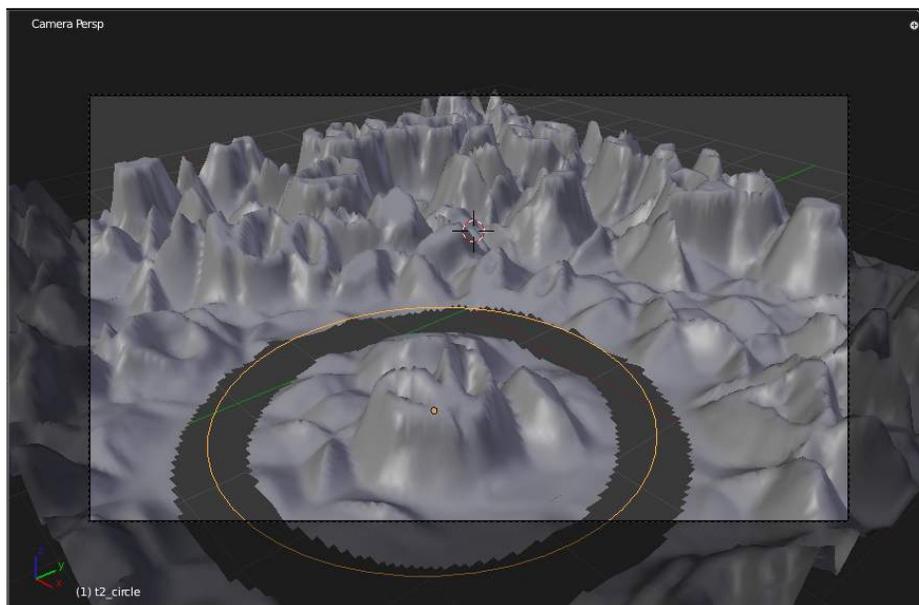
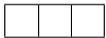


Fig. 2.601: Vertices with a computed weight below 0.1 removed from the vertex group.

You can also add a fifth *Mask* modifier, and enable *Vertex Weight Edit*'s *Group Remove* option, with a *Remove Threshold* of 0.1, to see the bottom of your valley disappear.

The Blender file, TEST\_2 scene.

**Using a Texture and the Mapping Curve** Here we are going to create a sort of strange alien wave (yes, another example with the *Wave* modifier... but it's a highly visual one; it's easy to see the vertex group effects on it...).

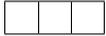
So as above, add a **100×100** grid. This time, add a vertex group, but without assigning any vertex to it - we'll do this dynamically.

Add a first *Vertex Weight Mix* modifier, set the *Vertex Group A* field with a *Default Weight A* of 0.0, and set *Default Weight B* to 1.0. Leave the *Mix Mode* to *Replace weights*, and select *All vertices* as *Mix Set*. This way, all vertices are affected. As none are in the affected vertex group, they all have a default weight of 0.0, which is replaced by the second default weight (1.0). And all those vertices are also added to the affected vertex group.

Now, select or create a masking texture - here I chose a default *Magic* one. The values of this texture will control how much of the "second weight" (1.0) replaces the "first weight" (0.0)... In other words, they are taken as weight values!

You can then select which texture coordinates and channel to use. Leave the mapping to the default *Local* option, and play with the various channels...

Table  
2.10:  
Using  
Saturation.

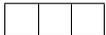


Don't forget to add a *Wave* modifier, and select your vertex group in it!

You can use the weights created this way directly, but if you want to play with the curve mapping, you must add the famous *Vertex Weight Edit* modifier, and enable its *Custom Curve* mapping.

By default, it's a one-to-one linear mapping - in other words, it does nothing! Change it to something like in (*A customized mapping curve*), which maps  $[0.0, 0.5]$  to  $[0.0, 0.25]$  and  $[0.5, 1.0]$  to  $[0.75, 1.0]$ , thus producing nearly only weights below  $0.25$ , and above  $0.75$ : this creates great "walls" in the waves...

Table  
2.11:  
Custom  
Mapping  
enabled.



The Blender file, TEST\_4 scene.

## See Also

- The [Development](#) page.

## 2.4.4 Generate

### Array Modifier

The Array modifier creates an array of copies of the base object, with each copy being offset from the previous one in any of a number of possible ways. Vertices in adjacent copies can be merged if they are nearby, allowing smooth [subsurf](#) frameworks to be generated.

This modifier can be useful when combined with tileable meshes for quickly developing large scenes. It is also useful for creating complex repetitive shapes.

Multiple array modifiers may be active for an object at the same time (e.g. to create complex three dimensional constructs).

### Options

**Fit Type menu** Controls how the length of the array is determined. There are three choices, activating respectively the display of the *Curve*, *Length* or *Count* settings explained below:

**Fit Curve** Generates enough copies to fit within the length of the curve object specified in *Curve*.

**Fit Length** Generates enough copies to fit within the fixed length given by *Length*.

**Fixed Count** Generates the number of copies specified in *Count*.

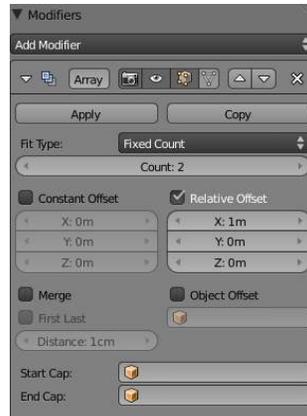


Fig. 2.614: Array modifier.

**Curve** The Curve object to use for *Fit Curve*.

**Length** The length to use for *Fit Length*.

**Count** The number of duplicates to use for *Fixed Count*.

---

**Note:**

- Both *Fit Curve* and *Fit Length* use the local coordinate system size of the base object, which means that scaling the base object in *Object* mode will not change the number of copies generated by the *Array* modifier.
  - *Fit Length* uses the local coordinate system length of the curve, which means that scaling the curve in *Object* mode will not change the number of copies generated by the *Array* modifier.
  - Applying the scale with `Ctrl-A` can be useful for each one.
- 

**Constant Offset, X, Y, Z** Adds a constant translation component to the duplicate object's offset. X, Y and Z constant components can be specified.

**Relative Offset, X, Y, Z**



Fig. 2.615: Relative offset example.

Adds a translation equal to the object's bounding box size along each axis, multiplied by a scaling factor, to the offset. X, Y and Z scaling factors can be specified.

**Object Offset**

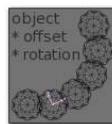


Fig. 2.616: Object offset example.

Adds a transformation taken from an object (relative to the current object) to the offset. It is good practice to use an Empty object centered or near to the initial object. E.g. by rotating this Empty a circle or helix of objects can be created.

**Merge** If enabled, vertices in each copy will be merged with vertices in the next copy that are within the given *Distance*.

**First Last** If enabled **and** *Merge* is enabled, vertices in the first copy will be merged with vertices in the last copy (this is useful for circular objects).

Table 2.12: First Last merge example.

Subsurf discontinuity caused by not merging vertices between first and last copies ( <i>First Last</i> off).	Subsurf discontinuity eliminated by merging vertices between first and last copies ( <i>First Last</i> on).
--	---

**Distance** Controls the merge distance for *Merge*.

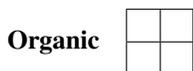
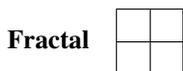
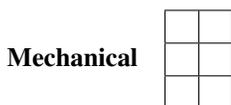
**Start cap** The mesh object to be used as a start cap. A single copy of this object will be placed at the “beginning” of the array - in fact, as if it was in position  $-1$ , i.e. one “array step” before the first “regular” array copy. Of course, if *Merge* is activated, and the *Start cap* is near enough to the first copy, they will be merged.

**End cap** The mesh object to be used as an end cap. A single copy of this object will be placed at the “end” of the array - in fact, as if it was in position  $n+1$ , i.e. one “array step” after the last “regular” array copy. And as *Start cap*, it can be merged with the last copy...

## Hints

**Offset Calculation** The transformation applied from one copy to the next is calculated as the sum of the three different components (*Relative*, *Constant* and *Object*), all of which can be enabled/disabled independently of the others. This allows, for example, a relative offset of  $1.0, 0.0, 0.0$  and a constant offset of  $0.1, 0.0, 0.0$ , giving an array of objects neatly spaced along the X axis with a constant  $0.1$  units between them, whatever the original object’s size.

## Examples



**Fig 01** Subsurfed cube array with 1 object offset, 4 cubes and a high vertex merge setting to give the effect of skinning.

**Fig 02** A double spiral created with two array modifiers and one subsurf modifier applied to a cube. As above, the vertex merge threshold is set very high to give the effect of skinning. [Sample blend file](#)

**Fig 03** A tentacle created with an Array modifier followed by a Curve modifier. The segment in the foreground is the base mesh for the tentacle; the tentacle is capped by two specially-modeled objects deformed by the same Curve object as the main part of the tentacle. [Sample blend file](#)

## Tutorials

- [Neal Hirsig's Array Modifier Screencast on Vimeo](#)
- [Creating A Double Helix With Modifiers](#)

The 'Double Helix' tutorial explains the Array modifier. It is for an old Blender Version (2.44) but except for the keyboard shortcuts it is still valid.

## Bevel Modifier

The Bevel modifier adds the ability to bevel the edges of the mesh it is applied to, allowing control of how and where the bevel is applied to the mesh.

The Bevel modifier is a non-destructive alternative to the [Bevel Operation](#) in edit mode.



The images above show the side views of a plain (unbeveled) cube and a beveled one.

## Options

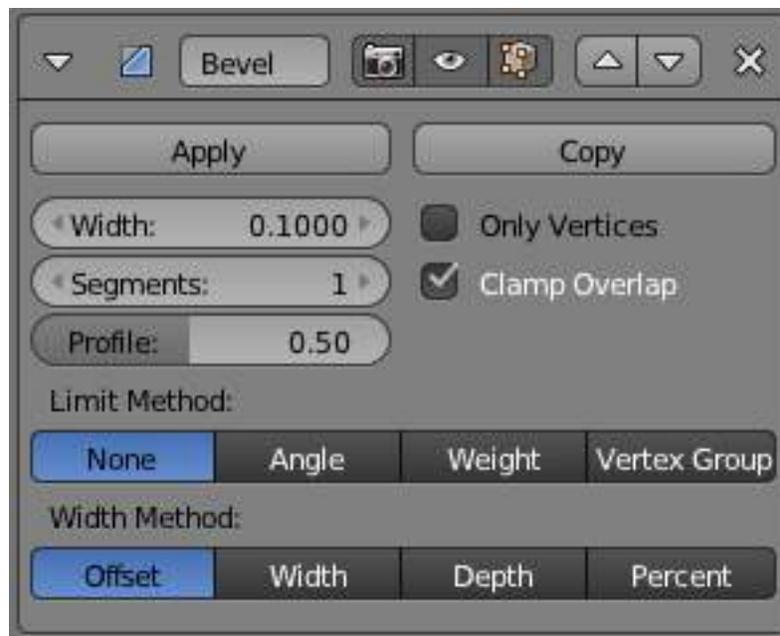


Fig. 2.643: Bevel modifier panel.

**Width** The size of the bevel affect. See *Width Method* below.

**Segments** The number of edge loops added along the bevel's face.

**Profile** The shape of the bevel, from concave to convex - has no effect if *Segments* is less than 2.

**Material** The index of the material slot to use for the bevel. When set to  $-1$ , the material of the nearest original face will be used.

**Only Vertices** When enabled, only the areas near vertices are beveled; the edges are left unbeveled.

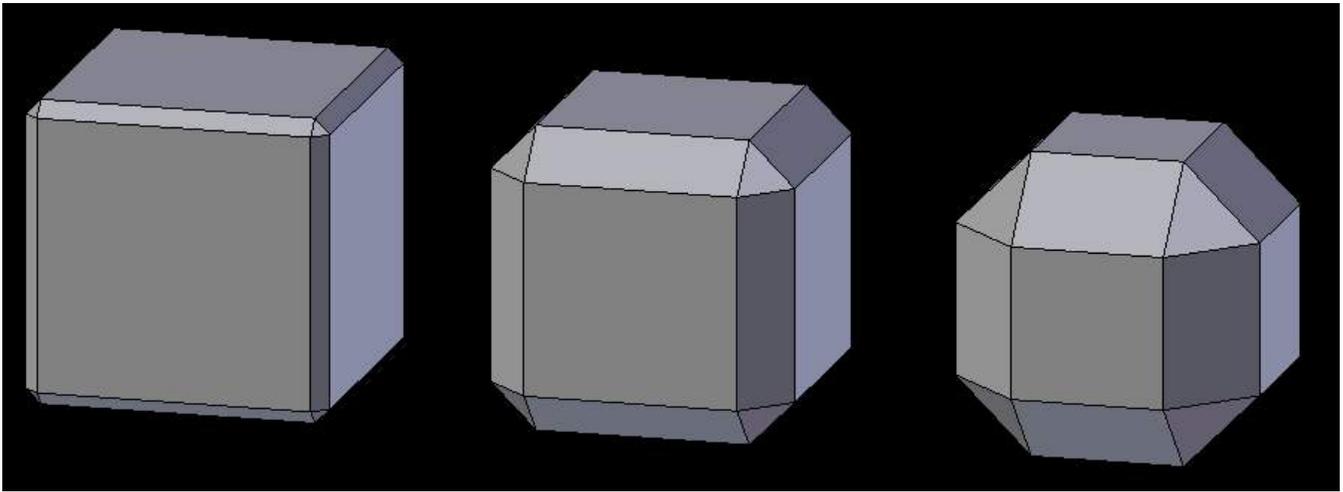


Fig. 2.644: Three Cubes with 0.1, 0.3 and 0.5 bevel Widths.

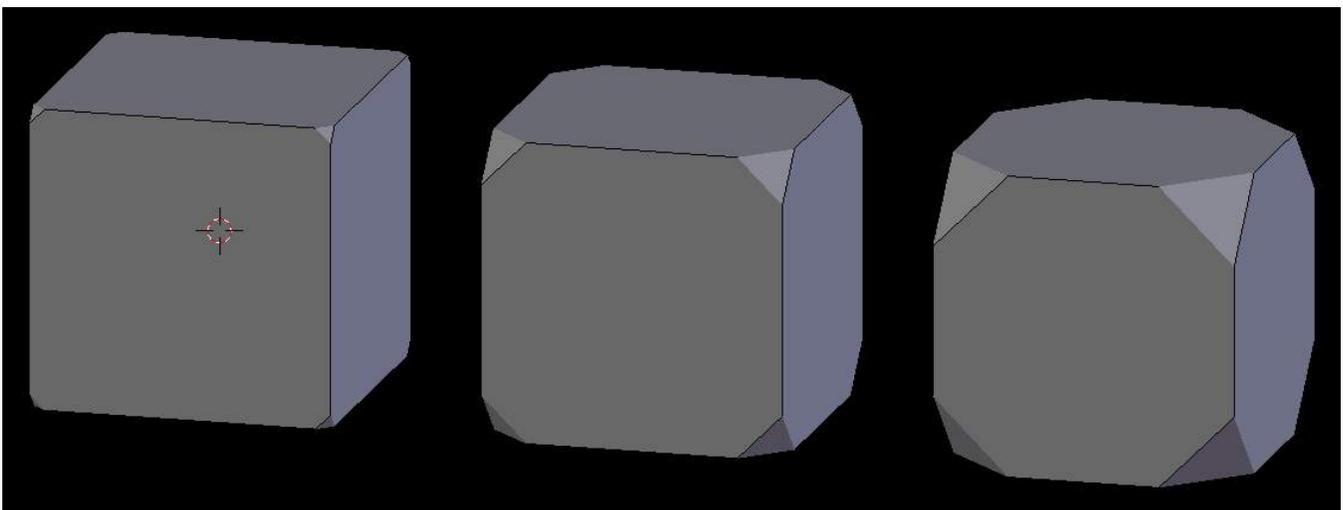


Fig. 2.645: Three cubes with 0.1, 0.3 and 0.5' bevel Widths, with Only Vertices option enabled.

**Clamp Overlap** When enabled, the width of each beveled edge will be limited such that they cannot intersect each other. Edges that are far apart will still bevel with the full width, only edges too close to each other are affected.

**Limit Method** Used to control where a bevel is applied to the mesh.

**None** No limit, all edges will be beveled.

**Angle** Only edges where the adjacent faces form an angle smaller than the defined threshold will be beveled. Intended to allow you to bevel only the sharp edges of an object without affecting its smooth surfaces.

**Weight** Use each edge's bevel weight to determine the width of the bevel. When the bevel weight is 0.0, no bevel is applied. See [here](#) about adjusting bevel weights.

**Vertex Group** Use weights from a vertex group to determine the width of the bevel. When the vertex weight is 0.0, no bevel is applied. An edge is only beveled if both of its vertices are in the vertex group. See [here](#) about adjusting vertex group weights.

**Width Method** Used to control how the *Width* is measured.

**Offset** Amount is offset of new edges from original.

**Width** Amount is width of new face.

**Depth** Amount is perpendicular distance from original edge to bevel face.

**Percent** Amount is percent of adjacent edge length.

## Boolean Modifier

### Introduction

The Boolean modifier performs operations on meshes that are otherwise too complex to achieve with as few steps by editing meshes manually, meaning you can achieve results with little effort to make mesh operations like Unions, Differences and Intersections.

The Boolean modifier uses one of three Boolean operations (*Difference* (negation), *Union* (conjunction), and *Intersect* (disjunction)) to create a single compound object out of two mesh objects.

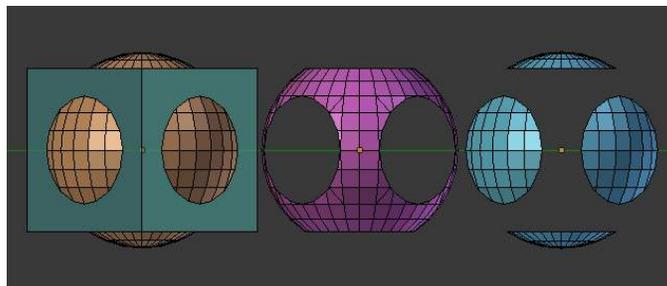


Fig. 2.646: Fig. 1 - The Union, Intersection and Difference between a Cube and a UV Sphere, with the modifier applied to the Sphere and using the cube as target. (Detail, Union is using Ngons).

### Description

The Boolean modifier can only be used on mesh objects.

It performs one of the three Boolean Operations for the faces of open or closed volumes that creates a complete topology in the faces it's being used. This means that this modifier will only work properly for the intersection of faces of the two meshes that will result in another closed loop of edges (filled with faces), creating a new resulting face topology.

The Boolean modifier is non-destructive for the target; it uses the topology of the target to make the calculations, but you will still have the target in the scene. In normal conditions, using face normals pointed outside, when you apply the Boolean modifier operation, the modified mesh will receive changes in topology, and you will have to move or hide the target to see the resulting mesh. The only exception is when you are using inverted normals; in this case, depending on the calculations, you will also change the topology of the target. You can see one example of a target being modified in the *Materials* section in this page, see Fig. 7 and 8.

The results of the mesh operation will only be shown in Object Mode of the 3D View.

---

**Note:**

- The Boolean modifier works with open and closed volumes.
  - The Boolean modifier doesn't work on edges without faces.
  - The target topology determines the new topology of the modified mesh.
  - The face normals are taken into account for the calculations.
  - Whether faces are marked for smooth or flat for shading doesn't affect the calculations of this modifier. (See Fig. 28 and 29.)
  - The line at which this modifier is calculated is delimited by the first tangential contact between faces of the modified mesh and target.
- 

**Tip:** This is a dynamic real-time modifier!

If you have marked your Objects to show the Edges (in Properties Window, Object context, Display panel, enable *Wire*), you will see the Edge creation process while you're moving your Objects. Depending on your mesh topology, you can also enable X-Ray and Transparency and see the topology being created in real time.

---

## Usage

Using the default Blender install, with the desired mesh Object selected, go to the *Properties Window* which is located at the right of your Blender Screen, below the Outliner. Click on the Modifiers Context, which is represented by a wrench (see Fig. 2 - The Boolean modifier; the wrench is highlighted in blue). Then, click on the *Add Modifier* Button and Blender will show you a list of all of the available Modifiers. The Boolean modifier is located on the third row in the *Generate* Column.

You can also click on the *Add Modifier* Button and use **N** to add the Boolean modifier, or use Blender search with the shortcut **Spacebar** and type "Add Modifier", click on *Add Modifier* and press **N**.

When you add the Boolean modifier for an Object, Blender will need a second Object to perform the operation. You can use open or closed Meshes, as long as they have faces for calculations.

You can add one or more modifiers of this type for an Object but you can only apply one operation for the Boolean modifier at a time.

## Options

### Operations

**Operation** Which boolean operation will be used.

**Difference** The modified mesh is subtracted from the target mesh.

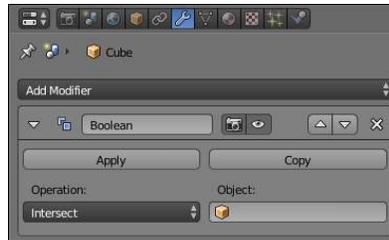


Fig. 2.647: Fig. 2 - The Boolean Modifier

- If the target Mesh has inverted normals, Blender will Intersect the modified mesh.
- If the modified Mesh has inverted normals, Blender will add both meshes (Union).
- If both Meshes use inverted normals, Blender will Intersect the target Mesh.

**Union** The target mesh is added to the modified mesh.

- If the target Mesh has inverted normals, Blender will Intersect the target Mesh.
- If the modified Mesh has inverted normals, Blender will subtract the target Mesh.
- If both Meshes use inverted normals, Blender will Intersect the modified Mesh.

**Intersect** The target mesh is subtracted from the modified mesh.

- If the target Mesh has inverted normals, Blender will subtract the target Mesh.
- If the modified Mesh has inverted normals, Blender will intersect the target Mesh.
- If both Meshes use inverted normals, Blender will add both meshes (Union).

**Object** The name of the target mesh object.

## Materials

The Boolean modifier preserves the Materials of the participant Meshes, including their basic textures and mappings, and the modified mesh will receive its first active material index assigned to its new topology (the first active material).

The only exception is the difference operation when the normals of the target and modified mesh are inverted (Fig 7 and 8). In this case, Blender will project the textures in an inverted direction over the target using the center contact of the meshes as a pivot and the resulting mesh will have the modified mesh subtracted from the target. For complex target meshes in some particular cases, you may have to reassign materials to faces because Blender will use the possible projection, and this may result in a sub-optimal texture assignment.

Below, some examples are shown to exemplify how materials work with the Boolean modifier; we took the cube as the modified mesh, and the icosphere as the target with one material (white). We added four different indexes to one of the faces of the cube, leaving another basic material in the other faces. Fig. 3 shows how the Boolean modifier interacts with the materials. Figs. 4, 5 and 6 show three different Boolean operations applied to the modified mesh. The meshes used have normals pointed outwards (Normal meshes). See their captions for more information.

In our last examples (Figs. 7 and 8) of how the Boolean modifier works with Materials, we have inverted normals for both the target (Icosphere) and modified mesh (Cube). As we said before, this is an exception rather than the rule. As you can see, the target received the materials of the modified mesh.

**UV Mappings** When you map UV Images to your target, Blender will add a map for each of the faces of the target. When you apply the Boolean modifier, Blender will follow the UV maps already assigned to the faces of the target topology that will

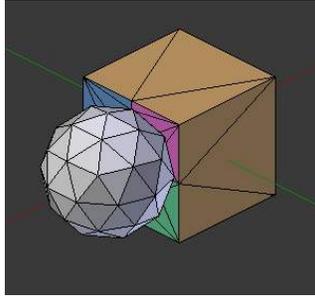


Fig. 2.648: Fig. 3 - Cube with Multi-Material Mesh (modified) and Icosphere (target) with basic Material

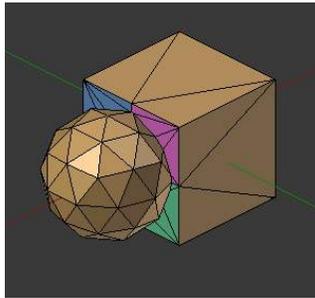


Fig. 2.649: Fig. 4 - Union - The first active Material of the Cube is added to the new topology; other materials remain in the old topology

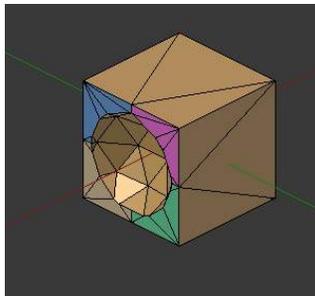


Fig. 2.650: Fig. 5 - Difference - The Icosphere was subtracted from the Cube; the new topology has received the first active Material of the Cube

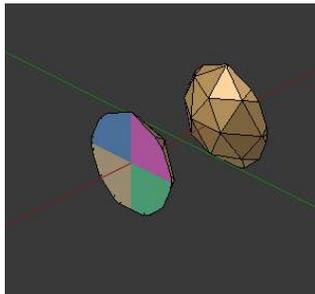


Fig. 2.651: Fig. 6 - Intersect - The resulting Mesh was copied and rotated 180- - You can see the first active material of the cube in the back face (new topology); the front face received the 4 basic materials of the cube.

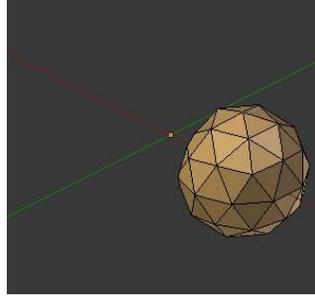


Fig. 2.652: Fig. 7 - Front of the target with the modified mesh materials

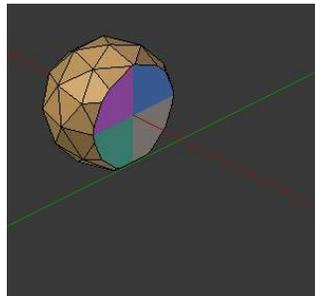


Fig. 2.653: Fig. 8 - Back of the target with the modified mesh materials

be the result of the operation on the modified mesh. Blender will also use the same image mapped to the target faces in the modified mesh.

**Warning:** Depending on the way you have assigned textures to the faces during the UV unwrap, and the complexity of your meshes, the boolean operation may generate imperfect UVs for the new faces.

Below we have four Images, a UV sphere mapped with a test grid tinted blue and the other face tinted in purple, one face of the cube tinted in a light orange and the other faces using the normal test grid. Fig. 9 shows the operation at the start (difference), and on the right (Fig. 10), the resulting mesh. In Figs. 11 and 12 we show the unwrap in the Blender UV/Image Editor Window.

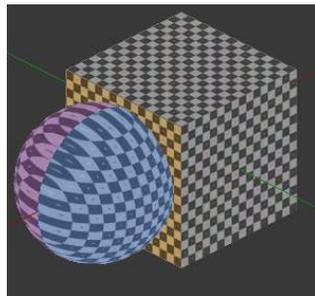


Fig. 2.654: Fig. 9 - A UV Sphere and a Cube with different UV Maps

### Other Modifiers

The Boolean modifier calculation is performed using the target modified mesh topology and dimensions. Other modifiers added to the modified mesh are bypassed. It means that if a target is using another modifier, like subsurf, the resulting topology for

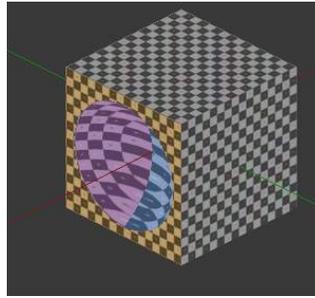


Fig. 2.655: Fig. 10 - Difference operation applied

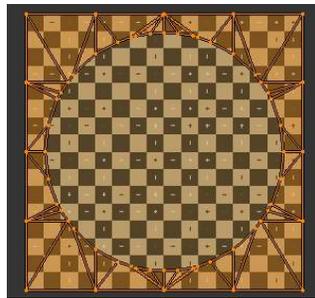


Fig. 2.656: Fig. 11 - Faces of the modified mesh mapped

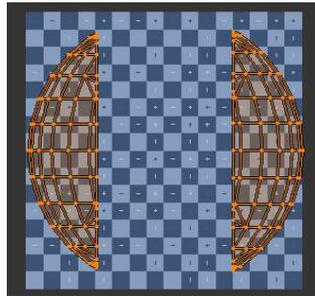


Fig. 2.657: Fig. 12 - New topology mapped and UV faces assigned; we have another image assigned to the purple tinted faces.



Fig. 2.658: Fig. 13 - Boolean modifier with error message

the modified mesh will take into account the subsurf of the target; but for the modified mesh, the basic topology is used anyway (see examples).

If you add subsurf to the modified mesh with a Boolean modifier, Blender will visually add the subsurf for the modified mesh, but not for its calculations; it will only take into account its basic mesh topology. If you want to have a subsurf added to the modified mesh, you have to apply the subsurf to the Boolean modified mesh before applying the Boolean operation.

The Boolean modifier can be added together with other modifiers in the modified mesh, but depending on the modifier, the calculations can't be done and/or the modifier cannot execute. When the modifier cannot execute, it will show the message "Cannot execute boolean operation" (see Fig. 13), and when the modifier cannot be applied to the mesh, Blender will show the message "Modifier is disabled, Skipping Apply.". In this case, you either have to remove some modifiers or apply the necessary ones.

The most common case is when you add or copy a Boolean modifier to use the modified mesh in conjunction with another target later; Blender will place the warning in the subsequent Boolean modifiers in the stack depending on the operation, because you may be creating concurrent Boolean operations for the same modified mesh, which in most cases is impossible to execute depending on the chosen target. In this case, you can apply the first Boolean modifier of the stack for the target and then use the other Boolean modifier(s) in the stack for subsequent operations.

Also, if some other modifiers are placed above this modifier and you click on Apply, Blender will warn you with the message "Applied Modifier was not first, results may not be as expected". The best usage scenario for this modifier is to prepare your modified mesh and target to work with the Boolean modifier.

When the Boolean modifier is the first of the stack and is applied, the other Modifiers will act over the resulting meshes using the resulting topology and will remain in the modifiers stack.

Below are two images: one with the subsurf added to the target (Fig. 14), and another with the resulting topology (Fig. 15).

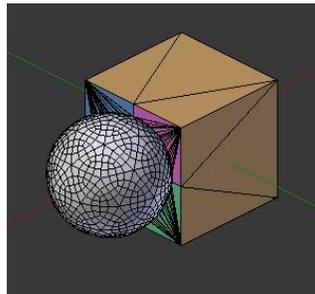


Fig. 2.659: Fig. 14 - The Subsurf is only added to the target (Icosphere), not applied

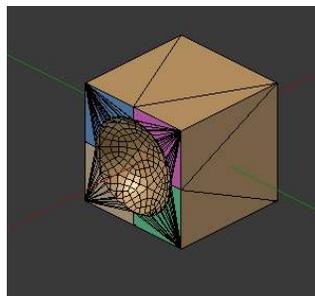


Fig. 2.660: Fig. 15 - The resulting topology. The Subsurf added to the target was taken into account

As you can see, the added (not applied) subsurf to the target was taken into consideration. The topology of the Icosphere with subsurf (Level 2) was completely transferred to the modified mesh.

**Tip:** The target topology determines the resulting topology

The target topology determines the results of the Boolean modifier operation. It means that any modifier added to the target which modifies its topology will affect the resulting mesh of the operation.

### Concurrent Operations

For the modified meshes, you can only apply one operation at a time, but you can use the same target for other modified meshes and use modified meshes as a target for other meshes as well. Also, you can copy or add the same modifier to the modifiers stack as many times as you wish to suit the number of operations you need, but be aware that if you choose concurrent targets which are, at the same time, modified meshes pointing to each other, you can cause Blender to crash with closed loops!

**Hints** Be aware that other modifiers and their stack position could cause this modifier to fail in certain circumstances.

**Tip:** The best way to work with this modifier when you need to make lots of sequential operations of the same modifier is to define the target at the time you need to apply the changes to the topology.

### Face Normals

When using the Boolean modifier, Blender will use the face normal directions to calculate the three Boolean operations. The direction of the normals will define the result of the three available operations. When one of the participants has inverted normals, you're in fact multiplying the operation by  $-1$  and inverting the calculation order. You can, at any time, select your modified mesh, enter Edit Mode and flip the normals to change the behavior of the Boolean modifier. See *Tips for Fixing Mixed Normals* below.

Blender also cannot perform any optimal Boolean operation when one or more of the mesh Normals of the participants that are touching has outwards/inwards normals mixed.

This means you can use the normals of the meshes pointed completely towards the inside or outside of your participants in the operation, but you cannot mix normals pointed inwards and outwards for the faces of the topology used for calculations. In this case, Blender will enable the modifier and you may apply the modifier, but with bad to no effects. We made some examples with a cube and an icosphere showing the results.

See Fig. 16 and 17 - All face normals are pointing outwards (Normal meshes).

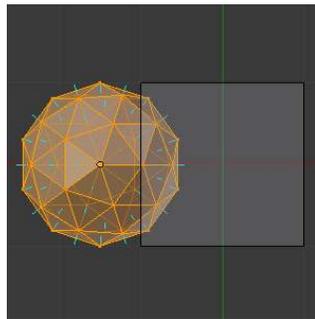


Fig. 2.661: Fig. 16 - Faces with normals pointing outwards

See Fig. 18 and 19 - All face normals are pointing inwards (Meshes with inverted normals)

Now, let's see what happens when the normal directions are mixed for one of the participants in the Boolean modifier operation. In Fig. 20 - Face normals mixed, pointed to different directions and 21 - Resulting operation, you can see that the modifier has bad effects when applied, leaving faces opened:

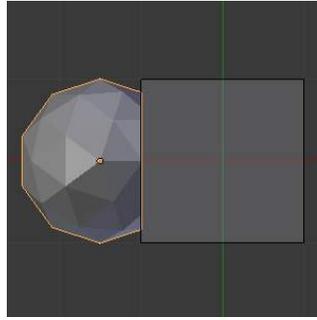


Fig. 2.662: Fig. 17 - Normal Boolean modifier operation (Difference operation)

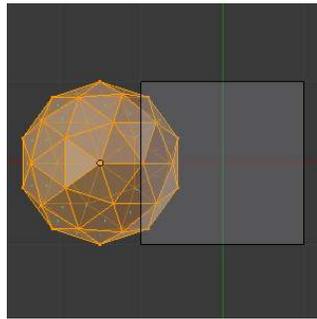


Fig. 2.663: Fig. 18 - Faces with normals pointing inwards

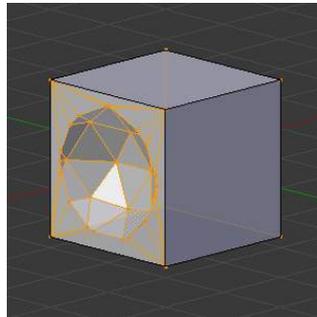


Fig. 2.664: Fig. 19 - Normal Boolean modifier operation (Intersection operation)

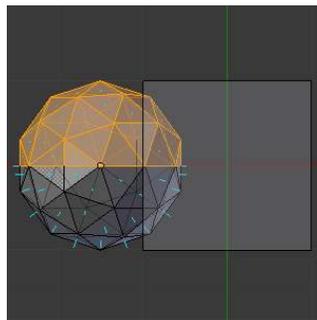


Fig. 2.665: Fig. 20 - Face normals mixed, pointed to different directions

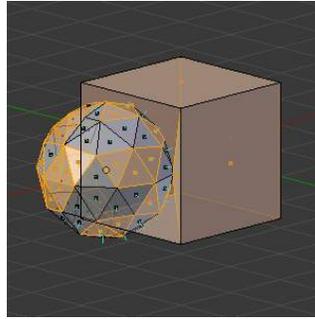


Fig. 2.666: Fig. 21 - Resulting operation, Modifier has bad effect when applied, leaving faces opened

As you can see, the normal directions can be pointing to any of the Mesh sides, but can't be mixed in opposite directions for the faces of the participants. The Library can't determine properly what's positive and negative for the operation, so the results will be bad or you will have no effect when using the Boolean modifier operation.

**Tip for Fixing Mixed Normals** You can fix mixed normals by recalculating them outside or inside; here we also give you a small hint on how to do this prior to Boolean modifier usage:

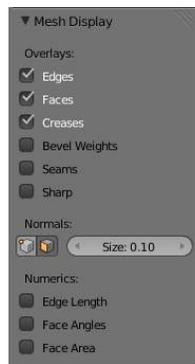


Fig. 2.667: Fig. 22 - Mesh Display in the Transform Panel

To show the normals of the faces, you can open the Transform Panel, find the Mesh display tab, and click on the small cube without the orange dot. (See Fig. 22 - Mesh Display in the Transform Panel.) You can also change the height of the axis that points the direction of the normal. The default is 0.1.

When some normal directions are mixed pointing inwards and outwards, you can recalculate them to the inside using `Ctrl-Shift-N` and to outside using `Ctrl-N`. If the normals still get mixed due to Mesh complexities, you can change to Face selection Mode while in Edit Mode using `Ctrl-Tab` and choosing *Face Mode*. Then select the faces that are pointing in the wrong direction using `Shift-RMB` and use the *Mesh* Menu entry in the Header of the 3D View, go to *Normals* and choose *Flip Normals*:



Fig. 2.668: Fig. 23 - Recalculate and Flip Normals in Mesh Menu Entry - 3D View

## Empty or Duplicated Faces

This modifier doesn't work when the modified and/or the target mesh uses empty faces in the topology used for calculations. If the modifier faces a situation where you have empty faces mixed with normal faces, the modifier will try, as much as possible, to connect the faces and apply the operation. For situations where you have two concurrent faces at the same position, the modifier will operate on the target mesh using both faces, but the resulting normals will get messed. To avoid duplicated faces, you can remove doubles for the vertices before recalculating the normals outside or inside. The button for remove doubles is located in the *Mesh Tools* Panel in the 3D View, while in Edit Mode.

The best usage scenario for this modifier is when you have clean meshes with faces pointing clearly to a direction (inwards/outwards)

Below we show an example of meshes with open faces mixed with normal faces being used to create a new topology. In this example, a difference between the cube and the icosphere is applied, but Blender connected a copy of the icosphere to the Cube mesh, trying to apply what was possible.

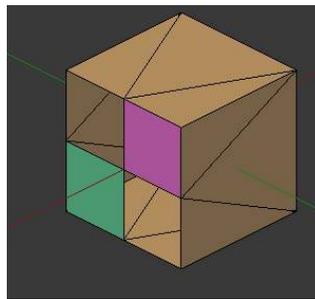


Fig. 2.669: Fig. 24 - Mesh with two empty faces mixed with normal faces

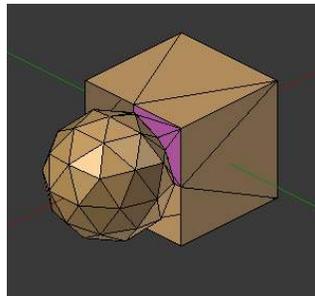


Fig. 2.670: Fig. 25 - Result of a difference operation applied - Blender connected what was possible.

## Open Volumes

The Boolean modifier permits you to use open meshes or non-closed volumes (not open faces).

When using open meshes or non-closed volumes, the Boolean modifier won't perform any operation in faces that don't create a new topology filled with faces using the faces of the target.

See Fig. 26 and Fig. 27 - Resulting operation using two non-closed volumes with faces forming a new topology.

Now, let's see what happens when we use meshes that are partially open, incomplete, or meshes that aren't forming a new topology.

As you can see in Fig. 28, the faces of one participant in the Boolean operation gives incomplete information to the modifier; the result is shown in Fig. 29 - Resulting operation using two open volumes that aren't forming a new topology. The resulting

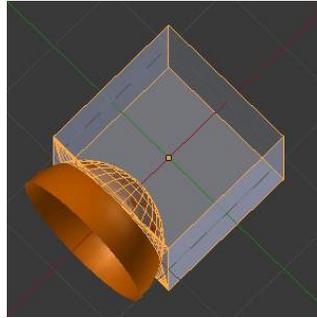


Fig. 2.671: Fig. 26 - Non-closed volumes forming a new topology

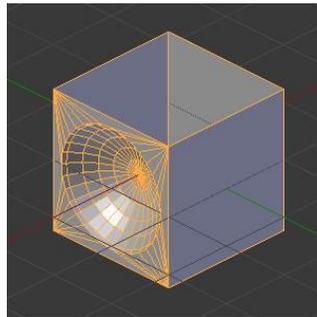


Fig. 2.672: Fig. 27 - Resulting operation using two open volumes performing a new closed topology

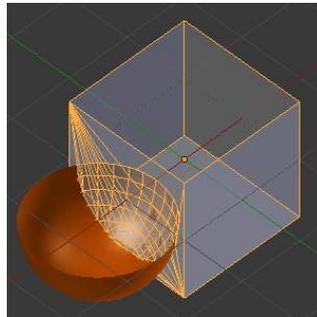


Fig. 2.673: Fig. 28 - Open volumes that aren't forming a new topology.

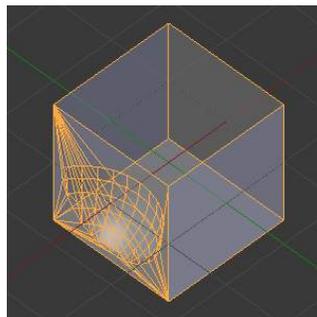


Fig. 2.674: Fig. 29 - Resulting operation using two open volumes that aren't forming a new topology.

edges get messy and there is not enough information to create faces for the resulting Mesh. This example uses a smooth shaded UVsphere cut in half. As explained before, the shading (smooth/flat) doesn't affect the calculations of the modifier.

### Useful Links

- [Carve Development Home](#) - GPLv2 C++ source at Google Code
- [Carve library](#) - Homepage for the Carve Library project.
- [Sculpt Tools](#) - Link for a Blender Add-on - This add-on uses another approach to use the Boolean operations, when you select two or more objects, the active one becomes the modified mesh and all the others becomes a target. This add-on will add the Boolean modifier and apply it to the meshes automatically.

### Build Modifier

The Build modifier causes the faces of the mesh object to appear one after the other over time.

By default, faces appear in the order in which they are stored in memory (by default, the order of creation). The face/vertex order can be altered in Edit Mode by selecting *Sort Faces* from the *Search Menu* (Spacebar)

---

**Note:** When using Blender Render, if the material of the mesh is a halo rather than a standard one, then the vertices of the mesh, not the faces, appear one after another.

---

### Options

Fig. 2.675: Build modifier in action

**Start** The start frame of the building process.

**Length** The number of frames over which to rebuild the object.

**Randomize** Randomizes the order in which the faces are built.

**Seed** The random seed. Changing this value gives a different “random” order when “*Randomize*” is checked - this order is always the same for a given seed/object set.

### Decimate Modifier

The Decimate modifier allows you to reduce the vertex/face count of a mesh with minimal shape changes.

This is not usually used on meshes which have been created by modeling carefully and economically (where all vertices and faces are necessary to correctly define the shape). But if the mesh is the result of complex modeling, sculpting and/or applied subsurf/multires modifiers, the Decimate modifier can be used to reduce the polygon count for a performance increase, or simply remove unnecessary vertices and edges.

The Decimate modifier is a quick and easy way of reducing the polygon count of a mesh non-destructively. This modifier demonstrates the advantages of a mesh modifier system because it shows how an operation which is normally permanent and destroys original mesh data, can be done interactively and safely using a modifier.

Unlike the majority of existing modifiers, the Decimate modifier does not allow you to visualize your changes in Edit Mode.



Fig. 2.676: decimate modifier

## Options

### Decimate Type

**Collapse** Merge vertices together progressively, taking the shape of the mesh into account.

**Ratio** The ratio of faces to keep after decimation.

- On 1.0, the mesh is unchanged.
- On 0.5, edges have been collapsed such that half the number of faces remain (See *Note* below).
- On 0.0, all faces have been removed.

---

**Note:** Although the *Ratio* is directly proportional to the number of remaining faces, triangles are used when calculating the ratio.

This means that if your mesh contains quads, the number of remaining faces will be larger than expected, because quads remain unchanged if their edges are not collapsed.

This is only true if the *Triangulate* option is disabled.

---

**Note:** N-Gons are currently not supported when collapsing edges, the operation will still succeed, but edges attached aren't considered for reduction.

---

**Un-Subdivide** Can be thought of as the reverse of subdivide. Attempts to remove edges that were the result of a subdivide operation. If additional editing has been done after the subdivide operation, the results may be unexpected.

**Iterations** The number of times to perform the un-subdivide operation. Two iterations is the same as one subdivide operation, so you will usually want to use even numbers.

**Planar** Dissolve geometry on the same plane.

**Angle Limit** Dissolve geometry which form angles lower than this setting.

**All Boundaries** When enabled, all vertices along the boundaries of faces are dissolved. This can give nicer results when using a high *Angle Limit*.

**Delimit** Prevent dissolving geometry in certain places.

**Normal** Does not dissolve edges on the borders of areas where the face normals are reversed.

**Material** Does not dissolve edges on the borders of where different materials are assigned.

**Seam** Does not dissolve edges marked as seams.

**Face Count** This field shows the number of remaining faces as a result of applying the Decimate modifier.

## Edge Split Modifier

The Edge Split modifier splits edges within a mesh. The edges to split can be determined from the edge angle (i.e. angle between faces forming this edge), and/or edges marked as sharp.

Splitting an edge affects vertex normal generation at that edge, making the edge appear sharp. Hence, this modifier can be used to achieve the same effect as *Auto Smooth*, making edges appear sharp when their angle is above a certain threshold. It can also be used for manual control of the smoothing process, where the user defines which edges should appear smooth or sharp (see [Mesh Smoothing](#) for other ways to do this). If desired, both modes can be active at once.

The output of the Edge Split modifier is available to export scripts, making it quite useful for creators of game content.

### Options

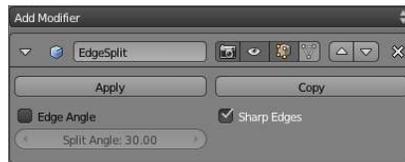


Fig. 2.677: Edge Split modifier.

**Edge Angle** When enabled, edges will be split if the angle between its two adjacent faces is greater than the *Split Angle*.

**Split Angle** On 0, all edges are split. On 180, no edges are split.

**Sharp Edges** When enabled, edges will be split if they were marked as sharp using *Edge Specials* → *Mark Sharp* (Menu shortcut: `Ctrl-E` in Edit Mode).

---

**Note:** *Non-manifold* edges (edges shared by more than two faces) will always be split.

---

### Examples

---

**Note:** Splitting edges can also be performed manually in Edit Mode using: *Edge Specials* → *Edge Split* (Menu shortcut: `Ctrl-E`).

---

## Mask Modifier

The Mask modifier allows vertices of an object to be hidden dynamically based on vertex groups.

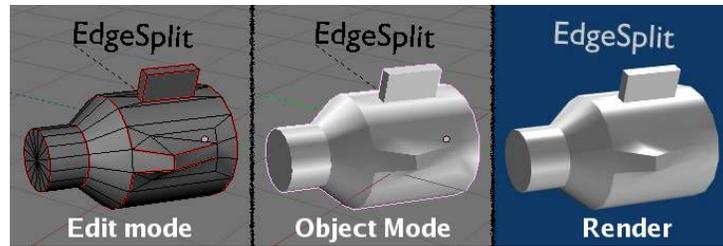


Fig. 2.678: Edge Split modifier output with From Marked As Sharp selected.

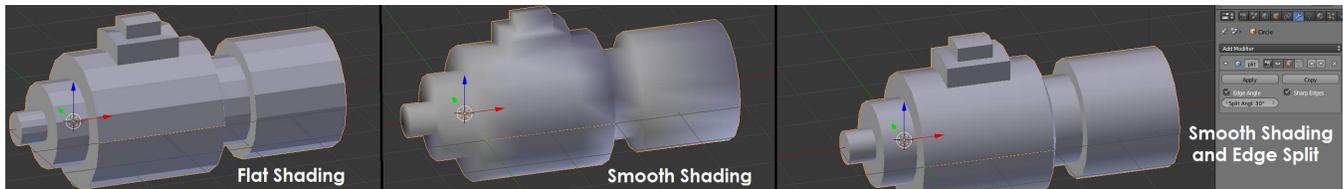


Fig. 2.679: (From Left to right): Flat Shading, Smooth Shading, Smooth Shading with Edge Split.

## Options

**Mode** The Mask modifier can hide parts of a mesh based on two different modes, selectable from this drop-down list.

### Vertex Group

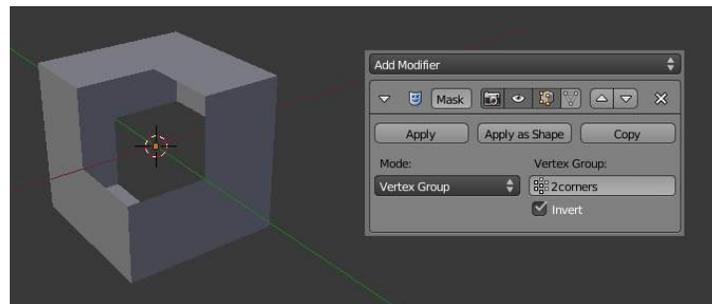


Fig. 2.680: Vertex Group

When the *Vertex Group* option is selected, all vertices belonging to the chosen Vertex Group will be visible, and all vertices which do not belong to that group will be hidden.

### Armature

When in Pose Mode, vertices belonging to the Vertex Group associated with the active bone (same names) will be visible. Vertices not in that group will be hidden.

**Inverse** Normally, vertices belonging to the selected Vertex Group (or group associated with the active pose-bone) will be shown. The *Invert* toggle allows you to reverse this behavior, instead showing all vertices which do not belong to the Vertex Group, and hiding those that do.

**Warning:** The Mask modifier only takes into account whether a vertex is part of the group or not, the weight is not taken into account.

Thus, a vertex with a weight of 0.0 will be treated the same as a vertex with a weight of 1.0, because even though it has no weight it is still a member of that group.

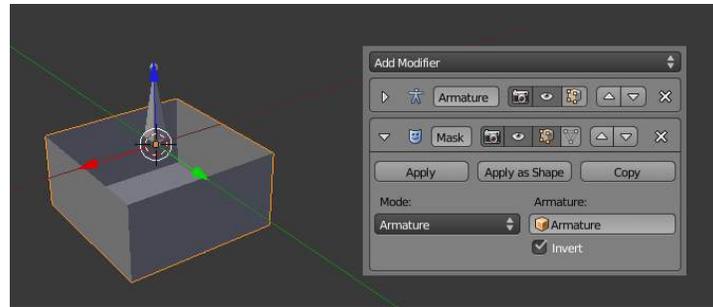


Fig. 2.681: Armature

## Mirror Modifier

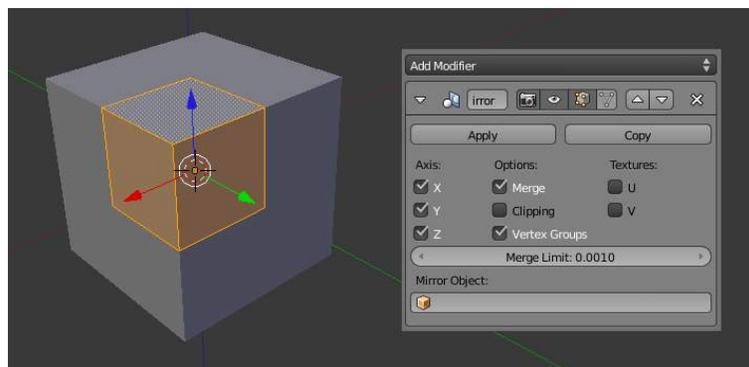


Fig. 2.682: The corner of a cube mirrored across three axes to form... well... a cube.

The Mirror modifier mirrors a mesh along its **local** X, Y and/or Z axes, across the object's center (the mirror plane is then defined by the two other axes).

It can also use another object as the mirror center, then use that object's local axes instead of its own.

## Options

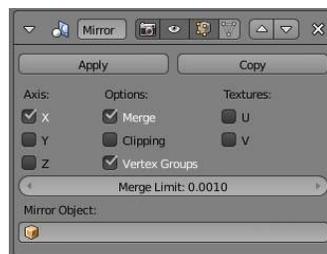


Fig. 2.683: Mirror modifier

**Axis** The axis (X, Y, or Z) along which to mirror (i.e. the axis perpendicular to the mirror plane of symmetry).

To understand how the axis applies to the mirror direction, if you were to mirror on the X axis, the positive X values of the original mesh would become the negative X values on the mirrored side.

You can select more than one of these axes - you'll then get more mirrored copies. With one axis you get a single mirror, with two axes four mirrors, and with all three axes eight mirrors.

**Options:**

**Merge** Where a vertex is in the same place (within the *Merge Limit* distance) as its mirror it will be merged with the mirrored vertex.

**Clipping** Prevents vertices from moving through the mirror plane(s) while the user is transforming them in Edit Mode.

If *Clipping* is enabled but vertices are beyond the mirror plane and outside of the *Merge Limit*, the vertices will not be merged. But as soon as the vertices are within *Merge Limit* they are snapped together and cannot be moved beyond the mirror plane.

---

**Note:** Vertices on the mirror plane will be unable to move away from the mirror plane as long as *Clipping* is enabled. You must disable *Clipping* to be able to move the vertices along the mirror axis again.

---

**Vertex Groups** When enabled, the Mirror modifier will try to mirror existing vertex groups.

A very nice feature, but one that has very specific prerequisites:

- The vertex groups you want to mirror must be named following the usual left/right pattern (i.e. suffixed by something like ".R", ".right", ".L", etc).
- The mirror side vertex group must already exist (it will not be created automatically). It must also be completely empty (no vertices assigned to it).

**Textures** The *U* and *V* options allows you to mirror the UV texture coordinates across the middle of the image.

E.g. if you have a vertex with UV coordinates of (0.3, 0.9), its mirror copy will have UV coordinates of (0.7, 0.1).

**Merge Limit** The maximum distance between a vertex and its mirror copy before they are merged together. In other words, a vertex may be half this distance away from the mirror plane before it snaps to it.

**Mirror Object** The name of another object (usually an empty), to be used as the reference for the mirror process: its center and axes will drive the plane(s) of symmetry. You can of course animate its position/rotation to animate the mirror effect.

**Hints**

Many modeling tasks involve creating objects that are symmetrical. However, there used to be no quick way to model both halves of an object without using one of the workarounds that have been discovered by clever Blender artists over the years. A common technique was to model one half of an object and use `Alt-D` to create a linked duplicate which can then be scaled on one axis by `-1` to produce a perfect mirror-image copy which updates in real time as you edit.

The Mirror modifier offers a simpler way to do this. Once your modeling is completed you can either click *Apply* to make a real version of your mesh or leave it as is for future editing.

**Using the Mirror Modifier with a Subdivision Surface Modifier** When using the Mirror modifier along with a `subsurf` modifier, the order in which the modifiers are placed is important.

The above image shows the `subsurf` modifier placed before the Mirror one; as you can see the effect of this is that the mesh is split down the center line of the mirror effect. This is because the `subsurf` calculation moves vertices away from the mirror plane, too far away from the *Merge Limit*.

The above image shows the Mirror modifier placed before the `subsurf` modifier. In this order, the mirror calculation is done and the vertices are merged together. Only after that does the `subsurf` modifier move any vertices.

**Accurately Positioning the Mirror Plane** To apply a Mirror modifier, it is common to have to move the object's center onto the edge or face that is to be the axis for mirroring. This can be tricky when attempted visually.

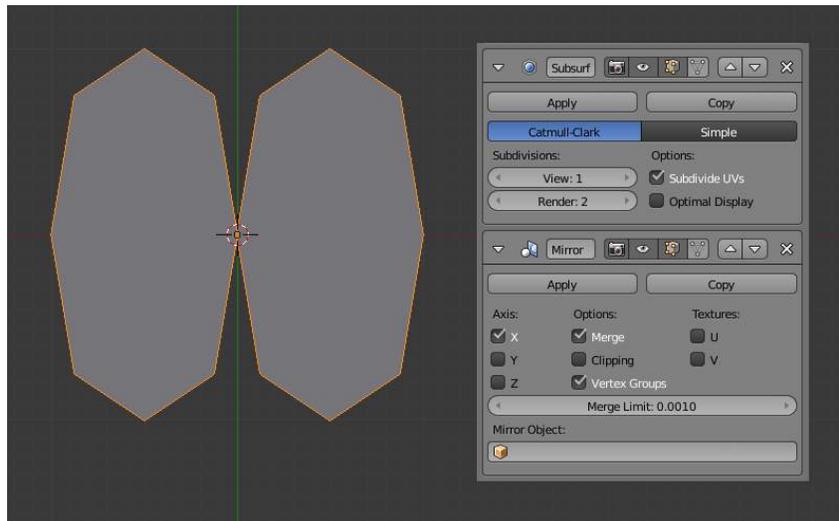


Fig. 2.684: Subsurf modifier before Mirror modifier

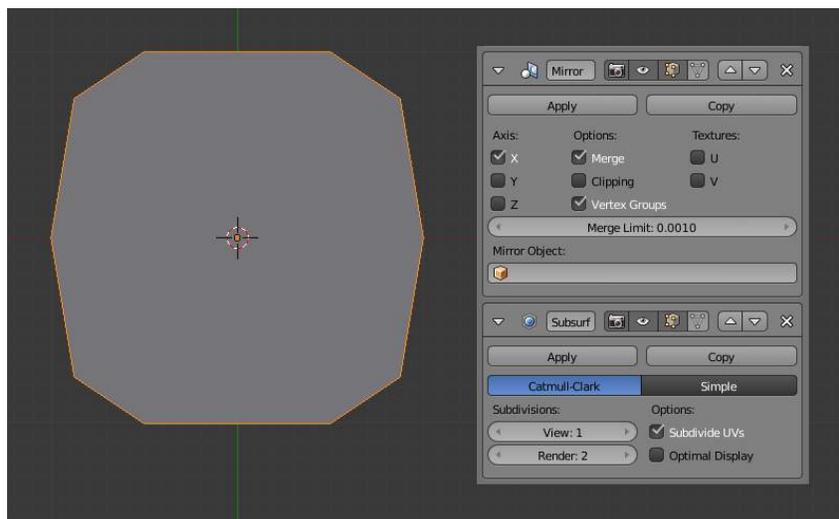


Fig. 2.685: Mirror modifier before Subsurf modifier

A good technique to achieve an exact position is to select the edge, then use `Shift-S` and choosing *Cursor to Selection*. This will position the 3D Cursor in the center of the edge. Finally, press `Ctrl-Alt-Shift-C` for the *Set Origin* menu, then select *Origin to 3D Cursor*. This will move the object's center (and thus, the mirror plane) to where the 3D cursor is located, and the mirroring will be exact.

An alternative is to use an Empty as a *Mirror Object* that you move to the correct position.

## Multiresolution Modifier



Fig. 2.686: Multires modifier

The Multiresolution modifier (often shortened to *Multires*) gives you the ability to subdivide a mesh similarly to the *Subsurf* modifier, but also allows you to edit the new subdivision levels in sculpt mode.

---

**Note:** The *Multiresolution Modifier* is the only modifier that cannot be repositioned in the stack if it means that there will be geometry or other object data created or removed before it (i.e. all *Generate*, some *Modify* and some *Simulate* modifiers cannot come before the Multiresolution modifier.)

---

### Options

**Catmull-Clark / Simple** Set the type of subdivision.

**Simple** Maintains the current shape, and simply subdivides edges.

**Catmull-Clark** Creates a smooth surface, usually smaller than the original, using the standard *Catmull-Clark* subdivision surface algorithm.

**Preview** Set the level of subdivisions to show in the 3D View.

**Sculpt** Set the number of subdivisions to use in Sculpt Mode.

**Render** Set the number of subdivisions to show when rendering.

**Subdivide** Add another level of subdivision.

**Delete Higher** Deletes all subdivision levels that are higher than the current one.

**Reshape** Copies vertex coordinates from another mesh. To use, first select a different mesh object with matching topology and vertex indexes, then `Shift` select the object you wish to copy vertex coordinates to and click *Reshape*.

**Apply Base** Modifies the original unsubdivided mesh to match the form of the subdivided mesh.

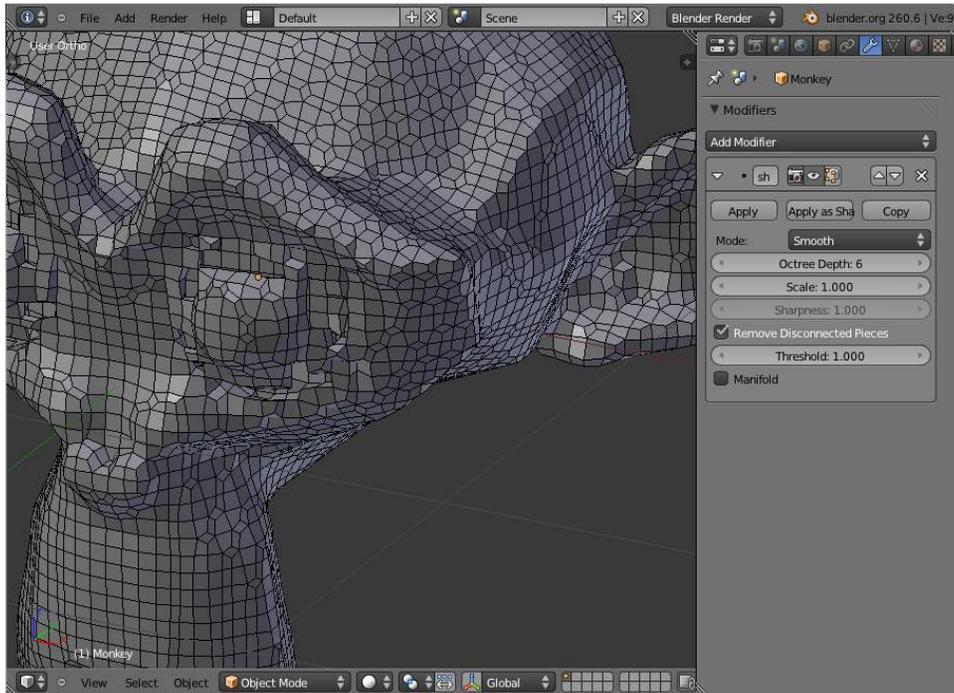
**Subdivide UVs** When enabled, the UV maps will also be subdivided. (i.e. Blender will add “virtual” coordinates for all sub-faces created by this modifier).

**Optimal Display** When drawing the wireframe of this object, the wires of the new subdivided edges will be skipped (only draws the edges of the original geometry).

**Save External** Saves displacements to an external .btx file.

## Remesh Modifier

The Remesh modifier is a tool for generating new mesh topology. The output follows the surface curvature of the input, but its topology contains only quads.



## Options

**Mode** There are three basic modes available in the remesh modifier: Blocks, Smooth and Sharp.

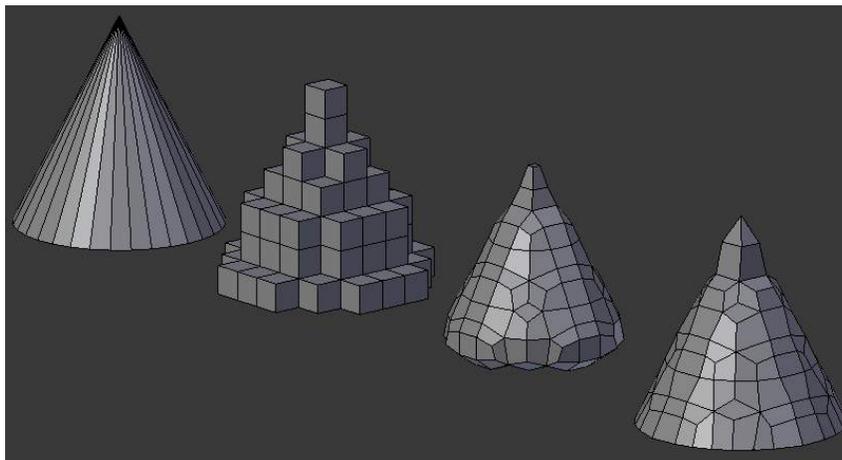


Fig. 2.687: This example shows a cone with each of the different remesh modes. From left to right: original cone, Blocks, Smooth, and Sharp

The output topology is almost identical between the three modes; what changes is the smoothing.

**Blocks** There is no smoothing at all.

**Smooth** Output a smooth surface.

**Sharp** Similar to *Smooth*, but preserves sharp edges and corners. In the above image, the circular bottom of the cone and the top point of the cone are more accurately reproduced in Sharp mode.

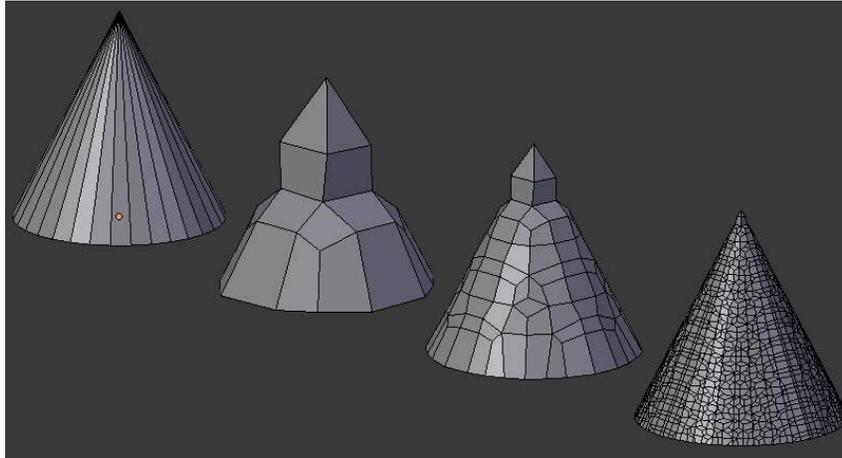


Fig. 2.688: Input mesh, and the low to high resolution output meshes

**Octree Depth** The Octree Depth sets the resolution of the output. Low values will generate larger faces relative to the input, higher values will generate a denser output.

**Scale** The result can be tweaked further by setting the Scale; lower values effectively decrease the output resolution.

**Sharpness** Shown when using the *Sharp Mode* - Higher values produce edges more similar to the input, while lower values filter out noise.

**Smooth Shading** Output faces with smooth shading rather than flat shading. The smooth/flat shading of the input faces is not preserved.

**Remove Disconnected Pieces** Filter out small disconnected pieces of the output.

**Threshold** Use this to control how small a disconnected component must be to be removed.

### Usage

In the modifier panel, add a Remesh modifier. The input mesh should have some thickness to it; if the input is completely flat, add a [solidify](#) modifier above the remesh modifier.

### Examples

#### Screw Modifier

The Screw modifier is similar to the *Screw tool* in the *Tool Shelf* in that it takes a profile object, a Mesh or a Curve, to create a helix-like shape.

The profile should be properly aligned to the cardinal direction of the object rather than to the screw axis.

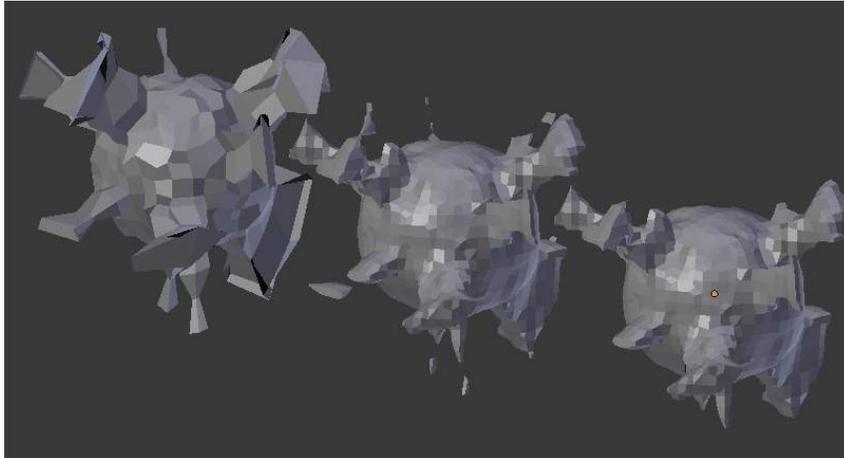


Fig. 2.689: The input mesh (left) is fairly noisy, so the initial output of the remesh modifier (center) contains small disconnected pieces. Enabling Remove Disconnected Pieces (right) deletes those faces.

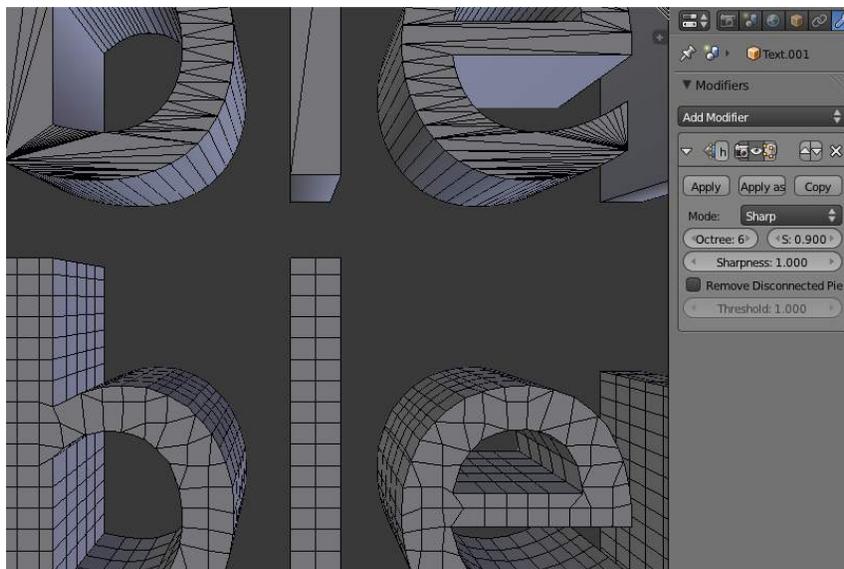


Fig. 2.690: Remesh modifier applied to text to improve topology



Fig. 2.691: Properly aligning the profile object is important

## Options



Fig. 2.692: Screw modifier

**Axis** The axis along which the helix will be built.

**Screw** The height of one helix iteration.

**AxisOb** The name of an object to define the axis direction.

**Object Screw** Use the distance from the *AxisOb* to define the height of one helix iteration.

**Angle** Degrees for a single helix revolution.

**Steps** Number of steps used for a single revolution displayed in the 3D view. Beware of setting this higher than *Render Steps*, which is the value used for rendering.

**Render Steps** As above, but used during render time. Increase to improve quality.

**Smooth Shading** Output faces with smooth shading rather than flat shading. The smooth/flat shading of the input geometry is not preserved.

**Calc Order** Order of edges is calculated to avoid problems with normals and shading. Only needed for meshes, not curves.

**Flip** Flip normals direction.

**Iterations** Number of revolutions.

**Stretch U/V** Stretch the UV coordinates from 0.0 to 1.0 when UVs are present.

## Skin Modifier

The Skin modifier uses vertices and edges to create a skinned surface, using a per-vertex radius to better define the shape. The output is mostly quads, although some triangles will appear around intersections.

It is a quick way to generate base meshes for sculpting and/or smooth organic shapes with arbitrary topology.

---

**Note:** Faces in the original geometry are ignored by the Skin modifier.

---

## Options

**Create Armature** Create an armature on top of the object - each edge becomes a bone.

---

**Note:** If the root vertex has more than one adjacent edge, an extra bone will be created to serve as the root.

---

This function does the following:

1. A new armature object is added with bones matching the input mesh. The active selection is switched to the new armature.

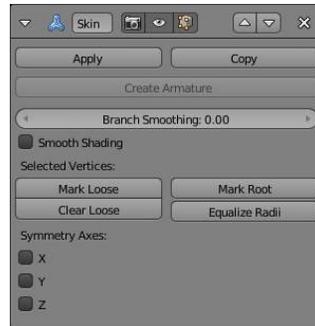


Fig. 2.693: Skin modifier UI.

2. Weight groups are added to the input mesh. The Skin modifier propagates these weights to the output as well.
3. An Armature modifier is added directly below the Skin modifier. Note that the Armature modifier is being applied after the Skin modifier because it should only deform the output, whereas if it were above the Skin modifier it might change the resulting topology.

**Branch Smoothing** A branch point is a vertex with three or more connected edges. These areas tend to produce more complicated topology, some of which may overlap. The *Branch Smoothing* setting relaxes the surface around these points, with the side effect of shrinking the surface.

**Smooth Shading** Output faces with smooth shading rather than flat shading. The smooth/flat shading of the input geometry is not preserved.

#### Selected Vertices

**Mark/Clear Loose** By default, a branch vertex (vertex with three or more connected edges) will generate extra edge loops along adjacent edges in order to keep the output tight. Branches can be made loose by clicking *Mark Loose*, which will allow the output to stretch between all adjacent vertices. This can be disabled again by clicking *Clear Loose* with the vertex selected.

**Mark Root** Marking a vertex as root causes that vertex to be used for calculating rotations for connected limbs. Root vertices also affect the armature output; they will be used as the origin for the root bones.

Roots are shown in the 3D View with a red dashed circle around the vertex.

Each set of connected vertices should have one root node. *Mark Root* enforces the one-root per set rule, so it is not necessary to manually unmark roots.

**Equalize Radii** Makes the skin radii of selected vertices equal on each axis.

**Symmetry Axes** The Symmetry Axes checkboxes are used to keep the output topology symmetrical in their respective axes. In other words, using it avoids merging triangles across an axis unless the triangles form a symmetric quad.

---

**Note:** These symmetry axes checkboxes do not add geometry flipped across an axis. For that, the Mirror modifier should be used, typically placed above the Skin modifier.

---

#### Usage

Add the Skin modifier to a mesh. Disable *Limit selection to visible* in the 3D view so that you can see the vertices inside the new geometry.

The skin modifier uses ordinary vertices and edges as input. All of the regular Edit Mode tools (such as extrude, subdivize, grab, scale, and rotate) can be used when building a skinned mesh.

The radius of selected vertices can be adjusted in the *Transform* panel of the *Properties* region (N)

## Examples

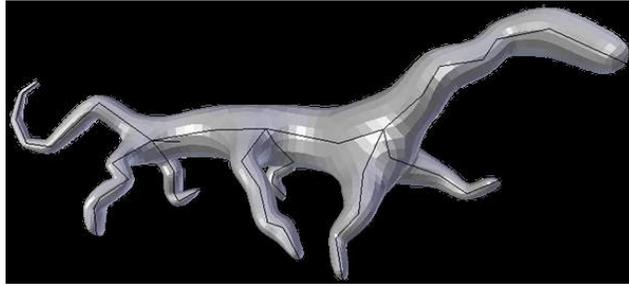


Fig. 2.694: Fig1: Simple creature, made with only the Skin modifier.

- In the modifiers menu, add a *Skin* modifier.
- Tab into edit mode and start extruding.
- Try to sketch results similar to *Fig. 1*, through extruding the vertices of the object.
- Use `Ctrl-A` to change the size of the different regions within the creature.
- Use *Mark Loose* at regions like the neck, to merge these faces more together.
- To get smoother results, activate *Smooth Shading* and add a *subsurf* (Shortcut: `Ctrl-3`) to the object.

## External links

- [Skin Modifier Development at Blender Nation](#) – An early demonstration of the skin modifier by Nicholas Bishop (March 2011)
- Ji, Zhongping; Liu, Ligang; Wang, Yigang (2010). [B-Mesh: A Fast Modeling System for Base Meshes of 3D Articulated Shapes](#), Computer Graphics Forum 29(7), pp. 2169-2178. – The work this modifier is based on ([direct link to PDF](#))
- [Related thread on Blender artists](#)

## Solidify Modifier

The Solidify modifier takes the surface of any mesh and adds depth to it.

### Options

**Thickness** The depth to be solidified.

**Offset** A value between  $-1$  and  $1$  to locate the solidified output inside or outside the original mesh. Set to  $0.0$ , the solidified output will be centered on the original mesh.

**Clamp** A value between  $0$  and  $2$  to clamp offsets to avoid self intersection.

**Vertex Group** Only vertices in this group are solidified - their weights are multiplied by the thickness, so vertices with lower weights will be less thick.

**Invert** Reverses the vertex group, so that only vertices which are **not** in the vertex group are solidified.

**Factor** How much the vertex weights are taken into account.

- On  $0.0$ , vertices with zero weight will have no thickness at all.

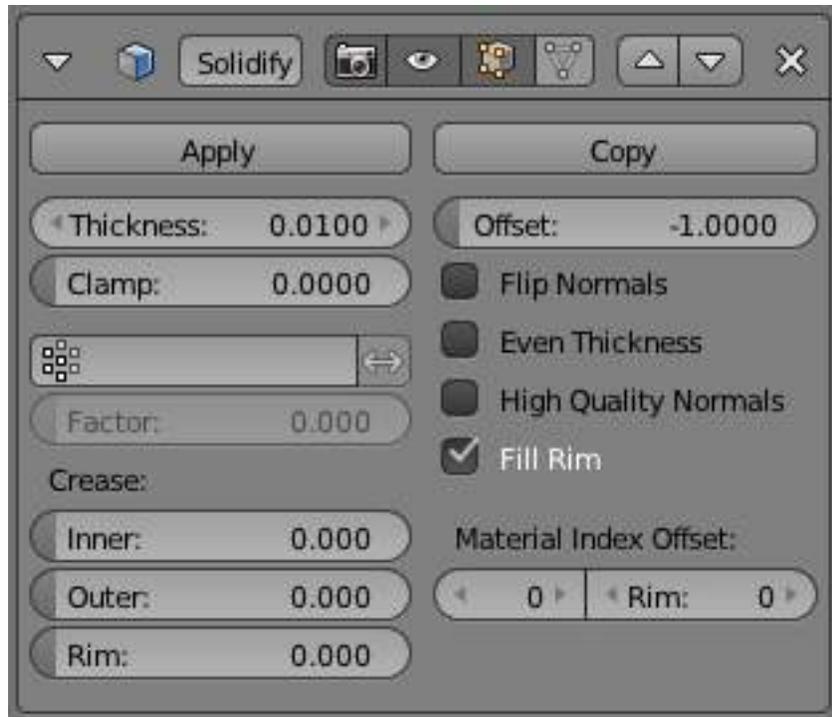


Fig. 2.695: Solidify modifier

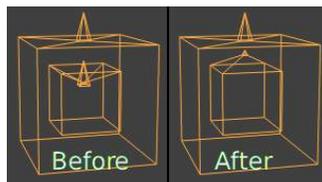


Fig. 2.696: Clamp Offset

- On 0.5, vertices with zero weight will be half as thick as those with full weight.
- On 1.0, the weights are ignored and the *thickness* value is used for every vertex.

**Crease** These options are intended for usage with the [Subdivision Surface](#) modifier.

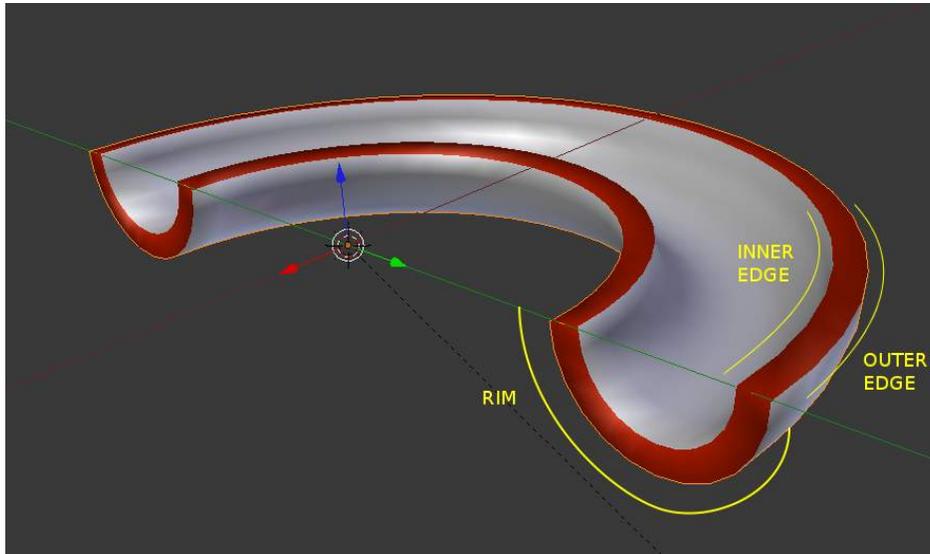


Fig. 2.697: Rim and edges. In this example, the object was assigned a second material used to color the rim red.

**Inner** Set a crease to the inner edges.

**Outer** Set a crease to the outer edges.

**Rim** Set a crease to the rim.

**Flip Normals** Reverse the normals of all geometry (both the inner and outer surfaces).

**Even Thickness** Maintain thickness by adjusting for sharp corners. Sometimes improves quality but also increases computation time.

**High Quality Normals** Normals are calculated to produce a more even thickness. Sometimes improves quality but also increases computation time.

**Fill Rim** Fills the gap between the inner and outer edges.

**Only Rim** TODO

---

**Note:** *Fill Rim* and *Only Rim* only make a difference on *non-manifold* objects, since the “rims” are generated from the borders of the original geometry.

---

**Material Index Offset** Choose a different material to use for the new geometry; this is applied as an offset from the original material of the face from which it was solidified.

- A value of 0 means it will use the same material.
- A value of 1 means it will use the material immediately below the original material.
- A value of -2 means the material two positions above the original material will be used.

These are clamped to the top-most and bottom-most material slots.

**Rim** Similarly, you can give another material to the rim faces.

**Warning:** The modifier thickness is calculated using local vertex coordinates. If the object has non-uniform scale, the thickness will vary on different sides of the object.  
To fix this, either apply (`Ctrl-A`) or clear (`Alt-S`) scale.

**Warning:** Solidify thickness is an approximation. While “Even Thickness” and “High Quality Normals” should yield good results, the final wall thickness isn’t guaranteed and may vary depending on the mesh topology.  
In order to maintain precise wall thickness in every case, we would need to add/remove faces on the offset shell - something this modifier doesn’t do since this would add a lot of complexity and slow down the modifier.  
Hence it is not recommended to use this for purposes requiring accuracy such as architectural/CAD modeling.

## Subdivision Surface Modifier



Fig. 2.698: Click to zoom; Subsurfs levels 0 to 3, without and with Smooth Shading. This was rendered from: [SubsurfDemo.blend](#)

Subdivision Surface (*Subsurf* in short) is a method of subdividing the faces of a mesh to give a smooth appearance, to enable modeling of complex smooth surfaces with simple, low-vertex meshes. This allows high resolution mesh modeling without the need to save and maintain huge amounts of data and gives a smooth *organic* look to the object.

This process creates virtual geometry that is generated non-destructively without modifying the original mesh, but it can be converted to real geometry that you could edit with the *Apply* button.

Also, like the rest of the Modifiers, order of execution has an important bearing on the results. For this, see the documentation on [The Stack](#).

Keep in mind that this is a different operation than its companion, [Smooth Shading](#). You can see the difference between the two in the grid image to the right.

**Tip:** The Subsurf modifier does not allow you to edit the new subdivided geometry without applying it, but the [Multires](#) modifier does (in sculpt mode).

## Options

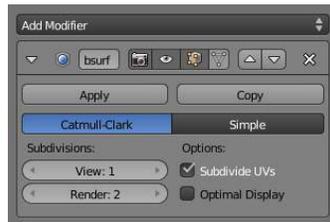


Fig. 2.699: Modifier's panel

**Type** This toggle button allows you to choose the subdivision algorithm:

**Catmull-Clark** The default option, subdivides and smooths the surfaces. According to [its Wikipedia page](#), the “arbitrary-looking formula was chosen by Catmull and Clark based on the aesthetic appearance of the resulting surfaces rather than on a mathematical derivation.”

**Simple** Only subdivides the surfaces, without any smoothing (the same as  $\bar{w} \rightarrow \text{Subdivide}$ , in Edit Mode). Can be used, for example, to increase base mesh resolution when using displacement maps.

**Subdivisions** Recursively adds more geometry. For details on polygon counts, see the [Performance Considerations](#) section.

**View** The number of subdivision levels shown in the 3D View.

**Render** The number of subdivision levels shown in renders.

The right combination of these settings will allow you to keep a fast and lightweight approximation of your model when interacting with it in 3D, but use a higher quality version when rendering.

**Warning:** Be careful not to set the *View* subdivisions higher than the *Render* subdivisions, this would mean the 3D View will be higher quality than the render.

### Options:

**Subdivide UVs** When enabled, the UV maps will also be subsurf'd (i.e. Blender will add “virtual” coordinates for all sub-faces created by this modifier). See this [example blend file](#).



Fig. 2.700: Subdivide UVs on and off – see the [.blend](#) for the source of this image.

**Optimal Display** When drawing the wireframe of this object, the wires of the new subdivided edges will be skipped (only draws the edges of the original geometry).

## Edit Cage

To view and edit the results of the subdivision while you're editing the mesh, you must enable the *Editing Cage* (the triangle button in the modifier's header). This lets you grab the vertices as they lie in their new smoothed locations, rather than on the original mesh.

Edit Cage Off (Default)	Edit Cage On

With the edit cage off, some vertices are buried under the subsurfed mesh. With dense vertex configurations, you might even have to temporarily disable the modifier or view [wireframe](#) shading so that you can see these vertices.

With the edit cage on, you do not have this problem. It does, however, have its own disadvantage—it can look *too* nice, hiding irregularities. Notice the three quads running in the middle of Suzanne's ear: you can only tell how crooked they are in the "edit cage off" version. When you are modeling, you will more often want to see your mesh deformities in their full ugliness so that you can apply your skills until it is sheer prettiness.

## Order of the Modifier Stack

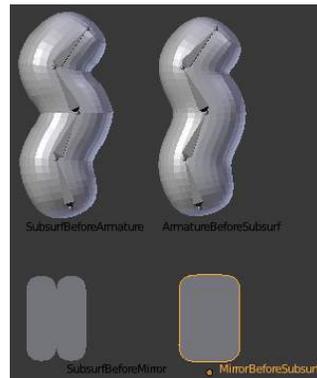


Fig. 2.701: Notice that the Armature Modifier before the Subsurf comes out much better in this case. Also, the Mirror before the Subsurf is clearly correct compared to the other way around.

The [Evaluation order](#) of Modifiers is often significant, but especially so in the case of the Subsurf. The key to deciding your Modifier stack order is to picture the changes at each step, perhaps by temporarily Apply'ing the Modifiers, or perhaps by simply tinkering with the order until things come out right. To see the file behind these screenshots, you can look at [Manual-Modifiers-Generate-Subsurf\\_OrderOfExecution.blend](#).

## Control

Subsurf rounds off edges, and often this is not what you want. There are several solutions.

## Weighted Creases Reference

Mode: Edit Mode (Mesh)

Panel: 3D View → *Transform Properties*

Menu: *Mesh* → *Edges* → *Crease Subsurf*

Hotkey:  $\aleph$  (*Transform Properties* Panel)

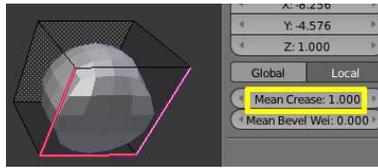


Fig. 2.702: A Subsurf'd Cube with Creased Edges

Weighted edge creases for subdivision surfaces allows you to change the way Subsurf subdivides the geometry to give the edges a smooth or sharp appearance.

The crease weight of selected edges can be changed in the *Transform* panel of the properties region (N), or by using the shortcut `Shift-E` and moving the mouse closer or further from the selected edges to adjust the crease weight. A higher value makes the edge “stronger” and more resistant to the smoothing effect of subdivision surfaces.

Another way to remember it is that the weight refers to the edge’s sharpness; Edges with a higher weight will be deformed less by subsurf. Recall that the subsurf'd shape is a product of all intersecting edges, so to make an area sharper, you have to increase the weight of all the surrounding edges.

## Edge Loops Reference

Mode: Edit Mode (Mesh)

Hotkey: `Ctrl-R`

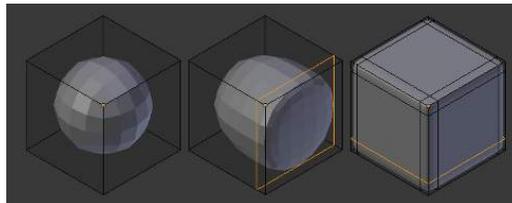


Fig. 2.703: A Subsurf Level 2 Cube, the same with an extra Edge Loop, and the same with six extra Edge Loops

The Subsurf modifier demonstrates why good, clean topology is so important. As you can see in the figure, the Subsurf modifier has a drastic effect on a default Cube. Until you add in additional Loops (with `Ctrl-R`), the shape is almost unrecognizable as a cube.

A mesh with deliberate topology has good placement of Edge Loops, which allow the placement of more Loops (or removal of Loops, with `[x] → Edge Loop`) to control the sharpness/smoothness of the resultant mesh.

**Combination** It is valuable to know the use of all three tools: Smooth/Flat Shading, Edge Creases and Edge Loops.

Consider the task of modeling a 2”x4” block of wood that has had a notch cut out. The factory edges are rounded off (a good task for Smooth Shading and some Edge Loops), but the edges where the saw touched are crisp (a good task for Flat Shading and Edge Crease).

Note that we had to add some extra edge loops near the Creased edges – this was only to limit the effects of Smooth Shading, which bleeds over onto the adjacent flat faces.

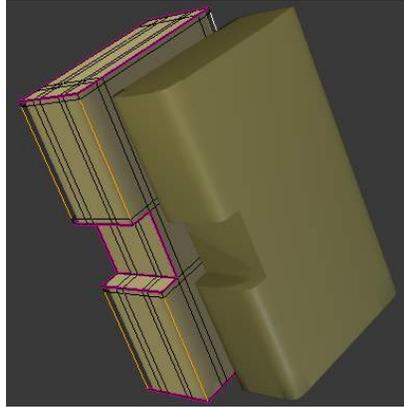


Fig. 2.704: Purple edges are created, orange (selected) are intended to be rounded off. See: [WoodBlock.blend](#)

### Limitations & Workarounds

Blender’s subdivision system produces nice smooth subsurfed meshes, but any subsurfed face (that is, any small face created by the algorithm from a single face of the original mesh), shares the overall normal orientation of that original face.

Abrupt normal changes can produce ugly black gouges (See: *Fig. 1*), even though these flipped normals are not an issue for the shape itself (See: *Fig. 2*).

A quick way to fix this (one which works 90% of the time) is to use Blender’s “Recalculate Normals” operation: In Edit Mode, select all with **A**, then press **Ctrl-N** to recalculate the normals to point outside. If you still have some ugly black gouges you will have to manually flip some normals. To do this (still in Edit Mode), use the *Specials* → *Flip Normals* functionality (shortcut: **W, N**) to fix them. Smoothing out normals is good for the mesh, and it’s good for the soul.

### Performance Considerations

Higher levels of subdivisions mean more vertices, and more vertices means more memory will be used (both video memory for display, and system RAM for rendering). Blender could potentially crash or hang if you do not have enough memory.

When using high levels of subdivision with a graphics card that has a low total amount of Vram, some parts of the geometry will disappear visually. Your mesh will actually be OK, because the render is generated using your Object Data, (even though it cannot be shown by your graphics card).

The total Vertex, Edge, and Face counts from the Modifier’s effect on a Cube can be found in this table:

Cube Subdivision Level	Resulting Verts	Resulting Edges	Resulting Faces
0	8	12	6
1	26	48	24
2	98	192	96
3	386	768	384
4	1538	3072	1536
5	6146	12288	6144
6	24578	49152	24576
Formula	$3 \cdot 2^{n+1} + 4$	$3 \cdot 4^{n+1}$	$verts - 2$

While we’re at it, here is the pattern for subdividing a single quadrilateral plane:

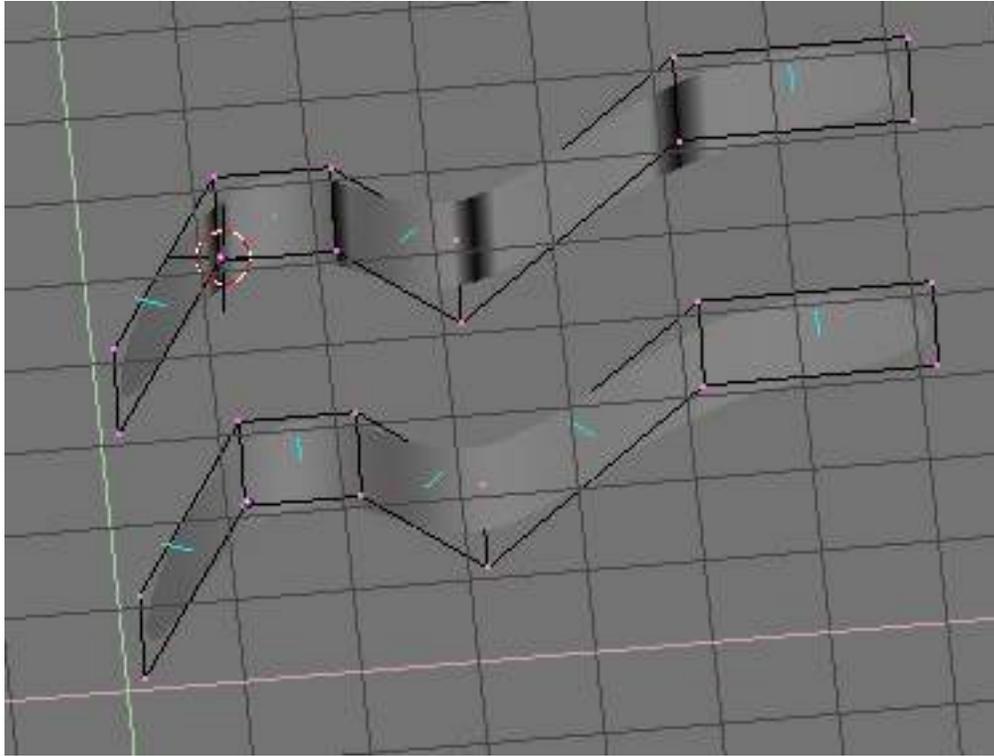


Fig. 2.705: Fig. 1: Solid view of subsurfed meshes with inconsistent normals (top) and consistent normals (bottom). Note the ugly dark areas that appear.

Quad Subdivision Level	Resulting Verts	Resulting Edges	Resulting Faces
0	4	4	1
1	9	12	4
2	25	40	16
3	81	144	64
4	289	544	256
5	1089	2112	1024
6	4225	8320	4096
Formula	$((2**n+2)**2)/4$	$2**(n-1)*(2**n+2)$	$4**(n-1)$

And, of course, triangles:

Tri Subdivision Level	Resulting Verts	Resulting Edges	Resulting Faces
0	3	3	1
1	7	9	3
2	19	30	12
3	61	108	48
4	217	408	192
5	817	1584	768
6	3169	6240	3072
Formula	Do you know it?	$3*(2**(n-3))*(2**n+2)$	

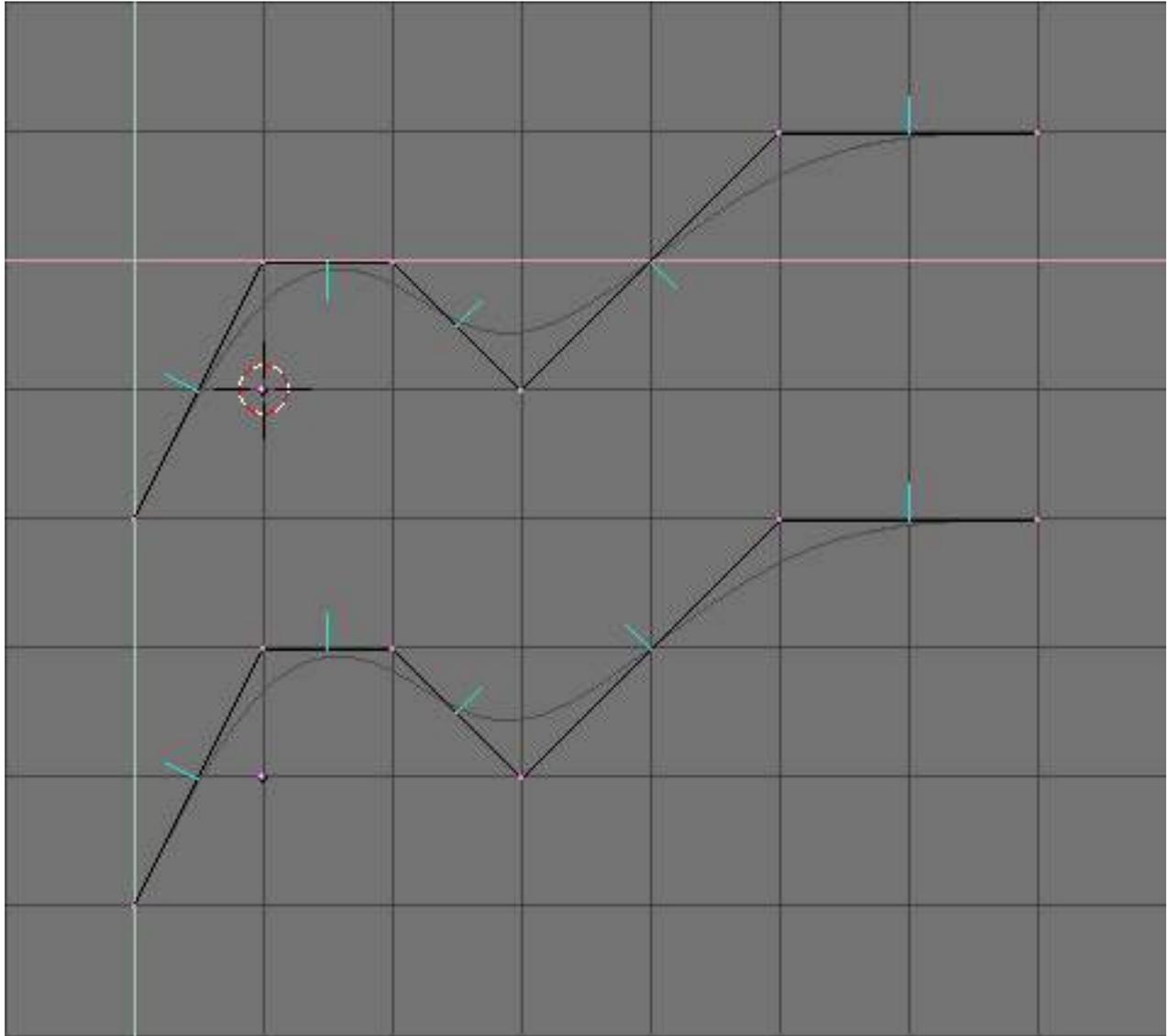


Fig. 2.706: Fig. 2: Side view of the above meshes' normals, with random normals (top) and with coherent normals (bottom).

## Keyboard Shortcuts

To quickly add a subsurf modifier to one or more objects, select it/them and press `Ctrl-1`. That will add a subsurf modifier with *View Subdivisions* on 1.

You can use other numbers too, such as `Ctrl-2`, `Ctrl-3`, etc, to add a subsurf with that number of subdivisions. The *Render Subdivisions* will always be on 2 when added like this.

If an object already has a subsurf modifier, doing this will simply change its subdivision level instead of adding another modifier.

## Triangulate Modifier

The Triangulate modifier converts all faces in a mesh (whether it be quads or N-gons) to triangular faces. This modifier does the exact same function as the triangulate function (`Ctrl-T`) in Edit Mode.



### Options

#### Quad Method:

- Beauty** Split the quads in nice triangles, slower method.
- Fixed** Split the quads on the 1st and 3rd vertices.
- Fixed Alternate** Split the quads on the 2nd and 4th vertices.
- Shortest Diagonal** Split the quads based on the distance between the vertices.

#### Ngon Method:

- Beauty** Arrange the new triangles nicely, slower method.
- Scanfill** Split the ngons using a scanfill algorithm.

## Wireframe Modifier

The Wireframe modifier transforms a mesh into a wireframe by iterating over its faces, collecting all edges and turning those edges into 4 sided polygons. Be aware of the fact that your mesh needs to have faces to be wireframed. You can define the thickness, the material and several other parameters of the generated wireframe dynamically via the given modifier options.

### Options

- Thickness** The depth or size of the wireframes.
- Offset** A value between  $-1$  and  $1$  to change whether the wireframes are generated inside or outside the original mesh. Set to zero, *Offset* will center the wireframes around the original edges.
- Vertex Group** Restrict the modifier to only this vertex group.
  - Invert** Inverts the vertex group weights.
- Crease Edges** This option is intended for usage with the [Subdivision Surface](#) modifier. Enable this option to crease edges on their junctions and prevent large curved intersections.
  - Crease Weight** Define how much crease (between  $0$  = no and  $1$  = full) the junctions should receive.

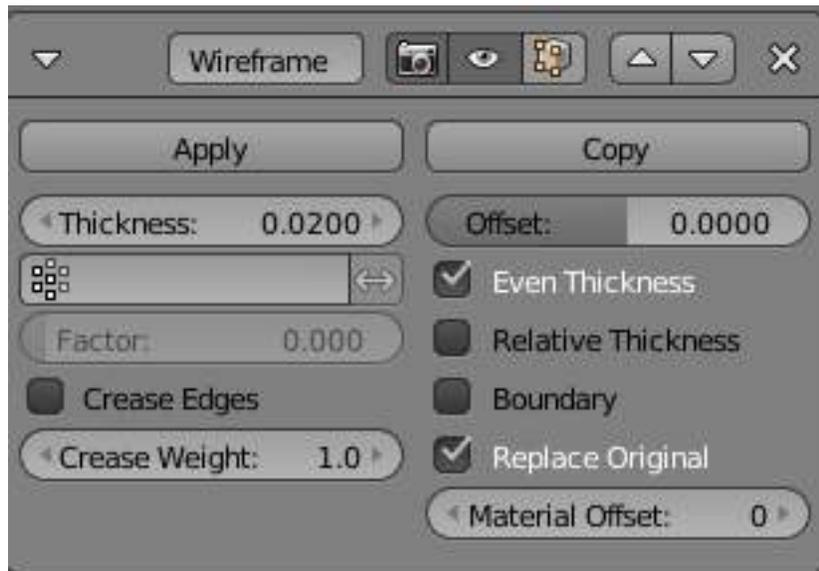


Fig. 2.711: Wireframe Modifier

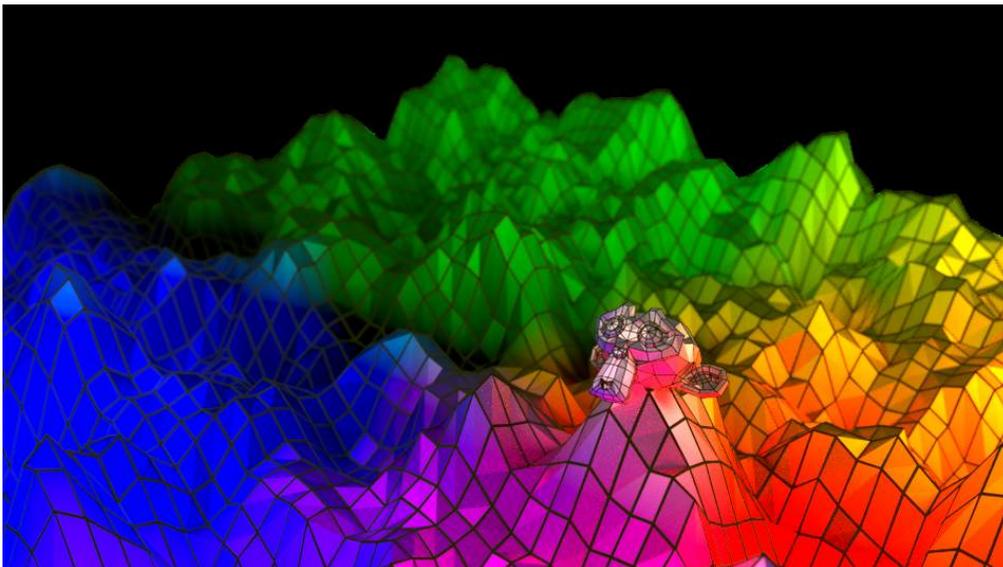


Fig. 2.712: Wireframes on a displaced plane. In this example, the wireframes carry a second (dark) material while the displaced plane uses its original one.

**Even Thickness** Maintain thickness by adjusting for sharp corners. Sometimes improves quality but also increases computation time.

**Relative Thickness** Determine edge thickness by the length of the edge - longer edges are thicker.

**Boundary** Creates wireframes on mesh island boundaries.

**Replace Original** If this option is enabled, the original mesh is replaced by the generated wireframe. If not, the wireframe is generated on top of it.

**Material Offset** Uses the chosen material index as the material for the wireframe; this is applied as an offset from the first material.

### Examples

When you got more Faces that meet at one point they are forming a star like pattern like seen in the examples below.

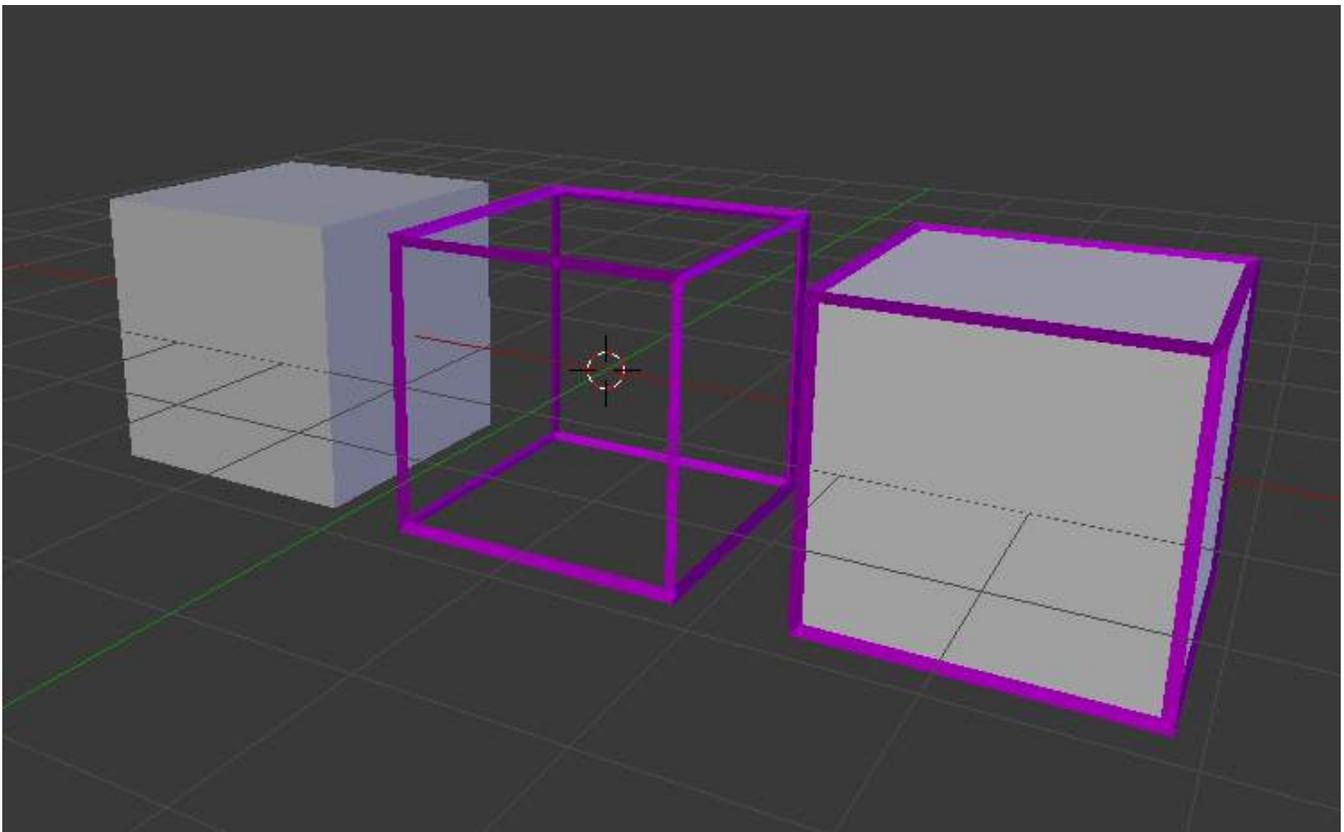


Fig. 2.713: Original / Wireframe / Original+Wireframe

**Warning:** Wireframe thickness is an approximation. While *Even Thickness* should yield good results in many cases, skinny faces can cause ugly spikes. In this case you can either reduce the extreme angles in the geometry or disable the *Even Thickness* option.

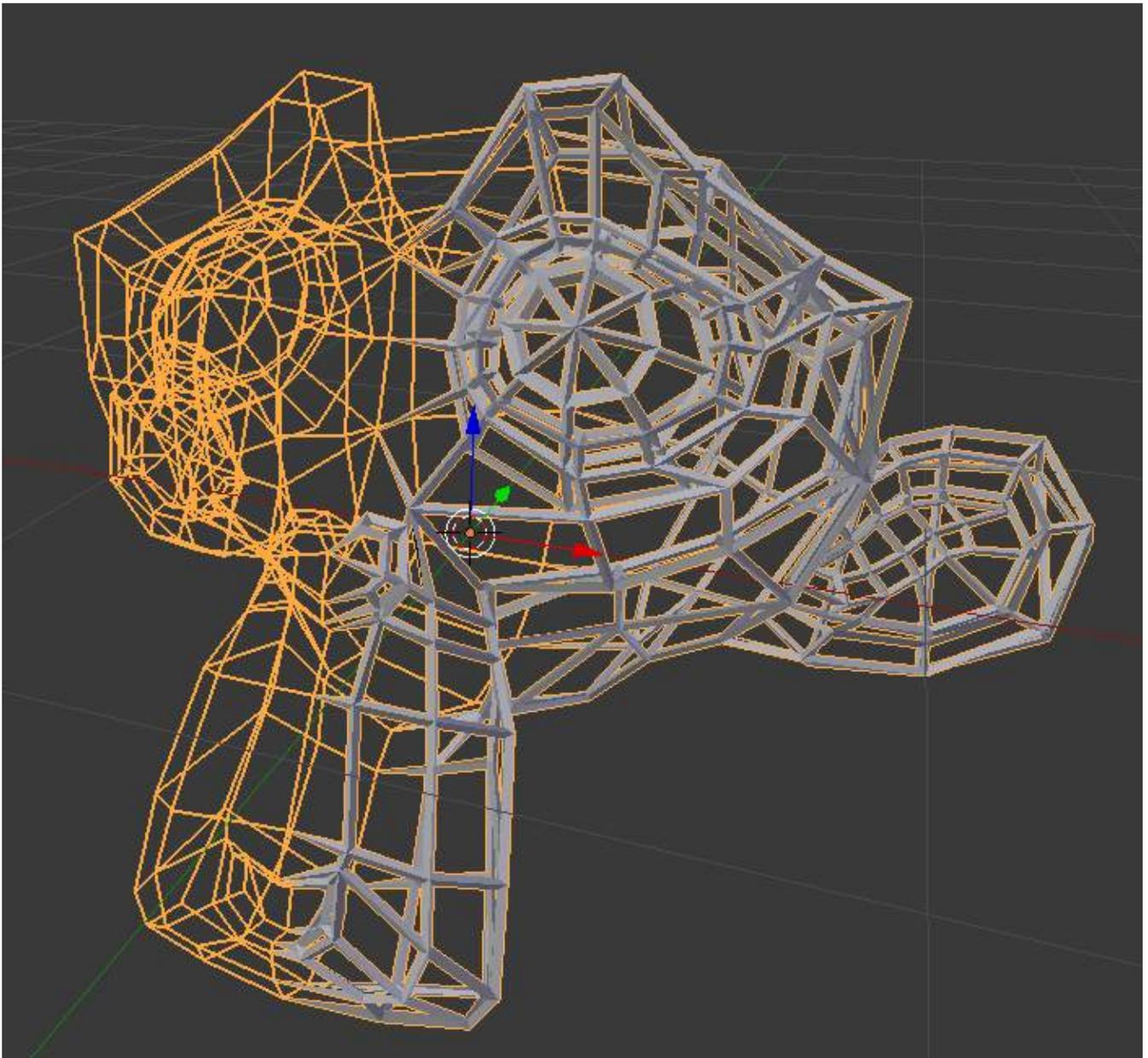


Fig. 2.714: VGroup weighting: One half 0 weighted, one half 1 weighted

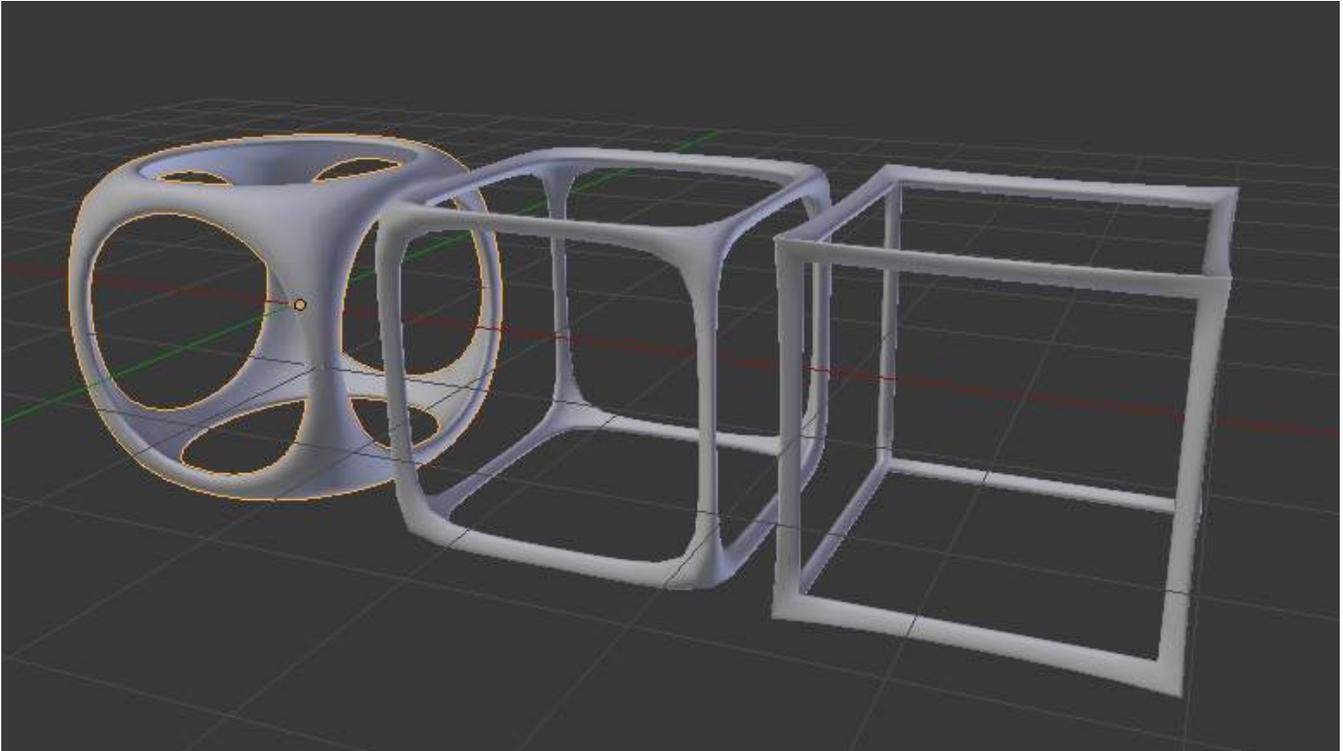


Fig. 2.715: Cube+Subsurf with 0 / 0.5 / 1 crease weight

## 2.4.5 Deform

### Armature Modifier

The *Armature* modifier is used for building skeletal systems for animating the poses of characters and anything else which needs to be posed.

By adding an armature system to an object, that object can be deformed accurately so that geometry doesn't have to be animated by hand.

---

**Note:** For more details on armatures usage, see [this chapter](#).

---

### Options

**Object** The name of the armature object used by this modifier.

**Preserve Volume** Use quaternions for preserving volume of object during deformation. It can be better in many situations.

**Vertex Group** The name of a vertex group of the object, the weights of which will be used to determine the influence of this *Armature* modifier's result when mixing it with the results from other *Armature* ones.

Only meaningful when having at least two of these modifiers on the same object, with *Multi Modifier* activated.

**Invert** Inverts the influence set by the vertex group defined in previous setting (i.e. reverses the weight values of this group).

**Multi Modifier** Use the same data as a previous modifier (usually also an *Armature* modifier) as input. This allows you to use several armatures to deform the same object, all based on the "non-deformed" data (i.e. this avoids having the second

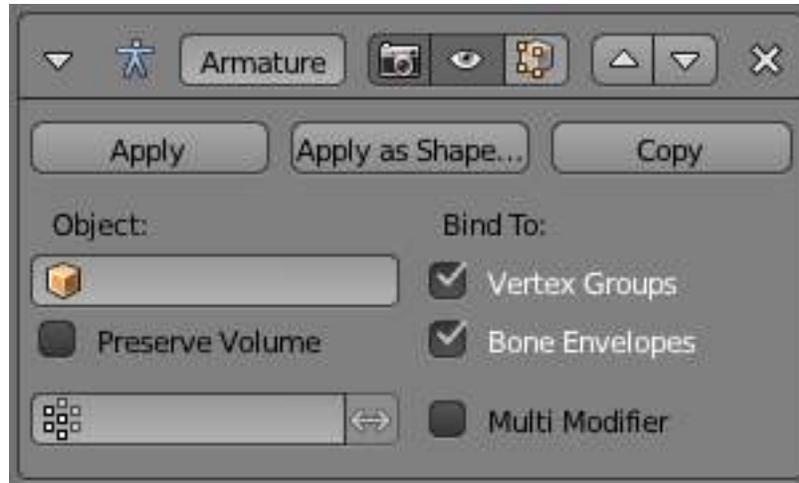


Fig. 2.716: Armature modifier

*Armature* modifier deform the result of the first one...).

The results of the *Armature* modifiers are then mixed together, using the weights of the *Vertex Group* as “mixing guides”.

**Bind To** Method to bind the armature to the mesh.

**Vertex Groups** When enabled, bones of a given name will deform vertices which belong to vertex groups of the same name.

**Bone Envelopes** When enabled, bones will deform vertices near them (defined by each bones envelope radius) Enable/Disable bone envelopes defining the deformation (i.e. bones deform vertices in their neighborhood).

---

**Tip:** Armature modifiers can quickly be added to objects using the parenting shortcut (`Ctrl-P`) when the active object is an armature.

---

## Cast Modifier

This modifier shifts the shape of a mesh, curve, surface or lattice to any of a few pre-defined shapes (sphere, cylinder, cuboid).

It is equivalent to the *To Sphere* tool in *Edit Mode* (*Mesh* → *Transform* → *To Sphere* or `Alt-Shift-S`) and what other programs call “Spherify” or “Spherize”, but, as written above, it is not limited to casting to a sphere.

---

**Tip:** The [Smooth](#) modifier is a good companion to *Cast*, since the cast shape sometimes needs smoothing to look nicer or even to fix shading artifacts.

---

**Note:** For performance reasons, this modifier only works with local coordinates. If the modified object looks wrong, you may need to apply its rotation (`Ctrl-A`), especially when casting to a cylinder.

---

## Options

**Cast Type** Menu to choose cast type (target shape): *Sphere*, *Cylinder* or *Cuboid*.

**X, Y, Z** Toggle buttons to enable/disable the modifier in the X, Y, Z axes directions (X and Y only for *Cylinder* cast type).

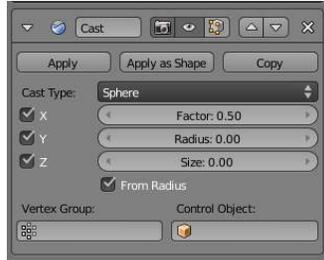


Fig. 2.717: Cast Modifier

**Factor** The factor to control blending between original and cast vertex positions. It's a linear interpolation: 0.0 gives original coordinates (i.e. modifier has no effect), 1.0 casts to the target shape. Values below 0.0 or above 1.0 exaggerate the deformation, sometimes in interesting ways.

**Radius** If non-zero, this radius defines a sphere of influence. Vertices outside it are not affected by the modifier.

**Size** Alternative size for the projected shape. If zero, it is defined by the initial shape and the control object, if any.

**From radius** If activated, calculate *Size* from *Radius*, for smoother results.

**Vertex Group** A vertex group name, to restrict the effect to the vertices in it only. This allows for selective, real-time casting, by painting vertex weights.

**Control Object** The name of an object to control the effect. The location of this object's center defines the center of the projection. Also, its size and rotation transform the projected vertices.

---

**Hint:** Animating (keyframing) this control object also animates the modified object.

---

### Example



Fig. 2.718: Top: Suzanne without modifiers. Middle: Suzanne with each type of Cast Modifier (Sphere, Cylinder and Cuboid). Bottom: Same as above, but now only X axis is enabled. [Sample blend file](#)

### Corrective Smooth

This modifier is used to reduce highly distorted areas of a mesh by smoothing the deformations.

This is typically useful *after* an armature modifier, where distortion around joints may be hard to avoid, even with careful weight painting.

To use this modifier effectively, it's useful to understand the basics of how it works.

**Rest State** Used as a reference to detect highly distorted areas.

*The original vertex locations are used by default.*

**Smoothing** Many options for this modifier relate to smoothing which is used internally to correct the distorted regions.

### Options

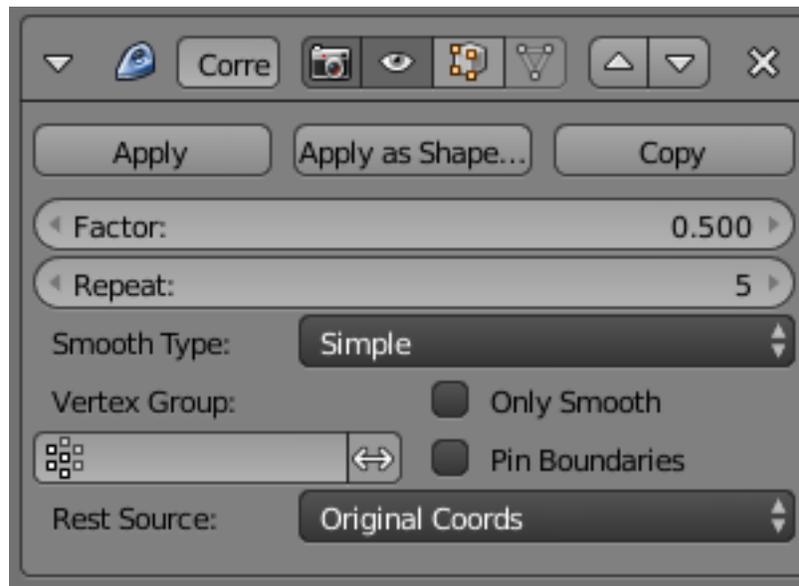


Fig. 2.719: Corrective smooth modifier

The modifier also uses a *Rest* state, to use as a reference Internally this modifier uses smoothing, so some of the options adjust the kind of smoothing.

**Factor** The factor to control the smoothing amount. Higher values will increase the effect. Values outside this range (above 1.0 or below 0.0) distort the mesh.

**Repeat** The number of smoothing iterations.

*Higher values generally improve the quality of the smoothing but slow down the operation also.*

**Smooth Type** Select the smoothing method used.

**Simple** This simply relaxes vertices to their connected edges.

**Length Weight** Uses a method of relaxing that weights by the distance of surrounding vertices.

*Can give higher quality smoothing in some cases, better preserving the shape of the original form.*

**Vertex Group** Use to manually select regions to smooth.

**Only Smooth** This option is included to preview the smoothing used, before correction is applied.

**Pin Boundaries** Prevent boundary vertices from smoothing.

**Rest Source** Select the source for reference vertex positions that defines the undeformed state.

**Original Coords** Use the original input vertex positions.

*This relies on the original mesh having the same number of vertices as the original mesh*

**Bind Coords** Optionally you may bind the modifier to a specific state.

*This is required if there are constructive modifiers such as subsurf or mirror being applied before this modifier in the stack.*

### Example

Here is an example of a character using a simple rig using only bone envelopes (*no weight painting*).

Armature Only	Armature & Corrective Smooth

### Curve Modifier

The Curve Modifier provides a simple but efficient method of deforming a mesh along a curve object.

The Curve Modifier works on a (global) dominant axis, X, Y, or Z. This means that when you move your mesh in the dominant direction (by default, the X-axis), the mesh will traverse along the curve. Moving the mesh perpendicularly to this axis, the object will move closer or further away from the curve.

When you move the object beyond the curve endings the object will continue to deform based on the direction vector of the curve endings.

### Options

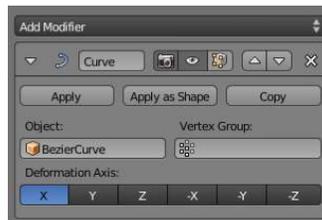


Fig. 2.720: Curve modifier

**Object** The name of the curve object that will affect the deformed object.

**Vertex Group** A vertex group name within the deformed object. The modifier will only affect vertices assigned to this group.

**Deformation Axis** X, Y, Z, -X, -Y, -Z: This is the axis that the curve deforms along.

### Example

Let's make a simple example:

- Remove default cube object from scene and add a Monkey (*Add* → *Mesh* → *Monkey*)
- Now add a curve (*Add* → *Curve* → *Bezier Curve*)
- While in Edit Mode, move the control points of the curve as shown in (*Edit Curve*), then exit Edit Mode (Tab).
- Select the Monkey (RMB) in *Object mode*

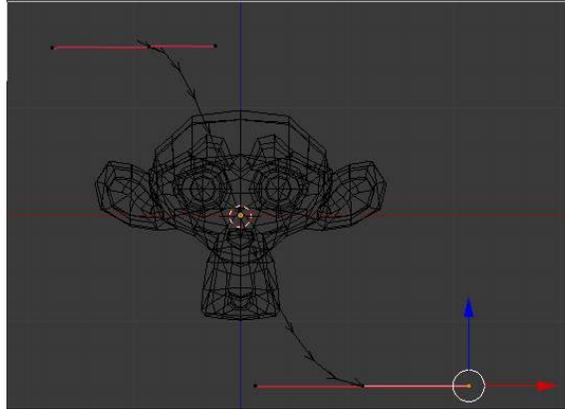


Fig. 2.721: Edit Curve.

- Assign the curve to the modifier, as shown below. The Monkey should be positioned on the curve:

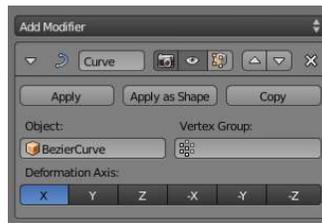


Fig. 2.722: Assign the Bezier curve to the Curve modifier (for Monkey)

- Now if you select the Monkey, and move it in the Y-direction (G, Y), the monkey will deform nicely along the curve.

**Tip:** If you press MMB while moving the Monkey you will constrain the movement to one axis only.

- In the image above you can see the Monkey at different positions along the curve. To get a cleaner view over the deformation, a [Subsurf](#) modifier with two subdivision levels was applied, and [smooth](#) shading was used.

## Displace Modifier

The Displace modifier displaces vertices in a mesh based on the intensity of a texture. Either procedural or image textures can be used. The displacement can be along a particular local axis, along the vertex normal, or the separate RGB components of the texture can be used to displace vertices in the local X, Y and Z directions simultaneously (sometimes referred to as *Vector Displacement*).

### Options

**Texture** The name of the texture from which the displacement for each vertex is derived. If this field is empty, the modifier defaults to 1.0 (white).

**Direction** The direction along which to displace the vertices. Can be one of the following:

**X, Y, Z** Displace along a local axis.

**Normal** Displace along vertex normal.

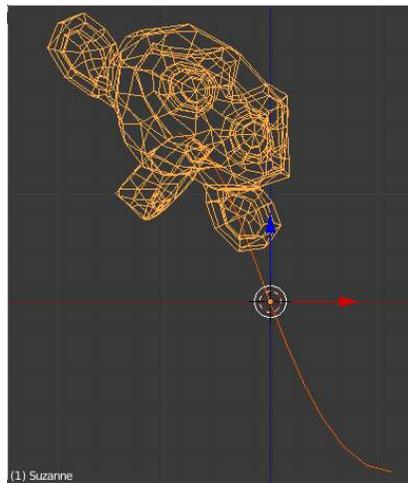


Fig. 2.723: Monkey on a Curve.

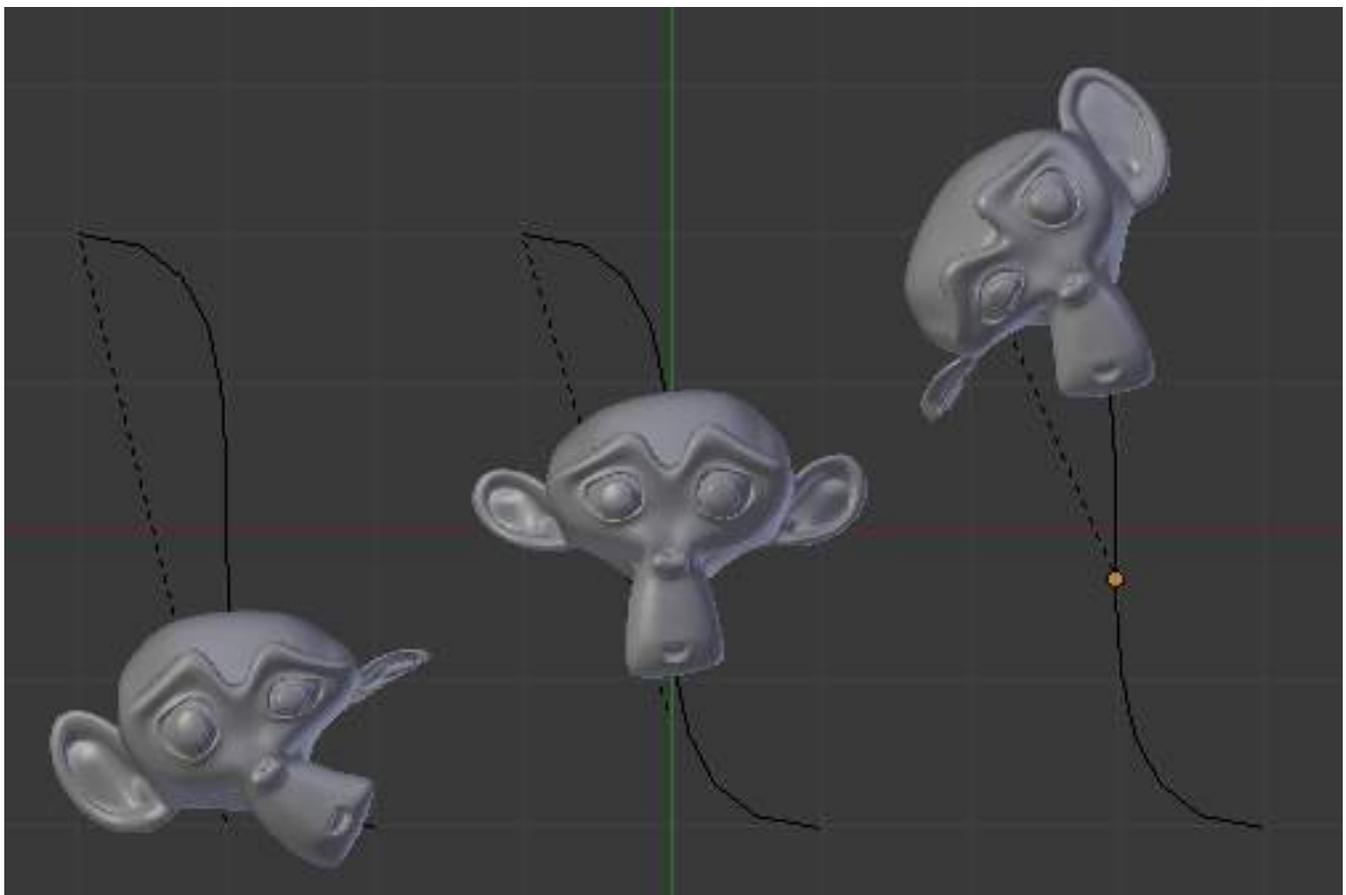


Fig. 2.724: Monkey deformations.

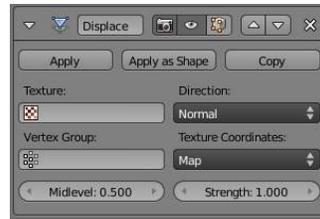


Fig. 2.725: Displace modifier

**RGB to XYZ** Displace along local XYZ axes individually using the RGB components of the texture (Red values displaced along the X-axis, Green along the Y, Blue along the Z). This is sometimes referred to as *Vector Displacement*.

**Texture Coordinates** The texture coordinate system to use when retrieving values from the texture for each vertex. Can be one of the following:

**UV** Take texture coordinates from face UV coordinates.

**UV Map** The UV coordinate layer from which to take texture coordinates. If the object has no UV coordinates, it uses the *Local* coordinate system. If this field is blank, but there is a UV coordinate layer available (e.g. just after adding the first UV layer to the mesh), it will be overwritten with the currently active UV layer.

---

**Note:** Since UV coordinates are specified per face, the UV texture coordinate system currently determines the UV coordinate for each vertex from the first face encountered which uses that vertex; any other faces using that vertex are ignored. This may lead to artifacts if the mesh has non-contiguous UV coordinates.

---

**Object** Take the texture coordinates from another object's coordinate system (specified by the *Object* field).

**Object** The object from which to take texture coordinates. Moving the object will therefore alter the coordinates of the texture mapping.

Take note that moving the original object will **also** result in a texture coordinate update. As such, if you need to maintain a displacement coordinate system while moving the modified object, consider parenting the coordinate object to the modified object.

If this field is blank, the *Local* coordinate system is used.

**Global** Take the texture coordinates from the global coordinate system.

**Local** Take the texture coordinates from the object's local coordinate system.

**Vertex Group** The name of a vertex group which is used to control the influence of the modifier. If left empty, the modifier affects all vertices equally.

**Midlevel** The texture value which will be treated as no displacement by the modifier. Texture values below this value will result in negative displacement along the selected direction, while texture values above this value will result in positive displacement.

This is achieved by the equation  $\text{displacement} = \text{texture\_value} - \text{Midlevel}$ . Recall that color/luminosity values are typically between 0.0 and 1.0 in Blender, and not between 0 and 255.

**Strength** The strength of the displacement. After offsetting by the *Midlevel* value, the displacement will be multiplied by the *Strength* value to give the final vertex offset. This is achieved by the equation  $\text{vertex\_offset} = \text{displacement} * \text{Strength}$ . A negative strength can be used to invert the effect of the modifier.

## Hook Modifier

The Hook modifier is used to deform a *Mesh*, *Curve* a *Lattice* using another object (usually an *Empty* or a *Bone* but it can be any object).

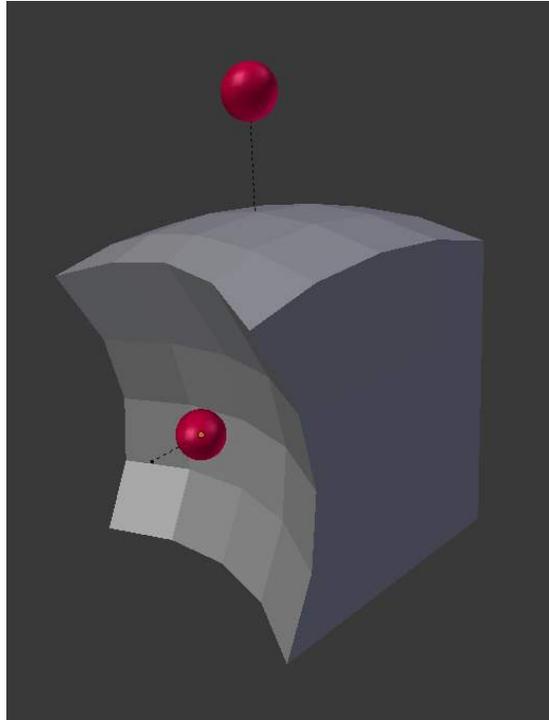


Fig. 2.726: Two spheres used as Hooks to deform a subdivided cube.

As the hook moves, it pulls vertices from the mesh with it. You can think of it as animated [proportional editing](#).

While hooks do not give you the fine control over vertices movement that shape keys do, they have the advantage that you can grab vertices directly for manipulation.

### Options

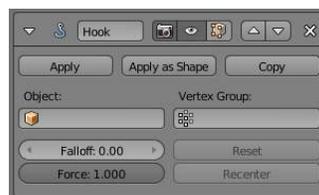


Fig. 2.727: Hook modifier

**Object** The name of the object to hook vertices to.

**Vertex Group** Allows you to define the influence per-vertex.

*Useful when you don't something other than a spherical field of influence.*

**Radius** The size of the hooks influence.

**Strength** Adjust this hooks influence on the vertices, were 0 . 0 makes no change and 1 . 0 follows the hook.

Since multiple hooks can work on the same vertices, you can weight the influence of a hook using this property.

**Falloff Type** This can be used to adjust the kind of curve that the hook has on the mesh, You can also define a custom-curve to get a much higher level of control.

**Uniform Falloff** This setting is useful when using hooks on scaled objects, especially in cases where non-uniform scale would stretch the result of the hook.

*This is especially useful for lattices, where its common to use non-uniform scaling.*

The following settings are only available in Edit Mode:

**Reset** Recalculate and clear the offset transform of hook.

**Recenter** Set hook center to the 3D cursor position.

**Select** Select the vertices affected by this hook.

**Assign** Assigns selected vertices to this hook.

---

**Note:** The hook modifier stores vertex indices from the original mesh to determine what to effect; this means that modifiers that generate geometry, like subsurf, should always be applied **after** the hook modifier; otherwise the generated geometry will be left out of the hook's influence.

---

## Laplacian Smooth Modifier

The Laplacian Smooth modifier allows you to reduce noise on a mesh's surface with minimal changes to its shape.

It can also exaggerate the shape using a negative *Factor*.

The Laplacian Smooth is useful for objects that have been reconstructed from the real world and contain undesirable noise. It removes noise while still preserving desirable geometry as well as the shape of the original model.

The Laplacian Smooth modifier is based on a curvature flow Laplace Beltrami operator in a diffusion equation.

### Options

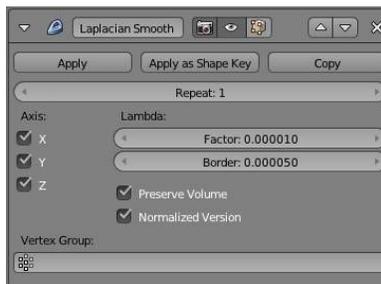


Fig. 2.728: Laplacian Smooth modifier

**Repeat** Repetitions allow you to run the Laplacian smoothing multiple times. Each repetition causes the flow curvature of the mesh to be recalculated again, and as a result it removes more noise with every new iteration using a small *Factor*  $< 1.0$ .

When on 0, no smoothing is done.

---

**Note:** More repetitions will take longer to calculate - beware of doing so on meshes with a large number of vertices.

---

Table 2.13:

Repeat: 10

--	--	--	--

Table 2.14:

Repeat: 10

--	--	--	--

Table 2.15:

Repeat: 10

--	--	--	--

**Factor** Controls the amount of displacement of every vertex along the curvature flow.

- Using a small *Factor*, you can remove noise from the shape without affecting desirable geometry.
- Using a large *Factor*, you get smoothed versions of the shape at the cost of fine geometry details.
- Using a negative *Factor*, you can enhance the shape, preserving desirable geometry.
- When the *Factor* is negative, multiple iterations can magnify the noise.


**Border** Since there is no way to calculate the curvature flow on border edges, they must be controlled separately. Border edges are smoothed using a much simpler method, using this property to control the influence.

Positive values will smooth the vertex positions, while negative values will “enhance” them by transforming them in the opposite direction.

Table 2.16:

Border:

10.0

--	--	--	--

Table 2.17:

Border:

20.0

--	--	--	--

Table 2.18:

Border:

-200.0

--	--	--	--

**X, Y, Z** Toggle buttons to enable/disable deforming vertices in the X, Y and/or Z axis directions.


**Preserve Volume** The smoothing process can produce shrinkage. That is significant for large *Factor* or large *Repeat* values; to reduce that effect you can use this option.

--	--	--	--

**Vertex Group** A vertex group name, to constrain the effect to a group of vertices only. Allows for selective, real-time smoothing or enhancing, by painting vertex weights.

Original Geometry	No Group Chosen	Vertex Weights	Result

**Normalized** When enabled, the results will depend on face sizes. When disabled, geometry spikes may occur.



### Hints

Meshes with a great number of vertices, more than ten thousand (10,000), may take several minutes for processing; you can use small portions of the mesh for testing before executing the modifier on the entire model.

### Examples



### See Also

- [Smooth Modifier](#)

## Laplacian Deform Modifier

The Laplacian Deform modifier allows you to pose a mesh while preserving geometric details of the surface.

The user defines a set of ‘anchor’ vertices, and then moves some of them around. The modifier keeps the rest of the anchor vertices in fixed positions, and calculates the best possible locations of all the remaining vertices to preserve the original geometric details.

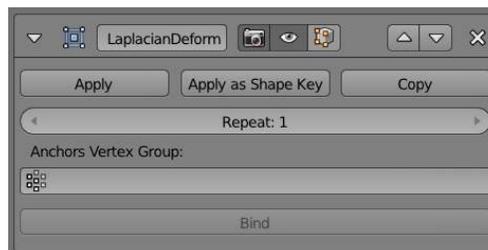
This modifier captures the geometric details with the uses of differential coordinates. The differential coordinates captures the local geometric information how curvature and direction of a vertex based on its neighbors.

---

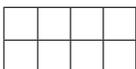
**Note:** You must define a *Anchor Vertex Group*. Without a vertex group modifier does nothing.

---

### Options



**Repeat** Repetitions iteratively improve the solution found. The objective is to find the rotation of the differential coordinates preserving the best possible geometric detail. Details are retained better if more repetitions are used, however it will take longer to calculate.



**Anchors Vertex Group** A vertex group name, to define the group of vertices that the user uses to transform the model. The weight of each vertex does not affect the behavior of the modifier; the method only takes into account vertices with weight greater than 0.

**Bind** The *Bind* button is what tells the Laplacian Deform modifier to actually capture the geometry details of the object, so that altering the anchors vertices actually alters the shape of the deformed object.

**Unbind** After binding the modifier, you may later decide to make changes to the Anchors Vertex Group. To do so you will first need to *Unbind* the modifier before binding again.

### Error Messages

**Vertex group *group\_name* is not valid** This message is displayed when a user deletes a Vertex Group or when the user changes the name of the Vertex Group.

**Verts changed from *X* to *Y*** This message is displayed when a user add or delete verts to the mesh.

**Edges changed from *X* to *Y*** This message is displayed when a user add or delete edges to the mesh.

**The system did not find a solution** This message is displayed if the solver SuperLU did not find a solution for the linear system.

---

**Note:** If the mesh is dense, with a number of vertices greater than 100,000, then it is possible that the nonlinear optimization system will fail.

---

### History

Laplacian Surface Editing is a method developed by Olga Sorkine and others in 2004. This method preserves geometric details as much as possible while the user makes editing operations. This method uses *differential coordinates* corresponding to the difference between a vector and the weighted average of its neighbors to represent the local geometric detail of the mesh.

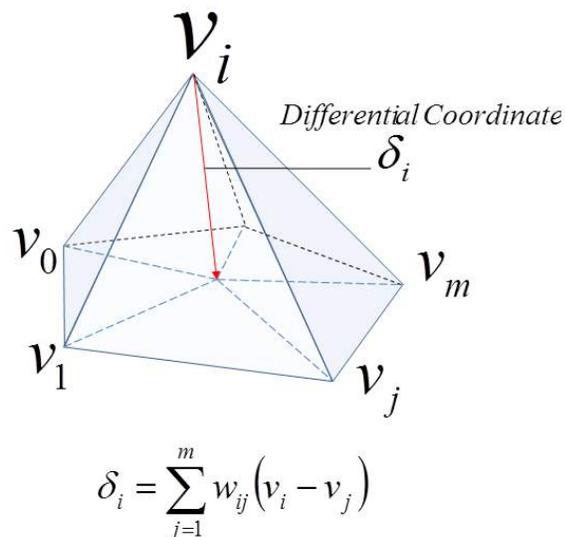


Fig. 2.853: Differential Coordinate

### See Also

- Laplacian Surface Editing (Original paper)

- Differential Coordinates for Interactive Mesh Editing

## Lattice Modifier

The Lattice modifier deforms the base object according to the shape of a Lattice object.

### Options



Fig. 2.854: Lattice modifier.

**Object** The Lattice object with which to deform the base object.

**Vertex Group** An optional vertex group name which lets you limit the modifier effect to a part of the base mesh.

**Strength** A factor to control blending between original and deformed vertex positions.

### Usage

A lattice consists of a three-dimensional non-renderable grid of vertices. Its main use is to give extra deformation capabilities to the underlying object it controls (either *via* a modifier, or as its parent). Objects to be deformed can be meshes, curves, surfaces, text, lattices and even particles.

The lattice should be scaled and moved to fit around your object in object mode. Any scaling applied to the object in edit mode will result in the object deforming. This includes applying scale with `Ctrl-A` as this will achieve the same result as scaling the lattice in edit mode, and therefore the object.

**Tip:** A Lattice Modifier can quickly be added to selected objects by selecting them all, then selecting the lattice object last and pressing `Ctrl-P` and choosing *Lattice Deform*. This will both add Lattice modifiers to the selected objects and parent them to the lattice.

### Hints

Why would you use a lattice to deform a mesh instead of deforming the mesh itself in Edit Mode ? There are a couple of reasons for that:

- If your object has a large number of vertices, it would be difficult to edit portions of it quickly in Edit Mode. Using a lattice will allow you to deform large portions efficiently.
- The smooth deformation you get from a lattice modifier can be hard to achieve manually in Edit Mode.
- Multiple objects can use the same lattice, thus allowing you to edit multiple objects at once.

- Like all modifiers, it is non-destructive. Meaning all changes happen on top of the original geometry, which you can still go back and edit without affecting the deformation.
- A lattice does not affect the texture coordinates of a mesh's surface.

**Note:** When using a lattice to deform particles, always remember to place the Lattice Modifier after the Particle System Modifier. Read more about the importance of the modifier stack [here](#).

## Mesh Deform Modifier

The Mesh Deform modifier allows an arbitrary mesh (of any closed shape) to act as a deformation cage around another mesh.

### Options

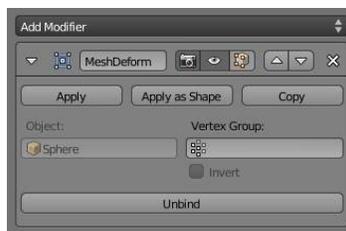


Fig. 2.855: Mesh Deform modifier

The Mesh Deform modifier is reasonably easy to use but it can be very slow to do the calculations needed to properly map the deform mesh cage to the deformed object.

**Object** The name of the mesh object to be used as a deforming mesh cage.

**Vertex Group** An optional vertex group that will be affected by the deforming mesh cage. Vertices not in this group will not be deformed.

**Invert** Inverts the influence of the vertex group defined in the previous setting (reverses the weight values of this group).

**Bind** Links the current vertex positions of both the modified geometry and the deformer *Object* chosen together. An unbound Mesh Deform modifier has no effect - it must be bound so that altering the shape of the deform mesh cage actually alters the shape of the modified object.

**Warning:** Depending on the settings of the Mesh Deform modifier and complexity of the deform mesh cage and/or deformed object, it can take a long time for this operation to complete. This can result in Blender not responding to user's actions until it has completed.

It is also possible that Blender will run out of memory and crash.  
To be safe, save your blend file before proceeding!

**Unbind** When a deformed object has been associated to a deform mesh cage, it can later be disassociated by clicking the *Unbind* button which replaced the *Bind* one. When *Unbind* is clicked, the *deform mesh cage* will keep its current shape; it will not reset itself back to its original start shape.

If you need its original shape, you will have to save a copy of it before you alter it.

The deformed object will, however, reset back to its original shape that it had before it was bound to the deform mesh cage.

**Precision** The *Precision* numeric slider field controls the accuracy with which the deform mesh cage alters the deformed object, when the points on the cage are moved. Raising this value higher can greatly increase the time it takes the *Mesh Deform* modifier to complete its binding calculations, but it will get more accurate cage mapping to the deformed object.

This setting becomes unavailable once a cage has been bound.

**Dynamic** When activated, other mesh altering features (such as other modifiers and shape keys) are taken into account when binding, increasing deformation quality.

It is deactivated by default to save memory and processing time when binding... Like with *Precision*, this setting is unavailable once a cage has been bound.

## Hints

- Ensure that the normals on the cage mesh point to the outside; they are used to determine the inside and outside of the cage.
- Besides the outer cage, more faces within the cage, either loose or forming another smaller cage, can be used for extra control. Such smaller cages may also overlap with the main cage; for example, to get extra control over eyes, two small sphere cages could be added around them.

## See Also

- The [Lattice modifier](#).
- [Original paper](#)

## Shrinkwrap Modifier

The *Shrinkwrap* modifier allows an object to “shrink” to the surface of another object. It moves each vertex of the object being modified to the closest position on the surface of the given mesh (using one of the three methods available).

It can be applied to meshes, lattices, curves, surfaces and texts.

## Options

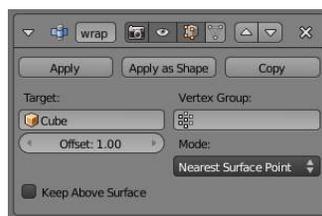


Fig. 2.856: Nearest Surface Point

**Target** Shrink target, the mesh to shrink to/wrap around.

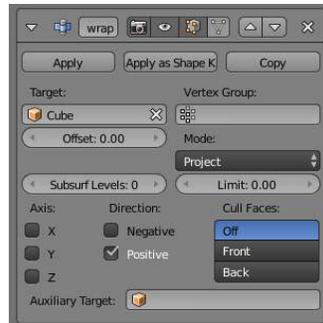
**Vertex Group** The vertex group to control whether and how much each vertex is displaced to its target position. If a vertex is not a member of this group, it is not displaced (same as weight 0).

**Offset** The distance that must be kept from the calculated target position, in Blender Units.

**Mode** This drop-down list specifies the method to be used to determine the nearest point on the target’s surface for each vertex of the modified object. Some options will add some extra, specific controls to the panel.

**Nearest Surface Point** This will select the nearest point over the surface of the shrink target. It adds the extra option *Above surface*, which always keep the computed vertices above their “floor faces”. This is only meaningful when *Offset* is not null.

## Projection



This will project vertices along a chosen axis until they touch the shrink target. Vertices that never touch the shrink target are left in their original position.

**Subsurf Levels** This applies a (temporary) *Catmull-Clark* subsurf to the modified object, before computing the wrap when using Projection mode.

**Limit** This is a distance limit between original vertex and surface. If the distance is larger than this limit vertex wouldn't be projected onto the surface,

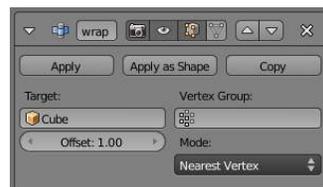
**X, Y, Z** Along which local axis of the modified object the projection is done. These options can be combined with each other, yielding a “median axis” of projection.

**Negative, Positive** This allows you to select the allowed direction(s) of the shrink along the selected axis. With more than one *Shrinkwrap* modifier, negative and positive axes can be combined.

**Cull Faces** This allows you to prevent any projection over the “front side” (respectively the “back side”) of the target's faces. The “side” of a face is determined by its normal (front being the side “from where” the normal “originates”).

**Auxiliary Target** An additional object to project over.

## Nearest Vertex



This will snap vertices to the nearest vertex of the shrink target. It adds no extra options.

### See also:

[Shrinkwrap Constraint](#)

## Simple Deform Modifier

The Simple Deform modifier allows easy application of a simple deformation to an object (meshes, lattices, curves, surfaces and texts are supported).

Using another object, it's possible to define the axis and origin of the deformation, allowing application of very different effects.

### Options

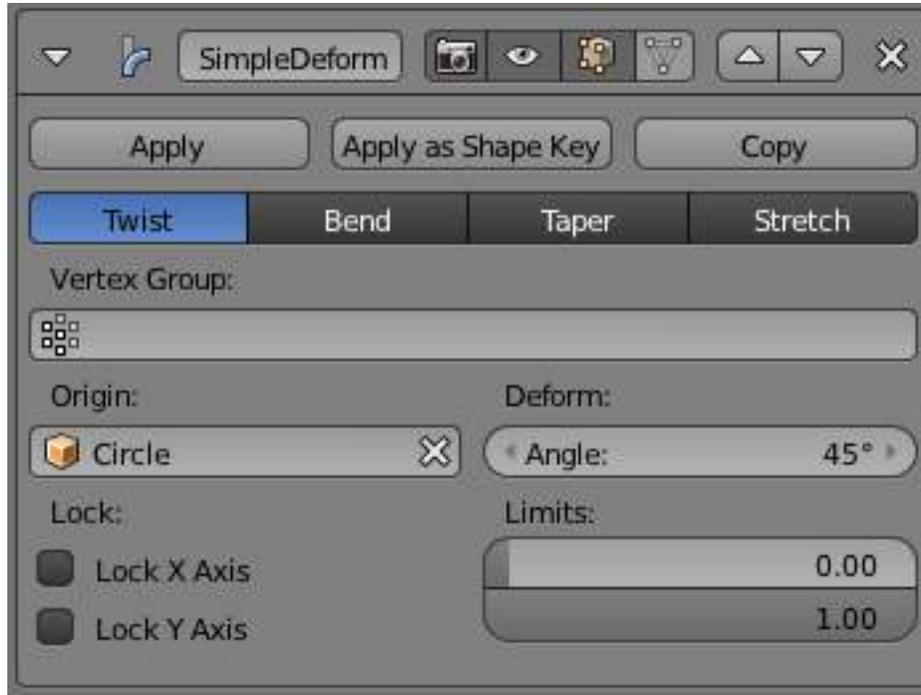


Fig. 2.857: Simple Deform

**Mode** This drop-down list defines the deform function applied, among four available:

**Twist** Rotates around the Z axis.

**Bend** Bends the mesh over the Z axis.

**Taper** Linearly scales along Z axis.

**Stretch** Stretches the object along the Z axis (negative *Factor* leads to squash), preserving volume by scaling inversely on the X and Y axes..

**Vertex Group** The name of the vertex group that indicates whether and how much each vertex is influenced by the deformation.

**Origin** The name of an object that defines the origin of deformation (usually an empty). This object can be:

- Rotated to control the axis (its local Z-axis is now used as the deformation axis).
- Translated to control the origin of deformation.
- Scaled to change the deform factor.

---

**Note:** When the object controlling the origin (the one in the *Origin* field) is a child of the deformed object, this creates a cyclic dependency in Blender's data system. The workaround is to create a new empty and parent both objects to it.

---

**Angle/Factor** The amount of deformation. Can be negative to reverse the deformation.

**Limits** These settings allow you to set the lower and upper limits of the deformation. The upper limit can't be lower than lower limit.

**Lock X Axis / Lock Y Axis (Taper and Stretch modes only)** These controls whether the X and/or Y coordinates are allowed to change or not. Thus it is possible to squash the X coordinates of an object and keep the Y coordinates intact.

## Smooth Modifier

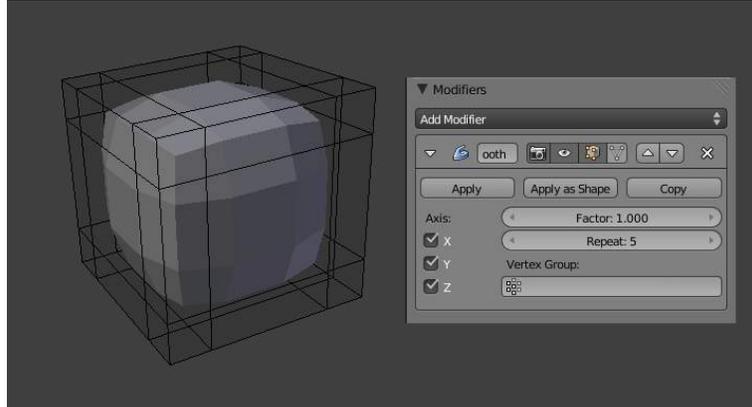


Fig. 2.858: Smooth modifier applied to a subdivided cube

This modifier smooths a mesh by flattening the angles between adjacent faces in it, just like *Specials* → *Smooth* in Edit Mode. It smooths without subdividing the mesh - the number of vertices remains the same.

This modifier is not limited to smoothing, though. Its control factor can be configured outside the  $0.0 - 1.0$  range (including negative values), which can result in interesting deformations.

### Options

**X, Y, Z** Toggle buttons to enable/disable the modifier in the X, Y and/or Z axes directions.

**Factor** The factor to control the smoothing amount. Higher values will increase the effect. Values outside this range (above  $1.0$  or below  $0.0$ ) distort the mesh.

**Repeat** The number of smoothing iterations, equivalent to pressing the `Smooth` button multiple times.

**Vertex Group** A vertex group name, to restrict the effect to the vertices in it only. This allows for selective, real-time smoothing, by painting vertex weights.

### Algorithm

The calculation done by the Smooth Modifier is a simple and logical one, and can be thought of as the geometric equivalent of blurring images.

Each new vertex position is simply the average position of surrounding vertices (the vertices connected to the same edge as it).

## Warp Modifier

This deformation modifier can be used to warp parts of a mesh to a new location in a very flexible way by using 2 objects to select the “from” and “to” regions, with options for using a curve falloff, texture and vertex group.

The Warp Modifier is a bit tricky at first, but it helps to understand how it works. The modifier requires two points, specified by object centers. The “from” point designates a point in space that is pulled toward the “to” point. It is akin to using the [Proportional Editing](#) in Edit Mode.

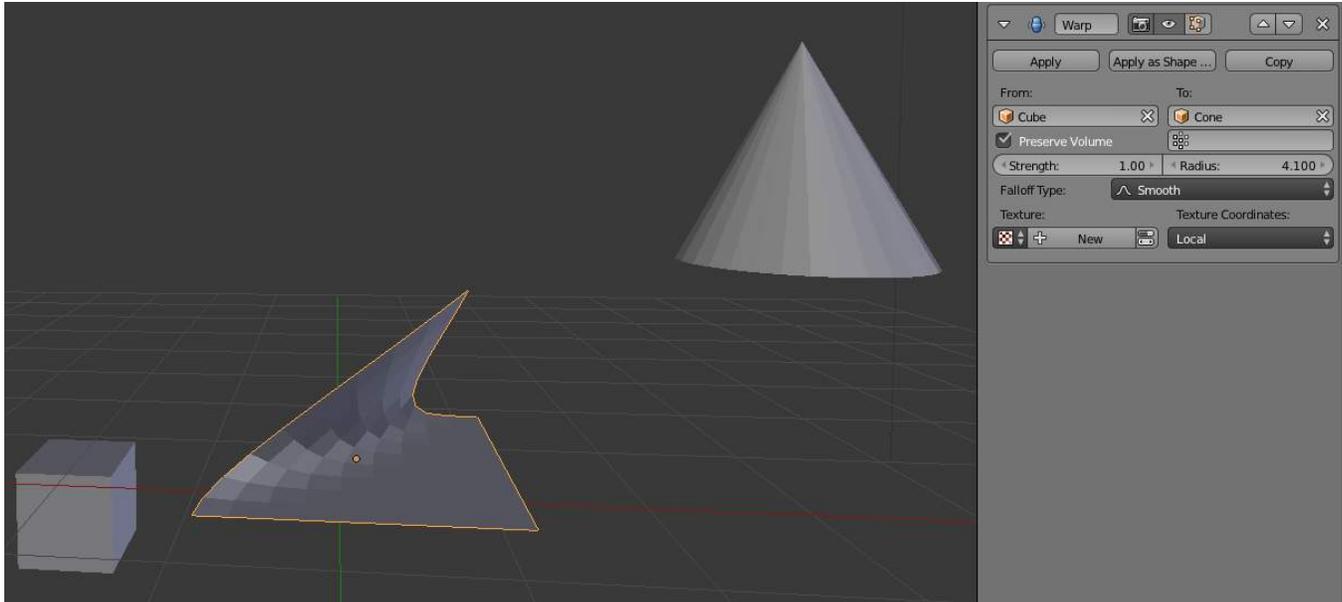


Fig. 2.859: Warp modifier applied to a grid

### Options

**From:** Specify the origin object transformation of the warp.

**To:** Specify the destination object transformation of the warp.

**Preserve Volume** Enables volume preservation when rotating one of the transforms.

**Vertex Group** Limit the deformation to a specific vertex group.

**Strength** Sets how strong the effect is.

**Radius** Sets the distance from the transforms that can be warped by the transform handles.

**Falloff Type** Sets the way the strength of the warp change as it goes from the center of the transform to the Radius value. See [Proportional Editing](#) for descriptions of the falloff types.

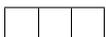
**Texture** Specify a texture the strength is offset by to create variations in the displacement.

**Texture Coordinates** Set the way textures are applied to the mesh when using a textured warp.

**Object** Specify an object to use when set to Object.

**UV Layer** Specify a UV layer when set to UV.

### Wave Modifier



The Wave modifier adds a ripple-like motion to an object's geometry.

This modifier is available for meshes, lattices, curves, surfaces and texts, with one restriction for non-mesh objects: Activating *Normals* or typing a name in *VGroup* will simply deactivate the modifier.

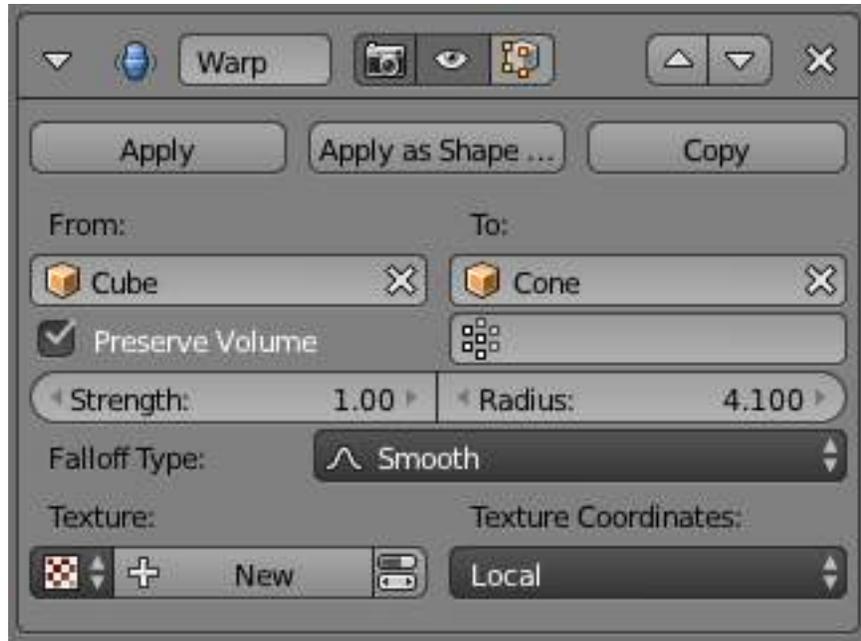


Fig. 2.860: Warp modifier

## Options

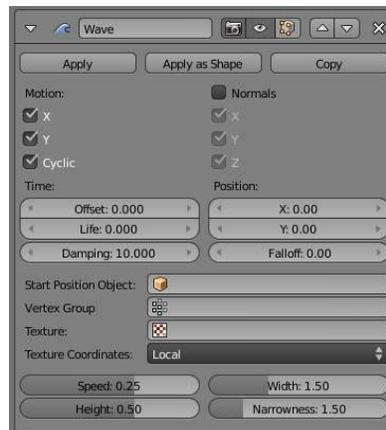


Fig. 2.867: Wave modifier

## Motion

**X, Y** The wave effect deforms vertices/control points in the Z direction, originating from the given starting point and propagating along the object with circular wave fronts (if both X and Y are enabled), or with rectilinear wave fronts (if only one axis is enabled), then parallel to the axis corresponding to the X or Y button activated.

**Cyclic** Repeats the waves cyclically, rather than a single pulse.

**Normals** For meshes only. Displaces the mesh along the surface normals (instead of the object's Z-axis).

**Time** Settings to control the animation.

**Offset** Time offset in frames. The frame at which the wave begins (if *Speed* is positive), or ends (if *Speed* is negative).

Use a negative frame number to prime and pre-start the waves.

**Life** Duration of animation in frames. When set to zero, loops the animation forever.

**Damping** An additional number of frames in which the wave slowly damps from the *Height* value to zero after *Life* is reached. The dampening occurs for all the ripples and begins in the first frame after the *Life* is over. Ripples disappear over *Damping* frames.

### Position

**X, Y** Coordinates of the center of the waves, in the object's local coordinates.

**Falloff** Controls how fast the waves fade out as they travel away from the coordinates above (or those of the *Start Position Object*).

**Start Position Object** Use another object as the reference for the starting position of the wave. Note that you then can animate this object's position, to change the wave's origin across time.

**Vertex Group** For meshes only. A vertex group name, used to control the parts of the mesh affected by the wave effect, and to what extent (using vertex weights).

**Texture** Use this texture to control the object's displacement level. Animated textures can give very interesting results here.

**Texture Coordinates** This menu lets you choose the texture's coordinates for displacement:

**Local** Object's local coordinates.

**Global** Global coordinates.

**Object** Adds an additional field just below, to type in the name of the object from which to get the texture coordinates.

**UV** Adds an extra *UV Layer* property, to select the UV layer to be used.

**Speed** The speed, in BU (for "Blender Units") per frame, of the ripple.

**Height** The height or amplitude, in BU, of the ripple.

**Width** Half of the width, in BU, between the tops of two subsequent ripples (if *Cyclic* is enabled). This has an indirect effect on the ripple amplitude - if the pulses are too near to each other, the wave may not reach the 0 Z-position, so in this case Blender actually lowers the whole wave so that the minimum is zero and, consequently, the maximum is lower than the expected amplitude. See **Technical Details and Hints** below.

**Narrowness** The actual width of each pulse: the higher the value the narrower the pulse. The actual width of the area in which the single pulse is apparent is given by  $4/\text{Narrowness}$ . That is, if *Narrowness* is 1 the pulse is 4 units wide, and if *Narrowness* is 4 the pulse is 1 unit wide.

**Warning:** All the values described above must be multiplied with the corresponding *Scale* values of the object to get the real dimensions.

### Technical Details and Hints

The relationship of the above values is described here:

To obtain a nice wave effect similar to sea waves and close to a sinusoidal wave, make the distance between following ripples and the ripple width equal; that is, the *Narrowness* value must be equal to  $2/\text{Width}$ . E.g. for *Width* = 1, set *Narrow* to 2.

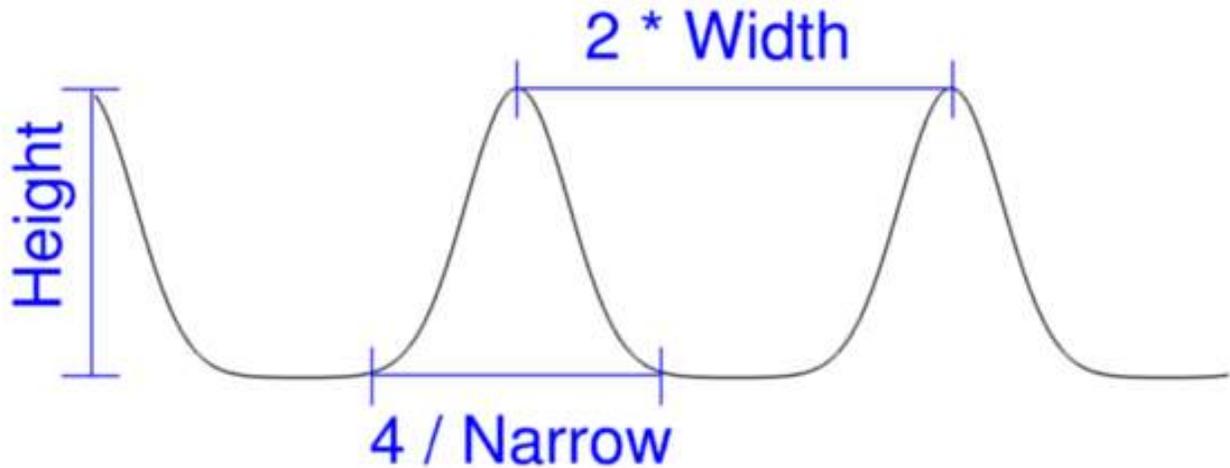


Fig. 2.868: Wave front characteristics.

## 2.4.6 Simulate

### Explode Modifier

The Explode Modifier is used to alter the mesh geometry by moving/rotating its faces in a way that roughly tracks particles emitted by that object, making it look as if the mesh is being exploded (broken apart and pushed outward).

For the Explode Modifier to have a visible effect, there needs to be a particle system on it. The particle system on the mesh is what controls how the mesh will be exploded, and therefore without the particle system the mesh won't appear to alter.

Both the number of emitted particles and number of faces determine how granular the Explode Modifier will be. More faces and more particles will mean more individual pieces.

Here is a [demo video](#) showing a cube with a particle system and Explode Modifier. ([Blend file](#))

---

**Note:** The Explode modifier must come after the Particle System Modifier because the Particle System Modifier has the information needed to drive the Explode Modifier.

---

### Options

**Vertex group** Vertices in this group may not be affected by the Explode Modifier. Vertices with full weight are not affected at all, while vertices with less weight have a higher chance of being affected.

Vertices with no weight will be treated like those which do not belong to the group at all and explode normally.

**Protect** Clean vertex group edges. Depending on the weights assigned to that vertex group; either completely protect those faces from being affected by the Explode Modifier (which would happen if the faces had a weight value of 1) or completely remove protection from those faces (which would happen if the faces had a weight value 0).

**Particle UV** UV map to change with particle age.

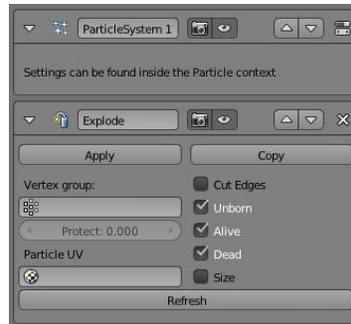


Fig. 2.869: Explode Modifier panel with Particle System Modifier above it

**Cut Edges** Cut face edges for nicer shrapnel

**Unborn** Show mesh when particles are unborn

**Alive** Show mesh when particles are alive

**Dead** Show mesh when particles are dead

**Size** Use particle size for shrapnel

**Refresh** Refresh data in the explode modifier

## Ocean Simulation

Blender's ocean simulation tools take the form of a modifier, to simulate and generate a deforming ocean surface, and associated texture, used to render the simulation data.

Ported from the open source Houdini Ocean Toolkit, it is intended to simulate deep ocean waves and foam.

### Options

#### Geometry Options

##### Geometry

**Generate** Creates a tiled mesh grid that exactly corresponds with the resolution of the simulation data

When generating a mesh surface, the existing mesh object is completely overridden with the ocean grid. A UV channel is also added, mapping the  $0.0 - 1.0$  UV space to the simulation grid.

**Displace** Uses the existing geometry rather than replacing it. Vertices are displaced along the local Z-axis.

**Repeat X, Repeat Y** When generating a mesh surface, controls the number of times the grid is tiled in X and Y directions. UVs for these tiled mesh areas continue outside of the  $0.0 - 1.0$  UV space.

#### Simulator Options

**Time** The time at which the ocean surface is being evaluated. To make an animated ocean, you will need to insert keyframes (RMB) and animate this time value - the speed that the time value is changing will determine the speed of the wave animation

**Depth** The constant depth of the ocean floor under the simulated area. Lower values simulate shallower waters by producing higher frequency details and smaller waves.

**Random Seed** A different seed will produce a different simulation result.



Fig. 2.870: Ocean Modifier Panel

**Resolution** The main control of quality vs speed in the simulation engine. This determines the resolution of the internal 2D grids generated by the simulation.

The internal grids are powers of two of the resolution value, so a resolution value of 16 will create simulation data of size  $256 \times 256$ . The higher the resolution, the more detail will be produced, but the slower it will be to calculate.

---

**Note:** When using the ‘Generate’ modifier geometry option, this resolution value also determines the resolution of the generated mesh surface, equal to the resolution of the internal simulation data.

---

**Size** A simple scaling factor that does not affect the height of the waves or behavior of the simulation.

**Spatial Size** The width of the ocean surface area being simulated, in meters. This also determines the size of the generated mesh, or the displaced area, in Blender units. Of course you can scale the object with ocean modifier in object mode to tweak the apparent size in your scene.

### Wave Options

**Choppiness** The choppiness of the wave peaks. With a choppiness of 0, the ocean surface is only displaced up and down in the Z direction, but with higher choppiness, the waves are also displaced laterally in X and Y, to create sharper wave peaks.

**Scale** An overall scale control for the amplitude of the waves. It approximates the height or depth of the waves above or below zero. Rather than just scaling the ocean object in Z, it scales all aspects of the simulation, displacement in X and Y, and corresponding foam and normals too.

**Alignment** The directionality of the wave shapes due to wind. At a value of 0, the wind and waves are randomly, uniformly oriented. With higher Alignment values, the wind is blowing in a more constant direction, making the waves appear more compressed and aligned to a single direction.

**Direction** When using Alignment, the direction in degrees that the waves are aligned to.

**Damping** When using Alignment, amount that inter-reflected waves are damped out. This has the effect of making the wave motion more directional (not just the wave shape). With damping of 0.0, waves are reflected off each other every direction, with damping of 1.0, these inter-reflected waves are damped out, leaving only waves traveling in the direction of the wind.

**Smallest Wave** A minimum limit for the size of generated waves. Acts similarly to a low-pass filter, removing higher frequency wave detail.

**Wind Velocity** Wind speed in meters/second. With a low velocity, waves are restricted to smaller surface waves.

**Simulation Data Generation Options** By default, the simulator only generates displacement data, since it takes the least amount of work and gives the fastest feedback. Additional simulation data can be generated for rendering as well.

**Generate Normals** Simulates additional normal map data. This can be used by the Ocean texture, when mapped to Normals, as a bump map, and enables generating normal map image sequences when baking.

**Generate Foam** Simulates additional foam data. This can be retrieved by the Ocean texture for use in texturing (perhaps as a mask), and enables generating foam map image sequences when baking.

**Coverage** Tweaks the amount of foam covering the waves, negative values will reduce the amount of foam (leaving only the topmost peaks), positive values will add it. Typically ranges from  $-1.0$  to  $1.0$

**Foam Data Layer Name** Optional name for the vertex data layer, used by the Ocean modifier to store foam maps as vertex colors. This is required for accessing the foam data in the renderer.

### Baking

Rather than simulating the ocean data live, the ocean data can be baked to disk. When a simulation is baked, the simulator engine is completely bypassed, and the modifier/texture retrieves all information from the baked files.

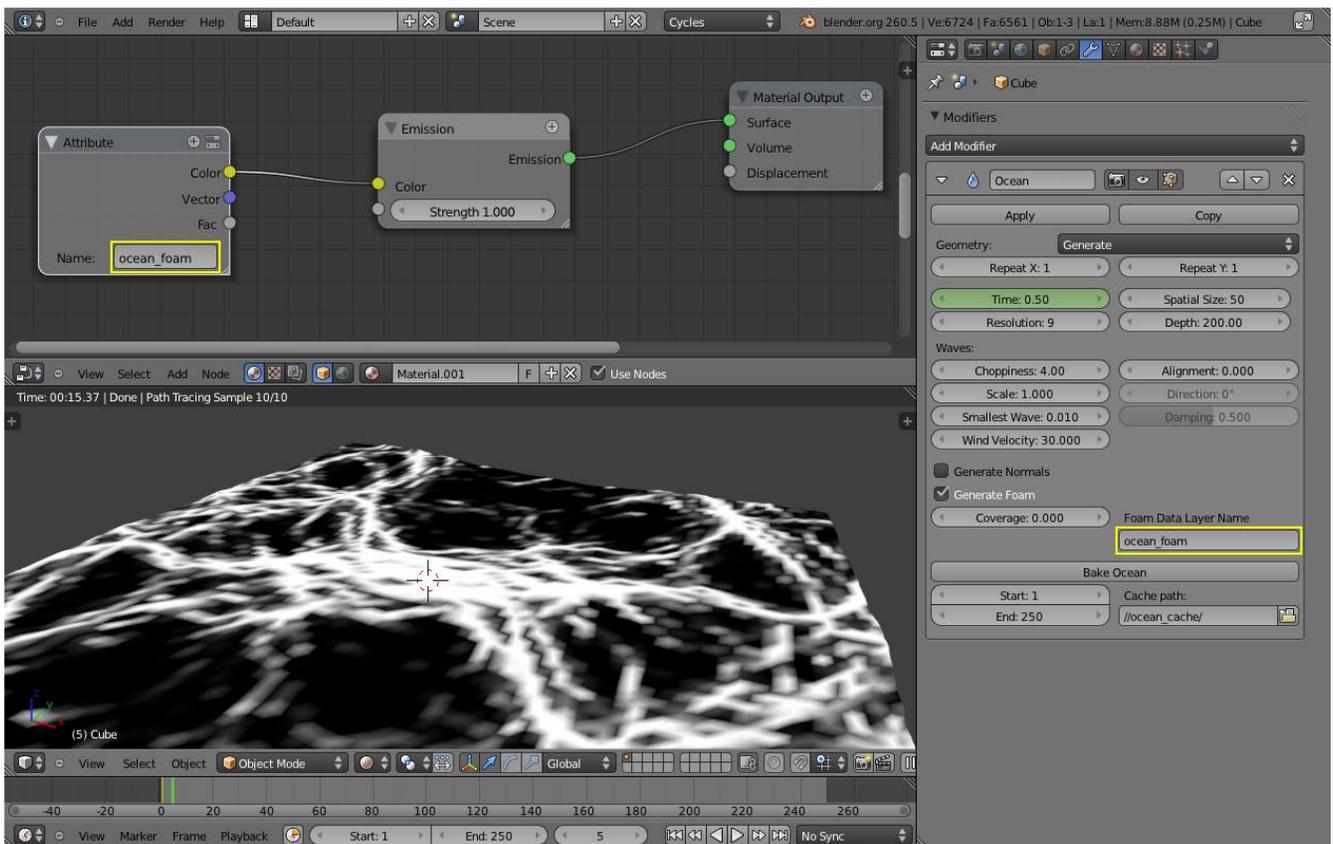


Fig. 2.871: Using foam vertex colors with a named data layer

Baking can be advantageous for a few reasons:

- It's faster to use the stored data rather than re-calculating it
- Allows rendering ocean data in external renderers
- Enables more advanced foam maps

**Data Files** Simulation data is stored in disk as sequences of OpenEXR image maps, one for each of displacement, normal and foam (if enabled to be generated). Upon loading the data from these baked files, when a frame of the bake sequence is read from disk, it is cached in memory. This means that accessing loaded frames subsequent times is fast, not incurring the overhead of disk access.

Since these baked files are plain OpenEXRs, they can also be opened and rendered in any other application or renderer that supports them.

**Baking Foam** Baking also provides improved foam capabilities. When simulating live, the ocean simulator retrieves data for that current frame only. In the case of the foam map, this represents the tips of wave crests for that given frame. In reality, after foam is created by wave interactions, it remains sitting on the top of the wave surface for a while, as it dissipates. With baking, it's possible to approximate that behaviour, by accumulating foam from previous frames, leaving it remaining on the surface.

### Baking Options

**Start, End** Frames of the simulation to bake (inclusive). The start and end frames of the bake are repeated when accessing frames outside the baked range.

**Cache Path** Folder to store the baked EXR files in. The sequences will be in the form `disp_####.exr`, `normal_####.exr`, and `foam_####.exr` where `####` is the four digit frame number. If the cache path folder does not exist, it will be created.

### Simulation Internals

The simulator itself uses FFT methods to generate 2D grids of simulation information internally, very similar to 2D texture maps. The simulator can generate three types of data - displacement, normals, and extra data that is used to calculate wave crest intersections (i.e. foam). After simulation, these maps are used to displace the ocean surface geometry in 3D, and also can be used for shading via the Ocean texture. The internal simulation engine is multi threaded with OpenMP to take advantage of multiple cores.

### Examples

Simulated and baked to image maps in Blender, rendered in 3Delight.

### Particle Instance Modifier

When a *ParticleInstance* modifier is added to an object, that object will be used as a particle shape on an object which has a particle system associated with it. This means that to use this modifier you must also have another object which has a particle system on it, otherwise the *ParticleInstance* modifier will appear to do nothing.

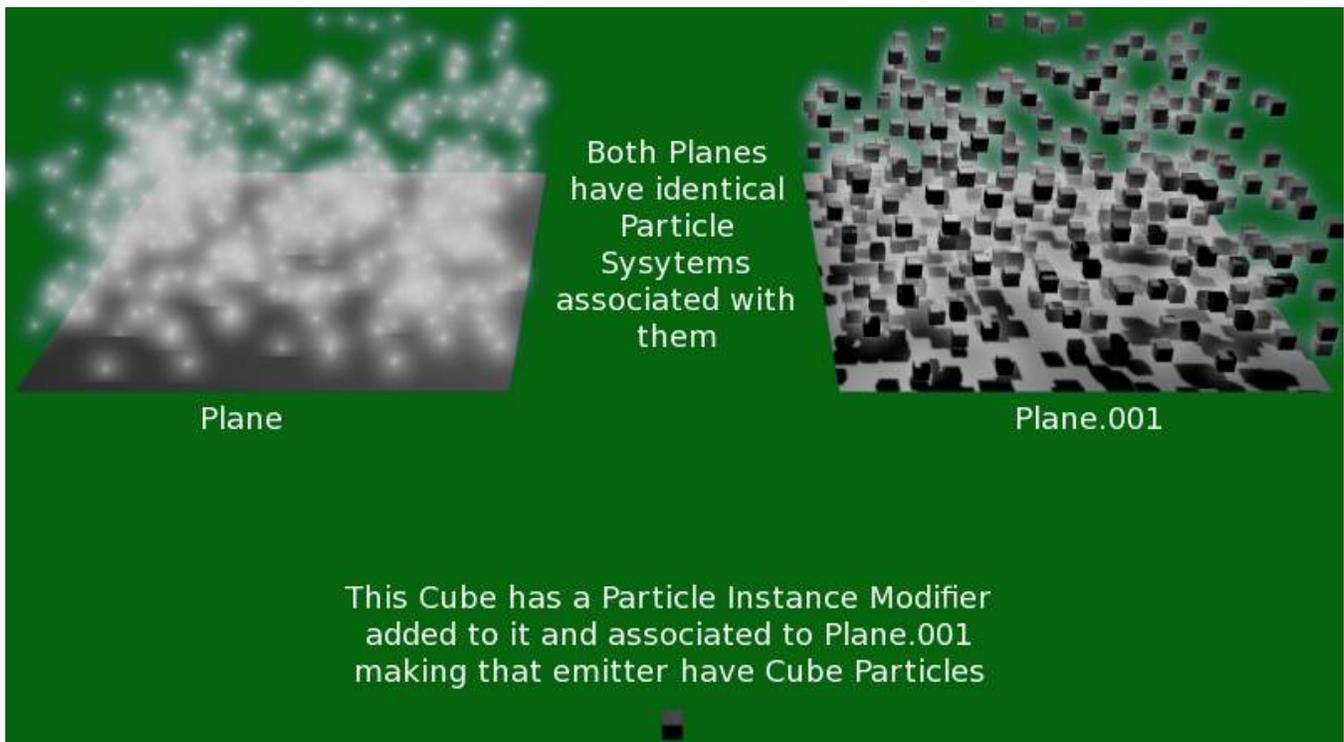


Fig. 2.872: Particle system on left has no ParticleInstance modified object associated with it. The one on the right is associated with cube shown by using a ParticleInstance modifier on the cube.

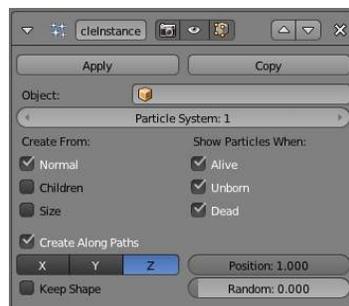


Fig. 2.873: Particle Instance Modifier

## Options

Because of the co-dependant way in which the *ParticleInstance* modifier is influenced by the underlying particle systems on other objects, some of the apparent effects generated by the *ParticleInstance* modifier can look and act vastly different, depending on the underlying settings of the particle systems it is associated with. This is worth taking account of if the *ParticleInstance* modifier settings don't appear to be giving the results expected, as it may indicate that the particle system settings may need altering rather than the *ParticleInstance* modifier settings.

**Object** The *Object* field, associates this *ParticleInstance* modifier with another object (usually an object having a particle system...). This indicates that when the object named in this field emits particles, those particles will have the mesh shape of the current *ParticleInstance* modifier's mesh. If for example a sphere has a *ParticleInstance* modifier added to it, when the *Object* field of this modifier is filled in with the name of an object that emits particles, those particle will be sphere shaped. Even though most of the time the *Object* field will have the name of an object with a particle system, this is not mandatory, you can enter an object's name which does not have a particle system, and it will be accepted by the *Object* field, as there do not appear to be any checks made to make sure the object's name entered into this field is "valid".

**Particle System** The *Particle System* field is used to select which particle system number to apply the *ParticleInstance* modifier to, when the mesh which has the particle system on it has more than one of these. The *Particle System* field can have a value between 1 and 10. It is possible to select any of the ten particle system numbers, however a check will **not** be made with the underlying particle emitting object specified previously in the *Object* field. If you select a particle system number which does not exist on the particle emitting object, then the particles on the emitting mesh will keep their normal particle shapes - no warning will be given that the chosen particle system does not exist on a particular particle emitting mesh.

As an example, below is a single plane mesh with two areas (the first area shown in red and the second in white), with different particle systems applied to each area. The left side using a *ParticleInstance* modifier which has the shape of a sphere and the right side having a *ParticleInstance* modifier which has the shape of a cube.

## Creation

**Normal** When selected, the *Normal* button tells the *ParticleInstance* modifier to draw instances of itself wherever normal particle types are emitted from the underlying particle system. So if the current *ParticleInstance* modifier is a sphere shape, when normal particles are emitted they will be spheres.

**Children** When selected, the *Children* button tells the *ParticleInstance* modifier to draw instances of itself wherever children/child particles are emitted/used on the underlying particle system. So if the current *ParticleInstance* modifier is a sphere shape, when children/child particles are emitted they will be spheres.

**Size** Scale the instanced objects by the particle size attribute. When this is disabled, all the copies appear the same size as the origin.

## Display

**Unborn** When selected, the *Unborn* button tells the *ParticleInstance* modifier to draw instances of itself wherever unborn particles will be emitted/used on the underlying particle system. So if the current *ParticleInstance* modifier is a sphere shape, when unborn particles are present they will be spheres.

**Alive** When selected, the *Alive* button tells the *ParticleInstance* modifier to draw instances of itself wherever alive particles will be emitted/used on the underlying particle system. So if the current *ParticleInstance* modifier is a sphere shape, when alive particles are present they will be spheres.

**Dead** When selected, the *Dead* button tells the *ParticleInstance* modifier to draw instances of itself wherever dead particles will occur on the underlying particle system. So if the current *ParticleInstance* modifier is a sphere shape, when dead particles are present they will be spheres.

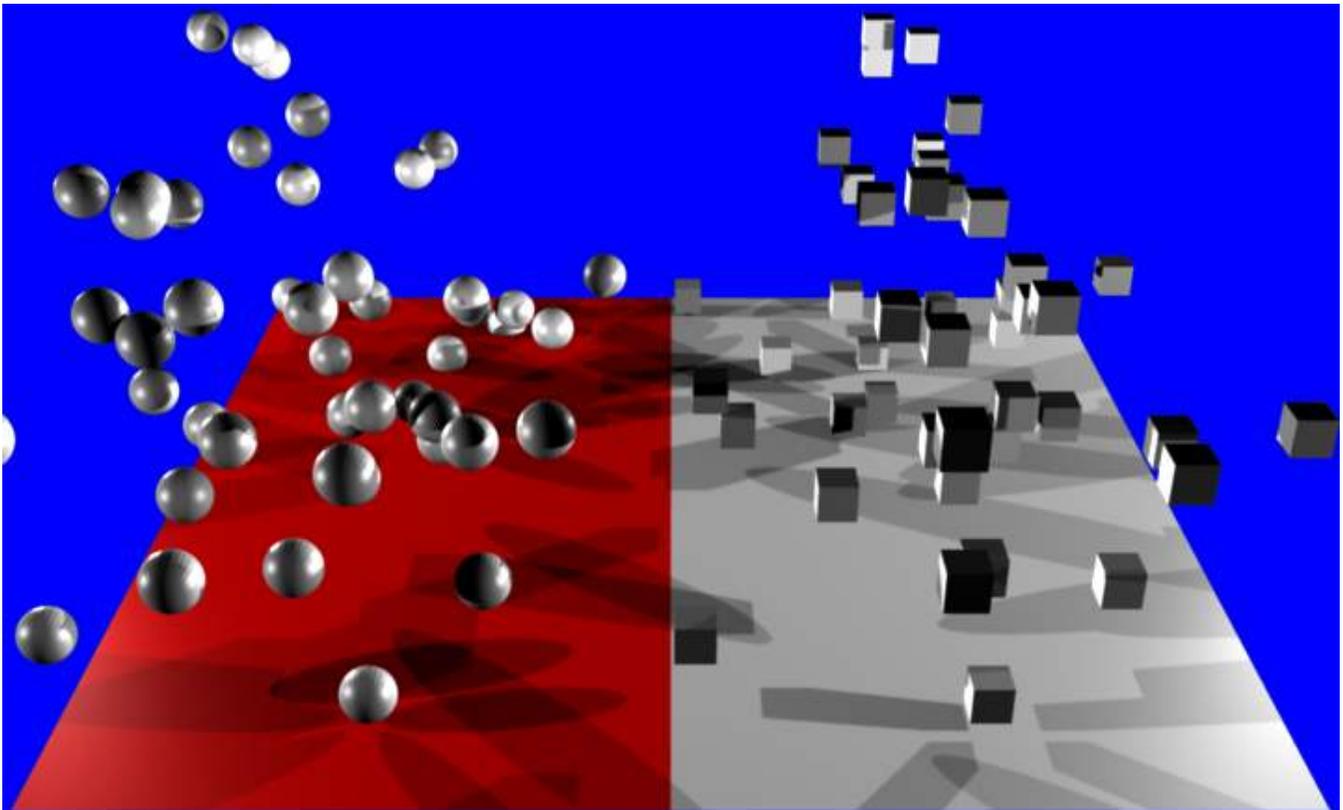


Fig. 2.874: Render showing a single Plain mesh object assigned to two different vertex groups and each of those vertex groups is assigned a separate and independent particle system, with each particle system being assigned a different ParticleInstance modifier. In the case shown the ParticleInstance modifiers are a sphere and a cube. [Example Blend file](#)

## Using Paths

**Create Along Paths** This option tries to make the underlying mesh object of the *Particle Instance* modifier deform its mesh shape in such a way as to try and match the path traveled by the particles/hair strands of the system associated with it. For example, below is a screen shot showing the path of a single keyed particle as it travels its way through each of the different way points 1 to 4 (target particle systems), when it reaches way point 4 the particle dies and ends its journey.

**X,Y,X Rotation Axis** Specify which pole axis to use for the rotation.

**Keep Shape** Enabling this prevents the object from being deformed. It instead simply aligns to the end of the path at the object's center.

**Position** Specify what percentage of the path the object fills. You could create a growing effect by animating this value over time.

**Random** Scales the position value of each instance a random value.

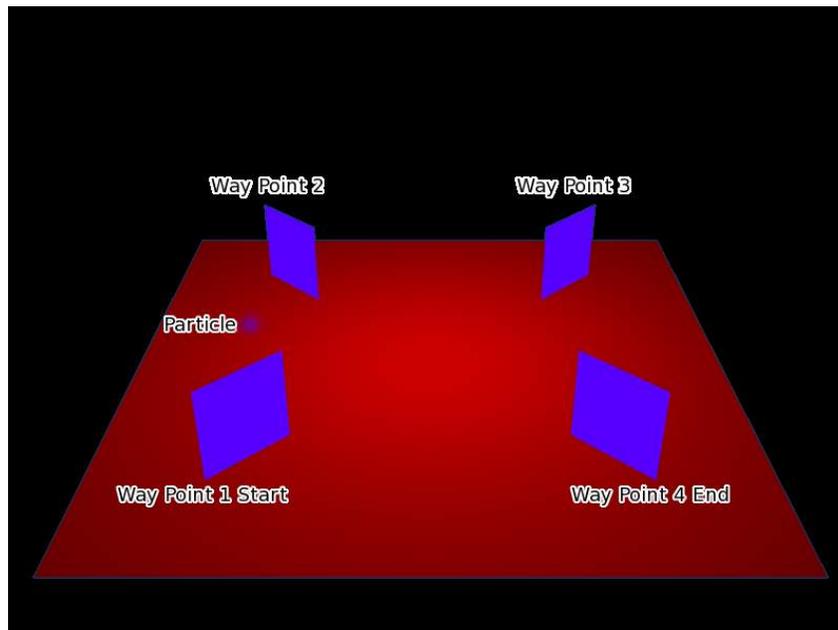


Fig. 2.875: Keyed particle following way points (showing one particle). [Example Blend file](#)

When a *ParticleInstance* modifier is added to a cylinder object and then associated with the way point particle system, the particle position is copied by the cylinder and placed at the particles position. So the mesh object follows the location of the particle. The cylinder does not alter any of its other properties when following the particle, only the cylinders location gets altered, shape and rotation do not get altered. See screenshot below:

---

**Note:** Strands when they are generated instantly die when created so for the *Path* button to be of any use, you must also have the *Dead* button activated. Otherwise the path a mesh took will not be visible!

---

**See also:**

[Particles](#)

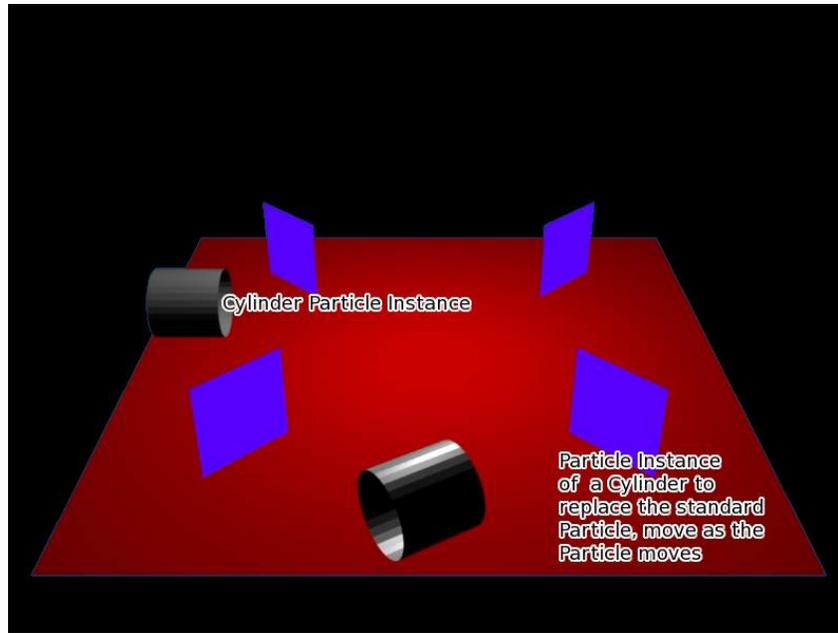


Fig. 2.876: Keyed particle following way points showing a mesh object (*ParticleInstance* modifier) in place of the original particle. [Example Blend file](#)

Both of the above examples had the *ParticleInstance* modifier *Path* button deactivated. When the *Path* button is activated the effect can be seen in the screenshot below:

## 2.5 Rigging

### 2.5.1 Introduction

After completing your character, you need to manipulate it for animation or just for posing. Rigging is the process of attaching a skeleton to your character mesh object so you can deform and pose it in different ways.

These actions do not fundamentally alter the mesh, and can easily be changed, undone, or combined with other poses.

---

**Note:** note for editors

below we need to add more introductory text for all steps in rigging!

---

The following is a typical workflow for rigging:

- Add an armature, which starts out with a single bone.
- Add more bones as needed and link them together to form “limbs”.

(you can either extrude an existing bone, which parents it to the extruded bone automatically; or add new bones first and then link them together.)

- Edit the bones to give proper proportions to the skeleton
- Apply constraints to the joints

(e.g. a human elbow can bend through about 170 degrees in one plane only)

- Give a ‘rest’ (default) position to the skeleton.
- Apply a mesh (body) to the armature (skinning)

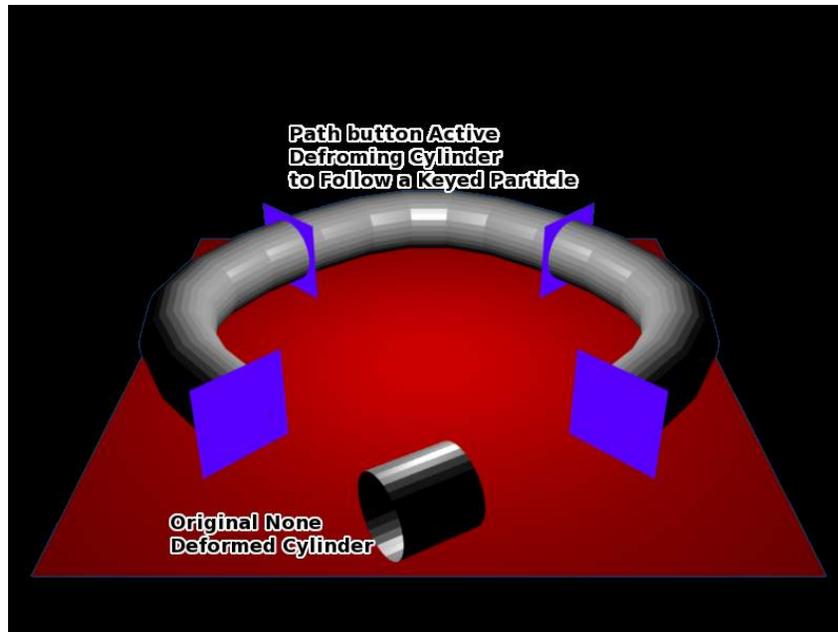


Fig. 2.877: Keyed particle following way points showing a mesh object (*ParticleInstance* modifier) in place of the original particle, that is also being deformed to fit the travel path of the original particle. [Example Blend file](#)

Instead of the cylinder location just following the position of the particle (and not altering its shape), the cylinder tries to fit its mesh to the shape of the path followed by the particle. The mesh geometry of the object which is trying to deform can have an impact on how well the deformation is carried out. In the case of the cylinder, it has many loop cuts along its length so that it can bend at those points to deform along the particle path. For example here is the same scene with the number of loop cuts along the length of the cylinder reduced, showing the effect on the deformation of the cylinder along the particle path.



Once all the extra edge loops around cylinder are removed so that there is only the top and bottom vertices left, meaning that the cylinder doesn't have enough geometry to bend, in that case it cannot follow the path of the particle, so it just goes from the start way point 1 to the ending way point 4. The *ParticleInstance* modifier *Path* button works for hair (strand) particles as well as with keyed particles. In this case the mesh of the *ParticleInstance* modifier will follow the length and profile of the hair strands paths. Below is a screenshot showing the effect of the *Path* button on hair:

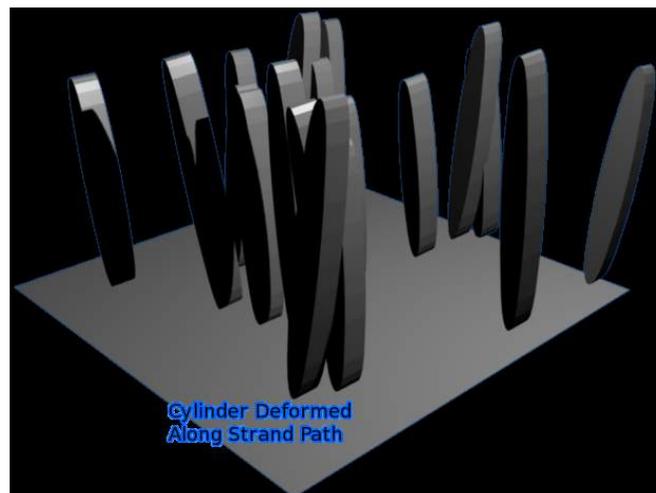


Fig. 2.886: Strand with a *ParticleInstance* modifier associated with it and deforming the cylinder along the hair profile. [Example Blend file](#)

- Define how the movement of the armature affects the skin

(folding, flexing, bulging)

- Give poses to the armature.

(There are multiple methods: By arranging each bone of the armature manually, or by copying a template armature, or by arranging the bones to follow a curve, or by making the armature follow externally collected motion-capture data.)

- Check how the armature movement affects the skin, and adjust the parameters.

(adjust the topology of the skin to make it look more natural)

All these steps are explained in details below.

done! Please check+edit [Raindrops](#) 19:32, 31 May 2013 (CEST)

## Armatures

Armatures are like the real-life skeletons; and provide the structure for a mesh for the purpose of posing or animation.

**Armature and Bone Panels** shows how to use the different panels in Blender to adjust the armatures and bones.

**Bones** explains properties of Bones, which are the basic elements of armatures.

**Visualization** how to display bones in four different ways.

**Structure** Explains the structure of bones in an armature.

**Selecting** Select only the section of your armature that matters to you.

## Editing

**Bones** Learn how to practically edit bones in Blender and see what that causes.

**Sketching** Use the Skeleton Sketching tool to easily sketch bones and bring them to reality in Blender.

**Templating** Templates offer a great way to quickly reuse already created rigs for your own models.

## Skinning

This section shows how to “flesh out” your character from a given armature.

In normal English, “to skin” means ‘to peel off skin’, but here it is just the reverse (used in the sense of covering the armature with a skin): You will be putting a body (mesh) around an armature.

**Linking Objects to Bones** How to parent a bone to an object, so that the bone controls that object. This type of linking is used to simulate mechanical linkage (for example, [Newton’s cradle](#)) or where the parts of the mesh are not deformed when the armature moves, as in case of modeling an insect body, crab, etc.

**Skinning to Objects’ Shapes** How to attach the armature so that each of its bones controls a specific part of the “skin” object’s geometry. This type of linkage is used when the object surface flexes when the armature moves, such as bulging of biceps when the arm is folded.

**Retargeting** How to apply motion-capture data (acquired from real world) to a rig, so that it mimics the original movements realistically. This method also avoids laborious programming of each movement.

## Posing

Posing means shaping and arranging the objects in your scene in a particular way to create an interesting composition. For example, look at the body language of [The Thinker](#), or think of a scorpion raising its tail to strike.

Poses are also used to create animation. For example, to create animation of a tennis player serving a ball, you would have to create poses at different moments of the stroke: (a) when she holds the ball and racket at waist height (b) when she tosses the ball up, (c) when she strikes the ball, and (d) when her racket reaches at the lowest point after the strike (follow through). Then Blender creates all the intermediate poses to create the animation.

**Visualization** describes the visual aids that help you in posing the armature; especially for animation.

**Editing Poses** how to create a pose, and how to edit it to create the snapshots of an animation at different moments.

**Pose Library** storing frequently used poses or existing poses from another armature, so that they can be quickly accessed and applied.

**Using Constraints** how to apply constraints to bones so that they cannot form an unnatural pose.

**Inverse Kinematics** a feature where you move the last bone in a chain, and Blender automatically moves the whole chain accordingly. This is like lifting someone's finger: His whole hand automatically follows that movement.

**Spline IK** a feature where you can align a chain of bones along a curve.

## 2.5.2 Armatures

An “armature” is a type of object used for [rigging](#). Armature object borrows many ideas from real life skeletons.

### Your first armature

In order to see what we're talking about, let's try to add the default armature in Blender.

(Note that armature editing details are explained in the [armatures editing section](#)).

Open a default scene, then:

- delete all objects in the scene
- make sure the cursor is in the world origin with `Shift-C`
- press `Numpad1` to see the world in Front view
- then, either: - in the Main Menu, Go to Add > Armature > Single Bone - -or- in the 3D view, add an armature with `Shift-A` *pop-up* → *Armature* → *Single Bone*
- press `NumpadDelete` to see the armature at maximum zoom

### The armature object

As you can see, an armature is like any other object type in Blender:

- It has a center, a position, a rotation and a scale factor.
- It has an `ObData` datablock, that can be edited in *Edit mode*.
- It can be linked to other scenes, and the same armature data can be reused on multiple objects.
- All animation you do in *Object mode* is only working on the whole object, not the armature's bones (use the *Pose mode* to do this).

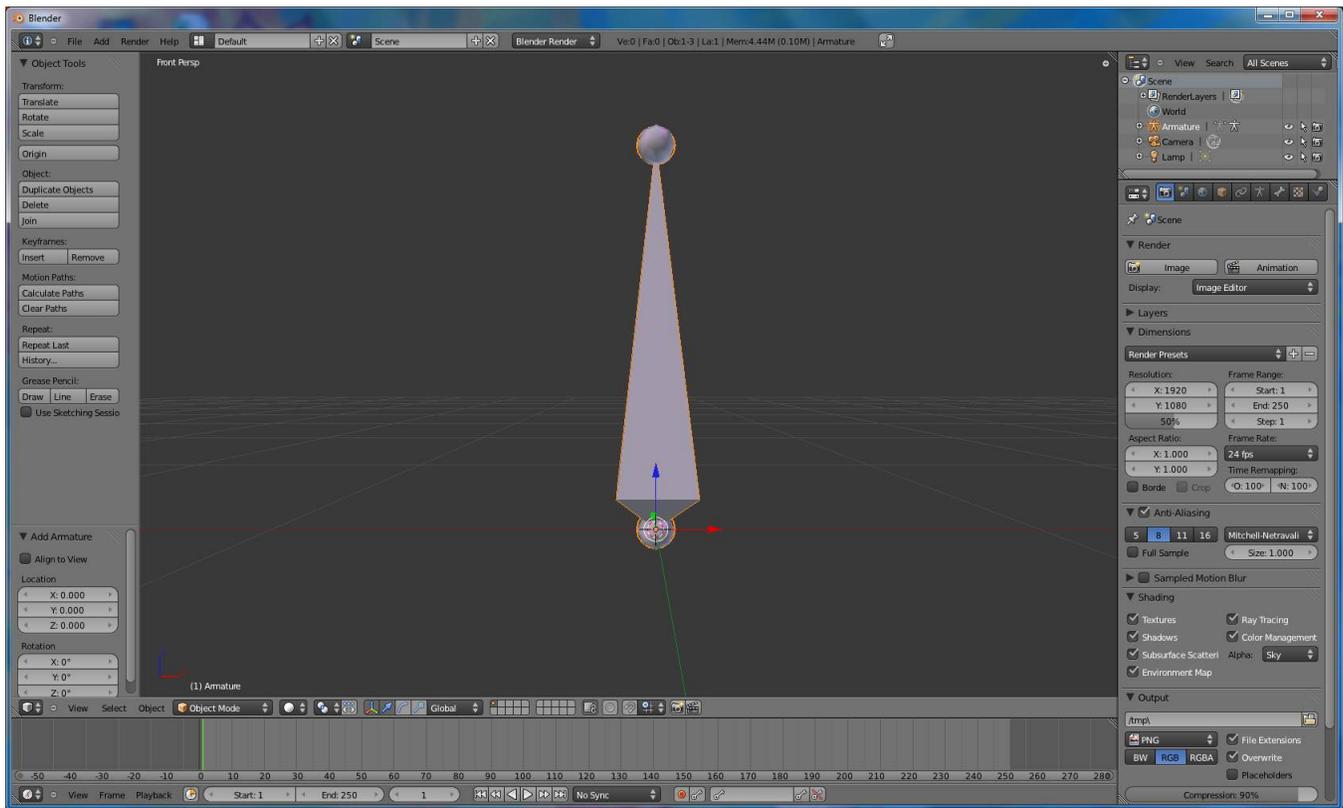


Fig. 2.887: The default armature Toolbox: → Add Armature → Single Bone

As armatures are designed to be posed, either for a static or animated scene, they have a specific state, called “rest position”. This is the armature’s default “shape”, the default position/rotation/scale of its bones, as set in *Edit mode*.

In *Edit mode*, you will always see your armature in rest position, whereas in *Object* and *Pose mode*, you usually get the current “pose” of the armature (unless you enable the *Rest Position* button of the *Armature* panel).

## Armature chapter overview

In the “Armatures” section, we will only talk about armatures themselves, and specifically we will talk about:

- the armature object panels
- the basics of bones
- the different armature visualizations
- the armature structure types
- how to select its parts,
- how to edit an armature
- how to Edit Bones
- how to edit bones properties
- how to sketch armatures with the Etch-a-Ton tool
- how to use templates

## 2.5.3 Armature Panels Overview

### Reference

Mode: *Object mode*, *Edit mode* and *Pose mode*

Panel: All in *Properties* window, *Object data* property

Let’s first have a general overview of the various panels gathering the armature settings, in *Properties* window, *Object data* context:

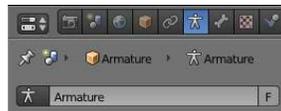


Fig. 2.888: The Object data property in the Properties window.

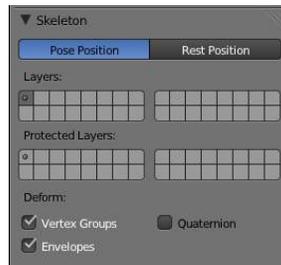


Fig. 2.889: The Skeleton panel.

In this panel you can arrange sets of bones into different layers for easier manipulation.

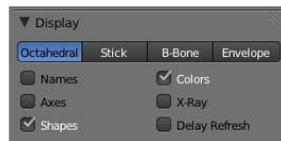


Fig. 2.890: The Display panel.

This controls the way the bones appear in 3D view; you have 4 different options you can select. There are several other options available which we will cover later on.



Fig. 2.891: The Bone Groups panel.

Lets you assign sets of bones into groups for easy manipulation and management.

**Skeleton panel (all modes)**

**Display panel (all modes)**

**Bone groups panel (pose mode)**

**Pose Library panel (Pose mode)**



Fig. 2.892: The Pose Library panel.

Allows you to save different settings (location, rotation, scale) for selected bones for later use.

**Ghost panel (all modes)**

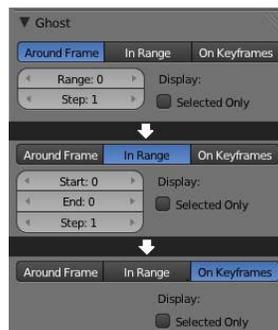


Fig. 2.893: The Ghost panel.

Allows you to see a set of different consecutive poses, very useful when animating.

**iTaSC parameters panel (all modes)**



Fig. 2.894: The iTaSC parameters panel.

Defines the type of IK solver used in your animation.

**Motion Paths panel (Pose mode)**

**Custom Properties panel (all modes)**

**Bone Panels Overview**

---

## Reference

Mode: *Object mode*, *Edit mode* and *Pose mode*

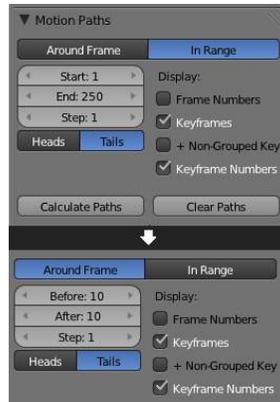


Fig. 2.895: The Motion Paths panel.

In this panel you can enable visualization of the motion path your skeleton leaves when animated.

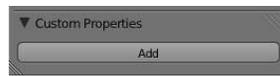


Fig. 2.896: The Custom Properties panel.

Panel for defining custom properties; this is used when scripting.

Panel: All in *Properties* window, *Bone* property

Let's first have a general grasp of the various panels gathering the bone settings, in *Properties* window, *Bone* context:

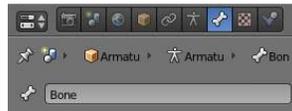


Fig. 2.897: The Bone context.

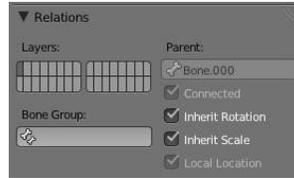


Fig. 2.898: The Relations panel.

In this panel you can arrange sets of bones in different layers for easier manipulation.



Fig. 2.899: The Display panel.

Display panel lets you customize the look of your bones taking the shape of a another existing object.

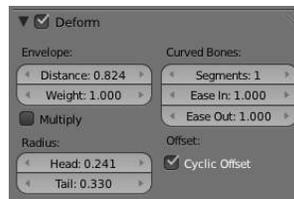


Fig. 2.900: The Deform panel.

In this panel you can set basic properties of the bones.

Turning the Deform option on and off, includes the active bone in the Automatic Weight Calculation when the Mesh is Parented to the Armature using the Armature Deform with the “With Automatic Weights” option.

Also it’s worth noting that by turning off a bone’s deform option, makes it not influence the mesh at all, overriding any weights that it might have been assigned before; It mutes its influence.

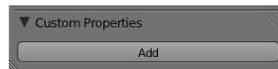


Fig. 2.901: The Custom Properties panel.

Panel for defining custom properties, this is used when scripting.

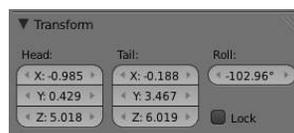


Fig. 2.902: The Transform panel(edit mode).

When in edit mode you can use this panel to control position and roll of individual bones.

When in pose mode you can only set location for the main bone, and you can now set rotation and scale.

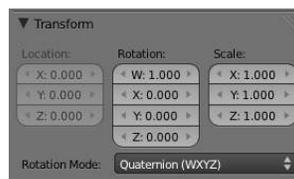


Fig. 2.903: The Transform panel(pose mode).

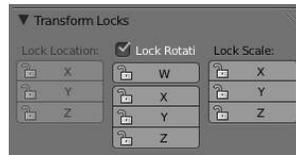


Fig. 2.904: The Transform Locks panel.

This panel appears only in pose mode and allows you to restrict position, rotation and scale by axis on each bone in the armature.

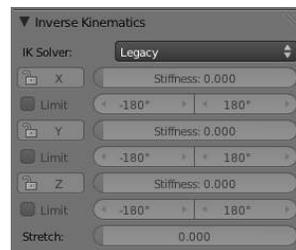


Fig. 2.905: The Inverse Kinematics panel.

This panel controls the way a bone or set of bones behave when linked in an inverse kinematic chain.



Fig. 2.906: The elements of a bone.

**Relations panel (edit mode)**

**Display panel (object mode)**

**Deform panel (all modes)**

**Custom Properties panel (all modes)**

**Transform panel (edit and pose mode)**

**Transform Locks panel (pose mode)**

**Inverse Kinematics panel (pose mode)**

## 2.5.4 Bones

Bones are the base elements of armatures.

They have three elements:

- the “start point” named **root** or **head**,
- the “body” itself,
- and the “end point” named **tip** or **tail**.

Select the default `armature` and press `Tab` to enter *Edit mode*. As you can see, in this mode you can select the root and the tip, and move them as you do with mesh vertices (don’t lose too much time here though, specific pages about selecting and editing will come later).

Both root and tip (the “ends”) define the bone by their respective position.

They also have a radius property, only useful for the envelope deformation method (see below).

### Bones Visualization

Bones can be visualized in various ways: *Octahedron*, *Stick*, *B-Bone*, *Envelope* and *Wire*. Custom shapes can be used, too!

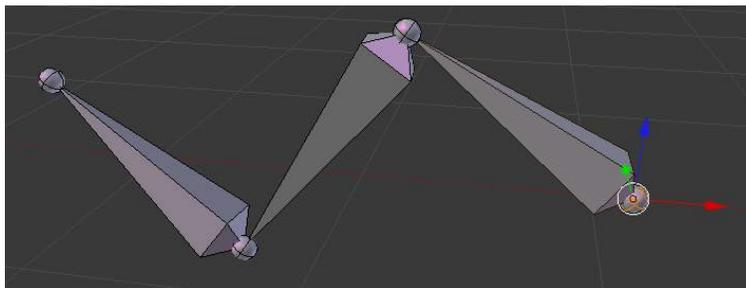


Fig. 2.907: Octahedral bone display.

Since armatures are made of bones, you’ll find more about this when we’ll talk about [Armatures Visualization](#).

Activating *Axes* checkmark on the *Armature / Display* panel, will show local axes for each bone’s tip. The Y axis is always aligned along the bone, oriented from root to tip. So, this is the “roll” axis of the bones.

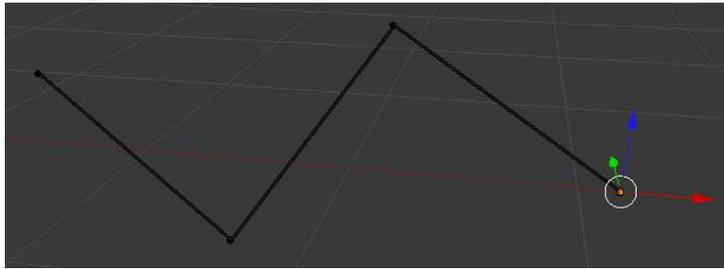


Fig. 2.908: Stick bone display.

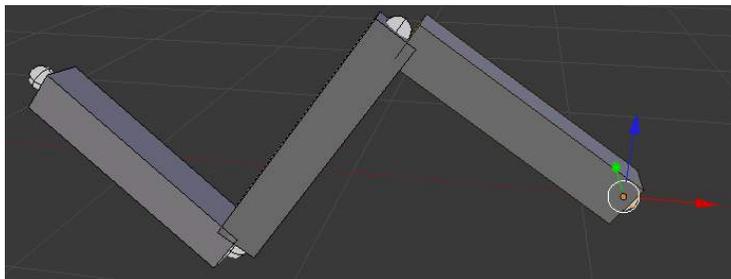


Fig. 2.909: B-Bone bone display.

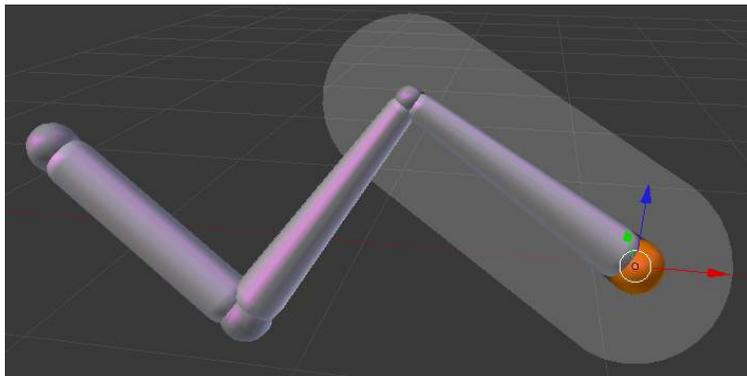


Fig. 2.910: Envelope bone display.



Fig. 2.911: The Bone context.

## Bones properties

When bones are selected (hence in *Edit mode* and *Pose mode*), their properties are shown in the *Bone* button context of the *Properties* window.

This shows different panels used to control features of each selected bone; the panels change depending on which mode you're working in.

## Bones Rigidity

Even though bones are rigid (i.e. behave as rigid sticks), they are made out of *segments*. *Segments* are small, rigid linked elements that can rotate between each other. By default, each new bone has only one segment and as such it cannot “bend” along its length. It is a rigid bone.

You can see these segments in *Object mode* and in *Pose mode*, and only if bones are visualized as *B-bones*; while in *Edit mode* bones are always drawn as rigid sticks. Note that in the special case of a single bone, you can't see these segments in *Object mode*, because they're aligned.

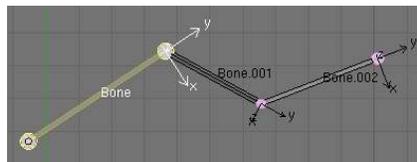


Fig. 2.912: An armature of B-Bones, in Edit mode

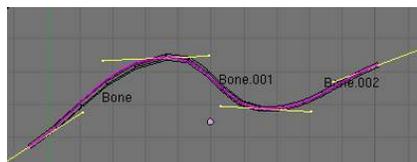


Fig. 2.913: The Bézier curve superposed to the chain, with its handles placed at bones' ends.

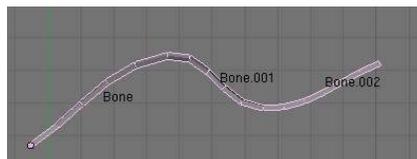


Fig. 2.914: The same armature in Object mode

When you connect bones to form a chain, Blender calculates a Bézier curve passing through all the bones' ends, and bones' segments in the chain will bend and roll to follow this invisible curve.

*You have no direct access to this curve*; you can only control it to some extent using bone properties, as explained in the editing pages.

In *An armature of B-Bones in Edit mode* we connected 3 bones, each one made of 5 segments. These are *B-bones* but as you see, in *Edit mode* they are shown as rigid elements. Look at *The same armature in Object mode*: now, in *Object mode*, we can see how the bones' segments smoothly “blend” into each other, even for roll.

Of course, a geometry influenced by the chain is smoothly deformed according to the Bézier curve! In fact, smooth bones are an easy way to replace long chains of many small rigid bones posed using IK...

However, if the chain has an influence on objects rather than geometry, the segments' orientation is not taken in account (details are explained in the [skinning part](#)).

When not visualized as *B-Bone*s, bones are always shown as rigid sticks, *even though the bone segments are still present and effective* (see [skinning to ObData](#)).

This means that even in e.g. *Octahedron* visualization, if some bones in a chain have several segments, they will nonetheless smoothly deform their geometry...

## Bones influence

Basically, a bone controls a geometry when vertices “follow” the bone. This is like how the muscles and skin of your finger follow your finger-bone when you move a finger.

To do this, you have to define **how much** a bone influences a certain vertex.

The simplest way is to have each bone affecting those parts of the geometry that are within a given range from it. This is called the *envelope technique*, because each bone can control only the geometry “enveloped” by its own influence area.

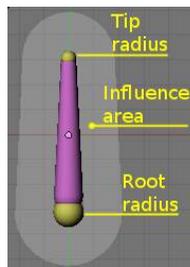


Fig. 2.915: A bone in Envelope visualization, in Edit mode.

If a bone is visualized as *Envelope*, in *Edit mode* and in *Pose mode* you can see the area of influence, which depends on:

- the *distance* property
- the root's radius and the tip's radius.

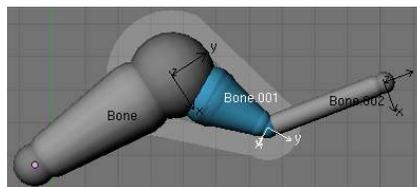


Fig. 2.916: Our armature in Envelope visualization, in Pose mode.

All these influence parameters are further detailed in the [skinning pages](#).

## 2.5.5 Armature visualization

We have 4 basic bone visualization: Octahedral, Stick, B-Bone, Envelope and Wire:

### Display Panel

### Reference

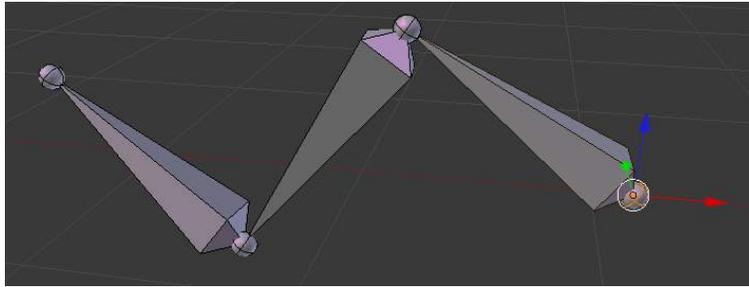


Fig. 2.917: Octahedral bone display.

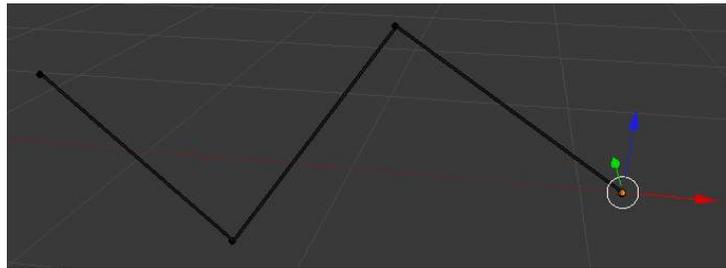


Fig. 2.918: Stick bone display.

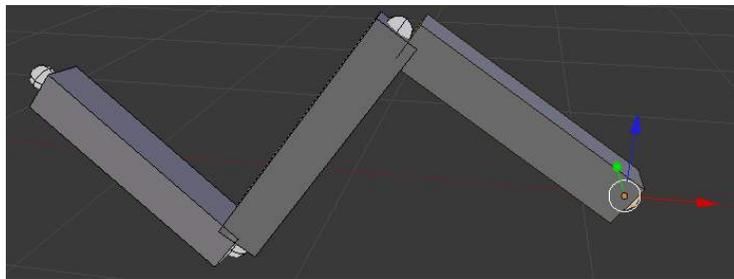


Fig. 2.919: B-Bone bone display.

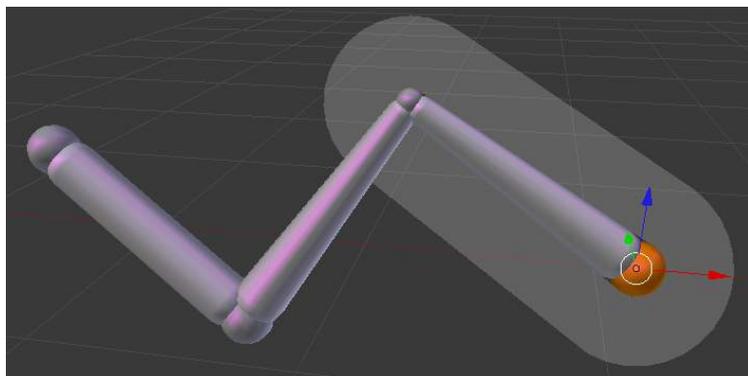


Fig. 2.920: Envelope bone display.

Mode: *Object*, *Edit* and *Pose* modes

Panel: *Display Object Data* context

But let's first see some general visualization properties of armatures, found in the *Display* panel of the *Object data* context.



Fig. 2.921: The Display panel.

## Bone types

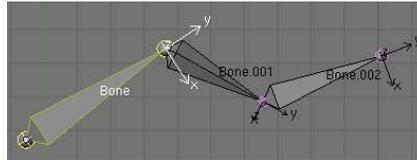


Fig. 2.922: A basic armature in Octahedron visualization, Edit mode.

Note the 40- rolled Bone.001 bone.

**Octahedral bone** This is the default visualization, well suited for most of editing tasks. It materializes:

- The bone root (“big” end) and tip (“small” end).
- The bone “size” (its thickness is proportional to its length).
- The bone roll (as it has a square section).

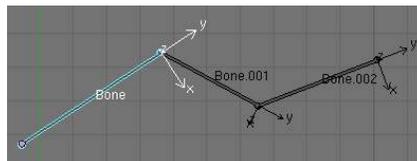


Fig. 2.923: The same armature in Stick visualization, Pose mode. Note that Bone.001 roll angle is not visible (except by its XZ axes).

**Stick bone** This is the simplest and most non-intrusive visualization. It just materializes bones by sticks of constant (and small) thickness, so it gives you no information about root and tip, nor bone size or roll angle.

**B-Bone bone** This visualization shows the curves of “smooth” multi-segmented bones; see the [bone](#) page for details.

**Envelope bone** This visualization materializes the bone deformation influence. More on this in the [bone](#) page.

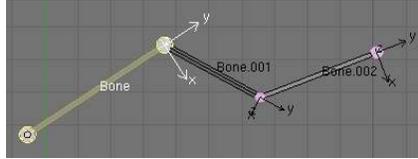


Fig. 2.924: The same armature in B-Bone visualization, Edit mode.

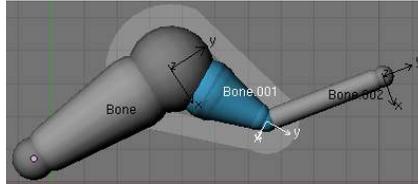


Fig. 2.925: The Bone Groups panel.

## Attributes

**Names** When enabled, the name of each bone is drawn.

**Colors** This is only relevant for *Pose* mode, and is described in detail [there](#).

**Axes** When enabled, the (local) axes of each bone are drawn (only relevant for *Edit* and *Pose* modes).

**X-Ray** When enabled, the bones of the armature will always be drawn on top of the solid objects (meshes, surfaces, ...) - i.e. they will always be visible and selectable (this is the same option as the one found in the *Display* panel of the *Object data* context. Very useful when not in *Wireframe* mode.

**Shapes** When enabled, the default standard bone shape is replaced, in *Object* and *Pose* modes, by the shape of a chosen object (see [Shaped Bones](#) for details).

**Delay Refresh** When enabled, the bone doesn't deform its children when manipulating the bone in pose mode.

## Shaped Bones

---

## Reference

Mode: *Object* and *Pose* modes

Panel: *Display* panel from *Bone* context.

---

Blender allows you to give to each bone of an armature a specific shape (in *Object* and *Pose* modes), using another object as “template”. First of all, you have to enable the *Shapes* button (*Armature* panel).



Fig. 2.926: The Display panel.

## Attributes

**Wireframe** When enabled, bone is displayed in wireframe mode regardless of the viewport drawing mode. Useful for non-obstructive custom bone chains.

**Hide** Bone is not visible when not in *Edit mode*.

**Custom Shape** Object that defines the custom shape of the selected bone.

**Custom At** Bone that defines the display transform of this shape bone

To assign a custom shape to a bone, you have to:

- Switch to *Pose mode* (Ctrl-Tab).
- Select the relevant bone (RMB click on it).
- Go to the *Display* panel *Custom Shape* field and select the 3D object previously created in the scene; in this example we are using a cube and a cone. You can optionally set the *At* field to another bone.

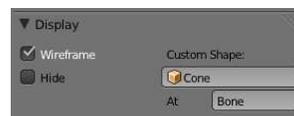


Fig. 2.927: The Display panel.

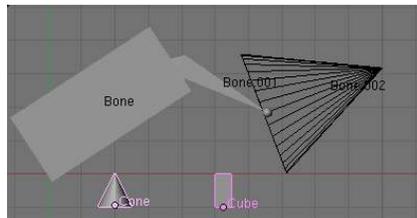


Fig. 2.928: The armature with shapes assigned to two bones, in Object mode. Note the centers of the Cone and Cube objects.

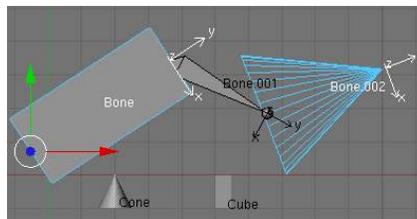


Fig. 2.929: The same armature in Pose mode...

Note that:

- These shapes will never be rendered - like any bone, they are only visible in 3D views.
- Even if any type of object seems to be accepted by the *OB* field (meshes, curves, even metas...), only meshes really work - all other types just make the bone invisible; nothing is drawn...
- The center of the shape object will be at the *root of the bone* (see the [bone page](#) for root/tip).
- The object properties of the shape are ignored (i.e. if you make a parallelepiped out of a cube by modifying its dimensions in *Object mode*, you'll still have a cube shaped bone...).
- The “along bone” axis is the Y one, and the shape object is always scaled so that one Blender Unit stretches along the whole bone length.

- If you need to remove the custom shape of the bone, just right click in the *Custom Shape* field and select *Reset to default value* in the pop-up menu.

So to summarize all this, you should use meshes as shape objects, with their center at their lower-Y end, and an overall Y length of **1.0 BU**.

## Armature Layers

### Reference

Mode: *Object*, *Edit* and *Pose* modes

Panel: *Skeleton* panel, *Object data* context

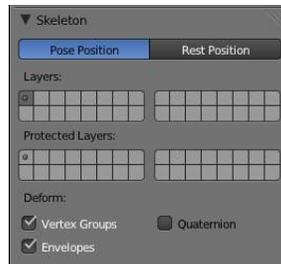


Fig. 2.930: The Skeleton panel.

Each armature has 32 “Armature layers” which allow you to organize your armature by “regrouping” sets of bones into layers; this works similar to scene layers (those containing your objects). You can then “move” a bone to a given layer, hide or show one or several layers, etc.

### Showing/hiding bone layers

Only bones in active layers will be visible/editable - but they will always be effective (i.e move objects or deform geometry), whether in an active layer or not. To (de)activate a layer, you have several options, depending in which mode you are in:

- In all modes, use the row of small buttons at the top of the *Display Options* group, *Armature* panel. If you want to enable/disable several layers at once, as usual, hold *Shift* while clicking...
- In *Edit* and *Pose* modes, you can also do this from the *3D View* s, either by using the menu (*Armature* → *Switch Armature Layers* or *Pose* → *Switch Armature Layers*), or the *Shift-M* shortcut, to display a small pop-up dialog containing the same buttons as described above (here again, you can use *Shift-LMB* clicks to (de)select several layers at once).

### Protected Layers

You can lock a given bone layer for all *proxies* of your armature, i.e. all bones in this layer won’t be editable. To do so, in the *Skeleton* panel, *Ctrl-LMB* click on the relevant button, the layer lock will be enabled.

Protected layers in proxy are restored to proxy settings on file reload and undo.

## Bone Layers

### Reference

Mode: *Object*, *Edit* and *Pose* modes

Panel: *Relations* panel *Bone* context

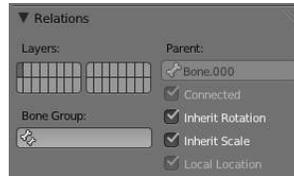


Fig. 2.931: The Relations panel.

### Moving bones between layers

Obviously, you have to be in *Edit* or *Pose* modes to move bones between layers - note that as with objects, bones can lay in several layers at once, just use the usual `Shift-LMB` clicks... First of all, you have to select the chosen bone(s)!

- In the *Button* window, use the “layer buttons” of each selected bone “sub-panel” (*Armature Bones* panel) to control in which layer(s) it lays.
- In the *3D View* window, use the menu (*Armature* → *Move Bone To Layer* or *Pose* → *Move Bone To Layer*) or press `M` to show the usual pop-up layers dialog. Note that this way, *you assign the same layers to all selected bones*.

### Hiding Bones

#### Reference

Mode: *Edit* and *Pose* modes

Panel: *Display* panel, *Bone* context



Fig. 2.932: The Display panel.

You do not have to use bone layers to show/hide some bones. As with objects, vertices or control points, you can use the `H` key:

- `H` will hide the selected bone(s).
- `Shift-H` will hide all bones *but the selected one(s)*.
- `Alt-H` will show all hidden bones.

You can also use the *Hide* check button of the *Display* panel, *Bone* context).

Note that hidden bones are specific to a mode - i.e. you can hide some bones in *Edit* mode, they will still be visible in *Pose* mode, and vice-versa. Hidden bone in *Pose* mode are also invisible in *Object* mode. And in *Edit* mode, the bone to hide must be fully selected, not just his root or tip...

## 2.5.6 Armature Structure

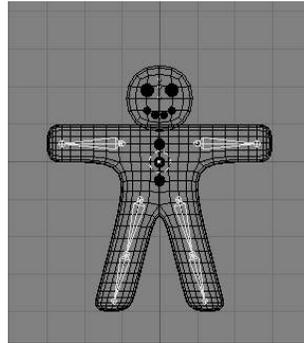


Fig. 2.933: The very basic armature of the Gingerbread Man tutorial.

Armatures mimic real skeletons. They are made out of bones, which are (by default) rigid elements. But you have more possibilities than with real skeletons: In addition to the “natural” rotation of bones, you can also translate and even scale them! And your bones do not have to be connected to each other; they can be completely free if you want. However, the most natural and useful setups imply that some bones are related to others, forming so-called “chains of bones”, which create some sort of “limbs” in your armature, as detailed in *Chains of Bones*.

### Chains of Bones

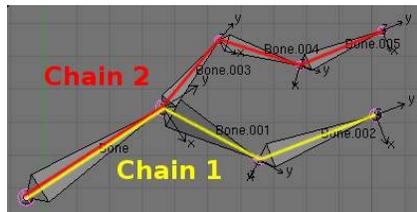


Fig. 2.934: An armature with two chains of bones.

The bones inside an armature can be completely independent from each other (i.e. the modification of one bone does not affect the others). But this is not often a useful set up: To create a leg, all bones “after” the thigh bone should move “with” it in a well-coordinated manner. This is exactly what happens in armatures - by parenting a bone to the next one in the limb, you create a “chains of bones”. These chains can be ramified. For example, five fingers attached to a single “hand” bone.

Bones are chained by linking the tip of the parent to the root of the child. Root and tip can be *connected*, i.e. they are always exactly at the same point; or they can be *free*, like in a standard parent-child object relationship.

A given bone can be the parent of several children, and hence be part of several chains at the same time.

The bone at the beginning of a chain is called its *root bone*, and the last bone of a chain is the *tip bone* (don’t confuse them with similar names of bones’ ends!).

Chains of bones are a particularly important topic in [posing](#) (especially with the standard *forward kinematics* versus “automatic” *inverse kinematics* posing techniques). You create/edit them in *Edit* mode, but except in case of connected bones, their relationships have no effect on bone transformations in this mode (i.e. transforming a parent bone won’t affect its children).

### Editing Bones Relationships

This is detailed in the [editing](#) pages, but let us have a quick look at this important feature.

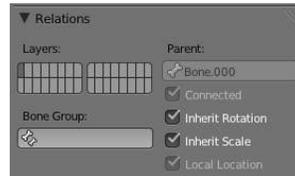


Fig. 2.935: The Armature Bones panel with two bones selected, and their Child of settings highlighted.

The easiest way to manage bones relationships is to use the *Relations* panel *Bone* context:

- First, [select](#) the bones you want to edit (selection order does not matter here).
- To *parent* a bone to another one, select the name of this parent in its drop-down *Parent* list.
- To *unparent* a bone, just select the void entry in the same *Parent* list.
- To *connect* a bone to its parent, enable its small *Con* button.
- To *disconnect* a bone, disable its *Con* button.

## 2.5.7 Selecting armature’s bones

### Reference

Mode: *Edit mode*

Panel: *Bone* panel

You can select and edit bones of armatures in *Edit mode* and in *Pose mode*. Here, we will see how to select bones in *Edit mode*. Selecting bones in *Pose mode* is similar to selecting in *Edit mode* with a few specific differences that will be detailed in the [posing part](#).

Similar to [vertices/edges selection](#) in meshes, there are two ways to select whole bones in *Edit mode*:

- directly, by selecting the bone’s body
- selecting both of its end points (root and tip)

This is an important point to understand, because selecting bones’ ends only might lead to non-obvious behavior, with respect to which bone you actually select, see the.

Note that unlike the mesh draw type the armature draw type has no effect on selection behavior. In other words, you can select a bone’s end or body the same way regardless of the bone visualization chosen.

## Selecting bones' ends

To select bones' ends you have the standard selection methods.

action	shortcut	menu	mouse
Select a bone's end			RMB -click on it
Add or Remove from the current selection			Shift-RMB
(De)select the ends of all bones	A	<i>Select</i> → <i>Select/Deselect All</i>	
Invert the current selection	Ctrl-I	<i>Select</i> → <i>Inverse</i>	
Box selection tool ON	B	<i>Select</i> → <i>Border Select</i>	
Box selection	<p>Click and drag LMB the box around the ends you want to add to the current selection</p> <p>Click and drag LMB to remove from the current selection</p> <p>release LMB to validate</p> <p>press Esc or click RMB to cancel</p>		
Box selection tool OFF	B or Esc		RMB
Lasso selection	<p>Click and drag Ctrl-LMB the lasso around the ends you want to add to the current selection</p> <p>Click and drag Ctrl-Shift-LMB to remove from the current selection</p> <p>Release LMB to validate</p> <p>Hit Esc or click RMB to cancel</p>		

## Inverse selection

As stated above, you have to remember that these selection tools are for bones' ends only, not the bones' bodies.

For example, the *Inverse* selection option (Ctrl-I) inverts the selection of bones' ends, not of bones (see *Inverse selection*).

Remember that a bone is selected only if both its ends are selected. So, when the selection status of bones' ends is inverted, a new set of bones is selected.

Table  
2.19: The  
result  
of the  
inverse  
selection  
Ctrl-I  
the bones  
ends  
selection  
has been  
inverted,  
and not  
the bones  
selection.



### Selecting connected bones' ends

Another example is: when you select the root of a bone connected to its parent, you also implicitly select the tip of its parent (and vice versa).

Remember: when selecting bones' ends, the tip of the parent bone is the “same thing” as the root of its children bones.

### Selecting Bones

By RMB -clicking on a bone's body, you will select it (and hence you will implicitly select its root and tip).

To each selected bone corresponds a sub-panel in the *Armature Bones* panel (*Editing* context). These sub-panels contain settings for some of the bones' properties (regarding e.g. relationships between bones, bones' influence on deformed geometry, etc.), as we will see later.

Using Shift-RMB, you can add to/remove from the selection.

You also have some **advanced selection** options, based on their relations.

You can select at once all the bones in the chain which the active (last selected) bone belongs to by using the *linked selection* tool, L.

Table  
2.20: Its  
whole  
chain  
selected  
with [L].



You can deselect the active bone and select its immediate parent or one of its children using respectively *Select → Select Parent* ([P]) or *Select → Select Child* ([C]). If you prefer to keep the active bone in the selection, use *Select → Extend Select Parent* (Ctrl-[P]) or *Select → Extend Select Child* (Ctrl-[C]).

## Deselecting connected bones

There is a subtlety regarding connected bones.

When you have several connected bones selected, if you deselect one bone, *you will in fact deselect its tip, but not its root if it is also the tip of another selected bone.*

To understand this, look at *Bone deselection in a selected chain.*

Table

2.21: A  
selected  
chain.



After Shift-RMB -clicking Bone .003:

- Bone .003 's tip (which is same as Bone .004 's root) is deselected
- Bone is Bone .003 's parent. Therefore Bone .003 's root is same as the tip of Bone. Since Bone is still selected, its tip is selected. Thus the root of Bone .003 remains selected.

## 2.5.8 Armature Editing

### Reference

Mode: *Edit mode*

Hotkey: Tab

As with any other object, you edit your armature in *Edit mode* (Tab).

Editing an armature means two main domains of action:

- **Editing the bones** - i.e. adding/inserting/deleting/extruding/sub-dividing/joining them...
- **Editing the bones' properties** - this includes key features, like transform properties (i.e. grab, scale, etc...) and relationships between bones (parenting and connecting), as well as bones' names, influence, behavior in *Pose mode*, etc.

These are standard editing methods, quite similar for example to [meshes](#) editing. Blender also features a more advanced "armature sketching" tool, called [Etch-a-Ton](#). The same tool might also be used in [templating](#), i.e. using another armature as template for the current one...

**Warning:** One important thing to understand about armature editing is that you **edit the rest position of your armature**, i.e. its "default state". An armature in its *rest position* has all bones with no rotation and scaled to **1.0** in their own local space.

The different [poses](#) you might create afterwards are based on this rest position - so if you modify it in *Edit mode*, all the poses already existing will also be modified. Thus you should in general be sure that your armature is definitive before starting to [skin](#) and [pose](#) it!

**Warning:** Please note that some tools work on bones' ends, while others work on bones themselves. Be careful not to get confused.

---

## 2.5.9 Editing Bones

---

### Reference

Mode: *Edit mode*

Hotkey: Tab

---

You'll learn here how to add (*Adding Bones*), delete (*Deleting Bones*) or subdivide (*Subdividing Bones*) bones. We will also see how to prevent any bone transformation (*Locking Bones*) in *Edit* mode, and the option that features an automatic mirroring (*X-Axis Mirror Editing*) of editing actions along the X axis.

### Adding Bones

To add bones to your armature, you have more or less the same options as when editing meshes:

- *Add* menu,
- extrusion,
- Ctrl-LMB clicks,
- fill between joints,
- duplication.

### Add Menu

---

### Reference

Mode: *Edit mode*

Hotkey: Shift-A

---

In the 3D view, Shift-Apop-up → *Bone* to add a new bone to your armature.

This bone will be:

- of one Blender Unit of length,
- oriented towards the positive Y axis of the view,
- with its root placed at the 3D cursor position,
- with no relationship with any other bone of the armature.

### Extrusion

---

### Reference

Mode: *Edit mode*

---

Menu: *Armature* → *Extrude*

Hotkey: E, Shift-E

When you press the E key, for each selected tip (either explicitly or implicitly), a new bone is created. This bone will be the child of “its” tip owner, and connected to it. As usual, once extrusion is done, only the new bones’ tips are selected, and in grab mode, so you can place them to your liking. See (*Extrusion example*).

Table

2.22:

The three  
extruded  
bones.



You also can use the rotating/scaling extrusions, as explained for meshes [here](#), by pressing respectively E-R and E-S - as well as [locked](#) extrusion along a global or local axis.

Table

2.23:

The two  
mirror-  
extruded  
bones.



Bones have an extra “mirror extruding” tool, called by pressing Shift-E. By default, it behaves exactly like the standard extrusion. But once you have enabled the X-Axis mirror editing option (see [X-Axis Mirror Editing](#)), each extruded tip will produce *two new bones*, having the same name except for the `_L/_R` suffix (for left/right, see the next page). The `_L` bone behaves like the single one produced by the default extrusion - you can grab/rotate/scale it exactly the same way. The `_R` bone is its mirror counterpart (along the armature’s local X axis), see (*Mirror extrusion example*).

**Warning:** Cancelling the extrude action causes the newly created bones to snap back to the source position, (*creating zero length bones*). These will be removed when exiting editmode, however they can cause confusion and it’s unlikely you want to keep them. If you realize the problem immediately undo the extrude action.

In case you’re wondering, you cannot just press X to solve this as you would in mesh editing, because extrusion selects the newly created tips, and as explained below the delete command ignores bones’ ends. To get rid of these extruded bones without undoing, you would have to move the tips, then select the bones and delete ([Deleting Bones](#)) them.

## Mouse Clicks

### Reference

Mode: *Edit mode*

Hotkey: Ctrl-LMB

If at least one bone is selected, Ctrl-LMB -clicking adds a new bone.

About the new bone's tip:

- after you `Ctrl-LMB`-clicked it becomes the active element in the armature,
- it appears to be right where you clicked, but...
- ... (as in mesh editing) it will be on the plane parallel to the view and passing through the 3D cursor.

The position of the root and the parenting of the new bone depends on the active element:



Fig. 2.954: Ctrl-clicking when the active element is a bone

If the active element is a **bone**

- the new bone's root is placed on the active bone's tip
- the new bone is parented and connected to the active bone (check the outliner in *Ctrl-clicking when the active element is a bone*).

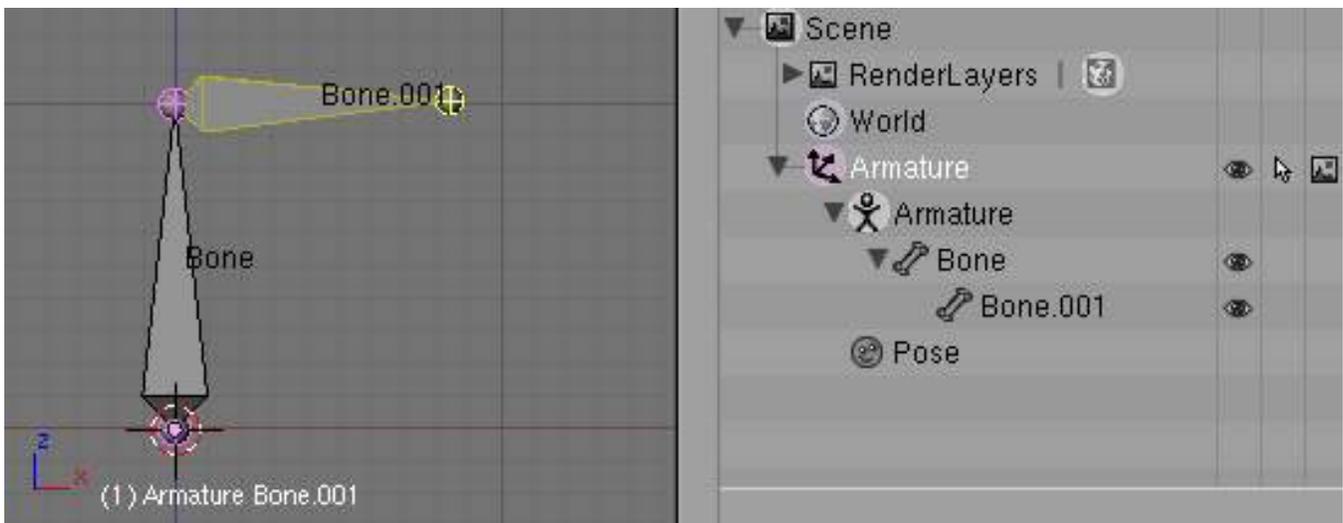


Fig. 2.955: Ctrl-clicking when the active element is a tip

If the active element is a **tip** :

- the new bone's root is placed on the active tip
- the new bone is parented and connected to the bone owning the active tip (check the outliner in *Ctrl-clicking when the active element is a tip*).

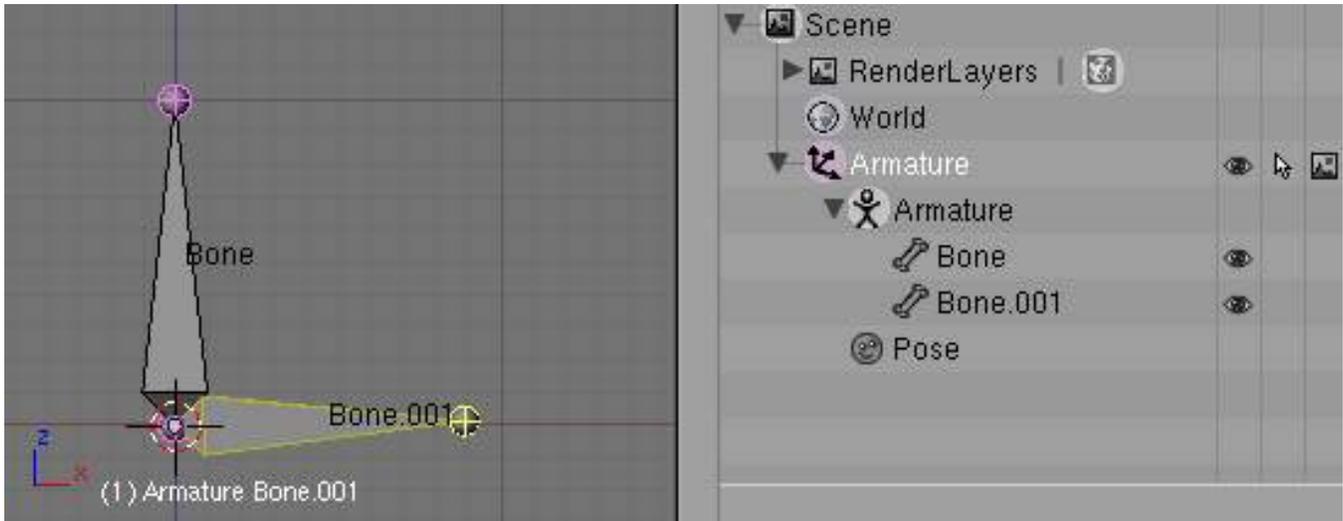


Fig. 2.956: Ctrl-clicking when the active element is a disconnected root

If the active element is a **disconnected root** :

- the new bone's root is placed on the active root
- the new bone is **NOT** parented to the bone owning the active root (check the outliner in *Ctrl-clicking when the active element is a disconnected root*).

And hence the new bone will **not** be connected to any bone.

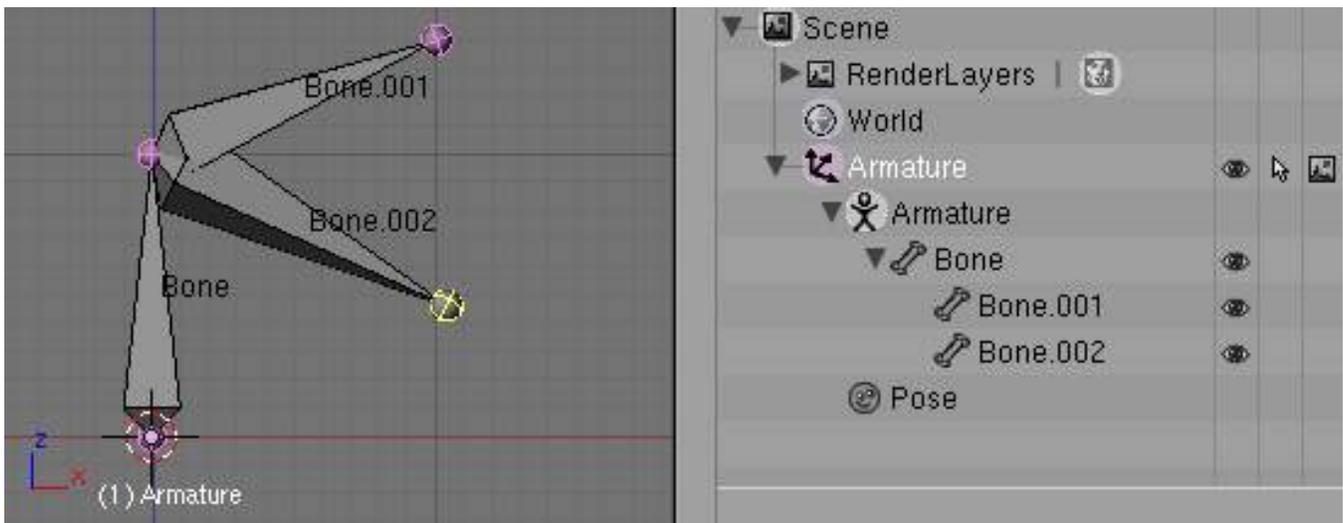


Fig. 2.957: Ctrl-clicking when the active element is a connected root

If the active element is a **connected root** :

- the new bone's root is placed on the active root

- the new bone **IS** parented and connected to the parent of the bone owning the active root (check the outliner in *Ctrl-clicking when the active element is a connected root*).

This should be obvious because if the active element is a connected root then the active element is also the tip of the parent bone, so it is the same as the second case.

As the tip of the new bone becomes the active element, you can repeat these ctrl-clicks several times, to consecutively add several bones to the end of the same chain.

### Fill between joints

---

#### Reference

Mode: *Edit mode*

Menu: *Armature → Fill Between Joints*

Hotkey: *F*

---

The main use of this tool is to create one bone between two selected ends by pressing *F*, similar to how in mesh editing you can “create edges/faces”.

If you have one root and one tip selected, the new bone:

- will have the root placed on the selected tip
- will have the tip placed on the selected root
- will be parented and connected to the bone owning the selected tip

Table

2.24:

Active  
tip on the  
right



If you have two tips selected, the new bone:

- will have the root placed on the selected tip closest to the 3D cursor
- will have the tip placed on the other selected tip
- will be parented and connected to the bone owning the tip used as the new bone’s root

Table

2.25: 3D

cursor on  
the right



If you have two roots selected, you will face a small problem due to the event system in Blender not updating the interface in real time.

When clicking *F*, similar to the previous case, you will see a new bone:

- with the root placed on the selected root closest to the 3D cursor

- with the tip placed on the other selected root
- parented and connected to the bone owning the root used as the new bone's root

If you try to move the new bone, Blender will update the interface and you will see that the new bone's root moves to the tip of the parent bone.

Table  
2.26:  
After UI  
update,  
correct  
visualiza-  
tion



Clicking **F** with only one bone end selected will create a bone from the selected end to the 3D cursor position, and it won't parent it to any bone in the armature.

Table  
2.27: Fill  
with only  
one root  
selected



You will get an error when:

- trying to fill two ends of the same bone, or
- trying to fill more than two bone ends.

## Duplication

---

### Reference

Mode: *Edit mode*

Menu: *Armature* → *Duplicate*

Hotkey: *Shift-D*

---

**Note:** This tool works on selected bones; selected ends are ignored.

---

As in mesh editing, by pressing *Shift-D*:

- the selected bones will be duplicated,
- the duplicates become the selected elements and they are placed in grab mode, so you can move them wherever you like.

If you select part of a chain, by duplicating it you'll get a copy of the selected chain, so the copied bones are interconnected exactly like the original ones.

The duplicate of a bone which is parented to another bone will also be parented to the same bone, even if the root bone is not selected for the duplication. Be aware, though, that if a bone is parented **and connected** to an unselected bone, its copy will be parented **but not connected** to the unselected bone (see *Duplication example*).

Table

2.28: The three duplicated bones.

Note that the selected chain is preserved in the copy, and that Bone.006 is parented but not connected to Bone.001, as indicated by the black dashed line.

Similarly, Bone.007 is parented but not connected to Bone.003.



## Deleting Bones

You have two ways to remove bones from an armature: the standard deletion, and merging several bones in one.

### Standard deletion

---

## Reference

Mode: *Edit mode*

Menu: *Armature* → *Delete*

Hotkey: X

---

**Note:** This tool works on selected bones: selected ends are ignored.

---

To delete a bone, you can:

- press the standard X key and confirm, or
- use the menu *Armature* → *Delete* and confirm.

If you delete a bone in a chain, its child(ren) will be automatically re-parented to its own parent, **but not connected**, to avoid deforming the whole armature.

Table

2.29:

The two bones have been deleted.

Note that Bone.002, previously connected to the deleted Bone.001, is now parented but not connected to Bone.



## Merge

---

### Reference

Mode: *Edit mode*

Menu: *Armature* → *Merge*

Hotkey: `Alt-M`

---

You can merge together several selected bones, *as long as they form a chain*. Each sub-chain formed by the selected bones will give one bone, whose root will be the root of the root bone, and whose tip will be the tip of the tip bone.

Confirm by clicking on *Within Chains* in the *Merge Selected Bones* pop-up.

If another (non-selected) chain originates from inside of the merged chain of bones, it will be parented to the resultant merged bone. If they were connected, it will be connected to the new bone.

Here's a strange subtlety (see *Merge example*): even though connected (the root bone of the unmerged chain has no root sphere), the bones are not visually connected - this will be done as soon as you edit one bone, differently depending in which chain is the edited bone (compare the bottom two images of the example to understand this better).

Table

2.30: The tip of Bone.006 has been translated, and hence the root of Bone.003 was moved to the tip of Bone.006



## Subdividing Bones

### Reference

Mode: *Edit mode*

Menu: *Armature* → *Subdivide*, *Armature* → *Subdivide Multi*

Hotkey:  $\bar{W}$ -1,  $\bar{W}$ -2

You can subdivide bones, to get two or more bones where there was just one bone. The tool will subdivide all selected bones, preserving the existing relationships: the bones created from a subdivision always form a connected chain of bones.

To create two bones out of each selected bone:

- press  $\bar{W}$ *pop-up* → *Subdivide*, same as  $\bar{W}$ -1, or
- select *Armature* → *Subdivide* from the header menu

To create an arbitrary number of bones from each selected bone:

- press  $\bar{W}$ *pop-up* → *Subdivide Multi*, same as  $\bar{W}$ -2, or
- select *Armature* → *Subdivide Multi* from the header menu, an

Then specify the number of cuts you want in the pop-up. As in mesh editing, if you set  $n$  cuts, you'll get  $n+1$  bones for each selected bone.

Table  
2.31: The  
selected  
bone  
has been  
“cut” two  
times,  
giving  
three sub-  
bones.



## Locking Bones

You can prevent a bone from being transformed in *Edit mode* in several ways:

- The active bone can be locked clicking on *Lock* in the *Transform Properties* panel (N in a 3D view);
- all bones can be locked clicking on the *Lock* button of their sub-panels in the *Armature Bones* panel;
- press `Shift-W` pop-up → *Toggle Settings* → *Locked*
- select *Armature* → *Bone Settings* → *Toggle a Setting*).

If the root of a locked bone is connected to the tip of an unlocked bone, it won't be locked, i.e. you will be able to move it to your liking. This means that in a chain of connected bones, when you lock one bone, you only really lock its tip. With unconnected bones, the locking is effective on both ends of the bone.

## X-Axis Mirror Editing

Another very useful tool is the *X-Axis Mirror* editing option (*Tool panel* > *Armature Options*, while *Armature* is selected in *Edit Mode*), working a bit like the same mesh editing tool. When you have pairs of bones of the same name with just a different “side suffix” (e.g. `.R` / `.L`, or `_right` / `_left` ...), once this option is enabled, each time you transform (move/rotate/scale...) a bone, its “other side” counterpart will be transformed accordingly, through a *symmetry along the armature local X axis*. As most rigs have at least one axis of symmetry (animals, humans, ...), it's an easy way to spare you half of the editing work! See also [next page](#) for more on naming bones.

## Separating Bones in a new Armature

You can, as with meshes, separate the selected bones in a new armature object (*Armature* → *Separate*, `Ctrl-Alt-P`) - and of course, in *Object* mode, you can join all selected armatures in one (*Object* → *Join Objects*, `Ctrl-J`).

## 2.5.10 Editing Bone Properties

In this page, you will learn how to edit and control most of the properties for Blender bones - For editing bones in an armature, you should read the [previous page](#) first! We will see how to manage the bones' relationships (*Chain Editing*), rename them (*Naming Bones*), etc.

## Transforming Bones

We won't detail here the various transformations of bones, nor things like axis locking, pivot points, and so on, as they are common to most object editing, and already described [here](#) (note however that some options, like snapping, do not seem to

work, even though they are available...). The same goes for mirroring, as it's nearly the same as with `mesh editing`. Just keep in mind that bones' roots and tips behave more or less like meshes' vertices, and bones themselves act like edges in a mesh.

As you know, bones can have two types of relationships: They can be parented, and in addition connected. Parented bones behave in *Edit* mode exactly as if they had no relations - you can grab, rotate, scale, etc. a parent bone without affecting its descendants. However, connected bones must always have parent's tips connected to child's roots, so by transforming a bone, you will affect all its connected parent/children/siblings.

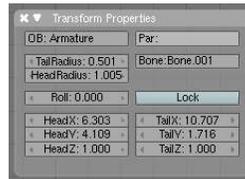


Fig. 2.994: The Transform Properties panel for armatures in Edit mode.

Finally, you can edit in the *Transform Properties* panel (N) the positions and radius of both ends of the active selected bone, as well as its `roll` rotation.

### Radius and Scaling in Envelope Visualization

#### Reference

Mode: *Edit mode*, *Envelope* visualization

Menu: *Armature* → *Transform* → *Scale*

Hotkey: S

When bones are displayed using *Octahedron*, *Stick* or *B-Bone* visualizations, scaling will behave as expected, similar to scaling mesh objects. When bones are displayed using *Envelope* visualization, scaling will have a different effect: it will scale the radius of the selected bones's ends. (see: [skinning part](#)). As you control only one value (the radius), there is no axis locking here. And as usual, with connected bones, you scale at the same time the radius of the parent's tip and of the children's roots.

Table

2.32:

...Scaled  
in En-  
velope  
visual-  
ization -  
its length  
remains  
the same,  
but its  
ends'  
radius are  
bigger.



Note that when you resize a bone (either by directly scaling it, or by moving one of its ends), Blender automatically adjusts the end-radii of its envelope proportionally to the size of the modification. Therefore, it is advisable to place all the bones first, and only then edit these properties.

## ScaleB and Envelope

---

### Reference

Mode: *Edit mode*

Hotkey: `Ctrl-Alt-S`

---

`Ctrl-Alt-S` activates a transform tool that is specific to armatures. It has different behavior depending on the active visualization, as explained below:

In *Envelope* visualization, it allows you to edit the influence of the selected bones (their *Dist* property, see the [skinning part](#)) - as with the “standard” scaling with this visualization (see the previous section), this is a one-value property, so there is no axis locking and such.

Table

2.33: Its envelope scaled with `[ctrl][alt][S]`.



In the other visualizations, it allows you to edit the “bone size”. This seems to only have a visible effect in *B-Bone* visualization, but is available also with *Octahedron* and *Stick ...* This tool in this situation has another specific behavior: While with other transform tools, the “local axes” means the object’s axes, here they are the bone’s own axes (when you lock to a local axis, by pressing the relevant key twice, the constraint is applied along the selected bone’s local axis, not the armature object’s axis).

**WARNING!** If you have more than one bone selected, using this tool crashes Blender!

Table

2.34: The same armature in Object mode and B-Bone visualization, with Bone.004’s size scaled up.



## Bone Direction

---

## Reference

Mode: *Edit* mode

Menu: *Specials* → *Switch Direction*

Hotkey:  $\bar{w}$ -3

---

This tool is not available from the *Armature* menu, but only from the *Specials* pop-up menu( $\bar{w}$ ). It allows you to switch the direction of the selected bones (i.e. their root will become their tip, and vice versa).

*Switching the direction of a bone will generally break the chain(s) it belongs to.* However, if you switch a whole (part of a) chain, the switched bones will still be parented/connected, but in “reversed order”. See the *Switching example*.

Table  
2.35: The selected bones have been switched. Bone.005 is no more connected nor parented to anything. The chain of switched bones still exists, but reversed (Now Bone.002 is its root, and Bone is its tip). Bone.003 is now a free bone.



## Bone Roll

---

### Reference

Mode: *Edit* mode

Menu: *Armature* → *Bone Roll* → ...

Hotkey: `Ctrl-R`, `Ctrl-N`

In *Edit* mode, you have options dedicated to the control of the bone roll rotation (i.e. the rotation around the Y axis of the bone). Each time you add a new bone, its default roll is so that its Z axis is as perpendicular to the current 3D view as possible. And each time you transform a bone, Blender tries to determine its best roll...

But this might lead to an unclear armature, with bones rolled in all angles... nasty! To address this problem, you have three options:

- *Armature* → *Bone Roll* → *Set Roll* (`Ctrl-R`) will start a roll-specific rotation, which behaves like any other transform operations (i.e. move the mouse and LMB click to validate, or type a numeric value and press `Return` or RMB click or press `Esc` to cancel everything).
- *Armature* → *Bone Roll* → *Clear Roll (Z-Axis Up)* (or `Ctrl-N-1pop-up` → *Recalculate Bone Roll Angles* → *Clear Roll (Z-Axis Up)*) will reset the selected bone roll so that their Z axis is as much as possible aligned with the global Z axis.
- *Armature* → *Bone Roll* → *Roll to Cursor* (or `Ctrl-N-2pop-up` → *Recalculate Bone Roll Angles* → *Align Z-Axis to 3D-Cursor*) will set the selected bone roll so that their Z axis is as much as possible pointed to the 3D cursor.

## Properties

### Reference

Mode: *Edit* mode

Panel: *Armature Bones* (*Editing* context)

Menu: *Armature* → *Bone Settings* → ...

Hotkey: `Shift-W`, `Ctrl-Shift-W`, `Alt-W`

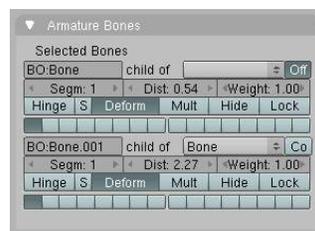


Fig. 2.1017: The Armature Bones panel in Edit mode.

Most bones' properties (excepted the transform ones) are regrouped in each bone's sub-panel, in the *Armature Bones* panel (*Editing* context'). Let's detail them.

Note that some of them are also available in the 3D views, through the three pop-up menus *Toggle Setting* (`Shift-W` or *Armature* → *Bone Settings* → *Toggle a Setting*), *Enable Setting* (`Ctrl-Shift-W` or *Armature* → *Bone Settings* → *Enable a Setting*), and *Disable Setting* (`Alt-W` or *Armature* → *Bone Settings* → *Disable a Setting*) - all three have the same entries, their respective effect should be obvious...

**BO** The bone name field, see *Naming Bones*.

**child of** These two settings control the bone relationship, as detailed in *Chain Editing*.

**Segm** This setting controls the number of segments that a bone has; see *Bone Rigidity Settings*.

**Dist, Weight, Deform** (also *[shift][W]* → *Deform & co*), **Mult** (also *[shift][W]* → *Mult VG & co*)

These settings control how the bone influences its geometry - along with the bones' ends radius. This will be detailed in the [skinning part](#).

**Hinge** (also *[shift][W]* → *Hinge & co*), **S** (also *[shift][W]* → *No Scale & co*) These settings affect the behavior of children bones while transforming their parent in *Pose* mode, so this will be detailed in the [posing part](#) !

**Hide** This will hide the bone (same as pressing H in the 3D views; see [this page](#)).

**Lock** (also *[shift][W]* → *Locked & co*) This will prevent all editing of the bone in *Edit* mode; see [previous page](#).

**Layers button** These small buttons allow you to control to which bone layer this bone belongs; see [this page](#).

## Bone Rigidity Settings

### Reference

Mode: *Edit* and *Pose* modes

Panel: *Armature Bones* (*Editing* context)



Fig. 2.1018: The Armature Bones panel in Pose mode.

Even though you have the *Segm* setting available in *Edit* mode (bones sub-panel, in the *Armature Bones* panel), you should switch to the *Pose* mode (*Ctrl-Tab*) to edit these “smooth” bones’ properties - one explanation to this strange need is that in *Edit* mode, even in *B-Bone* visualization, bones are drawn as sticks, so you can’t visualize the effects of these settings.

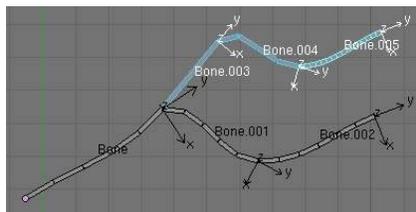


Fig. 2.1019: An armature in Pose mode, B-Bone visualization: Bone.003 has one segment, Bone.004 has four, and Bone.005 has sixteen.

We saw in [this page](#) that bones are made of small rigid segments mapped to a “virtual” Bézier curve. The *Segm* numeric field allows you to set the number of segments inside a given bone - by default, it is **1**, which gives a standard rigid bone! The higher this setting (max **32**), the smoother the bone, but the heavier the pose calculations...

Each bone’s ends are mapped to its “virtual” Bezier curve’s “auto” handle. Therefore, you can’t control their direction, but you can change their “length” using the *In* and *Out* numeric fields, to control the “root handle” and “tip handle” of the

bone, respectively. These values are proportional to the default length, which of course automatically varies depending on bone length, angle with previous/next bones in the chain, and so on.

<b>Bone In / Out settings example, with a materialized Bézier curve.</b>		
--	--	--

## Chain Editing

---

### Reference

Mode: *Edit* mode

Panel: *Armature Bones* (*Editing* context)

Menu: *Armature* → *Parent* → ...

Hotkey: `Ctrl-P`, `Alt-P`

---

You can edit the relationships between bones (and hence create/modify the chains of bones) both from the 3D views and the *Buttons* window. Whatever method you prefer, it's always a matter of deciding, for each bone, if it has to be parented to another one, and if so, if it should be connected to it.

To parent and/or connect bones, you can:

- In a 3D view, select the bone and *then* its future parent, and press `Ctrl-P` (or *Armature* → *Parent* → *Make Parent...*). In the small *Make Parent* menu that pops up, choose *Connected* if you want the child to be connected to its parent, else click on *Keep Offset*. If you have selected more than two bones, they will all be parented to the last selected one. If you only select one already-parented bone, or all selected bones are already parented to the last selected one, your only choice is to connect them, if not already done. If you select only one non-parented bone, you'll get the *Need selected bone(s)* error message...

*With this method, the newly-children bones won't be scaled nor rotated - they will just be translated if you chose to connect them to their parent's tip.*

- In the *Buttons* window, *Armature Bones* panel, for each selected bone, you can select its parent in the *Parent* drop-down list to the upper right corner of its sub-panel. If you want them to be connected, just enable the little *Con* button to the right of the list.

*With this method, the tip of the child bone will never be translated - so if *Con* is enabled, the child bone will be completely transformed by the operation.*

Table  
2.36:  
Bone.005  
parented  
and con-  
nected to  
Bone.002,  
using the  
Parent  
drop-  
down  
list of  
Bone.005  
sub-  
panel.



To disconnect and/or free bones, you can:

- In a 3D view, select the desired bones, and press `Alt-P` (or *Armature* → *Parent* → *Clear Parent...*). In the small *Clear Parent* menu that pops up, choose *Clear Parent* to completely free all selected bones, or *Disconnect Bone* if you just want to break their connections.
- In the *Buttons* window, *Armature Bones* panel, for each selected bone, you can select no parent in the *Parent* drop-down list of its sub-panel, to free it completely. If you just want to disconnect it from its parent, disable the *Con* button.

Note that relationships with non-selected children are never modified.

## Naming Bones

---

### Reference

Mode: *Edit* mode

Panel: *Armature Bones* (*Editing* context), *Transform Properties* (3D views, N)

---

You can rename your bones, either using the *Bone* field of the *Transform Properties* panel in the 3D views, for the active bone (N), or using the *BO* field in each bone sub-panel of the *Armature Bones* panel (*Editing* context).

Blender also provides you some tools that take advantage of bones named in a left/right symmetry fashion, and others that automatically name the bones of an armature. Let's look at this in detail.

### Naming Conventions

Naming conventions in Blender are not only useful for you in finding the right bone, but also to tell Blender when any two of them are counterparts.

In case your armature can be mirrored in half (i.e. it's bilaterally symmetrical), it's worthwhile to stick to a left/right naming convention. This will enable you to use some tools that will probably save you time and effort (like the *X-Axis Mirror* editing tool we saw above...).

- First you should give your bones meaningful base-names, like *leg*, *arm*, *finger*, *back*, *foot*, etc.

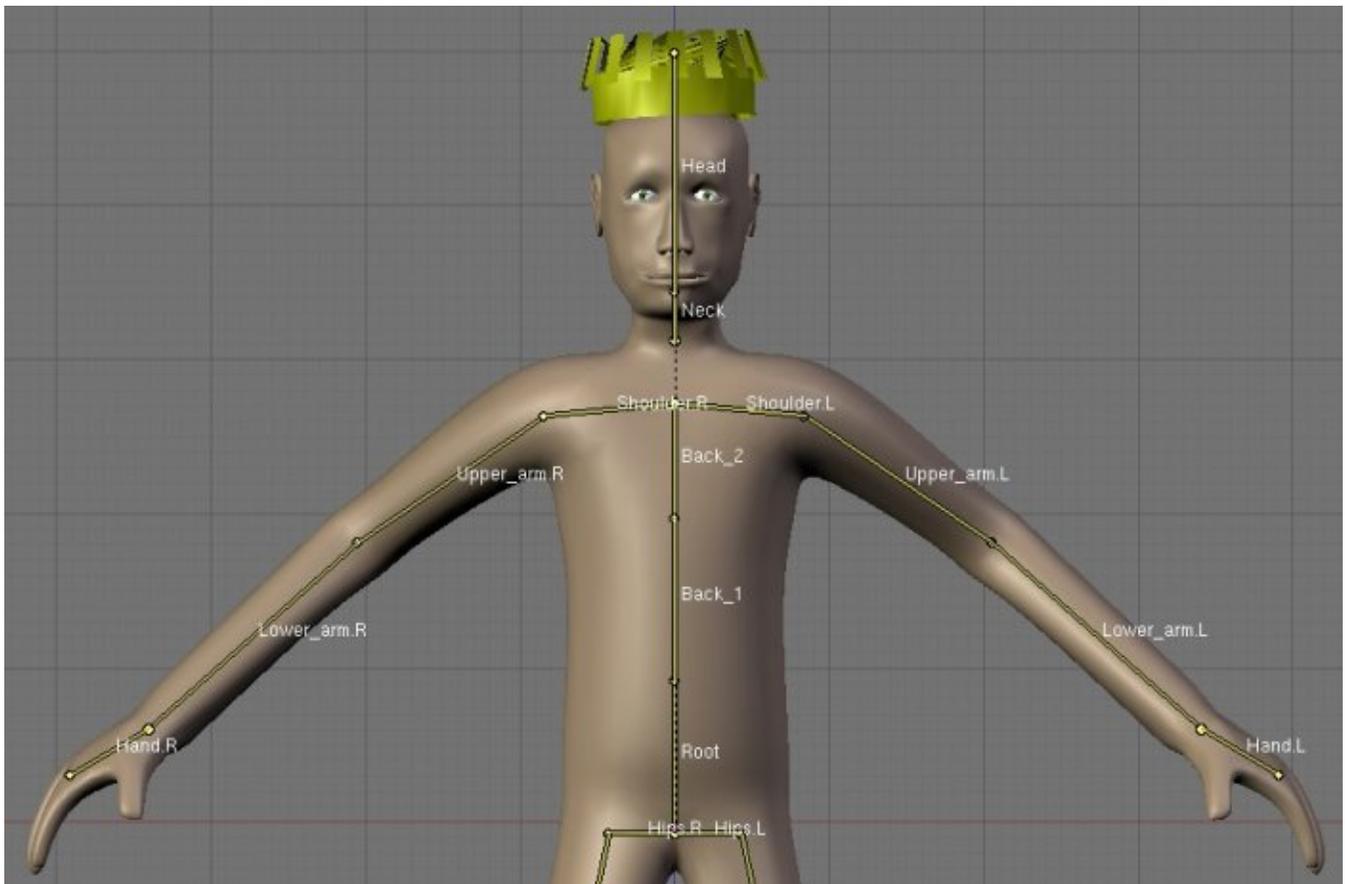


Fig. 2.1032: An example of left/right bone naming in a simple rig.

- If you have a bone that has a copy on the other side (a pair), like an arm, give it one of the following separators:
    - Left/right separators can be either the second position (`L_calfbone`) or last-but-one (`calfbone.R`)
    - If there is a lower or upper case L, R, left or right, Blender handles the counterpart correctly. See below for a list of valid separators. Pick one and stick to it as close as possible when rigging; it will pay off. Examples of **valid saporators**:
      - \* *(nothing)*: `hand Left -> hand Right`
      - \* *(underscore)*: `Hand _L -> Hand _R`
      - \* *(point)*: `hand .l -> hand .r`
      - \* *(dash)*: `Foot -l -> Foot -r`
      - \* *“ “ (space)*: `pelvis LEFT -> pelvis RIGHT`
- Note that all examples above are also valid with the left/right part placed before the name. You can only use the short L / R code if you use a separator (i.e. `handL / handR` won't work!).
- Before Blender handles an armature for mirroring or flipping, it first removes the number extension, if it's there (like `.001`)
  - You can copy a bone named `bla.L` and flip it over using `[W]` → *Flip Left-Right Names*. Blender will name the copy `bla.L.001` and flipping the name will give you `bla.R`.

## Bone name flipping

---

### Reference

Mode: *Edit mode*

Menu: *Armature* → *Flip Left & Right Names*

Hotkey: `W-4`

---

You can flip left/right markers (see above) in selected bone names, using either *Armature* → *Flip Left & Right Names*, or *Specials* → *Flip Left-Right Names* (`W-4`). This can be useful if you have constructed half of a symmetrical rig (marked for a left or right side) and duplicated and mirrored it, and want to update the names for the new side. Blender will swap text in bone names according to the above naming conventions, and remove number extensions if possible.

## Auto bone naming

---

### Reference

Mode: *Edit mode*

Menu: *Armature* → *AutoName Left-Right*, *Armature* → *AutoName Front-Back*, *Armature* → *AutoName Top-Bottom*

Hotkey: `W-5`, `W-6`, `W-7`

---

The three *AutoName* entries of the *Armature* and *Specials* (`W`) menus allows you to automatically add a suffix to all selected bones, *based on the position of their root relative to the armature center and its local coordinates* :

**AutoName Left-Right** will add the `.L` suffix to all bones with a positive X-coordinate root, and the `.R` suffix to all bones with a negative X-coordinate root. If the root is exactly at `0.0` on the X-axis, the X-coordinate of the tip is used. If both ends are at `0.0` on the X-axis, the bone will just get a period suffix, with no L/R (as Blender cannot decide whether it is a left or right bone...).

**AutoName Front-Back** will add the `.Bk` suffix to all bones with a positive Y-coordinate root, and the `.Fr` suffix to all bones with a negative Y-coordinate root. The same as with *AutoName Left-Right* goes for **0.0** Y-coordinate bones...

**AutoName Top-Bottom** will add the `.Top` suffix to all bones with a positive Z-coordinate root, and the `.Bot` suffix to all bones with a negative Z-coordinate root. The same as with *AutoName Left-Right* goes for **0.0** Z-coordinate bones...

## 2.5.11 Skeleton Sketching



Fig. 2.1033: The Bone Sketching panel in its default (inactive) state.

If you think that creating a whole rig by hand, bone after bone, is quite boring, be happy: some Blender developers had the same feeling, and created the Skeleton Sketching tool, formerly the Etch-a-ton tool, which basically allows you to “draw” (sketch) whole chains of bones at once.

Skeleton Sketching is obviously only available in *Edit* mode, in the 3D views. You control it through its *Skeleton Sketching* panel in the *Transform panel*, which you can open with `N`. Use mouse (LMB to draw strokes, and RMB for gestures). Showing its tool panel won’t enable sketching - *you must tick the checkbox next to Skeleton Sketching to start drawing bone chains* (otherwise, you remain in the standard *Edit* mode...).

Sketching is done in two steps:

- *Drawing Chains* (called “strokes”). Each stroke corresponds to a chain of bones.
- *Converting to Bones*, using different methods.

**The point of view is important**, as it determines the future bones’ roll angle: the Z axis of a future bone will be aligned with the view Z axis of the 3D view in which you draw its “parent” stroke (unless you use the\* *Template* converting method...). Strokes are drawn in the current view plane passing through the 3D cursor, but you can create somewhat “3D” strokes using the\* *Adjust* drawing option in different views (see below).

If you enable the small\* *Quick Sketch* option, the two steps are merged into one: once you have finalized the drawing of a stroke (see *Drawing Chains*), it is immediately converted to bones (using the current active method) and deleted. This option makes bone sketching quick and efficient, but you lose all the advanced stroke editing possibilities.

**Sketches are not saved into Blender files**, so you can’t interrupt a sketching session without losing all your work! Note also that the\* sketching is common to the whole Blender session, i.e. there is only one set of strokes (one sketch) in Blender, and not one per armature, or even per file...

### Drawing Chains

So, each stroke you draw will be a chain of bones, oriented from the starting point (the reddest or most orange part of the stroke) to its end (its whitest part). A stroke is made of several segments, delimited by small black dots - there will be at least one bone per segment (except with the\* *Template* conversion method, see [next page](#)), so all black points represents future bones’ ends. There are two types of segments, which can be mixed together:

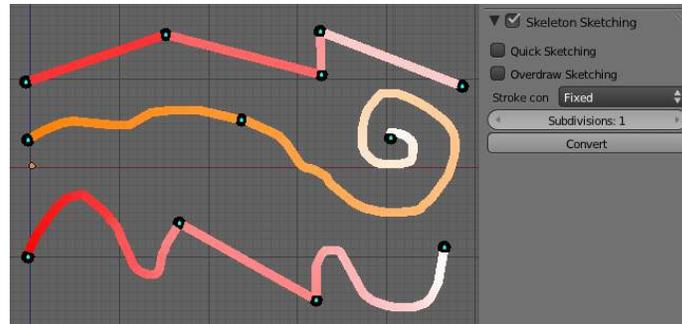


Fig. 2.1034: Strokes example. From top to bottom: A selected polygonal stroke of four straight segments, oriented from left to right. An unselected free stroke of two segments, oriented from left to right. A mixed stroke, with one straight segment between two free ones, right to left.

### Straight Segments

To create a straight segment, click\* **LMB** at its starting point. Then move the mouse cursor, without pressing any button - a dashed red line represents the future segment. Click **LMB** again to finalize it. Each straight segment of a stroke will always create one and only one bone, whatever convert algorithm you use (except for the\* *Template* conversion method).



### Free Segments

To create a free (curved) segment, click\* and hold **LMB** at its starting point. Then draw your segment by moving the mouse cursor - as in any paint program! Release **LMB** to finalize the segment - you will then be creating a new straight segment, so if you would rather start a new free segment, you must immediately re-press **LMB**. The free segments of a stroke will create different number of bones, in different manners, depending on the conversion method used. The future bones' ends for the current selected method are represented by small green dots for each one of those segments, for the selected strokes only. The free segment drawing uses the same\* *Manhattan Dist* setting as the [grease pencil tool](#) (*User Preferences window, Edit Methods* "panel", *Grease Pencil* group) to control where and when to add a new point to the segment - so if you feel your free segments are too detailed, raise this value a bit, and if you find them too jagged, lower it.

Table

2.37: But  
if you im-  
mediately  
click  
again and  
drag **LMB**  
you'll  
instead  
start a  
new free  
segment.



You finalize a whole stroke by clicking\* **RMB**. You can cancel the stroke you are drawing by pressing **Esc**. You can also snap strokes to underlying meshes by holding\* **Ctrl** while drawing. By the way, the *Peel Objects* button at the bottom of the *Bone Sketching* panel is the same thing as the "monkey" button of the snapping header bar controls shown when\* *Volume snap*

element is selected - see the `snap to mesh` page for details.

## Selecting Strokes

A stroke can be selected (materialized by a solid red-to-white line), or not (shown as an orange-to-white line) - see (Strokes example) above. As usual, you select a stroke by clicking\* RMB on it, you add one to/remove one from the current selection with a\* Shift-RMB click, and A (de)selects all strokes...

## Deleting

Hitting\* X or clicking on the *Delete* button (*Bone Sketching* panel) deletes the selected strokes (be careful, no warning/confirmation pop-up menu here). See also *Gestures*.

## Modifying Strokes

You can adjust, or “redraw” your strokes by enabling the *Overdraw Sketching option of the Bone Sketching* panel. This will modify the behavior of the strokes drawing (i.e. LMB clicks and/or hold): when you draw, you won’t create a new stroke, but rather modify the nearest one. The part of the old stroke that will be replaced by the new one are drawn in gray. This option does not take into account stroke selection, i.e. all strokes can be modified this way, not just the selected ones... Note also that even if it is enabled, when you draw too far away from any other existing stroke, you won’t modify any of them, but rather create a new one, as if\* *Overdraw Sketching* was disabled.

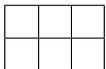
Adjusting stroke example.	

Finally, note that there is no undo/redo for sketch drawing...

## Gestures

There quite a few things about strokes editing that are only available through gestures. Gestures are started by clicking and holding FIXME(Template Unsupported: Shortcut/Keypress; {{Shortcut/Keypress|shift}}) LMB (when you are not already drawing a stroke), and materialized by blue-to-white lines. A gesture can affect several strokes at once.

There is no direct way to cancel a gesture once you’ve started “drawing” it. So the best thing to do, if you change your mind (or made a “false move”), is to continue to draw until you get a disgusting scribble, crossing your stroke several times - in short, something that the gesture system would never recognize!



## Cut

To\* **cut** a segment (i.e. add a new black dot inside it, making two segments out of one), “draw” a straight line crossing the chosen segment where you want to split it.



## Delete

To\* **delete** a stroke, draw a “V” crossing the stroke to delete twice.



## Reverse

To **reverse** a stroke (i.e. the future chain of bones will be reversed), draw a “C” crossing twice the stroke to reverse.



## Converting to Bones

Once you have one or more selected strokes, you can convert them to bones, using either the\* *Convert* button of the *Bone Sketching* panel, or the corresponding gesture (see *Gestures*). Each selected stroke will generate a chain of bones, oriented from its reddest end to its whitest one. Note that converting a stroke does not delete it.

There are four different conversion methods - three “simple” ones, and one more advanced and complex, *Template*, that reuses bones from the same armature or from another one as a template for the strokes to convert, and which is detailed in [the next page](#). Anyway, remember that\* straight segments are always converted to one and only one bone (except for the *Template* conversion method), and that the future bones’ ends are shown as green dots on selected free segments.

Remember also that the roll rotation of the created bones has been set during their “parent” stroke drawing (except for the *Template* conversion method) - their Z axis will be aligned with the view Z axis of the active 3D view at draw time.

## Fixed

With this method, each free segment of the selected strokes will be uniformly divided in *n* parts (set in *Num* numeric field), i.e. will give *n* bones.



## Adaptative

With this method, each free segment of the selected strokes will create as many bones as necessary to follow its shape closely enough - this “closely enough” parameter being set by the *Thres* hold numeric field; higher values giving more bones, following more closely the segments’ shape. So the more twisted a free segment, the more bones it will generate.



## Length

With this method, each free segment of the selected strokes will create as many bones as necessary, so that none of them is longer than the *Length* numeric field value (in Blender Units).



## Retarget

Retarget template bone chain to stroke.

**Template** Template armature that will be retargeted to the stroke. This is a more complex topic, detailed in its [own page](#).

### Retarget roll mode

**None** Don’t adjust roll.

**View** Roll bones to face the view.

**Joint** Roll bone to original joint plane offset.

**Autoname** ...

**Number** ...

**Side** ...

## 2.5.12 Armature Templating

The idea of templating is to use an already existing armature as base (“template”) to create a new armature. It differs from a simple copy in that you can directly define the new armature different in some aspects than its reference rig.

In Blender, the only templating tool is the bone sketching one (Etch-a-ton, described in [the previous page](#)), with its *Template* conversion method - so you should have read its page before this one!

### Using Bone Sketching

---

#### Reference

Mode: *Edit* mode

Panel: *Bone Sketching* (3D View window)

Menu: *Armature* → *Bone Sketching*

Hotkey: P

---

The *Template* conversion method of *Bone Sketching* tool maps a copy of existing bones to each selected stroke. The new bones will inherit some of their properties (influence, number of segments, etc.) from the corresponding bones in the template, but they will acquire their lengths, rolls and rotation from the sketch; so these properties would be different as compared to the template.

This is easier to understand with some examples.

In the following image, `armature.002` is set as the template, and the stroke maps with `chain_a` of this template. None of the bones are selected in the template. Note that there is no second stroke to map with chain `chain_b` of the template. The result is shown at right: Blender creates a copy of `chain_a` and matches the bones with the stroke.

Blender also creates a copy of `chain_b`, but this chain is not altered in any way; because this command can map only one selected chain with a stroke.

In the following example, no template is selected. (In other words, all the action is within the armature itself.)

Two bones are selected in `chain_b`, and the property panel is set to map the joints with the stroke. So these two selected bones are copied and the newly created copy of the chain is matched with the stroke. (Note that the newly created bones are named in continuation of the original chain.)

<p>If you had selected both the chains (<code>Chain_a</code> and <code>Chain_b</code>), you would have still got the same result as in the example above, because the command maps to stroke only one selected chain.</p> <p>In the following example also, only one chain is selected, but there are three strokes to map to. In this case, the same chain is copied three times (once for each stroke) and then mapped to individual strokes. Note how a two-bone chain is fitted to a three-segment stroke.</p> <p>The newly created bones are numbered sequentially, after the original bones' names.</p>	
---	--



OK now let us see some important ground rules:

- This conversion method can use as reference bones either *the selected bones in the currently edited armature*, or *all bones from another armature*. In general, it is a better idea to create new “templated” bones inside the “reference” armature, so you can precisely select which bones to use as template - if you want the new bones in a different armature, you can then use the *Separate* (Ctrl-Alt-P) and optionally *Join* (Ctrl-J in *Object* mode) commands...
- This tool only considers *one chain of bones*, so it’s better to select only one chain of bones inside the current armature (or use a single-chain armature object as template). Else, the chain of the template containing the first created bones will be mapped to the selected strokes, and *the other chains will just be “copied” as is*, without any modification.
- This tool maps the same chain of bones on all selected strokes, so you can’t use multiple strokes to map a multi-chains template - you will rather get a whole set of new bones for each selected stroke!
- If you have strokes only made of straight segments, *they must have at least as much segments as there are bones in the template chain* (else, the newly created chain is not mapped at all to the stroke, and remains an exact duplicate of its template). If there are more segments than necessary, the conversion algorithm will chose the best “joints” for the bones to fit to the reference chain, using the same influence settings as for free segments (*A*, *L* and *D* settings, see below).
- If you try to *Convert* without template bones (i.e. either an empty armature selected as template, or no bones selected in the current edited armature), you will get the error message *No Template and no deforming bones selected*, and nothing will occur.

Table  
2.38:  
With  
another  
arma-  
ture as  
template.



Now, let us see the settings of this conversion method:

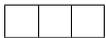
**No, View, Joint buttons** These three toggle buttons (mutually exclusive) control how the roll angle of newly created bones is affected:

**No** Do not alter the bones roll (i.e. the new bones’ rolls fit their reference ones).

**View** Roll each bone so that one of its X, Y or Z local axis is aligned (as much as possible) with the current view’s Z axis.

**Joint** New bones roll fit their original rotation (as *No* option), but with regards to the bend of the joint with its parent.

Table  
2.39:  
With  
Joint roll  
option.



**Template drop-down list** Here you select the armature to use as template. If you choose *None*, the selected bones from the currently edited armature will be used as reference, else all bones of the other armature will be used.

*A*, *L*, *D* are numeric fields.

Think of them as *A*(ngle of bones), *L*(ength of bones) and *D*(efinition of stroke).

These settings control how the template is mapped to the selected strokes. Each one can have a value between **0.0** and **10.0**, the default being **1.0**.

**A** controls the influence of the angle of the joints (i.e. angle between bones) - the higher this value, the more the conversion process will try to preserve these joints angle in the new chain.

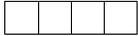
**L** controls the influence of the bones' length - the higher this value, the more the conversion process will try to preserve these lengths in the new bones.

**D** controls the influence of the stroke's shape - the higher this value, the more the conversion process will try to follow the stroke with the new chain.

Table 2.40:

A: 0.0; L:

0.0; D: 1.0.



**S and N text fields, “auto” button** These control how the new bones are named. By default, they just take the same names as the originals from the template - except for the final number, increased as needed. However, if the template bones have `&s` somewhere in their name, this “placeholder” will be replaced in the “templated” bones' names by the content of the *S* text field (“S” for “side”). Similarly, a `&n` placeholder will be replaced by the *N* field content (“N” for “number”). If you enable the small “auto” button, the *N* field content is auto-generated, producing a number starting from nothing, and increased each time you press the *Convert* button, and the `&s` placeholder is replaced by the side of the bone (relative to the local X axis: `r` for negative X values, `l` for positive ones).

Naming and placeholders, using a simple leg template.



Auto naming and placeholders, using a simple leg template.



**Static text line** The line just above the *Peel Objects* button gives you two informations:

- The *n joints* part gives you the number of joints (i.e. bones' ends, with connected ends considered as one joint), either from the selected bones of the edited armature, or in the whole other template armature.
- The second part is only present when another armature has been selected as template - it gives you *the root bone's name of the chain that will be mapped to the strokes*. Or, while you are drawing a stroke with straight segments, the name of the bone corresponding to the current segment (and *Done* when you have enough segments for all bones in the template chain).

## 2.5.13 Skinning

We have seen in [previous pages](#) how to design an armature, create chains of bones, etc. Now, having a good rig is not the final goal - unless you want to produce a “Dance Macabre” animation, you'll likely want to put some flesh on your skeletons! Surprisingly, “linking” an armature to the object(s) it should transform and/or deform is called the “skinning” process...

In Blender, you have two main skinning types:

- You can [Parent/Constrain Objects to Bones](#) - then, when you transform the bones in *Pose* mode, their “children” objects are also transformed, exactly as with a standard parent/children relationship... *The “children” are never deformed when using this method.*
- You can [Using the Armature modifier on entire Mesh](#), and then, some parts of this object to some bones inside this armature. This is the more complex and powerful method, and *the only way to really deform the geometry of the object*, i.e. to modify its vertices/control points relative positions.

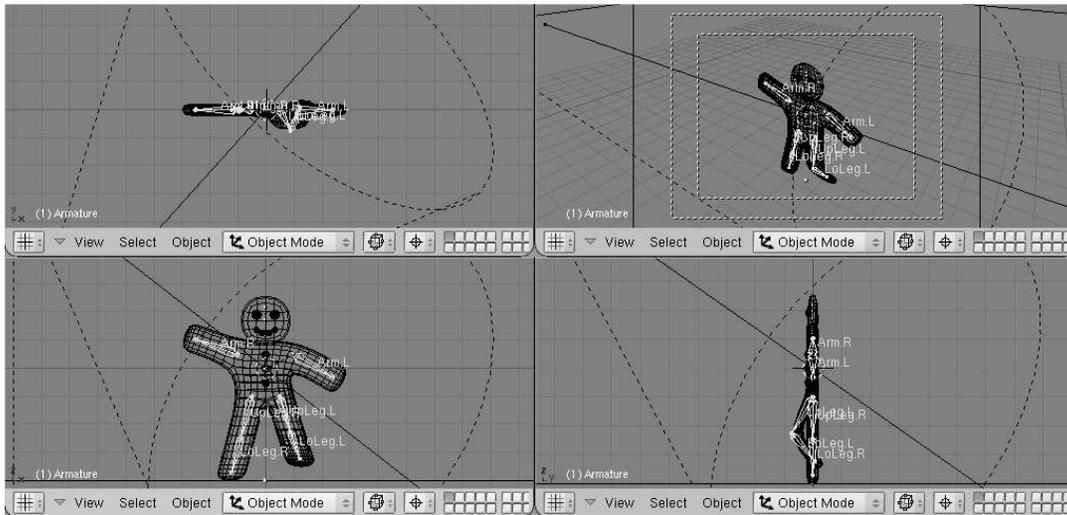


Fig. 2.1117: The ginebread mesh skinned on its armature.

## 2.5.14 Objects

### 2.5.15 Skinning to Shapes

We saw in the [previous page](#) how to link (parent) whole objects to armature bones - a way to control the transform properties of this object via a rig. However, armatures are much more powerful: they can deform the *shape* of an object (i.e. affect its ObData datablock - its vertices or control points...).

In this case, the child object is parented (skinned) to the whole armature, so that each of its bones controls a part of the “skin” object’s geometry. This type of skinning is available for meshes, lattices, curves, surfaces, and texts (with more options for the first two types).

Bones can affect the object’s shape in two ways:

- The *Envelope* process is available for all type of skinnable objects - it uses the “proximity” and “influence” of the bones to determine which part of the object they can deform.
- The *Vertex Groups* method is (obviously) reserved to meshes and lattices - one bone only affect the vertices in the *group* having the same name, using vertices’ *weights* as influence value. A much more precise method, but also generally longer to set up.

Both methods have some *Common Options*, and can be mixed together.

### Parenting to Whole Armatures

But before diving into this, let’s talk about the different ways to skin (parent) an object to a whole armature - as with *object skinning*, there is an “old parenting” method and a new, more flexible and powerful one, based on modifiers - which allows creation of very complex setups, with objects deformed by several armatures.

For *meshes and lattices only*, you can use the `Ctrl-P` parent shortcut in the 3D views (after having selected first the “skin” object, then the armature). The *Make Parent To* menu pops up, select the *Armature* entry. If the skinning object is a lattice, you’re done; no more options are available. But with a child mesh, another *Create Vertex Groups?* menu appears, with the following options - all regarding the “vertex groups” skinning method:

**With Empty Groups** will create, if they don’t already exist, empty groups, one for each bone in the skinned armature, with these bones’ names. Choose this option if you have already created (and weighted) all the vertex groups the mesh requires.

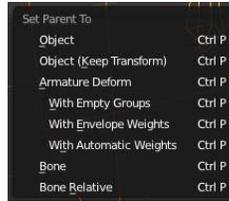


Fig. 2.1118: Set Parent menu

**With Envelope Weights** will create, as with *Name Groups* option, the needed vertex groups. However, it will also weight them according to the bones' envelope settings (i.e. it will assign to each groups the vertices that are inside its bone's influence area, weighted depending on their distance to this bone). This means that if you had defined vertex groups using same names as skinned bones, their content will be completely overridden! *You'll get the same behavior as if you used the envelopes skinning method, but with vertex groups?*

**Automatic Weights** Creates, as with *Envelope Weights* option, the needed vertex groups, with vertices assigned and weighted using the newer "bone heat" algorithm.

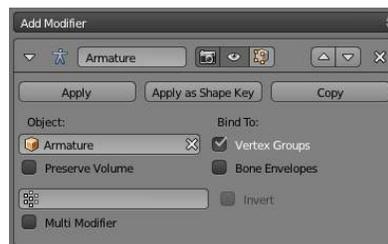


Fig. 2.1119: The Armature modifier.

This "parenting" method will create an\* [Armature modifier](#) in the skinning object's modifiers stack. And so, of course, adding an\* [Armature modifier](#) to an object is the second, new skinning method (which also works for curves/surfaces/texts...). Follow the above link to read more about this modifier's specific options. Note that there is a way with new *Armature* modifiers to automatically create vertex groups and weight them; see the [Vertex Groups](#) description below.

**Warning:** A single object can have several *Armature* modifiers (with e.g. different armatures, or different settings...), working on top of each other, **or** mixing their respective effects (depending whether their\* *Multi-Modifier* option is set, see their description for more details), and only one "virtual old parenting" one, which will always be at the top of the stack.

Note finally that for settings that are present in both the armature's *Armature panel* and in the objects' *Armature* modifier panel (namely, *Vertex Groups / VertGroups*, *Envelopes*, *Quaternion* and *B-Bone Rest*), the modifier ones always override the armature ones. This means that if, for example, you only enable the *Envelopes* deformation method of the armature, and then skin it with an object using an\* *Armature* modifier, where only *VertGroups* is enabled, the object will only be deformed based on its "bones" vertex groups, ignoring completely the bones' envelopes.

## Common Options

There are two armature-global skinning options that are common to both envelopes and vertex groups methods:\*

**Preserve Volume (Armature modifier)** This affects the way geometry is deformed, especially at bones' joints, when rotating them.

Without *Preserve Volume*, rotations at joints tend to scale down the neighboring geometry, up to nearly zero at 180d from rest position. With\* *Preserve Volume*, the geometry is no longer scaled down, but there is a "gap", a discontinuity when reaching\* 180d from rest position.

Table 2.41: 180.1- rotation, Preserve Volume enabled.

Note that the IcoSphere is deformed using the envelopes method.		

## Bone Deform Options

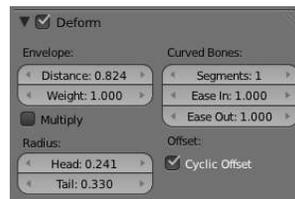


Fig. 2.1132: Bone Deform Options

The bones also have some deforming options in their sub-panels (\* *Armature Bones panel*), that you can therefore define independently for each of them

**Deform** By disabling this setting (enabled by default), you can completely prevent a bone from deforming the geometry of the skin object.

## Envelope

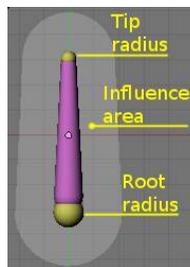


Fig. 2.1133: Bone influence areas for envelopes method.

Envelopes is the most general skinning method - it works with all available object types for skinning (meshes, lattices, curves, surfaces and texts). It is based on proximity between bones and their geometry, each bone having two different areas of influence, shown in the *Envelope* visualization:

- The inside area, materialized by the “solid” part of the bone, and controlled by both root and tip radius. Inside this zone, the geometry is fully affected by the bone.
- The outside area, materialized by the lighter part around the bone, and controlled by the *Dist* setting. Inside this zone, the geometry is less and less affected by the bone as it goes away - following a quadratic decay.

See the\* [editing pages](#) for how to edit these properties.

There is also a bone property, *Weight* (in each bone sub-panel, in\* *Edit mode only*, defaults to **1.0**), that controls the global influence of the bone over the deformed object, when using the envelopes method. It is only useful for the parts of geometry that are “shared”, influenced by more than one bone (generally, at the joints...) - a bone with a high weight will have more influence on the result than one with a low weight... Note that when set to\* **0.0**, it has the same effect as disabling the\* *Deform* option.

**Mult** Short for ‘Multiply’. This option controls how the two deforming methods interact when they are both enabled. By default, when they are both active, all vertices belonging to at least one vertex group are only deformed through the vertex groups method - the other “orphan” vertices being handled by the envelopes one. When you enable this option, the “deformation influence” that this bone would have on a vertex (based from its envelope settings) is multiplied with this vertex’s weight in the corresponding vertex group. In other words, the vertex groups method is further “weighted” by the envelopes method.

**Radius** Set the radius for the head and the tail of envelope bones.

### Curved Bone

*Curved Bones* (previously known as B-bones) allow you make bones act like bezier curve segments, which results in smoother deformations for longer bones.

See the [editing pages](#) for how to edit these properties.

### Vertex Groups

Vertex groups skinning method is only available for meshes and lattices - the only objects having [vertex groups](#). Its principle is very simple: each bone only affects vertices belonging to a vertex group having the same name as the bone. So if you have e.g. a forearm bone, it will only affect the forearm vertex group of its skin object(s).

The influence of one bone on a given vertex is controlled by the weight of this vertex in the relevant group. Thus, the [Weight Paint mode](#) (`Ctrl-Tab` with a mesh selected) is most useful here, to easily set/adjust the vertices’ weights.

However, you have a few goodies when weight-painting a mesh already parented to (skinning) an armature. For these to work, you must:

- Select the armature.
- Switch to *Pose* mode (`Ctrl-Tab`).
- Select the mesh to weight.
- Hit again `Ctrl-Tab` to switch to *Weight Paint* mode.

Now, when you select a bone of the armature (which remained in *Pose* mode), you automatically activate the corresponding vertex group of the mesh - Very handy! Obviously, you can only select one bone at a time in this mode (so `Shift-LMB` clicking does not work).

This way, you can also apply to the active bone/vertex group one of the same “auto-weighting” methods as available when doing an “old-parenting” to armature (`Ctrl-P`):

- Select the bone (and hence the vertex group) you want.
- Hit `W`, and in the *Specials* menu that pops up, choose either *Apply Bone Envelopes to Vertex Groups* or *Apply Bone Heat Weights to Vertex Groups* (names are self explanatory, I think). Once again, even though these names are plural, you can only affect *one* vertex group’s weights at a time with these options.

To automatically weight multiple bones, you can simply

- `Ctrl-Tab` out of *Weight Paint* Mode
- Select the Armature. It should be in *Pose* mode. If it isn’t, go `Ctrl-Tab`
- Select multiple bones `Shift-LMB` or press `A` (once or twice).
- Select Mesh again
- If not in weight paint already, toggle back into `Ctrl-Tab`
- Use the `W` menu to automatic weight. This will weight all the bones you selected in *Pose* Mode.

Table  
2.42: The  
same  
pose, but  
using  
envelopes  
method  
rather  
that  
vertex  
groups.



Obviously, the same vertex can belong to several groups, and hence be affected by several bones, with a fine tuning of each bone's influence using these vertex weights. Quite useful when you want to have a smooth joint. For example, when you skin an elbow, the upperarm vertex group contains the vertices of this part at full weight (\*\* 1.0 ), **and when reaching the elbow area, these weights decrease progressively to 0.0**' when reaching the forearm zone - and vice versa for the forearm group weights... Of course, this is a very raw example - skinning a realistic joint is a big job, as you have to carefully find good weights for each vertex, to have the most realistic behavior when bending - and this is not an easy thing!

### See Also

Making good but short examples about skinning to shapes is not an easy thing - so if you want better examples, have a look to this [BSoD tutorial](#), which illustrates (among many other things) the skinning of a simple human rig with a mesh object.

## 2.5.16 Retargeting

## 2.5.17 Posing

Once your armature is *skinned* by the needed object(s), you can start to pose it. Basically, by transforming its bones, you deform or transform the skin object(s). But you don't do that in *Edit* mode - remember that in this mode, you edit *the default, base, "rest" position of your armature*. You can't use the *Object* mode either, as here you can only transform whole objects...

So, armatures in Blender have a third mode, *Pose*, dedicated to this process. It's a sort of "object mode for bones". In rest position (as edited in *Edit* mode), each bone has its own position/rotation/scale to neutral values (i.e. 0.0 for position and rotation, and 1.0 for scale). Hence, when you edit a bone in *Pose* mode, you create an offset in its transform properties, from its rest position - this is quite similar to [meshes relative shape keys](#), in fact.

### Posing Section Overview

In this section, we will see:

- The [visualization features](#) specific to *Pose* mode.
- How to [select and edit bones](#) in this mode.
- How to [use pose library](#).
- How to [use constraints](#) to control your bones' DoF (degrees of freedom).
- How to [use inverse kinematics features](#).
- How to [use the Spline inverse kinematics features](#).

Even though it might be used for completely static purposes, posing is heavily connected with [animation features and techniques](#).

In this part, we will try to focus on animation-independent posing, but this isn't always possible. So if you know nothing about animation in Blender, it might be a good idea to read the [animation features and techniques](#) chapter first, and then come back here.

**See also:**

As usual, see the [tutorials](#) for more demonstrative examples, and especially [this BSoD one](#).

## 2.5.18 Visualization

We talk in [this page](#) about the armature visualization options available in all modes (the visualization types, the bones' shapes, etc.).

In *Pose* mode, you have extra features, *Colors* to help you visually categorize your bones, *Ghosts* and *Motion Paths* to help you visualize armatures' animations.

### Colors

In *Pose* mode, the bones can have different colors, following two different processes, controlled by the *Color* button (*Armature* panel, *Editing* context):

- When it is disabled, bones are colored based on their “state” (i.e. if they use constraints, if they are posed, etc.).
- When it is enabled, bones are colored depending on which bone group they belong to (or as above if they belong to no group).

You can also mix both coloring methods, see [Coloring from Bone Group](#) below ).

### Coloring from Bone State

This is the default and oldest way - there are six different color codes, ordered here by precedence (i.e. the bone will be of the color of the topmost valid state):

- **Purple:** The `Stride Root` bone.
- **Orange:** A bone with a targetless Solver constraint.
- **Yellow:** A bone with an [IK Solver constraint](#).
- **Green:** A bone with any other kind of constraint.
- **Blue:** A bone that is posed (i.e. has keyframes).
- **Gray:** Default state.

### Coloring from Bone Group

The bone groups panel is available in the Object data editor for an armature. Bone groups facilitate the coloring (theming) of multiple bones. Bone groups are managed mostly in the *Buttons* window, *Editing* context.

To create a new bone group, click on the *Add Group* button in the *Bone Groups: buttons set* (*Link and Materials* panel). Once created, you can use the top row of controls to select another group in the drop-down list (“arrows” button), rename the current group (text field), or delete it (“X” button).

To assign a selected bone to a given bone group you can do one of the following:

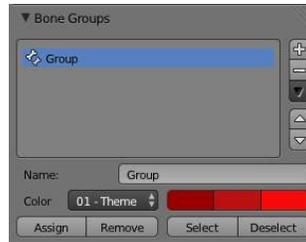


Fig. 2.1142: The Bone Groups panel with a bone group (default colors).

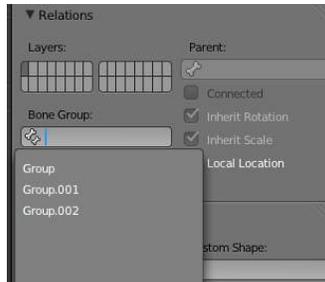


Fig. 2.1143: The Bone Group drop-down list of a bone sub-panel.

- In the Bone Groups, select an existing group, and click *Assign*
- In the Relations section of the *Bones* panel), use the *Bone Group* drop-down list to select the chosen one.

In the 3D views, using the *Pose* → *Bone Groups* menu entries, and/or the *Bone Groups* pop-up menu (Ctrl-G), you can:

**Assign to New Group** Assigns selected bones to a new bone group

**Assign to Group** Assigns selected bones to the selected Bone Groups

**Remove Selected from Bone Groups** Removes selected bones from all bone groups

**Remove Bone Group** Removes the active bone group

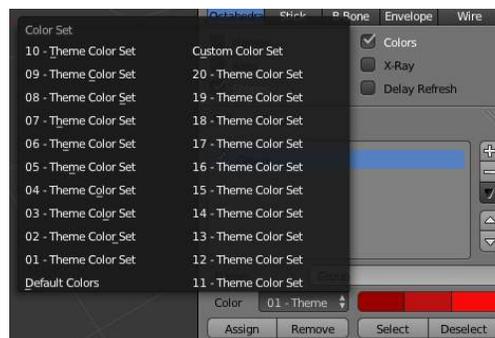


Fig. 2.1144: The Bone Color Set list of the bone group, and the color swatch of the chosen color theme.

You can also assign a “color theme” to a group (each bone will have these colors). Remember you have to enable the *Colors* button (*Armature* panel) to see these colors. Use the *Bone Color Set* drop-down list to select:

- The default (gray) colors (*Default Colors*).
- One of the twenty Blender presets (*nn - Theme Color Set*), common to all groups.
- A custom set of colors (*Custom Set*), which is specific to each group.

Below this list, you have three color swatches and a button.

- The first swatch is the color of unselected bones.
- The second swatch is the outline color of selected bones.
- The third swatch is the outline color of the active bone.

As soon as you click on a swatch (to change the color, through the standard color editing dialog), you are automatically switched to the *Custom Set* option.

## Ghosts

### Reference

Mode: *Pose* mode

Panel: *Visualisations*

Table  
2.43:  
Ghosts  
exam-  
ples.



If you are a bit familiar with traditional cartoon creation, you might know that drawing artists use tracing paper heavily, to see several frames preceding the one they are working on. This allows them to visualize the overall movement of their character, without having to play it back... Well, Blender features something very similar for armatures in *Pose* mode: the “ghosts”.



Fig. 2.1145: The Ghost panel showing the different options associated with different modes.

The ghosts are simply black drawings (more or less opaque) of the bones’ outlines as they are at certain frames.

The ghosts settings are found in the *Visualisations* panel (*Editing* context), only available in *Pose* mode. You have three different types of ghosts, sharing more or less the same options:

**Around Current Frame** This will display a given number of ghosts before and after the current frame. The ghosts are shaded from opaque at the current frame, to transparent at the most distant frames. It has three options:

**Range** This numeric field specifies how many ghosts you’ll have on both “sides” (i.e. a value of **5** will give you ten ghosts, five before the current frame, and five after).

**Step** This numeric field specifies whether you have a ghost for every frame (the default **1** value), or one each two frames, each three frames, etc.

**Selected Only** When enabled, you will only see the ghosts of selected bones (otherwise, every bone in the armatures has ghosts...)

**In Range** This will display the ghosts of the armature's bones inside a given range of frames. The ghosts are shaded from transparent for the first frame, to opaque at the last frame. It has four options:

**Start** This numeric field specifies the starting frame of the range (exclusive). Note that unfortunately, it cannot take a null or negative value - which means you can only see ghosts starting from frame **2** included...

**End** This numeric field specifies the ending frame of the range, and cannot take a value below *GSta* one.

**Step** Same as above.

**On Keyframes** This is very similar to the *In Range* option, but there are ghosts only for keyframes in the armature animation (i.e. frames at which you keyed one or more of the bones). So it has the same options as above, except for the *GStep* one (as only keyframes generate ghosts). Oddly, the shading of ghosts is reversed compared to *In Range* - from opaque for the first keyframe, to transparent for the last keyframe.

Finally, these ghosts are also active when playing the animation (**Alt-A**) - this is only useful with the *Around Current Frame* option, of course...

Note also that there is no “global switch” to disable this display feature - to do so, you have to either set *Ghost* to **0** (for *Around Current Frame* option), or the same frame number in both *GSta* and *GEnd* (for the two other ghosts types).

## Motion Paths

---

### Reference

Mode: *Pose* mode

Panel: *Visualisations*

Menu: *Pose* → *Motion Paths* → ...

Hotkey: **W-3**, **W-4**

---

This feature allows you to visualize as curves the paths of bones' ends (either their tips, by default, or their roots).

Before we look at its options (all regrouped in the same *Visualisations* panel, in the *Editing* context), let's first see how to display/hide these paths. Unlike *Ghosts*, you have to do it manually - and you have to first select the bones you want to show/hide the motion paths. Then,

- To show the paths (or update them, if needed), click on the *Calculate Path* button of the *Visualisations* panel, or, in the 3D views, select the *Pose* → *Motion Paths* → *Calculate Paths* menu entry (or use the *Specials* pop-up menu, **W-3**).
- To hide the paths, click on the *Clear Paths* button, or, in the 3D views, do *Pose* → *Motion Paths* → *Clear All Paths*, or **W-4**.

Remember: only selected bones and their paths are affected by these actions!

The paths are drawn in a light shade of gray for unselected bones, and a slightly blueish gray for selected ones. Each frame is materialized by a small white dot on the paths.

As with ghosts, the paths are automatically updated when you edit your poses/keyframes, and they are also active during animation playback (**Alt-A**, only useful when the *Around Current Frame* option is enabled).

And now, the paths options:

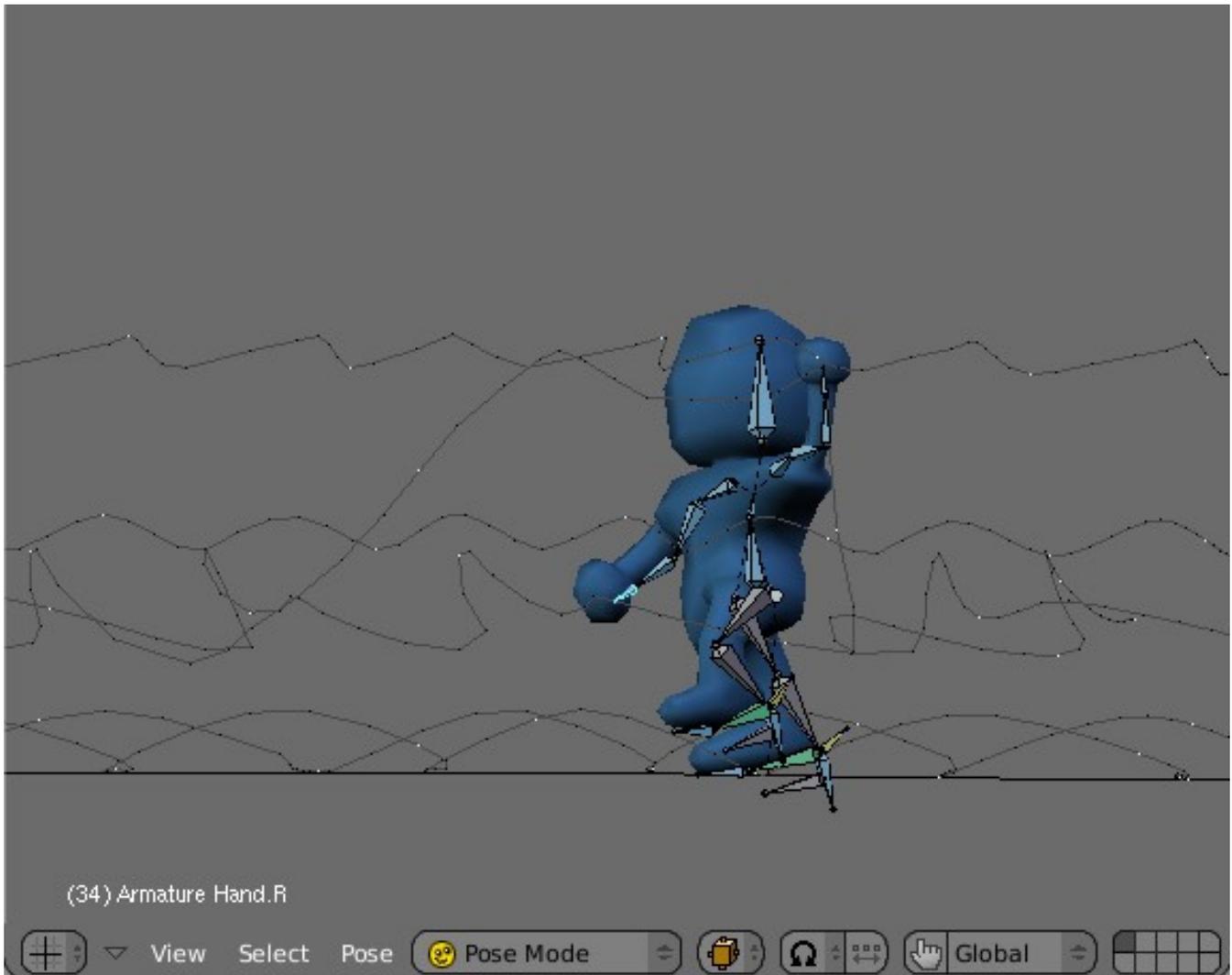


Fig. 2.1146: A motion paths example.

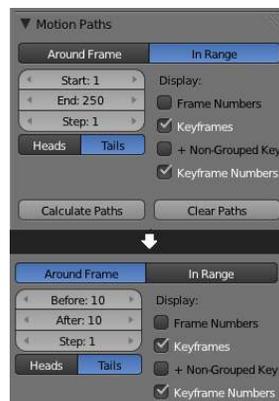


Fig. 2.1147: The Motion Paths Panel showing options for the different modes

**Around Frame** Around Frame, Display Paths of poses within a fixed number of frames around the current frame. When you enable this button, you get paths for a given number of frames before and after the current one (again, as with ghosts).

**In Range** In Range, Display Paths of poses within specified range.

#### Display Range

**Before/After** Number of frames to show before and after the current frame (only for ‘Around Current Frame’ Onion-skinning method)

**Start/End** Starting and Ending frame of range of paths to display/calculate (not for ‘Around Current Frame’ Onion-skinning method)

**Step** This is the same as the *GStep* for ghosts - it allows you to only display on the path one frame for each *n* ones. Mostly useful when you enable the frame number display (see below), to avoid cluttering the 3D views.

**Frame Numbers** When enabled, a small number appears next to each frame dot on the path, which is of course the number of the corresponding frame.

**Keyframes** When enabled, big yellow square dots are drawn on motion paths, materializing the keyframes of their bones (i.e. only the paths of keyed bones at a given frame get a yellow dot at this frame).

**Keyframe Nums** When enabled, you’ll see the numbers of the displayed keyframes - so this option is obviously only valid when *Show Keys* is enabled.

- **Non-Grouped Keyframes** For bone motion paths, search whole Action for keyframes instead of in group with matching name only (is slower)

#### Calculate

**Start / End** These are the start/end frames of the range in which motion paths are drawn. *You have to Calculate Paths again when you modify this setting*, to update the paths in the 3D views. Note that unlike with ghosts, the start frame is *inclusive* (i.e. if you set *PSta* to **1**, you’ll really see the frame **1** as starting point of the paths...).

**Bake Location** By default, you get the tips’ paths. By changing this setting to Tails, you’ll get the paths of the bone’s roots (remember that in Blender UI, bones’ roots are called “heads”...). *You have to Calculate Paths again when you modify this setting*, to update the paths in the 3D views.

## 2.5.19 Editing Poses

In *Pose* mode, bones behave like objects. So the transform actions (grab/rotate/scale, etc.) are very similar to the same ones in *Object* mode (all available ones are regrouped in the *Pose* → *Transform* sub-menu). However, there are some important specificities:

- Bones’ relationships are crucial (see *Effects of Bones Relationships*).
- The “transform center” of a given bone (i.e. its default pivot point, when it is the only selected one) is *its root*. Note by the way that some pivot point options seem to not work properly - in fact, except for the *3D Cursor* one, all others appear to always use the median point of the selection (and not e.g. the active bone’s root when *Active Object* is selected, etc.).

### Selecting Bones

Selection in *Pose* mode is very similar to the one in [Edit mode](#), with a few specificities:

- You can only select *whole bones* in *Pose* mode, not roots/tips...
- You can select bones based on their group and/or layer, through the *Select Grouped* pop-up menu (**Shift-G**):
  - To select all bones belonging to the same group(s) as the selected ones, use the *In Same Group* entry (**Shift-G-Numpad1**).

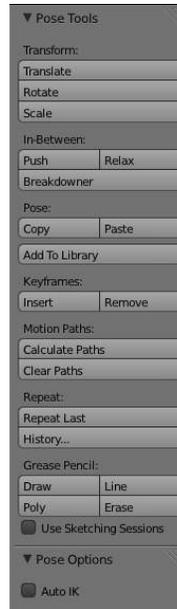


Fig. 2.1148: Pose Tools



Fig. 2.1149: The Select Grouped pop-up menu.

- To select all bones belonging to the same layer(s) as the selected ones, use the *In Same Layer* entry (Shift-G-Numpad2).

## Basic Posing

As previously noted, bones' transformations are performed based on the *rest position* of the armature, which is its state as defined in *Edit* mode. This means that in rest position, in *Pose* mode, each bone has a scale of **1.0**, and null rotation and position (as you can see it in the *Transform Properties* panel, in the 3D views, N).

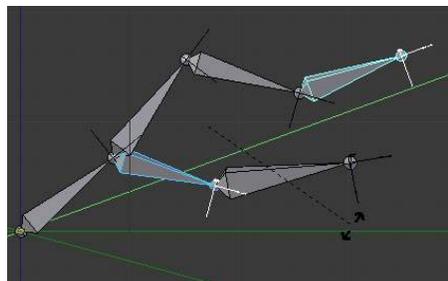


Fig. 2.1150: An example of locally-Y-axis locked rotation, with two bones selected. Note that the two green lines materializing the axes are centered on the armature's center, and not each bone's root...

Moreover, the local space for these actions is the bone's own one (visible when you enable the *Axes* option of the *Armature* panel). This is especially important when using axis locking - for example, there is no specific "bone roll" tool in *Pose* mode,

as you can rotate around the bone's main axis just by locking on the local Y axis (R-Y-Y)... This also works with several bones selected; each one is locked to its own local axis!

When you pose your armature, you are supposed to have one or more objects skinned on it! And obviously, when you transform a bone in *Pose* mode, its related objects or object's shape is moved/deformed accordingly, in real time. Unfortunately, if you have a complex rig set-up and/or a heavy skin object, this might produce lag, and make interactive editing very painful. If you experience such troubles, try enabling the *Delay Deform* button of the *Armature* panel - the skin objects will only be updated once you validate the transform operation.

## Auto IK

The auto IK option in the tool shelf enables a temporary ik constraint when posing bones. The chain acts from the tip of the selected bone to root of the uppermost parent bone. Note that this mode lacks options, and only works by applying the resulting transform to the bones in the chain.

## Rest Pose

Once you have transformed some bones, if you want to return to their rest position, just clear their transformations (usual Alt-G / Alt-R / Alt-S shortcuts, or *Pose* → *Clear Transform* → *Clear User Transform*, W-5, to clear everything at once... - commands also available in the *Pose* → *Clear Transform* sub-menu).

Note that in *Envelope* visualization, Alt-S does not clear the scale, but rather scales the *Distance* influence area of the selected bones (also available through the *Pose* → *Scale Envelope Distance* menu entry - only effective in *Envelope* visualization, even though it is always available...).

Conversely, you may define the current pose as the new rest position (i.e. “apply” current transformations to the *Edit* mode), using the *Pose* → *Apply Pose as Restpose* menu entry (or Ctrl-A and confirm the pop-up dialog). **When you do so, the skinned objects/geometry is also reset to its default, undeformed state**, which generally means you'll have to skin it again.

Whereas in *Edit* mode, you always see your armature in its rest position, in *Object* and *Pose* ones, you see it by default in its *pose position* (i.e. as it was transformed in the *Pose* mode). If you want to see it in the rest position in all modes, enable the *Rest Position* button in the *Armature* panel (*Editing* context).

## In-Betweens

There are several tools for editing poses in an animation.

**Relax Pose** (*Pose* → *In-Betweens* → *Relax Pose* or **Alt-E**) Relax pose is somewhat related to the above topic - but it is only useful with keyframed bones (see the [animation chapter](#)). When you edit such a bone (and hence take it “away” from its “keyed position”), using this command will progressively “bring it back” to its “keyed position”, with smaller and smaller steps as it comes near it.

**Push Pose** (*Pose* → *In-Betweens* → *Relax Pose* or **Ctrl-E**) Push pose exaggerates the current pose.

**Breakdownner** (*Pose* → *In-Betweens* → *Pose Breakdownner* or **Shift-E**) Creates a suitable breakdown pose on the current frame

There are also in *Pose* mode a bunch of armature-specific editing options/tools, like `auto-bones` naming, properties switching/enabling/disabling, etc., that we already described in the armature editing pages - follow the links above...

## Copy/Paste Pose

---

## Reference

Mode: *Pose* mode

Panel: *3D View* header

Menu: *Pose* → *Copy Current Pose*, *Pose* → *Paste Pose*, *Pose* → *Paste Flipped Pose*



Fig. 2.1151: Copy and paste pose buttons in the 3D View header in Pose mode.

Blender allows you to copy and paste a pose, either through the *Pose* menu, or directly using the three “copy/paste” buttons found at the right part of the 3D views header:

***Pose* → *Copy Current Pose*** to copy the current pose of selected bones into the pose buffer.

***Pose* → *Paste Pose*** paste the buffered pose to the currently posed armature.

***Pose* → *Paste Flipped Pose*** paste the **X axis mirrored** buffered pose to the currently posed armature.

Here are important points:

- This tool works at the Blender session level, which means you can use it across armatures, scenes, and even files. However, the pose buffer is not saved, so you lose it when you close Blender.
- There is only one pose buffer.
- Only the selected bones are taken into account during copying (i.e. you copy only selected bones’ pose).
- During pasting, on the other hand, bone selection has no importance. The copied pose is applied on a per-name basis (i.e. if you had a `forearm` bone selected when you copied the pose, the `forearm` bone of the current posed armature will get its pose when you paste it - and if there is no such named bone, nothing will happen...).
- What is copied and pasted is in fact the position/rotation/scale of each bone, in its own space. This means that the resulting pasted pose might be very different from the originally copied one, depending on: - The rest position of the bones, and - The current pose of their parents.

Examples of pose copy/paste.			

## Effects of Bones Relationships

Bones relationships are crucial in *Pose* mode - they have important effects on transformations behavior.

By default, children bones inherit:

- Their parent position, with their own offset of course.
- Their parent rotation (i.e. they keep a constant rotation relatively to their parent).
- Their parent scale, here again with their own offset.

Table  
2.44:  
Scaling  
of a root  
bone.

--	--	--

Exactly like standard children objects. You can modify this behavior on a per-bone basis, using their sub-panels in the *Armature Bones* panel:

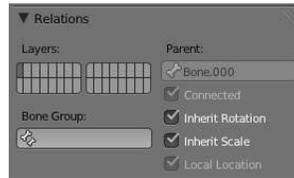


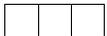
Fig. 2.1174: The Armature Bones panel in Pose mode.

**Inherit Rotation** When disabled, this will “break” the rotation relationship to the bone’s parent. This means that the child will keep its rotation in the armature object space when its parent is rotated.

**Inherit Scale** When disabled, this will “break” the scale relationship to the bone’s parent.

These inheriting behaviors propagate along the bones’ hierarchy. So when you scale down a bone, all its descendants are by default scaled down accordingly. However, if you set one bone’s *Inherit Scale* or *Inherit Rotation* property on in this “family”, this will break the scaling propagation, i.e. this bone *and all its descendants* will no longer be affected when you scale one of its ancestors.

Table  
2.45:  
Scaling  
of a bone  
with a  
Inherit  
Rotation  
disabled  
bone  
among its  
descen-  
dants.



Connected bones have another specificity: they cannot be translated. Indeed, as their root must be at their parent’s tip, if you don’t move the parent, you cannot move the child’s root, but only its tip - which leads us to a child rotation. This is exactly what happens - when you press G with a connected bone selected, Blender automatically switches to rotation operation.

Bones relationships also have important consequences on how selections of multiple bones behave when transformed. There are many different situations, so I’m not sure I list all possible ones below - but this should anyway give you a good idea of the problem:

- Non-related selected bones are transformed independently, as usual.

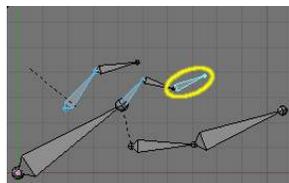


Fig. 2.1181: Scaling bones, some of them related.

- When several bones of the same “family” are selected, *only the “most parent” ones are really transformed* - the descendants are just handled through the parent relationship process, as if they were not selected (see *Scaling bones, some of*

*them related* - the third tip bone, outlined in yellow, was only scaled down through the parent relationship, exactly as the unselected ones, even though it is selected and active. Otherwise, it should have been twice smaller!).

- When connected and unconnected bones are selected, and you start a grab operation, only the unconnected bones are affected.
- When a child connected hinge bone is in the selection, and the “most parent” selected one is connected, when you press G, nothing happens - Blender remains in grab operation, which of course has no effect on a connected bone. This might be a bug, in fact, as I see no reason for this behavior...

So, when posing a chain of bones, you should always edit its elements from the root bone to the tip bone. This process is known as **forward kinematics**, or FK. We will see in a [later page](#) that Blender features another pose method, called **inverse kinematics**, or IK, which allows you to pose a whole chain just by moving its tip.

Note that this feature is somewhat extended/completed by the [pose library](#) tool.

## 2.5.20 Pose Library

### Intro

The *Pose Library* panel is used to save, apply, and manage different armature poses.

*Pose Libraries* are saved to *Actions*. They are not generally used as actions, but can be converted to and from.

### Pose Library Panel

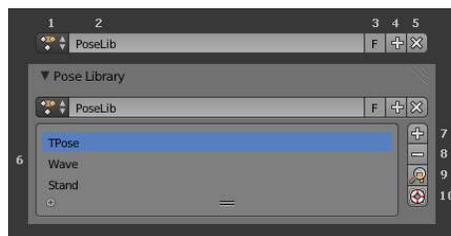


Fig. 2.1182: Properties > Armature > Pose Library.

1. Browse *Action / Pose Library* to be linked.
2. Name of the *Pose Library*.
3. Set Fake User. This will make blender save the *Pose Library* for if it has no users.
4. Add new *Pose Library* to the active object.
5. Remove the *Pose Library* from the active object.
6. A list of *Poses* for the active *Pose Library*.
7. Add Pose.

**Add New.** Add a new *Pose* to the active *Pose Library* with the current pose of the armature.

**Add New (Current Frame).** *Add New* and *Replace Existing* automatically allocate a *Pose* to an *Action* frame. *Add New (Current Frame)* will add a *Pose* to the *Pose Library* based on the current frame of the *Time Cursor*. Its not a well supported feature.

**Replace Existing.** Replace an existing *Pose* in the active *Pose Library* with the current pose of the armature.

8. Remove the active *Pose* from the *Pose Library*.

9. Apply the active *Pose* to the selected *Pose Bones*.
10. Sanitize Action. Make *Action* suitable for use as a *Pose Library*. This is used to convert an *Action* to a *Pose Library*. A *Pose* is added to the *Pose Library* for each frame with keyframes.

## Editing

3D View, Pose Mode.

Browse Poses. `Ctrl-L`.

Add Pose. `Shift-L`.

Rename Pose. `Shift-Ctrl-L`.

Remove Pose. `Alt-L`.

### 2.5.21 Applying Constraints to Bones

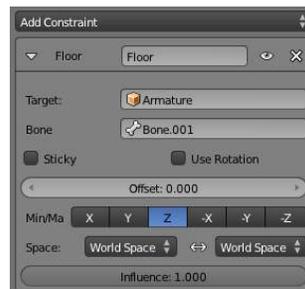


Fig. 2.1183: The Constraints panel in Pose mode, with one Floor constraint applied to the active bone (Bone.001).

As bones behave like objects in *Pose* mode, they can also be constrained. This is why the *Constraints* panel is shown in both *Object* and *Editing* contexts in this mode... This panel contains the constraints *of the active bone* (its name is displayed at the top of the panel, in the *To Bone:...* static text field).

Constraining bones can be used to control their degree of freedom in their pose transformations, using e.g. the *Limit* constraints. You can also use constraints to make a bone track another object/bone (inside the same object, or in another armature), etc. And the [inverse kinematics feature](#) is also mainly available through the *IK Solver* constraint - which is specific to bones.

For example, a human elbow can't rotate backward (unless the character has broken his hand), nor to the sides, and its forward and roll rotations are limited in a given range (for example, depending on the rest position of your elbow, it may be from 0 to 160, OR from -45 to 135).

So you should apply a *Limit Rotation* constraint to the forearm bone (as the elbow movement is the result of rotating the forearm bone around its root).

Using bones in constraints, either as owners or as targets, is discussed in detail in the [constraints pages](#).

### 2.5.22 Inverse Kinematics

IK simplifies the animation process, and makes it possible to make more advanced animations with lesser effort.

IK allows you to position the last bone in a bone chain and the other bones are positioned automatically. This is like how moving someone's finger would cause his arm to follow it. By normal posing techniques, you would have to start from the root bone, and set bones sequentially till you reach the tip bone: When each parent bone is moved, its child bone would inherit its

location and rotation. Thus making tiny precise changes in poses becomes harder farther down the chain, as you may have to adjust all the parent bones first.

This effort is effectively avoided by use of IK.

### Automatic IK

Automatic IK is a tool for quick posing, it can be enabled in the tool shelf in the 3D view, when in pose mode. When the Auto IK option is enabled, translating a bone will activate inverse kinematic and rotate bones higher up to follow the selected bone. By default, the length of the IK chain is as long as there are parent bones, and this length can be modified with `Shift PageUp`, `Shift PageDown`, or `Shift WheelUp`, `Shift WheelDown`.

This is a more limited feature than using an IK constraint, which can be configured, but it can be useful for quick posing.

### IK Constraints

IK is mostly done with bone constraints. They work by the same method but offer more choices and settings. Please refer to these pages for detail about the settings for the constraints:

- [IK Solver](#)
- [Spline IK](#)

### Armature IK Panel

This panel is used to select the IK Solver type for the armature. *Standard* or *iTaSC*.

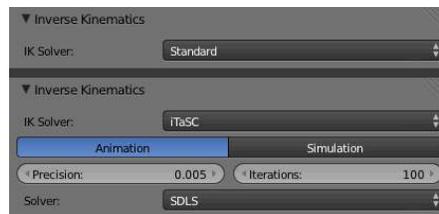


Fig. 2.1184: Properties > Armature > Inverse Kinematics Panel.

Most the time people will use the *Standard* IK solver. There is some documentation for the *iTaSC* “instantaneous Task Specification using Constraints” IK solver [here](#).

[Robot IK Solver](#)

### Bone IK Panel

This panel is used to control how the *Pose Bones* work in the IK chain.

**Lock** Disallow movement around the axis.

**Stiffness** Stiffness around the axis. Influence disabled if using *Lock*.

**Limit** Limit movement around the axis, specified by the sliders.

**Stretch** Stretch influence to IK target. 0.000 is the same as disabled.

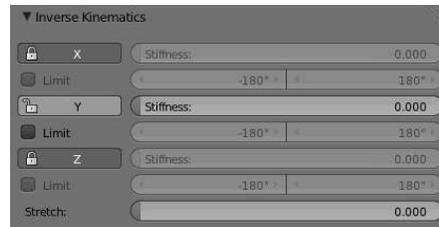


Fig. 2.1185: Properties > Bone > Inverse Kinematics Panel.

## Arm Rig Example

This arm uses two bones to overcome the twist problem for the forearm. IK locking is used to stop the forearm from bending, but the forearm can still be twisted manually by pressing R-Y-Y in *Pose Mode*, or by using other constraints.

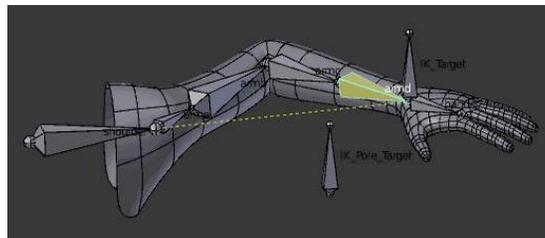


Fig. 2.1186: IK Arm Example.

### IK Arm Example.

Note that, if a *Pole Target* is used, IK locking will not work on the root bone.

## 2.5.23 Spline IK

Spline IK is a constraint which aligns a chain of bones along a curve. By leveraging the ease and flexibility of achieving aesthetically pleasing shapes offered by curves and the predictability and well integrated control offered by bones, Spline IK is an invaluable tool in the riggers' toolbox. It is particularly well suited for rigging flexible body parts such as tails, tentacles, and spines, as well as inorganic items such as ropes.

Full description of the settings for the spline IK are detailed on the [Spline IK](#) page.

### Basic Setup

The Spline IK Constraint is not strictly an 'Inverse Kinematics' method (i.e. IK Constraint), but rather a 'Forward Kinematics' method (i.e. normal bone posing). However, it still shares some characteristics of the IK Constraint, such as operating on multiple bones not being usable for Objects, and being evaluated after all other constraints have been evaluated. It should be noted that if a Standard IK chain and a Spline IK chain both affect a bone at the same time the Standard IK chain takes priority. Such setups are best avoided though, since the results may be difficult to control.

To setup Spline IK, it is necessary to have a chain of connected bones and a curve to constrain these bones to.

- With the last bone in the chain selected, add a [Spline IK](#) Constraint from the Bone Constraints tab in the Properties Editor.
- Set the 'Chain Length' setting to the number of bones in the chain (starting from and including the selected bone) that should be influenced by the curve.
- Finally, set the 'Target' field to the curve that should control the curve.

Congratulations, the bone chain is now controlled by the curve.

## Settings and Controls

### Roll Control

To control the ‘twist’ or ‘roll’ of the Spline IK chain, the standard methods of rotating the bones in the chain along their y-axes still apply. For example, simply rotate the bones in the chain around their y-axes to adjust the roll of the chain from that point onwards. Applying copy rotation constraints on the bones should also work.

### Offset Controls

The entire bone chain can be made to follow the shape of the curve while still being able to be placed at an arbitrary point in 3D-space when the ‘Chain Offset’ option is enabled. By default, this option is not enabled, and the bones will be made to follow the curve in its untransformed position.

### Thickness Controls

The thickness of the bones in the chain is controlled using the constraint’s ‘XZ Scale Mode’ setting. This setting determines the method used for determining the scaling on the X and Z axes of each bone in the chain.

The available modes are:

**None** this option keeps the X and Z scaling factors as 1 . 0

**Volume Preserve** the X and Z scaling factors are taken as the inverse of the Y scaling factor (length of the bone), maintaining the ‘volume’ of the bone

**Bone Original** this options just uses the X and Z scaling factors the bone would have after being evaluated in the standard way.

In addition to these modes, there is an option, ‘Use Curve Radius’. When this option is enabled, the average radius of the radii of the points on the curve where the endpoints of each bone are placed, are used to derive X and Z scaling factors. This allows the scaling effects, determined using the modes above, to be tweaked as necessary for artistic control.

### Tips for Nice Setups

- For optimal deformations, it is recommended that the bones are roughly the same length, and that they are not too long, to facilitate a better fit to the curve. Also, bones should ideally be created in a way that follows the shape of the curve in its ‘rest pose’ shape, to minimise the problems in areas where the curve has sharp bends which may be especially noticeable when stretching is disabled.
- For control of the curve, it is recommended that hooks (in particular, Bone Hooks, which are new in 2.5) are used to control the control-verts of the curve, with one hook per control-vert. In general, only a few control-verts should be needed for the curve (i.e. 1 for every 3-5 bones offers decent control).
- The type of curve used does not really matter, as long as a path can be extracted from it that could also be used by the Follow Path Constraint. This really depends on the level of control required from the hooks.
- When setting up the rigs, it is currently necessary to have the control bones (for controlling the curve) in a separate armature to those used for deforming the meshes (i.e. the deform rig containing the Spline IK chains). This is to avoid creating pseudo “Dependency Cycles”, since Blender’s Dependency Graph can only resolve the dependencies the control bones, curves, and Spline IK’ed bones on an object by object basis.

## 2.5.24 Constraints

### Introduction

Constraints are a way of connecting *transform properties* (position, rotation and scale) between objects. Constraints are in a way the object counterpart of the [modifiers](#), which work on the object *data* (i.e. meshes, curves, etc.).

All constraints share a basic [common interface](#), again with many similarities with the modifiers' one.

### Use of Constraints

Even though constraints might be very useful in static scenes (as they can help to automatically position/rotate/scale objects), they were first designed for animation, as they allow you to limit/control the freedom of an object, either in absolute (i.e. in global space), or relatively to other objects.

Also note that constraints internally work using 4x4 transformation matrices only. When you use settings for specific rotation or scaling constraining, this information is being derived from the matrix only, not from settings in a *Bone* or *Object*. Especially for combining rotations with non-uniform or negative scaling this can lead to unpredictable behavior.

**Constraining bones** Finally, there is a great rigging feature in Blender: in *Pose* mode, each bone of an armature behaves a bit like a standard object, and, as such, can be constrained. Most constraints work well with both objects and bones, but there are a few exceptions which are noted in the relevant constraints pages.

To learn more:

- Read about using constraints in object animation in the [Animation chapter](#)
- Read about using constraints in rigging in the [Armatures](#)

### Available Constraints

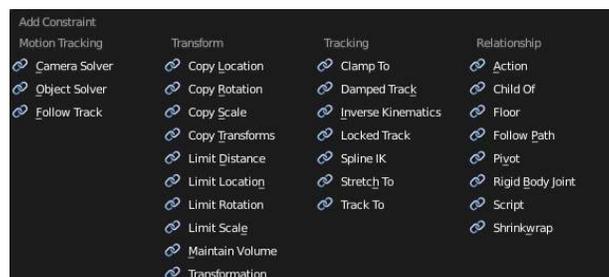


Fig. 2.1187: The Constraint Menu

There are several types of constraints. We can classify them into four families:

- Motion Tracking
- Transform
- Tracking
- Relationship

There are constraints that works with their *owner* object and others that need a second object (the *target*) to work, sometimes of a specific type (e.g. a curve). In this case targeted constraints are shown as a dark blue dashed line drawn in the 3D view between the owner and target objects.

**Motion Tracking**

- Camera Solver
- Object Solver
- Follow Track

**Transform Constraints** These constraints directly control/limit the transform properties of its owner, either absolutely or relatively in terms of its target properties.

**Copy Location** Copies the location of the target (with an optional offset) to the owner, so that both move together.

**Copy Rotation** Copies the rotation of the target (with an optional offset) to the owner, so that both rotate together.

**Copy Scale** Copies the scale of the target (with an optional offset) to the owner, so that both scale together.

**Copy Transforms** Copies the transforms of the target to the owner, so that both transform together.

**Limit Distance** Limits the position of the owner, so that it is nearer/further/exactly at the specified distance from the target.

**Limit Location** Limits the owner's location inside a given range.

**Limit Rotation** Limits the owner's rotation inside a given range.

**Limit Scale** Limits the owner's scale inside a given range.

**Transformation** Uses a property of the target (location, rotation or scale), to control a property (the same or a different one) of the owner.

**Maintain Volume** Maintains the volume of a bone or an object.

**Tracking Constraints** These constraints try, in various ways, to adjust their owner's properties so that it "points at" or "follows" the target.

**Clamp To** Clamps the owner to a given curve target.

**Damped Track** Constrains one local axis of the owner to always point towards Target.

**Inverse Kinematics** Bones only. Creates a chain of bones controlled by the target, using inverse kinematics.

**Locked Track** The owner is tracked to the given target, but with a given axis' orientation locked.

**Spline IK** Aligns a chain of bones along a curve.

**Stretch To** Stretch the owner to the given target.

**Track To** The owner is tracked to the given target.

**Relationship Constraints** These are "misc" constraints.

**Action** The owner executes an action, controlled by the target (driver).

**Child Of** Allows a selective application of the effects of parenting to another object.

**Floor** Uses the target's position (and optionally rotation) to define a "wall" or "floor" that the owner won't be able to cross.

**Follow Path** The owner moves along the curve target.

**Pivot** Allows the owner to rotate around a target object.

**Rigid Body Joint** Creates a rigid joint (like a hinge) between the owner and the "target" (child object).

**Script** Uses a Python script as constraint.

**Shrinkwrap** Limits the location of the owner at *the surface* (among other options) of the target.

## Constraints Common Interface

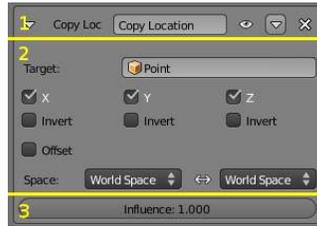


Fig. 2.1188: The three parts of a constraint interface

As with [modifiers](#), an object (or bone, see the [rigging chapter](#) for details) can use several constraints at once. Hence, these constraints are organized in a stack which controls their order of evaluation (from top to bottom).

All constraints share a common basic interface, packed up in a sort of sub-panel, that is split into three parts:

- The header, gathering most common settings.
- The constraint’s specific settings.
- The influence and animation controls (the *Rigid Body Joint* constraints have no influence setting).

### Constraints Header



Fig. 2.1189: A constraint header

The header of a constraint “sub-panel” is the same for all. From left to right, you have:

**A small arrow** This control allows you to show/hide the constraint’s settings.

**The constraint type** This is just static text showing you what this constraint is...

**The name field** Here you can give your constraint a more meaningful name than the default one. This control has another *important* purpose: it turns red when the constraint is not functional (as in *A constraint header*). As most constraints need a second “target” object to work (see below), when just added, they are in “red state”, as Blender cannot guess which object or bone to use as target. This can also happen when you choose an invalid set of settings, e.g. with a [Track To constraint](#) of which the *To* and *Up* vectors are both set to the same axis. As noted above, constraints in “red state” are ignored during the stack evaluation.

**The “up”/“down” buttons** As seen above, these allow you to move a constraint up/down in the stack.

**The “X” control** As seen above, this will delete (remove from the stack) the constraint.

### Constraints Settings



Fig. 2.1190: The central part of a constraint’s subpanel contains the constraint’s settings, the target, and constraint space

The constraints settings area is of course specific to each constraint type. However, there are two points that are common to many constraints, so we will detail them here.

**The target** Most constraints need another “target” object or bone to “guide” them. You select which by selecting its name in the *Target* field. Except for a few cases, you can use any type of object (camera, mesh, empty...); its object origin will be the target point.

When you type in the *OB* field a mesh or lattice name, a second *Vertex Group* field appears just below. If you leave it empty, the mesh or lattice will be used as a standard object target. But if you enter in this *Vertex Group* field the name of one of the mesh’s or lattice’s vertex groups, then the constraint will use the median point of this vertex group as target.

Similarly, if you type in the *OB* field an armature name, a second *Bone* field appears just below. If you enter in it the name of one of the armature’s bones, then the constraint will use this bone’s *root* as target. In some constraints, when you use a bone as target, another *Head/Tail* numeric field will also appear, that allows you to select where along the bone the target point will lay, from root (**0.0**) to tip (**1.0**) (remember that currently, in Blender UI, bones’ roots are called “heads”, and bones’ tips, “tails”...).

**The Constraint Space (Space)** For many constraints you can choose in which space it is evaluated/applied. In the Space drop-down lists, the right side one is the space that the owner is evaluated in (Owner Space). When such a constraint uses a target, you can also choose in which space the target is evaluated (Target Space). The Target Space drop-down list is on the left side. Both lists have the same options, depending on whether the element (owner or target) is a regular object, or a bone:

**Local Space** The object’s properties are evaluated in its own local space, i.e. based on its rest position (without taking into account its parents transformations in its chain, or its armature object’s transformation).

**Local With Parent (bones only)** The bone properties are evaluated in its own local space, *including* the transformations due to a possible parent relationship (i.e. due to the chain’s transformations above the bone).

**Pose Space (bones only)** The bone properties are evaluated in the armature object local space (i.e. independently from the armature transformations in *Object* mode). Hence, if the armature object has null transformations, *Pose Space* will have the same effect as *World Space*.

**Local (Without Parent) Space (objects only)** The object properties are evaluated in its own local space, *without* the transformations due to a possible parent relationship.

**World Space (default setting)** Here the object’s or bone’s properties are evaluated in the global coordinate system. This is the easiest to understand and most natural behavior, as it always uses the “visual” transform properties (i.e. as you see them in the 3D views).

Understanding the Constraint Space effects is not really easy (unless you are a geometry genius...). The best thing to do is to experiment with their different combinations, using e.g. two empties (as they materialize clearly their axes), and a *Copy Rotation* constraint (as rotations are the most demonstrative transformations, to visualize the various spaces specificities...).

## Influence



Fig. 2.1191: Influence

At the bottom of nearly all constraints, you have the *Influence* slider, which controls the influence of the constraint on its owner. As you might expect, **0.0** means that the constraint has no effect, and **1.0** means that the constraint has full effect. Using in-between values, you can have several constraints all working together on the same owner’s properties. Note that if a constraint has a full influence on a given property, all other constraints above in the stack working on that same property will have no effect at all.

But the best thing with influence is that you can animate it with an Fcurve - see [the constraints page of the animation chapter](#) for more details about this.

## The Constraints Stack

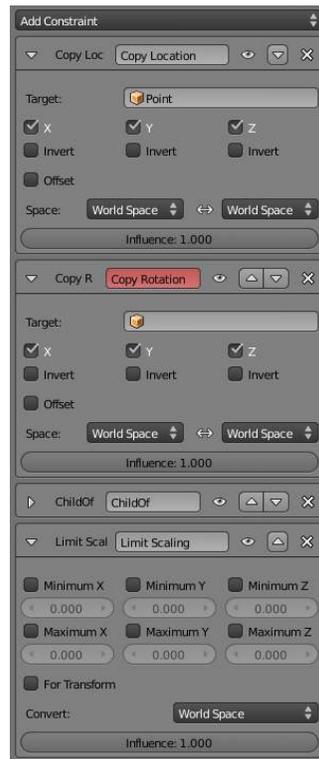


Fig. 2.1192: A constraint stack example

Constraints are evaluated from top to bottom of the constraint stack, shown in the *Constraints* panel.

- In (*A constraint stack example*), first the location of the lamp is copied to the owner object.
- The copy rotation constraint is ignored (red name, see below).
- So the next constraint evaluated is the *Child Of* one, which is currently reduced.
- Finally, the size of our cube is bounded by the *Limit Scale* last constraint.

So here, the size of the cube is first controlled by the target of the *Child Of* constraint, within the limits allowed by the next *Limit Scale* constraint... As with modifiers, order is crucial!

You can move a constraint up and down the stack by using the small up/down arrow buttons that are drawn in its header, to the right of the constraint name. These buttons are only visible when needed, i.e. the top constraint has only the “down” button, the bottom constraint, only the “up” one - and when there is only one constraint in the stack, both buttons are hidden.

### Adding/Removing a Constraint

To add a constraint, you can, in the *Constraints* panel, click on the... *Add Constraint* button! A menu shows up, listing all available constraints for the current active object (or bone in *Pose* mode (in which case the constraint will show up in the bone constraints menu)). The new constraint is *always* added at the bottom of the stack.

You can also, in a 3D view, either:

- Select only the future owner, press **Ctrl-Shift-C**, and in the *Add Constraint to New Empty Object* menu that pops up, select the constraint you want to add. If the chosen constraint needs it, a new *Empty* object will be automatically added as target, positioned at the owner’s center, and with null rotation.

- Select first the future target, and then the future owner, press `Ctrl-Shift-C`, and in the *Add Constraint to Active Object* (or *Add Constraint to Active Bone*) menu that pops up, select the constraint you want to add. If the chosen constraint needs it, the other selected object/bone will be used as target.

Note that these pop-up menus do not display all the available constraints.

To remove a constraint, click on the “X” button of the header of the constraint you want to delete, in the *Constraints* panel. You can also remove all constraints from the selected object(s), using the *Object* → *Constraints* → *Clear Object Constraints* (or *Pose* → *Constraints* → *Clear Pose Constraints...* or press `Ctrl-Alt-C`).

## Copy Location Constraint

### Description

The *Copy Location* constraint forces its owner to have the same location as its target.

**Warning:** Note that if you use such a constraint on a *connected* bone, it will have no effect, as it is the parent’s tip which controls the position of your owner bone’s root.

### Options



Fig. 2.1193: Copy Location panel

**Target** This constraint uses one target, and is not functional (red state) when it has none.

**Bone** If *Target* is an *Armature*, a new field is displayed offering the optional choice to set an individual bone as *Target*.

**Head/Tail** If a *Bone* is set as *Target*, a new field is displayed offering the optional choice of where along this bone the target point lies.

**Vertex Group** If *Target* is a *Mesh*, a new field is displayed offering the optional choice to set a *Vertex Group* as target.

**X, Y, Z** These buttons control which axes (i.e. coordinates) are constrained - by default, all three ones are.

**Invert** The *Invert* buttons invert their respective preceding coordinates.

**Offset** When enabled, this control allows the owner to be translated (using its current transform properties), relative to its target’s position.

**Space** This constraint allows you to choose in which space to evaluate its owner’s and target’s transform properties.

## Copy Rotation Constraint

The *Copy Rotation* constraint forces its owner to match the rotation of its target.

## Options

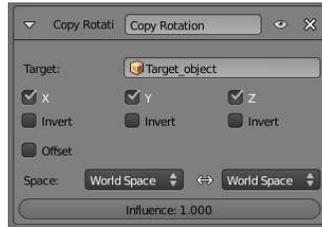


Fig. 2.1194: Copy Rotation panel

**Target** This constraint uses one target, and is not functional (red state) when it has none.

**Bone** If *Target* is an *Armature*, a new field is displayed offering the optional choice to set an individual bone as *Target*.

**Head/Tail** If a *Bone* is set as *Target*, a new field is displayed offering the optional choice of where along this bone the target point lies.

**Vertex Group** If *Target* is a *Mesh*, a new field is displayed offering the optional choice to set a *Vertex Group* as target.

**X, Y, Z** These buttons control which axes are constrained - by default, all three are on.

**Invert** The *Invert* buttons invert their respective rotation values.

**Offset** When enabled, this control allows the owner to be rotated (using its current transform properties), relative to its target's orientation.

**Space** This constraint allows you to choose in which space to evaluate its owner's and target's transform properties.

## Copy Scale Constraint

### Description

The *Copy Scale* constraint forces its owner to have the same scale as its target.

**Warning:** Here we talk of **scale**, not of **size**! Indeed, you can have two objects, one much bigger than the other, and yet both of them have the same scale. This is also true with bones: in *Pose* mode, they all have a unitary scale when they are in rest position, represented by their visible length.

## Options



Fig. 2.1195: Copy Scale panel

**Target** This constraint uses one target, and is not functional (red state) when it has none.

**Bone** If *Target* is an *Armature*, a new field is displayed offering the optional choice to set an individual bone as *Target*.

**Head/Tail** If a *Bone* is set as *Target*, a new field is displayed offering the optional choice of where along this bone the target point lies.

**Vertex Group** If *Target* is a *Mesh*, a new field is displayed offering the optional choice to set a *Vertex Group* as target.

**X, Y, Z** These buttons control along which axes the scale is constrained - by default, it is enabled along all three.

**Offset** When enabled, this control allows the owner to be scaled (using its current transform properties), relatively to its target's scale.

**Space** This constraint allows you to choose in which space to evaluate its owner's and target's transform properties.

## Copy Transforms Constraint

### Description

The *Copy Transforms* constraint forces its owner to have the same transforms as its target.

### Options



Fig. 2.1196: Copy Transforms panel

**Target** This constraint uses one target, and is not functional (red state) when it has none.

**Bone** If *Target* is an *Armature*, a new field is displayed offering the optional choice to set an individual bone as *Target*.

**Head/Tail** If a *Bone* is set as *Target*, a new field is displayed offering the optional choice of where along this bone the target point lies.

**Vertex Group** If *Target* is a *Mesh*, a new field is displayed offering the optional choice to set a *Vertex Group* as target.

**Space** This constraint allows you to choose in which space to evaluate its owner's and target's transform properties.

## Limit Distance Constraint

### Description

The *Limit Distance* constraint forces its owner to stay either further from, nearer to, or exactly at a given distance from its target. In other words, the owner's location is constrained either outside, inside, or at the surface of a sphere centered on its target.

When you specify a (new) target, the *Distance* value is automatically set to correspond to the distance between the owner and this target.

**Warning:** Note that if you use such a constraint on a *connected* bone, it will have no effect, as it is the parent's tip which controls the position of your owner bone's root.

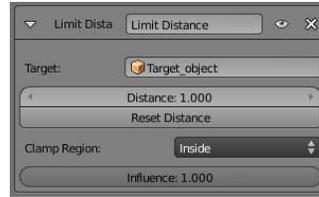


Fig. 2.1197: Limit Distance panel

## Options

**Target** This constraint uses one target, and is not functional (red state) when it has none.

**Bone** If *Target* is an *Armature*, a new field is displayed offering the optional choice to set an individual bone as *Target*.

**Head/Tail** If a *Bone* is set as *Target*, a new field is displayed offering the optional choice of where along this bone the target point lies.

**Vertex Group** If *Target* is a *Mesh*, a new field is displayed offering the optional choice to set a *Vertex Group* as target.

**Distance** This numeric field sets the limit distance, i.e. the radius of the constraining sphere.

**Reset Distance** When clicked, this small button will reset the *Distance* value, so that it corresponds to the actual distance between the owner and its target (i.e. the distance before this constraint is applied).

**Clamp Region** The *Limit Mode* drop-down menu allows you to choose how to use the sphere defined by the *Distance* setting and target's center:

**Inside (default)** The owner is constrained *inside* the sphere.

**Outside** The owner is constrained *outside* the sphere.

**Surface** The owner is constrained *on the surface* of the sphere.

## Limit Location Constraint

### Description

An object or *unconnected* bone can be moved around the scene along the X, Y and Z axes. This constraint restricts the amount of allowed translations along each axis, through lower and upper bounds.

The limits for an object are calculated from its center, and the limits of a bone, from its root.

It is interesting to note that even though the constraint limits the visual and rendered location of its owner, its owner's data block still allows (by default) the object or bone to have coordinates outside the minimum and maximum ranges. This can be seen in its *Transform Properties* panel (N). When an owner is grabbed and attempted to be moved outside the limit boundaries, it will be constrained to those boundaries visually and when rendered, but internally, its coordinates will still be changed beyond the limits. If the constraint is removed, its ex-owner will seem to jump to its internally specified location.

Similarly, if its owner has an internal location that is beyond the limits, dragging it back into the limit area will appear to do nothing until the internal coordinates are back within the limit threshold (unless you enabled the *For Transform* option, see below).

Setting equal the min and max values of an axis, locks the owner's movement along that axis... Although this is possible, using the *Transformation Properties* axis locking feature is probably easier!



Fig. 2.1198: Limit Location panel

## Options

**Minimum X, Minimum Y, Minimum Z** These buttons enable the lower boundary for the location of the owner's center along, respectively, the X, Y and Z axes of the chosen *Space*. The numeric field below them controls the value of their limit. Note that if a min value is higher than its corresponding max value, the constraint behaves as if it had the same value as the max one.

**Maximum X, Maximum Y, Maximum Z** These buttons enable the upper boundary for the location of the owner's center along, respectively, the X, Y and Z axes of the chosen *Space*. Same options as above.

**For Transform** We saw that by default, even though visually constrained, the owner can still have coordinates out of bounds (as shown by the *Transform Properties* panel). Well, when you enable this button, this is no longer possible - the owner's transform properties are also limited by the constraint. Note however that the constraint does not directly modify the coordinates: you have to grab its owner one way or another for this to take effect...

**Convert** This constraint allows you to choose in which space to evaluate its owner's transform properties.

## Limit Rotation Constraint

### Description

An object or bone can be rotated around the X, Y and Z axes. This constraint restricts the amount of allowed rotations around each axis, through lower and upper bounds.

It is interesting to note that even though the constraint limits the visual and rendered rotations of its owner, its owner's data block still allows (by default) the object or bone to have rotation values outside the minimum and maximum ranges. This can be seen in the *Transform Properties* panel (N). When an owner is rotated and attempted to be rotated outside the limit boundaries, it will be constrained to those boundaries visually and when rendered, but internally, its rotation values will still be changed beyond the limits. If the constraint is removed, its ex-owner will seem to jump to its internally specified rotation.

Similarly, if its owner has an internal rotation that is beyond the limit, rotating it back into the limit area will appear to do nothing until the internal rotation values are back within the limit threshold (unless you enabled the *For Transform* option, see below).

Setting equal the min and max values of an axis, locks the owner's rotation around that axis... Although this is possible, using the *Transformation Properties* axis locking feature is probably easier.

This transform does not constrain the bone if it is manipulated by the IK solver. For constraining the rotation of a bone for IK purposes, see the "Inverse Kinematics" section of Bone properties.

## Options

**Limit X, LimitY, LimitZ** These buttons enable the rotation limit around respectively the X, Y and Z axes of the owner, in the chosen *Space*. The *Min* and *Max* numeric fields to their right control the value of their lower and upper boundaries,

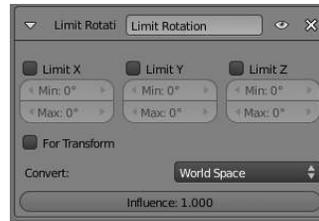


Fig. 2.1199: Limit Rotation panel

respectively.

Note that:

- If a min value is higher than its corresponding max value, the constraint behaves as if it had the same value as the max one.
- Unlike the [Limit Location constraint](#), you cannot enable separately lower or upper limits...

**For Transform** We saw that by default, even though visually constrained, the owner can still have rotations out of bounds (as shown by the *Transform Properties* panel). Well, when you enable this button, this is no more possible - the owner transform properties are also limited by the constraint. Note however that the constraint does not directly modifies the rotation values: you have to rotate one way or the other its owner, for this to take effect...

**Convert** This constraint allows you to chose in which space evaluate its owner's transform properties.

## Limit Scale Constraint

### Description

An object or bone can be scaled along the X, Y and Z axes. This constraint restricts the amount of allowed scalings along each axis, through lower and upper bounds.

**Warning:** This constraint does not tolerate negative scale values (those you might use to mirror an object...): when you add it to an object or bone, even if no axis limit is enabled, nor the *For Transform* button, as soon as you scale your object, all negative scale values are instantaneously inverted to positive ones... And the boundary settings can only take strictly positive values.

It is interesting to note that even though the constraint limits the visual and rendered scale of its owner, its owner's data block still allows (by default) the object or bone to have scale values outside the minimum and maximum ranges (as long as they remain positive!). This can be seen in its *Transform Properties* panel (N). When an owner is scaled and attempted to be moved outside the limit boundaries, it will be constrained to those boundaries visually and when rendered, but internally, its coordinates will still be changed beyond the limits. If the constraint is removed, its ex-owner will seem to jump to its internally-specified scale.

Similarly, if its owner has an internal scale that is beyond the limits, scaling it back into the limit area will appear to do nothing until the internal scale values are back within the limit threshold (unless you enabled the *For Transform* option, see below - or your owner has some negative scale values).

Setting equal the min and max values of an axis locks the owner's scaling along that axis. Although this is possible, using the *Transformation Properties* axis locking feature is probably easier.

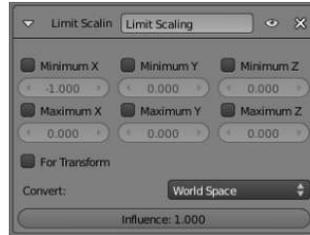


Fig. 2.1200: Limit Scale panel

## Options

**Minimum / Maximum X, Y, Z** These buttons enable the lower boundary for the scale of the owner along respectively the X, Y and Z axes of the chosen *Space*. The *Min* and *Max* numeric fields to their right control the value of their lower and upper boundaries, respectively. Note that if a min value is higher than its corresponding max value, the constraint behaves as if it had the same value as the max one.

**For Transform** We saw that by default, even though visually constrained, and except for the negative values, the owner can still have scales out of bounds (as shown by the *Transform Properties* panel). Well, when you enable this button, this is no longer possible - the owner transform properties are also limited by the constraint. Note however that the constraint does not directly modify the scale values: you have to scale its owner one way or another for this to take effect.

**Convert** This constraint allows you to choose in which space to evaluate its owner's transform properties.

## Maintain Volume Constraint

### Description

The *Maintain Volume* constraint limits the volume of a mesh or a bone to a given ratio of its original volume.

### Option



Fig. 2.1201: Maintain Volume panel

**Free X / Y / Z** The free-scaling axis of the object.

**Volume** The bone's rest volume. Default is 1.0.

**Space** This constraint allows you to choose in which space to evaluate its owner's transform properties.

### See also

- [Harkyman on the development of the Maintain Volume constraint, March 2010](#)

## Transformation Constraint

This constraint is more complex and versatile than the other “transform” constraints. It allows you to map one type of transform properties (i.e. location, rotation or scale) of the target, to the same or another type of transform properties of the owner, within a given range of values (which might be different for each target and owner property). You can also switch between axes, and use the range values not as limits, but rather as “markers” to define a mapping between input (target) and output (owner) values.

So, e.g. you can use the position of the target along the X axis to control the rotation of the owner around the Z axis, stating that **1 BU** along the target X axis corresponds to

10 around the owner Z axis. Typical uses for this include gears (see note below),

and rotation based on location setups.

### Options

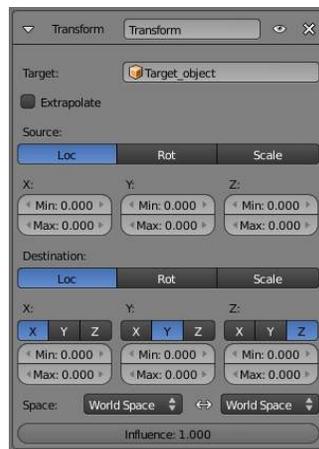


Fig. 2.1202: Transformation panel

**Target** This constraint uses one target, and is not functional (red state) when it has none.

**Bone** If *Target* is an *Armature*, a new field is displayed offering the optional choice to set an individual bone as *Target*.

**Head/Tail** If a *Bone* is set as *Target*, a new field is displayed offering the optional choice of where along this bone the target point lies.

**Vertex Group** If *Target* is a *Mesh*, a new field is displayed offering the optional choice to set a *Vertex Group* as target.

**Extrapolate** By default, the *min* and *max* values bound the input and output values; all values outside these ranges are clipped to them. When you enable this button, the *min* and *max* values are no longer strict limits, but rather “markers” defining a proportional (linear) mapping between input and corresponding output values. Let’s illustrate that with two graphs (*The Extrapolate principles*). In these pictures, the input range (in abscissa) is set to  $[1.0, 4.0]$ , and its corresponding output range (in ordinate), to  $[1.0, 2.0]$ . The yellow curve represents the mapping between input and output.

Table  
2.46:  
Extrapolate  
enabled:  
the output  
values are  
“free” to  
proportionally  
follow  
the input  
ones.



**Warning:** Note that:

- When mapping transform properties to location (i.e. *Loc*, *Destination* button is enabled), the owner’s existing location is added to the result of evaluating this constraint (exactly like when the *Offset* button of the [Copy Location](#) constraint is enabled...).
- Conversely, when mapping transform properties to rotation or scale, the owner’s existing rotation or scale is overridden by the result of evaluating this constraint.
- When using the rotation transform properties of the target as input, whatever the real values are, the constraint will always “take them back” into the  $-180, 180$  range (e.g. if the target has a rotation of 420 around its X axis, the values used as X input by the constraint will be  $((420 + 180) \bmod 360) - 180 = 60 - \dots$ ). This is why this constraint is not really suited for gears!
- Similarly, when using the scale transform properties of the target as input, whatever the real values are, the constraint will always take their absolute values (i.e. invert negative ones).
- When a *min* value is higher than its corresponding *max* one, both are considered equal to the *max* one. This implies you cannot create “reversed” mappings...

**Source** It contains the input (from target) settings. The three *Loc*, *Rot* and *Scale* toggle buttons, mutually exclusive, allow you to select which type of property to use. The *X*, *Y* and *Z*: *min* and *max* numeric fields control the lower and upper bounds of the input value range, independently for each axis. Note that if a min value is higher than its corresponding max value, the constraint behaves as if it had the same value as the max one.

**Destination** It contains the output (to owner) settings.

- The three *Loc*, *Rot* and *Scale* toggle buttons, mutually exclusive, allow you to select which type of property to control.
- The three *Axis Mapping* drop-down lists allow you to select which input axis to map to, respectively (from top to bottom), the X, Y and Z output (owner) axes.
- The *min* and *max* numeric fields control the lower and upper bounds of the output value range, independently for each mapped axis. Note that if a min value is higher than its corresponding max value, the constraint behaves as if it had the same value as the max one.

**Space** This constraint allows you to choose in which space to evaluate its owner’s and target’s transform properties.

## Clamp To Constraint

The *Clamp To* constraint clamps an object to a curve. The *Clamp To* constraint is very similar to the [Follow Path](#) constraint, but instead of using the evaluation time of the target curve, *Clamp To* will get the actual location properties of its owner (those shown in the *Transform Properties* panel, N), and judge where to put it by “mapping” this location along the target curve.

One benefit is that when you are working with *Clamp To*, it is easier to see what your owner will be doing; since you are

working in the 3D view, it will just be a lot more precise than sliding keys around on a time Ipo and playing the animation over and over.

A downside is that unlike in the [Follow Path constraint](#), *Clamp To* doesn't have any option to track your owner's rotation (pitch, roll, yaw) to the banking of the targeted curve, but you don't always need rotation on, so in cases like this it's usually a lot handier to fire up a *Clamp To*, and get the bits of rotation you do need some other way.

The mapping from the object's original position to its position on the curve is not perfect, but uses the following simplified algorithm (note, I am not the original code author so this may not be 100% accurate):

- A "main axis" is chosen, either by the user, or as the longest axis of the curve's bounding box (the default).
- The position of the object is compared to the bounding box of the curve in the direction of the main axis. So for example if X is the main axis, and the object is aligned with the curve bounding box's left side, the result is 0; if it is aligned with the right side, the result is 1.
- If the cyclic option is unchecked, this value is clamped in the range 0-1.
- This number is used as the curve time, to find the final position along the curve that the object is clamped to.

This algorithm does not produce exactly the desired result because curve time does not map exactly to the main axis position. For example an object directly in the centre of a curve will be clamped to a curve time of 0.5 regardless of the shape of the curve, because it is halfway along the curve's bounding box. However the 0.5 curve time position can actually be anywhere within the bounding box!

## Options

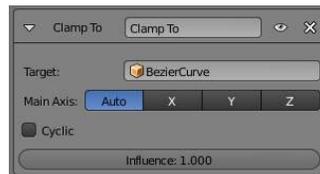


Fig. 2.1207: Clamp To panel

**Target** The Target: field indicates which curve object the Clamp To constraint will track along. The Target: field must be a curve object type. If this field is not filled in then it will be highlighted in red indicating that this constraint does not have all the information it needs to carry out its task and will therefore be ignored on the constraint stack.

**Main Axis** This button group controls which global axis (X, Y or Z) is the main direction of the path. When clamping the object to the target curve, it will not be moved significantly on this axis. It may move a small amount on that axis because of the inexact way this constraint functions.

For example if you are animating a rocket launch, it will be the Z axis because the main direction of the launch path is up. The default *Auto* option chooses the axis which the curve is longest in (or X if they are equal). This is usually the best option.

**Cyclic** By default, once the object has reached one end of its target curve, it will be constrained there. When the *Cyclic* option is enabled, as soon as it reaches one end of the curve, it is instantaneously moved to its other end. This is of course primarily designed for closed curves (circles & co), as this allows your owner to go around it over and over.

## Damped Track Constraint

The *Damped Track* constraint constrains one local axis of the owner to always point towards *Target*. In another 3D software you can find it with the name "Look at" constraint.

## Options

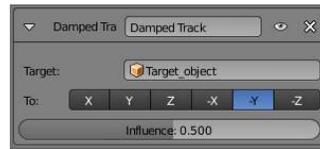


Fig. 2.1208: Damped Track panel

**Target (Mesh Object Type)** This constraint uses one target, and is not functional (red state) when it has none.

**Vertex Group** If *Target* is a *Mesh*, a new field is displayed offering the optional choice to set a *Vertex Group* as target.



Fig. 2.1209: Damped Track for Bones

### Target (Armature Object Type):

**Bone** If *Target* is an *Armature*, a new field is displayed offering the optional choice to set an individual bone as *Target*.

**Head/Tail** If *Target* is an *Armature*, a new field is displayed offering the optional choice to set whether the Head or Tail of a Bone will be pointed at by the *Target*. It is a slider value field which can have a value between 0 and 1. A value of 0 will point the Target at the Head/Root of a Bone while a value of 1 will point the Target at the Tail/Tip of a Bone.

**To** Once the owner object has had a Damped Track constraint applied to it, you must then choose which axis of the object you want to point at the Target object. You can choose between 6 axis directions (-X, -Y, -Z, X, Y, Z). The negative axis direction cause the object to point away from the Target object along the selected axis direction.

## IK Solver Constraint

The *Inverse Kinematics* constraint implements the *inverse kinematics* armature posing technique. Hence, it is only available for bones. To quickly create an IK constraint with a target, select a bone in pose mode, and press `Shift I`.

This constraint is fully documented in the [inverse kinematics page](#) of the rigging chapter.

## Options

**Target** Must be an armature

**Bone** A bone in the armature

**Pole Target** Object for pole rotation

**Iterations** Maximum number of solving iterations

**Chain Length** How many bones are included in the IK effect. Set to 0 to include all bones

**Use Tail** Include bone's tail as last element in chain

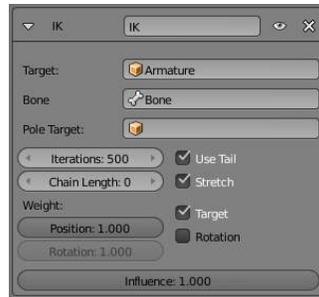


Fig. 2.1210: Inverse Kinematics panel

**Stretch** Enable IK stretching

#### Weight:

**Position** For Tree-IK: Weight of position control for this target

**Rotation** Chain follow rotation of target

**Target** Disable for targetless IK

**Rotation** Chain follows rotation of target

### Locked Track Constraint

The *Locked Track* constraint is a bit tricky to explain, both graphically and textually. Basically, it is a *Track To* constraint, but with a locked axis, i.e. an axis that cannot rotate (change its orientation). Hence, the owner can only track its target by rotating around this axis, and unless the target is in the plane perpendicular to the locked axis, and crossing the owner, this owner cannot really point at its target.

Let's take the best real world equivalent: a compass. It can rotate to point in the general direction of its target (the magnetic North, or a neighbor magnet), but it can't point *directly at it*, because it spins like a wheel on an axle. If a compass is sitting on a table and there is a magnet directly above it, the compass can't point to it. If we move the magnet more to one side of the compass, it still can't point *at* the target, but it can point in the general direction of the target, and still obey its restrictions of the axle.

When using a *Locked Track* constraint, you can think of the target as a magnet, and the owner as a compass. The *Lock* axis will function as the axle around which the owner spins, and the *To* axis will function as the compass' needle. Which axis does what is up to you!

If you have trouble understanding the buttons of this constraint, read the tool-tips, they are pretty good. If you don't know where your object's axes are, turn on the *Axis* button in the *Object* menu's *Draw* panel. Or, if you're working with bones, turn on the *Axes* button in the *Armature* menu's *Display* panel.

This constraint was designed to work cooperatively with the *Track To* constraint. If you set the axes buttons right for these two constraints, *Track To* can be used to point the axle at a primary target, and *Locked Track* can spin the owner around that axle to a secondary target.

This constraints also works very well for 2D billboarding.

This is all related to the topic discussed at length in the 2.49 BSoD tracking tutorial.

#### Options

**Target** This constraint uses one target, and is not functional (red state) when it has none.



Fig. 2.1211: Locked track panel

**To** The tracking local axis (*Y* by default), i.e. the owner's axis to point at the target. The negative options force the relevant axis to point away from the target.

### Lock

The locked local axis (*Z* by default), i.e. the owner's axis which cannot be re-oriented to track the target.

**Warning:** If you choose the same axis for *To* and *Lock*, the constraint will no longer be functional (red state).

## Spline IK Constraint

The *Spline IK* constraint aligns a chain of bones along a curve. By leveraging the ease and flexibility of achieving aesthetically pleasing shapes offered by curves and the predictability and well-integrated control offered by bones, *Spline IK* is an invaluable tool in the riggers' toolbox. It is particularly well suited for rigging flexible body parts such as tails, tentacles, and spines, as well as inorganic items such as ropes.

To set up *Spline IK*, it is necessary to have a chain of connected bones and a curve to constrain these bones to.

- With the last bone in the chain selected, add a *Spline IK* constraint from the *Bone Constraints* tab in the *Properties Editor*.
- Set the 'Chain Length' setting to the number of bones in the chain (starting from and including the selected bone) that should be influenced by the curve.
- Finally, set *Target* to the curve that should control the curve.

### Options

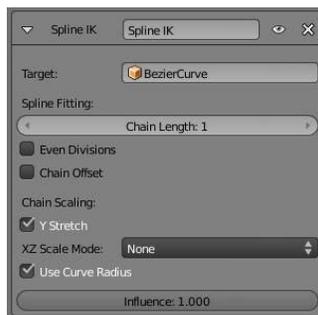


Fig. 2.1212: Spline IK panel

**Target** The target curve

### Spline Fitting:

**Chain Length** How many bones are included in the chain

**Even Division** Ignore the relative length of the bones when fitting to the curve

**Chain Offset** Offset the entire chain relative to the root joint

#### Chain Scaling:

**Y stretch** Stretch the Y axis of the bones to fit the curve

#### XZ Scale Mode:

**None** Don't scale the X and X axes (default)

**Bone Original** Use the original scaling of the bones

**Volume Preservation** Scale of the X and Z axes is the inverse of the Y scale

**Use Curve Radius** Average radius of the endpoints is used to tweak the X and Z scaling of the bones, on top of the X and Z scale mode

**See also** This subject is seen in depth in the [Rigging/Posing](#) section.

- [Blender.org 2.56 Release Log for Spline IK](#)

## Stretch To Constraint

The *Stretch To* constraint causes its owner to rotate and scale its Y axis towards its target. So it has the same tracking behavior as the *Track To* constraint. However, it assumes that the Y axis will be the tracking and stretching axis, and doesn't give you the option of using a different one.

It also optionally has some raw volumetric features, so the owner can squash down as the target moves closer, or thin out as the target moves farther away. Note however that it is not the real volume of the owner which is thus preserved, but rather the virtual one defined by its scale values. Hence, this feature works even with non-volumetric objects, like empties, 2D meshes or surfaces, and curves.

With bones, the “volumetric” variation scales them along their own local axes (remember that the local Y axis of a bone is aligned with it, from root to tip).

### Options

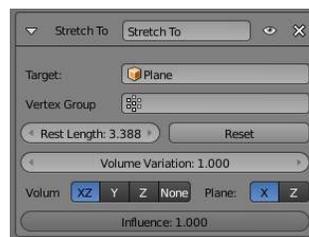


Fig. 2.1213: Stretch To panel for a Mesh Object

**Target (Mesh Object Type)** This constraint uses one target, and is not functional (red state) when it has none.

**Vertex Group** When *Target* is a mesh, a new field is display where a vertex group can be selected.

**Target (Armature Object Type)** This constraint uses one target, and is not functional (red state) when it has none.

**Bone** When *Target* is an armature, a new field for a bone is displayed.

**Head/Tail** When using a Bone *Target*, you can choose where along this bone the target point lies.

**Rest Length** This numeric field sets the rest distance between the owner and its target, i.e. the distance at which there is no deformation (stretching) of the owner.

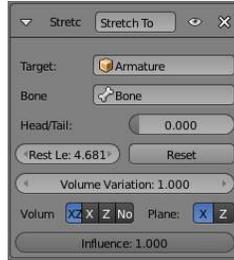


Fig. 2.1214: Stretch To panel for a Armature Object

**Reset** When clicked, this small button will recalculate the *Rest Length* value, so that it corresponds to the actual distance between the owner and its target (i.e. the distance before this constraint is applied).

**Volume Variation** This numeric field controls the amount of “volume” variation proportionally to the stretching amount. Note that the **0.0** value is not allowed, if you want to disable the volume feature, use the *None* button (see below).

**Volume** These buttons control which of the X and/or Z axes should be affected (scaled up/down) to preserve the virtual volume while stretching along the Y axis. If you enable the *NONE* button, the volumetric features are disabled.

**Plane** These buttons are equivalent to the *Up* ones of the [Track To constraint](#): they control which of the X or Z axes should be maintained (as much as possible) aligned with the global Z axis, while tracking the target with the Y axis.

## Track To Constraint

### Description

The *Track To* constraint applies rotations to its owner, so that it always points a given “To” axis towards its target, with another “Up” axis permanently maintained as much aligned with the global Z axis (by default) as possible. This tracking is similar to the “billboard tracking” in 3D (see note below).

This is the preferred tracking constraint, as it has a more easily controlled constraining mechanism.

This constraint shares a close relationship to the [Inverse Kinematics constraint](#) in some ways. It is very important in rig design, and you should be sure to read and understand the [2.49 BSoD tracking tutorial](#), as it centers around the use of both of these constraints.

---

**Tip:** Billboard tracking

The term “billboard” has a specific meaning in real-time CG programming (i.e. video games!), where it is used for plane objects always facing the camera (they are indeed “trackers”, the camera being their “target”). Their main usage is as support for tree or mist textures: if they were not permanently facing the camera, you would often see your trees squeezing to nothing, or your mist turning into a millefeuille paste, which would be funny but not so credible.

---

### Options

**Targets** This constraint uses one target, and is not functional (red state) when it has none.

**Bone** When *Target* is an armature, a new field for a bone is displayed.

**Head/Tail** When using a bone target, you can choose where along this bone the target point lies.

**Vertex Group** When *Target* is a mesh, a new field is display where a vertex group can be selected.

**To** The tracking local axis (*Y* by default), i.e. the owner’s axis to point at the target. The negative options force the relevant axis to point away from the target.

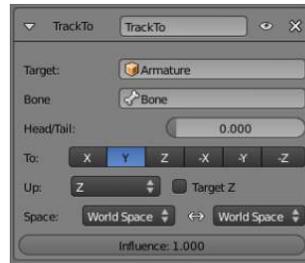


Fig. 2.1215: Track To panel

**Up** The “upward-most” local axis (*Z* by default), i.e. the owner’s axis to be aligned (as much as possible) with the global *Z* axis (or target *Z* axis, when the *Target* button is enabled).

**Target Z** By default, the owner’s *Up* axis is (as much as possible) aligned with the global *Z* axis, during the tracking rotations. When this button is enabled, the *Up* axis will be (as much as possible) aligned with the target’s local *Z* axis?

### Space

This constraint allows you to choose in which space to evaluate its owner’s and target’s transform properties.

**Warning:** If you choose the same axis for *To* and *Up*, the constraint will not be functional anymore (red state).

## Action Constraint

The *Action* constraint is powerful. It allows you control an *Action* using the transformations of another object.

The underlying idea of the *Action* constraint is very similar to the one behind the *Drivers*, except that the former uses a whole action (i.e. a bunch of *Fcurves* of the same type), while the latter controls a single *Fcurve* of their “owner”...

Note that even if the constraint accepts the *Mesh* action type, only the *Object*, *Pose* and *Constraint* types are really working, as constraints can only affect objects’ or bones’ transform properties, and not meshes’ shapes. Also note that only the object transformation (location, rotation, scale) is affected by the action, if the action contains keyframes for other properties they are ignored, as constraints do not influence those.

As an example, let’s assume you have defined an *Object* action (it can be assigned to any object, or even no object at all), and have mapped it on your owner through an *Action* constraint, so that moving the target in the  $[0.0, 2.0]$  range along its *X* axis maps the action content on the owner in the  $[0, 100]$  frame range. This will mean that when the target’s *X* property is  $0.0$  the owner will be as if in frame 0 of the linked action; with the target’s *X* property at  $1.0$  the owner will be as if in frame 50 of the linked action, etc.

### Options

**Target** This constraint uses one target, and is not functional (red state) when it has none.

**Bone:** When target is an armature object, use this field to select the target bone.

**Transform Channel** This drop-down list controls which transform property (location, rotation or scale along/around one of its axes) from the target to use as “action driver”.

**Target Space** This constraint allows you to choose in which space to evaluate its target’s transform properties.

### To Action

Select the name of the action you want to use.

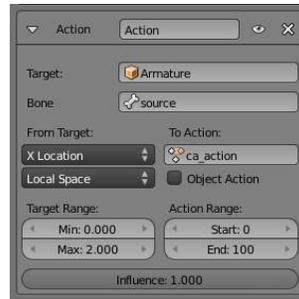


Fig. 2.1216: Action panel

**Warning:** Even though it might not be in red state (UI refresh problems...), this constraint is obviously not functional when this field does not contain a valid action.

**Object Action Bones only**, when enabled, this option will make the constrained bone use the “object” part of the linked action, instead of the “same-named pose” part. This allows you to apply the action of an object to a bone.

### Target Range Min / Max

The lower and upper bounds of the driving transform property value. By default, both values are set to 0.0

**Warning:** Unfortunately, here again we find the constraints limitations:

- When using a rotation property as “driver”, these values are “mapped back” to the  $[-180.0- , 180.0- [$  range.
- When using a scale property as “driver”, these values are limited to null or positive values.

**Action Range Start / End** The starting and ending frames of the action to be mapped. Note that:

- These values must be strictly positive.
- By default, both values are set to 0 which disables the mapping (i.e. the owner just gets the properties defined at frame 0 of the linked action...).

### Notes

- When the linked action affects some location properties, the owner’s existing location is added to the result of evaluating this constraint (exactly as when the *Offset* button of the [Copy Location](#) constraint is enabled...).
- When the linked action affects some scale properties, the owner’s existing scale is multiplied with the result of evaluating this constraint.
- When the linked action affects some rotation properties, the owner’s existing rotation is overridden by the result of evaluating this constraint.
- Unlike usual, you can have a *Start* value higher than the *End* one, or a *Min* one higher than a *Max* one: this will reverse the mapping of the action (i.e. it will be “played” reversed...), unless you have both sets reversed, obviously!
- When using a *Constraint* action, it is the constraint *channel’s names* that are used to determine to which constraints of the owner apply the action. E.g. if you have a constraint channel named “trackto\_empt1”, its keyed *Influence* and/or *Head/Tail* values (the only ones you can key) will be mapped to the ones of the owner’s constraint named “trackto\_empt1”.
- Similarly, when using a *Pose* action (which is obviously only meaningful and working when constraining a bone!), it is the bone’s name that is used to determine which bone *channel’s names* from the action to use (e.g. if the constrained

bone is named “arm”, it will use and only use the action’s bone channel named “arm”...). Unfortunately, using a *Pose* action on a whole armature object (to affect all the keyed bones in the action at once) won’t work...

- Note also that you can use the [pose library feature](#) to create/edit a *Pose* action datablock... just remember that in this situation, there’s one pose per frame!

## Child Of Constraint

*Child Of* is the constraint version of the standard parent/children relationship between objects (the one established through the `Ctrl-P` shortcut, in the 3D views).

Parenting with a constraint has several advantages and enhancements, compared to the traditional method:

- You can have several different parents for the same object (weighting their respective influence with the *Influence* slider).
- As with any constraint, you can key (i.e. animate) its Influence setting. This allows the object which has a Child Of constraint upon it to change over time which target object will be considered the parent, and therefore have influence over the Child Of constrained object.

**Warning:** Don’t confuse this “basic” object parenting with the one that defines the chains of bones inside of an armature. This constraint is used to parent an object to a bone (the so-called [object skinning](#)), or even bones to bones. But don’t try to use it to define chains of bones.

## Options

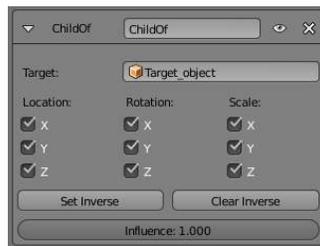


Fig. 2.1217: Child Of panel

**Target** The target object that this object will act as a child of. This constraint uses one target, and is not functional (red state) when it has none. If *Target* is an armature or a mesh, a new name field appears where a name of a *Bone* or a *Vertex Group* can be selected.

**Location X, Y, Z** Each of these buttons will make the parent affect or not affect the location along the corresponding axis.

**Rotation X, Y, Z** Each of these buttons will make the parent affect or not affect the rotation around the corresponding axis.

**Scale X, Y, Z** Each of these buttons will make the parent affect or not affect the scale along the corresponding axis.

**Set Inverse** By default, when you parent your owner to your target, the target becomes the origin of the owner’s space. This means that the location, rotation and scale of the owner are offset by the same properties of the target. In other words, the owner is transformed when you parent it to your target. This might not be desired! So, if you want to restore your owner to its before-parenting state, click on the *Set Inverse* button.

**Clear Inverse** This button reverses (cancels) the effects of the above one, restoring the owner/child to its default state regarding its target/parent.

## Tips

When creating a new parent relationship using this constraint, it is usually necessary to click on the *Set Inverse* button after assigning the parent. As noted above, this cancels out any unwanted transform from the parent, so that the owner returns to the location/rotation/scale it was in before the constraint was applied. Note that you should apply *Set Inverse* with all other constraints disabled (their *Influence* set to **0.0**) for a particular *Child Of* constraint, and before transforming the target/parent (see example below).

About the toggle buttons that control which target's (i.e. parent's) individual transform properties affect the owner, it is usually best to leave them all enabled, or to disable all three of the given Location, Rotation or Scale transforms.

### Technical Note

If you use this constraint with all channels on, it will use a straight matrix multiplication for the parent relationship, not decomposing the parent matrix into loc/rot/size. This ensures any transformation correctly gets applied, also for combinations of rotated and non-uniform scaled parents.

### Examples

<p><b>1. No constraint</b> Note the position of Owner empty - 1.0 BU along X and Y axes.</p>	<p><b>2. Child Of just added</b> Here you can see that Owner empty is now <b>1.0 BU</b> away from Target_1 empty along X and Y axes.</p>
<p><b>3. Offset set</b> <i>Set Inverse</i> has been clicked, and Owner is back to its original position.</p>	<p><b>4. Target/parent transformed</b> Target_1 has been translated in the XY plane, rotated around the Z axis, and scaled along its <i>local X</i> axis.</p>
<p><b>5. Offset cleared</b> <i>Clear Inverse</i> has been clicked - Owner is fully again controlled by Target_1.</p>	<p><b>6. Offset set again</b> <i>Set Offset</i> has been clicked again. As you can see, it does not give the same result as in (<i>Target/parent transformed</i>). As noted above, use <i>Set Inverse</i> only once, before transforming your target/parent.</p>

## Floor Constraint

### Description

The *Floor* constraint allows you to use its target position (and optionally rotation) to specify a plane with a “forbidden side”, where the owner cannot go. This plane can have any orientation you like. In other words, it creates a floor (or a ceiling, or a wall)! Note that it is only capable of simulating entirely flat planes, even if you use the *Vertex Group* option. It cannot be used for uneven floors or walls.

### Options



Fig. 2.1218: Floor panel

**Targets** This constraint uses one target, and is not functional (red state) when it has none.

**Bone** When *Target* is an armature, a new field for a bone is displayed.

**Vertex Group** When *Target* is a mesh, a new field is display where a vertex group can be selected.

**Sticky** This button makes the owner immovable when touching the “floor” plane (it cannot slide around on the surface of the plane any more). This is fantastic for making walk and run animations!

**Use Rotation** This button forces the constraint to take the target’s rotation into account. This allows you to have a “floor” plane of any orientation you like, not just the global XY, XZ and YZ ones...

**Offset** This numeric field allows you to offset the “floor” plane from the target’s center, by the given number of Blender Units. Use it e.g. to account for the distance from a foot bone to the surface of the foot’s mesh.

**Max / Min** This set of (mutually exclusive) buttons controls which plane will be the “floor”. The buttons’ names correspond indeed to the *normal* to this plane (e.g. enabling Z means “XY plane”, etc.) By default, these normals are aligned with the *global* axes. However, if you enable *Use Rotation* (see above), they will be aligned with the *local target’s axes*. As the constraint does not only define an uncrossable plane, but also a side of it which is forbidden to the owner, you can choose which side by enabling either the positive or negative normal axis... E.g, by default (Z), the owner is stuck in the positive Z coordinates.

**Space** This constraint allows you to choose in which space to evaluate its owner’s and target’s transform properties.

## Follow Path Constraint

The *Follow Path* constraint places its owner onto a *curve* target object, and makes it move along this curve (or path). It can also affect its owner’s rotation to follow the curve’s bends, when the *Follow Curve* option is enabled.

The owner is always evaluated in the global (world) space:

- Its location (as shown in the *Transform Properties* panel, N) is used as an offset from its normal position on the path. E.g. if you have an owner with the (1.0, 1.0, 0.0) location, it will be one BU away from its normal position on the curve, along the X and Y axis. Hence, if you want your owner *on its target path*, clear its location (Alt-G)!
- This location offset is also proportionally affected by the *scale of the target curve*. Taking the same (1.0, 1.0, 0.0) offset as above, if the curve has a scale of (2.0, 1.0, 1.0), the owner will be offset *two* BU along the X axis (and one along the Y one)...
- When the *Curve Follow* option is enabled, its rotation is also offset to the one given by the curve (i.e. if you want the Y axis of your object to be aligned with the curve’s direction, it must be in rest, non-constrained state, aligned with the global Y axis). Here again, clearing your owner’s rotation (Alt-R) might be useful...

The movement of the owner along the target curve/path may be controlled in two different ways:

- The most simple is to define the number of frames of the movement, in the Path Animation panel of the Object Data context, via the numeric field Frames, and its start frame via the constraint’s Offset option (by default, start frame: 1 [= offset of 0]), duration: 100).
- The second way, much more precise and powerful, is to define a *Evaluation Time* interpolation curve for the *Target* path (in the *Graph Editor*. See the [animation chapter](#) to learn more about Fcurves.
- If you don’t want your owner to move along the path, you can give to the target curve a flat *Speed* FCurve (its value will control the position of the owner along the path).

*Follow Path* is another constraint that works well with the [Locked Track one](#). One example is a flying camera on a path. To control the camera’s roll angle, you can use a *Locked Track* and a target object to specify the up direction, as the camera flies along the path.

---

**Note:** *Follow Path* and *Clamp To*

Do not confuse these two constraints. Both of them constraint the location of their owner along a curve, but *Follow Path* is an “animation-only” constraint, inasmuch that the position of the owner along the curve is determined by the time (i.e. current frame), whereas the [Clamp To constraint](#) determines the position of its owner along the curve using one of its location properties’ values.

**Note:** Note that you also need to keyframe Evaluation Time for the Path. Select the path, go to the path properties, set the overall frame to the first frame of the path (e.g. frame 1), set the value of Evaluation time to the first frame of the path (e.g. 1), right click on Evaluation time, select create keyframe, set the overall frame to the last frame of the path (e.g. frame 100), set the value of Evaluation time to the last frame of the path (e.g. 100), right click on Evaluation time, select create keyframe. ... Comment: <!-- from <http://overshoot.tv/node/1123> paragraph needs cleanup but this definitely needs to be in the documentation --> .

## Options

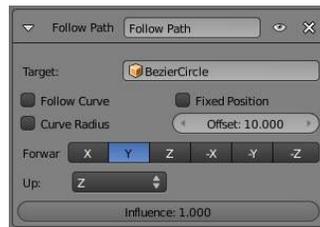


Fig. 2.1219: Follow Path panel

**Target** This constraint uses one target, which *must be a curve object*, and is not functional (red state) when it has none.

**Curve Radius** Objects scale by the curve radius. See [Curve Editing](#)

**Fixed Position** Object will stay locked to a single point somewhere along the length of the curve regardless of time

**Offset** The number of frames to offset from the “animation” defined by the path (by default, from frame 1).

**Follow Curve** If this option is not activated, the owner’s rotation isn’t modified by the curve; otherwise, it’s affected depending on the following options:

**Forward** The axis of the object that has to be aligned with the forward direction of the path (i.e. tangent to the curve at the owner’s position).

**Up** The axis of the object that has to be aligned (as much as possible) with the world Z axis. In fact, with this option activated, the behavior of the owner shares some properties with the one caused by a [Locked Track constraint](#), with the path as “axle”, and the world Z axis as “magnet”.

## Pivot Constraint

### Description

The *Pivot* constraint allows the owner to rotate around a target object.

It was originally intended for foot rigs.

### Options

**Target** The object to be used as a pivot point

**Bone** When *Target* is an armature, a new field for a bone is displayed.

**Head/Tail** When using a bone target, you can choose where along this bone the target point lies.

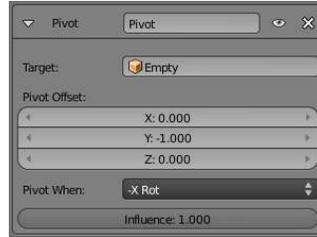


Fig. 2.1220: Pivot panel

**Vertex Group** When *Target* is a mesh, a new field is display where a vertex group can be selected.

**Pivot Offset** Offset of pivot from target

**Pivot When** Always, Z Rot, Y Rot...

### Example

### See also

- [Blender Artists Forum: Head-Tail pivot Constrain proposal \(with Video and .blend\)](#), the thread where the constraint was first proposed

## Rigid Body Joint Constraint

### Description

The *Rigid Body Joint* constraint is very special. Basically, it is used by the physical part of the Blender Game Engine to simulate a joint between its owner and its target. It offers four joint types: hinge type, ball-and-socket type, cone-twist, and generic six-DoF (degrees of freedom) type.

The joint point and axes are defined and fixed relative to the owner. The target moves as if it were stuck to the center point of a stick, the other end of the stick rotating around the joint/pivot point...

This constraint is of no use in most “standard” static or animated projects. However, you can use its results outside of the BGE, through the *Game* → *Record Animation* menu entry (from the main menu of the *User Preferences* window, see [Rigid Bodies](#) for more info on this topic).

For a demo file that shows some of the different types, see: [BGE-Physics-RigidBodyJoints.blend](#).

---

**Note:** In order for this constraint to work properly, both objects (so the owner and the target object) need to have “Collision Bounds” enabled.

---

### Options

**Target** This constraint uses one target, and is not functional (red state) when it has none.

### Joint Type:

**Ball** works like an ideal ball-and-socket joint, i.e. allows rotations around all axes like a shoulder joint.

**Hinge** works in one plane, like an elbow: the owner and target can only rotate around the X axis of the pivot (joint point).

**Limits** Angular limits for the X axis

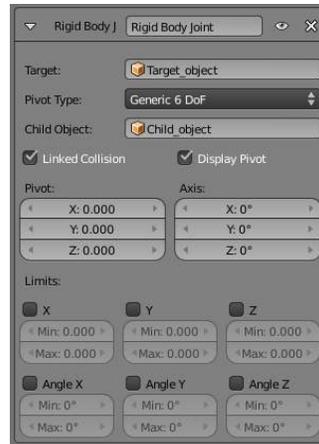


Fig. 2.1221: Rigid Body Joint panel

**Cone Twist** similar to *Ball*, this is a point-to-point joint with limits added for the cone and twist axis

**Limits** Angular limits

**Generic 6DOF** works like the *Ball* option, but the target is no longer constrained at a fixed distance from the pivot point, by default (hence the six degrees of freedom: rotation and translation around/along the three axes). In fact, there is no longer a joint by default, with this option, but it enables additional settings which allow you to restrict some of these DoF:

**Limits** Linear and angular limits for a given axis (of the pivot) in Blender Units and degrees respectively.

**Child Object** normally, leave this blank. You can reset it to blank by right clicking and selecting Reset to Default Value. Comment: <!-- Is this right? 2.4 just had a 'to object'. Now we have a 'target' and a 'child object'. These are not documented. It seems that we recreate the behaviour of 2.4 by leaving the child object blank. The target seems to be the 2.4 'to object'. What is the child object? Please explain: m.e -> .

**Linked Collision** When enabled, this will disable the collision detection between the owner and the target (in the physical engine of the BGE).

**Display Pivot** When enabled, this will draw the pivot of the joint in the 3D views. Most useful, especially with the *Generic 6DOF* joint type!

**Pivot** These three numeric fields allow you to relocate the pivot point, *in the owner's space*.

**Axis** These three numeric fields allow you to rotate the pivot point, *in the owner's space*.

## Script Constraints

This feature is not supported in Blender 2.6

## Shrinkwrap Constraint

The *Shrinkwrap* constraint is the “object counterpart” of the *Shrinkwrap modifier*. It moves the owner origin and therefore the owner object's location to the surface of its target.

This implies that the target *must* have a surface. In fact, the constraint is even more selective, as it can only use meshes as targets. Hence, the *Shrinkwrap* option is only shown in the *Add Constraint to Active Object* menu, `Ctrl-Alt-C`, (or its bone's equivalent), when the selected inactive object is a mesh.

## Options

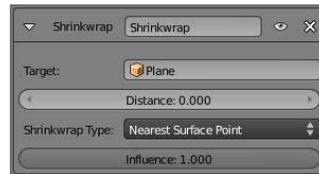


Fig. 2.1222: Shrinkwrap panel

**Target** This constraint uses one target, which *must be a mesh object*, and is not functional (red state) when it has none.

**Distance** This numeric field controls the offset of the owner from the shrunk computed position on the target’s surface. Positive values place the owner “outside” of the target, and negative ones, “inside” the target. This offset is applied along the straight line defined by the original (i.e. before constraint) position of the owner, and the computed one on the target’s surface.

**Shrinkwrap Type** This drop-down list allows you to select which method to use to compute the point on the target’s surface to which to translate the owner’s center. You have three options:

**Nearest Surface Point** The chosen target’s surface’s point will be the nearest one to the original owner’s location. This is the default and most commonly useful option.

**Projection** The target’s surface’s point is determined by projecting the owner’s center along a given axis. This axis is controlled by the three *X*, *Y* and *Z* toggle buttons that show up when you select this type. This means the projection axis can only be aligned with one of the global axes, median to both of them (*XY*, *XZ* or *YZ*), or to the three ones (*XYZ*). When the projection of the owner’s center along the selected direction does not hit the target’s surface, the owner’s location is left unchanged.

**Nearest Vertex** This method is very similar to the *Nearest Surface Point* one, except that the owner’s possible shrink locations are limited to the target’s vertices.

## 2.6 Animation

### 2.6.1 Introduction

Animation is making an object move or change shape over time. Objects can be animated in many ways:

**Moving as a whole object** Changing their position, orientation or size in time;

**Deforming them** Animating their vertices or control points;

**Inherited animation** Causing the object to move based on the movement of another object (e.g. its parent, hook, armature, etc...).

In this chapter we will cover the first two, but the basics given here are actually vital for understanding the following chapters as well.

Animation is typically achieved with the use of [Key Frames](#).

## Chapters

### General Principles and Tools

- [Key frames](#)

- Animation Editors
- Using The Timeline
- Markers

### The Graph Editor

- F-Curves
- Editing Curves
- F-Curve Modifiers

### The Action Editor

- Actions
- *Working with Actions*

### Animation Techniques

- Constraints
- Moving objects on a Path
- Game Engine Physics Recording

### Animating Deformation

- Shape Keys
- Deforming by a Lattice
- Deforming with Hooks

See also Hook Modifier

### Drivers

- Drivers
- Driven Shape Keys

The [Introduction to Character Animation tutorial](#) is a good starting point for learning character animation. Even if you never used Blender before.

### Animation Basics

**Actions** Actions are used to record the animation of objects and properties.

**Drivers** Drivers are used to control and animate properties.

**Keying Sets** Keying Sets are used to record a set of properties at the same time.

**Markers** Markers are used to mark key points/events within an animation.

**Motion Paths** Motion Paths are used to visualize an animation.

**Shape Keys** Shape Keys are used to deform objects into new shapes.

### Animation Editors

**Timeline** The Timeline Editor is a quick editor to set and control the time frame. This also has some tools for animation.

**Graph Editor** The Graph Editor is mostly used to edit the F-Curves and Keyframes for Channels and Drivers.

**Dope Sheet** The Dopes Sheet contains a collection of animation editors.

**NLA Editor** The NLA Editor is used to edit and blend Actions together.

### Categories

**Modifiers** Modifiers are automatic operations that affect an object in a non-destructive way. With modifiers, you can perform many effects automatically that would otherwise be tedious to do manually.

**Rigging** Rigging.

**Constraints** Constraints are a way of connecting transform properties (position, rotation and scale) between objects.

**Physical Simulation** This category covers various advanced Blender effects, often used to simulate real physical phenomena. There is the Particle System for things like hair, grass, smoke, flocks. Soft Bodies are useful for everything that tends to bend, deform, in reaction to forces like gravity or wind. Cloth simulation, to simulate clothes or materials. Rigid Bodies can simulate dynamic objects that are fairly rigid. Fluids, which include liquids and gasses, can be simulated, including Smoke. Force Fields can modify the behavior of simulations.

**Motion Tracking** Motion tracking is a new technique available in Blender. It is still under development, and currently supports basic operations for 2D motion tracking, 3D motion tracking, and camera solution.

**Animation Scripts** Add-on scripts for animation.

**Rigging Scripts** Add-on scripts for rigging.

## 2.6.2 Key Frames

A *Key Frame* is simply a marker in time which stores the value of a property.

For example, a key frame might indicate that the horizontal position of a cube is at 3m on frame 1.

The purpose of a key frame is to allow for interpolated animation, meaning, for example, that the user could then add another key on frame 10, specifying the cube's horizontal position at 20m, and Blender will automatically determine the correct position of the cube for all the frames between frame 1 and 10 depending on the chosen interpolation method (e.g. linear, bezier, quadratic, etc...).

### Adding Key Frames

There are several methods of adding new keys. Namely:

- In the 3D View, pressing **I** will bring up a menu to choose what to add a key frame to.
- Hovering over a property and pressing **I**.
- RMB a value and choose *Insert Keyframe* from the menu.

## Removing Key Frames

There are several methods of removing key frames

- In the 3D View press **Alt-I** to remove keys on the current frame for selected objects.
- When the mouse is over a value press **Alt-I**.
- RMB a value and choose *Delete Keyframe* from the menu.

## Editing Key Frames

For editing key frames go to the [Graph Editor](#) or to the [Dopesheet](#)

## 2.6.3 Actions

### Actions

When animating objects and properties in blender, Actions record and contain the data.

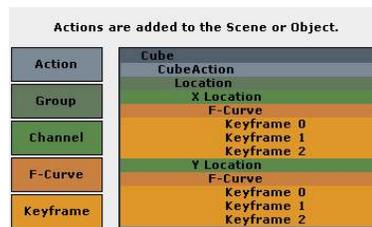


Fig. 2.1223: Actions.

So when you animate an object by changing its location with keyframes, the animation is saved to the Action.

Each property has a channel which it is recorded to, for example, `Cube.location.x` is recorded to Channel X Location.

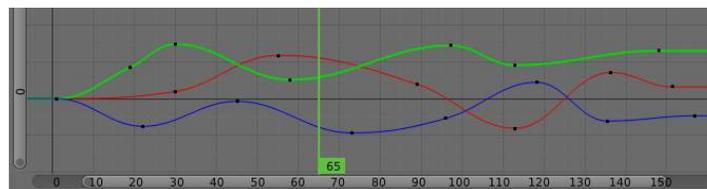


Fig. 2.1224: Graph Editor. Each Channel has an F-Curve represented by the lines between the keyframes.

**Actions** Record and contain animation data.

**Groups** Are groups of channels.

**Channels** Record properties.

**F-Curves** Are used to interpolate the difference between the keyframes.

**Keyframes** Are used to set the values of properties.

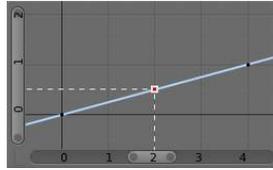


Fig. 2.1225: Graph Editor: Channel F-Curve.

## F-Curve Interpolation

The keyframes are set values by the user.

The *F-Curve* is used to interpolate the difference between the keyframes.

The *F-Curve* has different types of interpolation and also *F-Curve Modifiers*.

Most the settings for the *F-Curve* are found in the *Graph Editor*.

## Basic Animation

These are some common ways to animate objects. These methods can be used on different objects, like armature bones in pose mode.

### Insert Keyframes

This example shows you how to animate a cubes location, rotation, and scale.

- First, in the *Timeline*, or other animation editors, set the frame to 1.
- With the *Cube* selected in *Object Mode*, press **I** in the 3D View.
- From the *Insert Keyframe Menu* select *LocRotScale*.
- This will record the location, rotation, and scale, for the *Cube* on frame 1.
- Set the frame to 100.
- Use **Grab/Move G**, **Rotate R**, **Scale S**, to transform the cube.
- Press **I** in the 3D View. From the *Insert Keyframe Menu* select *LocRotScale*.

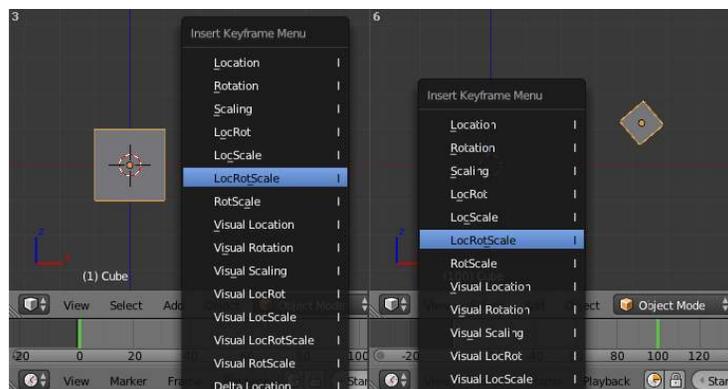


Fig. 2.1226: Insert Keyframes.

To test the animation, press **Alt-A** to play.

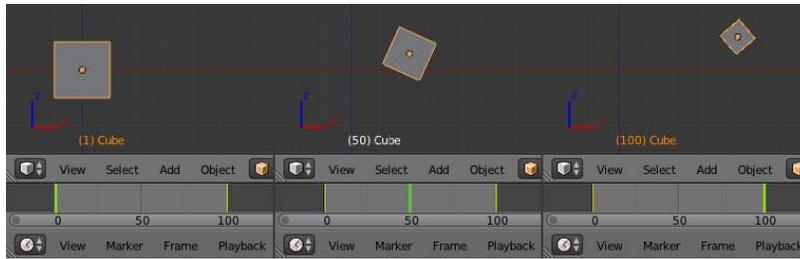


Fig. 2.1227: The animation on frames 1, 50, 100.

## Auto Keyframe

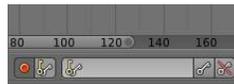


Fig. 2.1228: Timeline Auto Keyframe.

Auto Keyframe is the red record button in the *Timeline* header. Auto Keyframe adds keyframes automatically to the set frame if the value for transform type properties changes.

See *Timeline V Keyframe Control* for more info.

## Keying Sets

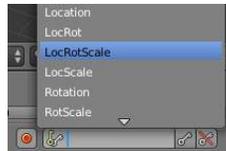


Fig. 2.1229: Timeline Keying Sets.

Keying Sets are a set of keyframe channels. They are used to record multiple properties at the same time. There are some built in keying sets, 'LocRotScale', and also custom keying sets can be made.

To use the keying set, first select a keying set from the *Timeline* header, or the *Keying Sets Panel*.

Now when you press **I** in the 3D view, blender will add keyframes for all the properties in the active keying set.

See *Keying Sets* for more info.

## Properties

Keyframes can be used to animate lots of different properties in blender. To add keyframes to a property in the UI, RMB the property, then select Insert Single Keyframe, or Insert Keyframes. Insert Keyframes **I** will add a keyframes for the set of properties.

Properties have different colors and menu items for different states.

**Gray - Property is not animated with Keyframes or Drivers.** Insert Keyframes **I**. Insert Single Keyframe. Add Drivers. Add Single Driver. Paste Driver.

**Purple - Property value is controlled with a Driver.** Delete Drivers. Delete Single Driver. Copy Driver. Paste Driver.



Fig. 2.1230: Keyframe properties.

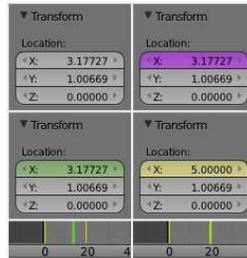


Fig. 2.1231: Properties, Drivers, Keyframes.

**Green - Property has Channel with Keyframes.** Insert Keyframes **I**. Insert Single Keyframe. Clear Keyframes **Alt-Shift-I** Clear Single Keyframes.

**Yellow - Property has Keyframes on the current Frame.** Replace Keyframes **I**. Replace Single Keyframe. Delete Keyframes **Alt-I**. Delete Single Keyframe. Clear Keyframes **Alt-Shift-I** Clear Single Keyframes.

**Each property also has some Keying Set options.** Add All to Keying Set **K**. Add Single to Keying Set. Remove from Keying Set.

## Editing

**3D View.** Insert Keyframes on current frame **I** Delete Keyframes on current frame **Alt-I**

## Working with Actions

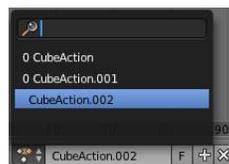


Fig. 2.1232: Action Browser.

When you first animate an object by adding keyframes, blender creates an *Action* to record the data.

*Actions* can be managed with the *Action Browser* in the *DopeSheet Action Editor* header, or the properties region of the *NLA Editor*.

If you are making multiple actions for the same object, press the **F** button for each action, this will give the actions a *Fake User* and will make blender save the unlinked actions.

Objects can only use one *Action* at a time for editing, the *NLA Editor* is used to blend multiple actions together.

## 2.6.4 Drivers

### Drivers

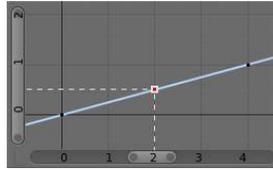


Fig. 2.1233: Graph Editor: Driver example.

Drivers can use properties, numbers, transformations, and scripts, to control the values of properties.

Using a F-Curve, the driver reads the value of the Driver Value and sets the value of the selected property it was added to.

So from this example, if the Driver Value is 2.0 the property will be 0.5.

The Driver Value is determined by Driver Variables or a Scripted Expression.

Most the settings for the drivers F-Curves are found in the [Graph Editor](#).

### Drivers Panel



Fig. 2.1234: Graph Editor: Drivers: Drivers Panel.

This panel is located in the [Graph Editor](#) with the mode set to Drivers.

The drivers panel is for setting up *Driver Variables* or a *Scripted Expression* which will determine the value of the *Driver Value*.

### Driver Settings

**Update Dependencies** This will force an update for the Driver Value dependencies.

**Remove Driver** Removes the driver from the object.

**Type** The type of calculation to use on the set of Driver Variables. (If you only have one driver variable there is no real difference between average, sum, minimum and maximum)

**Average Value** Uses the average value of the referenced Driver Variables.

**Sum Values** Uses the sum of the referenced Driver Variables.

**Scripted Expression** Uses a Scripted Expression. See Expr. You must write a python expression which performs your own calculations on the Driver Variables.

**Minimum Value** Uses the lowest value from the referenced Driver Variables.

**Maximum Value** Uses the highest value from the referenced Driver Variables.

**Expr** Scripted Expression. Here you can add real numbers, math operators, math functions, python properties, driver functions. See Driver Expression below for some examples.

**Show Debug Info** Shows the Driver Value. The current value of the variables or scripted expression.

**Add Variable** Adds a new Driver Variable.



Fig. 2.1235: Setup of a Single Property.

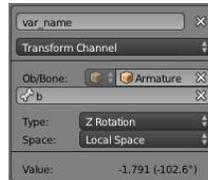


Fig. 2.1236: Transform Channel setup.



Fig. 2.1237: Distance setup.

## Driver Variables

**Name** Name to use for scripted expressions/functions. No spaces or dots are allowed and must start with a letter.

**Variable Type** The type of variable to use.

**Single Property** Use the value from some RNA property. For example, the Ambient shading color from a material. First select the type of ID-block, then the ID of the ID-block, then copy and paste an RNA property (Ctrl+V).

**ID-Type** The ID-Block type, example, Key, Image, Object, Material.

**ID** The ID of the ID-Block type, example, `Material.001`.

**RNA Path** The RNA id name of the property, example, 'ambient' from material shading.

**Transform Channel** Use one of the Transform channels from an object or bone.

**ID** ID of the object, example, Cube, Armature, Camera.

**Bone** ID of the Armature bone, example, Bone, `Bone.002`, `Arm.r`. This option is for armatures.

**Type** Example, X Location, X Rotation, X Scale.

**Space** World Space, Transform Space, Local Space.

**Rotational Difference** Use the rotational difference between two objects or bones.

**Distance** Use the distance between two objects or bones.

**Value** Shows the value of the variable.

## Workflow

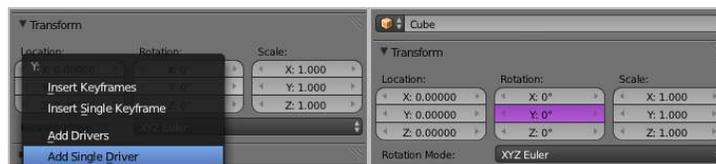
There are some different ways to add drivers in blender. These are some driver examples and workflow. After adding drivers they are usually modified in the *Graph Editor* with the mode set the *Drivers*.

## UI

The common way to add a driver to a property is to right click a property, then add a driver via the context menu.

**Add Drivers** This will add drivers to the set of properties related to the selected one. For example, it will add drivers to X, Y, and Z for Rotation.

**Add Single Driver** This will add a single driver to the selected property.



Drivers can also be added by pressing `D` with the mouse over the property set.

## Expression

This is quick way to add drivers with a scripted expression. First click the property you want add a driver to, then add a hash `#` and a scripted expression.

Some examples.

- `#frame`
- `#frame / 20.0`
- `#sin(frame)`
- `#cos(frame)`

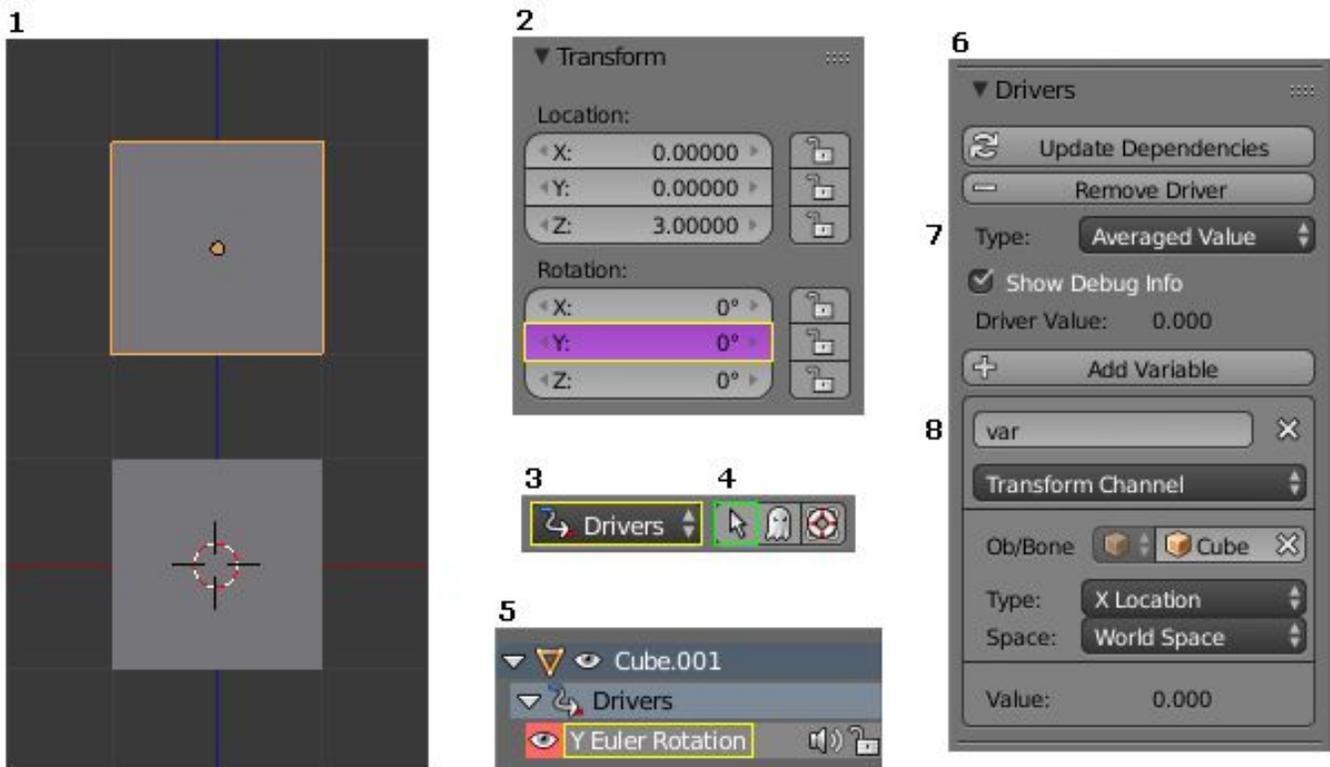
## Copy Paste

Drivers can be copied and pasted in the UI, via the context menu. When adding drivers with the same settings, this can save time modifying settings.

## Transform Driver

This examples shows you how setup a transform driver. First make sure you are in the Front Ortho view. Numpad5, Numpad1.

1. In object mode, select then duplicate the default Cube. Shift-D. Move Cube.001 to a new location.
2. With Cube.001 selected, add a single driver to the **Rotation Y** property.
3. Open the *Graph Editor*, set the *Mode* to *Drivers*.
4. *Show Only Selected* is useful disabled for drivers, marked green in the picture.
5. In the channels region, select the **Y Euler Rotation** property.
6. Press N to open the properties region, scroll down to *Drivers* panel.
7. Change the *Type* to *Averaged Value*, this will return the averaged value of the driver variables.
8. Modify the driver variable settings.
  - *Type* - *Transform Channel*
  - *Ob/Bone* - *Cube*
  - *Transform Type* - *X Location*
  - *Transform Space* - *World Space*



When finished, `Cube.001` should rotate on the Y axis when moving Cube left of right.

## Examples

Some Driver Examples.

### Driver Expression

Here are some examples using the scripted expression Expr to set the Driver Value.

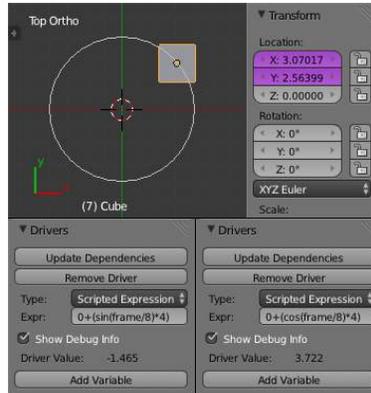


Fig. 2.1238: Object Rotation.

**Orbit a point** Here two drivers have been added to the Cube, X Location and Y Location.

The scripted expressions are being used to set the object location.

#### X Location Expr

$0 + (\sin(\text{frame} / 8) * 4)$  (`frame/8`): is the current frame of the animation, divided by 8 to slow the orbit down.  $(\sin(\ ) * 4)$ : This returns the sine of (`frame/8`), then multiplies by 4 for a bigger circle.  $0 +$ : is used to control the X Location offset of the orbit.

#### Y Location Expr

$0 + (\cos(\text{frame} / 8) * 4)$  (`frame / 8`): is the current frame of the animation, divided by 8 to slow the orbit down.  $(\cos(\ ) * 4)$ : This returns the cosine of (`frame/8`), then multiplies by 4 for a bigger circle.  $0 +$ : is used to control the Y Location offset of the orbit.

`frame` is the same as `bpy.context.scene.frame_current`.

**Driver Namespace** There is a list of built in driver functions and properties. These can be displayed via the python console.

```
>>> bpy.app.driver_namespace['
    __builtins__']
    __doc__']
    __loader__']
    __name__']
    __package__']
    acos']
    acosh']
    asin']
```

```

    asinh']
    atan']
    atan2']
    atanh']
    bpy']
    ceil']
    copysign']
    cos']
    cosh']
    ..

```

This script will add a function to the driver namespace, which can then be used in the expression `driver_func(frame)`

```

import bpy

def driver_func(val):
    return val * val    # return val squared

# add function to driver_namespace
bpy.app.driver_namespace['driver_func'] = driver_func

```

**Shape Key Driver** This example is a Shape Key Driver. The driver was added to the shape key Value.

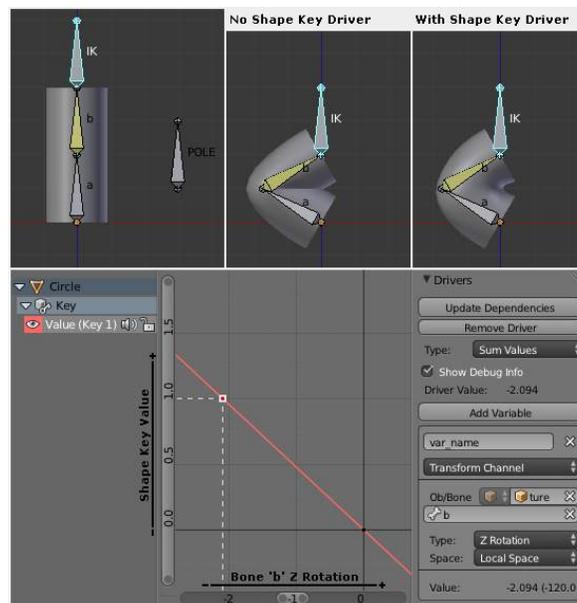


Fig. 2.1239: Shape Key Driver. Click to enlarge.

This example uses the Armature Bone 'b' Z Rotation to control the Value of a Shape Key. The bone rotation mode is set to XYZ Euler.

**The Driver F-Curve is mapped like so** Bone Z Rotation 0.0(0.0): Shape Key value 0.0 Bone Z Rotation -2.09(-120.0): Shape Key value 1.0

This kind of driver can also be setup with the Variable Type Rotational Difference.

See [Shape Keys](#) for more info.

## Drivers And Multiple Relative Shape Keys

The following screenshots illustrate combining shape keys, bones, and drivers to make multiple chained relative shape keys sharing a single root. While it lacks the convenience of the single Evaluation Time of an absolute shape key, it allows you to have more complex relationships between your shape keys.

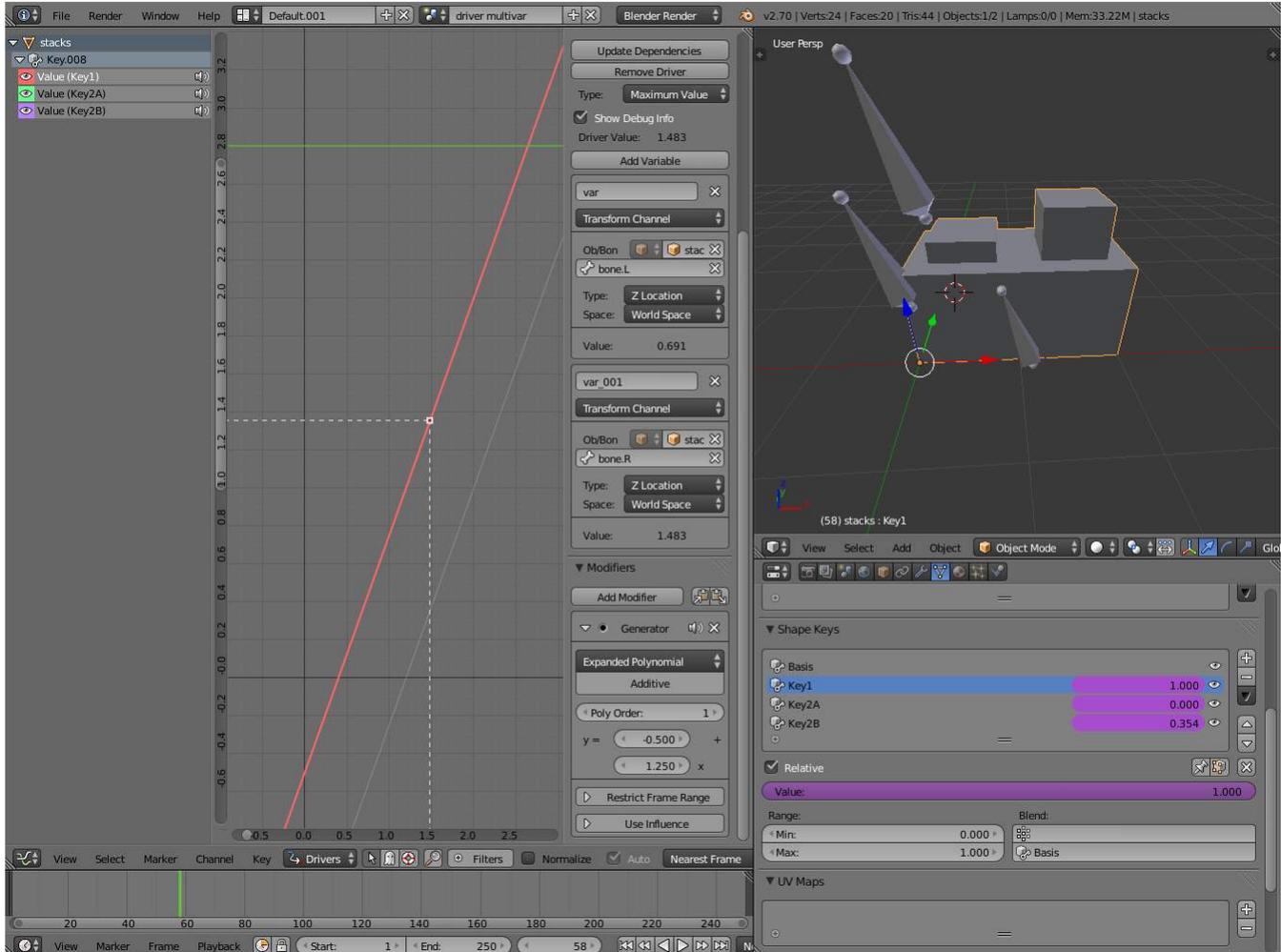


Fig. 2.1240: Key1 must handle conflicting values from the two bones

The Basis shape key has the stacks fully retracted. Key1 has the base fully extended. Key2A has the left stack fully extended. Key2B has the right stack fully extended. Key2A and Key2B are both relative to Key1 (as you can see in the field in the bottom right of the Shape Keys panel).

The value of Key1 is bound to the position of bones by a driver with two variables. Each variable uses the world Z coordinate of a bone and uses the maximum value to determine how much the base should be extended. The generator polynomial is crafted such that the top of the dominant stack should line up with the bone for that stack.

The value of Key2A is bound to the position of `Bone.L`. Its generator parameters are crafted such that when Key1's value reaches 1, the value of Key2A starts increasing beyond zero. In this way the top of the left stack will move with `bone.L` (mostly).

The value of Key2B is bound to the position of `Bone.R`. Its generator parameters are similar to Key2A so that the top of the right stack will move with `bone.R` (mostly).

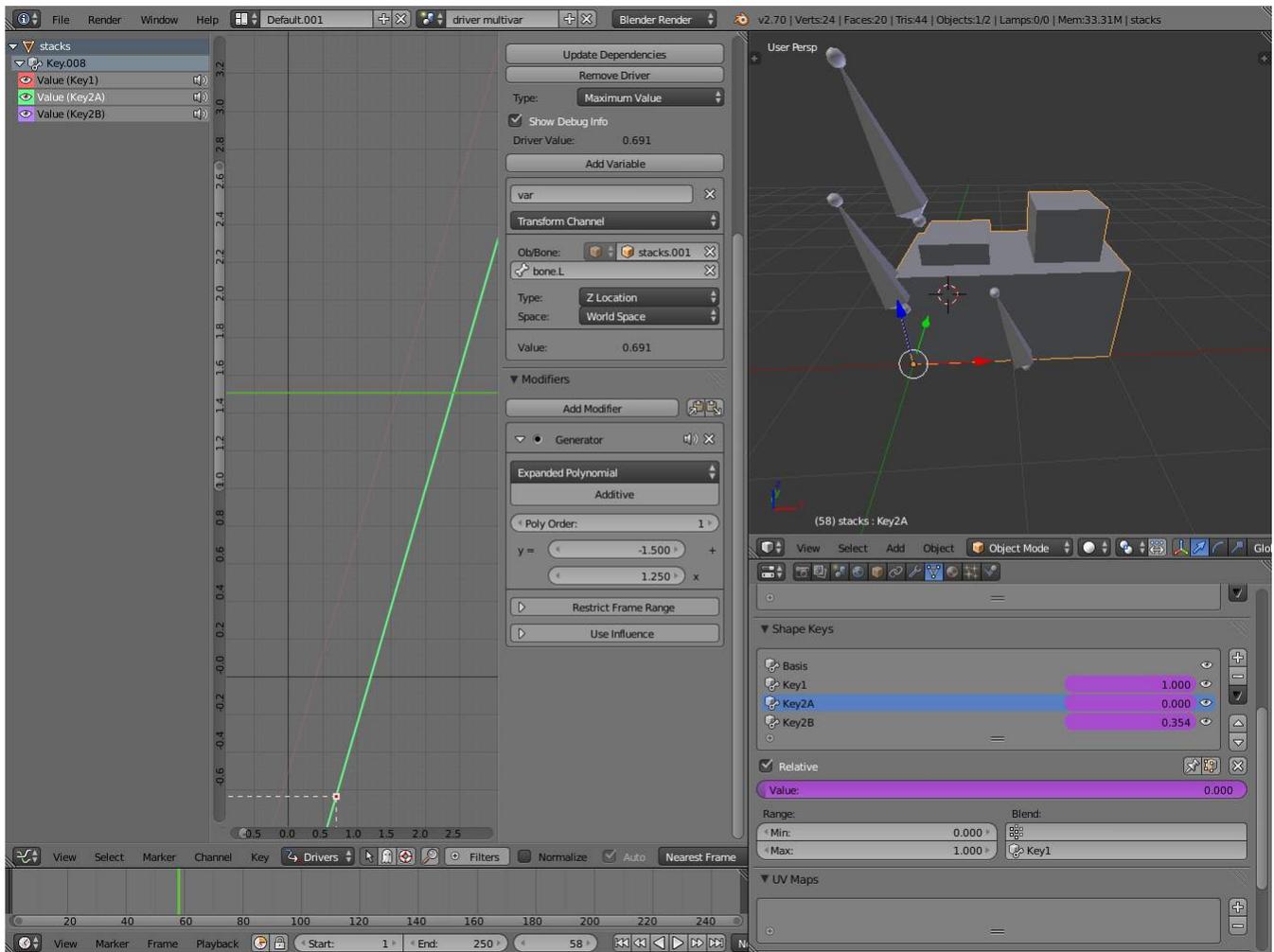


Fig. 2.1241: Key2A has different generator coefficients so it is activated in a different range of the bone's position.

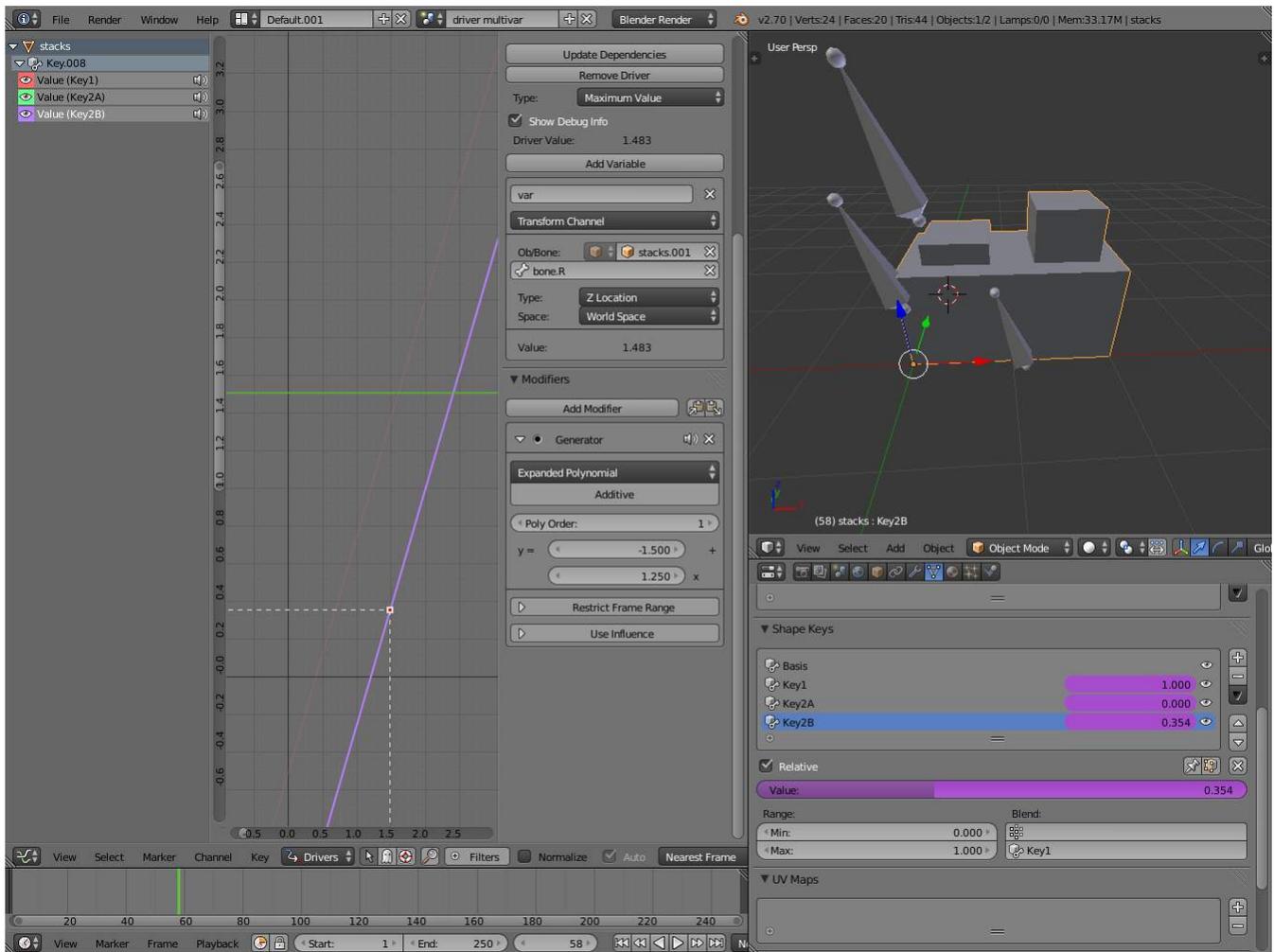


Fig. 2.1242: Key2B is the same as Key2A, but is controlled by the second bone.

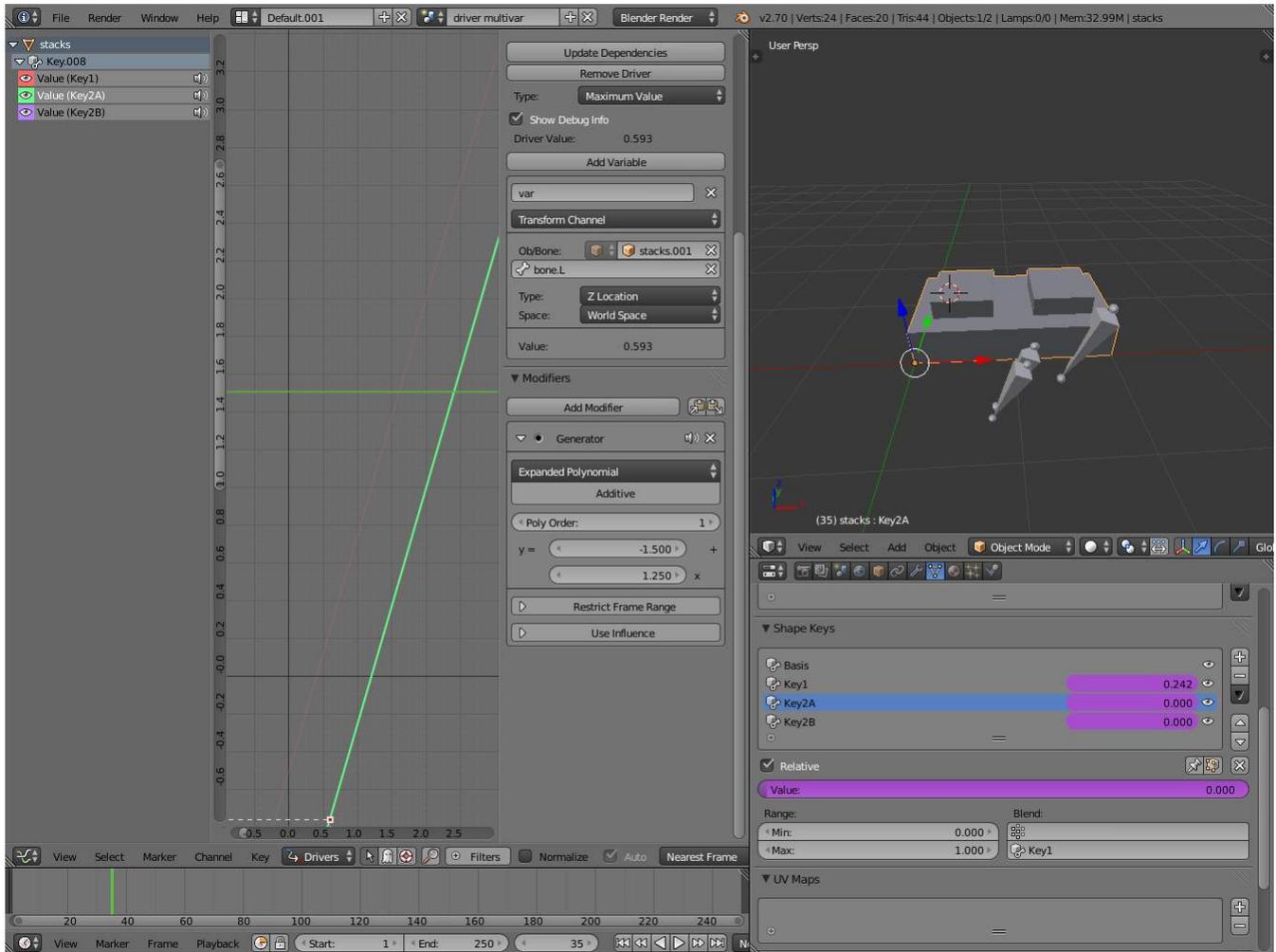
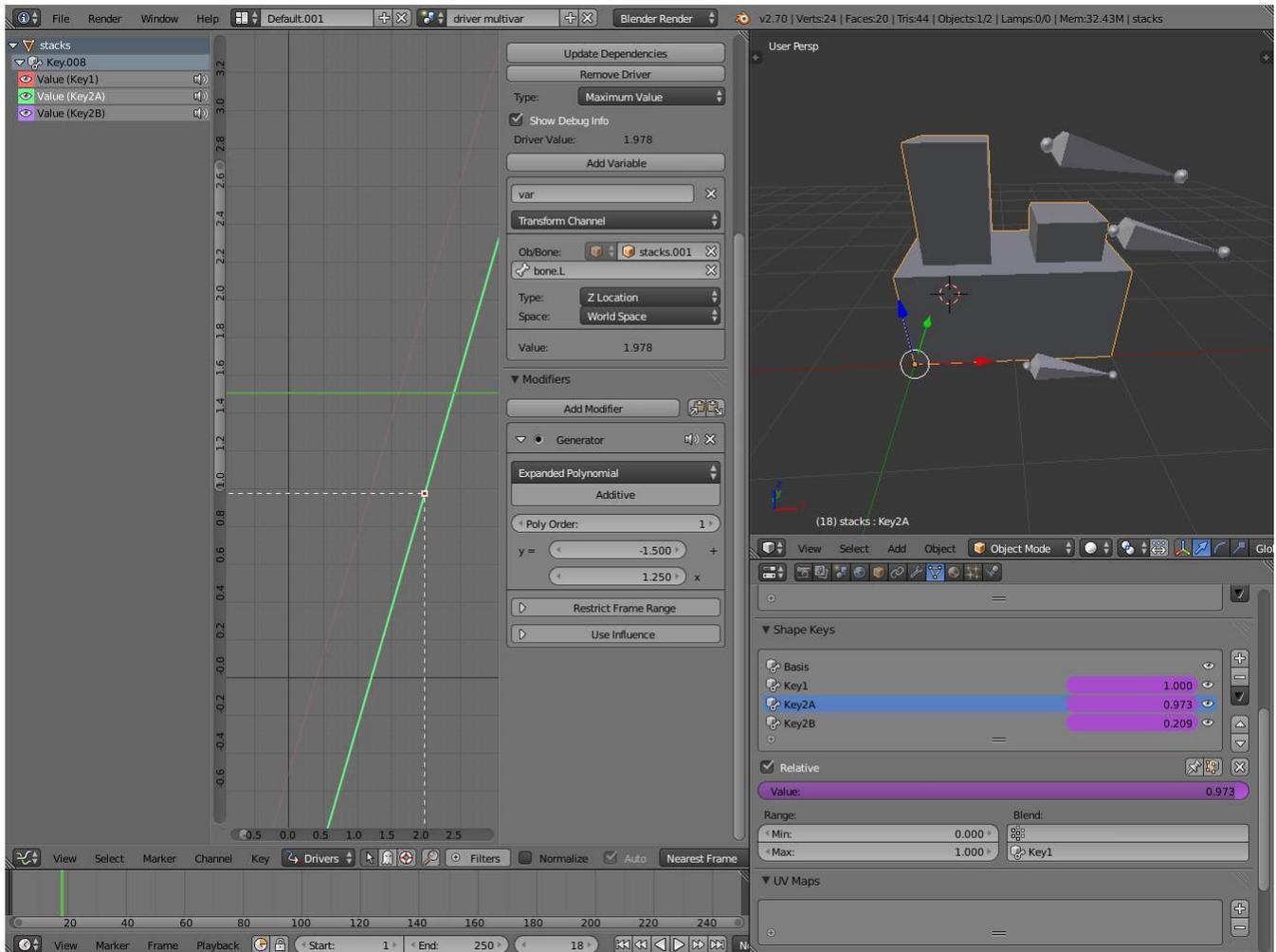


Fig. 2.1243: when both bones are low, Key2B and Key2A are deactivated and Key1 is at low influence.



Since it's quite easy for bone.L and bone.R to be in positions that indicate conflicting values for Key1 there will be times when the bones do not line up with the tops of their respective stacks. If the driver for Key1 were to use Average or Minimum instead of Maximum to determine the value of the shape key then “conflicts” between bone.L and bone.R would be resolved differently. You will chose according to the needs of your animation.

## Troubleshooting

Some common problems people may run in to when using drivers.

### Scripted Expression

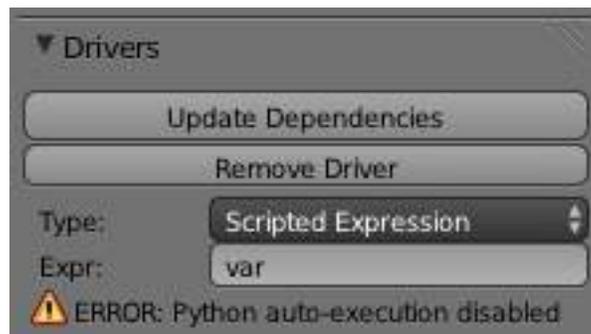


Fig. 2.1244: Graph Editor > Properties > Drivers.

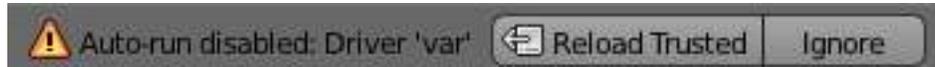


Fig. 2.1245: Info Header.

By default blender will not auto run python scripts.

If using a *Scripted Expression* Driver Type, you will have to open the file as *Trusted Source*, or set *Auto Run Python Scripts* in *User Preferences > File > Auto Execution*.

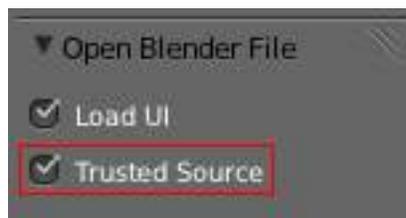


Fig. 2.1246: File Browser.

## Rotational Properties are Radians

Parts of the User Interface may use different units of measurements for angles, rotation. In the Graph Editor while working with Drivers, all angles are Radians.

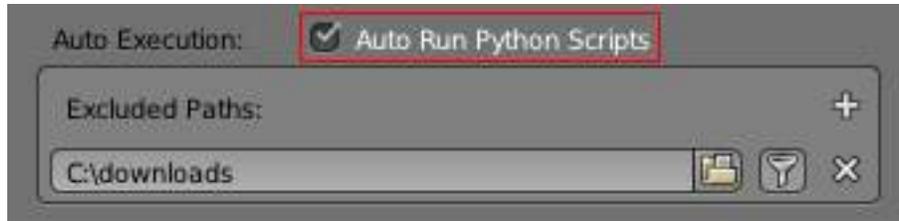


Fig. 2.1247: User Preference &gt; File &gt; Auto Execution.

### Intra-armature Bone Drivers Can Misbehave

There is a [well known limitation](#) with drivers on bones that refer to another bone in the same armature. Their values can be incorrectly calculated based on the position of the other bone as it was *before* you adjust the current\_frame. This can lead to obvious shape glitches when the rendering of frames has a jump in the frame number (either because the .blend file is currently on a different frame number or because you're skipping already-rendered frames).

### See Also

- [Animation](#)
- [Graph Editor](#)
- [F-Curves](#)
- [Extending Blender with python.](#)

### Links

- [Python and its documentation.](#)
- [functions.wolfram.com](http://functions.wolfram.com)

## 2.6.5 Keying Sets

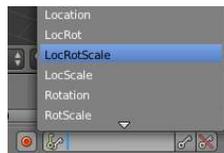


Fig. 2.1248: Timeline Keying Sets.

Keying Sets are a collection of properties. They are used to keyframe multiple properties at the same time, usually by pressing **I** in the 3D View.

There are some built in Keying Sets, and also custom Keying Sets called *Absolute Keying Sets*.

To select and use a Keying Set, set the *Active Keying Set* in the *Timeline Header*, or the *Keying Set Panel*, or press **Ctrl-Alt-Shift-I** in the 3D View.

## Keying Set Panel

This panel is used to add, select, manage *Absolute Keying Sets*.

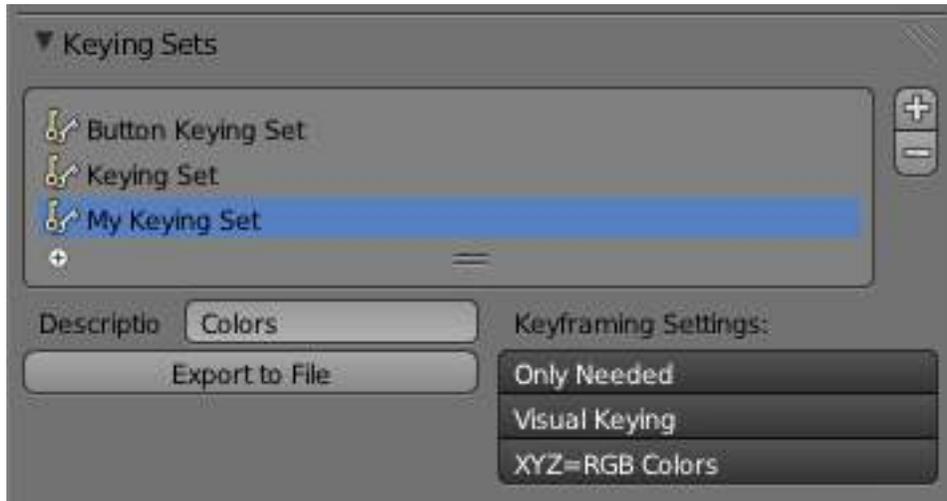


Fig. 2.1249: Properties > Scene > Keying Set Panel.

**Keying Set Name** The active Keying Set is highlighted in blue, LMB-x2 to rename.

- + Add new (Empty) keying set to the active Scene.
- Remove the active Keying Set.

### Active Keying Set properties

**Description** A short description of the keying set.

**Export to File** Export Keying Set to a python script *File.py*. To re add the keying set from the *File.py*, open then run the *File.py* from the Text Editor.

**Keyframing Settings** These options control all properties in the Keying Set. Note, the same settings in *User Preferences* override these settings if enabled.

**Only Needed** Only insert keyframes where they're needed in the relevant F-Curves.

**Visual Keying** Insert keyframes based on the visual transformation.

**XYZ=RGB Colors** For new F-Curves, set the colors to RGB for the property set, Location XYZ for example.

## Active Keying Set Panel

This panel is used to add properties to the active Keying Set.

**Paths** A collection of *Paths* each with a *Data Path* to a property to add to the active Keying Set. The active *Path* is highlighted in blue.

- Add new empty path to active Keying Set.
- Remove active path from the active Keying Set.

### Active Path properties

**ID-Block** Set the *ID-Type + Object ID Data Path* for the property.



Fig. 2.1250: Properties > Scene > Active Keying Set Panel.

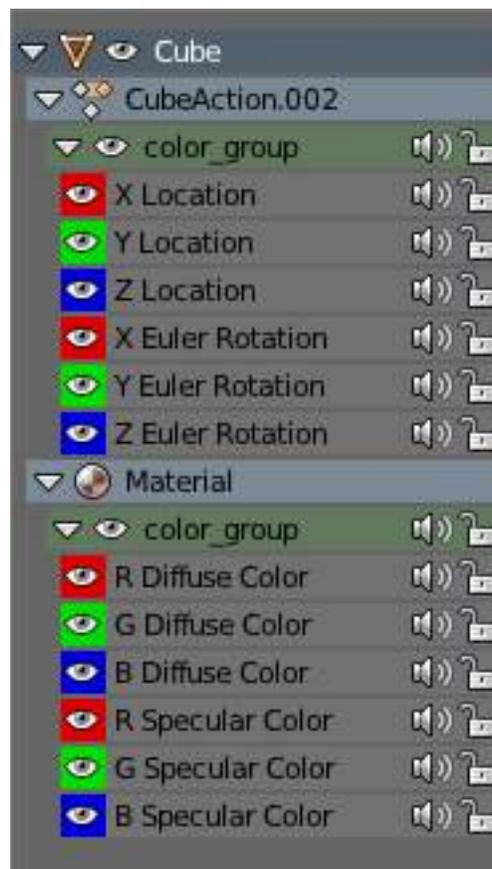


Fig. 2.1251: Properties > Graph Editor > Channels, Named Group.

**Data Path** Set the rest of the *Data Path* for the property.

**Array Target** Use *All Items* from the *Data Path* or select the array index for a specific property.

**F-Curve Grouping** This controls what *Group* to add the *Channels* to. *Keying Set Name, None, Named Group*.

**Keyframing Settings** These options control individual properties in the Keying Set.

**Only Needed** Only insert keyframes where they're needed in the relevant F-Curves.

**Visual Keying** Insert keyframes based on the visual transformation.

**XYZ=RGB Colors** For new F-Curves, set the colors to RGB for the property set, Location XYZ for example.

## Adding Properties

Some ways to add properties to keying sets.

RMB the property in the *User Interface*, then select *Add Single to Keying Set* or *Add All to Keying Set*. This will add the properties to the active keying set, or to a new keying set if none exist.

Hover the mouse over the properties, then press **K**, to add *Add All to Keying Set*.

## 2.6.6 Markers

Markers are used to denote frames at which something significant happens - it could be that a character's animation starts, the camera changes position, or a door opens, for example. Markers can be given names to make them more meaningful at a quick glance. They are available in many of Blender's windows, under different forms. Unlike the keyframes, markers are always placed at a whole frame number, you cannot e.g. set a marker at "frame 2.5".

Markers can be created and edited in all of the following editors (including their different modes):

- The [Graph Editor Window](#).
- The [Action Editor window](#).
- The [The Dope Sheet](#).
- The [NLA Editor window](#).
- The [Video Sequence Editor window](#).
- The [Timeline window](#). When you create

A marker created in one of these windows will also appear in all others that support them, including:

- The [3D View window](#).

### Pose markers

There is another type of markers, called "pose markers", which are specific to the armatures and the Action Editor window. They are related to the pose libraries, and are discussed in detail [here](#).



Fig. 2.1252: Markers: small but useful.

## Visualization

### Standard

Most of the window types visualize markers the same way: as small triangles at their bottom, white if unselected or yellow if selected.

If they have a name, this is shown to their right, in white when the marker is selected. See (Markers: small but useful).

### Sequencer



Fig. 2.1253: Markers in the Sequencer

The **Video Sequence Editor** just adds a vertical dashed line to each marker (gray if the marker is unselected, or white if it's selected).

### 3D View



Fig. 2.1254: Marker in a 3D View.

The View do not allow you to create/edit/remove markers, they just show their name between <> at there bottom left corner, near the active object's name, when you are at their frame (see Marker in a 3D view).

## Creating and Editing Markers

Unfortunately, there is no common shortcuts and menu for marker's editing, across the different window types that supports them... So in the refboxes of each action described below, I put the most-common shortcut and menu entry, with the known

exceptions between brackets.

## Creating Markers

---

### Reference

Mode: all modes

Menu: *Marker* → *Add Marker* (*Frame* → *Add Marker* in a timeline)

Hotkey: M (Ctrl-Alt-M in a VSE)

---

The simplest way to add a marker is to move to the frame where you would like it to appear, and press M (or Ctrl-Alt-M in a video sequence editor).

Alternatively, you can press Alt-A (or the “playback” button of the *Timeline* window) to make the animation play, and then press M (or Ctrl-Alt-M in VSE) at the appropriate points. This can be especially useful to mark the beats in some music.

## Selecting Markers

---

### Reference

Mode: all modes

Hotkey: RMB, Shift-RMB, A / Ctrl-A, B / Ctrl-B

---

Click RMB on the marker’s triangle to select it. Use Shift-RMB to (de)select multiple markers.

In the Ipo Curve Editor, Action Editor, NLA Editor and Video Sequence Editor windows, you can also (de)select all markers with Ctrl-A, and border-select them with Ctrl-B (as usual, LMB to select, RMB to deselect). The corresponding options are found in the Select menu of these windows.

In the Timeline and Audio windows, you can (de)select all markers with A, and border (de)select them with B ...

## Naming Markers

---

### Reference

Mode: all modes

Menu: *Marker* → *(Re)Name Marker* (*Frame* → *Name Marker* in a timeline)

Hotkey: Ctrl-M

---

Having dozens of markers scattered throughout your scene’s time won’t help you much unless you know what they stand for. You can name a marker by selecting it, pressing Ctrl-M, typing the name, and pressing the OK button.

---

## Moving Markers

---

### Reference

Mode: all modes

Menu: *Marker* → *Grab/Move Marker* (*Frame* → *Grab/Move Marker* in a timeline)

Hotkey: `Ctrl-G` (`G` in a timeline or audio)

---

Once you have one or more markers selected, press `Ctrl-G` (or `G` in Timeline or Audio windows) to move them, and confirm the move with `LMB` or `Return` (as usual, cancel the move with `RMB`, or `Esc`).

By default, you grab the markers in one-frame steps, but if you hold `Ctrl`, the markers will move in steps corresponding to one second - so if you have set your scene to **25 fps**, the markers will move in twenty-five-frames steps.

## Duplicating Markers

---

### Reference

Mode: all modes

Menu: *Marker* → *Duplicate Marker* (*Frame* → *Duplicate Marker* in a timeline)

Hotkey: `Ctrl-Shift-D` (`Shift-D` in a timeline or audio)

---

You can duplicate the selected markers by pressing `Ctrl-Shift-D` (or `Shift-D` in a Timeline or Audio window). Once duplicated, the new ones are automatically placed in grab mode, so you can move them where (or rather when) you want.

Note that unlike most other duplications in Blender, the names of the duplicated markers are not altered at all (no `.001` numeric counter append...).

## Deleting Markers

---

### Reference

Mode: all modes

Menu: *Marker* → *Delete Marker* (*Frame* → *Delete Marker* in a timeline)

Hotkey: `Shift-X` (`X` in a timeline or audio)

---

To delete the selected marker(s) simply press `Shift-X` (or `X` in a Timeline or Audio window), and confirm the pop-up message with `LMB`.

## 2.6.7 Keyframe Visualization

There are some important visualization features in the 3D views that can help animation.

## Keyframe Visualization

When the current frame is a keyframe for the current active object, the name of this object (shown in the bottom left corner of the 3D views) turns yellow.

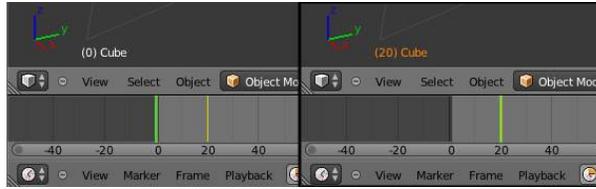


Fig. 2.1255: Left: Current frame at 0. Right: Current frame is a keyframe for Cube

## Motion Paths

### Reference

Mode: *Object* mode

Panel: *Object*

This feature allows you to visualize the animation of objects by displaying their position over a series of frames.

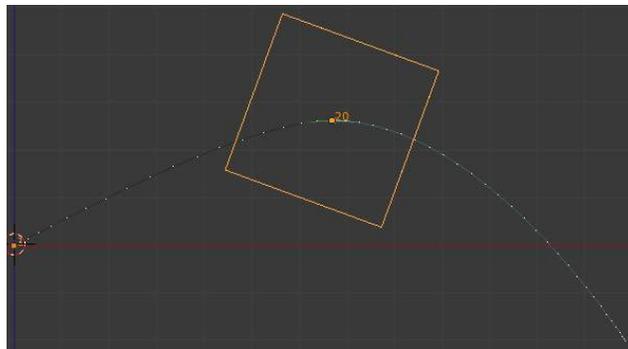


Fig. 2.1256: An animated cube with its motion path displayed

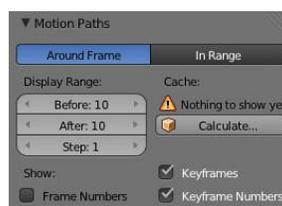


Fig. 2.1257: Motion paths panel

Before we look at its options (all regrouped in the same *Visualisations* panel, in the *Editing* context, let's first see how to display/hide these paths. You have to do it manually - and you have to first select the objects you want to show/hide the motion

paths. Then,

- To show the paths (or update them, if needed), click on the *Calculate Path* button.
- To hide the paths, click on the *Clear Paths* button

Remember: only selected object and their paths are affected by these actions!

The paths are drawn in black with white dots indicating frames, and a blue glow around the current frame.

## Options

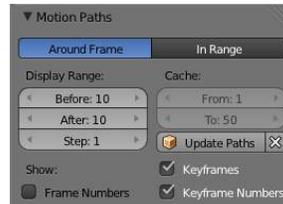


Fig. 2.1258: The Motion Paths Panel set to “Around Frame”

**Around Frame** Around Frame, Display Paths of poses within a fixed number of frames around the current frame. When you enable this button, you rather get paths for a given number of frames before and after the current one (again, as with ghosts).

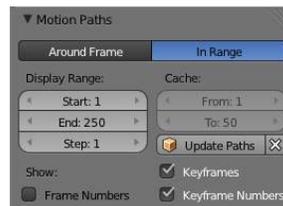


Fig. 2.1259: The Motion Paths Panel set to “In Range”

**In Range** In Range, Display Paths of poses within specified range.

### Display Range

**Before/After** Number of frames to show before and after the current frame (only for ‘Around Current Frame’ Onion-skinning method)

**Start/End** Starting and Ending frame of range of paths to display/calculate (not for ‘Around Current Frame’ Onion-skinning method)

**Step** This is the same thing as the *GStep* for ghosts - it allows you the only materialize on the path one frame each  $n$  ones. Mostly useful when you enable the frame number display (see below), to avoid cluttering the 3D views.

**Frame Numbers** When enabled, a small number appears next to each frame dot on the path, which is of course the number of the corresponding frame...

**Keyframes** When enabled, big yellow square dots are drawn on motion paths, materializing the keyframes of their bones (i.e. only the paths of keyed bones at a given frame get a yellow dot at this frame).

**Keyframe Numbers** When enabled, you’ll see the numbers of the displayed keyframes - so this option is obviously only valid when *Show Keys* is enabled.

### Cache

**From / To** These are the start/end frames of the range in which motion paths are drawn. You cannot modify this range without deleting the motion path first.

**Calculate Paths/ Update Paths** If no paths have been calculated, Calculate Paths will create a new motion path in cache. In the pop up box, select the frame range to calculate. If a path has already been calculated, Update Paths will update the path shape to the current animation. To change the frame range of the calculated path, you need to delete the path and calculate it again.

## 2.6.8 Animation Editors

### The Dopesheet



Fig. 2.1260: The DopeSheet

Classical hand-drawn animators often made a chart, showing exactly when each drawing, sound and camera move would occur, and for how long. They nicknamed this the ‘dopesheet’. While CG foundations dramatically differ from classical hand-drawn animation, Blender’s Dopesheet inherits a similar directive. It gives the animator a ‘birds-eye-view’ of every thing occurring within a scene.

### Dope Sheet Modes

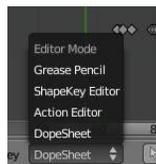


Fig. 2.1261: DopeSheet modes

There are four basic views for the Dopesheet. These all view different contexts of animation:

**DopeSheet** The DopeSheet allows you to edit multiple actions at once.

**Action Editor** *Action Editor* is the default, and most useful one. It’s here you can define and control your actions.

**Shape Key Editor** *ShapeKey Editor* is dedicated to the *Shape* Ipo datablocks. It uses/edits the same action datablock as the previous mode. It seems to be an old and useless thing, as the *Action Editor* mode handles *Shape* channels very well, and this mode adds nothing...

**Grease Pencil** *Grease Pencil* is dedicated to the *grease pencil tool*’s keyframes - for each grease pencil layer, you have a strip along which you can grab its keys, and hence easily re-time your animated sketches. As it is just another way to see and

edit the grease pencil data, this mode uses no datablock (and hence has nothing to do with actions...). Note that you'll have as much top-level grease pencil channels as you have sketched windows (3D views, *UV/Image Editor*, etc.)

## Interface

The *Action Editor* interface is somewhat similar to the *FCurve Editor* one, it is divided in three areas:

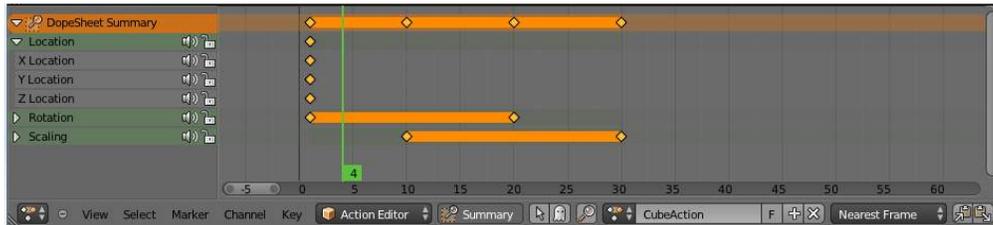


Fig. 2.1262: The Action Editor window, Action Editor mode, with an Object and Shape channels.

**The header bar** Here you find the menus, a first block of controls related to the editor “mode”, a second one concerning the action datablocks, and a few other tools (like the copy/paste buttons, and snapping type).

**The main area** It contains the keyframes for all visible action channels. As with the other “time” windows, the X-axis materializes the time. The Y-axis has no mean in itself, unlike with the *FCurve* editor, it’s just a sort of “stack” of action channels - each one being shown as an horizontal colored strip (of a darker shade “during” the animated/keyed period). On these channel strips lay the keyframes, materialized as light-gray (unselected) or yellow (selected) diamonds. One of the key feature of this window is that it allow you to visualize immediately which channel (i.e. *Ipo* curve) is *really* affected. When the value of a given channel does not change at all between two neighboring keyframes, a gray (unselected) or yellow (selected) line is drawn between them.

**The left “list-tree”** This part shows the action’s channel “headers” and their hierarchy. Basically, there are:

- “Top-level” channels, which represent whole *FCurve* datablocks (so there’s one for *Object* one, one for *Shape* one, etc.). They gather *all* keyframes defined in their underlying *FCurve* datablock.
- “Mid-level” channels, which seem currently to have no use (there’s one per top-level channel, they are all named *FCurves*, and have no option at all...).
- “Low-level” channels, which represent individual *FCurve*, with their own keyframes (fortunately, only keyed *Ipos* are shown!).

Each level can be expanded/collapsed by the small arrow to the left of its “parent” channel. To the right of the channel’s headers, there are some channel’s setting controls:

- Clicking on the small “eye” will allow you to mute that channel (and all its “children” channels, if any!).
- Clicking on the small “lock” will allow you to prevent this channel and its children to be edited (note that this is also working inside the *NLA*, but that it doesn’t prevent edition of the underlying *FCurve* ...).

A channel can be selected (text in white, strip in gray-blue color) or not (text in black, strip in pink-brown color.), use *LMB* clicks to toggle this state. You can access some channel’s properties by clicking *Ctrl-LMB* on its header. Finally, you can have another column with value-sliders, allowing you to change the value of current keyframes, or to add new ones. These are obviously only available for low-level channels (i.e. individual *FCurve*). See [View Menu](#) below for how to show these sliders.

## View Menu

**Realtime Updates** When transforming keyframes, changes to the animation data are flushed to other views

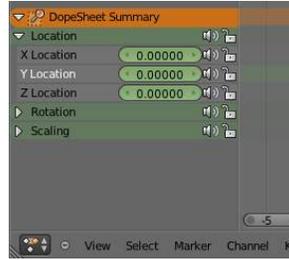


Fig. 2.1263: the action editor showing sliders

**Show Frame Number Indicator** Show frame number beside the current frame indicator line

**Show Sliders** A toggle option that shows the value sliders for the channels. See the *The Action Editor window, Action Editor mode, with a group and sliders* picture above).

**Use Group Colors** Draw groups and channels with colors matching their corresponding groups.

**AutoMerge Keyframes** Automatically merge nearby keyframes

**Sync Markers** Sync Markers with keyframe edits

**Show Seconds** Whether to show the time in the X-axis as frames or as seconds

**Set Preview Range P** Interactively define frame range used for playback. Allow you to define a temporary preview range to use for the `Alt-A` realtime playback (this is the same thing as the *Playback Range* option of the *timeline window header*).

**Clear Preview Range Alt-P** Clears the preview range

**Auto-Set Preview Range** Automatically sets the preview range to playback the whole action.

**Marker Menu** See the [Markers](#) page.

## Action Editor

In Blender *Actions* are a generic containers for F-Curves. Actions can contain any number of F-Curves, and can be attached to any data block. As long as the RNA data paths stored in the Action's F-Curves can be found on that data block, the animation will work. For example, an action modifying 'X location' and 'Y location' properties can be shared across multiple objects, since both objects have 'X location' and 'Y location' properties beneath them.

The *Action Editor* window enables you to see and edit the FCurve datablocks you defined as actions in the *FCurve Editor* window. So it takes place somewhere in-between the low-level [FCurves](#), and the high-level [NLA editor](#). Hence, you do not have to use them for simple Ipo curves animations - and they have not much interest in themselves, so you will mostly use this window when you do [NLA animation](#) (they do have a few specific usages on their own, though, like e.g. with the [Action constraint](#), or the [pose libraries](#)).

This is not a mandatory window, as you do can edit the actions used by the NLA directly in the *FCurve Editor* window (or even the *NLA Editor* one). However, it gives you a slightly simplified view of your FCurve datablocks (somewhat similar to the "key" mode of the FCurve window, even though more powerful in some ways) - and, more interesting, it can show you all "action" FCurve datablocks of a same object at once.

Additionally, it also allows you to affect timing of the different keys of the layers created with the [grease pencil tool](#).

Each "action" FCurve datablock forms a top-level channel (see below). Note that an object can have several *Constraint* (one per animated constraint) and *Pose* (for armatures, one per animated bone) FCurve datablocks, and hence an action can have several of these channels.

## Action Datablocks

As everything else in Blender, actions are datablocks. Unlike F-Curve ones, there is only one type of action, which can regroup all F-Curves of a given object. You'll find its usual datablock controls in the *Action Editor* header.

However, there is one specificity with action datablocks: they have by default a “fake user”, i.e. once created, they are always kept in Blender file, even if no object uses them. This is due to the fact that actions are designed to be used in the NLA, where you can affect several different actions to a same object! Yes, this is the only way to use different actions (and hence, different F-Curve datablocks of the same kind) to animate a same object. But as you have to assign an action to an object to be able to edit it (and an object can only have one action datablock at a time), to have “fake users” guarantees you that you won't lose your precious previously-edited actions when you start working on a new one!

This window shows, by default, the action datablock linked to the current active object. However, as with F-Curves, you can pin an *Action Editor* to a given action with the small “pin” button to the left of the datablock controls, in the header. This will force the window to always display this datablock, whatever the current selected object is.

## Channel Menu

### Delete (X)

Deletes the whole channel from the current action (i.e. unlink the underlying F-Curve datablock from this action datablock).

**Warning:** The X shortcut is area-dependent: if you use it in the left list part, it'll delete the selected channels, whereas if you use it in the main area, it'll delete the selected keyframes...

**Settings** → *Toogle/Enable/Disable a Setting (Shift-W / Ctrl-Shift-W / Alt-W)* Enable/disable a channel's setting (selected in the menu that pops-up) - currently, “lock” and/or “mute” only.

**Toggle Channel Editability Tab** Locks or unlocks a channel for editing

**Extrapolation Mode** Change the extrapolation between selected keyframes. More options are available in the Graph Editor.

**Expand Channels, Collapse Channels (NumpadPlus, NumpadMinus)** Expands or collapses selected channels.

**Move...** This allows you to move top-level channels up/down (Shift-PageUp / Shift-PageDown), or directly to the top/bottom (Ctrl-Shift-PageUp / Ctrl-Shift-PageDown).

**Revive Disabled F-Curves** Clears ‘disabled’ tag from all F-Curves to get broken F-Curves working again

## Dope Sheet

## Grease Pencil

## Shape Key

## Graph Editor

The graph editor is the main animation editor. It allows you to modify the animation for any properties using F-Curves.

The graph editor has two modes, *F-Curve* for *Actions*, and *Drivers* for *Drivers*. Both are very similar in function.

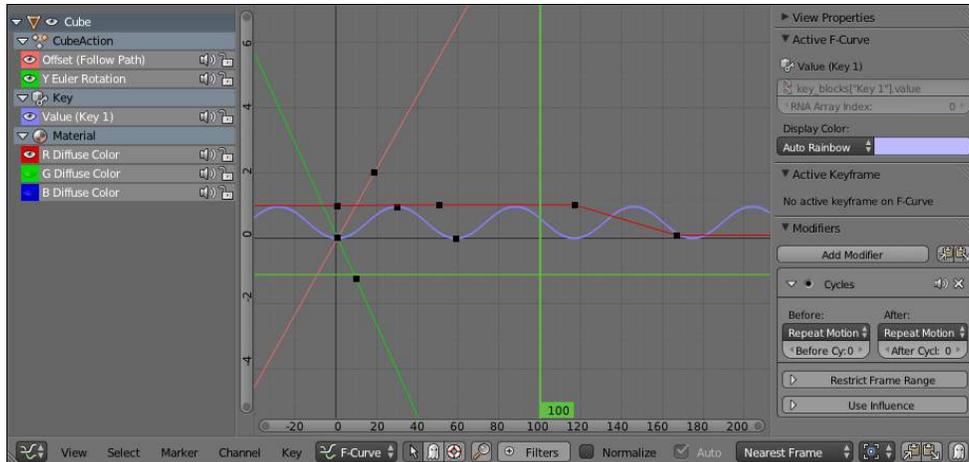


Fig. 2.1264: The Graph Editor.

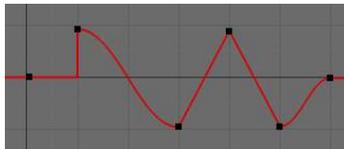


Fig. 2.1265: A curve with different types of interpolation.

### Curve Editor Area

Here you can see and edit the curves and keyframes.

See [F-Curves](#) for more info.

**Navigation** As with most windows, you can:

**Pan MMB** Pan the view vertically (values) or horizontally (time) with click and drag.

**Zoom Wheel** Zoom in and out with the mouse wheel.

**Scale View Ctrl-MMB** Scale the view vertically or horizontally.

These are some other useful tools.

**View All Home** Reset viewable area to show all keyframes.

**View Selected NumpadPeriod** Reset viewable area to show selected keyframes.

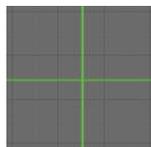


Fig. 2.1266: Graph Editor 2D Cursor.

**2D Cursor** The current frame is represented by a green vertical line called the *Time Cursor*.

As in the [Timeline](#), you can change the current frame by pressing or holding **LMB**.

The green horizontal line is called the *Cursor*. This can be disabled via the *View Menu* or the *View Properties* panel.

The *Time Cursor* and the *Cursor* make the *2D Cursor*. The *2D Cursor* mostly used for editing tools.

**View Axes** For *Actions* the X-axis represents time, the Y-axis represents the value to set the property.

For *Drivers* the X-axis represents the *Driver Value*, the Y-axis represents the value to set the property.

Depending on the selected curves, the values have different meaning: For example rotation properties are shown in degrees, location properties are shown in Blender Units. Note that *Drivers* use radians for rotation properties.

**Markers** Like with most animation editors, markers are shown at the bottom of the editor.



Fig. 2.1267: Graph Editor Markers.

*Markers* can be modified in the *Graph Editor* though its usually best to use the *Timeline*.

See [Markers](#) for more info.

## Header

Here you'll find.

- The menus.
- Graph Editor mode.
- View controls.
- Curve controls.



Fig. 2.1268: Graph Mode.

## Header Controls

**Mode** F-Curve for [Actions](#), and Drivers for [Drivers](#).



Fig. 2.1269: View Controls.

## View controls

**Show Only Selected** Only include curves related to the selected objects and data.

**Show Hidden** Include curves from objects/bones that are not visible.

**Show Only Errors** Only include curves that are disabled or have errors.

**Search Filter** Only include curves with keywords contained in the search text.

**Type Filter** Filter curves by property type.

**Normalize** Normalize curves so the maximum or minimum point equals 1.0 or -1.0.

**Auto** Automatically recalculate curve normalization on every curve edit.



Fig. 2.1270: Curve Controls.

### Curve controls

**Auto Snap** Auto snap the keyframes for transformations.

*No Auto-Snap Time Step Nearest Frame Nearest Marker*

**Pivot Point** Pivot point for rotation.

**Bounding Box Center** Center of the select keyframes.

**2D Cursor** Center of the *2D Cursor*. *Time Cursor + Cursor*.

**Individual Centers** Rotate the selected keyframe *Bezier* handles.

**Copy Keyframes **Ctrl-C**** Copy the selected keyframes to memory.

**Paste Keyframes **Ctrl-V**** Paste keyframes from memory to the current frame for selected curves.

**Create Snapshot** Creates a picture with the current shape of the curves.

### Channels Region

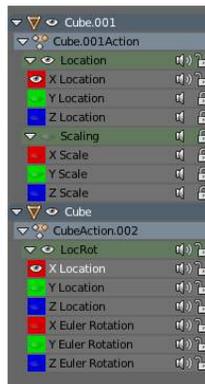


Fig. 2.1271: Channels Region.

The channels region is used to select and manage the curves for the graph editor.

**Hide curve** Represented by the eye icon.

**Deactive/Mute curve** Represented by the speaker icon.

**Lock curve from editing** Represented by the padlock icon.

**Channel Editing** *Select channel* LMB

*Multi Select/Deselect* Shift-LMB

*Toggle Select All* A

*Border Select* (LMB drag) or B (LMB drag)

*Border Deselect* (Shift-LMB drag) or B (Shift-LMB drag)

*Delete selected* X or Delete

*Lock selected* Tab

*Make only selected visible* V

*Enable Mute Lock selected* Shift-Ctrl-W

*Disable Mute Lock selected* Alt-W

*Toggle Mute Lock selected* Shift-W

## Properties Region

The panels in the *Properties Region*.

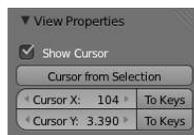


Fig. 2.1272: View Properties Panel.

## View Properties Panel

**Show Cursor** Show the vertical *Cursor*.

**Cursor from Selection** Set the *2D cursor* to the center of the selected keyframes.

**Cursor X** *Time Cursor X* position.

**To Keys** Snap selected keyframes to the *Time Cursor*.

**Cursor Y** Vertical *Cursor Y* position.

**To Keys** Snap selected keyframes to the *Cursor*.

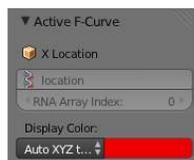


Fig. 2.1273: Active F-Curve Panel.

**Active F-Curve Panel** This panel displays properties for the active *F-Curve*.

**Channel Name (X Location)** *ID Type* + Channel name.

**RNA Path** *RNA Path* to property + Array index.

**Color Mode** *Color Mode* for the active *F-Curve*.

**Auto Rainbow** Increment the *HUE* of the *F-Curve* color based on the channel index.

**Auto XYZ to RGB** For property sets like location xyz, automatically set the set of colors to red, green, blue.

**User Defined** Define a custom color for the active *F-Curve*.



Fig. 2.1274: Active Keyframe Panel.

### Active Keyframe Panel

**Interpolation** Set the forward interpolation for the active keyframe.

**Constant** Keep the same value till the next keyframe.

**Linear** The difference between the next keyframe.

**Bezier** Bezier interpolation to the next keyframe.

### Key

**Frame** Set the frame for the active keyframe.

**Value** Set the value for the active keyframe.

**Left Handle** Set the position of the left interpolation handle for the active keyframe.

**Right Handle** Set the position of the right interpolation handle for the active keyframe.



Fig. 2.1275: Drivers Panel.

**Drivers Panel** See [Drivers Panel](#) for more info.



Fig. 2.1276: Modifiers Panel.

**Modifiers Panel** See [F-Modifiers](#) for more info.

## See Also

- Graph Editor - F-Curves
- Graph Editor - F-Modifiers
- Actions
- Drivers

## F-Curves

Once you have created keyframes for something, you can edit their corresponding curves. In Blender 2.5, IPO Curves have been replaced by FCurves, however, editing these curves is essentially still the same.

### The concept of Interpolation

When something is “animated,” it changes over time. In Blender, animating an object means changing one of its properties, such as its X location, or the Red channel value of its material diffuse color, and so on, during a certain amount of time.

As mentioned, Blender’s fundamental unit of time is the “frame”, which usually lasts just a fraction of a second, depending on the *frame rate* of the scene.

As animation is composed of incremental changes spanning multiple frames, usually these properties ARE NOT manually modified *frame by frame*, because:

- it would take ages!
- it would be very difficult to get smooth variations of the property (unless you compute mathematical functions and type a precise value for each frame, which would be crazy).

This is why nearly all direct animation is done using **interpolation**.

The idea is simple: you define a few Key Frames, which are multiple frames apart. Between these keyframes, the properties’ values are computed (interpolated) by Blender and filled in. Thus, the animators’ workload is significantly reduced.

For example, if you have:

- a control point of value 0 at frame 0,
- another one of value 10 at frame 25,
- linear interpolation,

then, at frame 5 we get a value of 2.

The same goes for all intermediate frames: with just two points, you get a smooth growth from 0 to 10 along the **25 frames**. Obviously, if you’d like the frame 15 to have a value of 9, you’d have to add another control point (or keyframe)...

## Settings

F-curves have three additional properties, which control the interpolation between points, extension behavior, and the type of handles.

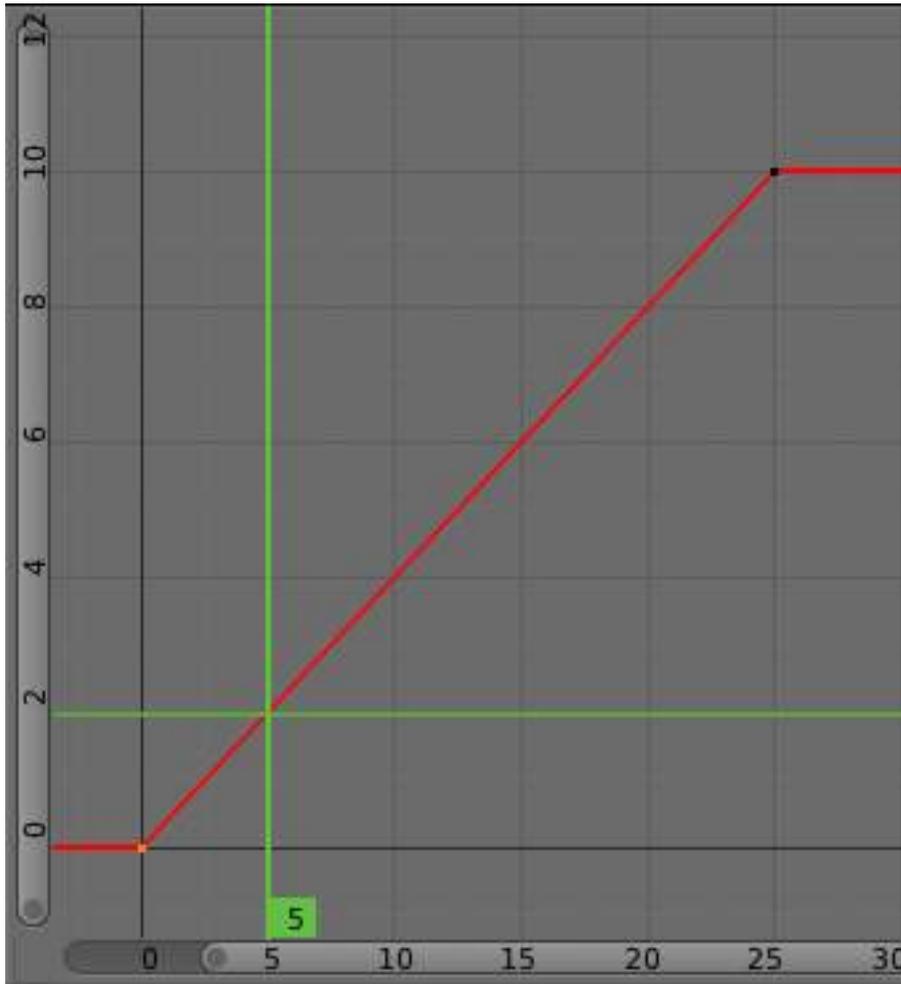


Fig. 2.1277: Example of interpolation



Fig. 2.1278: Constant.

**Interpolation Mode** You have three choices (T, or *Curve* → *Interpolation Mode*):

**Constant** There is no interpolation at all. The curve holds the value of its last keyframe, giving a discrete (stairway) “curve”. Usually only used during the initial “blocking” stage in pose-to-pose animation workflows.

**Linear** This simple interpolation creates a straight segment between each neighbor keyframes, giving a broken line. It can be useful when using only two keyframes and the *Extrapolation* extend mode, to easily get an infinite straight line (i.e. a linear curve).



Fig. 2.1279: Linear.

**Bezier** The more powerful and useful interpolation, and the default one. It gives nicely smoothed curves, i.e. smooth animations!

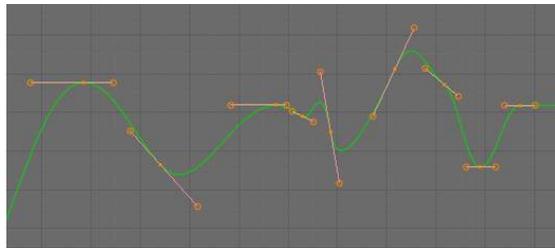


Fig. 2.1280: Bézier.

Remember that some FCurves can only take discrete values, in which case they are always shown as if constant interpolated, whatever option you chose.

**Extrapolation** (*Shift-E*, or *Channel* → *Extrapolation Mode*)

Extrapolation defines the behavior of a curve before the first and after the last keyframes.

There are two basic extrapolation modes:

**Constant** The default one, curves before their first keyframe and after their last one have a constant value (the one of these first and last keyframes).

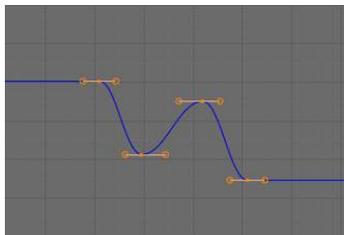


Fig. 2.1281: Constant extrapolation

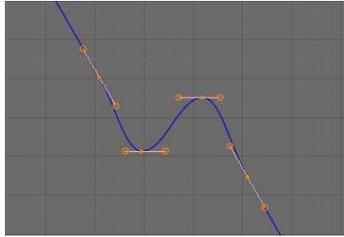


Fig. 2.1282: Linear extrapolation

**Linear** Curves ends are straight lines (linear), as defined by their first two keyframes (respectively their last two keyframes). Additional extrapolation tools (e.g. the “Cycles” F-Modifier) are located in the [F-Curve Modifiers](#)

**Handle Types** There is another curve option quite useful for Bézier-interpolated curves. You can set the type of handle to use for the curve points  $\nabla$

**Automatic** Keyframes are automatically interpolated

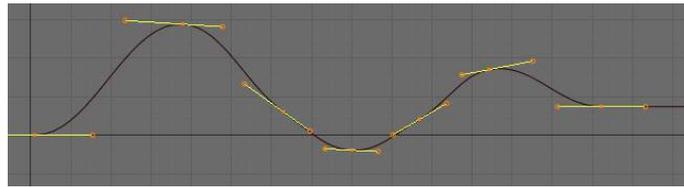


Fig. 2.1283: Auto handles

**Vector** Creates linear interpolation between keyframes. The linear segments remain if keyframe centers are moved. If handles are moved, the handle becomes Free.



Fig. 2.1284: Vector handles

**Aligned** Handle maintain rotation when moved, and curve tangent is maintained

**Free** Breaks handles tangents

**Auto Clamped** Auto handles clamped to not overshoot

### Direction of time

Although F-curves are very similar to *Bezier Curves*, there are some important differences.

For obvious reasons, **a property represented by a Curve cannot have more than one value at a given time**, hence:

- when you move a control point ahead of a control point that was previously ahead of the point that you are moving, the two control points switch their order in the edited curve, to avoid that the curve goes back in time
- for the above reason, it's impossible to have a closed Ipo curve

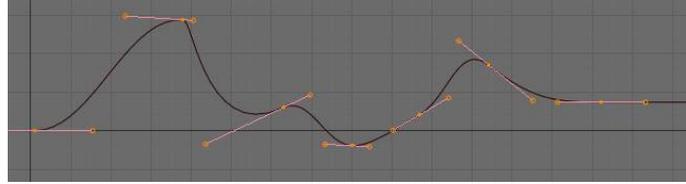


Fig. 2.1285: Aligned handles

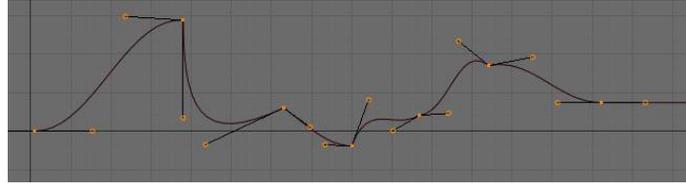


Fig. 2.1286: Free handles

Table  
2.47:  
After  
mov-  
ing the  
second  
keyframe



### Editing Tools

By default, when new channels are added, the *Graph Editor* sets them to *Edit Mode*. Selected channels can be locked by pressing `Tab`.

Many of the hotkeys are the same as the viewport ones, for example:

- `G` to grab
- `R` to rotate
- `S` to scale
- `B` for border select/deselect

And of course you can lock the transformation along the X (time frame) or Y (value) axes by pressing `X` or `Y` during transformation.

For precise control of the keyframe position and value, you can set values in the *Active Keyframe* of the Properties Region.

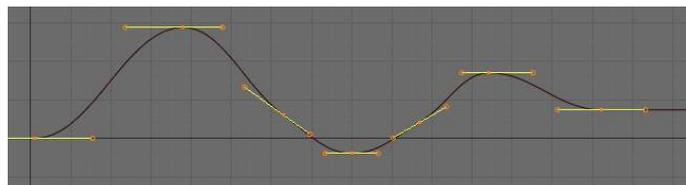


Fig. 2.1287: Auto clamped handles

**Transform Snapping** When transforming keyframes with G, R, S, the transformation can be snapped to increments.

Snap Transformation to 1.0 `Ctrl`

Divide Transformation by 10.0 `Shift`

Keyframes can be snapped to different properties by using the *Snap Keys* tool.

#### Snap Keys `Shift-S`

**Current Frame** Snap the selected keyframes to the *Time Cursor*.

**Cursor Value** Snap the selected keyframes to the *Cursor*.

**Nearest Frame** Snap the selected keyframes to their nearest frame individually.

**Nearest Second** Snap the selected keyframes to their nearest second individually, based on the *FPS* of the scene.

**Nearest Marker** Snap the selected keyframes to their nearest marker individually.

**Flatten Handles** Flatten the *Bezier* handles for the selected keyframes.

Table  
2.48:  
After  
Flatten  
Handles.



**Mirror** Selected keyframes can be mirrored over different properties using the *Mirror Keys* tool.

#### Mirror Keys `Shift-M`

**By Times Over Current Frame** Mirror horizontally over the *Time Cursor*.

**By Values over Cursor Value** Mirror vertically over the *Cursor*.

**By Times over Time 0** Mirror horizontally over frame 0.

**By Values over Value 0** Mirror vertically over value 0.

**By Times over First Selected Marker** Mirror horizontally the over the first selected *Marker*.

**Clean Keyframes** *Clean Keyframes* resets the keyframe tangents to their auto-clamped shape, if they have been modified.  
*Clean Keyframes* 



**Smoothing** (`Alt-O` or `Key` → *Smooth Keys*) There is also an option to smooth the selected curves, but beware: its algorithm seems to be to divide by two the distance between each keyframe and the average linear value of the curve, without any setting, which gives quite a strong smoothing! Note that the first and last keys seem to be never modified by this tool.



#### Sampling and Baking Keyframes

**Sample Keyframes** `Shift-O` Sampling a set a keyframes replaces interpolated values with a new keyframe for each frame.



**Bake Curves** `Alt-C` Baking a curve replaces it with a set of sampled points, and removes the ability to edit the curve.

## F-Curve Modifiers

F-Curve modifiers are similar to object modifiers, in that they add non-destructive effects, that can be adjusted at any time, and layered to create more complex effects.

### Adding a Modifier

The F-curve modifier panel is located in the Properties panel. Select a curve by selecting one of its curve points, or by selecting the channel list. Click on the *Add Modifier* button and select a modifier.

To add spin to an object or group, select the object/group and add a keyframe to the axis of rotation (x,y, or z)

Go to the Graph Editor.....make sure the f-curves properties panel is visible (View > Properties)

>Add Modifier > (e.g.) Generator

### Types of Modifiers

**Generator** Generator creates a Factorized or Expanded Polynomial function. These are basic mathematical formulas that represent lines, parabolas, and other more complex curves, depending on the values used.

**Additive** This option causes the modifier to be added to the curve, instead of replacing it by default.

**Poly Order** Specify the order of the polynomial, or the highest power of 'x' for this polynomial. (number of coefficients - 1).

Change the Coefficient values to change the shape of the curve.

**See also:**

[The Wikipedia Page](#) for more information on polynomials.

**Built-in Function** These are additional formulas, each with the same options to control their shape. Consult mathematics reference for more detailed information on each function.

- Sine
- Cosine
- Tangent
- Square Root
- Natural Logarithm
- Normalized Sine ( $\sin(x)/x$ )

**Amplitude** Adjusts the Y scaling

**Phase Multiplier** Adjusts the X scaling

**Phase Offset** Adjusts the X offset

**Value Offset** Adjusts the Y offset

**Envelope** Allows you to adjust the overall shape of a curve with control points.

**Reference Value** Set the Y value the envelope is centered around.

**Min** Lower distance from Reference Value for 1 : 1 default influence.

**Max** Upper distance from Reference Value for 1 : 1 default influence.

**Add Point** Add a set of control points. They will be created at the current frame.

**Fra:** Set the frame number for the control point.

**Min** Specifies the lower control point's position.

**Max** specifies the upper control point's position.

**Cycles** Cycles allows you add cyclic motion to a curve that has 2 or more control points. The options can be set for before and after the curve.

#### Cycle Mode

**Repeat Motion** Repeats the curve data, while maintaining their values each cycle.

**Repeat with Offset** Repeats the curve data, but offsets the value of the first point to the value of the last point each cycle.

**Repeat Mirrored** Each cycle the curve data is flipped across the X-axis.

**Before/After Cycles** Set the number of times to cycle the data. A value of 0 cycles the data infinitely.

**Noise** Modifies the curve with a noise formula. This is useful for creating subtle or extreme randomness to animated movements, like camera shake.

#### Blend Type

**Replace** Adds a -.5 to .5 range noise function to the curve.

**Add** Adds a 0 to 1 range noise function to the curve.

**Subtract** Subtracts a 0 to 1 range noise function to the curve.

**Multiply** Multiplies a 0 to 1 range noise function to the curve.

**Scale** Adjust the overall size of the noise. Values further from 0 give less frequent noise.

**Strength** Adjusts the Y scaling of the noise function.

**Phase** Adjusts the random seed of the noise.

**Depth** Adjusts how detailed the noise function is.

#### Python

**Limits** Limit curve values to specified X and Y ranges.

**Minimum/Maximum X** Cuts a curve off at these frames ranges, and sets their minimum value at those points.

**Minimum/Maximum Y** Truncates the curve values to a range.

**Stepped** Gives the curve a stepped appearance by rounding values down within a certain range of frames.

**Step Size** Specify the number of frames to hold each frame

**Offset** Reference number of frames before frames get held. Use to get hold for '1-3' vs '5-7' holding patterns.

**Use Start Frame** Restrict modifier to only act before its 'end' frame

**Use End Frame** Restrict modifier to only act after its 'start' frame

## Non-Linear Animation Editor

The NLA editor can manipulate and repurpose actions, without the tedium of keyframe handling. Its often used to make broad, significant changes to a scene's animation, with relative ease. It can also repurpose, and 'layer' actions, which make it easier to organize, and version-control your animation.

### Tracks

Tracks are the layering system of the NLA. At its most basic level, it can help organize strips. But it also layers motion much like an image editor layers pixels - the bottom layer first, to the top, last.



### Strips

There's three kinds of strips - Action, Transition, and Meta. Actions contain the actual keyframe data, Transitions will perform calculations between Actions, and Meta will group strips together as a whole.

**Creating Action Strips** Any action used by the NLA first must be turned into an Action strip. This is done so by clicking the



Fig. 2.1308: next to the action listed in the NLA. Alternatively, you can go to

---

### Reference

Menu: Add -> Action

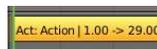


Fig. 2.1309: Action Strip.

**Creating Transition Strips** Select two or more strips on the same track, and go to

---

### Reference

Menu: Add -> Transition

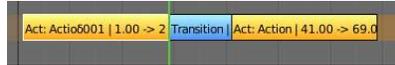


Fig. 2.1310: Transition Strip.

**Grouping Strips into Meta Strips** If you find yourself moving a lot of strips together, you can group them into a Meta strip. A meta strip can be moved and duplicated like a normal strip.

---

### Reference

Menu: Add → Add Meta-Strips

Hotkey: Shift-G



A meta strip still contains the underlying strips. You can ungroup a Meta strip.

---

### Reference

Menu: Add → Remove Meta-Strips

Hotkey: Alt-G

---

## Editing Strips

The contents of Action strips can be edited, but you must be in ‘Tweak Mode’ to do so.

---

### Reference

Menu: View → Enter Tweak Mode

Hotkey: Tab



If you try moving the strip, while in edit mode, you’ll notice that the keys will go along with it. On occasion, you’ll prefer the keys to remain on their original frames, regardless of where the strip is. To do so, hit the ‘unpin’ icon, next to the strip.

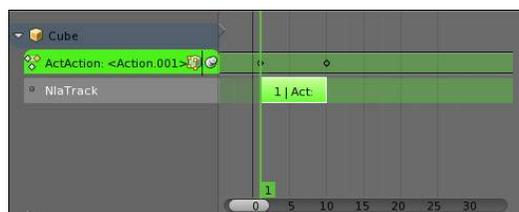


Fig. 2.1319: Nla strip with pinned keys.

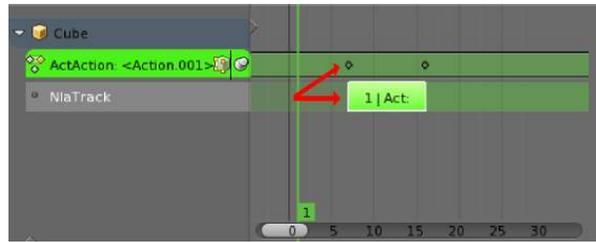


Fig. 2.1320: Strip moved, notice the keys move with it.

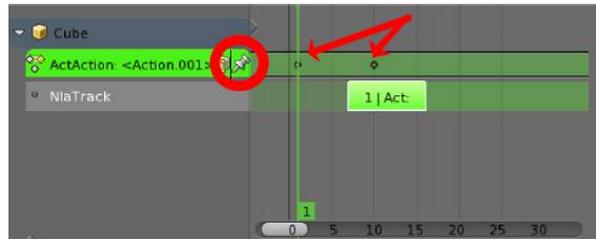


Fig. 2.1321: The unpinned keys return to their original frames.

When your finished editing the strip, simply go to View > Exit Tweak Mode. Note the default key for this is Tab.

### Re-Instancing Strips

The contents' of one Action strip can be instanced multiple times. To instance another strip, select a strip, go to

---

### Reference

Menu: Edit-> Duplicate Strips

---

Now, when any strip is tweaked, the others will change too. If a strip other than the original is tweaked, the original will turn to red.



### Strip Properties

Strip properties can be accessed via the NLA header.

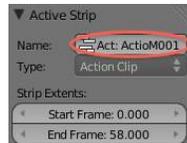
---

### Reference

Menu: View-> Properties

---

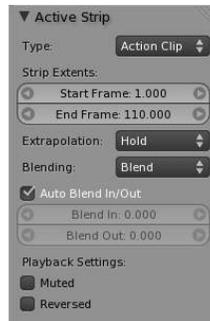
**Renaming Strips** All strips can be renamed, in the “Active Track” section in the Strip Properties.



**Active Track** This is which track the strip currently belongs to.



**Active Strip** Elements of the strip itself. An Action Strip can be either an Action Clip, or a Transition Clip. Note that the ‘Strip Extents’ fields determine strictly the strip, and not the action. Also, the “Hold” value in the Extrapolation section means hold both beginning, and after. This can cause previous clips to not work, if checked.



**Active Action** This represents the ‘object data’ of the strip. Much like the transform values of an object.

**Evaluation** This determines the degree of influence the strip has, and over what time.

If influence isn’t animated, the strips will fade linearly, during the overlap.

## Strip Modifiers

Like its close cousins in mesh and graph editing, Modifiers can stack different combinations of effects for strips. Obviously there will be more to come on this.

## The Timeline

The *Timeline* window, identified by a clock icon, is shown by default at the bottom of the screen.

The *Timeline* is not much of an editor, but more of a information and control window.

**Here you can have an overview of the animation part of your scene** What is the current time frame, either in frames or in seconds, where are the keyframes of the active object, the start and end frames of your animation, markers, etc...

The *Timeline* has *Player Controls*, to play, pause the animation, and to skip though parts of the scene.

It also has some tools for *Keyframes*, *Keying Sets*, and *Markers*.

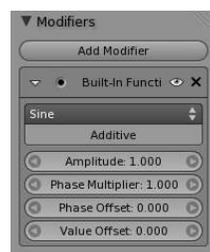
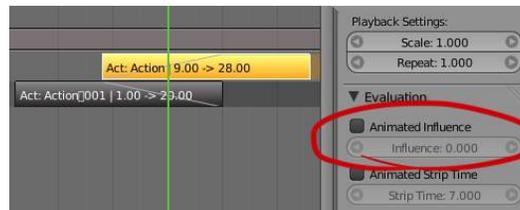
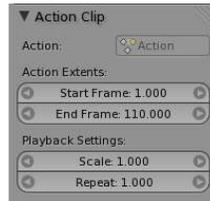


Fig. 2.1328: Timeline 2.69

## Timeline Elements

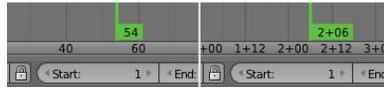


Fig. 2.1329: Time Cursor

**Time Cursor** The *Time Cursor* is the green line, its used to set and display the current time frame.

The *Time Cursor* can be set or moved to a new position by pressing or holding LMB in the Timeline window.

The current frame or second can be displayed on the *Time Cursor*, check the View menu for settings.

The *Time Cursor* can be moved in steps by pressing Left or Right, or in steps of 10 frames by pressing Shift-Up or Shift-Down.

**Keyframes** For the active and selected objects, keyframes are displayed as a yellow line.

For *Armatures*, the object keyframes and the pose bones keyframes are drawn.

*Only Selected Channels* can be enabled. *Timeline > View > Only Selected Channels*. For *Armatures*, this will draw the object keyframes, and the keyframes for the active and selected pose bones.

**Markers** Markers are the small triangles, with their name near them.

Markers are usually used to identify key parts of the animation.

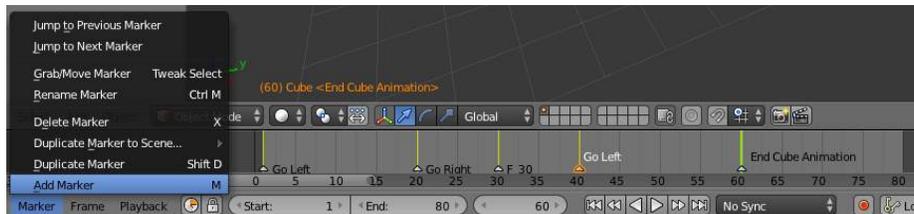


Fig. 2.1330: Markers

Markers can be selected by pressing RMB or Shift-RMB to select more.

See [Marker Menu](#) below or [Markers](#) for more info.

## Adjusting the View

**Timeline Area** The main *Timeline* area displays the animation frames over time.

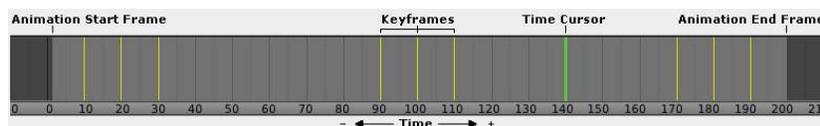


Fig. 2.1331: Timeline Main Area

The *Timeline* can be panned by holding MMB, then dragging the area left or right.

You can zoom the *Timeline* by using `Ctrl-MMB`, the mouse *Wheel*, or pressing the *Minus* and *Plus* keys on the numpad. By default, the *Playback/Rendering Range* (Frame Start 1 to Frame End 200) is a lighter shade of gray. The start and end frame can be set to the *Time Cursor* by pressing `S` or `E`. The *Playback Range* can also be set by pressing `P` then drawing a box.

### Timeline Header

**View Menu** The *View Menu* controls what you see, and what it looks like.

**Toggle Full Screen** Maximize or minimize the *Timeline* window. `Ctrl-Up` or `Ctrl-Down`

**Duplicate Area into New Window** This creates a new OS window, and sets the editor window to the *Timeline*.

**Bind Camera to Markers** This is used switch cameras during animation. It binds the active camera to the selected markers. First select a camera. Then select the marker(s). Then use the function. `Ctrl-B`

**Cache** This will display the baked *Cache Steps* for the active object.

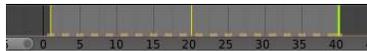


Fig. 2.1332: Timeline Cache

**Show Cache** Show all enabled types.

*Softbody, Particles, Cloth, Smoke, Dynamic Paint, Rigid Body.*

**Only Selected Channels** For *Armatures*, this will draw the object keyframes, and the keyframes for the active and selected pose bones.

**Show Frame Number Indicator** This will draw the current frame or seconds on the *Time Cursor*.

**View All** Maximize the *Timeline* area based on the Animation Range. `Home`

**Show Seconds** Show time in seconds for the *Timeline* and the *Time Cursor* based on the FPS. `Ctrl-T`

**Marker Menu** Jump to Previous Marker

Jump to Next Marker

**Grab/Move Marker** Grab/Move the selected markers. `G`

**Rename Marker** Rename the active marker. `Ctrl-M`

**Delete Marker** Delete selected markers. `X`

**Duplicate Marker to Scene...** Duplicate the selected markers to another scene.

**Duplicate Marker** Duplicate the selected markers. `Shift-D`

**Add Marker** Add marker to the current frame. `M`

### Frame Menu

**Auto-Keyframing Mode** This controls how the Auto Keyframe mode works. Only one mode can be used at a time.

**Add & Replace** Add or Replace existing keyframes.

**Replace** Only Replace existing keyframes.

## Playback Menu

- *Audio Scrubbing* If your animation has sound, this option plays bits of the sound wave while you move the time cursor with LMB or keyboard arrows.
- *Audio Muted* Mute the sound from Sequence Editors.
- *AV-sync* Play back and sync with audio clock, dropping frames if frame display is too slow. See 4. *Synchronize Playback* for more info.
- *Frame Dropping* Play back dropping frames if frames are too slow. See 4. *Synchronize Playback* for more info.
- *Clip Editors* While playing, updates the *Movie Clip Editor*.
- *Node Editors* While playing, updates the Node properties for the *Node Editor*.
- *Sequencer Editors* While playing, updates the *Video Sequence Editor*.

**Note:** Image Editors

TODO Not sure what is updated, maybe gif images or, image sequence.

- *Image Editors* Todo
- *Property Editors* When the animation is playing, this will update the property values in the UI.
- *Animation Editors* While playing, updates the *Timeline*, *Dope Sheet*, *Graph Editor*, *Video Sequence Editor*.
- *All 3D View Editors* While playing, updates the *3D View* and the *Timeline*.
- *Top-Left 3D Editor* While playing, updates the *Timeline* if *Animation Editors* and *All 3D View Editors* disabled.

**Header Controls** The Timeline header controls.



Fig. 2.1333: Timeline header controls.

### 1. Range Control

**Use Preview Range** This is an alternative range used to preview animations. This works for the UI playback, this will not work for rendering an animation.

**Lock Time Cursor to Playback Range** This limits the *Time Cursor* to the *Playback Range*.

### 2. Frame Control

**Start Frame** The start frame of the animation / playback range.

**End Frame** The end frame of the animation / playback range.

**Current Frame** The current frame of the animation / playback range. Also the position of the *Time Cursor*.

### 3. Player Control

These buttons are used to set, play, rewind, the *Time Cursor*.



Fig. 2.1334: Player Controls.

**Jump to start** This sets the cursor to the start of frame range. `Shift-Ctrl-Down` or `Shift-Left`

**Jump to previous keyframe** This sets the cursor to the previous keyframe. `Down`

**Rewind** This plays the animation sequence in reverse. `Shift-Alt-A` When playing the play buttons switch to a pause button.

**Play** This plays the animation sequence. `Alt-A` When playing the play buttons switch to a pause button.

**Jump to next keyframe** This sets the cursor to the next keyframe. `Up`

**Jump to end** This sets the cursor to the end of frame range. `Shift-Ctrl-Up` or `Shift-Right`

**Pause** This stops the animation. `Alt-A`

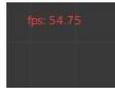


Fig. 2.1335: 3D View Red FPS. 60:54.75

When you play an animation, the FPS is displayed at the top left of the 3D View. If the scene is detailed and playback is slower than the set *Frame Rate* (see *Dimensions Presets*, these options are used to synchronize the playback.

**No Sync** Do not sync, play every frame.

**Frame Dropping** Drop frames if playback is too slow. This enables *Frame Dropping* from the *Playback Menu*.

**AV-sync** Sync to audio clock, dropping frames if playback is slow. This enables *AV-sync* and *Frame Dropping* from the *Playback Menu*.

## 4. Synchronize Playback



Fig. 2.1336: Timeline Auto Keyframe.

**Auto Keyframe** The “Record” red-dot button enables something called *Auto Keyframe* : It will add and/or replace existing keyframes for the active object when you transform it in the 3D view.

For example, when enabled, first set the *Time Cursor* to the desired frame, then move an object in the 3d view, or set a new value for a property in the UI.

When you set a new value for the properties, blender will add keyframes on the current frame for the transform properties.

*Auto Keying Set* - Optional if Auto Keyframe enabled. *Auto Keyframe* will insert new keyframes for the properties in the active *Keying Set*.

Note that *Auto Keyframe* only works for transform properties (objects and bones), in the 3D views (i.e. you cant use it e.g. to animate the colors of a material in the Properties window...).

## 5. Keyframe Control

**Note:** Layered

Todo.

### User Preferences

Some related user preferences from the **Editing** tab.



Fig. 2.1337: Timeline Layered.

*Layered* - Optional while playback. TODO.



Fig. 2.1338: Timeline Keying Sets.

**Active Keying Set** *Keying Sets* are a set of keyframe channels in one.

They are made so the user can record multiple properties at the same time.

With a keying set selected, when you insert a keyframe, blender will add keyframes for the properties in the active *Keying Set*.

There are some built in keying sets, 'LocRotScale', and also custom keying sets.

Custom keying sets can be defined in the in the panels *Properties > Scene > Keying Sets + Active Keying Set*.

**Insert Keyframes** Insert keyframes on the current frame for the properties in the active *Keying Set*.

**Delete Keyframes** Delete keyframes on the current frame for the properties in the active *Keying Set*.

## Playback

**Allow Negative Frames** Time Cursor can be set to negative frames with mouse or keyboard. When using *Use Preview Range*, this also allows playback.

## Keyframing

**Visual Keying** When an object is using constraints, the objects property value doesnt actually change. *Visual Keying* will add keyframes to the object property, with a value based on the visual transformation from the constraint.

**Only Insert Needed** This will only insert keyframes if the value of the property is different.

**Auto Keyframing** Enable *Auto Keyframe* by default for new scenes.

**Show Auto Keying Warning** Displays a warning at the top right of the *3D View*, when moving objects, if *Auto Keyframe* is on.

**Only Insert Available** With *Auto Keyframe* enabled, this will only add keyframes to channel F-Curves that already exist.

## 2.6.9 The Timeline

The *Timeline* window, identified by a clock icon, is shown by default at the bottom of the screen.



Fig. 2.1339: Timeline 2.69

The *Timeline* is not much of an editor, but more of a information and control window.

**Here you can have an overview of the animation part of your scene** What is the current time frame, either in frames or in seconds, where are the keyframes of the active object, the start and end frames of your animation, markers, etc...

The *Timeline* has *Player Controls*, to play, pause the animation, and to skip though parts of the scene.

It also has some tools for *Keyframes*, *Keying Sets*, and *Markers*.

## Timeline Elements

### Time Cursor



Fig. 2.1340: Time Cursor

The *Time Cursor* is the green line, its used to set and display the current time frame.

The *Time Cursor* can be set or moved to a new position by pressing or holding LMB in the Timeline window.

The current frame or second can be displayed on the *Time Cursor*, check the View menu for settings.

The *Time Cursor* can be moved in steps by pressing Left or Right, or in steps of 10 frames by pressing Shift-Up or Shift-Down.

### Keyframes

For the active and selected objects, keyframes are displayed as a yellow line.

For *Armatures*, the object keyframes and the pose bones keyframes are drawn.

*Only Selected Channels* can be enabled. *Timeline > View > Only Selected Channels*. For *Armatures*, this will draw the object keyframes, and the keyframes for the active and selected pose bones.

### Markers

Markers are the small triangles, with their name near them.

Markers are usually used to identify key parts of the animation.

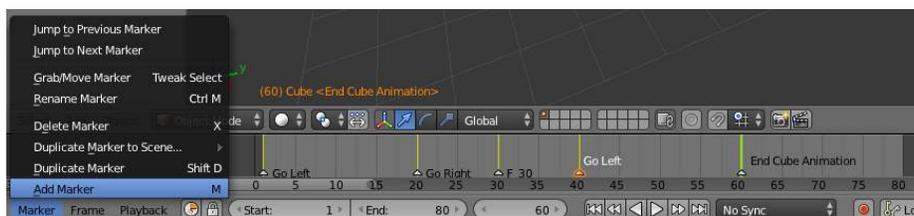


Fig. 2.1341: Markers

Markers can be selected by pressing RMB or Shift-RMB to select more.

See *Marker Menu* below or *Markers* for more info.

## Adjusting the View

### Timeline Area

The main *Timeline* area displays the animation frames over time.

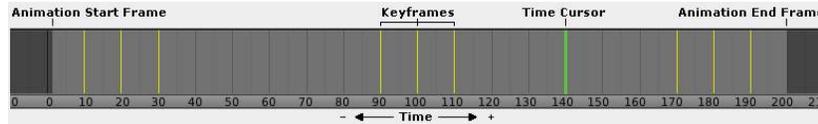


Fig. 2.1342: Timeline Main Area

The *Timeline* can be panned by holding MMB, then dragging the area left or right.

You can zoom the *Timeline* by using `Ctrl`-MMB, the mouse `Wheel`, or pressing the `Minus` and `Plus` keys on the numpad.

By default, the *Playback/Rendering Range* (Frame Start 1 to Frame End 200) is a lighter shade of gray. The start and end frame can be set to the *Time Cursor* by pressing `S` or `E`. The *Playback Range* can also be set by pressing `P` then drawing a box.

### Timeline Header

#### View Menu

The *View Menu* controls what you see, and what it looks like.

**Toggle Full Screen** Maximize or minimize the *Timeline* window. `Ctrl`-Up or `Ctrl`-Down

**Duplicate Area into New Window** This creates a new OS window, and sets the editor window to the *Timeline*.

**Bind Camera to Markers** This is used switch cameras during animation. It binds the active camera to the selected markers. First select a camera. Then select the marker(s). Then use the function. `Ctrl`-B

**Cache** This will display the baked *Cache Steps* for the active object.

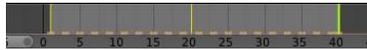


Fig. 2.1343: Timeline Cache

**Show Cache** Show all enabled types.

*Softbody, Particles, Cloth, Smoke, Dynamic Paint, Rigid Body.*

**Only Selected Channels** For *Armatures*, this will draw the object keyframes, and the keyframes for the active and selected pose bones.

**Show Frame Number Indicator** This will draw the current frame or seconds on the *Time Cursor*.

**View All** Maximize the *Timeline* area based on the Animation Range. `Home`

**Show Seconds** Show time in seconds for the *Timeline* and the *Time Cursor* based on the FPS. `Ctrl`-T

### Marker Menu

Jump to Previous Marker

Jump to Next Marker

**Grab/Move Marker** Grab/Move the selected markers. `G`

**Rename Marker** Rename the active marker. `Ctrl-M`

**Delete Marker** Delete selected markers. `X`

**Duplicate Marker to Scene...** Duplicate the selected markers to another scene.

**Duplicate Marker** Duplicate the selected markers. `Shift-D`

**Add Marker** Add marker to the current frame. `M`

### Frame Menu

**Auto-Keyframing Mode** This controls how the Auto Keyframe mode works. Only one mode can be used at a time.

**Add & Replace** Add or Replace existing keyframes.

**Replace** Only Replace existing keyframes.

### Playback Menu

- *Audio Scrubbing* If your animation has sound, this option plays bits of the sound wave while you move the time cursor with LMB or keyboard arrows.
- *Audio Muted* Mute the sound from Sequence Editors.
- *AV-sync* Play back and sync with audio clock, dropping frames if frame display is too slow. See 4. *Synchronize Playback* for more info.
- *Frame Dropping* Play back dropping frames if frames are too slow. See 4. *Synchronize Playback* for more info.
- *Clip Editors* While playing, updates the *Movie Clip Editor*.
- *Node Editors* While playing, updates the Node properties for the *Node Editor*.
- *Sequencer Editors* While playing, updates the *Video Sequence Editor*.

---

**Note:** Image Editors

TODO Not sure what is updated, maybe gif images or, image sequence.

---

- *Image Editors* Todo
- *Property Editors* When the animation is playing, this will update the property values in the UI.
- *Animation Editors* While playing, updates the *Timeline*, *Dope Sheet*, *Graph Editor*, *Video Sequence Editor*.
- *All 3D View Editors* While playing, updates the *3D View* and the *Timeline*.
- *Top-Left 3D Editor* While playing, updates the *Timeline* if *Animation Editors* and *All 3D View Editors* disabled.

### Header Controls

The Timeline header controls.



Fig. 2.1344: Timeline header controls.

### 1. Range Control

**Use Preview Range** This is an alternative range used to preview animations. This works for the UI playback, this will not work for rendering an animation.

**Lock Time Cursor to Playback Range** This limits the *Time Cursor* to the *Playback Range*.

### 2. Frame Control

**Start Frame** The start frame of the animation / playback range.

**End Frame** The end frame of the animation / playback range.

**Current Frame** The current frame of the animation / playback range. Also the position of the *Time Cursor*.

### 3. Player Control

These buttons are used to set, play, rewind, the *Time Cursor*.



Fig. 2.1345: Player Controls.

**Jump to start** This sets the cursor to the start of frame range. `Shift-Ctrl-Down` or `Shift-Left`

**Jump to previous keyframe** This sets the cursor to the previous keyframe. `Down`

**Rewind** This plays the animation sequence in reverse. `Shift-Alt-A` When playing the play buttons switch to a pause button.

**Play** This plays the animation sequence. `Alt-A` When playing the play buttons switch to a pause button.

**Jump to next keyframe** This sets the cursor to the next keyframe. `Up`

**Jump to end** This sets the cursor to the end of frame range. `Shift-Ctrl-Up` or `Shift-Right`

**Pause** This stops the animation. `Alt-A`



Fig. 2.1346: 3D View Red FPS. 60:54.75

When you play an animation, the FPS is displayed at the top left of the 3D View. If the scene is detailed and playback is slower than the set *Frame Rate* (see *Dimensions Presets*, these options are used to synchronize the playback.

**No Sync** Do not sync, play every frame.

**Frame Dropping** Drop frames if playback is too slow. This enables *Frame Dropping* from the *Playback Menu*.

**AV-sync** Sync to audio clock, dropping frames if playback is slow. This enables *AV-sync* and *Frame Dropping* from the *Playback Menu*.

### 4. Synchronize Playback

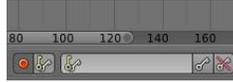


Fig. 2.1347: Timeline Auto Keyframe.

**Auto Keyframe** The “Record” red-dot button enables something called *Auto Keyframe* : It will add and/or replace existing keyframes for the active object when you transform it in the 3D view.

For example, when enabled, first set the *Time Cursor* to the desired frame, then move an object in the 3d view, or set a new value for a property in the UI.

When you set a new value for the properties, blender will add keyframes on the current frame for the transform properties.

*Auto Keying Set* - Optional if Auto Keyframe enabled. *Auto Keyframe* will insert new keyframes for the properties in the active *Keying Set*.

Note that *Auto Keyframe* only works for transform properties (objects and bones), in the 3D views (i.e. you cant use it e.g. to animate the colors of a material in the Properties window...).

## 5. Keyframe Control

**Note:** Layered

Todo.



Fig. 2.1348: Timeline Layered.

*Layered* - Optional while playback. TODO.

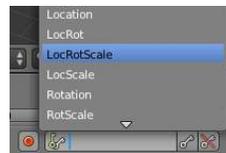


Fig. 2.1349: Timeline Keying Sets.

**Active Keying Set** *Keying Sets* are a set of keyframe channels in one.

They are made so the user can record multiple properties at the same time.

With a keying set selected, when you insert a keyframe, blender will add keyframes for the properties in the active *Keying Set*.

There are some built in keying sets, ‘LocRotScale’, and also custom keying sets.

Custom keying sets can be defined in the in the panels *Properties* > *Scene* > *Keying Sets* + *Active Keying Set*.

**Insert Keyframes** Insert keyframes on the current frame for the properties in the active *Keying Set*.

**Delete Keyframes** Delete keyframes on the current frame for the properties in the active *Keying Set*.

## User Preferences

Some related user preferences from the **Editing** tab.

### Playback

**Allow Negative Frames** Time Cursor can be set to negative frames with mouse or keyboard. When using *Use Preview Range*, this also allows playback.

### Keyframing

**Visual Keying** When an object is using constraints, the objects property value doesnt actually change. *Visual Keying* will add keyframes to the object property, with a value based on the visual transformation from the constraint.

**Only Insert Needed** This will only insert keyframes if the value of the property is different.

**Auto Keyframing** Enable *Auto Keyframe* by default for new scenes.

**Show Auto Keying Warning** Displays a warning at the top right of the *3D View*, when moving objects, if *Auto Keyframe* is on.

**Only Insert Available** With *Auto Keyframe* enabled, this will only add keyframes to channel F-Curves that already exist.

## 2.6.10 Graph Editor

The graph editor is the main animation editor. It allows you to modify the animation for any properties using *F-Curves*.

The graph editor has two modes, *F-Curve* for *Actions*, and *Drivers* for *Drivers*. Both are very similar in function.

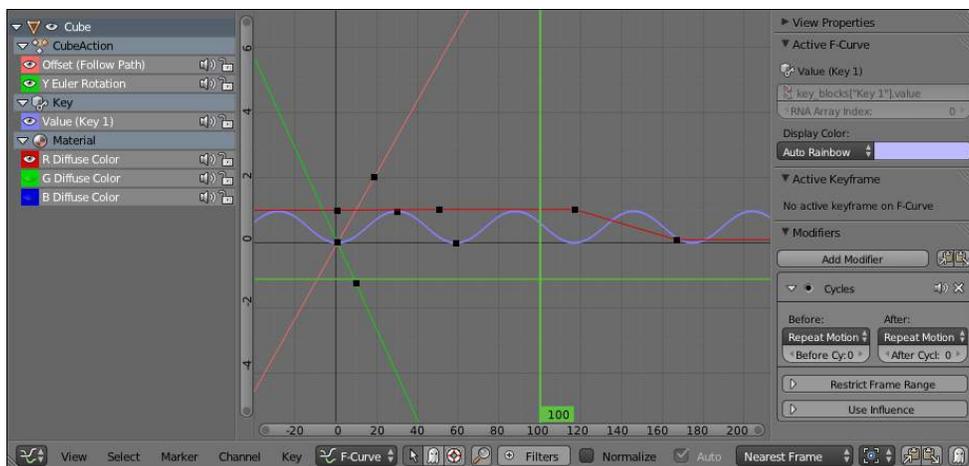


Fig. 2.1350: The Graph Editor.

### Curve Editor Area

Here you can see and edit the curves and keyframes.

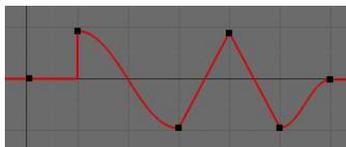


Fig. 2.1351: A curve with different types of interpolation.

See *F-Curves* for more info.

### Navigation

As with most windows, you can:

**Pan MMB** Pan the view vertically (values) or horizontally (time) with click and drag.

**Zoom Wheel** Zoom in and out with the mouse wheel.

**Scale View Ctrl-MMB** Scale the view vertically or horizontally.

These are some other useful tools.

**View All Home** Reset viewable area to show all keyframes.

**View Selected NumpadPeriod** Reset viewable area to show selected keyframes.

## 2D Cursor

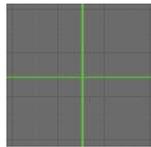


Fig. 2.1352: Graph Editor 2D Cursor.

The current frame is represented by a green vertical line called the *Time Cursor*.

As in the [Timeline](#), you can change the current frame by pressing or holding LMB.

The green horizontal line is called the *Cursor*. This can be disabled via the *View Menu* or the *View Properties* panel.

The *Time Cursor* and the *Cursor* make the *2D Cursor*. The *2D Cursor* mostly used for editing tools.

## View Axes

For *Actions* the X-axis represents time, the Y-axis represents the value to set the property.

For *Drivers* the X-axis represents the *Driver Value*, the Y-axis represents the value to set the property.

Depending on the selected curves, the values have different meaning: For example rotation properties are shown in degrees, location properties are shown in Blender Units. Note that *Drivers* use radians for rotation properties.

## Markers

Like with most animation editors, markers are shown at the bottom of the editor.



Fig. 2.1353: Graph Editor Markers.

*Markers* can be modified in the *Graph Editor* though its usually best to use the *Timeline*.

See [Markers](#) for more info.

## Header

Here you'll find.

- The menus.
- Graph Editor mode.
- View controls.
- Curve controls.

## Header Controls



Fig. 2.1354: Graph Mode.

**Mode** F-Curve for [Actions](#), and Drivers for [Drivers](#).



Fig. 2.1355: View Controls.

## View controls

**Show Only Selected** Only include curves related to the selected objects and data.

**Show Hidden** Include curves from objects/bones that are not visible.

**Show Only Errors** Only include curves that are disabled or have errors.

**Search Filter** Only include curves with keywords contained in the search text.

**Type Filter** Filter curves by property type.

**Normalize** Normalize curves so the maximum or minimum point equals 1.0 or -1.0.

**Auto** Automatically recalculate curve normalization on every curve edit.



Fig. 2.1356: Curve Controls.

## Curve controls

**Auto Snap** Auto snap the keyframes for transformations.

*No Auto-Snap Time Step Nearest Frame Nearest Marker*

**Pivot Point** Pivot point for rotation.

**Bounding Box Center** Center of the select keyframes.

**2D Cursor** Center of the *2D Cursor*. *Time Cursor + Cursor*.

**Individual Centers** Rotate the selected keyframe *Bezier* handles.

**Copy Keyframes** **Ctrl-C** Copy the selected keyframes to memory.

**Paste Keyframes Ctrl-V** Paste keyframes from memory to the current frame for selected curves.

**Create Snapshot** Creates a picture with the current shape of the curves.

## Channels Region

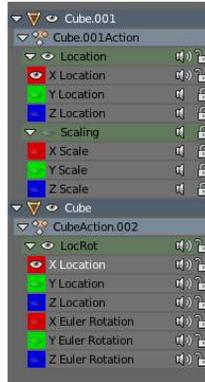


Fig. 2.1357: Channels Region.

The channels region is used to select and manage the curves for the graph editor.

**Hide curve** Represented by the eye icon.

**Deactive/Mute curve** Represented by the speaker icon.

**Lock curve from editing** Represented by the padlock icon.

## Channel Editing

*Select channel* LMB

*Multi Select/Deselect* Shift-LMB

*Toggle Select All* A

*Border Select* (LMB drag) or B (LMB drag)

*Border Deselect* (Shift-LMB drag) or B (Shift-LMB drag)

*Delete selected* X or Delete

*Lock selected* Tab

*Make only selected visible* V

*Enable Mute Lock selected* Shift-Ctrl-W

*Disable Mute Lock selected* Alt-W

*Toggle Mute Lock selected* Shift-W

## Properties Region

The panels in the *Properties Region*.

## View Properties Panel

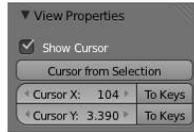


Fig. 2.1358: View Properties Panel.

**Show Cursor** Show the vertical *Cursor*.

**Cursor from Selection** Set the *2D cursor* to the center of the selected keyframes.

**Cursor X** *Time Cursor X* position.

**To Keys** Snap selected keyframes to the *Time Cursor*.

**Cursor Y** Vertical *Cursor Y* position.

**To Keys** Snap selected keyframes to the *Cursor*.

## Active F-Curve Panel



Fig. 2.1359: Active F-Curve Panel.

This panel displays properties for the active *F-Curve*.

**Channel Name (X Location)** *ID Type* + Channel name.

**RNA Path** *RNA Path* to property + Array index.

**Color Mode** *Color Mode* for the active *F-Curve*.

**Auto Rainbow** Increment the *HUE* of the *F-Curve* color based on the channel index.

**Auto XYZ to RGB** For property sets like location xyz, automatically set the set of colors to red, green, blue.

**User Defined** Define a custom color for the active *F-Curve*.

## Active Keyframe Panel

**Interpolation** Set the forward interpolation for the active keyframe.

**Constant** Keep the same value till the next keyframe.

**Linear** The difference between the next keyframe.

**Bezier** Bezier interpolation to the next keyframe.

**Key**

**Frame** Set the frame for the active keyframe.



Fig. 2.1360: Active Keyframe Panel.

**Value** Set the value for the active keyframe.

**Left Handle** Set the position of the left interpolation handle for the active keyframe.

**Right Handle** Set the position of the right interpolation handle for the active keyframe.

### Drivers Panel



Fig. 2.1361: Drivers Panel.

See [Drivers Panel](#) for more info.

### Modifiers Panel



Fig. 2.1362: Modifiers Panel.

See [F-Modifiers](#) for more info.

### See Also

- [Graph Editor - F-Curves](#)
- [Graph Editor - F-Modifiers](#)
- [Actions](#)
- [Drivers](#)

## 2.6.11 F-Curves

Once you have created keyframes for something, you can edit their corresponding curves. In Blender 2.5, IPO Curves have been replaced by FCurves, however, editing these curves is essentially still the same.

## The concept of Interpolation

When something is “animated,” it changes over time. In Blender, animating an object means changing one of its properties, such as its X location, or the Red channel value of its material diffuse color, and so on, during a certain amount of time.

As mentioned, Blender’s fundamental unit of time is the “frame”, which usually lasts just a fraction of a second, depending on the *frame rate* of the scene.

As animation is composed of incremental changes spanning multiple frames, usually these properties ARE NOT manually modified *frame by frame*, because:

- it would take ages!
- it would be very difficult to get smooth variations of the property (unless you compute mathematical functions and type a precise value for each frame, which would be crazy).

This is why nearly all direct animation is done using **interpolation**.

The idea is simple: you define a few Key Frames, which are multiple frames apart. Between these keyframes, the properties’ values are computed (interpolated) by Blender and filled in. Thus, the animators’ workload is significantly reduced.



Fig. 2.1363: Example of interpolation

For example, if you have:

- a control point of value 0 at frame 0,
- another one of value 10 at frame 25,
- linear interpolation,

then, at frame 5 we get a value of 2.

The same goes for all intermediate frames: with just two points, you get a smooth growth from 0 to 10 along the **25 frames**. Obviously, if you'd like the frame 15 to have a value of 9, you'd have to add another control point (or keyframe)...

## Settings

F-curves have three additional properties, which control the interpolation between points, extension behavior, and the type of handles.

### Interpolation Mode

You have three choices (T, or *Curve* → *Interpolation Mode*):

**Constant** There is no interpolation at all. The curve holds the value of its last keyframe, giving a discrete (stairway) “curve”. Usually only used during the initial “blocking” stage in pose-to-pose animation workflows.



Fig. 2.1364: Constant.

**Linear** This simple interpolation creates a straight segment between each neighbor keyframes, giving a broken line. It can be useful when using only two keyframes and the *Extrapolation* extend mode, to easily get an infinite straight line (i.e. a linear curve).

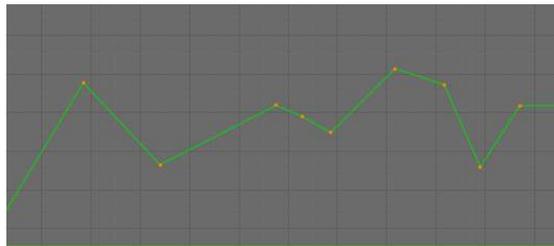


Fig. 2.1365: Linear.

**Bezier** The more powerful and useful interpolation, and the default one. It gives nicely smoothed curves, i.e. smooth animations!

Remember that some FCurves can only take discrete values, in which case they are always shown as if constant interpolated, whatever option you chose.



Fig. 2.1366: Bézier.

## Extrapolation

(`Shift-E`, or *Channel* → *Extrapolation Mode*)

Extrapolation defines the behavior of a curve before the first and after the last keyframes.

There are two basic extrapolation modes:

**Constant** The default one, curves before their first keyframe and after their last one have a constant value (the one of these first and last keyframes).

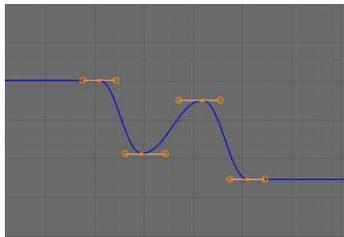


Fig. 2.1367: Constant extrapolation

**Linear** Curves ends are straight lines (linear), as defined by their first two keyframes (respectively their last two keyframes).

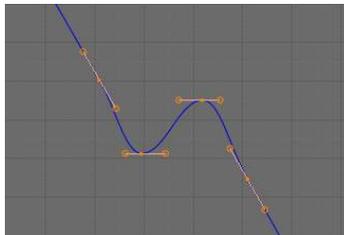


Fig. 2.1368: Linear extrapolation

Additional extrapolation tools (e.g. the “Cycles” F-Modifier) are located in the [F-Curve Modifiers](#)

## Handle Types

There is another curve option quite useful for Bézier-interpolated curves. You can set the type of handle to use for the curve points  $\nabla$

**Automatic** Keyframes are automatically interpolated

**Vector** Creates linear interpolation between keyframes. The linear segments remain if keyframe centers are moved. If handles are moved, the handle becomes Free.

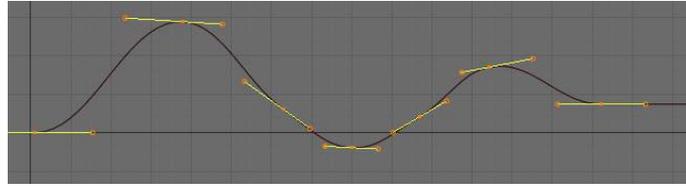


Fig. 2.1369: Auto handles

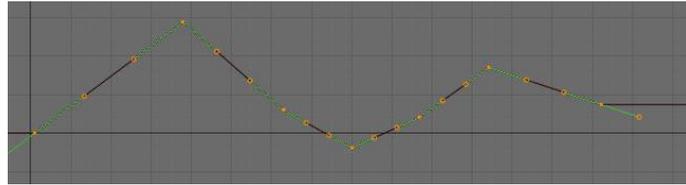


Fig. 2.1370: Vector handles

**Aligned** Handle maintain rotation when moved, and curve tangent is maintained

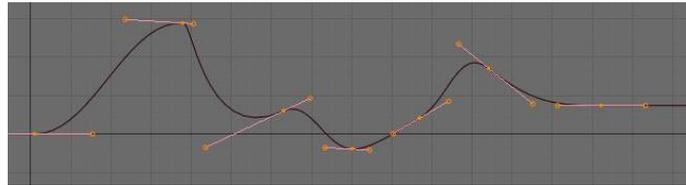


Fig. 2.1371: Aligned handles

**Free** Breaks handles tangents

**Auto Clamped** Auto handles clamped to not overshoot

### Direction of time

Although F-curves are very similar to *Bezier Curves*, there are some important differences.

For obvious reasons, **a property represented by a Curve cannot have more than one value at a given time**, hence:

- when you move a control point ahead of a control point that was previously ahead of the point that you are moving, the two control points switch their order in the edited curve, to avoid that the curve goes back in time
- for the above reason, it's impossible to have a closed Ipo curve

Table  
2.49:  
After  
mov-  
ing the  
second  
keyframe





Fig. 2.1372: Free handles

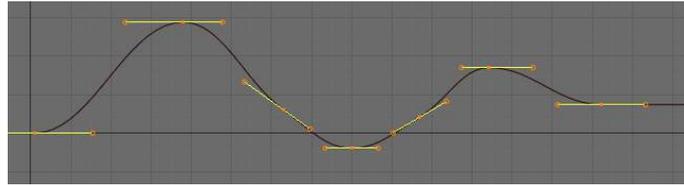


Fig. 2.1373: Auto clamped handles

## Editing Tools

By default, when new channels are added, the *Graph Editor* sets them to *Edit Mode*. Selected channels can be locked by pressing `Tab`.

Many of the hotkeys are the same as the viewport ones, for example:

- `G` to grab
- `R` to rotate
- `S` to scale
- `B` for border select/deselect

And of course you can lock the transformation along the `X` (time frame) or `Y` (value) axes by pressing `X` or `Y` during transformation.

For precise control of the keyframe position and value, you can set values in the *Active Keyframe* of the Properties Region.

## Transform Snapping

When transforming keyframes with `G`, `R`, `S`, the transformation can be snapped to increments.

Snap Transformation to 1.0 `Ctrl`

Divide Transformation by 10.0 `Shift`

Keyframes can be snapped to different properties by using the *Snap Keys* tool.

### Snap Keys `Shift-S`

**Current Frame** Snap the selected keyframes to the *Time Cursor*.

**Cursor Value** Snap the selected keyframes to the *Cursor*.

**Nearest Frame** Snap the selected keyframes to their nearest frame individually.

**Nearest Second** Snap the selected keyframes to their nearest second individually, based on the *FPS* of the scene.

**Nearest Marker** Snap the selected keyframes to their nearest marker individually.

**Flatten Handles** Flatten the *Bezier* handles for the selected keyframes.

Table  
2.50:  
After  
Flatten  
Handles.



## Mirror

Selected keyframes can be mirrored over different properties using the *Mirror Keys* tool.

### Mirror Keys **Shift-M**

**By Times Over Current Frame** Mirror horizontally over the *Time Cursor*.

**By Values over Cursor Value** Mirror vertically over the *Cursor*.

**By Times over Time 0** Mirror horizontally over frame 0.

**By Values over Value 0** Mirror vertically over value 0.

**By Times over First Selected Marker** Mirror horizontally the over the first selected *Marker*.

## Clean Keyframes

*Clean Keyframes* resets the keyframe tangents to their auto-clamped shape, if they have been modified. *Clean Keyframes* 



## Smoothing

(**Alt-O** or *Key* → *Smooth Keys*) There is also an option to smooth the selected curves, but beware: its algorithm seems to be to divide by two the distance between each keyframe and the average linear value of the curve, without any setting, which gives quite a strong smoothing! Note that the first and last keys seem to be never modified by this tool.



## Sampling and Baking Keyframes

**Sample Keyframes** **Shift-O** Sampling a set a keyframes replaces interpolated values with a new keyframe for each frame.



**Bake Curves** **Alt-C** Baking a curve replaces it with a set of sampled points, and removes the ability to edit the curve.

## 2.6.12 F-Curve Modifiers

F-Curve modifiers are similar to object modifiers, in that they add non-destructive effects, that can be adjusted at any time, and layered to create more complex effects.

## Adding a Modifier

The F-curve modifier panel is located in the Properties panel. Select a curve by selecting one of its curve points, or by selecting the channel list. Click on the *Add Modifier* button and select a modifier.

To add spin to an object or group, select the object/group and add a keyframe to the axis of rotation (x,y, or z)

Go to the Graph Editor.....make sure the f-curves properties panel is visible (View > Properties)

>Add Modifier > (e.g.) Generator

## Types of Modifiers

### Generator

Generator creates a Factorized or Expanded Polynomial function. These are basic mathematical formulas that represent lines, parabolas, and other more complex curves, depending on the values used.

**Additive** This option causes the modifier to be added to the curve, instead of replacing it by default.

**Poly Order** Specify the order of the polynomial, or the highest power of 'x' for this polynomial. (number of coefficients - 1).

Change the Coefficient values to change the shape of the curve.

**See also:**

[The Wikipedia Page](#) for more information on polynomials.

### Built-in Function

These are additional formulas, each with the same options to control their shape. Consult mathematics reference for more detailed information on each function.

- Sine
- Cosine
- Tangent
- Square Root
- Natural Logarithm
- Normalized Sine ( $\sin(x)/x$ )

**Amplitude** Adjusts the Y scaling

**Phase Multiplier** Adjusts the X scaling

**Phase Offset** Adjusts the X offset

**Value Offset** Adjusts the Y offset

### Envelope

Allows you to adjust the overall shape of a curve with control points.

**Reference Value** Set the Y value the envelope is centered around.

**Min** Lower distance from Reference Value for 1 : 1 default influence.

**Max** Upper distance from Reference Value for 1 : 1 default influence.

**Add Point** Add a set of control points. They will be created at the current frame.

**Fra:** Set the frame number for the control point.

**Min** Specifies the lower control point's position.

**Max** specifies the upper control point's position.

## Cycles

Cycles allows you add cyclic motion to a curve that has 2 or more control points. The options can be set for before and after the curve.

### Cycle Mode

**Repeat Motion** Repeats the curve data, while maintaining their values each cycle.

**Repeat with Offset** Repeats the curve data, but offsets the value of the first point to the value of the last point each cycle.

**Repeat Mirrored** Each cycle the curve data is flipped across the X-axis.

**Before/After Cycles** Set the number of times to cycle the data. A value of 0 cycles the data infinitely.

## Noise

Modifies the curve with a noise formula. This is useful for creating subtle or extreme randomness to animated movements, like camera shake.

### Blend Type

**Replace** Adds a -.5 to .5 range noise function to the curve.

**Add** Adds a 0 to 1 range noise function to the curve.

**Subtract** Subtracts a 0 to 1 range noise function to the curve.

**Multiply** Multiplies a 0 to 1 range noise function to the curve.

**Scale** Adjust the overall size of the noise. Values further from 0 give less frequent noise.

**Strength** Adjusts the Y scaling of the noise function.

**Phase** Adjusts the random seed of the noise.

**Depth** Adjusts how detailed the noise function is.

## Python

### Limits

Limit curve values to specified X and Y ranges.

**Minimum/Maximum X** Cuts a curve off at these frames ranges, and sets their minimum value at those points.

**Minimum/Maximum Y** Truncates the curve values to a range.

## Stepped

Gives the curve a stepped appearance by rounding values down within a certain range of frames.

**Step Size** Specify the number of frames to hold each frame

**Offset** Reference number of frames before frames get held. Use to get hold for '1-3' vs '5-7' holding patterns.

**Use Start Frame** Restrict modifier to only act before its 'end' frame

**Use End Frame** Restrict modifier to only act after its 'start' frame

## 2.6.13 The Dopesheet

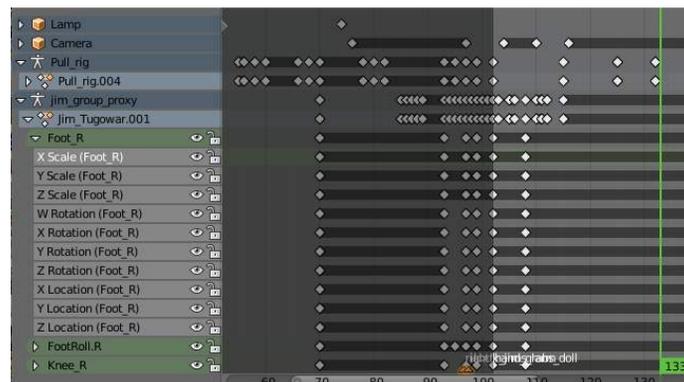


Fig. 2.1394: The DopeSheet

Classical hand-drawn animators often made a chart, showing exactly when each drawing, sound and camera move would occur, and for how long. They nicknamed this the 'dopesheet'. While CG foundations dramatically differ from classical hand-drawn animation, Blender's Dopesheet inherits a similar directive. It gives the animator a 'birds-eye-view' of every thing occurring within a scene.

## Dope Sheet Modes

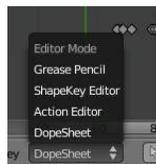


Fig. 2.1395: DopeSheet modes

There are four basic views for the Dopesheet. These all view different contexts of animation:

**DopeSheet** The dopeSheet allow you to edit multiple actions at once.

**Action Editor** *Action Editor* is the default, and most useful one. It's here you can define and control your actions.

**Shape Key Editor** *ShapeKey Editor* is dedicated to the *Shape* Ipo datablocks. It uses/edits the same action datablock as the previous mode. It seems to be an old and useless thing, as the *Action Editor* mode handles *Shape* channels very well, and this mode adds nothing...

**Grease Pencil** *Grease Pencil* is dedicated to the [grease pencil tool](#)'s keyframes - for each grease pencil layer, you have a strip along which you can grab its keys, and hence easily re-time your animated sketches. As it is just another way to see and edit the grease pencil data, this mode uses no datablock (and hence has nothing to do with actions...). Note that you'll have as much top-level grease pencil channels as you have sketched windows (3D views, *UV/Image Editor*, etc.)

## Interface

The *Action Editor* interface is somewhat similar to the *FCurve Editor* one, it is divided in three areas:

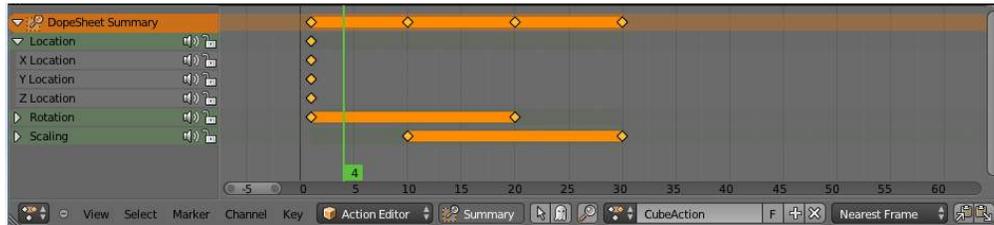


Fig. 2.1396: The Action Editor window, Action Editor mode, with an Object and Shape channels.

**The header bar** Here you find the menus, a first block of controls related to the editor “mode”, a second one concerning the action datablocks, and a few other tools (like the copy/paste buttons, and snapping type).

**The main area** It contains the keyframes for all visible action channels. As with the other “time” windows, the X-axis materializes the time. The Y-axis has no mean in itself, unlike with the *FCurve* editor, it’s just a sort of “stack” of action channels - each one being shown as an horizontal colored strip (of a darker shade “during” the animated/keyed period). On these channel strips lay the keyframes, materialized as light-gray (unselected) or yellow (selected) diamonds. One of the key feature of this window is that it allow you to visualize immediately which channel (i.e. Ipo curve) is *really* affected. When the value of a given channel does not change at all between two neighboring keyframes, a gray (unselected) or yellow (selected) line is drawn between them.

**The left “list-tree”** This part shows the action’s channel “headers” and their hierarchy. Basically, there are:

- “Top-level” channels, which represent whole *FCurve* datablocks (so there’s one for *Object* one, one for *Shape* one, etc.). They gather *all* keyframes defined in their underlying *FCurve* datablock.
- “Mid-level” channels, which seem currently to have no use (there’s one per top-level channel, they are all named *FCurves*, and have no option at all...).
- “Low-level” channels, which represent individual *FCurve*, with their own keyframes (fortunately, only keyed Ipos are shown!).

Each level can be expanded/collapsed by the small arrow to the left of its “parent” channel. To the right of the channel’s headers, there are some channel’s setting controls:

- Clicking on the small “eye” will allow you to mute that channel (and all its “children” channels, if any!).
- Clicking on the small “lock” will allow you to prevent this channel and its children to be edited (note that this is also working inside the NLA, but that it doesn’t prevent edition of the underlying *FCurve* ...).

A channel can be selected (text in white, strip in gray-blue color) or not (text in black, strip in pink-brown color.), use LMB clicks to toggle this state. You can access some channel’s properties by clicking `Ctrl-LMB` on its header. Finally, you can have another column with value-sliders, allowing you to change the value of current keyframes, or to add new ones. These are obviously only available for low-level channels (i.e. individual *FCurve*). See [View Menu](#) below for how to show these sliders.

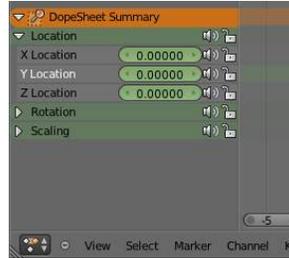


Fig. 2.1397: the action editor showing sliders

## View Menu

**Realtime Updates** When transforming keyframes, changes to the animation data are flushed to other views

**Show Frame Number Indicator** Show frame number beside the current frame indicator line

**Show Sliders** A toggle option that shows the value sliders for the channels. See the *The Action Editor window, Action Editor mode, with a group and sliders* picture above).

**Use Group Colors** Draw groups and channels with colors matching their corresponding groups.

**AutoMerge Keyframes** Automatically merge nearby keyframes

**Sync Markers** Sync Markers with keyframe edits

**Show Seconds** Whether to show the time in the X-axis as frames or as seconds

**Set Preview Range P** Interactively define frame range used for playback. Allow you to define a temporary preview range to use for the `Alt-A` realtime playback (this is the same thing as the *Playback Range* option of the *timeline window header*).

**Clear Preview Range Alt-P** Clears the preview range

**Auto-Set Preview Range** Automatically sets the preview range to playback the whole action.

## Marker Menu

See the [Markers](#) page.

## 2.6.14 Dope Sheet

### 2.6.15 Action Editor

In Blender *Actions* are a generic containers for F-Curves. Actions can contain any number of F-Curves, and can be attached to any data block. As long as the RNA data paths stored in the Action's F-Curves can be found on that data block, the animation will work. For example, an action modifying 'X location' and 'Y location' properties can be shared across multiple objects, since both objects have 'X location' and 'Y location' properties beneath them.

The *Action Editor* window enables you to see and edit the F-Curve datablocks you defined as actions in the *F-Curve Editor* window. So it takes place somewhere in-between the low-level [F-Curves](#), and the high-level [NLA editor](#). Hence, you do not have to use them for simple Ipo curves animations - and they have not much interest in themselves, so you will mostly use this window when you do [NLA animation](#) (they do have a few specific usages on their own, though, like e.g. with the [Action constraint](#), or the [pose libraries](#)).

This is not a mandatory window, as you do can edit the actions used by the NLA directly in the *FCurve Editor* window (or even the *NLA Editor* one). However, it gives you a slightly simplified view of your FCurve datablocks (somewhat similar to the “key” mode of the FCurve window, even though more powerful in some ways) - and, more interesting, it can show you all “action” FCurve datablocks of a same object at once.

Additionally, it also allows you to affect timing of the different keys of the layers created with the [grease pencil tool](#).

Each “action” FCurve datablock forms a top-level channel (see below). Note that an object can have several *Constraint* (one per animated constraint) and *Pose* (for armatures, one per animated bone) FCurve datablocks, and hence an action can have several of these channels.

## Action Datablocks

As everything else in Blender, actions are datablocks. Unlike FCurve ones, there is only one type of action, which can regroup all FCurve of a given object. You’ll find its usual datablock controls in the *Action Editor* header.

However, there is one specificity with action datablocks: they have by default a “fake user”, i.e. once created, they are always kept in Blender file, even if no object uses them. This is due to the fact that actions are designed to be used in the NLA, where you can affect several different actions to a same object! Yes, this is the only way to use different actions (and hence, different FCurve datablocks of the same kind) to animate a same object. But as you have to assign an action to an object to be able to edit it (and an object can only have one action datablock at a time), to have “fake users” guaranties you that you won’t lost your precious previously-edited actions when you start working on a new one!

This window shows, by default, the action datablock linked to the current active object. However, as with FCurvs, you can pin an *Action Editor* to a given action with the small “pin” button to the left of the datablock controls, in the header. This will force the window to always display this datablock, whatever the current selected object is.

## Channel Menu

### Delete (X)

Deletes the whole channel from the current action (i.e. unlink the underlying FCurve datablock from this action datablock).

**Warning:** The X shortcut is area-dependent: if you use it in the left list part, it’ll delete the selected channels, whereas if you use it in the main area, it’ll delete the selected keyframes...

**Settings** → **Toogle/Enable/Disable a Setting (Shift-W / Ctrl-Shift-W / Alt-W)** Enable/disable a channel’s setting (selected in the menu that pops-up) - currently, “lock” and/or “mute” only.

**Toggle Channel Editability Tab** Locks or unlocks a channel for editing

**Extrapolation Mode** Change the extrapolation between selected keyframes. More options are available in the Graph Editor.

**Expand Channels, Collapse Channels (NumpadPlus, NumpadMinus)** Expands or collapses selected channels.

**Move...** This allows you to move top-level channels up/down (Shift-PageUp / Shift-PageDown), or directly to the top/bottom (Ctrl-Shift-PageUp / Ctrl-Shift-PageDown).

**Revive Disabled F-Curves** Clears ‘disabled’ tag from all F-Curves to get broken F-Curves working again

## 2.6.16 Shape Key

## 2.6.17 Grease Pencil

## 2.6.18 Non-Linear Animation Editor

The NLA editor can manipulate and repurpose actions, without the tedium of keyframe handling. Its often used to make broad, significant changes to a scene's animation, with relative ease. It can also repurpose, and 'layer' actions, which make it easier to organize, and version-control your animation.

### Tracks

Tracks are the layering system of the NLA. At its most basic level, it can help organize strips. But it also layers motion much like an image editor layers pixels - the bottom layer first, to the top, last.



### Strips

There's three kinds of strips - Action, Transition, and Meta. Actions contain the actual keyframe data, Transitions will perform calculations between Actions, and Meta will group strips together as a whole.

### Creating Action Strips

Any action used by the NLA first must be turned into an Action strip. This is done so by clicking the



Fig. 2.1398: next to the action listed in the NLA. Alternatively, you can go to

### Reference

Menu: Add -> Action

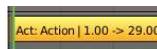


Fig. 2.1399: Action Strip.

### Creating Transition Strips

Select two or more strips on the same track, and go to

### Reference

Menu: Add -> Transition



Fig. 2.1400: Transition Strip.

### Grouping Strips into Meta Strips

If you find yourself moving a lot of strips together, you can group them into a Meta strip. A meta strip can be moved and duplicated like a normal strip.

#### Reference

Menu: Add -> Add Meta-Strips

Hotkey: Shift-G



A meta strip still contains the underlying strips. You can ungroup a Meta strip.

#### Reference

Menu: Add -> Remove Meta-Strips

Hotkey: Alt-G

### Editing Strips

The contents of Action strips can be edited, but you must be in 'Tweak Mode' to do so.

#### Reference

Menu: View -> Enter Tweak Mode

Hotkey: Tab



If you try moving the strip, while in edit mode, you'll notice that the keys will go along with it. On occasion, you'll prefer the keys to remain on their original frames, regardless of where the strip is. To do so, hit the 'unpin' icon, next to the strip.

When your finished editing the strip, simply go to View > Exit Tweak Mode. Note the default key for this is Tab.

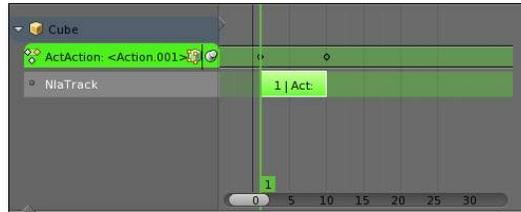


Fig. 2.1409: Nla strip with pinned keys.



Fig. 2.1410: Strip moved, notice the keys move with it.

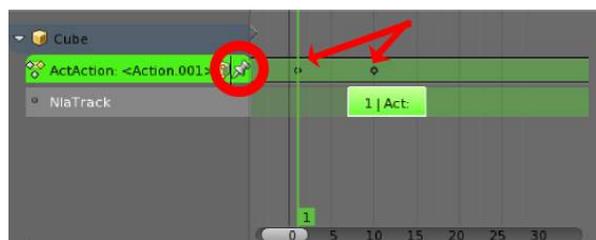


Fig. 2.1411: The unpinned keys return to their original frames.

## Re-Instancing Strips

The contents' of one Action strip can be instanced multiple times. To instance another strip, select a strip, go to

---

### Reference

Menu: Edit-> Duplicate Strips

---

Now, when any strip is tweaked, the others will change too. If a strip other than the original is tweaked, the original will turn to red.



## Strip Properties

Strip properties can be accessed via the NLA header.

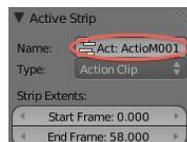
---

### Reference

Menu: View-> Properties

## Renaming Strips

All strips can be renamed, in the “Active Track” section in the Strip Properties.



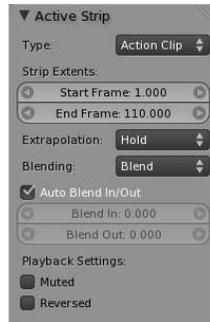
## Active Track

This is which track the strip currently belongs to.



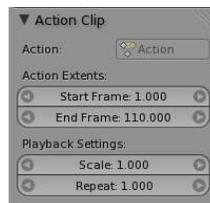
## Active Strip

Elements of the strip itself. An Action Strip can be either an Action Clip, or a Transition Clip. Note that the ‘Strip Extents’ fields determine strictly the strip, and not the action. Also, the “Hold” value in the Extrapolation section means hold both beginning, and after. This can cause previous clips to not work, if checked.



## Active Action

This represents the ‘object data’ of the strip. Much like the transform values of an object.



## Evaluation

This determines the degree of influence the strip has, and over what time.



If influence isn’t animated, the strips will fade linearly, during the overlap.

## Strip Modifiers

Like its close cousins in mesh and graph editing, Modifiers can stack different combinations of effects for strips. Obviously there will be more to come on this.

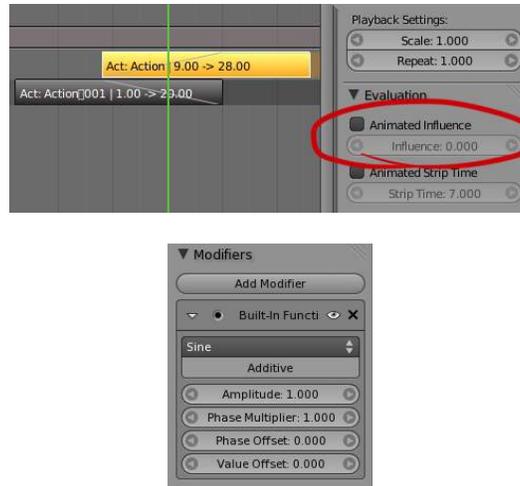
### 2.6.19 Techs

### 2.6.20 Object

### 2.6.21 Using Constraints in Animation

Constraints are a way to control an object’s properties (its location/rotation/scale), using either plain static values (like the “limit” ones), or (an)other object(s), called “targets” (like e.g. the “copy” ones).

Even though these constraints might be useful in static projects, their main usage is obviously in animation. There are two different aspects in constraints’ animation:



- You can control an object’s animation through the targets used by its constraints (this is a form of indirect animation).
- You can animate constraints’ settings

## Controlling Animation with Constraints

This applies only to constraints using target(s). Indeed, these targets can then control the constraint’s owner’s properties, and hence, animating the targets will indirectly animate the owner.

This indirect “constraint” animation can be very simple, like for example with the [Copy Location constraint](#), where the owner object will simply copy the location of its target (with an optional constant offset). But you can also have very complex behaviors, like when using the [Action constraint](#), which is a sort of [Animation Driver](#) for actions!

We should also mention the classical [Child Of constraint](#), which creates parent/child relationship. These relationships indeed imply indirect animation (as transforming the parent affects by default all its children). But the *Child Of* constraint is also very important, as it allows you to parent your objects to bones, and hence use [Armatures](#) to animate them!

Back to our simple *Copy Location* example, you can have two different behaviors of this constraint:

- When its *Offset* button is disabled (the default), the location of the owner is “absolutely” controlled by the constraint’s target, which means nothing (except other constraints below in the stack...) will be able to control the owner’s position. Not even the object’s animation curves.
- However, when the *Offset* button is enabled, the location of the owner is “relatively” controlled by the constraint’s target. This means that location’s properties of the owner are offset from the location of the target. And these owner’s location properties can be controlled e.g. by its *Loc...* curves (or actions, or NLA...)!

## Example

Let’s use the *Copy Location* constraint and its *Offset* button. For example, you can make your owner (let’s call it `moon`) describe perfect circles centered on the  $(0.0, 0.0, 0.0)$  point (using e.g. pydriven *LocX / LocY* animation curves, see [this page](#)), and then make it copy the location of a target (called, I don’t know... `earth`, for example) - with the *Offset* button enabled. Congratulations, you just modeled a satellite in a (simplified) orbit around its planet... Just do the same thing with its planet around its star (which you might call `sun`, what do you think?), and why not, for the star around its galaxy...

Here is a small animation of a “solar” system created using (among a few others) the technique described above:

Note that the this “solar” system is not realistic at all (wrong scale, the “earth” is rotating in the wrong direction around the “sun”, ...).

You can download the `.blend` file ([download here](#)) used to create this animation.

## Animating Constraints Influence

More “classically”, you can also animate a few properties of each constraint using animation curves.

You only have two animation curves (see also: [Graph Editor](#)):

- You can animate the *Influence* of a constraint. For example, in the *Example* above, I used it to first stick the camera to the “moon”, then to the “earth”, and finally to nothing, using two *Copy Location* constraints with *Offset* set, and their *Influence* cross-fading together...
- More anecdotal, you can also, for some constraints using an armature’s bone as target, animate where along this bone (between root and tip) lays the real target point (0 . 0 means “full-root”, and 1 . 0, “full-tip”).

## 2.6.22 Moving Objects on a Path

To make objects move along a path is a very common animation need. Think of a complex camera traveling, a train on his rails - and most other vehicles can also use “invisible” tracks! -, the links of a bicycle chain, etc. All these movements could obviously be done with standard Ipo curves, but this would be a nightmare! It’s much more easy and intuitive to define a path materializing the desired movement, and make your object(s) follow it.

Blender features you two different constraints to make an object follow a path, which have different ways to determine/animate the position of their owner along their path.

In Blender, any [curve object](#) can become a path. A curve becomes a path when its *Path Animation* button is enabled in the *Curve* data panel, but you don’t even have to bother about this: once a curve is selected as target for a “path” constraint, it automatically is enabled.

You can also directly add a “path” from the *Add → Curve → Path* menu entry (in a 3D view). This will insert in your scene a *three-dimensional* NURBS curve. This is an important point: by default, Blender’s curve are 2 dimensional, i.e. are laid on a plane, which is often not the desired behavior of a path. To turn a standard curve three-dimensional, enable its *3D* button, in the same *Curve and Surface* editing panel.

One last curve property that is important for a path is its *direction*, which is, for three-dimensional ones, materialized by its small arrows. You can switch it with the *Curve → Segments → Switch Direction* menu entry (or `W, 2`).

For more on editing path/curves, see the [modeling chapter](#).

---

### Note: Shapes on Curves

If you would rather like to have your object’s *shape* follow a path (like e.g. a sheet of paper inside a printer), you should use the [Curve Modifier](#)

---

## Parenting Method

Older versions of Blender did not have constraints to make an object follow a path. They used a different method (deprecated, but still available), based on parenting.

To use this method, select the object that will follow the path, then `Shift` select the curve, and use `Ctrl-P` to bring up the parenting menu. Choose *Follow Path*. The object will now be animated along the path.

The settings for the path animation are in the *Path Animation* panel of the Curve properties panel.

**Frames** Defines the number of frames it takes for the object to travel the path.

**Evaluation Time** Defines current frame of the animation. By default it is linked to the global frame number, but could be keyframed to give more control over the path animation.

**Follow** Causes the curve path children to rotate along the curvature of the path.

**Radius** Causes the curve path child to be scaled by the set curve radius. See [Curve Extruding](#)

**Offset Children** Causes the animation to be offset by the curve path child's time offset value, which can be found in its *Animation Hacks* section of the *Object Panel*.

### The Follow Path Constraint

The *Follow Path* constraint implements the most “classical” technique. By default, the owner object will walk the whole path only once, starting at frame one, and over 100 frames. You can set a different starting frame in the *Offset* field of the constraint panel, and change the length (in frames) of the path using its *Frames* property (*Curve and Surface* panel).

But you can have a much more precise control over your object's movement along its path by keyframing or defining a *Speed* animation curve for the path's *Evaluation Time* attribute. This curve maps the current frame to a position along the path, from 0.0 (start point) to 1.0 (end point).

For more details and examples, see the [Follow Path constraint page](#).

### The Clamp To Constraint

Another method of keeping objects on a path is to use the *Clamp To* constraint, which implements a more advanced technique. To determine where along the path should lay its owner, it uses the *location of this owner* along a given axis. So to animate the movement of your owner along its target path, you have to animate some way (Ipo curves or other indirect animation) its location.

This implies that here, the length of the path have no more any effect - and that by default, the object is static somewhere on the path!

For more details and examples, see the [Clamp To constraint page](#).

## 2.6.23 Shape

### 2.6.24 Shape Keys

#### Introduction

*Shape Keys* are used on Objects like *Mesh*, *Curve*, *Surface*, *Lattice*. They are used to deform the object vertices into a new shape.

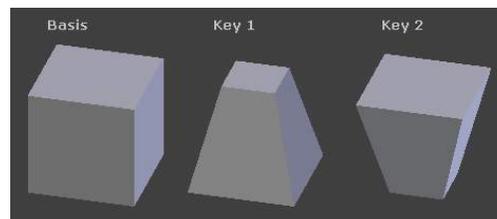


Fig. 2.1418: A mesh with different shape keys applied.

There are two types of Shape Keys.

**Relative** Which are relative to the Basis or selected shape key. They are mainly used as, for limb joints, muscles, or Facial Animation.

**Absolute** Which are relative to the previous and next shape key. They are mainly used to deform the objects into different shapes over time.

The shape key data, the deformation of the objects vertices, is usually modified in the 3D View by selecting a shape key, then moving the object vertices to a new position.

## Shape Keys Panel

### Reference

Mode: All modes

Panel: *Properties, Object Data, Shape Keys*

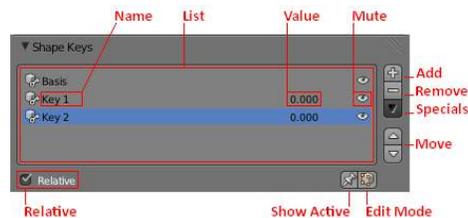


Fig. 2.1419: Shape Keys. Options.

**Relative** Set the shape keys to Relative or Absolute.

**Name** Name of the Shape Key.

**Value** Current Value of the Shape Key (0.0 to 1.0).

**Mute** This visually disables the shape key in the 3D view.

**Add** Add a new shape key to the list.

**Remove** Remove a shape key from the list.

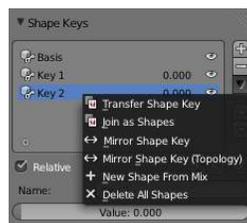


Fig. 2.1420: Shape Keys Specials.

**Specials** A menu with some operators.

**Transfer Shape Key** Transfer the active 'Shape Key' from a different object. Select two objects, the active Shape Key is copied to the active object.

**Join as Shapes** Transfer the 'Current Shape' from a different object. Select two objects, the Shape is copied to the active object.

**Mirror Shape Key** If your mesh is nice and symmetrical, in *Object Mode*, you can mirror the shape keys on the X axis. This won't work unless the mesh vertices are perfectly symmetrical. Use the *Mesh* → *Symmetrize* function in *Edit Mode*.

**Mirror Shape Key (Topology)** This is the same as *Mirror Shape Key* though it detects the mirrored vertices based on the topology of the mesh. The mesh vertices don't have to be perfectly symmetrical for this one to work.

**New Shape From Mix** Add a new shape key with the current deformed shape of the object.

**Delete All Shapes** Delete all shape keys.

**Move** Move shape key up or down in the list.

**Show Active** Show the shape of the active shape key in the 3D View. *Show Active* is enabled while the object is in *Edit Mode*, unless the setting below is enabled.

**Edit Mode** Modify the shape key settings while the object is in *Edit mode*.

## Relative Shape Keys

Relative shape keys deform from a selected shape key. By default all relative shape keys deform from the first shape key called the Basis shape key.

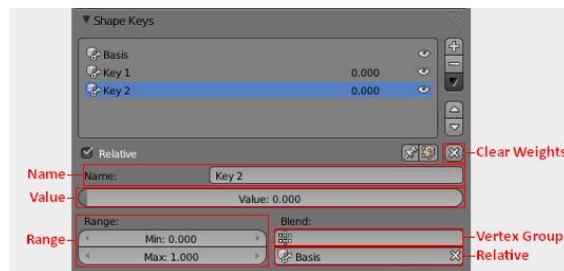


Fig. 2.1421: Relative Shape Keys. Options.

**Clear Weights** Set all values to 0.

**Name** Name of the active shape key.

**Value** Value of the active shape key.

**Range** Min and Max range of the active shape key value.

**Vertex Group** Limit the active shape key deformation to a vertex group.

**Relative** Select the shape key to deform from.

## Absolute Shape Keys

Absolute shape keys deform from the previous and to the next shape key. They are mainly used to deform the object into different shapes over time.

**Reset Timing** Reset the timing for absolute shape keys. For example, if you have the shape keys, Basis, Key\_1, Key\_2, in that order.

**Reset Timing will loop the shapekeys, and set the shape key frames to +0.1.** Basis 0.1 Key\_1 0.2 Key\_2 0.3

**Evaluation Time will show this as frame\*100.** Basis 10.0 Key\_1 20.0 Key\_2 30.0

**Name** Name of the active shape key.



Fig. 2.1422: Absolute Shape Keys. Options.

**Interpolation** This controls the interpolation between shape keys.



Fig. 2.1423: Different types of interpolation.

**Evaluation Time** This is used to control the shape key influence.

For example, if you have the shape keys, **Basis**, **Key\_1**, **Key\_2**, in that order, and you reset timing. Basis 10.0  
Key\_1 20.0 Key\_2 30.0

You can control the shape key influence with Evaluation Time. Here keyframes have been used to control Evaluation Time for animation.

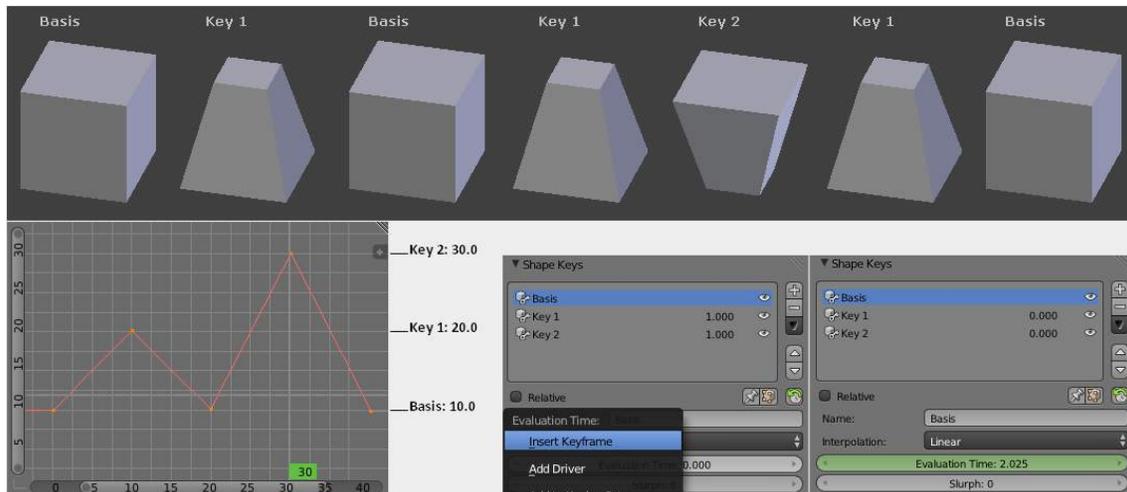


Fig. 2.1424: Animation with Evaluation Time.

## Workflow For Relative Shape Keys

This example shows you how to make a cube mesh transform in to a sphere.

- In *Object Mode* add two shape keys via the *Shape Key Panel*.
- *Basis* is the rest shape. *Key 1* will be the new shape.
- With *Key 1* selected, switch to *Edit Mode*.

- Press `Shift-Alt-S To Sphere`, move the mouse right, then LMB.
- Switch to *Object Mode*.
- Set the *Value* for *Key 1* to see the transformation between the shape keys.

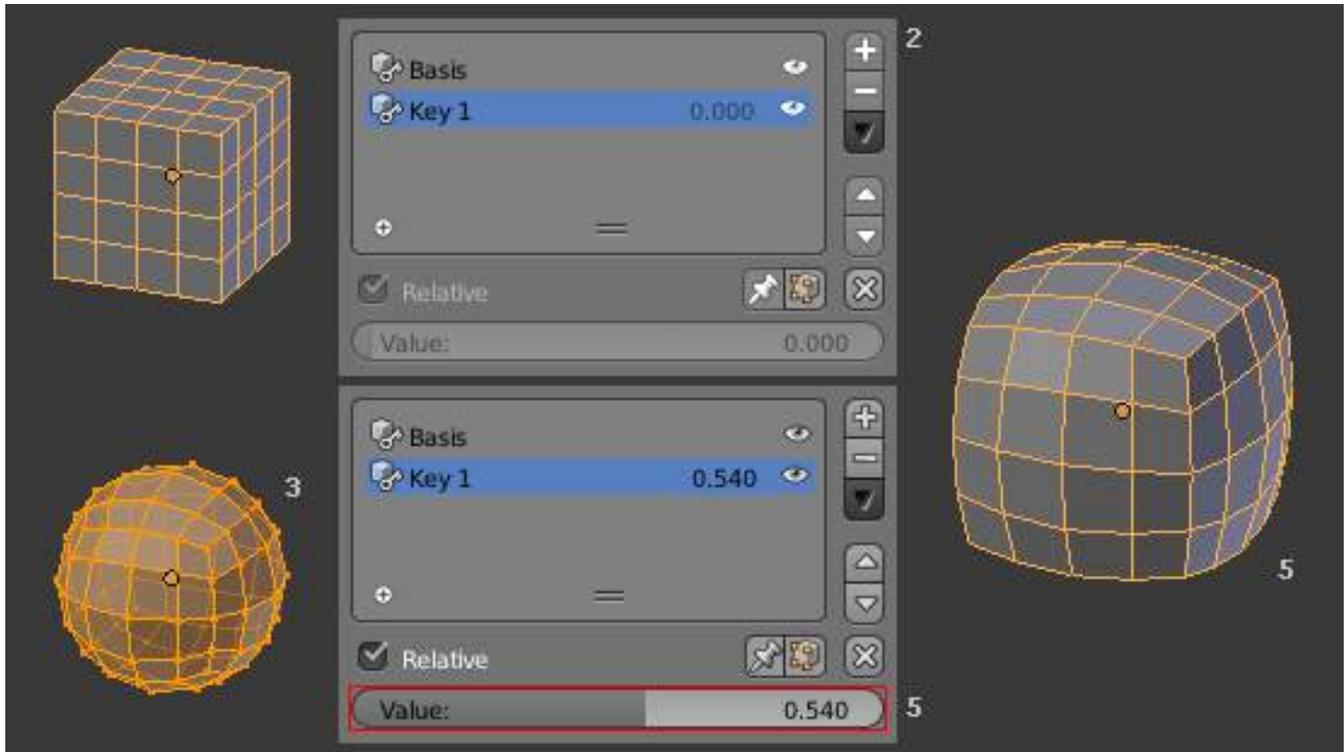
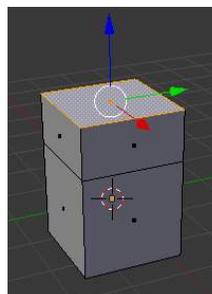


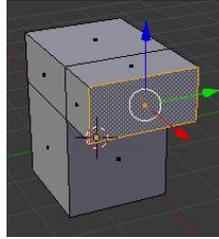
Fig. 2.1425: Shape Key workflow.

### Workflow For Absolute Shape Keys

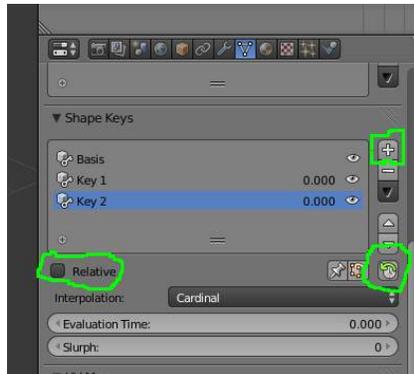
- Select the default Cube.
- Switch to Edit Mode.
- Switch to Face Select mode (if you are not already in it)



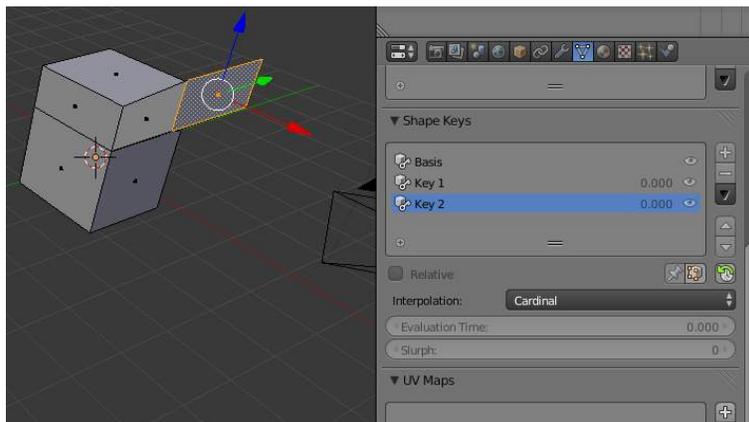
- Select the top face.
- Extrude up E 1 LMB.



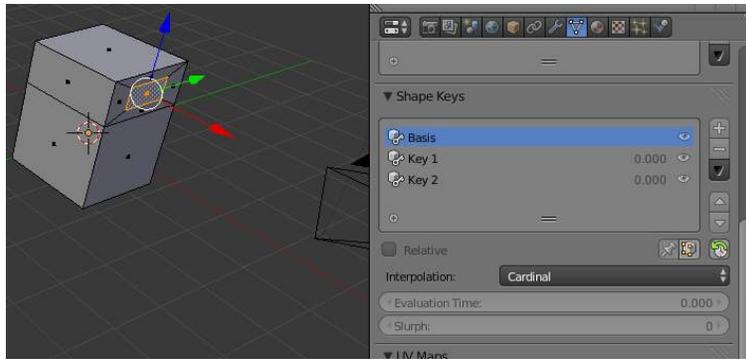
- Select a side face on the top half. (the one at x=1 if possible)
- Extrude out E 1 LMB.
- Switch back to Object Mode.



- Add a basis shape keys and two more via the + button on the Shape Key Panel.
- Uncheck the Relative checkbox.
- Click the Reset Timing button.
- Switch to Edit Mode.



- Select shape key Key 2 to edit the third shape key.
- Select the extruded side face and G Z 1 LMB
- Select shape key Basis to edit the first shape key.



- Select the extruded size face and S 0.5 LMB, then G X Minus1 LMB.
- Switch to Object Mode.
- Drag the Evaluation Time slider to make its value vary from 10 to 30.

### More Details On Absolute Shape Keys

The thing to remember about absolute shape keys is that they are incomplete until you click the Reset Timing button. When you create a shape key its “frame” property is zero (<https://developer.blender.org/T39897>), which is a completely useless value. This frame value is not displayed on the UI so you can’t easily tell if something is wrong or screwy until your animation starts misbehaving.

The number displayed to the right of the key name is the value and is used in relative shape keys. It has no effect on absolute shape keys, so ignore it.

When you reset the timings blender iterates through the shape keys assigning them frame values incrementing by 0.1 from key to key.

name	frame	evaluation time
Basis	0.1	10
Key 1	0.2	20
Key 2	0.3	30
Key 3	0.4	40

If you delete a shape key this does not automatically alter the frame values assigned to remaining shape keys.

name	frame	evaluation time
Basis	0.1	10
Key 1	0.2	20
Key 3	0.4	40

The Evaluation Time is how you choose which shape key is active, and how active it is. The interesting values range from 10 .. (n\*10) where n is the number of shape keys. (assuming you have not deleted or added any keys since the last Reset Timing). If you are using shape keys for animation, 99% of the time you will be putting keyframes on this Evaluation Time field.

Remember: if you are having problems with your absolute shape keys, there is a good chance that you need to Reset Timing.

### Shape Key Operators

**3D View > Edit Mode > Header > Mesh > Vertices > Shape Propagate** Apply selected vertex locations to all other shape keys.

**3D View > Edit Mode > Header > Mesh > Vertices > Blend From Shape** Blend in shape from a shape key.

### See Also

- 2.4 Shape Keys
- 2.4 Editing Shape Keys
- 2.4 Animating Shape Keys
- 2.4 Shape Keys Examples
- Add-on: Corrective Shape Key

## 2.6.25 Editing

## 2.6.26 Animating

## 2.6.27 Examples

## 2.6.28 Indirect Animation

## 2.6.29 Armatures

## 2.6.30 Motion Capture

## 2.6.31 Animating Lamps Properties

As of Blender 2.5, Everything is animatable. Read more about keyframes [Here](#).

### Example

Let's illustrate this with a flying torch deep in a cave.

We won't detail the cave and torch creation - the first one is an deformed icosphere with *Subsurf* and *Displace* modifiers, and the second one, a cylinder scaled and subdivided several times in its length, with a *particle system* to materialize its fire.

The torch will be the only light source of the scene. Add four *Lamp* lamps, all using the same lamp datablock. Place them around the tip of the torch, and parent them to it. Give them an orange color (e.g. (1.0, 0.8, 0.4)), a short *Distance* (2.0, but this depends on the size of your cave!), and an *Inverse Square* falloff. Also let them cast ray shadows (soft shadows, if you have enough computing power).

Right click on the *Energy* parameter and *Insert Keyframe* to create an Fcurve, then open the graph editor to edit the keyframes. You can for example start at zero (no energy, the scene is whole black), give a short and intense flash (10 over a few frames) to simulate a sort of lightning lighting the fire, then back to very low (0.25), and then a gently varying curve over the rest of the scene, to simulate the irregularities of the flames.

You might also use the same animation to control the particle system emission, to synchronize the quantity of particles with the luminosity of the lamps.

Once your torch is flying, you should get something as shown below - you can download the blend file [here](#).

## 2.6.32 Animating Cameras

As of Blender 2.5, *Everything is animatable*. Read more about keyframes [Here](#).

### Example

As an example, we are going to create a nice and impressive camera effect, which you can see e.g. in the first part of the Lord of the Ring: the *transtrav*. Basically, the idea is to combine a forward zoom with a backward traveling (or conversely), both controlled such as the point of interest keeps its scale in the image, while its environment scales up or down, depending whether it is nearer or more far from the camera...

Create a scene with a ground, and some objects laying on it.

Add a camera, place it as you like for the beginning of the transtrav (your “key” object should be more or less at the center of the picture, it’s easier to handle!). As we are going to do a “forward” transtrav, you should use a quite long lens at start. Go to frame 10, and insert a keyframe for both the location (and optionally the rotation) of the camera and its focal length.

Now, go to frame 140, and move forward your camera to your key object. Insert another keyframe for its position, and adjust its focal length until your key object have the same visual dimensions as at the beginning. Add a key to both attributes.

This won’t give you a fully perfect transtrav - to get such one, you would have to dive into trigonometric maths... But the result is visually quite satisfying.

## 2.6.33 Animating Material Attributes

As of Blender 2.5, *Everything is animatable*. Read more about keyframes [Here](#).

Before reading this page, you should know about Blender’s materials - if not, read [this chapter](#) first!

Animated materials can be a very powerful tool, for many different purposes. For example, you can use them to simulate the color changes of a chameleon’s skin, a video screen lighting up, the surface of a river or lake, a light lighting up (with an halo material), etc., etc.

The possibilities are nearly unlimited. For example, you could keyframe the *Glossiness* of raytraced reflection/transparency, which would enable the simulation of condensation spreading over a mirror or window...).

### Example

As an illustration, we’ll create a simple “psychedelic” background. This obviously won’t demonstrate all possibilities of material animation - but I think this would need at least a whole book!

Add a plane and a camera, such that the plane faces the camera and covers the whole view.

Add a material to the plane. As we won’t use any light, set its *Emit* value to 1.0.

Create Fcurves for *R*, *G* and *B*, with a few random control points, all in the  $[0.0, 1.0]$  range. Manage to have three different length between the first and last keyframes, and enable the *Cyclic* extend mode (E-Numpad2). This way, with the three curves cycling over various periods, you’ll get a never-the-same color animation! Unless you want to get a “time-tileable” animation, in which case you should manage to get exactly the same color at start and end... You can also create an *Emit* Fcurve, e.g. to create a fade in/out...

Now, let’s add a bit of fun in this plain colored background. Add a texture to the material and, in the *Texture* sub-context, select a procedural texture (*DistortedNoise*, for example, but any one will work - follow your taste!), and set it to your liking.

Back in the *Material* sub-context, choose to what you want to map the texture - for this example, I chose to map it to diffuse color in *Difference* mode, in a first texture channel, and to emit value in *Mul* tiply mode, in a second texture channel.

Finally, animate the Z offset of the mapping of both channels (define a first *OfsZ* Fcurve, and use the copy/paste buttons to exactly copy it to the second texture channel's curve). Here again, you can either have two different values for start and end, or the same if you want a cyclic animation...

Usually, you will create a slow, linear variation of the Z offset (i.e. a straight curve with low gradient), e.g. a decay of 1.0 over 500 to 1000 frames, but the only way to find the good value is to make preview renders!

You should get something similar to what shown below. You can download the blend file [here](#).

## 2.7 Physics

### 2.7.1 Introduction to Physics Simulation

This chapter covers various advanced Blender effects, often used to simulate real physical phenomena, such as:

- Smoke
- Rain
- Dust
- Cloth
- Water
- Jello

**Particle Systems** can be used to simulate many things: hair, grass, smoke, flocks.

**Hair** is a subset of the particle system, and can be used for strand-like objects, such as hair, fur, grass, quills, etc.

**Soft Bodies** are useful for everything that tends to bend, deform, in reaction to forces like gravity or wind, or when colliding with other objects... It can be used for skin, rubber, and even clothes, even though there is separate **Cloth Simulation** specific for cloth-like objects.

**Rigid Bodies** can simulate dynamic objects that are fairly rigid.

**Fluids**, which include liquids and gasses, can be simulated, including **Smoke**.

**Force Fields** can modify the behavior of simulations.

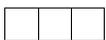
### Gravity

Gravity is a global setting that is applied the same to all physics systems in a scene, which can be found in the scene tab. This value is generally fine left at its default value, at  $-9.810$  in the Z axis, which is the force of gravity in the real world. Lowering this value would simulate a lower or higher force of gravity. Gravity denoted  $g$ , measurement [ $m \times s^{-2}$ ]).

Gravity is practically same around whole **Earth**. For rendering scenes from **Moon** use value 6 times smaller, e.g.  $1.622 m \times s^{-2}$ . The most popular and probably not colonized **Mars** has  $g = 3.69$ .

Note that you can scale down the gravity value per physics system in the *Field Weights tab*.

### 2.7.2 Cloth Simulation



Cloth simulation is one of the hardest aspects of CG, because it is a deceptively simple real-world item that is taken for granted, yet actually has very complex internal and environmental interactions. After years of development, Blender has a very robust cloth simulator that is used to make clothing, flags, banners, and so on. Cloth interacts with and is affected by other moving objects, the wind and other forces, as well as a general aerodynamic model, all of which is under your control.

## Description

A piece of cloth is any mesh, open or enclosed, that has been designated as cloth. The *Cloth* panels are located in the *Physics* sub-context and consist of three panels of options. Cloth is either an open or closed mesh and is mass-less, in that all cloth is assumed to have the same density, or mass per square unit.

Cloth is commonly modeled as a mesh grid primitive, or a cube, but can also be, for example, a teddy bear. However, Blender's *Softbody system* provides better simulation of closed meshes; Cloth is a specialized simulation of fabrics.

Once the object is designated as Cloth, a Cloth *modifier* will be added to the object's modifier stack automatically. As a *modifier* then, it can interact with other modifiers, such as *Armature* and *Smooth*. In these cases, the ultimate shape of the mesh is computed in accordance with the order of the modifier stack. For example, you should smooth the cloth *after* the modifier computes the shape of the cloth.

So you edit the Cloth settings in two places: use the Physics buttons to edit the properties of the cloth and use the Modifier stack to edit the Modifier properties related to display and interaction with other modifiers.

You can *Apply* the cloth modifier to freeze, or lock in, the shape of the mesh at that frame, which removes the modifier. For example, you can drape a flat cloth over a table, let the simulation run, and then apply the modifier. In this sense, you are using the simulator to save yourself a lot of modeling time.

Results of the simulation are saved in a cache, so that the shape of the mesh, once calculated for a frame in an animation, does not have to be recomputed again. If changes to the simulation are made, you have full control over clearing the cache and re-running the simulation. Running the simulation for the first time is fully automatic and no baking or separate step interrupts the workflow.

Computation of the shape of the cloth at every frame is automatic and done in the background; thus you can continue working while the simulation is computed. However it is CPU-intensive and depending on the power of your PC and the complexity of the simulation, the amount of CPU needed to compute the mesh varies, as does the lag you might notice.

---

**Note:** Don't jump ahead

If you set up a cloth simulation but Blender has not computed the shapes for the duration of the simulation, and if you jump ahead a lot of frames forward in your animation, the cloth simulator may not be able to compute or show you an accurate mesh shape for that frame, if it has not previously computed the shape for the previous frame(s).

---

## Workflow

A general process for working with cloth is to:

- Model the cloth object as a general starting shape.
- Designate the object as a "cloth" in the *Physics* tab of the *Properties* window.
- Model other deflection objects that will interact with the cloth. Ensure the Deflection modifier is last on the modifier stack, after any other mesh deforming modifiers.
- Light the cloth and assign materials and textures, UV-unwrapping if desired.
- If desired, give the object particles, such as steam coming off the surface.
- Run the simulation and adjust Options to obtain satisfactory results. The timeline window's VCR controls are great for this step.

- Optionally age the mesh to some point in the simulation to obtain a new default starting shape.
- Make minor edits to the mesh on a frame-by-frame basis to correct minor tears.

---

**Tip:** To avoid unstable simulation, ensure that the cloth object doesn't penetrate any of the deflection objects or an unstable simulation will result.

---

## Creating Cloth Simulations

This section discusses how to use those options to get the effect you want. First, enable *Cloth*. Set up for the kind of cloth you are simulating. You can choose one of the presets to have a starting point.

As you can see, the heavier the fabric, the more stiff it is and the less it stretches and is affected by air.

### Cloth Panel

**Presets** Contains a number of preset cloth examples, and allows you to add your own.

**Quality** Set the number of simulation steps per frame. Higher values result in better quality, but is slower.

### Material

**Mass** The mass of the cloth material.

**Structural** Overall stiffness of the cloth.

**Bending** Wrinkle coefficient. Higher creates more large folds.

### Damping

**Spring** Damping of cloth velocity. Higher = more smooth, less jiggling.

**Air** Air normally has some thickness which slows falling things down.

### Pinning

The first thing you need when pinning cloth is a [Vertex Group](#). There are several ways of doing this including using the [Weight Paint](#) tool to paint the areas you want to pin (see the [Weight paint](#) section of the manual). The weight of each vertex in the group controls how strongly it is pinned.

Once you have a vertex group set, things are pretty straightforward; all you have to do is press the *Pinning of cloth* button in the *Cloth* panel and select which vertex group you want to use, and the stiffness you want it at.

**Stiffness** Target position stiffness. You can leave the stiffness as it is; the default value of 1 is fine.

### Pinning Clothing To An Armature

Clothing can be simulated and pinned to an armature. For example, a character could have a baggy tunic pinned to the character's waist with a belt.

The typical workflow for pinning:

- Set the armature to its bind pose.

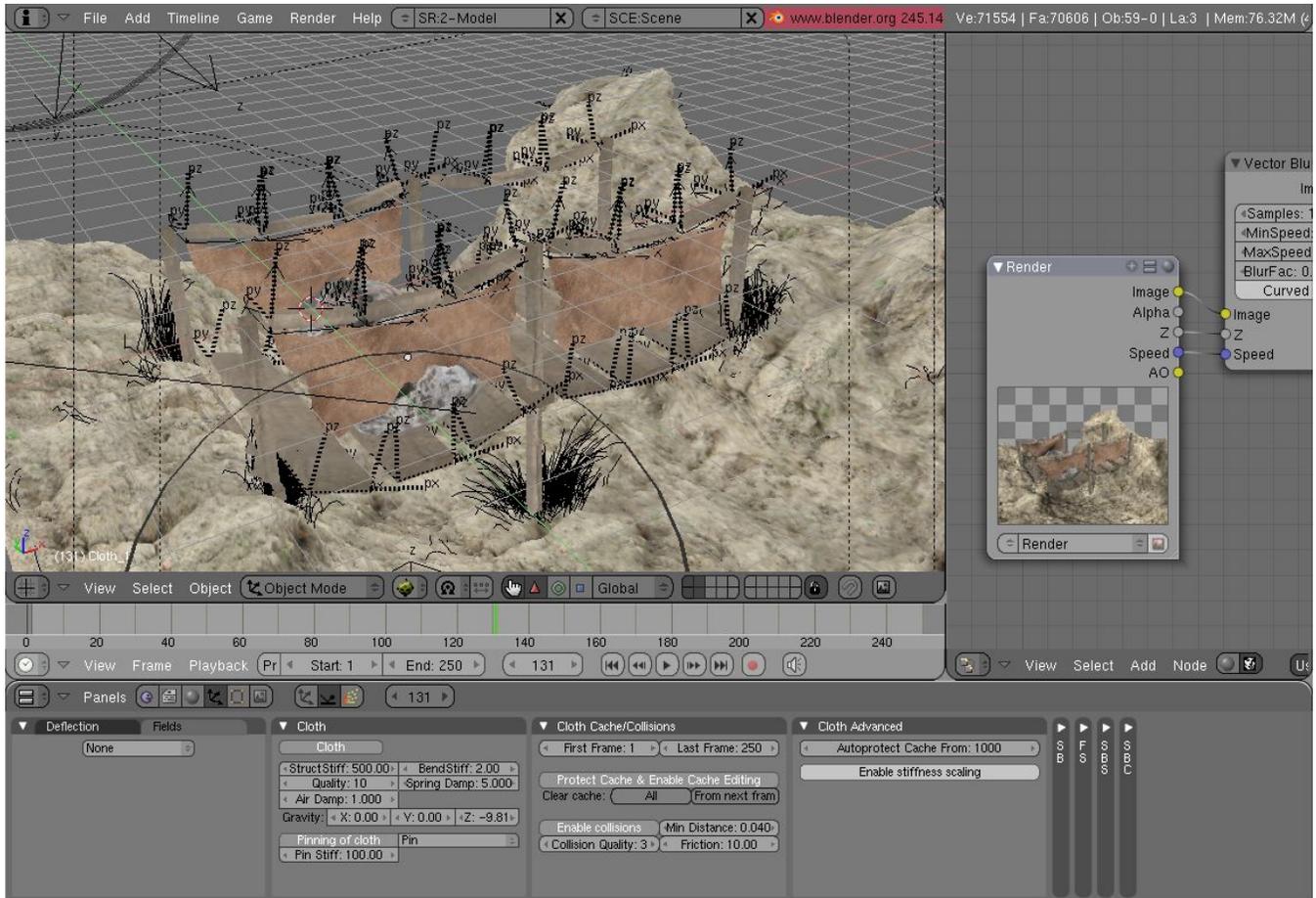


Fig. 2.1432: Cloth in action.

- Model clothing that encloses but does not penetrate the character’s mesh.
- Parent the clothing objects to the armature. The armature will now have several child meshes bound to it.
- Create a new vertex group on each cloth object for its pinned vertices
- Add vertexes to be pinned to this vertex group and give these vertices non-zero weights (you probably want weight = 1). For example the belt area of the tunic would be in the vertex group and have weight one.
- Designate the clothing objects as “cloth” in the Physics tab of the Properties window. Make sure the *Cloth* modifier is below the *Armature* modifier in the modifier stack.
- press the *Pinning of Cloth* button in the *Cloth* panel and select the vertex group.
- Designate the character’s mesh as “collision” object in the Physics tab of the Properties window.
- The clothing is now ready. Non-pinned vertices will be under control of the Cloth modifier. Pinned vertices will be under control of the Armature modifier.

---

**Note:** When animating or posing the character you must begin from the bind pose. Move the character to its initial pose over several frames so the physics engine can simulate the clothing moving. Very fast movements and teleport jumps can break the physics simulation.

---

### Cloth Sewing Springs

Another method of restraining cloth similar to pinning is sewing springs. Sewing springs are virtual springs that pull vertices in one part of a cloth mesh toward vertices in another part of the cloth mesh. This is different from pinning which binds vertices of the cloth mesh in place or to another object. A clasp on a cloak could be created with a sewing spring. The spring could pull two corners of a cloak about a character’s neck. This could result in a more realistic simulation than pinning the cloak to the character’s neck since the cloak would be free to slide about the character’s neck and shoulders.

Sewing springs are created by adding extra edges to a cloth mesh. These extra edges do not need to be included in faces. They should connect vertices in the mesh that should be pulled together. For example the corners of a cloak. The vertexes of these extra edges are added to a vertex group.

Enable the *Cloth Sewing Springs* panel and select the vertex group. Give the springs a non-zero force value and your cloth is ready to simulate.

### Collisions

In most cases, a piece of cloth does not just hang there in 3D space, it collides with other objects in the environment. To ensure proper simulation, there are several items that have to be set up and working together:

- The *Cloth* object must be told to participate in *Collision s*.
- Optionally (but recommended) tell the cloth to collide with itself.
- Other objects must be visible to the *Cloth* object *via* shared layers.
- The other objects must be mesh objects.
- The other objects may move or be themselves deformed by other objects (like an armature or shape key).
- The other mesh objects must be told to deflect the cloth object.
- The blend file must be saved in a directory so that simulation results can be saved.
- You then *Bake* the simulation. The simulator computes the shape of the cloth for a frame range.
- You can then edit the simulation results, or make adjustments to the cloth mesh, at specific frames.

- You can make adjustments to the environment or deforming objects, and then re-run the cloth simulation from the current frame forward.

## Collision Settings

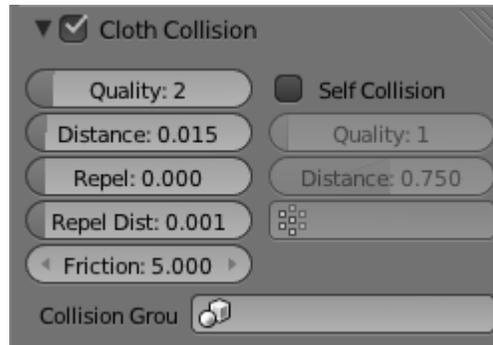


Fig. 2.1433: Cloth Collisions panel.

Now you must tell the *Cloth* object that you want it to participate in collisions. For the cloth object, locate the *Cloth Collision* panel, shown to the right:

**Enable Collisions** LMB click this to tell the cloth object that it needs to move out of the way.

**Quality** A general setting for how fine and good a simulation you wish. Higher numbers take more time but ensure less tears and penetrations through the cloth.

**Distance** As another object gets this close to it (in Blender Units), the simulation will start to push the cloth out of the way.

**Repel** Repulsion force to apply when cloth is close to colliding.

**Repel Distance** Maximum distance to apply repulsion force. Must be greater than minimum distance.

**Friction** A coefficient for how slippery the cloth is when it collides with the mesh object. For example, silk has a lower coefficient of friction than cotton.

**Self-collisions** Real cloth cannot permeate itself, so you normally want the cloth to self-collide.

**Enable Self Collisions** Click this to tell the cloth object that it should not penetrate itself. This adds to simulation compute time, but provides more realistic results. A flag, viewed from a distance does not need this enabled, but a close-up of a cape or blouse on a character should have this enabled.

**Quality** For higher self-collision quality just increase the *Quality* and more self collision layers can be solved. Just keep in mind that you need to have at least the same *Collision Quality* value as the *Quality* value.

**Distance** If you encounter problems, you could also change the *Min Distance* value for the self-collisions. The best value is 0.75; for fast things you better take 1.0. The value 0.5 is quite risky (most likely many penetrations) but also gives some speedup.

Regression blend file: [Cloth selfcollisions](#).

## Shared Layers

Suppose you have two objects: a pair of Pants on layers 2 and 3, and your Character mesh on layers 1 and 2. You have enabled the Pants as cloth as described above. You must now make the Character “visible” to the Cloth object, so that as your character

bends its leg, it will push the cloth. This principle is the same for all simulations; simulations only interact with objects on a shared layer. In this example, both objects share layer 2.

To view/change an object's layers, RMB click to select the object in *Object* mode in the 3D view. M to bring up the "Move Layers" pop-up, which shows you all the layers that the object is on. To put the object on a single layer, LMB click the layer button. To put the object on multiple layers, Shift-LMB the layer buttons. To remove an object from a selected layer, simply Shift-LMB the layer button again to toggle it.

### Mesh Objects Collide

If your colliding object is not a mesh object, such as a NURBS surface, or text object, you must convert it to a mesh object. To do so, select the object in object mode, and in the 3D View header, select *Object* → *Convert Object Type* (Alt-C), and select *Mesh* from the pop-up menu.

### Cloth - Object collisions

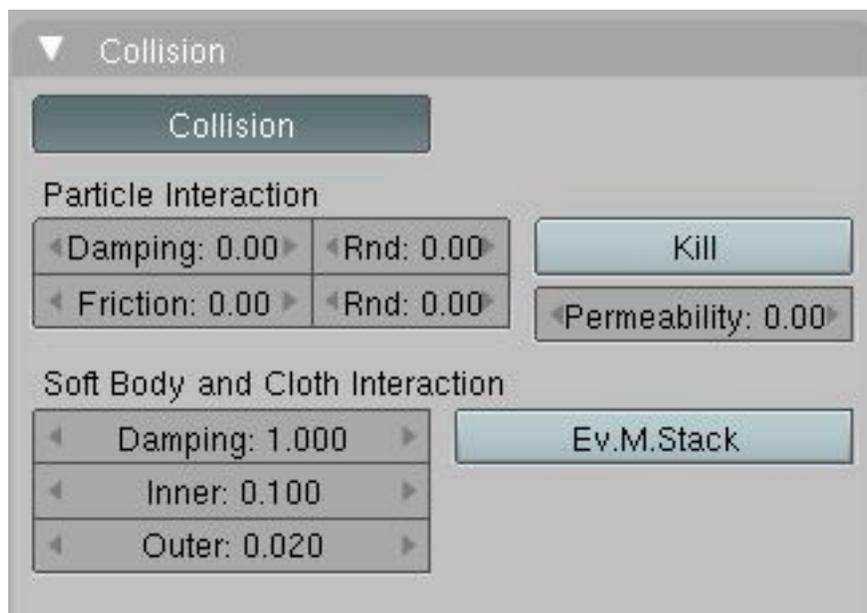


Fig. 2.1434: Collision settings.

The cloth object needs to be deflected by some other object. To deflect a cloth, the object must be enabled as an object that collides with the cloth object. To enable Cloth - Object collisions, you have to enable deflections on the collision object (not on the cloth object).

In the *Buttons* window, *Object* context, *Physics* sub-context, locate the *Collision* panel shown to the right. It is also important to note that this collision panel is used to tell all simulations that this object is to participate in colliding/deflecting other objects on a shared layer (particles, soft bodies, and cloth).

**Warning:** There are three different *Collision* panels, all found in the *Physics* sub-context. The first (by default), a tab beside the *Fields* panel, is the one needed here. The second panel, a tab in the *Soft Body* group, concern softbodies (and so has nothing to do with cloth). And we have already seen the last one, by default a tab beside the *Cloth* panel.

## Mesh Object Modifier Stack

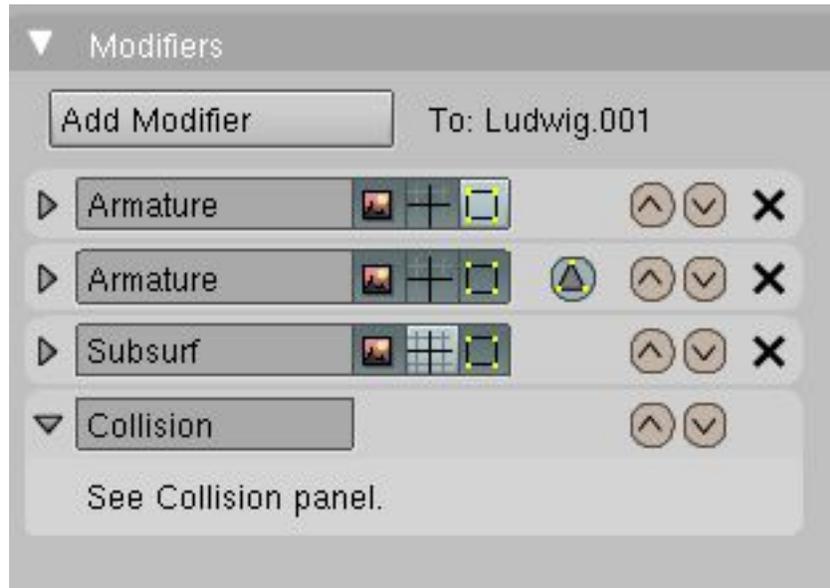


Fig. 2.1435: Collision stack.

The object’s shape deforms the cloth, so the cloth simulation must know the “true” shape of that mesh object at that frame. This true shape is the basis shape as modified by shape keys or armatures. Therefore, the *Collision* modifier must be **after** any of those. The image to the right shows the *Modifiers* panel for the Character mesh object (not the cloth object).

## Cloth Cache

Cache settings for cloth are the same as with other dynamic systems. See [Particle Cache](#) for details.

## Bake Collision

After you have set up the deflection mesh for the frame range you intend to run the simulation (including animating that mesh *via* armatures), you can now tell the cloth simulation to compute (and avoid) collisions. Select the cloth object and in the *Object* context, *Physics* sub-context, set the *Start* and *End* settings for the simulation frames you wish to compute, and click the *Bake* button.

You cannot change *Start* or *End* without clearing the bake simulation. When the simulation has finished, you will notice you have the option to free the bake, edit the bake and re-bake:

There’s a few things you’ll probably notice right away. First, it will bake significantly slower than before, and it will probably clip through the box pretty badly as in the picture on the right.

## Editing the cached simulation

The cache contains the shape of the mesh at each frame. You can edit the cached simulation, after you’ve baked the simulation and pressed the *Bake Editing* button. Just go to the frame you want to fix and **Tab** into *Edit mode*. There you can move your vertices using all of Blender’s mesh shaping tools. When you exit, the shape of the mesh will be recorded for that frame of the animation. If you want Blender to resume the simulation using the new shape going forward, **LMB** click *Rebake from next Frame* and play the animation. Blender will then pick up with that shape and resume the simulation.

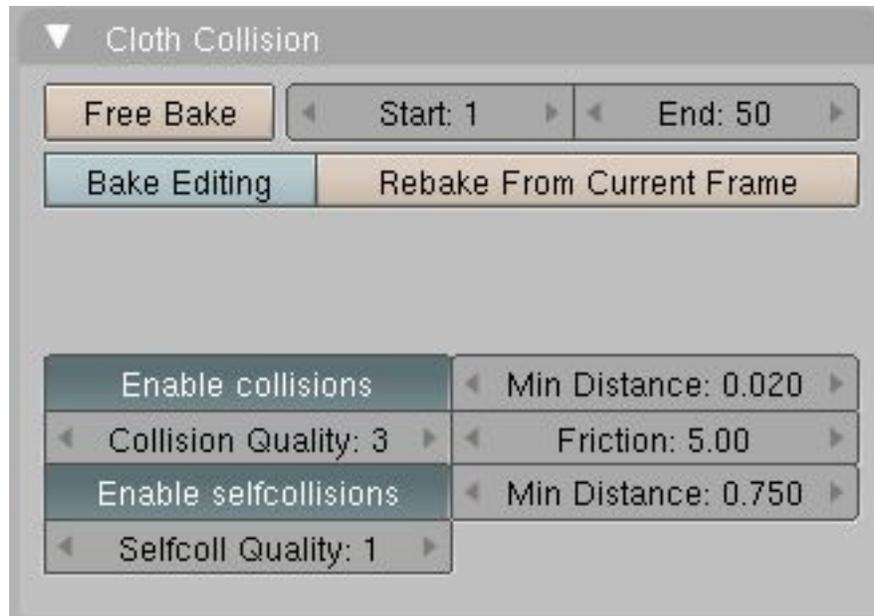


Fig. 2.1436: After Baking.

Edit the mesh to correct minor tears and places where the colliding object has punctured the cloth.

If you add, delete, extrude, or remove vertices in the mesh, Blender will take the new mesh as the starting shape of the mesh back to the *first frame* of the animation, replacing the original shape you started with, up to the frame you were on when you edited the mesh. Therefore, if you change the content of a mesh, when you Tab out of *Edit mode*, you should unprotect and clear the cache so that Blender will make a consistent simulation.

## Troubleshooting

If you encounter some problems with collision detection, there are two ways to fix them:

- The fastest solution is to increase the *Min Distance* setting under the *Cloth Collision* panel. This will be the fastest way to fix the clipping; however, it will be less accurate and won't look as good. Using this method tends to make it look like the cloth is resting on air, and gives it a very rounded look.
- A second method is to increase the *Quality* (in the first *Cloth* panel). This results in smaller steps for the simulator and therefore to a higher probability that fast-moving collisions get caught. You can also increase the *Collision Quality* to perform more iterations to get collisions solved.
- If none of the methods help, you can easily edit the cached/baked result in *Edit mode* afterwards.
- My Cloth is torn by the deforming mesh - he "Hulks Out": Increase its structural stiffness (*StructStiff* setting, *Cloth* panel), very high, like 1000.

---

**Note:** *Subsurf* modifier

A bake/cache is done for every subsurf level so please use **the equal** subsurf level for render and preview.

---

## Examples

To start with cloth, the first thing you need, of course, is some fabric. So, let's delete the default cube and add a plane. I scaled mine up along the Y axis, but you don't have to do this. In order to get some good floppy and flexible fabric, you'll need to

subdivide it several times. I did it 8 times for this example. So `Tab` into *Edit mode*, and press `W` → *Subdivide multi*, and set it to 8.

Now, we'll make this cloth by going to the *Object* context → *Physics* sub-context. Scroll down until you see the *Cloth* panel, and press the *Cloth* button. Now, a lot of settings will appear, most of which we'll ignore for now.

That's all you need to do to set your cloth up for animating, but if you press `Alt-A`, your lovely fabric will just drop very un-spectacularly. That's what we'll cover in the next two sections about pinning and colliding.

### Using Simulation to Shape/Sculpt a Mesh

You can *Apply* the *Cloth* modifier at any point to freeze the mesh in position at that frame. You can then re-enable the cloth, setting the start and end frames from which to run the simulation forward.

Another example of aging is a flag. Define the flag as a simple grid shape and pin the edge against the flagpole. Simulate for 50 frames or so, and the flag will drop to its "rest" position. Apply the *Cloth* modifier. If you want the flag to flap or otherwise move in the scene, re-enable it for the frame range when it is in camera view.

### Smoothing of Cloth

Now, if you followed this from the previous section, your cloth is probably looking a little blocky. In order to make it look nice and smooth like the picture you need to apply a *Smooth* and/or *Subsurf* modifier in the *Modifiers* panel under the *Editing* context. Then, in the same context, find the *Links and Materials* panel (the same one you used for vertex groups) and press *Set Smooth*.

Now, if you press `Alt-A`, things are starting to look pretty nice, don't you think?

### Cloth on armature

Cloth deformed by armature and also respecting an additional collision object: [Regression blend file](#).

### Cloth with animated vertex groups

Cloth with animated pinned vertices: [Regression blend file](#). UNSUPPORTED: Starting with a goal of 0 and increasing it, but still having the vertex not pinned will not work (e.g. from goal = 0 to goal = 0.5).

### Cloth with Dynamic Paint

Cloth with Dynamic Paint using animated vertex groups: [Regression blend file](#). UNSUPPORTED: Starting with a goal of 0 and increasing it, but still having the vertex not pinned will not work (e.g. from goal = 0 to goal = 0.5) because the necessary "goal springs" cannot be generated on the fly.

### Using Cloth for Softbodies

Cloth can also be used to simulate softbodies. It's for sure not its main purpose but it works nonetheless. The example image uses standard *Rubber* material, no fancy settings, just `Alt-A`.

Blend file for the example image: [Using Cloth for softbodies](#).



Fig. 2.1437: Using cloth for softbodies.



Fig. 2.1438: Flag with wind applied.

## Cloth with Wind

Regression blend file for Cloth with wind and self collisions (also the blend for the image above): [Cloth flag with wind and selfcollisions](#).

## 2.7.3 Collisions

Particles, Soft Bodies and Cloth objects may collide with mesh objects. Boids try to avoid Collision objects.

- The objects need to share at least one common layer to have effect.
- You may limit the effect on particles to a group of objects (in the [Field Weights](#) panel).
- *Deflection* for softbody objects is difficult, they often penetrate the colliding objects.
- Hair particles ignore deflecting objects (but you can animate them as softbodies which take deflection into account).

If you change the deflection settings for an object you have to recalculate the particle, softbody or cloth system (*Free Cache*), this is not done automatically. You can clear the cache for all selected objects with `Ctrl-B` → *Free cache selected*.

### Reference

Mode: *Object* Mode

Panel: *Object* context → *Physics* sub-context → *Collision*

### Options

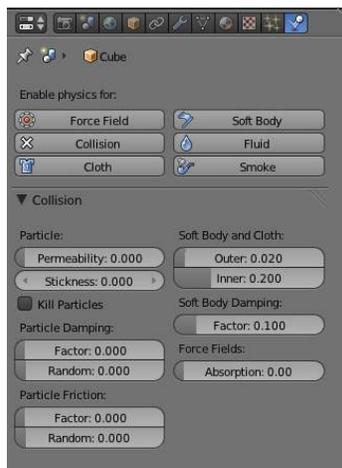


Fig. 2.1439: Image 1: Collision panel in the Physics sub-context.

**Permeability** Fraction of particles passing through the mesh. Can be animated with *Object* Ipos, *Perm* channel.

**Stickiness** How much particles stick to the object.

**Kill Particles** Deletes Particles upon impact.

**Damping Factor** Damping during a collision (independent of the velocity of the particles).

**Random damping** Random variation of damping.

**Friction Factor** Friction during movements along the surface.

**Random friction** Random variation of friction.

Fig. 2.1440: Image 1b: A softbody vertex colliding with a plane.

### Soft Body and Cloth Interaction

**Outer** Size of the outer collision zone.

**Inner** Size of the inner collision zone (padding distance).

Outside and inside is defined by the face normal, depicted as blue arrow in (*Image 1b*).

**Damping Factor** Damping during a collision.

*Softbody* collisions are difficult to get perfect. If one of the objects move too fast, the soft body will penetrate the mesh. See also the section about [Soft Bodies](#).

### Force Field Interaction

**Absorption** A deflector can also deflect effectors. You can specify some collision/deflector objects which deflect a specific portion of the effector force using the *Absorption* value. 100% absorption results in no force getting through the collision/deflector object at all. If you have 3 collision object behind each other with e.g. 10%, 43% and 3%, the absorption ends up at around 50% ( $100 \times (1 - 0.1) \times (1 - 0.43) \times (1 - 0.03)$ ).

### Examples



Fig. 2.1441: Image 2: Deflected Particles.

Here is a *Meta* object, dupliverted to a particle system emitting downwards, and deflected by a mesh cube:

### Hints

- Make sure that the normals of the mesh surface are facing towards the particles/points for correct deflection.
- Hair particles react directly to force fields, so if you use a force field with a short range you don't need necessarily collision.
- Hair particles avoid their emitting mesh if you edit them in *Particle* mode. So you can at least model the hair with collision.

## 2.7.4 Dynamic Paint

### Introduction

Dynamic paint is a new modifier and physics system that can turn objects into paint canvases and brushes, creating vertex colors, image sequences or displacement. This makes many effects possible that were previously difficult to achieve, for example footsteps in the snow, raindrops that make the ground wet, paint that sticks to walls, or objects that gradually freeze.

This guide explains the very basics of Dynamic Paint user interface and general features.

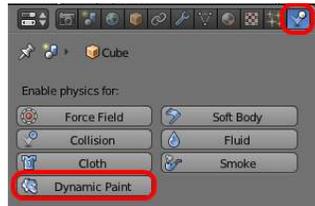


Fig. 2.1442: How to activate the Dynamic Paint

### Activating the modifier

Dynamic Paint can be activated from the “Physics” tab of the “Properties” editor.

### Types

Modifier itself has two different types:

**Canvas** Makes object receive paint from Dynamic Paint brushes.

**Brush** Makes object apply paint on the canvas.

---

**Note:** You can also enable brush and canvas simultaneously. In that case same object’s “brush” doesn’t influence it’s “canvas”, but can still interact with other objects in the scene.

---

### See also

- [A step-by step introduction](#)
- [A detailed guide that covers every setting with images and examples \(Currently not up-to-date\)](#)

## Dynamic Paint Brush

### Main Panel

From the first brush panel you can define how brush affects canvas color surfaces.

**Absolute Alpha** This setting limits brush alpha influence. Without it, brush is “added” on surface over and over again each frame, increasing alpha and therefore influence of brush on canvas. In many cases however, it’s preferred to not increase brush alpha if it already is on brushes level.

**Erase Paint** Makes brush dissolve exiting paint instead of adding it.

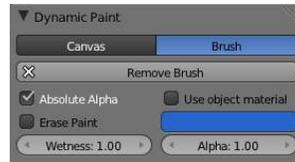


Fig. 2.1443: Brush main panel

**Wetness** Defines how “wet” new paint is. Wetness is visible on “Paint” surface “wetmap”. Speed of “Drip” and “Spread” effects also depends on how wet the paint is.

**Use object material** When enabled, you can define a material to be used as brush color. This includes material’s base color and all textures linked to it, eventually matching the rendered diffuse color. This setting is only available when using “Blender Internal” renderer at the moment.

Otherwise you can define a color for the brush from the color box below.

**Alpha** Defines brush alpha or visibility. Final wetness is also affected by alpha.

### Source Panel

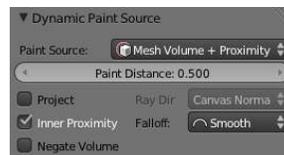


Fig. 2.1444: Brush source panel

Brush “Source” setting lets you define how brush influence/intersection is defined.

There are currently five brush behavior types to choose from, each having individual settings for further tweaking:

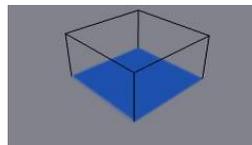


Fig. 2.1445: Brush Source - Volume

**Mesh Volume** This the default option. Brush affects all surface point inside the mesh volume.



**Proximity** Only uses defined distance to the closest point on brush mesh surface. Note that inside of the volume is not necessarily affected because it’s not close to the surface.

Proximity falloff type can be “Smooth”, “Sharp” or tweaked with a color ramp.

**Project** Projects brush to the canvas from a defined direction. Basically this can be considered as “direction aligned” proximity.



**Mesh Volume + Proximity** Same as volume type, but also has influence over defined distance. Same falloff types as for “Proximity” type are available.

**Inner Proximity** Applies proximity inside the mesh volume.

**Negate Volume** Negates brush alpha within mesh volume.

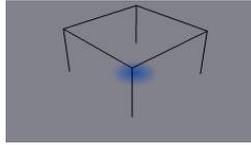


Fig. 2.1458: Brush Source - Object Center

**Object Center** Instead of calculating proximity to the brush object mesh, which can be quite slow in some cases, only distance to only center is calculated. This is much faster and often good enough.

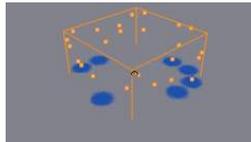


Fig. 2.1459: Brush Source - Particle System

**Particle System** Brush influence is defined by particles from a selected particle system.

### Velocity Panel

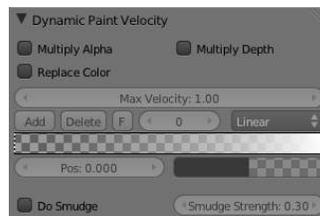


Fig. 2.1460: Velocity panel

This panel shows brush options that are based on object velocity.

On top you have a color ramp and several related settings. Basically the color ramp represents brush velocity values: left side being zero velocity and right side being the “Max velocity”. Speed is measured in “Blender units per frame”.

Tick boxes above can be used to define color ramp influence.

**Multiply Alpha** Uses color ramp’s alpha value depending on current velocity and multiplies brush alpha with it.

**Replace Color** Replaces the brush color with the ramp color.

**Multiply Depth** Multiplies brushes “depth intersection” effect. Basically you can adjust displace and wave strength depending on brush speed.

**Smudge settings** Enabling Smudge makes the brush “smudge” (or “smear”) existing colors on the surface as it moves. The strength of this effect can be defined from the “Smudge Strength” property.

Even when smudge is enabled brush still does it’s normal paint effect. If you want a purely smudging brush use zero alpha. It’s also possible to have “Erase” option enabled together with smudge.

## Waves Panel



Fig. 2.1461: Brush Waves panel

This panel is used to adjust brush influence to “Wave” surfaces.

You can use “Wave Type” menu to select what effect this brush has on the wave simulation. Below are two settings for further adjustments.

**Factor** Adjusts how strongly brush “depth” affects the simulation. You can also use negative values to make brush pull water up instead of down.

**Clamp Waves** In some cases the brush goes very deep inside the surface messing whole simulation up. You can use this setting to “limit” influence to only certain depth.

There are four “Wave Type” options available:

**Depth Change** This option makes brush create waves when the intersection depth with the surface is *changed* on that point. If the brush remains still it won’t have influence.

Using a negative “Factor” with this type can create a nice looking “wake” for moving objects like ships.

**Obstacle** Constantly affects surface whenever intersecting. Waves are also reflected off this brush type. However, due the nature of wave simulation algorithm this type creates an unnatural “dent” in the surface if brush remains still.

**Force** Directly affects the velocity of wave motion. Therefore the effect isn’t one to one with brush intersection depth, yet the force strength depends on it.

**Reflect Only** This type has no visible effect on the surface alone but reflects waves that are already on the surface.

## Dynamic Paint Canvas

### Main Panel

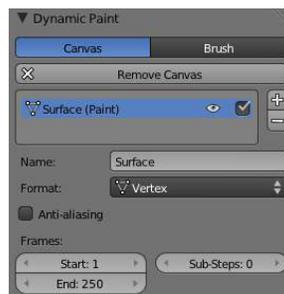


Fig. 2.1462: Canvas main panel

The first panel of canvas contains the list of Dynamic Paint surfaces. These surfaces are basically layers of paint, that work independently from each other. You can define individual settings for them and bake them separately.

If surface type/format allows previewing results in 3D-viewport, an eye icon is visible to toggle preview.

The checkbox toggles whether surface is active at all. If not selected, no calculations or previews are done.

You can also give each surface an unique name to easily identify them.

Below you can set surface type and adjust quality and timing settings.

Each surface has a certain format and type. Format determines how data is stored and outputted. Currently there are two formats available:

- Image Sequences. Dynamic Paint generates UV wrapped image files of defined resolution as output.
- Vertex. Dynamic Paint operates directly on mesh vertex data. Results are stored by point cache and can be displayed in viewports. However, using vertex level also requires a highly subdivided mesh to work.

From quality settings you can adjust image resolution (for image sequences) and anti-aliasing.

Then you can define surface processing start and end frame, and number of used sub-steps. Sub-steps are extra samples between frames, usually required when there is a very fast brush.

### Advanced Panel

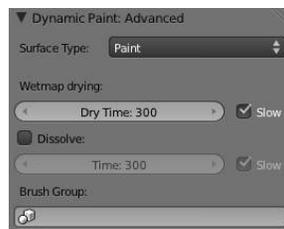


Fig. 2.1463: Canvas advanced panel

From “Advanced” panel you can adjust surface type and related settings.

Each surface has a “type” that defines what surface is used for. Available types are:

- Paint
- Displace
- Waves
- Weight

**Common options** For each surface type there are special settings to adjust. Most types have the settings *Dissolve* and *Brush* :

**Dissolve** used to make the surface smoothly return to its original state during a defined time period

**Brush Group** used to define a specific object group to pick brush objects from

**Paint** “Paint” is the basic surface type that outputs color and wetness values. In case of vertex surfaces results are outputted as vertex colors.

Wetmap is a black-and-white output that visualizes paint wetness. White being maximum wetness, black being completely dry. It is usually used as mask for rendering. Some “paint effects” affect wet paint only.

**Displace** This type of surface outputs intersection depth from brush objects.

---

**Tip:** If the displace output seems too rough it usually helps to add a “Smooth” modifier after Dynamic Paint in the modifier stack.

---

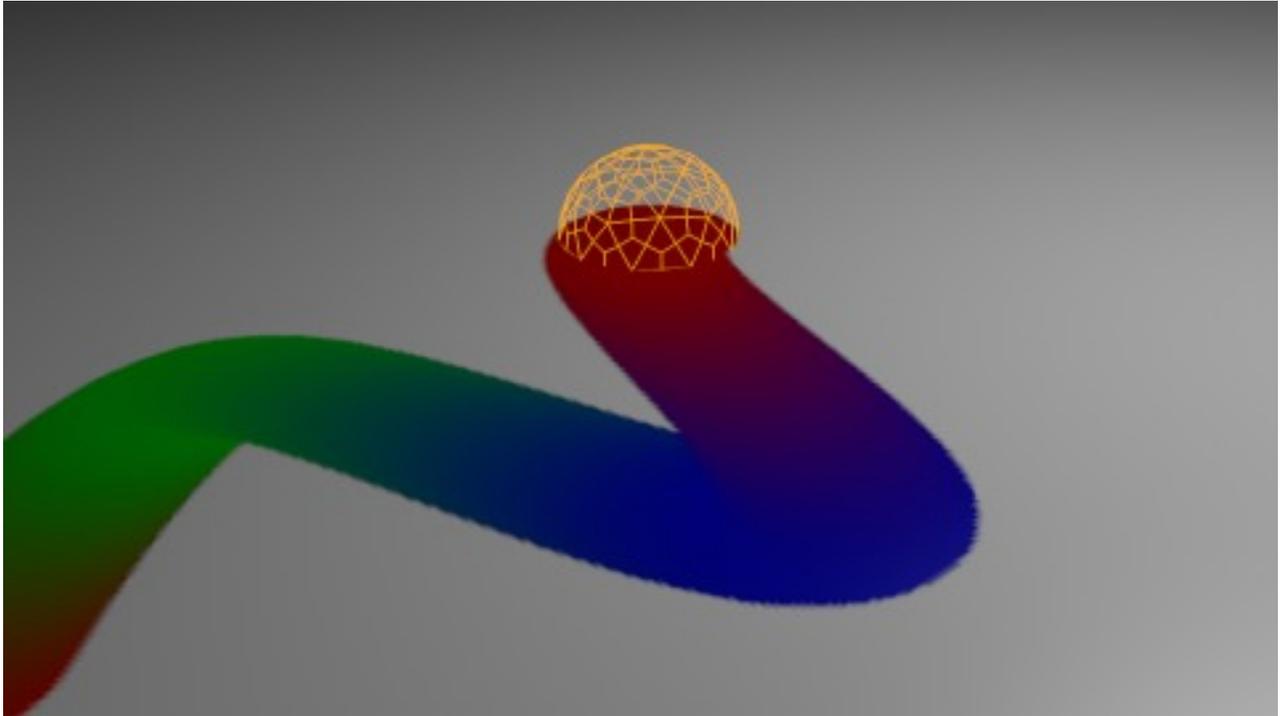


Fig. 2.1464: Paint Surface

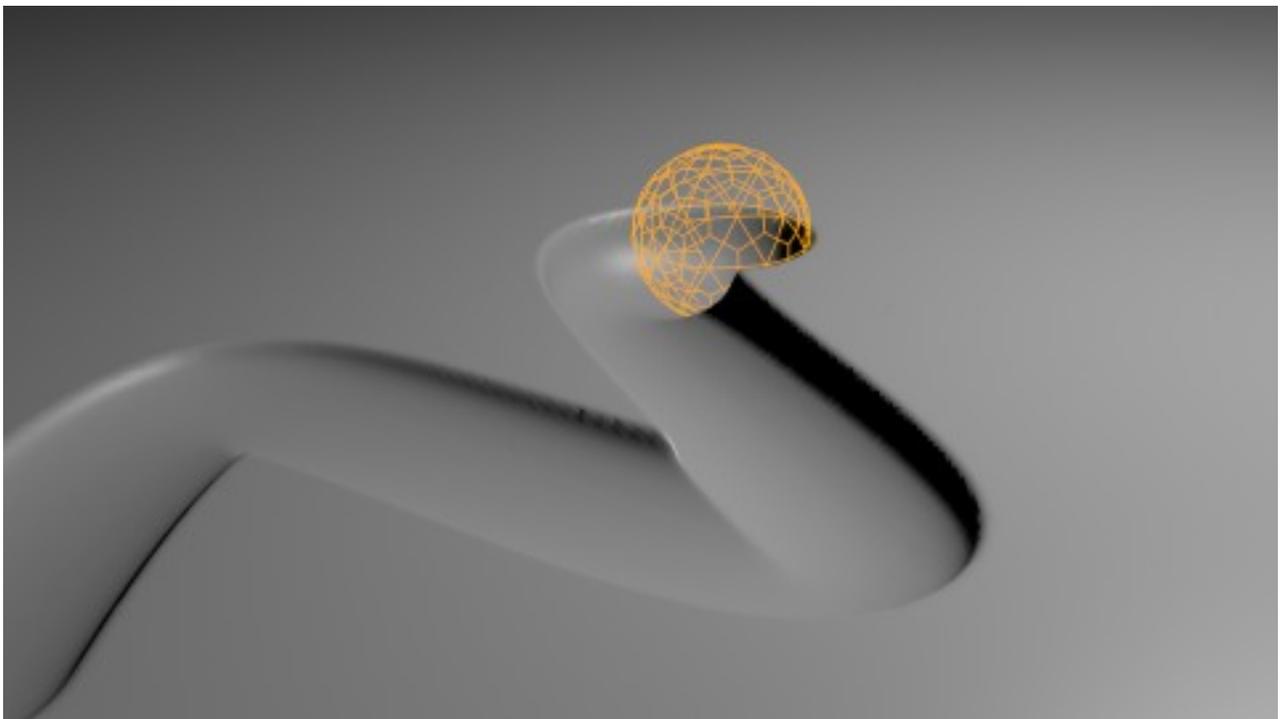


Fig. 2.1465: Displace Surface

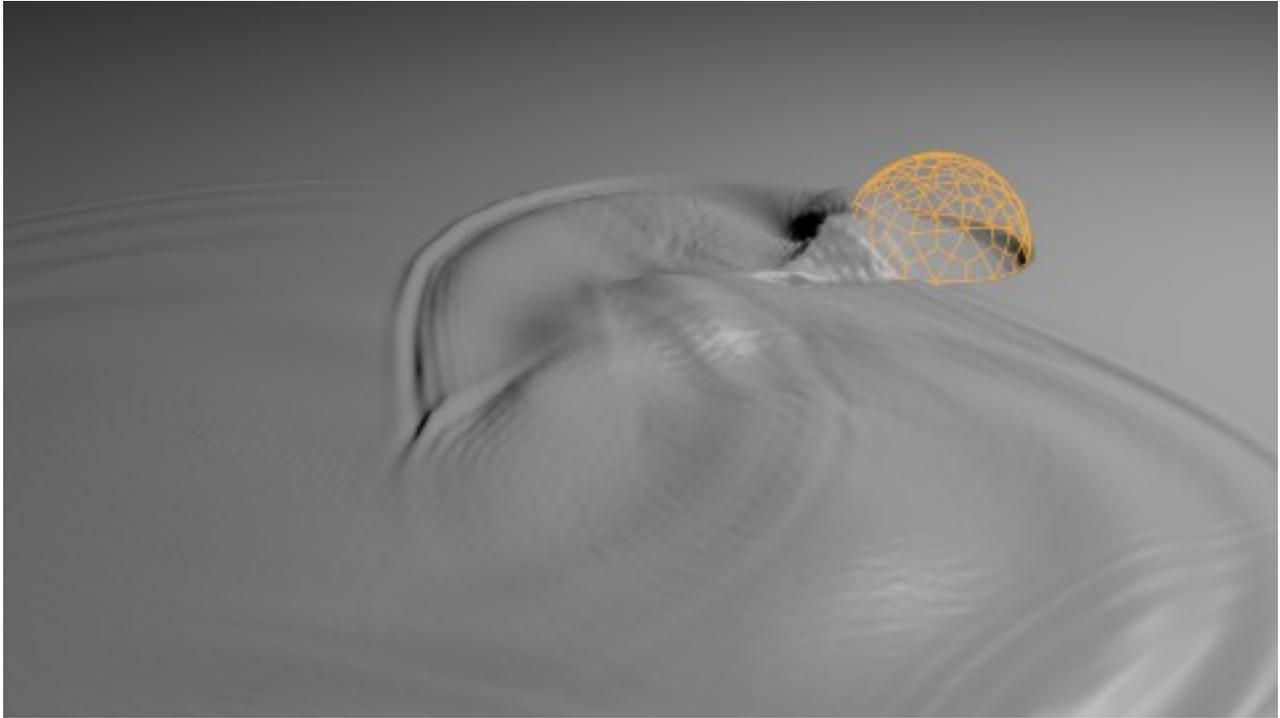


Fig. 2.1466: Waves Surface

**Waves** This surface type produces simulated wave motion. Like *displace*, wave surface also uses brush intersection depth to define brush strength.

You can use following settings to adjust the motion:

**Open Borders** Allows waves to pass through mesh “edges” instead of reflecting from them.

**Timescale** Directly adjusts simulation speed without affecting simulation outcome. Lower values make simulation go slower and otherwise.

**Speed** Affects how fast waves travel on the surface. This setting is also corresponds to the size of the simulation. Half the speed equals surface double as large.

**Damping** Reduces the wave strength over time. Basically adjusts how fast wave disappears.

**Spring** Adjusts the force that pulls water back to “zero level”.

---

**Tip:** In some cases the wave motion gets very unstable around brush. It usually helps to reduce wave speed, brush “wave factor” or even the resolution of mesh/surface.

---

**Weight** This is a special surface type only available for vertex format. It outputs vertex weight groups that can be used by other Blender modifiers and tools.

---

**Tip:** It’s usually preferred to use “proximity” based brushes for weight surfaces to allow smooth falloff between weight values.

---

### Output Panel

From “Output” panel you can adjust how surface outputs its results.

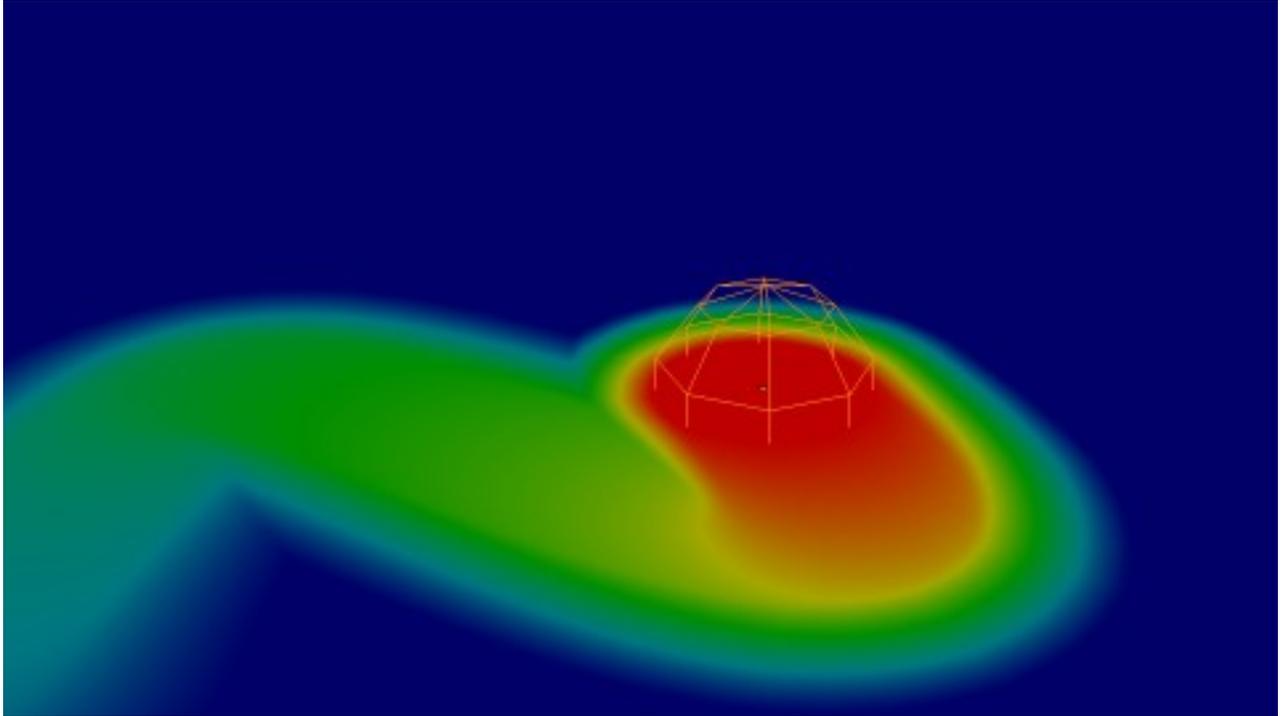


Fig. 2.1467: Weight Surface



Fig. 2.1468: Canvas output panel

For “Vertex” format surfaces, you can select a mesh data layer (color / weight depending on surface type) to generate results to. You can use the “+”/“-” icons to add/remove a data layers of given name. If layer with given name isn’t found, it’s shown as red.

For “Image Sequence” surfaces, you can define used “UV Layer” and output file saving directory, filenames and image format.

### Effects Panel



Fig. 2.1469: Canvas effects panel

This is a special feature for “Paint” type surface. It generates animated movement on canvas surface.

Currently there are 3 effects available:

**Spread** Paint slowly spreads to surrounding points eventually filling all connected areas.

**Drip** Paint moves in specific direction specified by Blender force fields, gravity and velocity with user defined influences.

**Shrink** Painted area slowly shrinks until disappears completely.

For spread and drip effects, only “wet paint” is affected, so as the paint dries, movement becomes slower until it stops.

### Cache Panel

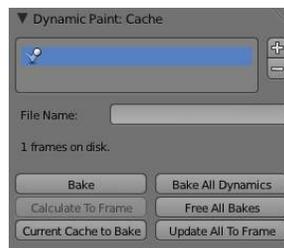


Fig. 2.1470: Canvas cache panel

This panel is currently only visible for “vertex” format surfaces. You can use it to adjust and bake point cache.

## 2.7.5 Fluid Simulation

### Introduction

### Reference

Mode: *Object mode / Edit mode (Mesh)*

Panel: *Physics sub-context → Fluid*

## Description

While modeling a scene with blender, certain objects can be marked to participate in the fluid simulation, e.g. as fluid or as an obstacle. The bounding box of another object will be used to define a box-shaped region to simulate the fluid in (the so called “simulation domain”). The global simulation parameters (such as viscosity and gravity) can be set for this domain object.

Using the *BAKE* button, the geometry and settings are exported to the simulator and the fluid simulation is performed, generating a surface mesh together with a preview for each animation frame, and saving them to hard disk. Then the appropriate fluid surface for the current frame is loaded from disk and displayed or rendered.

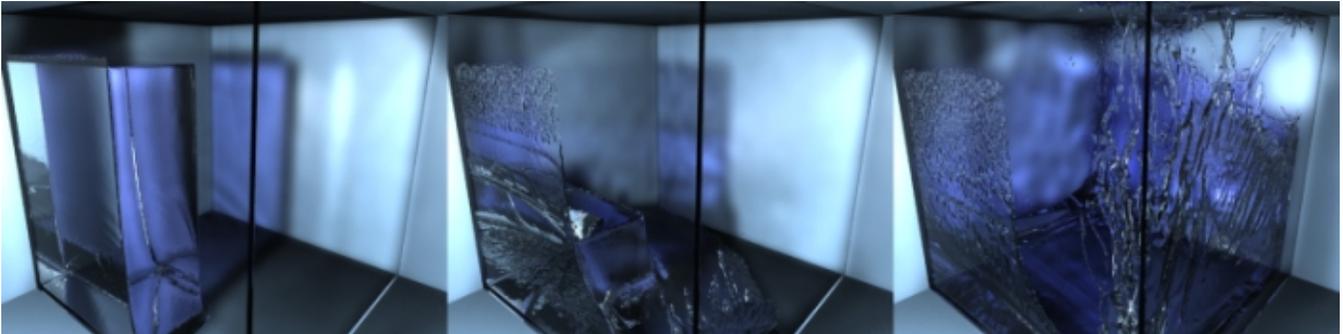


Fig. 2.1471: A breaking dam.

## Workflow

In general, you follow these steps:

- set the **simulation domain** (the portion of the scene where the fluid will flow),
- set the **fluid source(s)**, and specify its material, viscosity, and initial velocity,
- eventually, set other objects to **control the volume** of the fluid (inlets and outlets),
- eventually, set other objects related to the fluid, like: - **obstacles**, - **particles** floating on the fluid, - **fluid control**, to shape part of the fluid in the desired form,
- eventually, **animate the fluid properties**,
- **Bake the simulation** (eventually, revise as necessary and bake repeatedly).

---

**Tip:** Baking is done on the Domain object!

When you calculate the fluid simulation, **you bake the simulation on the domain object.**

For this reason:

- all the baking options are visible only when selecting the Domain Object,
  - baking options are explained in the `the baking section` of the Domain manual page.
- 

## More about the simulation

To know more about simulating fluids in Blender you can read:

- some **useful hint** about the simulation,

- some [technical details](#), to learn how to do a more realistic fluid simulation,
- the [fluids appendix](#) to learn limitations and workarounds, and some additional links.

## Fluid Domain

### The Domain Object

The bounding box of the object serves as the boundary of the simulation. **All fluid objects must be in the domain.** Fluid objects outside the domain will not bake. No tiny droplets can move outside this domain; it's as if the fluid is contained within the 3D space by invisible force fields. There can be only a single fluid simulation domain object in the scene.

**The shape of the object does not matter because it will *always* be treated like a box** (The lengths of the bounding box sides can be different). So, usually there won't be any reason to use another shape than a box. If you need obstacles or other boundaries than a box to interfere with the fluid flow, you need to insert additional obstacle objects *inside* the domain boundary.

This object will be *replaced* by the fluid during the simulation.

---

**Tip:** Baking is done on the Domain object

When you calculate the fluid simulation, **you bake the simulation on the domain object.** For this reason all the baking options are visible only when selecting the Domain Object.

For baking options, see: [Baking](#).

---

## Options

**Bake button** For baking options, see: [Baking](#).

### Resolution

**Render resolution** The granularity at which the actual fluid simulation is performed. This is probably the most important setting for the simulation as it determines the amount of details in the fluid, the memory and disk usage as well as computational time.



Note that the amount of required memory quickly increases: a resolution of 32 requires ca. 4MB, 64 requires ca. 30MB, while 128 already needs more than 230MB. Make sure to set the resolution low enough, depending on how much memory you have, to prevent Blender from crashing or freezing. Remember also that many operating systems limit the amount of memory that can be allocated by a single *process*, such as Blender, even if the *machine* contains much more than this. Find out what limitations apply to your machine.

---

**Note:** Resolution and Real-size of the Domain

Be sure to set the resolution appropriate to the real-world size of the domain (see the *Realworld-size* in the [Domain World](#)). If the domain is not cubic, the resolution will be taken for the longest side. The resolutions along the other sides will be reduced according to their lengths (therefore, a non-cubic domain will need less memory than a cubic one, resolutions being the same).

---

### Preview resolution

This is the resolution at which the preview surface meshes will be generated. So it does not influence the actual simulation. Even if “there is nothing to see” in the preview, there might be a thin fluid surface that cannot be resolved in the preview.

**Display quality** How to display a baked simulation in the 3d view (menu *Viewport Display*) and for rendering (menu *Render Display*):

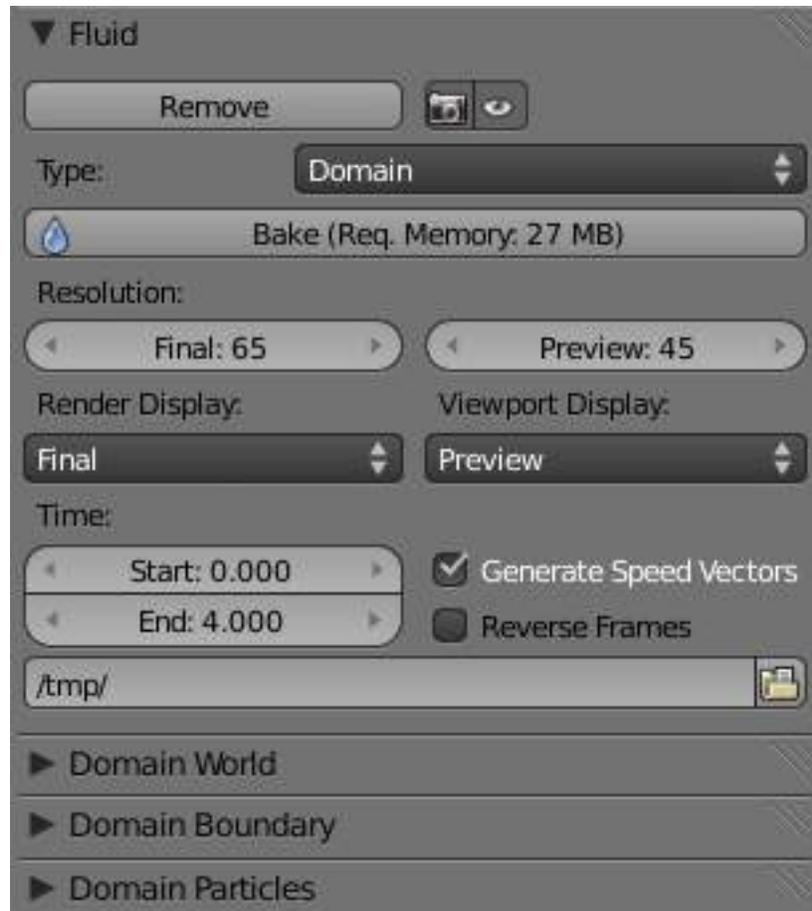


Fig. 2.1472: The fluid simulation options with Domain selected

**Geometry** use the original geometry (before simulation).

**Preview** use the preview mesh.

**Final** use the final high definition mesh.

When no baked data is found, the original mesh will be displayed by default.

After you have baked a domain, it is displayed (usually) in the Blender window as the preview mesh. To see the size and scope of the original domain box, select *Geometry* in the left dropdown.

## Time

**Start** It is the simulation start time (in seconds).

This option makes the simulation computation in Blender start later in the simulation. The domain deformations and fluid flow prior to the start time are not saved.

For example, if you wanted the fluid to appear to already have been flowing for 4 seconds before the actual first frame of data, you would enter 4.0 here.

**End** It is the simulation ending time (in seconds).

---

**Tip:** Start and end times have nothing to do with how many frames are baked

If you set *Start* time to 3.0, and *End* time to 4.0, you will simulate 1 second of fluid motion. That one second of fluid motion will be spread across however-many frames are set in the *Anim* panel (*Scene* context → *Render* sub-context → *Anim* and *Output* panel).

This means, for example, that if you have Blender set to make 250 frames at 25 fps, the fluid will look like it had already been flowing for 3 seconds at the start of the simulation, *but* will play in slow motion (one-tenth normal speed), since the 1 second fluid sim plays out over the course of 10 video seconds. To correct this, change the end time to 13.0 (3.0 + 10.0) to match the 250 frames at 25 fps. Now, the simulation will be real-time, since you set 10 seconds of fluid motion to simulate over 10 seconds of animation. Having these controls in effect gives you a “speed control” over the simulation.

---

**Generate Speed Vector** If this button is clicked, no speed vectors will be exported. So by default, speed vectors are generated and stored on disk. They can be used to compute image based motion blur with the compositing nodes.

**Reverse fluid frames** The simulation is calculated backward

**Bake directory** For baking options see: [Baking](#).

## Domain World

**Viscosity** The “thickness” of the fluid and actually the force needed to move an object of a certain surface area through it at a certain speed. You can either enter a value directly or use one of the presets in the drop down (such as honey, oil, or water).

For manual entry, please note that the normal real-world viscosity (the so-called dynamic viscosity) is measured in Pascal-seconds (Pa.s), or in Poise units (P, equal to 0.1 Pa.s, pronounced *pwaz*, from the Frenchman Jean-Louis Poiseuille, who discovered the laws on “the laminar flow of viscous fluids”), and commonly centiPoise units (cP, equal to 0.001 Pa.s, *sentipwaz*). Blender, on the other hand, uses the kinematic viscosity (which is dynamic viscosity in Pa.s, divided by the density in  $\text{kg}\cdot\text{m}^{-3}$ , unit  $\text{m}^2 \cdot \text{s}^{-1}$ ). The table below gives some examples of fluids together with their dynamic and kinematic viscosities.

Manual entries are specified by a floating point number and an exponent. These floating point and exponent entry fields (scientific notation) simplify entering very small or large numbers. The viscosity of water at room temperature is 1.002 cP, or 0.001002 Pa.s; the density of water is about  $1000 \text{ kg}\cdot\text{m}^{-3}$ , which gives us a kinematic viscosity of  $0.000001002 \text{ m}^2\cdot\text{s}^{-1}$  - so the entry would be 1.002 times 10 to the minus six ( $1.002 \cdot 10^{-6}$  in scientific notation). Hot Glass and melting

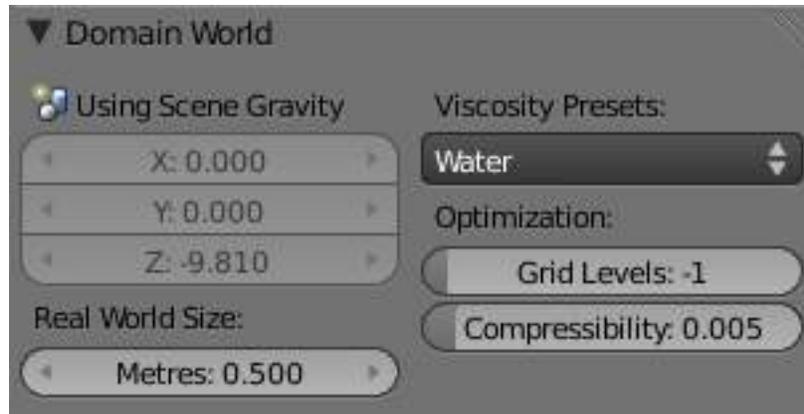


Fig. 2.1477: The Domain World options.

iron is a fluid, but very thick; you should enter something like  $1.0 \times 10^0$  (= 1.0) as its kinematic viscosity (indicating a value of  $1.0 \times 10^6$  cP).

Note that the simulator is not suitable for non-fluids, such as materials that do not “flow”. Simply setting the viscosity to very large values will not result in rigid body behavior, but might cause instabilities.

---

**Note:** Viscosity varies

The default values in Blender are considered typical for those types of fluids and “look right” when animated. However, actual viscosity of some fluids, especially sugar-laden fluids like chocolate syrup and honey, depend highly on temperature and concentration. Oil viscosity varies by SAE rating. Glass at room temperature is basically a solid, but glass at 1500 degrees Celsius flows (nearly) like water.

---

Table 2.51: Blender Viscosity Unit Conversion.

Fluid	dynamic viscosity (in cP)	kinematic viscosity (Blender, in $\text{m}^2 \cdot \text{s}^{-1}$ )
Water (20- C)	$1.002 \times 10^0$ (1.002)	$1.002 \times 10^{-6}$ (0.000001002)
Oil SAE 50	$5.0 \times 10^2$ (500)	$5.0 \times 10^{-5}$ (0.00005)
Honey (20- C)	$1.0 \times 10^4$ (10,000)	$2.0 \times 10^{-3}$ (0.002)
Chocolate Syrup	$3.0 \times 10^4$ (30,000)	$3.0 \times 10^{-3}$ (0.003)
Ketchup	$1.0 \times 10^5$ (100,000)	$1.0 \times 10^{-1}$ (0.1)
Melting Glass	$1.0 \times 10^{15}$	$1.0 \times 10^0$ (1.0)

**Realworld-size** Size of the domain object in the real world in meters. If you want to create a mug of coffee, this might be 10 cm (0.1 meters), while a swimming pool might be 10m. The size set here is for the longest side of the domain bounding box.

Optimization

**Gridlevel** How many adaptive grid levels to be used during simulation - setting this to -1 will perform automatic selection.

**Compressibility** If you have problems with large standing fluid regions at high resolution, it might help to reduce this number (note that this will increase computation times).

**Domain Boundary**

This box has all the slip and surface options.

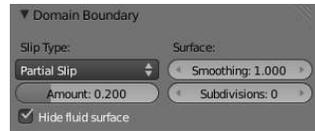


Fig. 2.1478: The Domain Boundary panel

FIXME(Template Unsupported: Doc:2.6/Manual/Physics/Fluid/split\_type;{{Doc:2.6/Manual/Physics/Fluid/split\_type}})

### Surface

**Surface Smoothing** Amount of smoothing to be applied to the fluid surface. 1.0 is standard, 0 is off, while larger values increase the amount of smoothing.

**Subdivisions** Allows the creation of high-res surface meshes directly during the simulation (as opposed to doing it afterwards like a subdivision modifier). A value of 1 means no subdivision, and each increase results in one further subdivision of each fluid voxel. The resulting meshes thus quickly become large, and can require large amounts of disk space. Be careful in combination with large smoothing values - this can lead to long computation times due to the surface mesh generation.

### Hide fluid surface

## Domain Particles

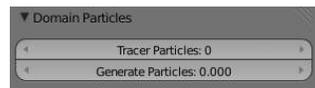


Fig. 2.1479: The Domain Particles panel

Here you can add particles to the fluid simulated, to enhance the visual effect.

**Tracer Particles** Number of tracer particles to be put into the fluid at the beginning of the simulation. To display them create another object with the *Particle* fluid type, explained below, that uses the same bake directory as the domain.

**Generate Particles** Controls the amount of fluid particles to create (0=off, 1=normal, >1=more). To use it, you have to have a surface subdivision value of at least 2.

## Baking

**Bake Button** Perform the actual fluid simulation. Blender will continue to work normally, except there will be a status bar in the top of the window, next to the render pulldown. Pressing `ESC` or the “x” next to the status bar will abort the simulation. Afterwards two `.bobj.gz` (one for the *Final* quality, one for the *Preview* quality), plus one `.bvel.gz` (for the *Final* quality) will be in the selected output directory for each frame.

### Bake directory **REQUIRED!**

Directory and file prefix to store baked surface meshes.

This is similar to the animation output settings, only selecting a file is a bit special: when you select any of the previously generated surface meshes (e.g. `test1_fluidsurface_final_0132.bobj.gz`), the prefix will be automatically set (`test1_` in this example). This way the simulation can be done several times with different settings, and allows quick changes between the different sets of surface data.

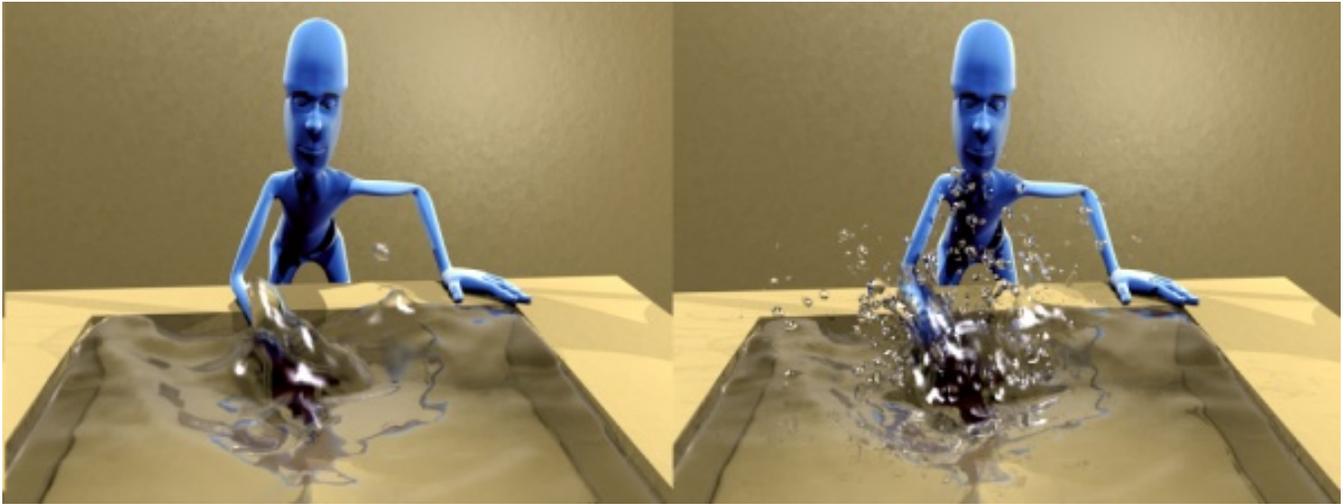


Fig. 2.1480: An example of the effect of particles can be seen here - the image to the left was simulated without, and the right one with particles and subdivision enabled.

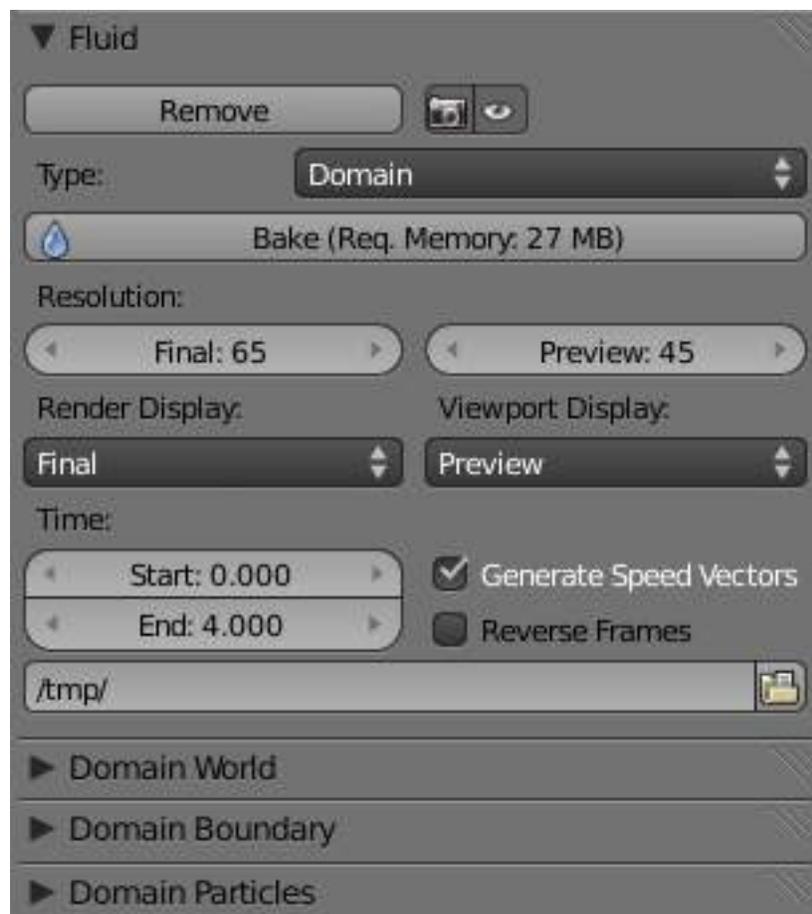


Fig. 2.1481: The fluid simulation options with Domain selected

## Notes

**Unique domain** Because of the possibility of spanning and linking between scenes, there can only be one domain in an entire .blend file.

**Selecting a Baked Domain** After a domain has been baked, it changes to the fluid mesh. To re-select the domain so that you can bake it again after you have made changes, go to any frame and select (RMB) the fluid mesh. Then you can click the *BAKE* button again to recompute the fluid flows inside that domain.

**Baking always starts at Frame #1** The fluid simulator disregards the *Sta* setting in the *Anim* panel, it will always bake from frame 1. If you wish the simulation to start later than frame 1, you must key the fluid objects in your domain to be inactive until the frame you desire to start the simulation. See [Animating the fluid properties](#) for more information.

**Baking always ends at the End Frame set in the Anim panel** If your frame-rate is 25 frames per second, and ending time is 4.0 seconds, then you should (if your start time is 0) set your animation to end at frame  $4.0 \times 25 = 100$

**Freeing the previous baked solutions** Deleting the content of the “Bake” directory is a destructive way to achieve this. Be careful if more than one simulation uses the same bake directory (be sure they use different filenames, or they will overwrite one another)!

**Reusing Bakes** Manually entering (or searching for) a previously saved (baked) computational directory and filename mask will switch the fluid flow and mesh deformation to use that which existed during the old bake. Thus, you can re-use baked flows by simply pointing to them in this field.

**Baking processing time** Baking takes a **lot** of compute power (hence time). Depending on the scene, it might be preferable to bake overnight.

If the mesh has modifiers, the rendering settings are used for exporting the mesh to the fluid solver. Depending on the setting, calculation times and memory use might exponentially increase. For example, when using a moving mesh with *Subsurf* as an obstacle, it might help to decrease simulation time by switching it off, or to a low subdivision level. When the setup/rig is correct, you can always increase settings to yield a more realistic result.

## Fluid Object

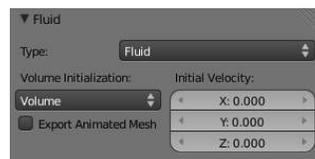


Fig. 2.1482: Fluid object settings

All regions of this object that are inside the domain bounding box will be used as actual fluid in the simulation. If you place more than one fluid object inside the domain, they should currently not intersect. Also make sure the surface normals are pointing outwards. In contrast to domain objects, the actual mesh geometry is used for fluid objects.

```
FIXME(Template Unsupported: Doc:2.6/Manual/Physics/Fluid/volume_init; {{Doc:2.6/Manual/Physics/Fluid/volume_init}}
)
```

```
FIXME(Template Unsupported: Doc:2.6/Manual/Physics/Fluid/animated_mesh_export; {{Doc:2.6/Manual/Physics/Fluid/animated_mesh_e
})
```

**Initial velocity** Speed of the fluid at the beginning of the simulation, in meters per second.

---

**Tip:** The direction of Surface Normals makes a big difference!

Blender uses the orientation of the Surface Normals to determine what is “inside of” the Fluid object and what is “outside”. You want all of the normals to face *outside* (in *Edit mode*, use *Ctrl-N* or press *Spacebar* and choose *Edit?? Normals??*)

*Calculate Outside*). If the normals face the wrong way, you'll be rewarded with a "gigantic flood of water" because Blender will think that the volume of the object is outside of its mesh! This applies regardless of the *Volume init* type setting.

## Fluid Obstacle

This object will be used as an obstacle in the simulation. As with a fluid object, obstacle objects currently should not intersect. As for fluid objects, the actual mesh geometry is used for obstacles. For objects with a volume, make sure that the normals of the obstacle are calculated correctly, and radiating properly (use the *Flip Normal* button, in *Edit mode*, *Mesh Tools* panel, *Editing* context), particularly when using a spinned container. Applying the Modifier *SubSurf* before baking the simulation could also be a good idea if the mesh is not animated.

FIXME(Template Unsupported: Doc:2.6/Manual/Physics/Fluid/volume\_init; {{Doc:2.6/Manual/Physics/Fluid/volume\_init}})

**Boundary type** Determines the stickiness of the obstacle surface, called "Surface Adhesion". Surface Adhesion depends in real-world on the fluid and the graininess or friction/adhesion/absorption qualities of the surface.

**Noslip** causes the fluid to stick to the obstacle (zero velocity).

**Free** (-slip) allows movement along the obstacle (only zero normal velocity).

**Part** (-slip) mixes both types, with 0 being mostly noslip, and 1 being identical to freeslip.

Note that if the mesh is moving, it will be treated as noslip automatically.

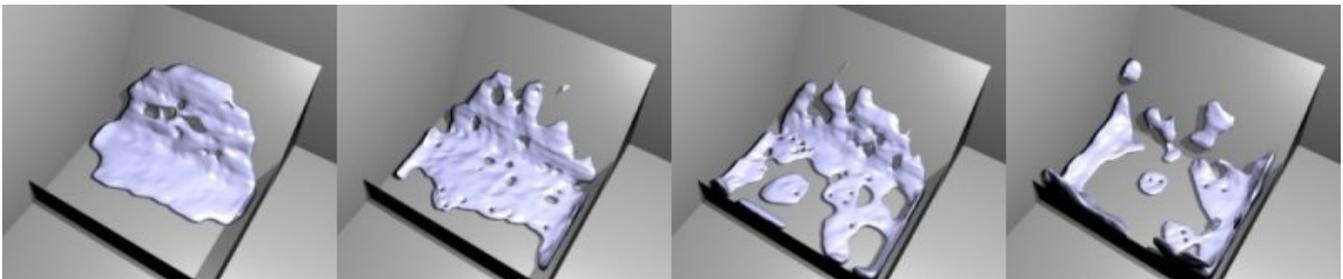


Fig. 2.1483: Example of the different boundary types for a drop falling onto the slanted wall. From left to right: no-slip, part-slip 0.3, part-slip 0.7 and free-slip.

FIXME(Template Unsupported: Doc:2.6/Manual/Physics/Fluid/animated\_mesh\_export; {{Doc:2.6/Manual/Physics/Fluid/animated\_mesh\_e}})

**PartSlip Amount** Amount of mixing between no- and free-slip, described above.

**Note:** Moving obstacles support

Blender supports now moving obstacles.

In the past, a moving obstacle was automatically treated as no slip (sticky), so if you wanted to splash off of a moving object, you had to put a transparent plane in the spot where the fluid will hit the moving object, exactly aligned and shaped as the object, to fake the splash. This is not needed anymore.

**Impact Factor** Amount of fluid volume correction for gain/loss from impacting with moving objects. If this object is not moving, this setting has no effect. However, if it is and the fluid collides with it, a negative value takes volume away from the Domain, and a positive number adds to it. Ranges from -2.0 to 10.0.

## Controlling the fluid volume

To control the volume of the fluid simulation, you can set objects in the scene to add or absorb fluid within the [Fluid Domain](#).

### Inflow

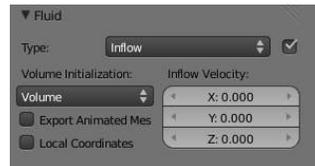


Fig. 2.1484: Fluid inflow settings

This object will put fluid into the simulation, like a water tap.

FIXME(Template Unsupported: Doc:2.6/Manual/Physics/Fluid/volume\_init; {{Doc:2.6/Manual/Physics/Fluid/volume\_init}}  
)

**Inflow velocity** Speed of the fluid that is created inside of the object.

**Local Coords / Enable** Use local coordinates for the inflow. This is useful if the inflow object is moving or rotating, as the inflow stream will follow/copy that motion. If disabled, the inflow location and direction do not change.

FIXME(Template Unsupported: Doc:2.6/Manual/Physics/Fluid/animated\_mesh\_export; {{Doc:2.6/Manual/Physics/Fluid/animated\_mesh\_e}}  
)

### Outflow



Fig. 2.1485: Fluid outflow settings

Any fluid that enters the region of this object will be deleted (think of a drain or a black hole). This can be useful in combination with an inflow to prevent the whole domain from filling up. When enabled, this is like a tornado (waterspout) or “wet vac” vacuum cleaner, and the part where the fluid disappears will follow the object as it moves around.

FIXME(Template Unsupported: Doc:2.6/Manual/Physics/Fluid/volume\_init; {{Doc:2.6/Manual/Physics/Fluid/volume\_init}}  
)

FIXME(Template Unsupported: Doc:2.6/Manual/Physics/Fluid/animated\_mesh\_export; {{Doc:2.6/Manual/Physics/Fluid/animated\_mesh\_e}}  
)

### Particle

This type can be used to display particles created during the simulation. For now only tracers swimming along with the fluid are supported. Note that the object can have any shape, position or type - once the particle button is pressed, a particle system with the fluid simulation particles will be created for it at the correct position. When moving the original object, it might be necessary to delete the particle system, disable the fluidsim particles, and enable them again. The fluidsim particles are currently also unaffected by any other particle forces or settings.

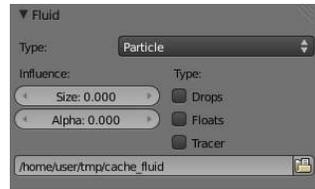


Fig. 2.1486: Fluid particle settings

## Influence

**Size Influence** The particles can have different sizes, if this value is 0 all are forced to be the same size.

**Alpha Influence** If this value is  $>0$ , the alpha values of the particles are changed according to their size.

## Particle type

**Drops** Surface splashes of the fluid result in droplets being strewn about, like fresh water, with low Surface Tension.

**Floats** The surface tension of the fluid is higher and the fluid heavier, like cold seawater and soup. Breakaways are clumpier and fall back to the surface faster than *Drops*, as with high Surface Tension.

**Tracer** Droplets follow the surface of the water where it existed, like a fog suspended above previous fluid levels. Use this to see where the fluid level has been.

**Path (bake directory)** The simulation run from which to load the particles. This should usually have the same value as the fluid domain object (e.g. copy by `Ctrl-C`, `Ctrl-V`).

## Control

### Description

Using the Lattice-boltzman method, the fluid is controlled using particles which define local force fields and are generated automatically from either a physical simulation or a sequence of target shapes. At the same time, as much as possible of the natural fluid motion is preserved.

### Examples

In this examples, we use the Fluid Control option to control part of the fluid so that it has a certain shape (the sphere drop or the teapot drop) before it falls in the rest of the fluid:



Fig. 2.1487: Falling drop



Fig. 2.1488: “Magic Fluid Control”

## Options

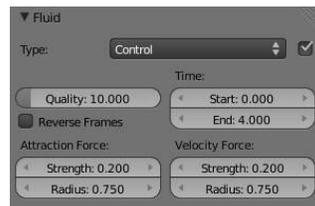


Fig. 2.1489: Fluid control options.

**Quality** Higher quality result in more control particles for the fluid control object.

**Reverse Frames** The control particle movement gets reversed.

**Time** You specify the start and end time during which time the fluid control object is active.

**Attraction force** The attraction force specifies the force which gets emitted by the fluid control object. Positive force results in attraction of the fluid, negative force in avoidance.

**Velocity force** If the fluid control object moves, the resulting velocity can also introduce a force to the fluid.

## See also

Release notes: [http://wiki.blender.org/index.php/Template:Release\\_Notes/2.48/FluidControl](http://wiki.blender.org/index.php/Template:Release_Notes/2.48/FluidControl)

## Animating the fluid properties

A new type of Ipo Curve, *FluidSim*, is available for fluid domain objects. Unlike most other animatable values in Blender, *FluidSim* Ipos cannot be keyframed by simply using the **I** key; you must manually set values by clicking in the Ipo window. In order to set a keyframe, you must select the property you wish to animate in the Ipo window and **Ctrl-LMB** click to set the keyframe to the desired location in the Ipo window.

---

### Tip: Enter Properties

Note that you do not have to be exact on where you click; we recommend that after you set the control point, open the *Transform Properties* panel (N) and round the X value to a whole frame number, and then set the Y value that you wish.

---

The fluid domain has several channels that control the fluid over time:

**Fac-Visc** A multiplicative factor in the fluid’s viscosity coefficient. It must be set before baking, and changes the viscosity of the fluid over time, so you can turn water into wi? oil, for example!

**Fac-Tim** Changes the speed of the simulation; like the Speed Control in the VSE can speed up or slow down a video, this curve can speed up or slow down the fluid motion during a frame sequence. If the value for *Fac-Tim* is less than or equal to zero, time (and the fluid) stands still; the fluid freezes. For values between 0.0 and 1.0, the fluid runs slower and will look thicker. 1.0 is normal fluid motion, and values greater than 1.0 will make the fluid flow faster, possibly appearing thinner.

**GravX / GravY / GravZ** The XYZ vector for gravity changes; aka inertia of the fluid itself (think drinking a cup of coffee while driving NASCAR at Talladega, or sipping an espresso on the autobahn, or watering the plants on the Space Shuttle). Changes in these curves make the fluid slosh around due to external forces.

The *Fluid*, *Obstacle*, *Inflow*, *Outflow* and *Particle* objects can use the following channels:

**VelX / VelY / VelZ** spurts of water from the garden hose can be simulated via these curves, to mimic changes in pressure and/or direction. It also can be used to simulate the effect of wind on a stream of water, for example.

**Active** When Active transitions from 0.0 to something greater than 0 (such as between 0.1 and 1.0), the object's function (designated as an *Inflow*, or *Outflow*, etc.) resumes its effect. Crossing down to 0.0 and then at some point, back up, re-establishes the effect and the resulting fluid sim. Use this for dripping, or any kind of intermittent inflow. This active status also works for objects designated as *Outflow* and *Obstacle*, so you can also simulate (for example) a drain plugging up.

You can also control the force settings of *Control* objects:

**AttrForceStr, AttrForceRa** These curves control the values of the attraction force settings.

**VelForceStr, VelForceRa** These curves control the values of the velocity force settings.

## Fluid Hints

Some useful hints about fluid simulation in Blender:

- Don't be surprised, but you'll get whole bunch of mesh (.obj.gz) files after a simulation. One set for preview, and another for final. Each set has a .gz file for each frame of the animation. Each file contains the simulation result - so you'll need them.
- Currently these files will not be automatically deleted, so it is a good idea to e.g. create a dedicated directory to keep simulation results. Doing a fluid simulation is similar to clicking the *ANIM* button - you currently have to take care of organizing the fluid surface meshes in some directory yourself. If you want to stop using the fluid simulation, you can simply delete all the \*fluid\*.obj.gz files.
- Before running a high resolution simulation that might take hours, check the overall timing first by doing lower resolution runs.
- Fluid objects must be completely inside the bounding box of the domain object. If not, baking may not work correctly or at all. Fluid and obstacle objects can be meshes with complex geometries. Very thin objects might not appear in the simulation, if the chosen resolution is too coarse to resolve them (increasing it might solve this problem).
- Note that fluid simulation parameters, such as inflow velocity or the active flag can be animated with *Fluidsim* Ipos (see above).
- Don't try to do a complicated scene all at once. Blender has a powerful compositor that you can use to combine multiple animations.

For example, to produce an animation showing two separate fluid flows while keeping your domain small, render one .avi using the one flow. Then move the domain and render another .avi with the other flow using an alpha channel (in a separate B&W .avi?). Then, composite both .avi's using the compositor's add function. A third .avi is usually the smoke and mist and it is laid on top of everything as well. Add a rain sheet on top of the mist and spray and you'll have quite a storm brewing! And then lightning flashes, trash blowing around, all as separate animations, compositing the total for a truly spectacular result.

- If you're having trouble, or something isn't working as you think it should - let me know: send the .blend file and a problem description to [nils at thuerey dot de](mailto:nils@thuerey.de). Please check these wiki pages and the [blenderartists-forum](#) before sending a mail!

## Fluid Technical Details

### Physical correctness

Fluid animation can take a lot of time - the better you understand how it works, the easier it will be to estimate how the results will look. The algorithm used for Blender's fluid simulation is the *Lattice Boltzmann Method* (LBM); other fluid algorithms include *Navier-Stokes* (NS) solvers and *Smoothed Particle Hydrodynamics* (SPH) methods. LBM lies somewhere between these two.

For Blender's LBM solver, the following things will make the simulation harder to compute:

- Large domains.
- Long duration.
- Low viscosities.
- High velocities.

The viscosity of water is already really low, so especially for small resolutions, the turbulence of water can not be correctly captured. If you look closely, most simulations of fluids in computer graphics do not yet look like real water as of now. Generally, don't rely on the physical settings too much (such as physical domain size or length of the animation in seconds). Rather try to get the overall motion right with a low resolution, and then increase the resolution as much as possible or desired.

### Acknowledgements

The integration of the fluid simulator was done as a Google Summer-of-Code project. More information about the solver can be found at [www.ntoken.com](http://www.ntoken.com). These Animations were created with the solver before its integration into blender: [Adaptive Grids](#), [Interactive Animations](#). Thanks to Chris Want for organizing the Blender-SoC projects, and to Jonathan Merrit for mentoring this one! And of course thanks to Google for starting the whole thing... SoC progress updates were posted here: [SoC-Blenderfluid-Blog at PlanetSoC](#).

The solver itself was developed by Nils Thuerey with the help and supervision of the following people: U. Ruede, T. Pohl, C. Koerner, M. Thies, M. Oechsner and T. Hofmann at the Department of Computer Science 10 (System Simulation, LSS) in Erlangen, Germany.

- <http://www10.informatik.uni-erlangen.de/~sinithue/img/lsslogo.png>
- <http://www10.informatik.uni-erlangen.de/~sinithue/img/unierlangenlogo.png>

## Fluid Appendix

### Limitations & Workarounds

- If the setup seems to go wrong, make sure all the normals are correct (hence, enter *Edit mode*, select all, and recalculate normals once in a while).
- Currently there's a problem with zero gravity simulation - simply select a very small gravity until this is fixed.
- If an object is initialized as *Volume*, it has to be closed and have an inner side (a plane won't work). To use planes, switch to *Shell*, or extrude the plane.



Fig. 2.1490: “My cup runneth over”

- Blender freezes after clicking *BAKE*. Pressing `ESC` makes it work again after a while - this can happen if the resolution is too high and memory is swapped to hard disk, making everything horribly slow. Reducing the resolution should help in this case.
- Blender crashes after clicking *BAKE* - this can happen if the resolution is really high and more than 2GB are allocated, causing Blender to crash. Reduce the resolution. Many operating systems limit the total amount of memory that can be allocated by a *process*, such as Blender, even if the *machine* has more memory installed. Sux...
- The meshes should be closed, so if some parts of e.g. a fluid object are not initialized as fluid in the simulation, check that all parts of connected vertices are closed meshes. Unfortunately, the Suzanne (monkey) mesh in Blender is not a closed mesh (the eyes are separate).
- If the fluid simulation exits with an error message (stating e.g. that the “init has failed”), make sure you have valid settings for the domain object, e.g. by resetting them to the defaults.
- To import a single fluid surface mesh you can use this script: `.bobj.-Import-Script`.
- You may not be able to bake a fluid that takes more than 1GB, not even with the LargeAddressAware build - it might be a limitation of the current fluid engine.
- Note that first frame may well take only a few hundred MBs of RAM memory, but latter ones go over one GB, which may be why your bake fails after awhile. If so, try to bake one frame at the middle or end at full res so you’ll see if it works.
- Memory used doubles when you set surface subdivision from 1 to 2.
- Using “generate particles” will also add memory requirements, as they increase surface area and complexity. Ordinary fluid-sim generated particles probably eat less memory.

### See also

- [Tutorial 1: Very Basic Introduction](#)
- [Tutorial 2: The Next Step](#)
- [Tutorial 1&2 Gui Changes for newer builds](#)
- [Another BSoD fluid tutorial](#)
- [Developer documentation \(implementation, dependencies, ...\)](#)

### External links

- [An Introduction to Fluid Simulations in Blender \(video\) \(Blendernation link\)](#)  
Learn the basics of how to set up a fluid simulation in Blender with an obstacle.
- [Fluid Simulator Tutorial \(video\) \(Blendernation link\)](#)  
Very easy to understand video-tutorial to fluid simulation newcomers. Also covers some of the most common pitfalls.
- [Guide on Blender Fluid Simulator’s Parameters \(Blendernation link\)](#)

## 2.7.6 Particles

### Introduction

Particles are lots of items emitted from mesh objects, typically in the thousands. Each particle can be a point of light or a mesh, and be joined or dynamic. They may react to many different influences and forces, and have the notion of a lifespan. Dynamic particles can represent fire, smoke, mist, and other things such as dust or magic spells.

**Hair** type particles are a subset of regular particles. Hair systems form strands that can represent hair, fur, grass and bristles.

You see particles as a *Particle* modifier, but all settings are done in the *Particle* tab.

Particles generally flow out from their mesh into space. Their movement can be affected by many things, including:

- Initial velocity out from the mesh.
- Movement of the emitter (vertex, face or object) itself.
- Movement according to “gravity” or “air resistance”.
- Influence of force fields like wind, vortexes or guided along a curve.
- Interaction with other objects like collisions.
- Partially intelligent members of a flock (herd, school, ...), that react to other members of their flock, while trying to reach a target or avoid predators.
- Smooth motion with softbody physics (only *Hair* particle systems).
- Or even manual transformation with **Lattices**.

Particles may be rendered as:

- **Halos** (for Flames, Smoke, Clouds).
- Meshes which in turn may be animated (e.g. fish, bees, ...). In these cases, each particle “carries” another object.
- **Strands** (for **Hair**, **Fur**, **Grass**); the complete way of a particle will be shown as a strand. These strands can be manipulated in the 3D window (combing, adding, cutting, moving, etc).

Every object may carry many particle systems. Each particle system may contain up to 100.000 particles. Certain particle types (*Hair* and *Keyed*) may have up to 10.000 children for each particle (children move and emit more or less like their respective parents). The size of your memory and your patience are your practical boundaries.

### Workflow

The process for working with standard particles is:

- Create the mesh which will emit the particles.
- Create one or more Particle Systems to emit from the mesh. Many times, multiple particle systems interact or merge with each other to achieve the overall desired effect.
- Tailor each Particle System’s settings to achieve the desired effect.
- Animate the base mesh and other particle meshes involved in the scene.
- Define and shape the path and flow of the particles.
- For **Hair** particle systems: Sculpt the emitter’s flow (cut the hair to length and comb it for example).
- Make final render and do physics simulation(s), and tweak as needed.

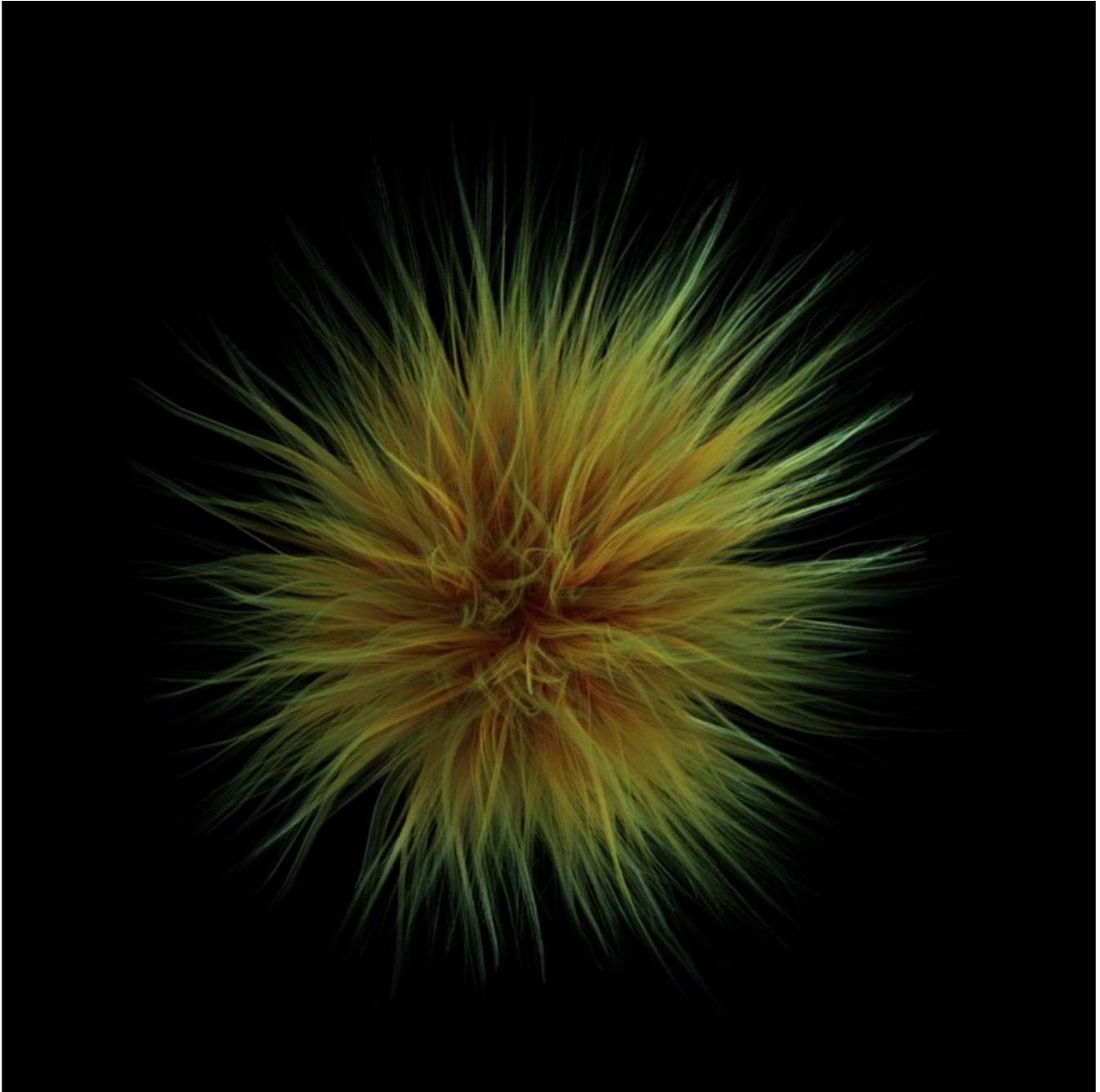


Fig. 2.1491: Image 1: Some fur made from particles (Blend file).

## Creating a Particle System

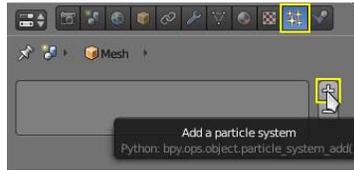


Fig. 2.1492: Image 2: Adding a particle system.

To add a new particle system to an object, go to the *Particles* tab of the object *Settings* editor and click the small + button. An object can have many Particle Systems.

Each particle system has separate settings attached to it. These settings can be shared among different particle systems, so one doesn't have to copy every setting manually and can use the same effect on multiple objects. Using the *Random* property they can be randomized to look slightly different, even when using the same settings.

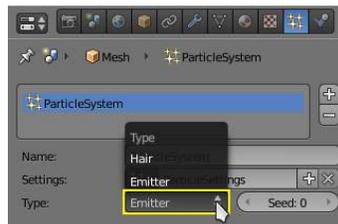


Fig. 2.1493: Image 3: Particle system types.

**Types of Particle systems** After you have created a particle system, the *Property* window fills with many panels and buttons. But don't panic! There are two different types of particle systems, and you can change between these two with the *Type* drop-down list:

**Emitter** This parallels the old system to the greatest extent. In such a system, particles are emitted from the selected object from the *Start* frame to the *End* frame and have a certain lifespan.

**Hair** This system type is rendered as strands and has some very special properties: it may be edited in the 3D window in realtime and you can also animate the strands with [Cloth Simulation](#).

The settings in the *Particle System* panel are partially different for each system type. For example, in *Image 3* they are shown for only system type *Emitter*.

**Common Options** Each system has the same basic sets of controls, but options within those sets vary based on the system employed. These sets of controls are:

**Emission** Settings for the initial distribution of particles on the emitter and the way they are born into the scene.

**Cache** In order to increase realtime response and avoid unnecessary recalculation of particles, the particle data can be cached in memory or stored on disk.

**Velocity** Initial speed of particles.

**Rotation** Rotational behavior of particles.

**Physics** How the movement of the particles behaves.

**Render** Rendering options.

**Display** Realtime display in the 3D View.

**Children** Control the creation of additional child particles.

**Field Weights** Factors for external forces.

**Force Field Settings** Makes particles force fields.

**Vertex Groups** Influencing various settings with vertex groups.

## Links

- [Tutorials](#)
- [Physics Caching and Baking](#)
- [Particle Rewrite Documentation](#)
- [Thoughts about the particle rewrite code](#)
- [Static Particle Fur Library](#)

## Particle Emission

The *Emitter* system works just like its name says: it emits/produces particles for a certain amount of time. In such a system, particles are emitted from the selected object from the *Start* frame to the *End* frame and have a certain lifespan. These particles are rendered default as *Halos*, but you may also render these kind of particles as objects (depending on the particle system's render settings, see [Visualization](#)).

## Options

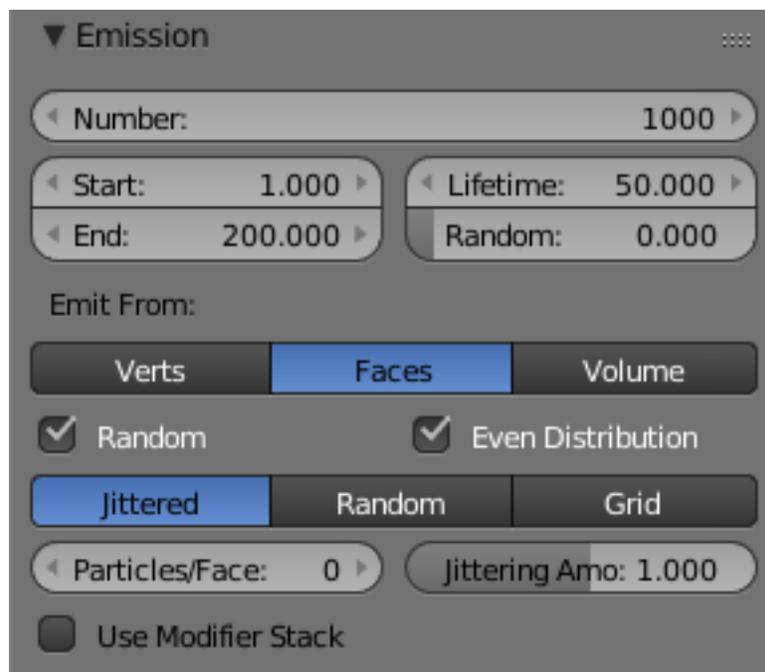


Fig. 2.1494: Particle emission settings

The buttons in the *Emission* panel control the way particles are emitted over time:

**Amount** The maximum amount of parent particles used in the simulation.

**Start** The start frame of particle emission. You may set negative values, which enables you to start the simulation before the actual rendering.

**End** The end frame of particle emission.

**Lifetime** The lifetime (in frames) of the particles.

**Random** A random variation of the lifetime of a given particle. The shortest possible lifetime is  $Lifetime \times (1 - Rand)$ . Values above 1.0 are not allowed. For example with the default *Lifetime* value of 50 a *Random* setting of 0.5 will give you particles with lives ranging from 50 frames to  $50 \times (1.0 - 0.5) = 25$  frames, and with a *Random* setting of 0.75 you'll get particles with lives ranging from 50 frames to  $50 \times (1.0 - 0.75) = 12.5$  frames.

**Emission Location** *Emit From* parameters define how and where the particles are emitted, giving precise control over their distribution. You may use vertex groups to confine the emission, that is done in the *Vertexgroups* panel.

**Verts** Emit particles from the vertices of a mesh.

**Faces** Emit particles from the surface of a mesh's faces.

**Volume** Emit particles from the volume of an enclosed mesh.

**Distribution Settings** These settings control how the emissions of particles are distributed throughout the emission locations

**Random** The emitter element indices are gone through in a random order instead of linearly (one after the other).

For Faces and Volume, additional options appear:

**Even Distribution** Particle distribution is made even based on surface area of the elements, i.e. small elements emit less particles than large elements, so that the particle density is even.

**Jittered** Particles are placed at jittered intervals on the emitter elements.

**Particles/Face** Number of emissions per face (0 = automatic).

**JitteringAmount** Amount of jitter applied to the sampling.

**Random** Particles are emitted from random locations in the emitter's elements.

**Grid** Particles are set in a 3d grid and particles near/in the elements are kept.

**Invert Grid** Invert what is considered the object and what is not.

**Hexagonal** Uses a hexagonal shaped grid instead of a rectangular one.

**Resolution** Resolution of the grid.

**Random** Add a random offset to grid locations.

---

**Tip:** Your mesh must be *manifold* to emit particles from the volume.

Some modifiers like *Edge Split* break up the surface, in which case volume emission will not work correctly!

---

**Use Modifier Stack** Take any [Modifiers](#) above the particle modifier in the [Modifier Stack](#) into account when emitting particles.

Note that particles may differ in the final render if these modifiers generate different geometry between the viewport and render.

## Particle Physics

### Introduction

The movement of particles may be controlled in a multitude of ways. With particles physics: there are five different systems:

**None (No Physics)** It doesn't give the particles any motion, which makes them belong to no physics system.

**Newtonian** Movement according to physical laws.

**Keyed** Dynamic or static particles where the (animated) targets are other particle systems.

**Boids** Particles with limited artificial intelligence, including behavior and rules programming, ideal for flocks of birds or schools of fishes, or predators vs preys simulations.

**Fluid** Movement according to fluid laws (based on Smoothed Particle Hydrodynamics technique).

Additional ways of moving particles:

- By softbody animation (only for Hair particle systems).
- By forcefields and along curves.
- By lattices.

Here we will discuss only the particle physics in the narrower sense, i.e. the settings in the Physics panel.

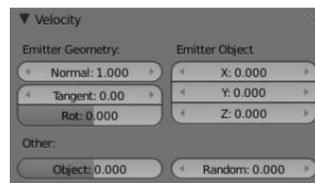


Fig. 2.1495: Image 3: Initial velocity.

**Velocity** The initial velocity of particles can be set through different parameters, based on the type of the particle system (see Particle System tab). If the particle system type is Emitter or Hair, then the following parameters give the particle an initial velocity in the direction of...

### Emitter Geometry

**Normal** The emitter's surface normals (i.e. let the surface normal give the particle a starting speed).

**Tangent** Let the tangent speed give the particle a starting speed.

**Rot** Rotates the surface tangent.

### Emitter Object

**Align X,Y,Z** Give an initial velocity in the X, Y, and Z axes.

**Object** The emitter objects movement (i.e. let the object give the particle a starting speed).

**Random** Gives the starting speed a random variation. You can use a texture to only change the value, see Controlling Emission, Interaction and Time).

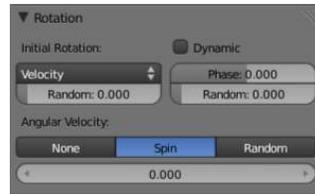


Fig. 2.1496: Image 4: Particles rotation.

**Rotation** These parameters specify how the individual particles are rotated during their travel. To visualize the rotation of a particle you should choose visualization type Axis in the Visualization panel and increase the Draw Size.

**Initial Rotation Mode** Sets the initial rotation of the particle by aligning the x-axis in the direction of:

**None** the global x-axis.

**Normal** Orient to the emitter's surface normal, the objects Y axis points outwards.

**Normal-Tangent** As with normal, orient the Y axis to the surface normal. Also orient the X axis to the tangent for control over the objects rotation about the normal. requires UV coordinates, the UV rotation effects the objects orientation, currently uses the active UV layer. This allow deformation without the objects rotating in relation to their surface.

**Velocity** the particle's initial velocity.

**Global X/Global Y/Global Z** one of the global axes

**Object X/Object Y/Object Z** one of the emitter object axes.

**Random** Randomizes rotation.

**Dynamic** If enabled, only initializes particles to the wanted rotation and angular velocity and let's physics handle the rest. Particles then change their angular velocity if they collide with other objects (like in the real world due to friction between the colliding surfaces). Otherwise the angular velocity is predetermined at all times (i.e. set rotation to dynamic/constant).

**Phase** Initial rotation phase

**Random** Rand allows a random variation of the Phase.

**Angular Velocity** The magnitude of angular velocity, the dropdown specifies the axis of angular velocity to be

**None** a zero vector (no rotation).

**Spin** the particles velocity vector.

**Random** a random vector.

If you use a Curve Guide and want the particles to follow the curve, you have to set Angular Velocity to Spin and leave the rotation on Constant (i.e. don't turn on Dynamic). Curve Follow does not work for particles.

### Common Physics Settings

**Size** Sets the size of the particles.

**Random Size** Give the particles a random size variation.

**Mass** Specify the mass of the particles.

**Multiply mass with particle size** Causes larger particles to have larger masses.

**No Physics** At first a Physics type that makes the particles do nothing could seem a bit strange, but it can be very useful at times. None physics make the particles stick to their emitter their whole life time. The initial velocities here are for example used to give a velocity to particles that are affected by a harmonic effector with this physics type when the effect of the effector ends.

Moreover, it can be very convenient to have particles at disposal (whose both Unborn and Died are visible on render) to groom vegetation and/or ecosystems using Object, Group or Billboard types of visualization.

**Field Weights** The Field Weight Panel allows you to control how much influence each type of external force field, or effector, has on the particle system. Force fields are external forces that give dynamic systems motion. The force fields types are detailed on the [Force Field Page](#).

**Effector Group** Limit effectors to a specified group. Only effectors in this group will have an effect on the current system.

**Gravity** Control how much the Global Gravity has an effect on the system.

**All** Scale all of the effector weights.

**Force Fields** The Force Field Settings Panel allows you to make each individual act as a force field, allowing them to affect other dynamic systems, or even, each other.

**Self Effect** Causes the particle force fields to have an effect on other particles within the same system.

**Amount** Set how many of the particles act as force fields. 0 means all of them are effectors.

You can give particle systems up to 2 force fields. By default they do not have any. Choose an effector type from the dropdowns to enable them. Settings are described on the [Force Field Page](#).

## Newtonian Physics

These are the “normal” particle physics. Particles start their life with the specified initial velocities and angular velocities, and move according to Newtonian forces. The response to environment and to forces is computed differently, according to any given integrator chosen by the animator.

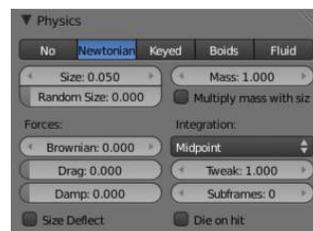


Fig. 2.1497: Image 5: Newtonian Physics.

## Forces

**Brownian** Specify the amount of Brownian motion. Brownian motion adds random motion to the particles based on a Brownian noise field. This is nice to simulate small, random wind forces.

**Drag** A force that reduces particle velocity in relation to its speed and size (useful in order to simulate Air-Drag or Water-Drag).

**Damp** Reduces particle velocity (deceleration, friction, dampening).

## Collision

**Size Deflect** Use the particle size in deflections.

**Die on Hit** Kill particle when it hits a deflector object.

**Integration** Integrators are a set of mathematical methods available to calculate the movement of particles. The following guidelines will help to choose a proper integrator, according to the behavior aimed at by the animator.

**Euler** Also known as “Forward Euler”. Simplest integrator. Very fast but also with less exact results. If no dampening is used, particles get more and more energy over time. For example, bouncing particles will bounce higher and higher each time. Should not be confused with “Backward Euler” (not implemented) which has the opposite feature, energies decrease over time, even with no dampening. Use this integrator for short simulations or simulations with a lot of dampening where speedy calculations is more important than accuracy.

**Varlet** Very fast and stable integrator, energy is conserved over time with very little numerical dissipation. +

**Midpoint** Also known as “2nd order Runge-Kutta”. Slower than Euler but much more stable. If the acceleration is constant (no drag for example), it is energy conservative. It should be noted that in example of the bouncing particles, the particles might bounce higher than they started once in a while, but this is not a trend. This integrator is a generally good integrator for use in most cases. +

**RK4** Short for “4th order Runge-Kutta”. Similar to Midpoint but slower and in most cases more accurate. It is energy conservative even if the acceleration is not constant. Only needed in complex simulations where Midpoint is found not to be accurate enough. +

**Timestep** The simulation time step per frame.

**Subframes** Subframes to simulate for improved stability and finer granularity in simulations. Use higher values for faster moving particles.

## Keyed Particles

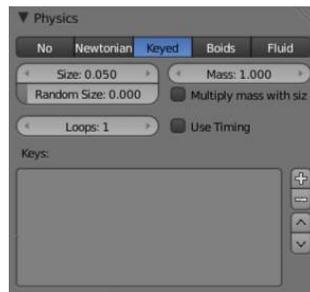


Fig. 2.1498: Image 6: Keyed Physics.

The particle paths of keyed particles are determined from the emitter to another particle system’s particles. This allows creation of chains of systems with keyed physics to create long strands or groovy moving particles. Basically the particles have no dynamics but are interpolated from one system to the next at drawtime. Because you have so much control over these kind of systems, you may use it

For example, for machines handling fibers (animation of a loom, ...). In (Image 3), the strands flow from the bottom system (First keyed) to the second keyed system in the middle, and from that to the top system that has None-Physics. Since you may animate each emitter object as you like, you can do arbitrarily complex animations.

**Setup** To setup Keyed particles you need at least two particle systems.

The first system has keyed physics, and it needs the option First activated. This will be the system that is visible.

- The second system may be another keyed system but without the option First, or a normal particle system. This second system is the target of the keyed system.

**Loops** Sets the number of times the keys are looped. Disabled if *Use Timing* is enabled.

## Keys

**Key Targets** You have to enter the name of the object which bears the target system and if there are multiple particle systems the number of the system.

Click the **Plus** to add a key, then select the object.

If you use only one keyed system the particles will travel in their lifetime from the emitter to the target. A shorter lifetime means faster movement. If you have more than one keyed system in a chain, the lifetime will be split equally. This may lead to varying particle speeds between the targets.

## Timing

**Use Timing** Timing works together with the Time slider for the other keyed systems in a chain. The Time slider allows to define a fraction of particle lifetime for particle movement.

An example: let's assume that you have two keyed systems in a chain and a third system as target. The particle lifetime of the first system shall be 50 keys. The particles will travel in 25 frames from the first keyed system to the second, and in further 25 frames from the second system to the target. If you use the Timed button for the first system, the Time slider appears in the second systems panel. Its default value is 0.5, so the time is equally split between the systems. If you set Time to 1, the movement from the first system to the second will get all the lifetime (the particles will die at the second system).

If you set Time to 0 the particles will start at the second Air system and travel to the target.

## Boids



Fig. 2.1499: Image 7: Boid Physics.

Boids particle systems can be set to follow basic rules and behaviors. They are useful for simulating flocks, swarms, herds and schools of various kind of animals, insects and fishes. They can react on the presence of other objects and on the members of their own system. Boids can handle only a certain amount of information, therefore the sequence of the Behaviour settings is very important. In certain situations only the first three parameter are evaluated.

To view the subpanel to the right, add a *Particle System* of type *Emitter* and look in the middle area of the *Particle System* tab.

**Physics** Boids try to avoid objects with activated Deflection. They try to reach objects with positive Spherical fields, and fly from objects with negative Spherical fields. The objects have to share one common layer to have effect. It is not necessary to render this common layer, so you may use invisible influences.

Boids can different physics depending on whether they are in the air, or on land (on collision object)

**Allow Flight** Allow boids to move in the air.

**Allow Land** Allow boids to move on land.

**Allow Climbing** Allow boids to climb goal objects.

**Max Air Speed** Set the Maximum velocity in the air.

**Min Air Speed** Set the Minimum velocity in the air.

**Max Air Acceleration** Lateral acceleration in air, percent of max velocity (turn). Defines how fast a boid is able to change direction.

**Max Air Angular Velocity** Tangential acceleration in air, percent 180 degrees. Defines how much the boid can suddenly accelerate in order to fulfill a rule.

**Air Personal Space** Radius of boids personal space in air. Percentage of particle size.

**Landing Smoothness** How smoothly the boids land.

**Max Land Speed** Set the Maximum velocity on land.

**Jump Speed** Maximum speed for jumping

**Max Land Acceleration** Lateral acceleration on land, percent of max velocity (turn). Defines how fast a boid is able to change direction.

**Max Land Angular Velocity** Tangential acceleration on land, percent 180 degrees. Defines how much the boid can suddenly accelerate in order to fulfill a rule.

**Land Personal Space** Radius of boids personal space on land. Percentage of particle size.

**Land Stick Force** How strong a force must be to start effecting a boid on land.

**Banking** Amount of rotation around velocity vector on turns. Banking of (1.0 == natural banking).

**Pitch** Amount of rotation around side vector.

**Height** Boid height relative to particle size.

## Battle

**Health** Initial boid health when born.

**Strength** Maximum caused damage per second on attack.

**Aggression** Boid will fight this times stronger than enemy.

**Accuracy** Accuracy of attack.

**Range** Maximum distance of which a boid can attack.

**Alliance** The relations box allows you to set up other particle systems to react with the boids. Setting the type to *Enemy* will cause the systems to fight with each other. *Friend* will make the systems work together. *Neutral* will not cause them to align or fight with each other.

**Deflectors and Effectors** As mentioned before, very much like Newtonian particles, Boids will react to the surrounding deflectors and fields, according to the needs of the animator:

**Deflection:** Boids will try to avoid deflector objects according to the Collision rule's weight. It works best for convex surfaces (some work needed for concave surfaces). For boid physics, Spherical fields define the way the objects having the field are seen by others. So a negative Spherical field (on an object or a particle system) will be a predator to all other boids particle systems, and a positive field will be a goal to all other boids particle systems.

When you select an object with a particle system set on, you have in the Fields tab a little menu stating if the field should apply to the emitter object or to the particle system. You have to select the particle system name if you want prey particles to flee away from predator particles.

**Spherical fields:** These effectors could be predators (negative Strength) that boids try to avoid or targets (positive Strength) that boids try to reach according to the (respectively) Avoid and Goal rules' weights. Spherical's effective Strength is multiplied by the actual relevant weight (e.g. if either Strength or Goal is null, then a flock of boids won't track a positive Spherical field). You can also activate Die on hit (Extras panel) so that a prey particle simply disappears when "attacked" by a predator particle which reaches it. To make this work, the predator particles have to have a spherical field with negative force, it is not sufficient just to set a positive goal for the prey particles (but you may set the predators force strength to -0.01). The size of the predators and the prey can be set with the Size button in the Extras panel.

**Boid Brain** The Boid Brain panel controls how the boids particles will react with each other. The boids' behavior is controlled by a list of rules. Only a certain amount of information in the list can be evaluated. If the memory capacity is exceeded, the remaining rules are ignored.

The rules are by default parsed from top-list to bottom-list (thus giving explicit priorities), and the order can be modified using the little arrows buttons on the right side.

The list of rules available are:

**Goal** Seek goal (objects with Spherical fields and positive Strength)

**Predict** Predict target's movements

**Avoid** Avoid "predators" (objects with Spherical fields and negative Strength)

**Predict** Predict target's movements

**Fear Factor** Avoid object if danger from it is above this threshold

**Avoid Collision** Avoid objects with activated Deflection

**Boids** Avoid collision with other boids

**Deflectors** Avoid collision with deflector objects

**Look Ahead** Time to look ahead in seconds

**Separate** Boids move away from each other

**Flock** Copy movements of neighboring boids, but avoid each other

**Follow Leader** Follows a leader object instead of a boid

**Distance** Distance behind leader to follow

**Line** Follow the leader in a line

**Average Speed** Maintain average velocity.

**Speed** Percentage of maximum speed

**Wander** How fast velocity's direction is randomized

**Level** How much velocity's Z component is kept constant

**Fight** Move toward nearby boids

**Fight Distance** Attack boids at a maximum of this distance

**Flee Distance** Flee to this distance

**Rule Evaluation** There are three ways control how rules are evaluated.

**Average** All rules are averaged.

**Random** A random rule is selected for each boid.

**Fuzzy** Uses fuzzy logic to evaluate rules. Rules are gone through top to bottom. Only the first rule that effect above fuzziness threshold is evaluated. The value should be considered how hard the boid will try to respect a given rule (a value of 1.000 means the Boid will always stick to it, a value of 0.000 means it will never). If the boid meets more than one conflicting condition at the same time, it will try to fulfill all the rules according to the respective weight of each.

Please note that a given boid will try as much as it can to comply to each of the rules he is given, but it is more than likely that some rule will take precedence on other in some cases. For example, in order to avoid a predator, a boid could probably “forget” about Collision, Crowd and Center rules, meaning that “while panicked” it could well run into obstacles, for example, even if instructed not to, most of the time.

As a final note, the Collision algorithm is still not perfect and in research progress, so you can expect wrong behaviors at some occasion. It is worked on.

## Fluid Physics

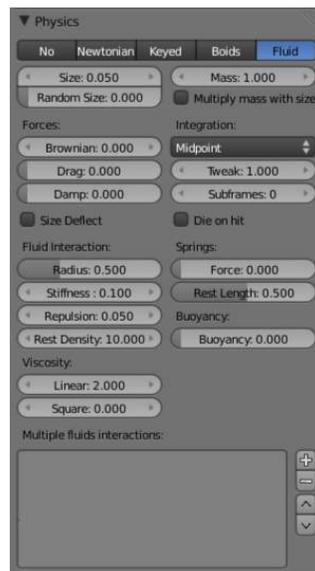


Fig. 2.1500: Image 8: Fluid Physics.

Fluid simulations are widely used in CG, and a very desired feature of any particle system, fluid particles are similar to newtonian ones but this time particles are influenced by internal forces like pressure, surface tension, viscosity, springs, etc. Blender particle fluids use the SPH techniques to solve the particles fluid equations.

Smoothed-particle hydrodynamics (SPH) is a computational method used for simulating fluid flows. It has been used in many fields of research, including astrophysics, ballistics, vulcanology, and oceanography. It is a mesh-free Lagrangian method (where the co-ordinates move with the fluid), and the resolution of the method can easily be adjusted with respect to variables such as the density.

From liquids to slime, goo to sand and wispy smoke the possibilities are endless.

**Settings** Fluid physics share options with [Newtonian Physics](#). These are covered on that page.

### Fluid Properties

**Stiffness** How incompressible the fluid is.

**Viscosity** Linear viscosity. Use lower viscosity for thicker fluids.

**Buoyancy** Artificial buoyancy force in negative gravity direction based on pressure differences inside the fluid.

### Advanced

**Repulsion Factor** How strongly the fluid tries to keep from clustering (factor of stiffness). Check box sets repulsion as a factor of stiffness.

**Stiff Viscosity** Creates viscosity for expanding fluid. Check box sets this to be a factor of normal viscosity.

**Interaction Radius** Fluid's interaction radius. Check box sets this to be a factor of 4\*particle size.

**Rest Density** Density of fluid when at rest. Check box sets this to be a factor of default density.

### Springs

**Force** Spring force

**Rest Length** Rest length of springs. Factor of particle radius. Check box sets this to be a factor of 2\*particle size.

**Viscoelastic Springs** Use viscoelastic springs instead of Hooke's springs.

**Elastic Limit** How much the spring has to be stretched/compressed in order to change its rest length

**Plasticity** How much the spring rest length can change after the elastic limit is crossed.

**Initial Rest Length** Use initial length as spring rest length instead of 2\*particle size.

**Frames** Create springs for this number of frames since particle's birth (0 is always).

### Particle Visualization

With the items in the *Display* and *Render* panel you can set the way the particles will be rendered or depicted in the view ports in various ways. Some options are valid only for the 3D window, the particles then are rendered always as [Halos](#). Some of the options will be rendered as shown in the 3D window.

### Viewport Display

The Display Panel controls how particles are displayed in the 3d viewport. This does not necessarily determine how they will appear when rendered.

**None** The particles are not shown in the 3D window and are not rendered. The emitter may be rendered though.

**Point** Particles are displayed as square points. Their size is independent of the distance from the camera.

**Circle** Particles are displayed as circles that face the view. Their size is independent of the distance from the camera.

**Cross** Particles are displayed as 6-point crosses that align to the rotation of the particles. Their size is independent of the distance from the camera.

**Axis** Particles are displayed as 3-point axes. This useful if you want to see the orientation and rotation of particles in the view port. Increase the *Draw Size* until you can clearly distinguish the axis.

Particles visualized like Point, Circle, Cross and Axis don't have any special options, but can be very useful when you have multiple particle systems at play, if you don't want to confuse particles of one system from another (e.g. in simulations using *Boids* physics).

**Display** Specifies the percentage of all particles to show in the viewport (all particles are still rendered).

**Draw Size** Specifies how large (in pixels) the particles are drawn in the viewport (0 = default).

**Size** Draw the size of the particles with a circle.

**Velocity** Draw the velocity of the particles with a line that points in the direction of motion, and length relative to speed.

**Number** Draw the id-numbers of the particles in the order of emission.

**Color** The Color Menu allows you to draw particles according to certain particle properties.

**None** Particles are black.

**Material** Particles are colored according to the material they are given.

**Velocity** Color particles according to their speed. The color is a ramp from blue to green to red, Blue being the slowest, and Red being velocities approaching the value of *Max* or above. Increasing *Max* allows for a wider range of particle velocities.

**Acceleration** Color particles according to their acceleration.

## Render Settings

The Render Panel controls how particles appear when they are rendered.

**Material Index** Set which of the object's material is used to shade the particles.

**Parent** Use a different object's coordinates to determine the birth of particles.

**Emitter** When disabled, the emitter is no longer rendered. Activate the button *Emitter* to also render the mesh.

**Parents** Render also parent particles if child particles are used. Children have a lot of different deformation options, so the straight parents would stand between their curly children. So by default *Parents* are not rendered if you activate *Children*. See [Children](#)

**Unborn** Render particles before they are born.

**Died** Render particles after they have died. This is very useful if particles die in a collision (*Die on hit*), so you can cover objects with particles.

**None** When set to *None* particles are not rendered. This is useful if you are using the particles to duplicate objects.

**Halo** Halo particles are rendered as [Halo Type Materials](#).

**Trail Count** Set the number of trail particles. When greater than 1, additional options appear.

**Length in Frames** Path timing is in absolute frames.

**Length** End time of drawn path.

**Random** Give path lengths a random variation.

**Line** The Line visualization mode creates (more or less thin) polygon lines with the strand renderer in the direction of particles velocities. The thickness of the line is set with the parameter *Start* of the *Strands* shader (*Material* sub-context, *Links and Pipeline* panel).

**Back** Set the length of the particle's tail.

**Front** Set the length of the particle's head.

**Speed** Multiply the line length by particles' speed. The faster, the longer the line.

**Trail Count** See description in *Halo*.

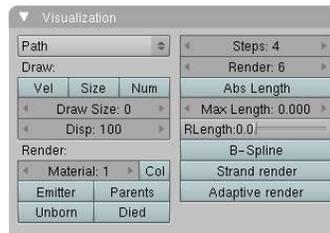


Fig. 2.1501: Image 3: The Visualization panel for Path visualization.

**Path** The *Path* visualization needs a [Hair](#) particle system or [Keyed](#) particles.

**Strand render** [Keypointstrands] Use the strand primitive for rendering. Very fast and effective renderer.

**Adaptive render** Tries to remove unnecessary geometry from the paths before rendering particle strands in order to make the render faster and easier on memory.

**Angle** How many degrees path has to curve to produce another render segment (straight parts of paths need fewer segments).

**Pixel** How many pixels path has to cover to produce another render segment (very short hair or long hair viewed from far away need fewer parts). (only for Adaptive render).

**B-Spline** Interpolate hair using B-Splines. This may be an option for you if you want to use low *Render* values. You loose a bit of control but gain smoother paths.

**Steps** Set the number of subdivisions of the rendered paths (the value is a power of 2). You should set this value carefully, because if you increase the render value by two you need four times more memory to render. Also the rendering is faster if you use low render values (sometimes drastically). But how low you can go with this value depends on the waviness of the hair.(the value is a power of 2). This means 0 steps give 1 subdivision, 1 give 2 subdivisions, 2->4, 3->8, 4->16, ...  $n \rightarrow 2^n$ .

### Timing Options

**Absolute Path Time** Path timing is in absolute frames.

**Start** Start time of the drawn path.

**End** End time of the drawn path.

**Random** Give the path length a random variation.

Please see also the manual page about [Strands](#) for an in depth description.

**Object** In the Object visualization mode the specified object (*Dupli Object:* field) is duplicated in place of each particle. The duplicated object has to be at the center of the coordinate system, or it will get an offset to the particle.

**Global** Use object's global coordinates for duplication.

**Size** Size of the objects

**Random Size** Give the objects a random size variation.

**Group** In the Group visualization mode, the objects that belong to the group (*GR:* field) are duplicated sequentially in the place of the particles.

**WholeGroup** Use the whole group at once, instead of one of its elements, the group being displayed in place of each particle.

**Use Count** Use objects multiple times in the same groups. Specify the order and number of times to repeat each object with the list box that appears. You can duplicate an object in the list with the **Plus** button, or remove a duplicate with the **Minus** button.

**Use Global** Use object's global coordinates for duplication.

**Pick Random** The objects in the group are selected in a random order, and only one object is displayed in place of a particle. Please note that this mechanism fully replaces old Blender particles system using parentage and DupliVerts to replace particles with actual geometry. This method is fully deprecated and doesn't work anymore.

**Size** Size of the objects

**Random Size** Give the objects a random size variation.

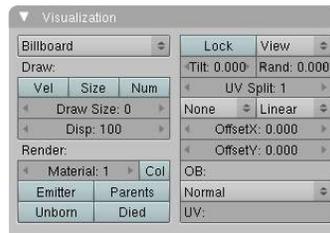


Fig. 2.1502: Image 4: Billboard visualization for particles.

**Billboard** *Billboards* are aligned square planes. They are aligned to the camera by default, but you can choose another object that they should be aligned to.

If you move a billboard around its target, it always faces the center of its target. The size of a billboard is set with the parameter *Size* of the particle (in Blender Units). You can use them e.g. for *Sprites*, or to replace *Halo* visualization. Everything that can be done with a halo can also be done with a billboard. But billboards are real objects, they are seen by raytracing, they appear behind transparent objects, they may have an arbitrary form and receive light and shadows. They are a bit more difficult to set up and take more render time and resources.

Texturing billboards (including animated textures with alpha) is done by using uv coordinates that are generated automatically for them so they can take an arbitrary shape. This works well for animations, because the alignment of the billboards can be dynamic. The textures can be animated in several ways:

- Depending on the particle lifetime (relative time).
- Depending on the particle starting time.
- Depending on the frame (absolute time).

You can use different sections of an image texture:

- Depending on the lifetime of the billboard.

- Depending on the emission time.
- Depending on align or tilt.

Since you use normal materials for the billboard you have all freedoms in mixing textures to your liking. The material itself is animated in absolute time.

The main thing to understand is that if the object doesn't have any *UV Layers*, you need to create at least one in the objects *Editing* context, for any of these to work. Moreover, the texture has to be set to UV coordinates in the *Map Input* panel. If you want to see examples for some of the animation possibilities, see the [Billboard Animation Tutorial](#).

An interesting alternative to billboards are in certain cases strands, because you can animate the shape of the strands. Because this visualization type has so much options it is explained in greater detail below.

You can limit the movement with these options. How the axis is prealigned at emission time.

**View** No prealignment, normal orientation to the target.

**X / Y / Z** Along the global X/Y/Z-axis respectively.

**Velocity** Along the speed vector of the particle.

**Lock** Locks the align axis, keeps this orientation, the billboard aligns only along one axis to it's target

**Billboard Object** The target object that the billboards are facing. By default, the active camera is used.

**Tilt Angle** Rotation angle of the billboards planes. A tilt of 1 rotates by 180 degrees (turns the billboard upside down).

**Random** Random variation of tilt.

**Offset X** Offset the billboard horizontally in relation to the particle center, this does not move the texture.

**Offset Y** Offset the billboard vertically in relation to the particle center.

**UV Channels** Billboards are just square polygons. To texture them in different ways we have to have a way to set what textures we want for the billboards and how we want them to be mapped to the squares. These can then be set in the texture mapping buttons to set wanted textures for different coordinates. You may use three different UV layers and get three different sets of UV coordinates, which can then be applied to different (or the same) textures.

**Billboard Normal UV** Coordinates are the same for every billboard, and just place the image straight on the square.

**Billboard Time-Index (X-Y)** Coordinates actually define single points in the texture plane with the x-axis as time and y-axis as the particle index. For example using a horizontal blend texture mapped to color from white to black will give us particles that start off as white and gradually change to black during their lifetime. On the other hand a vertical blend texture mapped to color from white to black will make the first particle to be white and the last particle to be black with the particles in between a shade of gray.

The animation of the UV textures is a bit tricky. The UV texture is split into rows and columns (N times N). The texture should be square. You have to use *UV Split* in the UV channel and fill in the name of the UV layer. This generated UV coordinates for this layer.

**Split UV's** The amount of rows/columns in the texture to be used. Coordinates are a single part of the *UV Split* grid, which is a n?n grid over the whole texture. What the part is used for each particle and at what time is determined by the *Offset* and *Animate* controls. These can be used to make each billboard unique or to use an "animated" texture for them by having each frame of the animation in a grid in a big image.

**Billboard Split UV** Set the name of the *UV layer* to use with billboards (you can use a different one for each *UV Channel*). By default, it is the active UV layer (check the *Mesh* panel in the *Editing* context).

**Animate** Dropdown menu, indicating how the split UVs could be animated (changing from particle to particle with time):

**None** No animation occurs on the particle itself, the billboard uses one section of the texture in it's lifetime.

**Age** The sections of the texture are gone through sequentially in particles' lifetimes.

**Angle** Change the section based on the angle of rotation around the *Align to* axis, if *View* is used the change is based on the amount of tilt.

**Frame** The section is changes according to the frame.

**Offset** Specifies how to choose the first part (of all the parts in the  $n \times n$  grid in the texture defined by the *UV Split* number) for all particles.

**None** All particles start from the first part.

**Linear** First particle will start from the first part and the last particle will start from the last part, the particles in between will get a part assigned linearly from the first to the last part.

**Random** Give a random starting part for every particle.

**Trail Count** See the description in *Halo*.

## Cache

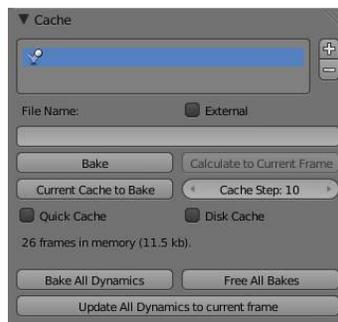


Fig. 2.1503: Image 4: Cache panel for particles.

*Emitter* systems use a unified system for caching and baking (together with softbody and cloth). The results of the simulation are automatically cached to disk when the animation is played, so that the next time it runs, it can play again quickly by reading in the results from the disk. If you *Bake* the simulation the cache is protected and you will be asked when you're trying to change a setting that will make a recalculating necessary.

**Tip:** Beware of the *Start* and *End* Settings

The simulation is only calculated for the positive frames in-between the *Start* and *End* frames of the *Bake* panel, whether you bake or not. So if you want a simulation longer than 250 frames you have to change the *End* frame!

## Caching

- As animation is played, each physics system writes each frame to disk, between the simulation start and end frames. These files are stored in folders with prefix `blendcache`, next to the `.blend` file. Note that for the cache to fill up, one has to start playback before or on the frame that the simulation starts.
- The cache is cleared automatically on changes - but not on all changes, so it may be necessary to free it manually e.g. if you change a force field.
- If it is impossible to write in the subdirectory there will be no caching.
- The cache can be freed per physics system with a button in the panels, or with the `Ctrl-B` shortcut key to free it for all selected objects.

- If the file path to the cache is longer than what is possible with your operating system (more than 250 characters for example), strange things might happen.

## Baking

- The system is protected against changes after baking.
- The *Bake* result is cleared also with `Ctrl-B` for all selected objects or click on *Free Bake* for a singular particle system.
- If the mesh changes the simulation is not calculated anew.
- Sorry: no bake editing for particles like for softbodies and clothes.

Two notes at the end:

- For renderfarms, it is best to bake all the physics systems, and then copy the blendcache to the renderfarm as well.
- Be careful with the sequence of modifiers in the modifier stack (as always). You may have a different number of faces in the 3D window and for rendering (e.g. when using subdivision surface), if so, the rendered result may be very different from what you see in the 3D window.

## Hair

When set to hair mode, particle system creates only static particles, which may be used for hair, fur, grass and the like.

## Growing

The first step is to create the hair, specifying the amount of hair strands and their lengths.

The complete path of the particles is calculated in advance. So everything a particle does a hair may do also. A hair is as long as the particle path would be for a particle with a lifetime of 100 frames. Instead of rendering every frame of the particle animation point by point there are calculated control points with an interpolation, the segments.

## Styling

The next step is to style the hair. You can change the look of base hairs by changing the [Physics Settings](#).

A more advanced way of changing the hair appearance is to use [Children](#). This adds child hairs to the original ones, and has settings for giving them different types of shapes.

You can also interactively style hairs in [Particle Mode](#). In this mode, the particle settings become disabled, and you can comb, trim, lengthen, etc. the hair curves.

## Animating

Hair can now be made dynamic using the cloth solver. This is covered in the [Hair Dynamics](#) page.

## Rendering

Blender can render hairs in several different ways. Materials have a Strand section, which is covered in the materials section in the [Strands Page](#).

Hair can also be used as a basis for the [Particle Instance modifier](#), which allows you to have a mesh be deformed along the curves, which is useful for thicker strands, or things like grass, or feathers, which may have a more specific look.

## Options

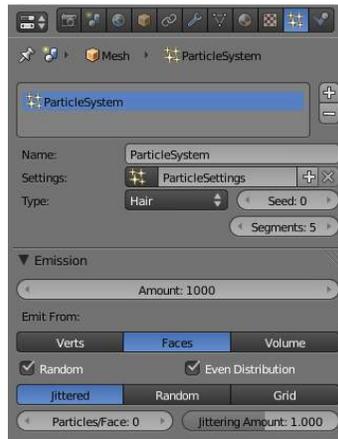


Fig. 2.1504: Image 4a: Settings for a Hair particle system.

**Regrow** Regrow Hair for each frame.

**Advanced** Enables advanced settings which reflect the same ones as working in Emitter mode.

### Emission

**Amount** Set the amount of hair strands. Use as little particles as possible, especially if you plan to use softbody animation later. But you need enough particles to have good control. For a “normal” haircut I found some thousand (very roughly 2000) particles to give enough control. You may need a lot more particles if you plan to cover a body with fur. Volume will be produced later with *Children*.

**Hair Dynamics** Settings for adding movement to hair see [Hair Dynamics](#).

### Display

**Rendered** Draw hair as curves.

**Path** Draw just the end points if the hairs.

**Steps** The number of segments (control points minus 1) of the hair strand. In between the control points the segments are interpolated. The number of control points is important:

- for the softbody animation, because the control points are animated like vertices, so more control points mean longer calculation times.
- for the interactive editing, because you can only move the control points (but you may recalculate the number of control points in *Particle Mode*).

10 Segments should be sufficient even for very long hair, 5 Segments are enough for shorter hair, and 2 or 3 segments should be enough for short fur.

**Children** See [Children](#).

**Render** Hair can be rendered as a Path, Object, or Group. See [Particle Visualization](#) for descriptions.

## Usage



Fig. 2.1505: Image 4b: Particle systems may get hairy...

- [Fur Tutorial](#), which produced (*Image 4b*). It deals especially with short hair.
- [Blender Hair Basics](#), a thorough overview of all of the hair particle settings.

## Hair Dynamics

Hair particles can now be made dynamic using Cloth physics.

To enable hair physics, click the check box beside *Hair Dynamics*.

### Material

**Stiffness** Controls how stiff the root of the hair strands are.

**Mass** Controls the mass of the cloth material.

**Bending** Controls the amount of bend along the hairs. Higher values cause less bending.

**Internal Friction** Amount of friction between individual hairs.

**Collider Friction** Amount of friction between hairs and external collision objects.

### Damping

**Spring** Damping of cloth velocity. (higher = more smooth, less jiggling).

**Air** Air has normally some thickness which slows falling things down.

### Quality

**Steps** Quality of the simulation in steps per frame. (higher is better quality but slower).

### Children

*Children* are *Hair* and *Keyed* particles assigned subparticles. They make it possible to work primarily with a relatively low amount of Parent particles, for whom the physics are calculated. The children are then aligned to their parents. Without recalculating the physics the number and visualization of the children can be changed.

- Children can be emitted from particles or from faces (with some different options). Emission from *Faces* has some advantages, especially the distribution is more even on each face (which makes it better suitable for fur and the like). However, children from particles follow their parents better, e.g. if you have a softbody animation and don't want the hair to penetrate the emitting mesh. But see also our manual page about [Hair](#).
- If you turn on children the parents are no longer rendered (which makes sense because the shape of the children may be quite different from that of their parents). If you want to see the parents additionally turn on the *Parents* button in the *Visualization* panel.
- Children carry the same material as their parents and are colored according to the exact place from where they are emitted (so all children may have different color or other attributes).

The possible options depend from the type of particle system, and if you work with *Children from faces* or *Children from particles*. We don't show every possible combination, only the settings for a *Hair* particle system.

### Settings

**Simple** Children are emitted from the parent hairs.

**Interpolated** Children are emitted between the *Parent* particles on the faces of a mesh. They interpolate between adjacent parents. This is especially useful for fur, because you can achieve an even distribution. Some of the children can become virtual parents, which are influencing other particles nearby.

**Display** The number of children in the 3D window.

**Render** The number of children to be rendered (up to 10.000).

#### For Simple Mode

**Size** Only for *Emitter*. A multiplier for children size.

**Random** Random variation to the size of child particles.

#### Interpolated Mode

**Seed** Offset the random number table for child particles, to get a different result.

**Virtual** Relative amount of virtual parents.

**Long Hair** Calculate children that suit long hair well.

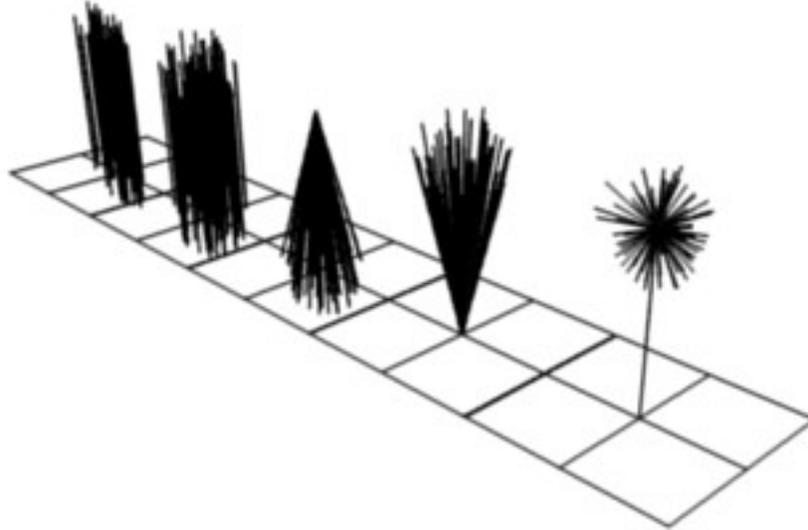


Fig. 2.1506: Image 2: From left to right: Round: 0.0 / Round: 1.0 / Clump: 1.0 / Clump: -1.0 / Shape: -0.99.

### Effects

**Clump** Clumping. The children may meet at their tip (1.0) or start together at their root (-1.0).

**Shape** Form of *Clump*. Either inverse parabolic (0.99) or exponentially (-0.99).

**Length** Length of child paths

**Threshold** Amount of particles left untouched by child path length

**Radius** The radius in which the children are distributed around their parents. This is 3D, so children may be emitted higher or lower than their parents.

**Roundness** The roundness of the children around their parents. Either in a sphere (1.0) or in-plane (0.0).

**Seed** Offset in the random number table for child particles, to get a different randomized result

### Roughness

**Uniform, Size** It is based on children location so it varies the paths in a similar way when the children are near.

**Endpoint, Shape** “Rough End” randomizes path ends (a bit like random negative clumping). Shape may be varied from <1 (parabolic) to 10.0 (hyperbolic).

**Random, Size, Threshold** It is based on a random vector so it’s not the same for nearby children. The threshold can be specified to apply this to only a part of children. This is useful for creating a few stray children that won’t do what others do.

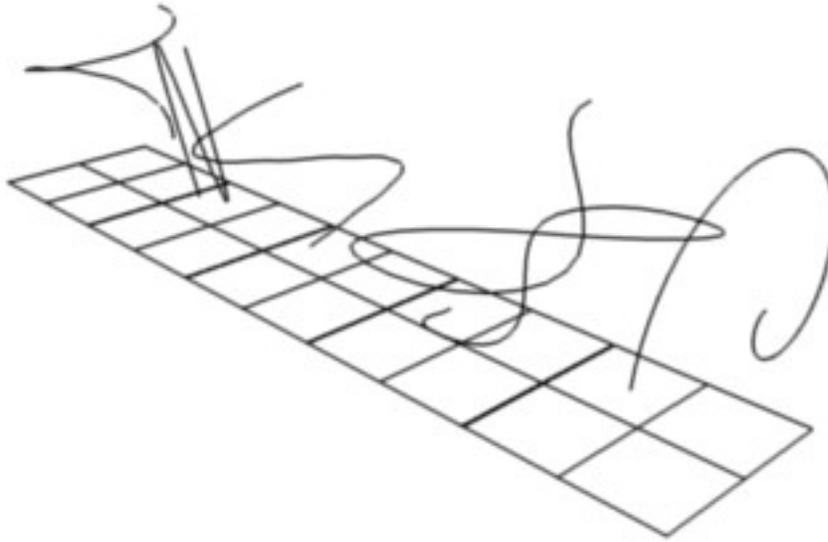


Fig. 2.1507: Image 3: Child particles with Kink. From left to right: Curl / Radial / Wave / Braid / Roll.

**Kink** With *Kink* you can rotate the children around the parent. See above picture (*Image 3*) for the different types of *Kink*.

**Curl** Children grow in a spiral around the parent hairs.

**Radial** Children form around the parent a wave shape that passes through the parent hair.

**Wave** Children form a wave, all in the same direction.

**Braid** Children braid themselves around the parent hair.

**Amplitude** The amplitude of the offset.

**Clump** How much clump effects kink amplitude.

**Flatness** How flat the hairs are.

**Frequency** The frequency of the offset ( $1/\text{total length}$ ). The higher the frequency the more rotations are done.

**Shape** Where the rotation starts (offset of rotation).

### Vertex Groups

The Vertexgroups panel allows you to specify vertex groups to use for several child particle settings. You can also negate the effect of each vertex group with the check boxes. You can affect the following attributes:

- Density
- Length
- Clump
- Kink

- Roughness 1
- Roughness 2
- Roughness End

## Examples

...

## Particle Mode

Using *Particle Mode* you can edit the key-points (key-frames) and paths of *Baked Hair*, *Particle*, *Cloth*, and *Soft Body* simulations. (You can also edit and style hair before baking).

Since working in particle mode is pretty easy and very similar to working with vertices in the 3D window, we will show how to set up a particle system and then give a reference of the various functions.

## Usage

### Ways to use Particle Mode

**Tip:** Only Frames Baked to Memory are Editable!

If you cannot edit the particles, check that you are not baking to a [Disk Cache](#).

### Setup for Hair Particles

- Create a *Hair* particle system - With your object selected, click the *Particle System* icon in the Properties panel. Create a new particle system by clicking the *Plus*.
- Give it an initial velocity in the *Normal* direction (first check the *Advanced* box, then modify the *Velocity* sub-panel), or adjust the *Hair Length*.
- Create a simulation - Place the camera at a good position (*pop-up* → *View* → *Cameras* → *Active Camera ...* or

FIXME(Template Unsupported: Shortcut/Keypress; {{Shortcut/Keypress|pad0}} ). Check the *Hair Dynamics* box. Select *pop-up* → *Render* → *Render OpenGL Animation* in *Render Engine* mode.

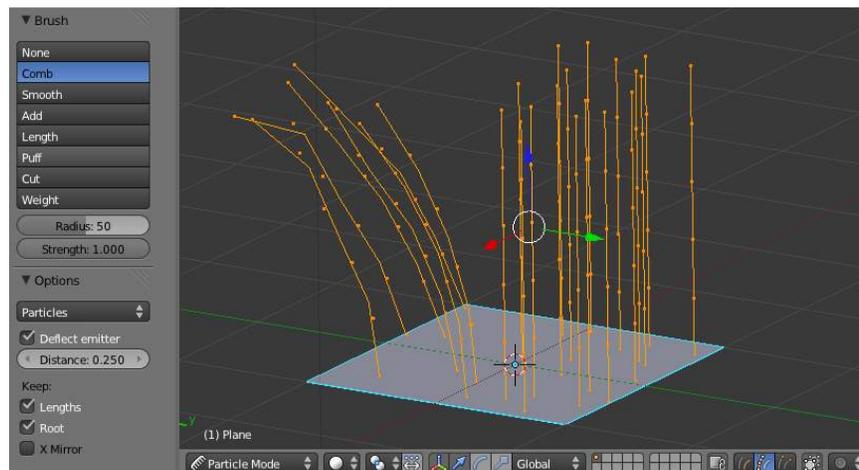


Fig. 2.1508: Editing hair strands in Particle Mode

Fig. 2.1509: Editing a baked particle simulation's particle paths in Particle Mode

### Setup for Particle, Cloth, and Soft Body Simulations

- Use *Emitter* particles, or a cloth/soft-body simulation
- Create a simulation - set up objects and or emitters, set your time range (use a small range if you are just starting out and experimenting), set up the simulation how you want it, using `Alt-A` to preview it.

#### Bake the Simulation

- Once you are happy with the general simulation, **bake** the simulation from object mode. The simulation must be baked to enable editing. (remember to bake to memory, a disk cache will not be editable in *Particle Mode*)

### Edit the Simulation

- Switch to *Particle Edit* from the *Mode dropdown menu* in the bottom menu bar of the *3D View* to edit the particle's paths/key-frames. You may need to press `T` from within the 3D viewport to see the *Particle Edit* panel. Move to the frame you want to edit and use the various *Particle Edit* tools to edit your simulation. Work slowly, previewing your changes with `Alt-A`, and save often so that you can go back to the previous version should something happen, or that you do not like the latest changes you have made.

To be able to clearly see what you are working on:

- Turn on the *Particle Edit Properties (PEP)* panel with `N`.
- Select *Point select mode*



Fig. 2.1510: in the header of the 3D window. This will display key points along the particle path.

---

**Tip:** Brush Size

Press `F` to resize the brush while working

---

### Using Particle Mode

#### Selecting Points

- Single: `RMB`.
- All: `A`.
- Linked: Move the mouse over a keypoint and press `L`.
- Border select: `B`.
- First/last: `W` -> *Select First / Select Last*.

You may also use the *Select Menu*.

---

**Tip:** Selections

Selections are extremely useful for modifying only the particles that you want. Hover over a particle path and press `L` to link-select it, hover over the next and press `L` to add that path to the selection. To remove a path, hold shift and press `L`. To Deselect all press `A`.

The method to select individual points is the same as in edit mode. click to select, shift+click to add/remove a point from the selection

---

---

**Tip:** Beware of Undo!

Using *Undo* in *Particle Mode* can have strange results. Remember to save often!

---

### Moving keypoints or particles

- To move selected keypoints press **G**, or use one of the various other methods to grab vertices.
- To move a particle root you have to turn off *Keep Root* in the *Tool Bar*.
- You can do many of the things like with vertices, including scaling, rotating and removing (complete particles or single keys).
- You may not duplicate or extrude keys or particles, but you can subdivide particles which adds new keypoints (**W** → *Subdivide* / **Numpad2**).
- Alternatively you can rekey a particle (**W** → *Rekey* / **Numpad1**) and choose the number of keys.

How smoothly the hair and particle paths are displayed depends on the *Path Steps* setting in the *Tool Bar*. Low settings produce blocky interpolation between points, while high settings produce a smooth curve.

### Mirroring particles

- If you want to create an X-Axis symmetrical haircut you have to do following steps: - Select all particles with **A**. - Mirror the particles with **Ctrl-M**, or use the *Particle* → *Mirror* menu. - Turn on *X-Axis Mirror Editing* in the *Particle* menu.

It may happen that after mirroring two particles occupy nearly the same place. Since this would be a waste of memory and render time, you can *Remove doubles* either from the *Specials* (**W**) or the *Particle* menu.

**Hiding/Unhiding** Hiding and unhiding of particles works similar as with vertices in the 3D window. Select one or more keypoints of the particle you want to hide and press **H**. The particle in fact doesn't vanish, only the key points.

Hidden particles (i.e. particles whose keypoints are hidden) don't react on the various brushes. But:

If you use *Mirror Editing* even particles with hidden keypoints may be moved, if their mirrored counterpart is moved.

To un-hide all hidden particles press **Alt+H**.



### Select Modes

**Path** No keypoints are visible, you can select/deselect only all particles.

**Point** You see all of the keypoints.

**Tip** You can see and edit (including the brushes) only the tip of the particles, i.e. the last keypoint.

**Brush** With the buttons you can select the type of “Comb” utility you want to use. Below the brush types, their settings appear:

#### Common Options:

**Radius** Set the radius of the brush.

**Strength** Set the strength of the brush effect (not for Add brush).

**Add/Sub Grow/Shrink** Sets the brush to add the effect or reverse it..

**None** No special tool, just edit the keypoints as “normal” vertices.

**Comb** Moves the keypoints (similar to “proportional editing”).

**Smooth** Parallels visually adjacent segments.

**Add** Adds new particles.

**Count** The number of new particles per step.

**Interpolate** Interpolate the shape of new hairs from existing ones.

**Steps** Amount of brush steps

**Keys** How many keys to make new particles with.

**Length** Scales the segments, so it makes the hair longer(*Grow*) or shorter(*Shrink*).

**Puff** Rotates the hair around it’s first keypoint (root). So it makes the hair stand up (*Add*) or lay down (*Sub*).

**Puff Volume** Apply puff to unselected end-points, (helps maintain hair volume when puffing root)

**Cut** Scales the segments until the last keypoint reaches the brush.

**Weight** This is especially useful for softbody animations, because the weight defines the softbody *Goal*. A keypoint with a weight of 1 won’t move at all, a keypoint with a weight of 0 subjects fully to softbody animation. This value is scaled by the *GMin* - *GMax* range of softbody goals...

### Options

**Deflect Emitter, Dist** Don’t move keypoints through the emitting mesh. *Dist* is the distance to keep from the Emitter.

### Keep

**Length** Keep the length of the segments between the keypoints when combing or smoothing the hair. This is done by moving all the other keypoints.

**Root** Keep first key unmodified, so you can’t transplant hair.

**X Mirror** Enable mirror editing across the local x axis.

### Draw

**Path Steps** Drawing steps, sets the smoothness of the drawn path.

**Show Children** Draws the children of the particles too. This allows to fine tune the particles and see their effects on the result, but it may slow down your system if you have many children.

## 2.7.7 Smoke Simulation

### Introduction

#### Development notes

Blender’s new smoke simulation is based on the paper [Wavelet Turbulence for Fluid Simulation](#) and associated sample code.

It has been implemented in Blender by Daniel Genrich and Miika Hamalainen.

## Inner working

The simulator uses a volumetric fluid-based model, with the end results output as voxel grids. This voxel data is visualized interactively in Blender's 3D view using custom OpenGL shading, and can be rendered using the Voxel Data texture. Blender's **smoke simulation** wraps Voxels around existing **Particles**. It requires a particle-emitting object and a 'domain' object within which smoke is rendered.

---

**Note:** This Part of the Documentation uses the 2.58 Release

---

## User workflow

The smoke simulation is similar to the Fluid simulation: a Domain and Flow object is required to do a smoke simulation:

- set as the simulation **Domain** an object that defines the bounds of the simulation volume,
- set as the **Flow object** an object which determines where the smoke will be produced from,
- set **Collision objects**, to make the smoke interact with objects in the scene.
- assign a **Material** to the smoke
- save the project
- **bake** the simulation

In case you are having troubles, please consult the [Appendix](#)

## Smoke Domain

Like the Fluid Sim, most of the settings are found when the Domain Object is selected.

### Creating the Domain

Before you can add smoke to your scene you need to define the area where the smoke simulation takes place. In Blender physics this is called a domain. A good idea is to choose a cube for that because you can scale it to the view of your camera later on. In our case we just make the default cube bigger by pressing **S** and dragging the mouse.

---

**Note:** Don't edit the domain's vertices!

If you want a bigger domain, scale the object. Changing it in edit mode will lead to your smoke appearing more than once during rendering, like a repeating texture.

---

Make sure you're in object mode and go to the physics tab. Add smoke and chose the radio button labeled 'Domain'. For now that's all, we will return to the new settings that popped up later on.



### Generic options

**Resolution** How detailed the smoke is. A resolution of 32 will bake in a few seconds, while a resolution of 100 can take up to a half hour on most PC's.

**Time Scale** Affects how fast the simulation plays.

### Border Collisions

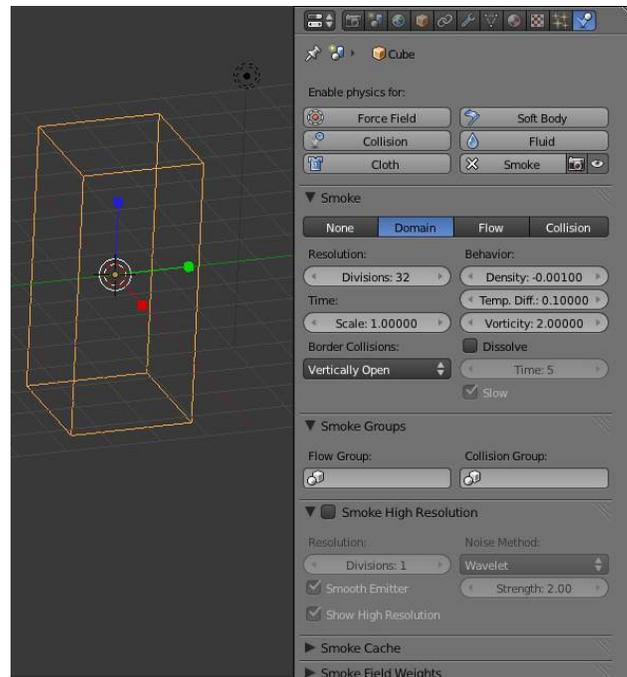


Fig. 2.1517: The Smoke Domain Object

**Vertically Open** Smoke disappears when it collides with the top and bottom of the domain.

**Open** Smoke disappears when it crosses the boundaries of the domain object.

**Collide All** Domain Boundaries are treated as collision objects, the smoke will collide and stay inside.

**Temperature and Density** How much Density and Temperature affect smoke motion. Higher Values make faster-rising smoke.

**Vorticity** Affects how turbulence/rotation, or swirly the smoke is.

**Dissolve** Allow the smoke to dissipate over time.

**Time** The speed of the smoke's dissipation.

**Slow** Use  $1/\text{Time}$  instead of Time, making the smoke dissolve slower.

### Smoke Groups options

#### Smoke High Resolution options

The High Resolution option lets you simulate at low resolution and then uses noise techniques to enhance the resolution without actually computing it. This allows animators to set up a low resolution simulation quickly and later add details without changing the overall fluid motion.

Various methods for this are available, including the default: Wavelet, which is an implementation of [Turbulence for Fluid Simulation](#).

**Resolution/Divisions** Enhance the resolution of smoke by this factor using noise.

**Smooth Emitter** Smoothens emitted smoke to avoid blockiness.

**Show High Resolution** Show high resolution using amplification.

**Noise Method** *Wavelet*

*FFT*

**Strength** Strength of noise.

### Smoke Field Weights options

Determines how much various forces and force fields affect the smoke.

**Gravity** How much the smoke is affected by Gravity.

**All** Changes the overall influence of all force fields.

The other settings determine how much various Force Fields affect the smoke.

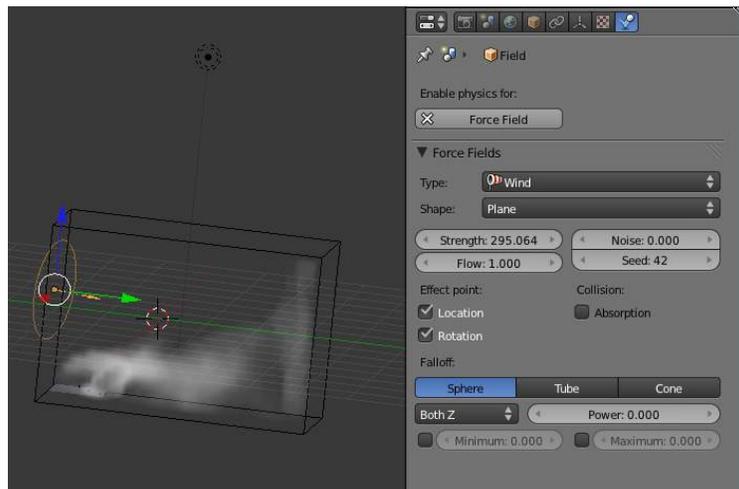


Fig. 2.1518: Smoke with a wind force field.

## Smoke Flow object

### Create a Flow Object

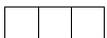
Once you have defined the volume that will contain smoke, we'll add an object from which the smoke will be emitted. Add another cube and make sure it's inside the domain cube *Shift-Apop-up* → *Mesh* → *Cube*; 3D view must be selected).

While in edit mode go to physics and add smoke to the small cube, too. This time chose Flow.

The smoke will not be emitted from the object itself but from particles the object emits. So we need to set up a particle system. With the small cube still selected go to the particle tab. Add a new particle system and turn off the physics because we want our smoke emit from stationary place. We also don't want to see the particles so turn off the render, too.



Now go back to the physics tab and chose the particle system in the smoke section. There should be a list with just one system to chose from that is called 'ParticleSystem' since we did not change the name. Now you can scrub through the timeline to see smoke coming from the cube. Another way to preview the smoke is starting the animation by *Alt-A* (stop it the same way).



## Settings

**Outflow** Delete smoke from simulation.

**Particle System** Particle system emitted from the object.

**Initial Velocity** Smoke inherits its velocity from the emitter particle.

**Multiplier** Multiplier to adjust velocity passed to smoke.

### Initial Values

**Absolute Density** Only allow given density value in emitter area.

**Density** Initial density value.

**Temp. Diff.** Temperature to ambient temperar.

## Collisions

Smoke can collide with mesh objects, using the 'Collision' option in smoke. Currently only static collision objects are supported.

## Forces

Blender's force fields (such as wind or vortex fields) are also supported, modifying the smoke simulation as they do for other physics systems such as particles.

## Smoke Material

### Create the Material

Simulating the smoke is easy, however rendering it is not.

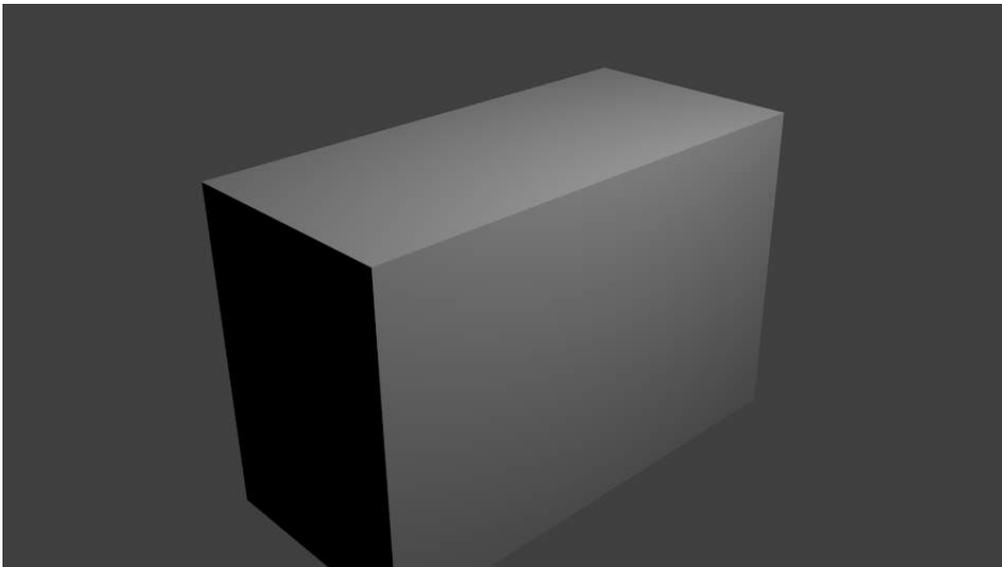


Fig. 2.1531: The Render without the correct smoke material.

Rendering at this point will result in just the big cube (Image, F12) or in a crash (Animation, Ctrl-F12).

The material must be a volumetric material with a Density of 0, and a high Density Scale.

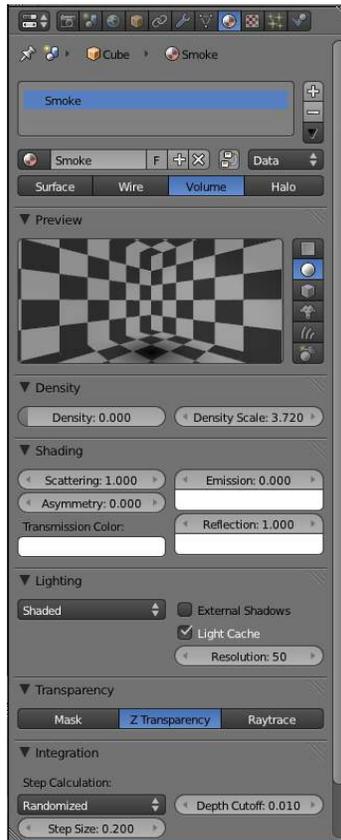


Fig. 2.1532: The material settings.

The first issue can easily be fixed by working on the material and texture of the domain cube. Smoke requires a complex material to render correctly. Select the big cube and go to the material tab. There change the material to 'Volume' and set the density to 0. If you set the density to values bigger than 0 the domain cube will be filled with the volume material. The other settings will affect the smoke, though. We'll cover those later.



### Add the Texture

In addition, Smoke requires its own texture, you can use a volumetric texture known as *Voxel Data*. You must remember to set the domain object and change the influence.

Go to the texture tab and change the type to *Voxel Data*. Under the Voxel Data-Settings set the domain object to our domain cube (it should be listed just as 'Cube' since we are using Blender's default cube. Under Influence check 'Density' and leave it at 1.000 (Emission should be automatically checked, too). Now you should be able to render single frames. You can choose to color your smoke as well, by turning *Emission Color* back on.

**Tip:** To see the smoke more clearly

Under the world tab, chose a very dark color for the horizon.

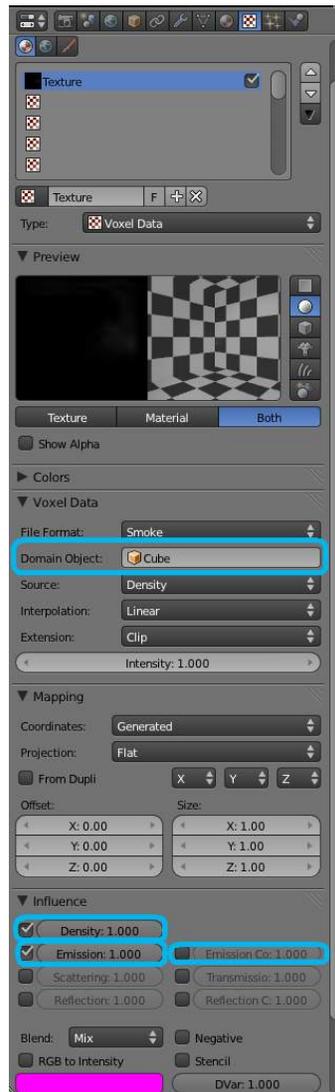


Fig. 2.1539: The texture settings.

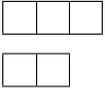


Fig. 2.1550: The rendered smoke. It's hard to see, but it's there.

### Extending the Smoke Simulator: Fire!

You can also turn your smoke into fire with another texture! To make fire, turn up the Emission Value in the Materials panel.

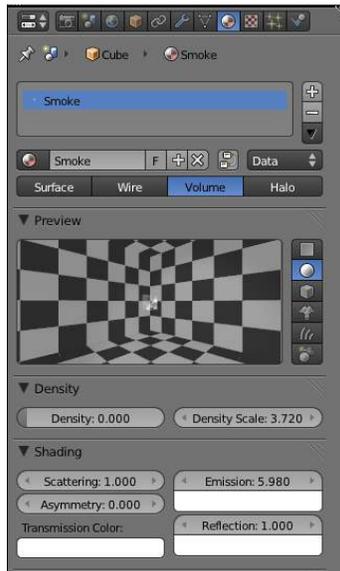


Fig. 2.1551: The Fire material.

Then, add another texture (Keep the old texture or the smoke won't show). Give it a fiery color ramp- which colors based on the alpha, and change the influence to emission and emission color. Change the blend to Multiply.

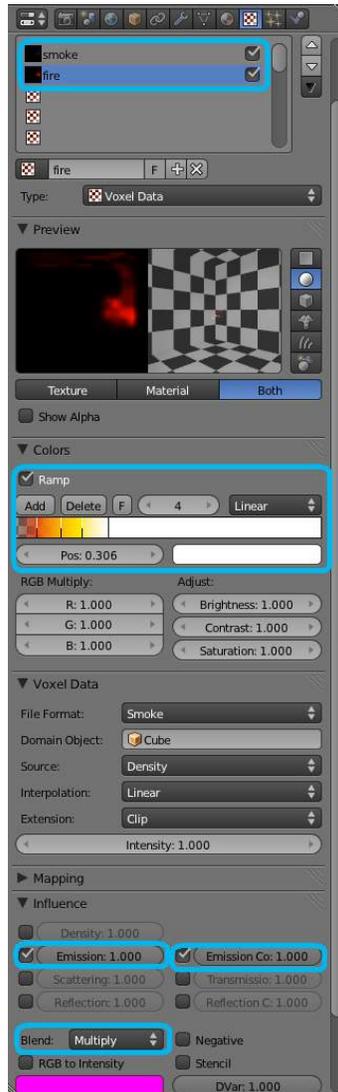


Fig. 2.1552: The fire texture settings.

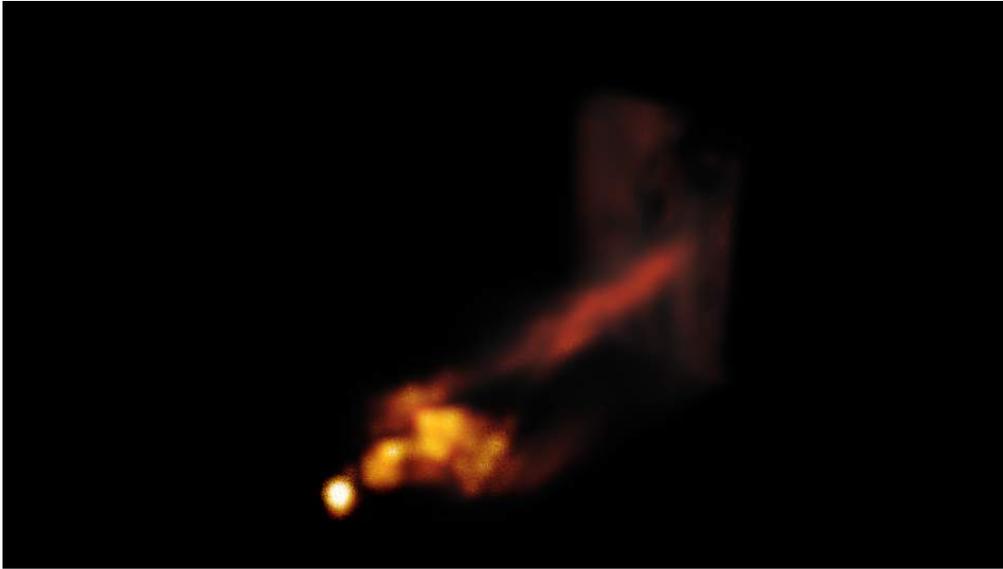


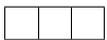
Fig. 2.1553: The fire render.

## Baking Smoke Simulations

If you want to render an animation, you need to bake your smoke first. Baking is simply calculating a simulation. To bake the smoke, the file must be saved. The calculations are stored in a cache file which can be named. On occasion, Baking may crash Blender. [See troubleshooting]

By scrubbing through the timeline or running the animation in the viewport via `Alt-A` you already did some realtime-baking to the memory. But for rendering the animation, the baked data must be on disk. And before you can bake, you need to save your Blendfile.

Next select the domain cube and go to the physics tab where you open the Smoke Cache section. Give your cache a file name by entering it into the text box and pressing `Return`. By pressing `Bake` your simulation data is computed and stored to disk. Notice that the scrubbing-bug in the timeline is gone now? At this point you should be able to render the animation.



## Smoke Appendix

### Troubleshooting

- **Q.** The smoke cache is greyed out!
- **A.** Save your .blend file.
- **Q.** Blender Crashes when I push the Domain Button of a smoke simulation!
- **A.** This is caused by outdated drivers. Update your Drivers.
- **Q.** Blender Crashes when Smoke Simulations bake!
- **A.** You ran out of RAM to compute it. Try Baking at a lower resolution.
- **Q.** The smoke isn't rendering!
- **A.** Go back and read the documentation.
- **Q.** When I try to make fire, it gives me strange results, or doesn't show up.

- **A.** Make sure you have a high emission Value for the Material, and that you have a Smoke Density texture, and that you set the fire texture to Multiply.

#### External links

- [In-Depth introduction to smoke and fire in Blender 2.5 covering most of the pitfalls](#)
- [Guide to realistic fire in Blender by MiikaH](#)

## 2.7.8 Soft Body

### Introduction



Fig. 2.1560: Image 1a: A softbody cloth uncovering a text. [Animation - Blend file](#)

A Soft Body in general, is a simulation of a soft or rigid deformable object. In Blender, this system is best for simple cloth objects and closed meshes. There is dedicated [Cloth Simulation](#) physics that use a different solver, and is better for cloth.

This simulation is done by applying forces to the vertices or controlpoints of the object. There are exterior forces like gravity or forcefields and interior forces that hold the vertices together. This way you can simulate the shapes that an object would take on in reality if it had volume, was filled with something, and was acted on by real forces.

Soft Bodies can interact with other objects (*Collision*). They can interact with themselves (*Self Collision*).

The result of the Soft Body simulation can be converted to a static object. You can also *bake edit* the simulation, i.e. edit intermediate results and run the simulation from there.

#### Typical scenarios for using Soft Bodies

Soft Bodies are well suited for:



Fig. 2.1561: Image 1b: A wind cone. The cone is a Soft Body, as the suspension. [Animation - Blend file](#)

- Elastic objects with or without collision.
- Flags, fabric reacting to forces.
- Certain modeling tasks, like a cushion or a table cloth over an object.
- Blender has another simulation system for clothing (see [Clothes](#)). But you can sometimes use Soft Bodies for certain parts of clothing, like wide sleeves.
- Hair (as long as you minimize collision).
- Animation of swinging ropes, chains and the like.

The following videos may give you some more ideas: <http://youtube.com/watch?v=qdusMZIBbQ4>, <http://de.youtube.com/watch?v=3du8ksOm9Fo>

### Creating a Soft Body

Soft Body simulation works for all objects that have vertices or control points:

- Meshes.
- Curves.
- Surfaces.
- Lattices.

To activate the Soft Body simulation for an object:

- In the *Properties* window, go to the *Physics* tab (it is all the way on the right, and looks like a bouncing ball).
- Activate the *Soft Body* button.

A lot of options appear. For a reference of all the settings see [this page](#).

- You start a Soft Body simulation with `Alt-A`.
- You pause the simulation with `Spacebar`, continue with `Alt-A`.
- You stop the simulation with `Esc`.

### Simulation Quality

The settings in the *Soft Body Solver* panel determine the accuracy of the simulation.

**Min Step** Minimum simulation steps per frame. Increase this value, if the Soft Body misses fast moving collision objects.

**Max Step** Maximum simulation steps per frame. Normally the number of simulation steps is set dynamically (with the *Error Limit*) but you have probably a good reason to change it.

**Auto-Step** Use Velocities for automatic step sizes.

**Error Limit** Rules the overall quality of the solution delivered. Default 0.1. The most critical setting that says how precise the solver should check for collisions. Start with a value that is 1/2 the average edge length. If there are visible errors, jitter, or over-exaggerated responses, decrease the value. The solver keeps track of how “bad” it is doing and the *Error Limit* causes the solver to do some “adaptive step sizing”.

**Fuzzy** Simulation is faster, but less accurate.

**Choke** Calms down (reduces the exit velocity of) a vertex or edge once it penetrates a collision mesh.

### Diagnostics

**Print Performance to Console** Prints on the console how the solver is doing.

**Estimate Matrix** Estimate matrix. Split to COM , ROT ,SCALE

## Cache and Bake

Soft Bodies and other physic simulations use a unified system for caching and baking. See [Particle Cache](#) for reference.

The results of the simulation are automatically cached to disk when the animation is played, so that the next time it runs, it can play again quickly by reading in the results from the disk. If you *Bake* the simulation the cache is protected and you will be asked when you're trying to change a setting that will make a recalculating necessary.

---

**Tip:** Beware of the *Start* and *End* settings

The simulation is only calculated for the frames in-between the *Start* and *End* frames (*Bake* panel), even if you don't actually bake the simulation! So if you want a simulation longer than the default setting of 250 frames you have to change the *End* frame.

---

- Caching:
  - As animation is played, each physics system writes each frame to disk, between the simulation start and end frames. These files are stored in folders with prefix `blendcache`, next to the `.blend` file.
  - The cache is cleared automatically on changes - but not on all changes, so it may be necessary to free it manually, e.g. if you change a force field. Note that for the cache to fill up, one has to start playback before or on the frame that the simulation starts.
  - If you are not allowed to write to the required sub-directory caching will not take place.
  - The cache can be freed per physics system with a button in the panels, or with the `Ctrl-B` shortcut key to free it for all selected objects.
  - You may run into trouble if your `.blend` file path is very long and your operating system has a limit on the path length that is supported.
- Baking:
  - The system is protected against changes after baking.
  - The *Bake* result is cleared also with `Ctrl-B` for all selected objects or click on *Free Bake* for the current Soft Body system.
  - If the mesh changes the simulation is not calculated anew.

For renderfarms, it is best to bake all the physics systems, and then copy the `blendcache` to the renderfarm as well.

## Interaction in real time

To work with a Soft Body simulation you will find it handy to use the *Timeline* window. You can change between frames and the simulation will always be shown in the actual state. The option *Continue Physics* in the *Playback* menu of the *Timeline* window lets you interact in real time with the simulation, e.g. by moving collision objects or shake a Soft Body object. And this is real fun!

---

**Tip:** *Continue Physics* does not work while playing the animation with `Alt-A`

Right. This works only if you start the animation with the *Play* button of the *Timeline* window.

---

You can then select the Soft Body object while running the simulation and *Apply* the modifier in the *Modifiers* panel of the *Editing* context. This makes the deformation permanent.

## Tips

- Soft Bodies work especially well if the objects have an even vertex distribution. You need enough vertices for good collisions. You change the deformation (the stiffness) if you add more vertices in a certain region (see the animation of *Image 1b*).
- The calculation of collisions may take a long time. If something is not visible, why calculate it?
- To speed up the collision calculation it is often useful to collide with an additional, simpler, invisible, somewhat larger object (see the example to *Image 1a*).
- Use Soft Bodies only where it makes sense. If you try to cover a body mesh with a tight piece of cloth and animate solely with Soft Body, you will have no success. Self collision of Soft Body hair may be activated, but that is a path that you have to wander alone. We will deal with [Collisions](#) in detail later.
- Try and use a *Lattice* or a *Curve Guide* Soft Body instead of the object itself. This may be magnitudes faster.

## Links

- [Developer Notes](#)
- [Swinging of a chain](#)
- [Softbodies for Rigged Characters](#)

## Exterior Forces

Exterior forces are applied to the vertices (and nearly exclusively to the vertices) of Soft Body objects. This is done using Newtons Laws of Physics:

- If there is no force on a vertex, it stays either unmoved or moves with constant speed in a straight line.
- The acceleration of a vertex depends on its mass and the force. The heavier the mass of a vertex the slower the acceleration. The larger the force the greater the acceleration.
- For every action there is an equal and opposite reaction.

Well, this is done only in the range of computing accurateness, there is always a little damping to avoid overshoot of the calculation.

## Example

We will begin with a very simple example - the default cube.

- To judge the effect of the external forces you should at first turn off the *Goal*, so that the vertices are not retracted to their original position.
- Press `Alt-A`.

What happens? The cube moves in negative Z-direction. Each of it's eight vertices is affected by a global, constant force - the gravitation. Gravitation without friction is independent from the weight of an object, so each object you would use as a Soft Body here would fall with the same acceleration. The object does not deform, because every vertex moves with the same speed in the same direction.

## Settings

## Soft Body Panel

**Friction** The friction of the surrounding medium. The larger the friction, the more viscous is the medium. Friction always appears when a vertex moves relative to its surround medium.

**Mass** Mass value for vertices. Larger mass slows down acceleration, except for gravity where the motion is constant regardless of mass. Larger mass means larger inertia, so also braking a Soft Body is more difficult.

**Mass Vertex Group** You can paint weight values for an mesh's mass, and select that vertex group here.

**Speed** You can control the internal timing of the Softbody system with this value. It sets the correlation between frame rate and tempo of the simulation. A free falling body should cover a distance of about five meters after one second. You can adjust the scale of your scene and your simulation with this correlation. If you render with 25 frames per second and 1 meter shall be 1 BU, you have to set *Speed* to 1.3.

## Force Fields

To create other forces you have to use another object, often *Empty* objects are used for that. You can use some of the forces on Soft Body vertices as on *Particles*. Soft Bodies react only to:

- Spherical
- Wind
- Vortex

Soft bodies do react on *Harmonic* fields, but not in a useful way. So if you use a *Harmonic* field for particles move the Soft body to another layer.

See the section [Force Fields](#) for details. The force fields are quite strong, a *Spherical* field with a strength of -1.0 has the same effect that gravity has - approximately - a force of 10 Newton.

## Aerodynamics

This special exterior force is not applied to the vertices but to the connecting edges. Technically, a force perpendicular to the edge is applied. The force scales with the projection of the relative speed on the edge (dot product). Note that the force is the same if wind is blowing or if you drag the edge through the air with the same speed. That means that an edge moving in its own direction feels no force, and an edge moving perpendicular to its own direction feels maximum force.

**Simple** Edges receive a drag force from surrounding media

**Lift Force** Edges receive a lift force when passing through surrounding media.

**Factor** How much aerodynamic force to use. Try a value of 30 at first.

## Using a Goal

A goal is a shape that a soft body object tries to conform to.

You have to confine the movement of vertices in certain parts of the mesh, e.g. to attach a Soft Body object at other objects. This is done with the *Vertex Group* (target). The target position is the original position of the vertex, like it would result from the "normal" animation of an object including *Shape Keys*, *Hooks* and *Armatures*. The vertex tries to reach its target position with a certain, adjustable intensity.

Fig. 2.1562: Image 2b: Shock absorber description.

Imagine the vertex is connected with its target through a spring (*Image 2b*).

**Default** This parameter defines how strong the influence of this spring is. A strength of 1 means, that the vertex will not move as Soft Body at all, instead keep its original position. 0 *Goal* (or no *Goal*) means, that the vertex moves only according to Soft Body simulation. If no vertex group is used/assigned, this numeric field is the default goal weight for all vertices. If a vertex group is present and assigned, this button instead shows an pop-up selector button that allows you to choose the name of the goal vertex group. If you use a vertex group the weight of a vertex defines its goal.

Often [weight painting](#) is used to adjust the weight comfortably. For non-mesh objects the *Weight* parameter of their vertices/controlpoints is used instead (*w* in *Edit mode*, or use the *Transform Properties* panel). The weight of *Hair* particles can also be painted in [Particle Mode](#).

**Minimum / Maximum** When you paint the values in vertex-groups (using *WeightPaint* mode), you can use the *G Min* and *G Max* to fine-tune (clamp) the weight values. The lowest vertex-weight (blue) will become *G Min*, the highest value (red) becomes *G Max* (please note that the blue-red color scale may be altered by User Preferences).

**Tip:** For now all is applied to single vertices

For now we have discussed vertex movement independent of each other, similar to particles. Every object without *Goal* would collapse completely if a non uniform force is applied. Now we will move to the next step, the forces that keep the structure of the object and make the Soft Body to a real Body.

**Stiffness** The spring stiffness for Goal. A low value creates very weak springs (more flexible “attachment” to the goal), a high value creates a strong spring (a stiffer “attachment” to the goal).

**Damping** The friction of the spring. With a high value the movement will soon come to an end (little jiggle).

### Interior Forces

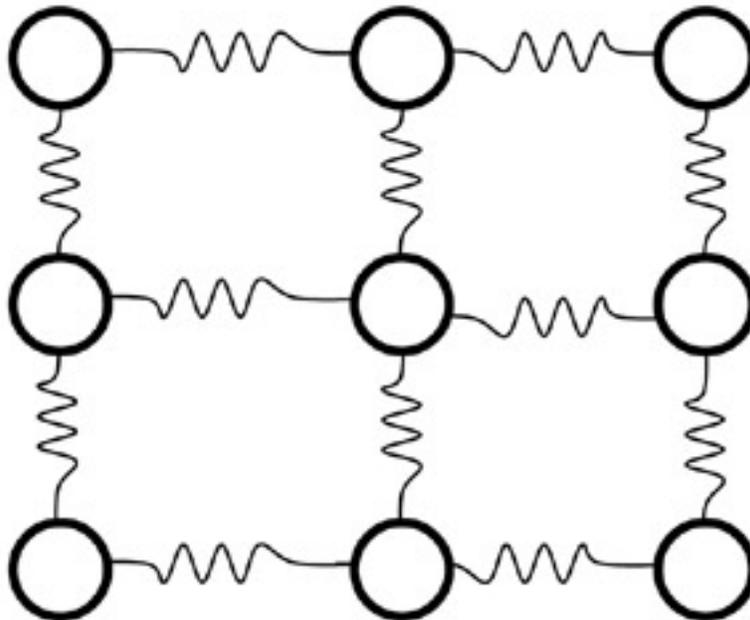


Fig. 2.1563: Image 1a: Vertices and forces along their connection edges.

To create a connection between the vertices of a Soft Body object there have to be forces that hold the vertices together. These forces are effective along the edges in a mesh, the connections between the vertices. The forces act like a spring. (*Image 1a*) illustrates how a 3x3 grid of vertices (a mesh plane in Blender) are connected in a Soft Body simulation.

But two vertices could freely rotate if you don't create additional edges between them. Have you ever tried building a storage shelf out of 4 planks alone? Well - don't do it, it will not be stable. The logical method to keep a body from collapsing would be to create additional edges between the vertices. This works pretty well, but would change your mesh topology drastically.

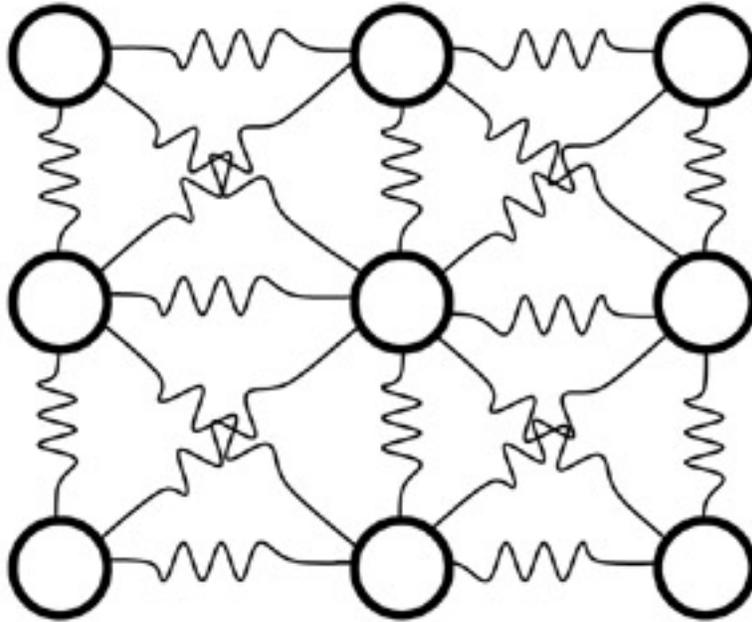


Fig. 2.1564: Image 1b: Additional forces with Stiff Quads enabled.

Luckily, Blender allows us to define additional *virtual* connections. On one hand we can define virtual connections between the diagonal edges of a quad face (*Stiff Quads*, *Image 1b*), on the other hand we can define virtual connections between a vertex and any vertices connected to its neighbours (*Bending Stiffness*). In other words, the amount of bend that is allowed between a vertex and any other vertex that is separated by two edge connections.

### Edges Settings

The characteristics of edges are set with the *Soft Body Edge* properties.

**Use Edges** Allow the edges in a Mesh Object to act like springs.

**Pull** The spring stiffness for edges (how much the edges are allowed to stretch). A low value means very weak springs (a very elastic material), a high value is a strong spring (a stiffer material) that resists being pulled apart. 0.5 is latex, 0.9 is like a sweater, 0.999 is a highly-starched napkin or leather. The Soft Body simulation tends to get unstable if you use a value of 0.999, so you should lower this value a bit if that happens.

**Push** How much the Softbody resist being scrunched together, like a compression spring. Low values for fabric, high values for inflated objects and stiff material.

**Damp** The friction for edge springs. High values (max of 50) dampen the *Push / Pull* effect and calm down the cloth.

**Plastic** Permanent deformation of the object after a collision. The vertices take a new position without applying the modifier.

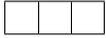
**Bending** This option creates virtual connections between a vertex and the vertices connected to its neighbors. This includes diagonal edges. Damping also applies to these connections.

**Length** The edges can shrink or been blown up. This value is given in percent, 0 disables this function. 100% means no change, the body keeps 100% of his size.

**Stiff Quads** For quad faces, the diagonal edges are used as springs. This stops quad faces to collapse completely on collisions (what they would do otherwise).

**Shear** Stiffness of the virtual springs created for quad faces.

**Preventing Collapse** To show the effect of the different edge settings we will use two cubes (blue: only quads, red: only tris) and let them fall without any goal onto a plane (how to set up collision is shown on the page [Collisions](#)).



In (*Image 3*), the default settings are used (without *Stiff Quads*). The “quad only” cube will collapse completely, the cube composed of tris keeps it’s shape, though it will deform temporarily because of the forces created during collision.



In (*Image 4*), *Stiff Quads* is activated (for both cubes). Both cubes keep their shape, there is no difference for the red cube, because it has no quads anyway.



The second method to stop an object from collapsing is to change it’s *Bending Stiffness*. This includes the diagonal edges (Damping also applies to these connections).

In (*Image 5*), *Be* is activated with a strength setting of 1. Now both cubes are more rigid.



Bending stiffness can also be used if you want to make a subdivided plane more plank like. Without *Be* the faces can freely rotate against each other like hinges (*Image 6b*). There would be no change in the simulation if you activated *Stiff Quads*, because the faces are not deformed at all in this example.

Bending stiffness on the other hand prevents the plane from being - well - bent.

## Collisions

There are two different collision types that you may use: collision between different objects and internal collision. We should set one thing straight from the start: the primary targets of the collision calculation are the vertices of a Soft Body. So if you have too few vertices too few collision takes place. Secondly, you can use edges and faces to improve the collision calculation.

### Collisions with other objects

For a *Soft Body* to collide with another object there are a few prerequisites:

- Both objects have to share a layer, but the layer does not necessarily have to be visible.
- The collision object has to be a mesh object.
- You have to activate the option *Collision* in the *Collision* panel of the *Physics* sub-context (*Image 1*) for the collision object. The collision object may also be a Soft Body.
- If you use modifiers such as *Array* and *Mirror* you have to activate *EV.M.Stack* to ensure that collision calculation is based on the modified object. The sequence of *Modifiers* is not important.

## Examples □□□

A cube colliding with a plane works pretty well (*Image 2a*), but a plane falls right through a cube that it is supposed to collide with (*Image 2b*). Why is that? Because the default method of calculation only checks to see if the four vertices of the plane collides with the cube as the plane is pulled down by gravity. You can activate *CFace* to enable collision between the face of the plane and the object instead (*Image 2c*), but this type of calculation takes much longer.

Let's have a closer look at the collision calculation, so you can get an idea of how we might optimize it.

## Calculating Collisions □□

Soft Body simulation is by default done on a per vertex basis. If the vertices of the Soft Body do not collide with the collision object there will be no interaction between the two objects.

In (*Image 3a*), you can see a vertex colliding with a plane. If a vertex penetrates the zone between *Outer* and *Inner*, it is repulsed by a force in the direction of the face normal. The position that a vertex finally ends up in is dependent on the forces that act upon it. In the example gravity and the repulsion force of the face balance out. The speed at which the vertex is pulled out of the collision zone is influenced by the *Choke* parameter (*Image 4*).

Now lets see what happens if we make vertices heavier and let them travel at a faster speed. In (*Image 3b*), you can see vertices traveling at different speeds. The two on the far right (5 and 6) are traveling so fast that they pass right through the collision zone (this is because of the default solver precision - which we can fix later). You will notice that the fourth vertex also travels quite fast and because it is heavier it breaches the inner zone. The first three vertices collide OK.



Fig. 2.1599: Image 3d: Also Edges and Faces can be used for the collision calculation.

You can set up your collision so that edges and even faces are included in the collision calculation (*Image 3d*). The collision is then calculated differently. It is checked whether the edge or face intersects with the collision object, the collision zones are not used.

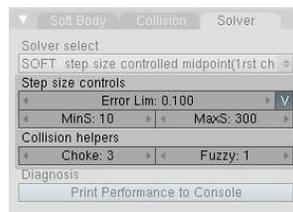


Fig. 2.1600: Image 4: Parameters for Soft Body calculation.

**Good collisions** If the collision you have set up is not behaving properly, you can try the following:

---

**Tip:** The best way

Add *Loop Cuts* to your Soft Body object in strategic areas that you know are most likely to be involved in a collision.

---

- The Soft Body object must have more subdivisions than the collision object.

- Check the direction of the face normals.
- If the collision object has sharp spikes they might penetrate the Soft Body.
- The resolution of the solver must match the speed at which Soft Body vertices are traveling. Lower the parameter *Error Lim* and carefully increase *Min S*.
- *Outer* and *Inner* should be large enough, but zones of opposite faces should not overlap, or you have forces in opposite directions.
- If you use strong forces you should use large zones.
- Set *Choke* to a high enough value (all the way up if necessary) if you have difficulties with repelled vertices.
- Colliding faces are difficult to control and need long calculation times. Try not to use them.

Often it is better to create a simplified mesh to use as your collision object, however this may be difficult if you are using an animated mesh.

### Self Collision

*Self Collision* is working only if you have activated *Use Edges*.

When enabled, allows you to control how Blender will prevent the Soft Body from intersecting with itself. Every vertex is surrounded with an elastic virtual ball. Vertices may not penetrate the balls of other vertices. If you want a good result you may have to adjust the size of these balls. Normally it works pretty well with the default options.

### Ball Size Calculation

**Man (“manual”)** The *Ball Size* directly sets the ball size (in BU).

**Av (“average”)** The average length of all edges attached to the vertex is calculated and then multiplied with the *Ball Size* setting. Works well with evenly distributed vertices.

**Min / Max** The ball size is as large as the smallest/largest spring length of the vertex multiplied with the *Ball Size*.

**AvMiMax (“average min/max”)**  $Size = ((Min + Max)/2) \times Ball\ Size$ .

**Ball Size** Default 0.49 BU or fraction of the length of attached edges. The edge length is computed based on the algorithm you choose. You know how when someone stands too close to you, and feel uncomfortable? We call that our “personal space”, and this setting is the factor that is multiplied by the spring length. It is a spherical distance (radius) within which, if another vertex of the same mesh enters, the vertex starts to deflect in order to avoid a self-collision.

Set this value to the fractional distance between vertices that you want them to have their own “space”. Too high of a value will include too many vertices all the time and slow down the calculation. Too low of a level will let other vertices get too close and thus possibly intersect because there won’t be enough time to slow them down.

**Stiffness** Default 1.0. How elastic that ball of personal space is.

**Damping** Default 0.5. How the vertex reacts. A low value just slows down the vertex as it gets too close. A high value repulses it.

Collisions with other objects are set in the (other) **Collision panel**. To collide with another object they have to share at least one common layer.

### Simple examples

some simple examples showing the power of softbody physics.

**bouncing cube**

change your start and end frames to 1 and 150.



Fig. 2.1601: The timeline

add a plane, and scale it 5 times. next go to the physics tab, and add a collision. the default settings are fine for this example. now add a cube, or use the default cube. Tab into edit mode and subdivide it thrice. then add a bevel modifier to it, to smoothen the edges. to add a little more, press r twice, and move your cursor a bit.

when finish, your scene should look like this:

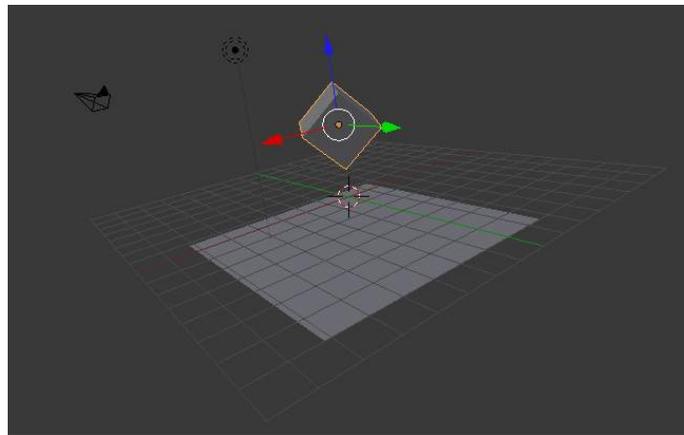


Fig. 2.1602: The scene, ready for softbody physics

Everything is ready to add the softbody physics. go to the physics tab and add 'softbody'. uncheck the soft body goal, and check softbody self collision. under soft body edges, increase the bending to 10.

playing the animation with alt a will now give a slow animation of a bouncing cube. to speed things up, we need to bake the softbody physics.

Under Soft Body Cache change start and end to your start and end frames. in this case 1 and 150. to test if everything is working, you can take a cache step of 5 or 10, but for the final animation it's better to reduce it to 1, to cache everything.

when finish, your physics panel should look like this:

you can now bake the simulation, give the cube materials and textures and render the animation.

**result**

The rendered bouncing cube:

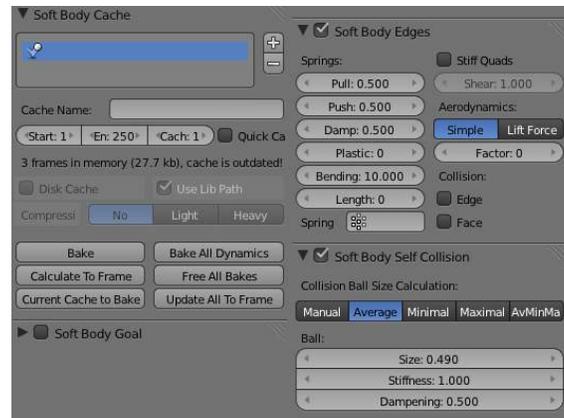


Fig. 2.1603: The physics settings.

## Combination With Armatures

## Combination With Hair Particles

## Soft Body settings

**Soft Body** This creates the soft body modifier on the selected object

**Render** Enable soft body during render

**Display** Display soft body in real time.

## Soft Body

**Friction** The friction of the surrounding medium. Generally friction dampens a movement.

**Mass** Mass value for vertices. Larger mass slows down acceleration, except for gravity where the motion is constant regardless of mass. Larger mass means larger inertia, so also braking a Soft Body is more difficult.

**Vertex Group Mass** Use a specified vertex group for mass values

**Speed** You can control the internal timing of the Softbody system with this value.

## Soft Body Cache

---

**Note:** Start- and Endframe

The *Start* and *End* settings in the *Collision* panel are not only valid for the baking, but for all Soft Body simulations. So if your animation lasts longer than 250 frames, you have to increase the *End* value.

---

**Cache** Select cache to use for simulation. Add, and remove caches.

**Cache Name** Specify the name of the cache.

**Start / End** First and last frame of the simulation. Always valid, not only for *baking*.

**Cache Step** Number of frames between cache steps.

**Disk Cache** Save cache files to disk. Blend file must be saved first.

**Use Lib Path** Use this files path when library linked into another file.

**Compression** Compression method to be used

**No** No compression.

**Light** Fast but not so effective compression.

**Heavy** Effective but slow compression.

**Bake** Calculates the simulation and protects the cache. You need to be in *Object* mode to bake.

**Free Bake** Clears the cache.

**Calculate to Frame** Bake physics to current frame

**Current Cache to Bake** Bake from Cache.

**Bake All Dynamics** Bake all physics

**Free All Bakes** Free all baked caches of all objects in the current scene

**Update All To Frame** Update cache to current frame

If you haven't saved the blend file the cache is created in memory, so save your file first or the cache may be lost.

### Soft Body Goal

**Use Goal** Soft Body Goal acts like a pin on a chosen set of vertices; controlling how much of an effect soft body has on them.

Enabling this tells Blender to use the position / animated position of a vertex in the simulation. Animating the vertices can be done in all the usual ways before the Soft Body simulation is applied. The *goal* is the desired end-position for vertices. How a softbody tries to achieve this goal can be defined using stiffness forces and damping.

**Default** If no vertex group is used, this numeric field is the default goal weight for all vertices. If a vertex group is present and assigned, this button instead shows an pop-up selector button that allows you to choose the name of the goal vertex group. A *Goal* value of 1.0 means no Soft Body simulation, the vertex stays at its original (animated) position. When setting *Goal* to 0.0, the object is only influenced by physical laws. By setting goal values between 0.0 and 1.0, you can blend between having the object affected only by the animation system, and having the object affected only by the soft body effect.

**Minimum / Maximum** When you paint the values in vertex-groups (using *Weight Paint* mode), you can use the *G Min* and *G Max* to fine-tune (clamp) the weight values. The lowest vertex-weight (blue) will become *G Min*, the highest value (red) becomes *G Max* (please note that the blue-red color scale may be altered by User Preferences).

**Stiffness** The spring stiffness for *Goal*. A low value creates very weak springs (more flexible “attachment” to the goal), a high value creates a strong spring (a stiffer “attachment” to the goal).

**Damping** The friction for *Goal*. Higher values dampen the effect of the goal on the soft body.

**Vertex Group** Use a vertex group to specify goal weights.

### Soft Body Edges

**Use Edges** The edges in a Mesh Object can act as springs as well, like threads in fabric.

**Pull** The spring stiffness for edges (how much the edges are stretched). A low value means very weak springs (a very elastic material), a high value is a strong spring (a stiffer material) that resists being pulled apart. 0.5 is latex, 0.9 is like a sweater, 0.999 is a highly-starched napkin or leather.

**Push** How much the Softbody resist being scrunched together, like a compression spring. Low values for fabric, high values for inflated objects and stiff material.

**Damp** The friction for edge springs. High values (max of 50) dampen the edge stiffness effect and calm down the cloth.

**Plastic** Plasticity, permanent deformation of the object.

**Bending** This option creates virtual connections between a vertex and the one after the next. This includes diagonal edges. Damping applies also to these connections.

**Length** The edges can shrink or been blown up. This value is given in percent, 0 disables this function. 100% means no change, the body keeps 100% of his size.

**Stiff Quads** For quad faces, the diagonal edges are used as springs. This stops quad faces to collapse completely on collisions (what they would do otherwise).

**Shear** Stiffness of the virtual springs for quad faces.

### Aerodynamics

**Simple** If you turn on *Aero* the force is not confined to the vertices, but has an effect also on the edges. The angle and the relative speed between medium and edge is used to calculate the force on the edge. This force results that vertices with little connecting edges (front of a plane) fall faster than vertices with more connecting edges (middle of a plane). If all vertices have the same amount of edges in a direction they fall with equal speed. An edge moving in its own direction feels no force, and an edge moving perpendicular to its own direction feels maximum force (think of a straw moving through air). Try it with an *Factor* of 30 at first.

**Lift Force** Use an aerodynamic model that is closer to physical laws and looks more interesting. Disable for a more muted simulation.

**Factor** How much aerodynamic effect to use

**Edge** Checks for edges of the softbody mesh colliding.

**Face** Checks for any portion of the face of the softbody mesh colliding (compute intensive!). While *CFace* enabled is great, and solves lots of collision errors, there doesn't seem to be any dampening settings for it, so parts of the softbody object near a collision mesh tend to "jitter" as they bounce off and fall back, even when there's no motion of any meshes. Edge collision has dampening, so that can be controlled, but Deflection dampening value on a collision object doesn't seem to affect the face collision.

### Soft Body Self Collision

*Self Collision* is working only if you have activated *Use Edges*.

**Self Collision** When enabled, allows you to control how Blender will prevent the Soft Body from intersecting with itself. Every vertex is surrounded with an elastic virtual ball. Vertices may not penetrate the balls of other vertices. If you want a good result you may have to adjust the size of these balls. Normally it works pretty well with the default options.

**Manual** The *Ball Size* directly sets the ball size (in BU).

**Average ("average")** The average length of all edges attached to the vertex is calculated and then multiplied with the *Ball Size* setting. Works well with evenly distributed vertices.

**Minimal / Maximal** The ball size is as large as the smallest/largest spring length of the vertex multiplied with the *Ball Size*.

**AvMiMax**  $Size = ((Min + Max)/2) \times Ball Size$ .

**Size** Default 0.49 BU or fraction of the length of attached edges. The edge length is computed based on the algorithm you choose. You know how when someone stands too close to you, and feel uncomfortable? We call that our "personal space", and this setting is the factor that is multiplied by the spring length. It is a spherical distance (radius) within which, if another vertex of the same mesh enters, the vertex starts to deflect in order to avoid a self-collision. Set this value to the fractional distance between vertices that you want them to have their own "space". Too high of a value will include too many vertices all the time and slow down the calculation. Too low of a level will let other vertices get too close and thus possibly intersect because there won't be enough time to slow them down.

**Stiffness** Default 1.0. How elastic that ball of personal space is.

**Dampening** Default 0.5. How the vertex reacts. A low value just slows down the vertex as it gets too close. A high value repulses it.

Collisions with other objects are set in the (other) [Collision panel](#). To collide with another object they have to share at least one common layer.

### Soft Body Solver

These settings determine the accurateness of the simulation.

**Min Step** Minimum simulation steps per frame. Increase this value, if the Soft Body misses fast moving collision objects.

**Max Step** Maximum simulation steps per frame. Normally the number of simulation steps is set dynamically (with the *Error Limit*) but you have probably a good reason to change it.

**Auto-Step** helps the Solver figure out how much work it needs to do based on how fast things are moving.

**Error Limit** Rules the overall quality of the solution delivered. Default 0.1. The most critical setting that says how precise the solver should check for collisions. Start with a value that is 1/2 the average edge length. If there are visible errors, jitter, or over-exaggerated responses, decrease the value. The solver keeps track of how “bad” it is doing and the *Error Limit* causes the solver to do some “adaptive step sizing”.

**Fuzzy** Fuzziness while on collision, high values make collision handling faster but less stable.

**Choke** Calms down (reduces the exit velocity of) a vertex or edge once it penetrates a collision mesh.

**Print Performance to Console** Prints on the console how the solver is doing.

**Estimate Matrix** Estimate matrix... split to COM, ROT, SCALE

## 2.7.9 Rigid Body

### Introduction

The rigid body simulation can be used to simulate the motion of solid objects. It affects the position and orientation of objects and does not deform them.

Unlike the other simulations in Blender, the rigid body sim works closer with the animation system. This means that rigid bodies can be used like regular objects and be part of parent-child relationships, animation constraints and drivers.

### Creating a Rigid Body

Creating the Rigid Body.

Right now only mesh objects can participate in the rigid body simulation. To create rigid bodies, either click on *Rigid Body* button in the *Physics* context of the *Properties* window or use the *Add Active/Add Passive* buttons in the *Physics* tab of the *Tool Shelf*.

There are two types of rigid body: active and passive. *Active* bodies are dynamically simulated, while *passive* bodies remain static. Both types can be driven by the animation system when using the *Animated* option.

During the simulation, the rigid body system will override the position and orientation of dynamic rigid body objects. Note however that the location and rotation of the objects is not changed, so the rigid body sim acts similar to a constraint. To apply the rigid body transformations you can use the *Apply Transformation* button in the *Physics* tab of the *Tool Shelf*.

The scale of the rigid body object also influences the simulation, but is always controlled by the animation system.

Rigid Body physics on the object can be *removed* with the *Rigid Body* button in the *Physics* context or *Remove* button in the *Physics* tab of the *Tool Shelf*.

## Rigid Body Properties

### Main properties

Rigid Body panel.

**Type** Role of the rigid body in the simulation. Active objects can be simulated dynamically, passive object remain static.

**Active** Object is directly controlled by simulation results. The possibility to select this type also available with *Add Active* button in the *Physics* tab of the *Tool Shelf*.

**Passive** Object is directly controlled by animation system. Thus, this type is not available for *Rigid Body Dynamics*. The possibility to select this type also available with *Add Passive* button in the *Physics* tab of the *Tool Shelf*.

**Dynamic** Enables/disables rigid body simulation for object.

**Animated** Allows the rigid body additionally to be controlled by the animation system.

**Mass** Specifies how heavy the object is and “weights” irrespective of gravity. There are predefined mass preset available with the *Calculate Mass* button in the *Physics* tab of the *Tool Shelf*.

**Calculate Mass** Automatically calculate mass values for Rigid Body Objects based on volume. There are many useful presets available from the menu, patching real-world objects.

Also you can have *Custom* mass material type, which is achieved by setting a custom density value ( $\text{kg/m}^3$ ).

### Rigid Body Collisions

Rigid Body Collisions panel.

#### General settings

##### Surface Response

**Friction** Resistance of object to movement. Specifies how much velocity is lost when objects collide with each other.

**Bounciness** Tendency of object to bounce after colliding with another (“0” - stays still, “1” - perfectly elastic). Specifies how much objects can bounce after collisions.

**Collision Groups** Allows rigid body collisions allocate on different groups (maximum 20).

**Collision shapes** The *Shape* option determines the collision shape of the object. The following Collision Shapes are available:

- **Primitive shapes** : these are best in terms of memory/performance but don’t necessarily reflect the actual shape of the object. They’re calculated based on the object’s bounding box. The center of gravity is always in the middle for now.
  - **Box** Box-like shapes (i.e. cubes), including planes (i.e. ground planes). The size per axis is calculated from the bounding box.
  - **Sphere** Sphere-like shapes. The radius is the largest axis of the bounding box.
  - **Capsule** This points up the Z-Axis.
  - **Cylinder** This points up the Z-Axis. The height is taken from the z-axis, while the radius is the larger of the x/y-axes.
  - **Cone** This points up the Z-Axis. The height is taken from the z-axis, while the radius is the larger of the x/y-axes.

- **Mesh based shapes** : these are calculated based on the geometry of the object so they are a better representation of the object. The center of gravity for these shapes is the object origin.
  - **Convex Hull** A mesh-like surface encompassing (i.e. shrinkwrap over) all vertices (best results with fewer vertices). Convex approximation of the object, has good performance and stability.
  - **Mesh Mesh** consisting of triangles only, allowing for more detailed interactions than convex hulls. Allows to simulate concave objects, but is rather slow and unstable.

The changing collision shape is available also with *Change Shape* button in the *Physics* tab of the *Tool Shelf*.

**Mesh source** Users can now specify the mesh *Source* for *Mesh* bases collision shapes:

**Base** The base mesh of the object.

**Deform** Includes any deformations added to the mesh (shape keys, deform modifiers).

**Deforming** Rigid body deforms during simulation.

**Final** Includes all modifiers.

### Collision Margin

**Margin** Threshold of distance near surface where collisions are still considered (best results when non-zero).

The collision margin is used to improve performance and stability of rigid bodies. Depending on the shape, it behaves differently: some shapes embed it, while others have a visible gap around them.

The margin is *embedded* for these shapes:

- Sphere
- Box
- Capsule
- Cylinder
- Convex Hull: Only allows for uniform scale when embedded.

The margin is *not embedded* for these shapes:

- Cone
- Active Triangle Mesh
- Passive Triangle Mesh: Can be set to 0 most of the time.

### Rigid Body Dynamics

Rigid Body Dynamics panel.

This panel is available only for *Active* type of rigid bodies.

#### Deactivation:

**Enable Deactivation** Enable deactivation of resting rigid bodies. Allows object to be deactivated during the simulation (improves performance and stability, but can cause glitches).

**Start Deactivated** Starts objects deactivated. They are activated on collision with other objects.

**Linear Vel** Specifies the linear deactivation velocity below which the rigid body is deactivated and simulation stops simulating object.

**Angular Vel** Specifies the angular deactivation velocity below which the rigid body is deactivated and simulation stops simulating object.

**Damping:**

**Translation** Amount of linear velocity that is lost over time.

**Rotation** Amount of angular velocity that is lost over time.

## Rigid Body World

Rigid Body World panel.

The rigid body world is a group of Rigid Body objects, which holds settings that apply to all rigid bodies in this simulation and can be found in *Rigid Body World* panel of *Scene* context.

When you add Rigid Body physics on an object, primary there is created a group of objects with default “RigidBodyWorld” name. Rigid body objects automatically are added to this group when you add Rigid Body physics for them.

You can be create several Rigid Body World groups and allocate there yours Rigid Body objects with *Groups* panel in *Object* context.

Rigid body objects and constraints are only taken into account by the simulation if they are in the groups specified in *Group* field of the *Rigid Body World* panel in the *Scene* context.

**Rigid Body World checkbox** Enable/disable evaluation of the Rigid Body simulation based on the rigid body objects participating in the specified group of Rigid Body World.

**Remove Rigid Body World button** Remove Rigid Body simulation from the current scene.

**Group** Containing rigid body objects participating in this simulation.

**Constraints** Containing rigid body object constraints participating in the simulation.

Simulation quality and timing settings:

**Speed** Can be used to speed up/slow down the simulation.

**Split Impulse** Enable/disable reducing extra velocity that can build up when objects collide (lowers simulation stability a little so use only when necessary). Limits the force with which objects are separated on collision, generally produces nicer results, but makes the simulation less stable (especially when stacking many objects).

**Steps Per Second** Number of simulation steps made per second (higher values are more accurate but slower). This only influences the accuracy and not the speed of the simulation.

**Solver Iterations** Amount of constraint solver iterations made per simulation step (higher values are more accurate but slower). Increasing this makes constraints and object stacking more stable.

## Rigid Body caching and baking

Rigid Body Cache panel.

Specifies the frame range in which the simulation is active. Can be used to bake the simulation.

**Start/End** First and last frame of the simulation.

**Bake** Calculates the simulation and protects the cache. You need to be in *Object* mode to bake.

**Free Bake** Active after the baking of simulation. Clears the baked cache.

**Calculate to Frame** Bake physics to current frame.

**Current Cache to Bake** Bake from Cache.

**Bake All Dynamics** Bake all physics.

**Free All Bakes** Free all baked caches of all objects in the current scene.

**Update All To Frame** Update cache to current frame.

If you haven't saved the blend file, the cache is created in memory, so save your file first or the cache may be lost.

### External Force Influence on Rigid Body

Rigid Body Cache panel.

As other physics dynamics systems, Rigid Body simulation are also influenced by external force effectors.

### Rigid Body Constraints

*Constraints* (also known as joints) for rigid bodies connect two rigid bodies.

The physics constraints available in the non-game modes are meant to be attached to an *Empty* object. The constraint then has fields which can be pointed at the two physics-enabled object which will be bound by the constraint. The *Empty* object provides a location and axis for the constraint distinct from the two constrained objects. The location of the entity hosting the physics constraint marks a location and set of axes on each of the two constrained objects. These two anchor points are calculated at the beginning of the animation and their position and orientation remain fixed *in the local coordinate system of the object* for the duration of the animation. The objects can move far from the constraint object, but the constraint anchor moves with the object. If this feature seems limiting, consider using multiple objects with a non-physics *Child-of* constraint and animate the relative location of the child.

The quickest way to constrain two objects is to select both and click the *Connect* button in the *Physics* tab of the *Tool Shelf*. This creates a new *Empty* object (named "Constraint") with a physics constraint already attached and pointing at the two selected objects.

Also you can create *Rigid Body Constraint* on of the two constrained objects with *Rigid Body Constraint* button of the *Physics* context in the *Properties* window. This constraint be depend on the object location and rotation which on it created. Thereafterat, there are no *Empty* object created for the constraint. The role of the *Empty* object is put on this object. The constrained object can be then set as *Passive* type for better driving the constrain.

Additional parameters appear in the *Rigid Body Constraint* panel of the *Physics* context in the *Properties* window for the selected *Empty* object or the one of the two constrained objects with the created constraint.

### Constraint Types

The menu of available constraint types.

There are several constraint types available:

**Fixed** Glue rigid bodies together. Fixes the relative position and orientation of two rigid bodies.

**Point** Constrains rigid bodies to move around a common pivot point.

**Hinge** Only allows rotation around the Z axis of the constraint object.

**Slider** Limits movement to the X axis of the constraint object.

**Piston** Restrict rigid body translation and rotation to the X axis of the constraint object.

**Generic** Restrict rigid body translation and rotation to specified axes. Allows customizable constraint axes.

**Generic Spring** Restrict rigid body translation and rotation to specified axes with springs. Like Generic with spring motion.

**Motor** Drive rigid body around or along an axis.

## Constraint options and settings

Rigid Body Constraint panel.

**Enabled** Specifies whether the constraint is active during the simulation.

**Disable Collisions** Allows constrained objects to pass through one another.

**Object 1** First object to be constrained.

**Object 2** Second object to be constrained.

**Breakable** Allows constraint to break during simulation. Disabled for the *Motor* constraint.

**Threshold** Impulse strength that needs to be reached before constraint breaks.

**Override Iterations** Allows to make constraints stronger (more iterations) or weaker (less iterations) than specified in the rigid body world.

**Iterations** Number of constraint solver iterations made per simulation step for this constraint.

**Limits:** By using limits you can constrain objects even more by specifying a translation/rotation range on/around respectively axis (see below for each one individually). To lock one axis, set both limits to 0.

## Fixed

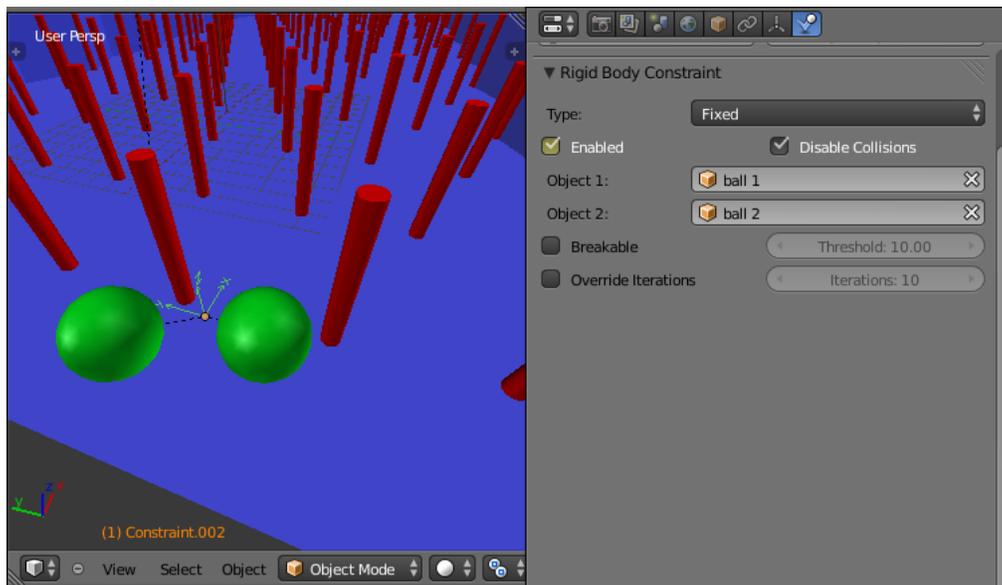
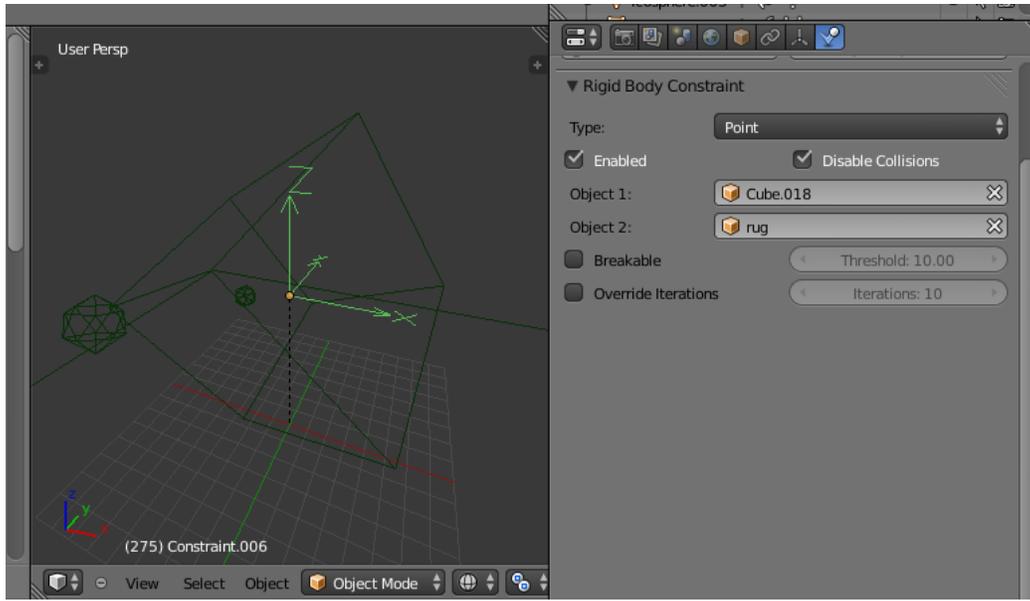


Fig. 2.1604: Options available to a *Fixed* constraint.

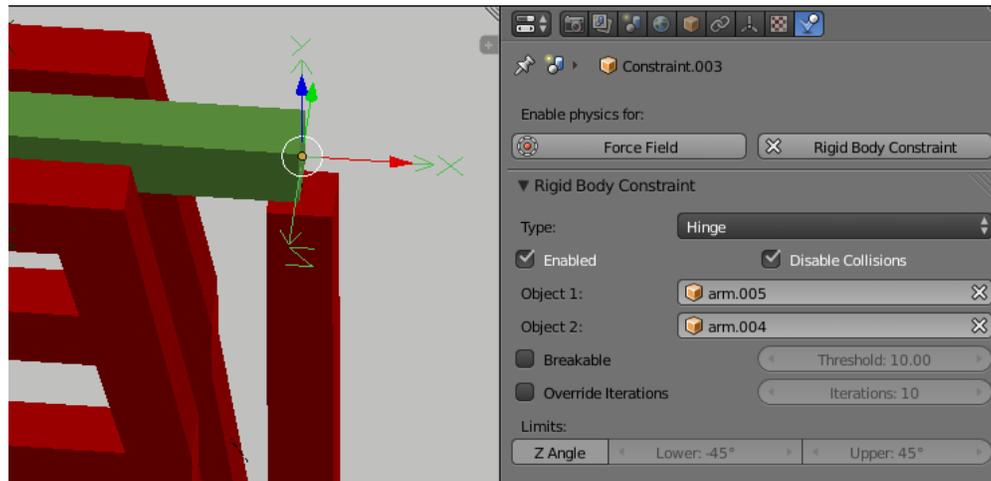
This constraint cause the two objects to move as one. Since the physics system does have a tiny bit of slop in it, the objects don't move as rigidly as they would if they were part of the same mesh.

## Point

The objects are linked by a point bearing allowing any kind of rotation around the location of the constraint object, but no relative translation is permitted. The physics engine will do its best to make sure that the two points designated by the constraint object on the two constrained objects are coincident.

Fig. 2.1605: Options available to a *Point* constraint.

## Hinge

Fig. 2.1606: Options available to a *Hinge* constraint.

The hinge permits 1 degree of freedom between two objects. Translation is completely constrained. Rotation is permitted about the Z axis of the object hosting the Physics constraint (usually an *Empty*, distinct from the two objects that are being linked). Adjusting the position and rotation of the object hosting the constraint allows you to control the anchor and axis of the hinge.

The Hinge is the only 1-axis rotational constraint that uses the Z axis instead of the X axis. If something is wrong with your hinge, check your other constraints to see if this might be the problem.

### Limits:

**Z Angle** Enables/disables limit rotation around Z axis.

**Lower** Lower limit of Z axis rotation.

**Upper** Upper limit of Z axis rotation.

## Slider

The Slider constraint allows relative translation along the X axis of the constraint object, but permits no relative rotation, or relative translation along other axes.

### Limits:

**X Axis** Enables/disables limit translation around X axis.

**Lower** Lower limit of X axis translation.

**Upper** Upper limit of X axis translation.

## Piston

A piston permits translation along the X axis of the constraint object. It also allows rotation around the X axis of the constraint object. It's like a combination of the freedoms of a slider with the freedoms of a hinge (neither of which is very free alone).

### Limits:

**X Axis** Enables/disables limit translation around X axis.

**Lower** Lower limit of X axis translation.

**Upper** Upper limit of X axis translation.

**X Angle** Enables/disables limit rotation around X axis.

**Lower** Lower limit of X axis rotation.

**Upper** Upper limit of X axis rotation.

## Generic

The generic constraint has a lot of available parameters.

The X, Y, and Z axis constraints can be used to limit the amount of translation between the objects. Clamping the min/max to zero has the same effect as the *Point* constraint.

Clamping the relative rotation to zero keeps the objects in alignment. Combining an absolute rotation and translation clamp would behave much like the *Fixed* constraint.

Using a non-zero spread on any parameter allows it to rattle around in that range throughout the course of the simulation.

### Limits:

**X Axis/Y Axis/Z axis** Enables/disables limit translation on X, Y or Z axis respectively.

**Lower** Lower limit of translation for X, Y or Z axis respectively.

**Upper** Upper limit of translation for X, Y or Z axis respectively.

**X Angle/Y Angle/Z Angle** Enables/disables limit rotation around X, Y or Z axis respectively.

**Lower** Lower limit of rotation for X, Y or Z axis respectively.

**Upper** Upper limit of rotation for X, Y or Z axis respectively.

## Generic Spring

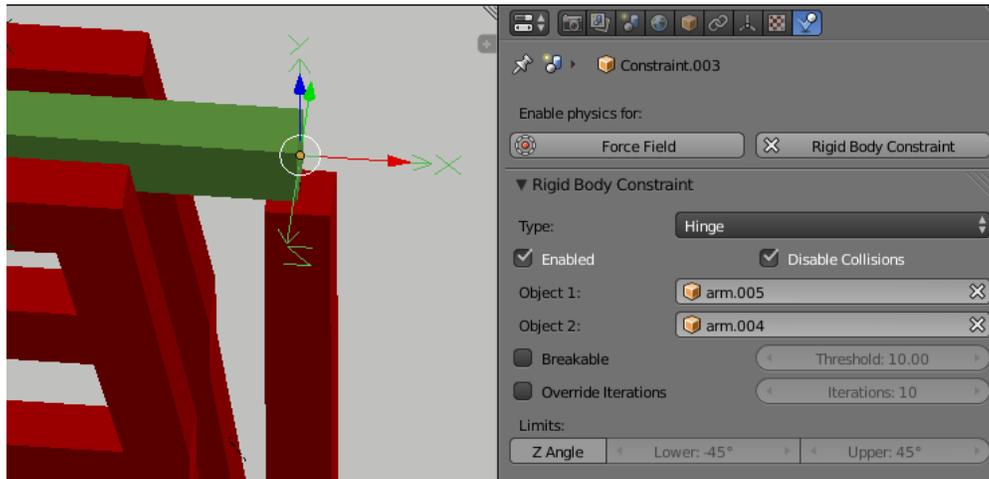


Fig. 2.1607: Options available to a *Generic Spring* constraint.

The generic spring constraint adds some spring parameters for the X/Y/Z axes to all the options available on the *Generic* constraint. Using the spring alone allows the objects to bounce around as if attached with a spring anchored at the constraint object. This is usually a little too much freedom, so most applications will benefit from enabling translation or rotation constraints.

If the damping on the springs is set to 1, then the spring forces are prevented from realigning the anchor points, leading to strange behavior. If your springs are acting weird, check the damping.

### Limits:

**X Axis/Y Axis/Z axis** Enables/disables limit translation on X, Y or Z axis respectively.

**Lower** Lower limit of translation for X, Y or Z axis respectively.

**Upper** Upper limit of translation for X, Y or Z axis respectively.

**X Angle/Y Angle/Z Angle** Enables/disables limit rotation around X, Y or Z axis respectively.

**Lower** Lower limit of rotation for X, Y or Z axis respectively.

**Upper** Upper limit of rotation for X, Y or Z axis respectively.

### Springs:

**X/Y/Z** Enables/disables springs on X, Y or Z axis respectively.

**Stiffness** Spring Stiffness on X, Y or Z axis respectively. Specifies how “bendy” the spring is.

**Damping** Spring Damping on X, Y or Z axis respectively. Amount of damping the spring has.

## Motor

The motor constraint causes translation and/or rotation between two entities. It can drive two objects apart or together. It can drive simple rotation, or rotation and translation (although it won't be constrained like a screw since the translation can be blocked by other physics without preventing rotation).

The rotation axis is the X axis of the object hosting the constraint. This is in contrast with the *Hinge* which uses the Z axis. Since the Motor is vulnerable to confusing perturbations without a matching *Hinge* constraint, special care must be taken to align the axes. Without proper alignment, the motor will appear to have no effect (because the hinge is preventing the motion of the motor).

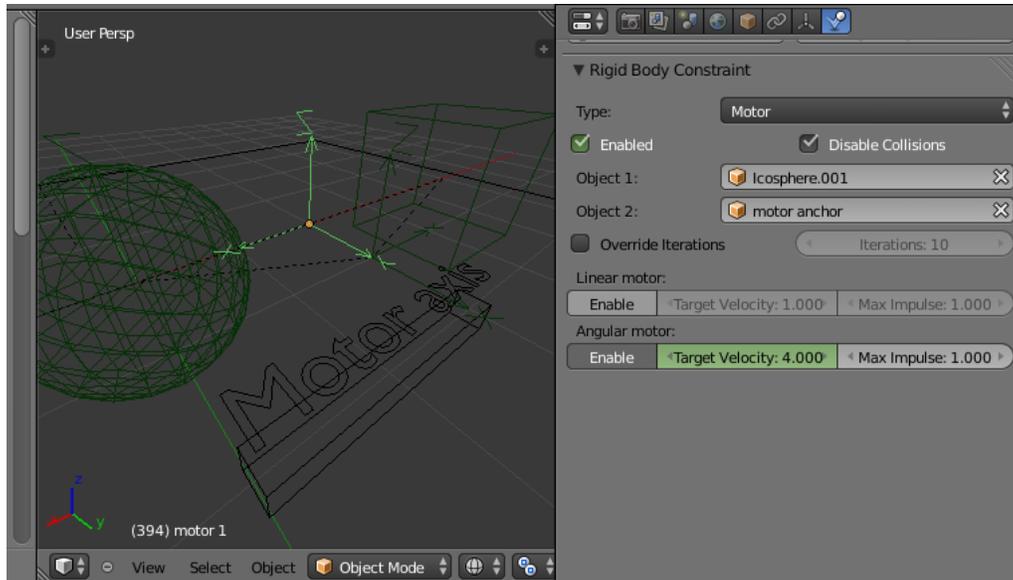


Fig. 2.1608: Options available to a *Motor* constraint.

#### **Linear motor/Angular motor:**

**Enable** Enable linear or angular motor respectively.

**Target Velocity** Target linear or angular motor velocity respectively.

**Max Impulse** Maximum linear or angular motor impulse respectively.

#### **Tips**

As with all physics-enabled objects, pay close attention to the *Animated* check box in the *Rigid Body* panel of the *Physics* context in the *Properties* window. A common mistake is to use keyframe animation on a *Passive* physics object without checking the *Animated* box. The object will move, but the physics engine will behave as if the *Passive* is still in its starting place, leading to disappointment.

#### **Animation**

The most common trick is to *keyframe* animate the location or rotation of an *Active* physics object as well as the *Animated* checkbox. When the curve on the *Animated* property switches to disabled, the physics engine takes over using the object's last known location, rotation and velocities.

Animating the strengths of various other parameters (a *Motor's* Target Velocity, a *Hinge's* limits, etc) can be used to accomplish a wide variety of interesting results.

Enabling a constraint during the physics simulation often has dramatic results as the physics engine tries to bring into alignment two objects which are often dramatically out of alignment. It is very common for the affected objects to build up enough kinetic energy to bounce themselves out of camera (and into orbit, although the physics engine is not yet capable of simulating a planet's gravity well, so scratch that).

Rigid Body dynamics can be baking to normal keyframes with *Bake To Keyframes* button in the *Physics* tab of the *Tool Shelf*.

### Simulation Stability

The simplest way of improving simulation stability is to increase the steps per second. However, care has to be taken since making too many steps can cause problems and make the simulation even less stable (if you need more than 1000 steps, you should look at other ways to improve stability).

Increasing the number of solver iterations helps making constraints stronger and also improves object stacking stability.

It's best to avoid small objects, as they're currently unstable. Ideally, objects should be at least 20 cm in diameter. If it's still necessary, setting the collision margin to 0, while generally not recommended, can help making small object behave more naturally.

When objects are small and/or move very fast, they can pass through each other. Besides what's mentioned above it's also good to avoid using mesh shapes in this case. Mesh shapes consist of individual triangles and therefore don't really have any thickness, so objects can pass through more easily. You can give them some thickness by increasing the collision margin.

### Combining Rigid Bodies with Other Simulations

Since the rigid body simulation is part of the animation system, it can influence other simulations just like the animation system can.

In order for this to work, the rigid body object needs to have a collision modifier. Simply click on *Collision* in the *Physics* context.

### Scaling Rigid Bodies

Rigid body objects can be scaled, also during the simulation. This work well in most cases, but can sometimes cause problems.

If dynamic scaling is not needed, rigid body objects should have the scale applied by using the *Apply Scale* command (Ctrl-A).

## 2.7.10 Force Fields

Force Fields offer a way to add extra movement to dynamic systems. [Particles](#), [Soft Bodies](#), [Rigid Bodies](#) and [Cloth](#) objects can all be affected by forces fields. Force Fields automatically affect everything. To remove a simulation or particle system from their influence, simply turn down the influence of that type of Force Field in its Field Weights panel.

- All types of objects and particles can generate fields, but only curve object can bear *Curve Guides* fields.
- Force Fields can also be generated from particles. See [Particle Physics](#)
- The objects need to share at least one common layer to have effect.

You may limit the effect on particles to a group of objects (see the [Particle Physics](#) page).

### Creating a Force Field

---

#### Reference

Mode: *Object Mode*

Panel: *Object* context → *Physics* sub-context → *Fields*

To create a single Force Field, you can select *Add* → *Force Field* and select the desired force field. This method creates an Empty with the force field attached.

To create a field from an existing object you have to select the object and change to the *Physics* sub-context. Select the field type in the *Fields* menu.

The fields have many options in common, these common options are explained for the *Spherical* field.

**Note:** After changing the fields (*Fields* panel) or deflection (*Collision* panel) settings, you have to recalculate the particle, softbody or cloth system (*Free Cache*), this is not done automatically. You can clear the cache for all selected objects with `Ctrl-B` → *Free cache selected*.

*Particles* react to all kind of *Force Fields*, *Soft Bodies* only to *Spherical* / *Wind* / *Vortex* (they react on *Harmonic* fields but not in a useful way).

## Common Field Settings

Most Fields have the same settings, even though they act very differently. Settings unique to a field type are described below. Curve Guide and Texture Fields have very different options.

**Shape** The field is either a *Point*, with omnidirectional influence, or a *Plane*, constant in the XY-plane, changes only in Z direction.

**Strength** The strength of the field effect. This can be positive or negative to change the direction that the force operates in. A force field's strength is scaled with the force object's scale, allowing you to scale up and down scene, keeping the same effects.

**Flow** Convert effector force into air flow velocity.

**Noise** Adds noise to the strength of the force.

**Seed** Changes the seed of the random noise.

**Effect Point** You can toggle the field's effect on particle *Location* and *Rotation*

**Collision Absorption** Force gets absorbed by collision objects.

## Falloff

Here you can specify the shape of the force field (if the *Falloff Power* is greater than 0).

**Sphere** Falloff is uniform in all directions, as in a sphere.

**Tube** Fall off results in a tube shaped force field. The Field's *Radial falloff* can be adjusted, as well as the *Minimum* and *Maximum* distances of the field.

**Cone** Fall off results in a cone shaped force field. Additional options are the same as those of *Tube* options.

**Z Direction** *Falloff* can be set to apply only in the direction of the positive Z Axis, negative Z Axis, or both.

**Power (Power)** How the power of the force field changes with the distance from the force field. If  $r$  is the distance from the center of the object, the force changes with  $1/r^{\text{Power}}$ . A *Falloff* of 2 changes the force field with  $1/r^2$ , which is the falloff of gravitational pull.

**Max Distance** Makes the force field only take effect within a specified maximum radius (shown by an additional circle around the object).

**Min Distance** The distance from the object center, up to where the force field is effective with full strength. If you have a *Fall-off* of 0 this parameter does nothing, because the field is effective with full strength up to *Max Dist* (or the infinity). Shown by an additional circle around the object.

## Field Types

### Force

The *Force* field is the simplest of the fields. It gives a constant force towards (positive strength) or away from (negative strength) the object's center. Newtonian particles are attracted to a field with negative strength, and are blown away from a field with positive strength.

For *Boids* a field with positive strength can be used as a *Goal*, a field with negative strength can be used as *Predator*. Whether *Boids* seek or fly goals/predators depends on the *Physics* settings of the Boids.

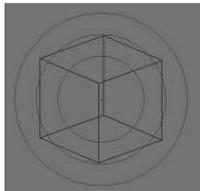


Fig. 2.1609: Image 2b: Spherical field indicator.

### Wind

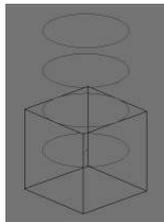


Fig. 2.1610: Image 3a: Wind field indicator.

Wind gives a constant force in a single direction, along the force object's local Z axis. The strength of the force is visualized by the spacing of the circles shown.

### Vortex Field

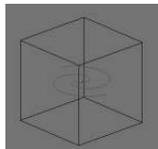


Fig. 2.1611: Image 3b: Vortex field indicator.

Vortex fields give a spiraling force that twists the direction of points around the force object's local Z axis. This can be useful for making a swirling sink, or tornado, or kinks in particle hair.

## Magnetic

This field depends on the speed of the particles. It simulates the force of magnetism on magnetized objects.

## Harmonic

The source of the force field is the zero point of a harmonic oscillator (spring, pendulum). If you set the *Damping* parameter to 1, the movement is stopped in the moment the object is reached. This force field is really special if you assign it to particles.

**Rest Length** Controls the rest length of the harmonic force.

**Multiple Springs** Causes every point to be affected by multiple springs.

Normally every particle of the field system influences every particle of the target system. Not with *Harmonic* ! Here every target particle is assigned to a field particle. So particles will move to the place of other particles, thus forming shapes. Tutorial: [Particles forming Shapes](#)

## Charge

It is similar to spherical field except it changes behavior (attract/repulse) based on the effected particles charge field (negative/positive), like real particles with a charge. This mean this field has only effect on particles that have also a *Charge* field (else, they have no “charge”, and hence are unaffected)!

## Lennard-Jones

This field is a very short range force with a behavior determined by the sizes of the effector and effected particle. At a distance smaller than the combined sizes the field is very repulsive and after that distance it's attractive. It tries to keep the particles at an equilibrium distance from each other. Particles need to be at a close proximity to each other to be effected by this field at all.

Particles can have for example both a charge and a Lennard-Jones potential - which is probably something for the nuclear physicists amongst us.

## Texture field

You can use a texture force field to create an arbitrarily complicated force field, which force in the 3 directions is color coded. Red is coding for the x-axis, green for the y-axis and blue for the z-axis (like the color of the coordinate axes in the 3D window). A value of 0.5 means no force, a value larger than 0.5 acceleration in negative axis direction (like -Z), a value smaller than 0.5 acceleration in positive axis direction (like +Z).

**Texture mode** This sets the way a force vector is derived from the texture.

**RGB** Uses the color components directly as the force vector components in the color encoded directions. You need an RGB texture for this, e.g. an image or a colorband. So a *Blend* texture without a colorband would not suffice.

**Gradient** Calculates the force vector as the 3d-gradient of the intensity (grayscale) of the texture. The gradient vector always points to the direction of increasing brightness.

**Curl** Calculates the force vector from the curl of the 3d-rgb texture (rotation of rgb vectors). This also works only with a color texture. It can be used for example to create a nice looking turbulence force with a color clouds texture with perlin noise.

**Nabla** It is the offset used to calculate the partial derivatives needed for *Gradient* and *Curl* texture modes.

**Use Object Coordinates** Uses the emitter object coordinates (and rotation & scale) as the texture space the particles use. Allows for moving force fields, that have their coordinates bound to the location coordinates of an object.

**Root Texture Coordinates** This is useful for hair as it uses the texture force calculated for the particle root position for all parts of the hair strand.

**2D** The *2D* button disregards the particles z-coordinate and only uses particles x&y as the texture coordinates.

Remember that only procedural texture are truly 3D.

### Examples

- A single colored texture 0.5/0.0/0.5 creates a force in the direction of the positive y-axis, e.g. hair is orientated to the y-axis.
- A blend texture with colorband can be used to created a force “plane”. E.g. on the left side 0.5/0.5/0.5, on the right side 1.0/0.5/0.5 you have a force plane perpendicular to XY (i.e. parallel to Z). If you use an object for the coordinates, you can use the object to push particles around.
- An animated wood texture can be used to create a wave like motion.

### Curve Guide

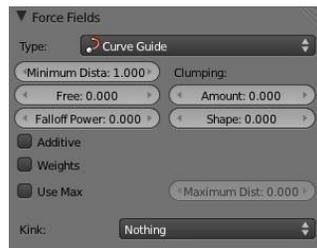


Fig. 2.1612: Image 4a: A Curve Guide field.

*Curve* objects can be the source of a *Curve Guide* field. You can guide particles along a certain path, they don't affect Softbodies. A typical scenario would be to move a red blood cell inside a vein, or to animate the particle flow in a motor. You can use *Curve Guide* s also to shape certain hair strands - though this may no longer be used as often now because we have the [Particle Mode](#). Since you can animate curves as Softbody or any other usual way, you may build very complex animations while keeping great control and keeping the simulation time to a minimum.

The option *Curve Follow* does not work for particles. Instead you have to set *Angular Velocity* (in the *Physics* panel of the *Particle* sub-context) to *Spin* and leave the rotation constant (i.e. don't turn on *Dynamic*).

*Curve Guide* s affect all particles on the same layer, independently from their distance to the curve. If you have several guides in a layer, their fields add up to each other (the way you may have learned it in your physics course). But you can limit their influence radius:

**Minimum Distance** The distance from the curve, up to where the force field is effective with full strength. If you have a *Falloff* of 0 this parameter does nothing, because the field is effective with full strength up to *MaxDist* (or the infinity). *MinDist* is shown with a circle at the endpoints of the curve in the 3D window.

**Free** Fraction of particle life time, that is not used for the curve.

**Falloff** This setting governs the strength of the guide between *MinDist* and *MaxDist*. A *Falloff* of 1 means a linear progression.

A particle follows a *Curve Guide* during it's lifetime, the velocity depends from it's lifetime and the length of the path.

**Additive** If you use *Additive*, the speed of the particles is also evaluated depending on the *Falloff*.

**Weights** Use Curve weights to influence the particle influence along the curve.

**Maximum Distance / Use Max** The maximum influence radius. Shown by an additional circle around the curve object.

The other settings govern the form of the force field along the curve.

**Clumping Amount** The particles come together at the end of the curve (1) or they drift apart (-1).

**Shape** Defines the form in which the particles come together. +0.99: the particles meet at the end of the curve. 0: linear progression along the curve. -0.99: the particles meet at the beginning of the curve.

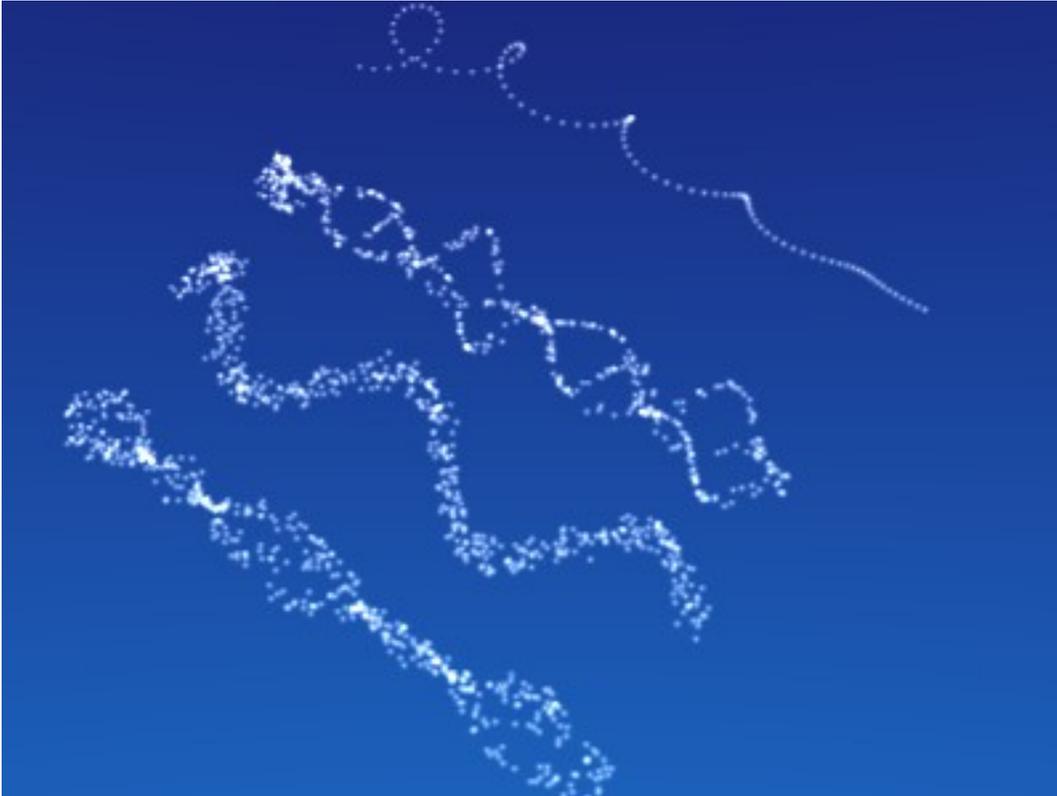


Fig. 2.1613: Image 4b: Kink options of a curve guide. From left to right: Radial, Wave, Braid, Roll. [Animation](#)

With the drop down box *Kink*, you can vary the form of the force field:

**Curl** The radius of the influence depends on the distance of the curve to the emitter.

**Radial** A three dimensional, standing wave.

**Wave** A two dimensional, standing wave.

**Braid** Braid.

**Roll** A one dimensional, standing wave.

It is not so easy to describe the resulting shapes, I hope it's shown clearly enough in (*Image 4b*).

**Frequency** The frequency of the offset.

**Shape** Adjust the offset to the beginning/end.

**Amplitude** The Amplitude of the offset.

## Boid

Boid probably comes from theoretical works. *Boids* is an artificial life program, developed by Craig Reynolds in 1986, which simulates the flocking behaviour of birds. His paper on this topic was published in 1987 in the proceedings of the ACM SIGGRAPH conference. The name refers to a “bird-like object”, but its pronunciation evokes that of “bird” in a stereotypical New York accent. As with most artificial life simulations, Boids is an example of emergent behavior; that is, the complexity of Boids arises from the interaction of individual agents (the boids, in this case) adhering to a set of simple rules. The rules applied in the simplest Boids world are as follows: separation: steer to avoid crowding local flockmates alignment: steer towards the average heading of local flockmates cohesion: steer to move toward the average position (center of mass) of local flockmates More complex rules can be added, such as obstacle avoidance and goal seeking.

## Turbulence

Create a random turbulence effect with a 3d noise.

**Size** Indicates the scale of the noise.

**Global** Makes the size and strength of the noise relative to the world, instead of the object it is attached to.

## Drag

Drag is a force that works to resist particle motion by slowing it down.

**Linear** Drag component proportional to velocity.

**Quadratic** Drag component proportional to the square of the velocity.

## Links

- [Wind & Deflector force update 2.48](#)

## 2.7.11 Converting Game Engine Physics

Sometimes, you may want to animate a wall being broken down by an object, or a bunch of objects collapsing, falling, or bouncing with accurate physics. You could manually insert keyframes and do trial and error adjusting with F-Curves to simulate physics and acceleration, or, you can do it much easier and automatically by taking advantage of Blender Game Engine Physics. Blender now has a feature which allows you to record animation in a blender game and turn it into Blender Animation Keyframes.

Animation can be recorded by going `Game>Record Animation`. The animation can then be recorded with `Alt-A`

If you just want a static pile of stuff, you can move to the last frame, and delete all the keyframes quickly by turning them into NLAs and deleting.

## 2.7.12 Performance

### OpenMP (Mac OSX)

How to use the distributed applescript to optimize simulation performance on OSX

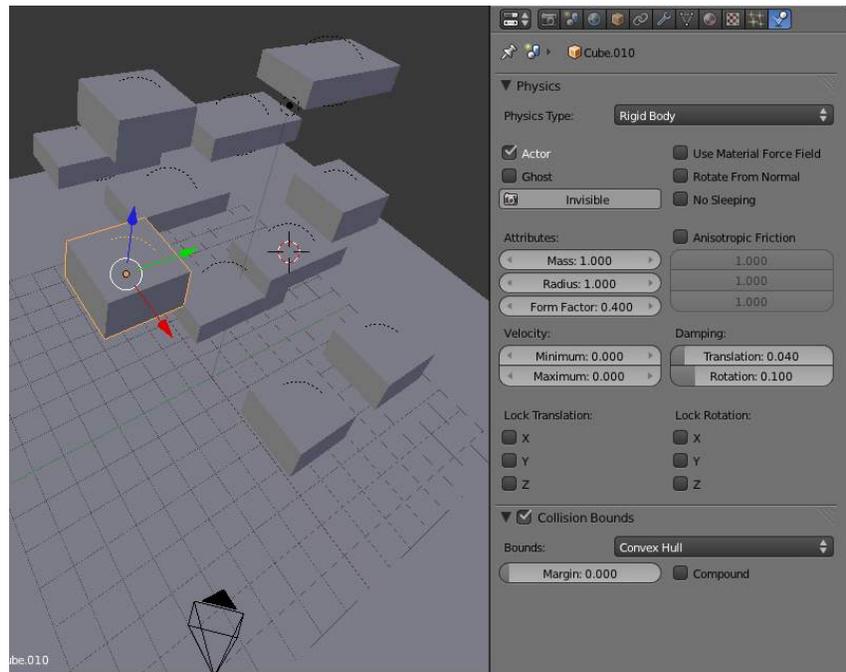


Fig. 2.1614: Some Blocks about to fall

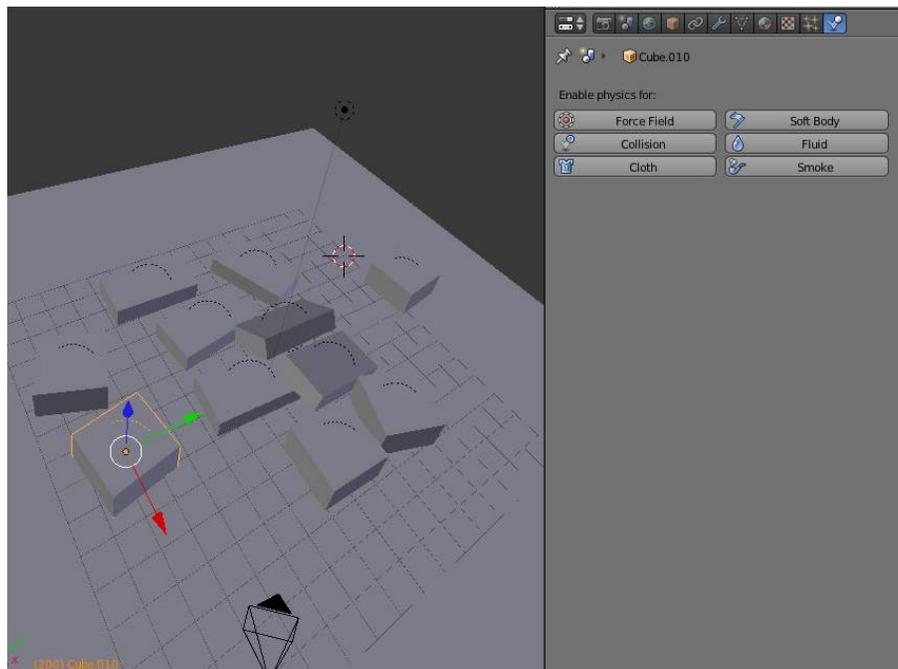


Fig. 2.1615: A Pile

### Suboptimal baking performance

Often you will encounter suboptimal baking performance with openMP (OMP) multithreaded simulations, especially fluids.

This is due how the domain splitting distributes threads to cells which are sometimes not “filled” whereas calculations, resulting in lot of threads not giving any speedboost. This is almost dependent on the resolution the simulation and object density.

If you have such an “undersaturated” simulation, it helps a lot to just reduce the OMP threads to a low number instead of letting OMP just use all available cores.

### Solution

For OSX openMP-enabled Blender you can now use a delivered applescript to tune the OMP-threads used. This makes usage of the terminal obsolete.

The default is for now set to 4 cores.

If you leave the input field empty, the script gets the corecount of your logical cpu-cores (including HyperTrading) This is the same what openMP would pull without the environment variable set.

### Steps to use the applescript

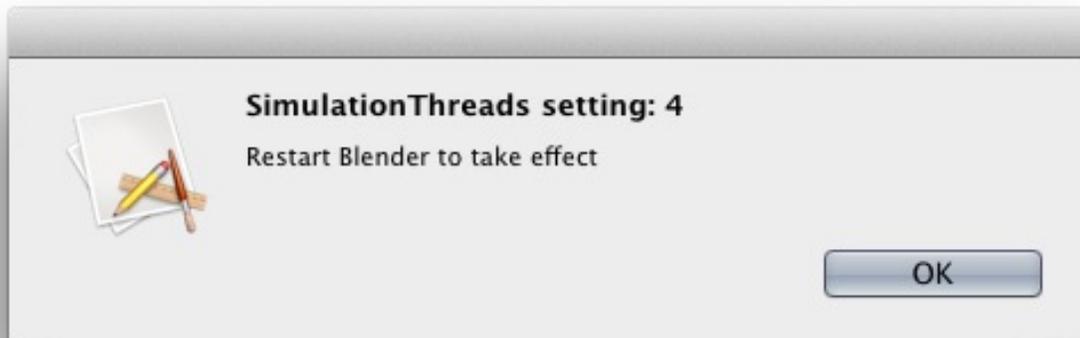
- In your Blenderfolder open the “set\_simulation\_threads” applescript



- Set the threadcount you want to use (leave empty to let OMP get all available cores)



- Press o.k. to set the new value, a messagebox will show you the new setting accepted.



- Close and reopen Blender to take over the setting.
- Watch your baking progress or simulation framerate and perhaps repeat steps 1- 4 to get the optimal value.

## 2.8 Render

### 2.8.1 Introduction

---

#### Reference

Editor: Properties

Context: Render

Hotkey: F12

---

Rendering is the process of creating a 2D image (or video) from your 3D scene. What that image looks like is based on four factors which the user can control:

- A [Camera](#)
- The [Lighting](#) in your scene
- The [Material](#) of each object
- Various render settings (quality, image size, layers etc)

Your computer will perform various complex calculations based on those factors in order to give you your rendered image. This process may take some time depending on the complexity of the scene and your hardware.

Once the render is complete, it is possible to do additional manipulation of the image, called [Post Processing](#).

Finally, the output can be saved to an image or video file using one of the [output formats](#).

## Workflow

In general, the process for rendering is:

1. Position the camera
2. Light the scene
3. Setup materials
4. Render a test image using lower quality settings
5. Change or fix anything you noticed in the render
6. Repeat the above two steps until you are satisfied
7. Render a high quality image, change or fix any issues and repeat until satisfied
8. Save your image to a file, or render the animation to a video or image sequence.

## Render Engines

The Render Engine is the set of code which controls how your materials and lighting are used, and ultimately what the rendered image looks like.

Some engines may be better at certain things than others due to the math they use or core principles around which they were written.

Blender includes two render engines by default:

- [Blender Render](#)
- [Cycles](#)

More render engines from third-party developers can also be added using [Add-ons](#)

## 2.8.2 Blender Render Engine

### Materials

#### Introduction to Materials

A material defines the artistic qualities of the substance that an object is made of. In its simplest form, you can use materials to show the substance an object is made of, or to “paint” the object with different colors. Usually, the substance is represented by its surface qualities (color, shininess, reflectance, etc.) but it can also exhibit more complicated effects such as transparency, diffraction and sub-surface scattering. Typical materials might be brass, skin, glass, or linen.

The basic (un-textured) Blender material is uniform across each face of an object (although the various pixels of each face of the object may appear differently because of lighting effects). However, different faces of the object may use different materials (see [Multiple Materials](#)).

In Blender, materials can (optionally) have associated textures. Textures *describe* the substance: e.g. *polished* brass, *dirty* glass or *embroidered* linen. The [Textures](#) chapter describes how to add textures to materials.

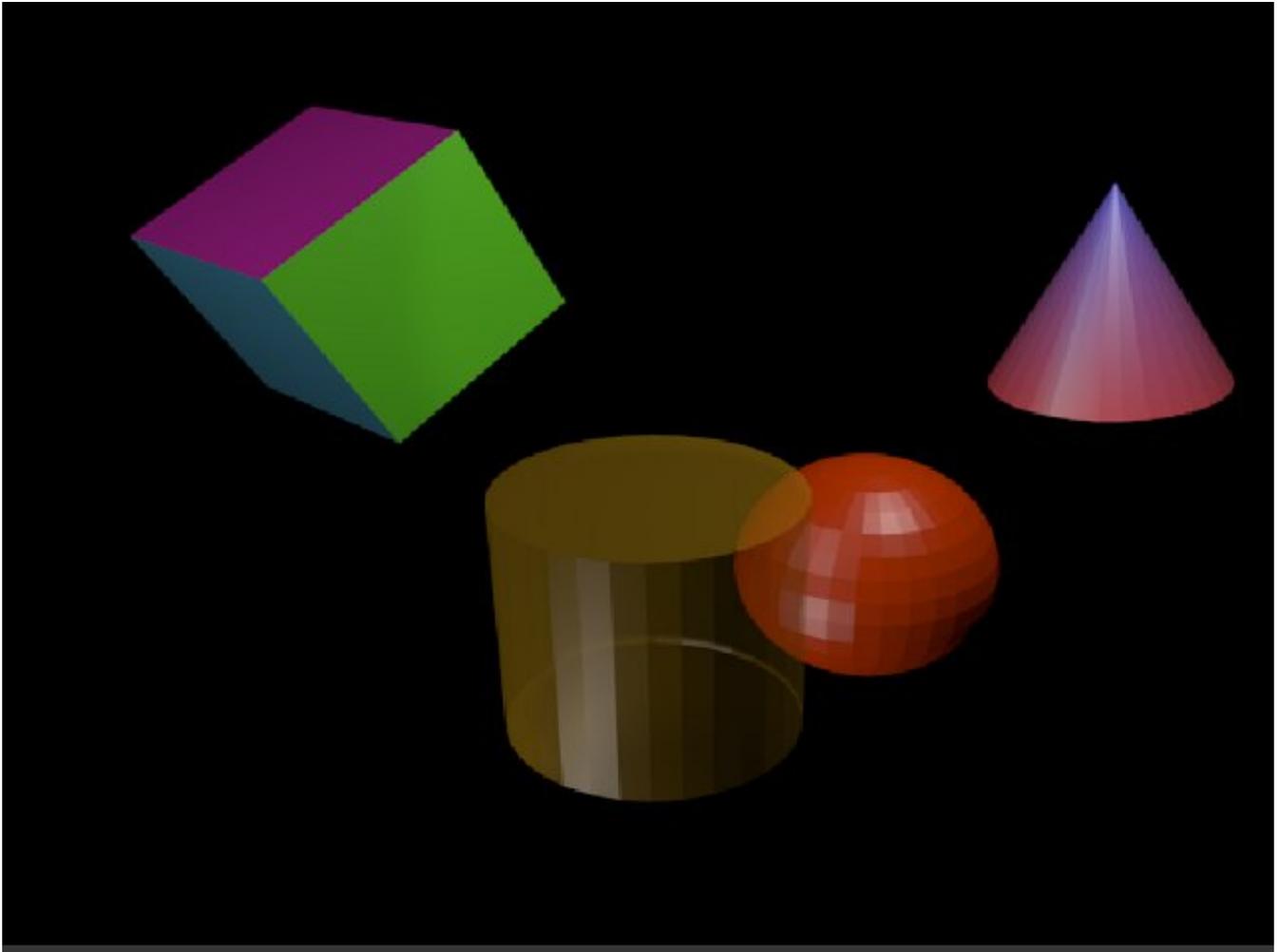


Fig. 2.1616: Various basic materials (single, multiple material, transparency, vertex paint).

**How Materials Work** Before you can understand how to design effectively with materials, you must understand how simulated light and surfaces interact in Blender’s rendering engine and how material settings control those interactions. A deep understanding of the engine will help you to get the most from it.

The rendered image you create with Blender is a projection of the scene onto an imaginary surface called the *viewing plane*. The viewing plane is analogous to the film in a traditional camera, or the rods and cones in the human eye, except that it receives simulated light, not real light.

To render an image of a scene we must first determine what light from the scene is arriving at each point on the viewing plane. The best way to answer this question is to follow a straight line (the simulated light ray) backwards through that point on the viewing plane and the focal point (the location of the camera) until it hits a renderable surface in the scene, at which point we can determine what light would strike that point.

The surface properties and incident light angle tell us how much of that light would be reflected back along the incident viewing angle (*Rendering engine basic principle*).

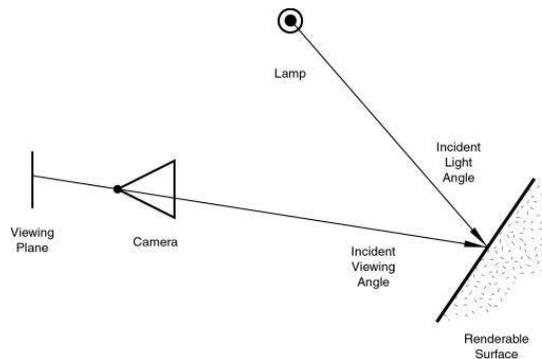


Fig. 2.1617: Rendering engine basic principle.

Two basic types of phenomena take place at any point on a surface when a light ray strikes it: diffusion and specular reflection. Diffusion and specular reflection are distinguished from each other mainly by the relationship between the incident light angle and the reflected light angle.

The shading (or coloring) of the object during render will then take into account the base color (as modified by the diffusion and specular reflection phenomenon) and the light intensity.

Using the internal ray tracer, other (more advanced) phenomena could occur. In ray-traced reflections, the point of a surface struck by a light ray will return the color of its surrounding environment, according to the rate of reflection of the material (mixing the base color and the surrounding environment’s) and the viewing angle.

On the other hand, in ray-traced refractions, the point of a surface struck by a light ray will return the color of its background environment, according to the rate of transparency (mixing the base color and the background environment’s along with its optional filtering value) of the material and the optional index of refraction of the material, which will distort the viewing angle.

Of course, shading of the object hit by a light ray will be about mixing all these phenomena at the same time during the rendering. The appearance of the object, when rendered, depends on many inter-related settings:

- World (Ambient color, Radiosity, Ambient Occlusion)
- Lights
- Material settings (including ambient, emission, and every other setting on every panel in that context)
- Texture(s) and how they are mixed
- Material Nodes
- Camera

- Viewing angle
- Obstructions and transparent occlusions
- Shadows from other opaque/transparent objects
- Render settings
- Object dimensions (SS settings are relevant to dimensions)
- Object shape (refractions, fresnel effects)

### Using Materials

**Tip:** Check your Render

When designing materials (and textures and lighting), frequently check the rendered appearance of your scene, using your chosen render engine/shader settings. The appearance might be quite different from that shown in the texture display in the 3D panel.

As stated above, the material settings usually determine the surface properties of the object. There are several ways in which materials can be set up in Blender. Generally speaking, these are not compatible - you must choose which method you are going to use for each particular object in your scene.

First, you can set the [Properties](#) in the various Material panels.

Second, you can use [Nodes](#); a graphical nodes editor is available.

Last, you can directly set the color of object surfaces using various special effects. Strictly speaking, these are not materials at all, but they are included here because they affect the appearance of your objects. These include [Vertex Painting](#), [Wire Rendering](#), [Volume Rendering](#), and [Halo Rendering](#).

The exact effect of Material settings can be affected by a number of system settings. First and foremost is the Render Engine used - Cycles and the Blender Render Engine (aka Blender Internal or BI) require quite different illumination levels to achieve similar results, and even then the appearance of objects can be quite different. Also, the material properties settings can be affected by the texture method used (Single Texture, Multitexture or GLSL). So it is recommended to always select the appropriate system settings before starting the design of materials.

### Assigning a Material

Materials available in the currently-open Blender file can be investigated by clicking on the Materials button



Fig. 2.1618: in the Properties Window Header. In this section we look at how to assign or remove a material to/from the Active Object in Blender, either by:

- creating a new material,
- re-using an existing material, or
- deleting a material.

We also give hints about practical material usage.

**Creating a new Material** Every time a new Object is created it has no material linked to it. You can create a new material for the object by

- Selecting the object

- In the Properties window, click on the object button
- Click on the Materials button in the Properties Panel Header (1)

The Shading context window then appears. This contains the following elements:

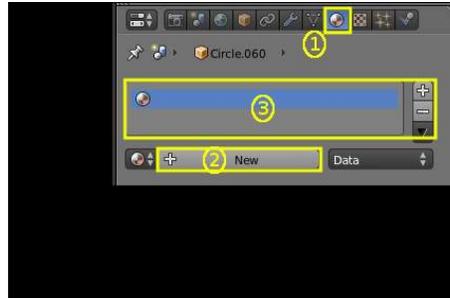


Fig. 2.1619: Add new material

- Context - The currently-selected scene and object
- Object Material Slots (3) - this window shows the “slots” for the material (or materials) that this object data contains.
- Active Material (2). Initially empty, asking for “New”.

To add a new material, click “+” in the Active Material box. This action has a series of effects:

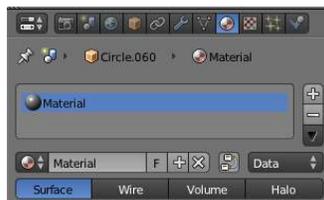


Fig. 2.1620: Materials Panel with New Entry

- opens the new material in the Active Material box,
- brings up additional buttons in the immediate panel,
- adds the new material to the Available Materials list,
- adds the new material to the Object Material Slots list for the active object (or its object data - see below)
- brings up a [preview](#) of the new material,
- provides you with a range of panels allowing you to select the [properties](#) of the new material.

**New Material Panel Buttons** Details of the additional buttons which appear in the Material panel for a new Active Material are as follows:

#### Active Material



Fig. 2.1621: Available Materials See Reusing Existing Materials below.

**Name** Like other datablocks, Blender will automatically set the name of the new material to `Material`, `Material.001` and so on. You can change this by over-typing with your own choice of name.

**Number of Users** Specifies the number of meshes which use this material.

**F - Fake User** When enabled, this material will always be saved within the Blender file, even if it has no meshes which use it (see [Deleting a Material](#)).

**X** Delete this material (see [Deleting a Material](#)).

---

**Tip:** Naming materials

It's a very good idea to give your materials clear names so you can keep track of them, especially when they're linked to multiple objects. Try to make your names descriptive of the material, not its function (e.g. "Yellow Painted" rather than "Kitchen Table Color")

---

**Nodes**



Fig. 2.1622: If dark, use the Shader Nodes to generate the material.

**Data** Specifies whether the material is to be linked to the Object or to the Object Data.



Fig. 2.1623: Link material to object or to object's data

The Link pop-up menu has two choices, Data and Object. These two menu choices determine whether the material is linked to the object or to the data, (in this case) the mesh (or curve, nurbs, etc.). The Data menu item determines that this material will be linked to the mesh's datablock which is linked to the object's datablock. The Object menu item determines that the material will be linked to the object's data block directly. This has consequences of course. For example, different objects may share the same mesh datablock. Since this datablock defines the shape of the object, any change in edit mode will be reflected on all of those objects. Moreover, anything linked to that mesh datablock will be shared by every object that shares that mesh. So, if the material is linked to the mesh, every object will share it. On the other hand, if the material is linked directly to the object datablock, the objects can have different materials and still share the same mesh. Short explanation: If connected to the object, you can have several instances of the same obData using different materials. If linked to mesh data, you can't. See [Data System](#) for more information.

**Object Render Format (menu)** This menu has four options which define how the object is to be rendered:

**Surface** Material applied to object planes.

**Wire** Material applied to wires following the object edges

**Volume** Material applied to the object volume.

**Halos** Material applied to halos around each object vertex.

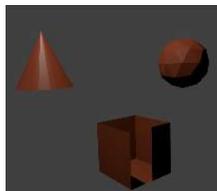


Fig. 2.1624: Surface

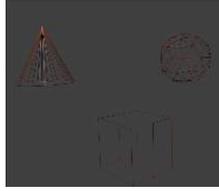


Fig. 2.1625: Wire



Fig. 2.1626: Volume

**Reusing Existing Materials** Blender is built to allow you to reuse *anything*, including material settings, between many objects. Instead of creating duplicate materials, you can simply re-use an existing material. There are several ways to do this using the Available Materials menu:

*Single Object* - With the object selected, click the sphere located to the left of the Material name. A drop-down list appears showing all the materials available in the current Blender file. To use one, just click on it.

---

**Tip:** Searching for Materials

The search field at the bottom of the material list allows you to search the names in the list. For example, by entering “wood” all existent materials are filtered so that only materials containing “wood” are displayed in the list.

*Multiple Objects* - In the 3D View, with `Ctrl-L` you can quickly link all selected objects to the material (and other aspects) of the active object. Very useful if you need to set a large number of objects to the same material; just select all of them, then the object that has the desired material, and `Ctrl-L` link them to that “parent”. (See Tip on Linking Data in Creating about data linking.)

---

**Deleting a Material** To delete a material, select the material and click X in the Available Materials List entry.

Although the material will seem to disappear immediately, the Delete action can depend on how the material is used elsewhere.

If the material is linked to the Object and there are other objects which use this material, then the material will be removed from that object (but remain on all its other objects).

If the “Fake User” button (F) has been lit in the Available Materials list, then the material will be retained when the file is saved, even if it has no users.



Fig. 2.1627: Halo

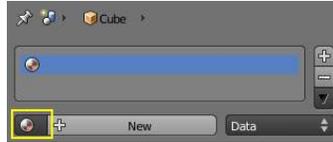


Fig. 2.1628: Select an existing material.

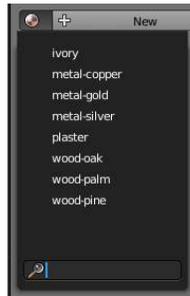


Fig. 2.1629: List of available materials

Only if it has 0 “real” users, and no “Fake” user, will the material be permanently deleted. Note that it will still remain in the Materials list until the Blender file is saved, but will have disappeared when the file is reloaded.

### Multiple Materials

Normally, different colors or patterns on an object are achieved by adding textures to your materials. However, in some applications you can obtain multiple colors on an object by assigning different materials to the individual faces of the object.

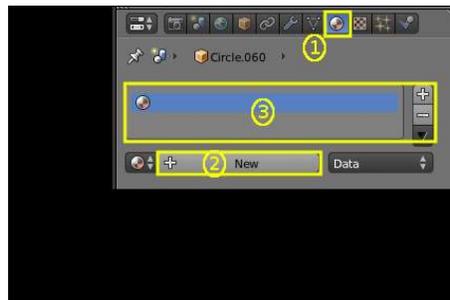


Fig. 2.1630: Add new material

To apply several materials to different faces of the same object, you use the Material Slots options (3) in the Materials header panel.

The workflow for applying a second material to some faces of an object covered by a base material is as follows:

- In Object mode, apply the base material to the whole object (as shown in [Assigning a material](#))
- Create/select the second material (the whole object will change to this new material).
- In the Active Material box (2), re-select the base material.
- Go to Edit Mode - Face Select (a new box appears above the Active Material box with Assign/Select/Deselect).
- Select the face/faces to be colored with the second material.

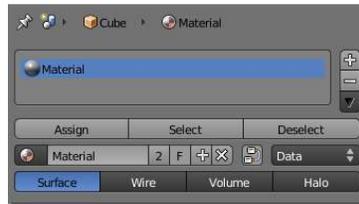


Fig. 2.1631: Material menu in edit mode

- In the Object Material Slots box (3), click the **P** L U S to create a new slot, and while this is still active, click on the second material in the Available Materials list.
- Click the Assign button, and the second material will appear on the selected object faces.
- You can also make this new material a copy of an existing material by adding the data block:

Select object, get the material, (R Click) - Copy data to clipboard. When you have renamed the material, click “Data - Data” to link to the existing material. Proceed to assign faces as required. NB: If you change the material on the original object, the new object color changes too.

## Introduction to Properties

**Material Properties** Materials can have a wide array of properties. It is the combination of all of these things that define the way a material looks, and how objects using that material will appear when rendered. These properties are set using the various Properties panels.

Remember that the appearance of your materials are affected by the way that they are rendered (surface, wire, volume or halo), and by the rendering engine (Blender, Cycles, or Game) used. Most properties for images rendered using Cycles can only be controlled using the Node system.

The list below sets out the various Properties panels available in Blender Render and Game Engine, and brief details of their scope. Details of their controls and settings are given on the relevant pages.

**Preview** A preview of the current material mapped on to one of several basic objects.

**Diffuse Shaders** The basic color of the material, together with different models for dispersion.

**Specular Shaders** The reflected highlights: color, strength and different models for dispersion.

**Color Ramps** How to vary the base color over a surface in both Diffuse and Specular shaders.

**Shading** Properties of various characteristics of the shading model for the material.

**Transparency** Can other objects be seen through the object, and if so, how?

**Mirror** (Only Blender Render): Reflective properties of the material.

**SubSurface Scattering** (Only Blender Render): Simulates semi-translucent objects in which light enters, bounces around, then exits in a different place.

**Strand** (Only Blender Render): For use when surfaces are covered with hair, fur, etc.

**Options** Various options for shading and coloring the object.

**Shadow:** Controls how objects using this material cast and receive shadows.

**Game Settings** (Only Blender Render): Controls settings for real-time rendering of Game Engine objects.

**Physics** (??)

---

## Material Preview

---

### Reference

Mode: All Modes

Panel: Shading/Material Context → Preview

---

The Preview panel gives a quick visualization of the active material and its properties, including its *Shaders*, *Ramps*, *Mirror Transp* properties and *Textures*. It provides several shapes that are very useful for designing new shaders: for some shaders (like those based on *Ramp* colors, or a Diffuse shader like *Minnaert*), one needs fairly complex or specific previewing shapes to decide if the shader being designed achieves its goal.

### Options

**Flat XY plane** Useful for previewing textures and materials of flat objects, like walls, paper and such.

**Sphere** Useful for previewing textures and materials of sphere-like objects, but also to design metals and other reflective/transparent materials, thanks to the checkered background.

**Cube** Useful for previewing textures and materials of cube-like objects, but also to design procedural textures. Features a checkered background.

**Monkey** Useful for previewing textures and materials of organic or complex non-primitive shapes. Features a checkered background.

**Hair strands** Useful for previewing textures and materials of strand-like objects, like grass, fur, feathers and hair. Features a checkered background.

**Large Sphere with Sky** Useful for previewing textures and materials of sphere-like objects, but also to design metals and other reflective materials, thanks to the gradient Sky background.

Preview uses OSA (oversampling). Whatever the preview option, it will make use of OSA (oversampling) in order to provide better quality. Disable this option if your computer is already slow or old.

### Examples




---

## Diffuse Shaders

---

### Reference

Mode: All Modes

Panel: Shading/Material Context → Diffuse

---

A diffuse shader determines, simply speaking, the general color of a material when light shines on it. Most shaders that are designed to mimic reality give a smooth falloff from bright to dark from the point of the strongest illumination to the shadowed areas, but Blender also has other shaders for various special effects.

**Common Options** All diffuse shaders have the following options:

**Color** Select the base *diffuse color* of the material.

**Intensity** The shader's brightness, or more accurately, the amount of incident light energy that is actually diffusely reflected towards the camera.

**Ramp** Allows you to set a range of colors for the *Material*, and define how the range will vary over a surface. See [Color Ramps](#) for details.

**Technical Details** Light striking a surface and then re-irradiated via a Diffusion phenomenon will be scattered, i.e., re-irradiated in all directions isotropically. This means that the camera will see the same amount of light from that surface point no matter what the *incident viewing angle* is. It is this quality that makes diffuse light *viewpoint independent*. Of course, the amount of light that strikes the surface depends on the incident light angle. If most of the light striking a surface is reflected diffusely, the surface will have a matte appearance (*Light re-irradiated in the diffusion phenomenon.*).

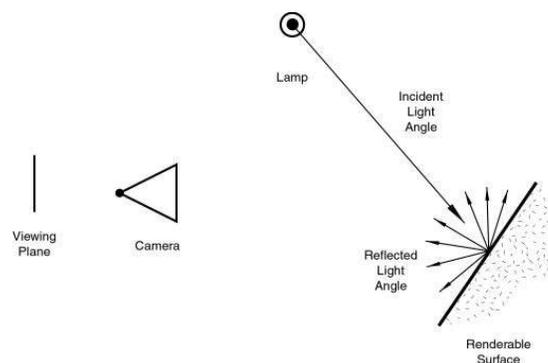


Fig. 2.1644: Light re-irradiated in the diffusion phenomenon.

---

**Tip:** Shader Names

Some shaders' names may sound odd - they are traditionally named after the people who first introduced the models on which they are based.

---

**Lambert Reference**

---

Mode: All Modes

Panel: Shading/Material Context → Shaders

---

This is Blender's default diffuse shader, and is a good general all-around workhorse for materials showing low levels of specular reflection.

**Johann Heinrich Lambert (1728-1777)** was a Swiss mathematician, physicist and astronomer who published works on the reflection of light, most notably the [Beer-Lambert Law](#) which formulates the law of light absorption.

This shader has only the default option, determining how much of available light is reflected. Default is 0.8, to allow other objects to be brighter.

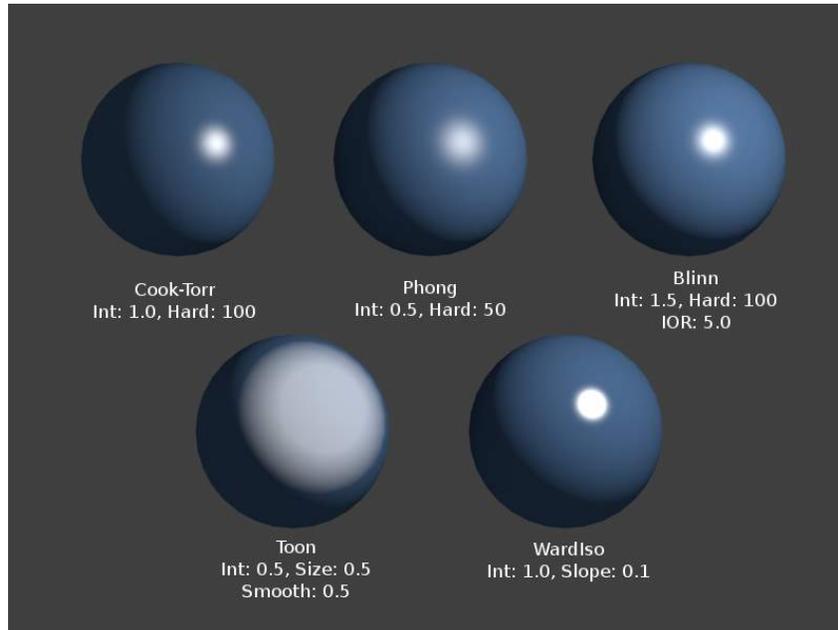


Fig. 2.1645: Lambert Shader



Fig. 2.1646: The Lambert diffuse shader settings.

## Oren-Nayar Reference

Mode: All Modes

Panel: Shading/Material Context → Shaders

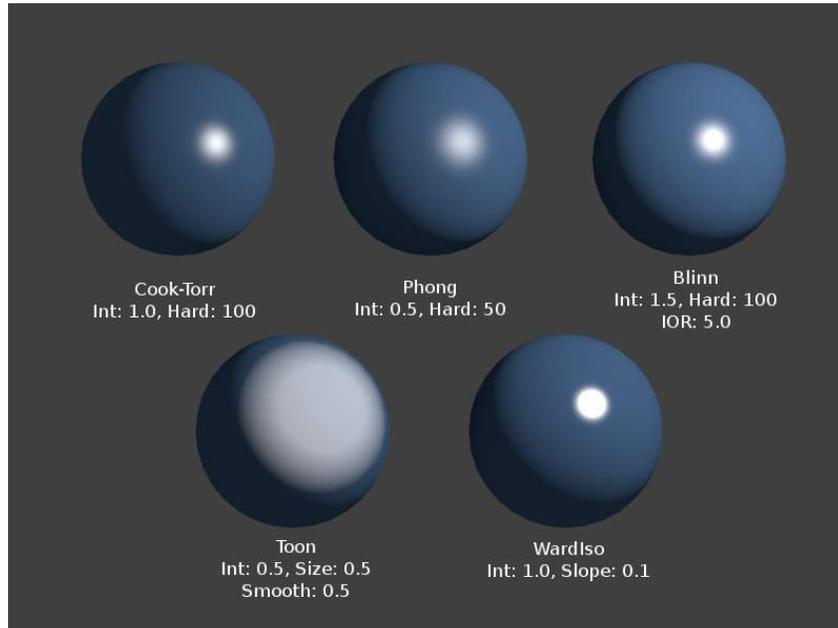


Fig. 2.1647: Oren-Nayar Shader

Oren-Nayar takes a somewhat more ‘physical’ approach to the diffusion phenomena as it takes into account the amount of microscopic roughness of the surface. [Michael Oren](#) and [Shree K. Nayar](#) Their [reflectance model](#), developed in the early 1990s, is a generalization of Lambert’s law now widely used in computer graphics.

### Options

**Roughness** The roughness of the surface, and hence, the amount of diffuse scattering.



Fig. 2.1648: The Oren-Nayar diffuse shader settings.

Toon

## Reference

Mode: All Modes

Panel: Shading/Material Context → Shaders

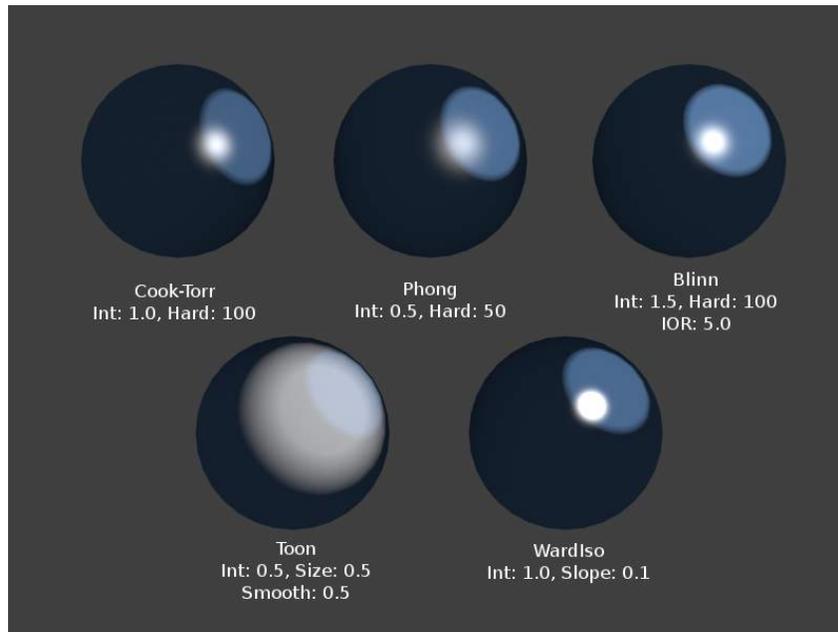


Fig. 2.1649: Toon Shader, Different Spec

The Toon shader is a very ‘un-physical’ shader in that it is not meant to fake reality but to produce cartoon cel styled rendering, with clear boundaries between light and shadow and uniformly lit/shadowed regions.

### Options

**Size** The size of the lit area

**Smooth** The softness of the boundary between lit and shadowed areas

### Minnaert

#### Reference

Mode: All Modes

Panel: Shading/Material Context → Shaders

Minnaert works by darkening parts of the standard Lambertian shader, so if *Dark* is 1 you get exactly the Lambertian result. Higher darkness values will darken the center of an object (where it points towards the viewer). Lower darkness values will lighten the edges of the object, making it look somewhat velvet. [Marcel Minnaert](#) (1893-1970) was a Belgian astronomer interested in the effects of the atmosphere on light and images who in 1954 published a book entitled *The Nature of Light and Color in the Open Air*.

### Options

**Dark** The darkness of the ‘lit’ areas (higher) or the darkness of the edges pointing away from the light source (lower).

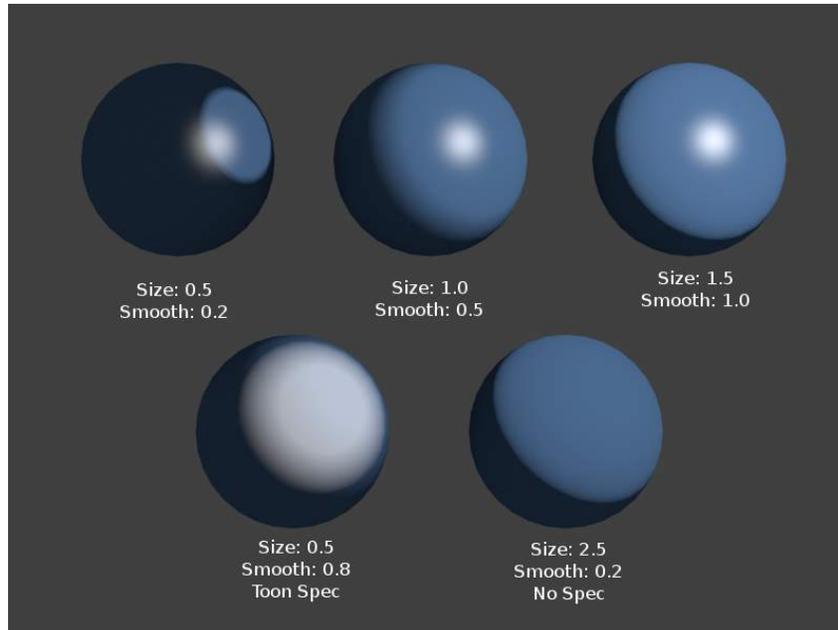


Fig. 2.1650: Toon Shader Variations



Fig. 2.1651: The Toon diffuse shader settings.

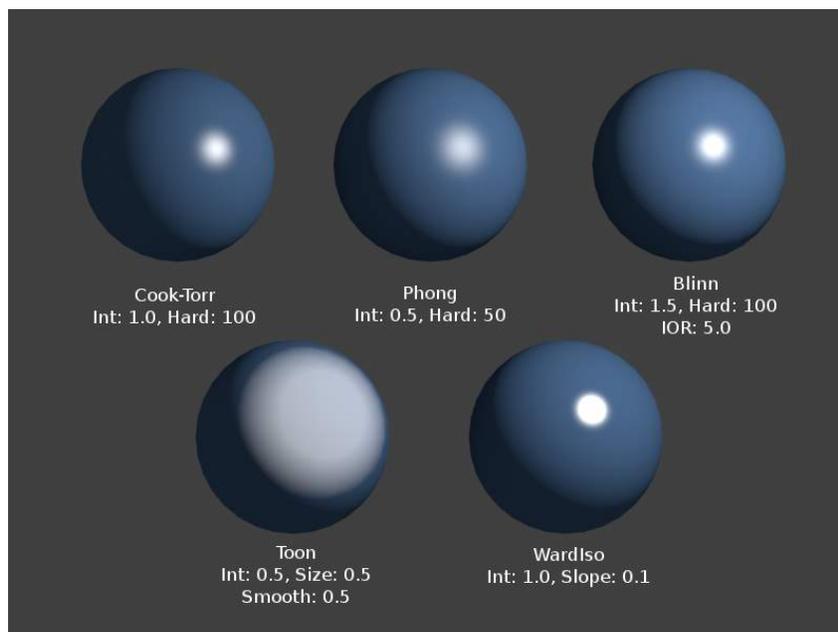


Fig. 2.1652: Minnaert Shader



Fig. 2.1653: The Minnaert diffuse shader settings.

## Fresnel Reference

Mode: All Modes

Panel: Shading/Material Context → Shaders

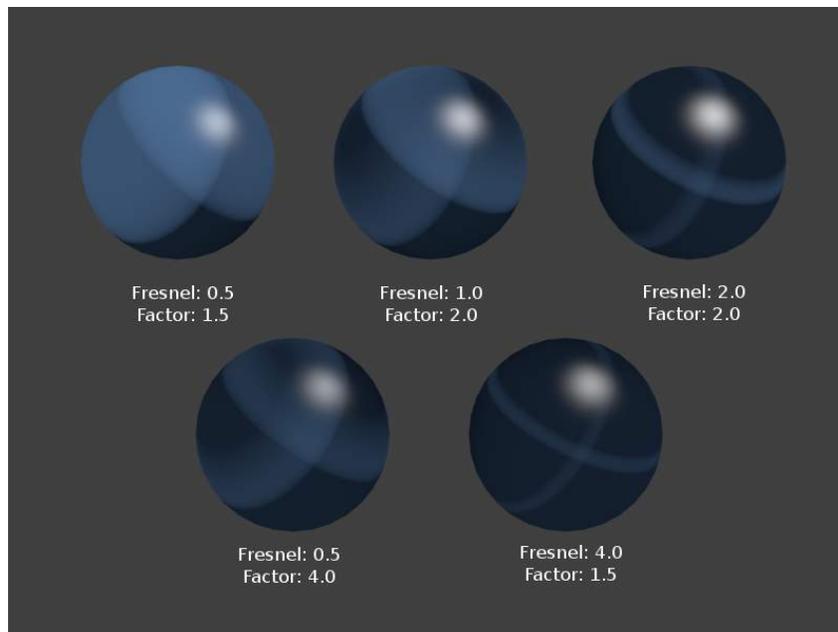


Fig. 2.1654: Various settings for the Fresnel shader, Cook-Torr Specular shader kept at Intensity 0.5, Hardness: 50

With a Fresnel shader the amount of diffuse reflected light depends on the incidence angle, i. e. from the direction of the light source. Areas pointing directly towards the light source appear darker; areas perpendicular to the incoming light become brighter. [Augustin-Jean Fresnel](#) (1788-1827) was a French physicist who contributed significantly to the establishment of the theory of wave optics.

### Options

**Fresnel** Power of the Fresnel effect, 5.0 is max.

**Factor** Blending factor of the Fresnel factor to blend in, 5.0 is max.

### Specular Shaders

## Reference

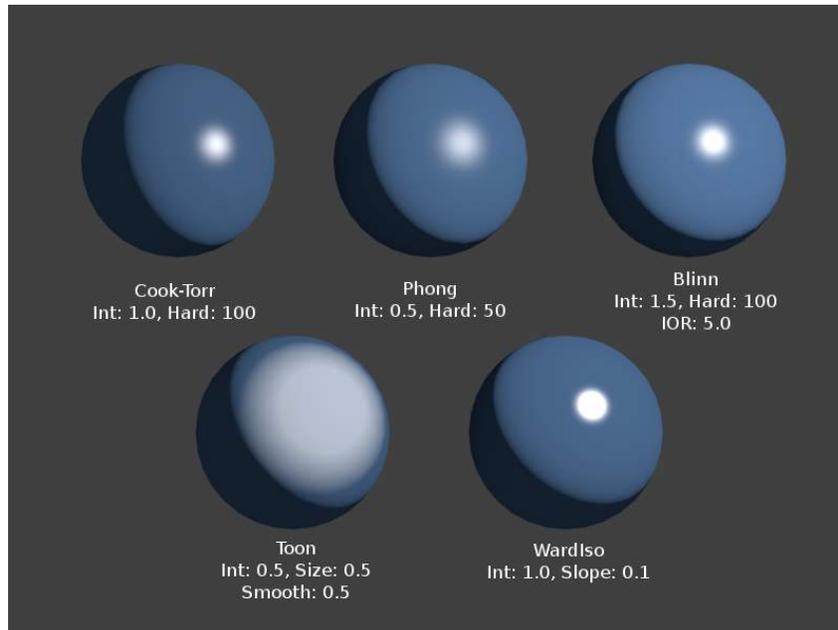


Fig. 2.1655: Fresnel Shader, Different Spec



Fig. 2.1656: The Fresnel diffuse shader settings.

Mode: All Modes

Panel: Shading/Material Context → Specular

Specular shaders create the bright highlights that one would see on a glossy surface, mimicking the reflection of light sources. Unlike [diffuse shading](#), specular reflection is *viewpoint dependent*. According to Snell's Law, light striking a specular surface will be reflected at an angle which mirrors the incident light angle (with regard to the surface's normal), which makes the viewing angle very important.

**Tip:** Not a Mirror!

It is important to stress that the *specular reflection* phenomenon discussed here is not the reflection we would see in a mirror, but rather the light highlights we would see on a glossy surface. To obtain true mirror-like reflections you would need to use the internal raytracer. Please refer to section [RENDERING](#) of this manual.

**Common Options** Each specular shader share the following common options:

**Specular Color** The color of the specular highlight

**Intensity** The intensity, or brightness of the specular highlight. This has a range of [0-1].

**Ramp** Allows you to set a range of specular colors for *Material*, and define how the range will vary over a surface. See [Ramps](#) for details.

As a result, a material has at least two different colors, a diffuse, and a specular one. The specular color is normally set to pure white (the same “pure white” as the reflected light source), but it can be set to different values for various effects (e.g. metals tend to have colored highlights).

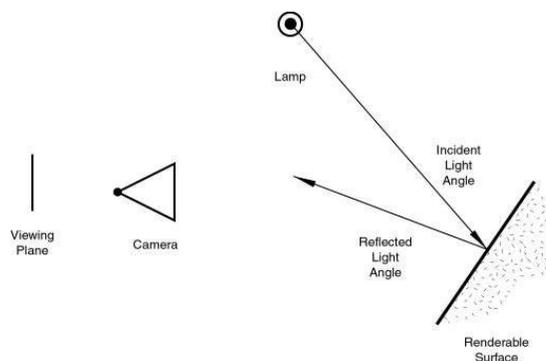


Fig. 2.1657: Specular Reflection.

**Technical Details** In reality, the quality of Diffuse and Specular reflection are generated during the same process of light scattering, but are not the same. Diffusion is actually subsurface scattering at a very small scale.

Imagine that a surface is made up of extremely microscopic semi-transparent, reflective facets. The sharpness of Specular reflection is determined by the distribution of the angle of these microfacets on the surface of an object. The more deep and jagged these facets are, the more the light spreads when it hits the surface. When these facets are flatter against the “macrosurface”, the surface will have a tighter reflection, closer to a mirror. This is a condensed explanation of the generally accepted microfacet theory of reflectance, which is the basis of all modern BRDFs (Bi-directional Reflectance Distribution Functions), or shading models.

Because these microfacets are transparent, some light that hits them travels into the surface and diffuses. The light that makes it back out is roughly Lambertian most of the time, meaning that it spreads evenly in all directions. It is also attenuated by the pigmentation in the surface, hence creating what we perceive as diffuse, and the color of an object.

Note that at glancing angles, the reflectivity of a surface will always go to 1.

If it is difficult for you to understand this relationship, try to imagine a ball (say, of centimeter scale): if you throw it against a wall of raw stones (with a scale of roughness of a decimeter), it will bounce in a different direction each time, and you will likely quickly lose it! On the other hand, if you throw it against a smooth concrete wall (with a roughness of, say, a millimeter scale), you can quite easily anticipate its bounce, which follow (more or less!) the same law as the light reflection.

---

## CookTorr Reference

Mode: All Modes

Panel: Shading/Material Context → Shaders

---

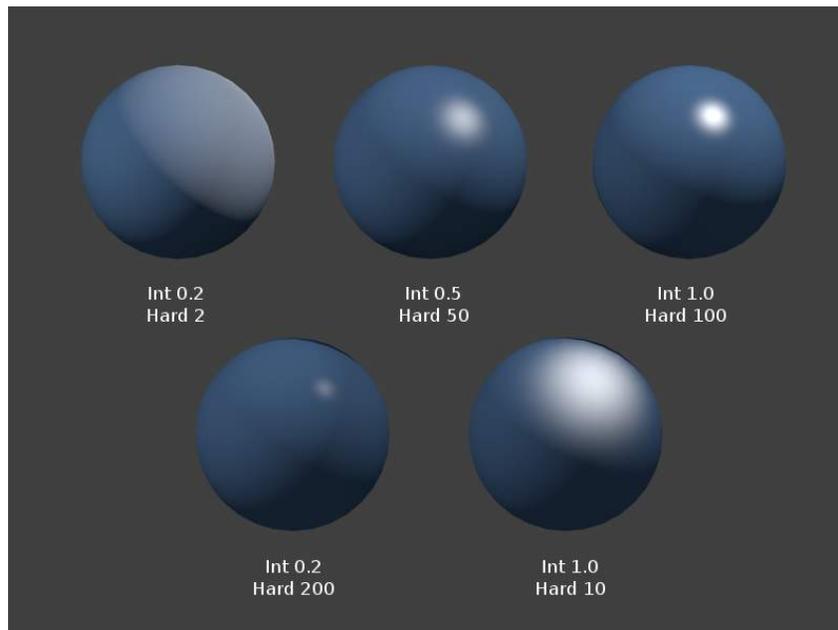


Fig. 2.1658: CookTorr Shader (Lambert 0.8)

CookTorr (Cook-Torrance) is a basic specular shader that is most useful for creating shiny plastic surfaces. It is a slightly optimized version of Phong. Robert L. Cook (LucasFilm) and Kenneth E. Torrance (Cornell University) In their 1982 paper [A Reflectance Model for Computer Graphics \(PDF\)](#), they described “a new reflectance model for rendering computer synthesized images” and applied it to the simulation of metal and plastic.

### Options

**Hardness** Size of the specular highlight

---

### Phong Reference

Mode: All Modes

Panel: Shading/Material Context → Shaders

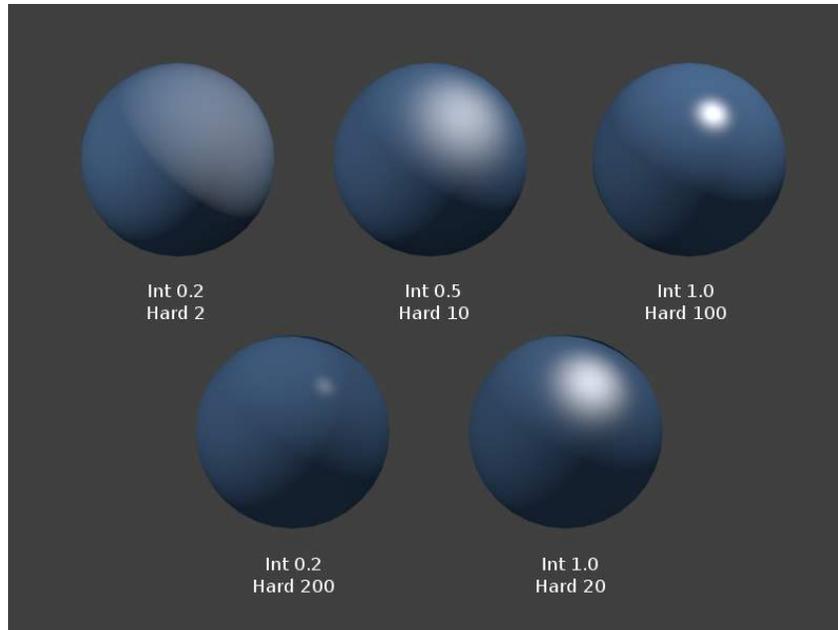


Fig. 2.1659: Phong Shader (Lambert 0.8)

Phong is a basic shader that's very similar to CookTorr, but is better for skin and organic surfaces. [Bui Tuong Phong](#) (1942-1975) was a Vietnamese-born computer graphics pioneer that developed the first algorithm for simulating specular phenomenon. His model included components not only for specular lighting, but also diffuse and ambient lighting.

### Options

**Hardness** Size of the specular highlight.

**Tip:** Planet Atmosphere

Because of its fuzziness, this shader is good for atmosphere around a planet. Add a sphere around the planet, slightly larger than the planet. For its material, use a phong specular shader. Set it to a very low alpha (.05), zero diffuse, low hardness (5) but high specularity (1).

### Blinn

#### Reference

Mode: All Modes

Panel: Shading/Material Context → Shaders

Blinn is a more 'physical' specular shader, often used with the Oren-Nayar diffuse shader. It can be more controllable because it adds a fourth option, an *index of refraction*, to the aforementioned three. [James F. Blinn](#) worked at NASA's Jet Propulsion

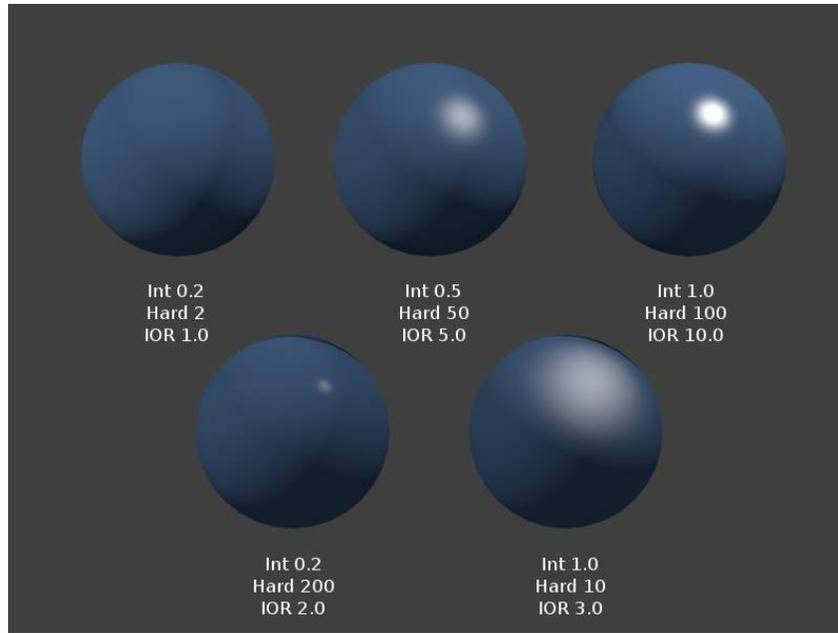


Fig. 2.1660: Blinn Shader (Oren-Nayar Int 0.8, Rough 0.5)

Laboratory and became widely known for his work on Carl Sagan's TV documentary *Cosmos*. The model he described in his 1977 paper [Models of Light Reflection for Computer Synthesized Pictures](#) (PDF) included changes in specular intensity with light direction and more accurately positioned highlights on a surface.

### Options

**Hardness** Size of the specular highlight. The Blinn shader is capable of much tighter specular highlights than Phong or CookTorr.

**IOR** 'Index of Refraction'. This parameter is not actually used to compute refraction of light rays through the material (a ray tracer is needed for that), but to correctly compute specular reflection intensity and extension via Snell's Law.

### Toon

#### Reference

Mode: All Modes

Panel: Shading/Material Context → Shaders

The Toon specular shader matches the Toon diffuse shader. It is designed to produce the sharp, uniform highlights of cartoon cels.

### Options

**Size** Size of the specular highlight.

**Smooth** Softness of the highlight's edge.

**Tip:** Alternative Method

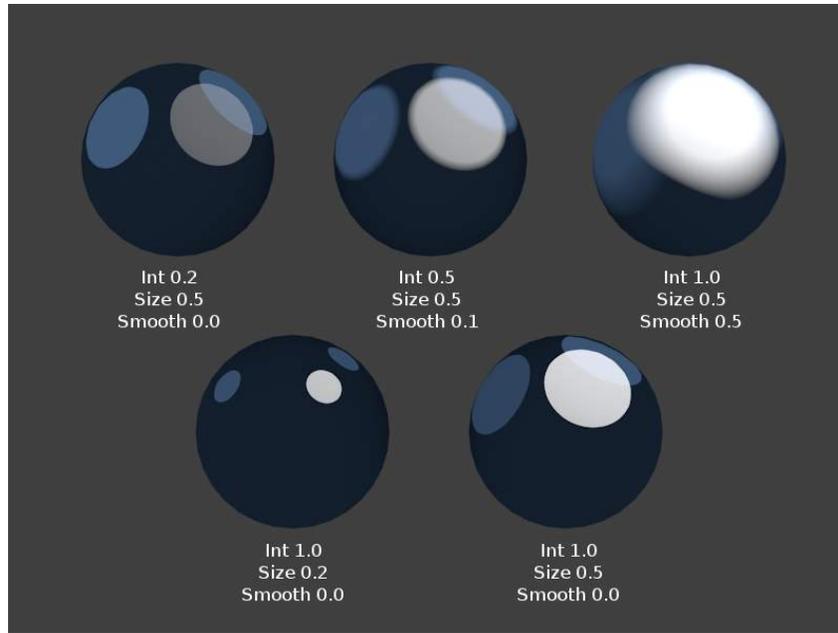


Fig. 2.1661: Toon Specular Shader (Toon Diffuse, Int 0.8, Size & Smooth match)

The Toon shader effect can also be accomplished in a more controllable way using ColorRamps.

---

## WardIso Reference

Mode: All Modes

Panel: Shading/Material Context → Shaders

---

WardIso is a flexible specular shader that can be useful for metal or plastic.

**Gregory J. Ward** developed a relatively simple model that obeyed the most basic laws of physics. In his 1992 paper, *Measuring and modeling anisotropic reflection*, Ward introduced a Bidirectional Reflectance Distribution Function (BRDF) since then widely used in computer graphics because the few parameters it uses are simple to control. His model could represent both isotropic surfaces (independent of light direction) and anisotropic surfaces (direction dependent). In Blender, the Ward specular shader is still called **Ward Isotropic** but is actually anisotropic. ([PDF](#))

### Options

**Slope** Standard deviation for of surface slope. Previously known as the [root-mean-square](#) or rms value, this parameter in effect controls the size of the specular highlight, though using a different method to that of the other specular shaders. It is capable of extremely sharp highlights.

### Color Ramps

---

### Reference

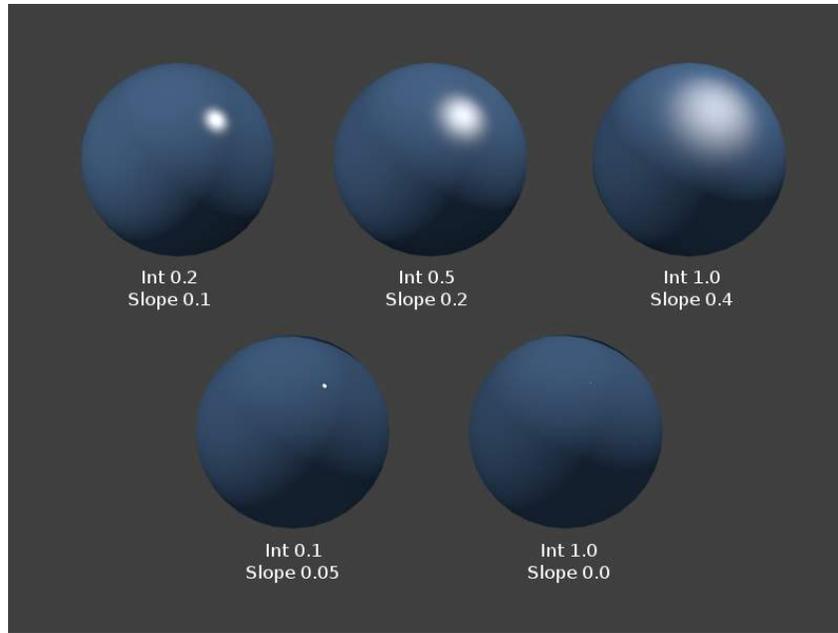


Fig. 2.1662: WardIso Shader

Mode: All Modes

Panel: Context *Shading* → sub-context *Material* → *Ramps*

In many real life situations - like skin or metals - the color of diffuse and specular reflections can differ slightly, based on the amount of energy a surface receives or on the light angle of incidence. The *Ramp Shader* options in Blender allow you to set a range of colors for a *Material*, and define how the range will vary over a surface, and how it blends with the ‘actual color’ (typically from a material or as output of a texture).

Ramps allow you to precisely control the color gradient across a material, rather than just a simple blend from a brightened color to a darkened color, from the most strongly lit area to the darkest lit area. As well as several options for controlling the gradient from lit to shadowed, ramps also provide ‘normal’ input, to define a gradient from surfaces facing the camera to surfaces facing away from the camera. This is often used for materials like some types of metallic car paint that change color based on viewing angle.

Since texture calculations in Blender happen before shading, the *Ramp Shader* can completely replace texture or material color. But by use of the mixing options and Alpha values it is possible to create an additional layer of shading in Blender materials.

**Options** The separate *Ramp* panels for the *Diffuse* shader and the *Specular* shader respectively can be toggled on and off using the

By default the Ramp panel opens with two colors; the first stop (*0*) is black and transparent (Alpha=0) and the second stop (*1*) is white and opaque (Alpha=1).

The position of the color stop markers can be altered by either (1) dragging the stop marker in the colorband or (2) by changing the *Pos* value in the

Color and alpha values for each marker can be set by clicking the

**Input** The input menu contains the following options for defining the gradient:

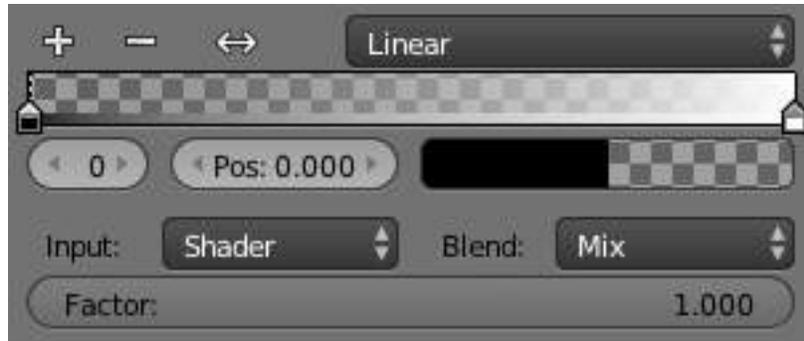


Fig. 2.1663: Ramps Panel



Fig. 2.1664: button.

**Shader** The value as delivered by the material's shader (*Lambert*, *CookTorr*) defines the color. Here the amount of light doesn't matter for color, only the direction of the light.

**Energy** As *Shader*, now also lamp energy, color, and distance are taken into account. This makes the material change color when more light shines on it.

**Normal** The surface normal, relative to the camera, is used for the *Ramp Shader*. This is possible with a texture as well, but added for convenience.

**Result** While all three previous options work per lamp, this option only works after shading calculations. This allows full control over the entire shading, including 'Toon' style results. Using alpha values here is most useful for tweaking a finishing touch to a material.

**Blend** A list of the various [blending modes](#) available for blending the ramp shader with the color from *Input*.

**Factor** This slider denotes the overall factor of the ramp shader with the color from *Input*.

## Colorbands

### Reference

Mode: All Modes

Panel: Context *Shading* → sub-context *Material* → *Ramps*

A colorband can contain a gradient through a sequence of many colors (with alpha), each color acting across a certain position in the spectrum. Colorbands are used in both materials and textures, as well in other places where a range of colors can be computed and displayed.

### Options

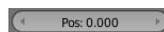


Fig. 2.1665: box.



Fig. 2.1666: box.



Fig. 2.1667: Blend pop-up menu

**Add** Add a new mark to the center of the colorband with the default color (neutral gray). New marks can also be added by `Ctrl-LMB` clicking in the colorband itself, which will add the mark at the position of the click with the same color that already exists underneath the mouse pointer.

**Delete** Remove the currently selected mark from the colorband.

**F** Flip the colorband.

**0** The number of the active mark. The values for this mark are those being displayed, and in the colorband, the active mark is displayed as a dashed line. Another marker can be selected (1) using the arrows in the



Fig. 2.1668: slider, (2) by clicking on the number being displayed and entering a number of a color mark, or (3) by LMB clicking a marker in the colorband.

**Pos** The position of the active color mark in the colorband (range 0.0–1.0). The position of the color marks can also be changed by LMB dragging them in the colorband.

---

**Note:** Reordering colors

If the position of the color marks are reordered, they will be automatically renumbered so that they always start with 0 from the left and increment to the right.

---

The *Colorswatch* right of the *Position* slider displays the color of the active mark. LMB click it to display a color picker in which values for color (*RGB*) and transparency (*Alpha*) can be set.

**Interpolation** Various modes of interpolation between marker's values can be chosen in the Interpolation menu:

**Ease** Ease by quadratic equation.

**Cardinal** Cardinal.

**Linear** Linear (default). A smooth, consistent transition between colors.

**B-Spline** B-Spline.



Fig. 2.1669: Interpolation pop-up menu

**Constant** Constant.

## Shading

In the separate *Shading* tab six more options are available:

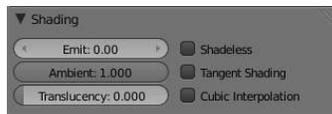


Fig. 2.1670: Shading menu, default settings

**Emit** Amount of light to emit

**Ambient** Amount of global ambient color the material receives. Each material has an *Ambient* slider that lets you choose how much ambient light that object receives. Set to 1.0 by default.

You should set this slider depending on the amount of ambient light you think the object will receive. Something deep in the cave will not get any ambient light, whereas something close to the entrance will get more. Note that you can animate this effect, to change it as the object comes out of the shadows and into the light.

Settings for *Ambient Occlusion* and *Environment Lighting* can be found in the *World* menu, with parameters affecting both these lighting components found in the *World Gather* menu.

**Translucency** Amount of diffuse shading on the back side

**Shadeless** Make this material insensitive to light or shadow; gives a solid, uniform color to the whole object.

**Tangent Shading** Use the material's tangent vector instead of the normal for shading - for anisotropic shading effects (e.g. soft hair and brushed metal). This shading was introduced in 2.42, see also settings for strand rendering in the menu further down and in the Particle System menu.

**Cubic Interpolation** Use cubic interpolation for diffuse values. Enhances the contrast between light areas and shadowed areas



## Transparency

---

## Reference

Mode: All Modes

Panel: Shading/Material Context → Transparency

---

Materials in Blender can be set to be transparent, so that light can pass through any objects using the material. Transparency is controlled using an “alpha” channel, where each pixel has an additional value, range 0-1, in addition to its RGB color values. If alpha=0, then the pixel is transparent, and the RGB values for the surface contribute nothing to the pixel’s appearance; for alpha=1, the surface is fully opaque, and the color of the surface determines the final color of the pixel.



Fig. 2.1675: Transparency Panel

In Blender, there are three ways in which the transparency of a material can be set: Mask, Z-Buffer and Ray-trace. Each of these is explained in more detail below. The [Material Preview](#) option with a sphere object gives a good demonstration of the capabilities of these three options.

**Common Options** The following property controls are available for all transparency options:

**Alpha** Sets the transparency of the material by setting all pixels in the alpha channel to the given value.

**Fresnel** Sets the power of the Fresnel effect. The Fresnel effect controls how transparent the material is, depending on the angle between the surface normal and the viewing direction. Typically, the larger the angle, the more opaque a material becomes (this generally occurs on the outline of the object).

**Specular** - Controls the alpha/falloff for the specular color.

**Blend** Controls the blending between transparent and non-transparent areas. Only used if Fresnel is greater than 0.

Mask

This option simply masks the Background. It uses the alpha channel to mix the color of each pixel on the active object plane with the color of the corresponding background pixel, according to the alpha channel of the pixel. Thus for alpha = 1, the object color is seen - the object is completely opaque; but if alpha = 0, only the background is seen - the object is transparent (but note that any other object behind the active object disappears).

This is useful for making textures of solid or semi-transparent objects from photographic reference material - a mask is made with alpha opaque for pixels within the object, and transparent for pixels outside the object.

See [Mask Transparency](#).

**Z Buffer** This uses the alpha buffer for transparent faces. The alpha value of each pixel determines the mix of the basic color of the material, and the color of the pixel is determined from the objects/background behind it. Only basic settings are available with this option; it does not calculate refractions.

**Raytraced Transparency** Uses ray tracing to calculate refractions. Ray tracing allows for complex refractions, falloff, and blurring, and is used for simulating the refraction of light rays through a transparent material, like a lens.

Note that the RayTrace option is only available in the Blender Render and Cycles render engines, but not in the Game Engine.

A ray is sent from the camera and travels through the scene until it encounters an object. If the first object hit by the ray is non-transparent, then the ray takes the color of the object.

If the object is transparent, then the ray continues its path through it to the next object, and so on, until a non-transparent object is finally encountered which gives the whole chain of rays its color. Eventually, the first transparent object inherits the colors of its background, proportional to its *Alpha* value (and the Alpha value of each transparent Material hit in between).

But while the ray travels through the transparent object, it can be deflected from its course according to the Index of Refraction (IOR) of the material. When you actually look through a plain sphere of glass, you will notice that the background is upside-down and distorted: this is all because of the Index of Refraction of glass.

---

**Note:** Enable Raytracing

To get ray-traced transparency, you need to:

- enable ray tracing in your Render settings. This is done in the Render context → Shading Panel. Ray tracing is enabled by default.
  - set your Alpha value to something other than 1.0.
  - in order for the background material to receive light passing through your transparent object, *Receive Transparent* must be turned on for that material in the Material → Shadow panel.
- 



Fig. 2.1676: The Transparency Panel.

**Options** In addition to the common options given above, the following property controls are available:

**IOR** Index of Refraction. Sets how much a ray traveling through the material will be refracted, hence producing a distorted image of its background. See *IOR values for Common Materials* below.

**Filter** Amount of filtering for transparent ray trace. The higher this value, the more the base color of the material will show. The material will still be transparent but it will start to take on the color of the material. Disabled (0.0) by default.

**Falloff** How fast light is absorbed as it passes through the material. Gives ‘depth’ and ‘thickness’ to glass.

**Limit** Materials thicker than this are not transparent. This is used to control the threshold after which the filter color starts to come into play.

**Depth** Sets the maximum number of transparent surfaces a single ray can travel through. There is no typical value. Transparent objects outside the *Depth* range will be rendered pitch black if viewed through the transparent object that the *Depth* is set for. In other words, if you notice black areas on the surface of a transparent object, the solution is probably to increase its *Depth* value (this is a common issue with ray tracing transparent objects). You may also need to turn on transparent shadows on the background object.

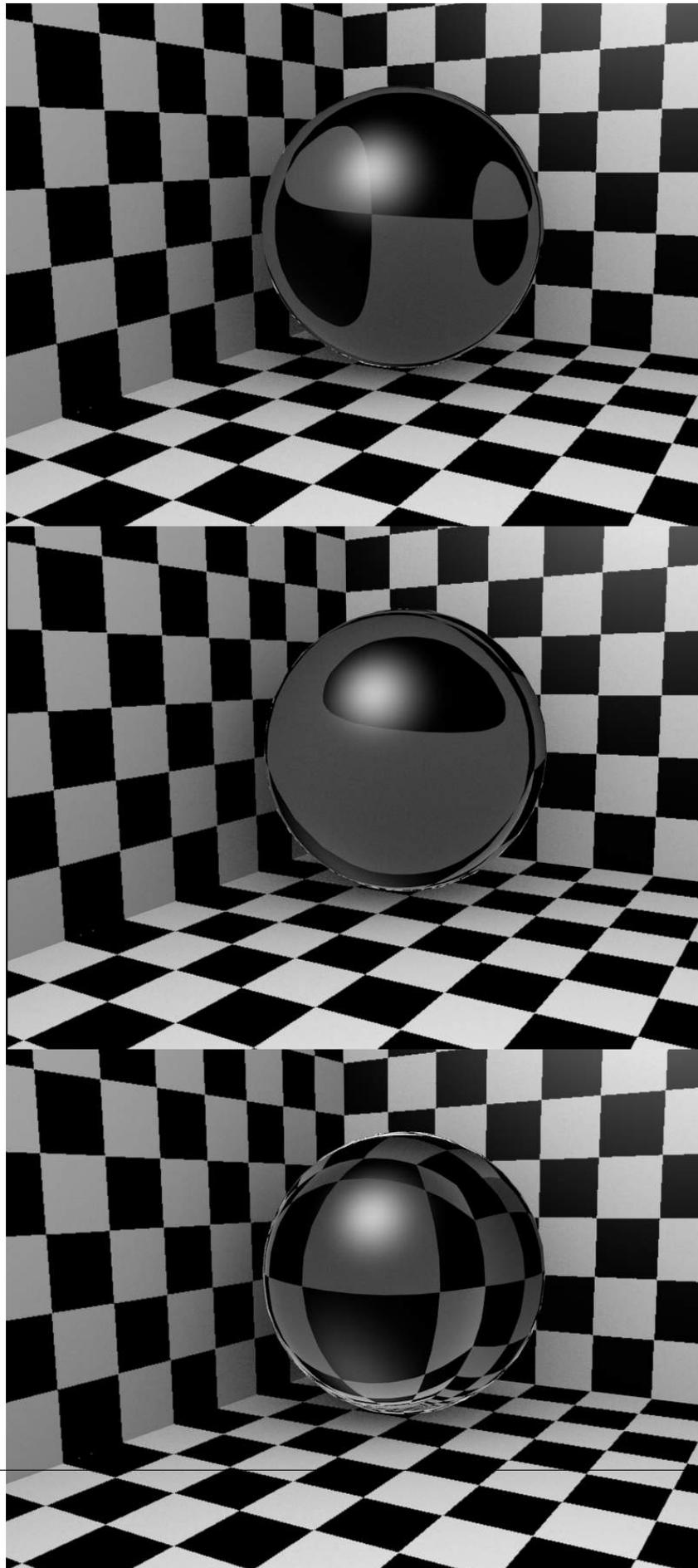
**Gloss** Settings for the glossiness of the material.

**Amount** The clarity of the refraction. Set this to something lower than zero to get a blurry refraction.

**Threshold** Threshold for adaptive sampling. If a sample contributes less than this amount (as a percentage), sampling is stopped.

**Samples** Number of cone samples averaged for blurry refraction.

## Examples



**Index of Refraction** (*Influence of the IOR of an Object on the distortion of the background: spheres of Water, Glass and Diamond (top to bottom).*). There are different values for typical materials: Air is **1.000** (no refraction), Alcohol is **1.329**, Glass is **1.517**, Plastic is **1.460**, Water is **1.333** and Diamond is **2.417**.

Fresnel	16 pieces of glass rotated in various directions demonstrate the angle-dependent Fresnel effect with ray-traced (left) and alpha buffered transparency (right). Note that the major difference is the lack of IOR effect in the latter case. (Download <a href="#">.blend</a> .)
	Settings for Fresnel using ray-traced (left) and Z transparency (right).

Note the specular highlight in the F4 glass tile (which is facing midway between the light and the camera); the Fresnel effect can be seen in row C and column 6 where the faces are turned away from the camera.

The amount of Fresnel effect can be controlled by either increasing the *Blend* value or decreasing the *Alpha* value.

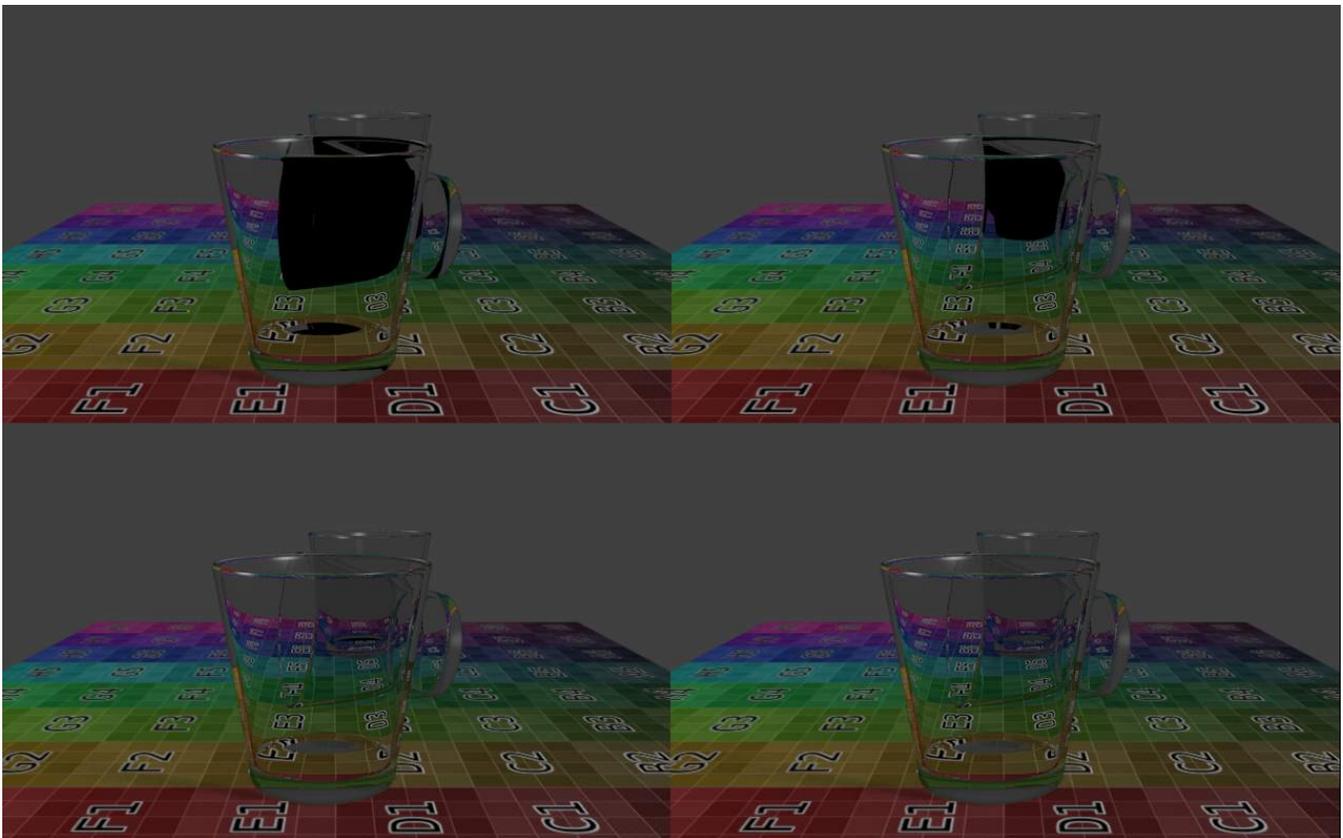


Fig. 2.1678: A simple scene with three glasses on a surface, and three lamps. Depth was set to 4, 8, 12, and 14, resulting in render times of 24 sec, 34 sec, 6 min, and 11 min respectively. (Download [.blend](#).)

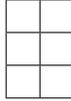
**Depth** Increasing *Depth* also considerably increases render time. Each time a light ray passes through a surface, the ray-tracing algorithm is called recursively. In the example above, each side of each glass has an exterior and an interior surface. Light rays thus have to pass through four surfaces for each glass.

But not only that, at every point on a surface, some of the light can be reflected, or mirrored off the surface in various directions. This results in multiple rays needing to be calculated for each point (often referred to as a *tree of rays*). In each of the rendered images above there are  $640 \times 400 = 256\,000$  pixels. By increasing *Depth*, at least one tree of rays is added to each pixel.

Be kind to your computer. Carefully placing objects in a scene to avoid overlapping transparent objects is often an interesting alternative.

## Hints

### Transparent shadows



By default, the shadows of transparent objects are rendered solid black, as if the object was not transparent at all. But in reality, the more transparent an object is, the lighter its shadow will be.

In Blender, transparent shadows are set on the materials that receive the shadows from the transparent object. This is enabled and disabled with the *Receive Transparent* button, in the *Material* context → *Shadow* panel. The shadow's brightness is dependent on the *Alpha* value of the shadow casting material.

Alternatives to transparent ray-traced shadows can be found in the *World* context, namely the *Ambient Occlusion*, *Environment Lighting*, and *Gather* panels. Alternatively, a texture can be used to control the *Intensity* value of the shadow-receiving material.

**IOR values for Common Materials** The following list provides some index of refraction values to use when ray-traced transparency is used for various liquids, solids (gems), and gases:

**Acetone** 1.36  
**Actinolite** 1.618  
**Agalmatolite** 1.550  
**Agate** 1.544  
**Agate** 1.540  
**Air** 1.000  
**Alcohol** 1.329  
**Alcohol, Ethyl (grain)** 1.36  
**Alexandrite** 1.745  
**Alexandrite** 1.750  
**Almandine** 1.83  
**Aluminum** 1.44  
**Amber** 1.545  
**Amblygonite** 1.611  
**Amethyst** 1.540  
**Ammolite** 1.600  
**Anatase** 2.490  
**Andalusite** 1.640  
**Anhydrite** 1.571  
**Apatite** 1.632  
**Apophyllite** 1.536  
**Aquamarine** 1.575

**Aragonite** 1.530  
**Argon** 1.000281  
**Asphalt** 1.635  
**Axinite** 1.674 - 1.704  
**Axinite** 1.675  
**Azurite** 1.730  
**Barite** 1.636  
**Barytocalcite** 1.684  
**Beer** 1.345  
**Benitoite** 1.757  
**Benzene** 1.501  
**Beryl** 1.57 - 1.60  
**Beryl, Red** 1.570 - 1.598  
**Beryllonite** 1.553  
**Brazilianite** 1.603  
**Bromine (liq)** 1.661  
**Bronze** 1.18  
**Brownite** 1.567  
**Calcite** 1.486  
**Calspar** 1.486  
**Cancrinite** 1.491  
**Carbon Dioxide (gas)** 1.000449  
**Carbon Disulfide** 1.628  
**Carbon Tetrachloride** 1.460  
**Carbonated Beverages** 1.34 - 1.356  
**Cassiterite** 1.997  
**Celestite** 1.622  
**Cerussite** 1.804  
**Ceylonite** 1.770  
**Chalcedony** 1.544 - 1.553  
**Chalk** 1.510  
**Chalybite** 1.630  
**Chlorine (gas)** 1.000768  
**Chlorine (liq)** 1.385  
**Chrome Green** 2.4  
**Chrome Red** 2.42

**Chrome Tourmaline** 1.61 - 1.64  
**Chrome Yellow** 2.31  
**Chromium** 2.97  
**Chrysoberyl** 1.745  
**Chrysoberyl, Cat's eye** 1.746 - 1.755  
**Chrysocolla** 1.500  
**Chrysoprase** 1.534  
**Citrine** 1.532 - 1.554  
**Citrine** 1.550  
**Clinohumite** 1.625 - 1.675  
**Clinozoisite** 1.724  
**Cobalt Blue** 1.74  
**Cobalt Green** 1.97  
**Cobalt Violet** 1.71  
**Colemanite** 1.586  
**Copper** 1.10  
**Copper Oxide** 2.705  
**Coral** 1.486  
**Coral** 1.486 - 1.658  
**Cordierite** 1.540  
**Corundum** 1.766  
**Cranberry Juice (25%)** 1.351  
**Crocoite** 2.310  
**Crystal** 2.000  
**Cuprite** 2.850  
**Danburite** 1.627 - 1.641  
**Danburite** 1.633  
**Diamond** 2.417  
**Diopside** 1.680  
**Dolomite** 1.503  
**Dumortierite** 1.686  
**Ebonite** 1.66  
**Ekanite** 1.600  
**Elaeolite** 1.532  
**Emerald** 1.560 - 1.605  
**Emerald Catseye** 1.560 - 1.605

**Emerald, Synth flux** 1.561  
**Emerald, Synth hydro** 1.568  
**Enstatite** 1.663  
**Epidote** 1.733  
**Ethanol** 1.36  
**Ethyl Alcohol** 1.36  
**Euclase** 1.652  
**Fabulite** 2.409  
**Feldspar, Adventurine** 1.532  
**Feldspar, Albite** 1.525  
**Feldspar, Amazonite** 1.525  
**Feldspar, Labradorite** 1.565  
**Feldspar, Microcline** 1.525  
**Feldspar, Oligoclase** 1.539  
**Flourite** 1.434  
**Formica** 1.47  
**Garnet, Andradite** 1.88 - 1.94  
**Garnet, Demantoid** 1.880 - 1.9  
**Garnet, Demantoid** 1.880  
**Garnet, Grossular** 1.738  
**Garnet, Hessonite** 1.745  
**Garnet, Mandarin** 1.790 - 1.8  
**Garnet, Pyrope** 1.73 - 1.76  
**Garnet, Rhodolite** 1.740 - 1.770  
**Garnet, Rhodolite** 1.760  
**Garnet, Spessartite** 1.810  
**Garnet, Tsavorite** 1.739 - 1.744  
**Garnet, Uvarovite** 1.74 - 1.87  
**Gaylussite** 1.517  
**Glass** 1.51714  
**Glass, Albite** 1.4890  
**Glass, Crown** 1.520  
**Glass, Crown, Zinc** 1.517  
**Glass, Flint, Dense** 1.66  
**Glass, Flint, Heaviest** 1.89  
**Glass, Flint, Heavy** 1.65548

**Glass, Flint, Lanthanum** 1.80  
**Glass, Flint, Light** 1.58038  
**Glass, Flint, Medium** 1.62725  
**Glycerine** 1.473  
**Gold** 0.47  
**Hambergite** 1.559  
**Hauyne** 1.490 – 1.505  
**Hauynite** 1.502  
**Helium** 1.000036  
**Hematite** 2.940  
**Hemimorphite** 1.614  
**Hiddenite** 1.655  
**Honey, 13% water content** 1.504  
**Honey, 17% water content** 1.494  
**Honey, 21% water content** 1.484  
**Howlite** 1.586  
**Hydrogen (gas)** 1.000140  
**Hydrogen (liq)** 1.0974  
**Hypersthene** 1.670  
**Ice** 1.309  
**Idocrase** 1.713  
**Iodine Crystal** 3.34  
**Iolite** 1.522 – 1.578  
**Iron** 1.51  
**Ivory** 1.540  
**Jade, Jadeite** 1.64 – 1.667  
**Jade, Nephrite** 1.600 – 1.641  
**Jadeite** 1.665  
**Jasper** 1.540  
**Jet** 1.660  
**Kornerupine** 1.665  
**Kunzite** 1.660 – 1.676  
**Kyanite** 1.715  
**Labradorite** 1.560 – 1.572  
**Lapis Gem** 1.500  
**Lapis Lazuli** 1.50 – 1.55

**Lazulite** 1.615  
**Lead** 2.01  
**Leucite** 1.509  
**Magnesite** 1.515  
**Malachite** 1.655  
**Meerschaum** 1.530  
**Mercury (liq)** 1.62  
**Methanol** 1.329  
**Milk** 1.35  
**Moldavite** 1.500  
**Moonstone** 1.518 – 1.526  
**Moonstone, Adularia** 1.525  
**Moonstone, Albite** 1.535  
**Morganite** 1.585 – 1.594  
**Natrolite** 1.480  
**Nephrite** 1.600  
**Nitrogen (gas)** 1.000297  
**Nitrogen (liq)** 1.2053  
**Nylon** 1.53  
**Obsidian** 1.489  
**Oil of Wintergreen** 1.536  
**Oil, Clove** 1.535  
**Oil, Lemon** 1.481  
**Oil, Neroli** 1.482  
**Oil, Orange** 1.473  
**Oil, Safflower** 1.466  
**Oil, vegetable (50- C)** 1.47  
**Olivine** 1.670  
**Onyx** 1.486  
**Opal, Black** 1.440 – 1.460  
**Opal, Fire** 1.430 – 1.460  
**Opal, White** 1.440 – 1.460  
**Oregon Sunstone** 1.560 – 1.572  
**Oxygen (gas)** 1.000276  
**Oxygen (liq)** 1.221  
**Padparadja** 1.760 – 1.773

**Painite** 1.787  
**Pearl** 1.530  
**Periclase** 1.740  
**Peridot** 1.635 – 1.690  
**Peristerite** 1.525  
**Petalite** 1.502  
**Phenakite** 1.650  
**Phosgenite** 2.117  
**Plastic** 1.460  
**Plexiglas** 1.50  
**Polystyrene** 1.55  
**Prase** 1.540  
**Prasiolite** 1.540  
**Prehnite** 1.610  
**Proustite** 2.790  
**Purpurite** 1.840  
**Pyrite** 1.810  
**Pyrope** 1.740  
**Quartz** 1.544 – 1.553  
**Quartz, Fused** 1.45843  
**Rhodizite** 1.690  
**Rhodochrisite** 1.600  
**Rhodonite** 1.735  
**Rock Salt** 1.544  
**Rubber, Natural** 1.5191  
**Ruby** 1.757 – 1.779  
**Rum, White** 1.361  
**Rutile** 2.62  
**Sanidine** 1.522  
**Sapphire** 1.757 – 1.779  
**Sapphire, Star** 1.760 – 1.773  
**Scapolite** 1.540  
**Scapolite, Yellow** 1.555  
**Scheelite** 1.920  
**Selenium, Amorphous** 2.92  
**Serpentine** 1.560

---

**Shampoo** 1.362  
**Shell** 1.530  
**Silicon** 4.24  
**Sillimanite** 1.658  
**Silver** 0.18  
**Sinhalite** 1.699  
**Smaragdite** 1.608  
**Smithsonite** 1.621  
**Sodalite** 1.483  
**Sodium Chloride** 1.544  
**Spessartite** 1.79 – 1.81  
**Sphalerite** 2.368  
**Sphene** 1.885  
**Spinel** 1.712 – 1.717  
**Spinel, Blue** 1.712 – 1.747  
**Spinel, Red** 1.708 – 1.735  
**Spodumene** 1.650  
**Star Ruby** 1.76 – 1.773  
**Staurolite** 1.739  
**Steatite** 1.539  
**Steel** 2.50  
**Stichtite** 1.520  
**Strontium Titanate** 2.410  
**Styrofoam** 1.595  
**Sugar Solution 30%** 1.38  
**Sugar Solution 80%** 1.49  
**Sulphur** 1.960  
**Synthetic Spinel** 1.730  
**Taaffeite** 1.720  
**Tantalite** 2.240  
**Tanzanite** 1.690–1.7  
**Teflon** 1.35  
**Thomsonite** 1.530  
**Tiger eye** 1.544  
**Topaz** 1.607 – 1.627  
**Topaz, Blue** 1.610

**Topaz, Imperial** 1.605 - 1.640  
**Topaz, Pink** 1.620  
**Topaz, White** 1.630  
**Topaz, Yellow** 1.620  
**Tourmaline** 1.603 - 1.655  
**Tourmaline** 1.624  
**Tourmaline, Blue** 1.61 - 1.64  
**Tourmaline, Catseye** 1.61 - 1.64  
**Tourmaline, Green** 1.61 - 1.64  
**Tourmaline, Paraiba** 1.61 - 1.65  
**Tourmaline, Red** 1.61 - 1.64  
**Tremolite** 1.600  
**Tugtupite** 1.496  
**Turpentine** 1.472  
**Turquoise** 1.610  
**Ulexite** 1.490  
**Uvarovite** 1.870  
**Wardite** 1.590  
**Variscite** 1.550  
**Water (0- C)** 1.33346  
**Water (100- C)** 1.31766  
**Water (20- C)** 1.33283  
**Water (gas)** 1.000261  
**Water (35- C, room temp)** 1.33157  
**Whisky** 1.356  
**Willemite** 1.690  
**Witherite** 1.532  
**Vivianite** 1.580  
**Vodka** 1.363  
**Wulfenite** 2.300  
**Zincite** 2.010  
**Zircon** 1.777 - 1.987  
**Zircon, High** 1.960  
**Zircon, Low** 1.800  
**Zirconia, Cubic** 2.173 - 2.21

## Mirror Reflections

Mirror reflections are computed in the Blender Render and Cycles render engines using ray tracing. (NB: Reflections are not available in the Game Engine.) Ray tracing can be used to make a material reflect its surroundings, like a mirror. The principle of ray-traced reflections is very simple: a ray is fired from the camera and travels through the scene until it encounters an object. If the first object hit by the ray is not reflective, then the ray takes the color of the object. If the object is reflective, then the ray bounces from its current location and travels up to another object, and so on, until a non-reflective object is finally met and gives the whole chain of rays its color.

Eventually, the first reflective object inherits the colors of its environment, proportional to its *Reflectivity* value. Obviously, if there are only reflective objects in the scene, then the render could last forever. This is why a mechanism for limiting the travel of a single ray is set through the *Depth* value: this parameter sets the maximum number of bounces allowed for a single ray.

---

**Note:** You need to enable ray tracing in your scene settings if you want to use ray-traced reflections. This is done in the Scene/Render context -> Render Panel. Ray tracing is enabled by default in Blender 2.37 and higher.

---

The *Color Swatch* in the mirror panel is the color of the light reflected back. Usually, for normal mirrors, use white. However, some mirrors color the reflection (e.g. metals), so you can change the color by clicking on the swatch. The amount of mirrored reflection is determined by the *Reflectivity* value. If set to something greater than 0, mirrored reflectivity will be activated and the reflection will be tinted the color set in the swatch.

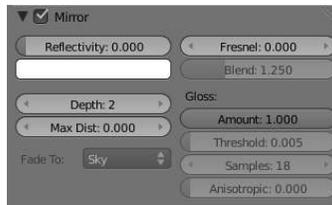


Fig. 2.1691: The Mirror Panel

### Options

**Enable ray-traced reflections** Enable or disable ray-traced reflections

**Reflectivity** Sets the amount of reflectiveness of the object. Use a value of 1.0 if you need a perfect mirror, or set it to 0.0 if you don't want any reflection.

**Color swatch** Color of mirrored reflection By default, an almost perfectly reflective material like chrome, or a mirror object, will reflect the exact colors of its surrounding. But some other equally reflective materials tint the reflections with their own color. This is the case for well-polished copper and gold, for example. In order to replicate this within Blender, you have to set the Mirror Color accordingly. To set a mirror color, simply click the color swatch in the mirror panel and select a color.

**Fresnel** Sets the power of the Fresnel effect. The Fresnel effect controls how reflective the material is, depending on the angle between the surface normal and the viewing direction. Typically, the larger the angle, the more reflective a material becomes (this generally occurs on the outline of objects).

**Blend** A controlling factor to adjust how the blending happens between the reflective and non-reflective areas.

**Depth** Maximum allowed number of light inter-reflections. If your scene contains many reflective objects and/or if the camera zooms in on such a reflective object, you will need to increase this value if you want to see surrounding reflections in the reflection of the reflected object (!). In this case, a Depth of 4 or 5 is typically a good value.

**Max Dist** Maximum distance of reflected rays away from camera (Z-Depth) in Blender units. Reflections further than this range fade out to reduce compute time.



Fig. 2.1692: Picking a mirror color

**Fade to** The color that rays with no intersection within the *Max Distance* take. *Material* color can be best for indoor scenes, *Sky* color (World settings) for outdoor scenes.

**Gloss** In paint, a high-gloss finish is very smooth and shiny. A flat, or low gloss, finish disperses the light and gives a very blurry reflection. Also, uneven or waxed-but-grainy surfaces (such as car paint) are not perfect and therefore slightly need a Gloss  $< 1.0$ . In the example to the right, the left mirror has a Gloss of 0.98, the middle is Gloss = 1.0, and the right one has Gloss of 0.90. Use this setting to make a realistic reflection, all the way up to a completely foggy mirror. You can also use this value to mimic depth of field in mirrors.

**Amount** The shininess of the reflection. Values  $< 1.0$  give diffuse, blurry reflections and activate the settings below.

**Threshold** Threshold for adaptive sampling. If a sampling contributes less than this amount (as percentage), sampling is stopped. Raising the threshold will make the adaptive sampler skip more often, however the reflections could become noisier.

**Samples** Number of cone samples averaged for blurry reflection. More samples will give a smoother result, but will also increase render time.

## Examples

**Fresnel** Let's undertake a small experiment in order to understand what Fresnel is really about. After a rainy day, go out and stand over a puddle of water. You can see the ground through the puddle. If you kneel just in front of the puddle, your face close to the ground, and look again at a distant point on the puddle of water, the liquid surface part which is closer to you lets you see the ground, but if you move your gaze towards the other end of the puddle, then the ground is gradually masked until all you see is the reflection of the sky. This is the Fresnel effect: having a surface sharing reflective and non-reflective properties according to the viewing angle and the surface normal.

In *Demonstration of Fresnel effect with values equal to (from top to bottom) 0.0, 2.5 and 5.0*, this behavior is demonstrated for a perfectly reflective Material (Mirror Reflectivity 1.0).

Fresnel 0.0 stands for a perfect mirror Material, while Fresnel 5.0 could stand for a glossy Material. It's barely noticeable but in the lower picture, the Material is perfectly reflective around the edges.

The smoothness of the Fresnel limit can be further controlled using the *Blend* slider.

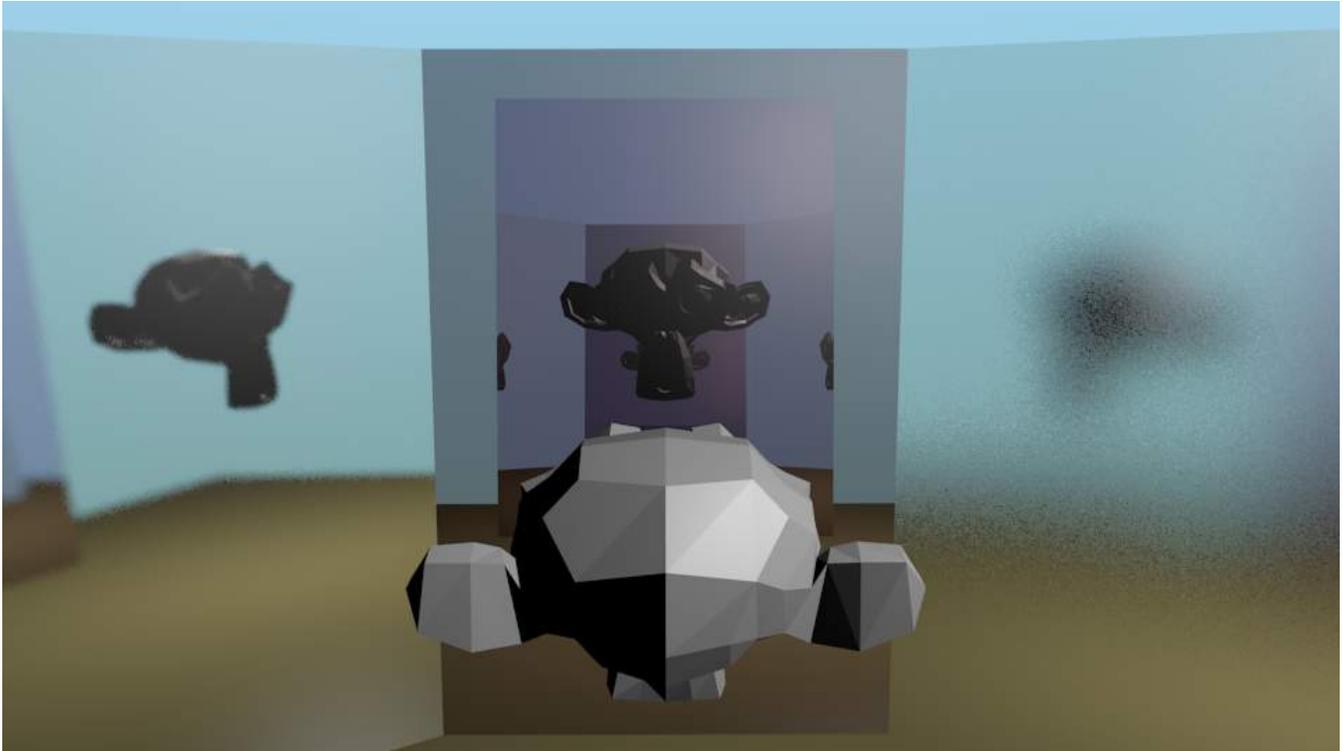


Fig. 2.1693: Suzanne in the Fun House (.blend)

### Subsurface Scattering

Many organic and inorganic materials are not totally opaque right at the surface, so light does not just bounce off the top surface. Instead, some light also penetrates the skin surface deeply, and scatters around inside, taking on the color of the insides and emerging back out at a different location. Human/animal skin, the skin of grapes, tomatoes, fruits, wax, gels (like honey, or Jello) and so on all have subsurface scattering (SSS), and photo-realism really cannot be achieved without it.

It is important to understand that subsurface scattering and diffuse are one and the same. The difference is in how far light can diffuse beneath the surface before it is absorbed or transmitted back out.

**How it works** Actually calculating the light path beneath the surface of an object is not practical. But it has been shown that it is not necessary to do this, and that one can use a different approach.

Blender calculates SSS in two steps:

- At first the irradiance, or brightness, of the surface is calculated, from the front side of the object as well as from its back side. This is pretty much the same as in a normal render. Ambient Occlusion, Radiosity, the type of diffuse Shader, the light color, etc. are taken into account.
- In the second step, the final image is rendered, but now the SSS shader replaces the diffuse shader. Instead of the lamps, the calculated lightmap is used. The brightness of a surface point is the calculated “Average” of the brightness of its surrounding points. Depending on your settings the whole surface may be taken into account, and it’s a bit more complicated than simply calculating the average, but don’t bother too much with the math behind it.

Instead let’s see what SSS does to a distinct light point.



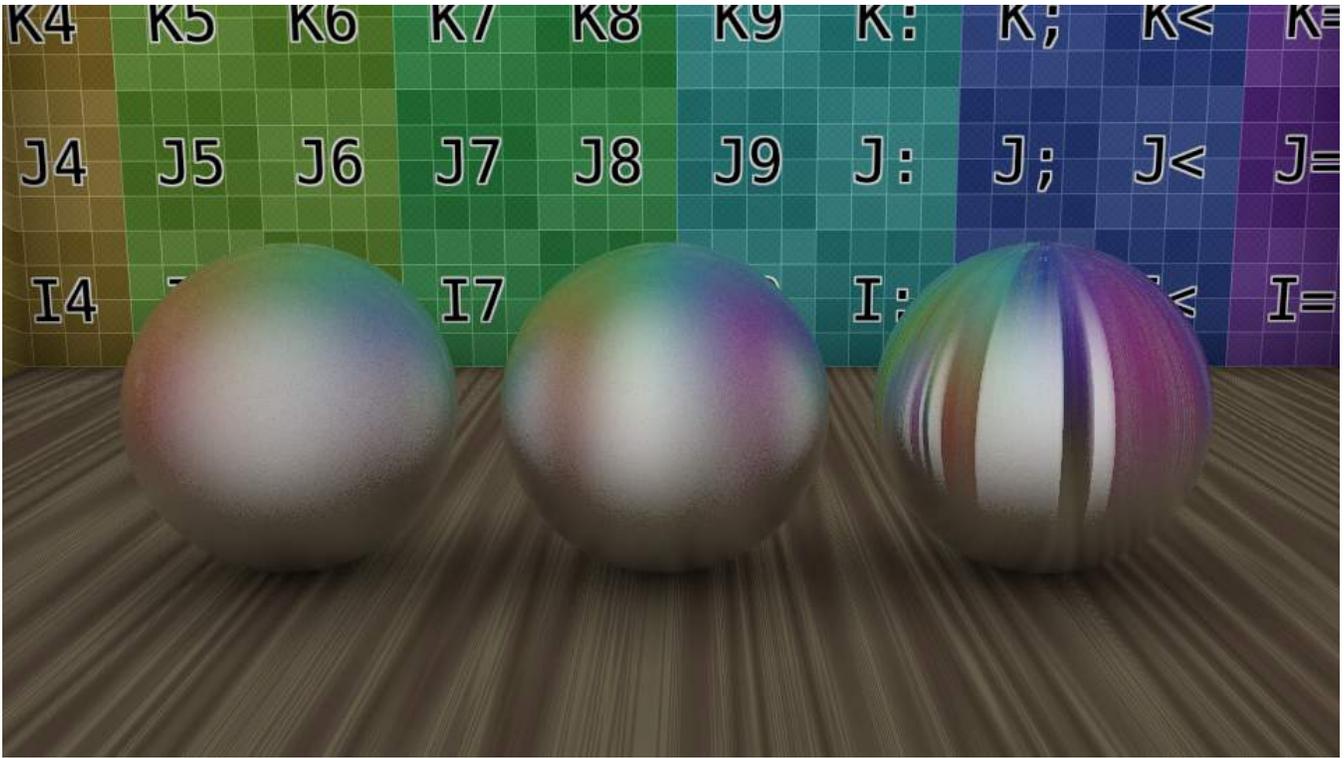
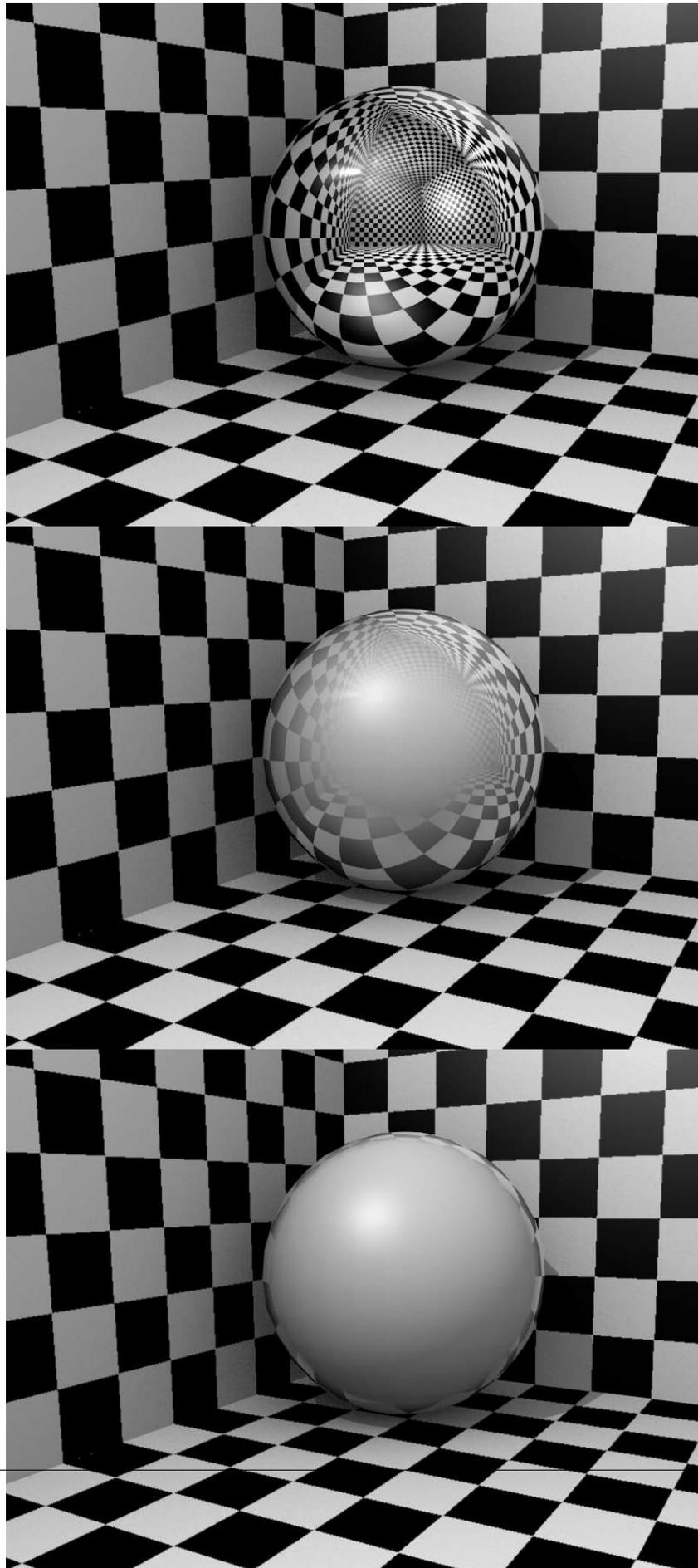


Fig. 2.1694: Anisotropic tangent reflecting spheres with anisotropic set to 0.0, 0.75, 1.0. (.blend)

**Anisotropic** The shape of the reflection, from 0.0 (circular) to 1.0 (fully stretched along the tangent). If the *Tangent Shading* is on, Blender automatically renders blurry reflections as anisotropic reflections. When *Tangent* is switched on, the *Anisotropic* slider controls the strength of this anisotropic reflection, with a range of 1.0 (default) being fully anisotropic and 0.0 being fully circular, as is when tangent shading on the material is switched off. Anisotropic ray-traced reflection uses the same tangent vectors as for tangent shading, so you can modify the angle and layout the same way, with the auto-generated tangents, or based on the mesh's UV co-ordinates.



If you turn on SSS, the light is distributed over a larger area. The size of this area depends on the radius values. Instead of distributing all colors with the same amount, you may choose different radius values for each of the RGB colors.

If you use a very large radius value for a color, its light is evenly distributed over the whole object.

**Note:** Note about scatter radius

Because of the way this scattering is calculated, when using large radius values, you will notice fringing artifacts that appear as the complementary color to the predominant color of the scattering. Above, you see in the last image a bluish band in the illuminated area. This is an unfortunate limitation. A way to lessen this effect is use multiple passes with different scatter radii, and average them.

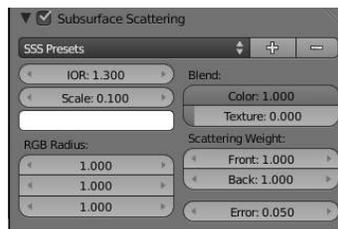


Fig. 2.1704: Image 4: The SSS Panel. SSS is already enabled.

### Enabling Subsurface Scattering

- Enable SSS by clicking on the *Subsurface Scattering* button.
- Accessible at the top are various presets. Add new or remove old presets by clicking the + and - buttons. When you select a preset, the *Radius* values, the *RGB Radius* and the *IOR* are set for you. The remaining options are not set (because they are mostly dependent on the size of your object).

*SubSurface Scattering* doesn't need ray tracing. But since it is dependent on the incident light and shadows, [you need proper shadow calculation (which may need ray tracing)].

**Options** The numeric sliders control how the light is scattered:

**IOR** The *Index Of Refraction* value determines the falloff of incident light. Higher values means that light falls off faster. The effect is quite subtle and changes the distribution function only a little bit. By the examination of many different materials, values of **1.3** to **1.5** have been found to work well for most materials. If you know the exact material you are trying to simulate, see [IOR values for Common Materials](#).

**Scale** The scale of your object, in Blender units, across which you want the scattering effect to take place. Scale **1.0** means **1** Blender unit equals **1** millimeter, scale **0.001** means **1** Blender unit equals **1** meter. If you want to work out what scale value to use in your scene, just use the formula: (size in blender units)/(real world size in millimeters)=scale.

**Scattering Color (Albedo)** Albedo is the probability that light will survive a scattering event. If you think of scattering as a filter, this is the height of the filter. It is multiplied by the surface color. In practice, this is unintuitive. It should be the same as the surface color, however changing this value has unintuitive results on the scattering effect:

The darker the color the more light is scattered. A value of 1 will produce no scattering effect.

So if you set it to green, the lit areas of the object will appear as green, and green is scattered only a little. Therefore the darker areas will appear in red and blue. You can compensate the different scattering by setting a larger radius for the color.

**RGB Radius** This is not in fact the radius of the subsurface scattering, but the average path length between scattering events. As the light travels through the object it bounces around then emerges from the surface at some other point. This value corresponds to the average length the light travels between each bounce. The longer the path length is, the further the



Fig. 2.1705: The SSS Color Swatch

light is allowed to scatter. This is the main source of a material's perceived "scatter color." A material like skin will have a higher red radius than green and blue. Subsurface scattering is the diffusion of light beneath the surface. You control how far the light spreads to achieve a specific result.

#### Blend :

**Color** This controls how much the R, G, B option modulates the diffuse color and textures. Note that even with this option set to **0.0**, the R, G, B option still influences the scattering behavior.

**Texture** How much the surface texture is blurred along with the shading.

#### Scattering Weight :

**Front** Factor to increase or decrease the front scattering. When light enters through the front of the object, how much is absorbed or added? (Normally **1.0** or **100%**).

**Back** Factor to increase or decrease the back scattering. Light hitting an object from behind can go all the way through the object and come out on the front of the object. This happens mostly on thin objects, like hands and ears.

**Error** This parameter controls how precisely the algorithm samples the surrounding points. Leaving it at **0.05** should give images without artifacts. It can be set higher to speed up rendering, potentially with errors.

Setting it at **1.0** is a good way to quickly get a preview of the look, with errors.

#### Developing your own SSS material

**The Traditional Approach** A more common but less intuitive approach is to use "layering". This is a simplified version of the layering approach. See the external links for more information:

- Set the SSS color on a value of your choice, normally the predominant color of the object. If you want to use different radii for the colors, don't make it too dark.
- Set the scale factor. If you want to see much translucency you need small objects or large scale values.
- Set the radius values.
- Adjust the brightness with the *Front* and *Back* values.

#### =A more intuitive approach

- Set the Scattering color to .5
- Set the Front weight to 2.
- Set the scale factor based on the size of your object relative to the scene. If you want to see much translucency you need small objects or large scale values.

- Set the radius values appropriately.

## Examples

### Skin

Table 2.52: 5



### See also

- [Development Release Log: Subsurface Scattering](#)
- [Ben Simonds: Three Layer SSS in Blender Demystified](#)

## Strands

The Strand section of the Material editor is specific to the rendering of Hair particles. There are two different strand methods available:

**Polygon strands** This is the default (old) method. The strands are rendered as flat polygons. The number of polygons depend on the *Steps* settings in the *Render* panel of the *Object* context, *Particles* sub-context.

**Strand Primitive** You activate Strand Primitive with the button *Strand render* in the *Render* panel of the particle system. The hair curves are not stored as polygons; only the key points are stored, which are then converted to polygons on the fly. A second difference is the way transparency works. Rather than rendering using the existing system, all strand segments in a part are sorted front to back and rendered in that order.

Strand Primitives:

- Are more memory efficient and faster, to make rendering of large amounts of fur and grass possible. For good performance, the render steps button should be lowered (e.g. 2 should be good enough fur), since the result will be a smoothed curve anyway. You need 1 to 2 render steps less than steps in the 3D window. Also, using more render parts helps to reduce memory usage.
- Have a distance of vision reduction (in the *Render* panel under *Child Simplification*) for children from faces.
- May be faded out towards the tip without an additional texture.
- Are not ray traced. So they are not visible through ray-transparent materials or in a ray mirror (you can use *Environment Mapping* for that).
- Have shape problems if they are rendered with a greater width.
- Cannot carry a UV-Texture along the strand.

Polygon strands:

- Work well with greater width, so you can use them as an alternative to billboards because the strands may have an animated shape.
- Can be textured with a UV-Texture along the strands.
- Are seen by ray tracing.

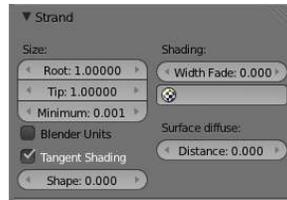


Fig. 2.1716: Image 1: Strands Panel.

**Strands Shading** Strands are rendered with the material of the underlying face/vertex, including shading with a UV-Texture. Since you can assign more than one material to each face, each particle system may have its own material and the material of the underlying face can be different from the material of the strands.

Additionally strands can be shaded along the strand (from root to tip) with a mono-dimensional texture; only polygon strands can carry a two-dimensional UV-Texture.

The options for strand shading are in the *Strands* section of the *Material* context.

**Root** Width of the hair at the root.

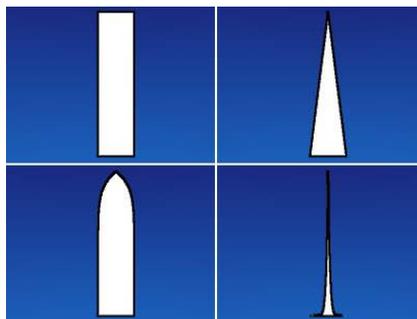
**Tip** Width of the hair at the tip.

**Minimum** This is the minimum thickness (in pixels) of the strands. Strands below that size are not rendered smaller, but are faded to alpha (well, the fading works only for strand primitives). This gives a much better rendering result for thin hair.

**Blender Units** Normally strands are quite thin; the thickness is given in screenpixels. If you use Blender units (BU) you may set the root value up to 2 BU, and the tip value up to 1 BU. You have to consider the overall object size, because the smallest possible size is 0.001 BU. So if you use 1 BU for 1 meter the smallest possible size would be 1 mm (too thick for thin hair).

**Use Tangent Shading** Calculates the light as if the strands were very thin and round. This makes the hair appear brighter and shinier. Disabling the “Tangent Shading” option will still render nicely, but resembles more solid strands, as though made of metal or wood.

**Shape** This slider allows you to control the interpolation. Default (0.0) is a linear interpolation between *Root* and *Tip*. A negative value will make the strand narrower (spiky), a positive value will make it fatter.



**Width Fade** To fade out along the width of the strand. This works only for Strand Primitives. 0.0 is no fading at all, 1.0 linear fading out.

**UV Layer** You can texture polygon strands with a UV-Texture. Fill in the name of the UV-Set (not the texture) here. You also have to load the texture in the *Shading* context, *Texture* and *Material* sub-contexts (*Mapping: UV*; you may use every *Influence* setting you like - especially the alpha value; see *Image 3*).

**Surface Diffuse** Computes the strand normal, taking the normal at the surface into account. This eases the coloring and lighting of hair a lot, especially for Strand Primitives. Essentially hair reacts similar to ordinary surfaces and don't show

exaggerated strong and large specular highlights.

**Distance** The distance in Blender units over which to blend in the normal at the surface (if you want to use *Surface Diffuse* only for Grass/Fur at greater distances).

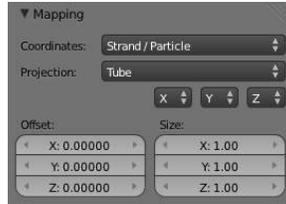


Fig. 2.1717: Image 4: Fading a strand to alpha...



Fig. 2.1718: Image 5: ...And the render result.

**Texturing along the Strand** Strands can be textured along the strand, i.e. from root to tip. To do that you have to select *Strand/Particle* in the *Coordinates* drop-down in the *Mapping* panel of the *Material* sub-context.

Pretty much the most important setting is shown in (*Image 4*), how to fade the tip of a strand to alpha to make nice, fuzzy-looking hair. Normally you would use a linear blend texture for this.

You may of course set any attribute you like, especially color. Be careful with specularity; hairs tend to get too shiny.



Fig. 2.1719: Image 5: Strand render child simplification.

**Strand render Simplification** If you use Strand Primitives (*Strand render* button) and have activated *Interpolated Children*, the *Child Simplification* option appears. The strand render has options to remove child strands as the object's faces become smaller.

**Reference Size** This is the approximate size of the object on screen (in pixels), after which simplification starts.

**Rate** How fast strands are removed.

**Transition** The transition period for fading out strands as they are removed.

**Viewport** This removes strands on faces that are outside of the viewport.

**Rate** Controls how fast these are removed.

## Options

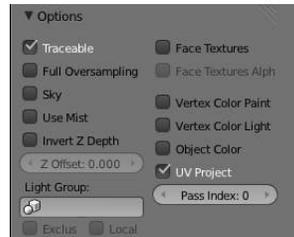


Fig. 2.1720: Material Options Panel

This panel provides a series of control options concerning how objects using this material will appear in the rendered image. All controls are default “Off” unless otherwise stated.

**Traceable (default On)** Include this material and the geometry that uses it in ray-tracing calculations. See [Transparency](#) for details of ray-tracing.

**Full Oversampling** Force this material to render full shading/textures for all anti-aliasing samples.

**Sky** Render this material with zero alpha, but with [sky background](#) in place (scanline only)

**Use Mist** Use [mist](#) on this material (see “World Settings” for more details)

**Invert Z depth** Render material’s faces with an inverted Z buffer (scanline only)

**Z Offset** Give faces an artificial Z offset for Z transparency.

**Light Group** Limit lighting to lamps in this [light group](#).

**Exclusive** Uses the [light group](#) exclusively - these lamps are excluded from other scene lighting

**Local** When linked in, uses local [light group](#) with the same name.

**Face Textures** Replace object’s base color with color from [UV map](#) image textures.

**Face Textures Alpha** Replace object’s base alpha with alpha from [UV map](#) image textures.

**Vertex Color Paint** Replace object’s base color with [vertex paint](#) colors (multiply with ‘texture face’ face assigned textures)

**Vertex Color Light** Add [vertex paint](#) colors as additional lighting. (This can be used to produce good incandescence effects).

**Object Color** Modulate the result with a per object color

**UV Project (default On)** Use to ensure UV interpolation is correct for camera projections (use with [UV project](#) modifier).

**Pass Index** Index number for the IndexMA render pass.

## Shadows

The shadows that appear in a scene are affected by a combination of the layout of objects, the shape of the objects, the materials of the objects, and the lighting. In Blender, the Display Mode (Single Texture, Multitexture, or GLSL) also affects the appearance of shadows. See [Shadows](#) for a more complete description of this subject.

**Tip:** Shadows in 3D mode

To see shadows in 3D (textured) mode, you must have switched to GLSL mode before making any materials. In MultiTexture mode, shadows only appear in the rendered image: none of these can be seen in the preview image.



Fig. 2.1721: Fig. 1: Shadow Panel.

The Shadow panel in the Materials Properties window (Fig. 1) controls the effects that the material can have on the shadows that appear in the scene. The various properties are described in the sections below.

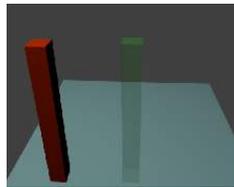


Fig. 2.1722: Fig. 2: Scene- all shadow properties off.

**Options** The following properties can be set for each individual material with which objects in the scene are shaded. The effects are illustrated with rendered images for a simple scene (Fig. 2) consisting of two “posts”, one with a red (totally non-transparent) material; one green (partially transparent) material, set up on a light blue plane to receive the shadows. The illustrations were all taken in Blender Render engine, with Multitexture mode.

**Shadow Receiving Object Material** The following options affect the material that receives shadows:

**Receive** Allows this material to receive full-intensity shadows (Fig. 3).

**Receive Transparent** Allows this material to receive shadows whose intensity is modified by the transparency and color of the shadow-casting object (Fig. 4).

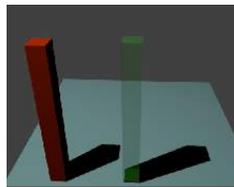


Fig. 2.1723: Fig. 3: Plane - Receive.

**Shadow Casting Object Material** The following options affect the material that casts shadows:

**Cast Only** Material appears transparent, but it still casts shadows (Fig. 5).

**Casting Alpha** ??

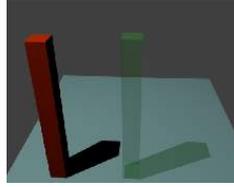


Fig. 2.1724: Fig. 4: Plane - Receive + Receive Transparency.

**Shadows Only** Material appears transparent except for where it receives shadows from other objects, and also it retains its own transparency (Fig. 6). Note the faint image of the partly-transparent post.

**Shadow and Distance** ???

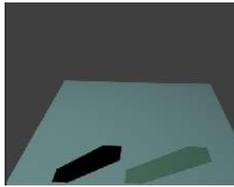


Fig. 2.1725: Fig. 5: Posts - Cast Only.



Fig. 2.1726: Fig. 6: Posts - Shadows Only.

**Buffered Shadow Options** In addition to the shadow options described above, there are further material properties which control buffered shadow features. See section on [Spot Buffered Shadows](#) for further discussion of these techniques.

**Cast Buffer Shadow** Casts shadows from shadow buffer lamps.

**Buffer Bias** Multiplication factor for Buffer shadows (0 = ignore)

**Auto Ray Bias** - Prevent raytraced shadow errors on surfaces with smooth shaded normals.

**Ray Bias** Bias value to be used.

**Cast Approximate** Allow this material to cast shadows when using approximate ambient occlusion.

### Game Settings

This panel contains properties that control how the object surfaces that use the material are rendered in real time by the Blender Game Engine.

Game settings are visible when using the game engine for rendering. Material physics usage is described [here](#).

**Backface Cull (default On)** Hide the back faces of objects rendered with this material. If “Off”, both sides of the surface are visible (at the expense of lower rendering speed). Note that this setting is applied per-material and not per-face; e.g. if the material is applied to a cube, only the back and front faces of the cube are visible, and not both sides of each face.



Fig. 2.1727: Game Settings Panel

**Invisible** Hide all faces of objects rendered with this material.

**Text** Use material as [Text object](#) in the Game Engine.

**Alpha Blend menu:** Controls how the alpha channel is used to create a transparent texture in the rendered image.

**Alpha Sort** Orders the sequence in which transparent objects are drawn on top of each other, so that ones in front receive more light than ones behind.

**Alpha Blend** Uses the alpha values present in the bitmap image sourced in the Image slot.

**Alpha Clip** Uses the alpha channel as a simple mask.

**Add** Render face transparent and add color of face.

**Opaque (default)** All alpha values are ignored; the scene is completely non-transparent.

**Face Orientation** menu: Provides options regarding the orientation (i.e. rotation transformation) of faces to which the material is applied.

**Shadow** Faces are used for shadow.

**Billboard** Billboard with Z-axis constraint.

**Halo** Screen aligned billboard.

**Normal (default)** No transformation.

**Note: Incomplete Page.** Descriptions of some controls either very brief or missing. If you have more information, please update this page (and remove this note when you feel the page is complete).

## Game Physics



Fig. 2.1728: Panel Physics in Material context

This panel contains physical properties that control how the object surfaces that use the material are rendered in real time by the Blender Game Engine.

Physics settings are visible when using the game engine for rendering. Game physics usage is described [Here](#)

**Friction** Coulomb friction coefficient when inside the physics distance area.

**Elasticity** Elasticity of collisions.

**Force Field** Controls force field settings.

**Force** Upward spring force when inside the physics distance area.

**Distance** Distance of physics area.

**Damping** Damping of the spring force when inside the physics distance area.

**Align to Normal** Align dynamic game objects along the surface normal when inside the physics distance area.

## Nodes

**Introduction** In addition to creating materials as just described using all the settings on all the materials panels, Blender allows you to create a material by routing basic materials through a set of nodes. Each node performs some operation on the material, changing how it will appear when applied to the mesh, and passes it on to the next node. In this way, very complex material appearances can be achieved.

You should already be familiar with general material concepts and how to create materials/textures using the material menu. You should also have a general understanding of the texture coordinate systems available in Blender (e.g. Generated, UV, etc.). Also, many aspects of a node will be skipped here because in later sections you will see the function expanded upon. Each section builds off the previous.

To start, the node system does not make the material menu obsolete. Many features and material settings are still only accessible through the material panel (e.g. Ray Mirror). However with the advent of nodes, more complex and fantastic materials can be created since we now have greater control.

Just in case you're not (yet) familiar with the concepts: when you create a system of nodes (otherwise known as a "noodle"), you're describing a data-processing pipeline of sorts, where data "flows from" nodes which describe various *sources*, "flows through" nodes which represent various processing and filtering stages, and finally "flows into" nodes which represent outputs or destinations. You can connect the nodes to one another in many different ways, and you can adjust "knobs," or parameters, that control the behavior of each node. This gives you a tremendous amount of creative control. And, it will very quickly become intuitive.

Having said all that, let's begin with a normal material.

Here we have the standard material we have added to a cube mesh. We could, as we have in the past, add color and other settings to this material and it would certainly look nice. But let's say we are just not getting what we are looking for? What if we want to control the creation more tightly or add more complexity? Here is where nodes come in.

Making this node map is accomplished by working in a [Node Editor window](#). This section covers:

- Enabling Material Nodes.
- The Node Editor window, its basic controls, and working with nodes.
- The specific types of nodes available for materials.

FIXME(Template Unsupported: Doc:2.6/Reference/Nodes/Concepts; {{Doc:2.6/Reference/Nodes/Concepts}})

**Accessing The Node Editor** First lets enter the [node editor](#) and make sure that the node editor has the material node button (the sphere icon) pressed, not the composite or texture node buttons.

**Enabling Node Materials in the Material Buttons** 

Let's take the base material and hit the Nodes button next to the material name in the material panel or the node editor. You will see a change in the material panel.

What you have just done is told Blender to make the material you were on to become the node tree. Most of the panels we normally find in the material menu are now gone.

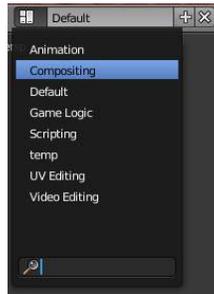


Fig. 2.1733: Accessing the Compositing screen

If you switch to the *Compositing* screen (`Ctrl-Left` if you are on the default screen) you'll find a *Node Editor* on the top half of the screen. When you enabled material nodes, a material node and an output node were automatically added to the node editor.

You can also split the 3D view in the default screen in two and change one into a *Node Editor*.



It is important to note that you can add a new material (which you can edit and change like any other material in the material panel), add an already created material or append a material from another blender file, and also use the material that you used to create the node tree.

Here, we added a new material in the *Node editor* (*Material.001*), and as we did, we can access the properties of this material in the material's menu.

## External Links

- [Blender Material Nodes](#) - Changelog for the Blender version that introduced material nodes.

## Node Editor

FIXME(Template Unsupported: Doc:2.6/Reference/Nodes/Node Editor; {{Doc:2.6/Reference/Nodes/Node Editor}} )

## Node Controls

FIXME(Template Unsupported: Doc:2.5/Reference/Nodes/Node\_Controls; {{Doc:2.5/Reference/Nodes/Node\_Controls}} )

## Using Nodes

FIXME(Template Unsupported: Doc:2.5/Reference/Nodes/Using\_Nodes; {{Doc:2.5/Reference/Nodes/Using\_Nodes}} )

## Node Groups

FIXME(Template Unsupported: Doc:2.5/Reference/Nodes/Node\_Groups; {{Doc:2.5/Reference/Nodes/Node\_Groups}} )

## Types of Material Nodes

This section is organized by type of node, which are grouped based on similar functions:

**Input** Introduces a material or component to the node map.

**Output** Displays the result in progress as a small image.

**Color** Manipulates the colors of the material.

**Vector** Change the way light is reflected off the material.

**Convertors** Convert colors to other material colors.

**Groups** User-defined groups of nodes.

**Dynamic** Custom nodes defined by Python. These are also known as PyNodes.

## Material Input Nodes

A starting material is created in the Materials Panel. The *Nodes* button is enabled to add that material to the list of noded materials shown in the Node Editor window header. Other inputs to the node map include:

- A value
- A color
- A texture
- Geometry
- Material
- Camera Data
- Lamp Data

---

## Material Node Reference

Panel: [Node Editor](#) -> [Material Nodes](#)

Menu: [Shift-A](#) -> [Input](#) -> [Material](#)

---

The Material node is used to add a material to the node program. Materials can be anything from pure shading to fully layered with textures. It inputs the main attributes of a material (color, alpha and normal vector) into the map.

**Output** Materials can output color (which includes shading and any textures assigned to it), alpha, and the final normal calculated from any textures it has.

**Color** value of the color, combined by the node.

**Alpha** value of the alpha, combined by the node.

**Normal** direction of the normal, combined by the node.

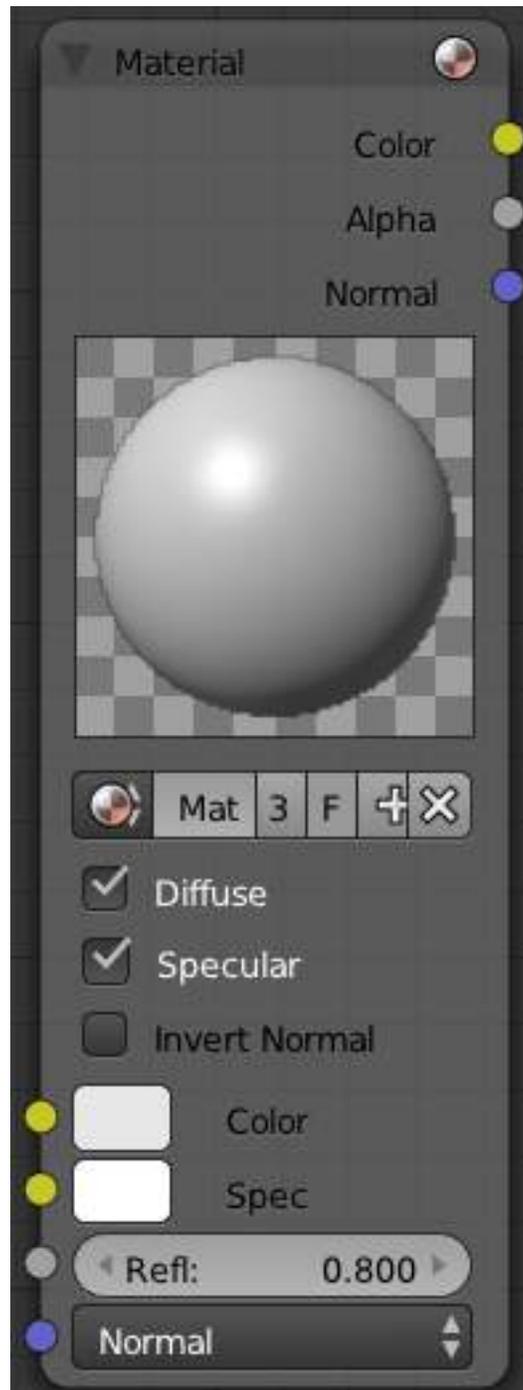


Fig. 2.1738: Material node

**Input** Materials can take inputs for colors, inputs for diffuse color and specular color, a value for reflectivity, and a normal.

**Color** The base color of the paint. Can be set

- manually by **LMB** clicking on the color swatch applet next to the socket, choosing a color using the control panel that pops up, and pressing **Return**
- based on an Active Material which is specified using the material panels, or
- plugged in from an RGB color generator.

**Spec** The color that is reflected as you get perpendicular to the light source reflecting off the surface. The color can be

- plugged in from another node or...
- set manually by **LMB** clicking on and using the color swatch applet.

**Refl:** The degree to which the material reflects light and gives off its color. The value can be provided by another node or set manually.

**Normal** The lighting condition.

### Controls

**Material field** You can browse and select materials here.

**Diffuse toggle** Turn on/off Diffuse Color.

**Specular toggle** Turns on/off Specularity calculation.

**Invert Normal toggle** Inverts the material input normal when activated (which, of course, is a combination of the 3D normal given to it by the 3D object plus the normal input point).

---

**Note:** Normal Override

The normal input socket does not in any way blend the source normal with the underlying geometry. Any plugged in Geometry here overrides the Normal lighting conditions.

---

**Using the Material Node with Specularity** To make a material node actually generate a color, you have to specify at least a basic input color, and optionally a specular color. The specular color is the color that shines under intense light.

For example, consider the mini-map to the right. The base color, a dark blue, is connected from an RGB color generator node to the *Color* input socket. The specular color, yellow, is connected to the *Spec* input. Under *Normal* lighting conditions on a flat surface, this material will produce a deep blue color and, as you approach a spot perpendicular to the light, you will see the yellow specular color mix in.

---

**Note:** Enable Spec

To see specularity, you have to enable it by clicking the blue Spec button located just below the material color swatch in the node.

---

**Extended Material Node** Adds additional input and output channels to the material node.

### Input

**Color** Includes a color swatch, allowing you to select the color directly on the node.

**Mirror Color** Color of mirrored reflection.

**Ambient** Amount of global ambient color the material receives.

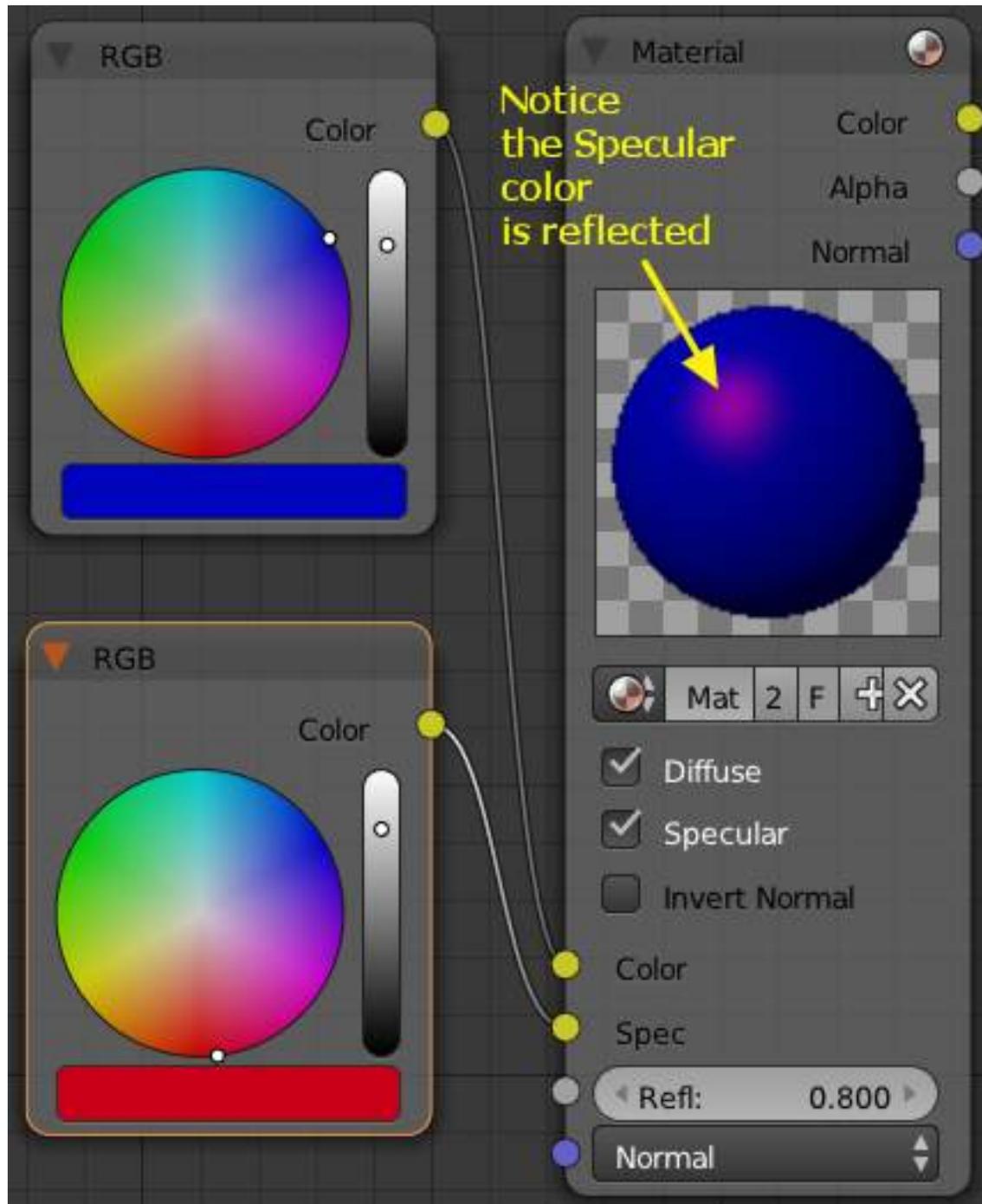
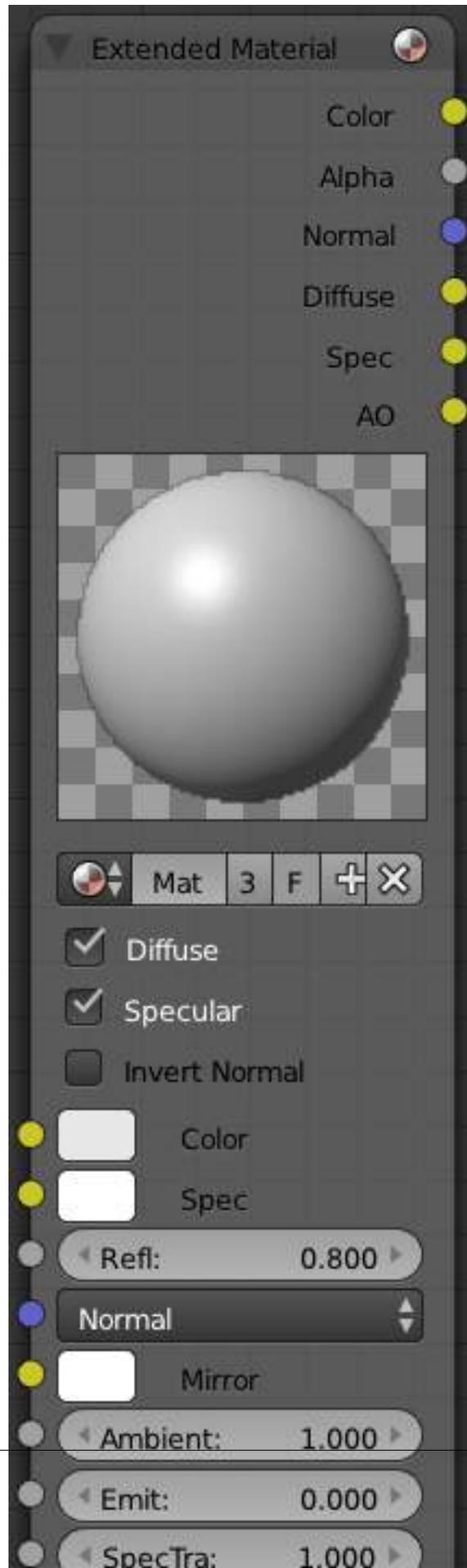


Fig. 2.1739: Material Node using Specularity



**Emit** Amount of light to emit.

**SpecTra** Alpha for the specular color.

**Ray Mirror** Amount of reflectiveness of the object.

**Alpha** Transparency of the material by setting all pixels in the alpha channel to the given value.

**Translucency** Amount of diffuse shading on the back side

**Output** Materials can additionally output the followings:

**Diffuse** value of the diffuse color, combined by the node.

**Spec** value of the specular color, combined by the node.

**AO** value of the Ambient Occlusion, combined by the node.

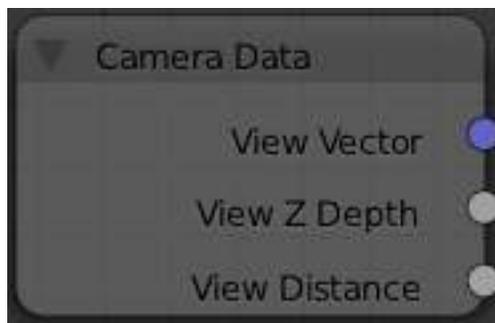


Fig. 2.1741: Camera Data node

### Camera Data Node

**View Vector** A Camera space vector from the camera to the shading point.

**View Z Depth** How far away each pixel is from the camera

**View Distance** Distance from the camera to the shading point.

**Lamp Data Node** The Lamp Data node is used to obtain information related to a specified lamp object. Select a lamp object listed in the Lamp field, then the following outputs will be available:

**Color** Lamp color multiplied by the lamp energy.

**Light Vector** An unit vector in the direction from the shading point to the lamp.

**Distance** Distance from the shading point to the lamp.

**Shadow** Shadow color that the other objects cast on the shading point.

**Visibility Factor** Light falloff ratio at the shading point.

The light textures and the shadow textures affect the Color and Shadow outputs, respectively.

---

**Note:** Portability to Various Scenes

If multiple materials use a Lamp Data node linking to the same lamp object, including the Lamp Data node into a node group is recommended. Otherwise, when the mesh objects are imported to the other scene, all the materials may need to be modified.

---

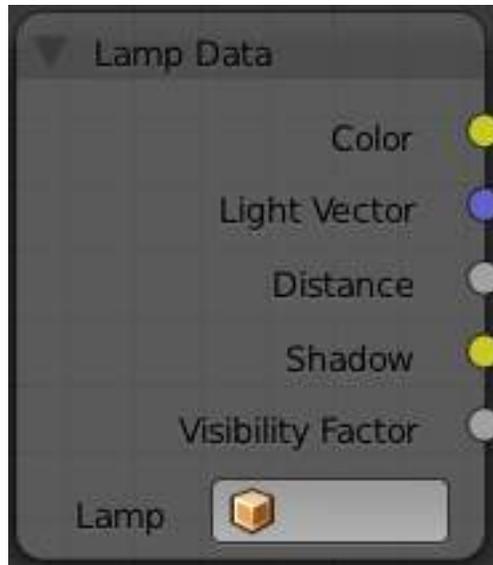


Fig. 2.1742: Lamp Data node

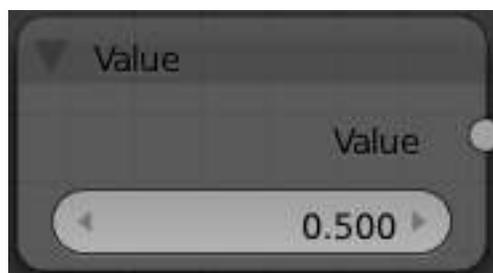


Fig. 2.1743: Value node

**Value Node** The Value node has no inputs; it just outputs a numerical value (floating point spanning 0.00 to 1.00) currently entered in the NumButton displayed in its controls selection.

Use this node to supply a constant, fixed value to other nodes' value or factor input sockets.



Fig. 2.1744: RGB node

**RGB Node** The RGB node has no inputs. It just outputs the value Color currently selected in its controls section.

**Texture Node** A texture, from the list of textures available in the current blend file, is selected and introduced through the value and/or color socket.

### Input

**Vector** Uses for map the texture to a specific geometric space.

### Outputs

**Value** Straight black-and-white value of the texture, combined by the node.

**Color** Texture color output, combined by the node.

**Normal** Direction of normal texture, combined by the node.

In the example to the right, a cloud texture, as it would appear to a viewer, is added to a base purple material, giving a velvet effect.

Note that you can have multiple texture input nodes. With nodes, you simply add the textures to the map and plug them into the map.

**Geometry Node** The geometry node is used to specify how light reflects off the surface. This node is used to change a material's Normal response to lighting conditions.

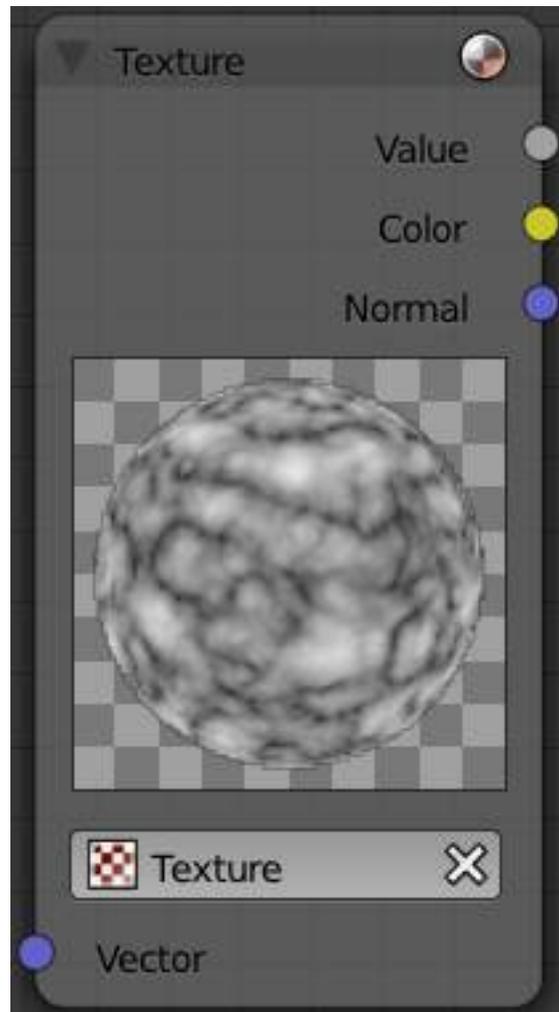


Fig. 2.1745: Texture node

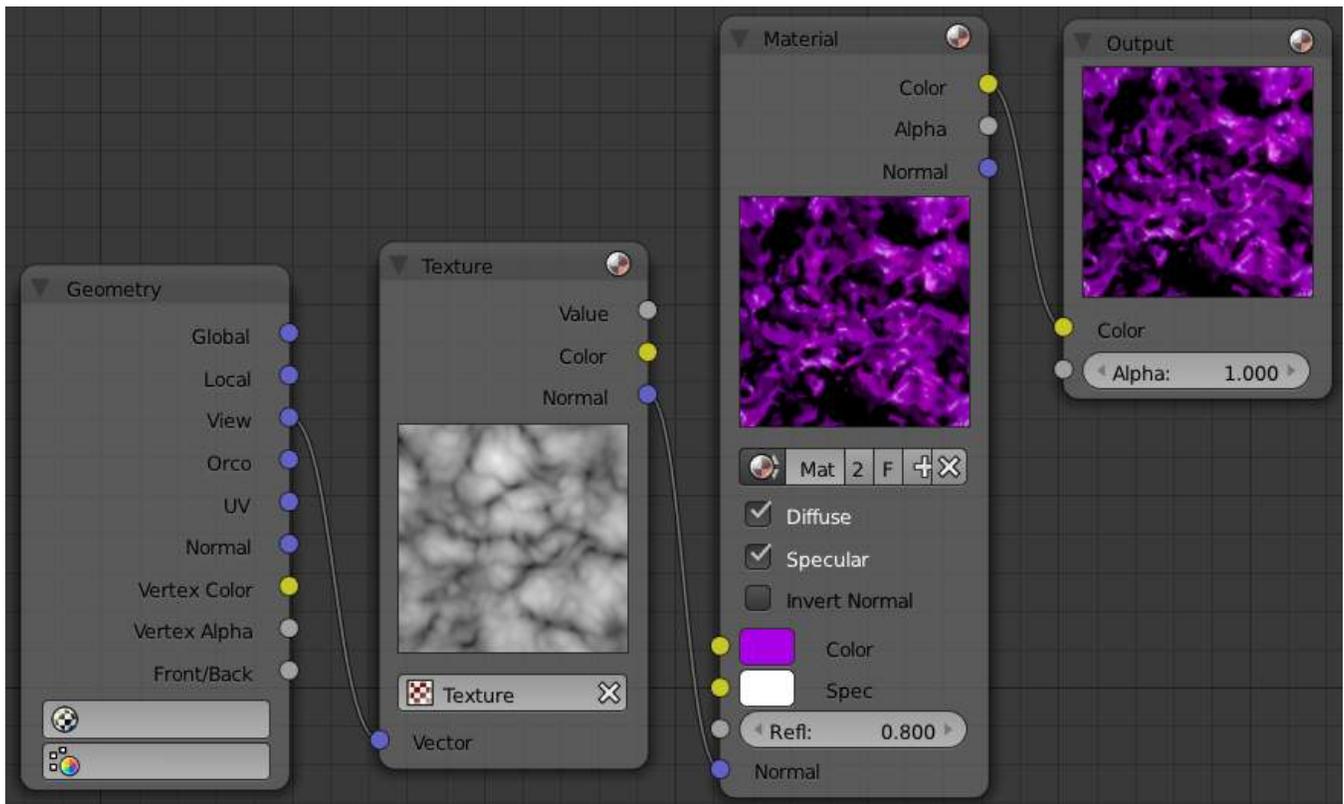


Fig. 2.1746: Example of the applying Texture node

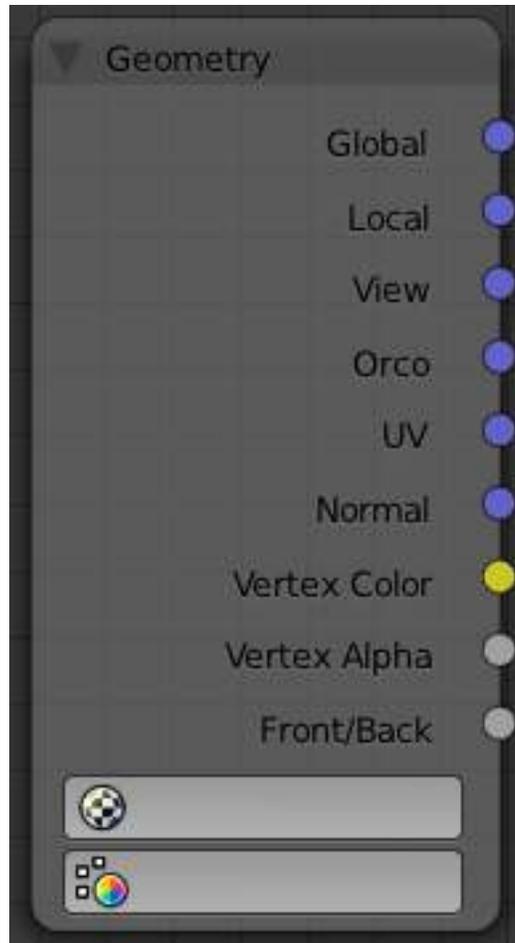


Fig. 2.1747: Geometry node

Use this node to feed the Normal vector input on the Material node, to see how the material will look (i.e. shine, or reflect light) under different lighting conditions. Your choices are:

**Global** Global position of the surface.

**Local** Local position of the surface.

**View** Viewed position of the surface.

**Orco** Using the Original Coordinates of the mesh.

**UV** Using the UV coordinates of the mesh, selected in the field in bottom node.

**Normal** Surface Normal; On a flat plane with one light above and to the right reflecting off the surface.

**Vertex Color** Allows for output value of group vertex colors, selected in the field in bottom node.

**Vertex Alpha** Allows for output alpha value of vertex.

**Front/Back** Allows for output to take into account front or back of surface is light relative the camera.

**Note:** These are exactly the same settings as in the [Mapping](#) panel for [Textures](#), though a few settings - like *Stress* or *Tangent* - are missing here. Normally you would use this node as input for a [Texture Node](#).

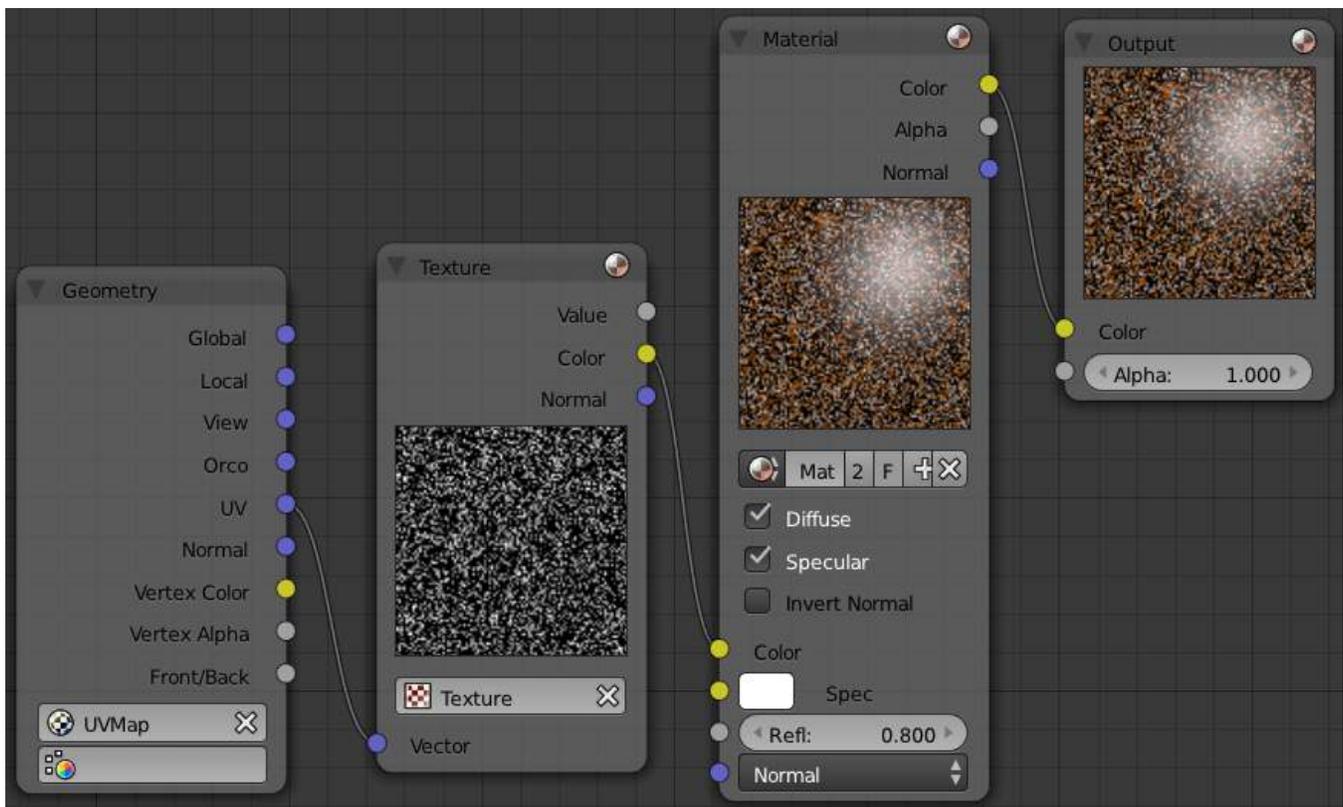


Fig. 2.1748: Setup to render an UV-Mapped Image Texture.

**Geometry Node Example using a UV image** E.g.: To render an UV-mapped image, you would use the *UV* output and plug it into the *Vector* Input of a texture node. Then you plug the color output of the texture node into the color input of the material node - which corresponds to the setting on the *Map To* panel.

## Material Output Node

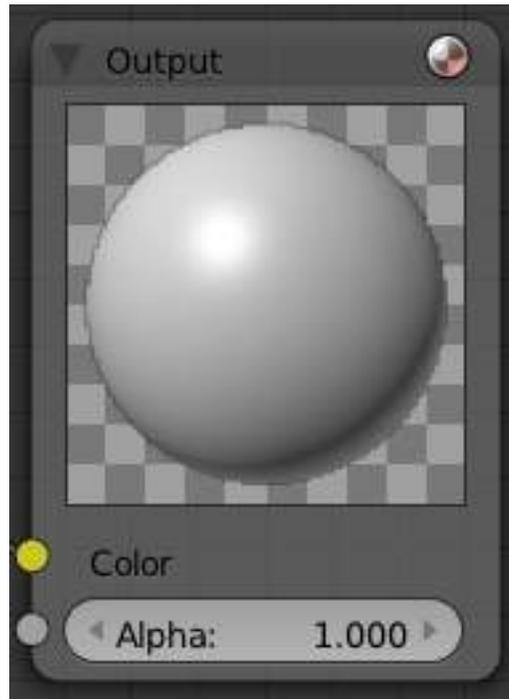


Fig. 2.1749: Output material node

At any point, you may want to see the work in progress, especially right after some operation by a node. Simply create another thread from the output socket of the node to the picture input socket of an Output node to see a mini-picture.

Connect the alpha channel to set/see transparency.

---

### Note: Effective Output Node

The only Output node that is used for the Material in the end (i.e the only non-Preview) has a little **red sphere** on the upper right.

---

## Material Color Nodes

**MixRGB** This node mixes a base color or image (threaded to the top socket) together with a second color or image (bottom socket) by working on the individual and corresponding pixels in the two images or surfaces. The way the output image is produced is selected in the drop-down menu. The size (output resolution) of the image produced by the mix node is the size of the base image. The alpha and Z channels (for compositing nodes) are mixed as well.

Not one, not two, but count 'em, sixteen mixing choices include:

### See also:

*Color Blend Modes* for details on each blending mode.

## Inputs



Fig. 2.1750: MixRGB node

**Fac** The amount of mixing of the bottom socket is selected by the Factor input field (Fac:). A factor of zero does not use the bottom socket, whereas a value of 1.0 makes full use. In Mix mode, 0.5 is an even mix between the two, but in Add mode, 0.5 means that only half of the second socket's influence will be applied.

**Color 1** Input color value. The value can be provided by another node or set manually. Includes a color swatch, allowing you to select the color directly on the node.

**Color 2** Input color value. The value can be provided by another node or set manually. Includes a color swatch, allowing you to select the color directly on the node.

### Outputs

**Color** Value of the color, combined by the node.

### Controls

**Clamp** Clamp result of the node to 0...1 range.

**RGB Curves** For each color component channel (RGB) or the composite (C), this node allows you to define a bezier curve that varies the input (x-axis) to produce an output value (y-axis). Clicking on one of the *C R G B* components displays the curve for that channel.

### See also:

- Read more about using the *Curve Widget*.
- [RGB Curves node in the compositor](#) (includes examples)

**Invert** This node simply inverts the input values and colors.

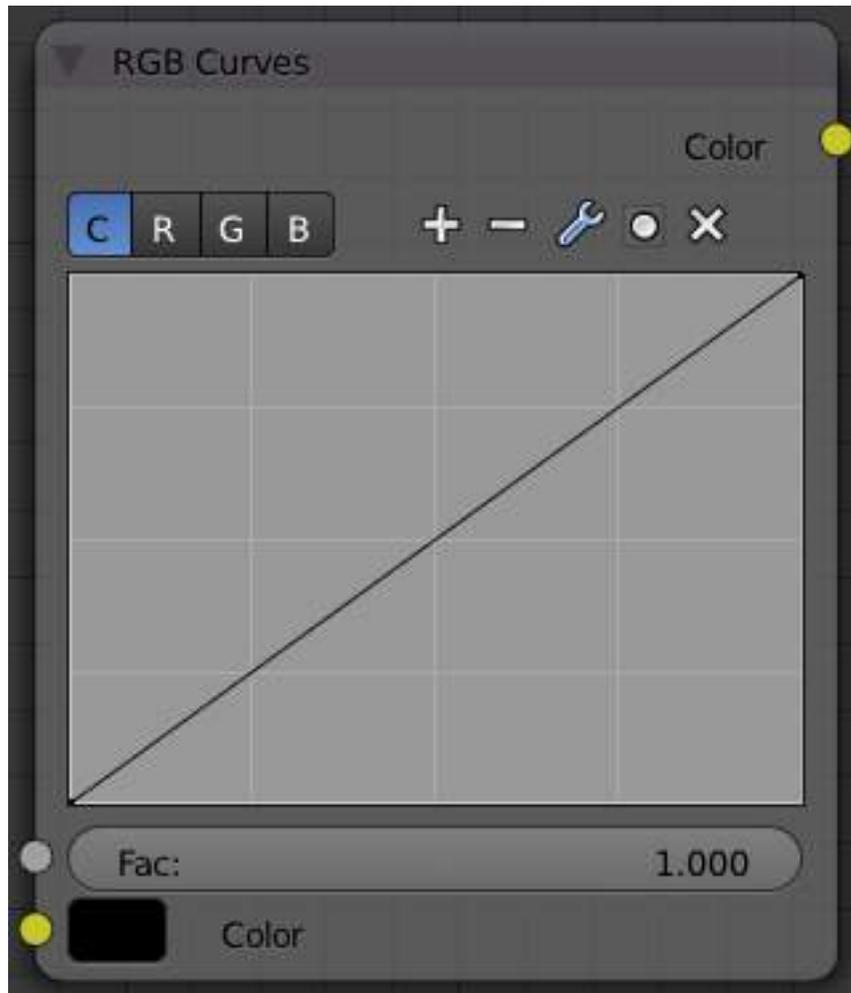


Fig. 2.1751: RGB Curves node



Fig. 2.1752: Invert node

**Inputs**

**Fac:** Factor. The degree of node's influence in node tree. The value can be provided by another node or set manually.

**Color** Input color value. The value can be provided by another node or set manually. Includes a color swatch, allowing you to select the color directly on the node.

**Outputs**

**Color** Value of the color, combined by the node.

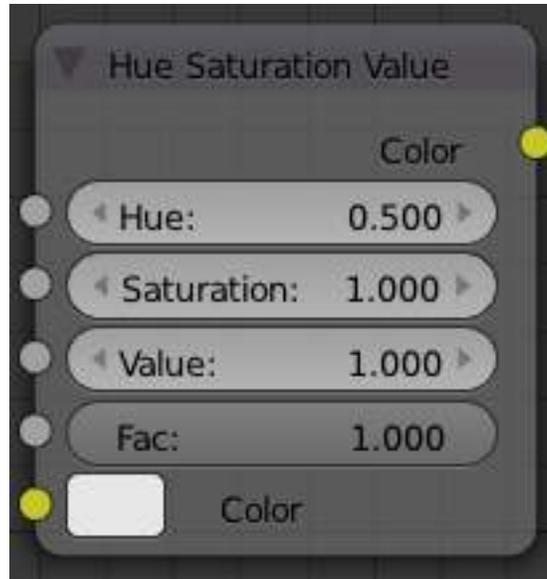


Fig. 2.1753: Hue Saturation Value node

**Hue Saturation Value** Use this node to adjust the Hue, Saturation, and Value of an input.

**Inputs**

**Fac** Factor. The degree of node's influence in node tree. The value can be provided by another node or set manually.

**Hue** Input hue value of color. The value can be provided by another node or set manually.

**Saturation** Input saturation value of color. The value can be provided by another node or set manually.

**Value** Input HSV-Value of color. The value can be provided by another node or set manually.

**Fac** Factor. The degree of node's influence in node tree. The value can be provided by another node or set manually.

**Color** Input color value. The value can be provided by another node or set manually. Includes a color swatch, allowing you to select the color directly on the node.

**Outputs**

**Color** Value of the color, combined by the node.

## Material Vector Nodes

Vector nodes manipulate information about how light interacts with the material, multiplying vector sets, and other wonderful things which can become quite advanced. Even if you don't have experience with vector maths, you'll find these nodes to be very useful.

Vectors, in general, are two or three element values, for example, surface normals are vectors. Vectors are also important for calculating shading.

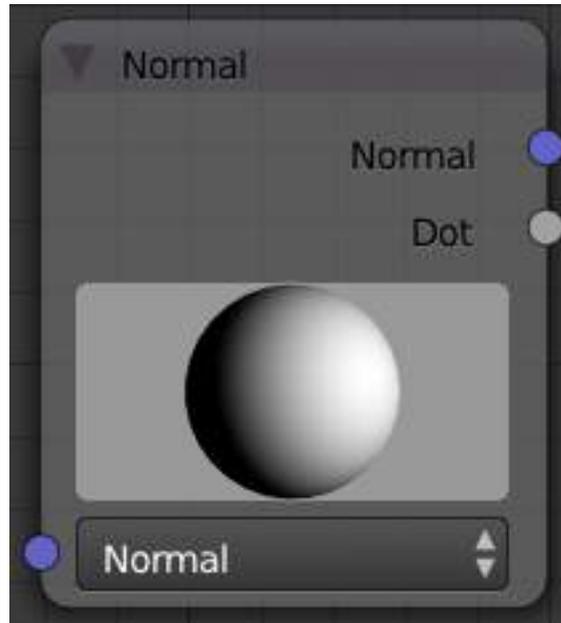


Fig. 2.1754: Normal node

**Normal Node** The Normal node generates a normal vector and a dot product. Click and Drag on the sphere to set the direction of the normal.

This node can be used to input a new normal vector into the mix. For example, use this node as an input to a Color Mix node. Use an Image input as the other input to the Mixer. The resulting colored output can be easily varied by moving the light source (click and dragging the sphere).

The (face) normal is the direction of the face in relation to the camera. You can use it to do the following:

- Use this node to create a fixed direction → output *Normal*.
- Calculate the *Dot* -Product with the *Normal* -Input. The *Dot* -Product is a scalar value (a number).
  - If two normals are pointing in the same direction the *Dot* -Product is 1.
  - If they are perpendicular the *Dot* -Product is zero (0).
  - If they are antiparallel (facing directly away from each other) the *Dot* -Product is -1. *And you never thought you would use the Vector Calculus class you took in college - shame on you!*

So now we can do all sorts of things that depends on the viewing angle (like electron scanning microscope effect). And the best thing about it is that you can manipulate the direction interactively.

---

**Note:** One caveat

The normal is evaluated per face, not per pixel. So you need enough faces, or else you don't get a smooth result

**Inputs**

**Normal** 3D-direction of the face in relation to the camera. The value can be provided by another node or set manually.

**Outputs**

**Normal** Fixed 3D-direction, combined by the node.

**Dot** Scalar value (a number), combined by the node.



Fig. 2.1755: Interactive Normal node preview

**Controls**

**Interactive node preview** Allows click and drag on the sphere in node center to set the direction of the normal.

**Mapping Node** Essentially mapping node allows the user to modify a mapping of system of 3D-coordinates. Typically used for modifying texture coordinates.

Mapping can be rotated and clamped if desired.

**Inputs**

**Vector** The input vector (3D-direction in relation to the camera) of some the coordinates' mapping. The value can be provided by another node or set manually.

**Outputs**

**Vector** The output vector, combined by the node.

**Controls** The controls of the node have been ordered in X, Y, Z order. If you want to use the clamping options, try enabling Min and Max.

**Vector type selector** Type of vector that the mapping transforms.

**Texture** Transform a texture by inverse mapping the texture coordinates.

**Point** Transform a point.

**Vector** Transform a direction vector.

**Normal** Transform a normal vector with unit length.

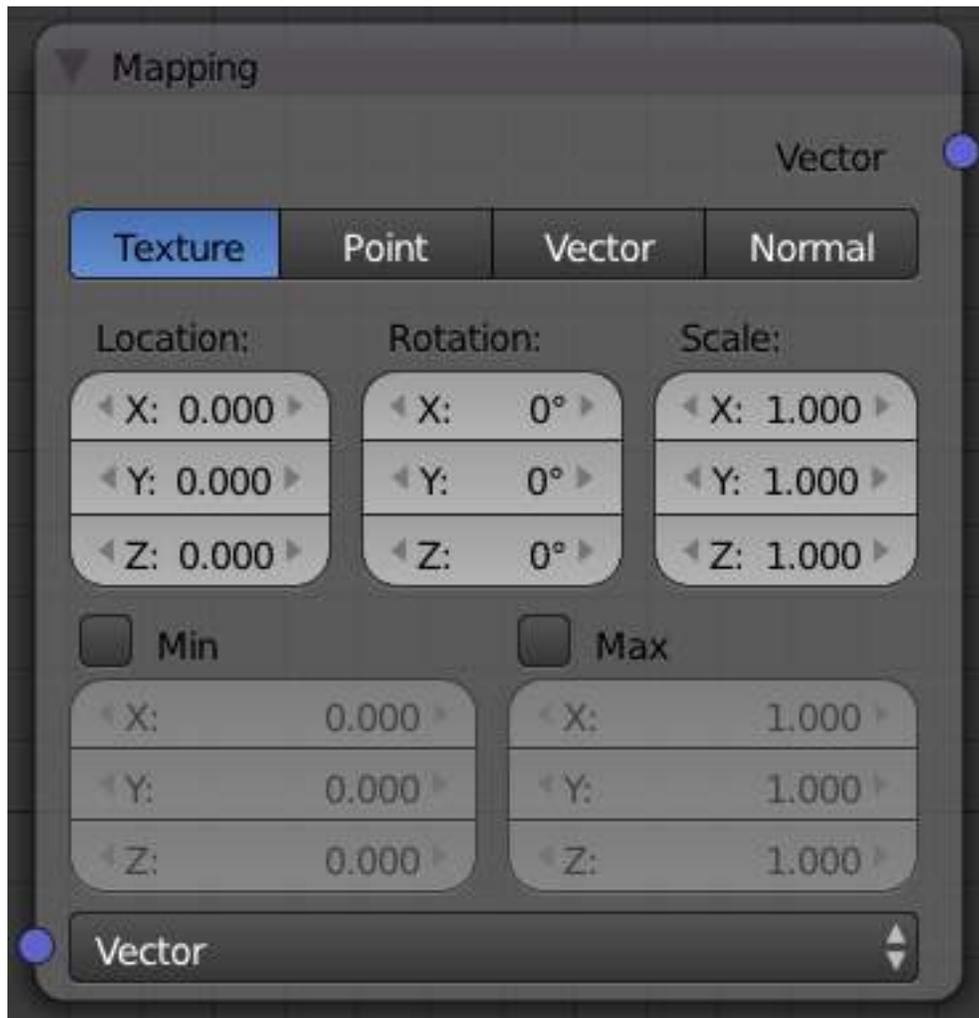


Fig. 2.1756: Mapping node

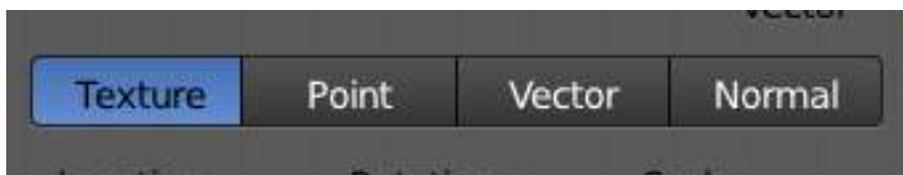


Fig. 2.1757: Mapping Node Vector Types controls

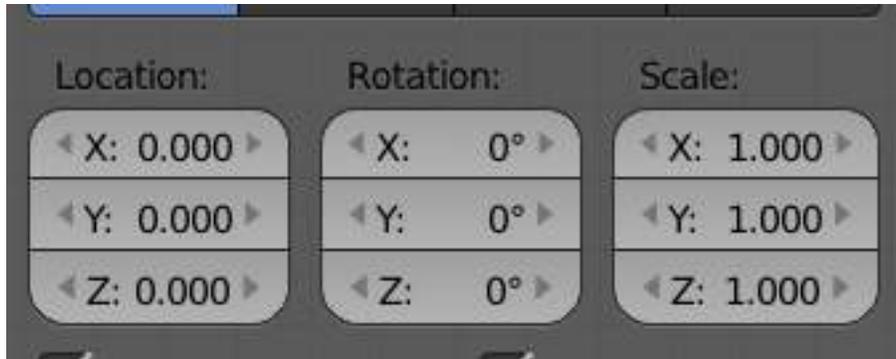


Fig. 2.1758: Mapping Node Transforms controls

**Location** Transform position vector.

**Rotation** Transform rotation vector.

**Scale** Transform scale vector.

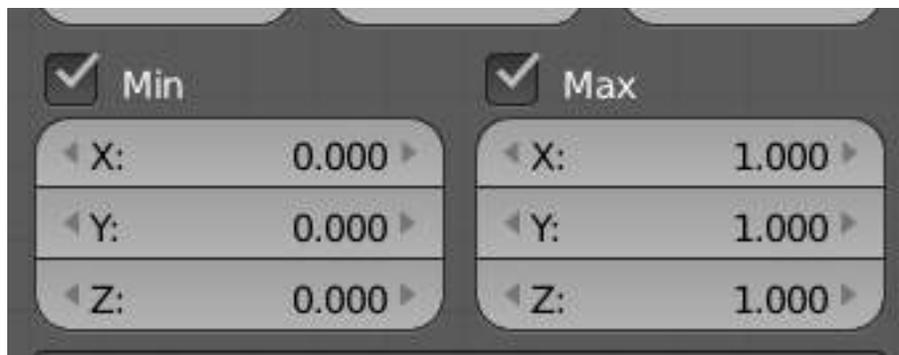


Fig. 2.1759: Mapping Node Clipping controls

**Min** Minimum clipping value.

**Max** Maximum clipping value.

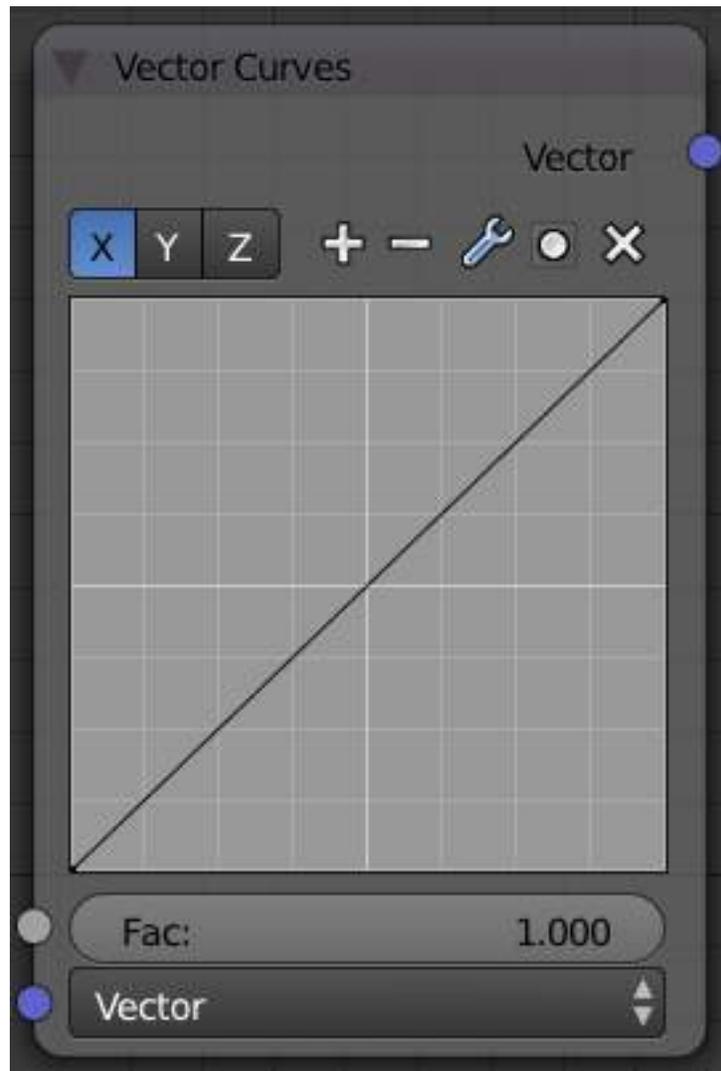


Fig. 2.1760: Vector Curves node

**Vector Curves** The Vector Curves node maps an input vector X, Y, and Z components to a diagonal curve. Use this node to remap a vector value using curve controls.

**See also:**

- Read more about using the *Curve Widget*.

**Inputs**

**Fac:** Factor. The degree of node's influence in node tree. The value can be provided by another node or set manually.

**Vector** The input vector (3D-direction in relation to the camera). The value can be provided by another node or set manually.

**Outputs**

**Vector** The output vector, combined by the node.

**Material Convertor Nodes**

As the name implies, these nodes convert the colors in the material in some way.



Fig. 2.1761: ColorRamp node

**ColorRamp Node** The ColorRamp Node is used for mapping values to colors with the use of a gradient. It works exactly the same way as a [Colorband for textures and materials](#), using the Factor value as a slider or index to the color ramp shown, and outputting a color value and an alpha value from the output sockets.

By default, the ColorRamp is added to the node map with two colors at opposite ends of the spectrum. A completely black black is on the left (Black as shown in the swatch with an Alpha value of 1.00) and a whitewash white is on the right.

To select a color, **LMB** click on the thin vertical line/band within the colorband. The example picture shows the black color selected, as it is highlighted white. The settings for the color are shown above the colorband as (left to right): color swatch, Alpha setting, and interpolation type.

### Inputs

**Fac:** Factor. The degree of node's influence in node tree. The value can be provided by another node or set manually.

### Outputs

**Color** Value of the color, combined by the node.

**Alpha** Value of the alpha, combined by the node.



Fig. 2.1762: Add a new mark to the center of the colorband with the default color (neutral gray).



Fig. 2.1763: Remove the currently selected mark from the colorband.



Fig. 2.1764: Flip the colorband.

### Controls

**Interpolation** Various modes of interpolation between marker's values can be chosen in the Interpolation menu:

**Ease** Ease by quadratic equation.

**Cardinal** Cardinal.

**Linear** Linear (default). A smooth, consistent transition between colors.

**B-Spline** B-Spline.

**Constant** Constant.

**Colorband** Contain a gradient through a sequence of many colors (with alpha), each color acting across a certain position in the spectrum.

**Color Selector** Allows set color and alpha values for each marker.

See more details about node controls' functions [here](#).

**RGB to BW Node** This node converts a color image to black-and-white.

### Inputs

**Color:** Input color value. Includes a color swatch, allowing you to select the color directly on the node.

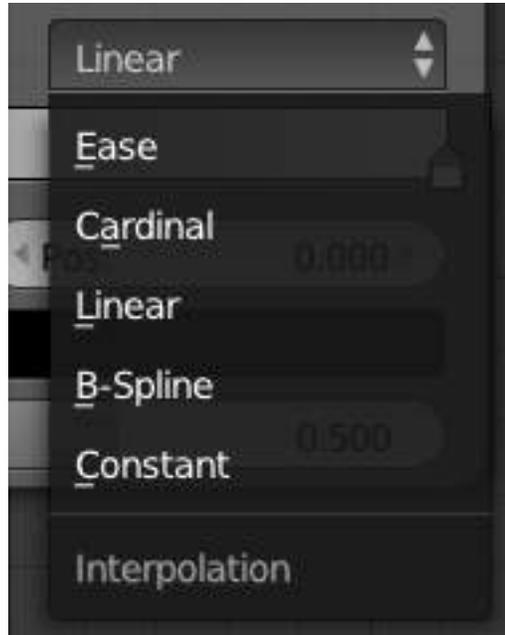


Fig. 2.1765: Modes of interpolation between marker's values color ramp



Fig. 2.1766: Colorband

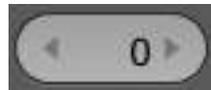


Fig. 2.1767: The number of the active mark.



Fig. 2.1768: *Pos*. The position of the active color mark in the colorband (range 0.0–1.0). The position of the color marks can also be changed by LMB dragging them in the colorband.



Fig. 2.1769: Color swatch to color selection for a mark

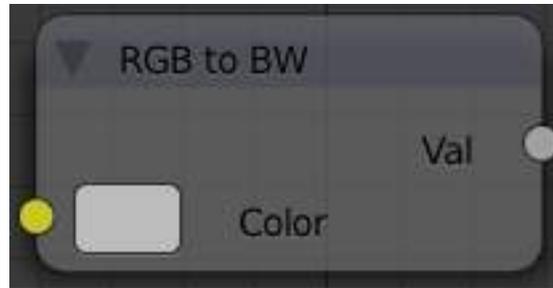


Fig. 2.1770: RGB to BW node

### Outputs

**Value** Black-and-white value of the input color, converted by the node.

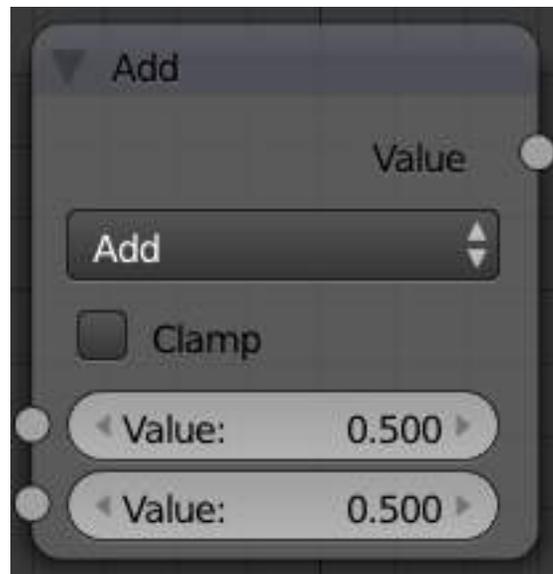


Fig. 2.1771: Math node

**Math Node** This node performs the selected math operation on an image or buffer. All common math functions are supported. If only an image is fed to one Value socket, the math function will apply the other Value consistently to every pixel in producing the output Value. Select the math function by clicking the up-down selector where the “Add” selection is shown.

### Inputs

**Value** Input value 1 (upper). The value can be provided by another node or set manually.

**Value** Input value 2 (lower). The value can be provided by another node or set manually.

### Outputs

**Value** Output value, converted by the node.

**Controls**

**Clamp** Clamps the result between 0 and 1.

**Operation** Selector the math function for conversion.

**Add** Add the two inputs

**Subtract** Subtract input 2 from input 1

**Multiply** Multiply the two inputs

**Divide** Divide input 1 by input 2

**Sine** The sine of input 1 (degrees)

**Cosine** The cosine of input 1 (degrees)

**Tangent** The tangent of input 1 (degrees)

**Arcsine** The arcsine (inverse sine) of input 1 (degrees)

**Arccosine** The arccosine (inverse cosine) of input 1 (degrees)

**Arctangent** The arctangent (inverse tangent) of input 1 (degrees)

**Power** Input 1 to the power of input 2 ( $\text{input1}^{\text{input2}}$ )

**Logarithm** Log base input 2 of input 1

**Minimum** The minimum of input 1 and input 2

**Maximum** The maximum of input 1 and input 2

**Round** Rounds input 1 to the nearest integer

**Less Than** Test if input 1 is less than input 2, returns 1 for true, 0 for false

**Greater Than** Test if input 1 is greater than input 2, returns 1 for true, 0 for false

**Modulo** Division of input 1 by input 2 with remainder.

**Absolute** Always return non-negative value from any operation input 2 between input 1.

**Vector Math Node** This node performs the selected math operation on vectors. Select the math function by clicking the up-down selector where the “Add” selection is shown.

**Inputs**

**Vector** Input vector 1 (upper). The value can be provided by another node or set manually.

**Vector** Input vector 2 (lower). The value can be provided by another node or set manually.

**Outputs**

**Vector** Output vector, converted by the node.

**Value** Output value, converted by the node.



Fig. 2.1772: Vector Math node

### Controls

**Operation** Selector the math function for conversion.

**Add** Adding input 1 and 2.

**Subtract** Subtracting input 1 and 2.

**Average** Averaging input 1 and 2.

**Dot Product** Algebraic operation that takes two equal-length sequences of vectors 1 and 2 and returns a single number.  
Result - scalar.

**Cross Product** Geometric binary operation on two vectors 1 and 2 in three-dimensional space. It results in a vector which is perpendicular to both and therefore normal to the plane containing them. Result - vector.

**Normalize** Normalizing input 1 and 2.

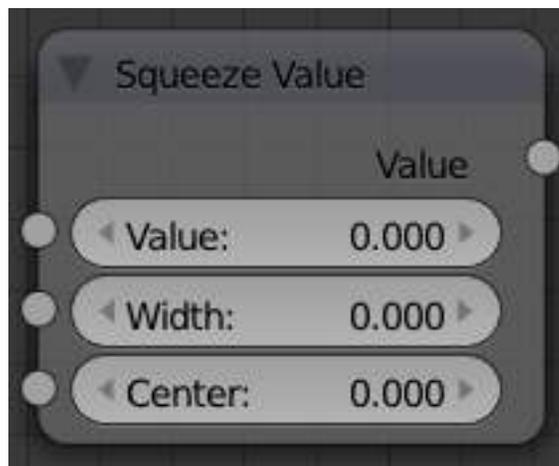


Fig. 2.1773: Squeeze Value node

**Squeeze Value Node** This node is used primarily in conjunction with the Camera Data node used. The camera data generate large output values, both in terms of the depth information as well as the extent in the width. With the squeeze Node high output values to an acceptable material for the node degree, ie to values between 0.0 - 1.0 scaled down.

### Inputs

**Value** Any numeric value. The value can be provided by another node or set manually.

**Width** Determines the curve between sharp S-shaped (width = 1) and stretched (Width = 0.1). Negative values reverse the course. The value can be provided by another node or set manually.

**Center** The center of the output value range. This input value is replaced by the output value of 0.5. The value can be provided by another node or set manually.

### Outputs

**Value** A value between 0 and 1, converted by the node.



Fig. 2.1774: Separate RGB node

**Separate RGB Node** This node separates an image into its red, green, blue channels - traditional primary colors, also broadcast directly to most computer monitors.

### Inputs

**Image** Input color value. Includes a color swatch, allowing you to select the color directly on the node.

### Outputs

**R** Value of the red color channel, separated out by the node.

**G** Value of the green color channel, separated out by the node.

**B** Value of the blue color channel, separated out by the node.

**Combine RGB Node** This node combines a color (image) from separated red, green, blue channels.

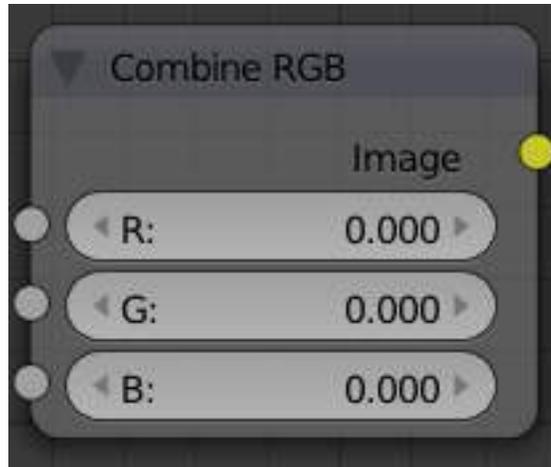


Fig. 2.1775: Combine RGB node

### Inputs

**R** Input value of red color channel. The value can be provided by another node or set manually.

**G** Input value of green color channel. The value can be provided by another node or set manually.

**B** Input value of blue color channel. The value can be provided by another node or set manually.

### Outputs

**Image** Output value of the color, combined by the node.



Fig. 2.1776: Separate HSV node

**Separate HSV Node** This node separates an image into image maps for the hue, saturation, value channels. Three values, often considered as more intuitive than the RGB system (nearly only used on computers)

Use and manipulate the separated channels for different purposes; i.e. to achieve some compositing/color adjustment result. For example, you could expand the Value channel (by using the multiply node) to make all the colors brighter. You could make an image more relaxed by diminishing (via the divide or map value node) the Saturation channel. You could isolate a specific range of colors (by clipping the Hue channel via the Colorramp node) and change their color (by the Add/Subtract mix node).

**Inputs**

**Color** Input color value. Includes a color swatch, allowing you to select the color directly on the node.

**Outputs**

**H** Value of the **hue** color channel, separated out by the node (choose a color of the rainbow).

**S** Value of the saturation color channel, separated out by the node (the *quantity* of hue in the color (from desaturate - shade of gray - to saturate - brighter colors)).

**V** Value of the value color channel, separated out by the node (the **luminosity** of the color (from 'no light' - black - to 'full light' - 'full' color, or white if Saturation is 0.0)).

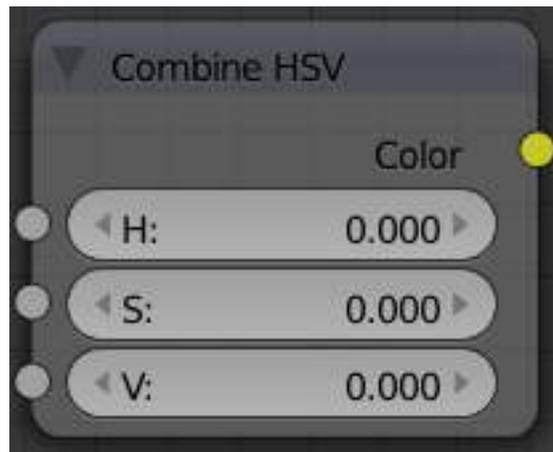


Fig. 2.1777: Combine HSV node

**Combine HSV Node** This node combines a color from separated hue, saturation, value color channels.

**Inputs**

**H** Input value of hue color channel. The value can be provided by another node or set manually.

**S** Input value of saturation color channel. The value can be provided by another node or set manually.

**V** Input value of value color channel. The value can be provided by another node or set manually.

**Outputs**

**Color** Output value of the color, combined by the node.

**Options**

**Materials** Materials can be linked to objects and Object's data in the materials panel, of the Shading/Material context. Here is where you can manage how materials are linked to objects, meshes, etc. and activate a material for editing in the rest of the panels.

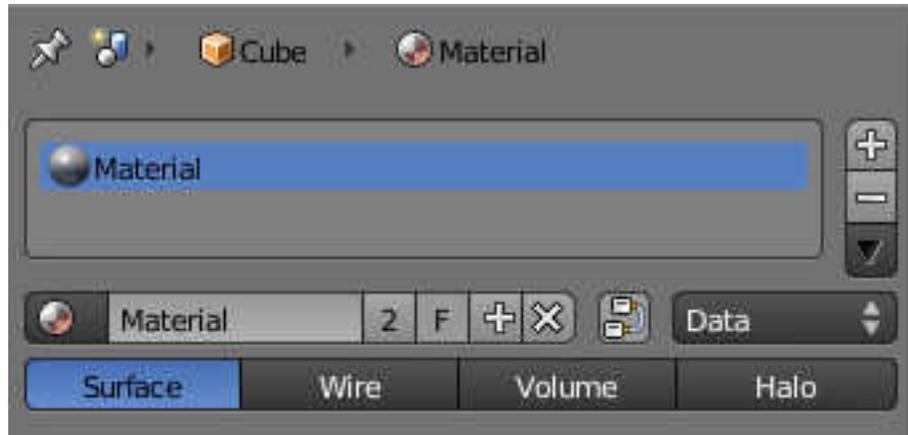


Fig. 2.1778: Material panel

**Context** At the top of the material menu a list of icons explains the context in which the material is being edited. In the example above, the material *Material* is linked to the object *Cube* which is linked to the scene *Scene*.

By toggling the pin symbol on the left side on and off, Blender can be told to display only the selected material or to follow context.

**Material slots** With a material linked or created, one or several material slots can be created and further options become available:

**Plus sign** Add a new material slot or copy the one selected

**Minus sign** Remove selected material slot

**Down arrow** Copy and paste the selected material slot

**Multiple materials** Meshes can handle having more than one material. Materials can be mapped on a per-face basis, as detailed on the [Multiple Materials](#) page. In edit mode, the following tools appear:

**Assign** Assign the material in the selected material slot to selected vertices

**Select** Select vertices assigned to the selected material slot

**Deselect** Deselect vertices assigned to the selected material slot



Fig. 2.1779: Link material to object or to object's data

### Material naming and linking

**Material's name field** click into this field to rename your material

**Number of users (number field)** The number of objects or object's data that use the material. This material is linked between the various objects, and will update across all of them when edited. Clicking this number will make a 'single user copy', duplicating the material, with it linked only to the active object/object's data.

**F (Fake user)** Gives the material a ‘fake user’, to keep the material datablock saved in the .blend file, even if it has no real users.

**Plus sign** Add a new material.

**X sign** Remove link to this material.

**Nodes** Designates this material to be a material node noodle, and not from the Material/Ramps/Shaders settings.

**Datablock links** The Link pop-up menu has two choices, Data and Object. These two menu choices determine whether the material is linked to the object or to the data, (in this case, the mesh). The Data menu item determines that this material will be linked to the mesh’s datablock which is linked to the object’s datablock. The Object menu item determines that the material will be linked to the object’s data block directly.

This has consequences of course. For example, different objects may share the same mesh datablock. Since this datablock defines the shape of the object any change in edit mode will be reflected on all of those objects. Moreover, anything linked to that mesh datablock will be shared by every object that shares that mesh. So, if the material is linked to the mesh, every object will share it.

On the other hand, if the material is linked directly to the object datablock, the objects can have different materials and still share the same mesh.

Short explanation: If connected to the object, you can have several instances of the same obData using different materials. If linked to mesh data, you can’t.

**Material type** Material added in edit mode These toggles tell Blender where this material fits into the Render Pipeline, and what aspects of the material are to be rendered.

**Surface** Render object as a surface

**Wire** Render the edges of faces as wires (not supported in ray tracing)

**Volume** Render object as a volume. See *Volume Material*

**Halo** Render object as halo particles. See *Halo Material*

**Material Properties Overview** The usage of each section of the material properties are detailed in the next section.

**Surface and Wire materials** Surface material types are the most common materials. They represent objects with a defined surface.

Wire materials simply turn all of an object’s edges into rods, which then become renderable, but uses the same shading options as surface materials.

**Preview** This is a preview of the current material mapped on to one of several objects.

- Flat Plane
- Sphere
- Cube
- Monkey
- Strands
- Large Sphere with Sky

See [Preview](#)

**Diffuse** Diffuse shading simulates light hitting a surface and bouncing off in a very wide angle. You can set the color of the diffuse shading, and set what model is used for the diffuse calculation.

See [Diffuse Shaders](#)

**Specular** Specularity simulates reflections of light sources, that are often sharp, bright spots. You can set the color of the specular shading, and set what model is used for the specular calculation.

See [Specular Shaders](#)

## Shading

**Emit** Adds extra illumination, as if the material is glowing.

**Ambient** Sets the global ambient light the material receives

**Translucency** Amount of shading on the back side that shows through. Use to simulate thin objects, like leaves or paper.

**Shadeless** This disables the calculation of any shading, so only color information is visible. This essentially makes it a “surface shader”

**Tangent Shading** Use the material’s tangent vector instead of the normal for shading - for anisotropic shading effects (e.g. soft hair and brushed metal). This shading was introduced in 2.42, see also settings for strand rendering in the menu further down and in the Particle System menu.

**Cubic Interpolation** Use cubic interpolation for diffuse values, for smoother transitions between light areas and dark areas

**Transparency** Set options for objects in which light can pass through

See [Transparency](#)

**Mirror** Here you can set options for materials that are reflective

See [Mirror](#)

**Subsurface Scattering** Subsurface scattering simulates semi translucent objects in which light enters, bounces around, then exits in a different place. Examples are candles, human skin, cheese, etc.

See [Subsurface Scattering](#)

**Strand** These settings are used when rendering the material on fur or hair

See [Strands](#)

## Options

**Traceable** Allows material to be calculated raytracing, for reflections and refractions.

**Full Oversampling** Forces material to render full shading and textures for all Anti-Aliasing Samples.

**Sky** Renders material with no alpha, replacing the background with the sky

**Use Mist** Uses Mist with this material.

**Invert Z Depth** Renders materials faces with an inverted Z buffer.

**Z Offset** If using Invert Z Depth, this is an artificial offset to z values.

**Light Group** Limit material’s lighting calculation to a specific light group

**Exclusive** Material uses light group exclusively

**Face Textures** Replaces object's base color with color from face assigned image textures.

**Face Textures Alpha** Replaces object's base alpha value with alpha from face assigned image textures.

**Vertex Color Paint** Replaces object's base color with vertex colors.

**Vertex Color Light** Adds vertex color as additional light.

**Object Color** Modulate the result with a per object color.

## Shadow

**Receive** Allows the material to receive shadows cast by other objects

**Receive Transparent** Allows material to receive transparent shadows cast by other transparent objects.

**Cast Only** Causes objects with the material to only cast a shadow, and not appear in renders.

**Casting Alpha** Sets the Alpha of shadow casting. Used for irregular and deep shadow buffering.

**Shadows Only** Renders shadows as materials alpha value, making materials transparent, except for shadowed areas.

**Shadow Only Type** Set the type of shadows used when Shadows Only is enabled

- Shadow and Distance
- Shadow Only
- Shadows and Shading

**Cast Buffer Shadow** Allows material to cast shadows from buffer lamps.

**Buffer Bias** Factor to multiply shadow buffer by.

**Auto Ray Bias** Prevents raytraced shadow errors on surfaces with smooth normals

**Ray Bias** Shadow raytracing bias value to prevent terminator artifacts on shadow boundary.

**Cast Approximate** Allow material to cast shadows when using Approximate Ambient Occlusion}}

**Volume Material** Volume materials represent volumes of tiny particles, like clouds or smoke. They are very different from standard materials, but are detailed in the [Volume Page](#).

**Halo Material** Halo materials renders each of the objects points as glowing dots. This is a useful material for simulating particle effects or lens flares. They are detailed on the [Halo Page](#).

## Vertex Painting

Vertex Painting is a simple way of painting color onto an object, by directly manipulating the color of vertices, rather than textures, and is fairly straightforward.

When a vertex is painted, the color of the vertex is modified according to the rules of the 'brush'. The color of all visible planes and edges attached to the vertex are then modified with a gradient to the color of the other connected vertices. (Note that the color of non-visible faces are not modified).

Vertex colors can be painted by first going into Edit Mode, then switching to *Vertex Paint Mode*; however, it will not show up in the render unless you check "Vertex Color Paint" in the Materials [Options Panel](#).

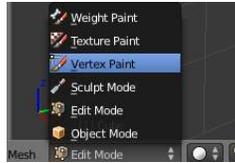


Fig. 2.1780: Vertex Painting Mode

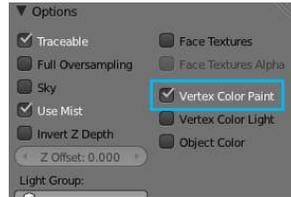


Fig. 2.1781: Check this box

**Settings** The Tools Shelf, shortcut T contains most of the options for vertex painting. The following sections describe the controls in each of the available panels.

### Brush

**Brush Datablock** The image, name panel and color selector at the top allows you to select brush presets, rename brushes, as well as add custom brushes, and delete them.

**Radius** Set the radius of the brush

**Strength** Set the strength of the brush's effect.

### Blend menu

**Mix** Mixes RGB values. When set to a strength of 1.0, it will cover the underlying “paint”.

**Add** Adds RGB values. Will eventually turn the entire object white as RGB values accumulate to 1.0-1.0-1.0: Pure White.

**Subtract** Subtracts RGB values. Usually results in Black.

**Multiply** Multiplies brush colors by the vertex colors.

**Blur** Blurs vertex colors.

**Lighten** Lightens the color of the vertices.

**Texture** Use the texture selector at the bottom of the paint panel to select a pre-loaded image or procedural texture to use as your brush pattern. Note that in order to use it, you must have a placeholder material defined, and that particular texture defined using the Material and Texture buttons. It is not necessary to have that material or texture applied to any mesh anywhere; it must only be defined.

**Brush Mapping Mode** Sets how the texture is applied to the brush

**View Plane** In 2D painting, the texture moves with the brush

**Tiled** The texture is offset by the brush location

**3D** Same as tiled mode

**Stencil** Texture is applied only in borders of the stencil.

**Random** Random applying of texture.



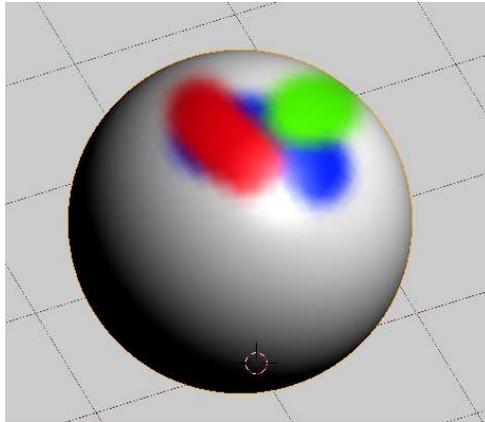


Fig. 2.1783: Mix overlay with full strength

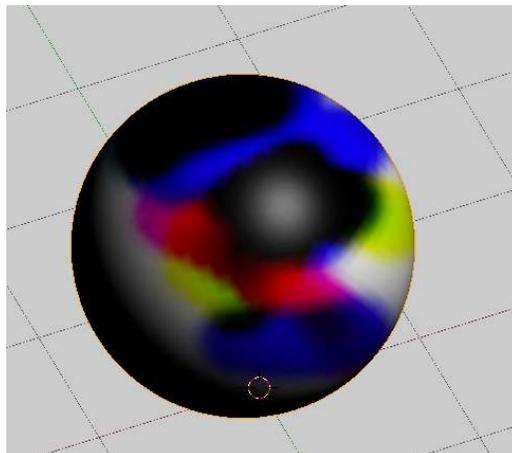


Fig. 2.1784: Subtract with full strength

**Darken** Darkens the color of the vertices.

**Angle** This is the rotation angle of the texture brush. It can be changed interactively via `Ctrl-F` in the 3D view. While in the interactive rotation you can enter a value numerically as well. Can be set to:

**User** Directly input the angle value.

**Rake** Angle follows the direction of the brush stroke. Not available with 3D textures.

**Random** Angle is randomized.

**Offset** Offset the texture in x, y, and z.

**Size** Set the scale of the texture in each axis.

## Stroke

**Stroke Method** Allows set the way applying strokes.

**Airbrush** Flow of the brush continues as long as the mouse click is held, determined by the *Rate* setting. If disabled, the brush only modifies the color when the brush changes its location.

**Rate** Interval between paints for airbrush

**Space** Creates brush stroke as a series of dots, whose spacing is determined by the *Spacing* setting.

**Spacing** Represents the percentage of the brush diameter. Limit brush application to the distance specified by spacing.

**Dots** Apply paint on each mouse move step

**Jitter** Jitter the position of the brush while painting

**Smooth stroke** Brush lags behind mouse and follows a smoother path. When enabled, the following become active:

**Radius** Sets the minimum distance from the last point before stroke continues.

**Factor** Sets the amount of smoothing.

**Input Samples** Average multiple input samples together to smooth the brush stroke.

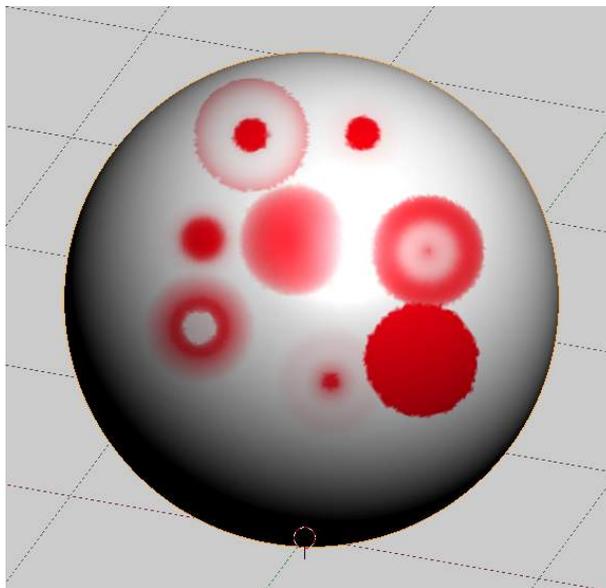


Fig. 2.1785: Various brush curves

**Curve** Brush Curves affect how strongly the color is applied depending on distance from the center of the brush. In other words, they allow you to edit the Falloff of the brush intensity.

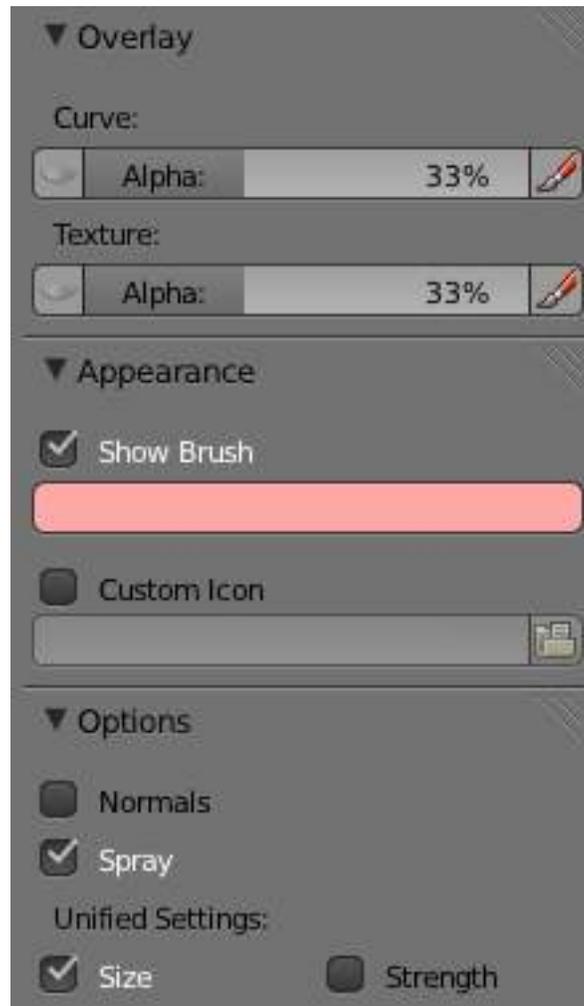


Fig. 2.1786: Options for vertex painting

### Options

**Overlay** Allows you to customize the display of curve and texture that applied to the brush.

**Appearance** Allows you to customize the color of the brush radius outline, as well as specify a custom icon.

### Options

**Normals** Applies the Vertex Normal before painting. This does not usually affect painting.

**Spray** Continues painting for as long as the mouse is held.

### Unified Settings

**Size** All brushes use the same size.

**Strength** All brushes use the same strength.

## Wire Render



Fig. 2.1787: Wire Render

The Wire Render option in the Materials section provides a way of showing a rendered image of the edges in an object. Each edge is rendered as a single-pixel image of the edges which make up the mesh. The colors, alpha and other relevant properties of the lines are selected with the same control panels as provided by the Surface rendered image.

## Volume Rendering

Volume rendering is a method for rendering light as it passes through participating media, within a 3d region. The implementation in Blender's sim-physics branch is a physically based model, which represents the various interactions of light in a volume relatively realistically.

The light bounces around off the various molecules, being scattered or absorbed, until some light passes through the volume and reaches the camera. In order for that volume to be visible, the renderer must figure out how much material the light has passed through and how it has acted and reacted within that volume, the volume object needs to contain a 3D region of space, for example a *manifold* closed mesh, such as a cube, not just a flat surface like a plane. To get an image, the renderer has to step through that region, and see how much 'stuff' is there (density) in order to see how light is absorbed or scattered or whatever. This can be a time consuming process since it has to check a lot of points in space and evaluate the density at each.

## Options

**Density** Many things can happen to the light as it passes through the volume, which will influence the final color that arrives at the camera. These represent physical interactions that happen in the real world, and most of these are dependent on the density of the volume, which can either be a constant density throughout, or varied, controlled by a texture. It is by controlling the density that one can get the typical 'volumetric' effects such as clouds or thick smoke.

**Density** The base density of the material - other density from textures is added on top

**Density Scale** A global multiplier to increase or decrease the apparent density. This can be useful for getting consistent results across different scene scales.

**Shading** When light enters a volume from an external source, it doesn't just pass straight through. Light gets scattered off tiny particles in the volume, and some proportion of that light reaches the camera. This property makes it possible to see light beams as they travel through a volume and are scattered towards the eye.

**Scattering** The amount of light that is scattered out of the volume. The more light that is scattered out of the volume, the less it will penetrate through the rest of the volume. Raising this parameter can have the effect of making the volume seem denser, as the light is scattered out quickly at the 'surface' of the volume, leaving the areas internal to the volume darker, as the light doesn't reach it.

Note in the examples below, the less light that is scattered out of the volume, the more easily it penetrates throughout the volume and to the shadow.



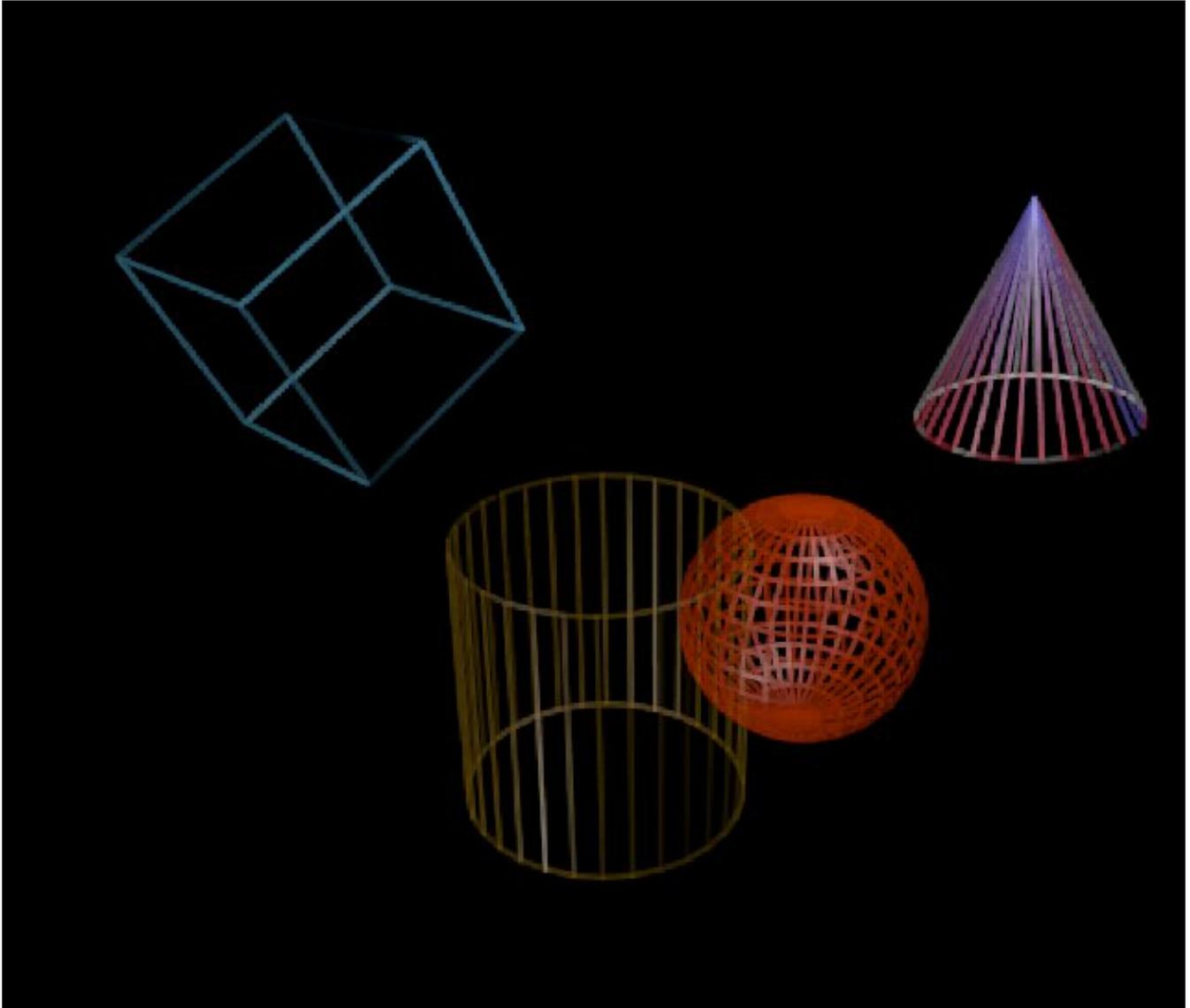


Fig. 2.1788: Wire Render

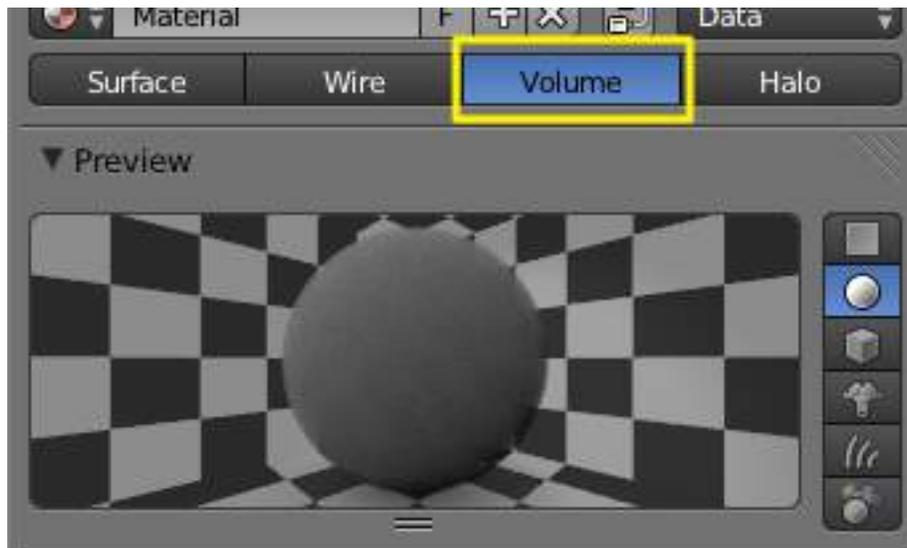


Fig. 2.1789: Activation volume rendering

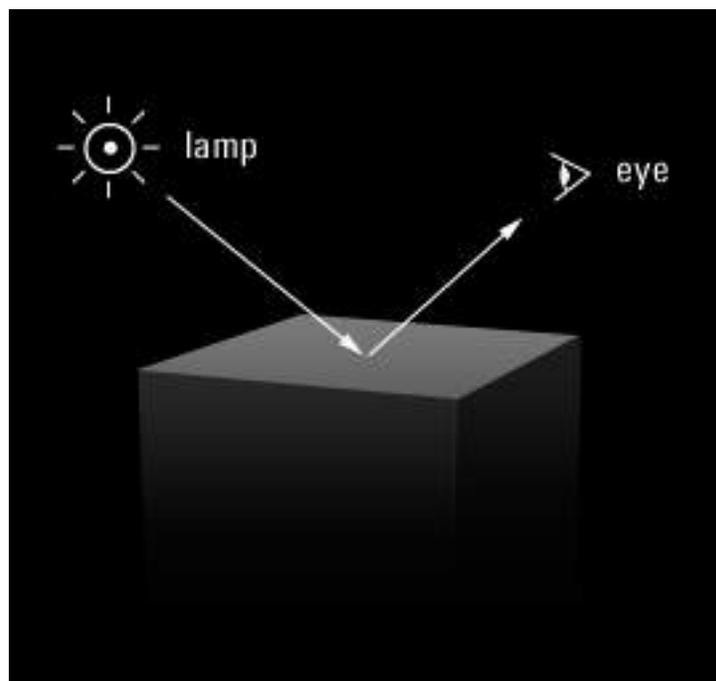


Fig. 2.1790: Solid rendering

The process of rendering a solid surface involves the camera finding a piece of geometry, then calculating the light that bounces from light sources (lamp objects, or other geometry), off the surface, and towards the camera. The light that arrives at the camera is the final color that's rendered.

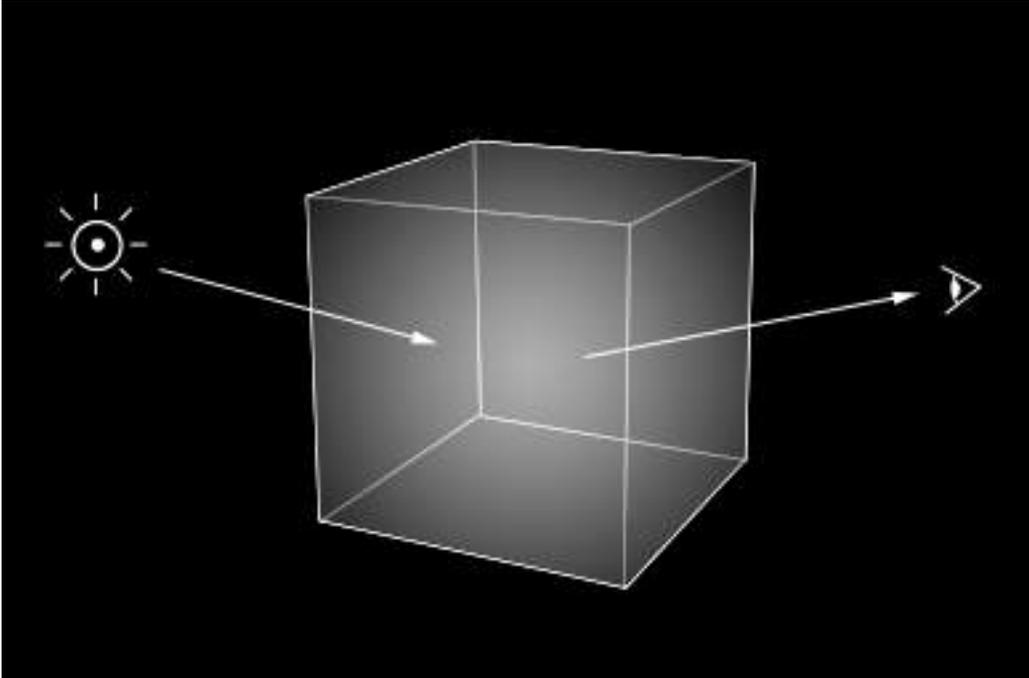


Fig. 2.1791: Volume rendering

Rendering a volume works differently. Light enters a 3D region of space (defined as the volume) that may be filled with small particles, such as smoke, mist or clouds.

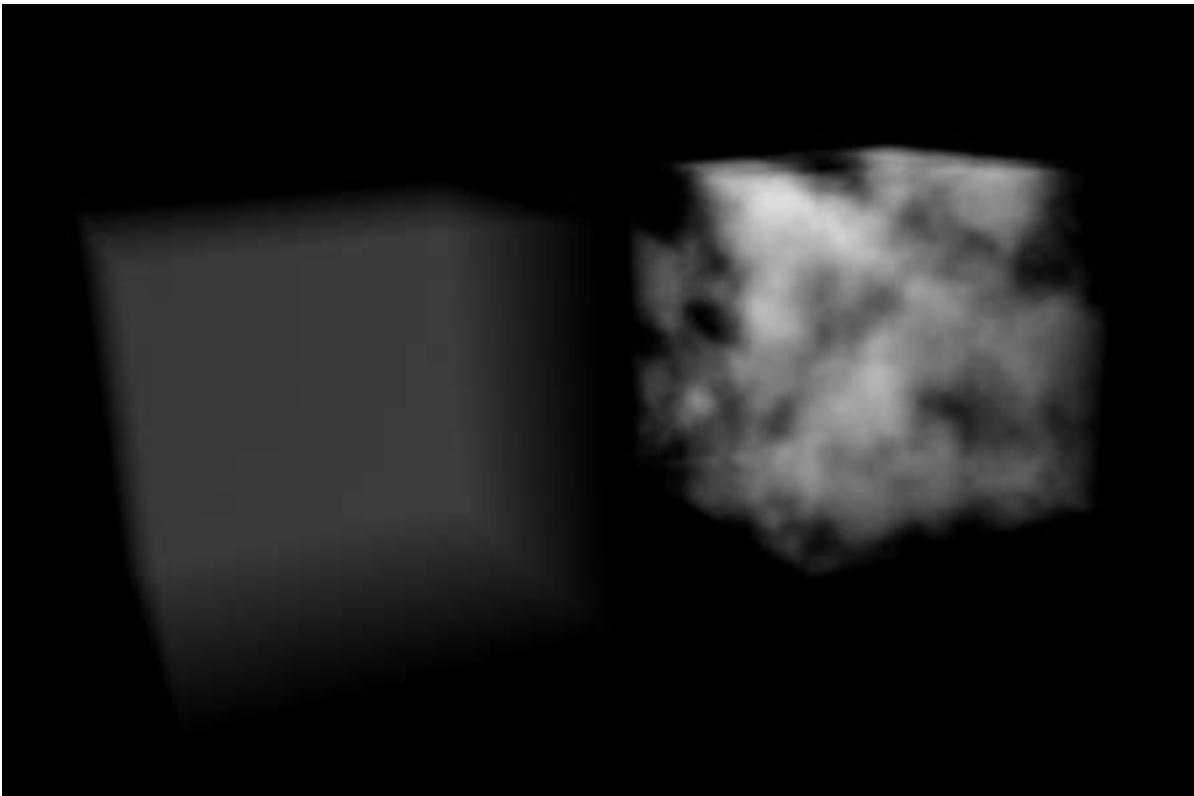


Fig. 2.1792: Constant density vs textured density



Fig. 2.1793: Density options



Fig. 2.1794: Spot lamp scattering in a constant volume

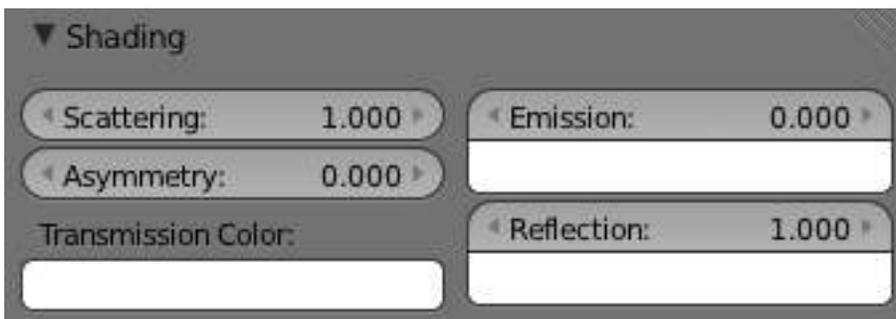


Fig. 2.1795: Shading options

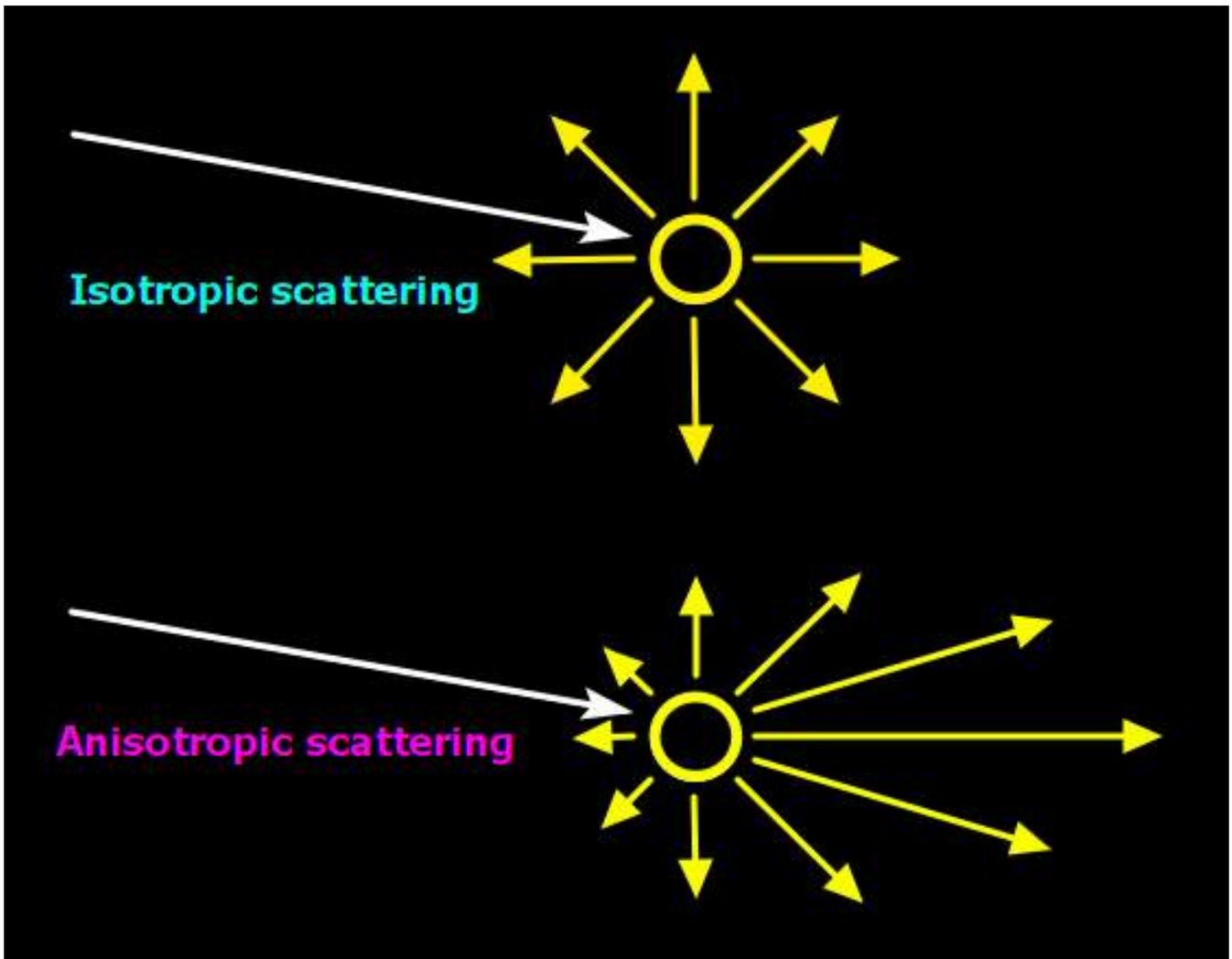


Fig. 2.1804: Isotropic and Anisotropic scattering

**Asymmetry** The default method for scattering light in a volume is for the light to be deflected evenly in all directions - known as Isotropic scattering. In real life different types of media can scatter light in different angular directions, known as Anisotropic scattering. Back-scattering means that light is scattered more towards the incoming light direction, and forward-scattering means it's scattered along the same direction as the light is travelling.

**Asymmetry** Asymmetry controls the range between back-scattering (-1.0) and forward-scattering (1.0). The default value of 0.0 gives Isotropic scattering (even in all directions).

**Transmission** Transmission is a general term for light that is transmitted throughout a volume.

This transmitted light can be the result of various different interactions, for example:

- the left over result of incoming light after it has reflected/scattered out of the volume
- the left over result of light after being absorbed by the volume (and converted to heat)

Here, the transmission color is used to set the end result color that light becomes after it is transmitted through the volume.

**Transmission Color** The resultant color of light that is transmitted through the volume.

Note in the examples below, as more light is scattered out of the volume, there is less available to be transmitted through.

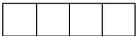


**Emission** Some volumes can emit light where there was none before, via chemical or thermal processes, such as fire. This light is generated from the volume itself and is independent of light coming from external sources.

Currently, this emitted light does not affect other volumes or surfaces (similar to surface material type, 'Emit' option).

**Emission Color** The color of light that is emitted by the volume.

**Emission** An intensity multiplier for the emitted color, for scaling up and down.



**Reflection** The 'reflection' parameters can be used to tint or scale the light that's scattered out of the volume. This only affects light that has come from lamps and been scattered out, it doesn't affect the color of transmitted or emitted light and is.

These settings are not physically correct because they don't conserve energy - the light scattering out doesn't affect the remaining light that is transmitted throughout the rest of the volume. For example, physically speaking, if the orange components of the light are scattered out of the volume towards the camera, only the inverse of that (blue) will remain to continue penetrating through the volume, causing the volume to take on a multi-colored appearance, which can be difficult to use. To make it a bit easier to plainly set the color of the volume, you can use the reflection parameters to quickly set an overall tint.

**Reflection Color** The color of light that is scattered out of the volume.

**Reflection** An intensity multiplier for the reflection, for scaling up and down.

**Hints** Ideally try to accomplish as much as you can with the other volume settings and lighting before using the reflection controls. If you stick to what's physically plausible, the material will act correctly, and be more predictable and usable in a wider range of lighting scenarios. Of course you can always break the rules too!



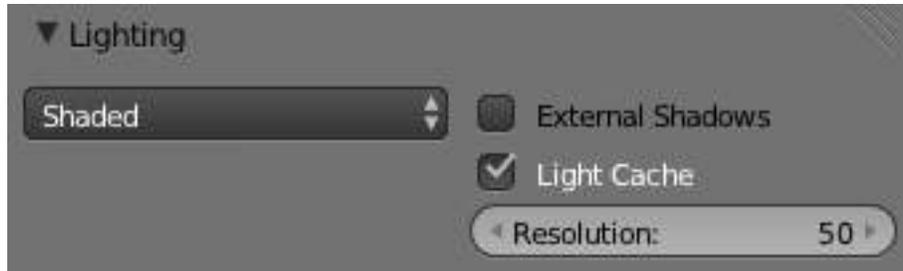


Fig. 2.1837: Lighting options

**Lighting** Several shading modes are available, providing a range of options between fast to render and physically accurate.

#### Lighting Mode

**Shadeless** Shadeless is the simplest, useful for thin, wispy mist or steam.

**Shadowed** Shadowed is similar, but with shadows of external objects.

**Shaded** Shaded uses a volumetric single-scattering method, for self-shading the volume as light penetrates through.

**Multiple Scattering** Allows multiple scatter calculations.

**Shaded+Multiple Scattering** Combines Shaded and Multiple Scattering functionality.

#### Shaded Options:

**External Shadows** Receive shadows from sources outside the volume (temporary).

**Light Cache** Pre-calculate the shading information into a voxel grid, speeds up shading at slightly less accuracy.

**Resolution** Resolution of the voxel grid, low resolutions are faster, high resolutions use more memory.

#### Multiple Scattering Options:

**Diffusion** Diffusion factor, the strength of the blurring effect.

**Spread** Proportional distance over which the light is diffused.

**Intensity** Multiplier for multiple scattered light energy.

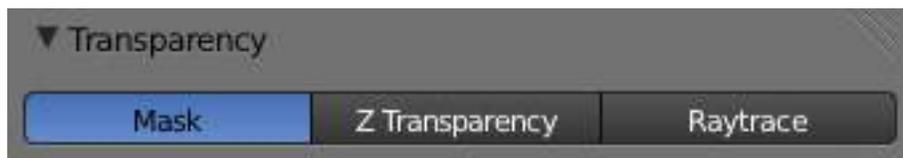


Fig. 2.1838: Transparency options

#### Transparency

**Mask** Mask the Background.

**Z Transparency** Use Alpha buffer for transparent faces.

**Raytrace** Use Raytracing for Transparent Refraction rendering.



Fig. 2.1839: Integration options

## Integration

**Step Calculation Method** Method of calculating the step through the volume.

**Randomized** Randomized method of calculating the step.

**Constant** Constant method of calculating the step.

**Step Size** Distance between subsequent volume depth samples. Step Sizes determine how noisy the volume is. Higher values result in lower render times and higher noise.

**Depth Cutoff** Stop ray marching early if transmission drops below this luminance - higher values give speedups in dense volumes at the expense of accuracy.



Fig. 2.1840: Material volume options

## Options

**Traceable** Allow this material to calculate raytracing.

**Full Oversample** Force this material to render full shading/textures for all anti-aliasing samples.

**Use Mist** Use mist with this material (in world settings).

**Light Group** Limit lighting of this material to lamps in this group.

**Exclusive** Material uses this group exclusively. Lamps are excluded from other scene lighting.

**Examples** <these are sandbox edits to the whole shading intro section of the wiki, which groups materials and textures, and gives us an entree into Volumetric shading. Note qualification of Mesh object. Need to investigate shading of other object types...>

Shading is the process and the code which enables an object to be seen in the final render output. Blender has four methods to shade a mesh object:

- Surface

- Volumetric
- Halo
- Wire

Surface shading indicates that the object is a tangible, skinned object that has a solid (but possibly pliable) surface, such as a chair, a sword, or a peach. The surface is described in terms of having a diffuse, specular, mirror, and transparency. It may also have a semi-transparent surface and something inside of it that scatters light, called sub-surface scattering. It may be reflective, such as chrome, smooth plastic, or metal, and may be partially transparent, such as glass, or liquid.

Volumetric shading treats the object as a volume of space that is filled with microscopic particles, such as a cloud, smoke, mist, fog, mystical spells, and steam. As light enters the volume, it is scattered by these particles, and some of that scattering reaches the eye/camera for us to see. The volume is described in terms of density, xxx. The particles may be uniformly colored but have a varying density within the volume, and so the shape may have darker areas. The density may be uniformly dispersed throughout the volume, or it may be clumped, giving a recognizable shape. Those microscopic particles may give off light themselves, as if they contained glowing embers or sparks, or were transmitting some energy field inside the cloud. That density may be driven by a particle system to create a well-defined jet or emission.

Halo shading turns each vertex of the object into a glob of light, an effect seen with sparks, pixie dust, glint, and sparkles from, for example, a diamond in bright sunlight. Halos can also be used to give a rough approximation of a lens flare, which is observed when a real camera lens looks directly at a bright light source such as the sun.

Wire shading renders each edge of the object as a thin line, like a wire cage, or net. Wire rendering is very fast and can be used as a proxy material for a more complicated surface to save time during intermediate renders.

There are two major components to shading: the Material and its Textures. The color that you see is a function of the light and the shading, so you need to also check out the lighting section as well. There are five types of objects in Blender that can be shaded: Mesh, Curve, Surface, Meta, and Text. The table below indicates which types of shading are available for each kind of object. Keep in mind that all types of non-mesh objects can be converted from their type to a Mesh, so, ultimately, all kinds of shading are available for all kinds of objects

Table 2.53: Shading available per Object type

Surface	Halo	Wire	Volumetric	no
Mesh	yes	full	yes	yes
Curve	if cyclic or extruded	no	no	no
Surface	yes	no	yes	no
Meta	yes	no	no	no
Text	yes	no	no	no

## Halo Rendering

Blender provides a set of materials which do not obey the face-shader paradigm and which are applied on a per-vertex rather than on a per-face basis. These are called *Halos* because you can see them, but they do not have any substance. They are like little clouds of light; although they are not really lights because they do not cast light into the scene like a lamp.

Halos come in very handy when creating certain special effects, when making an object glow, or when creating a viewable light or fog/atmospherics around an actual light.

**Options** To enable *Halos*, press the *Halo* button in the *Material* menu's top panel.

As you will see in the 3D View, the mesh faces are no longer rendered. Instead just the vertex is rendered, since that is where each halo will originate. Halos can be hard to find in a crowded scene, so name it well for easy location in the [outliner](#).

In the properties window, where we normally find the *Diffuse*, *Specular*, and *Shading* panels, we now see panels relative to the *Halo* characteristics:

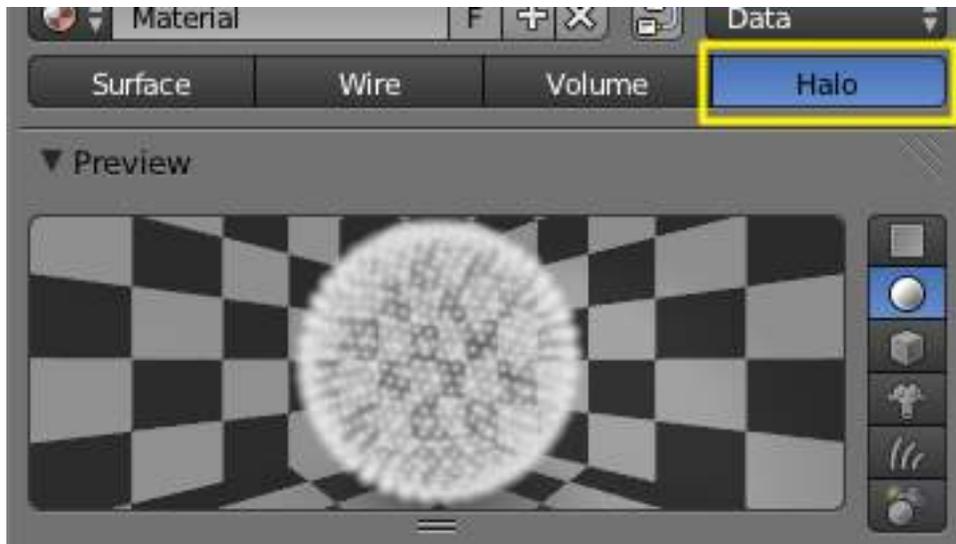


Fig. 2.1841: Activating halo rendering

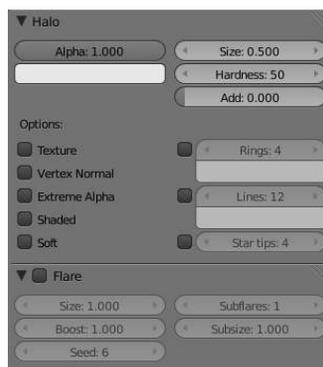


Fig. 2.1842: Halo panels

## Halo Panel

**Alpha** The transparency

**Color Swatch** The color of the halo itself

**Seed** If non-zero, randomizes the ring dimension and line location. To use, give any (integer) number to start the random-number generator.

**Size** Sets the dimension of the halo

**Hardness** Sets the hardness of the halo. Similar to specular hardness



Fig. 2.1843: Effect of Add

**Add** The *Add* slider determine how much the halo colors are ‘added to’, rather than mixed with, the colors of the objects behind and together with other halos. By increasing Add, the Halo will appear to light up objects that move behind it or through the Halo field.

**Texture** Gives halo a texture. By default, textures are applied to objects with Object coordinates and reflects on the halos by affecting their color, as a whole, on the basis of the color of the vertex originating the halo. Enable this feature to have the texture take effect *within* the halo, and hence to have it with varying colors or transparencies; this will map the whole texture to *every* halo. This technique proves very useful when you want to create a realistic rain effect using particle systems, or similar.

**Vertex Normal** Use the vertex normal to specify the dimension of the halo

**Extreme Alpha** Boosts alpha

**Shaded** Lets halo receive light and shadows from external objects

When shaded is enabled, the Halo will be affected by local light; a lamp will make it brighter and affect its diffuse color and intensity.

**Soft** Softens the edges of the halos at intersections with other geometry

In addition, several other special effects are available. To enable some or all of these effects, set the number of points/rings, or set the color of each effect individually:

**Rings** Adds circular rings around to the halo.

**Lines** Adds lines from the center of the halo.

**Star tips** Gives the halo a star shape.

You can not use color ramps. Lines, Rings and an assortment of special effects are available with the relevant toggle buttons, which include Flare, Rings, Lines, Star, Texture, Extreme Alpha, and Shaded. *Halo Variations* shows the result of applying a halo material to a single vertex mesh.

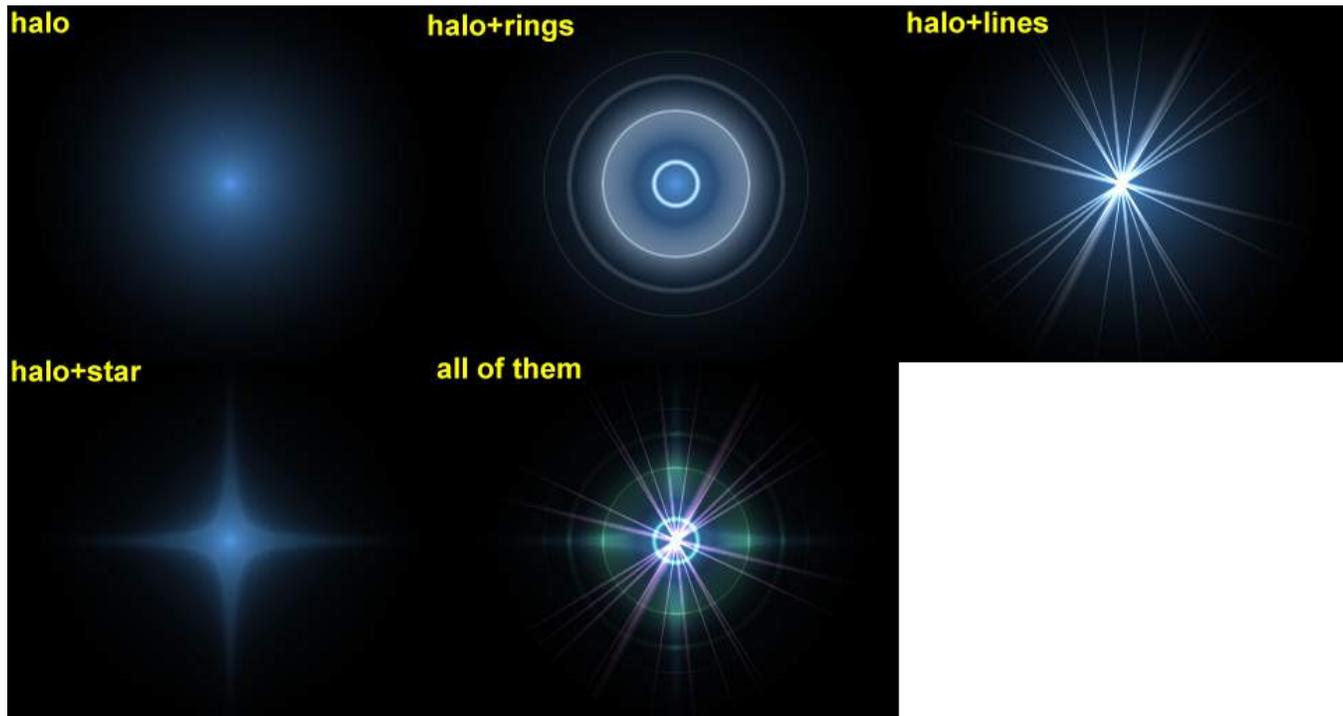


Fig. 2.1844: Halo Variations

The halo size, hardness and alpha can be adjusted with the pertinent sliders. These are very similar to traditional material settings

The *Add* slider determine how much the halo colors are ‘added to’, rather than mixed with, the colors of the objects behind and together with other halos. By increasing *Add*, the Halo will appear to light up objects that move behind it or through the Halo field.

To set the number of rings, lines, and star points independently, once they are enabled with the relative Toggle Button, use the Num Buttons *Rings:*, *Lines:* and *Star:*. Rings and lines are randomly placed and oriented, to change their pattern you can change the *Seed:* Num Button which sets the random numbers generator seed.

**Flare Panel** Enabling Flare Renders the halo as a lens flare

**Size** Sets the factor by which the flare is larger than the halo

**Boost** Give the flare extra strength.

**Seed** Specifies an offset in the flare seed table

**Subflares** Sets the number of subflares

**Subsize** Sets the dimensions of the subflares, dots, and circles



**Lens Flares** Our eyes have been trained to believe that an image is real if it shows artifacts that result from the mechanical process of photography. *Motion blur*, *Depth of Field*, and *lens flares* are just three examples of these artifacts. The first two are discussed in the *chapter\_rendering*; the latter can be produced with special halos. A simulated lens flare tells the viewer that the image was created with a camera, which makes the viewer think that it is authentic.

We create lens flares in Blender from a mesh object using first the *Halo* button and then the *Flare* options in the *Shaders* Panel of the material settings. Try turning on *Rings* and *Lines*, but keep the colors for these settings fairly subtle. Play with the *Flares*: number and *Fl. seed*: settings until you arrive at something that is pleasing to the eye. You might need to play with *Boost*: for a stronger effect (*Lens Flare settings*).

Note that this tool does not simulate the physics of photons traveling through a glass lens; it's just a eye candy.

Blender's lens flare looks nice in motion, and disappears when another object occludes the flare mesh.

**Halo Texturing** By default, textures are applied to objects with Object coordinates and reflects on the halos by affecting their color, as a whole, on the basis of the color of the vertex originating the halo. To have the texture take effect *within* the halo, and hence to have it with varying colors or transparencies press the *Texture* button; this will map the whole texture to *every* halo. This technique proves very useful when you want to create a realistic rain effect using particle systems, or similar.

Another Option is *Shaded*. When shaded is enabled, the Halo will be affect by local light; a lamp will make it brighter and affect its diffuse color and intensity.

## Examples

**Dotmatrix display** Let's use a halo material to create a dotmatrix display.

- To begin, add a grid with the dimensions 32x16. Then add a camera and adjust your scene so that you have a nice view of the billboard.
- Use a 2D image program to create some red text on a black background, using a simple and bold font (if you are a lazy lizard [I hope this not offensive, I just like how it sounds!], you can just save the picture below on your hard drive...). *Dot matrix image texture*. shows an image 512 pixels wide by 64 pixels high, with some black space at both sides.



Fig. 2.1845: Lens Flare



Fig. 2.1846: Dot matrix image texture.

- Add a material for the billboard, and set it to the type *Halo*. Set the *HaloSize* to 0.06 and when you render the scene you should see a grid of white spots.
- Add a Texture, then change to the Texture Buttons and make it an image texture. When you load your picture and render again you should see some red tinted dots in the grid.
- Return to the Material Buttons and adjust the *sizeX* parameter to about 0.5 then render again; the text should now be centered on the Billboard.
- To remove the white dots, adjust the material color to a dark red and render. You should now have only red dots, but the billboard is still too dark. To fix this enter EditMode for the board and copy all vertices using the `Shift-D` shortcut (take care not to move them!). Then adjust the brightness with the *Add* value in the MaterialButtons.



Fig. 2.1847: Dot Matrix display.

You can now animate the texture to move over the billboard, using the *ofsX* value in the *Texture* panel of the MaterialButtons. (You could use a higher resolution for the grid, but if you do you will have to adjust the size of the halos by shrinking them, or they will overlap. (*Dot Matrix display*)).

---

**Note:** Note about material indices

Halo materials only work when applied using the first material index. Any material(s) in a subsequent material index will not be rendered.

---

## Textures

### Introduction to Textures

In CGI, texture mapping is a method to add detail to surfaces by projecting images and patterns onto those surfaces. The projected images and patterns can be set to affect not only color, but also specularity, reflection, transparency, and even fake 3-dimensional depth. Most often, the images and patterns are projected during render time, but texture mapping is also used to sculpt, paint and deform objects.

In Blender, *Texture*s can be:

- applied to a *Material*
- applied to a *light*, that coming from lamp
- applied to the *World Background*
- applied to a *Brush*, see for example: - *Sculpt Mode - Painting the Texture*
- associated with *Modifiers*, see: - *Particles textures - Ocean textures*

**Material Textures** The material settings that we've seen so far produce smooth, *uniform* objects, but such objects aren't particularly true to reality, where uniformity tends to be uncommon and out of place. In order to deal with this unrealistic uniformity, Blender allows the user to apply *textures* which can modify the reflectivity, specularity, roughness and other surface qualities of a material.

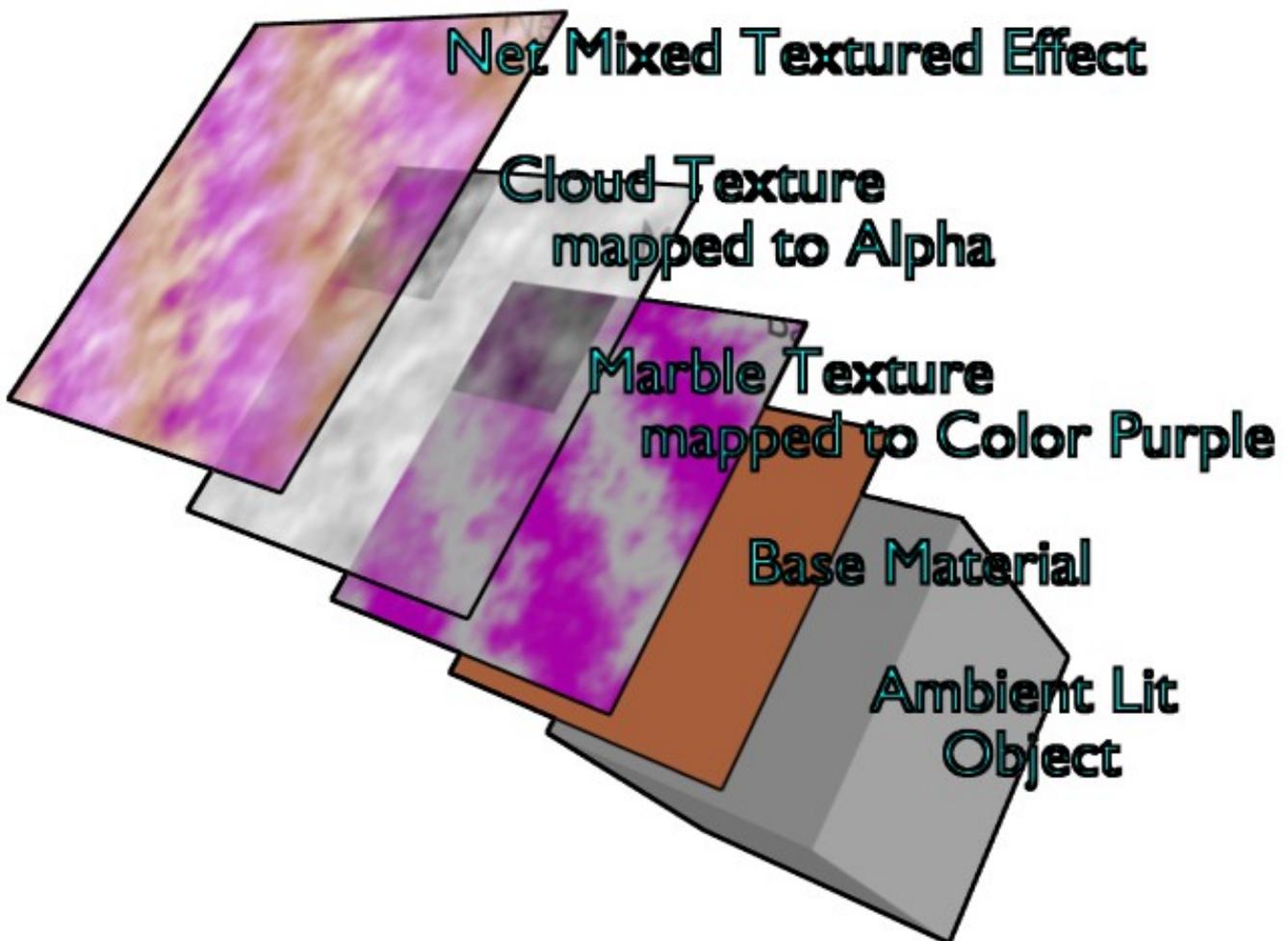


Fig. 2.1848: Textures Layer on base Material

Textures are like additional layers on top of the base material. Textures affect one or more aspects of the object's net coloring. The net color you see is a sort of layering of effects, as shown in this example image. The layers, if you will, are:

- Your object, lit with **ambient** light based on your world settings.

- Your base **material**, which colors the whole surface in a uniform color that reacts to light, giving different shades of the diffuse, specular, and mirror colors based on the way light passes through and into the surface of the object.
- A **primary texture** layer that overlays a purple marble coloring.
- A **second cloud texture** that makes the surface transparent in a misty/foggy sort of way by affecting the Alpha value.
- These two textures are **mixed** with the base material to provide the net effect: a cube of purplish-brown fog.

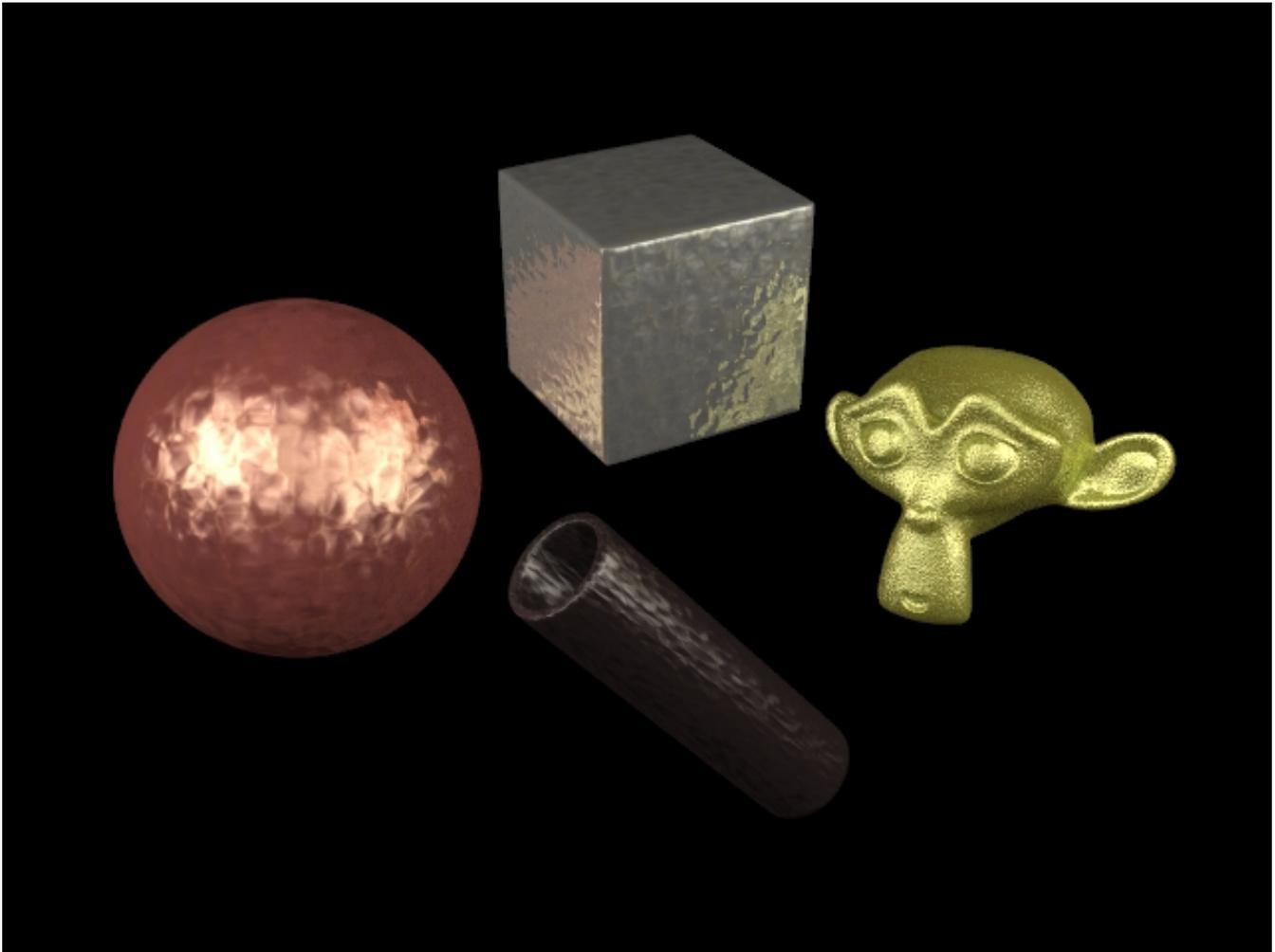


Fig. 2.1849: Some Metal Textures

This notion of using *more than one* texture, to achieve a combined effect, is one of the “hidden secrets” of creating realistic-looking objects. If you carefully “look at the light” while examining any real-life object, you will observe that the final appearance of that object is best described as the combination, in different ways and in different amounts, of several distinct underlying visual characteristics. These characteristics might be more (or less) strongly apparent at different angles, under different lighting conditions, and so forth. Blender allows you to achieve this in many ways. You can use “a stack of texture layers” as described in [this section](#), or you can also use arbitrarily-complex networks (“noodles”...) of “texture nodes” as discussed [here](#); the choice is yours.

Materials Textures fall into three primary categories:

**Procedural Textures** Textures generated by a mathematical formula. For example, *Wood*, *Clouds*, and *Distorted Noise*

**Images or Movies** Photos and films projected onto objects. For example, a flat map of Earth mapped to a sphere.

**Environment Maps** Textures used to create the impression of reflections and refractions. For example, an image of a street reflected in a car window.

**Data or Modifiers Textures** Textures obtained from raw data or obtained by a certain modifier in the scene. For example:

- volumetric materials use Voxel Data textures, or Point Density textures
- textures can be obtained from an Ocean Modifier

CRL 02:25, 26 May 2014 (UTC)(Sign)

---

## World Textures

### Reference

Mode: All Modes

Panel: Shading/World Context -> Preview

Hotkey:

---

**Description** The world buttons let you set up the shading of your scene in general. It can provide ambient color, and special effects such as mist, but a very common use of a *World* is to shade a background color.

**HoR, HoG, HoB** The RGB color at the horizon

**ZeR, ZeG, ZeB** The RGB color at the zenith (overhead)

These colors are interpreted differently, according to the Buttons in the *Preview Panel (Background colors)*:

**None Enabled** If none of these three buttons is checked, your background will just be plain flat color (using the horizon one).

**Blend** The background color is blended from horizon to zenith. If only this button is pressed, the gradient runs from the bottom to the top of the rendered image regardless of the camera orientation.

**Real** If this option is added, the gradient produced has two transitions, from nadir (same color as zenith) to horizon to zenith; the blending is also dependent on the camera orientation, which makes it more realistic. The horizon color is exactly at the horizon (on the x-y plane), and the zenith color is used for points vertically above and below the camera.

**Paper** If this option is added, the gradient keeps its characteristics, but it is clipped in the image (it stays on a horizontal plane (parallel to x-y plane): what ever the angle of the camera may be, the horizon is always at the middle of the image).

CRL 02:31, 26 May 2014 (UTC)(Sign)

**Brush Textures** Image textures can be loaded into blender. These images can then be applied to a mesh model that has been unwrapped and assigned an image of user defined size.

- Brush textures can be used to [paint](#) textures.
- Brush textures can be used to [paint](#) vertices.
- Brush textures can also be used in [sculpting](#) to create topology.

### Assigning a Texture

This page just shows how to add a texture to a slot. The textures' commons options are explained [here](#).

**Choosing the Texture context** In the Properties editor, choose the Texture context: this will show the Texture panel.

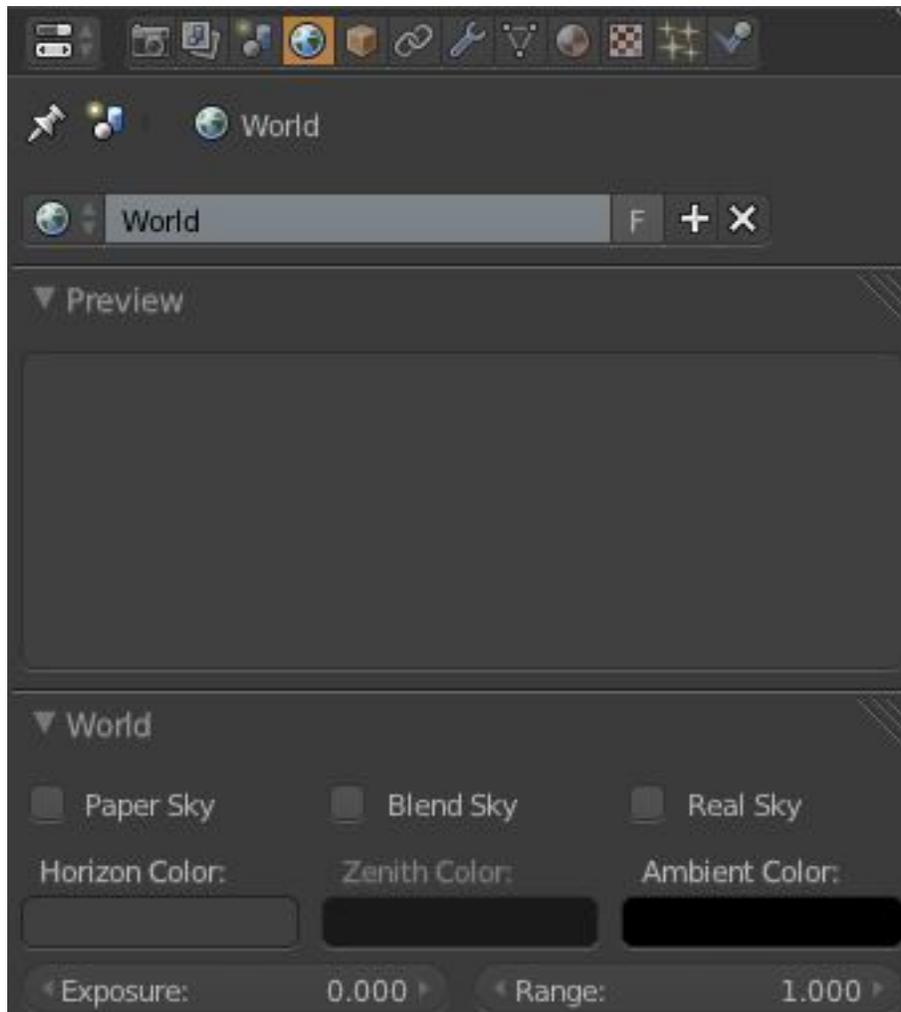


Fig. 2.1850: Textures Layer on base Material

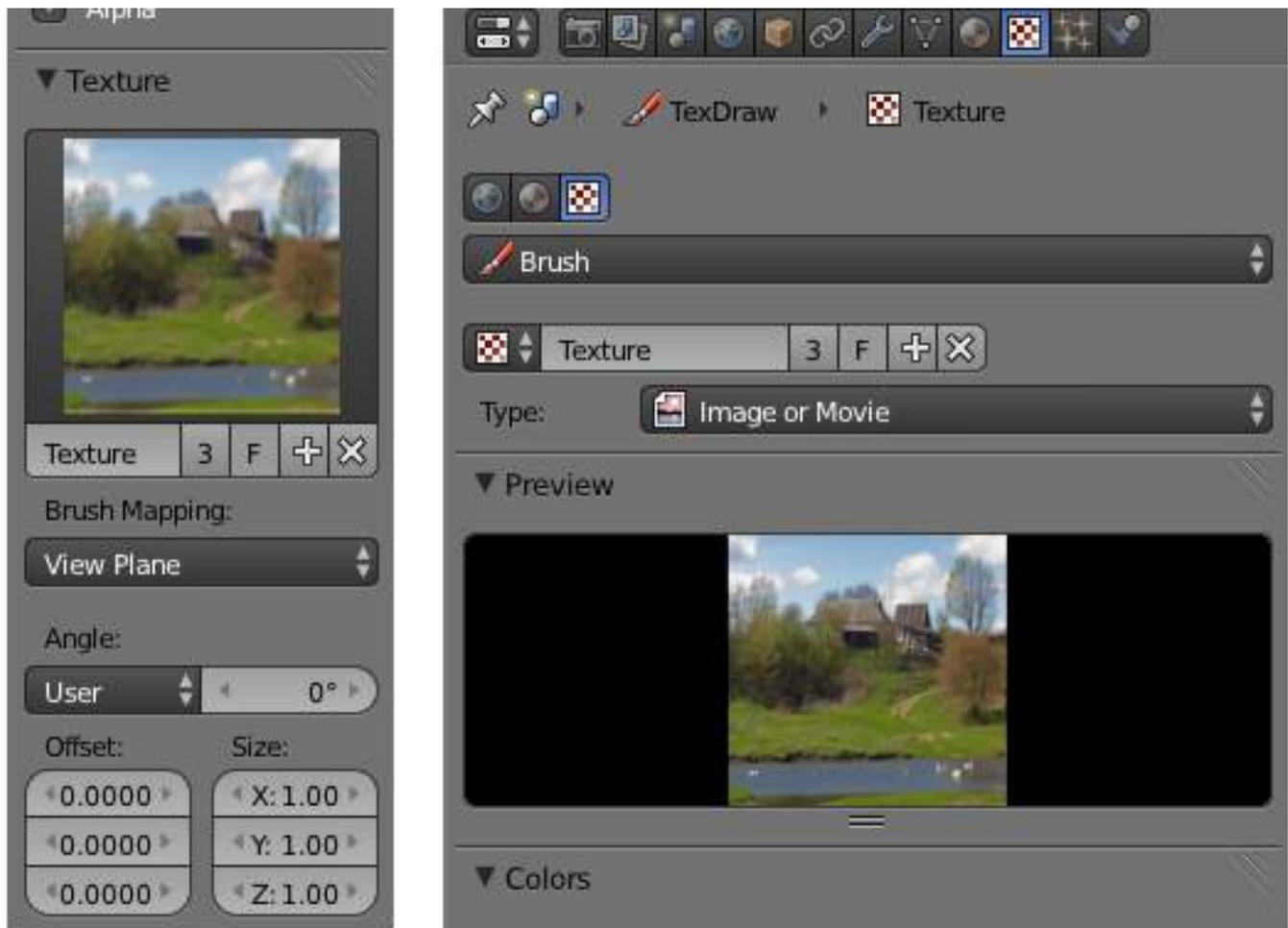


Fig. 2.1851: Applied Brush texture in different painting modes

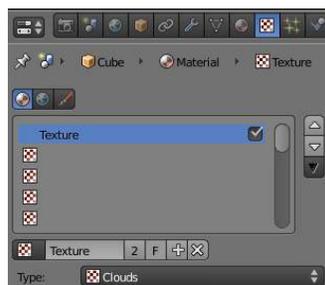


Fig. 2.1852: Texture panel

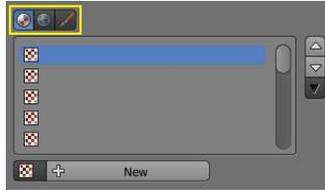


Fig. 2.1853: Texture panel with buttons for Material, World, and Brush textures highlighted

**Choosing the Texture data type** The three buttons *Material*, *World*, *Brush* at the top of the texture panel indicate the texture data type, that is, the kind of texture that is being edited.

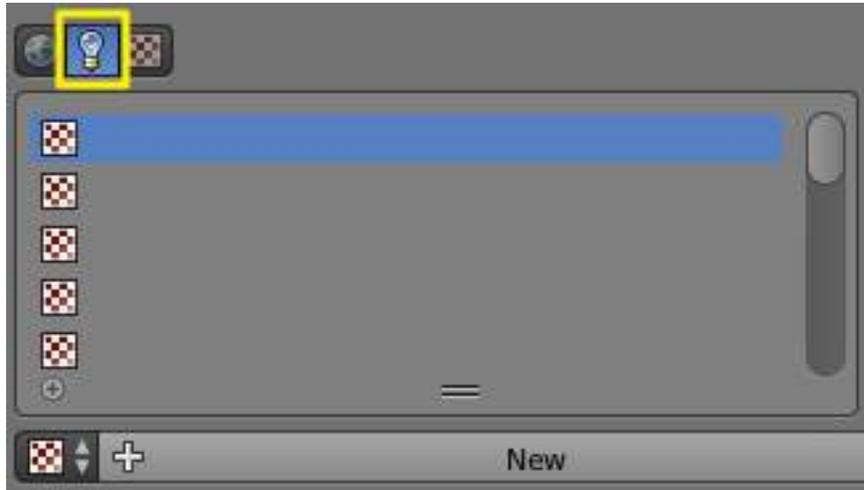


Fig. 2.1854: Texture panel with button for Lamp textures highlighted

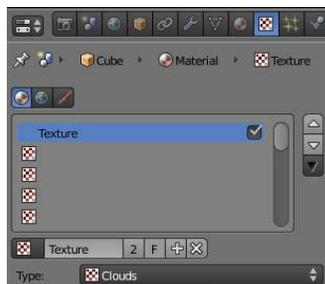


Fig. 2.1855: Texture panel

**Textures Slots** The list below these buttons represent the *Stack* of textures that we can manage. It can have up to eighteen *Texture Slots*:

- Tick or untick a texture to enable/disable it.
- Use the three buttons on the right side to move individual textures up and down in the stack or to copy/paste material's settings between slots.

**Creating a new Texture Datablock in a new Texture Slot** Select an empty slot, then click on the *New* button.

This will do two things:

- it will create a new texture datablock
- also, it will add a new slot in the textures stack

**Creating a new Texture Datablock in a non-empty slot** Select a non-empty slot, then click on the *Plus* button.

This will do two things:

- it will create a new texture datablock, with a new name, **making a copy of the texture datablock assigned to the selected slot**
- it will assign this new datablock to the selected slot

**Sharing a Texture Datablock in a non-empty slot**

- Select a non-empty slot, then click on the *Browse* button. This will open a menu showing all the available Texture Datablocks in this file.
- Choose a texture datablock in the menu to assign it to the selected slot. This will share the chosen texture with more than one object, hence the *Number of users* shown in the texture datablock will increase by one.

### Textures common options

In the Properties editor, choose the Texture context: this will show the Texture panel.



Fig. 2.1856: Textures Stack

**Textures Stack** The list below these buttons represents the *Stack* of textures that we can manage. It can have up to eighteen *Texture Slots*:

- Tick or untick a texture to enable/disable it.
- Use the three buttons on the right side to move individual textures up and down in the stack or to copy/paste material's settings between slots.

The order in the Stack Textures defines how textures overlay each other for finally result image.

**Texture Datablock** Select a slot in the Textures Stack to see its settings.

The first group of buttons below the stack displays the texture currently selected in the stack.

**Browse** The first button below the stack displays the all available textures in the current file. Textures are stored globally, and can be linked to more than one material. If you have already created a texture that you want to reuse, select from this list.

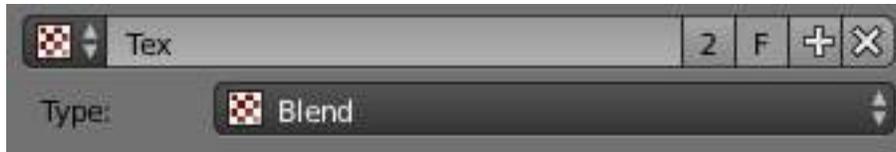


Fig. 2.1857: Active Texture Datablock

**Name** A name field where the name of the material can be changed.

**Number of users** If the active texture is used by another material, a 2 button appears that can be used to make a single-user copy of the active texture. Use this button to quickly create a new texture based on an existing texture.

**Fake** The *F* button assigns the active texture to a “Fake” material, so that the texture is saved with the file even if it has no “real” users.

**Add** Replaces the texture of the active slot with a new texture.

**Unlink** Removes the texture from the active slot.

**Texture Type** Choose the type of texture that is used for the current texture datablock.

- [Procedural Textures](#)
- [Image and Video Textures](#)
- [Environment Map](#)
- [Volume Textures](#)
- [Ocean Texture](#)

These types are described in detail [in this section](#).

**Preview** The texture preview panel provides a quick pre-visualisation of how the texture looks on its own, without mapping.

**Texture, Material, or Both** Choose to display only the texture, only the material, or both.

**Show Alpha** Show alpha in preview. If Alpha: Use is checked in the [Image Sampling](#) panel, the image’s alpha channel is displayed. If Alpha: Use is unchecked, an alpha channel based on averaged rgb values is displayed like it would be used by the Alpha slider in the [Influence](#) panel.

**Colors** The *Ramp* button activates a color ramp which allows you to remap the colors of a texture to new ones. See [Ramps](#) for information on using ramps.

The color of a texture can be modified with the *Brightness*, *Contrast*, and *Saturation* buttons. All textures with RGB-Values - including *Images* and *Environment Maps* - may be modified with the RGB sliders.

**R, G, B** Tint the color of a texture by brightening each red, green and blue channel.

**Brightness** Change the overall brightness/intensity of the texture

**Contrast** Change the contrast of the texture

**Saturation** Change the saturation of the texture

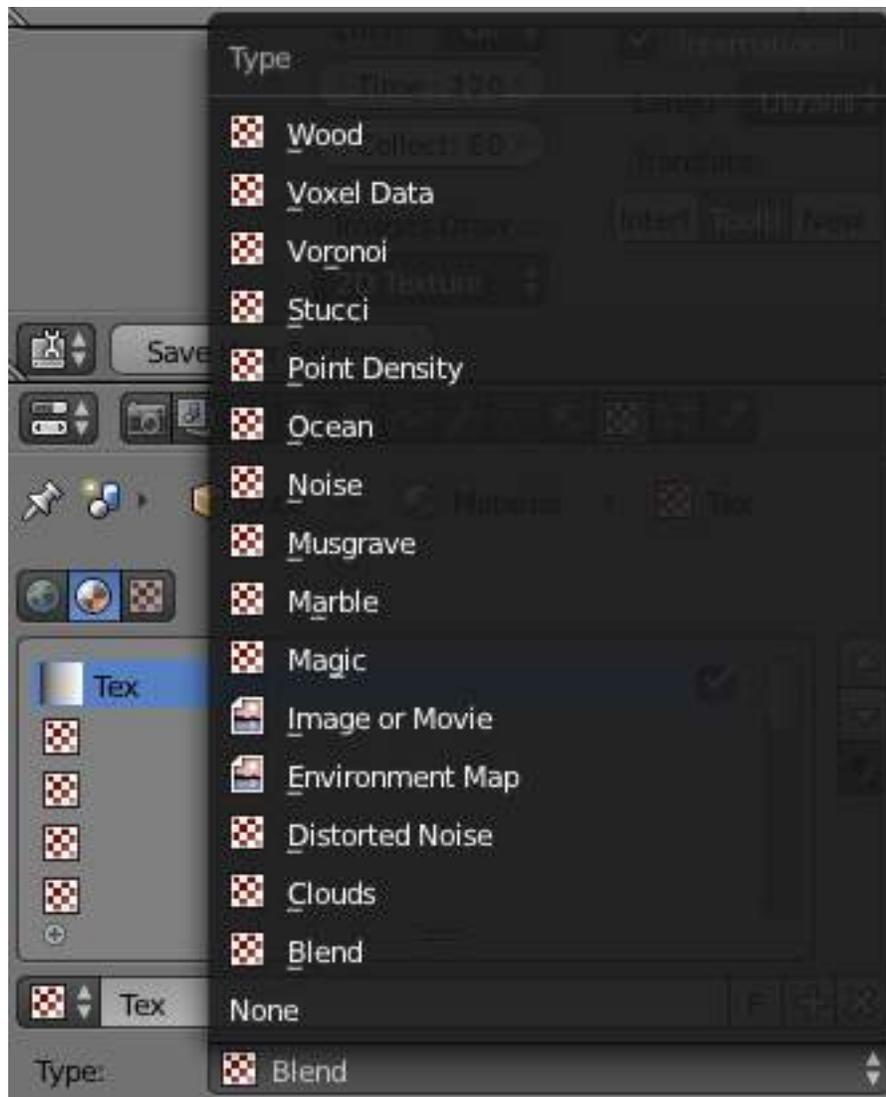


Fig. 2.1858: Texture Types

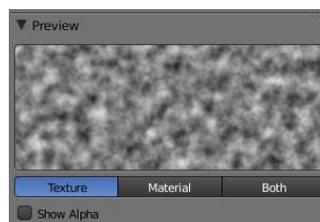


Fig. 2.1859: Preview panel

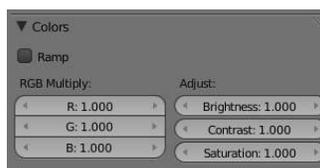


Fig. 2.1860: Colors panel

**Mapping** Here you can control how the texture will be mapped on the object.

---

**Note:** Brushes

These options are not available for brushes because they wouldn't make sense

---

See [Mapping](#) section for details.

**Influence** Here you can control what properties the texture will affect, and by how much.

They are detailed on the [Influence](#) section.

---

**Note:** Brushes

These options are not available for brushes because they wouldn't make sense

---

## UV Image Editor

FIXME(Template Unsupported: Doc:2.6/Reference/Textures/UV\_Image\_Editor; {{ Doc:2.6/Reference/Textures/UV\_Image\_Editor }}  
)

## Texture types

This are the available texture types:

- [Procedural Textures](#)
- [Image Textures](#)
- [Video Textures](#)
- [Combined Textures](#)
- [Volume Textures](#)
- [Ocean Textures](#)

## Procedural Textures

Procedural textures are textures that are defined mathematically. They are generally relatively simple to use, because they don't need to be mapped in a special way - which doesn't mean that procedural textures can't become very complex.

These types of textures are 'real' 3D. By that we mean that they fit together perfectly at the edges and continue to look like what they are meant to look like even when they are cut; as if a block of wood had really been cut in two. Procedural textures are not filtered or anti-aliased. This is hardly ever a problem: the user can easily keep the specified frequencies within acceptable limits.

These are the available types:

- [Blend](#)
- [Clouds](#)
- [Distorted Noise](#)
- [Magic](#)
- [Marble](#)

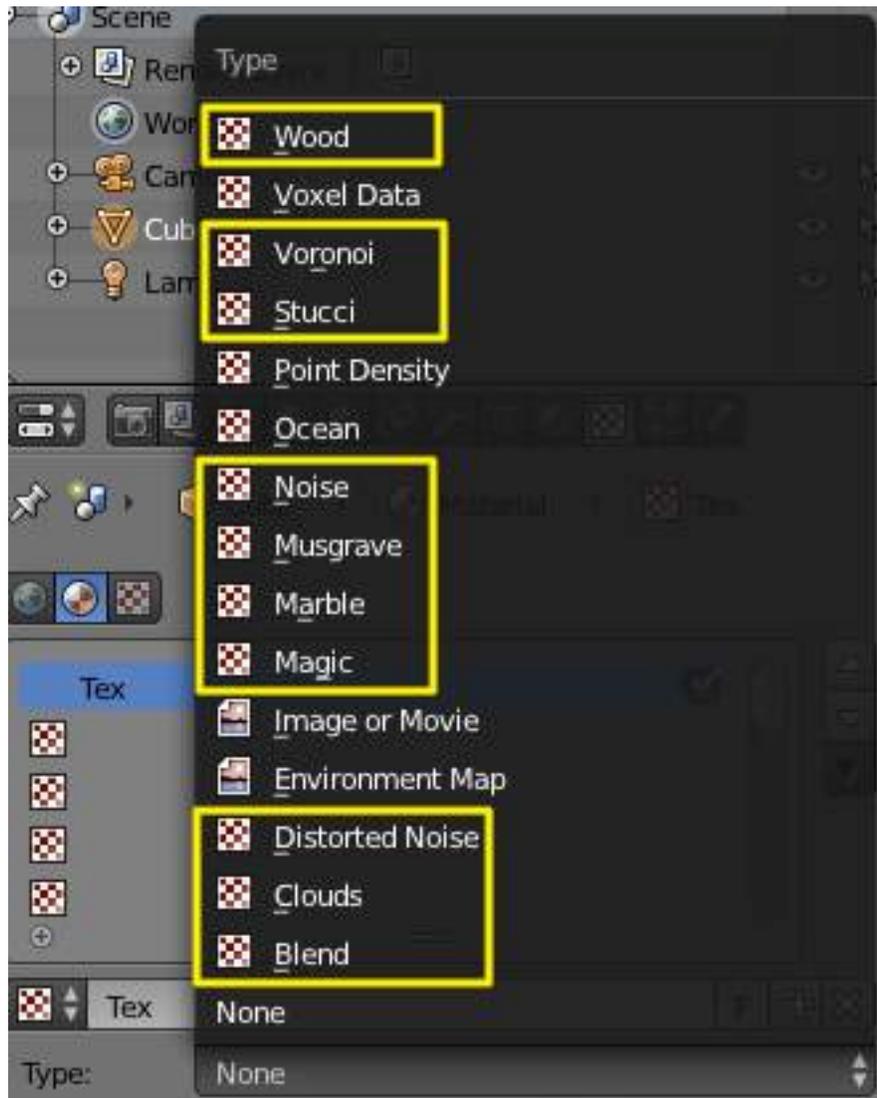


Fig. 2.1861: Textures generated by a mathematical formula.

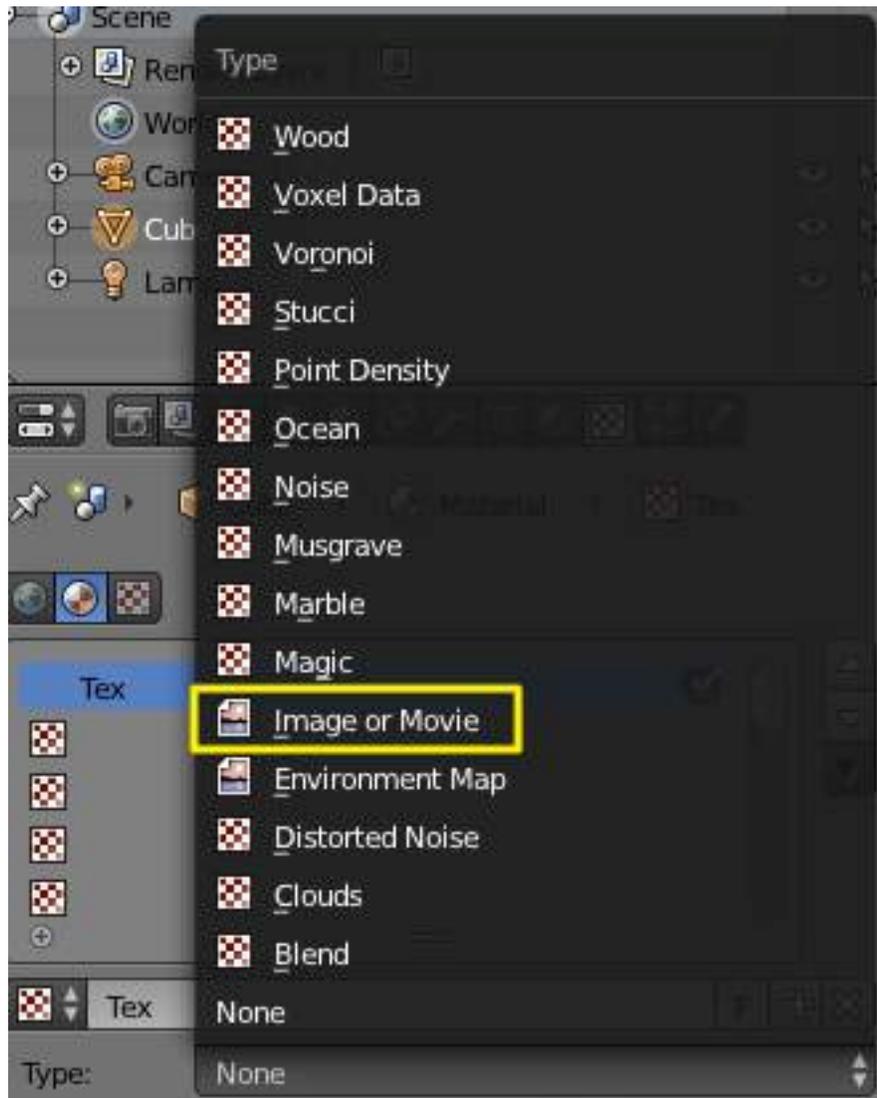


Fig. 2.1862: Photos and films projected onto objects.

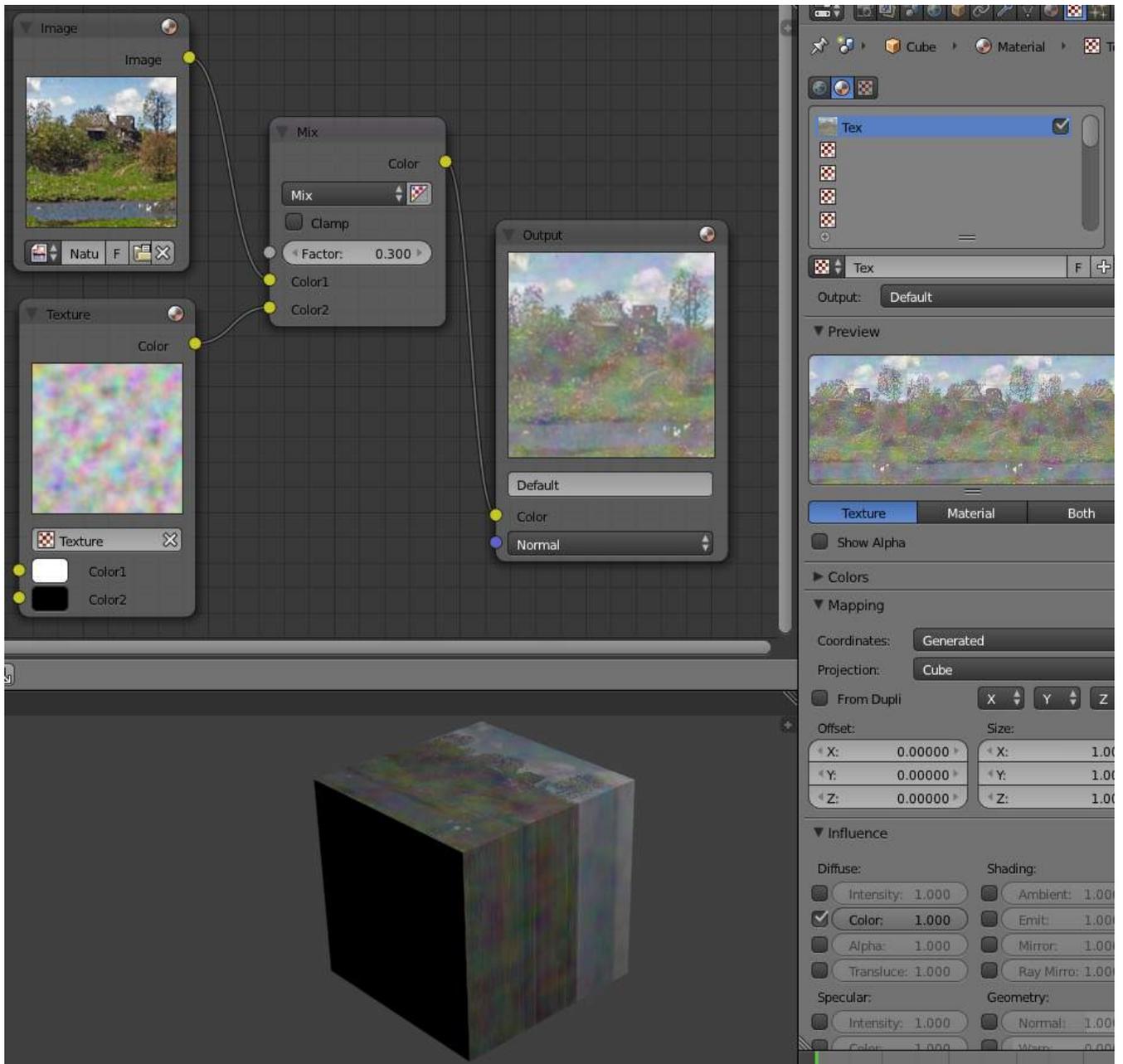


Fig. 2.1863: Combined textures based on nodes.

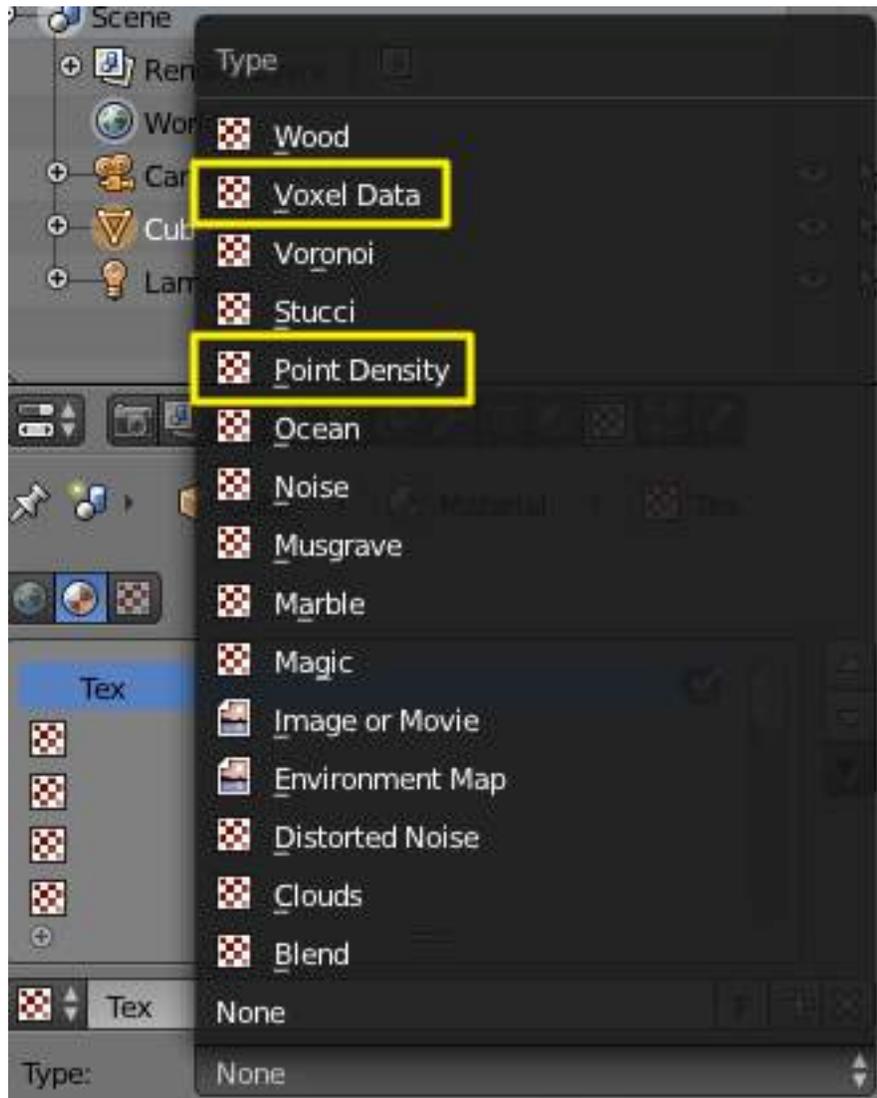


Fig. 2.1864: Textures that can be applied to volumetric data.

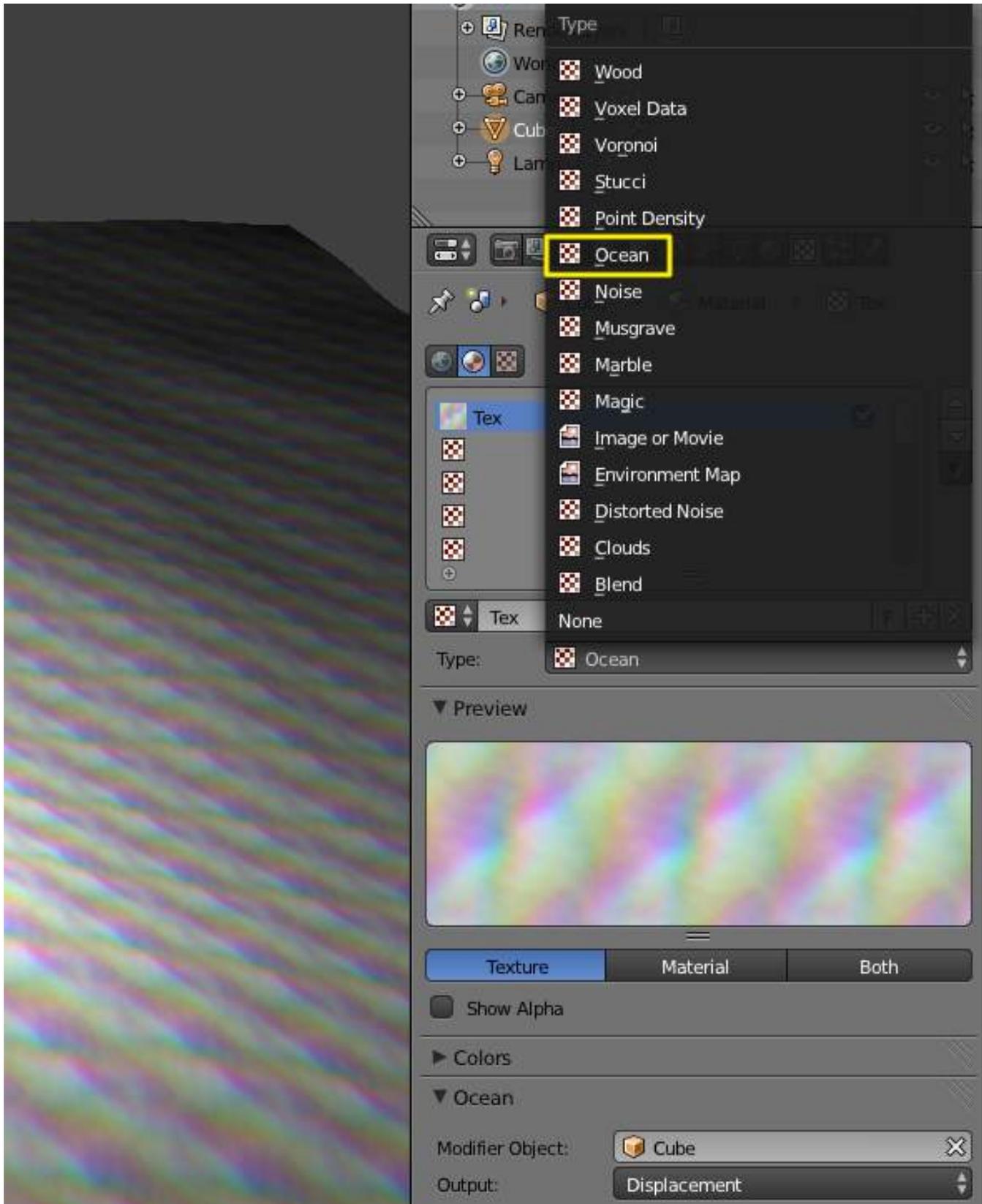


Fig. 2.1865: Texture generated by an Ocean modifier.

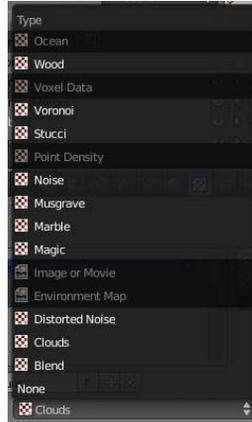


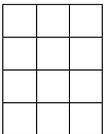
Fig. 2.1866: The Texture Type list in the Texture panel of the Texture Buttons. (Non procedural textures darkened out.)

- Musgrave
- Noise
- Stucci
- Voronoi
- Wood

### Common options

**Noise Basis** Each noise-based Blender texture (with the exception of Voronoi and simple noise) has a *Noise Basis* setting that allows the user to select which algorithm is used to generate the texture. This list includes the original Blender noise algorithm. The *Noise Basis* settings makes the procedural textures extremely flexible (especially *Musgrave*).

The *Noise Basis* governs the structural appearance of the texture :



There are two more possible settings for *Noise Basis*, which are relatively similar to *Blender Original*: Improved Perlin and Original Perlin

**Nabla** Almost all procedural textures in Blender use derivatives for calculating normals for texture mapping (with as exception *Blend* and *Magic*). This is important for Normal and Displacement Maps. The strength of the effect is controlled with the *Nabla* Number Button.

**Hints** Use the size buttons in the *Mapping* panel to set the size that the procedural textures are mapped to.

Procedural textures can either produce colored textures, intensity only textures, textures with alpha values and normal textures. If intensity only ones are used the result is a black and white texture, which can be greatly enhanced by the use of ramps. If on the other hand you use ramps and need an intensity value, you have to switch on *No RGB* in the *Mapping* panel.

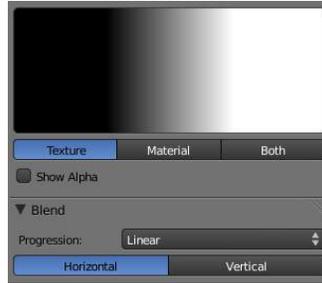


Fig. 2.1887: Blend Texture Panels

### Procedural textures: Blend

**Often used for** This is one of the most frequently used procedural textures. You can use blend textures to blend other textures together (with *Stencil*), or to create nice effects (especially with the *Mapping: Normal* trick). Just remember: if you use a ramp to create a custom blending, you may have to use *No RGB*, if the *Mapping* value needs an intensity input.

**Result(s)** Intensity. The Blend texture generates a smoothly interpolated progression.

#### Options

**Progression** Profile of blend

**Linear** A linear progression

**Quadratic** A quadratic progression

**Easing** A flowing, non-linear progression

**Diagonal** A diagonal progression

**Spherical** A progression with the shape of a three-dimensional ball

**Quadratic Sphere** A quadratic progression with the shape of a three-dimensional ball

**Radial** A radial progression **Horizontal / Vertical** The direction of the progression is flipped a quarter turn.

### Procedural textures: Clouds

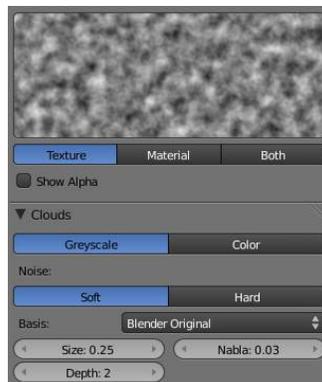


Fig. 2.1888: Clouds Texture Panels

Clouds represent Perlin noise. In addition, each noise-based Blender texture (with the exception of Voronoi and simple noise) has a “Noise Basis” setting that allows the user to select which algorithm is used to generate the texture.

**Often used for** Clouds, Fire, Smoke. Well-suited to be used as a Bump map, giving an overall irregularity to the material.

**Result(s)** *Greyscale* (default) or *RGB Color*

### Options

**Greyscale** The standard noise, gives an intensity

**Color** The noise gives an RGB value

**Noise** *Soft* or *Hard*, changes contrast and sharpness

**Size** The dimension of the Noise table

**Depth** The depth of the *Clouds* calculation. A higher number results in a long calculation time, but also in finer details.

**Technical Details** A three-dimensional table with pseudo-random values is used, from which a fluent interpolation value can be calculated with each 3D coordinate (thanks to Ken Perlin for his masterful article “An Image Synthesizer”, from the SIGGRAPH proceedings 1985). This calculation method is also called Perlin Noise.

### Procedural textures: Distorted Noise

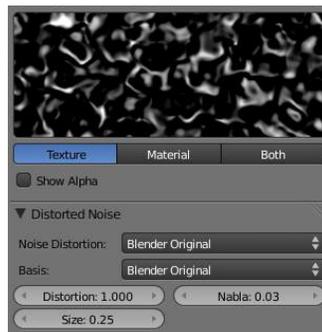


Fig. 2.1889: Distorted Noise Texture Panels

*Distortion Noise* takes the option that you pick from *Noise Basis* and filters it, to create hybrid pattern.

**Often used for** Grunge, very complex and versatile

**Result(s)** Intensity

### Options

**Noise Distortion** The texture to use to distort another

**Basis** The texture to be distorted

**Noise** The size of the noise generated

**Distortion** The amount that *Distortion Noise* affects *Basis*

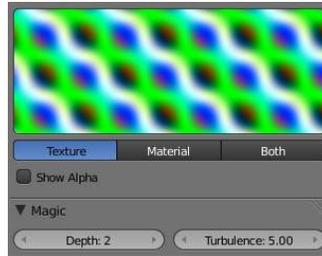


Fig. 2.1890: Magic Texture Panels

### Procedural textures: Magic

**Often used for** Not frequently used. It can be used for “Thin Film Interference”, if you set *Mapping* to *Reflection* and use a relatively high *Turbulence*.

**Result(s)** RGB color. The RGB components are generated independently with a sine formula.

#### Options

**Depth** The depth of the calculation. A higher number results in a long calculation time, but also in finer details.

**Turbulence** The strength of the pattern.

### Procedural textures: Marble

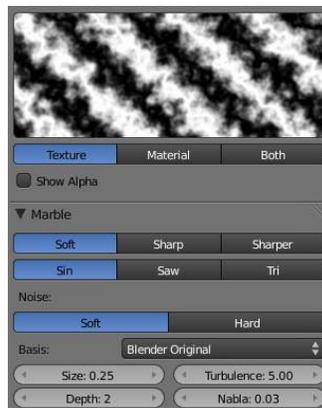


Fig. 2.1891: Marble Texture Panels

**Often used for** Marble, Fire, Noise with a structure

**Result(s)** Intensity value only

Bands are generated based on the sine, saw, or triangular formulae and noise turbulence.

#### Options

**Soft / Sharp / Sharper** Three presets for soft to more clearly defined *Marble*

**Sin / Saw / Tri** Shape of wave to produce bands

**Soft / Hard** The noise function works with two methods.

**Size** The dimensions of the noise table

**Depth** The depth of the *Marble* calculation. A higher value results in greater calculation time, but also in finer details.

**Turbulence** The turbulence of the sine bands.

### Procedural textures: Musgrave

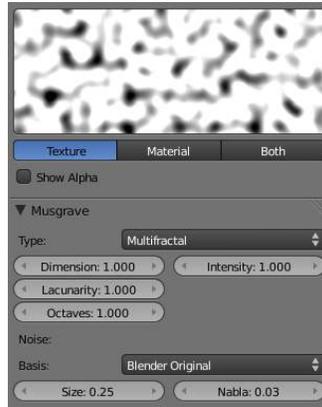


Fig. 2.1892: Musgrave Texture Panels

**Often used for** Organic materials, but it's very flexible. You can do nearly everything with it.

**Result(s)** Intensity

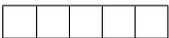
### Options

**Type** This procedural texture has five noise types on which the resulting pattern can be based and they are selectable from a dropdown menu at the top of the tab. The five types are:

- Hetero Terrain
- fBm
- Hybrid Multifractal
- Ridged Multifractal
- Multifractal

These noise types determine the manner in which Blender layers successive copies of the same pattern on top of each other at varying contrasts and scales.

Examples with Basis : Voronoi F1 - Dimension : 0.5 - Lacunarity : 0.15 - Octave: 2.0



The main noise types have four characteristics:

**Dimension** Fractal dimension controls the contrast of a layer relative to the previous layer in the texture. The higher the fractal dimension, the higher the contrast between each layer, and thus the more detail shows in the texture. Range: 0 to 2.

**Lacunarity** Lacunarity controls the scaling of each layer of the Musgrave texture, meaning that each additional layer will have a scale that is the inverse of the value which shows on the button. i.e. Lacunarity = 2 → Scale = 1/2 original. Range: 0 to 6.

**Octaves** Octave controls the number of times the original noise pattern is overlaid on itself and scaled/contrasted with the fractal dimension and lacunarity settings. Range: 0 to 8.

**Intensity** Light intensity. Called *Offset* for *Hetero Terrain*. Range: 0 to 10.

The *Hybrid Multifractal* and *Ridged Multifractal* types have these additional settings:

**Offset** Both have a “Fractal Offset” button that serves as a “sea level” adjustment and indicates the base height of the resulting bump map. Bump values below this threshold will be returned as zero. Range: 0 to 6.

**Gain** Setting which determines the range of values created by the function. The higher the number, the greater the range. This is a fast way to bring out additional details in a texture where extremes are normally clipped off. Range: 0 to 6.

### Procedural textures: Noise

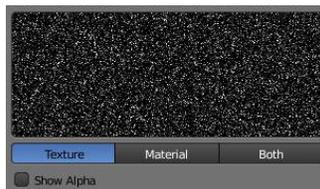


Fig. 2.1903: Noise Texture Panel

Although this looks great, it is not Perlin Noise! This is a true, randomly generated Noise. This gives a different result every time, for every frame, for every pixel.

There are no options for this noise

**Often used for** White noise in an animation. This is not well suited if you don't want an animation. For material displacement or bump, use clouds instead.

**Result(s)** Intensity

### Procedural textures: Stucci

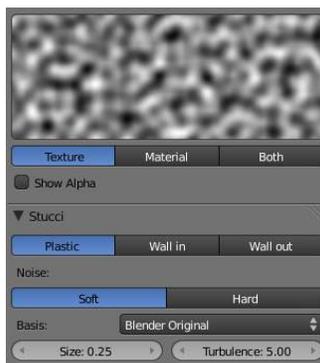


Fig. 2.1904: Stucci Texture Panels

The *Stucci* texture is based on noise functions.

**Often used for** Stone, Asphalt, Oranges. Normally for Bump-Mapping to create grainy surfaces.

**Result(s)** Normals and Intensity

## Options

**Plastic / Wall In / Wall out** Plastic is the standard Stucci, whilst the “walls” is where Stucci gets its name. This is a typical wall structure with holes or bumps.

**Soft / Hard** There are two methods available for working with Noise

**Size** Dimension of the Noise table

**Turbulence** Depth of the *Stucci* calculations

## Procedural textures: Voronoi

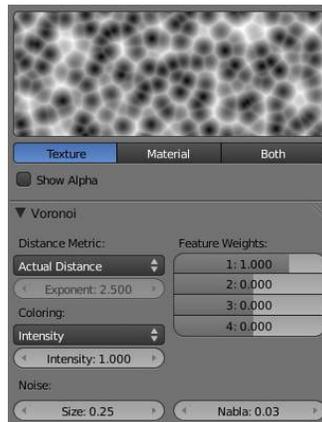


Fig. 2.1905: Voronoi Texture Panels

**Often used for** Very convincing Metal, especially the “Hammered” effect. Organic shaders (e.g. scales, veins in skin).

**Result(s)** Intensity (default) and Color

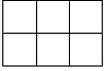
## Options

**Distance Metric** This procedural texture has seven Distance Metric options. These determine the algorithm to find the distance between cells of the texture. These options are:

- Minkovsky
- Minkovsky 4
- Minkovsky 1/2
- Chebychev
- Manhattan
- Distance Squared
- Actual Distance

The *Minkovsky* setting has a user definable value (the *Exponent* button) which determines the Minkovsky exponent ( $\epsilon$ ) of the distance function  $(x^\epsilon + y^\epsilon + z^\epsilon)^{1/\epsilon}$ . A value of one produces the *Manhattan* distance metric, a value less than one produces stars (at **0.5**, it gives a *Minkovsky 1/2*), and higher values produce square cells (at **4.0**, it gives a *Minkovsky 4*, at **10.0**, a *Chebychev*). So nearly all Distance Settings are basically the same - variations of *Minkovsky*.

You can get irregularly-shaped rounded cells with the *Actual Distance / Distance Squared* options.



**Feature Weights** These four sliders at the bottom of the Voronoi panel represent the values of the four Worley constants, which are used to calculate the distances between each cell in the texture based on the distance metric. Adjusting these values can have some interesting effects on the end result...

**Coloring** Four settings (*Intensity, Position, Position and Outline*, and *Position, Outline, and Intensity*) that can use four different noise basis as methods to calculate color and intensity of the texture output. This gives the Voronoi texture you create with the “Worley Sliders” a completely different appearance and is the equivalent of the noise basis setting found on the other textures.

**Technical Details** For a more in depth description of the Worley algorithm, see: [Worley Documentation](#).

### Procedural textures: Wood



Fig. 2.1918: Wood Texture Panels

**Often used for** Woods and ring-shaped patterns.

**Result(s)** Intensity only

#### Options

**Sin / Saw / Tri** Shape of wave to produce bands

**Bands / Rings / Band Noise / Ring Noise** Set the bands to either straight or ring-shaped, with or without turbulence

**Soft / Hard** There are two methods available for the Noise function

**Size** Dimension of the Noise table

**Turbulence** Turbulence of the Band Noise and Ring Noise types

#### Technical Details

**Generation** Bands are generated based on a sine formula. You can also add a degree of turbulence with the Noise formula.

**Coordinates** As the band is based on a sine formula, the texture repeats itself every pi units rather than every 1.0 units. To correct this, scale the texture by a value of pi for the dimension you wish.

## Image Textures

The term *Image Texture* simply means that a graphic image - a pixel grid composed of R, G, B, and sometimes Alpha values - is used as the input source to the texture. As with other types of textures, this information can be used in a number of ways, not only as a simple “decal”.

When the Texture Type *Image* or *Movie* is selected, three new panels present themselves allowing us to control most aspects of how image textures are applied: *Image*, *Image Sampling*, and *Image Mapping*.

**About Image Based Texturing** Texture images take up precious memory space, often being loaded into a special video memory bank that is very fast and very expensive, so it is often very small. So, keep the images as small as possible. A 64x64 image takes up only one fourth the memory of a 128x128 image.

For photo-realistic rendering of objects in animations, often larger image textures are used, because the object might be zoomed in on in camera moves. In general, you want to use a texture sized proportionally to the number of pixels that it will occupy in the final render. Ultimately, you only have a certain amount of physical RAM to hold an image texture and the model and to provide work space when rendering your image.

For the most efficient memory usage, image textures should be square, with dimensions as powers of 2, such as 32x32, 64x64, 128x128, 256x256, 1024x1024, 2048x2048, and 4096x4096.

If you can re-use images across different meshes, this greatly reduces memory requirements. You can re-use images if you map those areas of the meshes that “look alike” to a layout that uses the common image. In the overview below, the left image is re-used for both the sphere and a portion of the monkey. The monkey uses two layouts, one which has one UV map of a few faces, and another that has three maps.

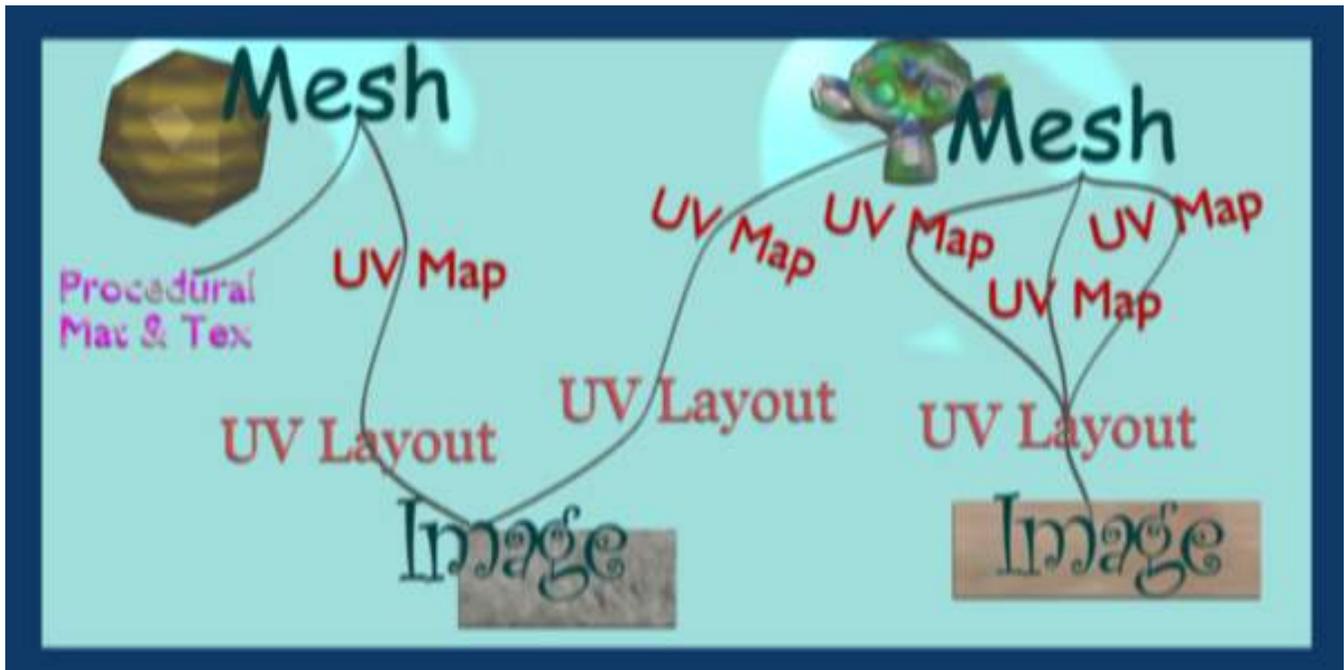


Fig. 2.1919: How all the parts of UV Texturing work together

When using file textures, it is very important that you have [Mapped the UVs](#) of the mesh, and they are laid out appropriately.

You don’t have to UV map the *entire* mesh. The sphere above on the left has some faces mapped, but other faces use procedural materials and textures. Only use UV Textures for those portions of your mesh where you want very graphic, precise detail. For

example, a model of a vase only needs UV Texture for the rim where decorative artwork is incorporated. A throw pillow does not need a different image for the back as the front; in fact many throw pillows have a fabric (procedural material) back.

As another example, you should UV map both eyes of a head to the same image (unless you want one bloodshot and the other clear). Mapping both sides of a face to the same image might not be advisable, because the location of freckles and skin defects are not symmetrical. You could of course change the UV map for one side of the face to slightly offset, but it might be noticeable. Ears are another example where images or section of an images can be mapped to similar faces.

**Workflow** The process consists of the following steps.

- Create the Mesh. [Unwrap](#) it into one or more [UV Layouts](#).
- Create one or more Materials for the Mesh.
- Create one or more images for each UV Layout and aspect of the texture. Either - paint directly on the mesh using Texture Paint in the 3D window, - load and/or edit an image in the UV Editor window, or - Bake the existing materials into an image for the UV Editor window.
- Apply those images as UV Textures to the mesh to affect one or more aspects of the mesh. This is done by using one or more of the numerous Map To options. For example, - map to Color to affect the diffuse coloring of the mesh, - map to Nor to affect the normal direction to give the surface a bumpy or creased look, or - map to Spec (specularity) to make certain areas look shiny and oily.
- Layer the Textures to create a convincing result.

**Using Images and Materials** To use an image as the color and alpha (transparency) of the texture, you can create an image in an external paint program and tell the UV/Image Editor to Open that file as the texture, or you can create a New image and save it as the texture.

If you want to start off by creating an image using an external paint program, you will want to save an outline of your UV faces by using the *Save UV Face Layout* tool located in the UVs menu. This is discussed [here](#).

**Creating an Image Texture** To create an image within Blender, you have to first create a [New Blank Image](#) with a uniform color or test grid. After that, you can color the image using the:

- Vertex colors as the basis for an image
- Render Bake image based on how the mesh looks in the scene

After you have created your image, you can modify it using Blender's built-in [Texture Paint](#) or any external image painting program.

---

**Note:** See [Texture in 3D View](#) but does not Render

You may be able to see the texture in Textured display mode in the 3D View; this is all that is required to have textures show up in Blender's Game Engine. Rendering, however, requires a material. You must have a *Face Textures* material assigned to the mesh for it to render using the UV Texture. In the Material settings, ADD NEW material to a selected object and enable *Face Textures*.

---

**Examples** There may be one UV Layout for the face of a character, and another for their clothes. Now, to texture the clothes, you need to create an image at least for the Color of the clothes, and possible a "bump" texture to give the fabric the appearance of some weave by creating a different image for the Normal of the clothes. Where the fabric is worn, for example at the elbows and knees, the sheen, or Specularity, of the fabric will vary and you will want a different image that tells Blender how to vary the Specularity. Where the fabric is folded over or creased, you want another image that maps Displacement to the mesh to physically deform the mesh. Each of these are examples of applying an image as a texture to the mesh.

As another example, the face is the subject of many questions and tutorials. In general, you will want to create a Material that has the basic skin color, appropriate shaders, and sub-surface scattering. Then you will want to layer on additional UV Textures for:

- Freckle map for Color and Normal aspects
- Subdermal veins and tendons for Displacement
- Creases and Wrinkles and skin cell stratification for Normal
- Makeup images for Color
- Oily maps for Specularity
- For a zombie, Alpha transparency where the flesh has rotted away (*ewwww....*)
- Under chin and inside nostrils that receive less Ambient light
- Thin skin is more translucent, so a map is needed for that

Each image is mapped by using another Texture Channel. Each of these maps are images which are applied to the different aspects (Color, Normal, Specularity) of the image. Tileable images can be repeated to give a smaller, denser pattern by using the Texture controls for repeat or size.

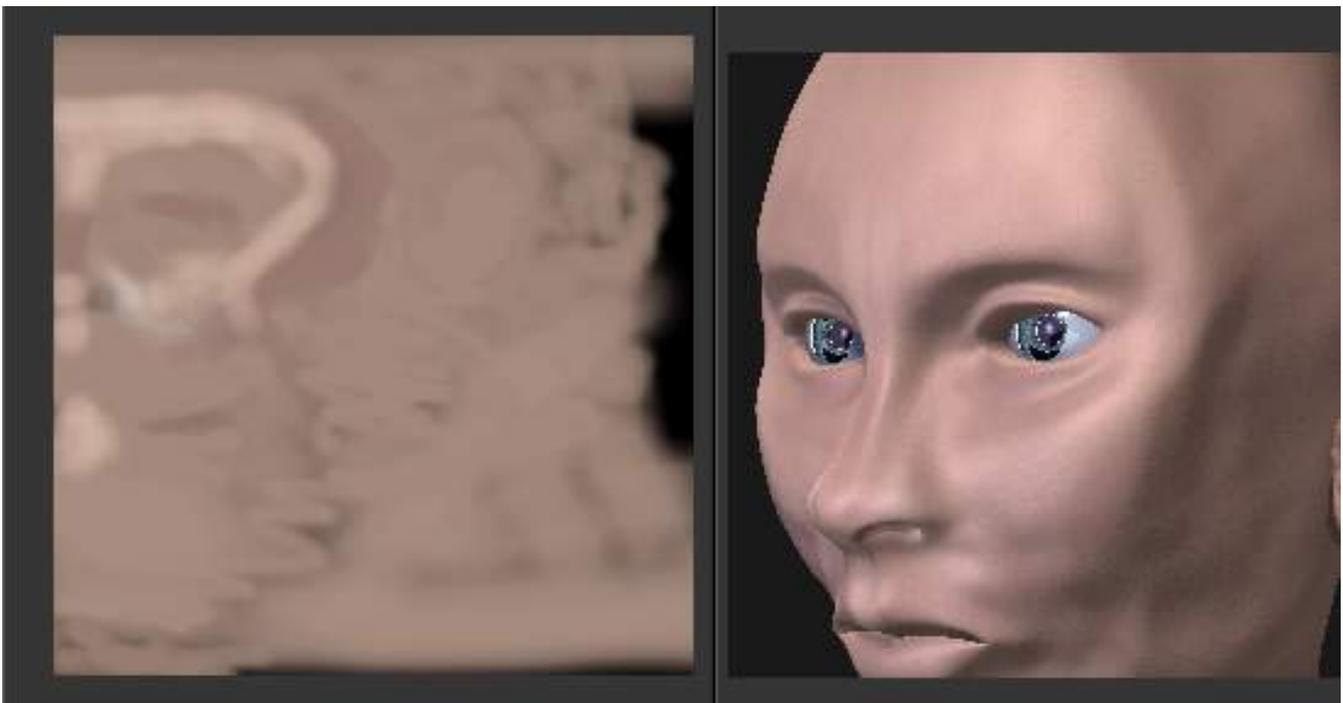


Fig. 2.1920: Base UV Texture

**Layering UV Textures** Great textures are formed by layering images on top of one another. You start with a base layer, which is the base paint. Each successive layer on top of that is somewhat transparent to let the bottom layers show through, but opaque where you want to add on to details.

To avoid massive confusion, all image textures for a mesh usually use the same UV map. If you do, each image will line up with the one below it, and they will layer on top of one another like the examples shown to the right. To do this, just create one UV Texture (map) as described in this section. Then, create material image textures as described in the procedural materials section. Instead of mapping to Original Coordinates (OrCo), map to UV.

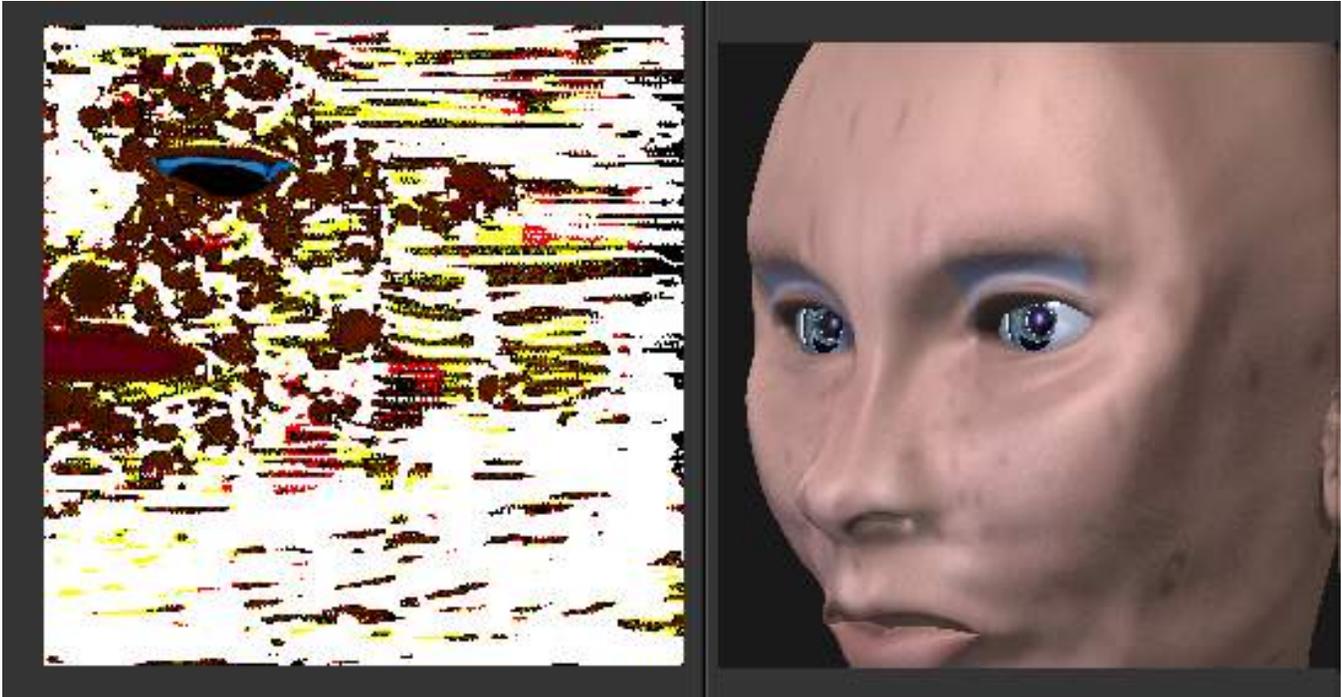


Fig. 2.1921: Layered UV Texture

Use that map name repeatedly in the Material->Textures->Map Input panel by selecting UV and typing the name in the text field. In the example to the right, our UV Texture is called “Head” (you may have to expand the image to see the panel settings). Then, the image texture shown will be mapped using the UV coordinates. In the “Base UV Texture” example to the right, the face has two textures UV mapped; one for a base color, and another for spots, blemishes and makeup.

Both textures use the same UV Texture map as their Map Input, and both affect Color. The Makeup texture is transparent except where there is color, so that the base color texture shows through. Note that the colors were too strong on the image, so they amount of Col affects is turned down to 60% in the second layer (the blemish layer).

Normally, we think of image textures affecting the color of a mesh. Realism and photo-realistic rendering is a combination of many different ways that light interacts with the surface of the mesh. The image texture can be Mapped To not only color, but also *Normal* (bumpiness) or *Reflection* or any of the other attributes specified in the Map To panel.

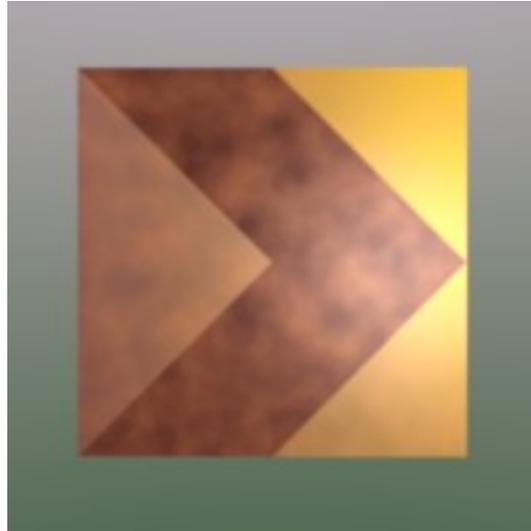
If you paint a grey-scale image (laid out according to the UV Layout) with white where the skin is oily and shiny, and dark where it is not, you would map that input image according to the UV Layout, but have it affect Specularity (not color).

To make portions of a mesh transparent and thus reveal another mesh surface underneath, you would paint a grey-scale image with black where you want the texture transparent, map input to UV, and map it to Alpha (not color). To make portions of a mesh, like a piece of hot metal, appear to glow, you would use a grey-scale image mapped to Emit.

Believe it or not, this is only “the tip of the iceberg!” If everything that’s been described here just isn’t enough for you, the *texture nodes* feature, introduced in recent versions of Blender, enables you to layer and combine textures in almost any way you can imagine.

**Mix and Match Materials** You can mix and match procedural materials and textures, vertex paint, and UV textures onto the same mesh.

The image to the right has a world with a red ambient light. The material has both VCol Paint and Face Textures enabled, and receives half of ambient light. A weak cloud texture affects color, mixing in a tan color. The right vertices are vertex painted yellow and the left is unpainted procedural gray. The UV Texture is a stock arrow image from the public domain texture CD.



Scene lighting is a white light off to the right. From this information and the User Manual thus far, you should now be able to recreate this image.

You can also assign [multiple materials](#) to the mesh based on which faces you want to be procedural and which you want to be texture-mapped. Just don't UV map the faces you want to be procedural.

You can use UV Textures and VertexPaint (V in the 3D View window) simultaneously, if both are enabled in the Material settings. The vertex colors are used to modulate the brightness or color of the UV image texture:

- UV Texture is at the base (*Face Textures*)
- Vertex paint affects its colors, then
- Procedural textures are laid on top of that,
- Area lights shine on the surface, casting shadows and what not, and finally
- Ambient light lights it up.

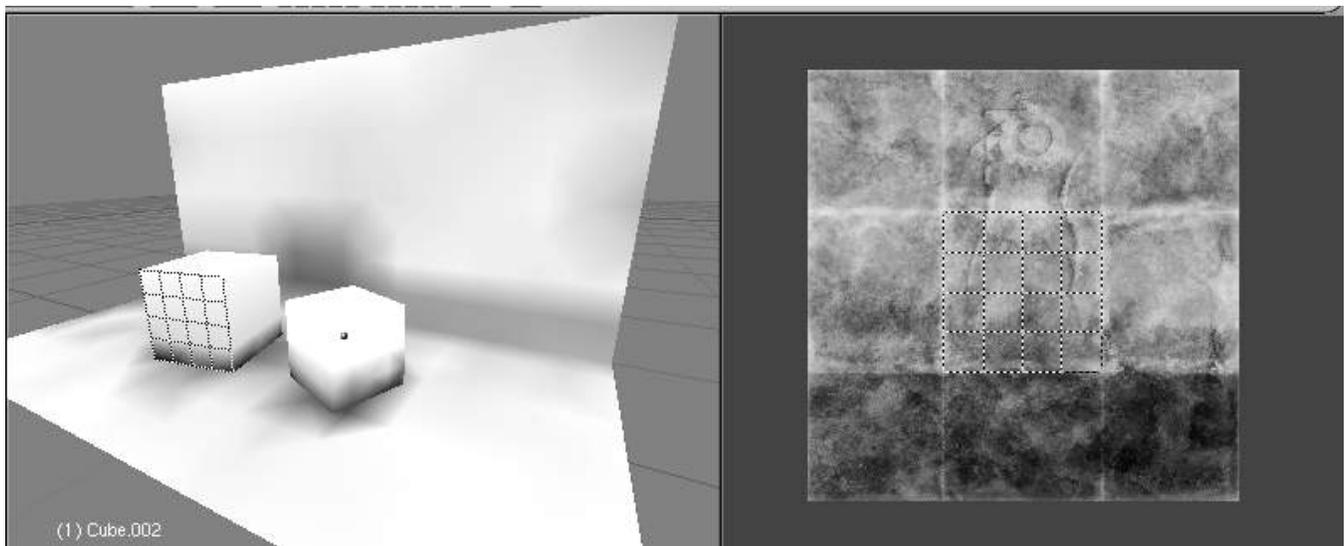


Fig. 2.1922: Vertex colors modulate texture.

A UV Layout can only have one image, although you can tile and animate the image. Since a layout is a bunch of arranged UV Maps, and a UV Map maps many mesh faces, a face can therefore only have one UV Texture image, and the UV coordinates for that face must fit entirely on the image. If you want a face to have multiple images, split the face into parts, and assign each part its own image. (*Or you can get fancy with Nodes, but that's another story ...*)

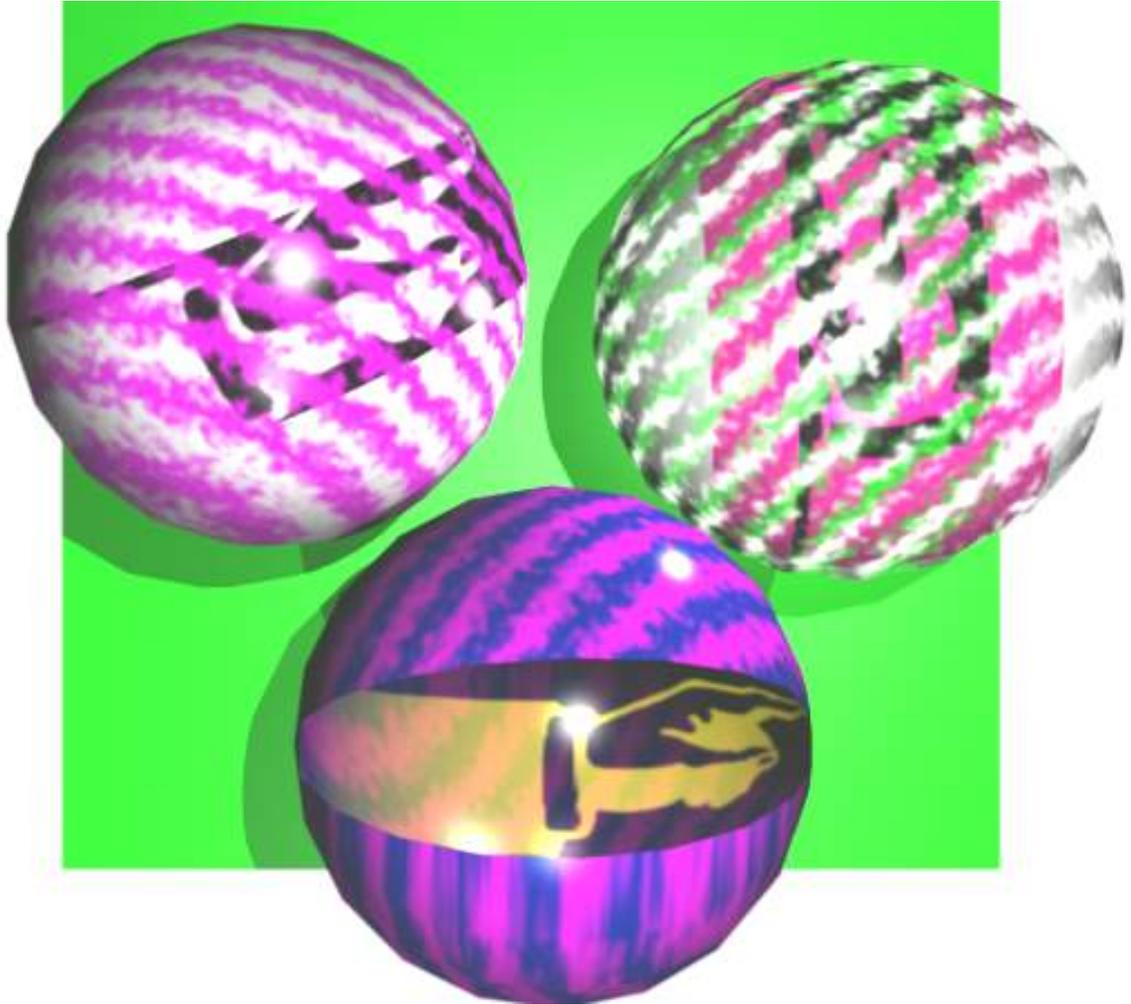


Fig. 2.1923: Alpha UV Textures

**Using Alpha Transparency** Alpha 0.0 (transparent) areas of a UV Image render as black. Unlike a procedural texture, they do not make the base material transparent, since UV Textures do not operate on the base procedural material. The UV texture overrides any procedural color underneath. Procedural Textures are applied on top of UV Textures, so a procedural image texture would override any UV Texture. Transparent (black) areas of a procedural texture mapped to alpha operate on top of anything else, making the object transparent in those places. The only thing that modulates visible parts of a UV Texture are the Vertex Colors. In the example to the right, the finger image is transparent at the cuff and top of the finger and is used as a UV Texture. All three balls have a base material of blue and a marbling texture. The base material color is not used whenever Face Textures is enabled.

The top left ball has not had any vertex painting, and the finger is mapped to the middle band, and the texture is mapped to a pink color. As you can see, the base material has VCol Paint and Face Textures enabled; the base color blue is not used, but the texture is. With no vertex painting, there is nothing to modulate the UV Texture colors, so the finger shows as white. Transparent areas of the UV Image show as black.

The top right ball has had a pink vertex color applied to the vertical band of faces (in the 3D View window, select the faces in UV Paint mode, switch to Vertex Paint mode, pick a pink color, and *Paint*→*Set Vertex Colors*). The finger is mapped to the middle vertical band of faces, and VCol and Face Textures are enabled. The texture is mapped to Alpha black and multiplies the base material alpha value which is 1.0. Thus, white areas of the texture are 1.0, and 1.0 times 1.0 is 1.0 (last time I checked, at least), so that area is opaque and shows. Black areas of the procedural texture, 0.0, multiply the base material to be transparent. As you can see, the unmapped faces (left and right sides of the ball) show the vertex paint (none, which is gray) and the painted ones show pink, and the middle stripe that is both painted and mapped change the white UV Texture areas to pink. Where the procedural texture says to make the object transparent, the green background shows through. Transparent areas of the UV Texture insist on rendering black.

The bottom ball uses multiple materials. Most of the ball (all faces except the middle band) is a base material that does not have Face Textures (nor Vertex Color Paint - VCol Paint) enabled. Without it enabled, the base blue material color shows and the pink color texture is mixed on top. The middle band is assigned a new material (2 Mat 2) that *does* have vertex paint and Face Textures enabled. The middle band of faces were vertex painted yellow, so the white parts of the finger are yellow. Where the pink texture runs over the UV texture, the mixed color changes to green, since pink and yellow make a green.

If you want the two images to show through one another, and mix together, you need to use Alpha. The base material can have an image texture with an Alpha setting, allowing the underlying UV Texture to show through.

To overlay multiple UV images, you have several options:

- Create multiple UV Textures which map the same, and then use different images (with Alpha) and blender will overlay them automatically.
- Use the [Composite Nodes](#) to combine the two images via the AlphaOver node, creating and saving the composite image. Open that composited image as the UV Texture.
- Use an external paint program to alpha overlay the images and save the file, and load it as the face's UV Texture
- Define two objects, one just inside the other. The inner object would have the base image, and the outer image the overlaid image with a material alpha less than one (1.0).
- Use the [Material nodes](#) to combine the two images via the AlphaOver or Mix node, thus creating a third noded material that you use as the material for the face. Using this approach, you will not have to UV map; simply assign the material to the face using the Multiple Materials

**UV Textures vs. Procedural Textures** A Material Texture, that has a Map Input of UV, and is an image texture that is mapped to Color, is equivalent to a UV Texture. It provides much more flexibility, because it can be sized and offset, and the degree to which it affects the color of your object can be controlled in the Map To panel. In addition, you can have different images for each texture channel; one for color, one for alpha, one for normals, one for specular, one for reflectivity, *etc.* Procedural textures, like Clouds, are INCREDIBLY simple and useful for adding realism and details to an image.

UV Texture	Procedural Texture
Image maps to precise coordinates on the selected faces of the mesh	Pattern is generated dynamically, and is mapped to the entire mesh (or portion covered by that material)
The Image maps once to a range of mesh faces specifically selected	Maps once to all the faces to which that material is assigned; either the whole mesh or a portion
Image is mapped once to faces.	Size XYZ in the MapInput allows tiling the texture many times across faces. Number of times depends on size of mesh
Affect the color and the alpha of the object.	Can also affect normals (bumpiness), reflectivity, emit, displacement, and a dozen other aspects of the mesh's appearance; can even warp or stencil subsequent textures.
Can have many for a mesh	Can be layered, up to 10 textures can be applied, layering on one another. Many mix methods for mixing multiple channels together.
Any Image type (still, video, rendered). Preset test grid available	Many different presents: clouds, wood grain, marble, noise, and even magic.
Provides the UV layout for animated textures	Noise is the only animated procedural texture
Takes very limited graphics memory	Uses no or little memory; instead uses CPU compute power

So, in a sense, a single UV texture for a mesh is simpler but more limited than using multiple textures (mapped to UV coordinates), because they do one specific thing very well: adding image details to a range of faces of a mesh. They work together if the procedural texture maps to the UV coordinates specified in your layout. As discussed earlier, you can map multiple UV textures to different images using the UV Coordinate mapping system in the Map Input panel.

## Settings

**Image** In the *Image Sampling* panel we tell Blender which source file to use. Image or Movie Datablock:

**Browse** Select an image or video among linked to the .blend file

**Name field** Internal name of image

**F** Create a fake user for the image texture

**+** Replace active texture with a new one

**Folder** Browse for an image on your computer

**X** Unlink this image or movie.

**Source:** Where the image come from. What kind of source file to use.

**Generated** Generated image in Blender.

**Movie** Movie file.

**Image Sequence** Multiple image files as a sequence.

**Single Image** Single image file.

**File for Image or Movie texture:** See about supported [Image](#) formats.

**Pack image** Embed image into current .blend file

**Path** Path to file

**File Browser** Find a file on your computer. Hold **Shift** to open the selected file and **Ctrl** to browse a containing directory.

**Reload** Reloads the file. Useful when an image has been rework in an external application.

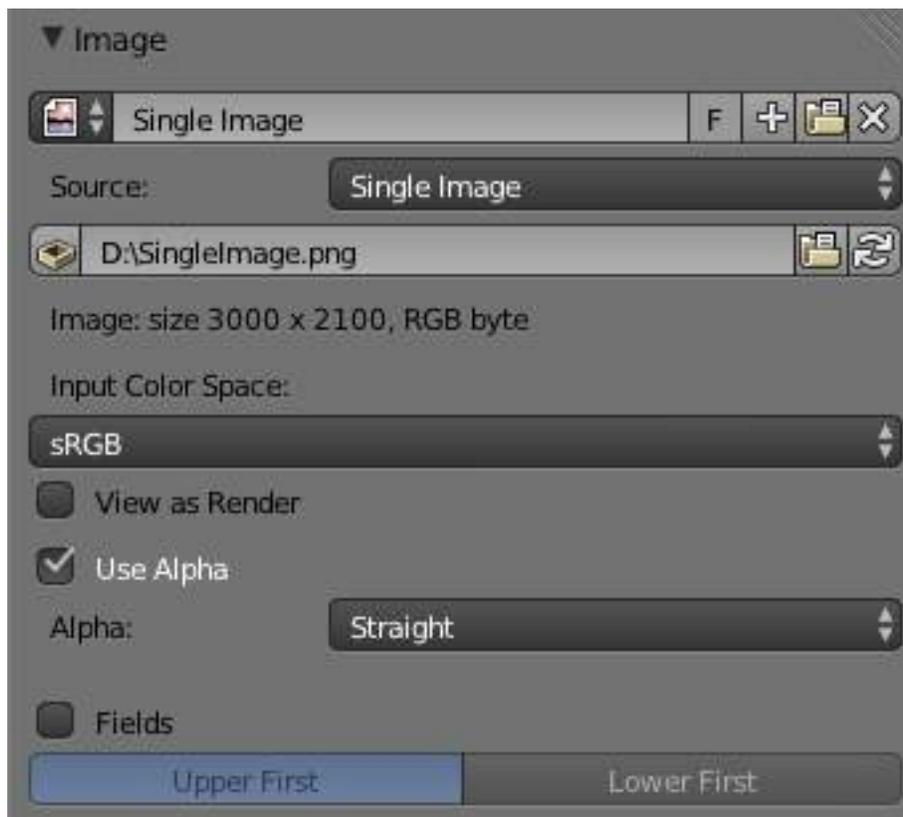


Fig. 2.1924: Image panel

**Input Color Space** Color space of the image or movie on disk

**XYZ** XYZ space.

**VD16** The simple video conversion from a gamma 2.2 sRGB space.

**sRGB** Standart RGB display space.

**Raw** Raw space.

**Non-Color** Color space used for images which contains non-color data (i.e. normal maps).

**Linear ACES** ACES linear space.

**Linear** 709 (full range). Blender native linear space.

**View as Render** Apply render part of display transformation when displaying this image on the screen.

**Use Alpha** Use the alpha channel information from the image or make image fully opaque

**Straight** Transparent RGB and alpha pixels are unmodified.

**Premultiplied** Transparent RGB pixels of an image are multiplied by the image's alpha value.

**Fields** Work with field images. Video frames consist of two different images (fields) that are merged. This option ensures that when *Fields* are rendered, the correct field of the image is used in the correct field of the rendering. *MIP Mapping* cannot be combined with *Fields*.

**Upper First** Order of video fields - upper field first.

**Lower First** Order of video fields - lower field first.

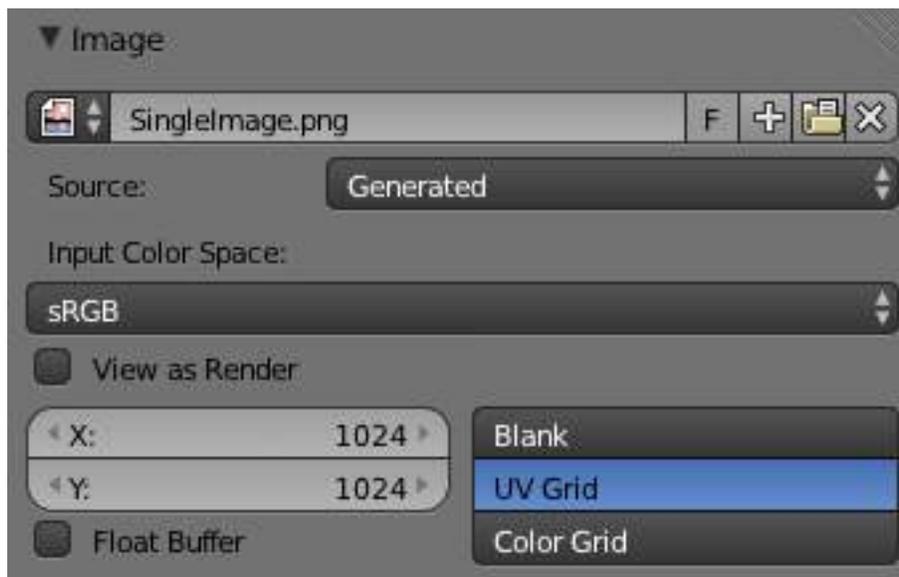


Fig. 2.1925: Image panel for Generated source of Image texture

For *Generated* source there are the specific options: *X* and *Y* size

Width and height of image to be generated.

**Generated Image Type** Which kind of image to be generated

**Blank** Generate a blank image.

**UV Grid** Generated grid to test UV mappings.

**Color Grid** Generated improved UV grid to test UV mappings.

**Float Buffer** Generate floating point buffer.

About specific options for **movie** and **image sequence** source. see [here](#)

**Image Sampling** In the *Image Sampling* panel we can control how the information is retrieved from the image.

Background image	Foreground image

The two images presented here are used to demonstrate the different image options. The *background image* is an ordinary JPG-file, the *foreground image* is a PNG-file with various alpha and greyscale values. The vertical bar on the right side of the foreground image is an Alpha blend, the horizontal bar has 50% alpha.

Foreground image with <i>Use</i> alpha. The alpha values of the pixels are evaluated	Foreground image with <i>Calculate</i> alpha

**Alpha** Options related to transparency

**Use** Works with PNG and TGA files since they can save transparency information (Foreground Image with UseAlpha). Where the alpha value in the image is less than 1.0, the object will be partially transparent and stuff behind it will show.

**Calculate** Calculate an alpha based on the RGB values of the Image. Black (0,0,0) is transparent, white (1,1,1) opaque. Enable this option if the image texture is a mask. Note that mask images can use shades of gray that translate to semi-transparency, like ghosts, flames, and smoke/fog.

**Invert** Reverses the alpha value. Use this option if the mask image has white where you want it transparent and vice-versa.

**Flip X/Y Axis** Rotates the image 90 degrees counterclockwise when rendered.

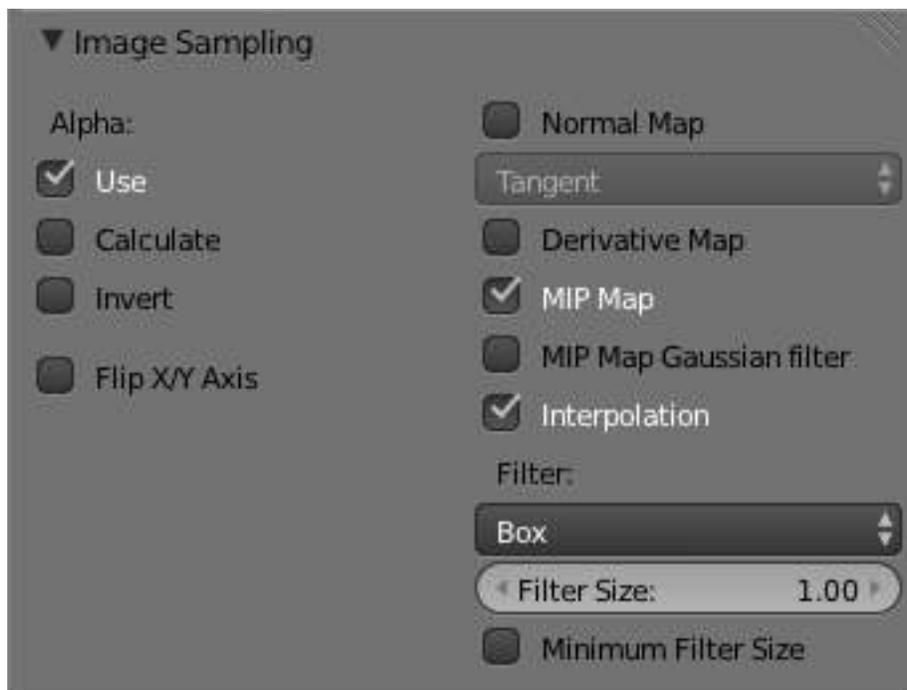


Fig. 2.1926: Image Sampling panel

**Normal Map** This tells Blender that the image is to be used to create the illusion of a bumpy surface, with each of the three RGB channels controlling how to fake a shadow from a surface irregularity. Needs specially prepared input pictures. See [Bump and Normal Maps](#).

**Normal Map Space:** *Tangent: Object: World: Camera:*

**Derivative Map** Use red and green as derivative values.

**MIP Map** **MIP Maps** are pre-calculated, smaller, filtered Textures for a certain size. A series of pictures is generated, each half the size of the former one. This optimizes the filtering process. By default, this option is enabled and speeds up rendering (especially useful in the game engine). When this option is OFF, you generally get a sharper image, but this can significantly increase calculation time if the filter dimension (see below) becomes large. Without MIP Maps you may get varying pictures from slightly different camera angles, when the Textures become very small. This would be noticeable in an animation.

**MIP Map Gaussian filter** Used in conjunction with MIP Map, it enables the MIP Map to be made smaller based on color similarities. In the game engine, you want your textures, especially your MIP Map textures, to be as small as possible to increase rendering speed and frame rate.

Table  
2.54:  
Enlarged  
Image  
texture  
without  
and with  
Interpolation



**Interpolation** This option interpolates the pixels of an image. This becomes visible when you enlarge the picture. By default, this option is on. Turn this option OFF to keep the individual pixels visible and if they are correctly anti-aliased. This last feature is useful for regular patterns, such as lines and tiles; they remain ‘sharp’ even when enlarged considerably. When you enlarge this 10x10 pixel Image



the difference with and without *Interpolation* is clearly visible. Turn this image off if you are using digital photos to preserve crispness.

**Filter** The filter size used in rendering, and also by the options *MipMap* and *Interpolation*. If you notice gray lines or outlines around the textured object, particularly where the image is transparent, turn this value down from 1.0 to 0.1 or so.

**Texture Filter Type** Texture filter to use for image sampling. Just like a *pixel* represents a *pic ture el ement*, a *texel* represents a *tex ture el ement*. When a texture (2D texture space) is mapped onto a 3D model (3D model space), different algorithms can be used to compute a value for each pixel based on samplings from several texels.

**Box** A fast and simple nearest-neighbor interpolation known as Monte Carlo integration

**EWA** **E**lliptical **W**eighted **A**verage - one of the most efficient direct convolution algorithms developed by Paul Heckbert and Ned Greene in the 1980s. For each texel, EWA samples, weights, and accumulates texels within an elliptical footprint and then divides the result by the sum of the weights.

**Eccentricity** Maximum Eccentricity. Higher values give less blur at distant/oblique angles, but is slower

**FELINE** **FELINE** (**F**ast **E**lliptical **L**ine **s**), uses several isotropic probes at several points along a line in texture space to produce an anisotropic filter to reduce aliasing artifacts without considerably increasing rendering time.

**Probes** Number of probes to use. An integer between 1 and 256. Further reading: McCormack, J; Farkas, KI; Perry, R; Jouppe, NP (1999) *Simple and Table Feline: Fast Elliptical Lines for Anisotropic Texture Mapping*, WRL

**Area** Area filter to use for image sampling

**Eccentricity** Maximum Eccentricity. Higher values give less blur at distant/oblique angles, but is slower

**Filter Size** The filter size used by MIP Map and Interpolation

**Minimum Filter Size** Use Filter Size as a minimal filter value in pixels

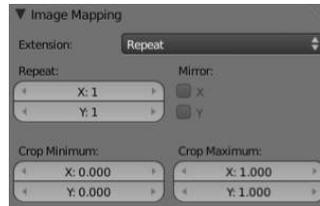


Fig. 2.1927: Image Mapping panel

**Image Mapping** In the *Image Mapping* panel, we can control how the image is mapped or projected onto the 3D model.

Extension:

**Extend** Outside the image the colors of the edges are extended

**Clip** Clip to image size and set exterior pixels as transparent. Outside the image, an alpha value of 0.0 is returned. This allows you to ‘paste’ a small logo on a large object.

**Clip Cube** Clips to cubic-shaped area around the images and sets exterior pixels as transparent. The same as Clip, but now the ‘Z’ coordinate is calculated as well. An alpha value of 0.0 is returned outside a cube-shaped area around the image.

**Repeat** The image is repeated horizontally and vertically

**Repeat** X/Y repetition multiplier

**Mirror** Mirror on X/Y axes. This buttons allow you to map the texture as a mirror, or automatic flip of the image, in the corresponding X and/or Y direction.

**Checker** Checkerboards quickly made. You can use the option *size* on the *Mapping* panel as well to create the desired number of checkers.

**Even / Odd** Set even/odd tiles

**Distance** Governs the distance between the checkers in parts of the texture size

**Crop Minimum / Crop Maximum** The offset and the size of the texture in relation to the texture space. Pixels outside this space are ignored. Use these to crop, or choose a portion of a larger image to use as the texture.

## Video Textures

**Video textures** are a some kind of **Image** textures and based on movie file or sequence of successive numbered separate images. They are added in the same way that image textures are.

## Options

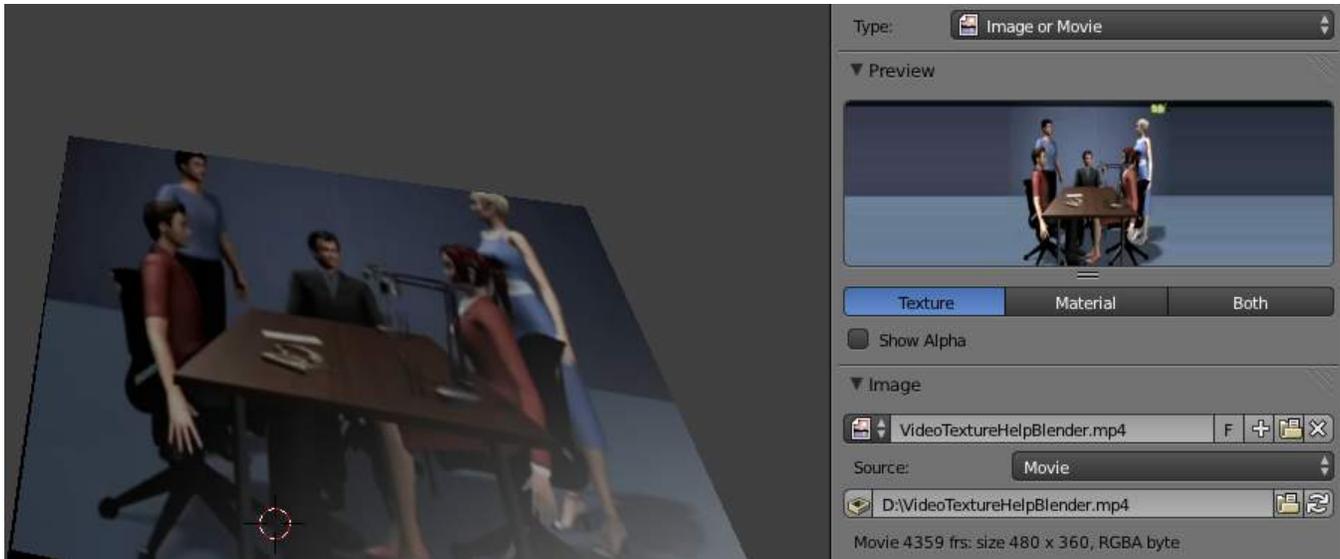


Fig. 2.1928: Video texture

## Image

**Source** For video texture the kind of source file to use is

**Movie** See about supported [Movie](#) formats.

**Image Sequence** See about supported [Image](#) formats. To load image sequence in any of the supported image file formats first click on the first frame and then Accept. Then change the Source to Image Sequence, and enter the ending frame number of this sequence.

More about loading source file for video texture see [here](#).

**Fields** Work with field images. Video frames consist of two different images (fields) that are merged. This option ensures that when *Fields* are rendered, the correct field of the image is used in the correct field of the rendering.

**Upper First** Order of video fields - upper field first.

**Lower First** Order of video fields - lower field first.

**Fields** Number of fields per rendered frame. Used with *Fields* and interlaced video, it says whether each image has both odd and even, or just one.

**Frames** Number of frames/images in the movie or sequence to use

**Start** Global starting frame of the sequence/movie

**Offset** Offset the number of the frame to use in the animation. What frame number inside the movie/sequence to start grabbing.

**Match Movie Length** This button set image's user's length to the one of selected movie/sequence.

**Auto Refresh** Automatically refresh images on frame changes

**Cyclic** When the video ends, it will loop around the to the start and begin playing again.

For *Movie* source:

**Use Alpha** Use the alpha channel information from the image or make image fully opaque

**Straight** Transparent RGB and alpha pixels are unmodified.

**Premultiplied** Transparent RGB pixels of an image are multiplied by the image's alpha value.

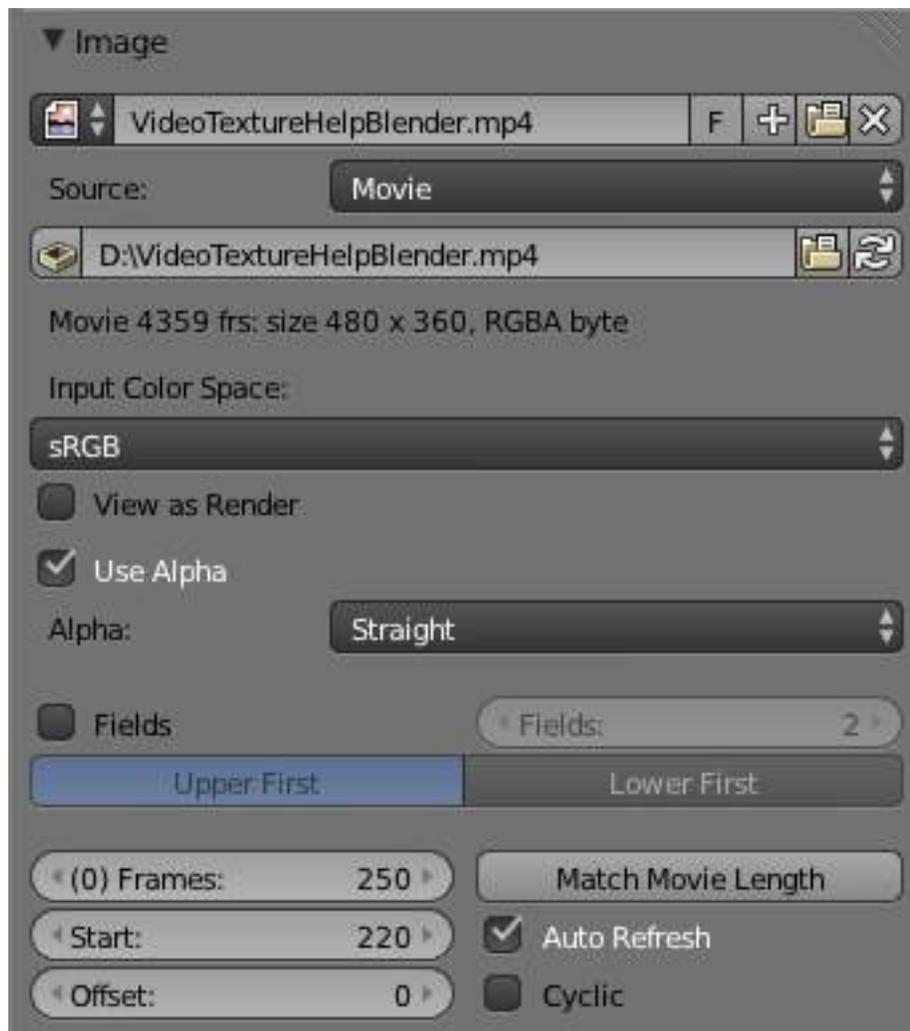


Fig. 2.1929: Image panel for video texture

About input color space for video texture see [here](#).

About video sampling for video texture see [here](#).

About video mapping for video texture see [here](#).

## Texture Nodes

As an alternative to using the [Texture Stack](#), Blender includes a node-based texture generation system which enables you to create textures by combining colors, patterns and other textures in much the same way that you combine [Material Nodes](#).

You can use these textures wherever you can use regular textures: you can place them in texture channels, in material nodes, in particle systems, and even inside other textures.

FIXME(Template Unsupported: Doc:2.6/Reference/Nodes/Concepts; { {Doc:2.6/Reference/Nodes/Concepts} } )

---

**Note:** Node-based textures do **not** work for realtime display, they will only be visible in rendered images.

---

**Using Texture Nodes** To use texture nodes with the current texture, open a [Node Editor window](#), set it to *Texture* mode by clicking the “Texture” icon (



Fig. 2.1930: Texture

) in its header.

To start adding nodes, you first need to select a material. Now you can either click the *New* button in the Node editor, or the *New* button in the texture panel. Once you have a texture selected, you can toggle it to function as a regular texture or a node texture by clicking the *Use Nodes* option in the Node Editor.

The default node setup will appear: a red-and-white checkerboard node connected to an *Output* named *Default*. For *texture* nodes, you can create as many *Outputs* as you like in your node setup. (Other types of node networks, as you may recall, are limited to only one *Output* node.) See the next section for details.

For instructions on how to add, remove and manipulate the nodes in the tree, see the [Node Editor manual](#).

**Using Multiple Outputs** Each texture that you define with Texture Nodes can have several outputs, which you can then use for different things. For example, you might want your texture to define both a diffuse (color) map and a normal map. To do this, you would:

- Create two texture slots in the texture list, and set them to the same texture datablock.
- Add two *Output* nodes to the node tree, and type new names into their *Name* text-boxes: e.g. *Diffuse* for one and *Normal* for the other.
- Underneath the texture picker in the texture panel, you’ll see a dropdown list with the names of your outputs. For each entry in the texture list, select the desired output by changing the menu entry e.g. set on to *Diffuse* and the other to *Normal*).

You can also use these named outputs if you’ve decided to define your material using [Material Nodes](#). In this case, you probably won’t be using [Texture Channels](#). Instead, you’ll insert *Texture* nodes into your [Material Node tree](#) using *Add* → *Input* → *Texture*. Then, inside the texture node that you’ve just added, you can select which output you want to use (e.g. *Diffuse* or *Normal*).

## Node Editor

FIXME(Template Unsupported: Doc:2.6/Reference/Nodes/Node Editor; {{Doc:2.6/Reference/Nodes/Node Editor}} )

## Node Controls

FIXME(Template Unsupported: Doc:2.6/Reference/Nodes/Node\_Controls; {{Doc:2.6/Reference/Nodes/Node\_Controls}} )

## Using Nodes

FIXME(Template Unsupported: Doc:2.6/Reference/Nodes/Using\_Nodes; {{Doc:2.6/Reference/Nodes/Using\_Nodes}} )

## Node Groups

FIXME(Template Unsupported: Doc:2.6/Reference/Nodes/Node\_Groups; {{Doc:2.6/Reference/Nodes/Node\_Groups}} )

## Input Nodes

Input nodes provide input data for other nodes.

Time

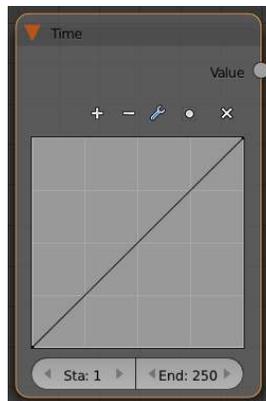


Fig. 2.1931: Time node

The time node uses a frame range to output a value between 0 and 1. By default the node output a linear transition from 0 to 1 from frame 1 to 250. The shape of the curve can be manipulated to vary the output over time in different ways.

Plus:Zoom in. Minus:Zoom out *Tools:*

**Reset View** Resets curve view

**Vector Handle** Breaks tangent at curve handle, making a angle.

**Auto Handle** Default smooth interpolation of curve segments

**Extend Horizontal** Causes the curve to stay horizontal before the first point and after the last point.

**Clipping Options:**

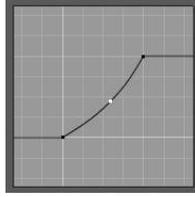


Fig. 2.1932: Extend Horizontal

**Extend Extrapolated** Causes the curve to extrapolate before the first point and after the last point, based on the shape of the curve.

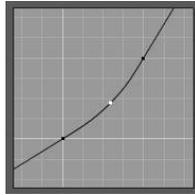


Fig. 2.1933: Extend Extrapolate

**Reset Curve** Resets shape of curve to original linear shape.

**Use Clipping** Forces curve points to stay between specified values.

**Min X/Y and Max X/Y** Set the minimum and maximum bounds of the curve points.

**X:**Delete curve points. The first and last points cannot be deleted. **X and Y** The coordinates of the selected edit point. **Sta:**Specify the start frame to use. **End:**Specify the end frame to use.

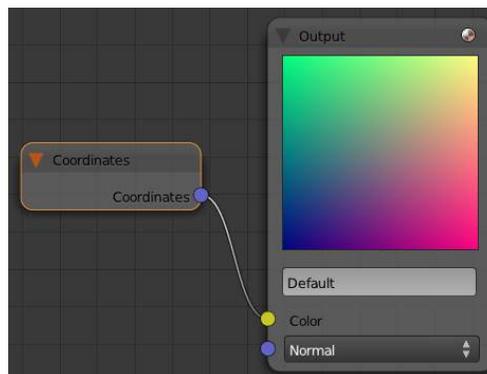


Fig. 2.1934: Coordinates node

**Coordinates** The Coordinates node outputs the geometry local coordinates, relative to its bounding box as RGB colors:

- Red channel corresponds to X value.
- Green channel corresponds to Y value.
- Blue channel corresponds to Z value.

**Texture Node** The texture node can be used to load a another node based or non-node based texture.

**Color 1 and Color 2** These can be used to remap a greyscale texture using two colors.

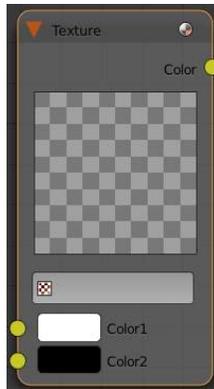


Fig. 2.1935: Texture node

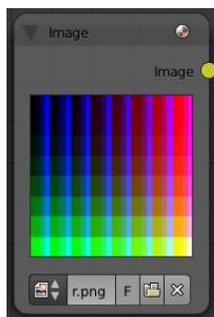


Fig. 2.1936: Image node

**Image Node** The image node can be used to load an external image.

**Browse for image** Select an image that already exists in the scene.

**Datablock name** Set the name of the image datablock.

**F** Save this image datablock, even if it has no users.

**Open image** Select image to use from file browser.

**Unlink datablock** Remove the image datablock from the node.

### Output Nodes

These node serves a outputs for node textures

**Output** This node contains the result of the node texture. Multiple output nodes can exist in a node texture, however only one of them is active. The active one is set in the Texture Panel in the *Output* drop down.

**Color** The color data that the texture renders

**Normal** The normal map that the texture will output.

**Viewer** The viewer node can be used to preview the results of a node.

## Texture Color Nodes



Fig. 2.1937: mix node

**Mix** This node mixes a base color or image (threaded to the top socket) together with a second color or image (bottom socket) by working on the individual and corresponding pixels in the two images or surfaces. The way the output image is produced is selected in the drop-down menu. The size (output resolution) of the image produced by the mix node is the size of the base image. The alpha and Z channels (for compositing nodes) are mixed as well.

**See also:**

[Color Blend Modes](#) for details on each blending mode.

**Note:** Color Channels

There are two ways to express the channels that are combined to result in a color: RGB or HSV. RGB stands for the Red,Green,Blue pixel format, and HSV stands for Hue,Saturation,Value pixel format.

**Clamp** Clamps the result of the mix operation between 0 and 1. Some of the mix types can produce results above 1 even if the inputs are both between 0 and 1, such as Add.

**Factor** The amount of mixing of the bottom socket is selected by the Factor input field (Fac:). A factor of zero does not use the bottom socket, whereas a value of 1.0 makes full use. In Mix mode, 0.5 is an even mix between the two, but in Add mode, 0.5 means that only half of the second socket's influence will be applied.

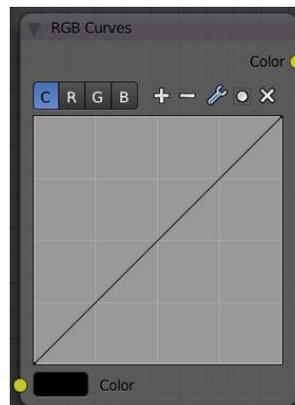


Fig. 2.1938: RGB Curves node

**RGB Curves** For each color component channel (RGB) or the composite (C), this node allows you to define a bezier curve that varies the input (x-axis) to produce an output value (y-axis). Clicking on one of the *C R G B* components displays the curve for that channel.

**See also:**

- Read more about using the *Curve Widget*.
- [RGB Curves node in the compositor](#) (includes examples)



Fig. 2.1939: invert node

**Invert** This node simply inverts the input values and colors.

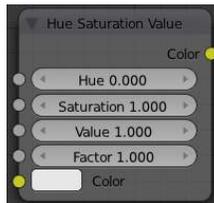


Fig. 2.1940: Hue Saturation Value node

**Hue Saturation Value** Use this node to adjust the Hue, Saturation, and Value of an input.



Fig. 2.1941: Combine RGB node

**Combine and Separate RGB** These two nodes allow you to convert between float values and color values. Colors are composed of 3 or 4 channels; red, green, blue, and sometimes alpha.

With Combine RGB, you can specify the values of each channel, and the node will combine them into a color value.

With Separate RGB, you can specify a color value, and get each channel value out of it.

### Pattern Nodes

**Checker** The checker node creates a checkerboard pattern

**color 1/color 2** Sets the color of the squares

**Size** The scale of the checker pattern

**Bricks** The Bricks node creates a brick like pattern

**Offset** The relative offset of the next row of bricks

**Frequency** Offset every N rows. The brick pattern offset repeats every N rows.



Fig. 2.1942: Separate RGB node

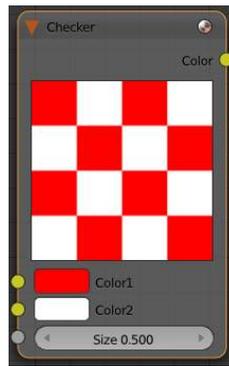


Fig. 2.1943: Checker node



Fig. 2.1944: Bricks node

**Squash** Scales the bricks in every N rows by this amount.

**Frequency** Squash every N rows.

**Bricks 1, Bricks 2** Sets the color range of the bricks. Brick colors are chosen randomly between these two colors.

**Mortar** Sets the mortar color, in between the bricks.

**Thickness** Sets the thickness of the mortar

**Bias** The bias of randomly chosen colors, between -1 and 1. -1 Makes all bricks Color 1, and a value of 1 makes them all Color 2.

**Brick Width** Sets the horizontal size of all the bricks.

**Row Height** Sets the verticalsize of all the bricks.

### Texture Nodes

These nodes generat procedural textures, and function just like their non node based counterparts.

### Common Options

**Color 1/Color 2** Remaps the procedural texture with these colors. These do not function in the Magic node.

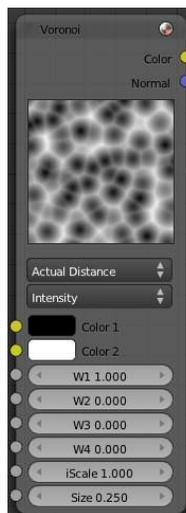


Fig. 2.1945: Voronoi node

**Voronoi** [See Here](#)

**Blend** [See Here](#)

**Magic** [See Here](#)

**Marble** [See Here](#)



Fig. 2.1946: Blend node

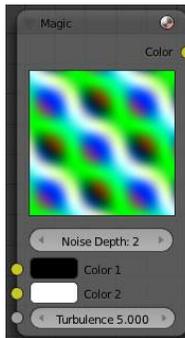


Fig. 2.1947: Magic node

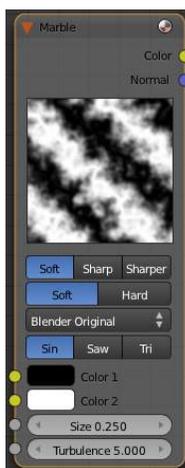


Fig. 2.1948: Marble node



Fig. 2.1949: Clouds node

**Clouds** [See Here](#)

**Wood**



Fig. 2.1950: Wood node

[See Here](#)

**Musgrave** [See Here](#)

**Noise** [See Here](#)

**Stucci** [See Here](#)

**Distorted Noise** [See Here](#)



Fig. 2.1951: Musgrave



Fig. 2.1952: Noise



Fig. 2.1953: Stucci



Fig. 2.1954: Distorted Noise node

### Texture Converter Nodes

As the name implies, these nodes convert the colors in the material in some way.

Math

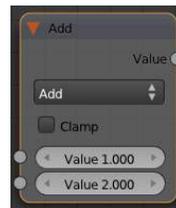


Fig. 2.1955: math node

The math node performs one of several math functions on one or two inputs

**Clamp** Clamps the result between 0 and 1.

**Add** Add the two inputs

**Subtract** Subtract input 2 from input 1

**Multiply** Multiply the two inputs

**Divide** Divide input 1 by input 2

**Sine** The sine of input 1 (degrees)

**Cosine** The cosine of input 1 (degrees)

**Tangent** The tangent of input 1 (degrees)

**Arcsine** The arcsine (inverse sine) of input 1 (degrees)

**Arccosine** The arccosine (inverse cosine) of input 1 (degrees)

**Arctangent** The arctangent (inverse tangent) of input 1 (degrees)

**Power** Input 1 to the power of input 2 ( $\text{input1}^{\text{input2}}$ )

**Logarithm** log base input 2 of input 1

**Minimum** The minimum of input 1 and input 2

**Maximum** The maximum of input 1 and input 2

**Round** Rounds input 1 to the nearest integer

**Less Than** Test if input 1 is less than input 2, returns 1 for true, 0 for false

**Greater Than** Test if input 1 is greater than input 2, returns 1 for true, 0 for false

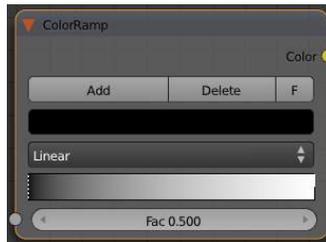


Fig. 2.1956: ColorRamp Node

**ColorRamp Node** The ColorRamp Node is used for mapping values to colors with the use of a gradient. It works exactly the same way as a [Colorband for textures and materials](#), using the Factor value as a slider or index to the color ramp shown, and outputting a color value and an alpha value from the output sockets.

By default, the ColorRamp is added to the node map with two colors at opposite ends of the spectrum. A completely black is on the left (Black as shown in the swatch with an Alpha value of 1.00) and a whitewash white is on the right. To select a color, LMB click on the thin vertical line/band within the colorband. The example picture shows the black color selected, as it is highlighted white. The settings for the color are shown above the colorband as (left to right): color swatch, Alpha setting, and interpolation type.

To change the hue of the selected color in the colorband, LMB click on the swatch, and use the pop-up color picker control to select a new color. Press `Return` to set that color.

To add colors, hold `Ctrl` down and `Ctrl-LMB` click inside the gradient. Edit colors by clicking on the rectangular color swatch, which pops up a color-editing dialog. Drag the gray slider to edit Alpha values. Note that you can use textures for masks (or to simulate the old “Emit” functionality) by connecting the alpha output to the factor input of an RGB mixer.

To delete a color from the colorband, select it and press the Delete button.

When using multiple colors, you can control how they transition from one to another through an interpolation mixer. Use the interpolation buttons to control how the colors should band together: Ease, Cardinal, Linear, or Spline.

Use the A: button to define the Alpha value of the selected color for each color in the range.



Fig. 2.1957: RGB to BW Node

**RGB to BW Node** This node converts a color image to black-and-white by computing the luminance of the rgb values.

**Value to Normal** Computes a normal map based on greyscale values of an input

**Val** The texture to compute the normal map from

**Nabla** Size of derivative offset used for calculating normals.



Fig. 2.1958: Value to Normal node

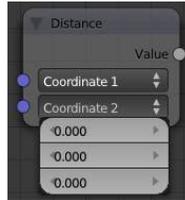


Fig. 2.1959: Distance node. Coordinate 2 dropdown is displayed

**Distance** Computes the distance between two 3d coordinates.

### Distort Nodes

These nodes allow you to change the mapping of a texture.



Fig. 2.1960: Rotate node

**Rotate** Rotate the texture coordinates of an image or texture.

**Turns** The number of times to rotate the coordinates 360 degrees about the specified axis.

**Axis** The axis to rotate the mapping about

**Translate** Translate the texture coordinates of an image or texture.

**Offset** The amount to offset the coordinates in each of the 3 axes.

**Scale** Scale the texture coordinates of an image or texture.

**Scale** The amount to scale the coordinates in each of the 3 axes.

**At** Returns the color of a texture at the specified coordinates. If the coordinates are not spatially varying, the node will return a single color.

**Coordinates** The point at which to sample the color. For images, the space is between -1 and 1 for x and y.



Fig. 2.1961: Translate node

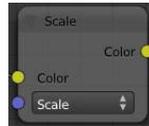


Fig. 2.1962: Scale node

## Volume Textures

Blender has two textures that can be applied to volumetric data:

**Voxel Data** Voxel data renders a voxel source. It can be used for rendering Blender's internal smoke simulations. Other sources include binary raw formats, and Image Sequence, which can be used to stack a sequence of images into a 3D representation

**Point Density** Point density renders a given point cloud (object vertices or particle system) as a 3D volume

### Voxel Data

Voxel data renders a voxel source, working very similarly to an image texture, but in 3d. Various input data source types are available (such as smoke voxel data, or external files), as well as various interpolation methods.

The voxels are stored in a flat z/y/x grid of floats. Functions for sampling this based on location within the (0,1) bounds are available in:

- `source/blender/blenlib/intern/voxel.c`

The default voxel data source, Smoke, is used for rendering Blender's internal smoke simulations. Other sources include binary raw formats, and Image Sequence, which can be used to stack a sequence of images into a 3D representation, which is a common format for medical volume data such as CT scans.

## Settings

### File Format

**Blender Voxel** Default binary voxel file format.

**8 bit RAW** 8 bit grayscale binary data.

**Image Sequence** Generate voxels from a sequence of image slices.

**Smoke** Render voxels from a Blender smoke simulation.



Fig. 2.1963: At node

**Source Path** The external source data file to use for 8 bit Raw data and Blender Voxel formats

**Domain Object (Smoke)** Object used as the smoke simulation domain

#### Source

**Smoke** Use smoke density and color as texture data.

**Flame** Use flame temperature as texture data.

**Heat** Use smoke heat as texture data. Values from -2.0 to 2.0 are used.

**Velocity** Use smoke velocity as texture data.

**Resolution** Resolution of the voxel grid when using 8 bit Raw data.

#### Interpolation

**Nearest Neighbor** No interpolation, fast but blocky and low quality.

**Linear** Good smoothness and speed.

**Quadratic** Mid-range quality and speed.

**Cubic Catmull-Rom** Smoothed high quality interpolation, but slower.

#### Extension

**Extend** Extend by repeating edge pixels of the image.

**Clip** Clip to image size and set exterior pixels as transparent.

**Repeat** Cause the image to repeat horizontally and vertically.

**Intensity** Multiplier for intensity values

### Point Density Texture

Point density renders a given point cloud (object vertices or particle system) as a 3D volume, using a user-defined radius for the points. Internally, the system uses a BVH data structure for fast range lookups.

The rendered points are spherical by default, with various smooth falloff options, as well as simple Turbulence options for displacing the result with noise, adding fine detail. When using Point Density with a particle system, additional particle info such as particle velocity, age, and speed, can be visualized using a color/alpha ramp gradient.

#### Options

**Particle System** Particle System, Generate point density from a particle system.

**Object Vertices** Object Vertices, Generate point density from an object's vertices.

Object Radius System Falloff

Standard Smooth Soft Softness

**Constant** Density is constant within lookup radius.

Root Particle Age Particle Velocity Velocity Scale

**Falloff Curve** Use a custom falloff

**Cache** Coordinate system to cache particles in Global Space Emit Object Space Emit Object Location

**Color Source** Data to derive the color results from

**Constant** Constant color

**Particle Age** Lifetime mapped as 0.0 - 1.0 intensity.

**Particle Speed** Particle speed (absolute magnitude of velocity) mapped as 0.0-1.0 intensity.

**Scale** Multiplier to bring particle speed within an acceptable range.

**Particle Velocity** XYZ velocity mapped to RGB colors.

**Scale** Multiplier to bring particle speed within an acceptable range.

**Turbulence** Adds directed noise to the density at render time

**Influence** Method for driving added turbulent noise

**Static** Noise patterns will remain unchanged, faster and suitable for stills.

**Particle Velocity** Turbulent noise driven by particle velocity.

**Particle Age** Turbulent noise driven by the particle's age between birth and death.

**Global Time** Turbulent noise driven by the global current frame.

**Noise Basis** See [Here](#)

**Size** Scale of the turbulent noise

**Depth** Level of detail in the added turbulent noise

**Turbulence Strength** Strength of the added turbulent noise

## Ocean

### Texture Painting

A UV Texture is a picture (image, sequence or movie) that is used to color the surface of a mesh. The UV Texture is mapped to the mesh through one or more UV maps. There are three ways to establish the image used by the UV Texture:

- Paint a flat image in the UV/Image Editor onto the currently selected UV Texture, using its UV map to transfer the colors to the faces of the mesh.
- Paint the mesh in the 3D View, and let Blender use the currently selected UV map to update the UV Texture (see [Projection Painting](#)).
- Use any image-editing (paint) program to create an image. In the UV/Image Editor, select the UV Texture and load the image. Blender will then use that texture's UV map to transfer the colors to the faces of the mesh

Blender features a built-in paint mode called Texture Paint which is designed specifically to help you edit your UV Textures and images quickly and easily in either the UV/Image Editor window or the 3D View window. Since a UV Texture is just a special-purpose image, you can also use any external paint program. For example, GIMP is a full-featured image manipulation program that is also open-source.

Since a mesh can have layers of UV Textures, there may be many images that color the mesh. However, each UV Texture only has one image.

Texture Paint works in both a 3D window and the UV/Image Editor window. In the 3D window in Texture Paint mode, you paint directly on the mesh by [projecting onto the UVs](#).

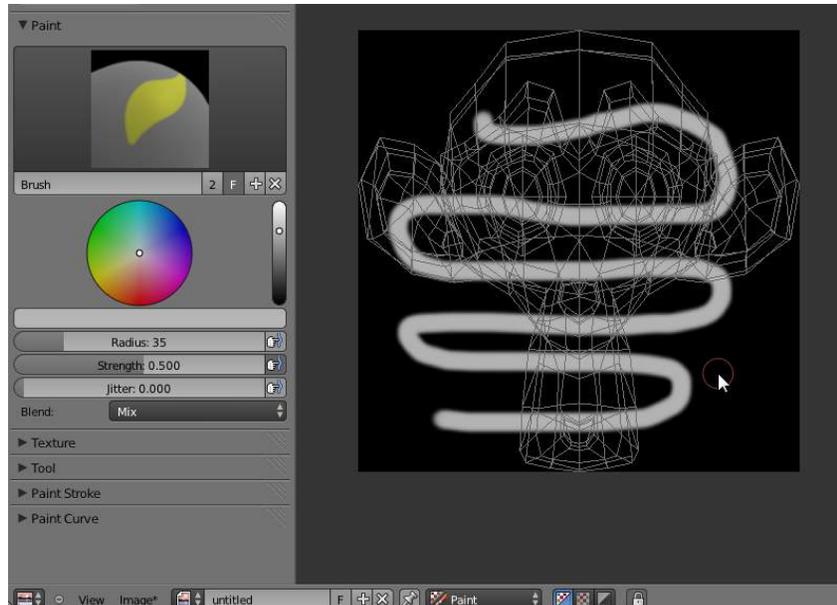


Fig. 2.1964: Texture painting in Blender

**Getting Started** Once you have unwrapped your model to a UV Map (as explained in previous pages), you can begin the texturing process. You cannot paint on a mesh in Texture Paint mode without **first** unwrapping your mesh, **and** doing one of the following steps. Either:

- Load an image into the UV/Image Editor (Image→Open→select file), or
- Create a new image (Image→New→specify size).

After you have done one of these two things, you can modify the image using the Texture Paint mode:

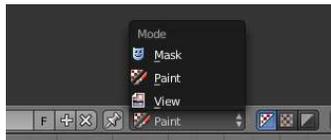


Fig. 2.1965: Enabling paint mode

- In the 3D View window, select Texture Paint mode from the mode selector in the window header, and you can paint directly onto the mesh.
- In the UV/Image Editor window, switch the editing context from View to Paint (shown to the right).

**Note:** Square Power of 2

Texture paint is very fast and responsive when working in the 3D window and when your image is sized as a square where the side lengths are a power of two, e.g. 256x256, 512x512, 1024x1024, etc.

Once you enable Texture Painting, your mouse becomes a brush. To work with the UV layout (for example, to move coordinates) you must go back to “View” mode.

As soon as you enable Texture Painting or switch to Texture Paint mode, brush settings become available in the Toolbar Panel (T-key).

In the UV/Image Editor window, you paint on a flat canvas that is wrapped around the mesh using UV coordinates. Any changes made in the UV/Image Editor window show up immediately in the 3D window, and vice versa.

A full complement of brushes and colors can be selected from the Properties panel in the UV/Image Editor. Brush changes made in either panel are immediately reflected in the other panel. However, the modified texture will **not** be saved automatically; you must explicitly do so by Image→Save in the UV/Image Editor window.

**Texture Preview** If your texture is already used to color, bump map, displace, alpha-transparent, etc., a surface of a model in your scene (in other techie words, is mapped to some aspect of a texture via a texture channel using UV as a map input), you can see the effects of your painting in the context of your scene as you paint.

To do this, set up side-by-side windows, one window in 3D View set to Textured display mode, and the second UV/Image Editor window loaded with your image. Position the 3D View to show the object that is UV mapped to the loaded image. Open a Preview window (see 3D View Options for more info) and position it over the object. In the image to the right, the texture being painted is mapped to the “Normal” attribute, and is called “bump mapping”, where the gray-scale image is used to make the flat surface appear bumpy. See Texture Mapping Output for more information on bump mapping.

**Brushes Settings** Press T in the UV/Image Editor to show the Toolbar panel. With this panel, you can create many brushes, each with unique settings (such as color and width). Use the Brush selector to switch between brushes, or to create a new brush. When you add a brush, the new brush is a clone of the current one. You can then change the setting for the new brush. Texture paint has an unlimited number of brushes and unique user-defined controls for those brushes which can be set in the Paint Tool panel.

To use a brush, click on its name. Use the selector up/down arrow, if there are more brushes on the flyout window than can be displayed at once. Name your brush by clicking on the name field and entering any name you wish, such as “Red Air” for a red airbrush. To toss out a brush, click the brush delete X button next to its name. If you want to keep this brush around for the next time you run Blender, click the F make user button next to the brush delete X button.

If you have a tablet pen with pressure sensitivity, toggle the small “P” button next to the opacity, size, falloff and spacing buttons to control these parameters using the pressure of the pen. Using your pen’s eraser end will toggle on the Erase Alpha mode.

Press S on any part of the image to sample that color and set it as the brush color.

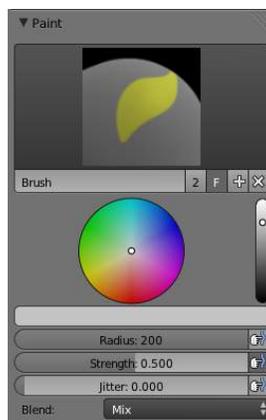


Fig. 2.1966: Brush Settings

## Brush

**Brush presets** Select a preset brush. Most brushes have common settings.

## Types of brushes

### There are four different types of brushes

**Draw** the normal brush; paints a swath of color

**Soften** blends edges between two colors

**Smear** when you click, takes the colors under the cursor, and blends them in the direction you move the mouse. Similar to the “smudge” tool of *Gimp*.

**Clone** copies the colors from the image specified (Tex.Dirt in the example), to the active image. The background image is shown when this brush is selected; use the *B* lend slider to control how prominent the background image is.

**Enable Pressure Sensitivity** The icon to the right of the following three settings will enable or disable tablet pressure sensitivity to control how strong the effect is.

**Color** The color of the brush

**Radius** The radius of the brush in pixels

**Strength** How powerful the brush is when applied}}

**Blend** Set the way the paint is applied over the underlying texture

- Mix: the brush color is mixed in with existing colors
- Add: the brush color is added to the existing color; green added to red gives yellow.
- Subtract: the brush color is subtracted; painting blue on purple gives red
- Multiply: the RGB value of the base is multiplied by the brush color
- Lighten: the RGB value of the base color is increased by the brush color
- Darken: tones down the colors
- Erase Alpha: makes the image transparent where painted, allowing background colors and lower-level textures to show through. As you ‘paint’, the false checkerboard background will be revealed
- Add Alpha: makes the image more opaque where painted

In order to see the effects of the Erase and Add Alpha mix modes in the UV/Image Editor, you must enable the alpha channel display by clicking the Display Alpha or the Alpha-Only button. Transparent (no alpha) areas will then show a checkered background.

**Image** When using the clone brush, this allows you to select an image as a clone source.

**Alpha** Opacity of the clone image display

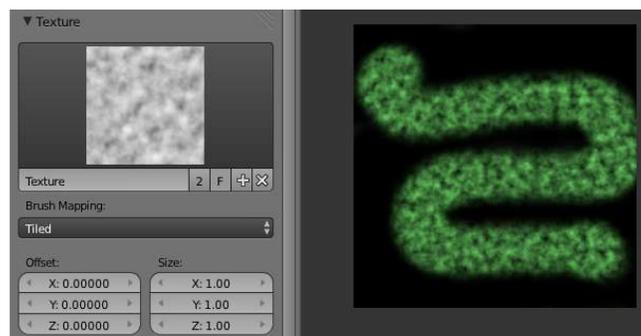


Fig. 2.1967: Texture options and example

**Texture** Use the texture selector at the bottom of the paint panel to select a pre-loaded image or procedural texture to use as your brush pattern. Note that in order to use it, you must have a placeholder material defined, and that particular texture defined using the Material and Texture buttons. It is not necessary to have that material or texture applied to any mesh anywhere; it must only be defined. The example to the right shows the effects of painting with a flat (banded) wood texture. Switching the texture to Rings makes a target/flower type of brush painting pattern.

---

**Note:** In Clone paint mode, this field changes to indicate the picture image or texture that you are cloning from.

---

**Brush Mapping** Sets how the texture is applied to the brush

**View Plane** In 2D painting, the texture moves with the brush

**Tiled** The texture is offset by the brush location

**3D** Same as tiled mode

**Stencil** Texture is applied only in borders of the stencil.

**Random** Random applying of texture.

**Angle** This is the rotation angle of the texture brush. It can be changed interactively via `Ctrl-F` in the 3D view. While in the interactive rotation you can enter a value numerically as well. Can be set to:

**User** Directly input the angle value.

**Rake** Angle follows the direction of the brush stroke. Not available with 3D textures.

**Random** Angle is randomized.

**Offset** Offset the texture in x, y, and z.

**Size** Set the scale of the texture in each axis.

## Stroke

**Stroke Method** Allows set the way applying strokes.

**Airbrush** Flow of the brush continues as long as the mouse click is held, determined by the *Rate* setting. If disabled, the brush only modifies the color when the brush changes its location.

**Rate** Interval between paints for airbrush

**Space** Creates brush stroke as a series of dots, whose spacing is determined by the *Spacing* setting.

**Spacing** Represents the percentage of the brush diameter. Limit brush application to the distance specified by spacing.

**Dots** Apply paint on each mouse move step

**Jitter** Jitter the position of the brush while painting

**Smooth stroke** Brush lags behind mouse and follows a smoother path. When enabled, the following become active:

**Radius** Sets the minimum distance from the last point before stroke continues.

**Factor** Sets the amount of smoothing.

**Input Samples** Average multiple input samples together to smooth the brush stroke.

**Wrap** wraps your paint to the other side of the image as your brush moves off the OTHER side of the canvas (any side, top/bottom, left/right). Very handy for making seamless textures.

**Curve** The paint curve allows you to control the falloff of the brush. Changing the shape of the curve will make the brush softer or harder.

**See also:**

- Read more about using the *Curve Widget*.

### Paint options

**Overlay** Allows you to customize the display of curve and texture that applied to the brush.

**Appearance** Allows you to customize the color of the brush radius outline, as well as specify a custom icon.

**Saving** If the header menu item Image has an asterisk next to it, it means that the image has been changed, but not saved. Use the *Image*→*Save Image* option to save your work with a different name or overwrite the original image.

---

**Note:** UV Textures

Since images used as UV Textures are functionally different from other images, you should keep them in a directory separate from other images.

---

The image format for saving is independent of the format for rendering. The format for saving a UV image is selected in the header of the Save Image window, and defaults to PNG (.png).

If Packing is enabled in the window header, or if you manually *Image*→*Pack Image*, saving your images to a separate file is not necessary.

**Using an External Image Editor** If you use an external program to edit your UV Texture, you must:

- run that paint program (GIMP, Photoshop, etc.)
- load the image or create a new one
- change the image, and
- re-save it within that program.
- Back in Blender, you reload the image in the UV/Image Editor window.

You want to use an external program if you have teams of people using different programs that are developing the UV textures, or if you want to apply any special effects that Texture Paint does not feature, or if you are much more familiar with your favorite paint program.

### Projection Texture Painting

Projection texture painting allows an artist to paint on texture mapped on a 3D model. Unlike painting in the image editor, projection texture painting is done in the 3D viewport of blender.

**Getting Started** To enter texture paint mode, you need to select a mesh object and select *Texture Paint* from the mode menu (the one which toggles between *Object*, *Edit* etc. modes).

Painting on a 3D model requires some setup before being possible. Blender needs a way to map an image to the 3D model. This is accomplished by using a UV map (see [UV Mapping](#) for more details), so if the model hasn't been unwrapped yet, it should be unwrapped prior to entering *Texture Paint* mode. The image assigned to the UV layer is also used for painting. That means that the user should either:

- unwrap the model while the target image is being displayed in the image editor window, or
- unwrap, and while still in edit mode, change the image in the UV editor window to the target image.

If the target image is not square, the first method is preferable, so that unwrapping accounts for the aspect ratio of the image.

**Hints** There are a known limitations in painting...

- Overlapping UVs are not supported (as with texture baking).
- When painting onto a face which is partially behind the view (in perspective mode), the face can't be painted on. To avoid, this zoom out or use an Ortho mode viewport.
- When painting onto a face in perspective mode onto a low poly object with normals pointing away from the view, painting may fail; to workaround disable the **Normal** option in the paint panel.

*Typically this happens when painting onto the side of a cube (see [Bug report T34665](#)).*

## Texture Mapping

Textures need mapping coordinates, to determine how they are applied to the object. The mapping specifies how the texture will ultimately wrap itself to the object.

For example, a 2D image texture could be configured to wrap itself around a cylindrical shaped object.



Fig. 2.1968: Mapping Coordinate menu

### Coordinates

**Coordinates** Mapping works by using a set of coordinates to guide the mapping process. These coordinates can come from anywhere, usually the object to which the texture is being applied to.

**Global** The scene's global 3D coordinates. This is also useful for animations; if you move the object, the texture moves across it. It can be useful for letting objects appear or disappear at a certain position in space.

### Object

Uses an object as source for coordinates. Often used with an *Empty*, this is an easy way to place a small image at a given point on the object. This object can also be animated, to move a texture around or through a surface.

**Object** Select the name of an object.

**Generated** The original undeformed coordinates of the object. This is the default option for mapping textures.

### UV

UV mapping is a very precise way of mapping a 2D texture to a 3D surface. Each vertex of a mesh has its own UV co-ordinates which can be unwrapped and laid flat like a skin. You can almost think of UV coordinates as a mapping that works on a 2D plane with its own local coordinate system to the plane on which it is operating on. This mapping is especially useful when using 2D images as textures, as seen in [UV Mapping](#). You can use multiple textures with one set of UV coordinates.

**Layer** Select your UV layer to use it for mapping.

**Strand/Particle** Uses normalized 1D strand texture coordinate or particle age(X) and trail position (Y). Use when texture is applied to hair strands or particles.

**Sticky** Uses a mesh's sticky coordinates, which are a form of per-vertex UV co-ordinates. If you have made sticky coordinates first (in (usually) Camera View' → Spacebar → type *Sticky* → choose *Add Sticky / Remove Sticky*) the texture can be rendered in camera view (so called Camera Mapping).

**Window** The rendered image window coordinates. This is well suited to blending two objects.

**Normal** Uses the direction of the surface's normal vector as coordinates. This is very useful when creating certain special effects that depend on viewing angle.

**Reflection** Uses the direction of the reflection vector as coordinates. This is useful for adding reflection maps - you will need this input when Environment Mapping.

**Stress** Uses the difference of edge length compared to original coordinates of the mesh. This is useful, for example, when a mesh is deformed by modifiers.

**Tangent** Uses the optional tangent vector as texture coordinates.

### Projection

**Flat** Flat mapping gives the best results on single planar faces. It does produce interesting effects on the sphere, but compared to a sphere-mapped sphere the result looks flat. On faces that are not in the mapping plane the last pixel of the texture is extended, which produces stripes on the cube and cylinder.

**Cube** Cube mapping often gives the most useful results when the objects are not too curvy and organic (notice the seams on the sphere).

**Tube** Tube mapping maps the texture around an object like a label on a bottle. The texture is therefore more stretched on the cylinder. This mapping is of course very good for making the label on a bottle or assigning stickers to rounded objects. However, this is not a cylindrical mapping so the ends of the cylinder are undefined.

**Sphere** Sphere mapping is the best type for mapping a sphere, and it is perfect for making planets and similar objects. It is often very useful for creating organic objects. It also produces interesting effects on a cylinder.

### Inheriting coordinates from the parent object

**From Dupli** Duplis instanced from vertices, faces, or particles, inherit texture coordinates from their parent.

**Todo: explanation**

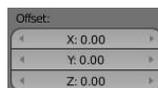


Fig. 2.1969: Offset panel

### Coordinate Offset, Scaling and Transformation

**Offset** The texture co-ordinates can be translated by an offset. Enlarging of the Ofs moves the texture towards the top left.

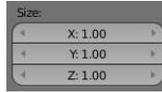


Fig. 2.1970: Size panel

**Size** These buttons allow you to change the mapping of axes between the texture’s own coordinate system, and the mapping system you choose (Generated, UV, etcetera.) More precisely, to each axis of the texture corresponds one of four choices, that allow you to select to which axis in the mapping system it maps! This implies several points:

- For 2D textures (such as images), only the first two rows are relevant, as they have no Z data.
- You can rotate a 2D picture a quarter turn by setting the first row (i.e. X texture axis) to Y, and the second row (Y texture axis) to X.
- When you map no texture axis (i.e. the three “void” buttons are set), you’ll get a solid uniform texture, as you use zero dimension (i.e. a dot, or pixel) of it (and then Blender extends or repeats this point’s color along all axes.)
- When you only map one texture axis (i.e. two “void” buttons are enabled) you’ll get a “striped” texture, as you only use one dimension (i.e. a line of pixel) of it, (and then Blender stretches this line along the two other axes).
- The same goes, for 3D textures (i.e. procedural ones), when one axis is mapped to nothing, Blender extends the plan (“slice”) along the relevant third axis.

So, all this is a bit hard to understand and master. Fortunately, you do not have to change these settings often, except for some special effects... Anyway, the only way to get used to them is to practice!

## Environment Maps

Environment maps take a render of the 3D scene and apply it to a texture, to use for faking reflections. If you want to achieve a very realistic result, raytraced reflections are a good solution. Environment Maps are another way to create reflective surfaces, but they are not so simple to set up.

So why should one use Environment Maps?

- The main reason is probably that they can be much faster than raytracing reflections. In certain situations they need to be calculated only once, and may be reused like any ordinary texture. You may even modify the precalculated Environment Map in an image editor.
- Environment maps can also be blurred and render even faster because the resolution can then be lowered. Blurring a reflection with the raytracer always adds to the render time, sometimes quite a lot.
- **Halos** (a visualization type for particles) are not visible to raytraced reflections, so you need to setup environment maps to reflect them.
- **Keypoint strands** (another visualization type for particles) are also not visible to raytraced reflections, so you need to setup environment maps to reflect them.

Just as we render the light that reaches the viewing plane using the camera to define a viewpoint, we can render the light that reaches the surface of an object (and hence, the light that might ultimately be reflected to the camera). Blender’s environment mapping renders a cubic image map of the scene in the six cardinal directions from any point. When the six tiles of the image are mapped onto an object using the *Refl* input coordinates, they create the visual complexity that the eye expects to see from shiny reflections.

---

**Note:** It’s useful to remember here that the true goal of this technique is *believability*, not *accuracy*. The eye doesn’t need a physically accurate simulation of the light’s travel; it just needs to be lulled into believing that the scene is real by seeing the complexity it expects. The most unbelievable thing about most rendered images is the sterility, not the inaccuracy.

---

## Options

**Important:** For correct results, the mapping of an environment map texture must be set to 'Refl' (reflection co-ordinates) in the Map Input panel of the Material context.

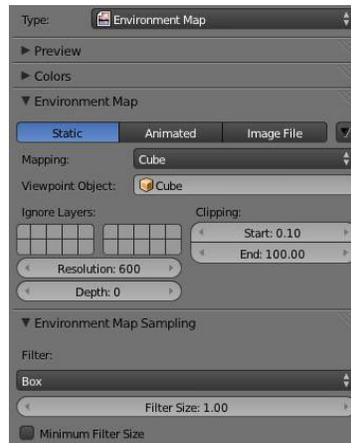


Fig. 2.1971: Reflecting plane EnvMap settings.

Blender allows three types of environment maps, as you can see in *Reflecting plane EnvMap settings*. :

**Static** The map is only calculated once during an animation or after loading a file.

**Animated** The map is calculated each time a rendering takes place. This means moving Objects are displayed correctly in mirroring surfaces.

**Image File** When saved as an image file, environment maps can be loaded from disk. This option allows the fastest rendering with environment maps, and also gives the ability to modify or use the environment map in an external application.

When using planar reflections, if the camera is the only moving object and you have a reflecting plane, the Empty must move too and you must use *Anim* environment map. If the reflecting object is small and the Empty is in its center, the environment map can be *Static*, even if the object itself rotates since the Empty does not move. If, on the other hand, the Object translates the Empty should follow it and the environment map be of *Anim* type.

Options in dropdown menu:

**Clear Environment Map** Clears the currently rendered environment map from memory. This is useful to refresh a *Static* environment maps and you have changed things in your scene since the last time the environment map was rendered. *Anim* environment maps do this automatically on every render.

**Save Environment Map** Saves the currently stored static environment map to disk as an image file. This can be loaded again with *Load*.

**Clear All Environment Maps** Does the same as *Free Data*, but with all environment maps in the scene. This is a useful shortcut when using recursive environment maps (when the *Depth* is greater than 0).

**Note:** EnvMap calculation can be disabled at a global level by the EnvMap Tog Button in the Render Panel of the Rendering Buttons.

**Viewpoint Object** Environment maps are created from the perspective of a specified object. The location of this object will determine how 'correct' the reflection looks, though different locations are needed for different reflecting surfaces. Usually, an Empty is used as this object.

- For planar reflections, the object should be in a location mirrored from the camera, on the other side of the plane of reflection (see Examples). This is the most accurate usage of Environment maps.

- For spherical reflections, the object should be in the center of the sphere. Generally, if the reflecting sphere's object center point is in the center of its vertices, you can just use the name of the actual sphere object as the *Ob*:
- For irregular reflections, there's no hard and fast rule, you will probably need to experiment and hope that the inaccuracy doesn't matter.

**Ignore Layers** The layers to exclude from the environment map creation. Since environment maps work by rendering the scene from the location of the *Ob*: object, you will need to exclude the actual reflecting surface from the environment map, otherwise it will occlude other objects that should be reflected on the surface itself.

Eg. If you are rendering an environment map from the center of a sphere, all the environment map will show by default is the inside of the sphere. You will need to move the sphere to a separate layer, then exclude that layer from the environment map render, so that the environment map will show (and hence reflect) all the objects outside the sphere.

**Resolution** The resolution of the cubic environment map render. Higher resolutions will give a sharper texture (reflection), but will be slower to render.

**Depth** The number of recursive environment map renders. If there are multiple reflecting objects using environment maps in the scene, some may appear solid, as they won't render each other's reflections. In order to show reflections within reflections, the environment maps need to be made multiple times, recursively, so that the effects of one environment map can be seen in another environment map. See Examples.

**Clipping Start/End** The clipping boundaries of the virtual camera when rendering the environment map. Sets the minimum and maximum distance from the camera that will be visible in the map.

## Environment Map Sampling

### Filter

**Box** Box Filter

**EWA** Elliptical Weighted Average - one of the most efficient direct convolution algorithms developed by Paul Heckbert and Ned Greene in the 1980s. For each texel, EWA samples, weights, and accumulates texels within an elliptical footprint and then divides the result by the sum of the weights.

**Eccentricity** Maximum eccentricity (higher gives less blur at distant/oblique angles, but is also slower)

**FELINE** FELINE (Fast Elliptical Lines), uses several isotropic probes at several points along a line in texture space to produce an anisotropic filter to reduce aliasing artifacts without considerably increasing rendering time.

**Probes** Maximum number of samples (higher gives less blur at distant/oblique angles, but is also slower)

Area

**Eccentricity** Maximum eccentricity (higher gives less blur at distant/oblique angles, but is also slower)

**Filter Size** The amount of blurring applied to the texture. Higher values will blur the environment map to fake blurry reflections.

**Minimum Filter Size** Use Filter Size as a minimal filter value in pixels

**Examples** In this example, an empty is used as the *Ob*: of the reflecting plane's environment map. It is located in the specular position of the camera with respect to the reflecting surface. (This is possible, strictly speaking, only for planar reflecting surfaces.) Ideally, the location of the empty would mirror the location of the camera across the plane of the polygon onto which it is being mapped.



The following images show the effect of the *Depth*. The first render has depth set to 0. This means the environment map on the plane has rendered before the environment map of the sphere, so the sphere's reflection isn't shown. By raising the *Depth*, the environment map is rendered recursively, in order to get reflections of reflections.



**Limitations** Because environment maps are calculated from the exact location of the *Viewpoint Object's* object center, and not from actual reflecting surface, they can often be inaccurate, especially with spheres. In the following image, the rectangular prism and the smaller spheres are touching the sides of the large reflecting sphere, but because the environment map is calculated from the center of the sphere, the surrounding objects look artificially far away.

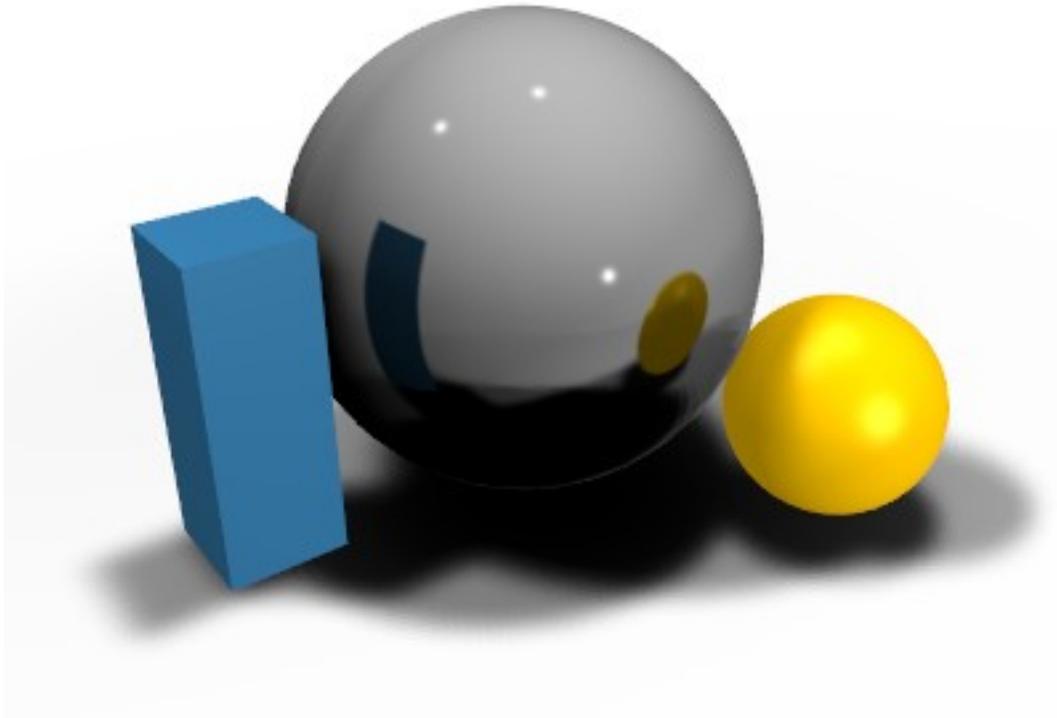


Fig. 2.1980: Inaccurate spherical reflection, the colored objects are artificially offset

### UV Mapping

The most flexible way of mapping a 2D texture over a 3D object is a process called “UV mapping”. In this process, you take your three-dimensional (X,Y & Z) mesh and unwrap it to a flat two-dimensional (X & Y ... or rather, as we shall soon see, “U & V”) image. Colors in the image are thus mapped to your mesh, and show up as the color of the faces of the mesh. Use UV texturing to provide realism to your objects that procedural materials and textures cannot do, and better details than Vertex Painting can provide.

**UVs Explained** The best analogy to understanding UV mapping is cutting up a cardboard box. The box is a three-dimensional (3D) object, just like the mesh cube you add to your scene.

If you were to take a pair of scissors and cut a seam or fold of the box, you would be able to lay it flat on a tabletop. As you are looking down at the box on the table, we could say that U is the left-right direction, is V is the up-down direction. This image is thus in two dimensions (2D). We use U and V to refer to these “texture-space coordinates” instead of the normal X and Y, which are always used (along with Z) to refer to “3D space.”



Fig. 2.1981: Box being inspected



Fig. 2.1982: Box mapped flat

When the box is reassembled, a certain UV location on the paper is transferred to an (X,Y,Z) location on the box. This is what the computer does with a 2D image in wrapping it around a 3D object.

During the UV unwrapping process, you tell Blender exactly how to map the faces of your object (in this case, a box) to a flat image in the UV/Image Editor window. You have complete freedom in how to do this. (Continuing our previous example, imagine that, having initially laid the box flat on the tabletop, you now cut it into smaller pieces, somehow stretch and/or shrink those pieces, and then arrange them in some way upon a photograph that's also lying on that tabletop ...)

**Cartography Example** Cartographers (map makers) have been dealing with this problem for millennia. A cartography (map-making) example is creating a projection map of the whole world. In cartography, we take the surface of the earth (a sphere) and make a flat map that can be folded up into the glove compartment aboard the space shuttle. We 'fill in' spaces toward the poles, or change the outline of the map in any of several ways:



Each of these is an example of a way to UV map a sphere. Each of the hundred or so commonly accepted projections has its advantages and disadvantages. Blender allows us to do the same thing any way we want to, on the computer.

On more complex models (like seen in the earth map above) there pops up an issue where the faces can't be 'cut', but instead they are stretched in order to make them flat. This helps making easier UV maps, but sometimes adds distortion to the final mapped texture. (Countries and states that are closer to the North or the South Pole look smaller on a flat map than do ones which are close to the Equator.)

**Half-Sphere Example** In this image you can easily see that the shape and size of the marked face in 3D space is different in UV space.

This difference is caused by the 'stretching' (technically called mapping) of the 3D part (XYZ) onto a 2D plane (i.e the UV map).

If a 3D object has a UV map, then, in addition to the 3D-coordinates X, Y, and Z, each point on the object will have corresponding U and V coordinates. (*P* in the image above is an example of how a point on a 3D object might be mapped onto a 2D image.)

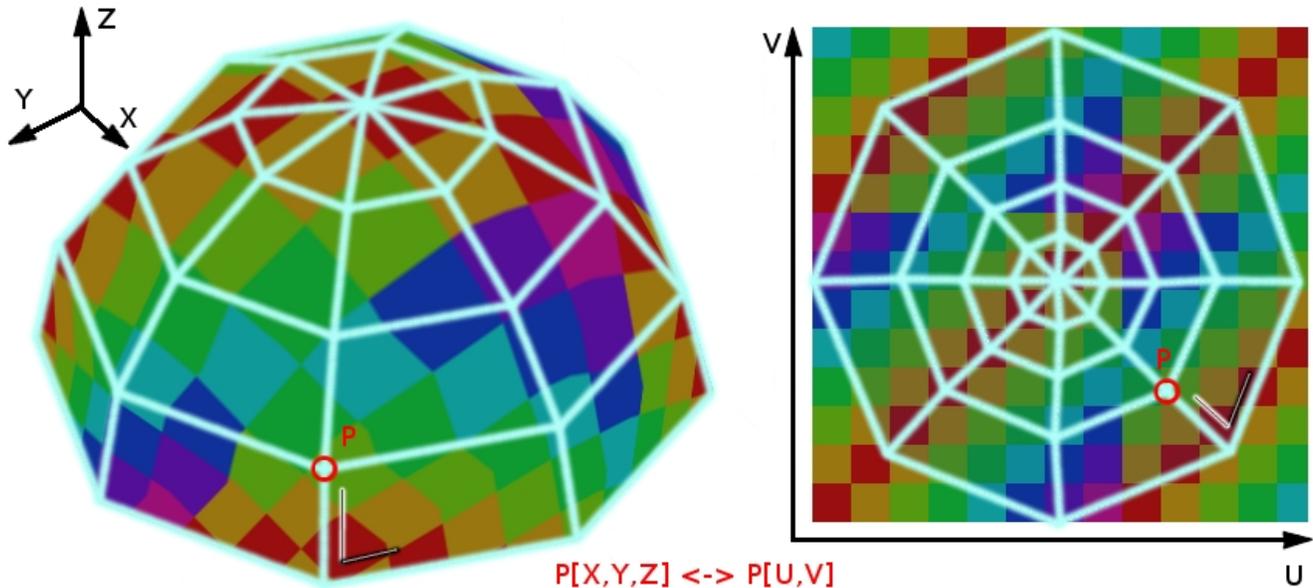


Fig. 2.1989: 3D Space (XYZ) versus UV Space (click to enlarge)

**The UV Editor** About functionalities for mapping UV see [UV/Image Editor](#) section for details.

**Advantages of UVs** While procedural textures (described in the previous chapters) are useful—they never repeat themselves and always “fit” 3D objects—they are not sufficient for more complex or natural objects. For instance, the skin on a human head will never look quite right when procedurally generated. Wrinkles on a human head, or scratches on a car do not occur in random places, but depend on the shape of the model and its usage. Manually-painted images, or images captured from the real world gives more control and realism. For details such as book covers, tapestry, rugs, stains, and detailed props, artists are able to control every pixel on the surface using a UV Texture.

A UV map describes what part of the texture should be attached to each polygon in the model. Each polygon’s vertex gets assigned to 2D coordinates that define which part of the image gets mapped. These 2D coordinates are called UVs (compare this to the XYZ coordinates in 3D). The operation of generating these UV maps is also called “unwrap”, since it is as if the mesh were unfolded onto a 2D plane.

For most simple 3D models, Blender has an automatic set of unwrapping algorithms that you can easily apply. For more complex 3D models, regular Cubic, Cylindrical or Spherical mapping, is usually not sufficient. For even and accurate projection, use seams to guide the UV mapping. This can be used to apply textures to arbitrary and complex shapes, like human heads or animals. Often these textures are painted images, created in applications like the Gimp, Photoshop, or your favorite painting application.

---

**Note:** Games

UV mapping is also essential in the Blender game engine, or any other game. It is the de facto standard for applying textures to models; almost any model you find in a game is UV mapped.

---

### UV Image Editor

```
FIXME(Template Unsupported: Doc:2.6/Reference/Textures/UV_Image_Editor; { {Doc:2.6/Reference/Textures/UV_Image_Editor} }
)
```

## UV Mapping a Mesh

The first step is to unwrap your mesh. You want to unwrap when you feel your mesh is complete with respect to the number of faces it needs to have. If you do add faces or subdivide existing faces when a model is already unwrapped, Blender will add those new faces for you, but you may need to do additional mapping or editing. In this fashion, you can use the UV Texture image to guide additional geometry changes.

This section covers techniques for Mapping Uvs. The next sections cover [Editing UVs](#), followed by methods of [Managing UV Layouts](#), and [Applying Images to UVs](#).

**About UVs** Every point in the UV map corresponds to a vertex in the mesh. The lines joining the UVs correspond to edges in the mesh. Each face in the UV map corresponds to a mesh face.

Each face of a mesh can have many UV Textures. Each UV Texture can have an individual image assigned to it. When you unwrap a face to a UV Texture in the UV/Image Editor, each face of the mesh is automatically assigned *four UV coordinates*: These coordinates define the way an image or a texture is mapped onto the face. These are 2D coordinates, which is why they're called UV, to distinguish them from XYZ coordinates. These coordinates can be used for rendering or for real-time OpenGL display as well.

Every face in Blender can have a link to a different image. The UV coordinates define how this image is mapped onto the face. This image then can be rendered or displayed in real time. A 3D window has to be in "Face Select" mode to be able to assign Images or change UV coordinates of the active Mesh Object. This allows a face to participate in many UV Textures. A face at the hairline of a character might participate in the facial UV Texture, *and* in the scalp/hair UV Texture.

These are described more fully in the next sections.

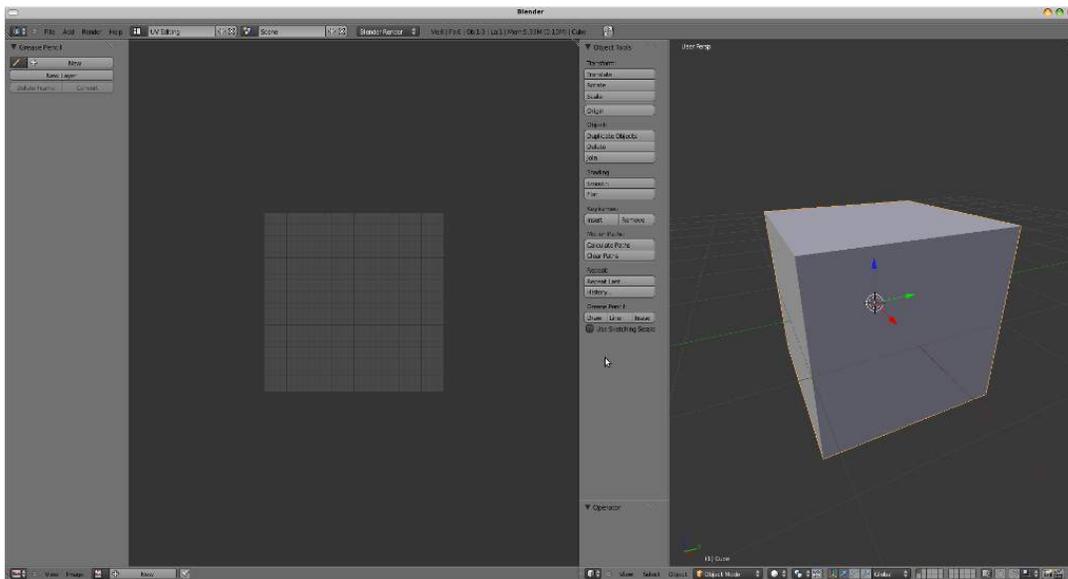


Fig. 2.1990: UV Editing screen layout

**Getting Started** By default, meshes are not created with UVs. First you must map the faces, then you can [edit them](#). The process of unwrapping your model is done within Edit Mode in the 3D View window. This process creates one or more UV Islands in the [UV/Image Editor window](#).

To begin, choose the [UV Editing screen layout](#) from the selection list at the top of your screen in the User Preferences window header. This sets one of the panes to show you the UV/Image Editor window ([Shift-F10](#)), and the other pane to the 3D window ([Shift-F5](#)).

Enter edit mode, as all unwrapping is done in Edit mode. You can be in vertex, face, or edge selection mode.

**Workflow** The process for unwrapping is straightforward, but there are tons of options available, each of which dramatically affect the outcome of the unwrap. By understanding the meaning behind the options, you will become more efficient at unwrapping. The process is:

- Mark Seams if necessary
- Select all of the mesh components
- Select a UV mapping method from the UV Unwrap menu
- Adjust the unwrap settings
- Add a test image to see if there will be any distortion. See [Applying Images to UVs](#)
- Adjust UVs in the UV editor. See [Editing UVs](#)

**Mapping Types** Blender offers several ways of mapping UVs. The simpler projection methods use formulas that map 3d space onto 2d space, by interpolating the position of points toward a point/axis/plane through a surface. The more advanced methods can be used with more complex models, and have more specific uses.

Basic:

**Cube** Maps the mesh onto the faces of a cube, which is then unfolded.

**Sphere** Projects the UVs onto a spherical shape. Useful only for spheres or spherical shapes, like eyes, planets, etc.

**Cylinder** Projects UVs onto a cylindrical surface.

**Project from View** Takes the current view in the 3D viewport and flattens it as it appears.

Advanced:

**Unwrap** Useful for organic shapes. Smooths the mesh into a flat surface by cutting along seams.

**Smart UV Project** Breaks the mesh into islands based on an angle threshold.

**Lightmap Pack** Separates each face and packs them onto the UV grid.

**Follow Active Quads** Follow UV from active quads along continuous face loops.

You can also *reset UVs*, which maps each face to fill the UV grid, giving each face the same mapping.

If we were to use an image that was tileable, the surface would be covered in a smooth repetition of that image, with the image skewed to fit the shape of each individual face. Use this unwrapping option to reset the map and undo any unwrapping (go back to the start).

**Basic Mapping** Based on the fundamental geometry of the object, and how it is being viewed, the *Mesh->UV Unwrap->Cube, Cylinder, and Sphere* UV Calculations attempt to unfold the faces for you as an initial best fit. Here, the view from the 3D window is especially important. Also, the settings for cube size or cylinder radius (Editing buttons, UV Calculation panel) should be set (in Blender units) to encompass the object.

The following settings are common for the Cube, Cylinder, and Sphere mappings:

**Correct Aspect** Map UVs taking image aspect ratios into consideration. If an image has already been mapped to the texture space that is non-square, the projection will take this into account and distort the mapping to appear correct.

**Clip to Bounds** Any UVs that lie outside the 0 to 1 range will be clipped to that range by being moved to the UV space border it is closest to.

**Scale to Bounds** If the UV map is larger than the 0 to 1 range, the entire map will be scaled to fit inside.

**Cube** Cube mapping projects a mesh onto six separate planes, creating 6 UV islands. In the UV editor, these will appear overlapped, but can be moved. See [Editing UVs](#).

**Cube Size** Set the size of the cube to be projected onto.

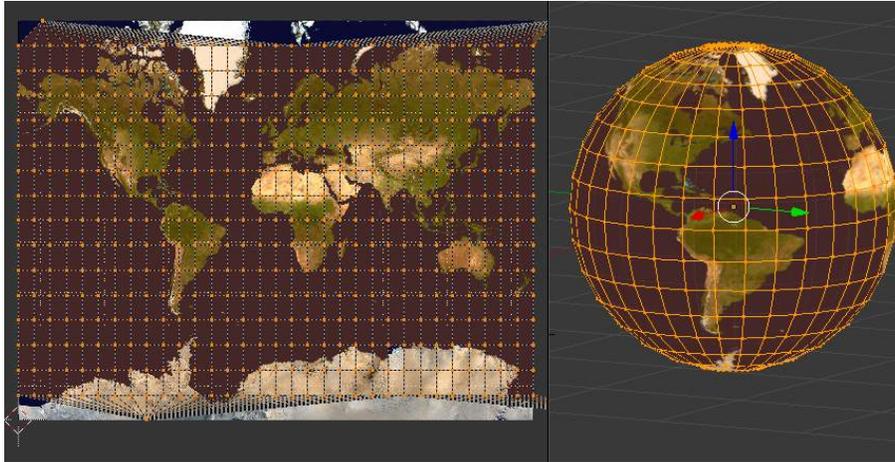


Fig. 2.1991: Using a Mercator image with a Sphere Projection

**Cylinder and Sphere** Cylindrical and Spherical mappings have the same settings. The difference is that a cylindrical mapping projects the UVs on a plane toward the cylinder shape, while a spherical map takes into account the sphere's curvature, and each latitude line becomes evenly spaced.

Normally, to unwrap a cylinder (tube) as if you slit it lengthwise and folded it flat, Blender wants the view to be vertical, with the tube standing 'up'. Different views will project the tube onto the UV map differently, skewing the image if used. However you can set the axis on which the calculation is done manually. This same idea works for the sphere mapping:

Recall the opening cartographer's approaching to mapping the world? Well, you can achieve the same here when unwrapping a sphere from different perspectives. Normally, to unwrap a sphere, view the sphere with the poles at the top and bottom. After unwrapping, Blender will give you a Mercator projection; the point at the equator facing you will be in the middle of the image. A polar view will give a very different but common projection map. Using a Mercator projection map of the earth as the UV image will give a very nice planet mapping onto the sphere.

#### Direction

**View on Poles** Use when viewing from the top (at a pole) by using an axis that is straight down from the view

**View on Equator** Use if view is looking at the equator, by using a vertical axis

**Align to Object** Uses the object's transform to calculate the axis

**Align** Select which axis is up

**Polar ZX** Polar 0 is on the x axis

**Polar ZY** Polar 0 is on the y axis

**Radius** The radius of the cylinder to use

**Project From View** In the 3D window, the *Face* → *Unwrap UVs* → *Project from View* option maps the face as seen through the view of the 3D window it was selected from. It is almost like you had x-ray vision or squashed the mesh flat as a pancake onto the UV map. Use this option if you are using a picture of a real object as a UV Texture for an object that you have modeled. You will get some stretching in areas where the model recedes away from you.

Using *Project from View (Bounds)* will do the same as above, but scales the UVs to the bounds of the UV space.

**Resetting UVs** In the 3D window, *Face*→*Unwrap*→*Reset* maps each selected face to the same area of the image, as previously discussed. To map all the faces of an object (a cube, for example) to the same image, select all the faces of the cube, and unwrap them using the Reset menu option.

### Advanced Mapping

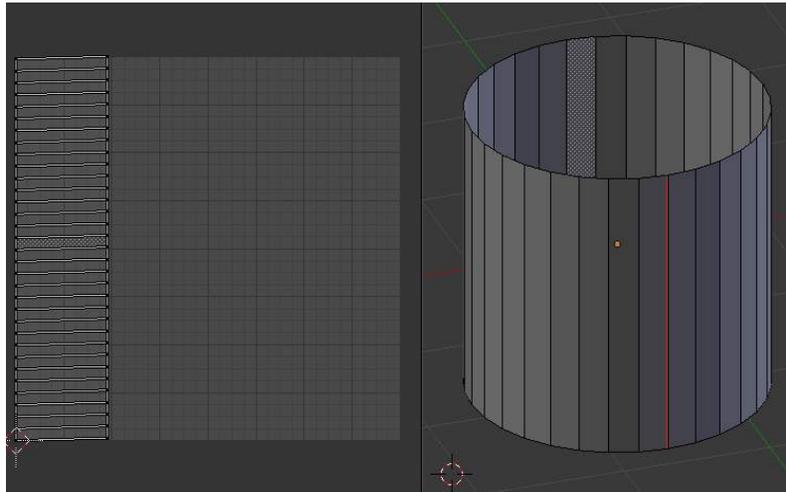


Fig. 2.1992: Simple Seam on a Cylinder

**Unwrapping Using Seams** For many cases, using the Unwrap calculations of Cube, Cylinder, Sphere, or best fit will produce a good UV layout. However, for more complex meshes, especially those with lots of indentations, you may want to define a **seam** to limit and guide any of the unwrapping processes discussed above.

Just like in sewing, a seam is where the ends of the image/cloth are sewn together. In unwrapping, the mesh is unwrapped at the seams. Think of this method as peeling an orange or skinning an animal. You make a series of cuts in the skin, then peel it off. You could then flatten it out, applying some amount of stretching. These cuts are the same as seams.

When using this method, you need to be aware of how much stretching there is. The more seams there are, the less stretching there is, but this is often an issue for the texturing process. It's a good idea to have as few seams as possible while having the least amount of stretching. Try to hide seams where they will not be seen. In productions where 3d Paint is used, this becomes less of an issue, as projection painting can easily deal with seams, as opposed to 2d texturing, where it is difficult to match the edges of different UV islands.

The workflow is the following:

- Create seams. A seam is marked in Edit mode by selecting edges to make the seam and then issuing the command to Mark Seam.
- Unwrap
- Adjust seams and repeat
- Manually adjust UVs. See the next section on Editing UVs.

**Marking Seams** To add an edge to a seam, simply select the edge and `Ctrl-E` *Mark Seam*. To take an edge out of a seam, select it, `Ctrl-E` and *Clear Seam*.

In the example to the right, the back-most edge of the cylinder was selected as the seam (to hide the seam), and the default unwrap calculation was used. In the UV/Image Editor window, you can see that all the faces are nicely unwrapped, just as if you cut the seam with a scissors and spread out the fabric.

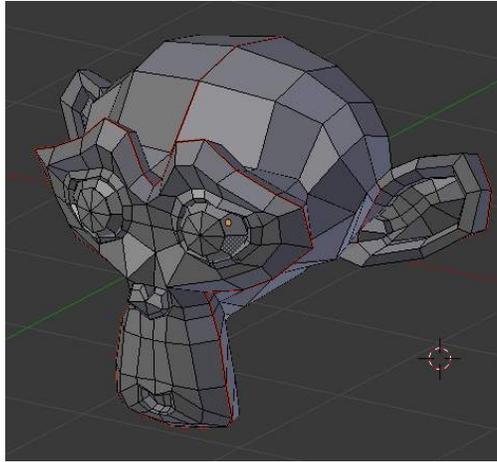


Fig. 2.1993: Seamed Suzanne

When marking seams, you can use the *Select*→*Linked Faces* or `Ctrl-L` in Face Select Mode to check your work. This menu option selects all faces connected to the selected one, up to a seam. If faces outside your intended seam are selected, you know that your seam is not continuous. You do not need continuous seams, however, as long as they resolve regions that may stretch.

Just as there are many ways to skin a cat, there are many ways to go about deciding where seams should go. In general though, you should think as if you were holding the object in one hand, and a pair of sharp scissors in the other, and you want to cut it apart and spread it on the table with as little tearing as possible. Note that we seamed the outside edges of her ears, to separate the front from the back. Her eyes are disconnected sub-meshes, so they are automatically unwrapped by themselves. A seam runs along the back of her head vertically, so that each side of her head is flattened out.

Another use for seams is to limit the faces unwrapped. For example, when texturing a head, you don't really need to texture the scalp on the top and back of the head since it will be covered in hair. So define a seam at the hairline. Then, when you select a frontal face, and then select linked faces before unwrapping, the select will only go up to the hairline seam, and the scalp will not be unwrapped.

When unwrapping anything that is bilateral, like a head or a body, seam it along the mirror axis. For example, cleave a head or a whole body right down the middle in front view. When you unwrap, you will be able to overlay both halves onto the same texture space, so that the image pixels for the right hand will be shared with the left; the right side of the face will match the left, etc.

Finally, remember that you *don't* have to come up with “one unwrapping that works perfectly for everything everywhere.” As we'll discuss later, you can easily have multiple UV unwrappings, using different approaches in different areas of your mesh.

**Unwrap** Begin by selecting all faces to be unwrapped in the 3D View. With our faces selected, it is now time to unwrap them. In the 3D View, select *Mesh* → *UV Unwrap* → *Unwrap* or `U` and select Unwrap.

You can also do this from the UV/Image Editor window with command *UVs* → *Unwrap* or `E`. This method will unwrap all of the faces and reset previous work. The *UVs* menu will appear in the UV/Image Editor window after unwrapping has been performed once.

This tool unwraps the faces of the object to provide the ‘best fit’ scenario based on how the faces are connected and will fit within the image, and takes into account any seams within the selected faces. If possible, each selected face gets its own different area of the image and is not overlapping any other faces UV's. If all faces of an object are selected, then each face is mapped to some portion of the image.

Blender has two ways of calculating the unwrapping. They can be selected in the tool setting in the tool panel in the 3D View.

**Angle Based** This method gives a good 2d representation of a mesh.

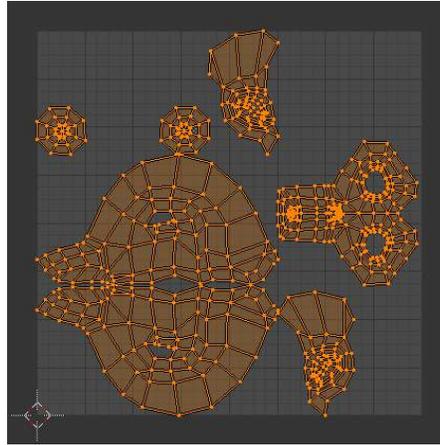


Fig. 2.1994: Result of unwrapping Suzanne

**Conformal** Uses LSCM (Least Squared Conformal Mapping). This usually gives a less accurate UV mapping than Angle Based, but works better for simpler objects.

**Fill Holes** Activating Fill Holes will prevent overlapping from occurring and better represent any holes in the UV regions.

**Correct Aspect** Map UVs taking image aspect into account

**Use Subsurf Modifier** Map UVs taking vertex position after subsurf modifier into account

**Margin** Space between UV islands

**This point is crucial to understanding mapping** later on: a face's UV image texture only has to use *part* of the image, not the *whole* image. Also, portions of the same image can be shared by multiple faces. A face can be mapped to less and less of the total image.

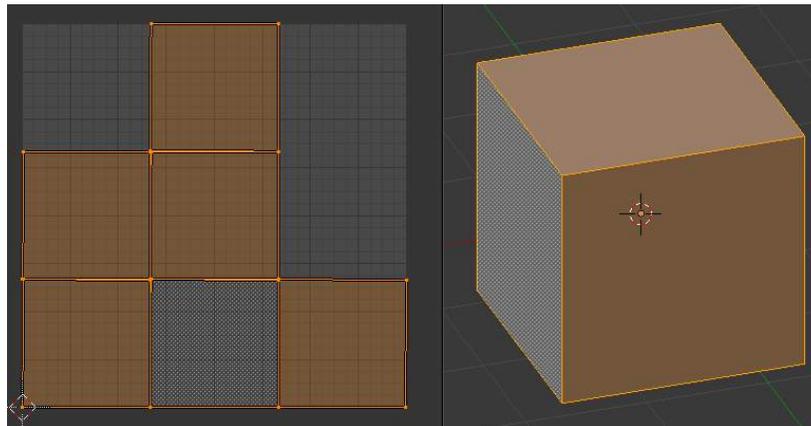


Fig. 2.1995: Smart UV project on a cube

**Smart UV Project** Smart UV Project, (previously called the Archimapper) gives you fine control over how automatic seams should be created, based on angular changes in your mesh. This method is good for simple and complex geometric forms, such as mechanical objects or architecture.

This function examines the shape of your object, the faces selected and their relation to one another, and creates a UV map based on this information and settings that you supply.

In the example to the right, the Smart Mapper mapped all of the faces of a cube to a neat arrangement of 3 sides on top, 3 sides on the bottom, for all six sides of the cube to fit squarely, just like the faces of the cube.

For more complex mechanical objects, this tool can very quickly and easily create a very logical and straightforward UV layout for you.

The Tool Settings panel in the Tool Shelf allows the fine control over how the mesh is unwrapped:

**Angle Limit** This controls how faces are grouped: a higher limit will lead to many small groups but less distortion, while a lower limit will create fewer groups at the expense of more distortion.

**Island Margin** This controls how closely the UV islands are packed together. A higher number will add more space in between islands.

**Area Weight** Weight projection's vector by faces with larger areas

**Lightmap** Lightmap Pack takes each of a mesh's faces, or selected faces, and packs them into the UV bounds. Lightmaps are used primarily in gaming contexts, where lighting information is baked onto texture maps, when it is essential to utilize as much UV space as possible. It can also work on several meshes at once. It has several options that appear in the Tool Shelf:

You can set the tool to map just *Selected Faces* or *All Faces* if working with a single mesh.

The *Selected Mesh Object* option works on multiple meshes. To use this, in *Object Mode* select several mesh objects, then go into *Edit Mode* and activate the tool.

**Share Tex Space** This is useful if mapping more than one mesh. It attempts to fit all of the objects' faces in the UV bounds without overlapping.

**New UV Layer** If mapping multiple meshes, this option creates a new UV layer for each mesh. See [Managing the Layout](#).

**New Image** Assigns new images for every mesh, but only one if *Shared Tex Space* is enabled.

**Image Size** Set the size of the new image.

**Pack Quality** Pre-packing before the more complex Box packing.

**Margin** This controls how closely the UV islands are packed together. A higher number will add more space in between islands.

**Follow Active Quads** The *Face->Unwrap->Follow Active Quads* takes the selected faces and lays them out by following continuous face loops, even if the mesh face is irregularly shaped. Note that it does not respect the image size, so you may have to scale them all down a bit to fit the image area.

Edge Length Mode:

**Even** Space all UVs evenly.

**Length** Average space UV's edge length of each loop.

Please note that it is the shape of the active quad in UV space that is being followed, not its shape in 3d space. To get a clean 90-degree unwrap make sure the active quad is a rectangle in UV space before using "Follow active quad".

## Managing UV Maps

After you finish editing a UV map, you may need to create additional maps on the same object, or transfer a UV map to another mesh.

**Transferring UV Maps** You can copy a UV Map from one mesh to another Mesh provided both meshes have the same geometry/vertex order. This is useful for example when you want to recreate a UV map from an earlier version of your model with intact UVs.

## Workflow

- RMB Select the target mesh (to which you want to copy the UV Map)
- Shift select the source mesh (that contains the intact UV map)
- *Object menu* → *Make Links...* → *Transfer UV Layouts* (Shortcut: `Ctrl-L ...`)

The target Mesh will now have a UV map that matches the original mesh.

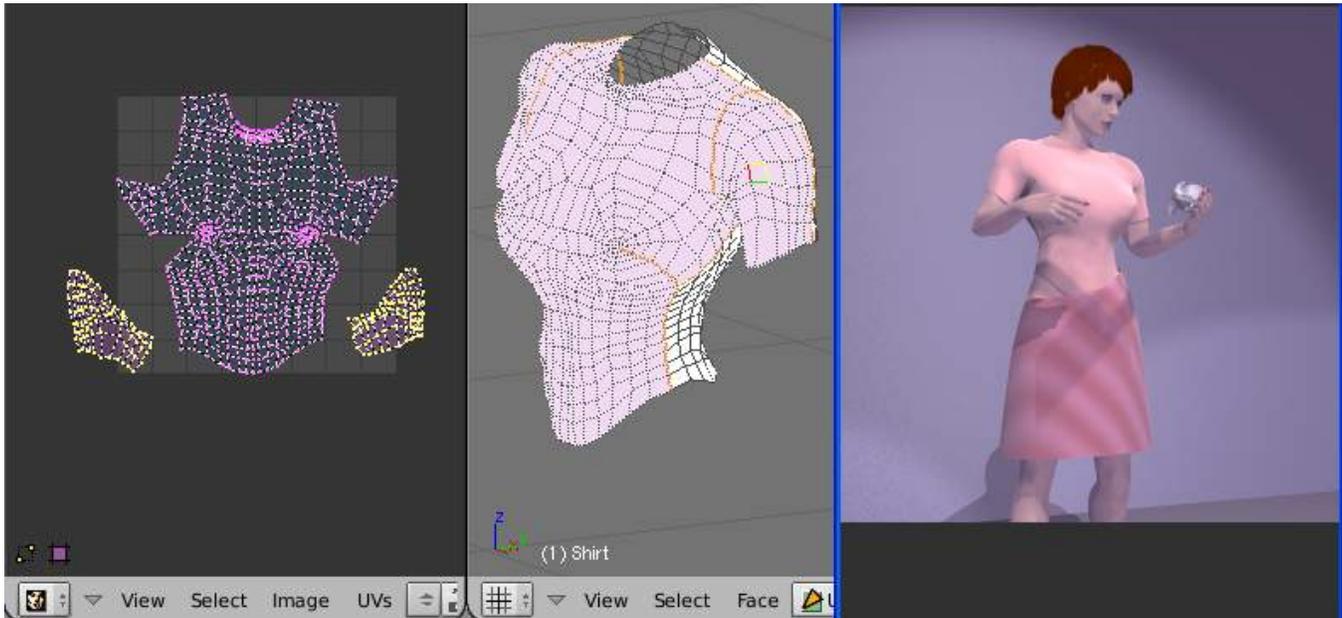


Fig. 2.1996: Mesh with Multiple UV Textures

**Multiple UV Maps** You are not limited to one UV Map per mesh. You can have multiple UV maps for parts of the mesh by creating new UV Textures. The first UV Texture is created for you when you select a face in UV Face Select mode. You can manually create more UV Textures by clicking the *New* button next to “UV Texture” on the Mesh panel in the Buttons Window, Editing Context) and unwrapping a different part of the mesh. Those faces will then go with that UV Texture, while the previously unwrapped faces will still go with the previous UV Texture. Note that if you unwrap the same face twice or more times (each time to a different UV Texture), the coloring for that face will be the alpha combination of the layers of those UV Textures.

In the example to the right, we have a mesh for a blouse. The mesh has been seamed as a normal blouse would, shown in the middle in UV Face Select mode. Wishing to make a cut pattern, the front of the blouse was unwrapped and basic rotation and scaling was done to center it in the UV/Image Editor window. It was then moved off to the side, while the left and right sleeves were unwrapped, each rotated and scaled. Then, select a sample face from each cloth piece, in the 3D View Select→Linked Faces, and the UV/Image Editor will show all those pieces (as shown to the right). You can then work with all pieces for that UV Map. The example shows all three pieces moved onto the image area for painting. As you can see, the pattern nicely fits a square yard of cloth.

Another UV Map was created by clicking the *New* button in the Mesh panel, and the process was repeated for the backs of the sleeves and the back of the blouse. Two images, one for the front and one for the back, are used to color the fabric. In this case, some faces map to the first texture, while other faces map to the second texture.

**UV Textures List** The Mesh panel (shown to the right) lists the UV Texture maps created for this mesh, and allows you to create *New* ones as placeholders for future unwrapping operations.



Click the + button to add a new UV texture, and the - to delete an existing one}}. Deleting a UV Map for the mesh destroys all work done in all unwrapping associated the mesh. Click with care. You've been warned.

Each map has a selector button. Click the camera icon to enable that UV texture for rendering. You can change the name by selecting one and changing the text in the *Name* box. The selected map is displayed in the UV/Image Editor window. The example shows a few UV maps created for a character, and the map for Clothes is selected.

Note that each texture can be mapped to a specific UV texture. See the [Mapping](#) section of the texture panel.

### Editing UVs

After unwrap, you will likely need to arrange the UV maps into something that can be logically textured or painted. Your goals for editing are:

- Stitch some pieces (UV maps) back together
- Minimize wasted space in the image
- Enlarge the 'faces' where you want more detail
- Re-size/enlarge the 'faces' that are stretched
- Shrink the 'faces' that are too grainy and have too much detail

With a minimum of dead space, the most pixels can be dedicated to giving the maximum detail and fineness to the UV Texture. A UV face can be as small as a pixel (the little dots that make up an image) or as large as an entire image. You probably want to make some major adjustments first, and then tweak the layout.

**Selecting UVs** Selection tools are available in the *Select Menu* and Header bar, and the shortcuts listed below:

**Border Select ; B** Use the box lasso to select UV coordinates.

**Select/Deselect All ; A** Selects or de-selects all UV coordinates. When initially unwrapping, you will want to select All UVs to rotate, scale, and move them around.

**Linked UVs:kbd:Ctrl-L** This menu item selects all UVs that are part of the same UV map. Recall that a map is made for every submesh and seamed part of the mesh, and is analogous to a piece of cloth. Selecting *Linked UVs* works similarly to the command in 3D View. It will select all UVs that are 'connected' to currently selected UVs.

**Pinned UVs ; Shift-P** You can pin UVs so they don't move between multiple unwrap operations. This menu item selects them all. See [Pinning](#)

**Border Select Pinned ; Shift-B** Use the box lasso to select only pinned UV coordinates.

**Unlink Selection ; Alt-L** Cuts apart the selected UVs from the map. Only those UVs which belong to fully selected faces remain selected following this command. As the name implies, this is particularly useful to unlink faces and move them elsewhere. The hotkey is analogous to the mesh Separate command.

**Selection Modes** Turning on the *Sync Selection* button in the header causes selection of components in the 3D view to sync with their corresponding elements in the UV editor. This is off by default. These two modes have very different results when transforming components in the UV editor.

When SyncSelection is **Off**: Only selected faces are displayed in the UV editor, and the following selection modes are available:

- *Vertex*

Select individual vertices

- *Edge*

Select edges

- *Face*

Select faces

- *Island*

Select contiguous groups of Faces

The *Sticky Selection Mode* menu is available in this mode. This controls how UVs are selected:

**Shared Vertex** Selects UVs that share a mesh vertex, even if they are in different UV locations.

**Shared Location** Selects UVs that are in the same UV location and share a mesh vertex. This mode is default and works best in most cases.

**Disabled** Disables Sticky Selection. When you move a UV in this mode, each face owns its own UVs, allowing them to be separated.

When *Sync Selection* is **On** the following can be selected:

- *Vertex*

- *Edge*

- *Face*

In this Mode, selection behaves differently. When selecting UVs or Edges, it behave like *Shared Vertex* mode above. When selecting Faces, it behaves as in *Disabled Stick Selection* above.

**Transforming UVs** UVs can be:

- Translated G

- Rotated R

- Scaled S

They can also be hidden or shown using the H and Alt-H respectively, the same way as in Edit Mode.

**Axis Locking** Transformations can be locked to an axis by pressing X or Y after one of the transform tools. Also, holding the MMB will constrain movement to the X or Y axis.

**Pivot Points** The UV editor has a 2D cursor. Its position can be changed by **LMB** clicking in the UV editor. You can also manually adjust its position in the Properties Panel. The range by default is from 0 to 256 starting from the lower left corner. By enabling *Normalized* under *Coordinates*, the range changes from 0 to 1.

The 2D Cursor can be snapped to nearest pixels or to selected elements, by selecting *UVs Menu* under *Snap*.

The Pivot Point can be changed to:

- Bounding Box Center
- Median Point
- 2D Cursor Location

**Proportional Editing** Proportional Editing is available in UV editing. The controls are the same as in the 3D view. See [Proportional Editing in 3D](#) for full reference.

**Snapping** Snapping in UV is also similar to [Snapping in 3D](#), but only snapping to UVs works, however, the *Snap to Pixels* option in the *UVs Menu* will force the UVs to snap to the pixels of an image if loaded.

Additional tools can be found in the *UVs Menu* under the *Snap* Submenu:

**Snap Pixels** Moves selection to nearest pixel

**Snap to Cursor** Moves selection to 2D cursor location

**Snap to Adjacent Unselected** Moves selection to adjacent unselected element

**Weld and Align** the *Weld* tool, **W-1** will move selected UVs to their average position

*Align*, **W-2**, **W-3**, and **W-4** will line up selected UVs on the X axis, Y axis, or automatically chosen axis.

**Mirror** Components can be mirrored on the Y axis or the X axis. You can select *Mirror X* and *Mirror Y* from the *Snap* sub menu in the *UV* menu.

You can also use the hotkey **Ctrl-M** then enter X or Y, or hold the **MMB** and drag in the mirror direction.

**Stitch** *Stitch*, **V**, will join selected UVs that share Vertices. You set the tool to limit stitching by distance in the Tool Settings, by activating *Use Limit* and adjusting the *Limit Distance*

**Minimize Stretch** the *Minimize Stretch* tool, **Ctrl-V** Reduces UV stretch by minimizing angles. This essentially relaxes the UVs

**Face Mirror and Rotate UVs** Recall how the orientation of the UV Texture is relative to each face? Well, you might find that, for example, the image is upside down or laying on its side. If so, use *Face->Rotate UVs* (in the 3D window in Face Select mode) menu to rotate the UVs per face in 90-degree turns.

The *Face->Mirror UVs* to flips the image over like a pancake in a pan, mirroring the UVs per face and showing you the image 'reversed'.

**Pinning** When Unwrapping a model it is sometimes useful to “Lock” certain UVs, so that parts of a UV layout stay the same shape, and/or in the same place.

Pinning is done selecting a UV, then by selecting *Pin* from the *UVs* menu, or the shortcut `P`. You can *Unpin a UV* with the shortcut `Alt-P`

Pinning is most effective when using the Unwrap method of UV mapping, for organic objects. An example is when you are modeling a symmetrical object using the [Mirror Modifier](#). Some of the UVs on the mirror axis may be shared across the mirrored counterparts. You could pin the UVs that correspond to the midline, then align them on the X axis, and they will stay in that location.

Pinning also work great with the Live Unwrap tool. If you pin two or more UVs, with Live Unwrap on, dragging pinned UVs will interactively unwrap the model. This helps with fitting a UV island to a certain shape or region.

**Optimizing the UV Layout** When you have unwrapped, possibly using seams, your UV layout may be quite disorganized and chaotic. You may need to proceed with the following tasks: Orientation of the UV mapping, arranging the UV maps, stitching several maps together.

The next step is to work with the UV layouts that you have created through the unwrap process. If you do add faces or subdivide existing faces when a model is already unwrapped, Blender will add those new faces for you. In this fashion, you can use the UV Texture image to guide additional geometry changes.

When arranging, keep in mind that the entire window is your workspace, but only the UV coordinates within the grid are mapped to the image. So, you can put pieces off to the side while you arrange them. Also, each UV unwrap is its own linked set of coordinates.

You can lay them on top of one another, and they will onionskin (the bottom one will show through the top one). To grab only one though, `RMB` select one of the UV coordinates, and use *Select* → *Linked UVs* (`Ctrl-L`) to select connected UVs, not border select because UVs from both will be selected.

**Combining UV Maps** Very often you will unwrap an object, such as the face example we have been using, and get it ‘mostly right’ but with parts of the mesh that did not unwrap properly, or are horribly confusing. The picture to the right shows an initial unwrap of the face using the Unwrap from sphere option. The issues are with the ear; it is just a mush of UVs, and the neck, it is stretched and folded under. Too much work to clean up.

We can tell that the ear would unwrap nicely with just a straightforward projection from the side view, and the neck with a tubular unwrap. So, our general approach will be to unwrap different parts of the object (face, ears, and so on) using different unwrap calculations, selecting each calculation according to whatever works best for that piece. So let’s begin: We select only the “face” faces, unwrap them using the *Sphere* calculation, and scale and rotate them somewhat to fit logically within the image area of the UV/Image Editor window pan.

Once we’re satisfied with the face, it’s time to turn our attention to the ear. First, unselect the faces you were working with. Their UVs will disappear from the UV/Image Editor, but they are still there, just not shown. (To verify this, you can select a few faces in 3D view and it will show up in the UV/Image Editor.)

To work on the ear, in the 3D View, we now select only the “ear” faces. You can use Vertex Groups to select the ear faces. Selecting sub-meshes is easy too, since they are not connected to the rest of the mesh. Simply selecting Linked vertices will select that entire submesh. Basically, since you are in edit mode, all of the selecting/unselecting features are available to you.

Now re-unwrap the ear using the *Project* calculation from side view, and scale and rotate them somewhat (discussed in the next section), and place them off to the side. You can do this repetitively, using different UV calculations; each re-calculation just puts those UVs for the selected faces somewhere else. Choose the calculation for each piece that gives you the best fit and most logical layout for subsequent painting of that piece.

When all of the pieces of the mesh have been unwrapped using the various calculations, you should end up with something that looks like to the Example to the right. All of the sections of the mesh have been mapped, and all those maps are laid out in the same UV Texture map. Congratulations! From here, it is a simple matter of “stitching” (discussed in the next section) to construct the entire UV Map as a single map.

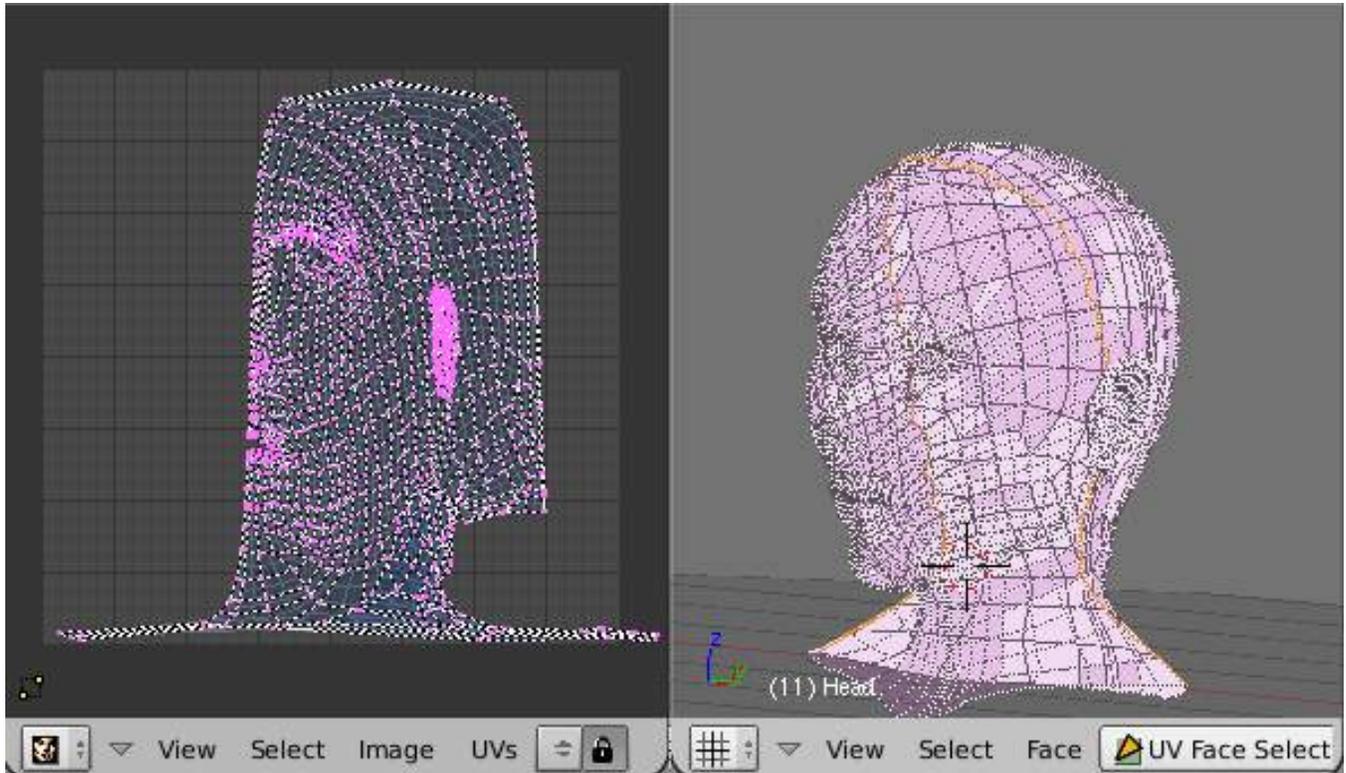


Fig. 2.1997: Bad Unwrap-Note Ear and Neck

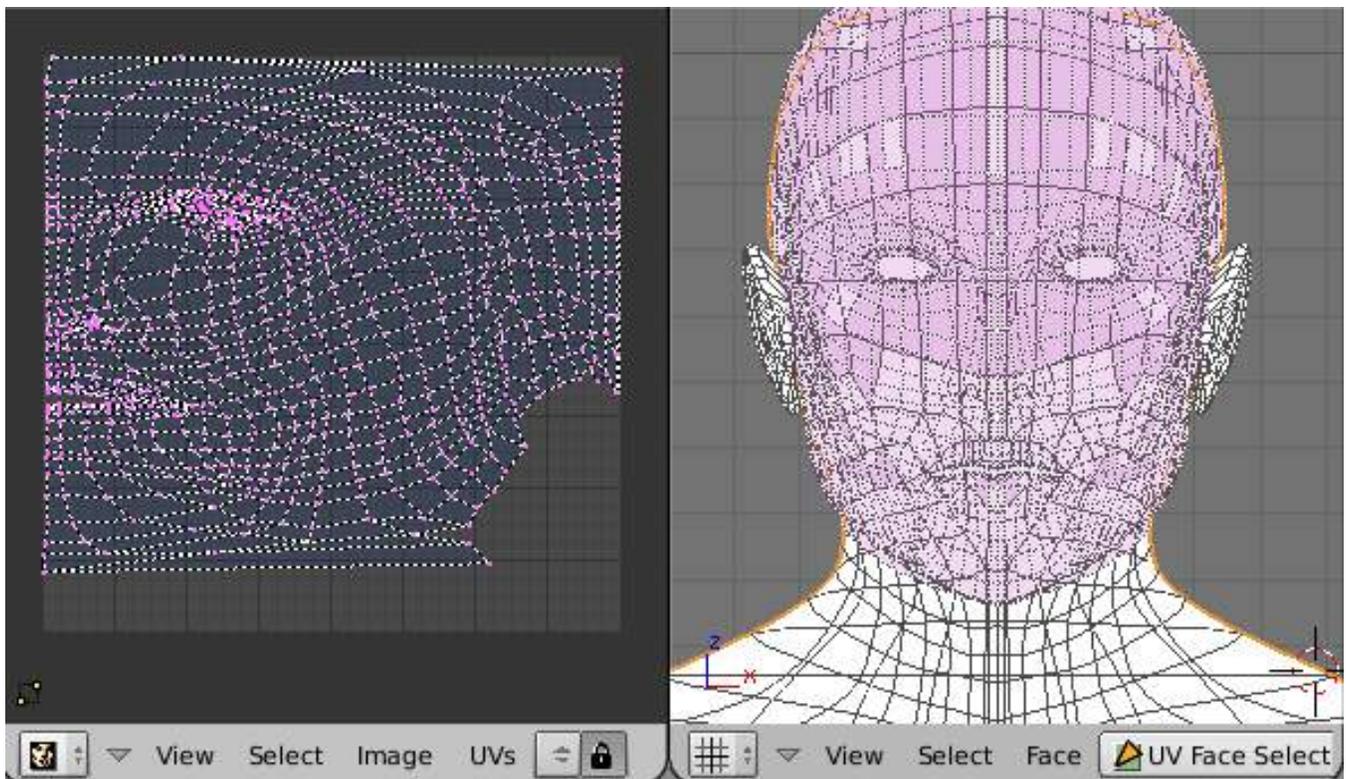


Fig. 2.1998: Unwrap Face Only, without Ear or Neck

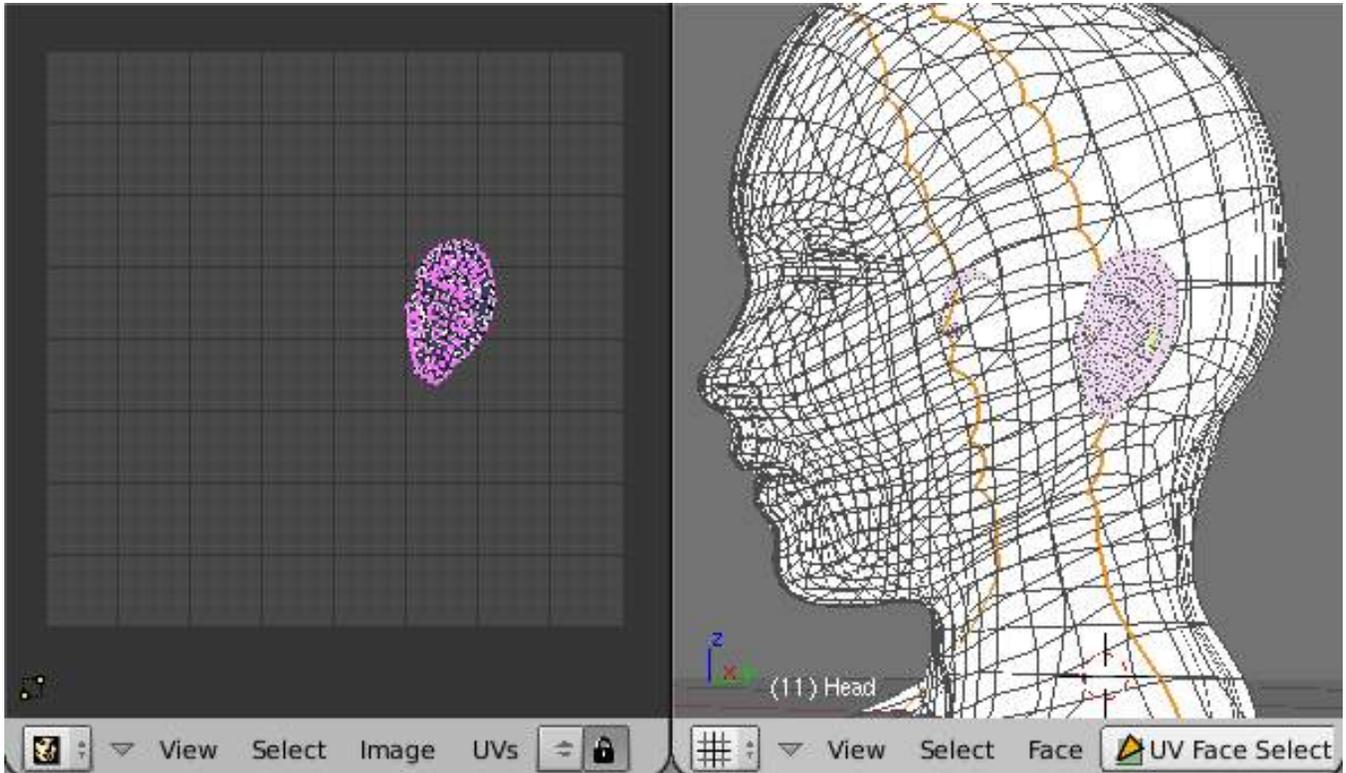


Fig. 2.1999: Unwrap Projection: Ear

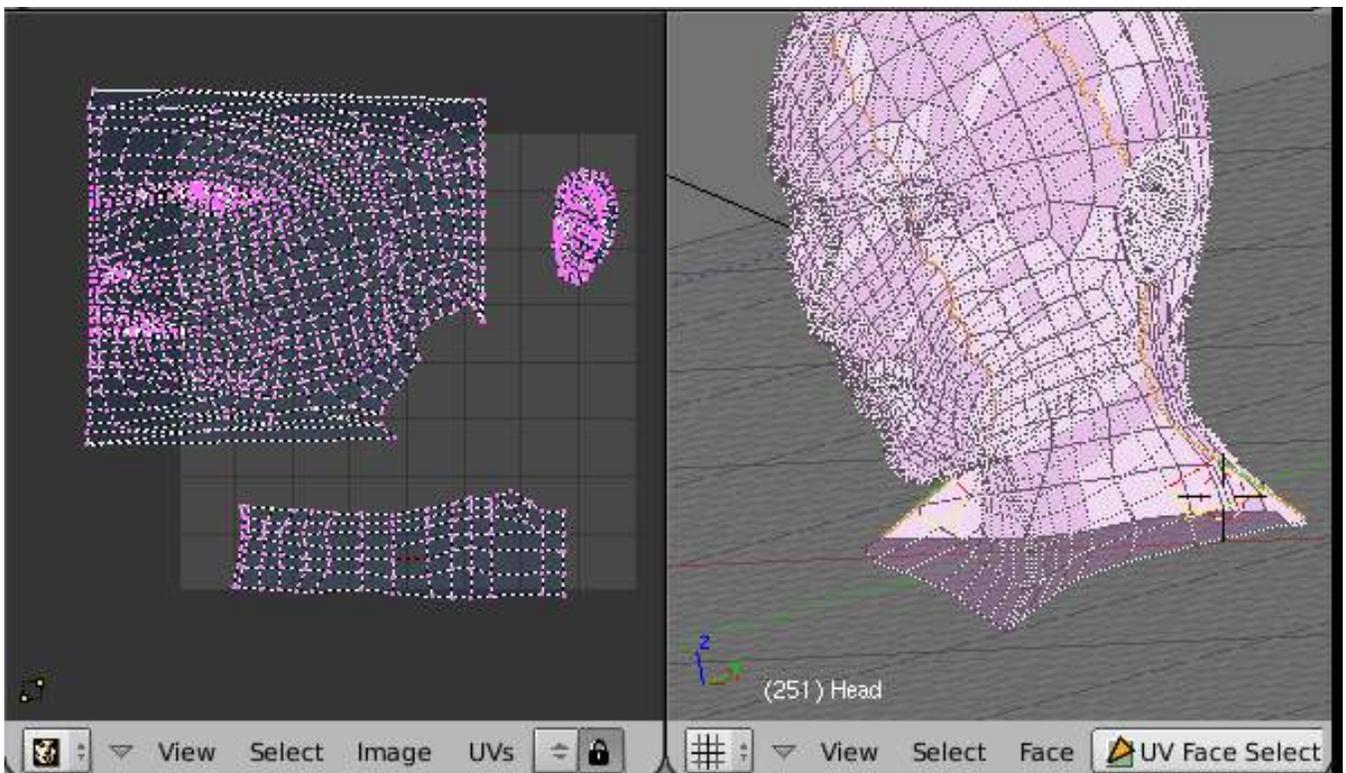


Fig. 2.2000: UV Maps together

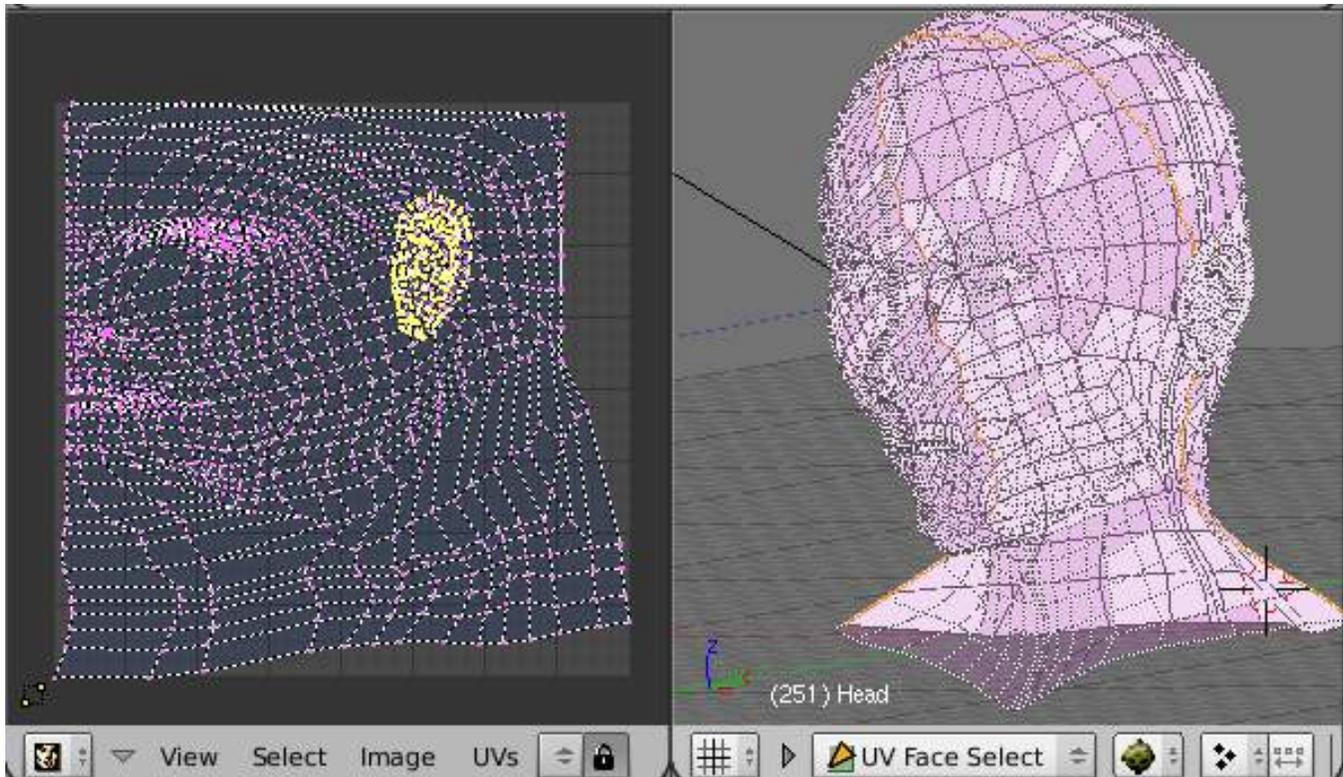


Fig. 2.2001: UV Maps Arranged and Stitched

When you have completed arranging and stitching, you will end up with a consolidated UV Map, like that shown to the right, arranged such that a single image will cover, or paint, all of the mesh that needs detailed painting. All of the detailed instructions on how to do this are contained in the next section. The point of this paragraph is to show you the ultimate goal. Note that the mesh shown is Mirrored along the Z axis, so the right side of the face is virtual; it is an exact copy of the left, so only one set of UVs actually exist. (If more realism is desired, the *Mirror* modifier would be applied, resulting in a physical mirror and a complete head. You could then make both sides physically different by editing one side and not the other. Unwrapping would produce a full set of UVs (for each side) and painting could thus be different for each side of the face, which is more realistic.)

**Average Island Scale** Using the *Average Island Scale* tool, shortcut `Ctrl-A`, will scale each UV island so that they are all approximately the same scale.

**Packing Islands** The *Pack Islands* tool, shortcut `Ctrl-P`, will uniformly scale, then individually transform each Island so that they fill up the UV space as much as possible. This is an important tool for efficiently making use of the texture space.

**Constraining to Image Bounds** Turning on *Constrain to Image Bounds* will prevent UVs from being moved outside the 0 to 1 UV range.

Grab/Move	G
Rotate	R
Scale	S
Weld/Align	W
Mirror...	M

Fig. 2.2002: UV Transformation Menu.

**Iteration and Refinement** At least for common people, we just don't "get it right the first time." It takes building on an idea and iterating our creative process until we reach that magical milestone called "Done." In software development, this is called the Spiral Methodology.

Applied to Computer Graphics, we cycle between modeling, texturing, animating, and then back to making some modifications to mesh, re-UV mapping, tweaking the animation, adding a bone or two, finding out we need a few more faces, so back to modeling, etc. We continue going round and round like this until we either run out of time, money, or patience, or, in some rare cases, are actually happy with our results.

**Refining the Layout** Refinement comes into play when we finally look at our character, and realize that we need more detail in a particular spot. For example, areas around the eyes might need crow's feet, or we need to add a logo to the vest. As you start to edit the image, you realize that there just aren't enough pixels available to paint the detail that you want.

Your only choice is to expand the size (scale out) that UV face. Using the minimize stretch or scale commands, you expand the UV faces around the eyes or chest, allocating more pixels to those areas, but at the same time taking away pixels (detail) from something else, like the back of the head. After refining the UV map, you then edit the image so that it looks right and contains the details you want.

**Reusing Textures** Another consideration is the need to conserve resources. Each image file is loaded in memory. If you can re-use the same image on different meshes, it saves memory. So, for example, you might want to have a generic 'face' painting, and use that on different characters, but alter the UV map and shape and props (sunglasses) to differentiate.

You might want to have a "faded blue jeans" texture, and unwrap just the legs of characters to use that image. It would be good to have a generic skin image, and use that for character's hands, feet, arms, legs, and neck. When modeling a fantasy sword, a small image for a piece of the sword blade would suffice, and you would Reset Unwrap the sword faces to re-use that image down the length of the blade.

## Applying Textures

Sooner or later, you may want to use an image texture on your model. If you are using an external application, you need to know where on the mesh you are painting. You may also need to test your UV mapping with a test image. This section covers how to export an outline of your UV map, and how to load images into the UV editor.

**Exporting UV Layout Image** As a way of communicating to an artist who is painting your UV Texture for you, Blender has a tool called *Save UV Face Layout* (located in the UV/Image Editor Window, UVs->Save UV Face Layout) that saves an image as a Targa (.tga), EPS, or an SVG format for the object you have selected.

The image is an outline of the UV face mapping. Activating the tool brings up the File Browser Window with options for saving the layout:

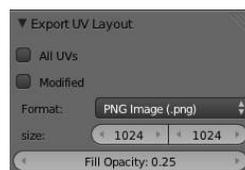


Fig. 2.2003: Export options

**All UVs** if disabled, then only the UV faces selected will be outlined

**Modified** Export UVs from the modified mesh.

**Format** Select the type of image file to save (.png, .eps, .svg)

**Size** select the size of the image in pixels. The image be square.

**Fill Opacity** Set the opacity of the fill

The image will be lines defining the UV edges that are within the image area of the UV mapping area. Edges outside the boundary, even if selected, will not be shown in the saved graphic.

The artist will use this as a transparent layer in their paint program as a guide when painting your texture. The example below shows Blender in the background, and the Gimp working on the texture, using the saved layout as a guide. Note that targa format supports the Alpha channel, so you can paint transparent areas of the mesh.

For using images as textures, see the page on [Image Textures](#)

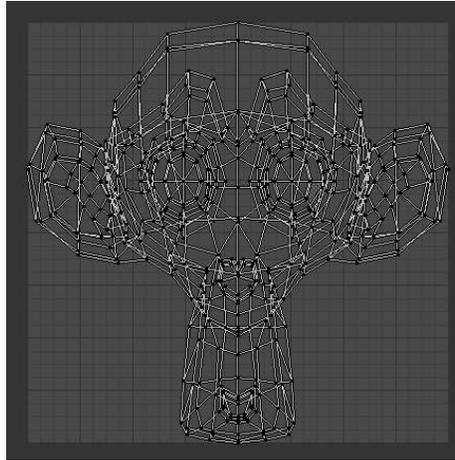


Fig. 2.2004: A uv layout in the uv editor

**Applying Textures to UVs** The UV/Image Editor allows you to map textures directly to the mesh faces. The 3D View window shows you the object being textured. If you set this window into Textured viewport shading, you will immediately see any changes made in the UV/Image Editor window in this window, and vice versa.

You can edit and load images, and even play a game in the Blender Game Engine with UV textures for characters and object, without a material, and still see them in the 3D window. This is because no ‘real’ rendering is taking place; it is all just viewport shading. If you were to apply an image to UVs then render, the texture would not show up by default

To render an image however, you must

- create a Material for the object, and
- tell Blender to use the UV Textures on faces when rendering.

To create a Material, you have to click *Add New Material* in the Shading context.

There are two ways to tell Blender to use the UV Texture when rendering: the Proper way and the Quick Way:

**Use UV Coordinates** In the Texture channel panel, Add a New Texture and define the texture as an image and load the image you want to use. In the Mapping section, choose UV from the Coordinates menu, and select the UV layer to use.

Make sure it is mapped to Color in the Influence section as well (it will be mapped to Color by default, and the UV Texture is named “UVTex” by default). If the image has an alpha channel and you want to use it, click “UseAlpha” in the Map Image panel.

Full details of using Image textures are on the [Image Textures](#) page.

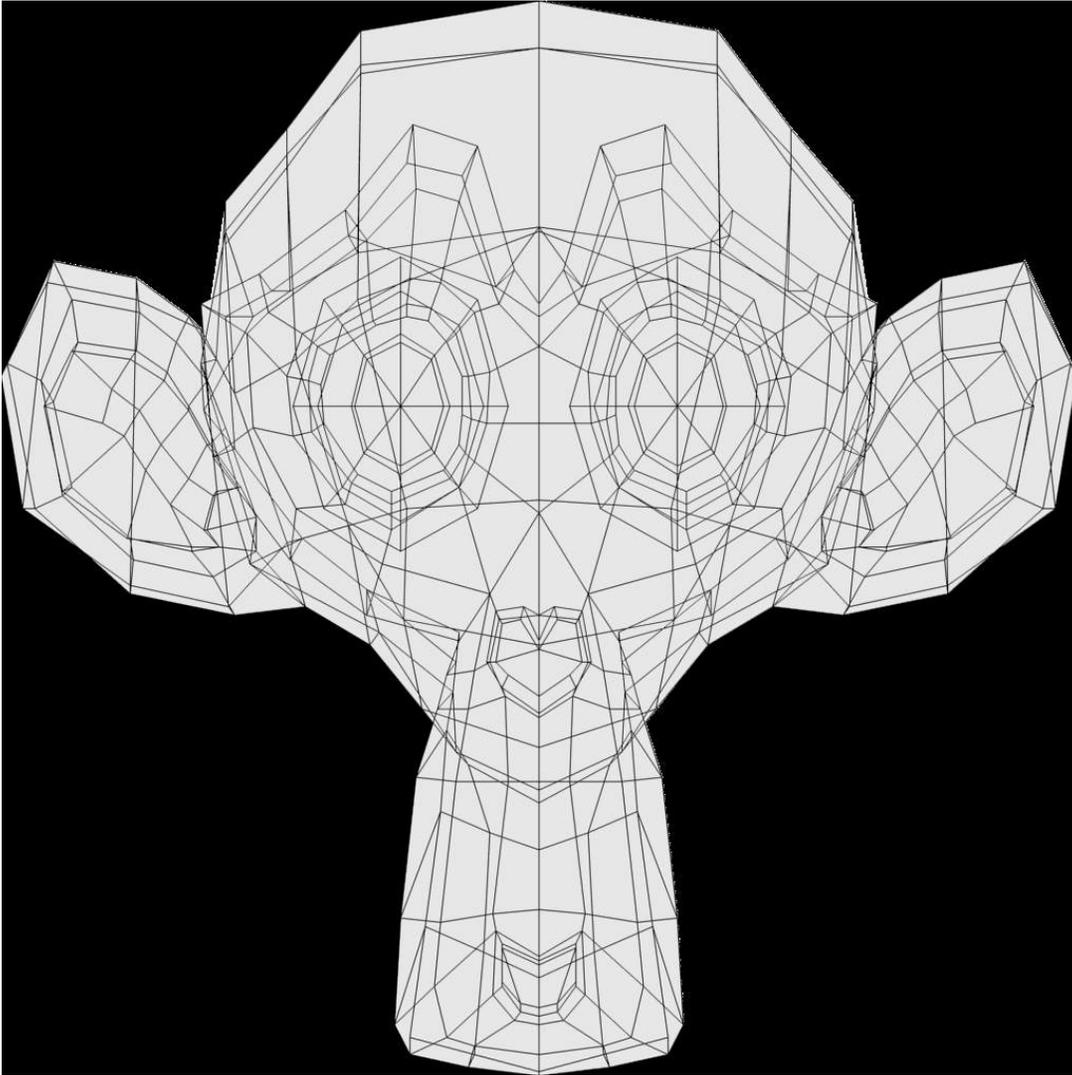


Fig. 2.2005: A snapshot of the uv layout to be used in an image editor

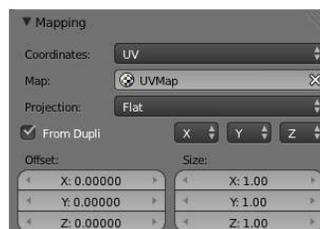


Fig. 2.2006: A texture setup to map using its UV coordinates

**Note:** Material is Required for Rendering

You can perform UV Texturing on a mesh within Blender without assigning a material, and you will even see it in your 3D View in textured viewport mode. However, when you render, you will just get a default gray if the object does not have a Material assigned. You will get a black if you do not load an image. If you do not create a texture that uses the image, or enable *Face Texture*, your object will render according to the procedural material settings.



Fig. 2.2007: The Material panel with activated Face Textures button.

**Face Textures** An alternate way is to set up a Face Textures Material as shown. To do so, with the buttons window displayed, press F5 to display the Shader Buttons. In the Buttons window, Material settings, click *ADD NEW* material.

On the Options panel, enable *Face Textures*. This way is quick, but bypasses the normal rendering system for fast results, but results which do not respect transparency and proper shading.

**Loading and Saving Images** In the UV editor, you can assign certain faces certain textures. To do so, first you need an image to work with. In the *Image Menu* you can open an image file with the *File Browser*. If you have images in the file already, that you want to use, click the *Browse* button in the *Header*, or make a new texture by clicking the *New* button.

In a team environment, or if you are using an external paint program to edit the image while the .blend file is active, and the file is updated and re-saved, use the UV/Image Editor to *Image*→*Reload* it and see the latest and greatest in Blender. Also, use *Reload* if you have mapped more faces to an image, and the 3D View will be updated with the latest image mapping back to faces.

If you move the image file, Blender may not be able to find it, and you will have to *Image*→*Replace* it. Use this option to map a UV layout to a different image altogether.

**Replacing the active Image** Recall that each face gets coordinates and a link to an image. To map a face to a different image, simply select that face (or faces) and use the UV/Image Editor window *Image*} } menu to *Replace* the current image with an existing file (such as a JPG or PNG file).



Fig. 2.2008: The new Image dialogue

**New Images** When you select *New Image* you are presented with several options. This *Generated* image can also be modified afterward in the *Properties Panel*:

**Image Name** Set the name if the generated image

**Width and Height** Set the size of the image in pixels

**Color** Sets the default fill color if creating a blank image.

**Alpha** Adds an alpha channel to the image

**Generated Type** The type of image to generate:

**UV Grid** Creates a checkerboard pattern with a colored + in each square.

**Color Grid** Creates a UV Test Grid, which is useful for testing how UVs have been mapped, and to reduce stretching. There are two types available, which can be set after the image has been created.

**Blank** Generates a blank image of the specified color.

**32 bit** Creates a 32 bit image. This is a larger file size, but holds much more color information than the standard 8 bit image. For close ups and large gradients, it may be better to use a 32 bit image.

**Using the Test Grid** Use the UV Test Grid option to check for undue stretching or distortion of faces. If your image is a base uniform pattern and you want the application of that image to your model to look like cloth, you do NOT want any stretching (unless you want the cloth to look like spandex).

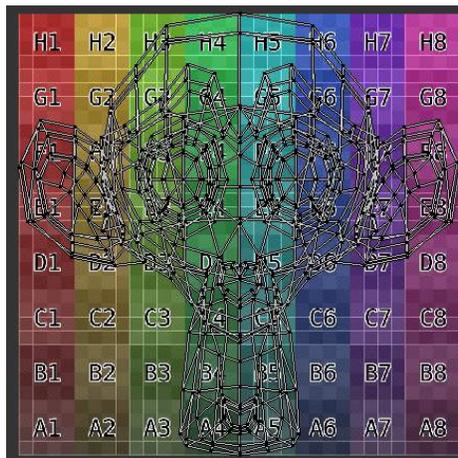


Fig. 2.2009: The test grid applied to the UVs

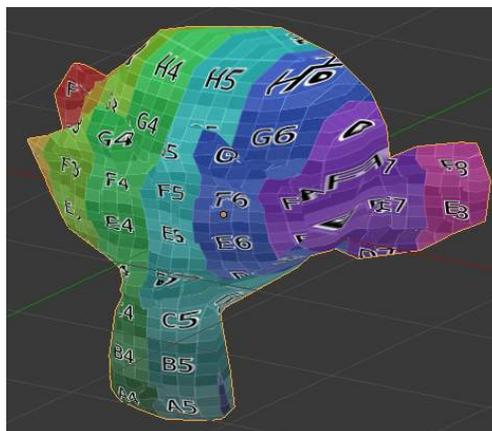


Fig. 2.2010: A preview of the texture on the geometry

When you render, the mesh will have the test grid as its colors, and the UV Texture will be the size image you specified. You can save the UV image using the Image→Save menu.

**Image Settings** When an image has been loaded or created in the UV editor, an additional section appears in the *Properties Panel*. The first row of buttons allow you to:

- Browse for an image
- Change the image name
- Set as *Fake User*
- Create a *New Image*
- *Open* an image
- *Unlink Datablock*

Select the image type in the *Source* menu. Each has different options:

**Generated** Generates a new image:

Width and Height of image in pixels

**Blank** Creates a Blank image

**UV grid** Creates a checkerboard pattern with colored plus symbols in each square.

**Color Grid** Creates a more complex colored grid with letters and numbers denoting locations in the grid.

**File** Use for loading image files:

**Fields** Use if image is made of fields. You can set it to use *Upper First* or *Lower First*

**Premultiply** Converts RGB from key alpha to premultiplied alpha.

**Movie and Sequence**

**Frames** Set the number of frames to use

**Start** Set the starting frame of the movie/sequence

**Offset** Offset the number of frame used in the animation

**Fields** Set the number fields per rendered frame to use(2 fields is 1 frame)

**Auto Refresh** Always refresh images on frame changes.

**Cyclic** Cycle the images in a movie/sequence.

**Saving Images** Images can be saved to external files if they were created or edited in Blender with tools in the *Image* menu. If images are already files, use the *Save* command (Alt+S). You can also *Save As* (F3) if the image was generated or you want to save as a different name. Using *Save as Copy*, (F3) will save the file to a specified name, but will keep the old one open in the Image editor.

**Modifying your Image Texture** To modify your new Texture, you can:

- [Render Bake](#) an image based on how the mesh looks
  - The Render Bake feature provides several tools to replace the current image based on a render of [Vertex Paint](#) colors, Normals (bumps), Procedural materials, textures and lighting, and ambient occlusion.
- Paint using [Texture Paint](#).
  - Use the UV/Image Editor menu *Image* → *New*. Then start painting your mesh with

- Use external software to create an image
  - Using your favorite image painting program, you could use an exported UV layout to create a texture. Then save your changes, and back in Blender, use the Image→Open menu command to load it as your UV image for the mesh in Face Select Mode for the desired (and active) UV Texture layer. Using the *Edit Externally* tool in the *Image* menu, Blender will open an image editor, as specified in the *User Preferences* and load in the image to be edited.
- Use the “projection painting” feature of recent versions of Blender
- Use the Bake uV-Textures to Vertex Colors add-on to create an image from vertex colors
- Some combination of the above.

The first three options, (UV Painter, Render Bake, and Texture Baker) replace the image with an image that they create. Texture paint and external software can be used to add or enhance the image. Regardless of which method you use, ultimately you must either

- save your texture in a separate image file (for example JPG for colors, PNG with RGBA for alpha),
- pack the image inside the blend file (UV/Image Editor Image→Pack as PNG),
- or do both.

The advantage to saving as a separate file is that you can easily switch textures just by copying other image files over it, and you can use external editing programs to work on it. The advantage of packing is that your whole project is kept in the .blend file, and that you only have to manage one file.

You can invert the colors of an image by selecting the *Invert* menu. in the *Image* menu

**Packing Images inside the Blend file** If you pack your .blend file, the current version of all UV Texture images are packed into the file. If those files later change, the updates will not be automatically re-packed; the old version of the image is what will be used. To update, you will have to re-pack or reload.

To pack an image, select *Pack Image* from the *Image* menu. To Unpack, select this option again and select *Remove Pack*.

The File→Append function automatically goes into .blend files and shows you the image textures packed in it. The public domain Blender Texture CD is also a great resource, and there are many other sources of public domain (and licensed) textures. All textures on the Elephants Dream CD are liberally licensed under [CC-BY 2.5](#).

## Influence

- [Doc:2.6/Manual/Textures/Influence/Material - Bump and Normal - Displacement](#)
- [Doc:2.6/Manual/Textures/Influence/World](#)
- [Doc:2.6/Manual/Textures/Influence/Particles](#)

## Material Textures Influence

Not only can textures affect the color of a material, they can also affect many of the other properties of a material. The different aspects of a material that a texture influences are controlled in the *Influence* panel.

---

**Note:** Texture options for *Surface* and *Wire* materials and in some cases also for *Volume* and *Halo* materials.

---

## Surface and Wire materials

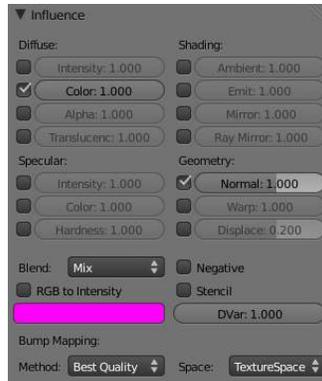


Fig. 2.2011: Texture Influence panel for a Surface material

### Diffuse

**Intensity** Amount texture affects affects diffuse reflectivity

**Color** Amount texture affect the basic color or RGB value of the material

**Alpha** Influences the opacity of the material. See [Use Alpha for Object Transparency](#). Also use *Z Transparency* for light and if combining multiple channels.

**Translucency** Influences the Translucency amount.

### Specular

**Intensity** Amount texture affect specular reflectivity

**Color** Influences the *Specular* color, the color of the reflections created by the lamps on a glossy material.

**Hardness** Influences the specular hardness amount. A DVar of 1 is equivalent to a Hardness of 130, a DVar of 0.5 is equivalent to a Hardness of 65.

### Shading

**Ambient** Influences the amount of Ambient light the material receives.

**Emit** Influences the amount of light Emitted by the material.

**Mirror** Influences the mirror color. This works with environment maps and raytraced reflection.

**Ray Mirror** Influences the strength of raytraced mirror reflection.

### Geometry

**Normal** Commonly called bump mapping, this alters the direction of the surface normal. This is used to fake surface imperfections or unevenness via bump mapping, or to create reliefs.

**Warp** *Warp* allows textures to influence/distort the texture coordinates of a next texture channel. The distortion remains active over all subsequent channels, until a new *Warp* has been set. Setting the factor at zero cancels out the effect.

**Displace** Influences the Displacement of vertices, for using [Displacement Maps](#).

## Other Controls

**Blend** Blending operation to perform. See [Texture Blending Modes](#) for details.

**RGB to intensity** With this option enabled, an RGB texture (affects color) is used as an intensity texture (affects a value).

**Blend Color** If the texture is mapped to Col, what color is blended in according to the intensity of the texture? Click on the swatch or set the RGB sliders.

**Negative** The effect of the Texture is negated. Normally white means on, black means off, *Negative* reverses that.

**Stencil** The active texture is used as a mask for all following textures. This is useful for semitransparent textures and “Dirt Maps”. Black sets the pixel to “untexturable”. The *Stencil* mode works similar to a layer mask in a 2D program. The effect of a stencil texture can not be overridden, only extended. You need an intensity map as input.

**DVar** Destination Value (not for RGB). The value with which the Intensity texture blends with the current value. Two examples:

- The *Emit* value is normally 0. With a texture mapped to *Emit* you will get maximal effect, because *DVar* is 1 by default. If you set *DVar* to 0 no texture will have any effect.
- If you want transparent material, and use a texture mapped to *Alpha*, nothing happens with the default settings, because the *Alpha* value in the *Material* panel is 1. So you have to set *DVar* to 0 to get transparent material (and of course *Z Transparency* also). This is a common problem for beginners. Or do it the other way round - set *Alpha* to 0 and leave *Dvar* on 1. Of course the texture is used inverted then.

**Bump Mapping** Settings for bump mapping. *Method Best Quality, Default, Compatible, Original Space*  
*Texture Space, Object Space, View Space*

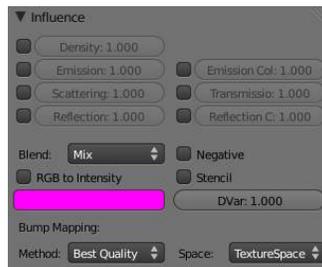


Fig. 2.2012: Texture Influence panel for Volume material

**Volume materials** Special texture options for *Volume* materials

**Density** Causes the texture to affect the volume’s density.

**Emission** Causes the texture to affect the volume’s emission.

**Scattering** Amount the texture affects scattering.

**Reflection** Amount the texture affects brightness of out-scattered light

**Emission Color** Amount the texture affects emission color.

**Transmission** Amount the texture affects result color after light has been scattered/absorbed.

**Reflection Color** Amount the texture affects color of out-scattered light.

**Halo materials** Special texture options for *Halo* materials

**Size** Amount the texture affects ray mirror.

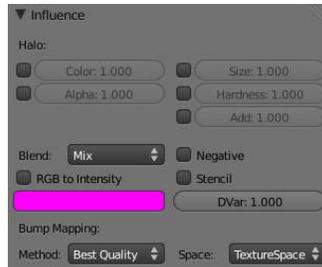


Fig. 2.2013: Texture Influence panel for a Halo material

**Hardness** Amount the texture affects hardness.

**Add** Amount the texture affects translucency.

### Texture Blending Modes

Blending Modes are different methods of controlling how the texture influences material properties. While a blending mode defines the specific operation performed, blending factor controls the amount, the overall “strength” of this operation. For textures such blending factor is set via sliders in the Influence panel. Throughout this section, the term *base layer* refers to the base material color being manipulated (as defined by texture’s Influence) and *blend layer* refers to the texture. Following is a list of available texture blending modes:

#### See also:

[Color Blend Modes](#) for details on each blending mode.

### Bump and Normal Maps

**Description** *Normal Maps* and *Bump Maps* both serve the same purpose: they simulate the impression of a detailed 3D surface, by modifying the shading as if the surface had lots of small angles, rather than being completely flat. Because it’s just modifying the shading of each pixel, this will not cast any shadows and will not obstruct other objects. If the camera angle is too flat to the surface, you will notice that the surface is not really shaped.

Both *Bump Maps* and *Normal Maps* work by modifying the normal angle (the direction pointing perpendicular from a face), which influences how a pixel is shaded. Although the terms *Normal Map* and *Bump Map* are often used synonymously, there are certain differences.

**Bump maps** These are textures that store an **intensity**, the relative height of pixels from the viewpoint of the camera. The pixels seem to be moved by the required distance in the direction of the face normals. (The “bump” consists only of a displacement, which takes place along the existing, and unchanged, normal-vector of the face.) You may either use greyscale pictures or the intensity values of a RGB-Texture (including images).

**Normal maps** These are images that store a **direction**, the direction of normals directly in the RGB values of an image. They are much more accurate, as rather than only simulating the pixel being away from the face along a line, they can simulate that pixel being moved at any direction, in an arbitrary way. The drawbacks to normal maps are that unlike bump maps, which can easily be painted by hand, normal maps usually have to be generated in some way, often from higher resolution geometry than the geometry you’re applying the map to.

Normal maps in Blender store a normal as follows:

- Red maps from (0-255) to X (-1.0 - 1.0)
- Green maps from (0-255) to Y (-1.0 - 1.0)
- Blue maps from (0-255) to Z (0.0 - 1.0)

Since normals all point towards a viewer, negative Z-values are not stored (they would be invisible anyway). In Blender we store a full blue range, although some other implementations also map blue colors (128-255) to (0.0 - 1.0). The latter convention is used in “Doom 3” for example.

**Workflow** The steps involved in making and using Bump and Normal Maps is:

- Model a highly detailed (“hi-poly”) model
- Bake the Bump and/or Normal maps
- Make a low-poly, less detailed model
- Map the map to the low-poly model using a common coordinate system

Consult the Modeling section for how to model a highly detailed model using the Mesh tools. How much detail you put in is totally up to you. The more ridges and details (knobs, creases, protrusions) you put in, the more detailed your map will be.

Baking a map, simply put, is to take the detail of a high polygon mesh, and apply it to a similar object. The similar object is identical to the high-poly mesh except with less vertices. Use the [Render Bake](#) feature in Blender to accomplish this.

Modeling a low-poly using Blender’s Mesh editing tools. In general, the same or similar faces should exist that reflect the model. For example, a highly detailed ear may have 1000 faces in the high-poly model. In the low-poly model, this may be replaced with a single plane, oriented in the same direction as the detailed ear mesh. (*Tip:* Blender’s [multi-resolution mesh](#) modeling feature can be used to good effect here.)

Mapping is the process of applying a texture to the low-poly mesh. Consult the [Textures Mapping](#) section for more information on applying a texture to a mesh’s material. Special considerations for Bump and Normal Maps is:

- When using a Bump map, map the texture to *Normal* and enable *No RGB*.
- When using a Normal map, map the texture to *Normal*.

The coordinate systems of the two objects must match. For example, if you bake using a UV map of the high-poly model, you must UV map the low poly model and line up its UV coordinates to match the outline of the high-poly image (see [UV unwrapping](#) to line up with the high-poly map edges.

## Displacement Maps

**Description** Displacement mapping allows a texture input to manipulate the position of vertices on rendered geometry. Unlike [Normal or Bump mapping](#), where the shading is distorted to give an illusion of a bump (discussed on the previous page), Displacement Maps create real bumps, creases, ridges, etc in the actual mesh. Thus, the mesh deformations can cast shadows, occlude other objects, and do everything that changes in real geometry can do, but, on the other hand, requires a lot more vertices to work.

**Options** In the [Influence](#) panel, the strength of the displacement is controlled by the *Displace* and *Normal* sliders.

- If a texture provides only normal information (e.g. *Stucci*), vertices move according to the texture’s normal data. The normal displacement is controlled by the *Normal* slider.
- If a texture provides only intensity information (e.g. *Magic*, derived from color), vertices move along the directions of their normals (a vertex has no normal itself, it’s the resulting vector of the adjacent faces). White pixels move outward in the direction of the normal, black pixels move in the opposite direction. The amount of displacement is controlled with the *Displace* slider.

The two modes are not exclusive. Many texture types provide both information (*Clouds, Wood, Marble, Image*). The amount of each type can be mixed using the respective sliders. Intensity displacement gives a smoother, more continuous surface, since the vertices are displaced only outward. Normal displacement gives a more aggregated surface, since the vertices are displaced in multiple directions.

The depth of the displacement is scaled with an object's scale, but not with the relative size of the data. This means if you double the size of an object in object mode, the depth of the displacement is also doubled, so the relative displacement appears the same. If you scale inside *Edit Mode*, the displacement depth is not changed, and thus the relative depth appears smaller.

**Hints** Displacement maps move the rendered faces, not the physical mesh faces. So, in 3D View the surface may appear smooth, but render bumpy. To give a detailed surface, there has to be faces to displace and have to be very small. This creates the trade-off between using memory and CPU time versus render quality.

From best to worst, displacement works with these object types using the methods listed to control the render face size:

**Subdivision Surface Meshes** Rendered face size is controlled with render subsurf level. Displacement really likes smooth normals.

**Manually ( *Edit Mode* ) subdivided meshes** Control render faces with number of subdivides. (This can be combined with the above methods.) Displaces exactly the same Simple Subsurf, but slows editing down because of the OpenGL overhead of drawing the extra faces. (You can't turn the edit subdivide level down this way).

**Meta Objects** Control render faces with render wiresize. Small wire == more faces.

The following are available, but currently don't work well. It is recommended that you convert these to meshes before rendering.

**Open NURBS Surfaces** Control render faces with *U/V Surface Resolution*. Higher numbers give more faces. (Note normal errors).

**Closed NURBS Surfaces** Control with *Surface Resolution* controls. (Note the normal errors, and how implicit seam shows).

**Curves and Text** Control with *Surface Resolution* controls. Higher gives more render faces. (Note that the large flat surfaces have few render faces to displace).

---

**Note:** Displace Modifier

If you want more control over your displacement, you'll probably want to use the [Displace Modifier](#). This feature has lots of different options so that you can customize the displacement exactly to your liking.

---

## World

## Particles

## Lighting

### Introduction

Lighting is a very important topic in rendering, standing equal to modeling, materials and textures. The most accurately modeled and textured scene will yield poor results without a proper lighting scheme, while a simple model can become very realistic if skillfully lit.

**Viewing Restrictions** The color of an object and the lighting of your scene is affected by:

- Your ability to see different colors (partial color blindness is common).
- The medium in which you are viewing the image (e.g. an LCD panel versus printed glossy paper).
- The quality of the image (e.g. a JPEG at 0.4 compression versus 1.0).
- The environment in which you are viewing the image (e.g. a CRT monitor with glare versus in a dark room, or in a sunshiny blue room).

- Your brain’s perception of the color and intensity relative to those objects around it and the world background color, which can be changed using color manipulation techniques using Blender [Composite Nodes](#).

**Global Influences** In Blender, the elements under your control which affect lighting are:

- The color of the world [ambient light](#).
- The use of [Ambient Occlusion](#) as a way to cast that ambient light onto the object.
- The degree to which the ambient light colors the [material](#) of the object.
- The use of [Indirect lighting](#), where the color of one object radiates onto another.
- The [lamps](#) in your scene.

The physics of light bouncing around in the real world is simulated by Ambient Occlusion (a world setting), buffer shadows (which approximate shadows being cast by objects), ray tracing (which traces the path of photons from a light source). Also, within Blender you can use [Indirect lighting](#). Ray tracing, ambient occlusion, and indirect lighting are computer-intensive processes. Blender can perform much faster rendering with its internal scan line renderer, which is a very good scan line renderer indeed. This kind of rendering engine is much faster since it does not try to simulate the real behavior of light, assuming many simplifying hypotheses.

**Lighting Settings** Only after the above global influences have been considered, do you start adding light from lamps in your scene. The main things under your control are the:

- Type of light used (*Sun, Spot, Lamp, Hemi, etc.*).
- Color of the light.
- Position of the light and its direction.
- Settings for the light, including energy and falloff.

Then you are back to how that material’s [shader](#) reacts to the light.

This chapter attempts to address the above, including how lights can work together in rigs to light your scene. In this chapter we will analyze the different type of lights in Blender and their behavior; we will discuss their strong and weak points. We also describe many lighting rigs, including the ever-popular three-point light method.

**Lighting in the Workflow** In this user manual we have placed Lighting before Materials; you should set up your lighting before assigning materials to your meshes. Since the material shaders react to light, without proper lighting, the material shaders will not look right, and you will end up fighting the shader, when it is really the bad lighting that is causing you grief. All of the example images in this section do not use any material setting at all on the ball, cube or background.

**Overriding Materials to Reset Lighting** If you have started down the road of assigning materials, and are now fiddling with the lighting, we suggest that you create a default, generic gray material—no *Vertex Color*, no *Face Texture*, no *Shadeless*, just plain old middle gray with RGB of (0.8, 0.8, 0.8). Name this `Gray`.

Next go to the *Render* context. In the *Render Layers* panel, select your new `Gray` material in the *Material* field. This will override any materials you may have set, and render everything with this color. Using this material, you can now go about adjusting the lighting. Just empty this field to get back to your original materials.

## Lights

**Introduction** As previously stated, there are multiple types of lighting in Blender, like indirect light or ambient light. However, one of the most used are “lights”, or “lamps”. In this section, we will discuss general info and settings for these lights (you will find more lamp-specific details in the [Lamps](#) section):

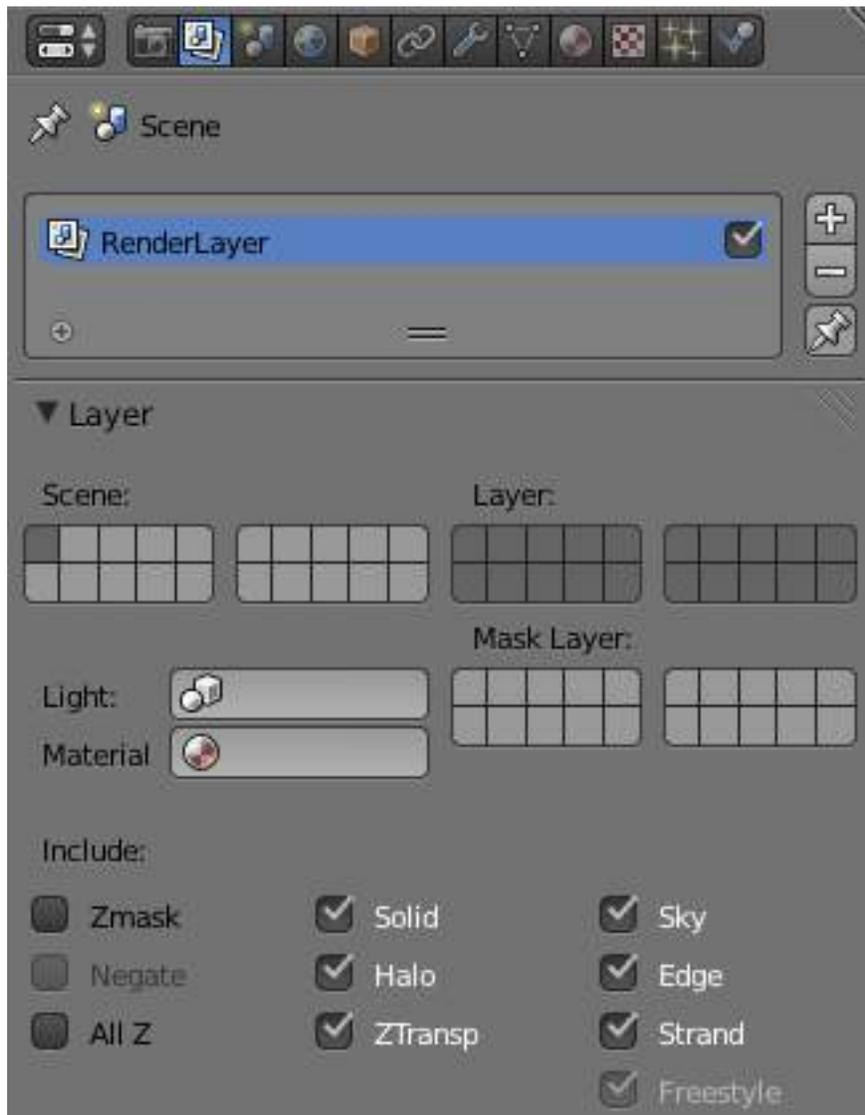


Fig. 2.2014: Material field in the Render Layers panel

- [Light Properties](#) - settings common to all lamps.
- [Light Attenuation](#).
- [Textures](#) - how to apply texture(s) to lamps.
- [What Light Affects](#).
- [Lights In Other Contexts](#) - lamp-related setting in other contexts.

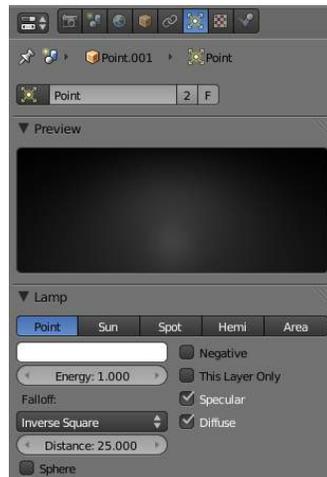


Fig. 2.2015: Lamp Properties panels

**Lights Common Options** There are five types of lamps in Blender. They share all or some of the options listed here:

### Object Data

**Browse Light Object Data** Click to view all lights in the current scene.

**Name** The name of the currently selected light object data. Edit to change the name.

**Number of Users** The number of light objects sharing the light object data.

**F** Create a fake user for this object data.

**Preview** A quick preview of the light settings.

Lamp

**Distance** The *Dist* field indicates the number of Blender Units (BU) at which the intensity of the current light source will be half of its intensity. Objects less than the number of BU away from the lamp will get more light, while objects further away will receive less light. Certain settings and lamp falloff types affect how the *Distance* field is interpreted, meaning that it will not always react the same; see the page about [light falloff](#).

- The *Sun* and *Hemi* Lamps are another class of Lamps which uses a constant falloff. Those lamps don't have a *Dist* field, and are often called "Base Lighting Lamps".

**Energy** The intensity of the light source's illumination (from 0.0 to 10.0).

**Color** The color of the light source's illumination. Opens a color swatch.

**Negative** Let the lamp cast negative light.

**This Layer Only** The Lamp only illuminates objects on the same layer the lamp is on.

**Specular** The Lamp creates specular highlights.

**Diffuse** The Lamp does diffuse shading.

### Light Attenuation

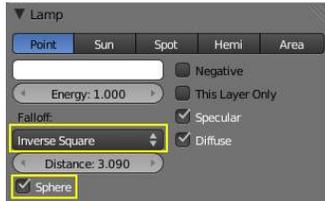


Fig. 2.2016: Lamp panel, falloff options highlighted

**Description** There are two main controls for light falloff for *Point* and *Spot* lamps.

- The lamp *Falloff* type drop-down list, and
- The *Sphere* button.

### Falloff types

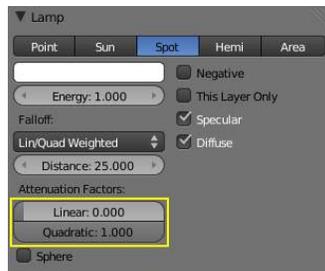


Fig. 2.2017: Lamp panel with Lin/Quad Weighted Falloff options highlighted

**Lin/Quad Weighted** When this setting is chosen, two sliders are shown, *Linear* and *Quadratic*, which control respectively the “linearness” and “quadraticness” of the falloff curve.

This lamp falloff type is in effect allowing the mixing of the two light attenuation profiles (linear and quadratic attenuation types).

**Linear** This slider input field can have a value between 0.0 and 1.0. A value of 1.0 in the *Linear* field and 0.0 in the *Quadratic* field in effect means that the light from this source is completely linear. This means that at the number of Blender Units distance specified in the *Distance* field, this light source’s intensity will be half the value it was originally.

When the *Quadratic* slider is set to 0.0, the formula for working out the attenuation at a particular range for full linear attenuation is:

$$I = E \times (D / (D + L \times r))$$

Where

- $I$  is the calculated Intensity of light.
- $E$  is the current *Energy* slider setting.
- $D$  is the current setting of the *Dist* field.
- $L$  is the current setting of the *Linear* slider.
- $r$  is the distance from the lamp where the light intensity gets measured.

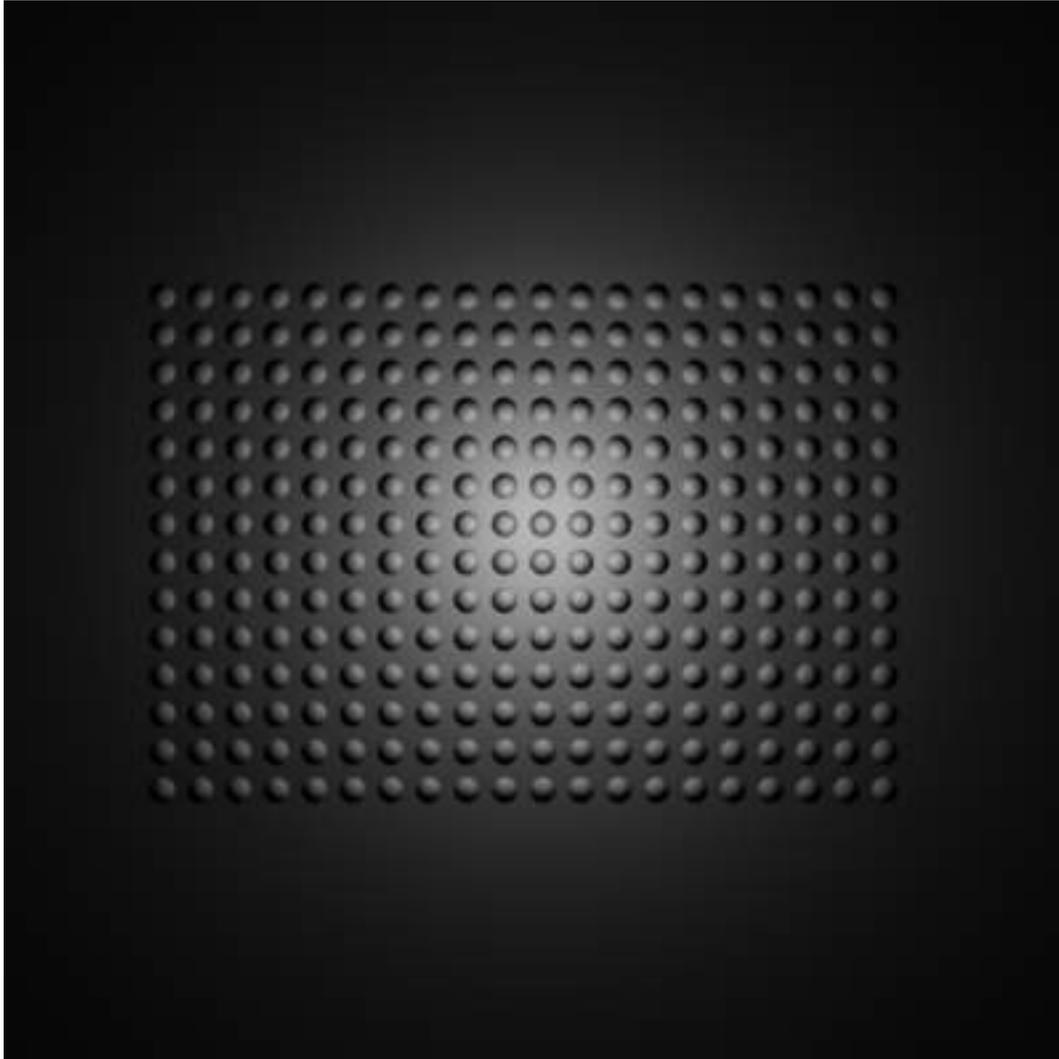


Fig. 2.2018: Lamp with Lin/Quad Weighted falloff default settings

**Quadratic** This slider input field can have a value between 0.0 and 1.0. A value of 1.0 in the *Quadratic* field and 0.0 in the *Linear* field means that the light from this source is completely quadratic.

Quadratic attenuation type lighting is considered a more accurate representation of how light attenuates (in the real world). In fact, fully quadratic attenuation is selected by default for *Lin/Quad Weighted* lamp falloff (see *Lamp with Lin/Quad Weighted falloff default settings*).

Here again, the light intensity is half when it reaches the *Distance* value from the lamp. Comparing the quadratic falloff to the linear falloff, the intensity decays much slower at distances lower than the set *Distance*, but it attenuates much quicker after *Distance* is reached.

When the *Linear* slider is set to 0.0, the formula for working out the attenuation at a particular range for full quadratic attenuation is:

$$I = E \times (D^2 / (D^2 + Q \times r^2))$$

Where

- I is the calculated Intensity of light.
- E is the current *Energy* slider setting.
- D is the current setting of the *Dist* field.
- Q is the current setting of the *Quad* slider.
- r is the distance from the lamp where the light intensity gets measured.

**Mixing “Linear” and “Quad”** If both the *Linear* and *Quad* slider fields have values greater than 0.0, then the formula used to calculate the light attenuation profile changes to this:

$$I = E \times (D / (D + L \times r)) \times (D^2 / (D^2 + Q \times r^2))$$

Where

- I is the calculated Intensity of light.
- E is the current *Energy* slider setting.
- D is the current setting of the *Dist* field.
- L is the current setting of the *Linear* slider.
- Q is the current setting of the *Quad* slider.
- r is the distance from the lamp where the light intensity gets measured.

**Zeroing both “Linear” and “Quad”** If both the *Linear* and *Quadratic* sliders have 0.0 as their values, the light intensity will not attenuate with distance. This does not mean that the light will not get darker - it will, but only because the energy the light has is spread out over a wider and wider distance. The total amount of energy in the spread-out light will remain the same, though. The light angle also affects the amount of light you see. It is in fact the behavior of light in the deep space vacuum.

If what you want is a light source that doesn't attenuate and gives the same amount of light intensity to each area it hits, you need a light with properties like the *Constant* lamp *Falloff* type.

Also, when the *Linear* and *Quad* sliders are both 0.0 values the *Distance* field ceases to have any influence on the light attenuation, as shown by the equation above.

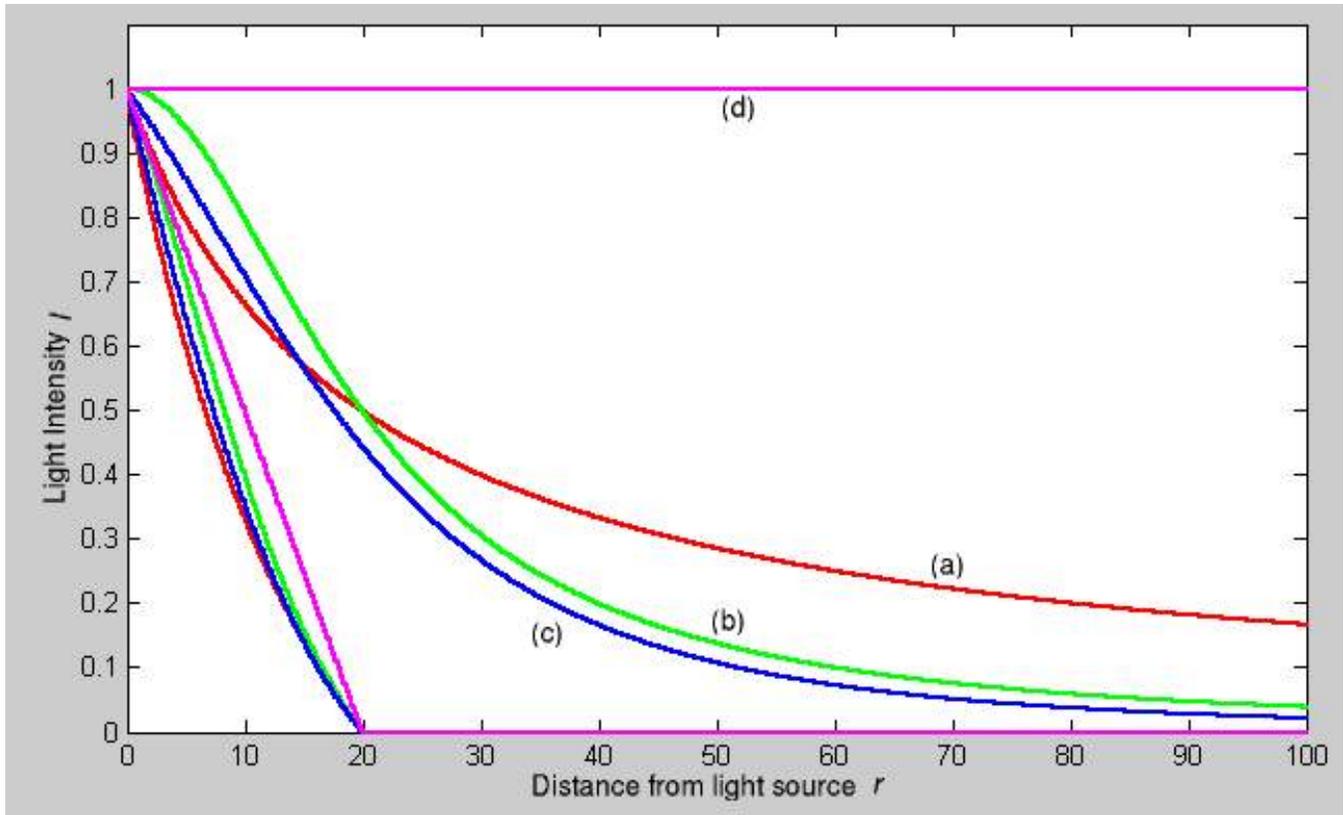
**Graphical Summary** Below is a graph summarizing the lin/quad attenuation type, showing attenuation with or without the *Sphere* option (described later).

**Custom Curve** The *Custom Curve* lamp *Falloff* type is very flexible.

Most other lamp falloff types work by having their light intensity start at its maximum (when nearest to the light source) and then with some predetermined pattern decrease their light intensity when the distance from the light source increases.

When using the *Custom Curve* Lamp Falloff type, a new panel is created called *Falloff Curve*. This *Falloff Curve* profile graph allows the user to alter how intense light is at a particular point along a light's attenuation profile (i.e. at a specific distance from the light source).

The *Falloff Curve* profile graph has two axes, the *Distance* axis and the *Intensity* axis.



**Distance axis** It represents the position at a particular point along a light source’s attenuation path. The far left is at the position of the light source and the far right is the place where the light source’s influence would normally be completely attenuated. I say “normally would” because the *Falloff Curve* can be altered to do the exact opposite if required.

**Intensity axis** It represents the intensity at a particular point along a light source’s attenuation path. Higher intensity is represented by being higher up the intensity axis, while lower intensity light is represented by being lower down on the intensity axis.

Altering the *Falloff Curve* profile graph is easy. Just LMB click on a part of the graph you want to alter and drag it where you want it to be. If when you click you are over or near one of the tiny black square handles, it will turn white, indicating that this handle is now selected, and you will be able to drag it to a new position. If when you click on the graph you are not near a handle, one will be created at the point that you clicked, which you can then drag where you wish. You can also create handles at specific parts of the graph, clicking with LMB while holding `Ctrl` key; it will create a new handle at the point you have clicked.

In the example below (the default for the *Falloff Curve* Profile Graph), the graph shows that the intensity of the light starts off at its maximum (when near the light), and linearly attenuates as it moves to the right (further away from the light source).



If you want to have a light attenuation profile that gets more intense as it moves away from the light source, you could alter the graph as below:



You are obviously not just limited to simple changes such as reversing the attenuation profile, you can have almost any profile you desire.

Here is another example of a different *Falloff Curve* profile graph, along with its resultant render output:



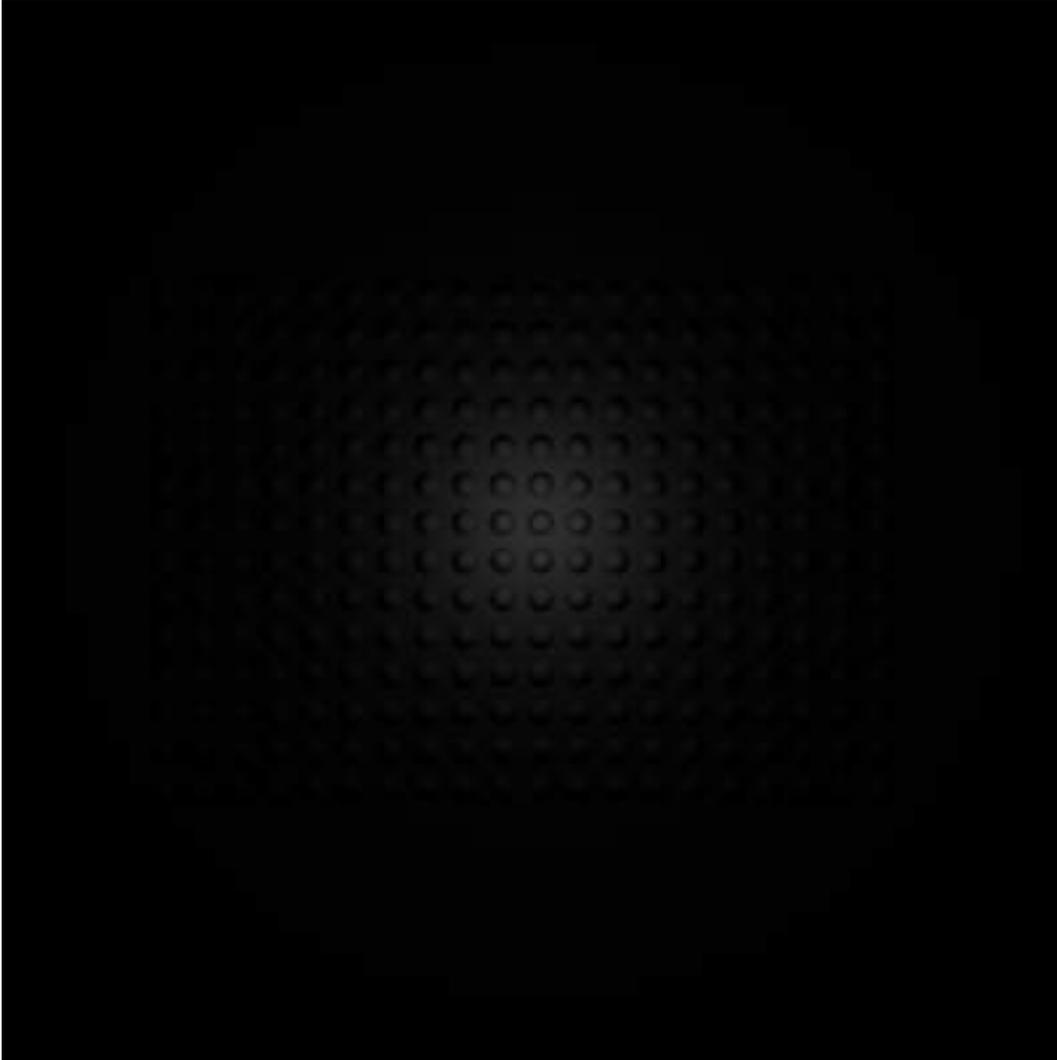


Fig. 2.2031: Render showing the Inverse Square lamp falloff type effect with default settings.

**Inverse Square** This lamp falloff type attenuates its intensity according to inverse square law, scaled by the *Distance* value. Inverse square is a sharper, realistic decay, useful for lighting such as desk lamps and street lights. This is similar to the old *Quad* option (and consequently, to the new *Lin/Quad Weighted* option with *Linear* to 0.0 and *Quad* to 1.0), with slight changes.

**Inverse Linear** This lamp falloff type attenuates its intensity linearly, scaled by the *Dist* value. This is the default setting, behaving the same as the default in previous Blender versions without *Quad* switched on, and consequently, like the new *Lin/Quad Weighted* option with *Linear* to 1.0 and *Quad* to 0.0. This isn't physically accurate, but can be easier to light with.

**Constant** This lamp falloff type does not attenuate its intensity with distance. This is useful for distant light sources like the sun or sky, which are so far away that their falloff isn't noticeable. *Sun* and *Hemi* lamps always have constant falloff.

**Sphere** The *Sphere* option restricts the light illumination range of a *Lamp* or *Spot* lamp, so that it will completely stop illuminating an area once it reaches the number of Blender Units away from the Lamp, as specified in the *Dist* field.

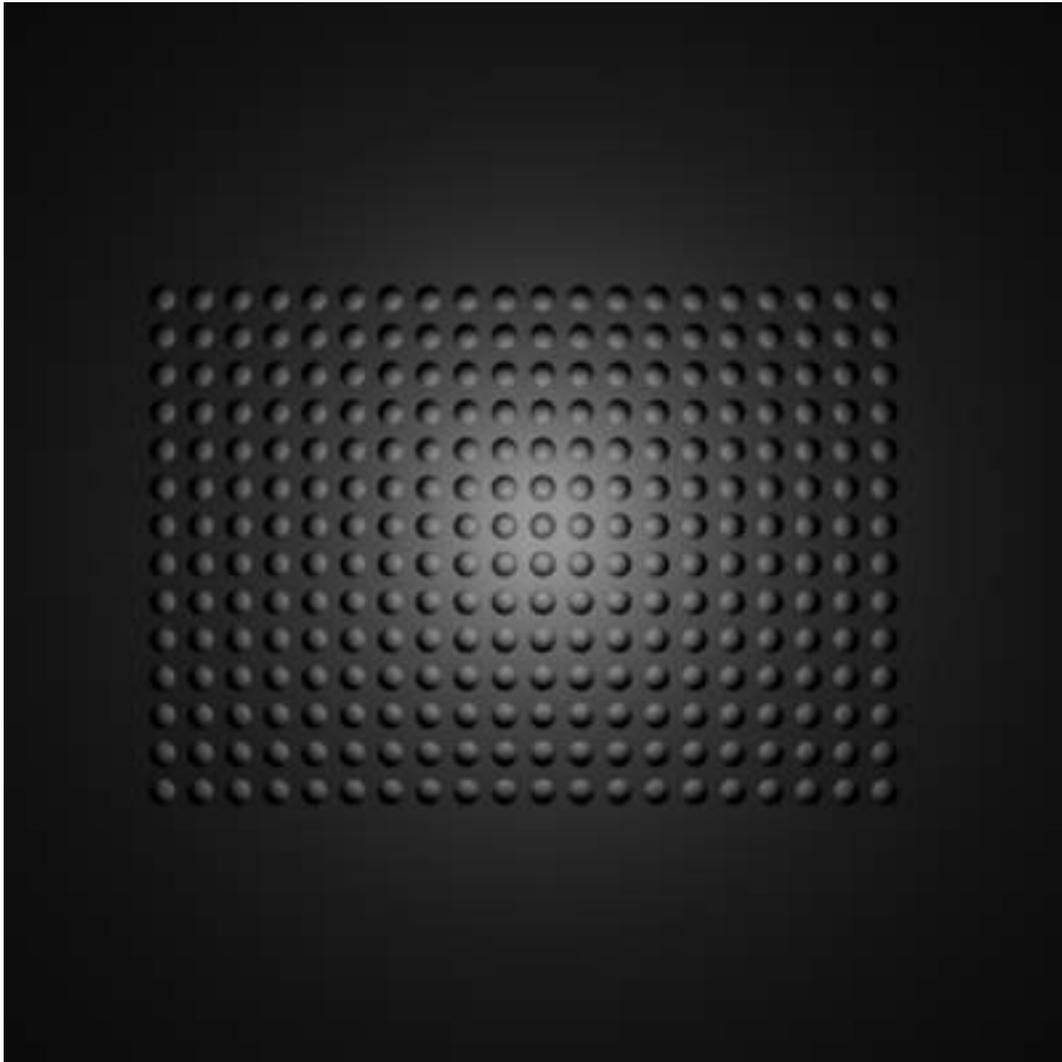


Fig. 2.2032: Render showing the Inverse Linear lamp falloff type effect with default settings.

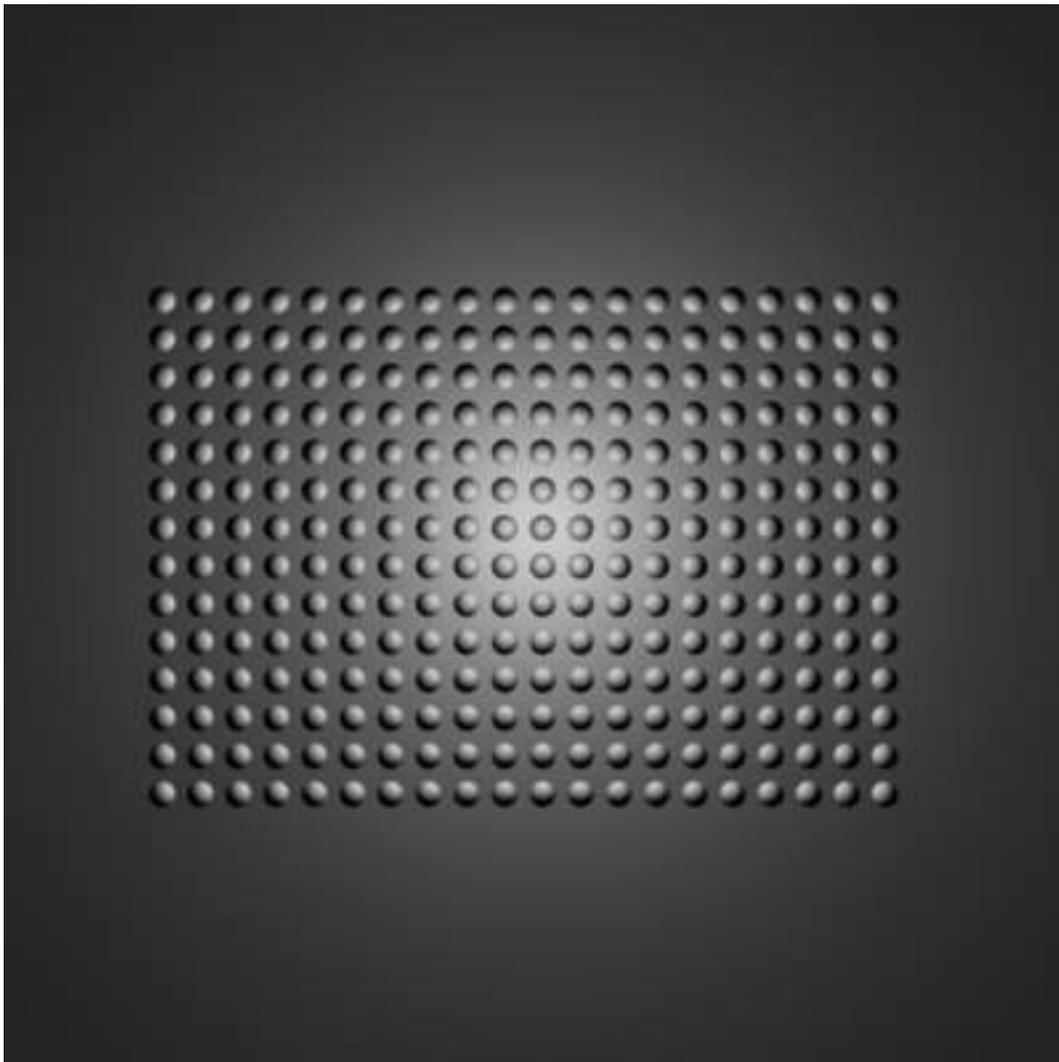


Fig. 2.2033: Render showing the Constant lamp falloff type effect with default settings.

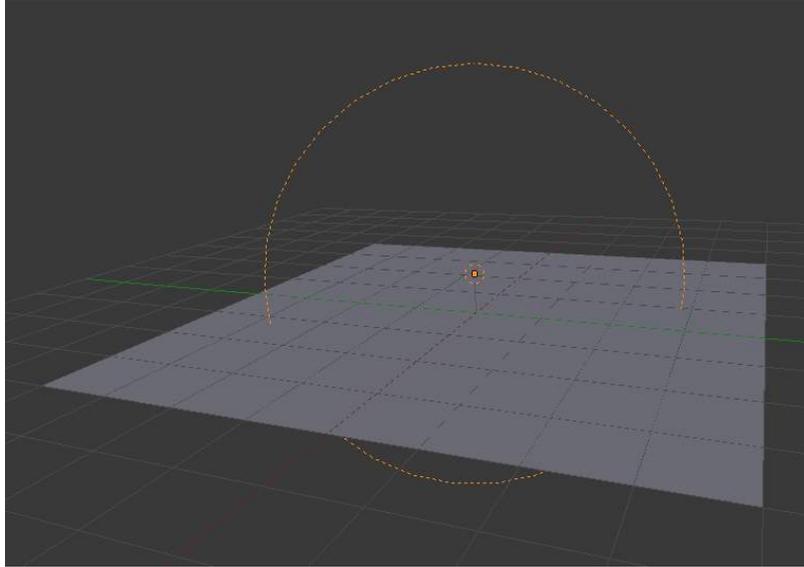


Fig. 2.2034: Screenshot of the 3D view window, showing the Sphere light clipping circle.

When the *Sphere* option is active, a dotted sphere will appear around the light source, indicating the demarcation point at which this light intensity will be null.

The *Sphere* option adds a term to the chosen attenuation law, whatever it is:

$$I' = I \times (D - r) / D \text{ if } r < D; 0 \text{ otherwise}$$

Where:

- $I'$  is the required Intensity of light (with the *Sphere* option activated).
- $I$  is the intensity of light calculated by the chosen attenuation law (without the *Sphere* option).
- $D$  is the current setting of the *Dist* field.
- $r$  is the distance from the lamp where the light intensity gets measured.

See the graphic at the end of the description of the *Lin/Quad Weighted* attenuation option.



## Examples

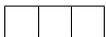
**Distance Example** In this example, the *Lamp* has been set pretty close to the group of planes. This causes the light to affect the front, middle and rear planes more dramatically. Looking at (*Various Distance settings*), you can see that as the *Dist* is increased, more and more objects become progressively brighter.

Table

2.55:

Distance:

1000.



The *Distance* parameter is controlling where the light is falling - at a linear rate by default - to half its original value from the light's origin. As you increase or decrease this value, you are changing where this half falloff occurs. You could think of *Distance* as the surface of a sphere and the surface is where the light's intensity has fallen to half its strength in all directions.

Note that the light's intensity continues to fall even after *Distance*. *Distance* just specifies the distance where half of the light's energy has weakened.

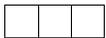
Notice in (*Distance* : 1000) that the farthest objects are very bright. This is because the falloff has been extended far into the distance, which means the light is very strong when it hits the last few objects. It is not until 1000 units that the light's intensity has fallen to half of its original intensity.

Contrast this with (*Distance* : 10), where the falloff occurs so soon that the farther objects are barely lit. The light's intensity has fallen by a half by time it even reaches the tenth object.

You may be wondering why the first few planes appear to be dimmer? This is because the surface angle between the light and the object's surface normal is getting close to oblique. That is the nature of a *Lamp* light object. By moving the light infinitely far away you would begin to approach the characteristics of the *Sun* lamp type.

**Inverse Square Example** *Inverse Square* makes the light's intensity falloff with a non-linear rate, or specifically, a quadratic rate. The characteristic feature of using *Inverse Square* is that the light's intensity begins to fall off very slowly but then starts falling off very rapidly. We can see this in the (*Inverse Square selected*) images.

Table  
2.56:  
Inverse  
Square  
with  
1000.



With *Inverse Square* selected, the *Distance* field specifies where the light begins to fall off faster, roughly speaking; see the light attenuation description in [Falloff types](#) for more info.

In (*Inverse Square with 10*), the light's intensity has fallen so quickly that the last few objects aren't even lit.

Both (*Inverse Square with 100*) and (*Inverse Square with 1000*) appear to be almost identical and that is because the *Distance* is set beyond the farthest object's distance which is at about **40 BU** out. Hence, all the objects get almost the full intensity of the light.

As above, the first few objects are dimmer than farther objects because they are very close to the light. Remember, the brightness of an object's surface is also based on the angle between the surface normal of an object and the ray of light coming from the lamp.

This means there are at least two things that are controlling the surface's brightness: intensity and the angle between the light source and the surface's normal.

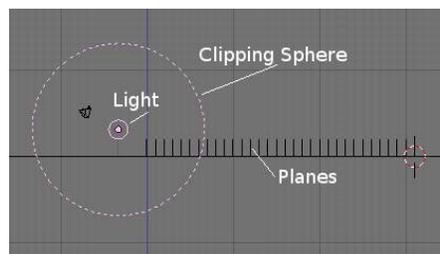


Fig. 2.2051: Clipping Sphere.

**Sphere Example** *Sphere* indicates that the light's intensity is null at the *Distance* distance and beyond, regardless of the chosen light's falloff. In (*Clipping Sphere*) you can see a side view example of the setup with *Sphere* enabled and a distance of 10.

Any objects beyond the sphere receive no light from the lamp.

The *Distance* field is now specifying both where the light's rays become null, and the intensity's ratio falloff setting. Note that there is no abrupt transition at the sphere: the light attenuation is progressive (for more details, see the descriptions of the *Sphere* and *Falloff types* above).

Table

2.57:

Sphere  
with 40.



In (*Sphere with 10*), the clipping sphere's radius is 10 units, which means the light's intensity is also being controlled by 10 units of distance. With a linear attenuation, the light's intensity has fallen very low even before it gets to the first object.

In (*Sphere with 20*), the clipping sphere's radius is now **20 BU** and some light is reaching the middle objects.

In (*Sphere with 40*), the clipping sphere's radius is now 40 units, which is beyond the last object. However, the light doesn't make it to the last few objects because the intensity has fallen to nearly 0.

---

**Hint:** If a *Lamp* light is set to not cast shadows, it illuminates through walls and the like. If you want to achieve some nice effects like a fire, or a candle-lit room interior seen from outside a window, the *Sphere* option is a must. By carefully working on the *Distance* value you can make your warm firelight shed only within the room, while illuminating outside with a cool moonlight, the latter achieved with a *Sun* or *Hemi* light or both.

---

**Lamps Textures** When a new lamp is added, it produces light in a uniform, flat color. While this might be sufficient in simple renderings, more sophisticated effects can be accomplished through the use of *textures*. Subtle textures can add visual nuance to a lamp, while hard textures can be used to simulate more pronounced effects, such as a disco ball, dappled sunlight breaking through treetops, or even a projector. These textures are assigned to one of ten channels, and behave exactly like material textures, except that they affect a lamp's color and intensity, rather than a material's surface characteristics.

**Options** The lamp textures settings are grouped into two panels. Here we will only talk about the few things that differ from object material textures; see the *Materials* and *Textures* chapters for details about the standard options.

The texture-specific and the *Mapping* panels remain the same. However, you'll note there are much fewer *Mapping* options - you can only choose between *Global*, *View* or another *Object*'s texture *coordinates* (since a lamp has no texture coordinates by itself), and you can scale or offset the texture.

The *Mapping* panel is also a subset of its regular material's counterpart. You can only map a lamp texture to its regular, basic *Color* and/or to its *Shadow* color. As you can only affect colors, and a lamp has no texture coordinates on its own, the *Diffuse*, *Specular*, *Shading*, and *Geometry* options have disappeared.

**What the Light Affects** Every lamp has a set of switches that control which objects receive its light, and how it interacts with materials.

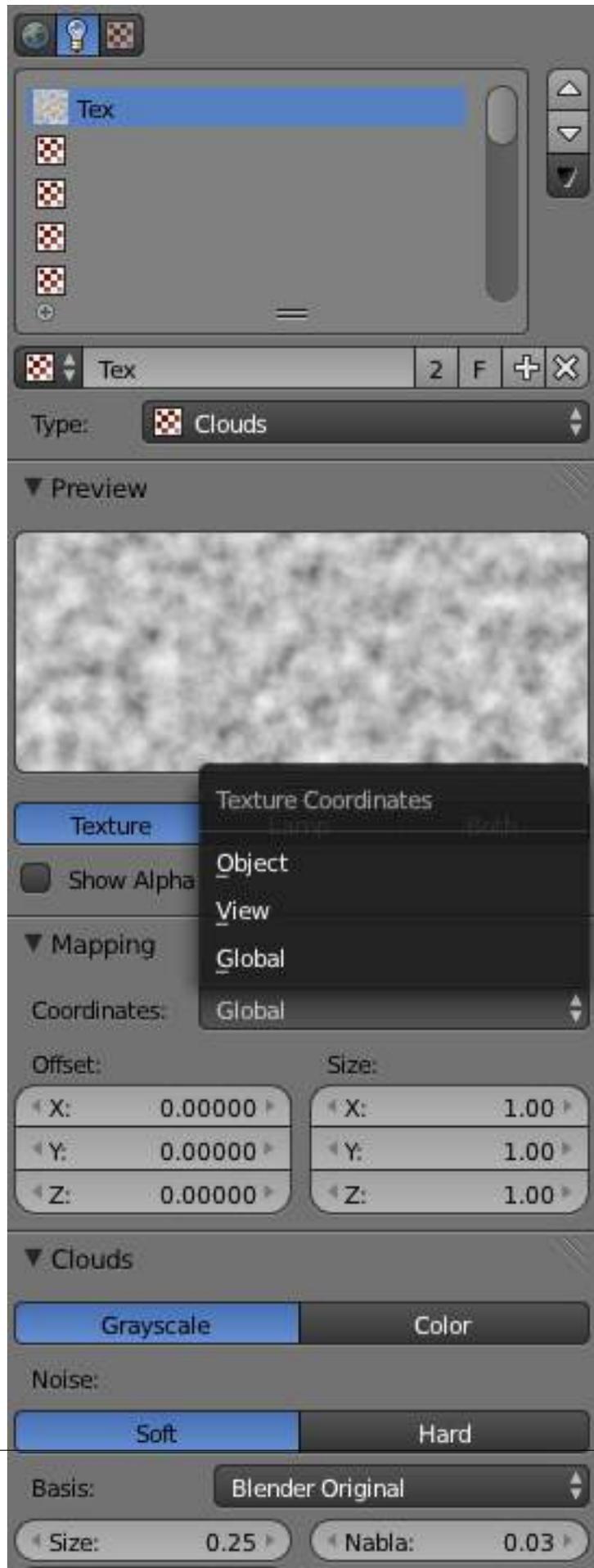
**Negative** The light produced by the lamp is **subtracted** from the one "available" on the surfaces it hits, which darkens these surfaces instead of brightening them.

**This Layer Only** Causes the lamp to only light objects on the same layer.

**Diffuse** Prevents the lamp from producing *diffuse light* (it doesn't really "light" things).

**Specular** Prevents the lamp from producing *specular highlights*.

**Lamps Related Settings** Here are some options closely related to light sources, without being lamps settings.



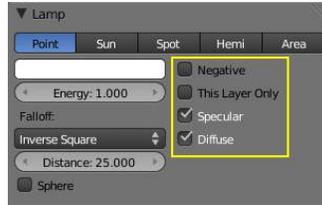


Fig. 2.2059: Lamp panel with the light affecting options highlighted

## Lighting Groups

**Materials** By default, materials are lit by all lamps in all visible layers, but a material (and thus all objects using that material) can be limited to a single group of lamps. This sort of control can be incredibly useful, especially in scenes with complex lighting setups. To enable this, navigate to the *Material* menu's *Options* panel and select a group of lamps in the *Light Group* field. Note that a [light group](#) must be created first.

If the *Exclusive* button is enabled, lights in the specified group will *only* affect objects with this material.

**Render Layers** There's a similar control located in the *Layer panel* of the context [Render Layers](#). If a light group name is selected in this *Light* field, the scene will be lit exclusively by lamps in the specified group.

## See Also

- [Lamps Introduction](#)
- [Shadows](#)
- [Materials Introduction](#)

## Shadows

**Introduction** Light wouldn't even exist without its counterpart: shadows. Shadows are a darkening of a portion of an object because light is being partially or totally blocked from illuminating the object. They add contrast and volume to a scene; there is nearly no place in the real world without shadows, so to get realistic renders, you will need them. Blender supports the following kinds of shadows:

- [Lamps: Ray-traced Shadows](#)
- [Lamps: Buffered Shadows](#)
- [Ambient occlusion](#)
- [Indirect lighting](#)

Ambient occlusion really isn't a shadow based on light *per se*, but based on geometry. However, it does mimic an effect where light is prevented from fully and uniformly illuminating an object, so it is mentioned here. Also, it is important to mention ambient lighting, since increasing *Ambient* decreases the effect of a shadow.

You can use a combination of ray-traced and buffer shadows to achieve different results. Even within ray-traced shadows, different lamps cast different patterns and intensities of shadow. Depending on how you arrange your lamps, one lamp may wipe out or override the shadow cast by another lamp.

Shadows is one of those trifectas in Blender, where multiple things have to be set up in different areas to get results:

- The lamp has to cast shadows (ability and direction).

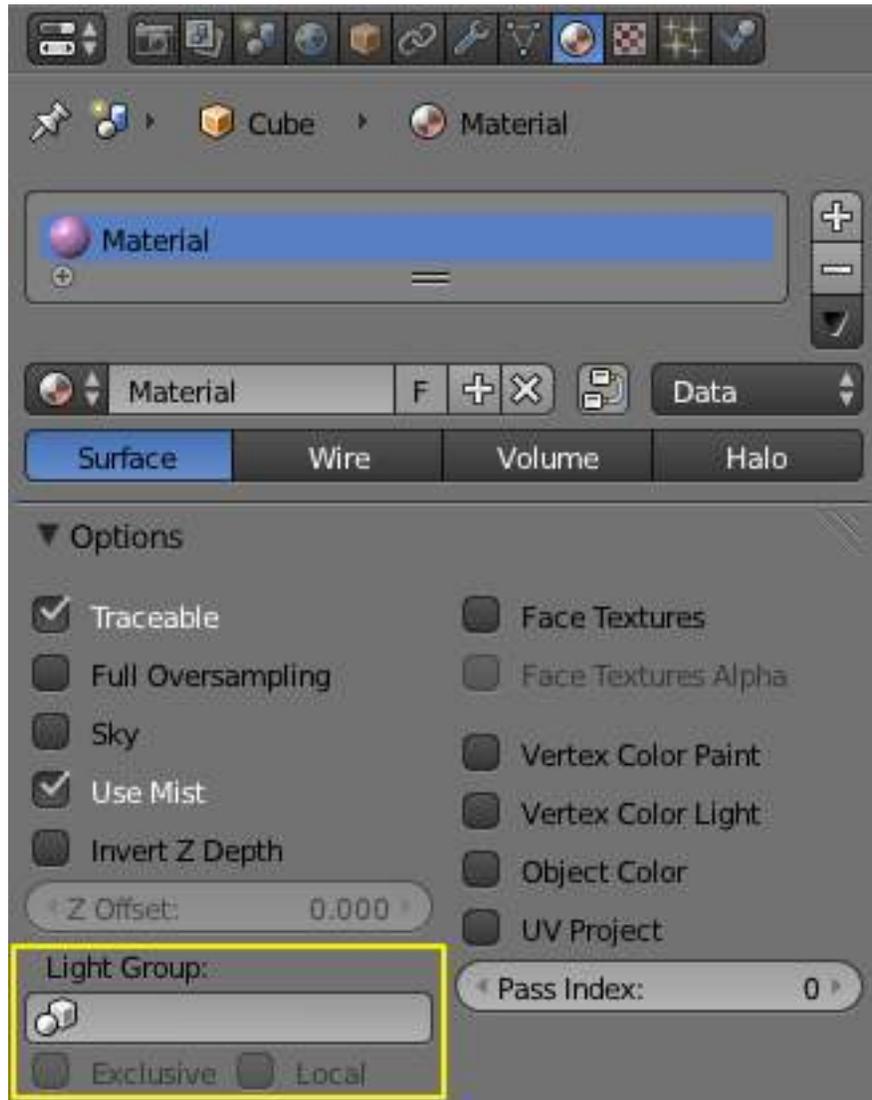


Fig. 2.2060: Light Group options for Materials

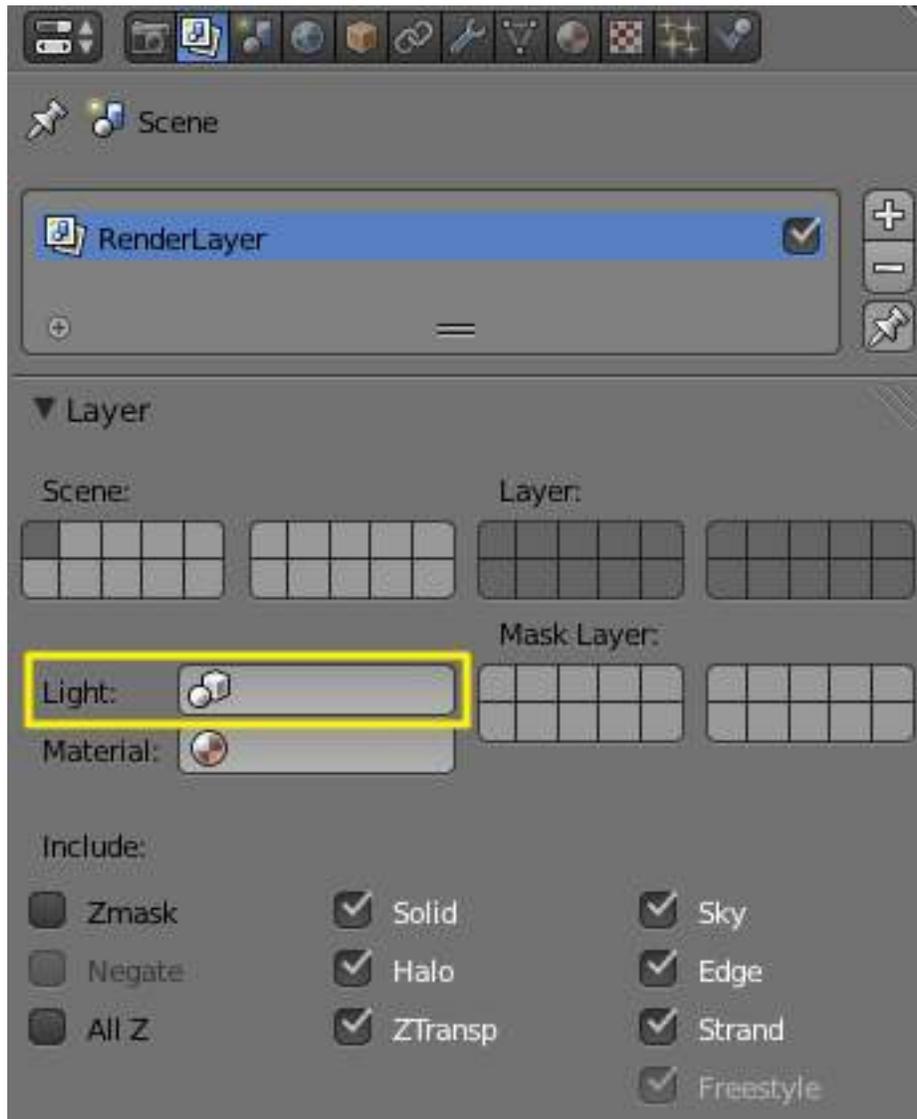


Fig. 2.2061: Light Group options for Render Layers

- An opaque object has to block light on its way (position and layer).
- Another object's material has to receive shadows (*Shadow* and *Receive Transparent* enabled).
- The render engine has to calculate shadows (*Shadow* for buffered shadows, *Shadow* and *Ray* for ray-traced shadows).

For example, the simple *Lamp*, *Area*, and *Sun* light has the ability to cast ray shadows, but not buffer shadows. The *Spot* light can cast both, whereas the *Hemi* light does not cast any. If a *Sun* lamp is pointing sideways, it will not cast a shadow from a sphere above a plane onto the plane, since the light is not traveling that way. All lamps able to cast shadows share some common options, described [here](#).

Just to give you more shadow options (and further confuse the issue), lamps and materials can be set to respectively **only** cast and receive shadows, and not light the diffuse/specular aspects of the object. Also, render layers can turn on/off the shadow pass, and their output may or may not contain shadow information...

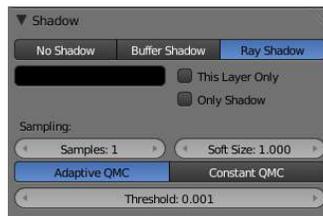


Fig. 2.2062: Ray Shadow enabled for a lamp

**Lamps: Ray-traced Shadows** Ray-traced shadows produce very precise shadows with very low memory use, but at the cost of processing time. This type of shadowing is available to all lamp types except *Hemi*.

As opposed to buffered shadows (*Lamps: Buffered Shadows*), ray-traced shadows are obtained by casting rays from a regular light source, uniformly and in all directions. The ray-tracer then records which pixel of the final image is hit by a ray light, and which is not. Those that are not are obviously obscured by a shadow.

Each light casts rays in a different way. For example, a *Spot* light casts rays uniformly in all directions within a cone. The *Sun* light casts rays from an infinitely distant point, with all rays parallel to the direction of the *Sun* light.

For each additional light added to the scene, with ray-tracing enabled, the rendering time increases. Ray-traced shadows require more computation than buffered shadows but produce sharp shadow borders with very little memory resource usage.

To enable ray-traced shadows, three actions are required:

- Enable *Shadows* globally in the *Render* menu's *Shading* panel.
- Enable *Ray tracing* globally from the same panel.
- Enable ray-traced shadows for the light using the *Ray Shadow* button in the *Light* menu's *Shadow* panel. This panel varies depending on the type of light.
  - All lamps able to cast ray-traced shadows share some common options, described in [Ray-traced Properties](#).

Ray-traced shadows can be cast by the following types of lamp:

- [Point lamp](#)
- [Spot lamp](#)
- [Area lamp](#)
- [Sun lamp](#)



Fig. 2.2063: Buffer Shadow enabled for a Spot lamp



Fig. 2.2064: Cast Buffer Shadows enabled for a material

**Lamps: Buffered Shadows** *Buffered* shadows provide fast-rendered shadows at the expense of precision and/or quality. Buffered shadows also require more memory resources as compared to ray tracing. Using buffered shadows depends on your requirements. If you are rendering animations or can't wait hours to render a complex scene with soft shadows, buffer shadows are a good choice.

For a scanline renderer - and Blender's built-in engine *is*, among other things, a scanline renderer - shadows can be computed using a *shadow buffer*. This implies that an "image", as seen from the spot lamp's point of view, is "rendered" and that the distance - in the image - for each point from the spot light is saved. Any point in the "rendered" image that is farther away than any of those points in the spot light's image is then considered to be in shadow. The shadow buffer stores this image data.

To enable buffered shadows these actions are required:

- Enable shadows globally from the *Scene* menu's *Gather* panel by selecting *Approximate*.
- Enable shadows for the light using the *Buffer Shadow* button in the *Lamp* menu's *Shadow* panel.
- Make sure the *Cast Buffer Shadows* options is enabled in each *Material*'s *Shadow* panel.
- The *Spot lamp* is the only lamp able to cast buffered shadows.

**Common Shadowing Lamps Options** All lamps able to cast shadows (*Lamp*, *Spot*, *Area* and *Sun*) share some options, described below:

**This Layer Only** When this option is enabled, only the objects on the same layer as the light source will cast shadows.

**Only Shadow** The light source will not illuminate an object but will generate the shadows that would normally appear. This feature is often used to control how and where shadows fall by having a light which illuminates but has no shadow, combined with a second light which doesn't illuminate but has *Only Shadow* enabled, allowing the user to control shadow placement by moving the "Shadow Only" light around.

**Shadow color** This color picker control allows you to choose the color of your cast shadows (black by default). The images below were all rendered with a white light and the shadow color was selected independently.



Although you can select a pure white color for a shadow color, it appears to make a shadow disappear.



Fig. 2.2065: Common shadowing options for lamps

**See Also**

- Shadows
- Common Raytraced Options
- Lamp Light Raytraced Shadows
- Spot Light Raytraced Shadows
- Area Light Raytraced Shadows
- Sun Light Raytraced Shadows
- Spot Light Buffered Shadows



Fig. 2.2072: Ray shadowing options for lamps

**Lamps Raytraced Shadows** Most lamp types ([Lamp](#), [Spot](#) and [Sun](#)) share the same options for the ray-traced shadows generation, which are described below. Note that the [Area](#) lamp, even though using most of these options, have some specifics described in its [own ray-traced shadows page](#).

**Ray Shadow** The *Ray Shadow* button enables the light source to generate ray-traced shadows. When the *Ray Shadow* button is selected, another set of options is made available, those options being:

**Shadow sample generator type** Method for generating shadow samples: Adaptive QMC is fastest, Constant QMC is less noisy but slower. This allows you to choose which algorithm is to be used to generate the samples that will serve to compute the ray-traced shadows (for now, mainly two variants of Quasi-Monte Carlo, see [What is Quasi-Monte Carlo?](#)):

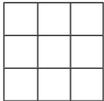
**Constant QMC** The *Constant QMC* method is used to calculate shadow values in a very uniform, evenly distributed way. This method results in very good calculation of shadow value but it is not as fast as using the *Adaptive QMC* method; however, *Constant QMC* is more accurate.

**Adaptive QMC** The *Adaptive QMC* method is used to calculate shadow values in a slightly less uniform and distributed way. This method results in good calculation of shadow value but not as good as *Constant QMC*. The advantage of using *Adaptive QMC* is that it is in general much quicker while being not much worse than *Constant QMC* in terms of overall results.

**Samples** Number of extra samples taken (samples x samples). This slider sets the maximum number of samples that both *Constant QMC* and *Adaptive QMC* will use to do their shadow calculations. The maximum value is 16 - the real number of samples is actually the square of it, so setting a sample value of 3 really means  $3^2 = 9$  samples will be taken.

**Soft Size** Light size for ray shadow sampling. This slider determines the size of the fuzzy/diffuse/penumbra area around the edge of a shadow. *Soft Size* only determines the width of the soft shadow size, not how graded and smooth the shadow is. If you want a wide shadow which is also soft and finely graded, you must also set the number of samples in the *Samples* field higher than 1; otherwise this field has no visible effect and the shadows generated will not have a soft edge. The maximum value for *Soft Size* is 100.0.

Below is a table of renders with different *Soft Size* and *Samples* settings showing the effect of various values on the softness of shadow edges:



Below is an animated version of the above table of images showing the effects:

Fig. 2.2091: Animated version renders with different *Soft Size* and *Samples* settings showing the effect of various values on the softness of shadow edges.

**Threshold** Threshold for Adaptive Sampling. This field is used with the *Adaptive QMC* shadow calculation method. The value is used to determine if the *Adaptive QMC* shadow sample calculation can be skipped based on a threshold of how shadowed an area is already. The maximum *Threshold* value is 1.0.

**What is Quasi-Monte Carlo?** The Monte Carlo method is a method of taking a series of samples/readings of values (any kind of values, such as light values, color values, reflective states) in or around an area at random, so as to determine the correct actions to take in certain calculations which usually require multiple sample values to determine overall accuracy of those calculations. The Monte Carlo method tries to be as random as possible; this can often cause areas that are being sampled to have large irregular gaps in them (places that are not sampled/read). This in turn can cause problems for certain calculations (such as shadow calculation).

The solution to this was the Quasi-Monte Carlo method.

The Quasi-Monte Carlo method is also random, but tries to make sure that the samples/readings it takes are also better distributed (leaving less irregular gaps in its sample areas) and more evenly spread across an area. This has the advantage of sometimes leading to more accurate calculations based on samples/reading.

## Volumetric Lighting

According to Wikipedia, **volumetric lighting** is a technique used in 3D computer graphics to add lighting effects to a rendered scene. It allows the viewer to see beams of light shining through the environment; seeing sunbeams streaming through an open window is an example of volumetric lighting, also known as God rays. The term seems to have been introduced from cinematography and is now widely applied to 3D modeling and rendering especially in the field of 3D gaming. In volumetric lighting, the light cone emitted by a light source is modeled as a transparent object and considered

*as a container of a “volume”: as a result, light has the capability to give the effect of passing through an actual three dimensional medium (such as fog, dust, smoke, or steam) that is inside its volume, just like in the real world.*

A classic example is the search light with a visible halo/shaft of light being emitted from it as the search light sweeps around.

By default Blender does not model this aspect of light. For example when Blender lights something with a *Spot* light, you see the objects and area on the floor lit but not the shaft/halo of light coming from the spotlight as it progresses to its target and would get scattered on the way.

The halo/shaft of light is caused in the real world by light being scattered by particles in the air, some of which get diverted into your eye and that you perceive as a halo/shaft of light. The scattering of light from a source can be simulated in Blender using various options, but by default is not activated.

The only lamp able to create volumetric effects is the [Spot lamp](#) (even though you might consider some of the “[Sky & Atmosphere](#)” effects of the *Sun* lamp as volumetric as well).

**Example** [Blend file of spotlight animation.](#)

#### See also

- [Mist](#)
- [Smoke](#)
- [Volumetric Materials](#)

## Lamps

**Introduction** Blender comes equipped with five different lamp types, each with its own unique strengths and limitations. Here are the available lamps:

- **Point** is an omni-directional point light source, similar to a light bulb.
- **Spot** is a directional point light source, similar to ... a spot.
- **Area** is a source simulating an area which is producing light, as windows, neons, TV screens.
- **Hemi** simulates a very wide and far away light source, like the sky.
- **Sun** simulates a very far away and punctual light source, like the sun.

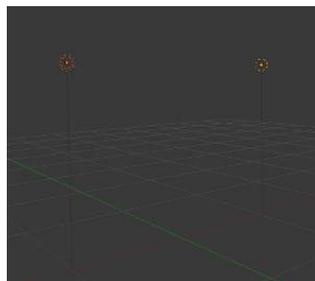


Fig. 2.2092: Visual height and shadow markers of two points lamps. Ray Shadow is enabled on the left lamp.

You can add new lamps to a scene using the *Add* menu in the top header, or with  $([Shift][A] \rightarrow Add \rightarrow Lamp)$ .

Once added, a lamp’s position is indicated in the 3D View by a solid dot in a circle, but most types also feature dashed wire-frames that help describe their orientation and properties. While each type is represented differently, there are some visual indicators common to all of them:

**Shadows** If shadows are enabled, an additional dashed circle is drawn around the solid circle. This makes it easier to quickly determine if a lamp has shadows enabled.

**Vertical Height Marker** This is a dim gray line, which helps locate the lamp's position relative to the global X-Y plane.

## Point

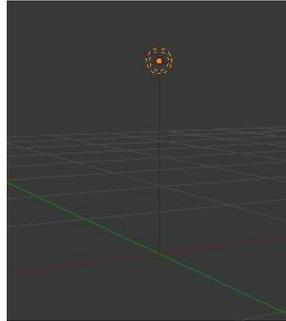


Fig. 2.2093: Point lamp

**Introduction** The *Point* lamp is an omni-directional point of light, that is, a point radiating the same amount of light in all directions. It's visualized by a plain, circled dot. Being a point light source, the direction of the light hitting an object's surface is determined by the line joining the lamp and the point on the surface of the object itself.

Light intensity/energy decays based on (among other variables) distance from the *Point* lamp to the object. In other words, surfaces that are further away are rendered darker.

## Lamp Options

**Distance, Energy and Color** These settings are common to most types of lamps, and are described in [Light Properties](#).

**Negative, This Layer Only, Specular, and Diffuse** These settings control what the lamp affects, as described in [What Light Affects](#).

**Falloff and Sphere** These settings control how the light of the *Lamp* decays with distance. See [Light Attenuation](#) for details.

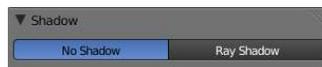


Fig. 2.2094: Without ray shadows



Fig. 2.2095: Point lamp with ray shadows and Adaptive QMC sample generator enabled

**Shadows** The *Point* light source can only cast ray-traced shadows. It shares with other lamp types the common shadow options described in [Shadow Properties](#).

The ray-traced shadows settings of this lamp are shared with other lamps, and are described [Raytraced Properties](#).



Fig. 2.2096: Shadow panel

**Raytraced Shadows** The *Point* light source can only cast raytraced shadows. It shares with other lamp types the same common shadowing options, described in [Shadows Properties](#).

The raytraced shadows settings of this lamp are shared with other ones, and are described in [Raytraced Properties](#).

## Spot

**Introduction** A *Spot* lamp emits a cone-shaped beam of light from the tip of the cone, in a given direction.

The *Spot* light is the most complex of the light objects and indeed, for a long time, among the most used thanks to the fact that it was the only one able to cast shadows. Nowadays, with a ray tracer integrated into Blender’s internal render engine, all lamps can cast shadows (except *Hemi*). Even so, *Spot* lamps’ shadow buffers are much faster to render than ray-traced shadows, especially when blurred/softened, and spot lamps also provide other functionality such as “volumetric” halos.

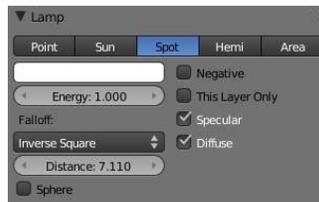


Fig. 2.2097: Common Lamp options of a Spot

### Lamp options

**Distance, Energy and Color** These settings are common to most types of lamps, and are described in [Light Properties](#).

**This Layer Only, Negative, Diffuse and Specular** These settings control what the lamp affects, as described in [What Light Affects](#).

**Light Falloff and Sphere** These settings control how the light of the *Spot* decays with distance. See [Light Attenuation](#) for details.

**Shadows** Spotlights can use either ray-traced shadows or buffered shadows. Either of the two can provide various extra options. Ray-traced shadows are generally more accurate, with extra capabilities such as transparent shadows, although they are quite slower to render.

**No Shadow** Choose this to turn shadows off for this spot lamp. This can be useful to add some discreet directed light to a scene.

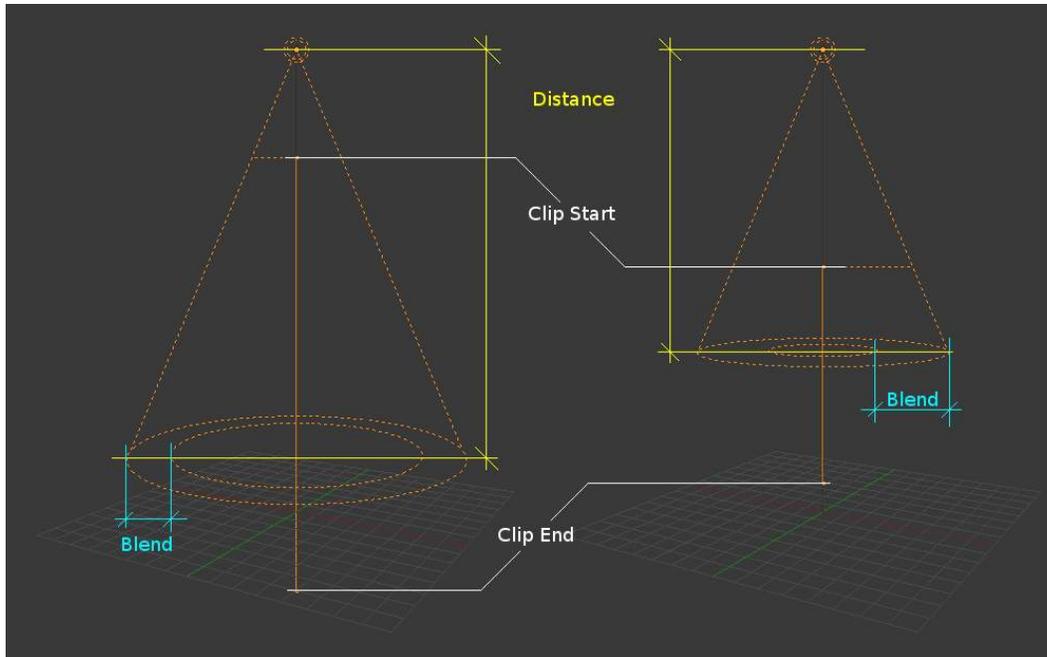


Fig. 2.2098: Changing the Spot options also changes the appearance of the spotlight as displayed in the 3D View



Fig. 2.2099: Shadow panel set to Ray Shadow

**Buffer Shadow** *Buffered Shadows* are also known as depth map shadows. Shadows are created by calculating differences in the distance from the light to scene objects. See [Buffered Shadows](#) for full details on using this feature. Buffered shadows are more complex to set up and involve more faking, but the speed of rendering is a definite advantage. Nevertheless, it shares with other lamp types common shadow options described in [Shadows Properties](#).

**Ray Shadow** The ray-traced shadows settings of this lamp are shared with other lamps, and are described in [Raytraced Properties](#).

### Spot Shape *Size*

The size of the outer cone of a *Spot*, which largely controls the circular area a *Spot* light covers. This slider in fact controls the angle at the top of the lighting cone, and can be between 1.0–° and °180.0.

Table  
2.58:  
Changing  
the spot  
Size  
option



**Blend** The *Blend* slider controls the inner cone of the *Spot*. The *Blend* value can be between 0.0 and 1.0. The value is proportional and represents that amount of space that the inner cone should occupy inside the outer cone (*Size*).

The inner cone boundary line indicates the point at which light from the *Spot* will start to blur/soften; before this point its light will mostly be full strength. The larger the value of *Blend* the more blurred/soft the edges of the spotlight will be, and the smaller the inner cone’s circular area will be (as it starts to blur/soften earlier).

To make the *Spot* have a sharper falloff rate and therefore less blurred/soft edges, decrease the value of *Blend*. Setting *Blend* to 0.0 results in very sharp spotlight edges, without any transition between light and shadow.

The falloff rate of the *Spot* lamp light is a ratio between the *Blend* and *Size* values; the larger the circular gap between the two, the more gradual the light fades between *Blend* and *Size*.

*Blend* and *Size* only control the *Spot* light cone’s aperture and softness (“radial” falloff); they do not control the shadow’s softness as shown below.

Notice in the picture above that the object’s shadow is sharp as a result of the ray tracing, whereas the spotlight edges are soft. If you want other items to cast soft shadows within the *Spot* area, you will need to alter other shadow settings.

**Square** The *Square* button makes a *Spot* light cast a square light area, rather than the default circular one.

**Show Cone** Draw a transparent cone in 3D view to visualize which objects are contained in it.

**Halo** Adds a volumetric effects to the spot lamp. See [Spot Halos](#).

**Raytraced Shadows** The *Spot* light source can only cast raytraced shadows. It shares with other lamp types the same common shadowing options, described in [Shadows Properties](#).

The raytraced shadows settings of this lamp are shared with other ones, and are described in [Raytraced Properties](#).

**Spot Buffered Shadows** Spotlights can use either `Raytraced Shadows` or buffered shadows. Either of the two can provide various extra options.

Raytraced shadows are generally more accurate, with extra capabilities such as transparent shadows, although they are quite slower to render.

Buffered shadows are more complex to set up and involve more faking, but the speed of rendering is a definite advantage. Nevertheless, it shares with other lamp types common shadows options described in [Shadows Properties](#).

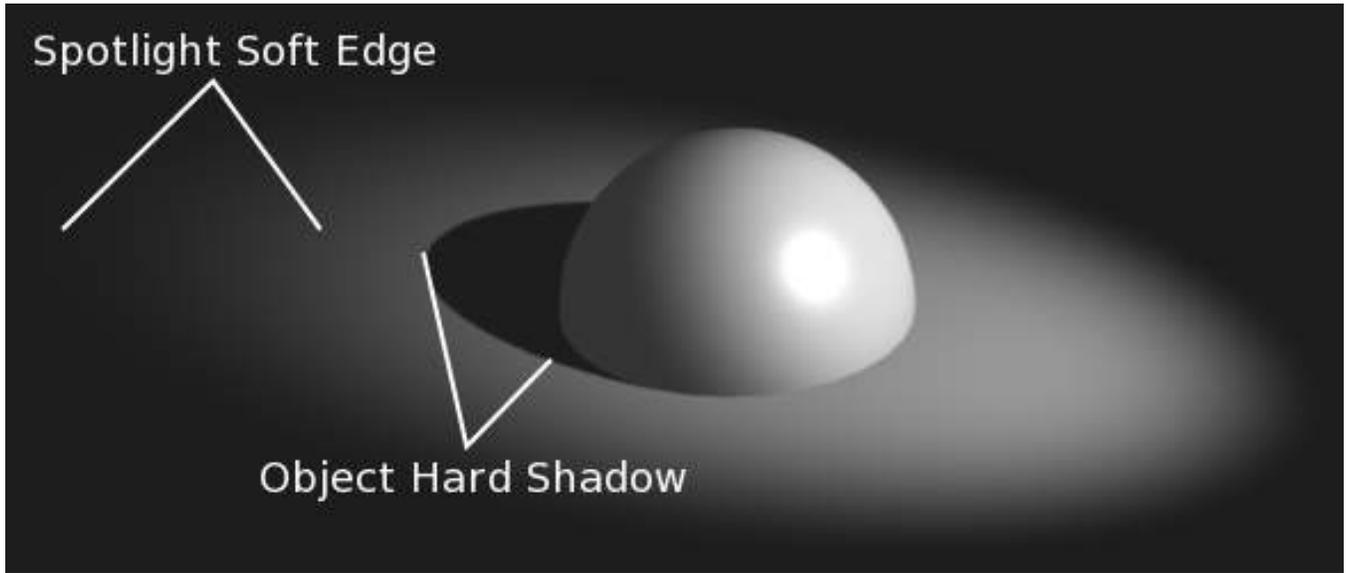


Fig. 2.2100: Render showing the soft edge spotlighted area and the sharp/hard object shadow



Fig. 2.2101: Shadow panel

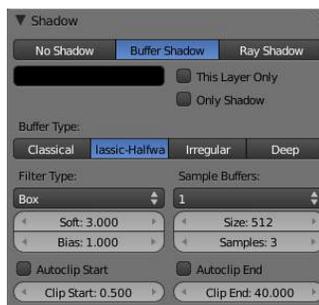


Fig. 2.2102: Buffer Shadow enabled for a Spot lamp

**Shadow Buffer Types** When the *Buffer Shadow* button is activated, the currently selected *Spot* light generates shadows, using a “shadow buffer” rather than using raytracing, and various extra options and buttons appear in the *Shadow* panel.

**Buffer Type** There more than one way to generate buffered shadows. The shadow buffer generation type controls which generator to use.

There are four shadow generation types, those being:

- Classical
- Classic-Halfway
- Irregular
- Deep

For more information on the different shadow generation methods see these links:

- [Development Release Logs 2.43: Irregular Shadow Buffer](#)
- [Blender Nation: Blender Gets Irregular Shadow Buffers](#)
- [Development Release Logs 2.43: Shadow Buffer Halfway Average](#)

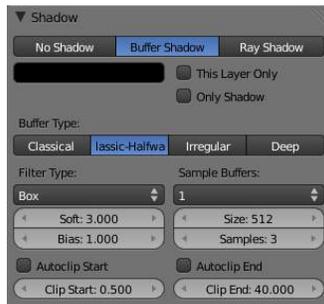


Fig. 2.2103: Buffer Shadowset to Classic-Halfway

### “Classical” and “Classic-Halfway”

**Classical** A shadow generation which used to be the Blender default and unique method for generation of buffered shadows. It used an older way of generating buffered shadows, but it could have some problems with accuracy of the generated shadows and can be very sensitive to the resolution of the shadow buffer (*Shadow Buffer* → *Size*), different *Bias* values, and all the self-shadowing issues that brings up.

The *Classical* method of generating shadows is obsolete and is really only still present to allow for backward compatibility with older versions of Blender. In most other cases you will want to use *Classic-Halfway* instead.

**Classic-Halfway** This shadow buffer type is an improved shadow buffering method and is the default option selected in Blender. It works by taking an averaged reading of the first and second nearest Z depth values allowing the *Bias* value to be lowered and yet not suffer as much from self-shadowing issues.

Not having to increase *Bias* values helps with shadow accuracy, because large *Bias* values can mean small faces can lose their shadows, as well as preventing shadows being overly offset from the larger *Bias* value.

*Classic-Halfway* doesn’t work very well when faces overlap, and biasing problems can happen.

Here are now the options specific to these generation methods:

**Size** The *Size* numeric field can have a value from 512 to 10240. *Size* represents the resolution used to create a shadow map. This shadow map is then used to determine where shadows lay within a scene.

As an example, if you have a *Size* with a value of 1024, you are indicating that the shadow data will be written to a buffer which will have a square resolution of **1024×1024** pixels/samples from the selected spotlight.

The higher the value of *Size*, the higher resolution and accuracy of the resultant shadows, assuming all other properties of the light and scene are the same, although more memory and processing time would be used. The reverse is also true - if the *Size* value is lowered, the resultant shadows can be of lower quality, but would use less memory and take less processing time to calculate.

As well as the *Size* value affecting the quality of generated shadows, another property of *Spot* lamps that affects the quality of their buffered shadows is the angle of the spotlights lighted area (given in the *Spot Shape* panel's *Size* field).

As the spot shape *Size* value is increased, the quality of the cast shadows degrades. This happens because when the *Spot* lighted area is made larger (by increasing spot shape *Size*), the shadow buffer area has to be stretched and scaled to fit the size of the new lighted area.

The *Size* resolution is not altered to compensate for the change in size of the spotlight, so the quality of the shadows degrades. If you want to keep the generated shadows the same quality, as you increase the spot shape *Size* value, you also need to increase the buffer *Size* value.

---

**Note:** The above basically boils down to

If you have a spotlight that is large you will need to have a larger buffer *Size* to keep the shadows good quality. The reverse is true also - the quality of the generated shadows will usually improve (up to a point) as the *Spot* lamp covers a smaller area.

---

**Filter Type** The *Box*, *Tent*, and *Gauss* filter types control what filtering algorithm to use to anti-alias the buffered shadows.

They are closely related to the *Samples* numeric field, as when this setting is set to 1, shadow filtering is disabled, so none of these buttons will have any effect whatsoever.

**Box** The buffered shadows will be anti-aliased using the “box” filtering method. This is the original filter used in Blender. It is relatively low quality and is used for low resolution renders, as it produces very sharp anti-aliasing. When this filter is used, it only takes into account oversampling data which falls within a single pixel, and doesn't take into account surrounding pixel samples. It is often useful for images which have sharply angled elements and horizontal/vertical lines.

**Tent** The buffered shadows will be anti-aliased using the “tent” filtering method. It is a simple filter that gives sharp results, an excellent general purpose filtering method. This filter also takes into account the sample values of neighboring pixels when calculating its final filtering value.

**Gauss** The buffered shadows will be anti-aliased using the “Gaussian” filtering method. It produces a very soft/blurred anti-aliasing. As result, this filter is excellent with high resolution renders.

The *Anti-Aliasing* page in the *Render* chapter will give more information on the various filtering/distribution methods and their uses.

**Samples** The *Samples* numeric field can have a value between 1 and 16. It controls the number of samples taken per pixel when calculating shadow maps.

The higher this value, the more filtered, smoothed and anti-aliased the shadows cast by the current lamp will be, but the longer they will take to calculate and the more memory they will use. The anti-aliasing method used is determined by having one of the *Box*, *Tent* or *Gauss* buttons activated (see above).

Having a *Samples* value of 1 is similar to turning off anti-aliasing for buffered shadows.

**Soft** The *Soft* numeric field can have a value between 1.0 and 100.0. It indicates how wide an area is sampled when doing anti-aliasing on buffered shadows. The larger the *Soft* value, the more graduated/soft the area that is anti-aliased/softened on the edge of generated shadows.

**Sample Buffers** The *Sample Buffers* setting can be set to values 1, 4 or 9, and represents the number of shadow buffers that will be used when doing anti-aliasing on buffered shadows.

This option is used in special cases, like very small objects which move and need to generate really small shadows (such as strands). It appears that normally, pixel width shadows don't anti-alias properly, and that increasing *Buffer Size* doesn't help much.

So this option allows you to have a sort of extra sample pass, done above the regular one (the one controlled by the *Box / Tent / Gauss, Samples* and *Soft* settings).

The default 1 value will disable this option.

Higher values will produce a smoother anti-aliasing - but be careful: using a *Sample Buffers* of 4 will require four times as much memory and process time, and so on, as Blender will have to compute that number of sample buffers.

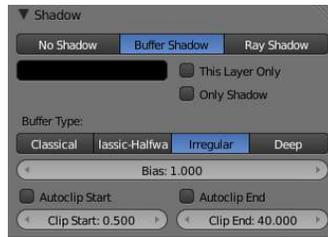


Fig. 2.2104: Buffer Shadow set to Irregular

**“Irregular”** *Irregular* shadow method is used to generate sharp/hard shadows that are placed as accurately as raytraced shadows. This method offers very good performance because it can be done as a multi-threaded process.

This method supports transparent shadows. To do so, you will first need to setup the shadow setting for the object which will receive the transparent shadow. (*Material* → *Shadow* → *Cat Buffer Shadows* and *Buffer Bias*)

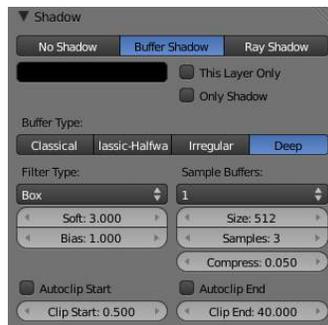


Fig. 2.2105: Buffer Shadow set to Deep

### Deep generation method

**Deep Shadow buffer supports transparency and better filtering , at the cost of more memory usage and processing time**

*Compress*: Deep shadow map compression threshold

**Common options** The following settings are common to all buffered shadow generation method.

**Bias** The *Bias* numeric field can have a value between 0.001 and 5.0. *Bias* is used to add a slight offset distance between an object and the shadows cast by it. This is sometimes required because of inaccuracies in the calculation which determines whether an area of an object is in shadow or not.

Making the *Bias* value smaller results in the distance between the object and its shadow being smaller. If the *Bias* value is too small, an object can get artifacts, which can appear as lines and interference patterns on objects. This problem is usually called “self shadowing”, and can usually be fixed by increasing the *Bias* value, which exists for that purpose!

Other methods for correcting self shadowing include increasing the size of the *Shadow Buffer Size* or using a different buffer shadow calculation method such as *Classic-Halfway* or *Irregular*.

Self shadowing interference tends to affect curved surfaces more than flat ones, meaning that if your scene has a lot of curved surfaces it may be necessary to increase the *Bias* value or *Shadow Buffer Size* value.

Having overly large *Bias* values not only places shadows further away from their casting objects, but can also cause objects that are very small to not cast any shadow at all. At that point altering *Bias*, *Shadow Buffer Size* or *Spot Size* values, among other things, may be required to fix the problem.

---

**Note:** Finer Bias tuning

You can now refine the *Bias* value independently for each [Material](#), using the *Bias* slider (*Material* menu, *Shadow* panel). This value is a factor by which the *Bias* value of each *Spot* buffered shadows lamp is multiplied, each time its light hits an object using this material. The 0.0 and 1.0 values are equivalent - they do not alter the lamp's *Bias* original value.

---

**Clip Start & Clip End** When a *Spot* light with buffered shadows is added to a scene, an extra line appears on the *Spot* 3D view representation.

The start point of the line represents *Clip Start*'s value and the end of the line represents *Clip End*'s value. *Clip Start* can have a value between 0.1 and 1000.0, and *Clip End*, between 1.0 and 5000.0. Both values are represented in Blender Units.

*Clip Start* indicates the point after which buffered shadows can be present within the *Spot* light area. Any shadow which could be present before this point is ignored and no shadow will be generated.

*Clip End* indicates the point after which buffered shadows will not be generated within the *Spot* light area. Any shadow which could be present after this point is ignored and no shadow will be generated.

The area between *Clip Start* and *Clip End* will be capable of having buffered shadows generated.

Altering the *Clip Start* and *Clip End* values helps in controlling where shadows can be generated. Altering the range between *Clip Start* and *Clip End* can help speed up rendering, save memory and make the resultant shadows more accurate.

When using a *Spot* lamp with buffered shadows, to maintain or increase quality of generated shadows, it is helpful to adjust the *Clip Start* and *Clip End* such that their values closely bound around the areas which they want to have shadows generated at. Minimizing the range between *Clip Start* and *Clip End*, minimizes the area shadows are computed in and therefore helps increase shadow quality in the more restricted area.

**Autoclip Start & Autoclip End** As well as manually setting *Clip Start* and *Clip End* fields to control when buffered shadows start and end, it is also possible to have Blender pick the best value independently for each *Clip Start* and *Clip End* field.

Blender does this by looking at where the visible vertices are when viewed from the *Spot* lamp position.

**Hints** Any object in Blender can act as a camera in the 3D view. Hence you can select the *Spot* light and switch to a view from its perspective by pressing `Ctrl-Numpad0`.



Fig. 2.2106: Spot lamps's Halo options

**Spot Volumetric Effects** *Spot* lights also can produce “volumetric” effects. See [Volumetric Light](#) for more information about what it means.

**Halo** The *Halo* button allows a *Spot* lamp to have a volumetric effect applied to it. This button must be active if the volumetric effect is to be visible. Note that if you are using buffered shadows, you have extra options described in the [Spot Buffered Shadows](#) page.

**Intensity** The *Intensity* slider controls how intense/dense the volumetric effect is that is generated from the light source. The lower the value of the *Intensity* slider, the less visible the volumetric effect is, while higher *Intensity* values give a much more noticeable and dense volumetric effect.

**Step** This field can have a value between 0 and 12. It is used to determine whether this *Spot* will cast volumetric shadows, and what quality those volumetric shadows will have. If *Step* is set to a value of 0, then no volumetric shadow will be generated. Unlike most other controls, as the *Step* value increases, the quality of volumetric shadows decreases (but take less time to render), and *vice versa*.

---

**Tip:** *Step* values

A value of 8 for *Halo Step* is usually a good compromise between speed and accuracy.

Blender only simulates volumetric lighting in *Spot* lamps when using its internal renderer. This can lead to some strange results for certain combinations of settings for the light's *Energy* and the halo's *Intensity*. For example, having a *Spot* light with null or very low light *Energy* settings but a very high halo *Intensity* setting can result in a dark/black halo, which would not happen in the real world. Just be aware of this possibility when using halos with the internal renderer.

---

**Note:** The halo effect can be greatly enhanced when using buffered shadows: when the halo's *Step* is not null, they can create "volumetric shadows". See the page about [Spot Buffered Shadows](#) for more information.

---

## See Also

- [Shadows](#)
- [Spot Lamp](#)
- [Spot Buffered Shadows](#)

## Area

**Introduction** The *Area* lamp simulates light originating from a surface (or surface-like) emitter. For example, a TV screen, your supermarket's neon lamps, a window, or a cloudy sky are just a few types of area lamp. The area lamp produces shadows with soft borders by sampling a lamp along a grid the size of which is defined by the user. This is in direct contrast to point-like artificial lights which produce sharp borders.

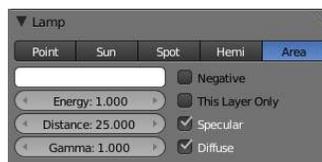


Fig. 2.2107: Commons Options

## Lamp options

**Distance, Energy and Color** These settings are common to most types of lamps, and are described in [Light Properties](#).

Note that the *Distance* setting is much more sensitive and important for *Area* lamps than for others; usually any objects within the range of *Distance* will be blown out and overexposed. For best results, set the *Distance* to just below the distance to the object that you want to illuminate.

**Gamma** Amount to gamma correct the brightness of illumination. Higher values give more contrast and shorter falloff.

The *Area* lamp doesn't have light falloff settings. It uses an "inverse quadratic" attenuation law. The only way to control its falloff is to use the *Distance* and/or *Gamma* settings.

**This Layer Only, Negative, Specular and Diffuse** These settings control what the lamp affects, as described in [What Light Affects](#).

**Shadows** Area light ray-traced shadows are described here: [Raytraced Shadows](#).

When an *Area* light source is selected, the *Shadow* panel has the following default layout:

Table 2.59: The Shadow panel when Area light source is selected.

Adaptive QMC settings	Constant Jittered settings

**Area Shape** The shape of the area light can be set to *Square* or *Rectangle*.

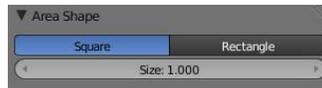


Fig. 2.2108: Square options

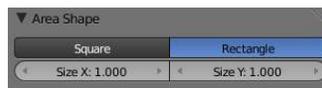


Fig. 2.2109: Rectangle options

**Square / Rectangular** Emit light from either a square or a rectangular area

**Size / Size X / Size Y** Dimensions for the *Square* or *Rectangle*

---

**Note:** Shape Tips

Choosing the appropriate shape for your *Area* light will enhance the believability of your scene. For example, you may have an indoor scene and would like to simulate light entering through a window. You could place a *Rectangular* area lamp in a window (vertical) or from neons (horizontal) with proper ratios for *Size X* and *Size Y*. For the simulation of the light emitted by a TV screen a vertical *Square* area lamp would be better in most cases.

---

**Area Raytraced Shadows** The *Area* light source can only cast ray-traced shadows. The ray-traced shadows settings of this lamp are mostly shared with other lamps, as described in [Raytraced Properties](#). However, there are some specifics with this lamp, which are detailed below:



Fig. 2.2110: Adaptive QMC settings

### Shadow Samples

**Samples** This has the same role as with other lamps, but when using a *Rectangle Area* lamp, you have two samples settings: *Samples X* and *Samples Y*, for the two axes of the area plane. Note also that when using the *Constant Jittered* sample generator method, this is more or less equivalent to the number of virtual lamps in the area. With QMC sample generator methods, it behaves similarly to with *Lamp* or *Spot* lamps.

### Sample Generator Types

**Adaptive QMC / Constant QMC** These common settings are described in [Shadow Properties](#).



Fig. 2.2111: Constant Jittered settings

**Constant Jittered** The *Area* lamp has a third sample generator method, *Constant Jittered*, which is more like simulating an array of lights. It has the same options as the old one: *Umbra*, *Dither* and *Jitter*.

The following three parameters are only available when using the *Constant Jittered* sample generator method, and are intended to artificially boost the “soft” shadow effect, with possible loss in quality:

**Umbra** Emphasizes the intensity of shadows in the area fully within the shadow rays. The light transition between fully shadowed areas and fully lit areas changes more quickly (i.e. a sharp shadow gradient). You need *Samples* values equal to or greater than 2 to see any influence of this button.

**Dither** Applies a sampling over the borders of the shadows, similar to the way anti-aliasing is applied by the *OSA* button on the borders of an object. It artificially softens the borders of shadows; when *Samples* is set very low, you can expect poor results, so *Dither* is better used with medium *Samples* values. It is not useful at all with high *Samples* values, as the borders will already appear soft.

**Jitter** Adds noise to break up the edges of solid shadow samples, offsetting them from each other in a pseudo-random way. Once again, this option is not very useful when you use high *Samples* values where the drawback is that noise generates quite visible graininess.

**Technical Details** The *(Principles behind the Area light)* picture helps to understand how the soft shadows are simulated.

(a) is the *Area* light as defined in Blender. If its shape is *Square*, then the softness of the shadow is defined by the number of light *Samples* in each direction of the shape. For example, (b) illustrates the equivalent case of an *Area* light (*Square* shape), with *Samples* set at 3 on the *Shadow and Spot* panel.

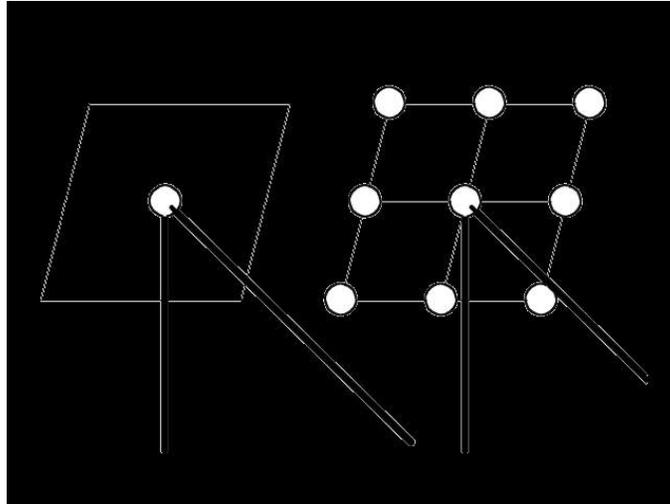


Fig. 2.2112: Principles behind the Area light

The *Area* lamp is then considered as a grid with a resolution of three in each direction, and with a light “duplivered” at each node for a total of nine lights.

In case (a), the energy ( $E$ ) is  $E/1$ , and in case (b), the energy of each individual pseudo-light is equal to  $E/9$  (Nbr of lights). Each pseudo-light produces a faint shadow (proportional to its energy), and the overlay of the shadows produces the soft shadow (it is darker where the individual shadows overlap, and lighter everywhere else).

**Hints** You will note that changing the *Size* parameter of your area lamp doesn’t affect the lighting intensity of your scene. On the other hand, rescaling the lamp using the *S* in the 3D View could dramatically increase or decrease the lighting intensity of the scene. This behavior has been coded this way so that you can fine tune all your light settings and then decide to scale up (or down) the whole scene without suffering from a drastic change in the lighting intensity. If you only want to change the dimensions of your *Area* lamp, without messing with its lighting intensity, you are strongly encouraged to use the *Size* button(s) instead.

If your computer isn’t very fast, when using the *Constant Jittered* sample generator method, you could find it useful to set a low *Samples* value (like 2) and activate *Umbra*, *Dither*, and/or *Jitter* in order to simulate slightly softer shadows. However, these results will never be better than the same lighting with high *Samples* values.

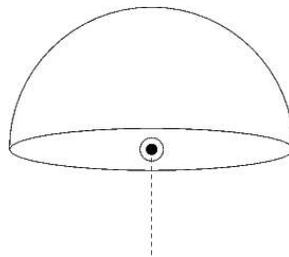


Fig. 2.2113: Hemi light conceptual scheme

**Hemi Lamp** The *Hemi* lamp provides light from the direction of a 180- hemisphere, designed to simulate the light coming from a heavily clouded or otherwise uniform sky. In other words, it is a light which is shed, uniformly, by a glowing dome surrounding the scene.

Similar to the *Sun* lamp, the *Hemi* ‘s location is unimportant, while its orientation is key.

The *Hemi* lamp is represented with four arcs, visualizing the orientation of the hemispherical dome, and a dashed line representing the direction in which the maximum energy is radiated, the inside of the hemisphere.

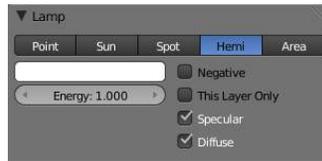


Fig. 2.2114: Hemi lamp's panel

## Options

**Energy and Color** These settings are common to most types of lamps, and are described in [Light Properties](#).

**Layer, Negative, Specular, and Diffuse** These settings control what the lamp affects, as described in [What Light Affects](#).

The *Hemi* lamp has no light falloff settings: it always uses a constant attenuation (i.e. no attenuation).

Since this lamp is the only lamp which cannot cast any shadow, the *Shadow* panel is absent.

## Sun

**Introduction** A *Sun* lamp provides light of constant intensity emitted in a single direction. A *Sun* lamp can be very handy for a uniform clear daylight open-space illumination. In the 3D view, the *Sun* light is represented by an encircled black dot with rays emitting from it, plus a dashed line indicating the direction of the light.

This direction can be changed by rotating the *Sun* lamp, like any other object, but because the light is emitted in a constant direction, the location of a *Sun* lamp does not affect the rendered result (unless you use the “sky & atmosphere” option).

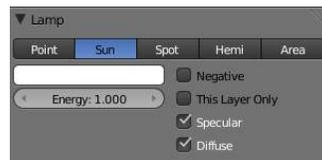


Fig. 2.2115: Sun lamp panel

## Lamp options

**Energy and Color** These settings are common to most types of lamps, and are described in [Light Properties](#).

**Negative, This Layer Only, Specular, and Diffuse** These settings control what the lamp affects, as described in [What Light Affects](#).

The *Sun* lamp has no light falloff settings: it always uses a constant attenuation (i.e. no attenuation!).

**Sky & Atmosphere** Various settings for the appearance of the sun in the sky, and the atmosphere through which it shines, are available. For details, see [Sky and Atmosphere](#).

**Shadow** The *Sun* light source can only cast ray-traced shadows. It shares with other lamp types the same common shadowing options, described in [Shadows Properties](#).

The ray-traced shadows settings of this lamp are shared with other lamps, and are described in [Raytraced Properties](#).



Fig. 2.2116: Sky &amp; Atmosphere panel



Fig. 2.2117: Shadow panel



Fig. 2.2118: Shadow panel

**Raytraced Shadows** The *Sun* light source can only cast raytraced shadows. It shares with other lamp types the same common shadowing options, described in [Shadows Properties](#).

The raytraced shadows settings of this lamp are shared with other ones, and are described in [Raytraced Properties](#).

**Sun: Sky & Atmosphere** This panel allows you to enable an effect that simulates various properties of real sky and atmosphere: the scattering of sunlight as it crosses the kilometers of air overhead. For example, when the Sun is high, the sky is blue (and the horizon, somewhat whitish). When the Sun is near the horizon, the sky is dark blue/purple, and the horizon turns orange. The dispersion of the atmosphere is also more visible when it is a bit foggy: the farther away an object is, the more “faded” in light gray it is... Go out into the countryside on a nice hot day, and you will see.

To enable this effect, you have to use a *Sun* light source. If, as usual, the *position* of the lamp has no importance, its *rotation* is crucial: it determines which hour it is. As a starting point, you should reset rotation of your *Sun* (with `Alt-R`, or typing 0 in each of the three *Rotation* fields *X / Y / Z* in the *Transform Properties* panel - *N*). This way, you’ll have a nice mid-day sun (in the tropics).

Now, there are two important angles for the *Sky/Atmosphere* effect: the “incidence” angle (between the light direction and the X-Y plane), which determines the “hour” of the day (as you might expect, the default rotation - straight down - is “mid-day”, a light pointing straight up is “midnight”, and so on...). And the rotation around the Z axis determines the position of the sun around the camera.



Fig. 2.2119: Sky &amp; Atmosphere panel

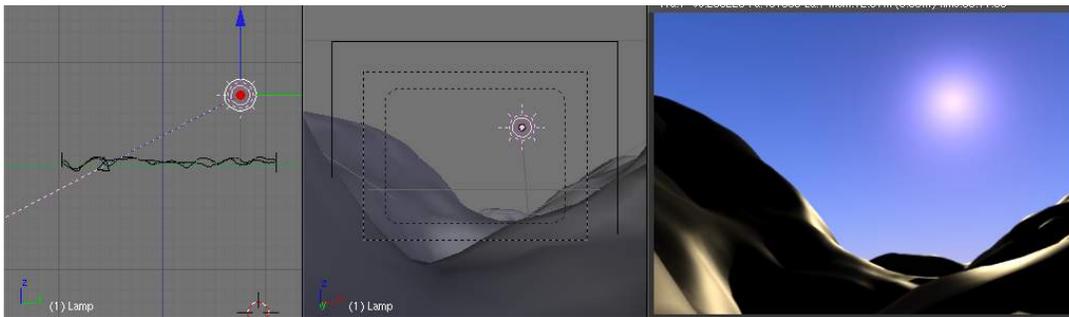


Fig. 2.2120: The dashed “light line” of the Sun lamp crossing the camera focal point.

In fact, to have a good idea of where the sun is in your world, relative to the camera in your 3D view, you should always try to have the dashed “light line” of the lamp crossing the center of the camera (its “focal” point), as shown in (*The dashed “light line” of the Sun lamp crossing the camera focal point*). This way, in camera view (Numpad0, center window in the example picture), you will see where the “virtual” sun created by this effect will be.

It is important to understand that the *position* of the sun has no importance for the effect: only its *orientation* is relevant. The position just might help you in your scene design.

#### Options *Sun & Sky Presets:*

- Classic
- Desert
- Mountain

#### Sky

**Sky** This button enables the sky settings: it will create a “sky”, with a “sun” if visible, and mix it with the background as defined in *World* settings.

**Turbidity** This is a general parameter that affects sun view, sky and atmosphere; it’s an atmosphere parameter where low values describe clear sky, and high values shows more foggy sky. In general, low values give a clear, deep blue sky, with “little” sun; high values give a more reddish sky, with a big halo around the sun. Note that this parameter is one which can really modify the “intensity” of the sun lighting. See examples below.

Here are its specific controls:

Blending

The first drop-down list shows you a menu of various mix methods. The one selected will be used to blend the sky and sun with the background defined in the *World* settings. The mixing methods are the same as described e.g. in the [Mix Compositing Node](#) page.

**Factor** Controls how much the sky and sun effect is applied to the World background.

**Color space** These buttons allows you to select which color space the effect uses, with the following choices:

- CIE
- REC709
- SMPTE
- Exposure

This numeric field allows you to modify the exposure of the rendered Sky and Sun (0.0 for no correction).

### Horizon

**Brightness** Controls brightness of colors at the horizon. Its value should be in the range 0.0 to 10.0; values near zero means no horizontal brightness, and large values for this parameter increase horizon brightness. See examples below.

**Spread** Controls spread of light at the horizon. Its value should be in the range 0.0 to 10.0; values low in the range result in less spread of light at horizon, and values high in the range result in horizon light spread in through all the sky.

### Sun

**Brightness** Controls the sun brightness. Its value should be in the range 0.0 to 10.0; with low values the sky has no sun and with high values the sky only has sun.

**Size** Controls the size of sun. Its values should be in the range 0.0 to 10.0, but note that low values result in large sun size, and high values result in small sun size. Note that the overall brightness of the sun remains constant (set by *Brightness*), so the larger the sun (the smaller *Size*), the more it “vanishes” in the sky, and *vice versa*.

**Back Light** For “Back Scatter Light”, result on sun’s color, high values result in more light around the sun. Its values range is -1.0 to 1.0. Negative values result in less light around sun.

### Atmosphere

**Atmosphere** This button enables the atmosphere settings. It will not modify the background, but it tries to simulate the effects of an atmosphere: scattering of the sunlight in the atmosphere, its attenuation, ...

### Intensity

**Sun** Sets sun intensity. Its values are in range 0.0 to 10.0. High values result in bluer light on far objects.

**Distance** This factor is used to convert Blender units into an understandable unit for atmosphere effect, it starts from 0 and high values result in more yellow light in the scene.

### Scattering

**Inscattering** This factor can be used to decrease the effect of light inscattered into atmosphere between the camera and objects in the scene. This value should be 1.0 but can be changed to create some nice, but not realistic, images.

**Extinction** This factor can be use to decrease the effect of extinction light from objects in the scene. Like *Inscattering* factor, this parameter should be 1.0 but you can change it; low values result in less light extinction. Its value is in the range 0.0 to 1.0.

**Examples** First, let's see what happens when we modify the orientation of the sun:

Table  
2.60: Sun  
slightly  
below the  
horizon  
(end of  
twilight).



The 2.4 .blend file of these examples.

And now, the effects of various settings (examples created with [this 2.4 .blend file](#)):

Table  
2.61:  
Turbidity:  
10.0.



**Sky** Table  
2.62:  
Horizon  
Bright-  
ness:  
1.13.



Table  
2.63:  
Horizon  
Spread:  
5.0.



Table  
2.64: Sun  
Bright-  
ness:  
1.0.



Table

2.65:

Sun Size:

10.0.



Table

2.66:

Back

Light:

1.0.



**Atmosphere** For all renders below, *Hor.Bright* is set to 0.2, and *Sun Bright* to 2.0.

Table

2.67: Sun

Intensity:

10.0.



Table

2.68:

Inscattering:

1.0.



Table

2.69:

Extinction:

1.0.



Table

2.70:

Distance:

4.0.



**Hints and limitations** To always have the *Sun* pointing at the camera center, you can use a [TrackTo constraint](#) on the sun object, with the camera as target, and -Z as the “To” axis (use either *X* or *Y* as “Up” axis). This way, to modify height/position of the sun in the rendered picture, you just have to move it; orientation is automatically handled by the constraint. Of course, if your camera itself is moving, you should also add e.g. a [Copy Location constraint](#) to your *Sun* lamp, with the camera as target - and the *Offset* option activated... This way, the sun light won’t change as the camera moves around.

If you use the default *Add* mixing type, you should use a very dark-blue world color, to get correct “nights”...

This effect works quite well with a *Hemi* lamp, or some ambient occlusion, to fill in the *Sun* shadows.

Atmosphere shading currently works incorrectly in reflections and refractions and is only supported for solid shaded surfaces. This will be addressed in a later release.

**Lighting Rigs** A rig is a standard setup and combination of objects; there can be lighting rigs, or armature rigs, etc. A rig provides a basic setup and allows you to start from a known point and go from there. Different rigs are used for different purposes and emulate different conditions; the rig you start with depends on what you want to convey in your scene. Lighting can be very confusing, and the defaults do not give good results. Further, very small changes can have a dramatic effect on the mood and colors.

In all the lighting rigs, the default camera is always positioned nearly 15 degrees off dead-on, about **25 BU** (Blender Units) back and **9 BU** to the side of the subject, at eye level, and uses a long lens (**80mm**). Up close, a **35mm** lens will distort the image. A long lens takes in more of the scene. A dead-on camera angle is too dramatic and frames too wide a scene to take in. So now you know; next time you go to a play, sit off-center and you won't miss the action happening on the sidelines and will have a greater appreciation for the depth of the set. Anyway, enough about camera angles; this is about lighting.

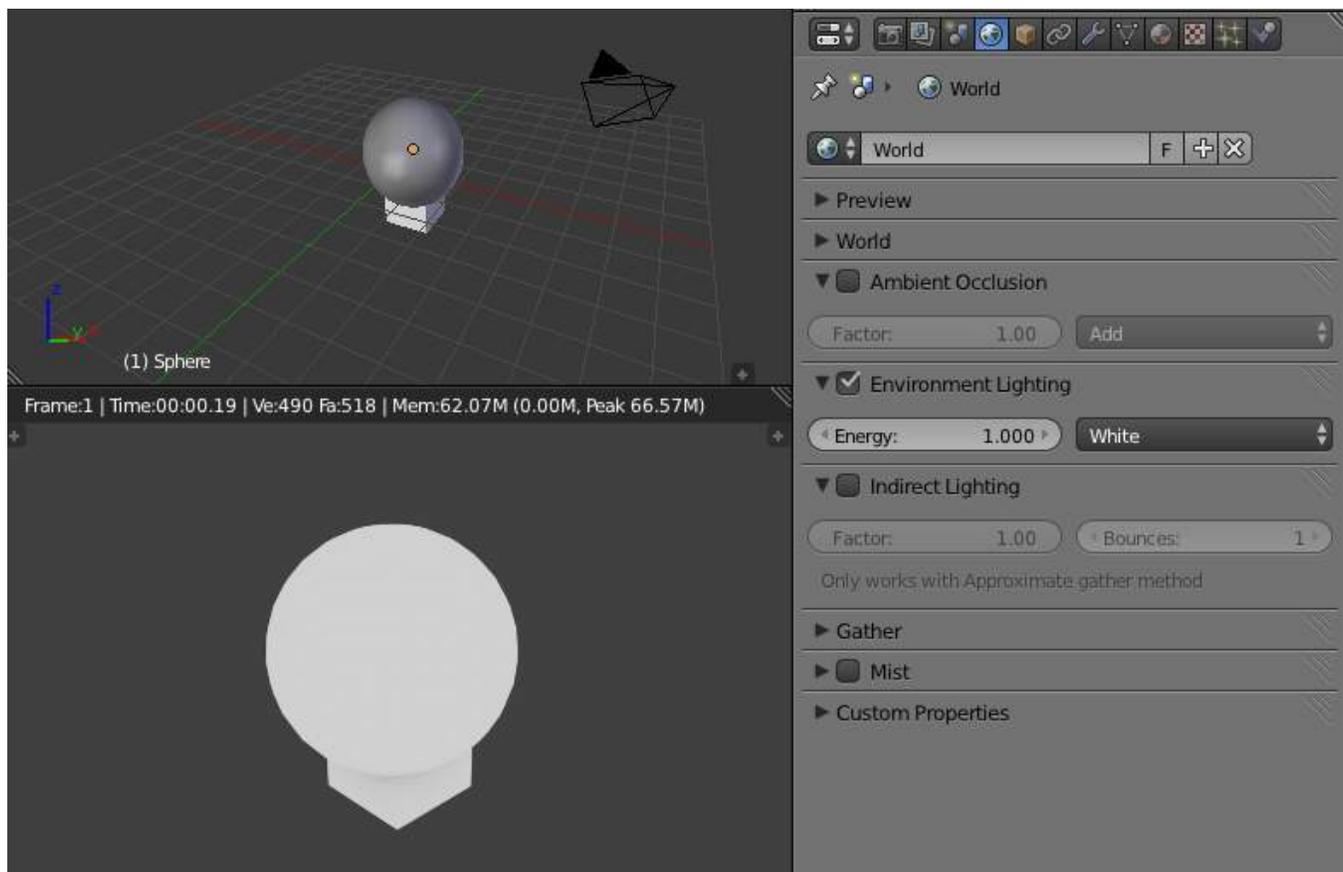


Fig. 2.2209: Environment (Ambient) lighting only.

**Environment or Ambient Only** In the *World* context, there is a panel *Environment Lighting*, where you enable environment or ambient lighting of your scene. Ambient light is the scattered light that comes from sunlight being reflected off every surface it hits, hitting your object, and traveling to camera.

Ambient light illuminates, in a perfectly balanced, shadeless way, without casting shadows. You can vary the intensity of the ambient light across your scene via [ambient occlusion](#). The ambient color is a sunny white.

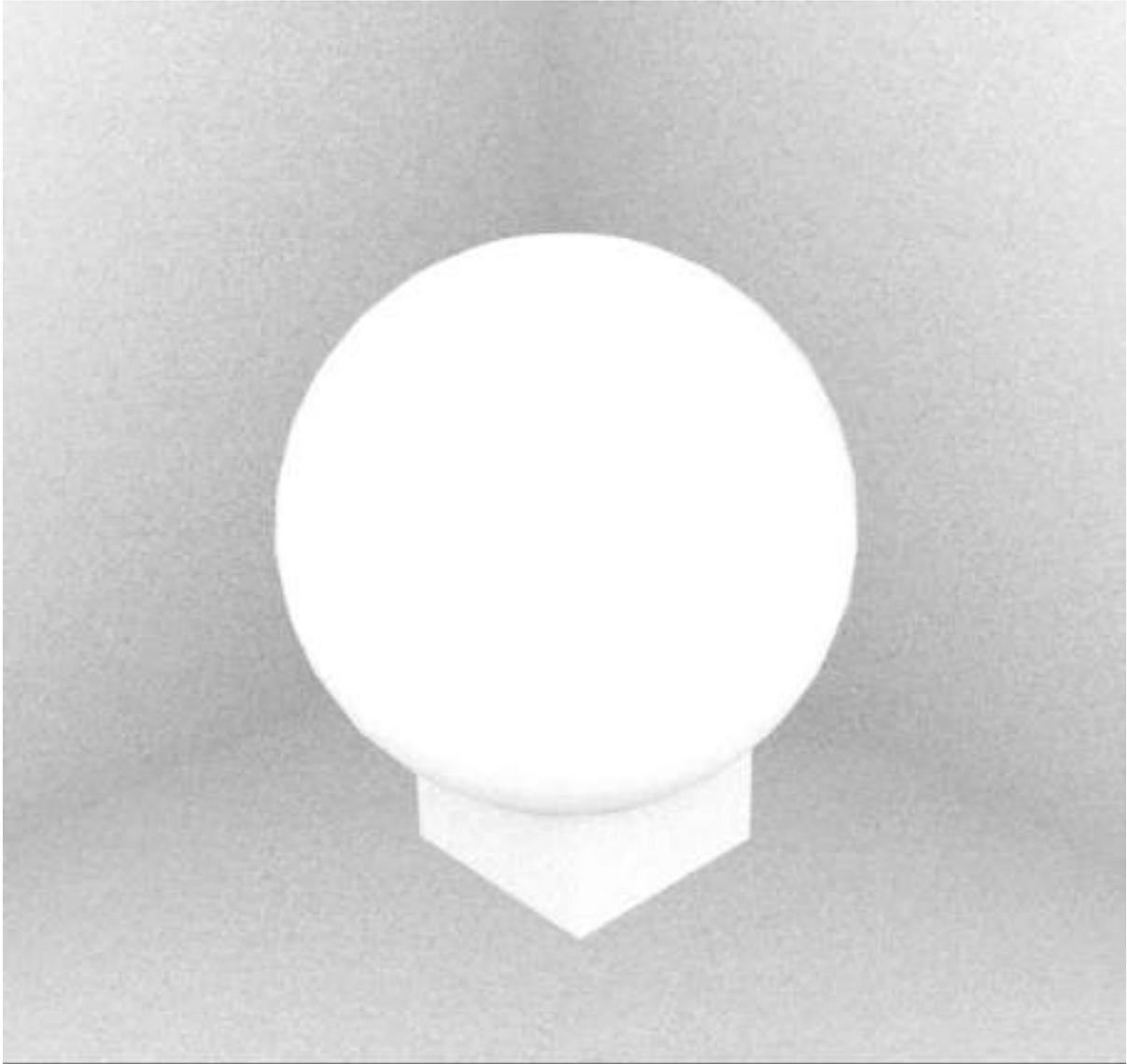


Fig. 2.2210: Ambient occlusion.

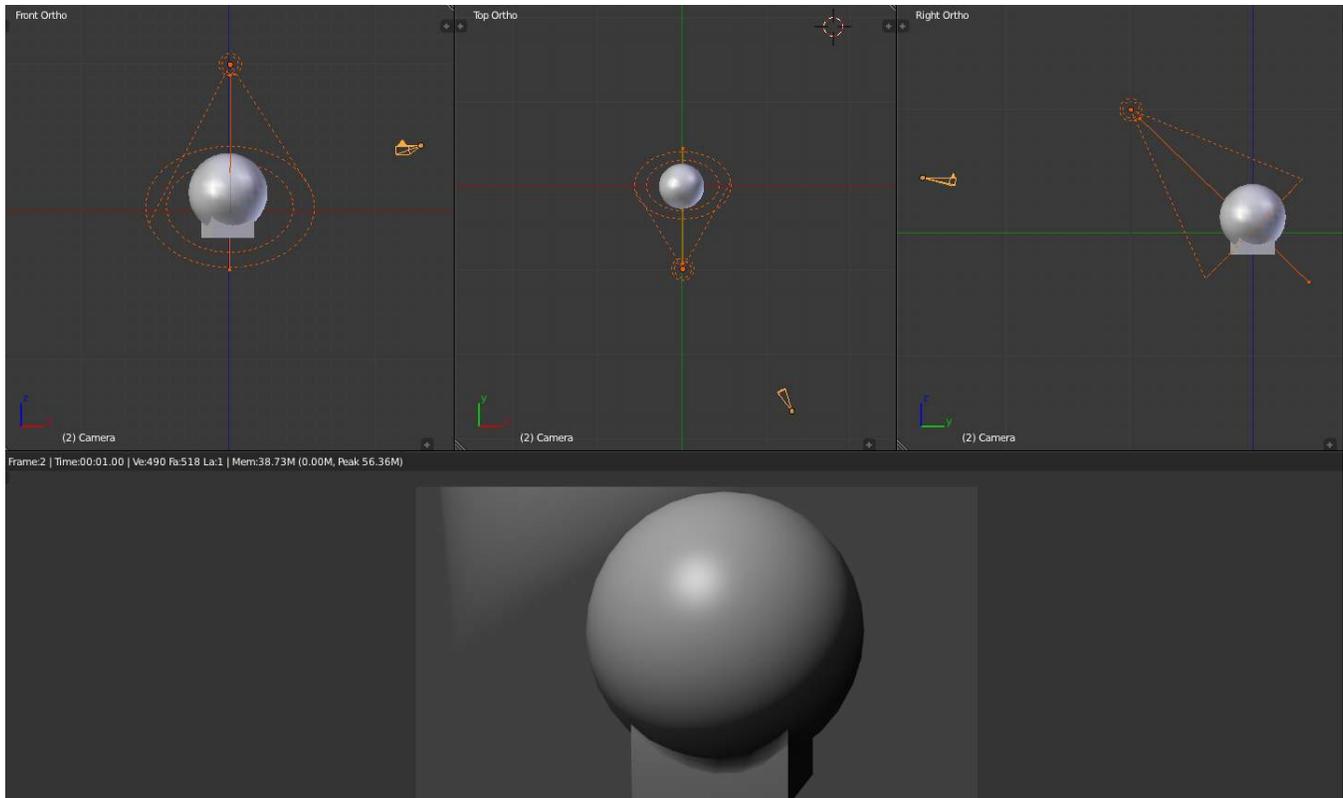


Fig. 2.2211: Standard Spot light rig.

**Single Rig** The sole, or key, spot light rig provides a dramatic, showy, yet effective illumination of one object or a few objects close together. It is a single *Spot* light, usually with a hard edge. Halos are enabled in this render to remind you of a smoky nightclub scene. It is placed above and directly in front of the subject; in this case **10 BU** in front and **10 BU** high, just like a stage, it shines down at about a 40 degrees angle. We use quadratic attenuation.

You can make the spot wider by increasing *Size Spot Shape* and softening the edge by increasing *Blend Spot Shape*, and parent it to the main actor, so that the spot follows him as he moves around. Objects close to the main actor will naturally be more lit and your viewer will pay attention to them.

Moving this spot directly overhead and pointing down gives the interrogation effect. At the opposite end of the show-off emotional spectrum is one soft candlelight (*Point* lamp, short falloff *Distance*, yellow light) placed really up close to the subject, dramatizing the fearful “lost in the darkness” effect.

Somewhere in the macabre spectrum is a hard spot on the floor shining upward. For fun, grab a flashlight, head into the bathroom and close the door. Turn out the light and hold the flashlight under your chin, pointing up. Look in the mirror and turn it on. Ghoulies! Don’t blame me for nightmares, and I hope you get the point: lighting, **even with a single light, varying the intensity, location and direction, changes everything** in a scene.

Use this rig, with *Environment Lighting* light (and props receiving and being lit by ambient light in their material settings) for scenes that feature one main actor or a product being spotlighted. Do not use this rig for big open spaces or to show all aspects of a model.

**Two-Point Rig** The two-point lighting rig provides a balanced illumination of an object. Shown to the right are the views of the standard two-point lighting rig. It is called the two-point because there are two points of light. The standard two-point lighting rig provides a balanced illumination of untextured objects hanging out there in 3D space. This rig is used in real studios for lighting a product, especially a glossy one.

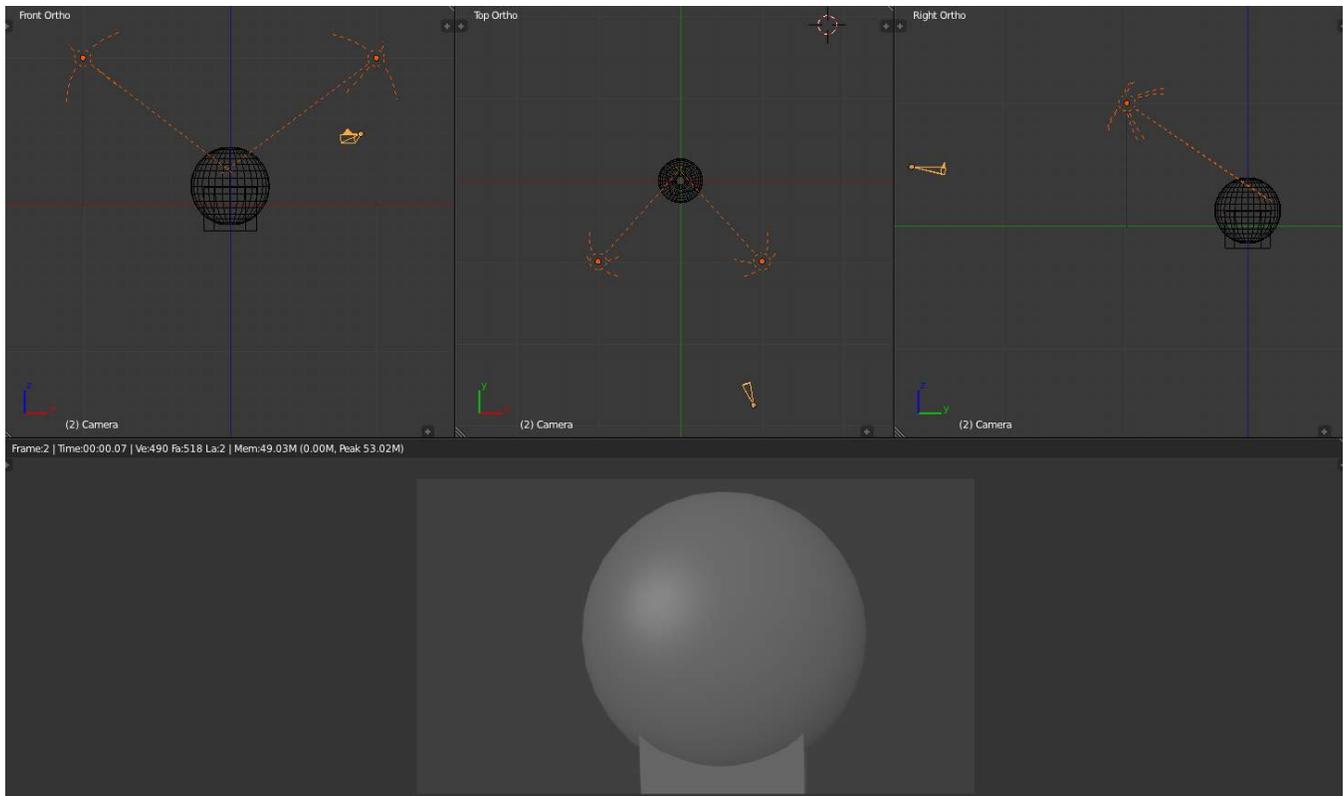


Fig. 2.2212: Standard two-point light rig.

Both lights are almost the same but do different things. Both emulate very wide, soft light by being *Hemi*. In real life, these lights bounce light off the inside of a silver umbrella.

Notice how we use low *Energy* to bring out the dimensionality of the sphere; I can't stress that enough. Hard, bright lights actually flatten it and make you squint. Soft lights allow your eye to focus. We disable specular for right *Hemi*, so we don't get that shiny forehead or nose.

The lamp on the left however, lets it be known that it is there by enabling specular; specular flare is that bright spot that is off center above midline on the sphere.

Use this rig to give even illumination of a scene, where there is no main focus. The *Hemi*'s will light up background objects and props, so *Environment Lighting* is not that important. At the opposite end of the lighting spectrum, two narrow *Spot* lights at higher power with a hard edge gives a "This is the Police, come out with your hands up" kind of look, as if the subject is caught in the crossfire.

**Three-Point Rigs** The standard three-point lighting rig is the most common illumination of objects and scenes bar none. If you want to show off your model, use this rig. As you can see, the untextured unmaterialized sphere seems to come out at you. There are multiple thesis on this rig, and you will use one of two:

- **Studio** - used in a real studio to film in front of a green screen or backdrop. Use this rig when you are rendering your CG objects to alpha into the scene so that the lighting on the actors *and* your CG objects is the same.
- **Standard** - used in real life to light actors on a set, and gives some backlighting to highlight the sides of actors, making them stand out more and giving them depth.

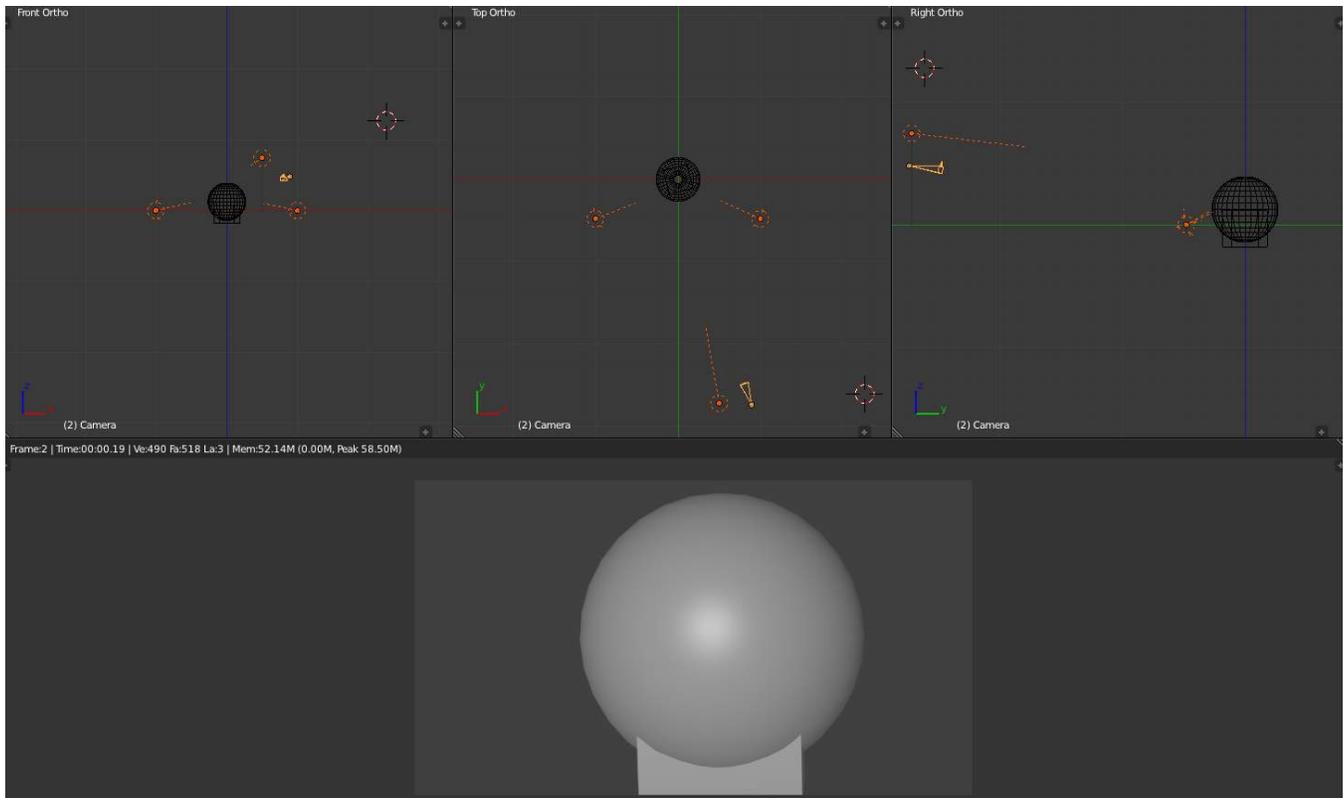


Fig. 2.2213: Studio three-point light rig.

**Studio rig** Shown to the right are the “Studio” top, front, and side views of the standard three-point lighting rig. It changes the dynamics of the scene, by making a brighter “key” light give some highlights to the object, while two side “fill” lights soften the shadows created by the key light.

In the studio, use this rig to film a talking head (actor) in front of a green screen, or with multiple people, keeping the key light on the main actor. This rig is also used to light products from all angles, and the side fill lights light up the props.

The key light is the *Area* light placed slightly above and to the left of the camera. It allows the specular to come out. It is about **30 BU** back from the subject, and travels with the camera. A little specular shine lets you know there’s a light there, and that you’re not looking at a ghost. In real life, it is a spot with baffles, or blinders, that limit the area of the light.

The two sidelights are reduced to only fill; each of them are *Hemi* lights placed **20 BU** to the side and **5 BU** in front of the subject, at ground level. They don’t cause a spotshine on the surface by disabling specular, and at ground level, light under the chin or any horizontal surfaces, countering the shadows caused by the key light.

Use this rig to give balanced soft lighting that also highlights your main actor or object. It combines the best of both the single rig and the two-point rig, providing balanced illumination and frontal highlights. For a wide scene, you may have to pull the sidelights back to be more positioned like the two-point rig.

**Standard Rig** Without a curtain in back of your main subject, you have depth to work with. The left fill light has been moved behind the subject (so it is now called a backlight) and is just off-camera, while the right side fill light remains the same. The keylight gives you specular reflection so you can play with specularity and hardness in your object’s material settings. The key light gives that “in-the-spotlight” feel, highlighting the subject, while the backlight gives a crisp edge to the subject against the background. This helps them stand out.

In this rig, the key light is a fairly bright spot light. Use a slighter tinge of yellow because the light is so bright; it is the only light for that side. The other sidelight has been moved in back and raised to eye (camera) level. You need to cut the energy of

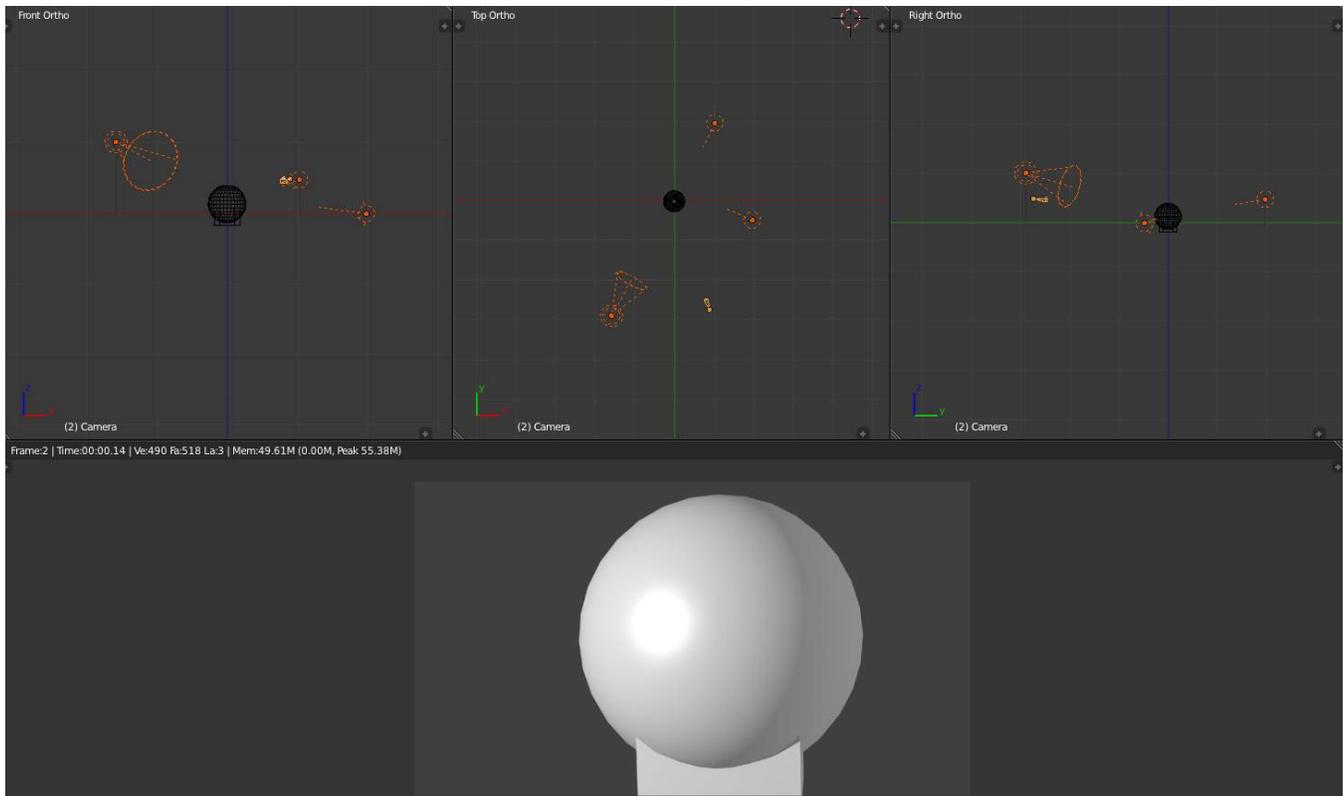


Fig. 2.2214: Standard three-point light rig.

the backlight in half, or when it is added to the remaining sidelight, it will light up the side too much and call too much attention to itself. You can vary the angle and height of the backlight to mimic a sun lighting up the objects.

Use this rig in normal 3D animations to light the main actor. Use this rig especially if you have transparent objects (like glass) so that there is plenty of light to shine through them to the camera. The tricky part here is balancing the intensities of the lights so that no one light competes with or overpowers the others, while making sure all three work together as a team.

**Four-point Rig** The four-point lighting rig provides a better simulation of outside lighting, by adding a *Sun* lamp 30 Blender Units above, 10 to the side, and 15 BU behind the subject. This sunlight provides backlighting and fills the top of the subject; even producing an intentional glare on the top of their head, telling you there is a sun up there. Notice it is colored yellow, which balances out the blue sidelights.

Changing the key light to a *Spot*, select *Inverse Square*, disable *Specular* and pure white light combines with and softens the top sun flare while illuminating the face, resulting in a bright sunshine effect. Two lights above means sharper shadows as well, so you might want to adjust the side fill lights. In this picture, they are still *Hemi*, disable *Specular*.

Use this rig when the camera will be filming from behind the characters, looking over their shoulder or whatnot, because the sun provides the backlight there. Also use this rig when you have transparent objects, so there is light to come through the objects to the camera.

Another spot for the fill light is shining up onto the main actor's face, illuminating the underside of his chin and neck. This gets rid of a sometimes ugly shadow under the chin, which if not corrected, can make the actor look fat or like they have a double chin; otherwise distracting. It evens out the lighting of the face.

**Troubleshooting** If you run into a problem with your render, where there are really bright areas, or really dark ones, or strange shadows, or lines on your objects, here is what I suggest you do:

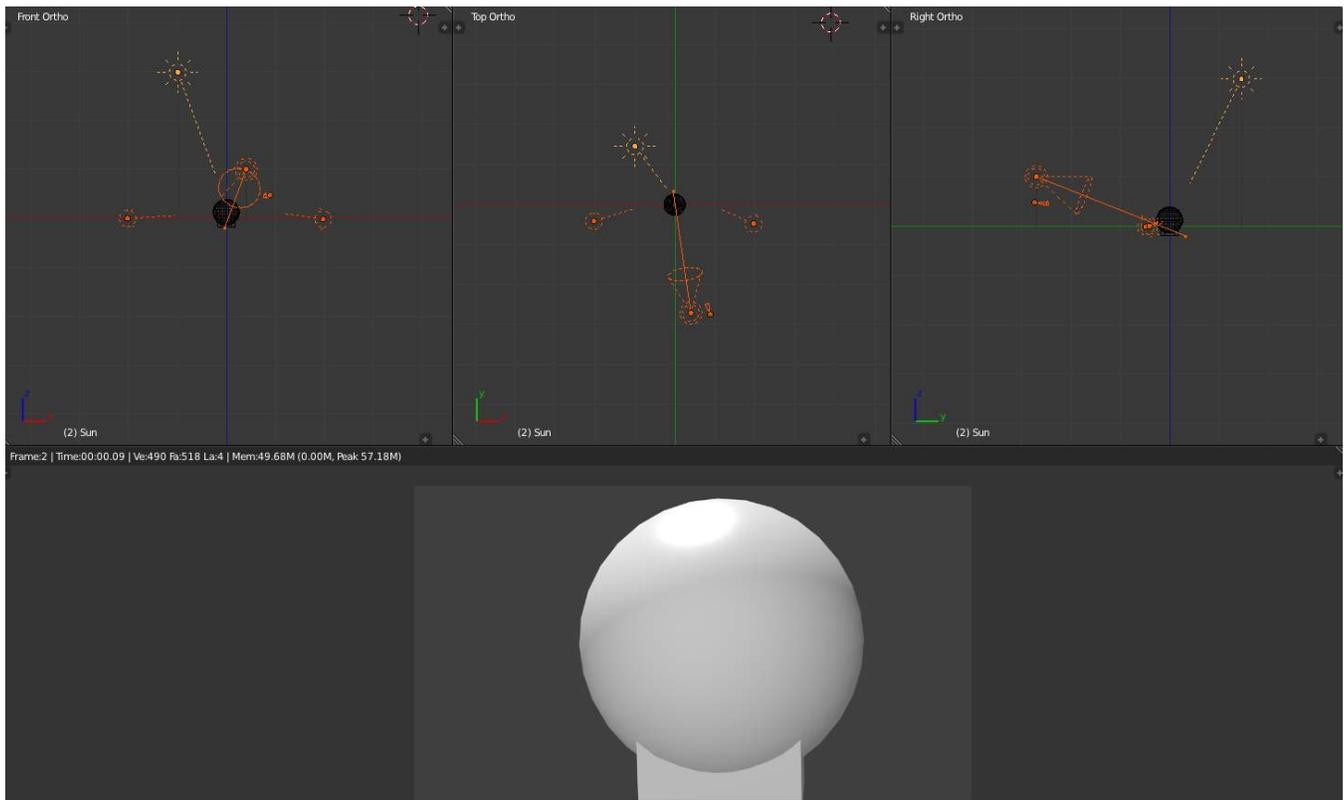


Fig. 2.2215: Four-point light rig.

- First, try deactivating all materials (create a default, gray one, and enter its name in the *Mat* field, *Layer* panel, *Render Layers* context - to get back all your normal materials, just erase this text field!). See if you get those problems with just grayness objects. If you don't have the problem anymore, that should tell you that you've got a materials-interacting-with-light problem. Check the material settings, especially ambient, reflection and all those little buttons and sliders in the *Material* context. You can set some lights to affect only certain materials, so if there's an issue with only a few objects being really bright, start with those.
- Then start "killing" lights (e.g. moving them to an unused layer); regress all the way back to one light, make sure it's smooth, then add them in one by one. As they add together, reduce power in the tested ones so they merge cleanly, or consider not adding it at all, or, especially, reduce the energy of the lamp you just introduced.
- You can also set lights to only light objects on a layer, so again, if some of the gray spheres have weirdness, check for that as well. Again, you may have done some of this accidentally, so sometimes deleting the light and re-adding it with defaults helps you reset to a known-good situation.
- Negative lights can be very tricky, and make your model blotchy, so pay special attention to your use of those special lights. Shadow-only lights can throw off the look of the scene as well. Overly textured lights can make your scene have random weird colors. Don't go too far off a slight tinge of blue or yellow or shades of white, or your material may show blue in the *Material* context but render green, and you will be very confused.
- Look at your environment settings *World* context: *Horizon*, *Zenith*, and *Environment Lighting*.

## Environment Lighting

Environment light provides light coming from all directions.

Light is calculated with a ray-traced method which is the same as that used by Ambient Occlusion. The difference is that Environment lighting takes into account the "ambient" parameter of the material shading settings, which indicates the amount of ambient light/color that that material receives.

Also, you can choose the environment color source (white, sky color, sky texture) and the light energy.

**Energy** Defines the strength of environment light.

**Environment Color** Defines where the color of the environment light comes from.

Using both settings simultaneously produces better global lighting.

It's good for mimicking the sky in outdoor lighting. Environment lighting can be fairly noisy at times.

## Ambient Occlusion

Ambient Occlusion is a sophisticated ray-tracing calculation which simulates soft global illumination shadows by faking darkness perceived in corners and at mesh intersections, creases, and cracks, where ambient light is occluded, or blocked.

There is no such thing as AO in real life; AO is a specific not-physically-accurate (but generally nice-looking) rendering trick. It basically samples a hemisphere around each point on the face, sees what proportion of that hemisphere is occluded by other geometry, and shades the pixel accordingly.

It's got nothing to do with light at all; it's purely a rendering trick that tends to look nice because generally in real life surfaces that are close together (like small cracks) will be darker than surfaces that don't have anything in front of them, because of shadows, dirt, etc.

The AO process, though, approximates this result; it's not simulating light bouncing around or going through things. That's why AO still works when you don't have any lights in the scene, and it's why just switching on AO alone is a very bad way of "lighting" a scene.

You must have ray tracing enabled as a *Render* panel option in the *Shading* section for this to work.

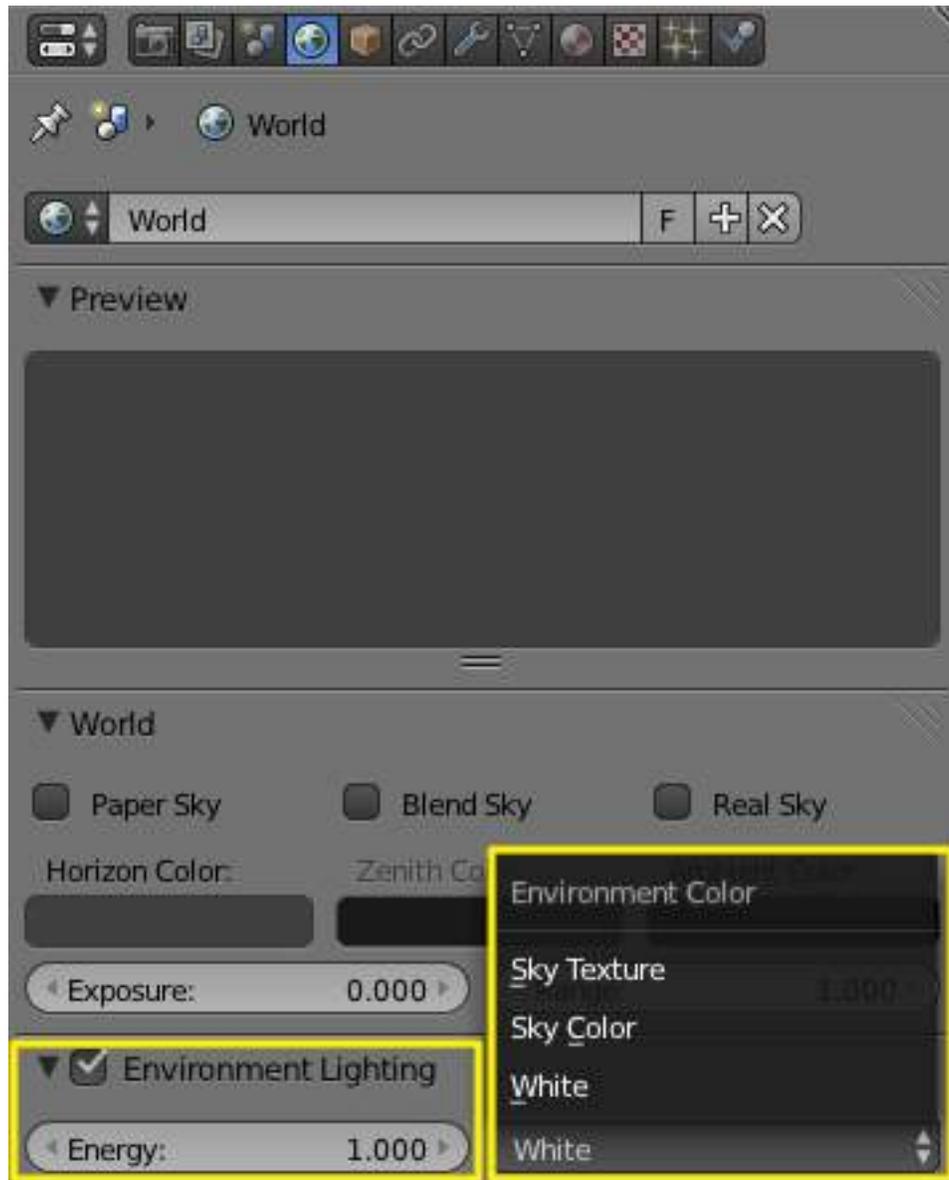


Fig. 2.2216: Environment Lighting panel.

You must have an ambient light color set as you desire. By default, the ambient light color (world) is black, simulating midnight in the basement during a power outage. Applying that color as ambient will actually darken all colors. A good outdoor mid-day color is RGB (0.9, 0.9, 0.8) which is a whitish yellow sunny kind of color on a bright-but-not-harshly-bright day.



Fig. 2.2217: The World panel with ambient color sliders highlighted.

### Options

**Factor** The strength of the AO effect, a multiplier for addition.

Ambient Occlusion is composited during the render. Two blending modes are available:

**Add** The pixel receives light according to the number of non-obstructed rays. The scene is lighter. This simulates global illumination.

**Multiply** Ambient occlusion is multiplied over the shading, making things darker.

**Note:** If *Multiply* is chosen, there must be other light sources; otherwise the scene will be pitch black. In the other two cases the scene is lit even if no explicit light is present, just from the AO effect. Although many people like to use AO alone as a quick shortcut to light a scene, the results it gives will be muted and flat, like an overcast day. In most cases, it is best to light a scene properly with Blender's standard lamps, then use AO on top of that, set to *Multiply*, for the additional details and contact shadows.

The *Gather* panel contains settings for the ambient occlusion quality. Note that these settings also apply to Environment Lighting and Indirect Lighting.

Ambient occlusion has two main methods of calculation: *Raytrace* and *Approximate*.

### Gather

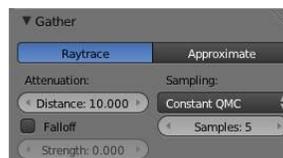


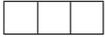
Fig. 2.2218: The Amb Occ panel, Raytrace method.

**Raytrace** The *Raytrace* method gives the more accurate, but also the more noisy results. You can get a nearly noiseless image, but at the cost of render time... It is the only option if you want to use the colors of your sky's texture.

**Attenuation** Length of rays defines how far away other faces may be and still have an occlusion effect. The longer this distance, the greater impact that far-away geometry will have on the occlusion effect. A high *Distance* value also means that the renderer has to search a greater area for geometry that occludes, so render time can be optimized by making this distance as short as possible for the visual effect that you want.

*Sampling*

**Samples** The number of rays used to detect if an object is occluded. Higher numbers of samples give smoother and more accurate results, at the expense of slower render times. The default value of 5 is usually good for previews. The actual number of rays shot out is the square of this number (i.e. *Samples* at 5 means 25 rays). Rays are shot at the hemisphere according to a random pattern (determined by the sample methods described above); this causes differences in the occlusion pattern of neighboring pixels unless the number of shot rays is big enough to produce good statistical data.



You have the three standard sampling options:

**Constant QMC** The base Quasi-Monte Carlo, gives evenly and randomly distributed rays.

**Adaptive QMC** An improved method of QMC, that tries to determine when the sample rate can be lowered or the sample skipped, based on its two settings:

**Threshold** The limit below which the sample is considered fully occluded (“black”) or un-occluded (“white”), and skipped.

**Adapt to Speed** A factor to reduce AO sampling on fast-moving pixels. As it uses the *Vec* render pass, that must also be enabled (see [render passes page](#)).

---

**Note:** About QMC

See also the [raytraced shadows](#) page for more info about the Quasi-Monte Carlo sampling method.

**Constant Jittered** The historical sample method, more prone to “bias” artifacts...

**Bias** The angle (in radians) the hemisphere will be made narrower (i.e. the hemisphere will no longer be a real hemisphere: its section will no longer be a semicircle, but an arc of a circle of  $\pi - \text{bias}$  radians).

The bias setting allows you to control how smooth “smooth” faces will appear in AO rendering. Since AO occurs on the original faceted mesh, it is possible that the AO light makes faces visible even on objects with “smooth” on. This is due to the way AO rays are shot, and can be controlled with the *Bias* slider. Note that while it might even happen with QMC sampling methods, it is much more visible with the *Constant Jittered* one - and anyway, you have no *Bias* option for QMC.

---

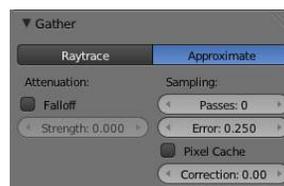


Fig. 2.2229: The Amb Occ panel, Approximate method.

**Approximate** The *Approximate* method gives a much smoother result for the same amount of render time, but as its name states, it is only an approximation of the *Raytrace* method, which implies it might produce some artifacts - and it cannot use the sky’s texture as the base color

This method seems to tend to “over-occlude” the results. You have two complementary options to reduce this problem:

**Passes** Set the number of pre-processing passes, between 0 (no pre-processing) to 10. Keeping the pre-processing passes high will increase render time but will also clear some artifacts and over-occlusions.

**Error** This is the tolerance factor for approximation error (i.e. the max allowed difference between approximated result and fully computed result). The lower, the slower the render, but the more accurate the results... Ranges between 0.0 and 10.0, defaults to 0.250.

**Pixel Cache** When enabled, it will keep values of computed pixels to interpolate it with its neighbors. This further speeds up the render, generally without visible loss in quality...

**Correction** A correction factor to reduce over-occlusion. Ranges between 0.0 (no correction) to 1.0.

### Common Settings

**Falloff** When activated, the distance to the occluding objects will influence the “depth” of the shadow. This means that the further away the occluding geometry is, the lighter its “shadow” will be. This effect only occurs when the *Strength* factor is higher than 0.0. It mimics light dispersion in the atmosphere...

**Strength** Controls the attenuation of the shadows enabled with *Use Falloff*. Higher values give a shorter shadow, as it falls off more quickly (corresponding to a more foggy/dusty atmosphere). Ranges from 0.0 (default, no falloff) to 10.0.

**Technical Details** Ambient occlusion is calculated by casting rays from each visible point, and by counting how many of them actually reach the sky, and how many, on the other hand, are obstructed by objects.

The amount of light on the point is then proportional to the number of rays which have “escaped” and have reached the sky. This is done by firing a hemisphere of shadow rays around. If a ray hits another face (it is occluded) then that ray is considered “shadow”, otherwise it is considered “light”. The ratio between “shadow” and “light” rays defines how bright a given pixel is.

**Hints** Ambient occlusion is a ray-tracing technique (at least with the *Raytrace* method), so it tends to be slow. Furthermore, performance severely depends on octree size, see the [rendering chapter](#) for more information.

### Indirect Lighting

*Indirect Lighting* adds indirect light bouncing of surrounding objects. It’s modes the light that is reflected from other surfaces to the current surface. Is more comprehensive, more physically correct, and produces more realistic images. It is also more computationally expensive. Take a look at the following examples of a scene lit with Direct Lighting and both Direct+Indirect Lighting:

Images courtesy of [rastermon.com](http://rastermon.com)

Indirect Lighting only works with Approximate gather method.

**Options** The *Indirect Lighting* panel contains two options:

**Factor** Defines how much surrounding objects contribute to light.

**Bounces** Number of indirect diffuse light bounces.

The *Gather* panel contains settings for the indirect lighting quality. Note that these settings also apply to Environment Lighting and Ambient Occlusion.

**Approximate** The *Approximate* method gives a much smoother result for the same amount of render time, but as its name states, it is only an approximation of the *Raytrace* method, which implies it might produce some artifacts - and it cannot use the sky’s texture as the base color

This method seems to tend to “over-occlude” the results. You have two complementary options to reduce this problem:

**Passes** Set the number of pre-processing passes, between 0 (no pre-processing) to 10. Keeping the pre-processing passes high will increase render time but will also clear some artifacts and over-occlusions.

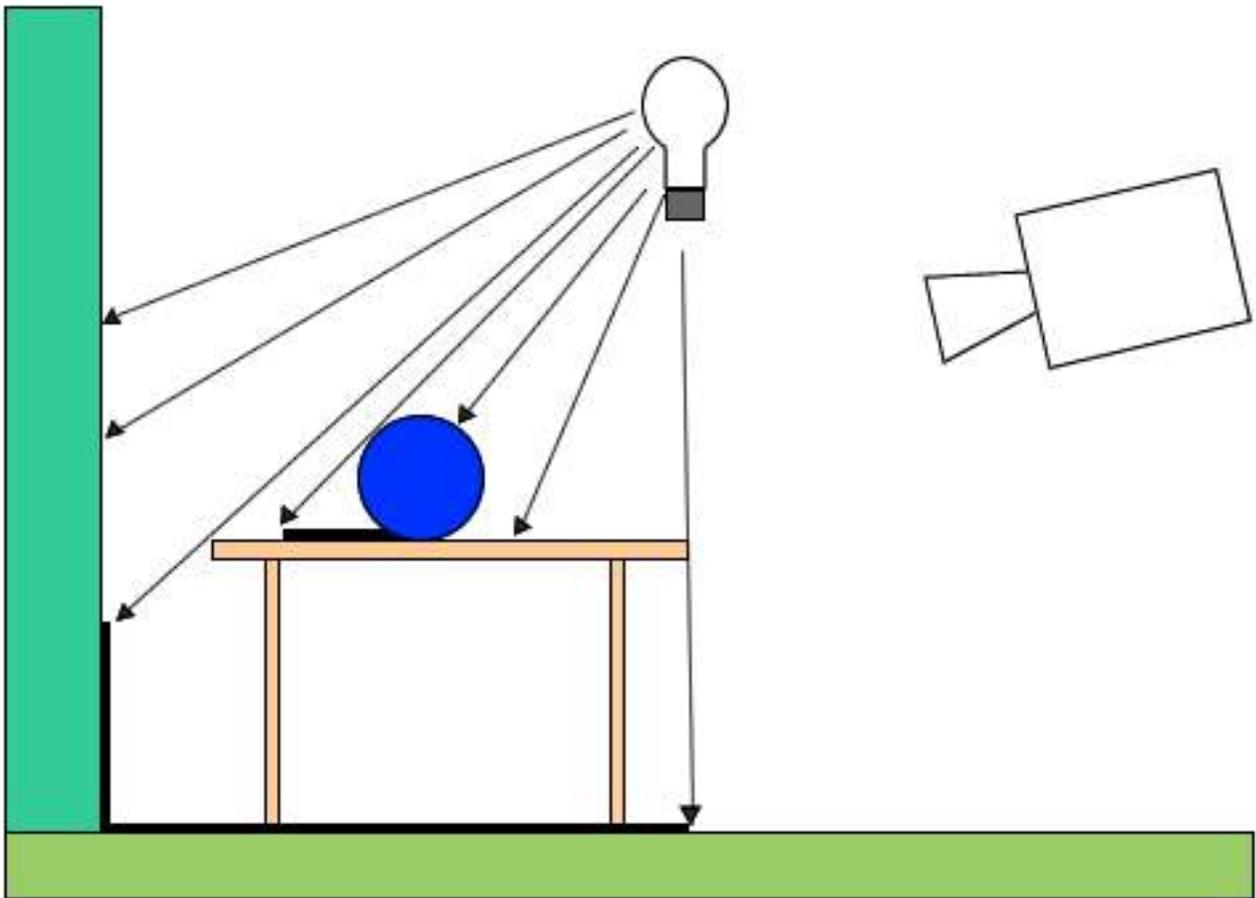


Fig. 2.2230: Direct Lighting Schematic.

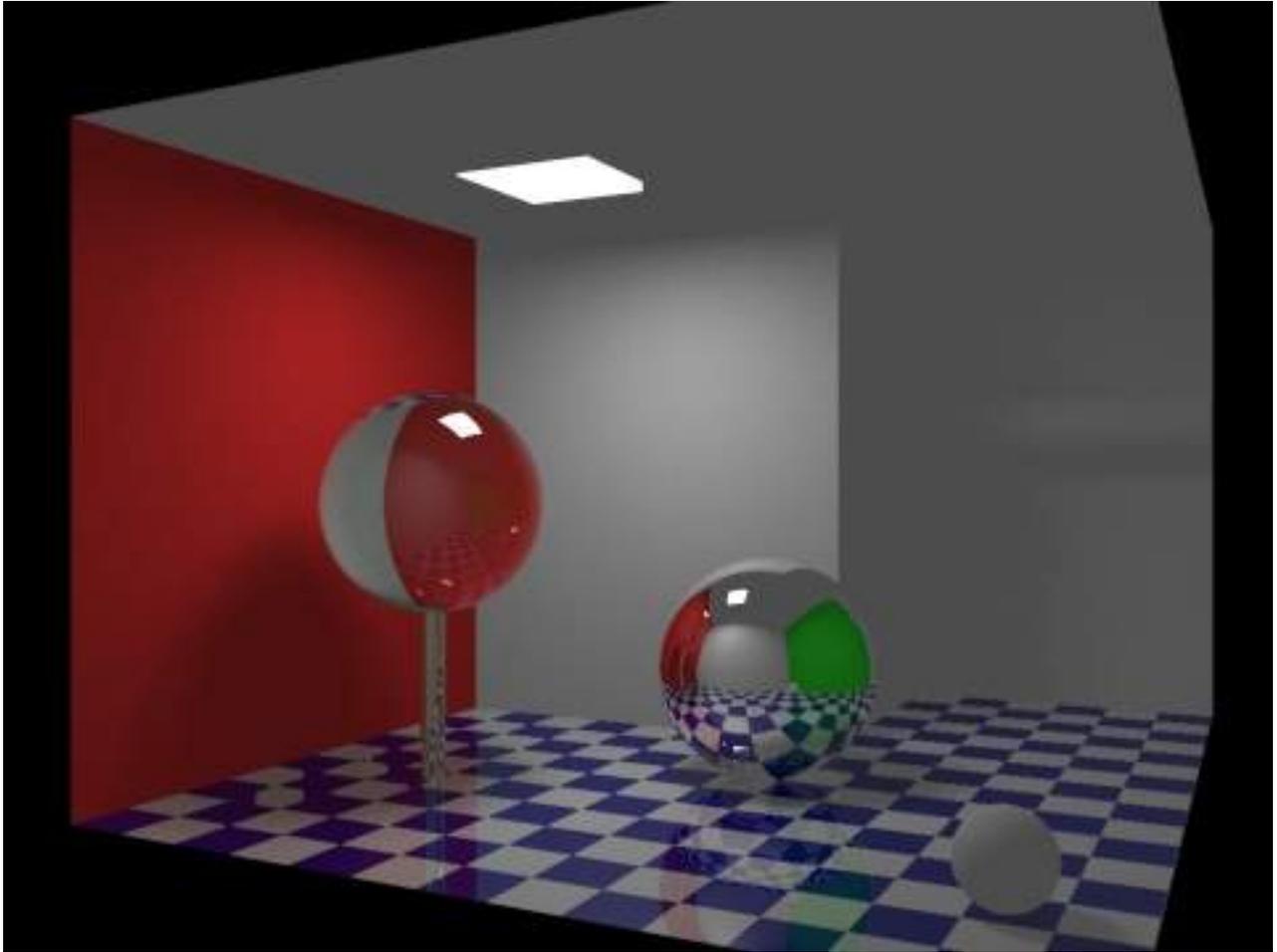


Fig. 2.2231: Direct Lighting Render

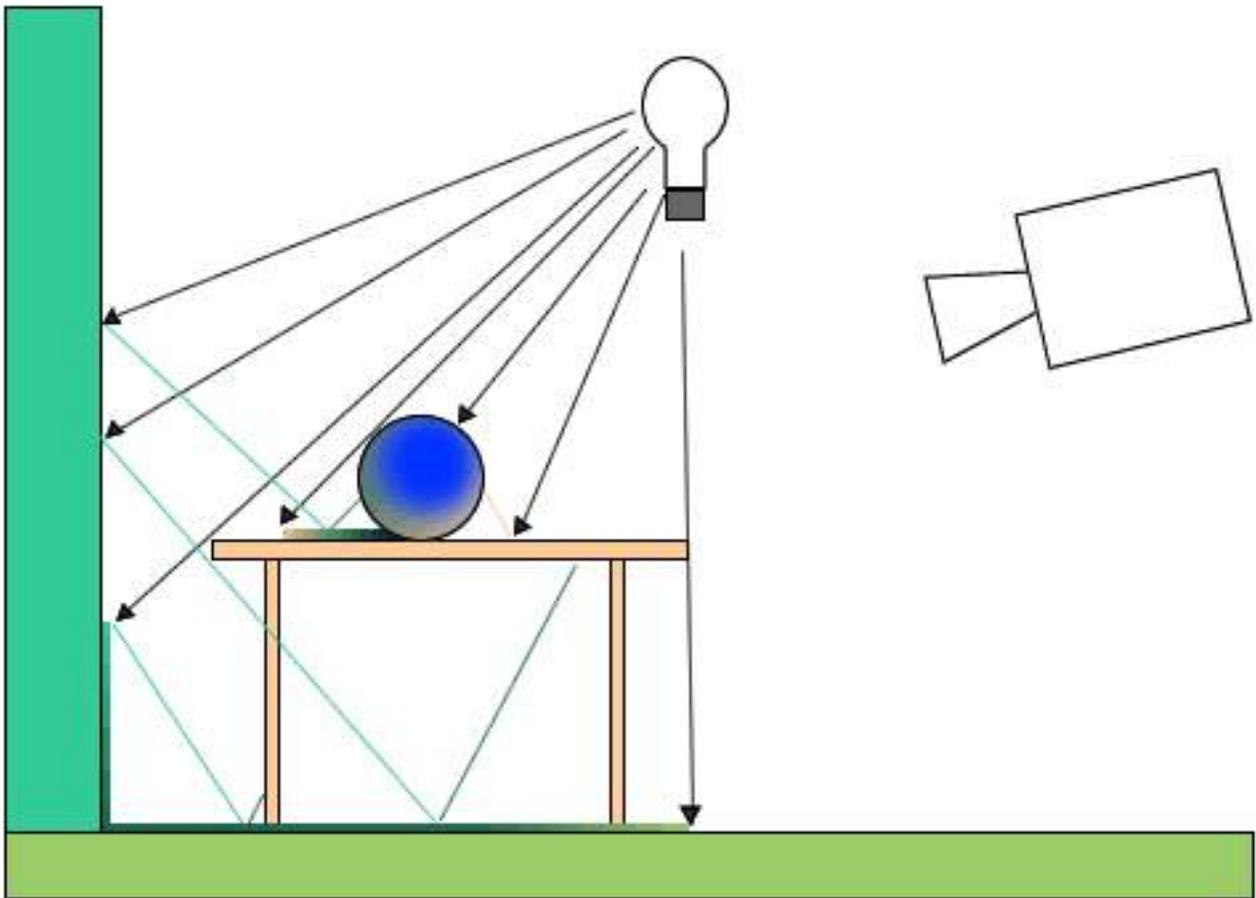


Fig. 2.2232: Direct+Indirect Lighting Schematic

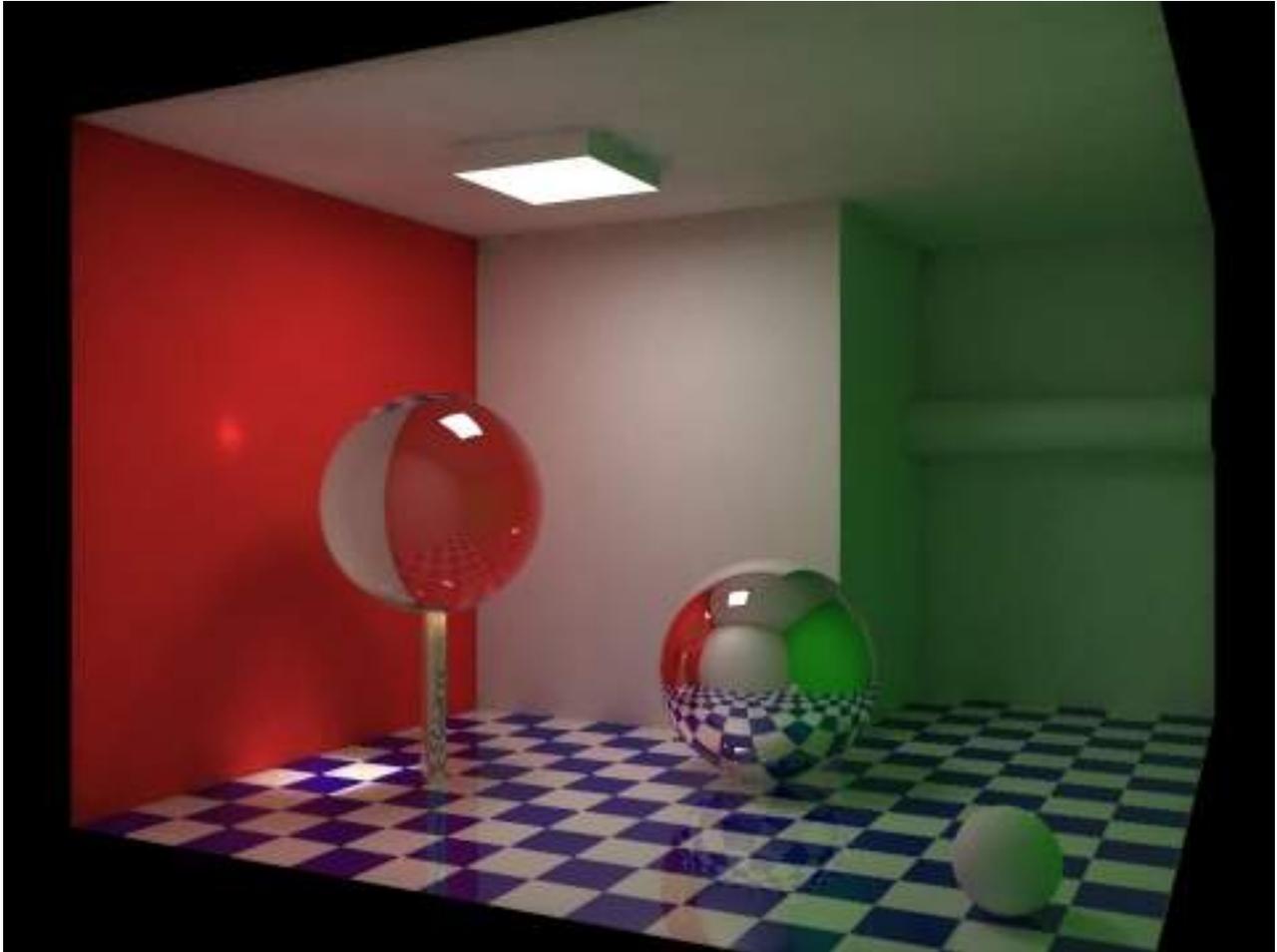


Fig. 2.2233: Direct+Indirect Lighting Render



Fig. 2.2234: Indirect Lighting parameters.

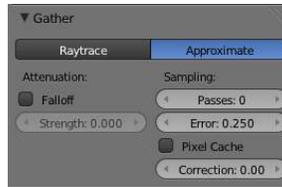


Fig. 2.2235: The Indirect Lighting panel, Approximate method.

**Error** This is the tolerance factor for approximation error (i.e. the max allowed difference between approximated result and fully computed result). The lower, the slower the render, but the more accurate the results... Ranges between 0.0 and 10.0, defaults to 0.250.

**Pixel Cache** When enabled, it will keep values of computed pixels to interpolate it with its neighbors. This further speeds up the render, generally without visible loss in quality...

**Correction** A correction factor to reduce over-occlusion. Ranges between 0.0 (no correction) to 1.0.

## Exposure and Range

### Reference

Mode: All modes

Panel: *World* (*Shading* context, *World* sub-context)

**Description** *Exposure* and *Range* are similar to the “Color Curves” tool in Gimp or Photoshop.

These controls affect the rendered image, and the results are baked into the render. For information on achieving similar affects with render controls, see [Color Management and Exposure](#).

Previously Blender clipped color directly with 1.0 (or 255) when it exceeded the possible RGB space. This caused ugly banding and overblown highlights when light overflowed (*An overexposed teapot*).

Using an exponential correction formula, this now can be nicely corrected.



Fig. 2.2236: Exposure and Range sliders.

### Options

**Exposure** The exponential curvature, with 0.0 being linear, and 1.0 being curved.

**Range** The range of input colors that are mapped to visible colors (0.0 – 1.0).

So without *Exposure* we will get a linear correction of all color values:

**Range**  $> 1.0$  the picture will become darker; with  $Range = 2.0$ , a color value of  $1.0$  (the brightest by default) will be clipped to  $0.5$  (half bright) ( $Range : 2.0$ ).

**Range**  $< 1.0$  the picture will become brighter; with  $Range = 0.5$ , a color value of  $0.5$  (half bright by default) will be clipped to  $1.0$  (the brightest) ( $Range : 0.5$ ).

**Examples** With a linear correction every color value will get changed, which is probably not what we want. *Exposure* brightens the darker pixels, so that the darker parts of the image won't be changed at all ( $Range : 2.0$ ,  $Exposure : 0.3$ ).



**Hints** Try to find the best *Range* value, so that overexposed parts are barely not too bright. Now turn up the *Exposure* value until the overall brightness of the image is satisfying. This is especially useful with area lamps.

## World

### Introduction

Blender provides a number of very interesting settings to complete your renderings by adding a nice background, and some interesting 'depth' effects. These are accessible via the *World* context. By default a very plain uniform world is present. You can edit it or add a new World.

You have:

**Background** The color and texture of the world background, with special settings for mapping coordinates.

**Mist** Add a mist to your scene to enhance the feeling of depth.

While these world settings offers a simple way of adding effects to a scene, **compositing nodes** are often preferred, though more complex to master, for the additional control and options they offer. For example, filtering the Z value (distance from camera) or normals (direction of surfaces) through compositing nodes can further increase the depth and spacial clarity of a scene.

---

**Note:** Some of the settings under the World panel in Blender affect lighting so you find them under the **Lighting** chapter (see **Ambient Light**, **Exposure** and **Ambient Occlusion**). When using a *Sun Lamp* options for *Sky & Atmosphere* are available in the *Lamp* menu.

---

### World Background

**Description** The world buttons let you set up the shading of your scene in general. It can provide ambient color, and special effects such as mist, but a very common use of a *World* is to shade a background color.

---

**Note:** Background Image in Render

To use an image as your render background, see [BackBuf images specified in the Output Panel](#)

---

**Note:** Background Image in 3D

To use an image as a background image in your 3D view, for example as a reference when doing a model, see [using a Background Image](#)

---

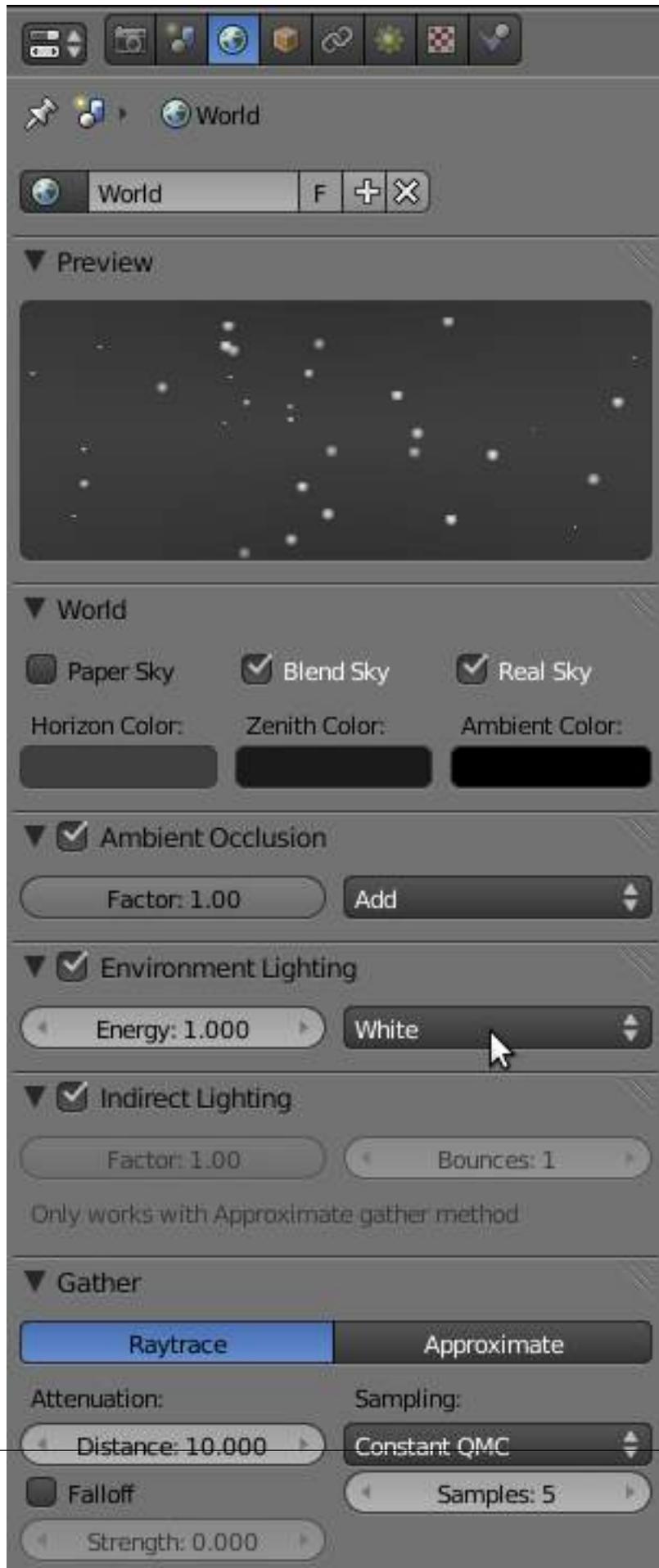




Fig. 2.2246: World panel

### Options

**Horizon Color** The RGB color at the horizon

**Zenith Color** The RGB color at the zenith (overhead)

How these colors are interpreted depends on which kind of *Sky* is chosen.

**None Enabled** If none of these three buttons is checked, your background will just be plain flat color (using the horizon one).

**Paper Sky** If this option is added, the gradient keeps its characteristics, but it is clipped in the image (it stays on a horizontal plane (parallel to x-y plane): what ever the angle of the camera may be, the horizon is always at the middle of the image).

**Blend Sky** The background color is blended from horizon to zenith. If only this button is pressed, the gradient runs from the bottom to the top of the rendered image regardless of the camera orientation.

**Real Sky** If this option is added, the gradient produced has two transitions, from nadir (same color as zenith) to horizon to zenith; the blending is also dependent on the camera orientation, which makes it more realistic. The horizon color is exactly at the horizon (on the x-y plane), and the zenith color is used for points vertically above and below the camera.

**Textures** Instead of a color, or blend of two colors, Blender can use an 2D image which it maps to a very large Box or sphere which encompasses the entire scene, or which it maps to a virtual space around the scene.

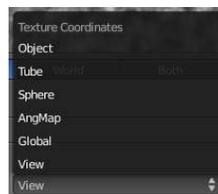


Fig. 2.2247: Texture Coordinates pop-up menu

The World textures are accessible in the texture menu (just select *World* first, then *Texture*. They are used much like the Materials textures, except for a couple of differences. The textures can be mapped according to:

**View** The default orientation, aligned with the co-ordinates of the final render

**Global** Uses global coordinates

**AngMap** Used to wrap a standard hemisphere angular map around the scene in a dome. This can be used for image based lighting with *Ambient Occlusion* set to sky color. You'll generally need a high dynamic range image (HDRI) angular map. (It will look like a weird spherical image).

**Sphere** Sphere mapping, similar to that of materials

**Tube** Wrap the rectangular texture around in a cylinder, similar to that of materials

**Object** Position the texture relative to a specified object's local texture space

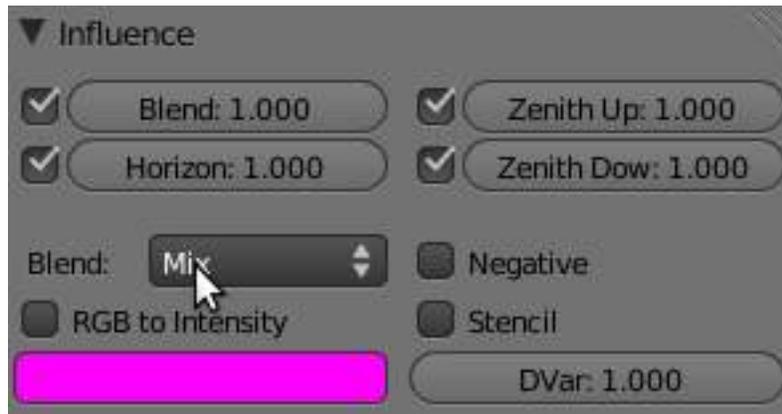


Fig. 2.2248: Texture Influence panel

The texture affects color only, but in four different ways:

**Blend** Makes the Horizon color appear where the texture is non-zero

**Horizon** Affect the color of the horizon

**Zenith Up** Affect the zenith color overhead

**Zenith Down** Affect the zenith color underneath

If you are disappointed that your camera appears to carry the texture with it rather than rotate through the texture, you should check the Real Sky checkbox in the World tab of the Properties view.

## Mist

**Description** Mist can greatly enhance the illusion of depth in your rendering. To create mist, Blender makes objects farther away more transparent (decreasing their Alpha value) so that they mix more of the background color with the object color. With Mist enabled, the further the object is away from the camera the less it's alpha value will be.



Fig. 2.2249: Mist panel

## Option

**Mist check box** Toggles mist on and off

**Minimum** An overall minimum intensity, or strength, of the mist.

**Start** The distance from the camera at which the mist starts to fade in

**Depth** The distance from *Start* of the mist, that it fades in over. Objects further from the camera than *Start+Depth* are completely hidden by the mist.

**Height** Makes the mist intensity decrease with height, for a more realistic effect. If greater than 0, it sets, in Blender units, an interval around  $z=0$  in which the mist goes from maximum intensity (below) to zero (above).

**Falloff** The decay rate of the mist (*Quadratic / Linear / Inverse Quadratic*). These settings control the rate of change of the mist's strength further and further into the distance.

---

**Note:** Mist distances

To visualize the mist distances in the 3D View, select your camera, go to the camera menu, and enable *Show Mist*.

The camera will show mist limits as a line projecting from the camera starting from *Start* and of distance *Depth*.

To get a better view to evaluate the *Mist* visualization, `Shift-Numpad1` with the camera selected (`Numpad5` to toggle perspective view on and off). This will place the 3D view right over the camera looking down.

---

**Transparency** Because *Mist* works by adjusting transparency, this can sometimes cause objects to be partially transparent when they shouldn't be. One workaround is to set the Mist settings as desired, but turn Mist off. The Mist data is still available for compositing even though it is off. Use [Do Composite](#) and the [Nodes Editor](#) to feed the Mist pass to an [AlphaOver](#) to blend the background color (or a render layer with just the sky) with the rendered image. This produces the mist effect but since Mist is off the object transparency (or lack of) is preserved.

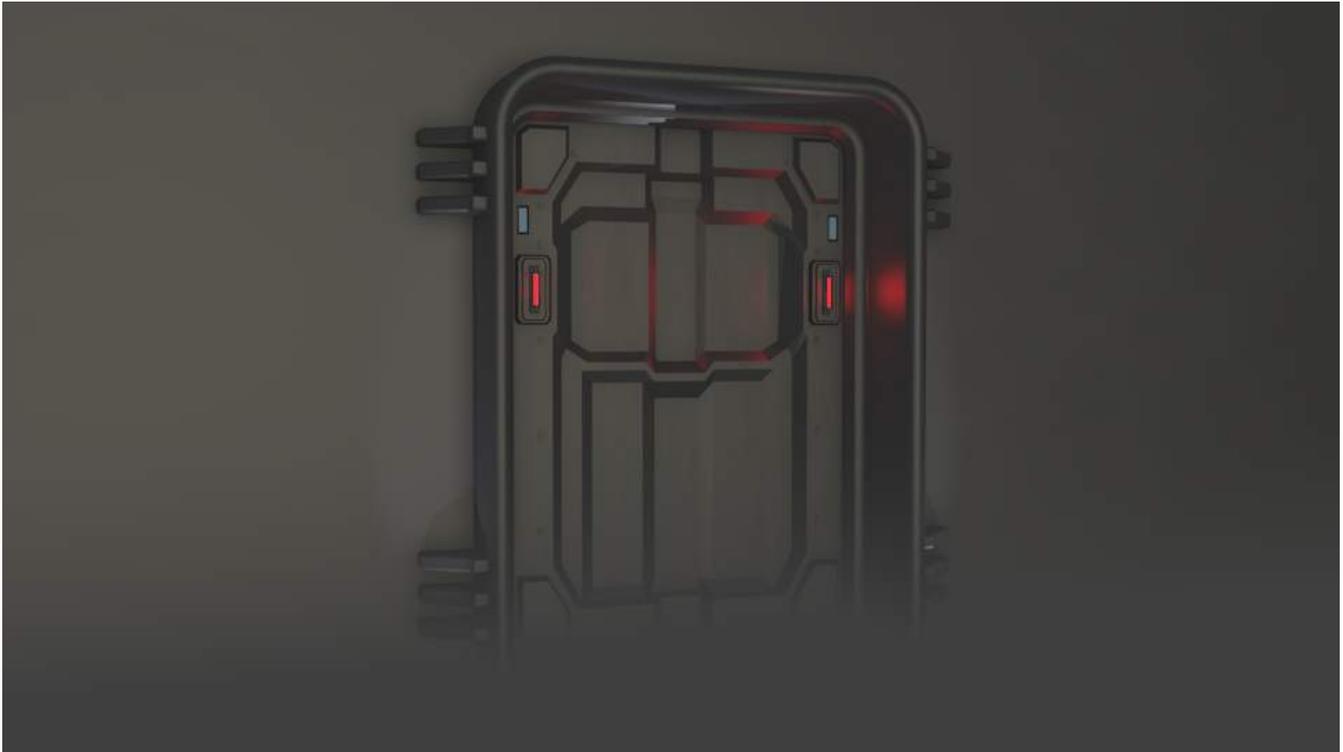


Fig. 2.2250: Mist example

**Examples** In this example (`.blend`) the *Mist Height* options has been limited to create smoke covering the floor. This simple scene was inspired by [Stefan Morell's Arc Sci-Fi Corridor](#).

## Render Layers

This section covers only the Render Layer settings appropriate for the Blender Render engine. For the engine-independent settings, see [this section](#).

**Light Override** Enter the name of a light group, and the scene will be lit with only those lights.

Examples of where this might be used:

- Using multiple Render Layers with different light group overrides can allow you to tweak light intensity and color in the compositor (avoiding re-renders).
- Speed up a draft render by using only a few lights instead of all of them.

## Include Options

Each render layer has its own set of features which can be enabled/disabled to save time and give you control when working with passes:

**Zmask** Only render what's in front of the solid Z values.

**Negate** Only render what's Behind the solid Z values.

**All Z** Z-values are computed for everything in view, not just those things that are rendered. When disabled, objects not included in the render have no ("infinite") z value.

**Solid** Solid faces are rendered. All normal meshes are solid faced.

**Halo** Halo materials are rendered.

**ZTransp** Transparency may be Z-based or Ray-traced. If Z-based, enabling *Ztra* renders transparent areas with the z-value of what is behind the transparent area.

**Sky** Turning on Sky renders the sky, as defined in your material world settings. Otherwise, a black alpha transparent background is rendered.

**Edge** If Edge is enable in the Output panel, objects in this Render Layer are given an outline edge. Turning on Edge pulls in the Edge settings from the Output tab, and adds an outline to the objects. Edges also have to be enabled on the Output tab.

**Strand** Strands are strings of static particles that are colored as part of the material settings; they look like strands of hair or grass.

**Freestyle** Render the Freestyle strokes on this layer.

## Render Passes

Render Passes are necessary because of the different things the Blender Render Engine must calculate to give you the final image. In each 'pass' the engine calculates different interactions between objects.

### Render Passes In Detail

Everything you see in a render must be calculated for the final image. All interactions between objects in your scene, lighting, cameras, background images, world settings, etc., must all be separately calculated in different passes for different reasons, such as calculating shadows.

In a render, every pixel has been calculated several times to make sure it will show the right color for the right part of the image. Various things that are calculated in a standard render include:

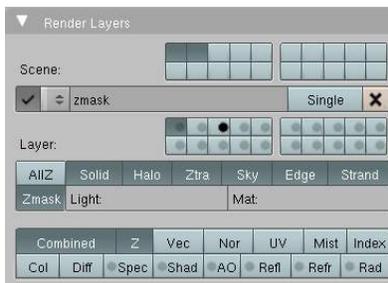


Each Render Pass puts out an image or a map. For the purposes of this example, a Render Layer was defined to produce all possible outputs. When a Render Layer input-node was added to the node diagram and the Render Layer input-node was subsequently associated with the Render Layer, all of the layer's outputs appeared as connection points on the right-hand (output) side of the node.

Render Passes that produce Images can be directly viewed in a viewer, or, if they are the only pass that is rendered, saved as the render image. If the pass is enabled, it can be saved in a multilayer OpenEXR format.

If the Render Pass output is not an image but is a map, it needs to be translated into something that we can see. For example, the Z-depth map is an array of values that specifies how far away from the camera each pixel is; values range between +/-3,000,000 Blender Units or so. The intermediate node you see above, between the Render Layer output socket and the Viewer node input socket (such as Map Value) does this translation or scaling. You must use that specific kind of translation node to get good results if you intend on operating on that map as an image. You must then, after making any adjustments, run the map back through that node to re-scale it back to the original before saving.

### Selecting Render Passes



Render Passes are the various distinct outputs that the renderer is able to generate. All of the following render outputs are normally combined into a single output known, appropriately enough, as the **Combined** output. But you can also select any of them to be output as a separate pass. (If you do so, in most cases you can choose whether to *also* continue to include it in the Combined output.)

Some of these outputs must be enabled and used within your scene (and not just selected in the Render Layer panel) in order to show anything. For example, if you do not have any lights in your scene, or those lights have been set to not cast shadows, or objects in the limelight do not have materials which have been set to receive shadows, the **Shadow** pass will be blank; there's simply nothing to show you. If you have not enabled *Ambient Occlusion* in your World environment settings, the **AO** pass will be blank, even if you select it here.

To save time and disk space, you have to tell Blender each of the passes to render in the Render Layers panel (which we first introduced on [the previous page](#)):

**Combined** This renders everything in the image, even if it's not necessary. ("The whole enchilada," so to speak.) This is all the options below, blended into a single output, *except* those options which you've indicated should be omitted from this pass, as indicated with the camera button.

**Z** The Z-depth map; how far away each pixel is from the camera. Used for Depth-Of-Field (DOF). The depth map is inverse linear (*1/distance*) from the camera clip start.

**Vector** The direction and speed things are moving. Used with Vector Blur.

**Normal** Calculates lighting and apparent geometry for a bumpmap (an image which is used to fake detail on an object) or for changing the apparent direction of light falling on an object.

**UV** Allows texturing after rendering. See UV node.

**Mist** Deliver Mist factor pass.

**Object Index** Masks selected objects. See MaskObj node.

**Color** The color of materials without shading.

**Diffuse** The diffuse shading of materials.

**Specular** Specular highlights.

**Shadow** Shadows cast. Make sure shadows are cast by your lights (positive or negative), and received by materials. To use this pass, mix multiply it with the Diffuse pass.

**Emit** Emission pass.

**AO** Ambient Occlusion. Make sure it's turned on in your environment and that RayTracing is enabled.

**Environment** Environment lighting.

**Indirect** Indirect lighting pass.

**Reflection** Reflection off mirrors and other reflective surfaces (highly waxed white floors, for example). Mix Add this pass to Diffuse to use it.

**Refraction** Refraction of colors through transparent meshes. Mix Add this pass to the Diffuse pass to use it.

When you enable a pass, the appropriate socket on the Render Layers node shows up like magic, and can be used as shown in the example above.

### Excluding Render Passes

As we said, the **Combined** output is an amalgam of several outputs which are *also* available separately. When you select one of these outputs, they will be provided separately *and also* included in the Combined pass.

When you enable the Camera icon that is beside several of the pass options, the particular pass will be excluded from the combined pass. They will be made available separately *but not* included in the combined pass.

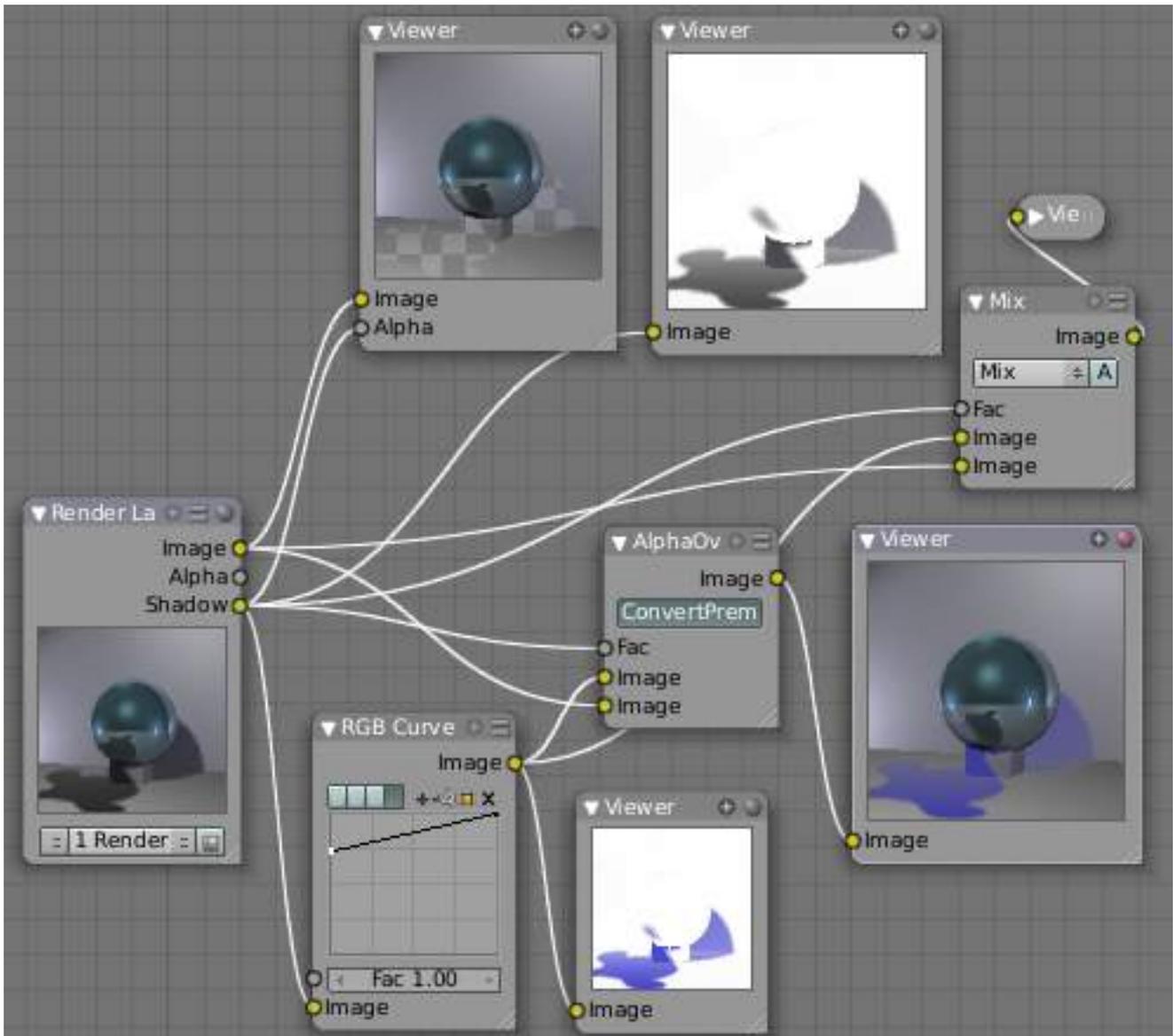
### Using Render Passes

The primary purpose of Render Passes is to enable you to process the various outputs in different ways, by constructing networks of render nodes. You can achieve many special effects, and economize considerably on the render times of complicated scenes, by creative and effective use of this facility. We'll show you a few examples of this in just a moment.

Quite a bit of information about the typical uses for some of the passes is discussed elsewhere:

- Image: Since this is the main product, all of Blender uses it.
- Alpha: See the *AlphaOver* node and all of the *Matte* nodes.
- Z: See the *Defocus* node.
- Vec: See the *Vector Blur* node.
- Normal: See the *Normal* node.

**Recoloring Shadows** Let's run the Shadow buffer through a colorization noodle, then recombine it; all your shadows will be artificially colored. Lots of threads in this noodle are shown to the right, so let's walk through it. On the left is the Render Layer input node: it refers to one of the Render Layers that we have defined for our scene. In the scene, we have a reflective ball on a pedestal standing in front of a backdrop. Everything (except the ball) is gray. We use a standard four-light rig: backfill placed high, two sidefills at ground level, and a key light above and to the left of camera. Suzanne, a monkey-shaped geometry, is standing in front of the key light, so her shadow is cast into the scene on the floor. The ball casts shadows onto the backdrop and floor.



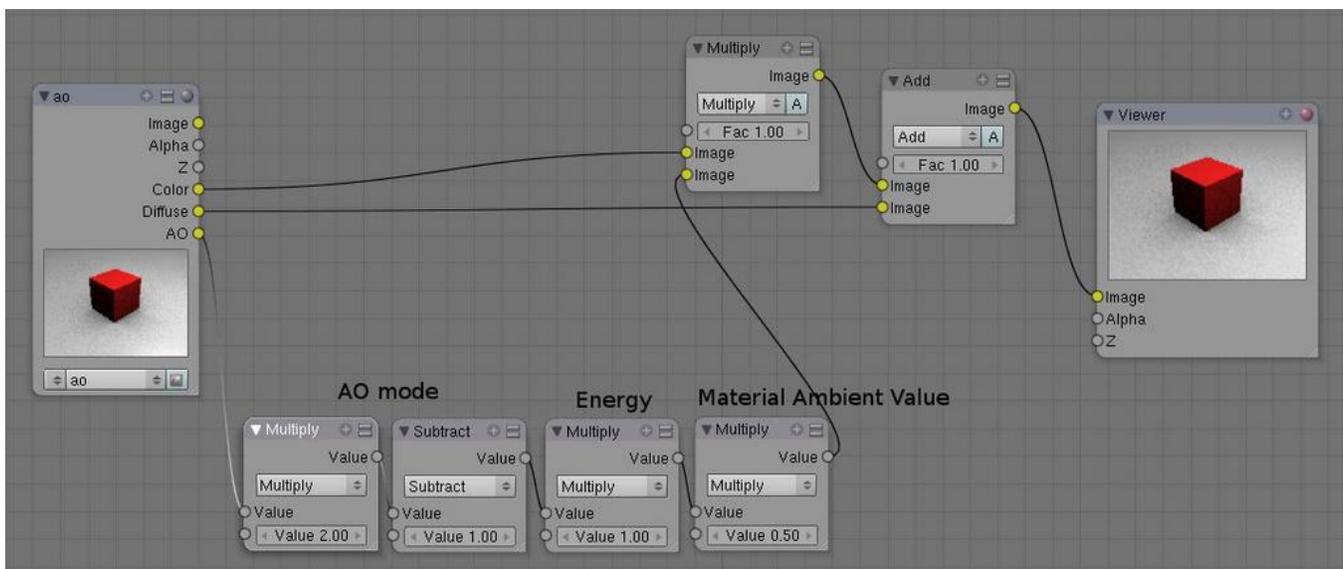
The output channels of the Render Layer node are determined by which buttons we selected when defining our Render Layer. The top two viewers show you the image output using the Shadow as the Alpha channel, and the node next to it just the Shadow channel. Where the Shadow is dark, the image in the left viewer is transparent. We have used the Shadow to cut out parts of the image.

We then take the shadow through an RGB Curve, which is set to magnify just the Blue by 75%; so a gray shadow of (R:40, G:40, B:40) becomes (R:40, G:40, B:40x1.75=70). That blue-tinged shadow is shown in the bottom viewer. Now we have two options: AlphaOver and Mix. For either option:

- Use the Shadow map as a Factor.
- Feed the Blue Shadow to the Top Socket.
- Feed the core or base image to the Bottom Socket.

The resulting image is the same in either case; a blue shadow. Note that Suzanne's reflection is not blue; there's a different Render Pass for that.

You could just as easily swap in another image entirely; for example, the shadow map from another render layer. You can even take an image from another project entirely and use that instead (using the Image Input node), to get a different effect. (For example, an effect similar to a *Star Wars Episode One* movie poster, where Anakin Skywalker already casts the shadow of Darth Vader.)



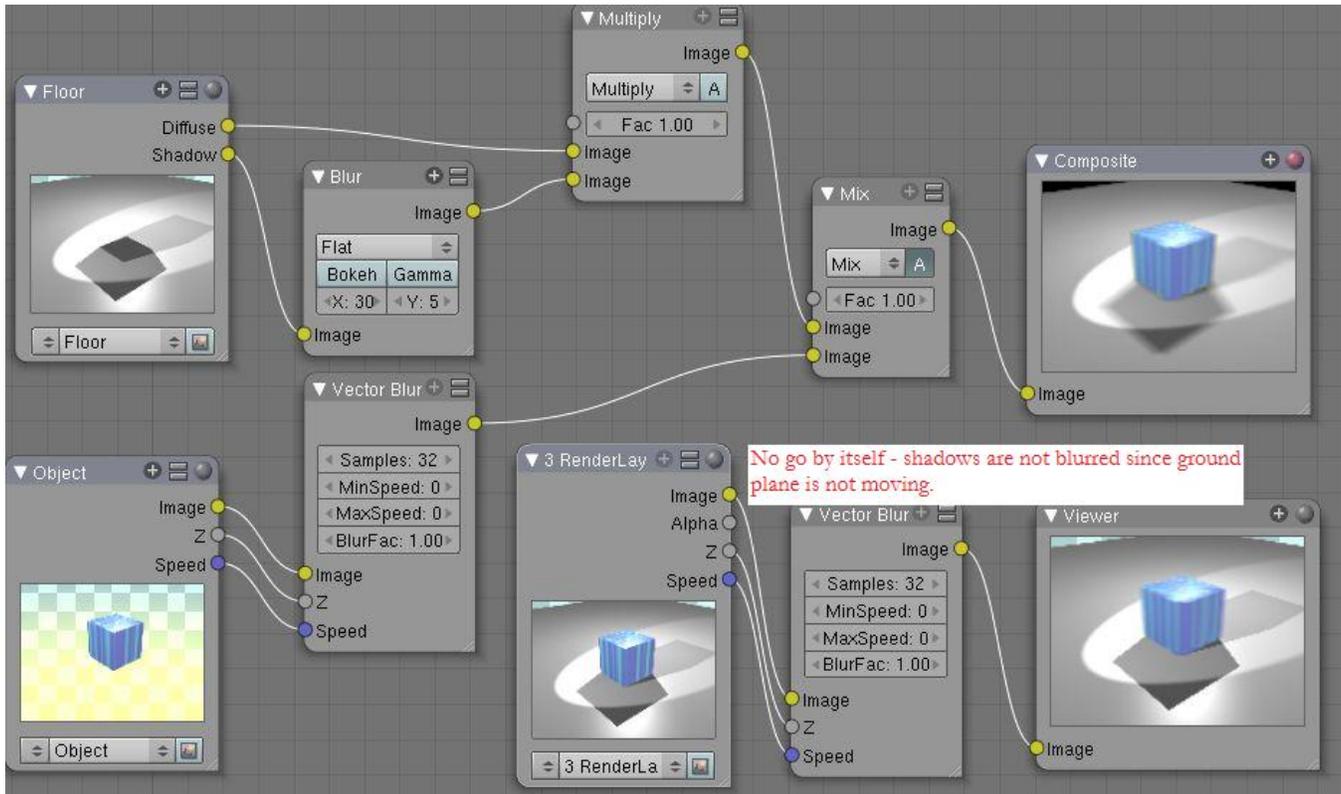
**Compositing Ambient Occlusion** AO is a geometry-based dirt shader, making corners darker. It is separately enabled in the World settings and computed as a separate pass. When enabled, it has one of three Modes (*Add*, *Subtract*, *Both*), and variable *Energy* level (which changes the intensity of the shading). The third variable is the amount of Ambient light that the material receives. If it does not receive any, then ambient occlusion does not affect it. Based on these variables, Blender computes an AO pass. If you call it out as a separate pass and wish to composite it back into your image, you will need to enable the Color and Diffuse pass as well.

To configure your noodle, consider the example image above.

- First, depending on the AO mode do one of the following: If AO mode is Add: directly use the AO pass. If AO mode is Sub: Calculate  $AO - 1$ , or if AO mode is Both: Calculate  $2 * AO - 1$ .
- Multiply the output of Step 1 with the AO energy level.
- Multiply the output of Step 2 with the material's ambience value. If you have materials which receive different ambience light levels (0.5 is the default), one would have to create an ambience map based on Object ID.

- Multiply the output of Step 3 with the color pass.
- Add the output of Step 4 to the diffuse pass.

If shadows, colored ambient light, specularity, reflections, and/or refractions are involved they have to be added to the diffuse pass before adding the converted AO pass.



**Vector Blurring Shadows** When using Vector Blur instead of Motion Blur, objects in motion are blurred, but objects at rest (with respect to the camera) are not blurred. The crossover is the shadow of the object in motion. Above, we have a cube in motion across a ground plane. If we just ran the combined pass through Vector Blur, you can see the result in the lower right-hand corner; the box is blurred, but its shadow is sharply in focus, and thus the image does not look realistic.

Therefore, we need to separate out the diffuse and shadow passes from the floor by creating a “Floor” render layer. That render layer has Diffuse and Shadow passes enabled, and only renders the floor object (layer 2). Another render layer (“Cube”) renders the Z and Vector passes, and only renders the cube (on layer 1). Using the Blur node, we blur the shadow pass, and then combine the diffuse and blurred shadow by multiplying them together in a Mix Multiply node; we then have a blurred shadow on a crisp ground plane. We can then mix the vector-blurred object to provide a realistic-looking image.

## Conclusion

Render Passes can be manipulated to give you almost complete control over your final image. Causing objects to cast shadows that aren’t really their shadows, making objects appear out of focus or sharply in focus like a real camera, manipulating colors just for final post-processing or just reconfiguring your render passes to save render time, are all things which you might wish to manipulate the render engine for.

## Motion Blur

Blender's animations are by default rendered as a sequence of *perfectly still* images. While great for stop-motion and time-lapses, this is unrealistic, since fast-moving objects do appear to be blurred in the direction of motion, both in a movie frame and in a photograph from a real-world camera.

Blender has two ways to achieve motion blur:

### Sampled Motion Blur

Blender can be made to render the current frame and some more 'virtual' frames in between it and the next frame, then merge them all together to obtain an image where moving objects are 'blurred'.

This method is slow, but produces good results. It can be activated in the *Sampled Motion Blur* panel of the render settings. This kind of motion blur is done during the render.

**Motion Samples** Set the number of samples to take for each frame. The higher the samples, the smoother the blur effect, but the longer the render, as each virtual intermediate frame has to be rendered.

**Shutter** Time (in frames) the shutter is open. If you are rendering at 24 fps, and the Shutter is set to 0.5, the time in between frames is 41.67 ms, so the shutter is open for half that, 20.83 ms.

---

**Note:** Samples are taken only from the *next* frame, not the previous one. Therefore the blurred object will appear to be slightly ahead of how it would look without motion blur.

---

## Vector Blur

**Vector Blur** is faster but sometimes has unwanted side-effects which are sometimes difficult to avoid.

Vector blur is a process done in compositing (post-render time), by rendering the scene without any blur, plus a pass that has movement information for each pixel. This information is a vector map which describes a 2d or 3d direction and magnitude. The compositor uses that data to blur each pixel in the given direction.

## Examples

To better grasp the concept, let's assume that we have a cube 2 units wide, uniformly moving 1 unit to the right at each frame.

*Image 1* shows a render of frame 1 without Motion Blur; *Image 2* shows a render of frame 2. The scale beneath the cube helps in appreciating the movement of 1 Blender unit.

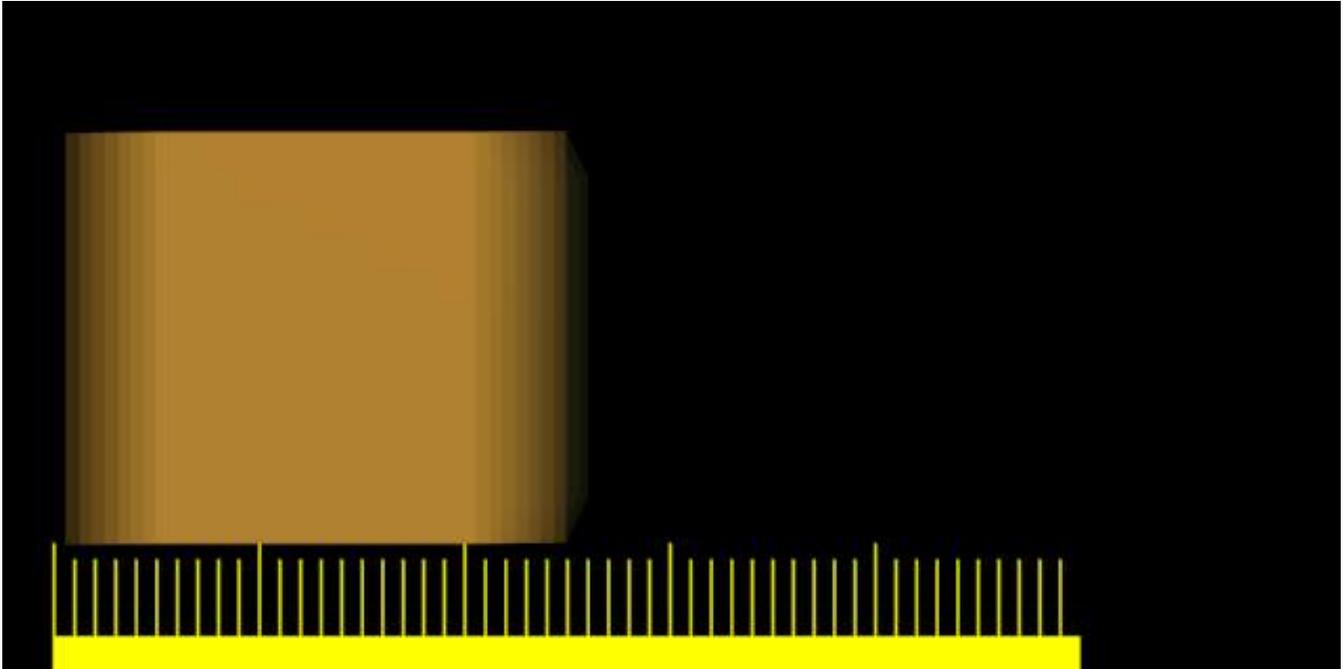


*Image 3* shows the rendering of frame 1 when Sampled Motion Blur is enabled and 8 'intermediate' frames are computed. *Shutter* is set to 0.5 - thus the image 8 samples are rendered between frame 1 and halfway to frame 2.

*Image 4* and *Image 5* show the effect of increasing shutter values. A value greater than 1 is physically impossible in a real-world camera, but can be used to exaggerate the effect.



Better results than those shown can be obtained by using higher samples than 8, but, of course, since as many *separate* renders as samples are needed, a Motion Blur render takes that many times more time than a non-Motion Blur one.



## Hints

Sampled Motion Blur can be used as an additional form of *Anti-Aliasing*, since aliasing artifacts are computed differently for each sample and averaged together at the end.

## Anti-Aliasing

A computer generated image is made up of pixels; each pixel can of course only be a single color. In the rendering process the rendering engine must therefore assign a single color to each pixel on the basis of what object is shown in that pixel. This often leads to poor results, especially at sharp boundaries, or where thin lines are present, and it is particularly evident for oblique lines.

To overcome this problem, which is known as *Aliasing*, it is possible to resort to an *Anti-Aliasing* technique. Basically, each pixel is ‘oversampled’, by rendering it as if it were 5 pixels or more, and assigning an ‘average’ color to the rendered pixel.

The buttons to control *Anti-Aliasing*, or *OverSampling (OSA)*, are below the rendering button in the *Render Panel (Render Panel)*.

## Options

**Anti-Aliasing (check box)** Enables oversampling

**5 / 8 / 11 / 16** The number of samples to use. The values 5, 8, 11, 16 are preset numbers in specific sample patterns; a higher value produces better edges, but slows down the rendering.

By default, we use in Blender a fixed “Distributed Jitter” table. The samples within a pixel are distributed and jittered in a way that guarantees two characteristics:

- Each sample has equal distances to its neighbor samples
- The samples cover all sub-pixel positions equally, both horizontally and vertically

The images below show Blender sample patterns for 5, 8, 11 and 16 samples. To show that the distribution is equalized over multiple pixels, the neighbor pixel patterns were drawn as well. Note that each pixel has an identical pattern.



**Full Sample** For every anti-aliasing sample, save the entire Render Layer results. This solves anti-aliasing issues with compositing.

### Filtering

When the samples have been rendered, we've got color and alpha information available per sample. It then is important to define how much each sample contributes to a pixel.

The simplest method is to average all samples and make that the pixel color. This is called using a "Box Filter". The disadvantage of this method is that it doesn't take into account that some samples are very close to the edge of a pixel, and therefore could influence the color of the neighbor pixel(s) as well.

Filter menu: Set The filter type to use to 'average' the samples:

**Box** A low-quality box-shaped curve

---

**Note:** This filter is relatively low quality. you can see that only the samples within the pixel itself are added to the pixel's color. For the other filters, the formula ensures that a certain amount of the sample color gets distributed over the other pixels as well.

---

**Tent** A simplistic filter that gives sharp results

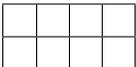
**Quadratic** A Quadratic curve

**Cubic** A Cubic curve

**Gauss** Gaussian distribution, the most blurry

**Catmull-Rom** Catmull-Rom filter, gives the most sharpening

**Mitchell-Netravali** a good all-around filter that gives reasonable sharpness



### Filter Size

Making the filter size value smaller will squeeze the samples more into the center, and blur the image more. A larger filter size makes the result sharper. Notice that the last two filters also have a negative part; this will give an extra sharpening result.

### Examples

### Render Quality

Many factors go into the quality of the rendered image. Rendering a scene without changing any of the render settings is probably going to produce a rather unpleasant image. In previous chapters, you have learned how to model, shade, texture, and light scenes. Optimizing settings with respect to those areas will help to produce quality images, but there are some important settings that come into play before pressing the render button. These can directly affect the look of the rendered image.

The next section covers render layers and render passes, both of which allow you to compose an image from several elements of a scene. In some cases it is necessary to render effects straight out of the renderer, rather than creating them in "post."

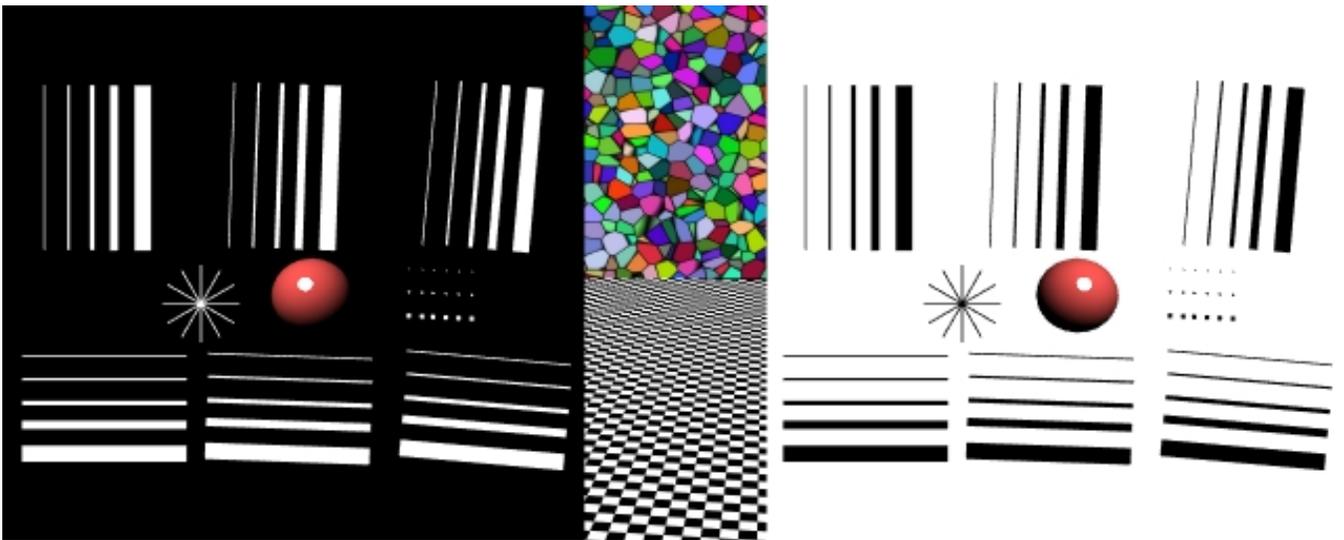
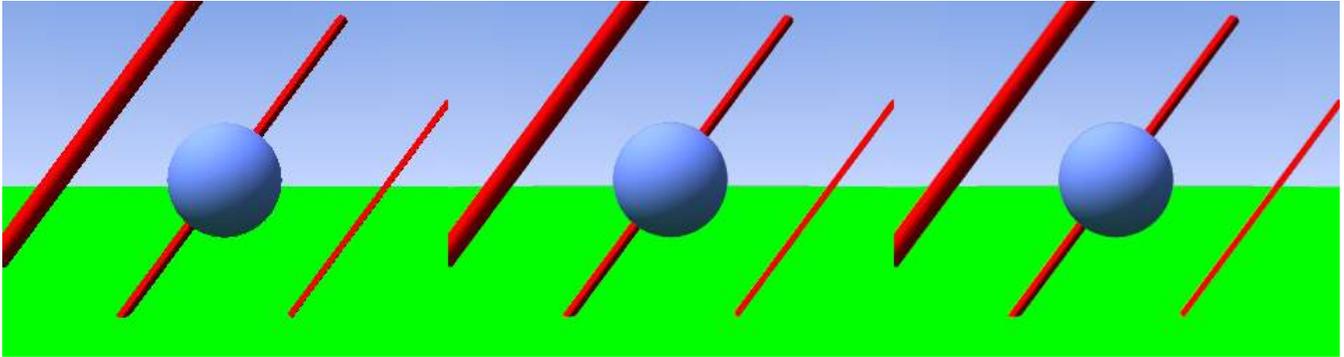


Fig. 2.2277: AA 8, Box filter

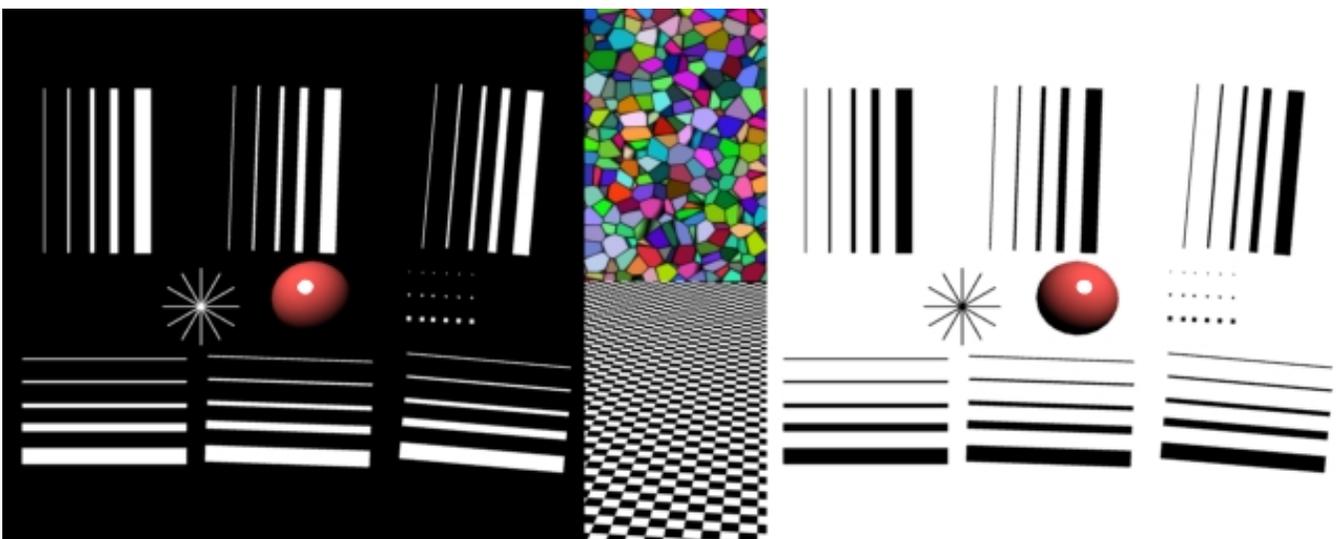


Fig. 2.2278: AA 8, Tent filter

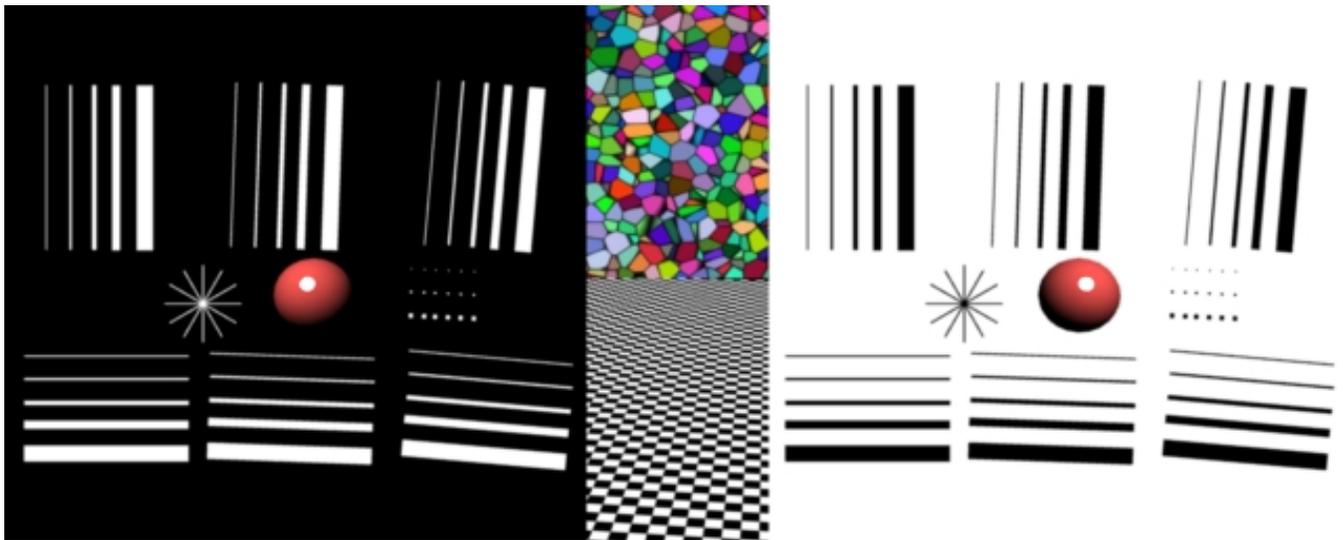


Fig. 2.2279: AA 8, Quadratic filter

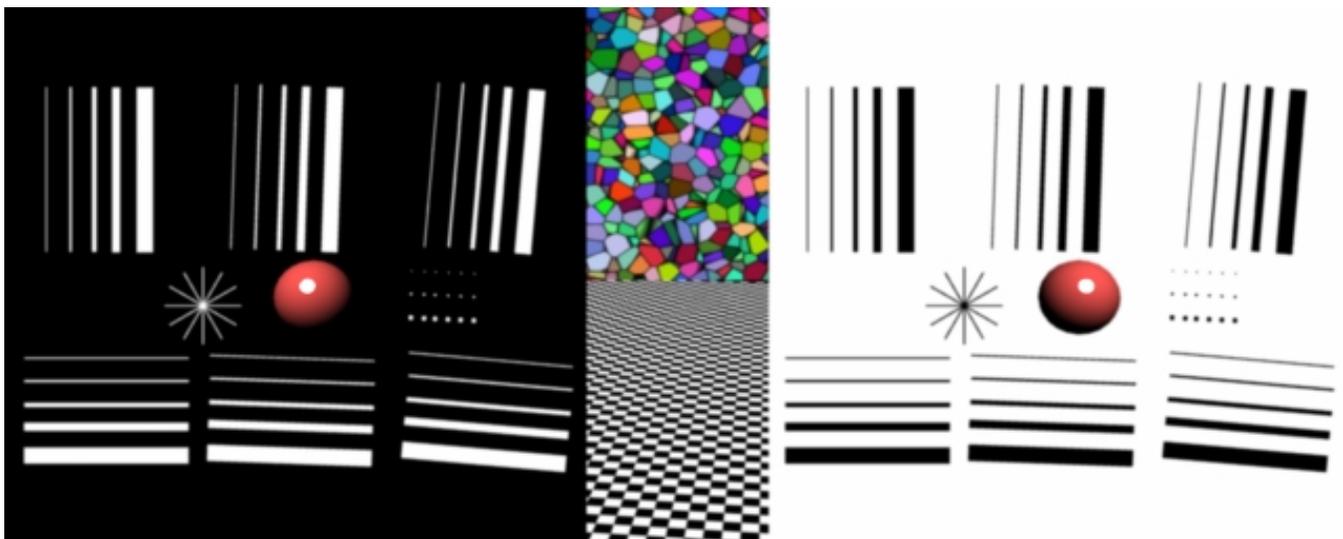


Fig. 2.2280: AA 8, Cubic filter

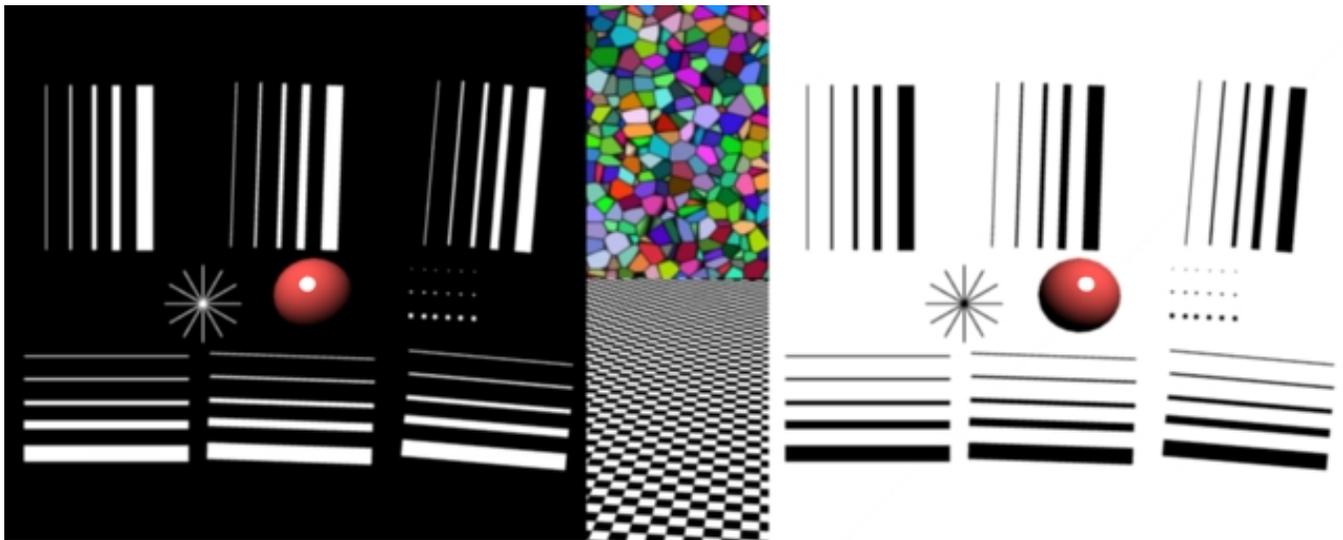


Fig. 2.2281: AA 8, Gaussian filter

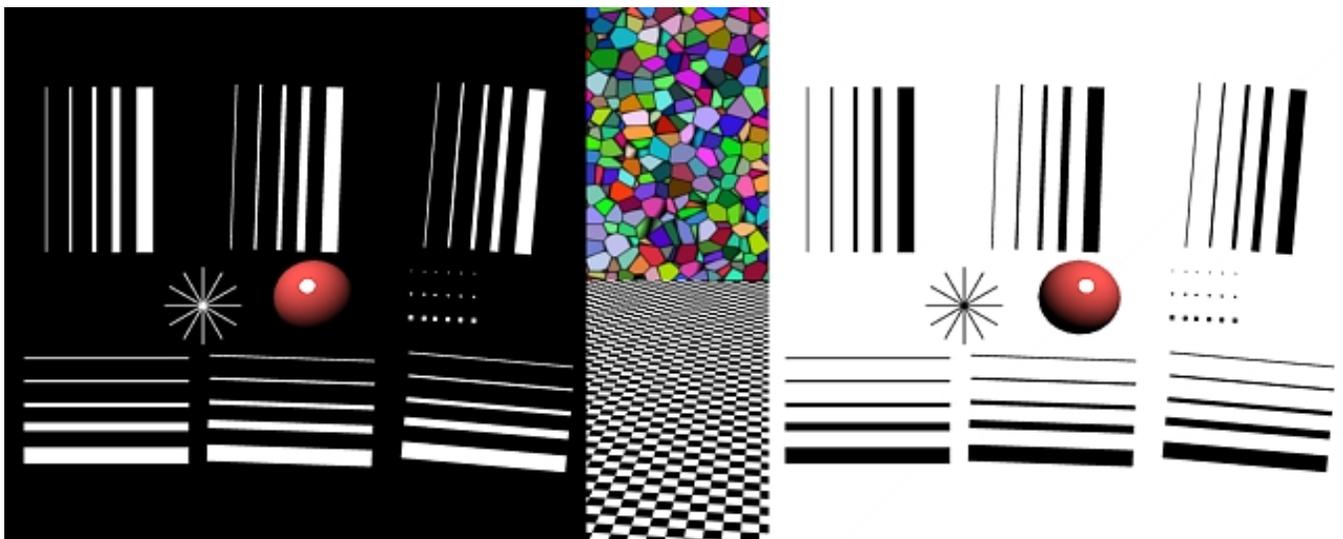


Fig. 2.2282: AA 8, Catmull-Rom filter

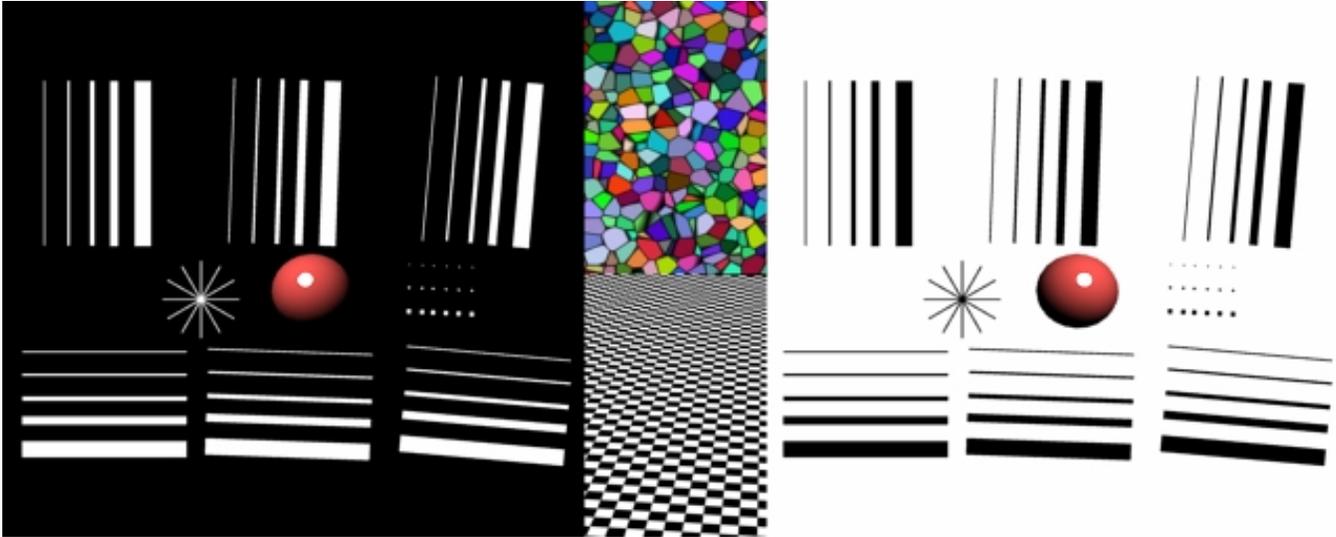


Fig. 2.2283: AA 8, Mitchell-Netravali filter

### ***Color Management and Exposure***

One important aspect of 3D rendering that is often overlooked is color management. Without color management, or more commonly, linear rendering, render engines interpret scene lighting correctly, but display them incorrectly on your monitor. Blender simplifies this workflow, but it is important to know how the color space of a rendered image factors into your pipeline.

### ***Anti-Aliasing***

Anti-Aliasing removes jagged edges that appear in contrasting areas of color. This is a very important aspect of render quality. Without this render setting, images usually appear particularly CG and amateur.

### ***Exposure (Lighting)***

Exposure is, in physical terms, the length of time a camera's film or sensor is exposed to light. Longer exposure times create a brighter image. In CG, the recorded light values are offset to simulate longer or shorter exposures. This can be achieved through lighting settings, or better, through [Color Management settings](#)

### ***Motion Blur***

Cameras have a certain shutter speed, or the length of time the film is exposed to the image. Things that are in motion while the picture is taken will have some degree of blurring. Faster-moving objects will appear more blurred than slower objects. This is an important effect in CG because it is an artifact that we expect to see, and when it is missing, an image may not be believable.

## **Performance Considerations**

### **Optimizing Render Performance**

"A watched pot never boils" is the old saying, but you may wonder why your render takes so long to create, or worse, crashes mid-way through! Well, there is lots going on and lots you can do to speed up rendering or enable a complicated render to

complete. Also, it is possible to render a very complicated scene on a mediocre PC by being “render-smart”. Here’s a “top ten” list of things to do or not do in order to speed up rendering or even avoid crashes during scene render. Some options may decrease the quality of your render, but for draft renders you may not care.

If you get the message “Malloc returns nil”, in plain English that means the memory allocator tried to get more physical memory for Blender but came back empty-handed. This means that you do not have enough memory available to render the scene, and Blender cannot continue. You will need to do one or more of the following tasks on this page in order to render.

### Hardware Improvements

- Install more system memory.
- Upgrade your CPU to a multi-core/multiprocessor
- Upgrade your OpenGL video drivers
- Get faster memory, up to your PC’s motherboard limit.
- Use or set up a render farm using all available PCs in your house, or use a render farm.

### Operating System Configuration

- Increase Blender’s processing priority through your OS.
- Increase your swap file space used by the OS for memory swapping. Also called virtual memory pagefile size, up to the size of your physical memory.
- Use a system-monitor to check if any other processes are using significant CPU or RAM, which can be closed.
- Render in *background mode* (from the command line), saves extra memory.

### Blender Settings

- Increase the MEM Cache Limit in the User Preferences System & OpenGL tab.
- Switch to an Orthographic camera, and render your own “parts” of the scene as separate images, and then paste those parts together in GIMP. An old trick in making your own panorama with a real camera is to take three or so pictures of a very wide (beach sunset) scene, where you take one picture, rotate to the right, snap another, then another, and when you get the pictures developed, you overlap them to make a very wide landscape image. Do the same in Blender: render out one shot to a file, then move the camera to look at a different area of the scene, and render that shot. Each shot will be of a smaller area and thus take in fewer polygons/faces. Be sure that when you position your camera that you snap overlapping shots, so that you can then match them up. If you don’t want to use GIMP, you can use compositing nodes and the Translate node to match them up in Blender.
- Minimize the render window (and Blender if rendering to an internal window). ATI users report dramatic speedup on a per frame basis, which adds up over the frame range.
- Use the Big Render script to render sub-sections of the overall image, and then paste them together.

### Scene and Specific Objects

- Remove lamps, or move them to unrendered layers, or tie them to layers.
- Turn off some lamp’s shadows, using only one or two main sun lamps to cast shadows. A few “shadows only” lights will render faster than every light having shadows on.
- Use Buffer Shadows rather than ray-traced Shadows

- Bake your shadows using Render Baking Full Render bake on surfaces that do not move. Use that texture for that mesh, then disable shadows for that material.
- Simplify meshes (remove polygons). The more vertices you have in camera, the more time it takes to render.
- Remove Doubles, or use the Decimator mesh edit feature.
- Remove Subsurf and Multires modifiers.
- Delete backsides of meshes (removing unseen geometry).
- Render just a few objects at a time; in the beginning of your project, render the background objects and sets that will not change and will always be in the background.
- Put the buildings on another layer, and through render layers, don't render them. Then composite them back in later.
- Make the camera static so that you can better accomplish the above two ideas.
- Avoid use of Area lights.
- Make materials Shadeless.
- Render Bake AO and textures, and then make those materials Shadeless.
- Decrease the Clip distance for spot lights.
- Decrease the Clip distance for the camera.
- Turn off world AO.
- Turn off Material SSS.
- Use smaller image textures. A 256x256 image takes only 1% of the memory that a 2k image does, often with no loss of quality in the ultimate render.
- Reduce Subsurf. Each level quadruples (4x) the number of faces from the previous level.
- Reduce Multires.
- Make a matte render of background objects, like buildings, and put the image of them on a billboard in the scene instead of the object themselves. This will reduce vertex/face count.
- if you have lots of linked instances of an object, use DupliFaces, as these are instanced. If you have 100 of them, Blender will only store the geometry for 1 (Instances themselves take a small amount of memory).

## Render Settings

- **Output Panel** - Disable *Edge* rendering. - *Save Buffers*.
  - Render to an Image Editor window, not a pop-up. **Render Window**.
  - Use multiple *Threads* on a multi-core CPU (with multiple *Parts*).
- **Render Layers Panel** - Render only the Layers of interest. - Render with all lights set to one simple spot (enter its name in the *Light:* field). - Render with one material override (enter its name in the *Mat:* field).
  - Disable unnecessary Render Passes, such as *Z*, or only render the pass of interest, such as *Diffuse*.
- **Render Panel** - Turn off *Shadows*. - Turn off *Environment Mapping*. - Turn off *Panoramic Rendering*. - Turn off *Raytracing*. - Turn off SSS Subsurface Scattering. - Turn off or lower oversampling/aliasing *OSA*. - Turn off or lower *Motion Blur*.
  - Render in *Parts*. This will also allow you to render HUGE images on a weak PC. On a multi-core PC, it will assign a thread to each part as well.
  - Increase the octree resolution.

- Render at a percentage size of your final resolution (like 25%).
- Turn off *Fields* rendering.
- Use *Border* rendering to render a subset of the full image.
- **Anim Panel**
  - Decrease the frame count of the animation (and use a lower framerate for the same duration of animation). For example, render 30 frames at 10 frames per second for a 3-second animation, instead of 75 frames at 25 frames per second.
- **Bake Panel**
  - Bake Full Render - create a UV Texture that colors the objects based on materials, and then use that UV Texture shadeless instead of the material.
  - Bake Ambient Occlusion only.
  - Bake textures for objects.
  - Baking Normals or Displacement does not speed up render time, and are used for other things.
- **Format Panel** - Render at a lower resolution. Smaller pictures take less time to render. - Choose a faster CODEC or CODEC settings. - Render in black and white (*BW* button). - If using FFMPEG, do not activate *Multiplex audio*. - If using FFMPEG, *Autosplit Output* (*Video* panel button).
  - Render only RGB if you just need color; the A channel (*RGBA* button) takes more memory and is unused when saving a movie file.

## Multi-Pass Compositing

Another strategy that can be used to address the problem of long (re-)render times is to structure your workflow from the ground up so that you make aggressive use of *compositing*, as described in the “Post-Production” section. In this approach, you break down each shot into components that can be rendered separately, then you combine those separately-rendered elements to achieve the finished clip. For instance:

- If the camera isn’t moving, then neither is the background: only a single frame is needed. (The same is true of any non-moving object within the frame.) These individual elements, having been generated *once*, can be re-used as many times as necessary over as many frames as necessary.
- Both shadows and highlights can be captured separately from the objects that are being illuminated or shadowed, such that the intensity, color, and depth of the effect can be adjusted later without re-rendering.
- Start by using lights that do not cast shadows. (Shadow calculations are big time-killers.) Then, use “shadow-only” lights (which cast shadows, but do not cast light) to create shadows *only* where you judge that they are actually necessary. (It is very often the case that only a few of the shadows which could exist in the scene actually matter, and that the rest of them simply won’t be noticed.)
- Tricky lighting situations can be avoided by handling the objects separately, then combining the individually-rendered clips and “tweaking” the result.

This is a very familiar idea. Modern sound recordings, for example, always use a “multi-track” approach. Individual components of the song are captured separately and in isolation, then the components are “mixed” together. The “final mix” then goes through additional processing stages, called *mastering*, to produce the finished product(s). (In fact, the features and design of modern sound-processing software are directly comparable to that of Blender’s node-based compositor.)

There are compelling advantages to this approach:

- You have options. If something is “not quite right,” you don’t necessarily have to start over from scratch.

- In practice, the deadline-killer is *re-* rendering, which ordinarily must be done (in its entirety) just because “‘one little thing’ about the shot is wrong.” Compositing helps to avoid this, because (ideally...) only the specific parts that are found to be in error must be repeated. (Or, maybe, the error can be blocked out with a “garbage matte” and a corrected version can be inserted in its place. No one will ever know!)
- It’s also possible that you find yourself saying, “okay, that’s *almost* what I wanted, but now I’d like to *add* this and maybe *take away* that.” A compositing-based approach enables you to do just that, and furthermore, to do so *non-destructively*. In other words, having generated the “addition” (or the “mask”) as a separate channel of information, you can now fine-tune its influence in the overall “mix, ” or even change your mind and remove it altogether, all without permanently altering anything.
- By and large, these stages work *two-* dimensionally, manipulating what is by that time “a raster bitmap with R, G, B, Alpha (*transparency...*) and Z-Depth information,” so they’re consistently fast.
- Since each discrete rendering task has been simplified, the computer can carry them out using much fewer resources.
- The tasks can be distributed among several different computers ... even less-powerful ones
- “After all, the scene doesn’t actually have to be *physically perfect*, to be *convincing*. ” A compositing-based approach lets you take full advantage of this. You can focus your attention (and Blender’s) upon those specific aspects of the scene which will actually make a noticeable difference. It is possible to save a considerable amount of time by consciously choosing to exclude less-important aspects which (although “technically correct”) probably won’t be noticed.

Of course, this approach is not without its own set of trade-offs. You must devise a workable asset-management system for keeping track of exactly what material you have, where it is, whether it is up-to-date, and exactly how to re-create it. You must understand and use the “library linking” features of Blender to allow you to refer to objects, nodes, materials, textures and scenes in a carefully-organized collection of other files. You need to have a very clear notion, *in advance*, of exactly what the finished shot must consist of and what the task breakdown must be. You must be a scrupulous note-taker and record-keeper. But sometimes this is the best way, if not the *only* way, to accomplish a substantial production.

## 2.8.3 Cycles Render Engine

### Introduction

Cycles is a ray tracing renderer focused on interactivity and ease of use, while still supporting many production features.

It is bundled as an add-on that is enabled by default. To use Cycles, it must be set as the active render engine in the top header. Once that is done, interactive rendering can be started by setting a 3D view editor to draw mode Rendered using `Shift-Z`. The render will keep updating as modifications are done, such as changing a material color, changing a lamp’s intensity or moving objects around.

Cycles may be able to use your GPU (Graphics Processing Unit, or Graphics Card) to render. To see if and how you can use your GPU for rendering, see the documentation on [GPU Rendering](#).

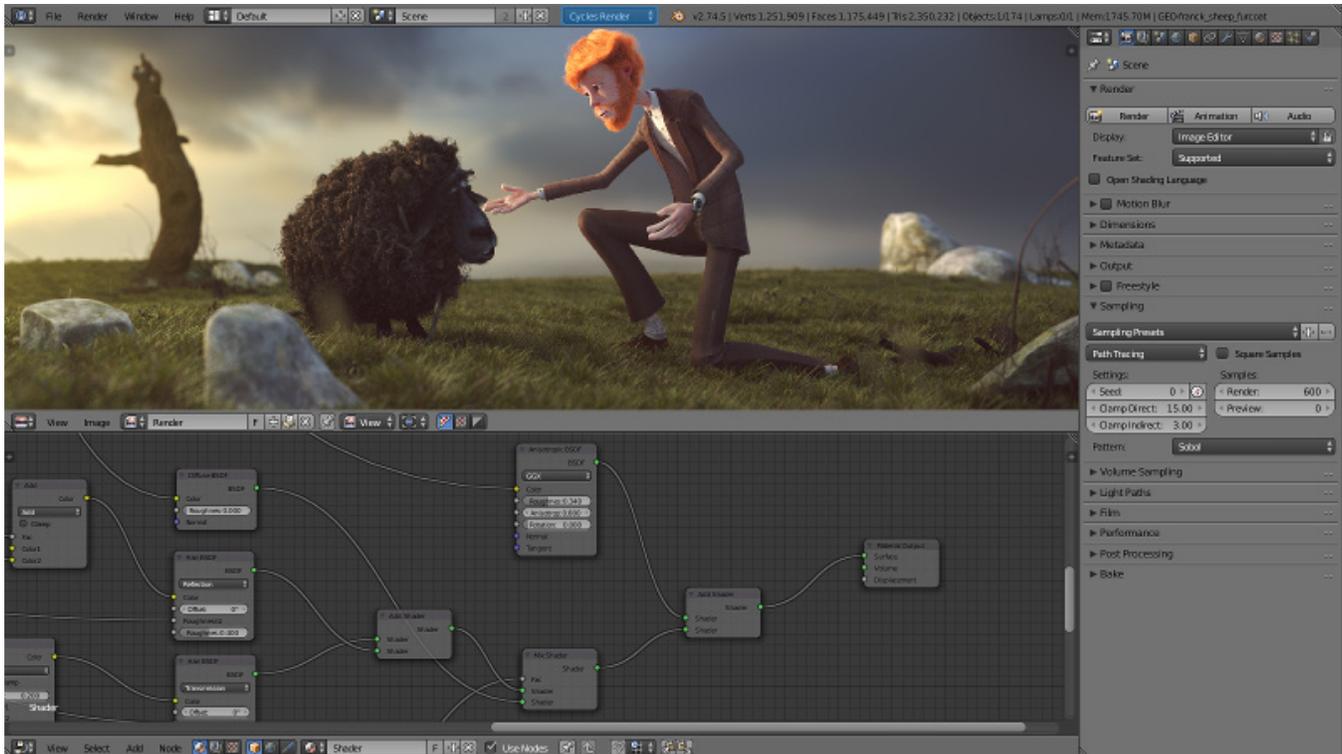
#### See also:

[Developer documentation](#) is also available.

### Render Settings

#### Integrator

The integrator is the rendering algorithm used to compute the lighting. Cycles currently supports a path tracing integrator with direct light sampling. It works well for various lighting setups, but is not as suitable for caustics and some other complex lighting situations.



Rays are traced from the camera into the scene, bouncing around until they find a light source such as a lamp, an object emitting light, or the world background. To find lamps and surfaces emitting light, both indirect light sampling (letting the ray follow the surface BSDF) and direct light sampling (picking a light source and tracing a ray towards it) are used.

## Scene Settings

**Sampling** There are two integrator modes that can be used: path tracing and branched path tracing. The **path tracing integrator** is a pure path tracer; at each hit it will bounce light in one direction and pick one light to receive lighting from. This makes each individual sample faster to compute, but will typically require more samples to clean up the noise.

**Render Samples** Number of paths to trace for each pixel in the final render. As more samples are taken, the solution becomes less noisy and more accurate.

**Preview Samples** Number of samples for viewport rendering.

The **branched path tracing integrator** (formerly called non-progressive integrator) is similar, but at the first hit it will split the path for different surface components and will take all lights into account for shading instead of just one. This makes each sample slower, but will reduce noise, especially in scenes dominated by direct or one-bounce lighting. To get the same number of diffuse samples as in the path tracing integrator, note that e.g. 250 path tracing samples = 10 AA samples x 25 diffuse samples. The Sampling panel shows this total number of samples.

**AA Render Samples** Number of samples to take for each pixel in the final render. More samples will improve antialiasing.

**AA Preview Samples** Number of samples for viewport rendering.

**Diffuse Samples** Number of diffuse bounce samples to take for each AA sample.

**Glossy Samples** Number of glossy bounce samples to take for each AA sample.

**Transmission Samples** Number of transmission bounce samples to take for each AA sample.

**AO Samples** Number of ambient occlusion samples to take for each AA sample.

**Mesh Light Samples** Number of mesh light samples to take for each AA sample.

**Subsurface Samples** Number of subsurface scattering samples to take for each AA sample.

For both integrators the noise pattern can be controlled.

**Seed** Random number generator seed; each different value gives a different noise pattern.

## Bounces

**Max Bounces** Maximum number of light bounces. For best quality, this should be set to the maximum. However, in practice, it may be good to set it to lower values for faster rendering. Setting it to maximum 0 bounces results in direct lighting only.

**Min Bounces** Minimum number of light bounces for each path, after which the integrator uses Russian Roulette to terminate paths that contribute less to the image. Setting this higher gives less noise, but may also increase render time considerably. For a low number of bounces, it's strongly recommended to set this equal to the maximum number of bounces.

**Diffuse Bounces** Maximum number of diffuse bounces.

**Glossy Bounces** Maximum number of glossy bounces.

**Transmission Bounces** Maximum number of transmission bounces.

## Transparency

**Transparency Max** Maximum number of transparency bounces.

**Transparency Min** Minimum number of transparency bounces, after which Russian Roulette termination is used.

**Transparent Shadows** For direct light sampling, use transparency of surfaces in between to produce shadows affected by transparency of those surfaces.

## Tricks

**No Caustics** While in principle path tracing supports rendering of caustics with a sufficient number of samples, in practice it may be inefficient to the point that there is just too much noise. This option makes it possible to disable them entirely.

**Filter Glossy** When using a value higher than 0.0, this will blur glossy reflections after blurry bounces, to reduce noise at the cost of accuracy. 1.0 is a good starting value to tweak.

Some light paths have a low probability of being found while contributing much light to the pixel. As a result these light paths will be found in some pixels and not in others, causing fireflies. An example of such a difficult path might be a small light that is causing a small specular highlight on a sharp glossy material, which we are seeing through a rough glossy material. In fact in such a case we practically have a caustic.

With path tracing it is difficult to find the specular highlight, but if we increase the roughness on the material, the highlight gets bigger and softer, and so easier to find. Often this blurring will hardly be noticeable, because we are seeing it through a blurry material anyway, but there are also cases where this will lead to a loss of detail in lighting.

**Clamp Samples** This option will clamp all samples to a maximum intensity they can contribute to the pixel, again to reduce noise at the cost of accuracy. With value 0.0 this option is disabled; lower values clamp more light away.

If the image has fireflies, there will be samples that contribute very high values to pixels, and this option provides a way to limit that. However note that as you clamp out such values, bright colors in other places where there is no noise will be lost as well. So this is a balance between reducing the noise and keeping the image from losing its intended bright colors.

**Motion Blur** Camera and object motion blur rendering can be enabled per scene, and affects all render layers. This will take the camera and object motion into account to blur objects along 3 points through the previous, current and next frame. Currently scale motion is not supported, only object transformations like translation and rotation. Viewport rendering currently will not show motion blur.

If there are particles or other physics system in a scene, be sure to bake them before rendering, otherwise you might not get correct or consistent motion.

**Shutter** Time between frames over which motion blur is computed. Shutter time 1.0 blurs over the length of 1 frame, 2.0 over the length of two frames, from the previous to the next.

**Warning:** An object modifier setup that changes mesh topology over time will cause severe problems. Common examples of this are animated booleans, deformation before edge-split, remesh, skin or decimate modifiers.

## Material Settings

**Multiple Importance Sample** By default objects with emitting materials use both direct and indirect light sampling methods, but in some cases it may lead to less noise overall to disable direct light sampling for some materials. This can be done by disabling the *Multiple Importance Sample* option. This is especially useful on large objects that emit little light compared to other light sources.

This option will only have an influence if the material contains an emission node; it will be automatically disabled otherwise.

## World Settings

**Multiple Importance Sample** By default lighting from the world is computed solely with indirect light sampling. However for more complex environment maps this can be too noisy, as sampling the BSDF may not easily find the highlights in the environment map image. By enabling this option, the world background will be sampled as a lamp, with lighter parts automatically given more samples.

**Map Resolution** When Multiple Importance Sample is enabled, this specifies the size of the importance map (resolution x resolution). Before rendering starts, an importance map is generated by “baking” a grayscale image from the world shader. This will then be used to determine which parts of the background are light and so should receive more samples than darker parts. Higher resolutions will result in more accurate sampling but take more setup time and memory.

## Lamp Settings

**Multiple Importance Sample** By default lamps use only direct light sampling. For area lights and sharp glossy reflections, however, this can be noisy, and enabling this option will enable indirect light sampling to be used in addition to reduce noise.

**Samples** For the branch path tracing integrator, this specifies the number of direct light samples per AA sample. Point lamps might need only one sample, while area lamps typically need more.

**Max Bounces** The maximum amount of bounces this light will contribute to the scene.

**Portal** Only available for Area lamps. This setting enables area lamps to function as a light portal, helping to sample the environment lamp and therefore improving convergence. Note that this will make the area lamp itself invisible.

**Volume Render Settings** The scene has these settings:

**Step Size** Distance between volume shader samples when rendering the volume. Lower values give more accurate and detailed results but also increased render time.

**Max Steps** Maximum number of steps through the volume before giving up, to protect from extremely long render times with big objects or small step sizes.

The world and materials have the following setting:

**Homogeneous Volume** Assume volume has the same density everywhere (not using any textures), for faster rendering. For example absorption in a glass object would typically not have any textures, and by knowing this we can avoid taking small steps to sample the volume shader.

**Sampling Method** Options are “Multiple Importance”, “Distance” or “Equiangular”. If you’ve got a pretty dense volume that’s lit from far away then distance sampling is usually more efficient. If you’ve got a light inside or near the volume then equiangular sampling is better. If you have a combination of both, then the multiple importance sampling will be better.

## Light Paths

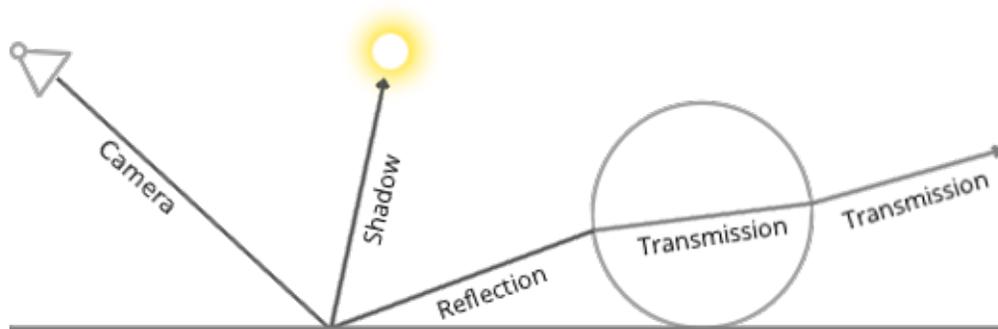
**Ray Types** Ray types can be divided into four categories:

- Camera: the ray comes straight from the camera
- Reflection: the ray is generated by a reflection off a surface
- Transmission: the ray is generated by a transmission through a surface
- Shadow: the ray is used for (transparent) shadows

Reflection and transmission rays can further have these properties:

- Diffuse: the ray is generated by a diffuse reflection or transmission (translucency)
- Glossy: the ray is generated by a glossy specular reflection or transmission
- Singular: the ray is generated by a perfectly sharp reflection or transmission

The Light Path node can be used to find out the type of ray the shading is being computed for.



**Bounce Control** The maximum number of light bounces can be controlled manually. While ideally this should be infinite, in practice a smaller number of bounces may be sufficient, or some light interactions may be intentionally left out for faster convergence. The number of diffuse reflection, glossy reflection and transmission bounces can also be controlled individually.

Light paths are terminated probabilistically when specifying a minimum number of light bounces lower than the maximum. In that case paths longer than minimum will be randomly stopped when they are expected to contribute less light to the image. This will still converge to the same image, but renders faster while possibly being noisier.

A common source of noise is caustics, which are diffuse bounces followed by a glossy bounce (assuming we start from the camera). An option is available to disable these entirely.

**Transparency** The transparent BSDF (Bidirectional scattering distribution function) shader is given special treatment. When a ray passes through it, light passes straight on, as if there was no geometry there. The ray type does not change when passing through a transparent BSDF.

Alpha pass output is also different for the transparent BSDF. Other transmission BSDFs are considered opaque, because they change the light direction. As such they can't be used for alpha-over compositing, while this is possible with the transparent BSDF.

The maximum number of transparent bounces is controlled separately from other bounces. It is also possible to use probabilistic termination of transparent bounces, which might help rendering many layers of transparency.

Note that while semantically the ray passes through as if no geometry was hit, rendering performance is affected as each transparency step requires executing the shader and tracing a ray.

**Ray Visibility** Objects can be set to be invisible to particular ray types:

- Camera
- Diffuse reflection
- Glossy reflection
- Transmission
- Shadow

This can be used, for example, to make an emitting mesh invisible to camera rays. For duplicators, visibility is inherited; if the parent object is hidden for some ray types, the children will be hidden for these too.

In terms of performance, using these options is more efficient than using a shader node setup that achieves the same effect. Objects invisible to a certain ray will be skipped in ray traversal already, leading to fewer rays cast and shaders executed.

## Render Layers and Passes

**Layers** This section covers only the Render Layer settings appropriate for the Blender Render engine. For the engine-independent settings, see [this section](#).

**Exclude** Scene layers are shared between all render layers; however sometimes it's useful to leave out some object influence for a particular render layer. That's what this option allows you to do.

### Lighting Passes

**Diffuse Direct** Direct lighting from diffuse BSDFs. We define direct lighting as coming from lamps, emitting surfaces, the background, or ambient occlusion after a single reflection or transmission off a surface. BSDF color is not included in this pass.

**Diffuse Indirect** Indirect lighting from diffuse BSDFs. We define indirect lighting as coming from lamps, emitting surfaces or the background after more than one reflection or transmission off a surface. BSDF color is not included in this pass.

**Diffuse Color** Color weights of diffuse BSDFs. These weights are the color input socket for BSDF nodes, modified by any Mix and Add Shader nodes.

**Glossy Direct, Indirect, Color** Same as above, but for glossy BSDFs.

**Transmission Direct, Indirect, Color** Same as above, but for transmission BSDFs.

**Subsurface Direct, Indirect, Color** Same as above, but for subsurface BSDFs.

**Emission** Emission from directly visible surfaces.

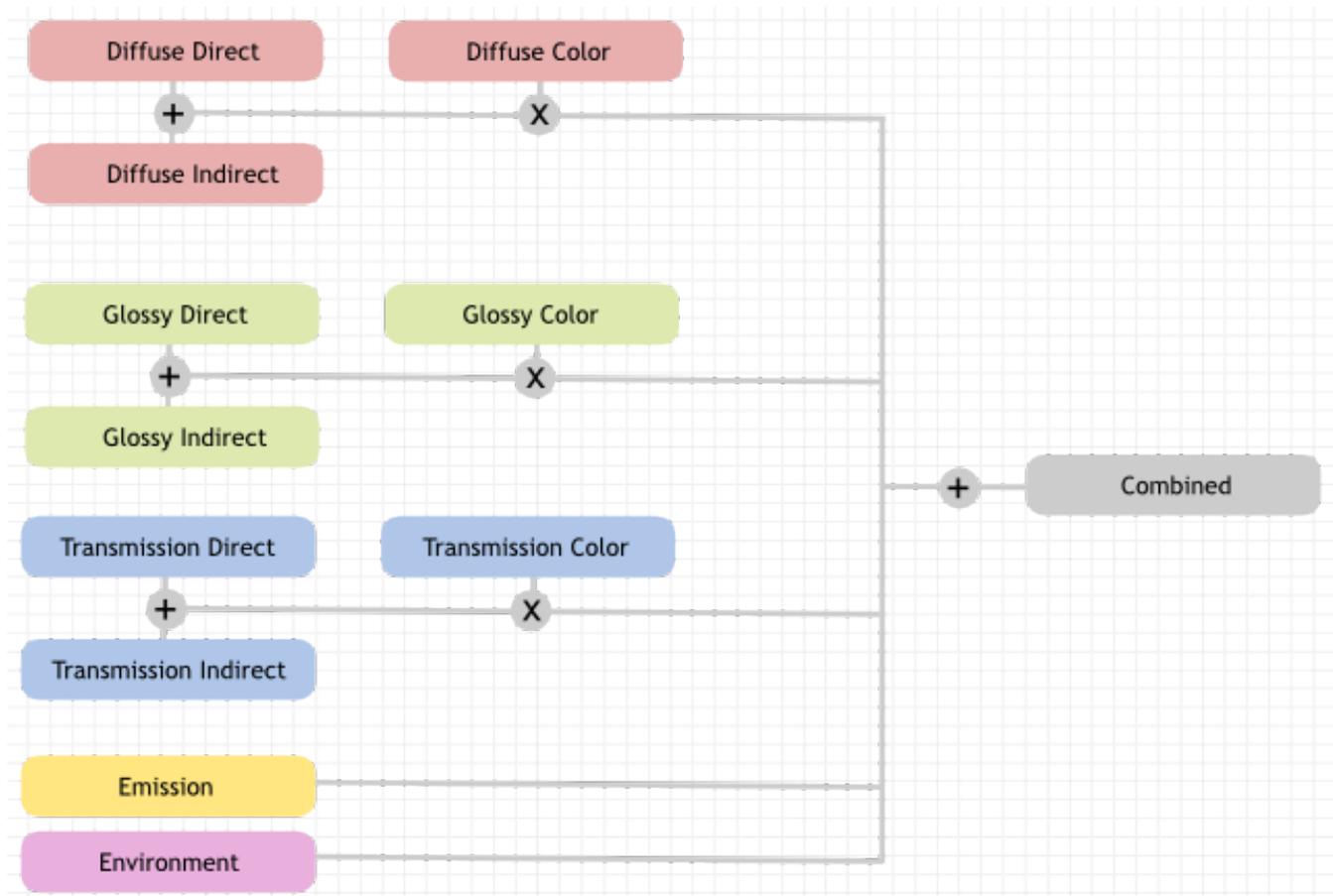
**Environment** Emission from the directly visible background. When the film is set to transparent, this can be used to get the environment color and composite it back in.

**Shadow** Shadows from lamp objects.

**Ambient Occlusion** Ambient occlusion from directly visible surfaces. BSDF color or AO factor is not included; i.e. it gives a 'normalized' value between 0 and 1.

Note that transparent BSDFs are given special treatment a fully transparent surface is treated as if there is no surface there at all; a partially transparent surface is treated as if only part of the light rays can pass through. This means it is not included in the Transmission passes; for that a glass BSDF with index of refraction 1.0 can be used.

**Combining** All these lighting passes can be combined to produce the final image as follows:



### Data Passes

**Combined** The final combination of render passes with everything included.

**Z** The

**Mist** Mist value between 0.0 and 1.0, using settings from the Mist Pass panel in world properties.

**Normal** Surface normal used for shading.

**Vector** Motion vectors for the vector blur node. The four components consist of 2D vectors giving the motion towards the next and previous frame position in pixel space.

**UV** Default render UV coordinates.

**Object Index** Pass index of object.

**Material Index** Pass index of material.

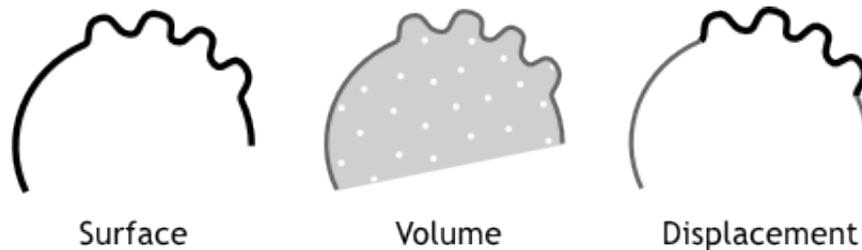
The Z, Object Index and Material Index passes are not anti-aliased. This is done intentionally because such values can't really be blended correctly.

**Alpha Threshold** Z, Index, normal, UV and vector passes are only affected by surfaces with alpha transparency equal to or higher than this threshold. With value 0.0 the first surface hit will always write to these passes, regardless of transparency. With higher values surfaces that are mostly transparent can be skipped until an opaque surface is encountered.

## Materials

### Introduction

Materials define the appearance of meshes, curves and other objects. They consist of three shaders, defining the appearance of the surface of the mesh, the volume inside the mesh, and displacement of the surface of the mesh.



**Surface Shader** The surface shader defines the light interaction at the surface of the mesh. One or more BSDF s specify if incoming light is reflected back, refracted into the mesh, or absorbed.

Emission defines how light is emitted from the surface, allowing any surface to become a light source.

**Volume Shader** When the surface shader does not reflect or absorb light, it enters into the volume. If no volume shader is specified, it will pass straight through to the other side of the mesh.

If it is defined, a volume shader describes the light interaction as it passes through the volume of the mesh. Light may be scattered, absorbed, or emitted at any point in the volume.

A material may have both a surface and a volume shader, or only one of either. Using both may be useful for materials such as glass, water or ice, where you want some of the light to be absorbed as it passes through the surface, combined with e.g. a glass or glossy shader at the surface.

**Displacement** The shape of the surface and the volume inside it may be altered by displacement shaders. This way, textures can then be used to make the mesh surface more detailed.

Depending on the settings, the displacement may be virtual, only modifying the surface normals to give the impression of displacement, which is known as bump mapping, or a combination of real and virtual displacement.

**Energy Conservation** The material system is built with physics-based rendering in mind, cleanly separating how a material looks and which rendering algorithm is used to render it. This makes it easier to achieve realistic results and balanced lighting, though there are a few things to keep in mind.

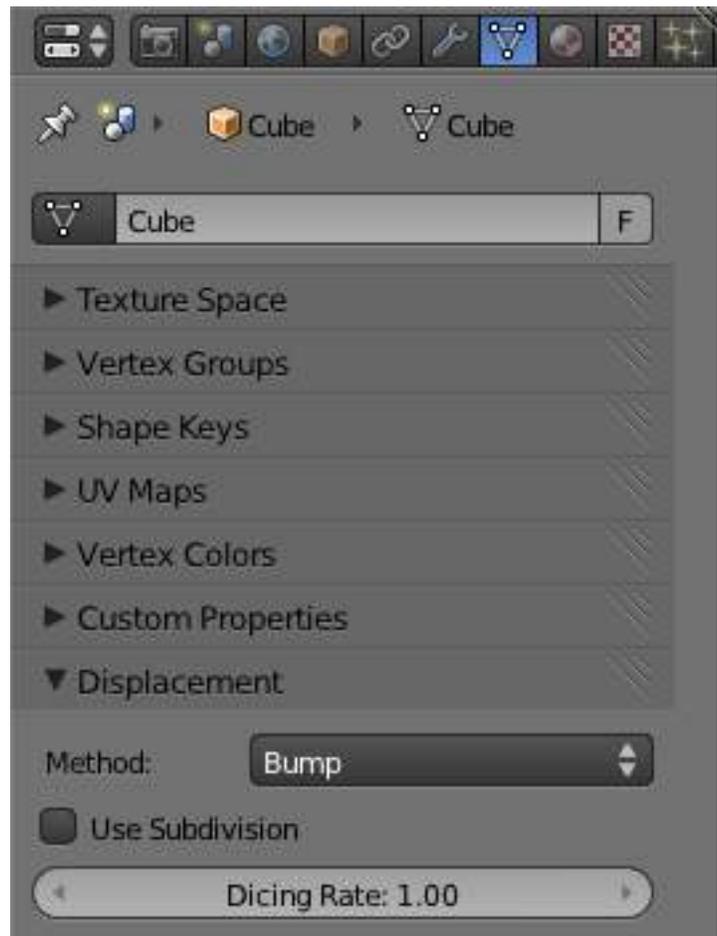
In order for materials to work well with global illumination, they should be, speaking in terms of physics, energy conserving. That means they cannot reflect more light than comes in. This property is not strictly enforced, but if colors are in the range 0.0 to 1.0, and BSDF s are only mixed together with the Mix Shader node, this will automatically be true.

It is however possible to break this, with color values higher than 1.0 or using the Add Shader node, but one must be careful when doing this to keep materials behaving predictably under various lighting conditions. It can result in a reflection adding light into the system at each bounce, turning a BSDF into a kind of emitter.

## Displacement

*Implementation not finished yet, marked as an experimental feature.*

The shape of the surface and the volume inside its mesh may be altered by the displacement shaders. This way, textures can then be used to make the mesh surface more detailed.



**Type** Depending on the settings, the displacement may be virtual, only modifying the surface normals to give the impression of displacement, known as bump mapping, or a combination of real and virtual displacement. The displacement type options are:

**True Displacement** Mesh vertices will be displaced before rendering, modifying the actual mesh. This gives the best quality results, if the mesh is finely subdivided. As a result this method is also the most memory intensive.

**Bump Mapping** When executing the surface shader, a modified surface normal is used instead of the true normal. This is a quick alternative to true displacement, but only an approximation. Surface silhouettes will not be accurate and there will be no self-shadowing of the displacement.

**Displacement + Bump** Both methods can be combined, to do displacement on a coarser mesh, and use bump mapping for the final details.

**Subdivision** *Implementation not finished yet, marked as an experimental feature.*

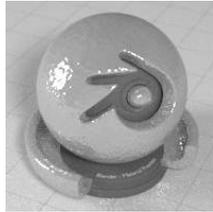


Fig. 2.2284: Bump Mapped Displacement

When using *True Displacement* or *Displacement + Bump* and enabling *Use Subdivision* you can reduce the **Dicing Rate** to subdivide the mesh. This only affects the render and does not show in the viewport (but does show in *Rendered Shading Mode*). Displacement can also be done manually by use of the Displacement Modifier.

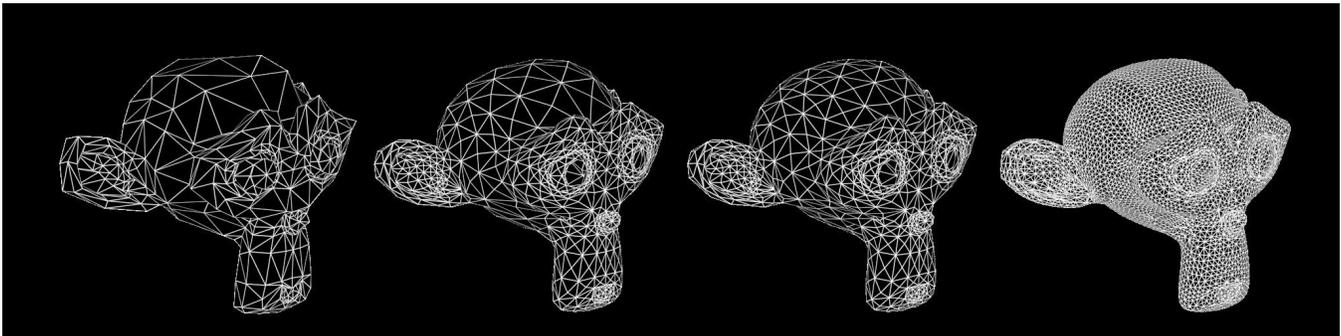


Fig. 2.2285: Subdivision Off - On, Dicing Rate 1.0 - 0.3 - 0.05 (Monkeys look identical in viewport, no modifiers)

## Surface

The surface shader defines the light interaction at the surface of the mesh. One or more BSDF s specify if incoming light is reflected back, refracted into the mesh, or absorbed.

Emission defines how light is emitted from the surface, allowing any surface to become a light source.

## Terminology

**BSDF stands for bidirectional scattering distribution function.** It defines how light is reflected and refracted at a surface.

**Reflection BSDF s** Reflect an incoming ray on the same side of the surface.

**Transmission BSDF s** Transmit an incoming ray through the surface, leaving on the other side.

**Refraction BSDF s are a type of Transmission**, Transmitting an incoming ray and changing its direction as it exits on the other side of the surface.

**BSDF Parameters** A major difference from non-physically based renderers is that direct light reflection from lamps and indirect light reflection of other surfaces are not decoupled, but rather handled using a single BSDF. This limits the possibilities a bit, but we believe overall it is helpful in creating consistent-looking renders with fewer parameters to tune.

For glossy BSDF s, **roughness** parameters control the sharpness of the reflection, from 0.0 (perfectly sharp) to 1.0 (very soft). Compared to **hardness** or **exponent** parameters, it has the advantage of being in the range 0.0..1.0, and as a result gives more linear control and is more easily textureable. The relation is roughly:  $roughness = 1 - 1/hardness$

## Volume

Volume rendering can be used to render effects like fire, smoke, mist, absorption in glass, and many other effects that can't be represented by surface meshes alone.

To set up a volume, you create a mesh that defines the bounds within which the volume exists. In the material you typically remove the surface nodes and instead connect volume nodes to define the shading inside the volume. For effects such as absorption in glass you can use both a surface and volume shader. The world can also use a volume shader to create effects such as mist.

**Volume Shaders** We support three volume shader nodes, that model particular effects as light passes through the volume and interacts with it.

- Volume Absorption will absorb part of the light as it passes through the volume. This can be used to shade for example black smoke or colored glass objects, or mixed with the volume scatter node. This node is somewhat similar to the transparent BSDF node, it blocks part of the light and lets other light pass straight through.
- Volume Scatter lets light scatter in other directions as it hits particles in the volume. The anisotropy defines in which direction the light is more likely to scatter. A value of 0 will let light scatter evenly in all directions (somewhat similar to the diffuse BSDF node), negative values let light scatter mostly backwards, and positive values let light scatter mostly forward. This can be used to shade white smoke or clouds for example.
- Emission will emit light from the volume. This can be used to shade fire for example.



Fig. 2.2286: Volume Shader: Absorption / Absorption + Scatter / Emission

**Density** All volume shaders have a density input. The density defines how much of the light will interact with the volume, getting absorbed or scattered, and how much will pass straight through. For effects such as smoke you would specify a density field to indicate where in the volume there is smoke and how much (density bigger than 0), and where there is no smoke (density equals 0).

Volumes in real life consist of particles, a higher density means there are more particles per unit volume. More particles means there is a higher chance for light to collide with a particle and get absorbed or scattered, rather than passing straight through.

## Volume Material

**Interaction with the Surface Shader** A material may have both a surface and a volume shader, or only one of either. Using both may be useful for materials such as glass, water or ice, where you want some of the light to be absorbed as it passes through the surface, combined with e.g. a glass or glossy shader at the surface.

When the surface shader does not reflect or absorb light, it enters into the volume. If no volume shader is specified, it will pass straight through to the other side of the mesh. If it is defined, a volume shader describes the light interaction as it passes through the volume of the mesh. Light may be scattered, absorbed, or emitted at any point in the volume.

**Mesh Topology** Meshes used for volume render should be closed and *manifold*. That means that there should be no holes in the mesh. Each edge must be connected to exactly 2 faces such that there are no holes or T-shaped faces where 3 or more faces are connected to an edge.

Normals must point outside for correct results. The normals are used to determine if a ray enters or exits a volume, and if they point in a wrong direction, or there is a hole in the mesh, then the renderer is unable to decide what is the inside or outside of the volume.

These rules are the same as for rendering glass refraction correctly.

**Volume World** A volume shader can also be applied to the entire world, filling the entire space.

Currently this is most useful for night time or other dark scenes, as the world surface shader or sun lamps will have no effect if a volume shader is used. This is because the world background is assumed to be infinitely far away, which is accurate enough for the sun for example. However for modeling effects such as fog or atmospheric scattering, it is not a good assumption that the volume fills the entire space, as most of the distance between the sun and the earth is empty space. For such effects it is better to create a volume object surrounding the scene. The size of this object will determine how much light is scattered or absorbed.

**Scattering Bounces** Real world effects such as scattering in clouds or subsurface scattering require many scattering bounces. However unbiased rendering of such effects is slow and noisy. In typical movie production scenes only 0 or 1 bounces might be used to keep render times under control. The effect you get when rendering with 0 volume bounces is what is known as “single scattering”, the effect from more bounces is “multiple scattering”.

For rendering materials like skin or milk, the subsurface scattering shader is an approximation of such multiple scattering effects that is significantly more efficient but not as accurate.

For materials such as clouds or smoke that do not have a well defined surface, volume rendering is required. These look best with many scattering bounces, but in practice one might have to limit the number of bounces to keep render times acceptable.

**Limitations** Currently we do not support:

- Correct ray visibility for volume meshes

Not available on GPU:

- Smoke/Fire rendering
- Equi Angular / MIS Volume Sampling
- Volume Multi Light sampling

## Texture Editing

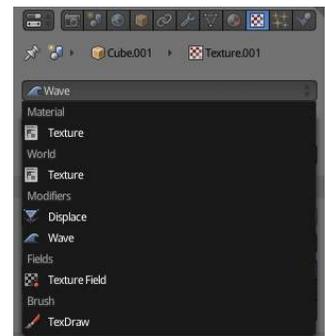
3D viewport draw types, UV mapping, and texture painting work somewhat differently when Cycles is enabled. UV Maps no longer get image textures assigned themselves; rather they must always be assigned by adding an image texture node to a material.

**3D Viewport Draw Types** The Texture draw types used for Blender Internal have been replaced by three others in Cycles:

**Texture** This draw mode is used for editing, painting and mapping individual textures. Lighting is the same as in solid mode, so this is similar to the existing textured solid for Blender Internal. The texture drawn is the active image texture node for the material.

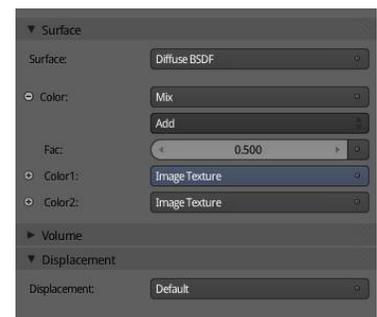
**Material** A simplified version of the entire material is drawn using GLSL shaders. This uses solid lighting, and also is mostly useful for editing, painting and mapping textures, but while seeing how they integrate with the material.

**Rendered** In this draw mode the render engine does the drawing, interactively refining the full rendered image by taking more samples. Unlike offline rendering, objects still use the viewport rather than render resolution and visibility.



**Texture Properties** In the texture properties, the texture can now be selected from a list that contains all texture nodes from the world, lamps and materials, but also from e.g. modifiers, brushes and physics fields.

For shading nodes, the available textures are Cycles textures. For others, Blender textures are still used, but this will change in the future.



**Painting & UV Editing** For texture paint mode, the image that is painted on is taken from the active image texture node. This can be selected in the node editor or the texture properties, and it is indicated as blue in the material properties.

For UV mapping, the active UV map as specified in the mesh properties is used. Assigning images in the image editor also affects the active image texture node.

## Nodes

### Introduction

Materials, lights and backgrounds are all defined using a network of shading nodes. These nodes output values, vectors, colors and shaders.

**Shaders** An important concept to understand when building node setups is that of the **shader socket**. The output of all surface and volume shaders is a shader, describing lighting interaction at the surface or of the volume, rather than the color of the surface.

There are a few types of shaders available as nodes:

- **BSDF** shader describing light reflection, refraction and absorption at an object surface.
- **Emission** shader describing light emission at an object surface or in a volume.
- **Volume** shader describing light scattering inside a volume.
- **Background** shader describing light emission from the environment.

Each shader node has a color input, and outputs a shader. These can then be mixed and added together using Mix and Add Shader nodes. No other operations are permitted. The resulting output can then be used by the render engine to compute all light interactions, for direct lighting or global illumination.

**Textures** Each texture type in Cycles corresponds to a node, with a texture coordinate and various parameters as input, and a color or value as output. No texture datablocks are needed; instead node groups can be used for reusing texture setups.

For UV mapping and texture painting in the viewport, the Image texture node must be used. When setting such a node as active, it will be drawn in Textured draw mode, and can be painted on in texture paint mode.

The default texture coordinates for all nodes are Generated coordinates, with the exception of Image textures that use UV coordinates by default. Each node includes some options to modify the texture mapping and resulting color, and these can be edited in the texture properties.

**More** Nodes for geometric data, texture coordinates, layering shaders and non-physically based tricks.

**Open Shading Language** Custom nodes can be written using the Open Shading Language.

### Input Nodes

#### Camera Data

**View Vector** A Camera space vector from the camera to the shading point.

**View Z Depth** TODO

**View Distance** Distance from the camera to the shading point.

**Value** Input a scalar value.

**Value** Value output.

**RGB** Input an RGB color.

**Color** RGB color output.

**Attribute** Retrieve attribute attached to the object or mesh. Currently UV maps and vertex color layers can be retrieved this way by their names, with layers and attributes planned to be added. Also internal attributes like  $P$  (position),  $N$  (normal),  $N_g$  (geometric normal) may be accessed this way, although there are more convenient nodes for this.

**Name** Name of the attribute.

**Color output** RGB color interpolated from the attribute.

**Vector output** XYZ vector interpolated from the attribute.

**Fac output** Scalar value interpolated from the attribute.

**Geometry** Geometric information about the current shading point. All vector coordinates are in *World Space*. For volume shaders, only the position and incoming vector are available.

**Position** Position of the shading point.

**Normal** Shading normal at the surface (includes smooth normals and bump mapping).

**Tangent** Tangent at the surface.

**True Normal** Geometry or flat normal of the surface.

**Incoming** Vector pointing towards the point the shading point is being viewed from.

**Parametric** Parametric coordinates of the shading point on the surface.

**Backfacing** 1.0 if the face is being viewed from the back side, 0.0 for the front side.

**Pointiness** An approximation of the curvature of the mesh per-vertex. Lighter values indicate convex angles, darker values indicate concave angles.

**Light Path** Node to find out for which kind of incoming ray the shader is being executed; particularly useful for non-physically based tricks. More information about the meaning of each type is in the [Light Paths](#) documentation.

**Is Camera Ray output** 1.0 if shading is executed for a camera ray, 0.0 otherwise.

**Is Shadow Ray output** 1.0 if shading is executed for a shadow ray, 0.0 otherwise.

**Is Diffuse Ray output** 1.0 if shading is executed for a diffuse ray, 0.0 otherwise.

**Is Glossy Ray output** 1.0 if shading is executed for a glossy ray, 0.0 otherwise.

**Is Singular Ray output** 1.0 if shading is executed for a singular ray, 0.0 otherwise.

**Is Reflection Ray output** 1.0 if shading is executed for a reflection ray, 0.0 otherwise.

**Is Transmission Ray output** 1.0 if shading is executed for a transmission ray, 0.0 otherwise.

**Ray Length output** Distance traveled by the light ray from the last bounce or camera.

**Object Info** Information about the object instance. This can be useful to give some variation to a single material assigned to multiple instances, either manually controlled through the object index, based on the object location, or randomized for each instance. For example a Noise texture can give random colors or a Color ramp can give a range of colors to be randomly picked from.

**Location** Location of the object in world space.

**Object Index** Object pass index, same as in the Object Index pass.transformed.

**Material Index** Material pass index, same as in the Material Index pass.

**Random** Random number unique to a single object instance.

**Fresnel** Dielectric fresnel, computing how much light is refracted through and how much is reflected off a layer. The resulting weight can be used for layering shaders with the *Mix Shader* node. It is dependent on the angle between the surface normal and the viewing direction.

**IOR input** Index of refraction of the material being entered.

**Fresnel output** Fresnel weight, indicating the probability with which light will reflect off the layer rather than passing through.

**Layer Weight** Output weights typically used for layering shaders with the *Mix Shader* node.

**Blend input** Blend between the first and second shader.

**Fresnel output** Dielectric fresnel weight, useful for example for layering diffuse and glossy shaders to create a plastic material. This is like the Fresnel node, except that the input of this node is in the often more-convenient 0.0 to 1.0 range.

**Facing output** Weight that blends from the first to the second shader as the surface goes from facing the viewer to viewing it at a grazing angle.

**Texture Coordinates** Commonly used texture coordinates, typically used as inputs for the *Vector* input for texture nodes.

**Generated** Automatically-generated texture coordinates from the vertex positions of the mesh without deformation, keeping them sticking to the surface under animation. Range from 0.0 to 1. 0 over the bounding box of the undeformed mesh.

**Normal** Object space normal, for texturing objects with the texture staying fixed on the object as it transformed.

**UV** UV texture coordinates from the active render UV layer.

**Object** Position coordinate in object space.

**Camera** Position coordinate in camera space.

**Window** Location of shading point on the screen, ranging from 0.0 to 1. 0 from the left to right side and bottom to top of the render.

**Reflection** Vector in the direction of a sharp reflection, typically used for environment maps.

**Particle Info** For objects instanced from a particle system, this node give access to the data of the particle that spawned the instance.

**Index** Index number of the particle (from 0 to number of particles).

**Age** Age of the particle in frames.

**Lifetime** Total lifespan of the particle in frames.

**Location** Location of the particle.

**Size** Size of the particle.

**Velocity** Velocity of the particle.

**Angular Velocity** Angular velocity of the particle.

**Hair Info** This node gives access to strand information.

**Is strand** Returns 1 when the shader is acting on a strand, otherwise 0.

**Intercept** The point along the strand where the ray hits the strand (1 at the tip and 0 at the root).

**Thickness** The thickness of the strand at the point where the ray hits the strand.

**Tangent Normal** Tangent normal of the strand.

**Tangent** Generates a tangent direction for the Anisotropic BSDF.

**Direction Type** The tangent direction can be derived from a cylindrical projection around the X, Y or Z axis (Radial), or from a manually created UV Map for full control.

**Tangent Output** The tangent direction vector.

### Output Nodes

Output nodes are the final node in every node tree. Although you can add more than one, only one will be used (indicated by a colored or darkened header). Output nodes are always preceded by [Shaders](#) except in the case of the [Displacement](#) of a Material Output.

#### *Material Output*

**Surface** The surface output of the material

**Volume** *Currently under independent development, does nothing*

**Displacement** Used to create bump mapping or actual subdivided [Displacement](#)

#### *Lamp Output*

**Surface** Not an actual surface, but the final output of a [Lamp](#) Object

#### *World Output*

**Surface** The appearance of the environment, usually preceded by a [Background](#) shader

**Volume** *Currently under independent development, does nothing*

### Shader Nodes

**Diffuse** Lambertian and Oren-Nayar diffuse reflection.

**Color input** Color of the surface, or physically speaking, the probability that light is reflected or transmitted for each wavelength.

**Roughness input** Surface roughness; 0.0 gives standard Lambertian reflection, higher values activate the Oren-Nayar BSDF.

**Normal input** Normal used for shading; if nothing is connected the default shading normal is used.

**BSDF output** Diffuse BSDF shader.



**Translucent** Lambertian diffuse transmission.

**Color input** Color of the surface, or physically speaking, the probability that light is transmitted for each wavelength.

**Normal input** Normal used for shading; if nothing is connected the default shading normal is used.

**BSDF output** Translucent BSDF shader.

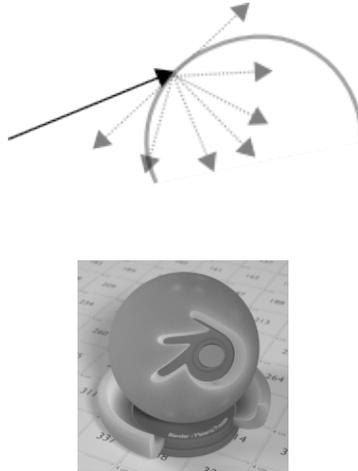


Fig. 2.2289: Translucent Shader

**Glossy** Glossy reflection with microfacet distribution, used for materials such as metal or mirrors.

**Distribution** Microfacet distribution to use. *Sharp* results in perfectly sharp reflections like a mirror, while *Beckmann*, *GGX* and *Ashikhmin-Shirley* can use the *Roughness* input for blurry reflections.

**Color input** Color of the surface, or physically speaking, the probability that light is reflected for each wavelength.

**Roughness input** Influences sharpness of the reflection; perfectly sharp at 0.0 and smoother with higher values.

**Normal input** Normal used for shading; if nothing is connected the default shading normal is used.

**BSDF output** Glossy BSDF shader.

Sharp Glossy	Rough Glossy

**Anisotropic** Anisotropic glossy reflection, with separate control over U and V direction roughness. The tangents used for shading are derived from the active UV map. If no UV map is available, they are automatically generated using a sphere mapping based on the mesh bounding box.

**Distribution** Microfacet distribution to use. *Sharp* results in perfectly sharp reflections like a mirror, while *Beckmann*, *GGX* and *Ashikhmin-Shirley* can use the *Roughness* input for blurry reflections.

**Color input** Color of the surface, or physically speaking, the probability that light is reflected for each wavelength.

**Roughness input** Sharpness of the reflection; perfectly sharp at 0.0 and smoother with higher values.

**Anisotropy input** Amount of anisotropy in the reflection; 0.0 gives a round highlight. Higher values give elongated highlights orthogonal to the tangent direction; negative values give highlights shaped along the tangent direction.

**Rotation input** Rotation of the anisotropic tangent direction. Value 0.0 equals 0- rotation, 0.25 equals 90- and 1.0 equals 360- = 0- . This can be used to texture the tangent direction.

**Normal input** Normal used for shading; if nothing is connected the default shading normal is used.

**Tangent input** Tangent used for shading; if nothing is connected the default shading tangent is used.

**BSDF output** Anisotropic glossy BSDF shader.



**Toon** Diffuse and Glossy Toon BSDF for creating cartoon light effects.

**Color input** Color of the surface, or physically speaking, the probability that light is reflected for each wavelength.

**Size input** Parameter between 0.0 and 1.0 that gives a angle of reflection between 0- and 90- .

**Smooth input** This value specifies an angle over which a smooth transition from full to no reflection happens.

**Normal input** Normal used for shading; if nothing is connected the default shading normal is used.

**BSDF output** Toon BSDF shader.



Fig. 2.2294: Toon Shader

**Transparent** Transparent BSDF without refraction, passing straight through the surface, as if there were no geometry there. Useful with alpha maps, for example. This shader *affects light paths somewhat differently* than other BSDF s. Note that only pure white transparent shaders are completely transparent.

**Color input** Color of the surface, or physically speaking, the probability for each wavelength that light is blocked or passes straight through the surface.

**BSDF output** Transparent BSDF shader.



**Glass** Glass-like shader mixing refraction and reflection at grazing angles. Like the transparent shader, only pure white will make it transparent. The glass shader tends to cause noise due to caustics. Since the Cycles path tracing integrator is not very good at rendering caustics, it helps to combine this with a transparent shader for shadows; for *more details see here*

**Distribution** Microfacet distribution to use. *Sharp* results in perfectly sharp refractions like clear glass, while *Beckmann* and *GGX* can use the *Roughness* input for rough glass.

**Color input** Color of the surface, or physically speaking, the probability that light is transmitted for each wavelength.

**Roughness input** Influences sharpness of the refraction; perfectly sharp at 0.0 and smoother with higher values.

**IOR input** Index of refraction defining how much the ray changes direction. At 1.0 rays pass straight through like transparent; higher values give more refraction.

**Normal input** Normal used for shading; if nothing is connected the default shading normal is used.

**BSDF output** Glass BSDF shader.

Sharp Glass	Rough Glass

**Refraction** Glossy refraction with sharp or microfacet distribution, used for materials that transmit light. For best results this node should be considered as a building block and not be used on its own, but rather mixed with a glossy node using a fresnel factor. Otherwise it will give quite dark results at the edges for glossy refraction.

**Distribution** Microfacet distribution to use. *Sharp* results in perfectly sharp refractions, while *Beckmann* and *GGX* can use the *Roughness* input for blurry refractions.

**Color input** Color of the surface, or physically speaking, the probability that light is refracted for each wavelength.

**Roughness input** Influences sharpness of the refraction; perfectly sharp at 0.0 and smoother with higher values.

**Normal input** Normal used for shading; if nothing is connected the default shading normal is used.

**BSDF output** Glossy BSDF shader.



Fig. 2.2301: Refraction Shader.

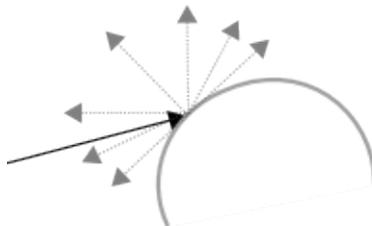
**Velvet** Velvet reflection shader for materials such as cloth. It is meant to be used together with other shaders (such as a *Diffuse Shader*) and isn't particularly useful on its own.

**Color input** Color of the surface, or physically speaking, the probability that light is reflected for each wavelength.

**Sigma input** Variance of the normal distribution, controlling the sharpness of the peak - can be thought of as a kind of *roughness*.

**Normal input** Normal used for shading; if nothing is connected the default shading normal is used.

**BSDF output** Velvet BSDF shader.



**Subsurface Scattering** Simple subsurface multiple scattering, for materials such as skin, wax, marble, milk and others. For these materials, rather than light being reflect directly off the surface, it will penetrate the surface and bounce around internally before getting absorbed or leaving the surface at a nearby point.

How far the color scatters on average can be configured per RGB color channel. For example, for skin, red colors scatter further, which gives distinctive red-colored shadows, and a soft appearance.

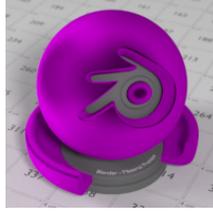


Fig. 2.2302: The Velvet Shader

**Falloff** Lighting distance falloff function. **Cubic** is a sharp falloff useful for many simple materials. The function is  $(\text{radius} - x)^3$  **Gaussian** gives a smoother falloff following a normal distribution, which is particularly useful for more advanced materials that use measured data that was fitted to one or more such Gaussian functions. The function is  $e^{-8x^2/\text{radius}^2}$ , such that the radius roughly matches the maximum falloff distance. To match a given measured variance  $v$ , set  $\text{radius} = \sqrt{16*v}$ .

**Color input** Color of the surface, or physically speaking, the probability that light is reflected for each wavelength.

**Scale input** Global scale factor for the scattering radius.

**Radius input** Scattering radius for each RGB color channel, the maximum distance that light can scatter.

**Sharpness input** Used only with **Cubic** falloff. Values increasing from 0 to 1 prevents softening of sharp edges and reduces unwanted darkening.

**Normal input** Normal used for shading; if nothing is connected the default shading normal is used.

**Texture Blur input** How much of the texture will be blurred along with the lighting, mixing the texture at the incoming and outgoing points on the surface. Note that the right choice depends on the texture. Consider for example a texture created from a photograph of skin, in this cases the colors will already be pre-blurred and texture blur could be set to 0. Even for hand painted textures no or minimal blurring might be appropriate, as a texture artist would likely paint in softening already, one would usually not even know what an unblurred skin texture looks like, we always see it blurred. For a procedural texture on the other hand this option would likely have a higher value.

**BSSRDF output** BSSRDF (Bidirectional subsurface scattering distribution function) shader.



Fig. 2.2303: A skin-toned SSS shader with color radius: 1.0, 0.8, 0.5.

**Emission** Lambertian emission, to be used for material and lamp surface outputs.

**Color input** Color of the emitted light.

**Strength input** Strength of the emitted light. For point and area lamps, the unit is Watts. For materials, a value of 1.0 will ensure that the object in the image has the exact same color as the Color input, i.e. make it 'shadeless'.

**Emission output** Emission shader.



Cycles uses a physically correct light falloff by default, whereas Blender Internal uses a smoothed falloff with a Distance parameter. A similar effect can be found by using the Light Falloff node with the Smooth parameter.

Lamp strength for point, spot and area lamps is specified in Watts. This means you typically need higher values than Blender Internal, as you couldn't use a 1W lamp to light a room; you need something stronger like a 100W lamp.

Sun lamps are specified in Watts/m<sup>2</sup>, which require much smaller values like 1 W/m<sup>2</sup>. This can be confusing, but specifying strength in Watts wouldn't have been convenient; the real sun for example has strength 384600000000000000000000000000W. Emission shaders on meshes are also in Watts/m<sup>2</sup>.

**Background** Background light emission. This node should only be used for the world surface output; it is ignored in other cases.

**Color input** Color of the emitted light.

**Strength input** Strength of the emitted light.

**Background output** Background shader.

**Holdout** The holdout shader creates a “hole” in the image with zero alpha transparency, which is useful for compositing (see [alpha channel](#)).

Note that the holdout shader can only create alpha when *Properties* → *Render* → *Film* → *Transparent* is enabled. If it's disabled, the holdout shader will be black.

**Holdout output** Holdout shader.

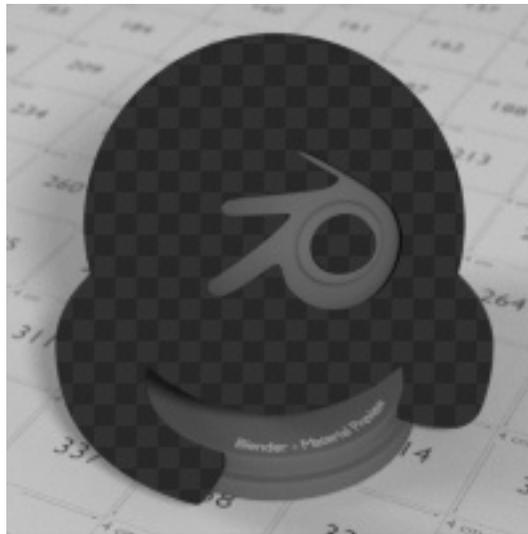


Fig. 2.2308: The checkered area is a region with zero alpha.

**Ambient Occlusion** The ambient occlusion node gives per-material control for the amount of AO. When AO is enabled in the world, it affects all diffuse BSDFs in the scene. With this option it's possible to let only some materials be affected by AO, or to let it influence some materials more or less than others.

**Color input** surface reflection color.

**AO output** Ambient Occlusion shader.

**Mix and Add** Mix or add shaders together. Mixing can be used for material layering, where the *Fac* input may, for example, be connected to a Blend Weight node.

**Shader inputs** Shaders to mix, such that incoming rays hit either with the specified probability in the *Fac* socket.



Fig. 2.2309: White AO shader.

**Fac input** Blend weight to use for mixing two shaders; at zero it uses the first shader entirely and at one the second shader.

**Shader output** Mixed shader.



Fig. 2.2310: A mix of a glossy and a diffuse shader makes a nice ceramic material.

### Texture Nodes

Fig. 2.2311: Image texture from [GoodTextures.com](http://GoodTextures.com)

**Image Texture** Use an image file as a texture.

**Image Datablock** Image datablock used as the image source. Currently not all images supported by Blender can be used by Cycles. In particular, generated, packed images or animations are not supported currently.

**Projection** Projection to use for mapping the textures. *Flat* will use the XY coordinates for mapping. *Box* will map the image to the 6 sides of a virtual box, based on the normal, using XY, YZ and XYZ coordinates depending on the side.

**Projection Blend** For *Box* mapping, the amount to blend between sides of the box, to get rid of sharp transitions between the different sides. Blending is useful to map a procedural-like image texture pattern seamlessly on a model. 0.0 gives no blending; higher values give a smoother transition.

**Color Space** Type of data that the image contains, either *Color* or *Non-Color Data*. For most color textures the default of *Color* should be used, but in case of e.g. a bump or alpha map, the pixel values should be interpreted as *Non-Color Data*, to avoid doing any unwanted color space conversions.

**Vector input** Texture coordinate for texture lookup. If this socket is left unconnected, UV coordinates from the active UV render layer are used.

**Color output** RGB color from image. If the image has alpha, the color is premultiplied with alpha if the Alpha output is used, and unpremultiplied or straight if the Alpha output is not used.

**Alpha output** Alpha channel from image.



Fig. 2.2312: HDR image from [OpenFootage.net](http://OpenFootage.net)

**Environment Texture** Use an environment map image file as a texture. The environment map is expected to be in Latitude/Longitude or ‘latlong’ format.

**Image Datablock** Image datablock used as the image source. Currently not all images supported by Blender can be used by Cycles. In particular, generated, packed images or animations are not supported currently.

**Color Space** Type of data that the image contains, either Color or Non-Color Data. For most color textures the default of Color should be used, but in case of e.g. a bump or alpha map, the pixel values should be interpreted as Non-Color Data, to avoid doing any unwanted color space conversions.

**Vector input** Texture coordinate for texture lookup. If this socket is left unconnected, the image is mapped as environment with the Z axis as up.

**Color output** RGB color from the image. If the image has alpha, the color is premultiplied with alpha if the Alpha output is used, and unpremultiplied if the Alpha output is not used.

**Alpha output** Alpha channel from image.

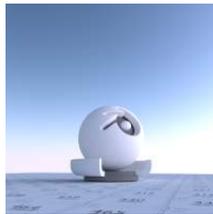


Fig. 2.2313: Sky Texture

**Sky Texture** Procedural Sky texture.

**Sky Type** Sky model to use (Preetham or Hosek / Wilkie).

**Sun Direction** Sun direction vector.

**Turbidity** Atmospheric turbidity. (2: Arctic like, 3: clear sky, 6: warm/moist day, 10: hazy day)

**Ground Albedo** Amount of light reflected from the planet surface back into the atmosphere. (RGB 0,0,0 is black, 1,1,1 is white).

**Vector** Texture coordinate to sample texture at; defaults to Generated texture coordinates if the socket is left unconnected.

**Color output** Texture color output.

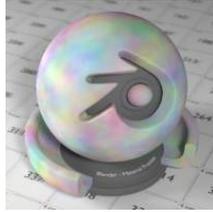


Fig. 2.2314: Noise Texture with high detail

**Noise Texture** Procedural Perlin noise texture, similar to the Clouds texture in Blender Internal.

**Vector input** Texture coordinate to sample texture at; defaults to Generated texture coordinates if the socket is left unconnected.

**Scale input** Overall texture scale.

**Detail input** Amount of noise detail.

**Distortion input** Amount of distortion.

**Color output** Texture color output.

**Fac output** Texture intensity output.

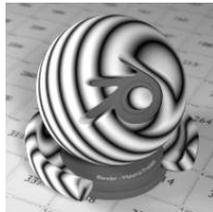


Fig. 2.2315: Default wave texture

**Wave Texture** Procedural bands or rings texture with noise distortion.

**Type** *Bands* or *Rings* shaped waves.

**Vector input** Texture coordinate to sample texture at; defaults to Generated texture coordinates if the socket is left unconnected.

**Scale input** Overall texture scale.

**Distortion input** Amount of distortion of the wave (similar to the Marble texture in Blender Internal).

**Detail input** Amount of distortion noise detail.

**Detail Scale input** Scale of distortion noise.

**Color output** Texture color output.

**Fac output** Texture intensity output.

**Voronoi Texture**

Procedural texture producing Voronoi cells.

**Type** *Intensity* or *Cells* output.

**Vector input** Texture coordinate to sample texture at; defaults to Generated texture coordinates if the socket is left unconnected.

**Scale input** Overall texture scale.

**Color output** Texture color output.

**Fac output** Texture intensity output.

**Musgrave Texture** Advanced procedural noise texture. Note that it often needs some adjustments (multiplication and addition) in order to see more detail.



**Type** Multifractal, Ridged Multifractal, Hybrid Multifractal, fBM, Hetero Terrain.

**Vector input** Texture coordinate to sample texture at; defaults to Generated texture coordinates if the socket is left unconnected.

**Scale input** Overall texture scale.

**Detail input** Amount of noise detail.

**Dimension input** *TBD*

**Lacunarity input** *TBD*

**Offset input** *TBD*

**Gain input** *TBD*

**Color output** Texture color output.

**Fac output** Texture intensity output.



Fig. 2.2324: Gradient texture using object coordinates

**Gradient Texture** A gradient texture.

**Type** The gradient can be *Linear*, *Quadratic*, *Easing*, *Diagonal*, *Spherical*, *Quadratic Sphere* or *Radial*.

**Vector input** Texture coordinate to sample texture at; defaults to Generated texture coordinates if the socket is left unconnected.

**Color output** Texture color output.

**Fac output** Texture intensity output.

**Magic Texture** Psychedelic color texture.

**Depth** Number of iterations.

**Vector input** Texture coordinate to sample texture at; defaults to Generated texture coordinates if the socket is left unconnected.

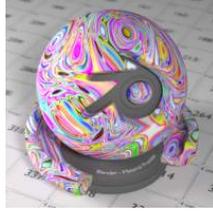


Fig. 2.2325: Magic texture: Depth 10, Distortion 2.0

**Distortion input** Amount of distortion.

**Color output** Texture color output.

**Fac output** Texture intensity output.



Fig. 2.2326: Default Checker texture

**Checker Texture** Checkerboard texture.

**Vector input** Texture coordinate to sample texture at; defaults to Generated texture coordinates if the socket is left unconnected.

**Color1/2 input** Color of the checkers.

**Scale input** Overall texture scale.

**Color output** Texture color output.

**Fac output** Checker 1 mask (1 = Checker 1).

**Brick Texture** Procedural texture producing Bricks.

### Options

**Offset** Determines the brick offset of the various rows.

**Frequency** Determines the offset frequency. A value of 2 gives a even/uneven pattern of rows.

**Squash** Amount of brick squashing.

**Frequency** Brick squashing frequency.

### Sockets

**Color 1/2 and Mortar** Color of the bricks and mortar.

**Scale** Overall texture scale.

**Mortar Size** The Mortar size; 0 means no Mortar.



Fig. 2.2327: Brick texture: Colors changed, Squash 0.62, Squash Frequency 3.

**Bias** The color variation between Brick color 1 / 2. Values of -1 and 1 only use one of the two colors; values in between mix the colors.

**Brick Width** The width of the bricks.

**Row Height** The height of the brick rows.

**Color output** Texture color output.

**Fac output** Mortar mask (1 = mortar).

#### More Nodes

**Value** Input a scalar value.

**Value** Value output.

**RGB** Input an RGB color.

**Color** RGB color output.

**Geometry** Geometric information about the current shading point. All vector coordinates are in *World Space*. For volume shaders, only the position and incoming vector are available.

**Position** Position of the shading point.

**Normal** Shading normal at the surface (includes smooth normals and bump mapping).

**Tangent** Tangent at the surface.

**True Normal** Geometry or flat normal of the surface.

**Incoming** Vector pointing towards the point the shading point is being viewed from.

**Parametric** Parametric coordinates of the shading point on the surface.

**Backfacing** 1.0 if the face is being viewed from the backside, 0.0 for the frontside.

**Wireframe** Node for a wireframe shader (Triangles only for now).

**Pixel Size** Use screen pixel size instead of world units.

**Size** Controls the thickness of the wireframe.

**Fac output** 1.0 if shading is executed on an edge, 0.0 otherwise.

**Wavelength** A wavelength to rgb converter.

**Wavelength** The color wavelength from 380 to 780 nanometers.

**Color** RGB color output.

**Blackbody** A blackbody temperature to RGB converter.

**Temperature** The temperature in Kelvin.

**Color** RGB color output.

**Texture Coordinates** Commonly used texture coordinates, typically used as inputs for the *Vector* input for texture nodes.

**Generated** Automatically-generated texture coordinates from the vertex positions of the mesh without deformation, keeping them sticking to the surface under animation. Range from 0.0 to 1. 0 over the bounding box of the undeformed mesh.

**Normal** Object space normal, for texturing objects with the texture staying fixed on the object as it transformed.

**UV** UV texture coordinates from the active render UV layer.

**Object** Position coordinate in object space.

**Camera** Position coordinate in camera space.

**Window** Location of shading point on the screen, ranging from 0.0 to 1. 0 from the left to right side and bottom to top of the render.

**Reflection** Vector in the direction of a sharp reflection, typically used for environment maps.

**Bump** Generate a perturbed normal from a height texture, for bump mapping. The height value will be sampled at the shading point and two nearby points on the surface to determine the local direction of the normal.

**Invert** Invert the bump mapping, to displace into the surface instead of out.

**Strength Input** Strength of the bump mapping effect, interpolating between no bump mapping and full bump mapping.

**Distance Input** Multiplier for the height value to control the overall distance for bump mapping.

**Height Input** Scalar value giving the height offset from the surface at the shading point; this is where you plug in textures.

**Vector Transform** Allows converting a Vector, Point or Normal between World <=> Camera <=> Object coordinate space.

**Type** Specifies the input/output type: Vector, Point or Normal.

**Convert From** Coordinate Space to convert from: World, Object or Camera.

**Convert To** Coordinate Space to convert to: World, Object or Camera.

**Vector Input** The input vector.

**Vector Output** The transformed output vector.

**Tangent** Generate a tangent direction for the Anisotropic BSDF.

**Direction Type** The tangent direction can be derived from a cylindrical projection around the X, Y or Z axis (Radial), or from a manually created UV Map for full control.

**Tangent Output** The tangent direction vector.

**Normal Map** Generate a perturbed normal from an RGB normal map image. This is usually chained with an Image Texture node in the color input, to specify the normal map image. For tangent space normal maps, the UV coordinates for the image must match, and the image texture should be set to Non-Color mode to give correct results.

**Space** The input RGB color can be in one of 3 spaces: Tangent, Object and World space. Tangent space normal maps are the most common, as they support object transformation and mesh deformations. Object space normal maps keep sticking to the surface under object transformations, while World normal maps do not.

**UV Map** Name of the UV map to derive normal mapping tangents from. When chained with an Image Texture node, this UV map should be the same as the UV map used to map the texture.

**Strength** Strength of the normal mapping effect.

**Color Input** RGB color that encodes the normal in the specified space.

**Normal Output** Normal that can be used as an input to BSDF nodes.

**Object Info** Information about the object instance. This can be useful to give some variation to a single material assigned to multiple instances, either manually controlled through the object index, based on the object location, or randomized for each instance. For example a Noise texture can give random colors or a Color ramp can give a range of colors to be randomly picked from.

Note that this node only works for material shading nodes; it does nothing for lamp and world shading nodes.

**Location** Location of the object in world space.

**Object Index** Object pass index, same as in the Object Index pass.transformed.

**Material Index** Material pass index, same as in the Material Index pass.

**Random** Random number unique to a single object instance.

**Particle Info** For objects instanced from a particle system, this node give access to the data of the particle that spawned the instance. This node currently only supports parent particles, info from child particles is not available.

**Index** Index number of the particle (from 0 to number of particles).

**Age** Age of the particle in frames.

**Lifetime** Total lifespan of the particle in frames.

**Location** Location of the particle.

**Size** Size of the particle.

**Velocity** Velocity of the particle.

**Angular Velocity** Angular velocity of the particle.

**Hair Info** This node gives access to strand information.

**Is strand** Returns 1 when the shader is acting on a strand, otherwise 0.

**Intersect** The point along the strand where the ray hits the strand (1 at the tip and 0 at the root).

**Thickness** The thickness of the strand at the point where the ray hits the strand.

**Tangent Normal** Tangent normal of the strand.

**Attribute** Retrieve attribute attached to the object or mesh. Currently UV maps and vertex color layers can be retrieved this way by their names, with layers and attributes planned to be added. Also internal attributes like  $P$  (position),  $N$  (normal),  $N_g$  (geometric normal) may be accessed this way, although there are more convenient nodes for this.

**Name** Name of the attribute.

**Color output** RGB color interpolated from the attribute.

**Vector output** XYZ vector interpolated from the attribute.

**Fac output** Scalar value interpolated from the attribute.

**Mapping** Transform a coordinate; typically used for modifying texture coordinates.

**Location** Vector translation.

**Rotation** Rotation of the vector along XYZ axes.

**Scale** Scale of the vector.

**Vector input** Vector to be transformed.

**Vector output** Transformed vector.

**Layer Weight** Output weights typically used for layering shaders with the *Mix Shader* node.

**Blend input** Blend between the first and second shader.

**Fresnel output** Dielectric fresnel weight, useful for example to layer diffuse and glossy shaders to create a plastic material. This is like the *Fresnel* node, except that the input of this node is in the often more-convenient 0.0 to 1.0 range.

**Facing output** Weight that blends from the first to the second shader as the surface goes from facing the viewer to viewing it at a grazing angle.

**Fresnel** Dielectric fresnel, computing how much light is reflected off a layer, where the rest will be refracted through the layer. The resulting weight can be used for layering shaders with the *Mix Shader* node. It is dependent on the angle between the surface normal and the viewing direction.

The most common use is to mix between two BSDFs using it as a blending factor in a mix shader node. For a simple glass material you would mix between a glossy refraction and glossy reflection. At grazing angles more light will be reflected than refracted as happens in reality.

For a two-layered material with a diffuse base and a glossy coating, you can use the same setup, mixing between a diffuse and glossy BSDF. By using the fresnel as the blending factor you're specifying that any light which is refracted through the glossy coating layer would hit the diffuse base and be reflected off that.

**IOR input** Index of refraction of the material being entered.

**Fresnel output** Fresnel weight, indicating the probability with which light will reflect off the layer rather than passing through.

**Light Path** Node to find out for which kind of incoming ray the shader is being executed; particularly useful for non-physically based tricks. More information about the meaning of each type is in the [Light Paths](#) documentation.

**Is Camera Ray output** 1.0 if shading is executed for a camera ray, 0.0 otherwise.

**Is Shadow Ray output** 1.0 if shading is executed for a shadow ray, 0.0 otherwise.

**Is Diffuse Ray output** 1.0 if shading is executed for a diffuse ray, 0.0 otherwise.

**Is Glossy Ray output** 1.0 if shading is executed for a glossy ray, 0.0 otherwise.

**Is Singular Ray output** 1.0 if shading is executed for a singular ray, 0.0 otherwise.

**Is Reflection Ray output** 1.0 if shading is executed for a reflection ray, 0.0 otherwise.

**Is Transmission Ray output** 1.0 if shading is executed for a transmission ray, 0.0 otherwise.

**Ray Length output** Distance travelled by the light ray from the last bounce or camera.

**Ray Depth output** Returns the current light bounce.

**Transparent Depth output** Returns the number of transparent surfaces passed through.

**Light Falloff** Manipulate how light intensity decreases over distance. In reality light will always fall off quadratically; however it can be useful to manipulate as a non-physically based lighting trick. Note that using Linear or Constant falloff may cause more light to be introduced with every global illumination bounce, making the resulting image extremely bright if many bounces are used.

**Strength input** Light strength before applying falloff modification.

**Smooth input** Smooth intensity of light near light sources. This can avoid harsh highlights, and reduce global illumination noise. 0.0 corresponds to no smoothing; higher values smooth more. The maximum light strength will be strength/smooth.

**Quadratic output** Quadratic light falloff; this will leave strength unmodified if smooth is 0.0 and corresponds to reality.

**Linear output** Linear light falloff, giving a slower decrease in intensity over distance.

**Constant output** Constant light falloff, where the distance to the light has no influence on its intensity.

**Nodes shared with the Compositor** Some nodes are common with Composite nodes, their documentation can be found at their relevant pages rather than repeated here.

- [Brightness Contrast](#)
- [Separate RGB](#)
- [Combine RGB](#)
- [Separate HSV](#)
- [Combine HSV](#)
- [Gamma](#)
- [Hue Saturation Value](#)
- [Invert](#)
- [Math](#)
- [Mix RGB](#)
- [RGB Curves](#)
- [RGB to BW](#)
- [Vector Curve](#)

## World

The world environment can emit light, ranging from a single solid color, physical sky model, to arbitrary textures.



Fig. 2.2328: Lighting with an HDR image

### Surface Shader

The surface shader defines the light emission from the environment into the scene. The world surface is rendered as if it is very distant from the scene, and as such there is no two-way interacting between objects in the scene and the environment, only light coming in. The only shader accepted is the Background node with a color input and strength factor for the intensity of the light.

**Image Based Lighting** For image based lighting, use the Environment Texture node rather than the Image Texture node for correct mapping. This supports Equirectangular (also known as Lat/Long) for environment maps, and Mirror Ball mapping for converting photos of mirror balls to environment maps.

### Volume Shader

A volume shader can be applied to the entirely world, filling the entire space.

Currently this is most useful for night time or other dark scenes, as the world surface shader or sun lamps will have no effect if a volume shader is used. This is because the world background is assumed to be infinitely far away, which is accurate enough for the sun for example. However for modeling effects such as fog or atmospheric scattering, it is not a good assumption that the volume fills the entire space, as most of the distance between the sun and the earth is empty space. For such effects it is better to create a volume object surrounding the scene. The size of this object will determine how much light is scattered or absorbed.

### Ambient Occlusion

Ambient occlusion is a lighting method based on how much a point on a surface is occluded by nearby surfaces. This is a trick that is not physically accurate, but it is useful to emphasize shapes of surfaces, or as a cheap way to get an effect that looks a bit like indirect lighting.

**Factor** The strength of the ambient occlusion; value 1.0 is like a white world shader.

**Distance** Distance from shading point to trace rays. A shorter distance emphasizes nearby features, while longer distances make it also take objects further away into account.

Lighting from ambient occlusion is only applied to diffuse reflection BSDFs; glossy or transmission BSDFs are not affected. Transparency of surfaces will be taken into account, i.e. a half-transparent surface will only half occlude.

An alternative method of using Ambient Occlusion on a per-shader basis is to use the *Ambient Occlusion* shader.

### Sampling

**Multiple Importance Sample** Enabling this will sample the background texture such that lighter parts are favored, producing less noise in the render. It is almost always a good idea to enable this when using an image texture to light the scene, otherwise noise can take a very long time to converge.

**Map Resolution** Sets the resolution of the ‘Multiple Importance Sample’ map. Higher values may produce less noise when using high-res images, but will take up more memory and render slightly slower.

Below is a comparison between Multiple Importance Sample Off and On - both images rendered for 25 seconds (Off: 1500 samples, On: 1000 samples)



For interior scenes, noise can be significantly reduce by setting up area lamps as [light portals](#).



## Ray Visibility

As with other objects, *Ray Visibility* allows you to control which other shaders can “see” the environment.

## Tricks

Sometimes it may be useful to have a different background that is directly visible versus one that is indirectly lighting the objects. A simple solution to this is to add a Mix node, with the Blend Factor set to *Is Camera Ray*. The first input color is then the indirect color, and the second the directly visible color. This is useful when using a high-res image for the background and a low-res image for the actual lighting.

Similarly, adding the *Is Camera* and *Is Glossy* rays will mean that the high-res image will also be visible in reflections.

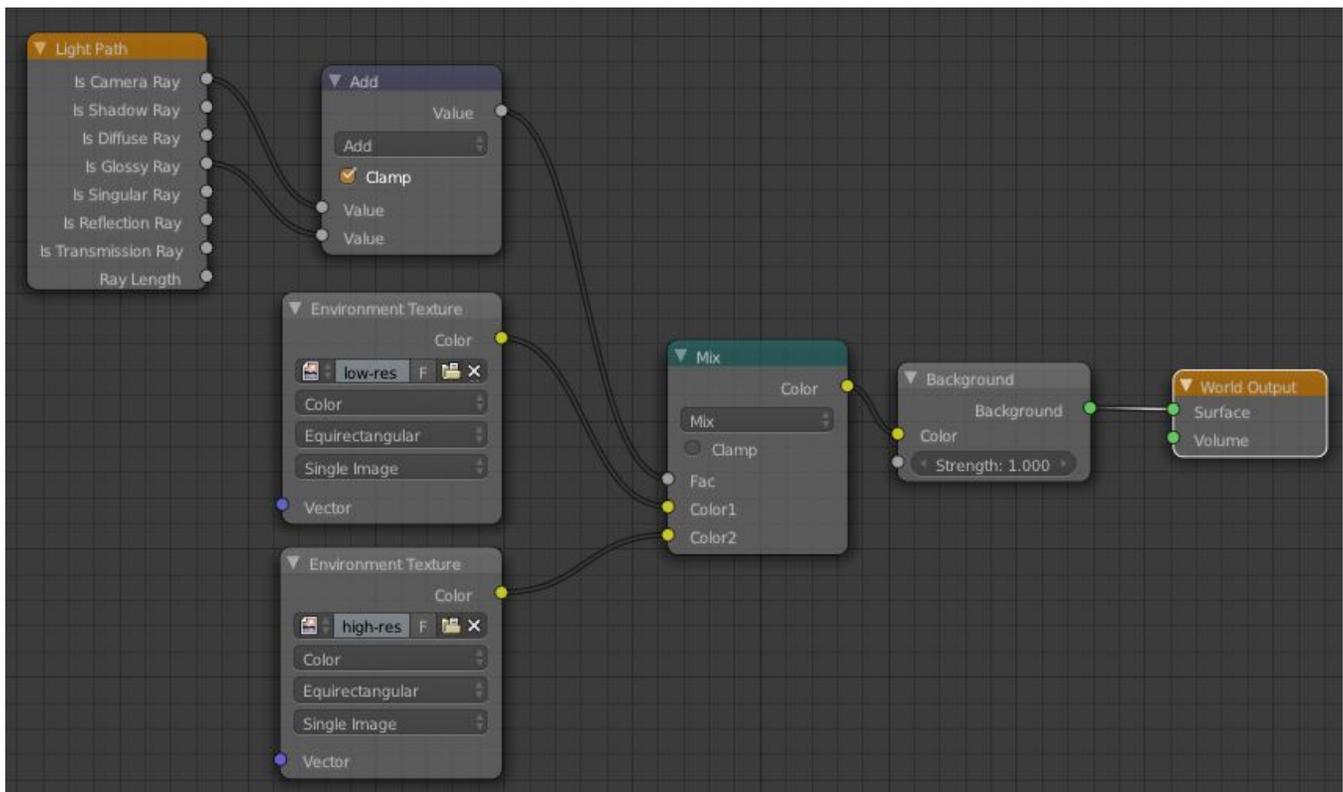


Fig. 2.2337: Nodes for the trick above

## Lamps

Next to lighting from the background and any object with an emission shader, lamps are another way to add light into the scene. The difference is that they are not directly visible in the rendered image, and can be more easily managed as objects of their own type.

**Type** Currently *Point*, *Spot*, *Area* and *Sun* lamps are supported. *Hemi* lamps are not supported, and will be rendered as point and sun lamps respectively, but they may start working in the future, so it's best not to enable them to preserve compatibility.

**Size** Size of the lamp in Blender Units; increasing this will result in softer shadows and shading.

**Cast Shadow** By disabling this option, light from lamps will not be blocked by objects in-between. This can speed up rendering by not having to trace rays to the light source.

### Point Lamp

Point lamps emit light equally in all directions. By setting the *Size* larger than zero, they become spherical lamps, which give softer shadows and shading. The strength of point lamps is specified in Watts.

### Spot Lamp

Spot lamps emit light in a particular direction, inside a cone. By setting the *Size* larger than zero, they can cast softer shadows and shading. The size parameter defines the size of the cone, while the blend parameter can soften the edges of the cone.

### Area Lamp

Area lamps emit light from a square or rectangular area with a Lambertian distribution.

**Light Portals** Area lamps can also function as light portals to help sample the environment light, and significantly reduce noise in interior scenes. Note that rendering with portals is usually slower, but as it converges more quickly, less samples are required.

Light portals work by enabling the Portal option, and placing areas lamps in windows, door openings, and any place where light will enter the interior.

### Sun Lamp

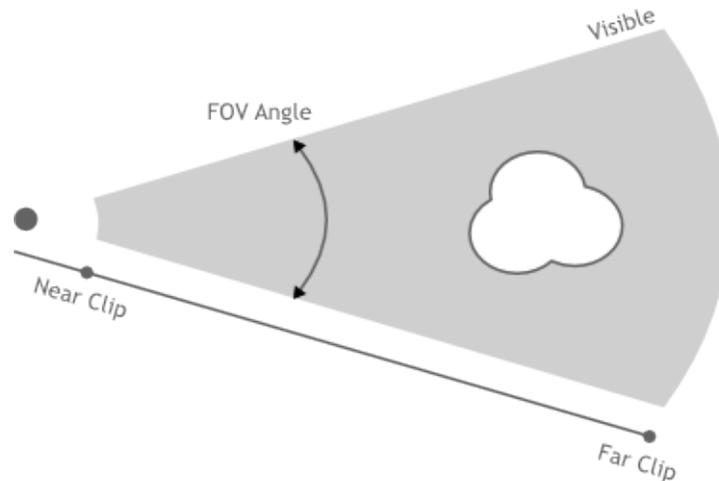
Sun lamps emit light in a given direction. Their position is not taken into account; they are always located outside of the scene, infinitely far away, and will not result in any distance falloff.

Because they are not located inside the scene, their strength uses different units, and should typically be set to lower values than other lights.

## Camera

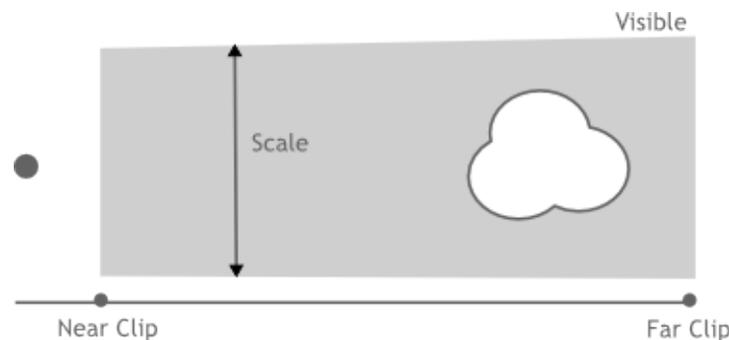
### Perspective

**Lens Size and Angle** Control the field of view angle.



### Orthographic

**Scale** Controls the size of objects projected on the image.



### Panoramic

Cycles supports Equirectangular and Fisheye panoramic cameras. Note that these can't be displayed with OpenGL rendering in the viewport; they will only work for rendering.

**Equirectangular** Render a panoramic view of the scenes from the camera location and use an equirectangular projection, always rendering the full 360° over the X-axis and 180° over the Y-axis.

This projection is compatible with the environment texture as used for world shaders, so it can be used to render an environment map. To match the default mapping, set the camera object rotation to (90, 0, -90) or pointing along the positive X-axis. This corresponds to looking at the center of the image using the default environment texture mapping.

**Fisheye** Fisheye lenses are typically wide angle lenses with strong distortion, useful for creating panoramic images for e.g. dome projection, or as an artistic effect. The *Fisheye Equisolid* lens will best match real cameras. It provides a lens focal length and field of view angle, and will also take the sensor dimensions into account.

The *Fisheye Equidistant* lens does not correspond to any real lens model; it will give a circular fisheye that doesn't take any sensor information into account but rather uses the whole sensor. This is a good lens for full dome projection.

**Lens** Lens focal length in millimeter.

**Field of View** Field of view angle, going to 360 and more to capture the whole environment.

### Depth of Field

**Aperture Type** Method with which to specify the size of the camera opening through which light enters. With Radius the radius of the opening can be specified, while F/Stop specifies the size relative to the camera focal length, a measure more common in photography. Their relation is:  $aperture\ radius = focal\ length / (2 \cdot f-stop)$

**Aperture Size** Also called lens radius. If this is zero, all objects will appear in focus, while larger values will make objects farther than the focal distance appear out of focus.

**Aperture F/Stop** Also called F-number or relative aperture. Lower numbers give more depth of field; higher numbers give a sharper image.

**Aperture Blades** If this setting is 3 or more, a polygonal-shaped aperture will be used instead of a circle, which will affect the shape of out of focus highlights in the rendered image.

**Aperture Rotation** Rotation of the *Aperture Blades*.

**Focal Distance** Distance at which objects are in perfect focus. Alternatively, an object can be specified whose distance from the camera will be used.

### Clipping

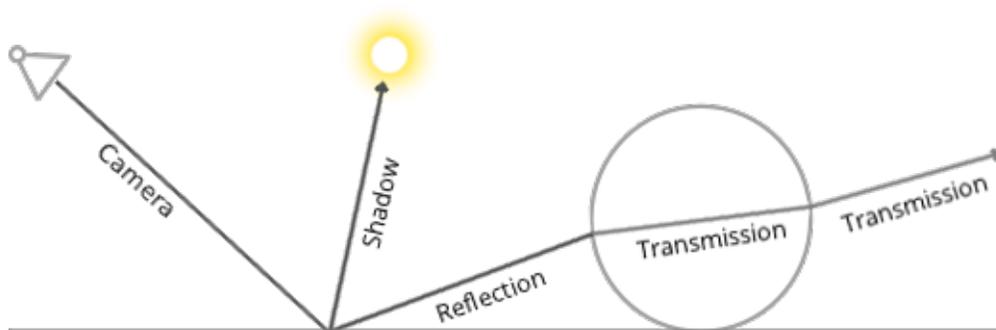
**Clip Start and End** The interval in which objects are directly visible. Any objects outside this range still influence the image indirectly, as further light bounces are not clipped. For OpenGL rendering, setting clipping distances to limited values is important to ensure sufficient rasterization precision. Ray tracing does not suffer from this issue much, and as such more extreme values can safely be set.

### Reducing Noise

When performing a final render, it is important to reduce noise as much as possible. Here we'll discuss a number of tricks that, while breaking the laws of physics, are particularly important when rendering animations within a reasonable time. Click to enlarge the example images to see the noise differences well.

### Path Tracing

Cycles uses path tracing with next event estimation, which is not good at rendering all types of light effects, like caustics, but has the advantage of being able to render more detailed and larger scenes compared to some other rendering algorithms. This is because we do not need to store, for example, a photon map in memory, and because we can keep rays relatively coherent to use an on-demand image cache, compared to e.g. bidirectional path tracing.



We do the inverse of what reality does, tracing light rays from the camera into the scene and onto lights, rather than from the light sources into the scene and then into the camera. This has the advantage that we do not waste light rays that will not end up in the camera, but also means that it is difficult to find some light paths that may contribute a lot. Light rays will be sent either according to the surface BRDF, or in the direction of known light sources (lamps, emitting meshes with Sample as Lamp).

For more details, see the [Light Paths](#) and [Integrator](#) documentation.

### Where Noise Comes From

To understand where noise can come from, take for example this scene. When we trace a light ray into the specified location, this is what the diffuse shader “sees”. To find the light that is reflected from this surface, we need to find the average color from all these pixels. Note the glossy highlight on the sphere, and the bright spot the lamp casts on the nearby wall. These hotspots are 100x brighter than other parts of the image and will contribute significantly to the lighting of this pixel.



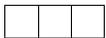
The lamp is a known light source, so it will not be too hard to find, but the glossy highlight (s) that it causes are a different matter. The best we can do with path tracing is to distribute light rays randomly over the hemisphere, hoping to find all the important bright spots. If for some pixels we miss some bright spot, but we do find it for another, that results in noise. The more samples we take, the higher the probability that we cover all the important sources of light.

With some tricks we can reduce this noise. If we blur the bright spots, they become bigger and less intense, making them easier to find and less noisy. This will not give the same exact result, but often it’s close enough when viewed through a diffuse or soft glossy reflection. Below is an example of using Filter Glossy and Smooth Light Falloff.



### Bounces

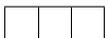
In reality light will bounce a huge number of times due to the speed of light being very high. In practice more bounces will introduce more noise, and it might be good to use something like the Limited Global Illumination preset that uses **fewer bounces for different shader types**. Diffuse surfaces typically can get away with fewer bounces, while glossy surfaces need a few more, and transmission shaders such as glass usually need the most.



Also important is to **use shader colors that do not have components of value 1.0** or values near that; try to keep the maximum value to 0.8 or less and make your lights brighter. In reality, surfaces are rarely perfectly reflecting all light, but there are of course exceptions; usually glass will let most light through, which is why we need more bounces there. High values for the color components tend to introduce noise because light intensity then does not decrease much as it bounces off each surface.

### Caustics and Filter Glossy

Caustics are a well-known source of noise, causing fireflies. They happen because the renderer has difficulty finding specular highlights viewed through a soft glossy or diffuse reflection. There is a *No Caustics* option to disable glossy behind a diffuse reflection entirely. Many render engines will typically disable caustics by default.



However using No Caustics will result in missing light, and it still does not cover the case where a sharp glossy reflection is viewed through a soft glossy reflection. There is a *Filter Glossy* option to reduce the noise from such cases at the cost of accuracy. This will blur the sharp glossy reflection to make it easier to find, by increasing the shader Roughness.

The above images show default settings, no caustics, and filter glossy set to 1.0.

## Light Falloff

In reality light in a vacuum will always fall off at a rate of  $1/(\text{distance}^2)$ . However as distance goes to zero, this value goes to infinity and we can get very bright spots in the image. These are mostly a problem for indirect lighting, where the probability of hitting such a small but extremely bright spot is low and so happens only rarely. This is a typical recipe for fireflies.



To reduce this problem, the *Light Falloff* node has a **Smooth factor, that can be used to reduce the maximum intensity** a light can contribute to nearby surfaces. The images above show default falloff and smooth value 1.0.

## Sample as Lamp

Materials with emission shaders can be configured to be **sampled as lamp** (*Material Settings*). This means that they will get rays sent directly towards them, rather than ending up there based on rays randomly bouncing around. For very bright mesh light sources, this can reduce noise significantly. However when the emission is not particularly bright, this will take samples away from other brighter light sources for which it is important to find them this way.

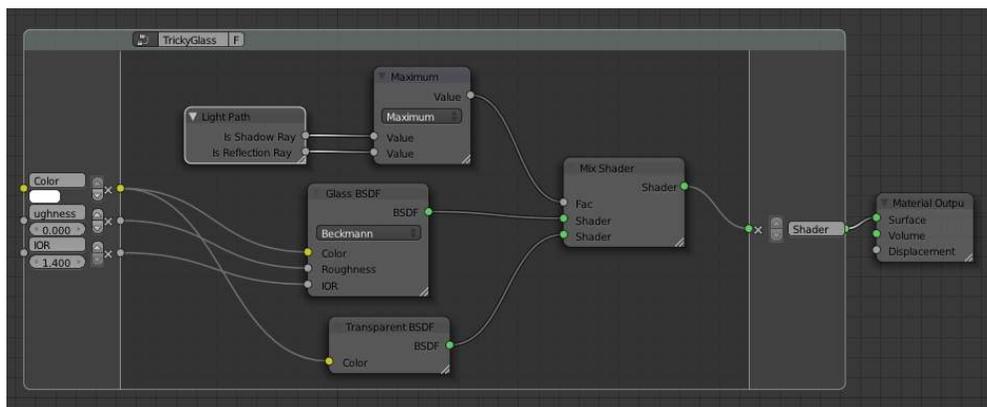
The optimal setting here is difficult to guess; it may be a matter of trial and error, but often it is clear that a somewhat glowing object may be only contributing light locally, while a mesh light used as a lamp would need this option enabled. Here is an example where the emissive spheres contribute little to the lighting, and the image renders with slightly less noise by disabling Sample as Lamp on them.



The world background also has a *Sample as Lamp* (*World Settings*) option. This is mostly useful for environment maps that have small bright spots in them, rather than being smooth. This option will then, in a preprocess, determine the bright spots, and send light rays directly towards them. Again, enabling this option may take samples away from more important light sources if it is not needed.

## Glass and Transparent Shadows

With caustics disabled, glass will miss shadows, and with filter glossy they might be too soft. We can make a glass shader that will **use a Glass BSDF when viewed directly, and a Transparent BSDF when viewed indirectly**. The Transparent BSDF can be used for transparent shadows to find light sources straight through surfaces, and will give properly-colored shadows, but without the caustics. The Light Path node is used to determine when to use which of the two shaders.



Above we can see the node setup used for the glass transparency trick; on the left the render has too much shadow due to missing caustics, and on the right the render with the trick.

## Window Lights

When rendering a daylight indoor scene where most of the light is coming in through a window or door opening, it is difficult for the integrator to find its way to them. We can replace the opening with a plane with an emission shader, so that the integrator knows in which direction to fire rays. For camera rays we can make this mesh light invisible, so that we can still look into the outside scene. This is done either by disabling camera ray visibility on the object, or by switching between glass and emission shaders in the material.

The two renders below have the same render time, with the second render using a mesh light positioned in the window.



In newer versions, [light portals](#) provide a better solution.

## Clamp Fireflies

Ideally with all the previous tricks, fireflies would be eliminated, but they could still happen. For that, **the intensity that any individual light ray sample will contribute to a pixel can be clamped** to a maximum value with the integrator *Clamp setting*. If set too low this can cause missing highlights in the image, which might be useful to preserve for camera effects such as bloom or glare.



## Features

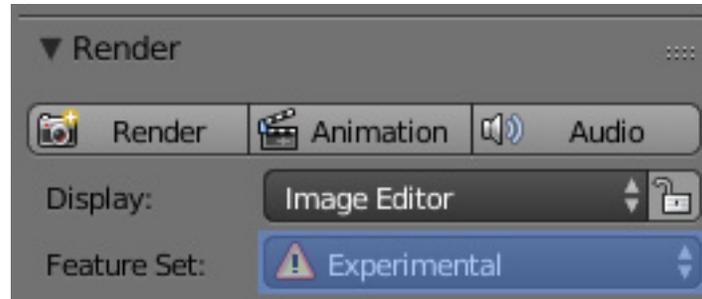
This page offers a comparison of available features on CPU, CUDA and OpenCL.

Feature	CPU	CUDA (NVIDIA GPU)	OpenCL (AMD GPU)
BASIC SHADING (Includes Node Shaders and Textures, Ambient Occlusion, Global Illumination...)			
Transparent Shadows			
Motion Blur			
Hair			
Volume			
Smoke / Fire			
Subsurface Scattering		(experimental)	
Open Shading Language			
CMJ sampling		(experimental)	
Branched Path integrator			
Displacement / Subdivision	(experimental)	(experimental)	(experimental)

## Experimental Features

Experimental features are disabled / hidden by default, but can be enabled by setting *Feature Set* to *Experimental* in the Render properties. They may not work properly, crash Blender or change their behaviour in later versions.

**Warning:** The experimental GPU kernel requires a lot of memory and thus may not work at all on cards without enough ram.



## GPU Rendering

### Introduction

GPU rendering makes it possible to use your graphics card for rendering, instead of the CPU. This can speed up rendering, because modern GPUs are designed to do quite a lot of number crunching. On the other hand, they also have some limitations in rendering complex scenes, due to more limited memory, and issues with interactivity when using the same graphics card for display and rendering.

Cycles has two GPU rendering modes: **CUDA**, which is the preferred method for NVIDIA graphics cards; and **OpenCL**, which supports rendering on AMD graphics cards.

### Configuration

To enable GPU rendering, go into the User Preferences, and under the System tab, select the Compute Device(s) to use. Next, for each scene, you can configure to use CPU or GPU rendering in the Render properties.

**CUDA** NVIDIA CUDA (Compute Unified Device Architecture) is supported for GPU rendering with **NVIDIA graphics cards**. We support graphics cards starting from GTX 4xx (computing capability 2.0).

Cycles requires recent NVIDIA drivers to be installed, on all operating systems.

[List of CUDA cards with shader model](#)

**OpenCL** OPENCL (Open Computing Language) is supported for GPU rendering with **AMD graphics cards**. We only support graphics cards with GCN (Graphics Core Next) architecture (HD 7xxx and above). Not all HD 7xxx cards are GCN cards though, you can check if your card is [here](#).

Cycles requires recent AMD drivers to be installed, on all operating systems.

### Supported Features and Limitations

For an overview of supported features, check the comparison in the [Features](#).

**CUDA:** The maximum amount of individual textures is limited to 95 byte-image textures (PNG, JPEG, ..) and 5 float-image textures (OpenEXR, 16 bit TIFF, ..) on GTX 4xx/5xx cards, and 145 byte-image textures and 5 float-image textures on GTX6xx cards and above.

**OpenCL:** No support for HDR (float) textures at the moment.

## Frequently Asked Questions

**Why is Blender unresponsive during rendering?** While a graphics card is rendering, it can not redraw the user interface, which makes Blender unresponsive. We attempt to avoid this problem by giving back control over the GPU as often as possible, but a completely smooth interaction can't be guaranteed, especially on heavy scenes. This is a limitation of graphics cards for which no true solution exists, though we might be able to improve this somewhat in the future.

If possible, it is best to install more than one GPU, using one for display and the other(s) for rendering.

**Why does a scene that renders on the CPU not render on the GPU?** There maybe be multiple causes, but the most common is that there is not enough memory on your graphics card. We can currently only render scenes that fit in graphics card memory, and this is usually smaller than that of the CPU. Note that, for example, 8k, 4k, 2k and 1k image textures take up respectively 256MB, 64MB, 16MB and 4MB of memory.

We do intend to add a system to support scenes bigger than GPU memory, but this will not be added soon.

**Can I use multiple GPUs for rendering?** Yes, go to User Preferences > System > Compute Device Panel, and configure it as you desire.

**Would multiple GPUs increase available memory?** No, each GPU can only access its own memory.

**What renders faster, NVIDIA or AMD, CUDA or OpenCL?** Currently NVIDIA with CUDA is rendering faster. There is no fundamental reason why this should be so - we don't use any CUDA - specific features - but the compiler appears to be more mature, and can better support big kernels. OpenCL support is still in an early stage and has not been optimized as much.

## Error Messages

**Unsupported GNU version! gcc 4.7 and up are not supported!** On Linux, depending on your GCC version you might get this error.

If so, delete the following line in `/usr/local/cuda/include/host_config.h`

```
#error -- unsupported GNU version! gcc 4.7 and up are not supported!
```

**CUDA Error: Invalid kernel image** If you get this error on Windows 64-bit, be sure to use the 64-bit build of Blender, not the 32-bit version.

**CUDA Error: Out of memory** This usually means there is not enough memory to store the scene on the GPU. We can currently only render scenes that fit in graphics card memory, and this is usually smaller than that of the CPU. See above for more details.

**The NVIDIA OpenGL driver lost connection with the display driver** If a GPU is used for both display and rendering, Windows has a limit on the time the GPU can do render computations. If you have a particularly heavy scene, Cycles can take up too much GPU time. Reducing Tile Size in the Performance panel may alleviate the issue, but the only real solution is to use separate graphics cards for display and rendering.

Another solution can be to increase the timeout, although this will make the user interface less responsive when rendering heavy scenes. <http://msdn.microsoft.com/en-us/windows/hardware/gg487368.aspx>

**CUDA error: Unknown error in cuCtxSynchronize()** An unknown error can have many causes, but one possibility is that it's a timeout. See the above answer for solutions.

## Open Shading Language

Users can now create their own nodes using [Open Shading Language \(OSL\)](#). Note that these nodes will only work for CPU rendering; there is no support for running OSL code on the GPU.

To enable it, select Open Shading Language as the shading system in the render settings.

---

**Note:** On Linux, C/C++ compiler tools (in particular /usr/bin/cpp) must be installed to compile OSL scripts.

---

### Script Node

OSL was designed for node-based shading, and **each OSL shader corresponds to a node** in a node setup. To add an OSL shader, add a script node and link it to a text datablock or an external file. Input and output sockets will be created from the shader parameters on clicking the update button in the node or the text editor.

OSL shaders can be linked to the node in a few different ways. With the **Internal** mode, a text datablock is used to store the OSL shader, and the OSO bytecode is stored in the node itself. This is useful for distributing a .blend file with everything packed into it.

The **External** mode can be used to specify a .osl file on disk, and this will then be automatically compiled into a .oso file in the same directory. It is also possible to specify a path to a .oso file, which will then be used directly, with compilation done manually by the user. The third option is to specify just the module name, which will be looked up in the shader search path.

The shader search path is located in the same place as the scripts or configuration path, under:

**Linux** /home/\$user/.config/blender/ *Version Number* /shaders/

**Windows** C:\Users\\$user\AppData\Roaming\Blender Foundation\Blender\ *Version Number* \shaders\

**Mac OS X** /Users/\$user/Library/Application Support/Blender/ *Version Number* /shaders/

(Replace *Version Number* with the release number of your current Blender installation, e.g. 2.65 or 2.66.)

For use in production, we suggest to **use a node group to wrap shader script nodes**, and link that into other .blend files. This makes it easier to make changes to the node afterwards as sockets are added or removed, without having to update the script nodes in all files.

### Writing Shaders

For more details on how to write shaders, see the [OSL specification](#). Here is a simple example:

```
shader simple_material(
    color Diffuse_Color = color(0.6, 0.8, 0.6),
    float Noise_Factor = 0.5,
    output closure color BSDF = diffuse(N)
{
    color material_color = Diffuse_Color * mix(1.0, noise(P * 10.0), Noise_Factor);
    BSDF = material_color * diffuse(N);
}
```

## Closures

OSL is different from, for example, RSL or GLSL, in that it does not have a light loop. There is no access to lights in the scene, and the material must be built from closures that are implemented in the render engine itself. This is more limited, but also makes it possible for the render engine to do optimizations and ensure all shaders can be importance sampled.

The available closures in Cycles correspond to the shader nodes and their sockets; for more details on what they do and the meaning of the parameters, see the [shader nodes manual](#).

## BSDF

- `diffuse(N)`
- `oren_nayar(N, roughness)`
- `diffuse_ramp(N, colors[8])`
- `phong_ramp(N, exponent, colors[8])`
- `diffuse_toon(N, size, smooth)`
- `glossy_toon(N, size, smooth)`
- `translucent(N)`
- `reflection(N)`
- `refraction(N, ior)`
- `transparent()`
- `microfacet_ggx(N, roughness)`
- `microfacet_ggx_aniso(N, T, ax, ay)`
- `microfacet_ggx_refraction(N, roughness, ior)`
- `microfacet_beckmann(N, roughness)`
- `microfacet_beckmann_aniso(N, T, ax, ay)`
- `microfacet_beckmann_refraction(N, roughness, ior)`
- `ashikhmin_shirley(N, T, ax, ay)`
- `ashikhmin_velvet(N, roughness)`

## Hair

- `hair_reflection(N, roughnessu, roughnessv, T, offset)`
- `hair_transmission(N, roughnessu, roughnessv, T, offset)`

## BSSRDF

- `bssrdf_cubic(N, radius, texture_blur, sharpness)`
- `bssrdf_gaussian(N, radius, texture_blur)`

## Volume

- `henyey_greenstein(g)`
- `absorption()`

**Other**

- `emission()`
- `ambient_occlusion()`
- `holdout()`
- `background()`

**Attributes**

Some object, particle and mesh attributes are available to the built-in `getattribute()` function. UV maps and vertex colors can be retrieved using their name. Other attributes are listed below:

**geom:generated** Generated texture coordinates

**geom:uv** Default render UV map

**geom:dupli\_generated** For instances, generated coordinate from duplicator object

**geom:dupli\_uv** For instances, UV coordinate from duplicator object

**geom:trianglevertices** 3 vertex coordinates of the triangle

**geom:numpolyvertices** Number of vertices in the polygon (always returns 3 currently)

**geom:polyvertices** Vertex coordinates array of the polygon (always 3 vertices currently)

**geom:name** Name of the object

**geom:is\_curve** Is object a strand or not

**geom:curve\_intercept** Point along the strand, from root to tip

**geom:curve\_thickness** Thickness of the strand

**geom:curve\_tangent\_normal** Tangent Normal of the strand

**path:ray\_length** Ray distance since last hit

**object:location** Object location

**object:index** Object index number

**object:random** Per object random number generated from object index and name

**material:index** Material index number

**particle:index** Particle instance number

**particle:age** Particle age in frames

**particle:lifetime** Total lifespan of particle in frames

**particle:location** Location of the particle

**particle:size** Size of the particle

**particle:velocity** Velocity of the particle

**particle:angular\_velocity** Angular velocity of the particle

## Trace

We support the `trace(point pos, vector dir, ...)` function, to trace rays from the OSL shader. The “shade” parameter is not supported currently, but attributes can be retrieved from the object that was hit using the `getmessage("trace", ...)` function. See the OSL specification for details on how to use this.

This function can’t be used instead of lighting; the main purpose is to allow shaders to “probe” nearby geometry, for example to apply a projected texture that can be blocked by geometry, apply more “wear” to exposed geometry, or make other ambient occlusion-like effects.

## Render Baking

Refer to the Blender Render page for [general baking guidelines](#)

Cycles uses the render settings (samples, bounces, ...) for baking. This way the quality of the baked textures should match the result you get from the rendered scene.

The baking happens into the respective active textures of the object materials. The active texture is the last selected Image Texture node of the material nodetree. That means the active object (or the selected objects, when not baking ‘Selected to Active’) needs a material, and that material needs at least an Image Texture node, with the image to be used for the baking. Note, the node doesn’t need to be connected to any other node. The active texture is what projection painting and the viewport use as a criteria to which image to use. This way after the baking is done you can automatically preview the baked result in the Texture mode.

## Options

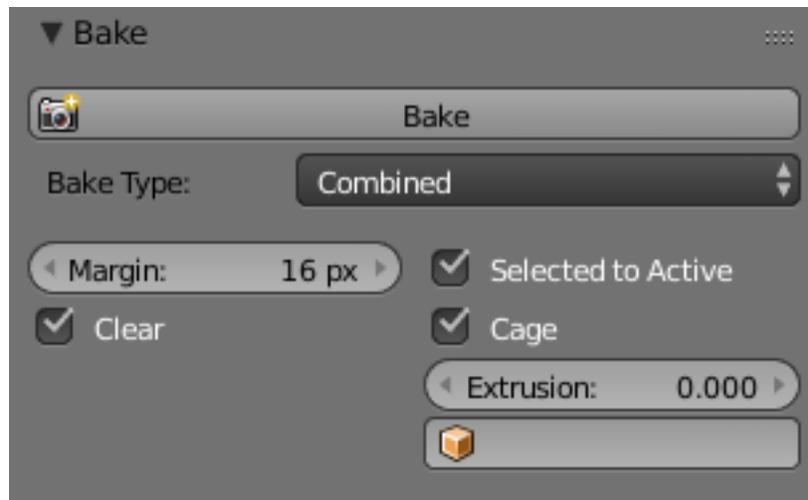


Fig. 2.2338: Combined Pass

## Bake Mode

**Combined** Bakes all materials, textures, and lighting except specularly and SSS.

**Ambient Occlusion** Bakes ambient occlusion as specified in the World panels. Ignores all lights in the scene.

**Shadow** Bakes shadows and lighting.

**Normals** Bakes normals to an RGB image.

**Normal Space** Normals can be baked in different spaces:

**World space** Normals in world coordinates, dependent on object transformation and deformation.

**Object space** Normals in object coordinates, independent of object transformation, but dependent on deformation.

**Tangent space** Normals in tangent space coordinates, independent of object transformation and deformation. This is the default, and the right choice in most cases, since then the normal map can be used for animated objects too.

**Normal Swizzle** Axis to bake into the red, green and blue channel.

For materials the same spaces can be chosen in the image texture options next to the existing *Normal Map* setting. For correct results, the setting here should match the setting used for baking.

**UV** Bakes colors of materials and textures only, without shading.

**Emit** Bakes Emission, or the Glow color of a material.

**Environment** Bakes the environment as seen from the center of the object.

**Diffuse Color/Direct/Indirect** Bakes the diffuse pass of a material. Diffuse Color is a property of the surface and independent of sampling refinement.

**Glossy Color/Direct/Indirect** Bakes the glossiness of a material. Glossy Color is a property of the surface and independent of sampling refinement.

**Transmission Color/Direct/Indirect** Bakes the transmission of a material. Transmission Color is a property of the surface and independent of sampling refinement.

**Subsurface Color/Direct/Indirect** Bakes the subsurface pass of a material. Subsurface Color is a property of the surface and independent of sampling refinement.

### Additional Options

**Margin** Baked result is extended this many pixels beyond the border of each UV “island,” to soften seams in the texture.

**Clear** If selected, clears the image before baking render.

**Select to Active** Bake shading on the surface of selected objects to the active object. The rays are cast from the lowpoly object inwards towards the highpoly object. If the highpoly object is not entirely involved by the lowpoly object, you can tweak the rays start point with *Ray Distance* or *Cage Extrusion* (depending on whether or not you are using cage). For even more control you can use a *Cage Object*.

---

### Note: Memory Usage

There is a CPU fixed memory footprint for every object used to bake from. In order to avoid crashes due to lack of memory the highpoly objects can be joined before the baking process. The render tiles parameter also influence the memory usage, so the bigger the tile the less overhead you have, but the more memory it will take during baking (either in GPU or CPU).

---

**Cage** Cast rays to active object from a cage. A cage is a ballooned-out version of the lowpoly mesh created either automatically (by adjusting the ray distance) or manually (by specifying an object to use). When not using a cage the rays will conform to the mesh normals. This produces glitches on the edges, but it’s a preferable method when baking into planes to avoid the need of adding extra loops around the edges.

**Ray Distance** Distance to use for the inward ray cast when using selected to active. Ray distance is only available when not using *Cage*.

**Cage Extrusion** Distance to use for the inward ray cast when using *Selected to Active* and *Cage*. The inward rays are casted from a version of the active object with disabled Edge Split modifiers. Hard splits (e.g., when the Edge Split modifier is applied) should be avoided because they will lead to non-smooth normals around the edges.

**Cage** Object to use as cage instead of calculating the cage from the active object with the *Cage Extrusion*.

---

**Note: Cage**

When the base mesh extruded doesn't give good results, you can create a copy of the base mesh and modify it to use as a *Cage*. Both meshes need to have the same *topology* (number of faces and face order).

---

## 2.8.4 Camera

### Introduction

A *Camera* is an object that provides a means of rendering images from Blender. It defines which portions of a scene is visible in the rendered image.

A scene can contain more than one camera, but only one of them will be used at a time.

### Add a New Camera

---

#### Reference

Mode: *Object* mode

Menu: *Add* → *Camera*

Hotkey: *Shift-A* to add new.

---

In *Object* mode simply press *Shift-A* and in the pop-up menu, choose *Add* → *Camera*.

The default scene in Blender includes a camera, so you'll probably only need to add a new one if you have deleted the default one, or need to animate a cut between two cameras.

### Changing the Active Camera

---

#### Reference

Mode: *Object* mode

Hotkey: *Ctrl-Numpad0*

---

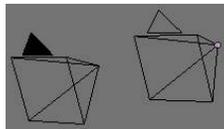


Fig. 2.2339: Active camera (left one).

The *active* camera is the camera that is currently being used for rendering and camera view (*Numpad0*).

---

Select the camera you would like to make active and press `Ctrl-Numpad0` (by doing so, you also switch the view to camera view). In order to render, each scene **must** have an active camera.

The active camera can also be set in the *Scene* context of the *Properties Editor*

The camera with the solid triangle on top is the active camera.

**Warning:** The active camera, as well as the layers, can be specific to a given view, or global (locked) to the whole scene - see [Local Camera](#).

## Camera Settings

### Reference

Mode: Object mode

Editor: Properties

Context: Object Data

Cameras are invisible in renders, so they don't have any material or texture settings. However, they do have *Object* and *Editing* setting panels available which are displayed when a camera is the selected (active!) object.

**Lens** The camera lens options control the way 3D objects are represented in a 2D image. See [Camera Lens](#) for details.

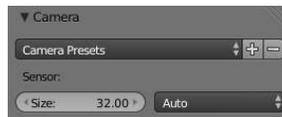


Fig. 2.2340: Camera Presets panel.

## Camera

**Sensor size** This setting is an alternative way to control the focal-length, it's useful to match the camera in Blender to a physical camera & lens combination, e.g. for [motion tracking](#).

**Depth of Field** Real world cameras transmit light through a lens that bends and focuses it onto the sensor. Because of this, objects that are a certain distance away are in focus, but objects in front and behind that are blurred.

The area in focus is called the *focal point* and can be set using either an exact value, or by using the distance between the camera and a chosen object:

**Focus Object** Choose an object which will determine the focal point. Linking an object will deactivate the distance parameter. Typically this is used to give precise control over the position of the focal point, and also allows it to be animated or constrained to another object.

**Distance** Sets the distance to the focal point when no *Focus Object* is specified. If *Limits* are enabled, a yellow cross is shown on the camera line of sight at this distance.

**Hint:** Hover the mouse over the *Distance* property and press `E` to use a special *Depth Picker*. Then click on a point in the 3D View to sample the distance from that point to the camera.

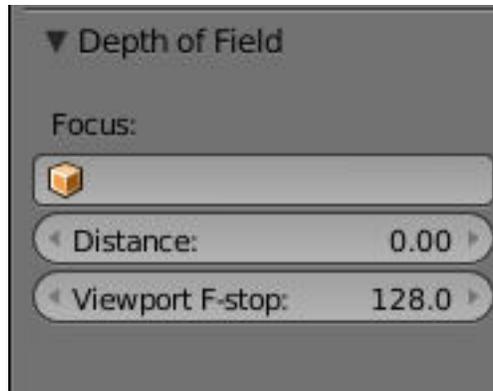


Fig. 2.2341: Camera Depth of Field Panel

**Viewport F-stop** Controls the real-time focal blur effect used during sequencer or OpenGL rendering and, when enabled, camera views in the 3D viewport. The amount of blur depends on this setting, along with Focal Length and Sensor Size. Smaller Viewport F-stop values result in more blur.

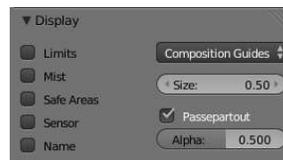


Fig. 2.2342: Camera Display panel

## Display

**Limits** Shows a line which indicates *Start* and *End Clipping* values.

**Mist** Toggles viewing of the mist limits on and off. The limits are shown as two connected white dots on the camera line of sight. The mist limits and other options are set in the *World* panel, in the [Mist](#) section.

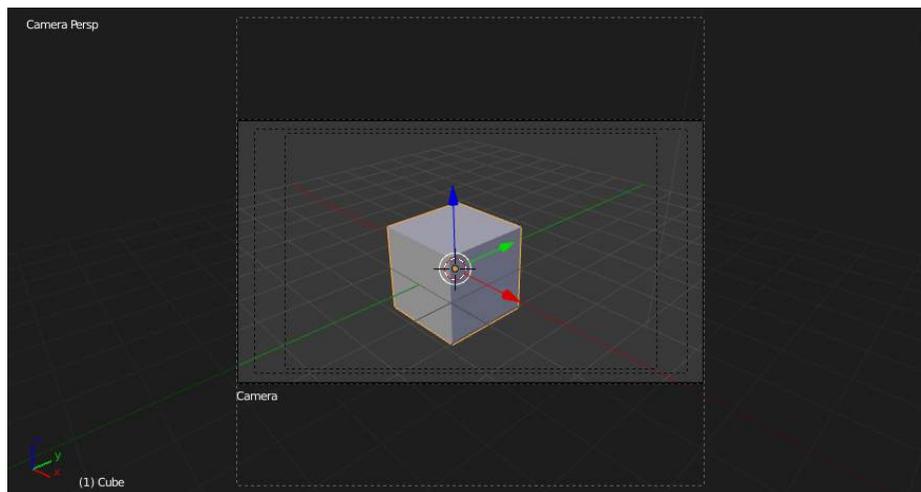


Fig. 2.2343: Camera view displaying safe areas, sensor and name

**Sensor** Displays a dotted frame in camera view.

**Name** Toggle name display on and off in camera view.

**Size** Size of the camera icon in the 3D view. This setting has no effect on the render output of a camera, and is only a cosmetic setting. The camera icon can also be scaled using the standard Scale *S* transform key.

**Passepartout, Alpha** This mode darkens the area outside of the camera's field of view, based on the *Alpha* setting.

**Composition Guides** *Composition Guides* are available from the drop-down menu, which can help when framing a shot. There are 8 types of guides available:

**Center** Adds lines dividing the frame in half vertically and horizontally.

**Center Diagonal** Adds lines connecting opposite corners.

**Thirds** Adds lines dividing the frame in thirds vertically and horizontally.

**Golden** Divides the width and height into Golden proportions (About 0.618 of the size from all sides of the frame).

**Golden Triangle A** Draws a diagonal line from the lower-left to upper-right corners, then adds perpendicular lines that pass through the top left and bottom right corners.

**Golden Triangle B** Same as A, but with the opposite corners.

**Harmonious Triangle A** Draws a diagonal line from the lower-left to upper-right corners, then lines from the top left and bottom right corners to 0.618 the lengths of the opposite side.

**Harmonious Triangle B** Same as A, but with the opposite corners.

**Safe Areas** When this is enabled, extra dotted frames are drawn when in camera view, delimiting the area considered as "safe" for important elements. More information about them in the *safe areas* section.

## Camera Navigation

There are several different ways to navigate and position the camera in your scene, some of them are explained below.

**Note:** Remember that the active "camera" might be any kind of object. So these actions can be used, for example, to position and aim a lamp.

---

### Move active camera to view

#### Reference

Mode: *Object* mode

Hotkey: `Ctrl-Alt-Numpad0`

---

This feature allows you to position and orient the active camera to match your current viewport.

Select a camera and then move around in the 3D view to a desired position and direction for your camera (so that you're seeing what you want the camera to see). Now press `Ctrl-Alt-Numpad0` and your selected camera positions itself to match the view, and switches to camera view.

**Camera View Positioning** By enabling *Lock Camera to View* in the View menu of the View Properties panel, while in camera view, you can navigate the 3d viewport as usual, while remaining in camera view. Controls are exactly the same as when normally moving in 3d.

**Roll, Pan, Dolly, and Track** To perform these camera moves, the camera must first be *selected*, so that it becomes the active object (while viewing through it, you can RMB -click on the solid rectangular edges to select it). The following actions also assume that you are in camera view (Numpad0)! Having done so, you can now manipulate the camera using the same commands that are used to manipulate any object:

**Roll** Press R to enter object rotation mode. The default will be to rotate the camera in its local Z-axis (the axis orthogonal to the camera view), which is the definition of a camera “roll”.

**Vertical Pan or Pitch** This is just a rotation along the local X-axis. Press R to enter object rotation mode, then X twice (the first press selects the *global* axis - pressing the same letter a second time selects the *local* axis - this works with any axis; see the [axis locking page](#)).

**Horizontal Pan or Yaw** This corresponds to a rotation around the camera’s local Y axis... Yes, that’s it, press R, and then Y twice!

**Dolly** To dolly the camera, press G then MMB (or Z twice).

**Sideways Tracking** Press G and move the mouse (you can use X twice or Y to get pure-horizontal or pure-vertical sideways tracking).

**Aiming the camera in Flymode** When you are in *Camera* view, the `fly` mode actually moves your active camera...

## Camera Lens

### Reference

Editor: Properties

Context: Object Data

Panel: Lens



Fig. 2.2344: Camera Lens panel.

The camera lens options control the way 3D objects are represented in a 2D image.

### Lens Types

There are three different lens types:

- *Perspective*
- *Orthographic*
- *Panoramic*

**Perspective** This matches how you view things in the real-world. Objects in the distance will appear smaller than objects in the foreground, and parallel lines (such as the rails on a railroad) will appear to converge as they get farther away.



Fig. 2.2345: Render of a train track scene with a *Perspective* camera.

Settings which adjust this projection include:

- Focal length
- *Shift*
- *Sensor size*

**Focal length** The *focal length* setting controls the amount of zoom, i.e. the amount of the scene which is visible all at once. Longer focal lengths result in a smaller FOV (Field of View) (more zoom), while short focal lengths allow you to see more of the scene at once (larger FOV, less zoom).

**Lens Unit** The focal length can be set either in terms of millimeters or the actual *Field of View* as an angle.

**Orthographic** With *Orthographic* perspective objects always appear at their actual size, regardless of distance. This means that parallel lines appear parallel, and do not converge like they do with *Perspective*.

**Orthographic Scale** This controls the apparent size of objects in the camera.

Note that this is effectively the only setting which applies to orthographic perspective. Since parallel lines do not converge in orthographic mode (no vanishing points), the lens shift settings are equivalent to translating the camera in the 3D view.

**Panoramic** Panoramic cameras are only supported in the Cycles render engine. See *the Cycles documentation*.



Fig. 2.2346: Render of the same scene as above, but with a focal length of 210mm instead of 35mm.



Fig. 2.2347: Render from the same camera angle as the previous examples, but with orthographic perspective.

## Shift

The *Shift* setting allows for the adjustment of *vanishing points*. *Vanishing points* refer to the positions to which parallel lines converge. In this example, the most obvious vanishing point is at the end of the railroad.

To see how this works, take the following examples:

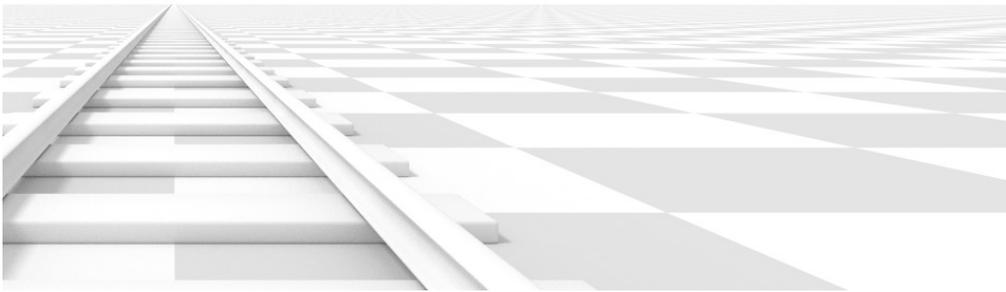


Fig. 2.2348: Render of a train track scene with a horizontal lens shift of 0.330.

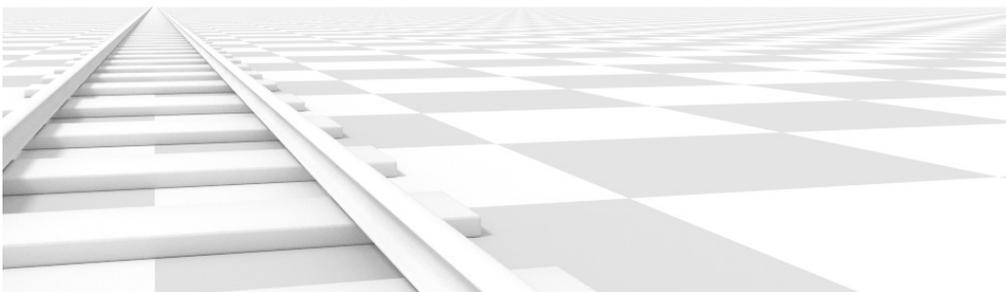


Fig. 2.2349: Render of a train track scene with a rotation of the camera object instead of a lens shift.

Notice how the horizontal lines remain perfectly horizontal when using the lens shift, but do get skewed when rotating the camera object.

Using lens shift is equivalent to rendering an image with a larger FOV and cropping it off-center.

## Clipping

Set the clipping limits with the *Start* and *End* values. Only objects within the limits are rendered. If *Limits* in the *Display* panel is enabled, the clip bounds will be visible as two yellow connected dots on the camera line of sight.

**Note:** The *3D View* window contains settings similar to the camera, see the [3D view options](#) page for more details.

## Safe Areas

Safe areas are guides used to position elements to ensure that the most important parts of the content can be seen across all screens.

Different screens have varying amounts of *overscan*. (specially older TV sets). That means that not all content will be visible to all viewers, since parts of the image surrounding the edges are not shown. To work around this problem TV producers defined two areas where content is guaranteed to be shown: action safe and title safe.

Modern LCD/plasma screens with purely digital signals have no *overscan*, yet safe areas are still considered best practice and may be legally required for broadcast.

In Blender, safe areas can be set from the Camera and Sequencer views.

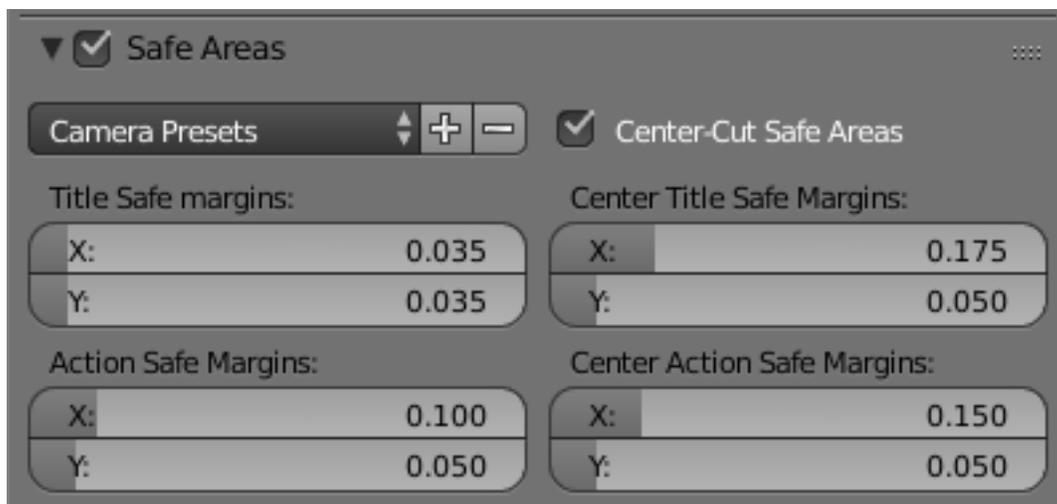


Fig. 2.2350: The Safe areas panel found in the camera properties, and the view mode of the sequencer.

## Main Safe Areas

**Title Safe** Also known as *Graphics Safe*. Place all important information (graphics or text) inside this area to ensure it can be seen by the majority of viewers.

**Action Safe** Make sure any significant action or characters in the shot are inside this area. This zone also doubles as a sort of “margin” for the screen which can be used to keep elements from piling up against the edges.

### Tip: Legal Standards

Each country sets a legal standard for broadcasting. These include, among other things, specific values for safe areas. Blender defaults for safe areas follow the EBU (European Union) standard. Make sure you’re using the correct values when working for broadcast to avoid any trouble.

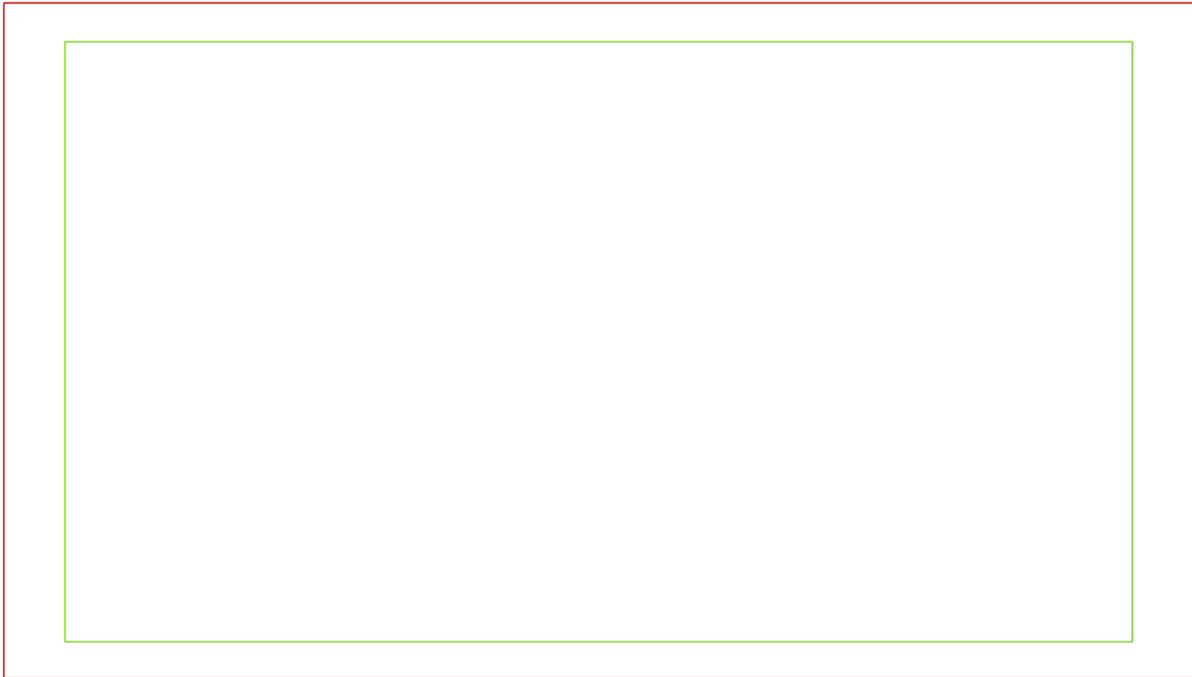


Fig. 2.2351: **Red line:** Action safe. **Green line:** Title safe

### Center-Cuts

Center-cuts are a second set of safe areas to ensure content is seen correctly on screens with a different aspect ratio. Old TV sets receiving 16:9 or 21:9 video will cut off the sides. Position content inside the center-cut areas to make sure the most important elements of your composition can still be visible in these screens.

Blender defaults show a 4:3 (square) ratio inside 16:9 (wide-screen).

## 2.8.5 Displaying and Saving Images

Rendering still images is fairly simple. [Rendering Animations](#) is a bit more complex and is covered in the next sections.

To render an image from the active camera, in the Render Panel, press the *Render* button. By default the 3D view is replaced with the UV/Image Editor and the render appears.

### Displaying Renders

Renders are displayed in the Image Editor. You can set the way this is displayed to several different options in the Display menu:

**Keep UI** The image is rendered to the Image Editor, but the UI remains the same. You will need to open the Image Editor manually to see the render result.

**New Window** A new floating window opens up, displaying the render.

**Image Editor** One of the existing editors is replaced with the Image Editor, showing the render.

**Full Screen** The Image editor replaces the UI, showing the render.

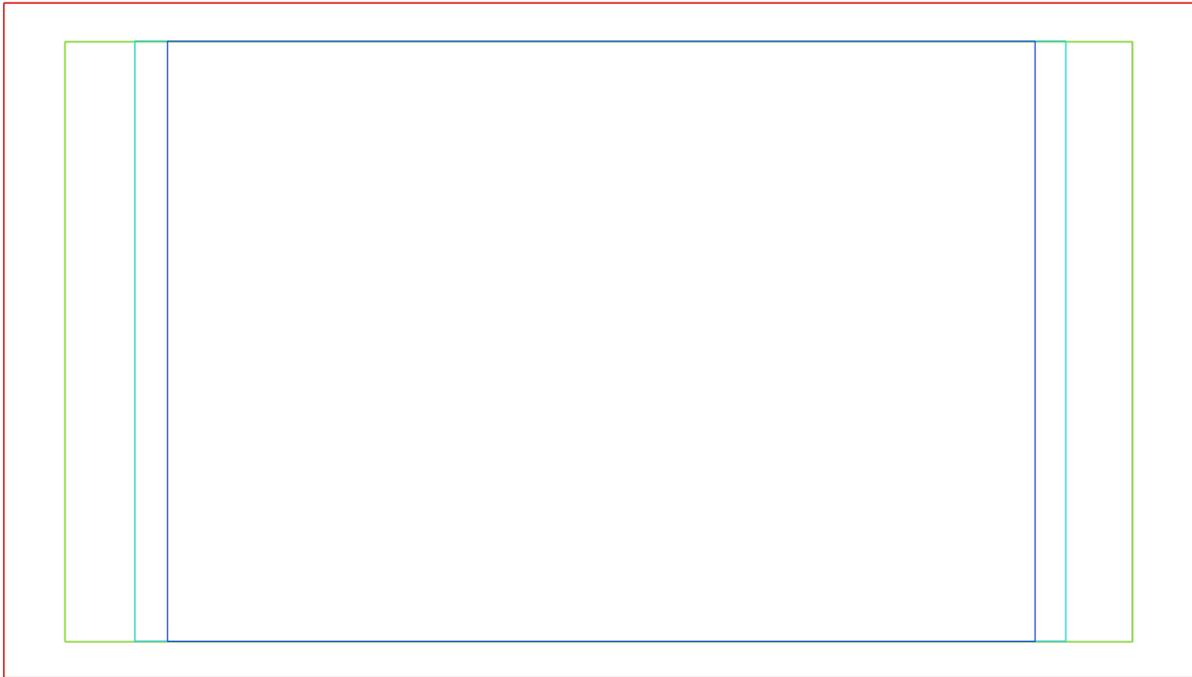


Fig. 2.2352: **Cyan line:** action center safe. **Blue line:** title center safe

For each of these options, pressing `ESC` will close the render view and return to the previous view.

## Saving

Rendered images can be saved like any other image: Using *Image* → *Save As Image* or by pressing `F3`

## Display Options

When a rendered image is displayed in the Image Editor, several new menu items become available.

**Slot Menu** You can save successive renders into the render buffer by selecting a new slot before rendering. If an image has been rendered to a slot, it can be viewed by selecting that slot. Empty slots appear as blank grids in the image editor. Use the `J` and `Alt-J` to cycle forwards and backwards through saved renders.

**Render Layer** If you are using [Render Layers](#), use this menu to select which layer is displayed.

**Render Pass** If you are using [Render Passes](#), use this menu to select which pass is displayed.

**Display Mode** The last four buttons set how the image is displayed.

**RGB** Draw image as rendered, without alpha channel.

**RGBA** Replaces transparent pixels with background checkerboard, denoting the alpha channel.

**Alpha Channel** Displays a gray-scale image. White areas are opaque, black areas have an alpha of 0.

**Z Depth** Display the depth from the camera, from Clip Start to Clip End, as specified in the [Camera settings](#).

## Animation Playback

The ‘Play’ button in the render panel will play back your rendered animation in a new window.

You can also drop images or movie files in a running animation player. It will then restart the player with the new data.

The following table shows the available hotkeys for the animation player:

Hotkey	Action
A	Toggle frame skipping.
P	Toggle ping-pong.
F	Flip drawing on the X axis.
Shift-F	Flip drawing on the Y axis.
Return	Start playback (when paused).
Numpad0	Toggle looping.
NumpadPeriod	Manual frame stepping.
Left	Step back one frame.
Right	Step forward one frame.
Down	Step back 10 frames.
Up	Step forward 10 frames.
Shift-Down	Use backward playback.
Shift-Up	Use forward playback.
Shift	Hold to show frame numbers.
LMB	Scrub in time.
Ctrl-Plus	Zoom in
Ctrl-Minus	Zoom out
Esc	Quit
Numpad1	60 fps
Numpad2	50 fps
Numpad3	30 fps
Numpad4	25 fps
Shift-Numpad4	24 fps
Numpad5	20 fps
Numpad6	15 fps
Numpad7	12 fps
Numpad8	10 fps
Numpad9	6 fps
NumpadSlash	5 fps
Minus	Slow down playback.
Plus	Speed up playback.

An external player can be used instead by selecting it in the [User Preferences](#).

## 2.8.6 Output Options

The first step in the rendering process is to determine and set the output options. This includes render size, frame rate, pixel aspect ratio, output location, and file type.

### Dimensions

#### Resolution

**X/Y** The number of pixels horizontally and vertically in the image.

**Percentage slider** Reduce or increase the size of the rendered image relative to the X/Y values above. This is useful for small test renders that are the same proportions as the final image.

**Aspect Ratio** Older televisions may have non-square pixels, so this can be used to control the shape of the pixels along the respective axis.

See [Video Output](#) for details on pixel aspect ratio.

**Border** You can render just a portion of the view instead of the entire frame. While in Camera View, press `Ctrl-B` and drag a rectangle to define the area you want to render. `Ctrl-Alt-B` is the shortcut to disable the border.

---

**Note:** This disables the *Save Buffers* option in *Performance* and *Full Sample* option in *Anti-Aliasing*.

---

Enabling *Crop* will crop the rendered image to the *Border* size, instead of rendering a black region around it.

**Frame Range** Set the *Start* and *End* frames for [Rendering Animations](#). *Step* controls the number of frames to advance by for each frame in the timeline.

**Frame Rate** For an [Animation](#) the frame rate is how many frames will be displayed per second.

**Time Remapping** Use to remap the length of an animation.

## Presets

To make life easier the topmost menu provides some common presets. You can add your own or remove one with the + and - buttons:

## Output Panel

This panel provides options for setting the location of rendered frames for animations, and the quality of the saved images.

**File Path** Choose the location to save rendered frames.

When rendering an animation, the frame number is appended at the end of the file name with 4 padded zeros (e.g. *image0001.png*). You can set a custom padding size by adding the appropriate number of #’s at the end of the file name (e.g. *image\_##.png* would translate to *image\_01.png*).

**Overwrite** Overwrite existing files when rendering

**Placeholders** Create empty placeholder frames while rendering

**File Extensions** Adds the correct file extensions per file type to the output files

**Cache Result** Saves the rendered image to your hard drive. This is helpful for heavy compositing.

**Output Format** Choose the file format to save to. Based on which format is used, other options such as channels, bit-depth and compression level are available.

## 2.8.7 Video Output

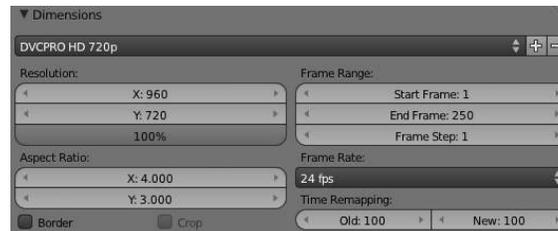
### Preparing your work for video

Once you have mastered the trick of animation you will surely start to produce wonderful animations, encoded with your favorite codecs, and possibly you’ll share them on the Internet with the rest of the community.

Sooner or later you will be struck with the desire to build an animation for television, or maybe burn your own DVDs. To spare you some disappointment, here are some tips specifically targeted at Video preparation. The first and principal one is to remember the double-dashed white lines in the camera view!

If you render for PC then the whole rendered image which lies within the *outer* dashed rectangle will be shown. For television, some lines and some part of the lines will be lost due to the mechanics of the electron beam scanning in your TV's cathode ray tube. You are guaranteed that what is within the *inner* dashed rectangle in camera view will be visible on the screen. Everything within the two rectangles may or may not be visible, depending on the given TV set that your audience watches the video on.

## Dimensions Presets



The rendering size is strictly dictated by the TV standard. There are various popular presets included, more can be added for your convenience.

Saved information is:

- Resolution** X, Y & percentage scale
- Aspect ratio** pixel aspect ratio
- Frame rate** frames per second, for animation

See also [Dimensions](#)

## Pixel Aspect Ratio

Unlike regular computer monitors, some screens (typically older TV sets) do *not* have the square pixels making it necessary to generate *pre-distorted* images which will look stretched on a computer but which will display correctly on a TV set. It is important that you use the correct pixel aspect ratio when rendering to prevent re-scaling, resulting in lowered image quality.

## Color Saturation

Most video tapes and video signals are not based on the RGB model but on the YCrCb model: more precisely, the YUV in Europe (PAL), and the YIQ in the USA (NTSC), the latter being quite similar to the former. Hence some knowledge of this is necessary too.

The YCrCb model sends information as 'Luminance', or intensity (Y) and two 'Chrominance' signals, red and blue (Cr and Cb). Actually a Black and White TV set shows only luminance, while color TV sets reconstruct color from Chrominances (and from luminance). Construction of the YCrCb values from the RGB ones takes two steps (the constants *in italics* depend on the system: PAL or NTSC):

First, the Gamma correction (*g* varies: 2.2 for NTSC, 2.8 for PAL):

- $R' = R^{1/g}$  ;  $G' = G^{1/g}$
- $B' = B^{1/g}$

Then, the conversion itself:

- $Y = 0.299R' + 0.587G' + 0.114B'$
- $Cr = a_1 (R' - Y) + b_1 (B' - Y)$
- $Cb = a_2 (R' - Y) + b_2 (B' - Y)$

Whereas a standard 24 bit RGB picture has 8 bits for each channel, to keep bandwidth down, and considering that the human eye is more sensitive to luminance than to chrominance, the luminance signal is sent with more bits than the two chrominance signals. This bit expansion results in a smaller dynamic of colors in video, than what you are used to on monitors. You hence have to keep in mind that not all colors can be correctly displayed.

A rule of thumb is to keep the colors as 'grayish' or 'unsaturated' as possible; this roughly means keeping the dynamics of your colors within 80% of one another. In other words, the difference between the highest RGB value and the lowest RGB value should not exceed 0.8 ([0-1] range) or 200 ([0-255] range).

This is not strict - something more than 0.8 is acceptable - but an RGB display with color contrast that ranges from 0.0 to 1.0 will appear to be very ugly (over-saturated) on video, while appearing bright and dynamic on a computer monitor.

### Rendering to fields

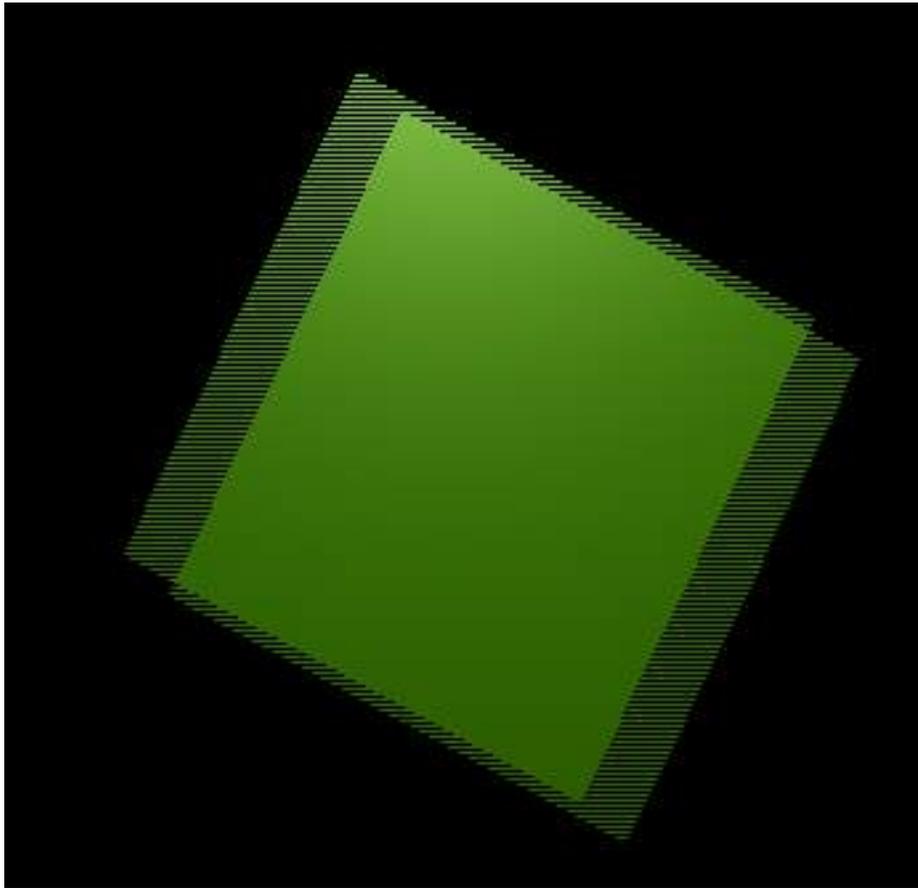


Fig. 2.2353: Field Rendering result.

The TV standards prescribe that there should be 25 frames per second (PAL) or 30 frames per second (NTSC). Since the phosphors of the screen do not maintain luminosity for very long, this could produce a noticeable flickering.

To minimize this, a TV does not represent frames as a computer does ('progressive' mode), but rather represents half-frames, or *fields* at a double refresh rate, hence 50 half frames per second on PAL and 60 half frames per second on NTSC. This was

originally bound to the frequency of power lines in Europe (50Hz) and the US (60Hz).

In particular, fields are “interlaced” in the sense that one field presents all the even lines of the complete frame and the subsequent field the odd ones.

Since there is a non-negligible time difference between each field (1/50 or 1/60 of a second) merely rendering a frame the usual way and splitting it into two half frames does not work. A noticeable jitter of the edges of moving objects would be present.

## Options

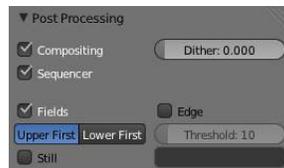


Fig. 2.2354: Field Rendering setup.

**Fields** Enable field rendering. When the *Fields* button in the *Render Panel* is pressed (*Post Processing* section), Blender prepares each frame in two passes. On the first it renders only the even lines, then it *advances in time by half a time step* and renders all the odd lines. This produces odd results on a PC screen (*Field Rendering result*), but will show correctly on a TV set.

**Upper First / Lower First** Toggles between rendering the even and odd frames first.

**Still** Disables the half-frame time step between fields (x).

**Note:** Setting up the correct field order

Blender’s default setting is to produce Even fields *before* Odd fields; this complies with European PAL standards. Odd fields are scanned first on NTSC.

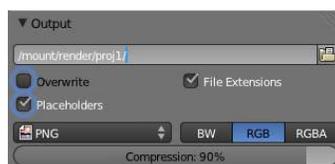
Of course, if you make the wrong selection things are even worse than if no Field rendering at all was used!

If you are really confused, a simple trick to determine the correct field order is to render a short test animation of a white square moving from left to right on a black background. Prepare one version with odd field order and another with even field order, and look at them on a television screen. The one with the right field order will look smooth and the other one horrible. Doing this simple test will save you *hours* of wasted rendering time...

**Note:** Fields and Composite Nodes

Nodes are currently not field-aware. This is partly due to the fact that in fields, too much information is missing to do good neighborhood operations (blur, vector blur etc.). The solution is to render your animation at double the frame rate without fields and do the interlacing of the footage afterwards.

## Home-made Render Farm



An easy way to get multiple machines to share the rendering workload is to:

- Set up a shared directory (such as a Windows Share or an NFS mount)
- Un-check “Overwrite” and check “Placeholders”
- Start as many machines as you wish rendering to that directory – they will not step on each other’s toes.

## 2.8.8 Post Processed Effects

There are several effects you can enable in the Render Settings that add visual elements to rendered images, after the rendering has completed. These are not done in camera, but rather composited on top of the image.

*Composited* and *Sequence* are discussed in [Output Options](#).

*Fields* are discussed in [Video Output](#).

## Render Layers

---

### Reference

Editor: *Properties*

Context: *Render Layers*

---

Render layers allow you to render your scene in separate layers, usually with the intension of compositing them back together afterwards.

This can be useful for several purposes, such as color correcting certain elements differently, blurring the foreground as a fast manual method of creating DoF, or reducing the render quality for unimportant objects.

Using Render Layers can also save you from having to re-render your entire image each time you change something, allowing you to instead re-render only the layer(s) that you need.

### Layer List

This is a list of all the Render Layers in the current scene.

Only layers which are enabled (checkbox on right is ticked) will be rendered. If the *Pin* icon at the bottom right of the list is enabled, only the active (highlighted) layer will be rendered.

Render Layers can be added and removed using the + and – buttons on the right, and existing layers can be renamed by double clicking on their name.

### Layer Panel

The Layer Panel shows the settings of the active Render Layer from the list above.

You can select multiple layers using `Shift-LMB`.

**Scene** The Scene Layers, showing which are currently visible and will be rendered.

**Layer** The Scene Layers which are associated with the active Render Layer. Objects in those Scene Layers will be rendered in that Render Layer. When an object is in the Scene Layers but not the Render Layer, it will still cast shadows and be visible in reflections, so it is still indirectly visible.

**Mask Layer** Objects on these will mask out other objects appearing behind them.

**Material Override** Overrides all material settings to use the Material chosen here.

Examples of where this might be used:

- To check lighting by using a plain diffuse material on all objects
- Render a wireframe of the scene
- Create a custom render pass such as an anti-aliased matte or global coordinates.

**See also:**

Additional options shown in this panel are different for each render engine. See these options for:

- [Blender Render](#)
- [Cycles](#)

### Using Render Layers

Each Render Layer has an associated set of [Scene Layers](#). Objects which are on one of the associated Scene Layers are shown in that Render Layer, as long as that Scene Layer is also visible.

**Warning:** Only the objects in visible Scene Layers will be rendered. So, if only Scene Layer 1 is visible and your Render Layer set specifies to render only Layers 2 and 3, nothing will be rendered.

### Edge (Toon) Rendering

Blender's toon shaders can give your rendering a comic-book-like or manga-like appearance, affecting the shades of colors. The effect is not perfect since real comics and manga also usually have china ink outlines. Blender can add this feature as a post-processing operation.

#### Options

**Edge** This makes Blender search for edges in your rendering and add an 'outline' to them.

Before repeating the rendering it is necessary to set some parameters:

**Threshold** The threshold of the angle between faces for drawing edges, from 0 to 255. A value of 10 would just give outline of object against the background, whereas higher settings start to add outlines on surface edges, starting with sharper angles. At maximum intensity, Edge will even faintly display geometry subsurf edge lines in areas of imperfect smoothing.

**Color / R/G/B** The color of the rendered edges (black by default). Click on the swatch to see the color picker

#### Examples

It is possible to separate out the edge layer using a render layer dedicated to that purpose. The alpha channel is 0 where there is no edge, and 1 where the edge is. By separating out the edge layer, you can blur it, change its color, mask it, etc. The image to the right shows how to do this. I created an Edge render layer that only has the Sky and Edge layers (I included sky so that we get the world color later on in the composite output). The other render layer omits the Edge layer, so it returns just the normal image. On the output panel I enabled Edge with a width of 10 in black. I run that layer through a blur node. Using the Alphaover node, I then composite the cube on top of the blurred edge. The result gives a soft-shadow kind of effect. Note that Premultiply is set because the Edge image already has an alpha of 1.0 set.



Fig. 2.2355: A scene with Toon materials.

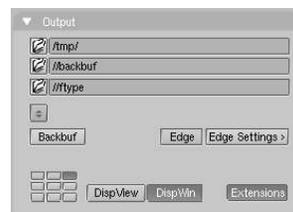


Fig. 2.2356: Toon edge buttons.

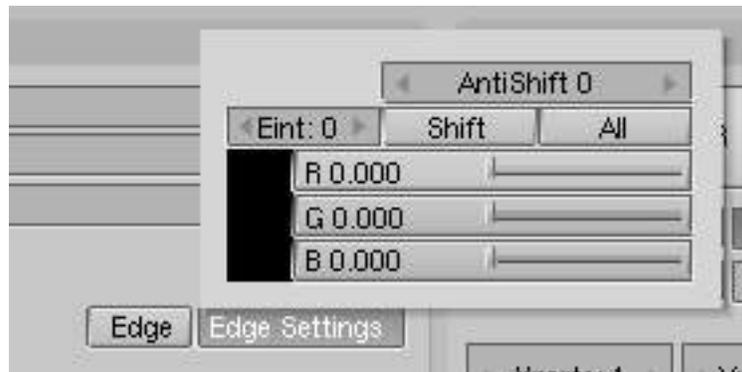


Fig. 2.2357: Toon edge settings.



Fig. 2.2358: Scene re-rendered with toon edge set.

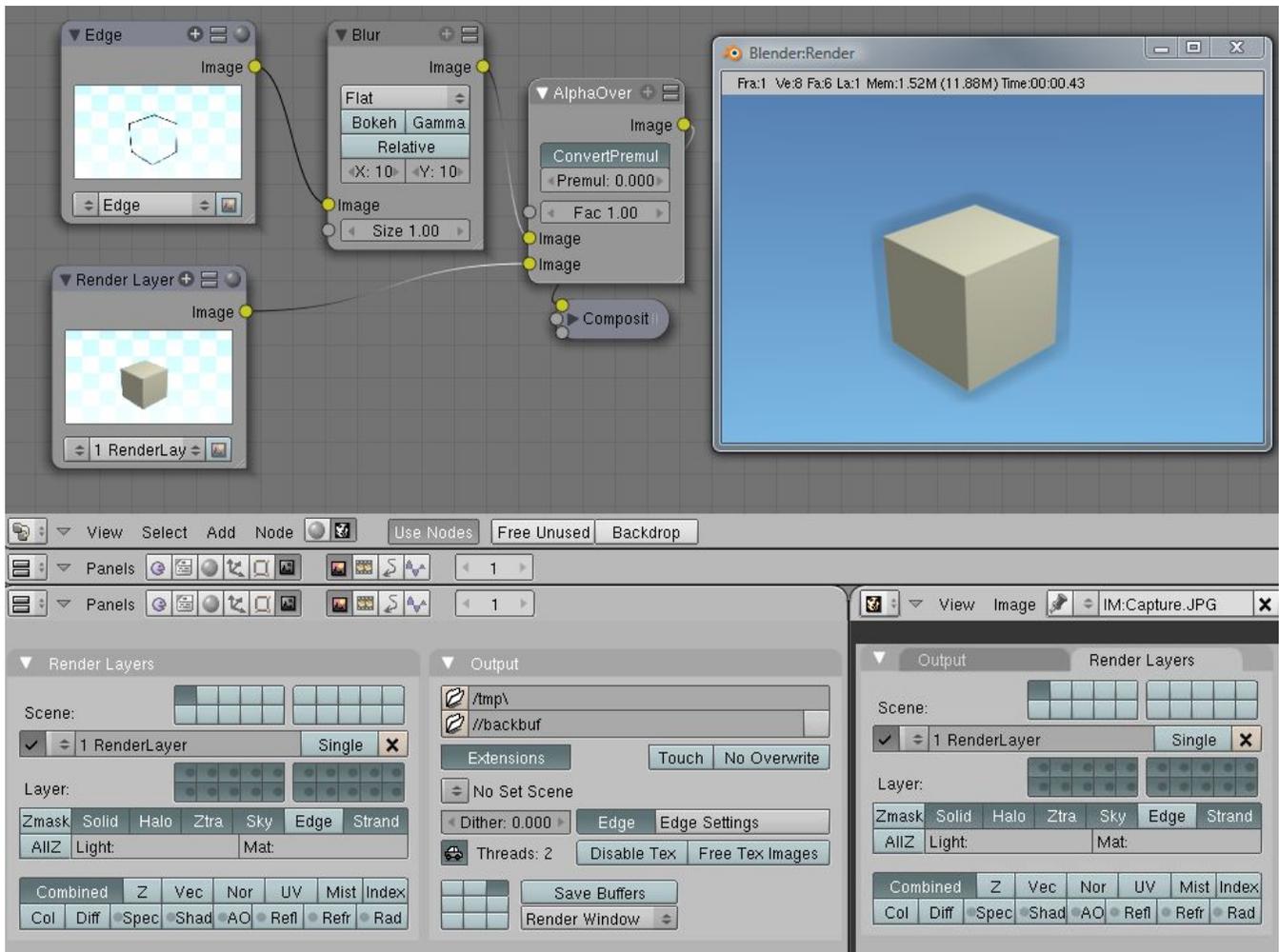


Fig. 2.2359: Post-processing Edge and Renderlayers

**Dithering** Dithering is a technique for blurring pixels to prevent banding that is seen in areas of gradients, where stair-stepping appears between colors. Banding artifacts are more noticeable when gradients are longer, or less steep. Dithering was developed for graphics with low bit depths, meaning they had a limited range of possible colors.

Dithering works by taking pixel values and comparing them with a threshold and neighboring pixels then does calculations to generate the appropriate color. Dithering creates the perceived effect of a larger color palette by creating a sort of visual color mixing. For example, if you take a grid and distribute red and yellow pixels evenly across it, the image would appear to be orange.

The *Dither* value ranges from 0 to 2.

## Metadata

The *Metadata* panel includes options for writing meta-data into render output.

Stamping can include the following data:

**Time** Include the current scene time and render frame as `HH:MM:SS.FF`

**Date** Include the current date and time.

**RenderTime** Include the render time in the stamp image.

**Frame** Include the frame number.

**Scene** Include the name of the active scene.

**Camera** Include the name of the active camera.

**Lens** Include the name of the active camera's lens value.

**Filename** Include the filename of the .blend file.

**Marker** Include the name of the last marker.

**Seq. Strip** Include the name of the foreground sequence strip.

**Note** Include a custom note.

---

**Note:** Only some image formats support metadata: See [image formats](#).

---

**Stamp Output** You can optionally stamp this into the image its self (adding text over the rendered image) which can be useful for test renders and animation previews.

**Stamp Text Color** Set the color and alpha of the stamp text.

**Stamp Background** Set the color and alpha of the color behind the text.

**Font Size** Set the size of the text.

---

**Hint:** It can be useful to use the *Note* field if you're setting up a render-farm.

Since you can script any information you like into it, such as an identifier for the render-node or the job-number.

For details on stamping arbitrary values, see: [this page](#)

---

## Color Management

[OpenColorIO](#) is integrated into Blender, meaning many color spaces are supported with fine control over which color transformations are used.



Fig. 2.2360: Different views and exposures of the same render

## Scene Linear Color Space

For correct results, different color spaces are needed for rendering, display and storage of images. Rendering and compositing is best done in **scene linear color space**, which corresponds more closely to nature, and makes computations more physically accurate.

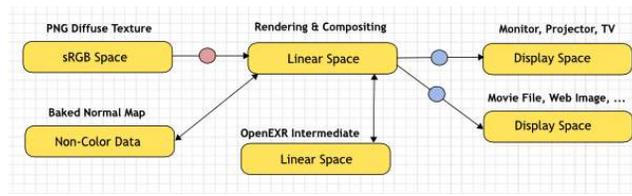


Fig. 2.2361: Example linear workflow

If the colors are linear, it means that if in reality we double the amount of photons, the color values are also doubled. Put another way, if we have two photos/renderers each with one of two lights on, and add those images together, the result would be the same as a render/photo with both lights on. It follows that such a radiometrically linear space is best for photorealistic rendering and compositing.

However these values do not directly correspond to human perception or the way display devices work, and image files are often stored in different color spaces, so we have to take care to do the right conversion into and out of this linear color space.

## Settings

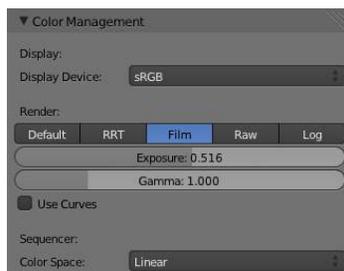


Fig. 2.2362: Scene settings for color management

These settings are found in the scene context of the properties editor, under the *Color Management* panel.

**Display** Correct display of renders requires a **conversion to the display device color space**, which can be configured here. A computer monitor works differently from a digital cinema project or HDTV. The scene properties have these settings:

**Display Device** The device that the image is being viewed on.

Most computer monitors are configured for the sRGB color space, and so when working on a computer usually this option should just be left to the default. It would typically be changed when viewing the image on another display device connected to the computer, or when writing out image files intended to be displayed on another device.

Rec709 is commonly used for HDTVs, while XYZ and DCI-P3 are common for digital projectors.

Color management can be disabled by setting the device to None.

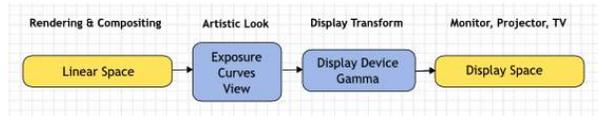


Fig. 2.2363: Conversion from linear to display device space

**Render** There is also an **artistic choice** to be made for renders. Partially that's because display devices can't display the full spectrum of colors and only have limited brightness, so we can squeeze the colors to fit in the gamut of the device. Besides that it can also be useful to give the renders a particular look, e.g. as if they have been printed on real film.

Another common use case is when you want to inspect renders, to see details in dark shadows or bright highlights, or identify render errors. Such settings would be only used temporarily and not get used for final renders.

**View** These are different ways to view the image on the same display device.

**Default** Does no extra conversion besides the conversion for the display device.

**RRT** Uses the ACES Reference Rendering Transform, to simulate a film-like look.

**Film** This option is another film-like look.

**Raw and Log** Intended for inspecting the image but not for final export. Raw gives the image without any color space conversion, while Log gives a more "flat" view of the image without very dark or light areas.

**Exposure** Used to control the image brightness (in stops) applied before color space conversion.

**Gamma** Extra gamma correction applied after color space conversion. Note that the default sRGB or Rec709 color space conversions already include a gamma correction of approximately 2.2 (except the Raw and Log views), so this would be applied in addition to that.

**Look** Choose an artistic effect from set of measured film response data which roughly emulates the look of certain film types. Applied before color space conversion.

**Use Curves** Adjust RGB Curves to control image colors before color space conversion. Read more about using the [Curve Widget](#).

## Sequencer

**Color Space** The color space that the sequencer operates in. By default the sequencer operates in sRGB space, but it can also be set to work in Linear space like the Compositing nodes, or another color space. Different color spaces will give different results for color correction, cross fades, and other operations.

## Image Files

The other place to keep color management in mind is when **loading and saving image files**. File formats such as PNG or JPEG will typically store colors in a color space ready for display, not in a linear space. When they are, for example, used as textures in renders, they need to be converted to linear first, and when saving renders for display on the web, they also need to be converted to a display space. Other file formats like OpenEXR store linear color spaces and as such are useful as intermediate files in production.

When working with image files, the default color space is usually the right one. If this is not the case, the color space of the image file can be configured in the image settings. A common situation where manual changes are needed is when working with or baking normal maps or displacement maps, for example. Such maps do not actually store colors, just data encoded as colors. In such cases they should be marked as **Non-Color Data**.

Image datablocks will always store float buffers in memory in the scene linear color space, while a byte buffer in memory and files on disk are stored in the color space specified with this setting:

**Color Space** The color space of the image on disk. This depends on the file format, for example PNG or JPEG images are often stored in sRGB, while OpenEXR images are stored in a linear color space. Some images such as normal, bump or stencil maps do not strictly contain ‘colors’, and on such values no color space conversion should ever be applied. For such images the color space should be set to None.

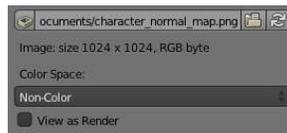


Fig. 2.2364: Image settings for color management

By default only renders are displayed and saved with the render view transformations applied. These are the Render Result and Viewer image datablocks, and the files saved directly to disk with the Render Animation operator. However when loading a render saved to an intermediate OpenEXR file, Blender can't detect automatically that this is a render (it could be e.g. an image texture or displacement map). We need to specify that this is a render and that we want the transformations applied, with these two settings:

**View as Render** Display the image datablock (not only renders) with view transform, exposure, gamma, RGB curves applied. Useful for viewing rendered frames in linear OpenEXR files the same as when rendering them directly.

**Save as Render** Option in the image save operator to apply the view transform, exposure, gamma, RGB curves. This is useful for saving linear OpenEXR to e.g. PNG or JPEG files in display space.

## OpenColorIO Configuration

Blender comes with a standard OpenColorIO configuration that contains a number of useful display devices and view transforms. The reference linear color space used is the linear color space with Rec. 709 chromaticities and D65 white point.

However OpenColorIO was also designed to give a consistent user experience across [multiple applications](#), and for this a single shared configuration file can be used. Blender will use the standard OCIO environment variable to read an OpenColorIO configuration other than the default Blender one. More information about how to set up such a workflow can be found on the [OpenColorIO website](#).

We currently use the following color space roles:

**scene\_linear** color space used for rendering, compositing, and storing all float precision images in memory.

**default\_sequencer** default color space for sequencer, *scene\_linear* if not specified

**default\_byte** default color space for byte precision images and files, *texture\_paint* if not specified.

**default\_float** default color space for float precision images and files, *scene\_linear* if not specified.

The standard Blender configuration also includes some support for ACES ([code and documentation](#)), even though we have a different linear color space. It's possible to load and save EXR files with the Linear ACES color space, and the RRT view transform can be used to view images with their standard display transform. However the ACES gamut is larger than the Rec. 709 gamut, so for best results an ACES specific configuration file should be used. OpenColorIO provides an [ACES configuration](#), though it may need a few more tweaks to be usable in production.

## Compatibility

Compatibility with existing files should mostly be preserved. Files that had color management enabled should be entirely compatible, while older files with the color management option disabled are mostly compatible but different for vertex colors and viewport colors.

## See Also

- [Developer Documentation](#)
- [User:Sobotka/Color\\_Management](#)

## 2.8.9 Freestyle

### Introduction

#### What is FreeStyle?

Freestyle is an edge- and line-based non-photorealistic (NPR) rendering engine. It relies on mesh data and z-depth information to draw lines on selected edge types. Various line styles can be added to produce artistic (“hand drawn”, “painted”, etc.) or technical (hard line) looks.

The two operating modes - [Python Scripting](#) and [Parameter Editor](#) - allow a powerful diversity of line styles and results. Line styles such as Japanese big brush, cartoon, blueprint, thickness-with-depth are already pre-scripted in Python. The Parameter Editor mode allows intuitive editing of features such as dotted lines and easy setup of multiple line types and edge definitions. On top of all of that, with the introduction of line style modifiers, the sky is the limit!

More artwork can be found at [http://wiki.blender.org/index.php/Dev:Ref/Release\\_Notes/2.67/FreeStyle#Freestyle\\_Artwork\\_Showcase](http://wiki.blender.org/index.php/Dev:Ref/Release_Notes/2.67/FreeStyle#Freestyle_Artwork_Showcase)

### The Big Picture

- Activate FreeStyle by *Properties* window → *Render* tab → *FreeStyle* panel, tick check box. Please note that FreeStyle is only available for the Blender Internal renderer.
- Freestyle settings are located in the new *Render Layers* context.
- One render layer can only have one viewmap. A viewmap holds the edge detection settings (Crease Angle, Culling toggle, Face Smoothness toggle, Material Boundaries toggle, Sphere Radius and Kr Derivative Epsilon advanced options).
- A viewmap can have multiple line sets.
- A line set controls which line types and selections will be rendered, from lines based on your scene.
- Each line set uses one line style (which can be shared between multiple line sets).
- A line style tells Freestyle how to render the linked line sets in terms of color, alpha, thickness and other aspects.



Fig. 2.2365: A cartoon scene from OHA Studio (the .blend file). © Mechanimotion Entertainment.

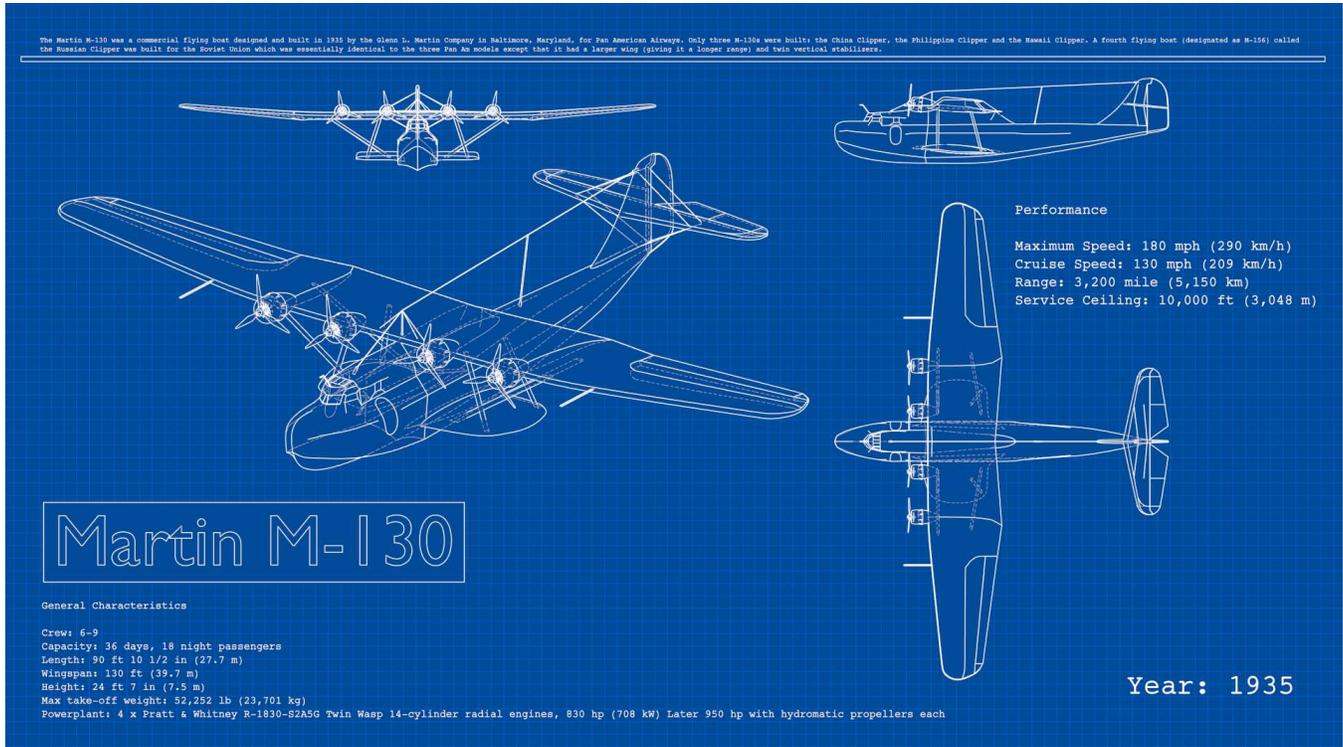


Fig. 2.2366: Blueprint render of Martin M-130 from 1935 by LightBWK. CC0. WARNING: HEAVY FILE! DESIGNED FOR STRESS TEST BLENDER TO THE LIMITS & MAY CRASH BLENDER. (File:M-130Blueprint.zip)

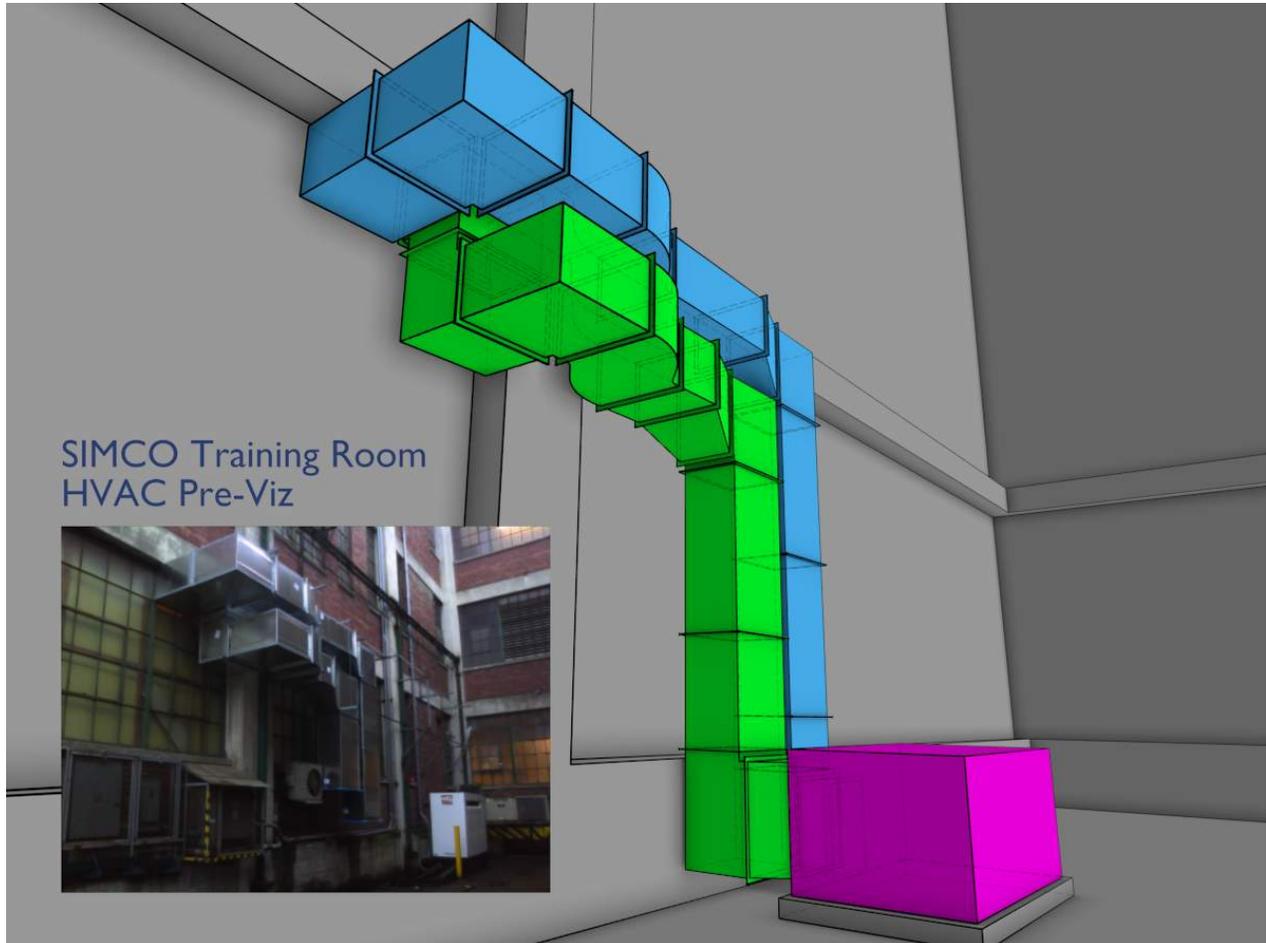


Fig. 2.2367: HVAC Pre-Viz by Lee Posey. CC0 (File:HVACPreViz.zip)



Fig. 2.2368: Kitchen by Vicente Carro. © AnigoAnimation

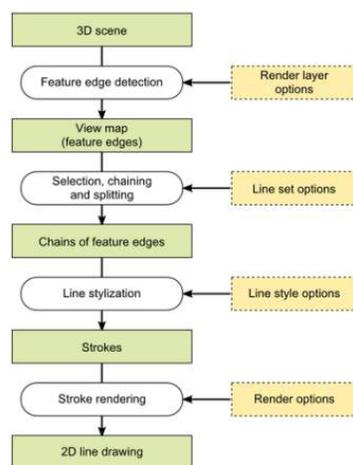


Fig. 2.2369: block diagram of Freestyle view map and processes

## Known Limitations

- Highly memory demanding: All mesh objects in a render layer are loaded at once.
- Only faced mesh objects are supported. The following kinds of meshes are ignored:
  - Mesh faces with wire materials.
  - Mesh faces with completely transparent materials.
  - Mesh faces with the *Cast Only* material option enabled.
- Transparent faces are treated as opaque faces.
- No edges at face intersections are detected yet.
- Layer masks do not work with Freestyle.
- Freestyle rendering results do not have any Z depth information.
- Panoramic cameras are not supported.

## Core Options



Fig. 2.2370: Freestyle core options.

Activating Freestyle in the *Render* context of the *Buttons* window will give you the following options:

**Line Thickness** There are two different modes for defining the base line thickness:

**Absolute** The line thickness is given by a user-specified number of pixels. The default value is **1.0**.

**Relative** The unit line thickness is scaled by the proportion of the present vertical image resolution to **480** pixels. For instance, the unit line thickness is **1.0** with the image height set to **480**, **1.5** with **720**, and **2.0** with **960**.

**Line Thickness** Only for *Absolute* line thickness: base line thickness in pixels, **1.0** by default.

## Viewmaps

There is only one viewmap per render layer. It controls the edge detection parameters. Which detected edges are actually rendered, and how, can be controlled either through the user-friendly [parameter editor](#), or powerful but complex [Python scripting](#).

**Face Smoothness** When enabled, Face Smoothness will be taken into account for edges calculation.

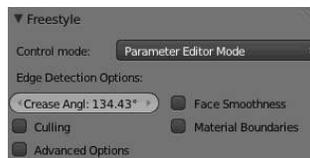


Fig. 2.2371: Parameter Editor Mode UI

**Crease Angle** If two adjacent faces form an angle less than the defined *Crease Angle*, the edge between them will be rendered when using *Crease* edge type selection in a line set. The value also affects *Silhouette* edge type selection.

**Culling** Ignore the edges that are out of view (saves some processing time and memory, but may reduce the quality of the result in some cases).

## Advanced Options

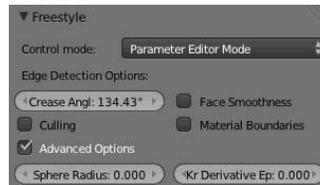


Fig. 2.2372: Advanced Options

*Sphere Radius* It affects the calculation of curvatures for *Ridge*, *Valley* and *Suggestive Contour* edge type selection in a line set.

**Kr Derivative Epsilon** It provides you with control over the output of *Suggestive Contour* and *Silhouette* edge type selection (further information in [this pdf](#)).

## Parameter Editor

### Introduction to Parameter Editor

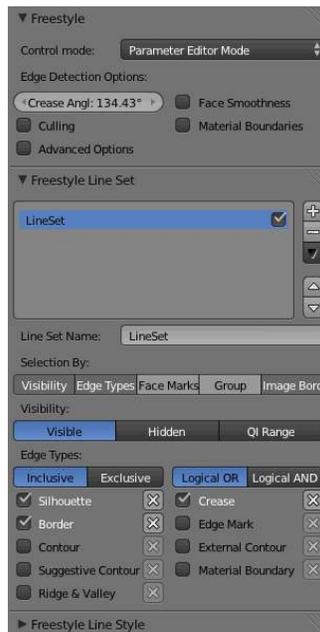


Fig. 2.2373: Parameter Editor

The Freestyle *Parameter Editor* is a user-friendly interface to define and control line sets and line styles.

**Line Sets** control which of the edges detected by Freestyle will actually be used (rendered).

**Line Styles** control how the selected edges are rendered.

A view map (hence a render layer) can have multiple line sets, and each line set is linked to one line style.

## Line Set

A line set selects, among the lines (edges) detected by Freestyle, which ones will be rendered using its attached [line style](#), through various methods.

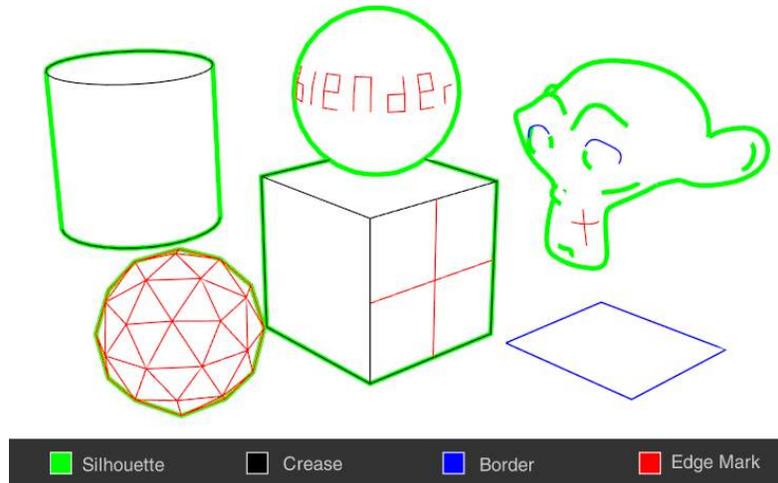


Fig. 2.2374: Examples of some basic edge types by LightBWK (File:EdgeType.zip)

**Selection by Visibility** There are three choices for selecting edges by visibility.

**Visible** Only lines occluded by no surfaces are rendered.

**Hidden** Lines occluded by at least one surface are rendered.

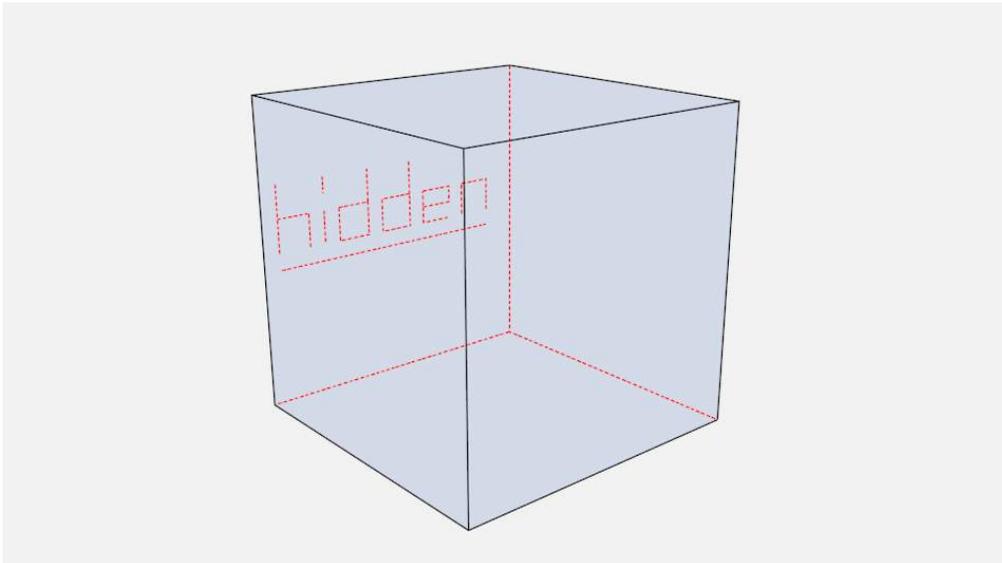


Fig. 2.2375: Proof of concept of visible and hidden edges by LightBWK (Sample .blend)

**QI Range** QI stands for *Quantitative Invisibility*. Lines occluded by a number of surfaces in the given range are rendered.

**Start and End** Only with *QI Range*, min/max number of occluding surfaces for a line to be rendered.

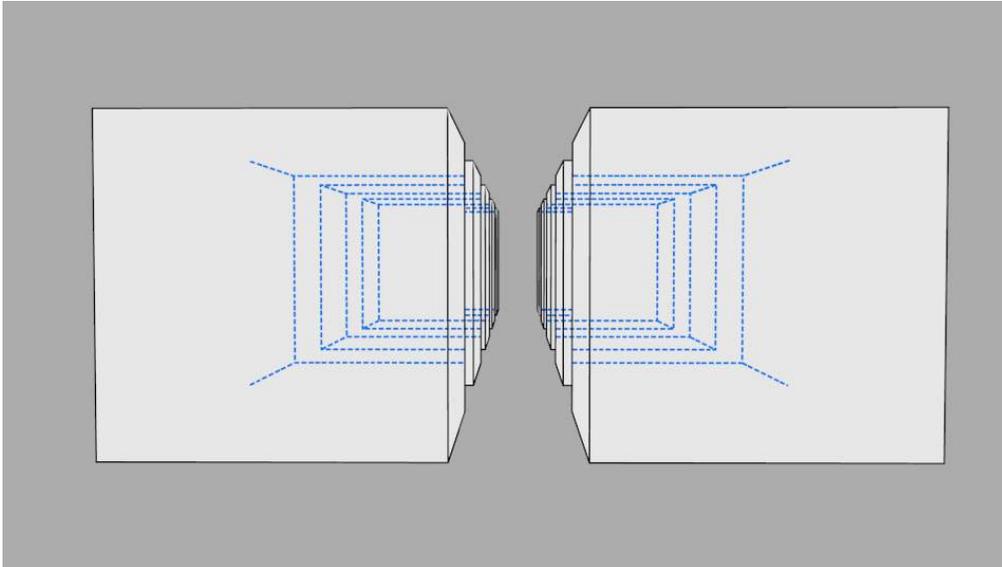


Fig. 2.2376: QI Range proof of concept demo, Start: 3, End: 7, by LightBWK (Sample .blend)

**Selection by Edge Types** Edge types are basic algorithms for the selection of lines from geometry. When using the parameter editor you have to choose at least one edge type in order to get a render output, but several edge types can be combined in one line set. Edge types can also be excluded from calculation by pressing the X next to them.

**Silhouette** Draws silhouettes around your closed objects; it is often good for organic objects (like Suzanne & Sphere), and bad for sharp edges, like a box. It can't render open mesh objects like open cylinders and flat planes. The output is affected by the *Kr Derivative Epsilon* viewmap setting.

**Crease** Shows only edges whose adjacent faces form an angle greater than the defined viewmap's *Crease Angle*.

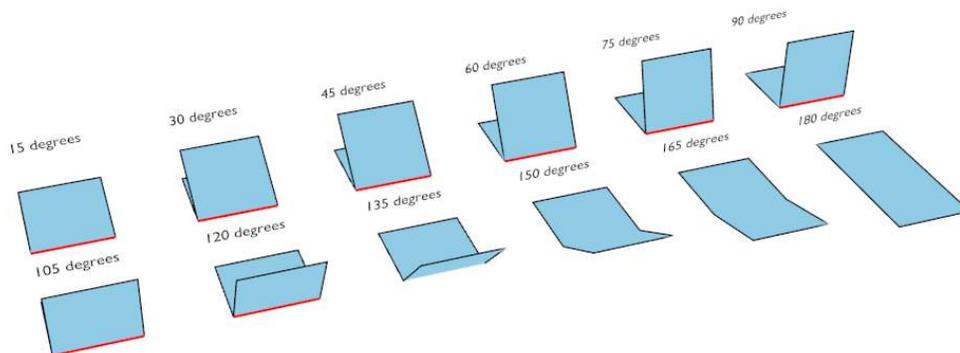


Fig. 2.2377: Crease Angle proof of concept for 121° by LightBWK ( the .blend file)

**Border** Border is for open/unclosed edge meshes; an open cylinder has an open edge at the top and bottom, and a plane is open all around. Suzanne's eye socket is an open edge. All open edges will have lines rendered. This depends on the mesh structure.

**Edge Marks** Renders marked edges. See [Edge Marks](#) for details.

**Contour** Draws the outer edges and inner open border.

**External Contour** Draws the contour lines, but only on the outer edges.

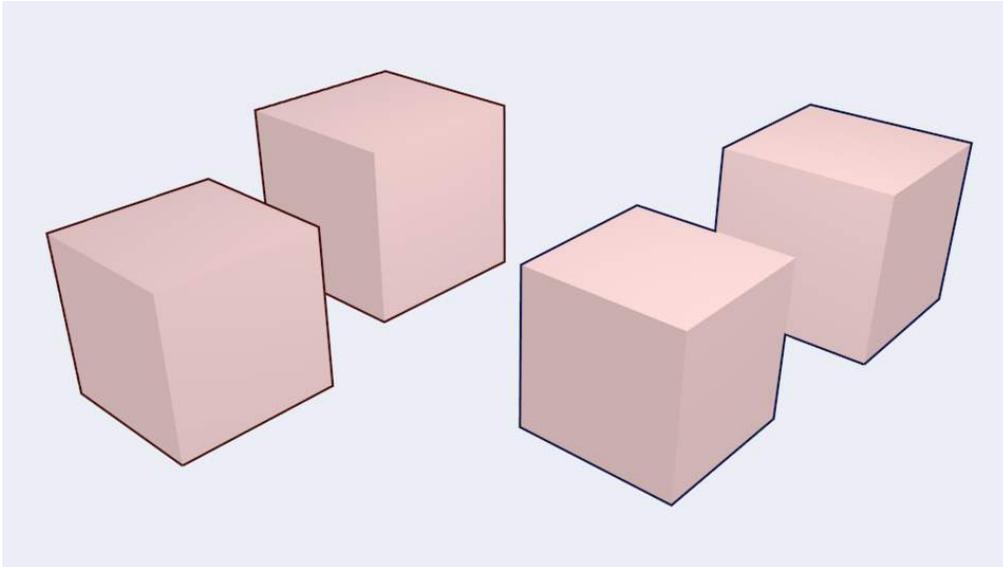


Fig. 2.2378: Left pair: Contour; Right pair: External Contour

**Suggestive Contour** Draws some lines which would form the contour of the mesh if the viewport was shifted. Depends on your viewmap settings for *Kr Derivative Epsilon* and *Sphere Radius* (further information: [File:Manual-2.6-Render-Freestyle-PrincetownLinestyle.pdf](#)).

**Material Boundary** Draws lines where two materials meet on the same object. Must be activated in the viewmap settings.

**Ridge & Valley** Draws ridges and valleys. Depends on your *Sphere Radius* viewmap settings.

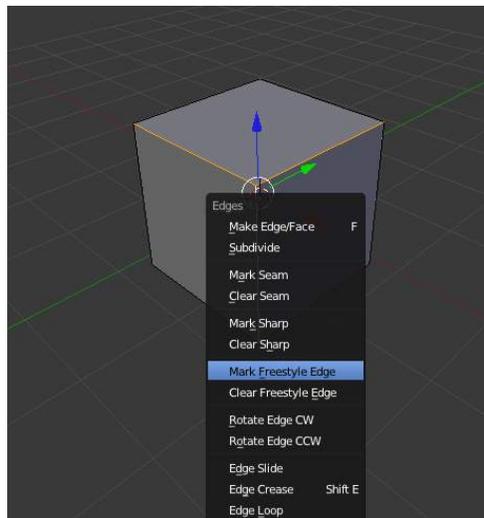


Fig. 2.2379: Select and mark Freestyle edges.



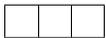
Fig. 2.2380: Edge Mark setting in the Line Sets tab.

**Edge Marks** In edit mode you can mark “Freestyle Edges” in the same manner you can mark “Seams” for UV unwrapping or “Sharp” for edge split. These marked edges are available to render when you select *Edge Mark*.

This is done as follows:

- Select your mesh and tab into *Edit* mode.
- Select the edges you want to be marked.
- Press `Ctrl-E` and select *Mark Freestyle Edge*.

Edge marks are useful when you want to draw lines along particular mesh edges. The examples below explain the use of edge marks.



The image on the left shows a sphere in *Edit* mode. The green lines are the edge marks. On the right you see a render without edge marks enabled.

With edge marks enabled, the previously-marked lines are always rendered. You can see the black contour lines and the blue lines that are made with edge marks.

What are edge marks good for?

- When you need to render marks on an almost-flat plane, when other edge types can’t detect any line.
- When you want full control of edge rendering. Often used for edges of squarish shapes.
- Mark the whole base mesh to be rendered for base mesh preview.

What are edge marks not good for?

- Round outer edges (use instead *Contour* / *External Contour* / *Silhouette*).

**Selection by Face Marks** To set a face mark:

- Select a mesh and tab into *Edit* mode.
- Select the faces you want to be marked.
- Press `Ctrl-F` and select *Mark Freestyle Face*.

Face marks are useful for removing lines from certain areas of a mesh.

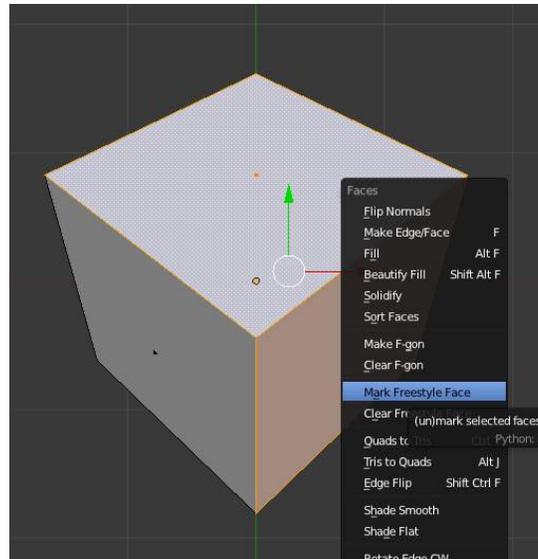


Fig. 2.2387: Mark Freestyle Faces.

In this example, two faces of the default cube are marked like the image on the left. On the right is a render without face marks activated.

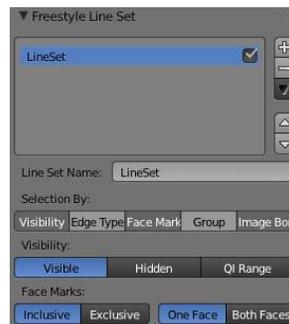


Fig. 2.2392: Face mark options.

The line selection can be controlled via inclusion and faces options:

**Inclusive / Exclusive** Whether to include or exclude edges matching defined face mark conditions from the line set.

**One Face** (De)select all edges which have one or both neighbor faces marked.

**Both Faces** (De)select all edges which have both of their neighbor faces marked.

The image below shows the resulting combinations.



**Selection by Group** You can include or exclude objects for line calculation, based on their belonging to a group.

**Group** The name of the object group to use.

**Inclusive / Exclusive** Whether to include or exclude lines from those objects in this line set.

**Selection by Image Border** If enabled, Freestyle only takes geometry within the image border into consideration for line calculation. This reduces render times but increases continuity problems when geometry is moved out of and into camera view.

## Line Style & Modifiers



Fig. 2.2401: Line Style UI

In Freestyle, the line style settings define the appearance of a line set using five main aspects:

- stroke
- color
- alpha
- thickness
- geometry

These allow you to get many different styles of renders (technical draw, rough sketch, cartoon, oriental calligraphy, etc.).

You can create as many line styles as you wish, and reuse a given line style for several line sets by selecting it from the dropdown menu next to its name.

---

### Note: Length Unit

Unless otherwise specified, all lengths in line style settings are in pixels (either relative or absolute, as specified in the [core options](#)).

---



Fig. 2.2402: Line Style demo [File:LineStyle.zip](#)

## Stroke

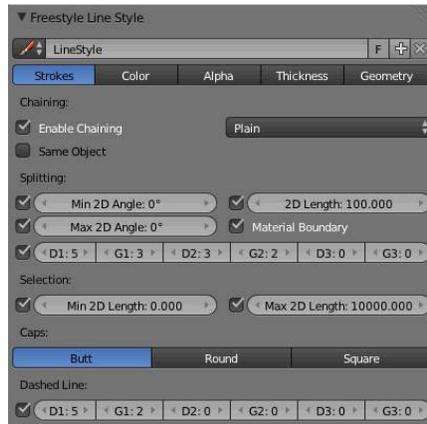


Fig. 2.2403: Stroke Line Style

Strokes are the final rendered lines. Yet you can tweak them, for example, by removing the ones longer/shorter than some threshold, chaining lines into a single stroke or breaking a stroke into several ones based on angles, dashed pattern, etc.

**Chaining** By default all retrieved lines from the line set are chained together. There are two basic chaining methods:

**Plain** The default chaining method; it creates simple chains.

**Sketchy** This chaining option allows for generating chains of feature edges with sketchy multiple strokes. Basically, it generates *Round* strokes instead of a single one. It is only really useful if you use some random-driven modifiers in the line style!

**Rounds** It specifies the number of rounds in sketchy strokes.

Chaining can also be turned off to render each line separately, which can be useful for line styles which depend on accurate representation of the line set.

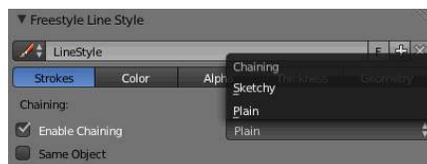


Fig. 2.2404: Chaining

**Splitting** You can split up chains of Freestyle lines by checking one of the following:

**Material Boundary** Splits chains of feature edges if they cross from one material to another.

**Min 2D Angle and Max 2D Angle** Splits chains of feature edges when they make a 2D angle above (or below) a minimum (or maximum) threshold.

**2D Length** Splits chains when they are longer than the given value.

**D1 / G1 / D2 / G2 / D3 / G3** Splits the chains using the given dashed pattern (“D” stands for “dash”, “G” stands for “gap”; see also *Dashed Line*).



Fig. 2.2405: Splitting



Fig. 2.2406: Selection

**Selection** You can also choose to only select (i.e. render) chains longer than *Min 2D Length* and/or shorter than *Max 2D Length*.

**Caps** You can choose between three types of line caps:

**Butt** Flat cap, exactly at the point the line ends.



Fig. 2.2407: Line tip caps

**Round** A half circle centered on the end point of the line.

**Square** A square centered on the end point of the line (hence, like the circle, the drawn end of the line is slightly extended compared to its computed value).

**Dashed Line** By enabling the *Dashed Line* check box, you can specify three pairs of dash and gap lengths. Dash values define the lengths of dash strokes, while gap values specify intervals between two dashes.

If a zero gap is specified, then the corresponding dash is ignored even if it has a non-zero value.

Dashes are treated as separate strokes, meaning that you can apply line caps, as well as color, alpha and thickness modifiers.

## Color

In this tab you control the color of your strokes.

**Base Color** The base color for this line style.

**Modifiers** There are four color modifiers available, which can be mixed with the base color using the usual methods (see for example the [Mix compositing node](#) for further discussion of this topic). As with other modifier stacks in Blender, they are applied from top to bottom.

**Influence** How much the result of this modifier affects the current color.

**Along Stroke** The *Along Stroke* modifier alters the base color with a new one from a given color ramp mapped along each stroke's length. In other words, it applies a color ramp along each stroke.

**Color Ramp** A standard Blender color ramp.

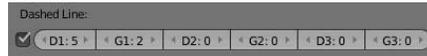


Fig. 2.2408: Dashes Line UI

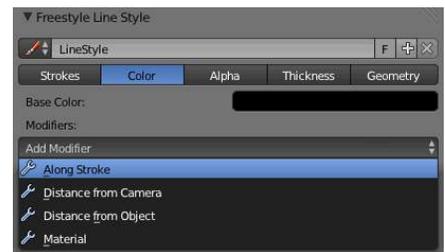
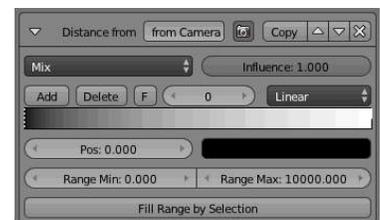


Fig. 2.2409: Line Style Color UI

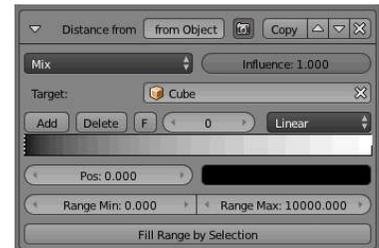


**Distance from Camera** The *Distance from Camera* color modifier alters the base color with a new one from a given color ramp, using the distance to the active camera as the parameter.

**Range Min and Range Max** The limits of the mapping from “distance to camera” to “color in ramp”. If the current point of the stroke is at *Range Min* or less from the active camera, it will take the start color of the ramp, and conversely, if it is at *Range Max* or more from the camera, it will take the end color of the ramp. These values are in the current scene’s units, not in pixels!

**Fill Range by Selection** Set the min/max range values from the distances between the current selected objects and the camera. The other settings are those of the standard Blender color ramp!

**Distance from Object** The *Distance from Object* color modifier alters the base color with a new one from a given color ramp, using the distance to a given object as the parameter.



**Target** The object to measure distance from.

**Range Min and Range Max** The limits of the mapping from “distance to object” to “color in ramp”. If the current point of the stroke is at *Range Min* or less from the target, it will take the start color of the ramp, and conversely, if it is at *Range Max* or more from the target, it will take the end color of the ramp. These values are in the current scene’s units, not in pixels!

**Fill Range by Selection** Set the min/max range values from the distances between the current selected objects and the target. The other settings are those of the standard Blender color ramp!

**Material** The *Material* color modifier alters the base color with a new one taken from the current material under the stroke.



You can use various properties of the materials, among which many are mono-component (i.e. give B&W results). In this case, an optional color ramp can be used to map these grayscale values to colored ones.

If used with the *Split by Material* option in the *Stroke* tab, the result will not be blurred between materials along the strokes.

**Noise** The *Noise* modifier uses a pseudo-random number generator to variably distribute color along the stroke.

**Amplitude** The maximum value of the noise. A higher amplitude means a less transparent (more solid) stroke.

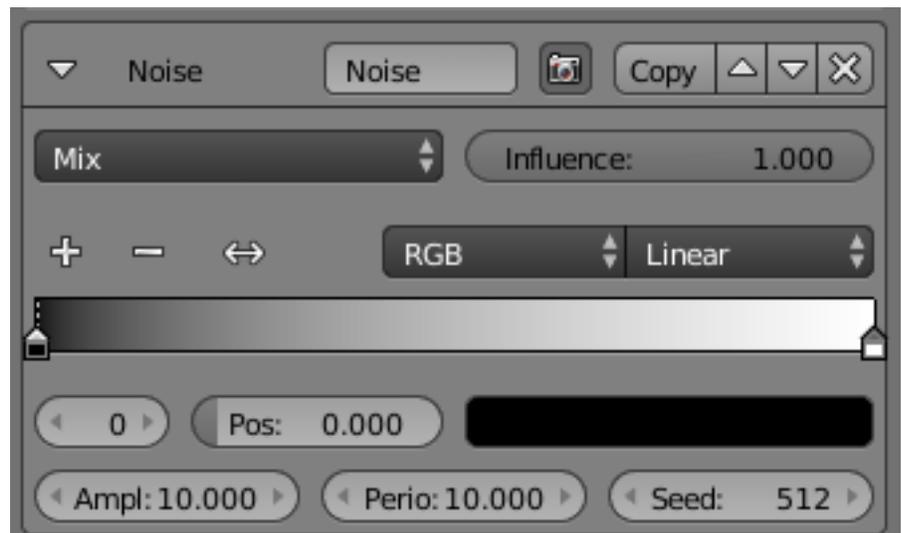
**Period** The period of the noise. This means how quickly the color value can change. A higher value means a more smoothly changing color along the stroke.

**Seed** Seed used by the pseudo-random number generator.

**Color Ramp** A standard Blender color ramp that maps noise values to a stroke color.



Fig. 2.2410: Material modifiers demo by T.K. File:Lilies\_Color\_Material.zip



**Tangent** This modifier bases its effect on the traveling direction of the stroke evaluated at the stroke's vertices.



**Color Ramp** A standard Blender color ramp that maps the traveling direction to a stroke color.

**Min Angle and Max Angle** The range of input values to the mapping. Out-of-range input values will be clamped by the Min and Max angles and their corresponding color values.

**3D Curvature** A modifier based on radial curvatures of the underlying 3D surface. The [curvature](#) of a 2D curve at a point is a measure of how quickly the curve turns at the point. The quicker the turn is, the larger the curvature is at the point. The curvature is zero if the curve is a straight line. Radial curvatures are those computed for a 2D curve that appears at the cross-section between the 3D surface and a plane defined by the view point (camera location) and the normal direction of the surface at the point.

For radial curvatures to be calculated (and therefore for this modifier to have any effect), the *Face Smoothness* option has to be turned on and the object needs to have *Smooth Shading*.

**Color Ramp** A standard Blender color ramp that maps the radial curvature to a stroke color.

**Min Curvature and Max Curvature** The limits of the color ramp. If the current point of the stroke is at *Min Curvature* or less from the target, it will take the start color of the mapping, and conversely, if it is at *Max Curvature* or more from the target, it will take the end color of the mapping.

**Crease Angle** A modifier based on the Crease Angle (angle between two adjacent faces). If a stroke segment doesn't lie on a crease (i.e., the edge doesn't have the [Crease Angle nature](#)), its color values are not touched by the modifier.

**Color Ramp** A standard Blender color ramp that maps the crease angle to a stroke color.

**Min Angle and Max Angle** The range of input values to the mapping. Out-of-range crease angle values will be clamped by the Min and Max angles and their corresponding color values.

## Alpha

In this tab you control the alpha (transparency) of your strokes.

**Base Transparency** The base alpha for this line style.

**Modifiers** There are four alpha modifiers available, which can be mixed with the base alpha using a subset of the usual methods (see for example the [Mix compositing node](#) for further discussion of this topic). As with other modifier stacks in Blender, they are applied from top to bottom.

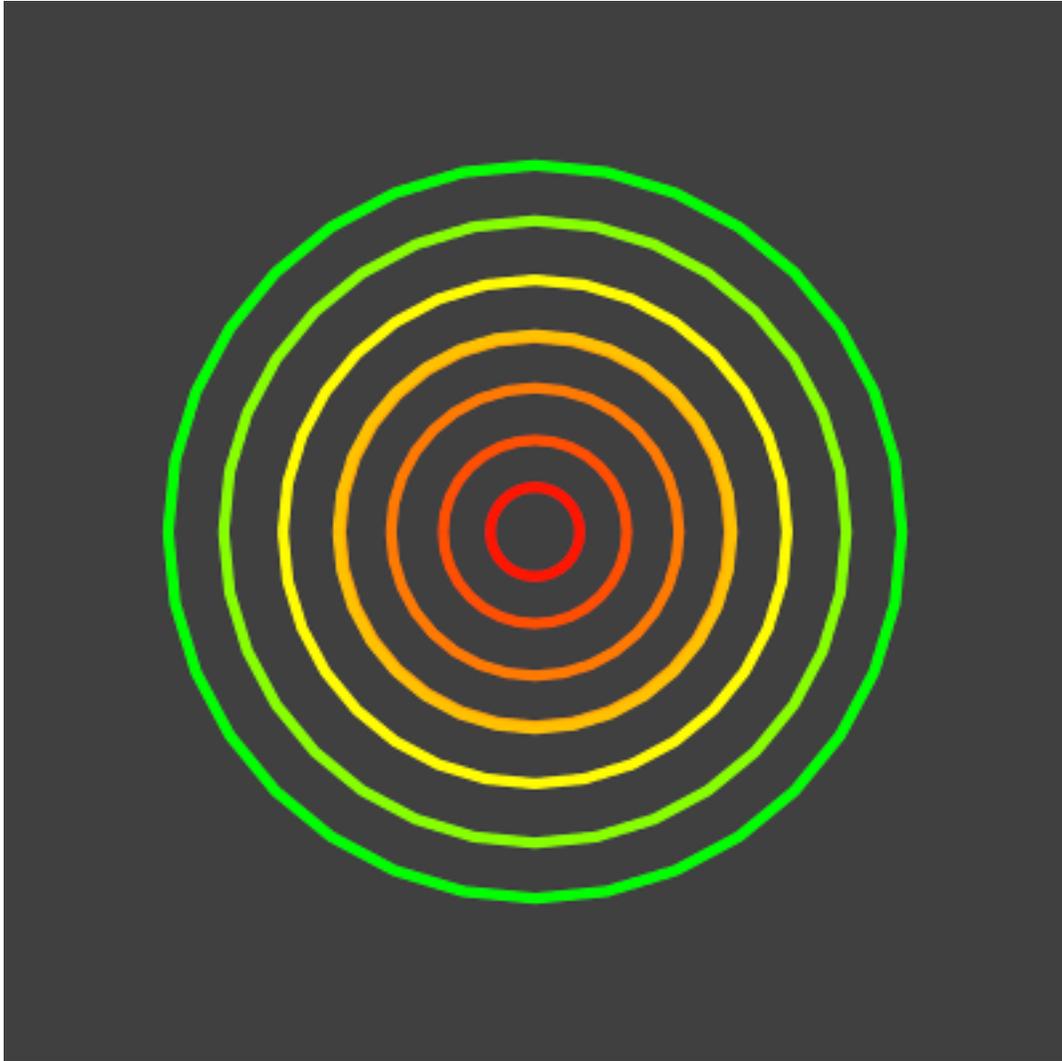
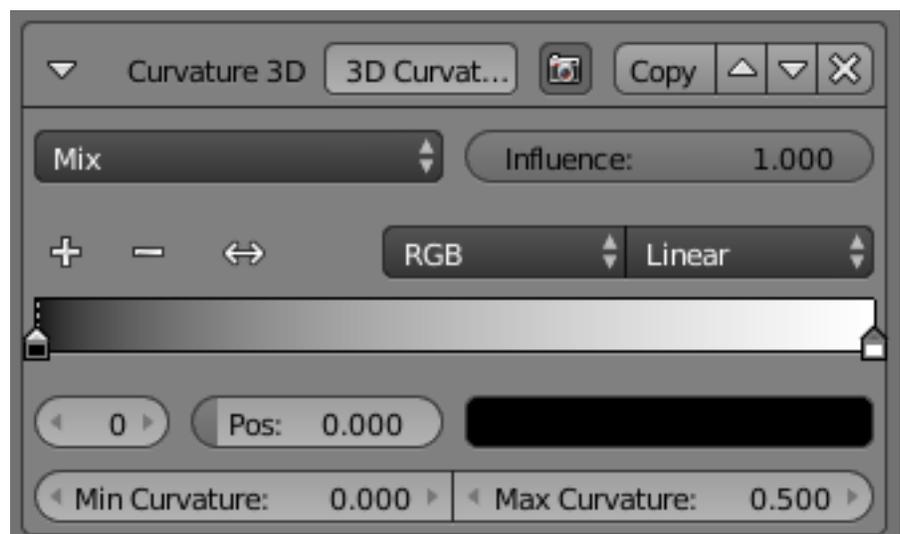


Fig. 2.2411: 3D Curvature modifier demo by T.K. File:Render\_freestyle\_modifier\_curvature\_3d.blend



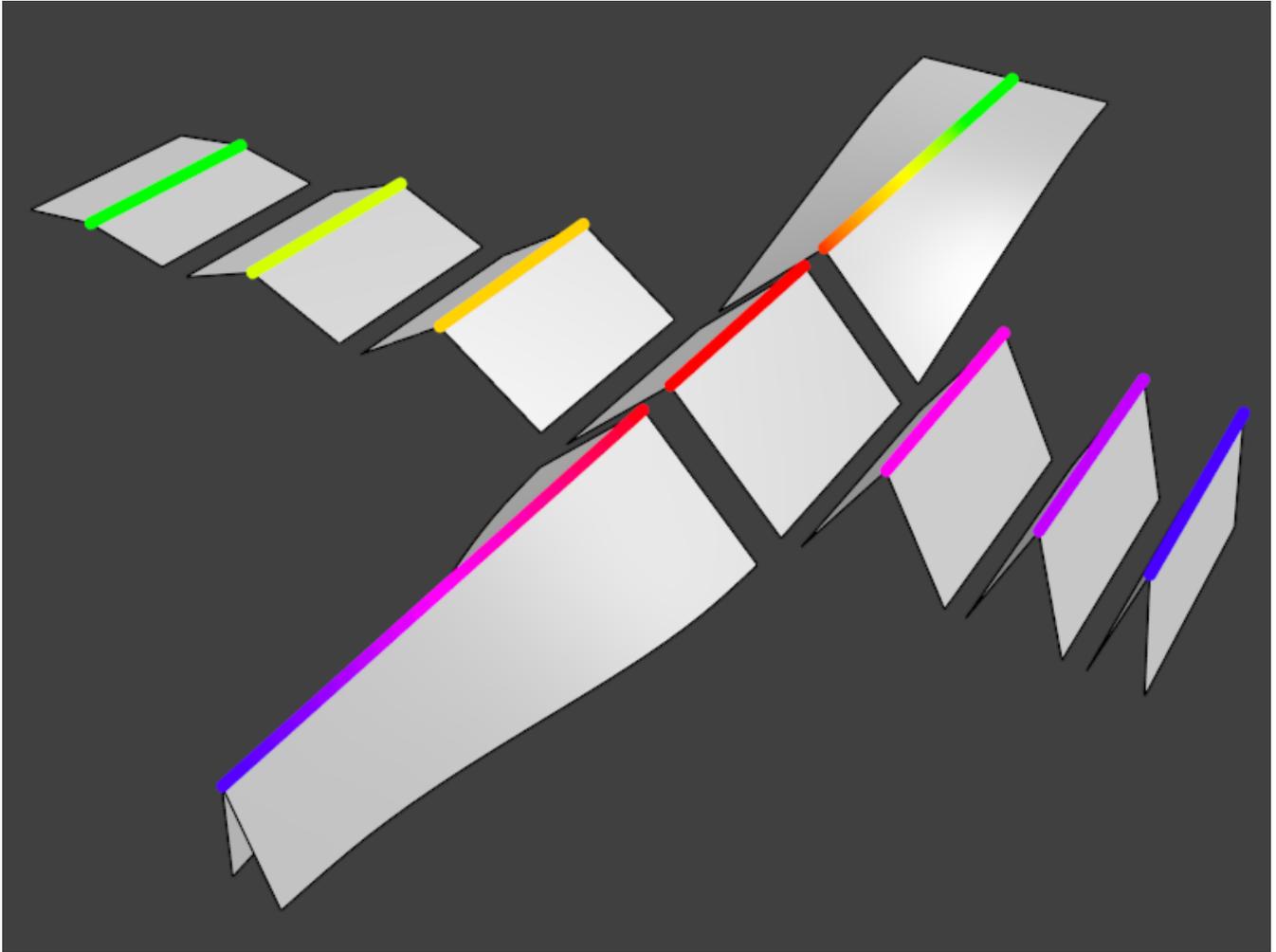
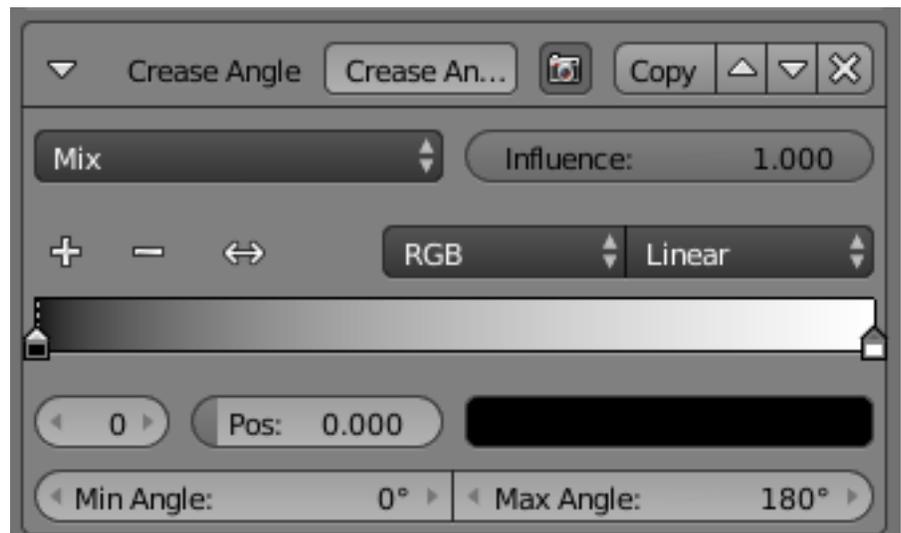


Fig. 2.2412: Crease Angle modifier demo by T.K. File:Render\_freestyle\_modifier\_crease\_angle.blend



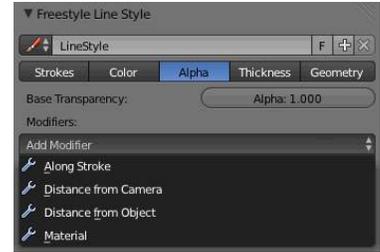
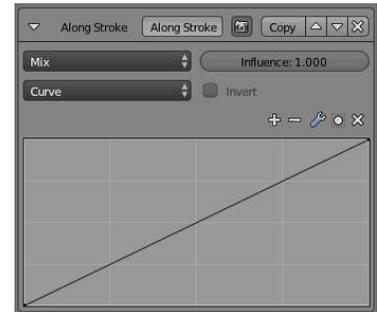


Fig. 2.2413: Line Style Alpha UI

**Influence** How much the result of this modifier affects the current transparency.



**Along Stroke** The *Along Stroke* modifier alters the base alpha with a new one from either a linear progression or a custom curve, mapped along each stroke’s length. In other words, it applies the selected progression along each stroke.

**Mapping** Either a linear progression (from 0.0 to 1.0, which may be inverted with the *Invert* option), or a custom mapping curve.

**Distance from Camera** The *Distance from Camera* modifier alters the base alpha with a new one from either a linear progression or a custom curve, using the distance to the active camera as parameter.



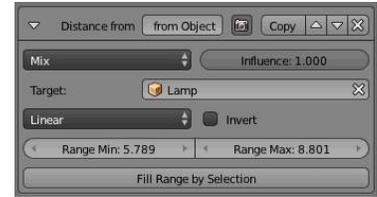
**Mapping** Either a linear progression (from 0.0 to 1.0, which may be inverted with the *Invert* option), or a custom mapping curve.

**Range Min and Range Max** The limits of the mapping from “distance to camera” to “alpha in mapping”. If the current point of the stroke is at *Range Min* or less from the active camera, it will take the start alpha of the mapping, and conversely, if it is at *Range Max* or more from the camera, it will take the end alpha of the mapping. These values are in the current scene’s units, not in pixels!

**Fill Range by Selection** Set the min/max range values from the distances between the current selected objects and the camera.

**Distance from Object** The *Distance from Object* modifier alters the base alpha with a new one from either a linear progression or a custom curve, using the distance to a given object as parameter.

**Target** The object to measure distance from.



**Mapping** Either a linear progression (from 0.0 to 1.0, which may be inverted with the *Invert* option), or a custom mapping curve.

**Range Min and Range Max** The limits of the mapping from “distance to object” to “alpha in mapping”. If the current point of the stroke is at *Range Min* or less from the target, it will take the start alpha of the mapping, and conversely, if it is at *Range Max* or more from the target, it will take the end alpha of the mapping. These values are in the current scene’s units, not in pixels!

**Fill Range by Selection** Set the min/max range values from the distances between the current selected objects and the target.

**Material** The *Material* modifier alters the base alpha with a new one taken from the current material under the stroke.

You can use various properties of the materials, among which some are multi-components (i.e. give RGB results). In that case, the mean value will be used.



**Mapping** Either a linear progression (from 0.0 to 1.0, which may be inverted with the *Invert* option), or a custom mapping curve. Note the linear non-inverted option is equivalent to “do nothing”, as original values from materials are already in the [0.0, 1.0] range.

If used with the *Split by Material* option in the *Stroke* tab, the result will not be blurred between materials along the strokes.

**Noise** The *Noise* modifier uses a pseudo-random number generator to variably distribute transparency along the stroke.

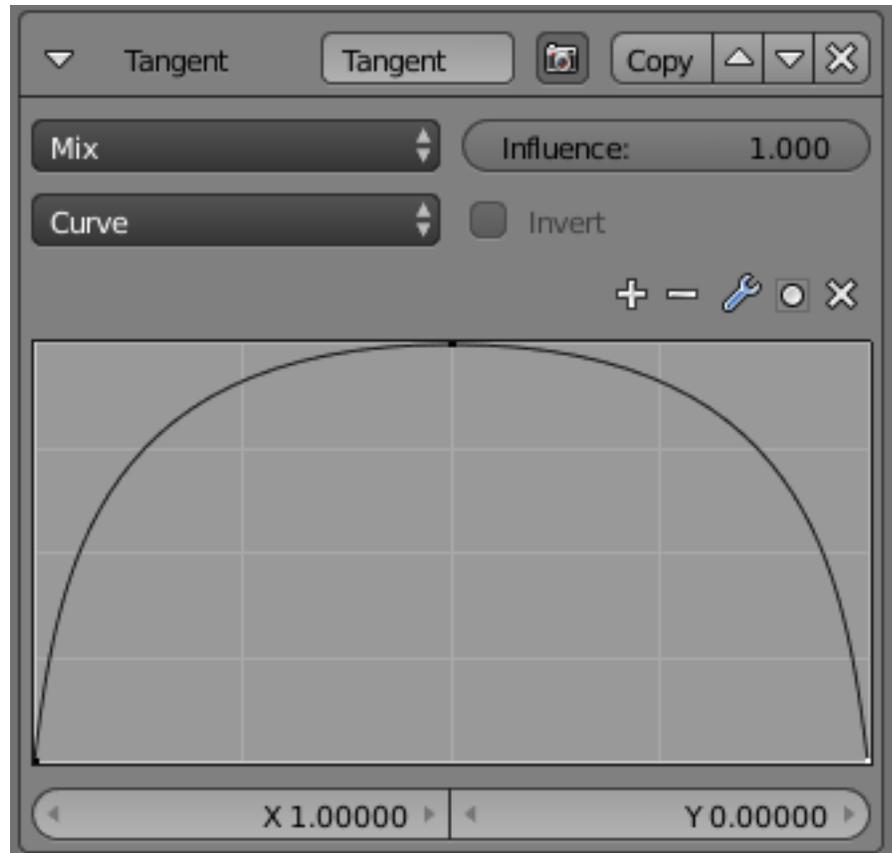


**Amplitude** The maximum value of the noise. A higher amplitude means a less transparent (more solid) stroke.

**Period** The period of the noise. This means how quickly the alpha value can change. A higher value means a more smoothly changing transparency along the stroke.

**Seed** Seed used by the pseudo-random number generator.

**Mapping** Either a linear progression (from 0.0 to 1.0, which may be inverted with the *Invert* option), or a custom mapping curve. Note the linear non-inverted option is equivalent to “do nothing”, as original values from materials are already in the [0.0, 1.0] range.



**Tangent** This modifier bases its effect on the traveling direction of the stroke evaluated at the stroke’s vertices.

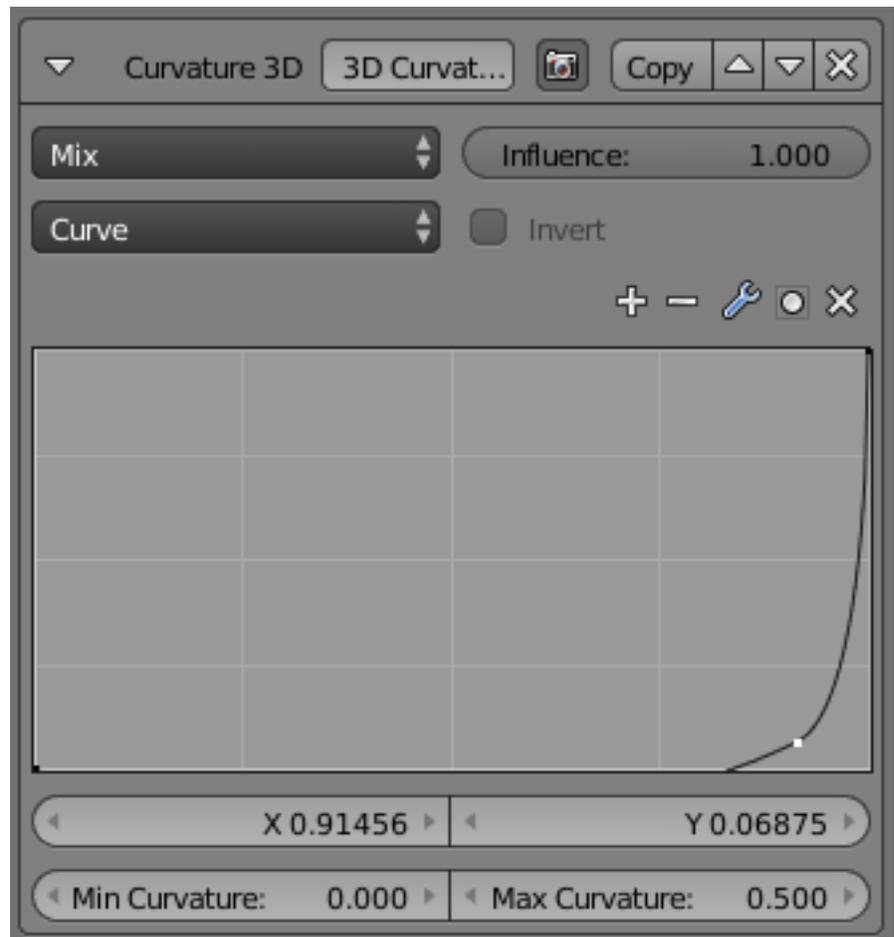
**Mapping** Either a linear progression (from 0.0 to 1.0, which may be inverted with the *Invert* option), or a custom mapping curve. Note the linear non-inverted option is equivalent to “do nothing”, as original values from materials are already in the [0.0, 1.0] range.

**Min Angle and Max Angle** The range of input values to the mapping. Out-of-range input values will be clamped by the Min and Max angles and their corresponding alpha values.

**3D Curvature** A modifier based on radial curvatures of the underlying 3D surface. The *curvature* of a 2D curve at a point is a measure of how quickly the curve turns at the point. The quicker the turn is, the larger the curvature is at the point. The curvature is zero if the curve is a straight line. Radial curvatures are those computed for a 2D curve that appears at the cross-section between the 3D surface and a plane defined by the view point (camera location) and the normal direction of the surface at the point.

For radial curvatures to be calculated (and therefore for this modifier to have any effect), the *Face Smoothness* option has to be turned on and the object needs to have *Smooth Shading*.

**Mapping** Either a linear progression (from 0.0 to 1.0, which may be inverted with the *Invert* option), or a custom mapping curve. Note the linear non-inverted option is equivalent to “do nothing”, as original values from materials are already in the [0.0, 1.0] range.



**Min Curvature and Max Curvature** The limits of the mapping. If the current point of the stroke is at *Min Curvature* or less from the target, it will take the start alpha of the mapping, and conversely, if it is at *Max Curvature* or more from the target, it will take the end alpha of the mapping.

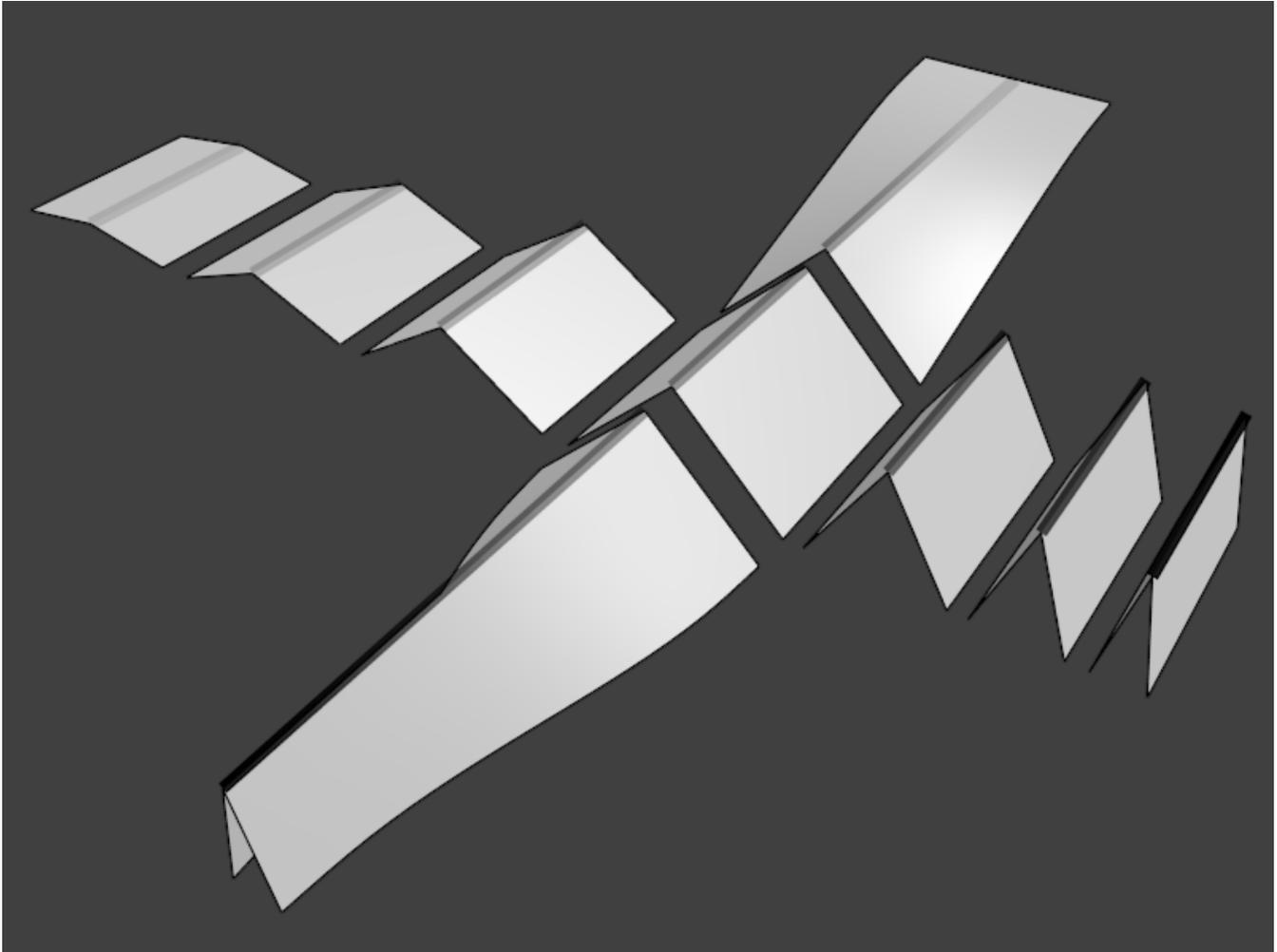


Fig. 2.2414: Crease Angle modifier demo by T.K. [File:Render\\_freestyle\\_modifier\\_crease\\_angle.blend](#)

**Crease Angle** A modifier based on the Crease Angle (angle between two adjacent faces). If a stroke segment doesn't lie on a crease (i.e., the edge doesn't have the [Crease Angle nature](#)), its alpha value is not touched by this modifier.

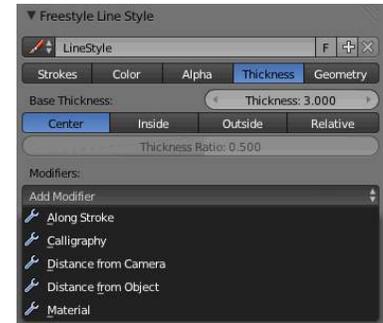
**Mapping** Either a linear progression (from 0.0 to 1.0, which may be inverted with the *Invert* option), or a custom mapping curve. Note the linear non-inverted option is equivalent to “do nothing”, as original values from materials are already in the [0.0, 1.0] range.

**Min Angle and Max Angle** The range of input values to the mapping. Out-of-range input values will be clamped by the Min and Max angles and their corresponding alpha values.

### Thickness

In this tab you control the thickness of your strokes.

**Base Thickness** The base thickness for this line style.



**Thickness Position** Control the position of stroke thickness from the original (backbone) stroke geometry. There are four choices:

**Center** The thickness is evenly split to the left and right side of the stroke geometry.

**Inside** The strokes are drawn within object boundary.

**Outside** The strokes are drawn outside the object boundary.

**Relative** This allows you to specify the relative position by a number between 0.0 (inside) and 1.0 (outside), in the *Thickness Ratio* numeric field just below.

The thickness position options are applied only to strokes of edge types *Silhouette* and *Border*, since these are the only edge types defined in terms of the object boundary. Strokes of other edge types are always drawn using the *Center* option.

**Modifiers** There are five thickness modifiers available, which can be mixed with the base thickness using a subset of the usual methods (see for example the [Mix compositing node](#) for further discussion of this topic). As with other modifier stacks in Blender, they are applied from top to bottom.

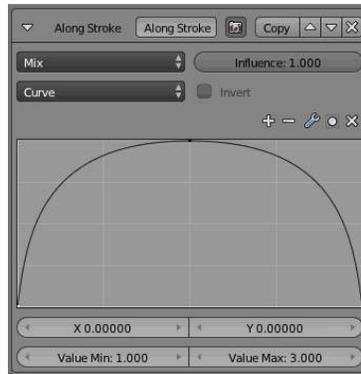
**Influence** How much the result of this modifier affects the current thickness.

**Along Stroke** The *Along Stroke* modifier alters the base thickness with a new one from either a linear progression or a custom curve, mapped along each stroke's length. In other words, it applies the selected progression along each stroke.

**Mapping** Either a linear progression (from 0.0 to 1.0 which may be inverted with the *Invert* option), or a custom mapping curve.

**Calligraphy** The *Calligraphy* modifier mimics some broad and flat pens for calligraphy. It generates different thickness based on the orientation of the stroke.

**Orientation** The angle (orientation) of the virtual drawing tool, from the vertical axis of the picture. For example, an angle of 0.0 mimics a pen aligned with the vertical axis, hence the thickest strokes will be the vertical ones, and the thinnest, the horizontal ones.



**Min Thickness and Max Thickness** The minimum and maximum generated thickness (as explained above, minimum is used when the stroke's direction is perpendicular to the main *Orientation*, and maximum, when aligned with it).

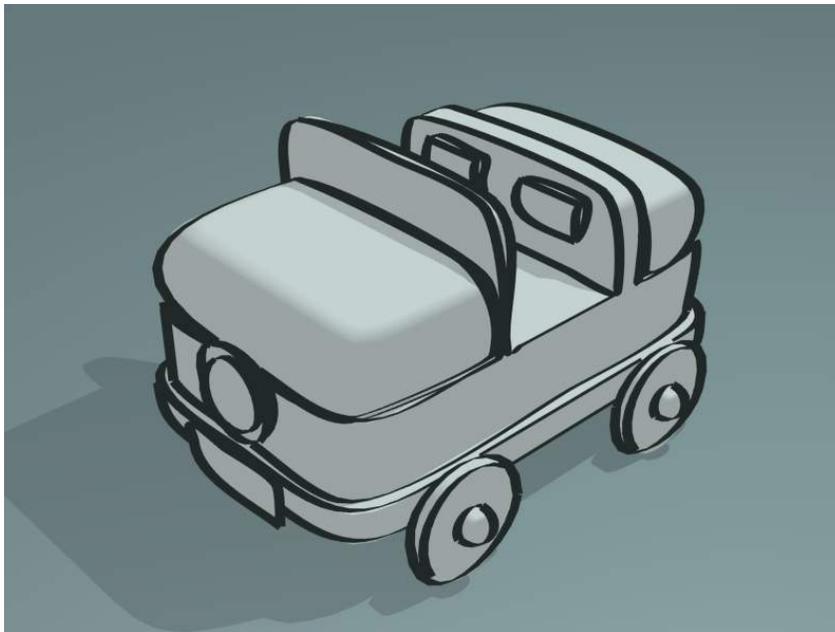
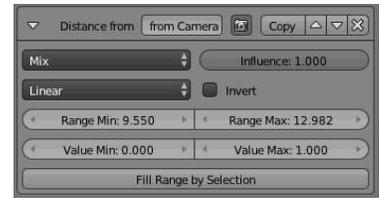


Fig. 2.2415: Calligraphy modifier demo by T.K. [File:Toycar\\_Calligraphy.zip](#)

**Distance from Camera** The *Distance from Camera* modifier alters the base thickness with a new one from either a linear progression or a custom curve, using the distance to the active camera as the parameter.

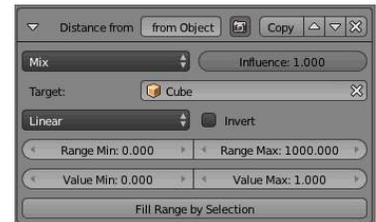
**Mapping** Either a linear progression (from 0.0 to 1.0 which may be inverted with the *Invert* option), or a custom mapping curve.

**Range Min and Range Max** The limits of the mapping from “distance to camera” to “thickness in mapping”. If the current point of the stroke is at *Range Min* or less from the active camera, it will take the start thickness of the mapping, and conversely, if it is at *Range Max* or more from the camera, it will take the end thickness of the mapping. These values are in the current scene's units, not in pixels!



**Fill Range by Selection** Set the min/max range values from the distances between the current selected objects and the camera.

**Distance from Object** The *Distance from Object* modifier alters the base thickness with a new one from either a linear progression or a custom curve, using the distance to a given object as parameter.



**Target** The object to measure distance from.

**Mapping** Either a linear progression (from 0.0 to 1.0 which may be inverted with the *Invert* option), or a custom mapping curve.

**Range Min and Range Max** The limits of the mapping from “distance to object” to “alpha in mapping”. If the current point of the stroke is at *Range Min* or less from the target, it will take the start thickness of the mapping, and conversely, if it is at *Range Max* or more from the target, it will take the end thickness of the mapping. These values are in the current scene’s units, not in pixels!

**Fill Range by Selection** Set the min/max range values from the distances between the current selected objects and the target.

**Material** The *Material* modifier alters the base thickness with a new one taken from the current material under the stroke.

You can use various properties of the materials, among which some are multi-components (i.e. give RGB results). In that case, the mean value will be used.



**Mapping** Either a linear progression (from 0.0 to 1.0 which may be inverted with the *Invert* option), or a custom mapping curve. Note the linear non-inverted option is equivalent to “do nothing”, as original values from materials are already in the [0.0, 1.0] range...

If used with the *Split by Material* option in the *Stroke* tab, the result will not be blurred between materials along the strokes.

**Noise** The *Noise* modifier uses a pseudo-random number generator to variably distribute thickness along the stroke.

**Min Thickness and Max Thickness** The minimum and maximum assigned thickness.

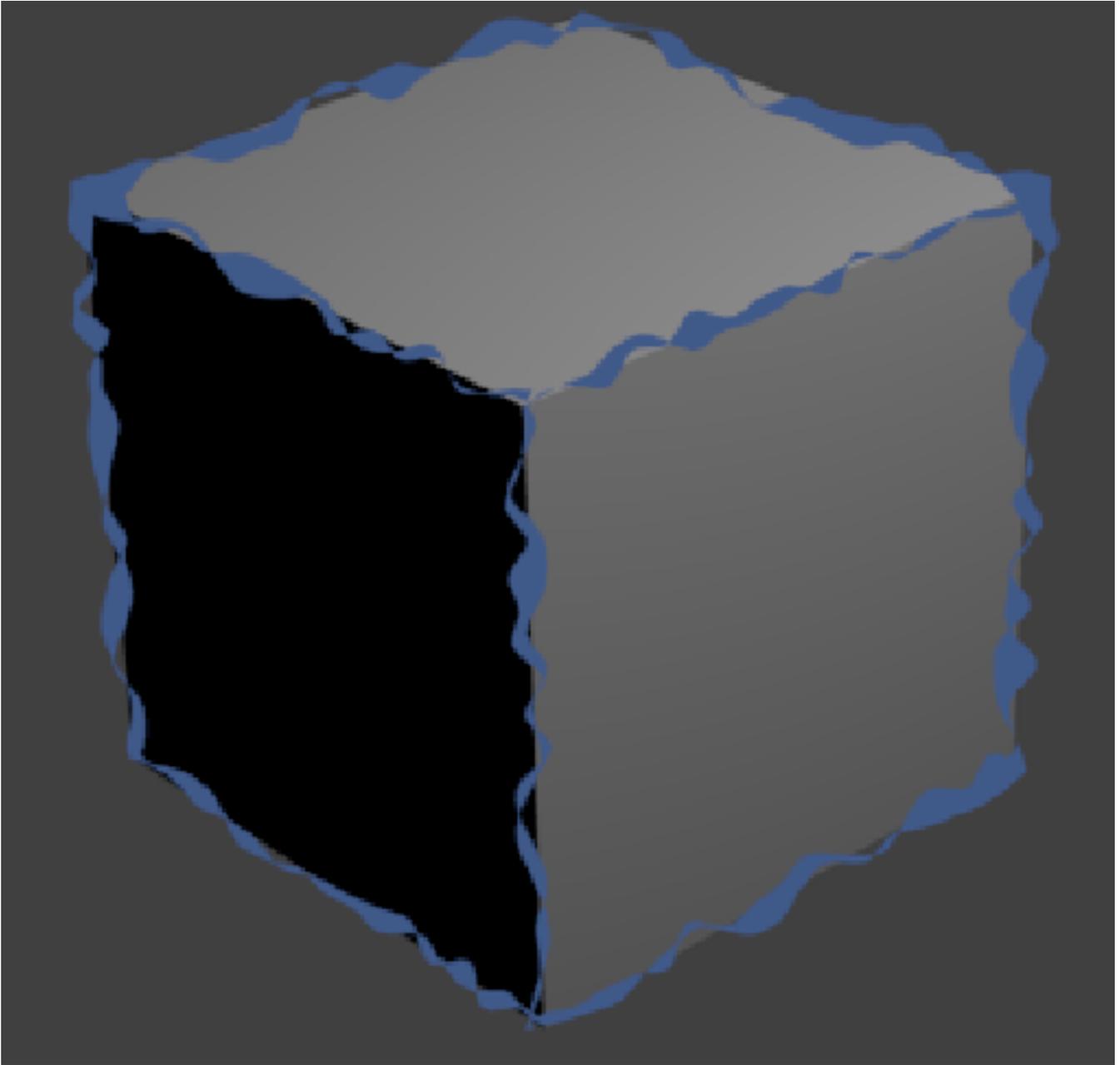
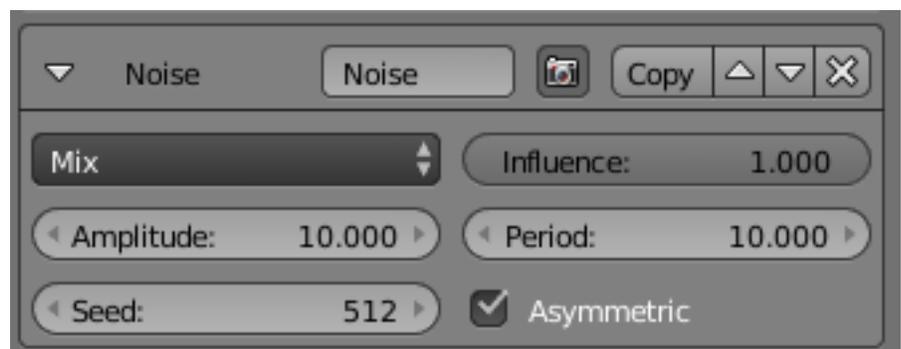
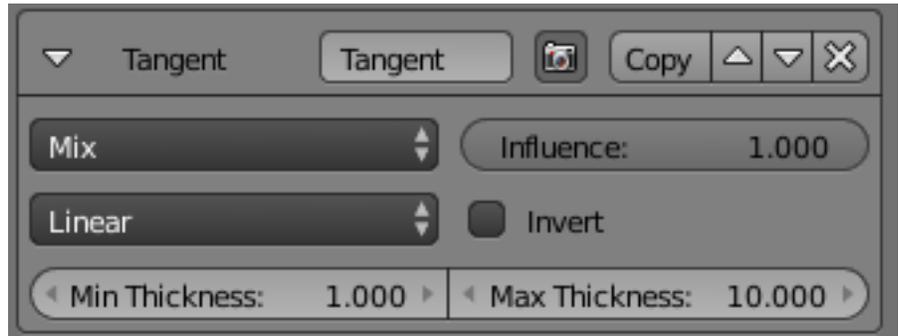


Fig. 2.2416: Effect generated with a noise thickness modifier using asymmetric thickness.



**Asymmetric** Allows the thickness to be distributed unevenly at every point. Internally, the stroke is represented as a backbone with a thickness to the right and left side. All other thickness shaders make sure that the left and right thickness values are equal. For the Noise shader however, a meaningful (and good-looking) result can be created by assigning different values to either side of the backbone.

**Tangent** This modifier bases its effect on the traveling direction of the stroke evaluated at the stroke's vertices.



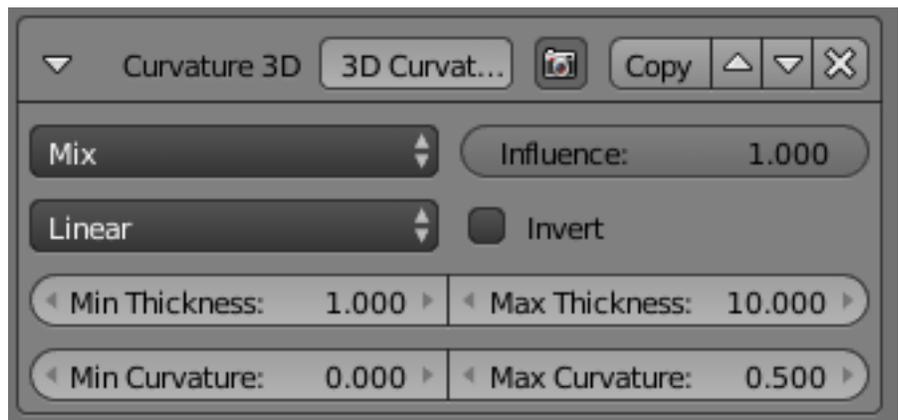
**Min Thickness and Max Thickness** The minimum and maximum assigned thickness.

**Mapping** Either a linear progression (from *Min Thickness* to *Max Thickness*, which may be inverted with the *Invert* option), or a custom mapping curve (on the same range).

**Min Angle and Max Angle** The range of input values to the mapping. Out-of-range input values will be clamped by the Min and Max angles and their corresponding thickness values.

**3D Curvature** A modifier based on radial curvatures of the underlying 3D surface. The **curvature** of a 2D curve at a point is a measure of how quickly the curve turns at the point. The quicker the turn is, the larger the curvature is at the point. The curvature is zero if the curve is a straight line. Radial curvatures are those computed for a 2D curve that appears at the cross-section between the 3D surface and a plane defined by the view point (camera location) and the normal direction of the surface at the point.

For radial curvatures to be calculated (and therefore for this modifier to have any effect), the *Face Smoothness* option has to be turned on and the object needs to have *Smooth Shading*.



**Min Thickness and Max Thickness** The minimum and maximum assigned thickness.

**Mapping** Either a linear progression (from *Min Thickness* to *Max Thickness*, which may be inverted with the *Invert* option), or a custom mapping curve (on the same range).

**Min Curvature and Max Curvature** The limits of the mapping of the Min and Max Thickness. If the current point of the stroke is at *Min Curvature* or less from the target, it will take the start thickness of the mapping, and conversely, if it is at *Max Curvature* or more from the target, it will take the end thickness of the mapping.

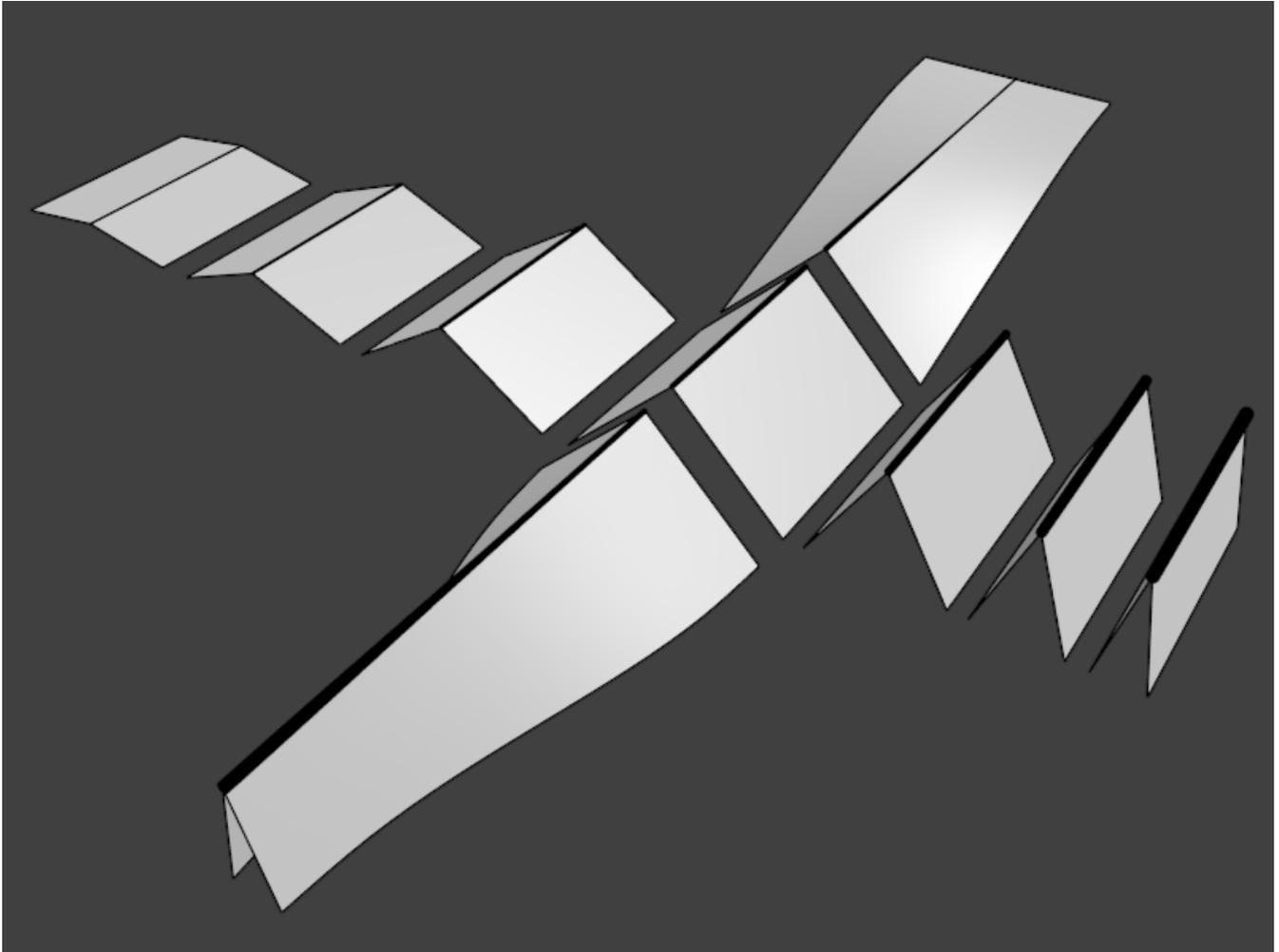


Fig. 2.2417: Crease Angle modifier demo by T.K. File:Render\_freestyle\_modifier\_crease\_angle.blend

**Crease Angle** A modifier based on the Crease Angle (angle between two adjacent faces). If a stroke segment doesn't lie on a crease (i.e., the edge doesn't have the [Crease Angle nature](#)), its thickness value is not touched by this modifier.

**Min Thickness and Max Thickness** The minimum and maximum assigned thickness.

**Mapping** Either a linear progression (from *Min Thickness* to *Max Thickness*, which may be inverted with the *Invert* option), or a custom mapping curve (on the same range).

### Geometry

In this tab you control the geometry of your strokes.

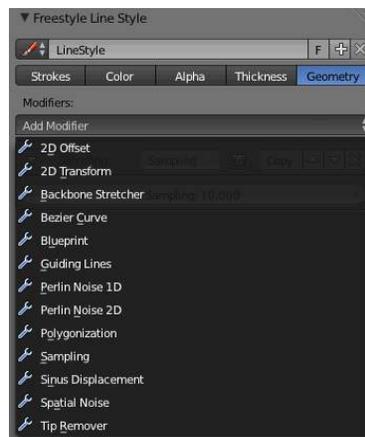
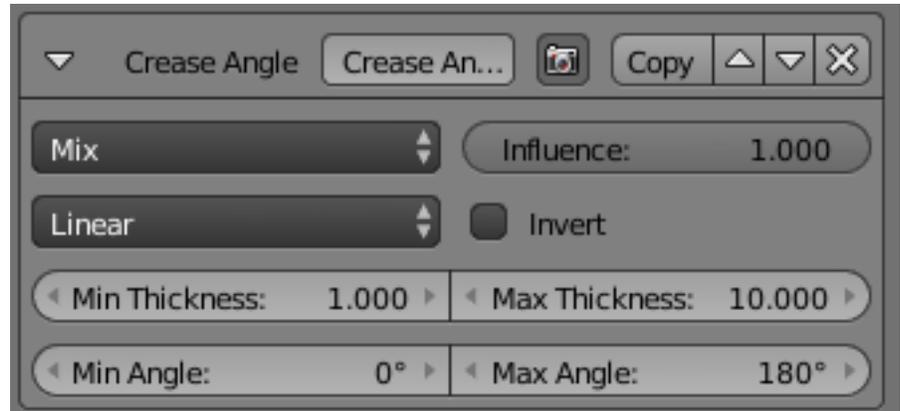
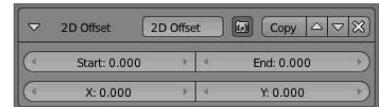


Fig. 2.2418: Line Style Geometry Overall UI

**Modifiers** There are thirteen geometry modifiers available. These modifiers have no mix nor influence settings, as they always completely apply to the strokes' geometry (like object modifiers do). They take the resulting two-dimensional strokes from the Freestyle line set and displace or deform them in various ways.

As with other modifier stacks in Blender, they are applied from top to bottom.

**2D Offset** The *2D Offset* modifier adds some two-dimensional offsets to the stroke backbone geometry. It has two sets of independent options/effects:

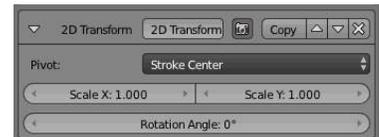


**Start and End** These two options add the given amount of offset to the start (or end) point of the stroke, along the (2D) normal at those points. The effect is blended over the whole stroke, so if you, for example, set only *Start* to **50**, the start of the stroke is offset 50 pixels along its normal, the middle of the stroke, 25 pixels along its own normal, and the end point isn't moved.

**X and Y** These two options simply add a constant horizontal and/or vertical offset to the whole stroke.

**2D Transform** The *2D Transform* modifier applies two-dimensional scaling and/or rotation to the stroke backbone geometry. Scale is applied before rotation.

The center (pivot point) of these 2D transformations can be:



**Stroke Center** The median point of the stroke.

**Stroke Start** The beginning point of the stroke.

**Stroke End** The end point of the stroke.

**Stroke Point Parameter** The *Stroke Point Parameter* factor controls where along the stroke the pivot point is (0 . 0 means start point; 1 . 0 end point).

**Absolute 2D Point** The *Pivot X* and *Pivot Y* allows you to define the position of the pivot point in the final render (from the bottom left corner). **WARNING** : Currently, you have to take into account the *real* render size, i.e. resolution **and** resolution percentage!

**Scale X and Scale Y** The scaling factors, in their respective axes.

**Rotation Angle** The rotation angle.

**Backbone Stretcher** The *Backbone Stretcher* modifier stretches (adds some length to) the beginning and end of the stroke.

**Backbone Length** Length to add to the strokes' ends.

**Bezier Curve** The *Bezier Curve* modifier replaces the stroke by a Bezier approximation of it.

**Error** The maximum distance allowed between the new Bezier curve and the original stroke.

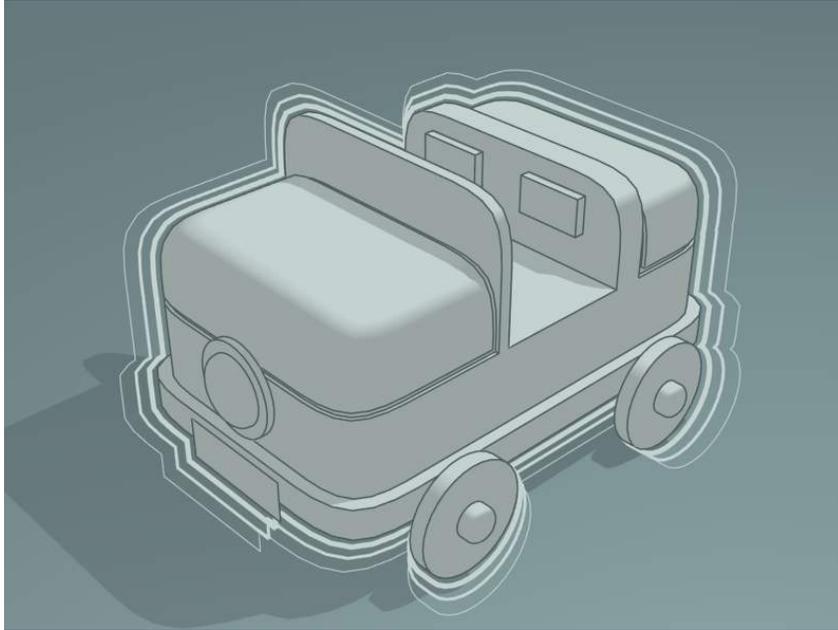


Fig. 2.2419: 2D Transform modifier File:ToyCar\_Three\_Contours.zip

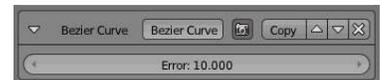


Fig. 2.2420: Bezier Curve modifier demo by T.K. File:toyCar\_bezier.zip

**Blueprint** The *Blueprint* modifier produces blueprint-like strokes using either circular, elliptical, or square contours. A blueprint here refers to those lines drawn at the beginning of free-hand drawing to capture the silhouette of objects with a simple shape such as circles, ellipses and squares.



**Shape** Which base shapes to use for this blueprint: *Circles*, *Ellipses* or *Squares*.

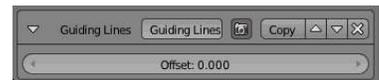
**Rounds** How many rounds are generated, as if the pen draws the same stroke several times (i.e. how many times the process is repeated).

**Random Radius and Random Center** For the *Circles* and *Ellipses* shapes. Adds some randomness to each round in the relevant aspect. Using more than one round with no randomness would be meaningless, as they would draw over each other exactly.

**Backbone Length and Random Backbone** For the *Squares* shapes. The first adds some extra length to each edge of the generated squares (also affected by the second parameter). The second adds some randomness to the squares.

Note that the *Min 2D Length* feature from the *Strokes* settings is quite handy here, to avoid the noise generated by small strokes...

**Guiding Lines** The *Guiding Lines* modifier replaces a stroke by a straight line connecting both of its ends.



**Offset** Offset the start and end points along the original stroke, before generating the new straight one.

This modifier will produce reasonable results when strokes are short enough, because shorter strokes are more likely to be well approximated by straight lines. Therefore, it is recommended to use this modifier together with one of the splitting options (by 2D angle or by 2D length) from the *Strokes* panel.

**Perlin Noise 1D** The *Perlin Noise 1D* modifier adds one-dimensional Perlin noise to the stroke. The curvilinear abscissa (value between 0 and 1 determined by a point's position relative to the first and last point of a stroke) is used as the input to the noise function to generate noisy displacements.

This means that this modifier will give an identical result for two strokes with the same length and sampling interval.

**Frequency** How dense the noise is (kind of a scale factor along the stroke).

**Amplitude** How much the noise distorts the stroke in the *Angle* direction.

**Seed** The seed of the random generator (the same seed over a stroke will always give the same result).

**Octaves** The "level of detail" of the noise.

**Angle** In which direction the noise is applied (0 . 0 is fully horizontal).

**Perlin Noise 2D** The *Perlin Noise 2D* modifier adds one-dimensional Perlin noise to the stroke. The modifier generates noisy displacements using 2D coordinates of stroke vertices as the input of the noise generator.

Its settings are exactly the same as the *Perlin Noise 1D* modifier.

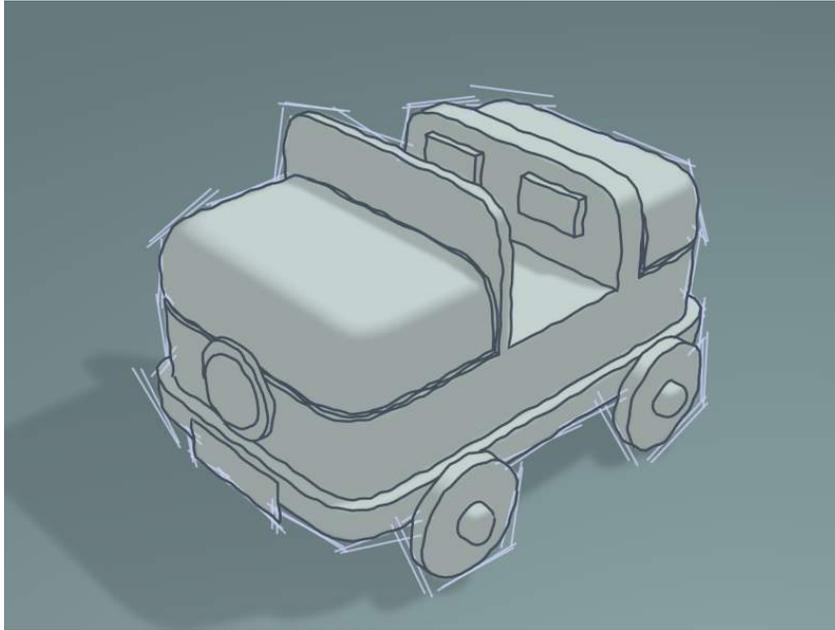
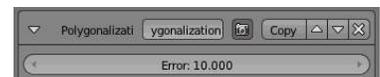


Fig. 2.2421: Guiding Lines modifier Demo by T.K. File:ToyCar\_Guiding\_Line.zip



**Polygonization** The *Polygonization* modifier simplifies strokes as much as possible (in other words, it transforms smooth strokes into jagged polylines).

**Error** The maximum distance allowed between the new simplified stroke and the original one (the larger this value is, the more jagged/approximated the resulting polylines are).

**Sampling** The *Sampling* modifier changes the definition, precision of the stroke, for the following modifiers.



**Sampling** The smaller this value, the more precise are the strokes. Be careful; too small values will require a huge amount of time and memory during render!

**Sinus Displacement** The *Sinus Displacement* modifier adds a sinusoidal displacement to the stroke.



**Wavelength** How wide the undulations are along the stroke.

**Amplitude** How high the undulations are across the stroke.

**Phase** Allows “offsetting” (“moving”) the undulations along the stroke.

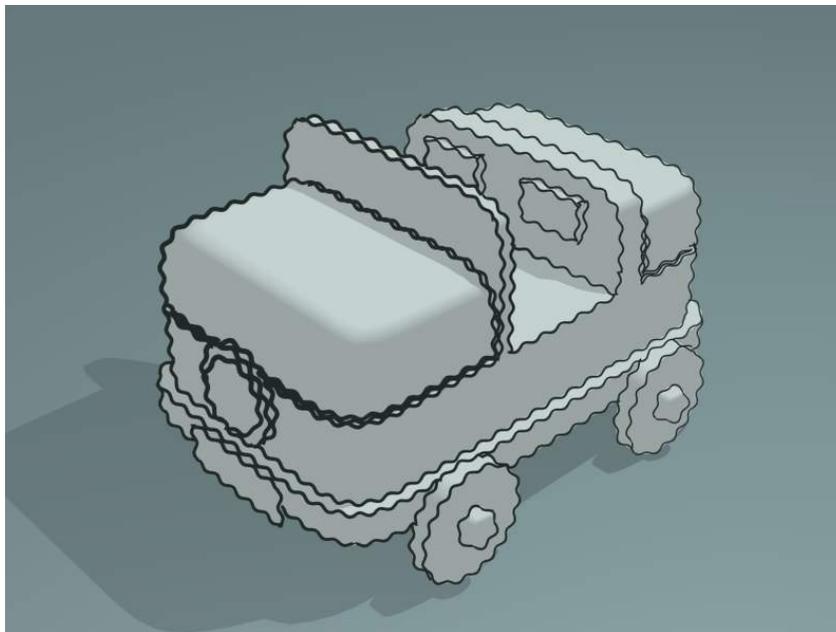


Fig. 2.2422: Sinus Displacement modifier demo by T.K. File:ToyCar\_Sinus.zip

**Spatial Noise** The *Spatial Noise* modifier adds some spatial noise to the stroke. Spatial noise displacements are added in the normal direction (i.e., the direction perpendicular to the tangent line) evaluated at each stroke vertex.



**Amplitude** How much the noise distorts the stroke.

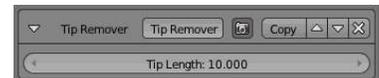
**Scale** How wide the noise is along the stroke.

**Octaves** The level of detail of the noise.

**Smooth** When enabled, apply some smoothing over the generated noise.

**Pure Random** When disabled, the next generated random value depends on the previous one; otherwise they are completely independent. Disabling this setting gives a more “consistent” noise along a stroke.

**Tip Remover** The *Tip Remover* modifier removes a piece of the stroke at its beginning and end.



**Tip Length** Length of stroke to remove at both of its tips.



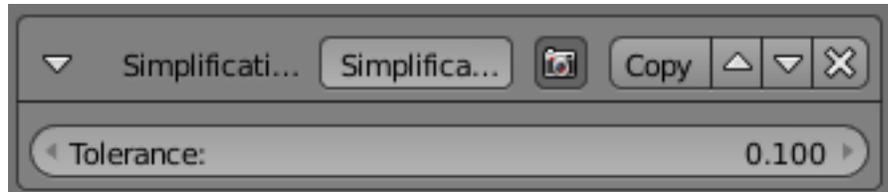
**Simplification** The *Simplification* modifier merges stroke vertices that lie close to one another, like the *Decimate* modifier for meshes.

**Tolerance** Measure for how close points have to be to each other to be merged. A higher tolerance means more vertices are merged.

## Python Scripting Mode

The Python Scripting mode offers full programmability for line stylization. In this control mode, all stylization operations are written as Python scripts referred to as style modules in the Freestyle terminology. The input to a style module is a view map (i.e., a set of detected feature edges), and the output is a set of stylized strokes.

A style module is composed of successive calls of five basic operators: selection, chaining, splitting, sorting and stroke creation. The selection operator identifies a subset of input feature edges based on one or more user-defined selection conditions (predicates). The selected edges are processed with the chaining, splitting and sorting operators to build chains of feature edges. These operators are also controlled by user-supplied predicates and functions in order to determine how to transform the feature



edges into chains. Finally, the chains are transformed into stylized strokes by the stroke creation operator, which takes a list of user-defined stroke shaders.

Python style modules are stored within .blend files as text datablocks. External style module files first need to be loaded in the Text Editor window. Then the pull-down menu within an entry of the style module stack allows you to select a module from the list of loaded style modules.

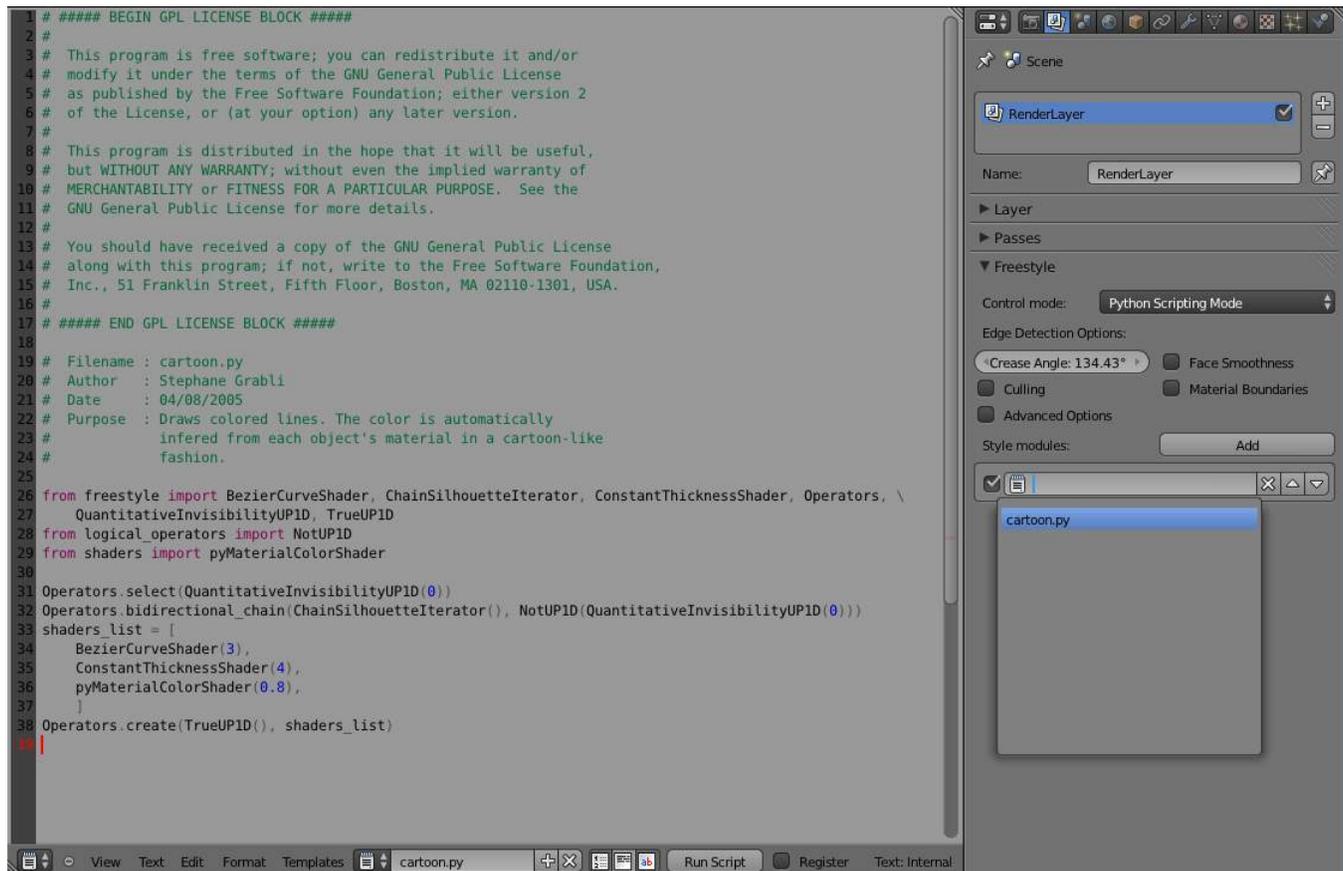


Fig. 2.2423: A screen capture of a style module (cartoon.py) loaded in the Text Editor window (left), as well as Freestyle options in the Python Scripting mode in the Render Layers buttons (right)

Freestyle for Blender comes with a number of Python style modules that can serve as a starting point of your own style module writing. See also the section of the Freestyle Python API in the Blender Python API reference manual for the full detail of style module constructs.



## Writing Style Modules

A style module is a piece of code responsible for the stylization of Freestyle line drawing. The input of a style module is a set of feature edges called view map (ViewMap). The output is a set of stylized lines also referred to as strokes. A style module is structured as a pipeline of operations that allow for building strokes from the input edges within the view map.

There are five kinds of operations (listed with corresponding operator functions):

- Selection `Operators.select()`
- Chaining `Operators.chain()`, `Operators.bidirectional_chain()`
- Splitting `Operators.sequential_split()`, `Operators.recursive_split()`
- Sorting `Operators.sort()`
- Stroke creation `Operators.create()`

The input view map is populated with a set of `ViewEdge` objects. The selection operation is used to pick up `ViewEdges` of interest to artists based on user-defined selection conditions (predicates). Chaining operations take the subset of `ViewEdges` and build `Chains` by concatenating `ViewEdges` according to user-defined predicates and functions. The `Chains` can be further refined by splitting them into smaller pieces (e.g., at points where edges make an acute turn) and selecting a fraction of them (e.g., to keep only those longer than a length threshold). The sorting operation is used to arrange the stacking order of chains to draw one line on top of another. The chains are finally transformed into stylized strokes by the stroke creation operation applying a series of stroke shaders to individual chains.

`ViewEdges`, `Chains` and `Strokes` are generically referred to as one-dimensional (1D) elements. A 1D element is a polyline that is a series of connected straight lines. Vertices of 1D elements are called 0D elements in general.

All the operators act on a set of active 1D elements. The initial active set is the set of `ViewEdges` in the input view map. The active set is updated by the operators.

**Selection** The selection operator goes through every element of the active set and keeps only the ones satisfying a certain predicate. The `Operators.select()` method takes as the argument a unary predicate that works on any `Interface1D` that represents a 1D element. For example:

```
Operators.select(QuantitativeInvisibilityUP1D(0))
```

This selection operation uses the `QuantitativeInvisibilityUP1D` predicate to select only the visible `ViewEdge` (more precisely, those whose quantitative invisibility is equal to 0). The selection operator is intended to selectively apply the style to a fraction of the active 1D elements.

It is noted that `QuantitativeInvisibilityUP1D` is a class implementing the predicate that tests line visibility, and the `Operators.select()` method takes an instance of the predicate class as argument. The testing of the predicate for a given 1D element is actually done by calling the predicate instance, that is, by invoking the `__call__` method of the predicate class. In other words, the `Operators.select()` method takes as argument a functor which in turn takes an `Interface0D` object as argument. The Freestyle Python API employs functors extensively to implement predicates, as well as functions.

**Chaining** The chaining operators act on the set of active `ViewEdge` objects and determine the topology of the future strokes. The idea is to implement an iterator to traverse the `ViewMap` graph by marching along `ViewEdges`. The iterator defines a chaining rule that determines the next `ViewEdge` to follow at a given vertex (see `ViewEdgeIterator`). Several such iterators are provided as part of the Freestyle Python API (see `ChainPredicateIterator` and `ChainSilhouetteIterator`). Custom iterators can be defined by inheriting the `ViewEdgeIterator` class. The chaining operator also takes as argument a `UnaryPredicate` working on `Interface1D` as a stopping criterion. The chaining stops when the iterator has reached a `ViewEdge` satisfying this predicate during the march along the graph.

Chaining can be either unidirectional `Operators.chain()` or bidirectional `Operators.bidirectional_chain()`. In the latter case, the chaining will propagate in the two directions from the starting edge.

The following is a code example of bidirectional chaining:

```
Operators.bidirectional_chain(
    ChainSilhouetteIterator(),
    NotUP1D(QuantitativeInvisibilityUP1D(0)),
)
```

The chaining operator uses the `ChainSilhouetteIterator` as the chaining rule and stops chaining as soon as the iterator has come to an invisible `ViewEdge`.

The chaining operators process the set of active `ViewEdge` objects in order. The active `ViewEdges` can be previously sorted using the `Operators.sort()` method (see below). It starts a chain with the first `ViewEdge` of the active set. All `ViewEdges` that have already been involved in the chaining process are marked (in the case of the example above, the time stamp of each `ViewEdge` is modified by default), in order not to process the same `ViewEdge` twice. Once the chaining reaches a `ViewEdge` that satisfies the stopping predicate, the chain is terminated. Then a new chain is started from the first unmarked `ViewEdge` in the active set. This operation is repeated until the last unmarked `ViewEdge` of the active set was processed. At the end of the chaining operation, the active set is set to the Chains that have just been constructed.

**Splitting** The splitting operation is used to refine the topology of each Chain. Splitting is performed either sequentially or recursively. Sequential splitting `Operators.sequentialSplit()` in its basic form, parses the Chain at a given arbitrary resolution and evaluates a unary predicate (working on 0D elements) at each point along the Chain. Every time the predicate is satisfied, the chain is split into two chains. At the end of the sequential split operation, the active set of chains is set to the new chains.

```
Operators.sequentialSplit(TrueUP0D(), 2)
```

In this example, the chain is split every 2 units. A more elaborated version uses two predicates instead of one: One to determine the starting point of the new chain and the other to determine its ending point. This second version can lead to a set of Chains that are disjoint or that overlap if the two predicates are different. (see `Operators.sequentialSplit()` for more details).

Recursive splitting `Operators.recursiveSplit()` evaluates a function on the 0D elements along the Chain at a given resolution and find the point that gives the maximum value for the function. The Chain is then split into two at that point. This process is recursively repeated on each of the two new Chains, until the input Chain satisfies a user-specified stopping condition.

```
func = Curvature2DAngleF0D()
Operators.recursive_split(func, NotUP1D(HigherLengthUP1D(5)), 5)
```

In the code example above, the Chains are recursively split at points of the highest 2D curvature. The curvature is evaluated at points along the Chain at a resolution of 5 units. Chains shorter than 5 units won't be split anymore.

**Sorting** The sorting operator `Operators.sort()` arranges the stacking order of active 1D elements. It takes as argument a binary predicate used as a “smaller than” operator to order two 1D elements.

```
Operators.sort(Length2DBP1D())
```

In this code example, the sorting uses the `Length2DBP1D` binary predicate to sort the `Interface1D` objects in the ascending order in terms of 2D length.

The sorting is particularly useful when combined with causal density. Indeed, the causal density evaluates the density of the resulting image as it is modified. If we wish to use such a tool to decide to remove strokes whenever the local density is too high, it is important to control the order in which the strokes are drawn. In this case, we would use the sorting operator to insure that the most “important” lines are drawn first.

**Stroke creation** Finally, the stroke creation operator `Operators.create()` takes the active set of Chains as input and build Strokes. The operator takes two arguments. The first is a unary predicate that works on `Interface1D` that is designed

to make a last selection on the set of chains. A Chain that doesn't satisfy the condition won't lead to a Stroke. The second input is a list of shaders that will be responsible for the shading of each built stroke.

```
shaders_list = [
    SamplingShader(5.0),
    ConstantThicknessShader(2),
    ConstantColorShader(0.2, 0.2, 0.2, 1),
]
Operators.create(DensityUP1D(8, 0.1, IntegrationType.MEAN), shaders_list)
```

In this example, the `DensityUP1D` predicate is used to remove all Chains whose mean density is higher than 0.1. Each chain is transformed into a stroke by resampling it so as to have a point every 5 units and assigning to it a constant thickness of 2 units and a dark gray constant color.

**User control on the pipeline definition** Style module writing offers different types of user control, even though individual style modules have a fixed pipeline structure. One is the sequencing of different pipeline control structures, and another is through the definition of functor objects that are passed as argument all along the pipeline.

Different pipeline control structures can be defined by sequencing the selection, chaining, splitting, and sorting operations. The stroke creation is always the last operation that concludes a style module.

Predicates, functions, chaining iterators, and stroke shaders can be defined by inheriting base classes and overriding appropriate methods. See the reference manual entries of the following base classes for more information on the user-scriptable constructs.

- UnaryPredicate0D
- UnaryPredicate1D
- BinaryPredicate0D
- BinaryPredicate1D
- UnaryFunction0DDouble
- UnaryFunction0DEdgeNature
- UnaryFunction0DFloat
- UnaryFunction0DId
- UnaryFunction0DMaterial
- UnaryFunction0DUnsigned
- UnaryFunction0DVec2f
- UnaryFunction0DVec3f
- UnaryFunction0DVectorViewShape
- UnaryFunction0DViewShape
- UnaryFunction1DDouble
- UnaryFunction1DEdgeNature
- UnaryFunction1DFloat
- UnaryFunction1DUnsigned
- UnaryFunction1DVec2f
- UnaryFunction1DVec3f
- UnaryFunction1DVectorViewShape
- UnaryFunction1DVoid
- ViewEdgeIterator
- StrokeShader

## Freestyle SVG Exporter

SVG exporting for Freestyle is available through an addon.

This addon can be enabled via *User Preferences > Addons > Render:Freestyle SVG Exporter*. The GUI for the exporter should now be visible in the render tab of the properties window. The exported .svg file is written to the default output path (*Properties > Render > Output*).

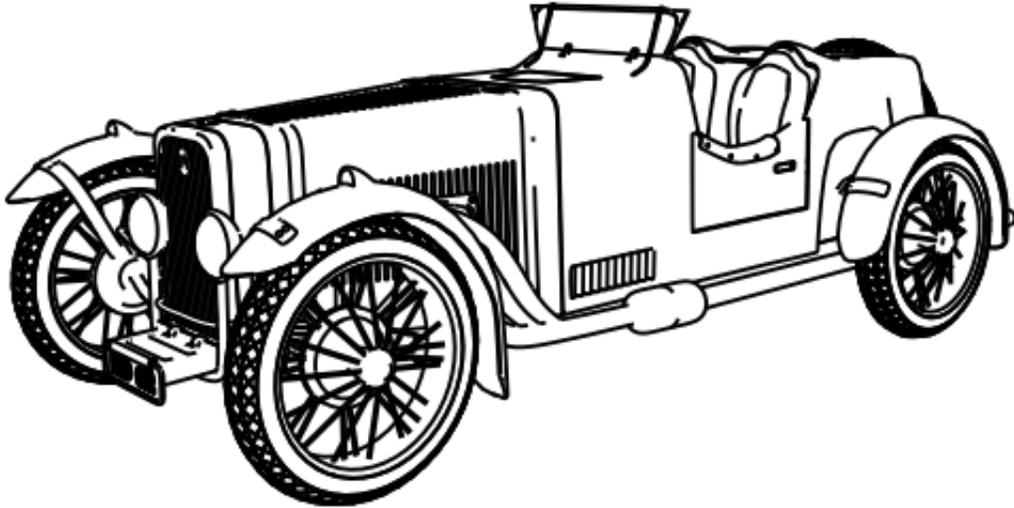
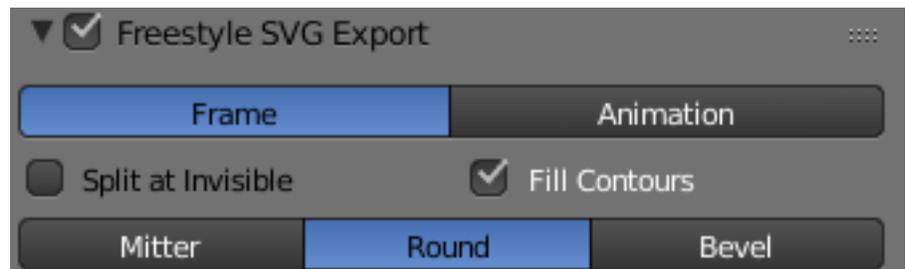


Fig. 2.2428: An example of a .svg result produced by the Freestyle SVG Exporter. Model by [Blendergoodies](#)

## Options



**Mode** Option between Frame and Animation. Frame will render a single frame, Animation will bundle all rendered frames into a single .svg file.

**Split at Invisible** By default the exporter won't take invisible vertices into account and export them like they are visible. Some stroke modifiers, like Blueprint, mark vertices as invisible to achieve a certain effect. Enabling this option will make the paths split when encountering an invisible vertex, which leads to a better result.

**Fill Contours** The contour of objects is filled with their material color. Note that this features is somewhat unstable - especially with animations.

**Stroke Cap Style** Defines the style the stroke caps will have in the SVG output.

## Exportable Properties

Because the representation of Freestyle strokes and SVG path objects is fundamentally different, a one on one translation between Freestyle and SVG is not possible. The main shortcoming of SVG compared to Freestyle is that Freestyle defines

style per-point, where SVG defines it per-path. This means that Freestyle can produce much more complex results that are impossible to achieve in SVG.

The properties that can be exported are:

- Base color
- Base alpha
- Base thickness
- Dashes

## Animations

The exporter supports the creation of SVG animations. When the Mode is set to Animation, all frames from a render - one when rendering a frame (f12) or all when rendering an animation (shift f12) - into a single file. Most modern browsers support the rendering of SVG animations.

Fig. 2.2429: An SVG animation rendered with the exporter.

**Exporting Fills** Fills are colored areas extracted from a Freestyle render result. Specifically, they are defined by a combination of the Contour and External Contour edge type, combined with some predicates. The fill result can be unexpected, when the SVG renderer cannot correctly draw the path that the exporter has generated. This problem is extra apparent in animations.

Fig. 2.2430: An example of a .svg result produced by the Freestyle SVG Exporter. Model by [Julien Deswaef](#)

Fills support holes and layering. When using layers, the exporter tries to render objects with the same material as the patch. The exporting of fills and especially the order in which they are layered is by no means perfect. In most cases, these problems can be easily solved in Inkscape or a text editor.

## Links

Here are some links to external data regarding Freestyle.

## Videos

The Light At The End.

[mmd\\_tools test2 with Blender+Freestyle \( AM4:30\)](#)

## Video Tutorials

[An introduction to Freestyle plugin for Blender: “sketching” Suzanne / HD](#)

[Using freestyle in blender](#)

[Tutorial: Blender 3D - Freestyle and Composite](#)

[Blender Tutorial: Freestyle](#)

## Tutorials

**Freestyle basics** <http://studiollb.wordpress.com/2012/02/29/freestyle-introductory-tutorial/> <http://jikz.net/archives/364>  
<http://jikz.net/archives/329>

**Edge types** <https://studiollb.wordpress.com/2012/09/08/freestyle-101-edge-types/>

**Line style basic** <http://studiollb.wordpress.com/2012/09/08/freestyle-101-line-style-basic/>

**Line style modifiers** <http://studiollb.wordpress.com/2012/09/08/freestyle-101-line-style-modifier-part-1/> <http://studiollb.wordpress.com/2012/09/08/freestyle-101-line-style-modifier-part-2/>  
<http://studiollb.wordpress.com/2012/09/15/freestyle-101-planning-and-along-stroke-line-style-modifier/>

**Tips and tricks** <http://studiollb.wordpress.com/2012/02/03/freestyle-tips/> (Old)

Misc

- FreeStyle Users' improvement suggestions.
- FreeStyle integration into Blender blog
- Early documentation of FreeStyle

## 2.8.10 Workflows

### Rendering Animations

While rendering stills will allow you to view and save the image from the render buffer when it's complete, animations are a series of images, or frames, and are automatically saved directly out to disk after being rendered.

After rendering the frames, you may need to edit the clips, or first use the Compositor to do green-screen masking, matting, color correction, DOF, and so on to the images. That result is then fed to the Sequencer where the strips are cut and mixed and a final overlay is done.

Finally you can render out from the Sequencer and compress the frames into a playable movie clip.

### Workflow

Generally, you do a lot of intermediate renders of different frames in your animation to check for timing, lighting, placement, materials, and so on. At some point, you are ready to make a final render of the complete animation for publication.

There are two approaches you can use when making a movie, or animation, with or without sound. The approach you should use depends on the amount of CPU time you will need to render the movie. You can render a "typical" frame at the desired resolution, and then multiply by the number of frames that will ultimately go into the movie, to arrive at a total render time.

If the total render time is an hour or more, you want to use the "Frame Sequence" approach. For example, if you are rendering a one-minute video clip for film, there will be (60 seconds per minute) \* (24 frames per second) or 1440 frames per minute. If each frame takes 30 seconds to render, then you will be able to render two frames per minute, or need 720 minutes (12 hours) of render time.

Rendering takes all available CPU time; you should render overnight, when the computer is not needed, or set Blender to a low priority while rendering, and work on other things (be careful with the RAM space!).

The **Direct Approach** - highly **not** recommended and not a standard practice - is where you set your output format to an AVI or MOV format, and click ANIM to render your scene directly out to a movie file. Blender creates one file that holds all the frames of your animation. You can then use Blender's VSE to add an audio track to the animation and render out to an MPEG format to complete your movie.

The **Frame Sequence** is a much more stable approach, where you set your output format to a still format (such as JPG, PNG or MultiLayer), and click ANIM to render your scene out to a set of images, where each image is the frame in the sequence.

Blender creates a file for each frame of the animation. You can then use Blender's compositor to perform any frame manipulation (post processing). You can then use Blender's VSE to load that final image sequence, add an audio track to the animation, and render out to an MPEG format to complete your movie. The Frame Sequence approach is a little more complicated and takes more disk space, but gives you more flexibility.

Here are some guidelines to help you choose an approach.

#### Direct Approach

- short segments with total render time < 1 hour
- stable power supply
- computer not needed for other uses

#### Frame Sequence Approach

- total render time > 1 hour
- **post-production work needed**
  - Color/lighting adjustment
  - Green screen / matte replacement
  - Layering/compositing
  - Multiple formats and sizes of ultimate product
- intermediate frames/adjustments needed for compression/codec
- precise timing (e.g. lip-sync to audio track) needed in parts
- may need to interrupt rendering to use the computer, and want to be able to resume rendering where you left off.

#### Frame Sequence Workflow

- First prepare your animation.
- In the *Dimensions* panel, choose the render size, Pixel Aspect Ratio, and the Range of Frames to use, as well as the frame rate, which should already be set.
- In the Output panel set up your animation to be rendered out as images, generally using a format that does not compromise any quality (I prefer PNG or MultiLayer because of their loss-less nature).
- Choose the output path and file type in the Output panel as well, for example `//render/my-anim-`.
- Confirm the range of your animation frame Start and End.
- Save your .blend file.
- Press the big *Animation* button. Do a long task [like sleeping, playing a video game, or cleaning your driveway] while you wait for your computer to finish rendering the frames.
- Once the animation is finished, use your OS file explorer to navigate into the output folder ("`render` in this example). You will see lots of images (.png or .exr, etc... depending on the format you chose to render) that have a sequence number attached to them ranging from 0000 to a max of 9999. These are your single frames.
- In Blender, now go into the [video sequence editor](#).
- Choose *Add Image* from the add menu. Select all the frames from your output folder that you want to include in your animation (Press A to Select All easily). They will be added as a strip to the sequence editor.

- Now you can edit the strip and add effects or simply leave it like it is. You can add other strips, like an audio strip.
- Scrub through the animation, checking that you have included all the frames.
- In the Scene Render buttons, in the Post Processing panel, activate *Sequencer*.
- In the Format panel, choose the container and codec you want (e.g. MPEG H.264) and configure it. The video codecs are described on the previous page: [Output Options](#).
- Click the ANIMATION render button and Blender will render out the sequence editor output into your movie.

Why go through all this hassle? Well, first of all, if you render out single frames you can stop the render at any time by pressing `ESC` in the render window. You will not lose the frames you have already rendered, since they have been written out to individual files. You can always adjust the range you want to continue from where you left off.

You can edit the frames afterwards and post-process them. You can add neat effects in the sequence editor. You can render the same sequence into different resolutions (640x480, 320x240, etc) and use different codecs (to get different file sizes and quality) with almost no effort whatsoever.

## Options

### Post Processing Panel

**Sequencer** Renders the output of the sequence editor, instead of the view from the 3D scene's active camera. If the sequence contains scene strips, these will also be rendered as part of the pipeline. If Do Composite is also enabled, the Scene strip will be the output of the Compositor.

**Compositing** Renders the output from the Compositing noodle, and then pumps all images through the Composite node map, displaying the image fed to the Composite Output node.

## Hints

**You accidentally turned off you're PC right in the middle of rendering my movie!** Unless your animation renders in a few minutes, it's best to render the animation as separate image files. Instead of rendering directly to a compressed movie file, use a loss-less format (PNG for example).

This allows you an easy recovery if there is a problem and you have to re-start the rendering, since the frames you have already rendered will still be in the output directory.

Just disable the *Overwrite* option to start rendering where you left off.

You can then make a movie out of the separate frames with Blender's sequence editor or using 3rd party encoding software.

**Animation Preview** It can be useful to render a subset of the animated sequence, since only part of an animation may have an error.

Using an image format for output, you can use the *Frame Step* option to render every *N*'th frame. Then disable *Overwrite* and re-render with *Frame Step* set to 1.

## Command Line

In some situations we want to increase the render speed, access blender remotely to render something or build scripts that use the command line.

One advantage of using the command line is that we don't need the X server (in the case of Linux) and consequently we can render remotely by SSH or telnet.

To see a list of available flags (for example to specify which scene to render, the end frame number, etc...), simply run:

```
blender --help
```

**Note:** Arguments are executed in the order they are given!

The following command won't work, since the output and extension is set after blender is told to render:

```
blender -b file.blend -a -x 1 -o //render
```

The following command will behave as expected.

```
blender -b file.blend -x 1 -o //render -a
```

**Always position `-f` or `-a` as the last arguments.**

## Platforms

How to actually execute Blender from the command line depends on the platform and where you have installed Blender. Here are basic instructions for the different platforms.

**Linux** Open a terminal, then go to the directory where Blender is installed, and run the blender command like this.

```
cd <blender installation directory>  
./blender
```

If you have Blender installed in your PATH (usually when Blender is installed through a distribution package), you can simply run:

```
blender
```

**Mac OS X** Open the terminal application, go to the directory where Blender is installed, and run the executable within the app bundle, with commands like this:

```
cd /Applications/Blender  
./blender.app/Contents/MacOS/blender
```

If you need to do this often, you can make an alias so that typing just `blender` in the terminal works. For that you can run a command like this in the terminal (with the appropriate path).

```
echo "alias blender=/Applications/Blender/blender.app/Contents/MacOS/blender" >> ~/.profile
```

If you then open a new terminal, the following command will work:

```
blender
```

**Windows** Open the Command Prompt, go to the directory where Blender is installed, and then run the blender command.

```
cd c:\<blender installation directory>  
blender
```

You can also add the Blender folder to your system PATH so that do you do not have to change to it each time.

## Examples

Here are some common examples of command line rendering:

## Single Image

```
blender -b file.blend -f 10
```

**-b** Render in the background (without UI).

**file.blend** Path to the blend file to render.

**-f 10** Render only the 10th frame.

```
blender -b file.blend -o /project/renderers/frame_##### -F EXR -f -2
```

**-o /project/renderers/frame\_#####** Path of where to save the rendered image, using 5 padded zeros for the frame number.

**-F EXR** Override the image format specified in the blend file and save to an OpenEXR image.

**-f -2** Render only the second last frame.

**Warning:** Arguments are case sensitive! **-F** and **-f** are not the same.

## Animation

```
blender -b file.blend -a
```

**-a** Render the whole animation using all the settings saved in the blend file.

```
blender -b file.blend -E BLENDER_RENDER -s 10 -e 500 -t 2 -a
```

**-E BLENDER\_RENDER** Use the “Blender Render” engine. For a list of available renderers, run `blender -E help`.

**-s 10 -e 500** Set the start frame to 10 and the end frame to 500.

**-t 2** Use only two threads.

## Render Baking

Baking, in general, is the act of pre-computing something in order to speed up some other process later down the line. Rendering from scratch takes a lot of time depending on the options you choose. Therefore, Blender allows you to “bake” some parts of the render ahead of time, for select objects. Then, when you press Render, the entire scene is rendered much faster, since the colors of those objects do not have to be recomputed.

Render baking creates 2D bitmap images of a mesh object’s rendered surface. These images can be re-mapped onto the object using the object’s UV coordinates. Baking is done for each individual mesh, and can only be done if that mesh has been UV-unwrapped. While it takes time to set up and perform, it saves render time. If you are rendering a long animation, the time spent baking can be much less than time spent rendering out each frame of a long animation.

Use Render Bake in intensive light/shadow solutions, such as AO or soft shadows from area lights. If you bake AO for the main objects, you will not have to enable it for the full render, saving render time.

Use *Full Render* or *Textures* to create an image texture; baked procedural textures can be used as a starting point for further texture painting. Use *Normals* to make a low-resolution mesh look like a high-resolution mesh. To do that, UV-unwrap a high-resolution, finely sculpted mesh and bake its normals. Save that normal map, and *Mapping* (texture settings) the UV of a similarly unwrapped low-resolution mesh. The low-resolution mesh will look just like the high-resolution, but will have much fewer faces/polygons.

### Advantages

- Can significantly reduce render times
- Texture painting made easier

- Reduced polygon count
- Repeated renders are made faster, multiplying the time savings

### Disadvantages

- Object must be UV-unwrapped.
- If shadows are baked, lights and object cannot move with respect to each other.
- Large textures (eg 4096x4096) can be memory intensive, and be just as slow as the rendered solution.
- Human (labor) time must be spent unwrapping and baking and saving files and applying the textures to a channel.

### Options



Fig. 2.2431: Ambient Occlusion

### Bake Mode

**Full Render** Bakes all materials, textures, and lighting except specularly and SSS.

**Ambient Occlusion** Bakes ambient occlusion as specified in the World panels. Ignores all lights in the scene.

**Normalized** Normalize without using material's settings.

**Shadow** Bakes shadows and lighting.

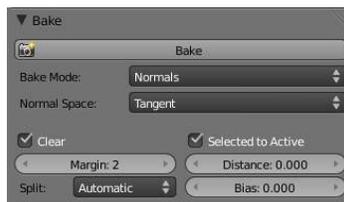


Fig. 2.2432: Normals

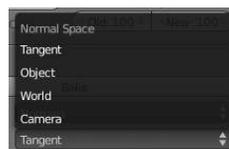


Fig. 2.2433: Normal Space

**Normals** Bakes tangent and camera-space normals (amongst many others) to an RGB image.

**Normal Space** Normals can be baked in different spaces:

**Camera space** Default method.

**World space** Normals in world coordinates, dependent on object transformation and deformation.

**Object space** Normals in object coordinates, independent of object transformation, but dependent on deformation.

**Tangent space** Normals in tangent space coordinates, independent of object transformation and deformation. This is the new default, and the right choice in most cases, since then the normal map can be used for animated objects too.

For materials the same spaces can be chosen as well, in the image texture options, next to the existing *Normal Map* setting. For correct results, the setting here should match the setting used for baking.

**Textures** Bakes colors of materials and textures only, without shading.

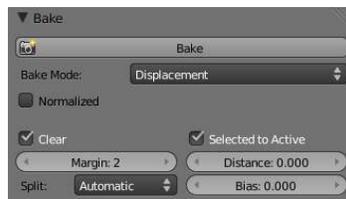


Fig. 2.2434: Displacement

**Displacement** Similar to baking normal maps, displacement maps can also be baked from a high-res object to an unwrapped low-res object, using the *Selected to Active* option.

**Normalized** Normalize to the distance.

When using this in conjunction with a subsurf and displacement modifier within Blender, it's necessary to temporarily add a heavy subsurf modifier to the 'low res' model before baking. This means that if you then use a displacement modifier on top of the subsurf, the displacement will be correct, since it's stored as a relative difference to the subsurfed geometry, rather than the original base mesh (which can get distorted significantly by a subsurf). The higher the render level subsurf while baking, the more accurate the displacements will be. This technique may also be useful when saving the displacement map out for use in external renderers.

**Emission** Bakes Emit, or the Glow color of a material.

**Alpha** Bakes Alpha values, or transparency of a material.

**Mirror Color and Intensity** Bakes Mirror color or intensity values.

**Specular Color and Intensity** Bakes specular color or specular intensity values.

### Additional Options

**Clear** If selected, clears the image to selected background color (default is black) before baking render.

**Margin** Baked result is extended this many pixels beyond the border of each UV "island," to soften seams in the texture.

**Split**

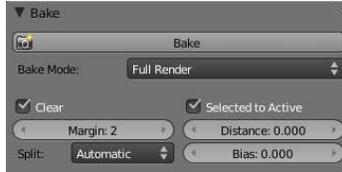


Fig. 2.2435: Full Render

**Fixed** Slit quads predictably (0,1,2) (0,2,3).

**Fixed alternate** Slit quads predictably (1,2,3) (1,3,0).

**Automatic** Split quads to give the least distortion while baking.

**Select to Active** Enable information from other objects to be baked onto the active object.

**Distance** Controls how far a point on another object can be away from the point on the active object. Only needed for *Selected to Active*. A typical use case is to make a detailed, high poly object, and then bake it's normals onto an object with a low polygon count. The resulting normal map can then be applied to make the low poly object look more detailed.

**Bias** Bias towards further away from the object (in blender units)

---

**Note:** Mesh Must be Visible in Render

If a mesh is not visible in regular render, for example because it is disabled for rendering in the Outliner or has the DupliVerts setting enabled, it cannot be baked to.

---

## Workflow

- In a 3D View window, select a mesh and enter UV/Face Select mode
- [Unwrap the mesh object](#)
- In a UV/Image Editor window, either create a new image or open an existing one. If your 3D view is in textured display mode, you should now see the image mapped to your mesh. Ensure that all faces are selected.
- In the Bake panel at the bottom of the *Render menu*, bake your desired type of image (*Full Render* etcetera.)
- When rendering is complete, Blender replaces the image with the Baked image.
- Save the image.
- Apply the image to the mesh as a UV texture. For displacement and normal maps, refer to [Bump and Normal Maps](#). For full and texture bakes, refer to [Textures](#).
- Refine the image using the process described below, or embellish with [Texture Paint](#) or an external image editor.

## Multi-View Render

For this 5-minute guide we will take an existent `.blend` file that was made for monoscopic rendering and transform it in stereo-3d ready.

---

**Note:** Multi-View drawing requires capable graphics card and drivers with *Triple Buffer* support. If the *Automatic* mode doesn't work, set the *Window Draw Method* in the [System User Preferences](#).

---



Fig. 2.2436: Creature Factory 2 by Andy Goralczyk Rendered in Stereo 3D (anaglyph)

## Introduction

Start opening up your project file, in this case `turntable.blend` from the **Creature Factory 2** Open Movie Workshop series from the Blender Institute by **Andy Goralczyk**.

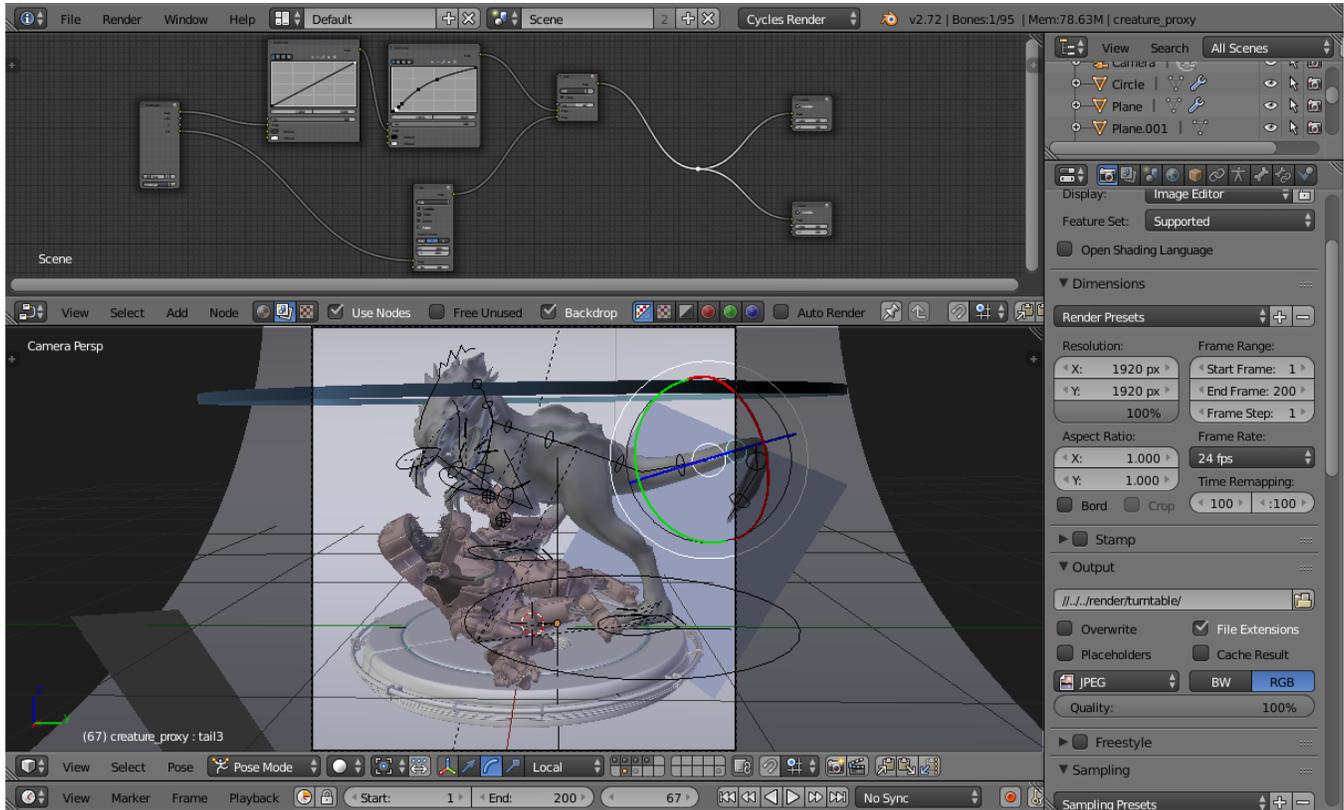


Fig. 2.2437: Turn Table Creature Factory 2

## Views Setup

Go to the Render Layers panel and enable *Views* for this scene.

**Note:** When you turn on *Views* in the scene you get 3d preview in the viewport, as well as multiple panels that are now accessible all over the user interface.

## Camera

To tweak the stereo 3d parameters select the camera in the Outliner. In the Camera panel go to the Stereoscopy tab and change the *Convergence Distance*.

The viewport will respond in real-time to those changes allowing you to preview the current depth value of the scene.

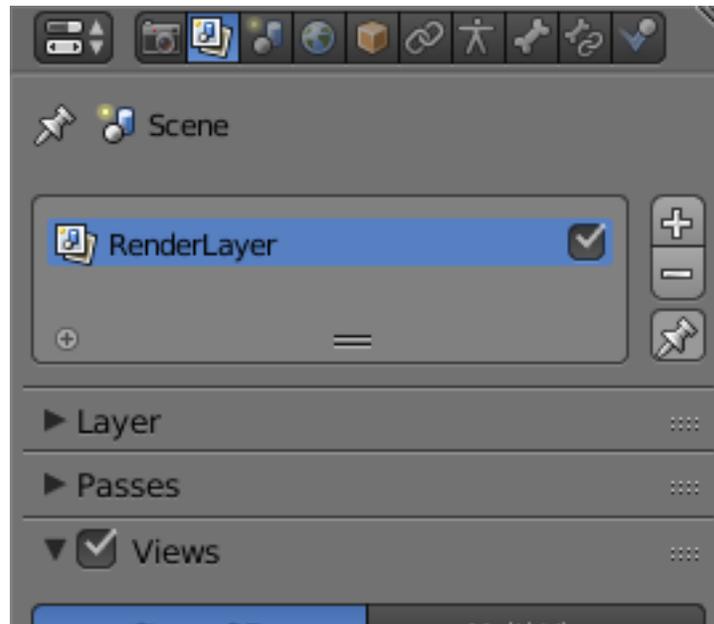


Fig. 2.2438: Scene Render Views

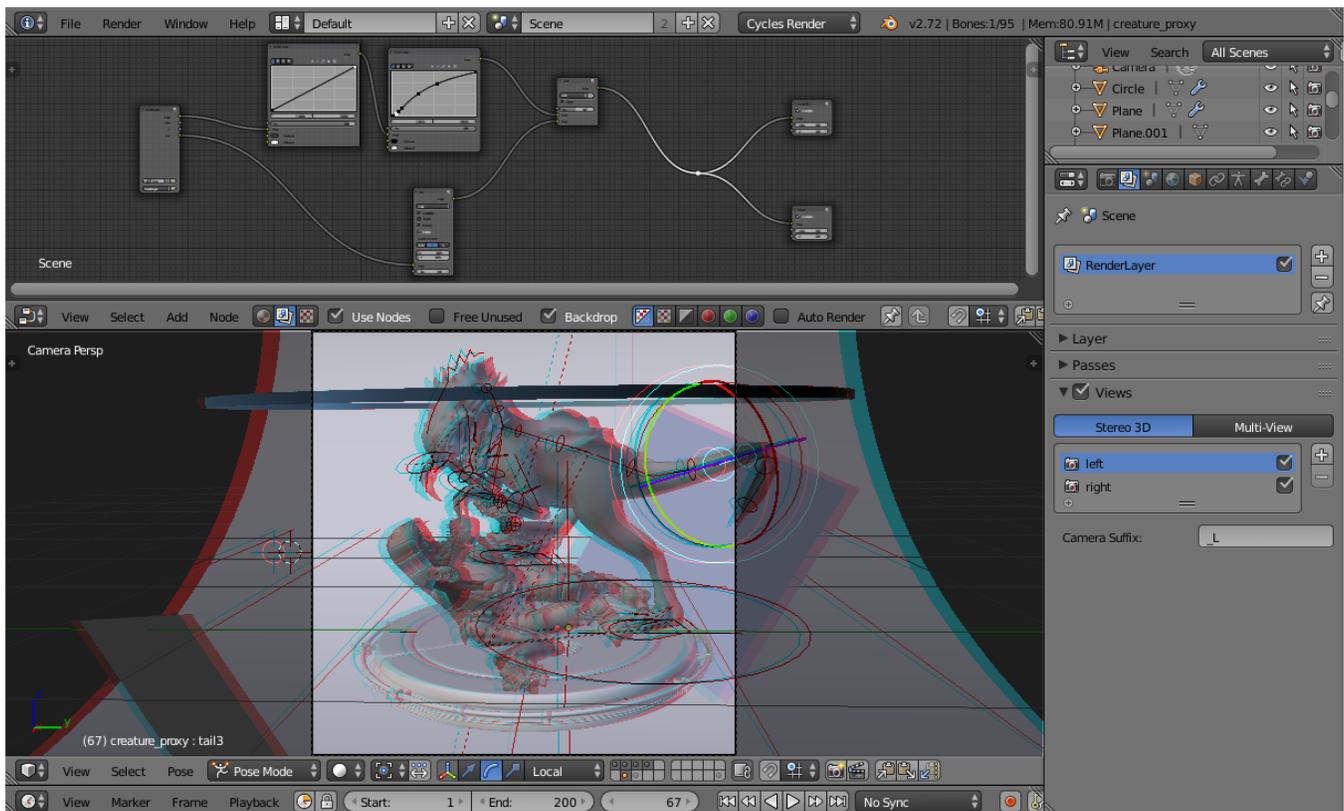


Fig. 2.2439: Viewport with 3D visualization



Fig. 2.2440: Stereo Convergence Distance

## Viewport

Before fine-tuning the camera parameters you can set the convergence plane in the viewport based in your scene depth layout. Go outside the camera view and you will instantly see the convergence plane in front of the camera.

You can toggle this and other display settings in the Stereoscopy tab of the viewport properties panel. In the following image the cameras frustum volumes are also visible.

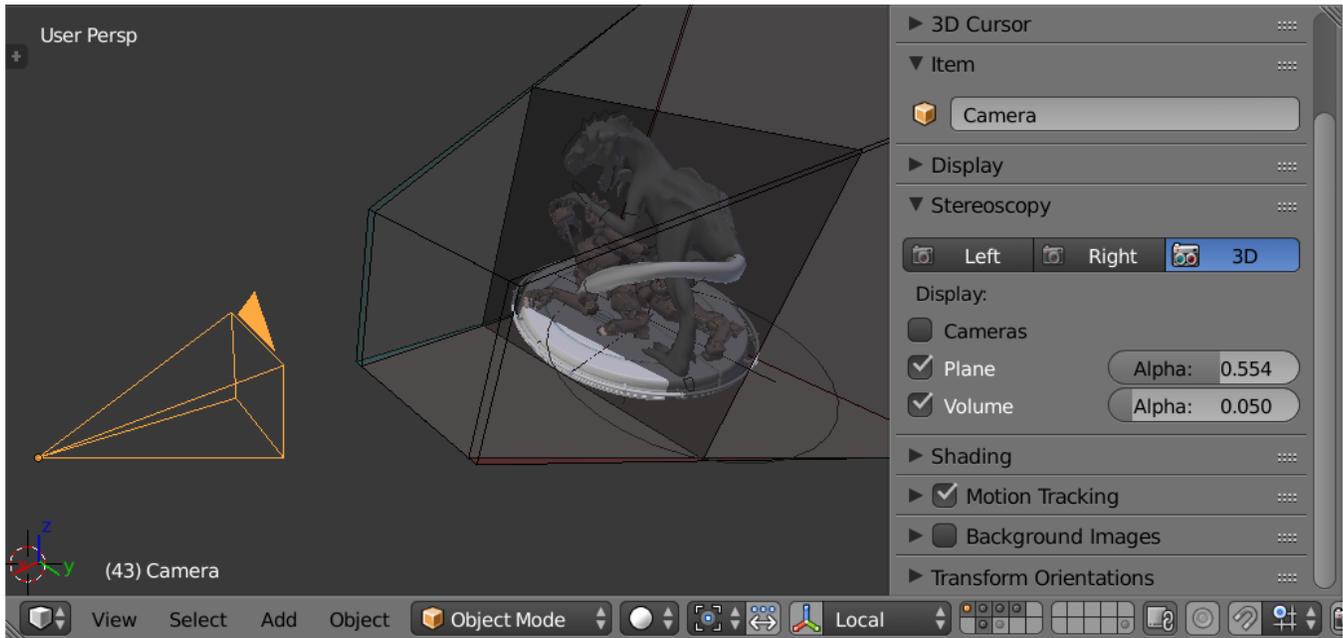


Fig. 2.2441: Viewport Plane and Volume Stereo Preview

## Stereo 3D Display

If you have a real 3d display at some point you can change the 3D display mode in the Window menu, by calling the Stereo 3D operator. Be aware that some modes require a fullscreen editor to work.

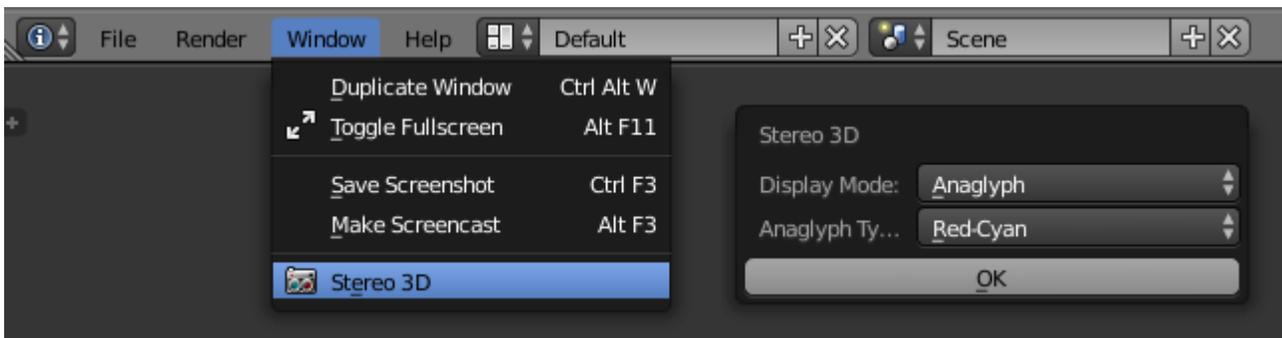


Fig. 2.2442: Window Menu, Stereo 3D Operator

## OpenGL Preview

Before rendering your scene you can save an OpenGL preview of the animation for testing in the final display. In the Render Output panel you can chose the output *Views Format*.

Fig. 2.2443: Turn Table OpenGL Rendering Preview

The options include individual files per view, top-bottom, anaglyph among others. Pick the one that fits your display requirements.

### Rendering and Image Editor

Once you are happy with the results you can render out the final animation. In the Image Editor you can inspect the individual views and the stereo result.

### Image Formats

Your final animation can be saved in more robust formats than the ones used by the OpenGL render preview. In this example we saved as cross-eyed side-by-side stereo 3d.



Fig. 2.2444: Side by Side Cross-Eye Format

### Final Considerations

As this guide showed, there is more to stereo 3d rendering than just generate two images. The earlier the stereo pipeline is considered the smoother it will get. The following sections are a more in-depth view of the individual components we visited in the workflow.

## Window Stereo 3D Display

An essential component of the Stereoscopy pipeline is the ability to display the stereo image in a proper display. Blender supports from high-end 3D displays to simple red-cyan glasses. On top of that you can set a different display mode for each window.

The display mode can be changed via the Window menu or if you create your own shortcuts for the `wm.set_stereo_3d` operator.

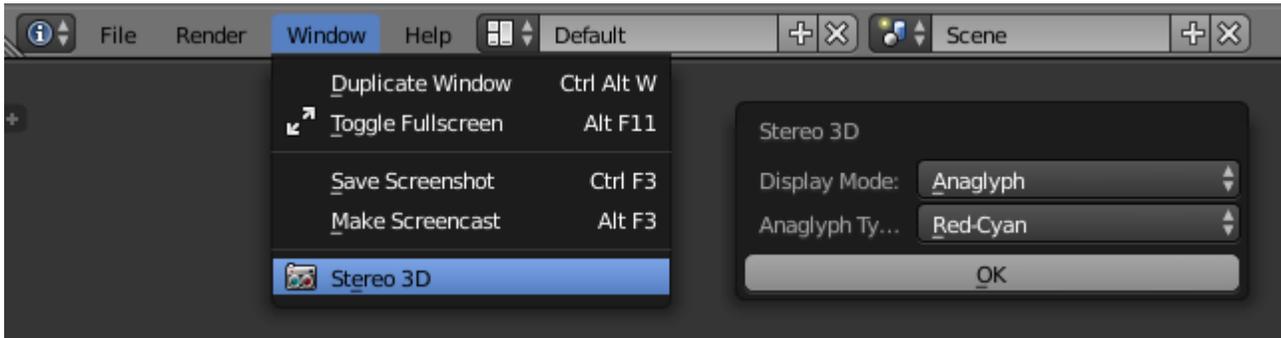


Fig. 2.2445: Window Menu, Stereo 3D Operator

### Display Mode

**Anaglyph** Render two differently filtered colored images for each eye. Anaglyph glasses are required. We support Red-Cyan, Green-Magenta and Yellow-Blue glasses.

**Interlace** Render two images for each eye into one interlaced image. A 3D-ready monitor is required. We support Row, Column and Checkerboard Interleaved. An option to Swap Left/Right helps to adjust the image for the screen. This method works better in fullscreen.

**Time Sequential** Renders alternate eyes. This method is also known as Page Flip. This requires the graphic card to support Quad Buffer and it only works in fullscreen.

**Side-by-Side** Render images for left and right eye side-by-side. There is an option to support Cross-Eye glasses. It works only in fullscreen, and it should be used with the Full Editor operator.

**Top-Bottom** Render images for left and right eye one above another. It works only in fullscreen, and it should be used with the Full Editor operator.

---

#### Note: Full Screen Stereo 3D Modes

If you have a 3D display most of the time you will use it to see in stereo 3D you will have to go to the fullscreen mode. In fact some modes will only work in the full window mode that hides most of the user interface from the work area. In this case it is recommended to work with two monitors, using the 3D screen for visualizing the stereo result while the other screen can be used for the regular Blender work.

---

### Stereo 3D Camera

When using the Stereo 3D scene view setup a stereo pair is created on-the-fly and used for rendering and previsualization. For all the purposes this works as two cameras that share most parameters (focal length, clipping, ...). The stereo pair, however, is offsetted, and can have unique rotation and shift between itself.

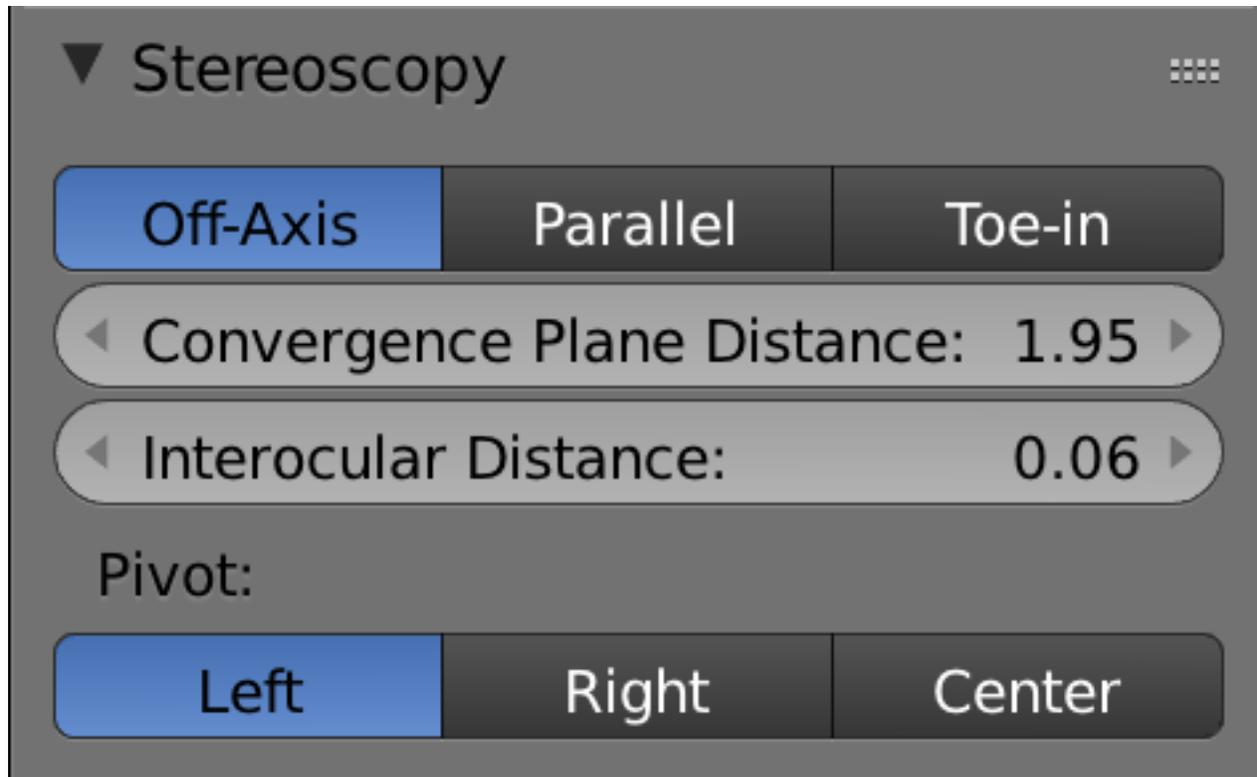


Fig. 2.2446: Stereo 3D Camera Settings

**Interocular Distance** Set the distance between the camera pair. Although the convergence of a stereo pair can be changed in post-production, different interocular distances will produce different results due to the parts of the scene being occluded from each point of view.

**Convergence Plane Distance** The converge point for the stereo cameras. This is often the distance between a projector and the projection screen. You can visualize this in the 3D Viewport.

#### Convergence Mode

**Off-Axis** The stereo camera pair is separated by the interocular distance, and shifted inwards so it converges in the convergence plane. This is the ideal format since it is the one closest to how the human vision works.

**Parallel** This method produces two parallel cameras that do not converge. Since this method needs to be manually converged it can't be used for viewing. This method is common when combining real footage with rendered elements.

**Toe-in** A less common approach is to rotate the cameras instead of shifting their frustum. The Toe-in method is rarely used in modern 3D productions.

**Pivot** The stereo pair can be constructed around the active camera with a new camera built for each eye (Center Pivot) or using the existing camera and creating (Left or Right). The latter is what is used when only one eye needs to be rendered for an existing mono 2D project.

#### Viewport Stereo 3D

When you enable 'Views' in the Render Layer panel a new are is available in the 3D Viewport properties panel. In this panel you can pick whether to see the stereo 3d in the viewport, or which camera to see. It also allow you to see the Cameras, the Plane and the Volume of the stereo cameras.

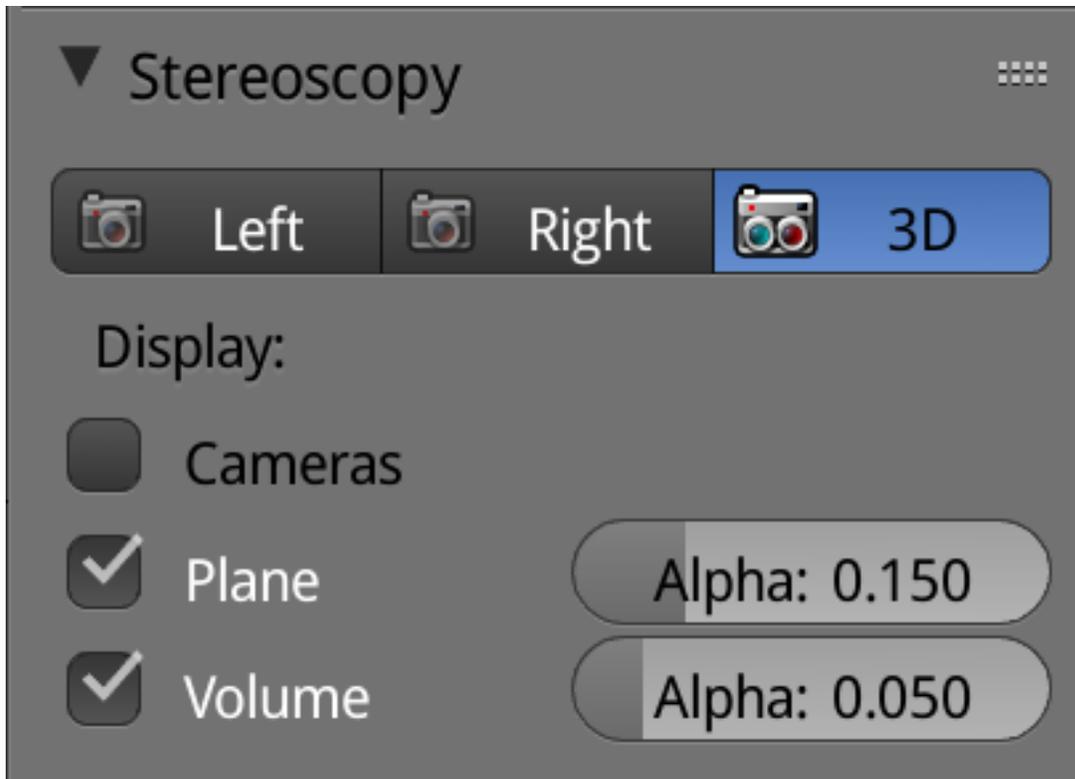


Fig. 2.2447: Viewport Stereo 3D Settings

**Cameras** When working with the Stereo 3D views setup you can inspect what each individual generated camera is looking or the combined result of them. In the Multi-View mode you can see the combined result of the left and right cameras (when available) or the current selected camera.

**Plane** The convergence plane represents the screen as it is perceived by the audience. Visualizing it in the 3D Viewport allows you to layout your scene based on your depth script outside the camera view.

**Volume** The intersection of the stereo cameras frustums helps planning the show by avoiding elements being visible by only one camera. The volume is defined by the cameras start and end clipping distances. The areas that are in the frustum of one camera only are known as **retinal rivalry areas**. They are tolerated in the negative space (the region from the convergence plane into the image) but are to be avoided at all costs in the positive space (the area from the convergence plane to the camera).

### Multi-View and Stereo 3D Image I/O

**Multi-View and Stereo 3D** Multi-View images can be saved in special formats according to the production requirements. By default the system saves each view as an individual file, thus generating as many files as views to be rendered. In stereo 3d productions, for the final deployment or even intermediary previews it's convenient to save stereo 3d images, that are ready to use with 3D displays or simple anaglyph glasses. The formats supported match the display modes available for the window.

**Lossy-Formats** Some stereo 3D formats represent a considerable loss of data. For example, the Anaglyph format will cap out entire color channels from the original image. The Top-Bottom compressed will discard half of your vertical resolution data. The Interlace will mash your data considerably. Once you export in those formats, you can still import the image back in Blender, for it to be treated as Stereo 3D. You will need to match the window stereo 3d display mode to the image stereo 3d format though.

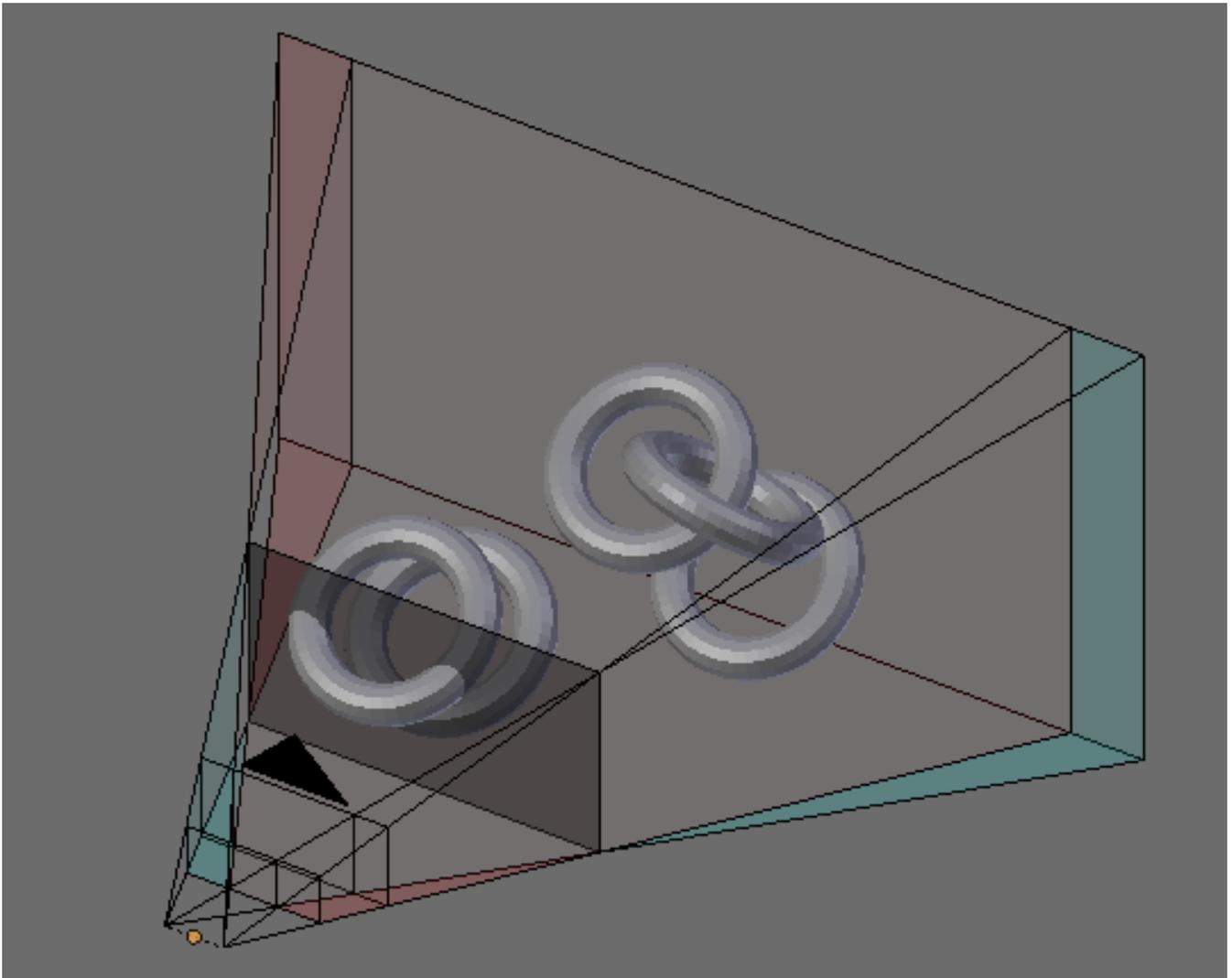


Fig. 2.2448: Viewport 3D: Convergence Plane and Volume Display

**Lossless Formats** Some formats will preserve the original data, leading to no problems on exporting and importing the files back in Blender. The Individual option will produce separate images that (if saved in a lossless encoding such as PNG or OpenEXR) can be loaded back in production with no loss of data. For the Stereo 3D formats the only lossless options are *Top-Bottom* and *Side-by-Side* without the Squeezed Frame option.

**Multi-View Openexr** Another option is to use Multi-View OpenEXR files. This format can save multiple views in a single file and is backward compatible with old OpenEXR viewers (you see only one view though). Multi-View native support is only available to OpenEXR.

## Image Editor

**View Menu** After you render your scene with Stereo 3D you will be able to see the rendered result in the combined stereo 3d or to inspect the individual views. This works for Viewer nodes, render results or opened images.



Fig. 2.2449: Stereo 3D and View menu

**Views Format** When you drag and drop an image into the Image Editor, Blender will open it as a individual images at first. If your image was saved with one of the Stereo 3D formats you can change how Blender should interpret the image by switching the mode to Stereo 3D, turning on Use Multi-View and picking the corresponding stereo method.

## Compositor

The compositor works smoothly with Multi-View. The compositing of a view is completed before the remaining views start to be composited. The pipeline is the same as the single-view workflow, with the difference that you can use Image, Movies or Image Sequences in any of the supported Multi-View formats.

The views to render are defined in the current scene views, in a similar way as you define the composite output resolution in the current scene render panel, regardless of the Image nodes resolutions or RenderLayers from different scenes.

---

### Note: Single-View Images

If the image from an Image Node does not have the view you are trying to render, the image will be treated as a single-view image.

---

**Switch View Node** If you need to treat the views separately you can use the Switch View node to combine the views before an output node.

**Performance** By default when compositing and rendering from the user interface all views are rendered and then composited. During test iterations you can disable all but one view from the Scene Views panel, and re-enable it after you get the final look.



Fig. 2.2450: Views Formats and Stereo 3D

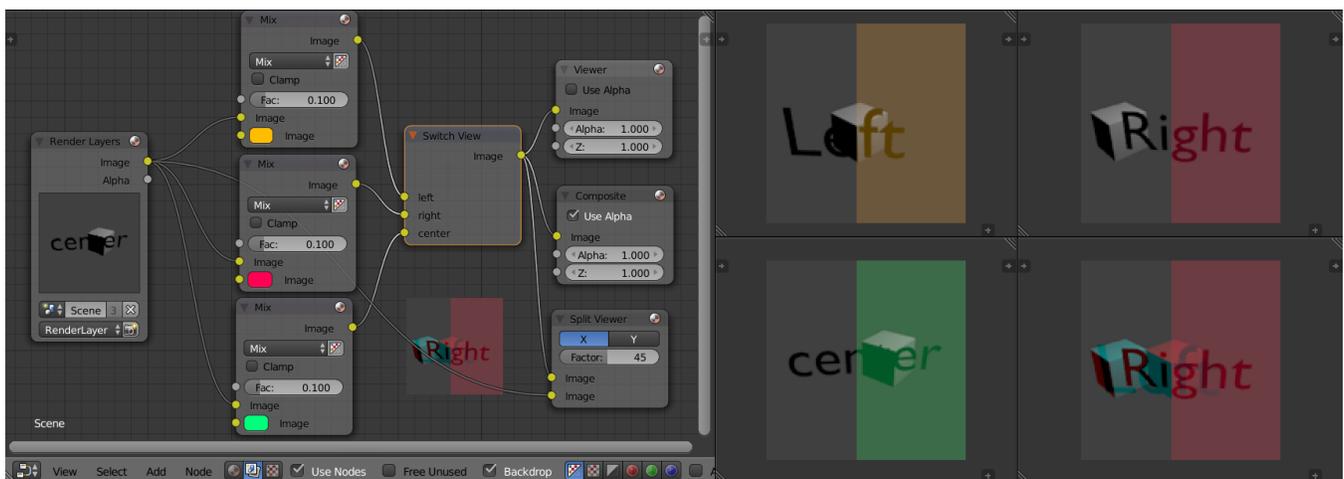


Fig. 2.2451: Compositor, Backdrop and Split Viewer Node

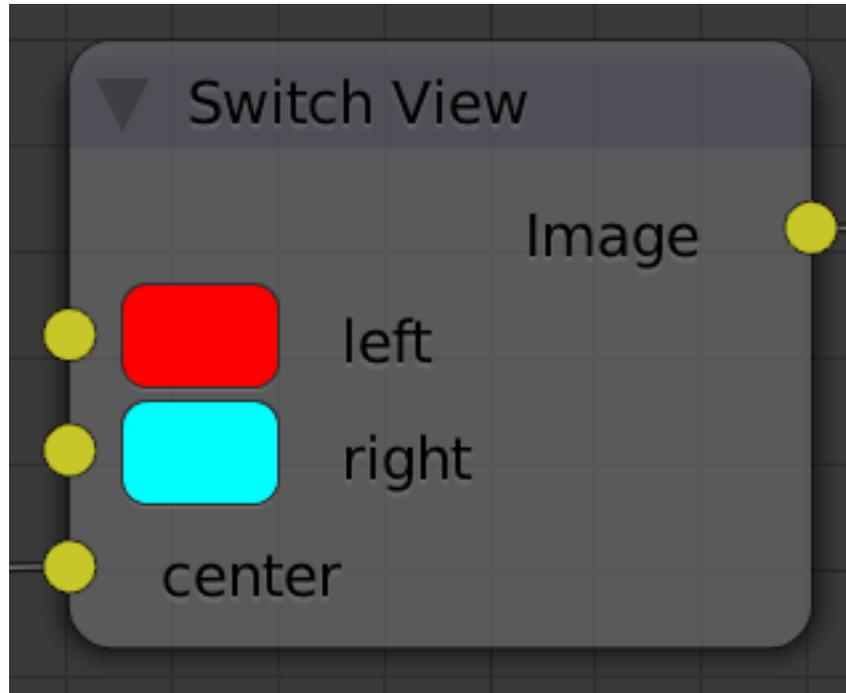


Fig. 2.2452: Switch View Node

## 2.9 Composite Nodes

### 2.9.1 Introduction

Compositing Nodes allow you to assemble and enhance an image (or movie). Using composition nodes, you can glue two pieces of footage together and colorize the whole sequence all at once. You can enhance the colors of a single image or an entire movie clip in a static manner or in a dynamic way that changes over time (as the clip progresses). In this way, you use composition nodes to both assemble video clips together, and enhance them.

**Note:** Term: Image

We use the term *Image* to refer to a single picture, a picture in a numbered sequence of images, or a frame of a movie clip. A node layout processes one image at a time, no matter what kind of input you provide.



Fig. 2.2453: Default Composition Noodle

To process your image, you use nodes to import the image into Blender, change it, optionally merge it with other images, and finally save it.

The example to the right shows the simplest noodle; an input node threads the camera view to an output node so it can be saved.  
 FIXME(Template Unsupported: Doc:2.6/Reference/Nodes/Concepts; { {Doc:2.6/Reference/Nodes/Concepts} } )

## Accessing and Activating Nodes

Access the [Node Editor](#) and enable *Composite Nodes* by clicking on the *Image* icon.



Fig. 2.2454: Node Editor Header with Composite Nodes enabled

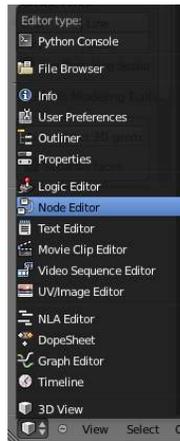


Fig. 2.2455: Select the Node Editor window

To activate nodes for compositing, click the *Use Nodes* checkbox. Blender creates a default starting noodle, consisting of two nodes threaded together.

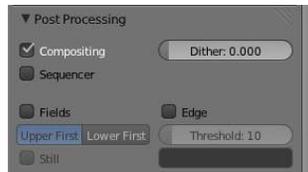


Fig. 2.2456: Use Composition Nodes

To use this mini-map, you must now tell Blender to use the Compositing Node map that has been created, and to composite the image using composition nodes. To do so, switch to the *Render* button area and activate the *Compositing* button located below the *Post Processing* tab. This tells Blender to composite the final image by running it through the composition node map.

You now have your first noodle, a RenderLayer input node threaded to a Composite output node. From here, you can add and connect many [types of compositing nodes](#), in a sort of map layout, to your heart's content (or physical memory constraints, whichever comes first).

## Examples

You can do just about anything with images using nodes.

Raw footage from a foreground actor in front of a blue screen, or a rendered object doing something, can be layered on top of a background. Composite both together, and you have composited footage.

You can change the mood of an image:

- To make an image ‘feel’ colder, a blue tinge is added.
- To convey a flashback or memory, the image may be softened.
- To convey hatred and frustration, add a red tinge or enhance the red. The film ‘Sin City’ is the most extreme example of this I have ever seen.
- A startling event may be sharpened and contrast-enhanced.
- A happy feeling - you guessed it - add yellow (equal parts red and green, no blue) for bright and sunny.
- Dust and airborne dirt is often added as a cloud texture over the image to give a little more realism.

## 2.9.2 Editor

FIXME(Template Unsupported: Doc:2.6/Reference/Nodes/Node Editor; {{Doc:2.6/Reference/Nodes/Node Editor}} )

### Buttons for work with Compositing nodes



Fig. 2.2457: Buttons for work with Compositing nodes

#### Free Unused Button

This button frees up memory space when you have a very complex node map. Recommended.

#### Backdrop

Use the active viewer node output as a backdrop. When enabled, additional settings appear in the Header and the Properties Panel:

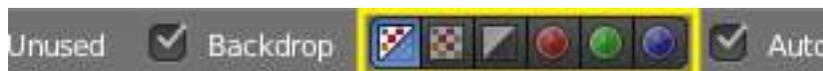


Fig. 2.2458: Backdrop Channels.

**Backdrop Channels** Set the image to be displayed with *Color*, *Color and Alpha*, or just *Alpha*.

**Zoom** Sets how big the backdrop image is.

**Offset** Change the screen space position of the backdrop, or click the *Move* button, or shortcut **Alt**–**MMB** to manually move it.

#### Auto Render

Re-render and composite changed layer when edits to the 3d scene are made.

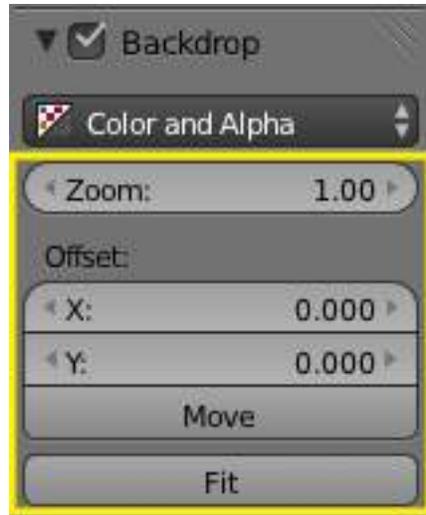


Fig. 2.2459: Options of Zoom and Offset of Backdrop.

### Performance for Compositing Nodes in Node Editor

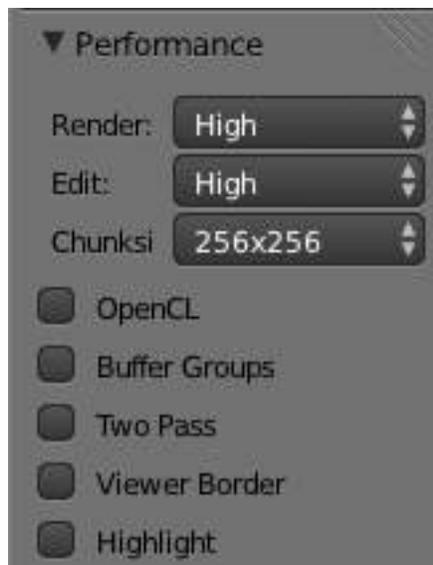


Fig. 2.2460: Performance for Compositing Nodes in Node Editor

**Render** Set quality when rendering in Node Editor.

**Edit** Set quality when editing in Node Editor

**Chunksi** Max size of a tile (smaller values give better distribution of multiple threads, but more overhead).

**OpenCL** Enable GPU calculations when working in Node Editor.

**Buffer Groups** Enable buffering of group nodes.

**Two Pass** Use two pass execution during editing: first calculate fast nodes, second pass calculate all nodes.

**Viewer Border** Use boundaries for viewer nodes and composite backdrop.

**Highlight** Highlight nodes that are being calculated.

### 2.9.3 Node Controls

This page explains the widgets to control a node.

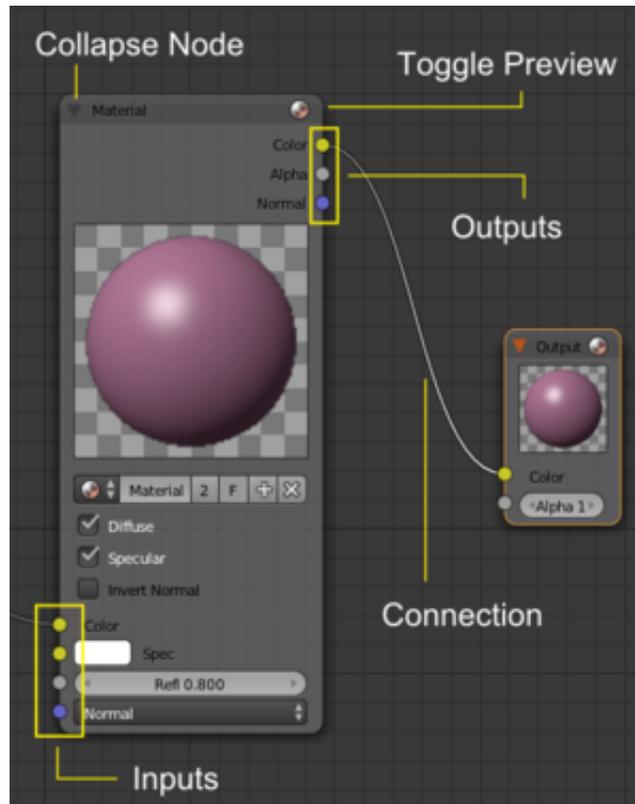


Fig. 2.2461: Nodes main controls

**Title bar** This contains the node's label, along with several different collapse buttons.

**Input and Output sockets** The colored dots on the bottom left and top right of the node are used to make connections between other nodes. See *Sockets* below.

**Image preview** Inside the node there's an area to show the image preview being output by the node or the curves that control the node behavior (for example in a RGB node).

**Buttons and menus** Below the image preview there are buttons and menus to control the node behavior.

**Link** A curved line shows a connection from an output socket to an input socket.

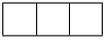
Connections associated with the active node are highlighted for better visibility.

#### Collapsing toggles

At the top of a node there are up to 4 visual controls for the node. Clicking these controls influences how much information the node shows.

**Node toggle** The triangle arrow on the left collapses/expands the node.

**Preview image toggle** The sphere button on the far right of the title bar hides/shows the preview image.



## Resizing the node

Fine resizing of an individual node can also be accomplished by clicking LMB and dragging on the left or right edge of the node.

## Sockets

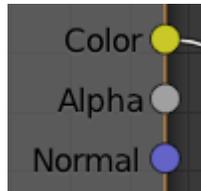


Fig. 2.2468: Node sockets

Each Node will have “sockets” which are small colored circles to which input data and output data can be linked.

The sockets on the left side of a node are *inputs*, while the sockets on the right side are *outputs*.

For your convenience, sockets are *color-coded* according to the type of information they expect to send or receive:

**Yellow** Indicates that **color** information needs to be input or will be output from the node. This may or may not include an alpha channel.

**Gray** Indicates values (**numeric**) information. It can either be a single numerical value or a so-called “value map”. (You can think of a value map as a grayscale-map where the different amount of bright/dark reflects the value for each point.) If a single value is used as an input for a “value map” socket, all points of the map are set to this same value. Common use: Alpha maps and value options for a node.

**Blue** Indicates **vector/coordinate/normal** information.

**Green** Used for **shaders** in *Cycles*

---

**Note:** Usually the socket types will match (e.g. color output to color input), although they do not always have to:

- If a color socket (yellow) is plugged into a value socket (gray), a conversion is done automatically.
  - Colors and vectors can be used interchangeably, because they are both simply sets of three-channel values.
- 

Next to the colored dot you will see the name of that socket. This name usually explains what that socket is meant to be used for, however nothing is stopping you from using it for something else. An example of this is a common technique used in the game industry, where low file size and memory usage are important: The alpha channel of a diffuse texture is used for some other component of the material (e.g. specular intensity), instead of having to include a whole new image.

## 2.9.4 Using Nodes

### Adding Nodes

Nodes are added in two ways to the node editor by using the *Add* menu (`Shift-A`) and picking the desired type of node.

## Arranging Nodes

In general, try to arrange your nodes within the window such that the image flows from left to right, top to bottom. Move a node by clicking on a benign area and drag it around. Nodes can be clicked almost anywhere and dragged about; connections will reshape as a bezier curve as best as possible.

## Connecting nodes

LMB-click and drag a socket: you will see a branch coming out of it: this is called a “thread”.

Keep dragging and connect the thread to an input socket of another node, then release the LMB.

In this case, a copy of each output is routed along a thread. However, only a single thread can be linked to an input socket.

## Disconnecting nodes

To break a link between sockets `Ctrl-LMB`-click in an empty areas near the thread you want to disconnect and drag: you will see a little cutter icon appearing at your mouse pointer. Move it over the thread itself, and release the LMB.

## Duplicating a node

Click LMB or RMB on the desired node, press `Shift-D` and move the mouse away to see the duplicate of the selected node appearing under the mouse pointer.

---

**Note:** When you duplicate a node, the new node will be positioned *exactly* on top of the node that was duplicated. If you leave it there (and it’s quite easy to do so), you can **not** easily tell that there are *two* nodes there! When in doubt, grab a node and move it slightly to see if something’s lurking underneath.

---

## 2.9.5 Node Groups

Both material and composite nodes can be grouped. Grouping nodes can simplify the node network layout in the node editor, making your material or composite ‘noodle’ (node network) easier to work with. Grouping nodes also creates what are called NodeGroups (inside a .blend file) or NodeTrees (when appending).

Conceptually, “grouping” allows you to specify a *set* of nodes that you can treat as though it were “just one node”. You can then re-use it one or more times in this or some other .blend file(s).

As an example: If you have created a material using nodes that you would like to use in another .blend file, you *could* simply append the material from one .blend file to another. However, what if you would like to create a new material, and use a branch from an existing material node network? You could re-create the branch. Or you could append the material to the new .blend file, then cut and paste the branch that you want into the new material. Both of these options work, but are not very efficient when working across different .blend files. A better method of re-use, for either material node branches or composite node networks, would be to create groups of nodes.

Once a group has been defined, it becomes an opaque object; a reusable software component. You can (if you choose) ignore exactly how it is *defined*, and simply use it as many times as you like.

## Grouping Nodes

To create a node group, in the node editor, select the nodes you want to include, then press `Ctrl-G` or `Group → Make Group` (`Shift-A`). A node group will have a green title bar. All of the selected nodes will now be contained within the group node. Default naming for the node group is *NodeGroup*, *NodeGroup.001* etc. There is a name field in the node group you can click

into to change the name of the group. Change the name of the node group to something meaningful. When appending node groups from one .blend file to another, Blender does not make a distinction between material node groups or composite node groups, so it's recommended some naming convention that will allow you to easily distinguish between the two types.

---

**Note: What not to include in your groups (all types of Node editors)**

Remember that the essential idea is that a group should be an easily-reusable, self-contained software component. Material node groups should **not include**:

**Input nodes** if you include a source node in your group, you'll end up having the source node appearing *twice*: once inside the group, and once outside the group in the new material node-network.

**Output node** if you include an output node in the group, there won't be an output socket available *from* the group!

---

## Editing Node Groups

With a group node selected, `Tab` expands the node to a window frame, and the individual nodes within it are shown. You can move them around, play with their individual controls, re-thread them internally, etc. just like you can if they were a normal part of your editor window. You will not be able, though, to thread them to a node outside the group; you have to use the external sockets on the side of the group node. To add or remove nodes from the group, you need to ungroup them.

## Ungrouping Nodes

The `Alt-G` command removes the group and places the individual nodes into your editor workspace. No internal connections are lost, and now you can thread internal nodes to other nodes in your workspace.

## Appending Node Groups

Once you have appended a NodeTree to your .blend file, you can make use of it in the node editor by pressing `Shift-A: Add → Group`, then select the appended group. The “control panel” of the Group is the individual controls for the grouped nodes. You can change them by working with the Group node like any other node.

## 2.9.6 Types of Nodes

This section is organized by type of nodes, which are grouped based on similar functions:

### Input Nodes

Input nodes produce information from some source. For instance, an input could be:

- Taken directly from the active camera in a selected scene,
- from a JPG, PNG, etc. file as a static picture,
- a movie clip (such as an image sequence or video), or
- just a color or value.

These nodes generate the information that feed other nodes. As such, they have no input-connectors; only outputs.

## Render Layers Node

### Reference

Panel: Node Editor -> Node Composition

Menu: Shift-A -> Input -> Render Layers

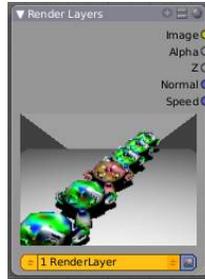


Fig. 2.2469: Render Layers Node

This node is the starting place to getting a picture of your scene into the compositing node map.

This node inputs an image from a scene within your blend file. Select the scene and the active render layer from the yellow selection list at the bottom of the node. Blender uses the active camera for that scene to create an image of the objects specified in the [RenderLayer](#).

The *Image* is input into the map, along with the following data:

- *Alpha* (transparency) mask

Depending on the Renderlayer passes that are enabled, other sockets are available. By default the Z is enabled:

- *Z* depth map (how far away each pixel is from the camera)

The example shows that two other passes are enabled:

- *Normal* vector set (how light bounces off the surface)
- *Speed* vector set (how fast an object is moving from one frame to the next)

Use the re-render button (Small landscape icon - to the right of the Renderlayer name) to re-render the scene and refresh the image and map.

You may recall that a .blend file may contain many scenes. The Renderlayer node can pick up the scene info from any available scene by selecting the scene from the left-hand selector. If that *other* scene also uses the compositor and/or sequencer, you should note that the scene information taken is the raw information (pre-compositing and pre-sequencing). If you wish to use composited information from another scene, you will have to render that scene to a multilayer OpenEXR frameset as an intermediate file store, and then use the Image input node instead.

**Using the Alpha Socket** Using the *Alpha* output socket is crucial in overlaying images on top of one another and letting a background image “show through” the image in front of it.

In a Blender scene, your objects are floating out there in virtual space. While some objects are in front of one another (*Z* depth), there is no ultimate background. Your world settings can give you the illusion of a horizon, but it’s just that: an illusion. Further, some objects are semi-transparent; this is called having an Alpha value. A semi-transparent object allows light (and any background image) to pass through it to the camera. When you render an image, Blender puts out, in addition to a pretty

image, a map of what solid objects actually are there, and where infinity is, and a map of the alpha values for semi-transparent objects. You can see this map by mapping it to a blue screen:

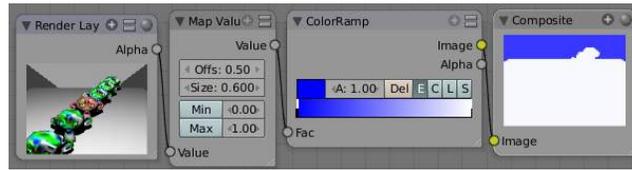


Fig. 2.2470: Viewing the Alpha values

In the little node map above, we have connected the Alpha output socket of the RenderLayer node to a Map Value node (explained later, but basically this node takes a set of values and maps them to something we can use). The Color Ramp node (also explained later in detail) takes each value and maps it to a color that we can see with our eyes. Finally, the output of the Color Ramp is output to a Composite viewer to show you, our dear reader, a picture of the Alpha values. Notice that we have set up the map so that things that are perfectly solid (opaque) are white, and things that are perfectly transparent (or where there is nothing) are blue.

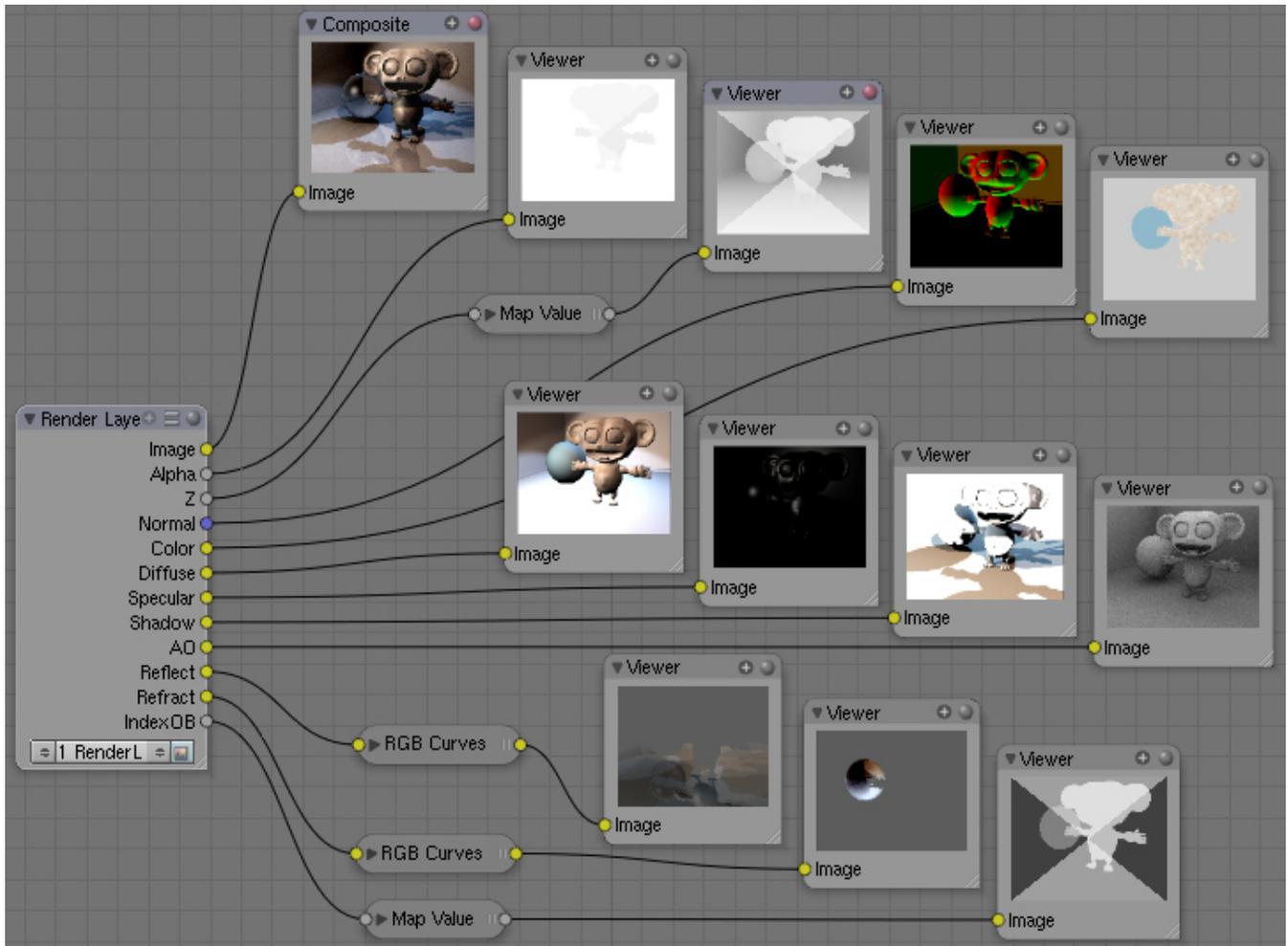
**Optional Sockets** For any of the optional sockets to appear on the node, you MUST have the corresponding pass enabled. In order for the output socket on the RenderLayer node to show, that pass must be enabled in the RenderLayer panel in the Buttons window. For example, in order to be able to have the Shadow socket show up on the RenderLayer input node, you must have the “Shad” button enabled in the Buttons window, Scene Render buttons, Renderlayer panel. See the RenderLayer tab (Buttons window, Output frame, Render Layers tab, Passes selector buttons) for Blender to put out the values corresponding to the socket.

For a simple scene, a monkey and her bouncy ball, the following picture expertly provides a great example of what each pass looks like:

The available sockets are:

- Z: distance away from the camera, in Blender Units
- Normal (Nor): How the color is affected by light coming from the side
- UV: how the image is distorted by the UV mapping
- Speed (Vec): How fast the object is moving, and in what direction
- Color (Col): the RGB values that color the image that you see
- Diffuse: the softening of colors as they diffuse through the materials
- Specular: the degree of shininess added to colors as they shine in the light
- Shadow: shadows cast by objects onto other objects
- AO: how the colors are affected by Ambient Occlusion in the world
- Reflect (Ref): for mirror type objects, the colors they reflect and are thus not part of their basic material
- Refract: how colors are bent by passing through transparent objects
- Radio (Radiosity): colors that are emitted by other objects and cast onto the scene
- IndexOB: a numeric ordinal (index) of each object in the scene, as seen by the camera.

**Using the Z value Socket** Using the Z output socket is crucial in producing realistic images, since items farther away are blurrier (but more on that later).



Imagine a camera hovering over an X-Y plane. When looking through the camera at the plane, Y is up/down and X is left/right, just like when you are looking at a graph. The camera is up in the air though, so it has a Z value from the X-Y plane, and, from the perspective of the camera, the plane, in fact all the objects that the camera can see, have a Z value as a distance that they are away from it. In addition to the pretty colors of an image, a RenderLayer input node also generates a Z value map. This map is a whole bunch of numbers that specify how far away each pixel in the image is away from the camera. You can see this map by translating it into colors, or shades of gray:

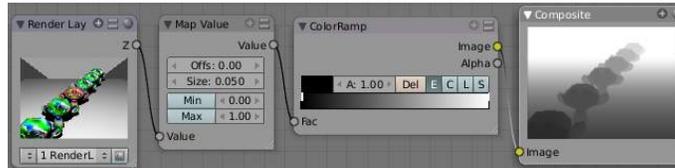


Fig. 2.2471: Viewing the Z values

In the little node map above, we have connected the Z output socket of the RenderLayer node to a Map Value node (explained later). This node takes a set of values and maps them to something we can use. The Color Ramp node (also explained later in detail) takes each value and maps it to a shade of gray that we can see with our eyes. Finally, the output of the colorramp is output to a Composite viewer to show you, our dear reader, a picture of the Z values. Notice that we have set up the Map Value node so that things closer to the camera appear blacker (think: black is 0, less Z means a smaller number) and pixels/items farther away have an increasing Z distance and therefore get whiter. We chose a Size value of 0.05 to see Z values ranging from 0 to 20 (20 is  $1/0.05$ ).

**Using the Speed Socket** Even though things may be animated in our scene, a single image or frame from the animation does not portray any motion; the image from the frame is simply where things are at that particular time. However, from the *Render Layers* node, Blender puts out a vector set that says how particular pixels are moving, or will move, to the next frame. You use this socket to create a [blurring effect](#).

## Image node

### Reference

Panel: [Node Editor](#) -> [Node Composition](#)

Menu: [Shift-A](#) -> [Input](#) -> [Image](#)

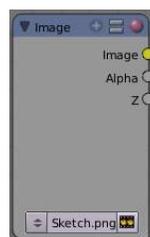


Fig. 2.2472: Image node

The *Image* node injects any image format that is supported by Blender. Besides inputting the actual image, this node can also input *Alpha* and depth (*Z*) values if the image has them. If the image is a MultiLayer format, all saved render passes are input. Use this node to input:

- A single image from a file (such as a JPG picture)
- Part or all of an animation sequence (such as the 30th to 60th frame)
- Part or all of a movie clip (such as an AVI file)
- the image that is currently in the UV/Image Editor (and possibly being painted)
- an image that was loaded in the UV/Image Editor

Animated image sequences or video files can also be used. See [Animations](#) below.

To select an image file or generated image from the UV/Image Editor, click on the small arrow selector button to the left of the name and pick an existing image (e.g. loaded in the UV editor or elsewhere) or click on *LOAD NEW* to select a file from your hard disk via a file-browser. These images can be e.g. previously rendered images, matte paintings, a picture of your cat, whatever. Blender really doesn't care.

If the image is part of a sequence, manually click the Image Type selector to the right of the name, and select *Sequence*. Additional controls will allow you to define how much of the sequence to pull in (see Animations below). If the file is a video file, these controls will automatically appear.

**Image Channels** When the image is loaded, the available channels will be shown as sockets on the node. As a minimum, the Image, Alpha, and Z channels are made available. The picture may or may not have an alpha (transparency) and/or Z (depth) channel, depending on the format. If the image format does not support A and/or Z, default values are supplied (1.0 for A, 0.0 for Z).

#### Alpha/Transparency Channel

- If a transparency channel is detected, the *Alpha* output socket will supply it.
- If it does not have an Alpha channel (e.g. JPG images), Blender will supply one, setting the whole image to completely opaque (an Alpha of 1.00, which will show in a *Viewer* node as white - if connected to the *Image* input socket).

#### Z/depth Channel

- If a Z (depth) channel is detected, the *Z* output socket will supply it.
- If it does not have a Z channel (e.g. JPG or PNG images), Blender will supply one, setting the whole image to be at the camera (a depth of 0.00). To view the Z-depth channel, use the Map Value to ColorRamp noodle given above in the Render Layer input node (see [Using the Z value Socket](#)).

---

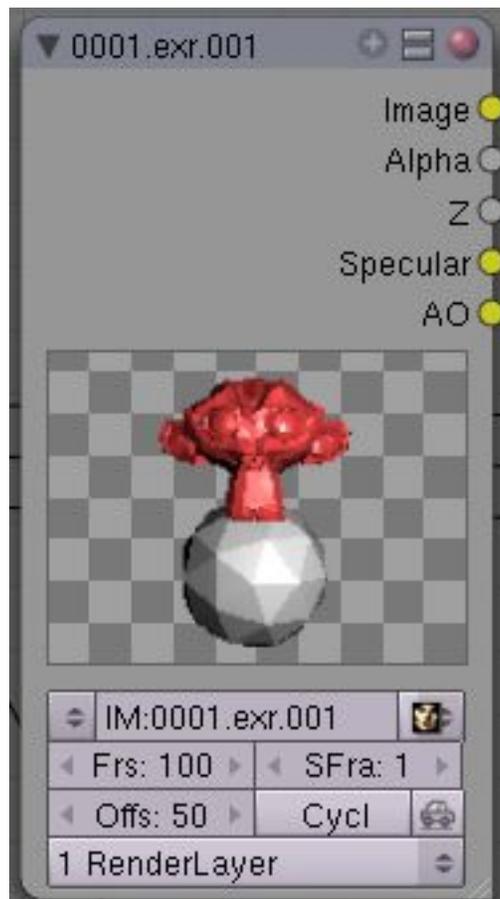
#### Note: Formats

Blender supports many image formats. Currently only the OpenEXR image format stores RGB (color), A (alpha), and Z (depth) buffer information in a single file, if enabled.

---

**Saving/Retrieving Render Passes** Blender can save the individual Render Layers and specific passes in a MultiLayer file format, which is an extension of the OpenEXR format. In this example, we are reading in frames 50 to 100 of a RenderLayer that were generated some time ago. The passes that were saved were the Image, Alpha, Z, Specular and AO passes.

To create a MultiLayer image set when initially rendering, simply disable Do Composite, set your Format to MultiLayer, enable the Render Layer passes you wish to save over the desired frame range, and Animate. Then, in Blender, enable Compositing Nodes and Do Composite, and use the Image input node to read in the EXR file. When you do, you will see each of the saved passes available as sockets for you to use in your compositing noodle.

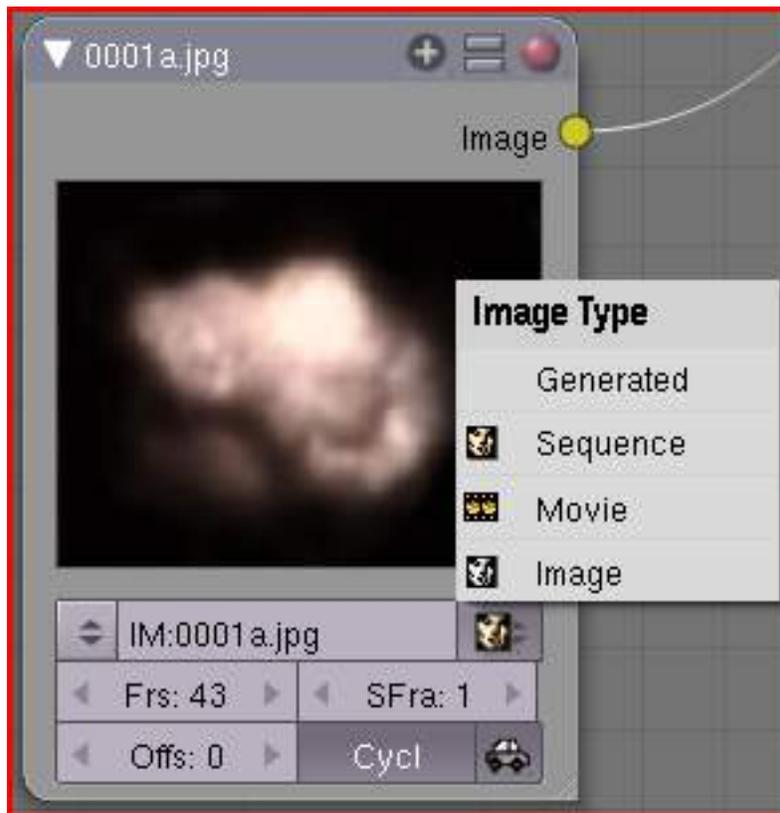


**Image Size** Size matters - Pay attention to image resolution and color depth when mixing and matching images. Aliasing (rough edges), color *flatness*, or distorted images can all be traced to mixing inappropriate resolutions and color depths.

The compositor can mix images with any size, and will only perform operations on pixels where images have an overlap. When nodes receive inputs with differently sized Images, these rules apply:

- The first/top Image input socket defines the output size.
- The composite is centered by default, unless a translation has been assigned to a buffer using a *Translate* node.

So each node in a composite can operate on different sized images, as defined by its inputs. Only the *Composite* output node has a fixed size, as defined by the *Scene buttons* (Format Panel). The *Viewer* node always shows the size from its input, but when not linked (or linked to a value) it shows a small 320x256 pixel image.



**Animations** To use image sequences or movies within your composition, press the face or little film strip button located to the right of the selector. As you click, a pop-up will offer you four choices:

- Generated -
- Sequence - a sequence of frames, each frame in a separate file.
- Movie - a sequence of frames packed into a single `.avi` or `.mov` file
- Image - a single frame or still image in a file

A Movie or Image can be named anything, but a Sequence must have a digit sequence somewhere in its filename, for example `fire0001set.jpg`, `fire0002set.jpg`, `fire0003set.jpg` and so on. The number indicates the frame.

If a Sequence or Movie is selected, an additional set of controls will appear that allows you to select part or all of the sequence. Use these controls to specify which frames, out of the original sequence, that you want to introduce into the animation you are

about to render. You can start at the beginning and only use the beginning, or even pick out a set of frames from the middle of an existing animation.

The *Frs* number button is the number of frames in the sequence that you want to show. For example, if you want to show 2 seconds of the animation, and are running 30 fps, you would put 60 here.

The *SFra* number button sets the start frame of the animation; namely, at what point in the animation that you *are going to render* do you want this sequence to start playing. For example, if you want to introduce this clip ten seconds into the composite output, you would put 300 here (at 30 fps).

The *First* number button sets the first number in the animated sequence name. For example, if your images were called “credits-0001.png”, “credits-0002.png” through “credits-0300.png” and you wanted to start picking up with frame 20, you’d put 20 here.

To have the movie/sequence start over and repeat when it is done, press the *Cyclic* button. For example, if you were compositing a fan into a room, and the fan animation lasted 30 frames, the animation would start over at frame 31, 61, 91, and so on, continuously looping. As you scrub from frame to frame, to see the actual video frame used for the current frame of animation, press the auto button to the right of the *Cyclic* button.

**Generated Images** Using the Nodes to modify a painting in progress in the UV/Image window Blender features **Texture Paint** which works in the UV/Image Editor, that allows you to paint on the fly, and the image is kept in memory or saved. If sync lock is enabled (the lock icon in the header), changes are broadcast throughout Blender as soon as you lift the mouse button. One of the places that the image can go is to the Image Input node. The example shows a painting session going on in the right-hand UV/Image Editor window for the painting “Untitled”. Create this image via Image?New in the UV/Image Editor. Refer to the texture paint section of the user manual for more info on using Texture Paint.

In the left-hand window, the Image input node was used to select that “Untitled” image. Notice that the Image type icon is blank, indicating that it is pulling in a Generated image. That image is colorized by the noodle, with the result used as a backdrop in the Node Editor Window.

Using this setup and the Generated Image type is like painting and post-processing as you continue painting. Changes to either the painting or the post-pro noodle are dynamic and real-time.

**Notes No Frame Stretching or Compression:** If the input animation (avi or frame set) was encoded at a frame rate that is *different* from your current settings, the resultant animation will appear to run faster or slower. Blender Nodes do not adjust input video frame rates. Use the scale control inside the **Video Sequence Editor** to stretch or compress video to the desired speed, and input it here. You can incorporate “Slow-Mo” into your video. To do so, *ANIM* ate a video segment at 60 frames per second, and input it via this node, using Render settings that have an animation frame rate of the normal 30 fps; the resulting video will be played at half speed. Do the opposite to mimic Flash running around at hyperspeed.

AVI (Audio Video Interlaced) files are encoded and often compressed using a routine called a *Codec*. You must have a codec installed on your machine and available to Blender that understands and is able to read the file, in order for Blender to be able to de-code and extract frames from the file. If you get the error message **FFMPEG or unsupported video format** when trying to load the file, you need to get a Codec that understands the video file. Contact the author of the file and find out how it was encoded. An outside package, such as VirtualDub, might help you track this information down. Codecs are supplied by video device manufacturers, Microsoft, DivX, and Xvid, among others, and can often be downloaded from their web sites for free.

**Splicing Video Sequences using Nodes** The above animation controls, coupled with a little mixing, is all you need to splice video sequences together. There are many kinds of splices:

- Cut Splice - literally the ends of the footage are just stuck together
- Fade In - The scene fades in, usually from black
- Fade Out - The scene fades out, usually to black
- Mix - Toward the end of one scene, the images from the next scene meld in as the first scene fades

- Winking and Blinking - fading one cut out while the other fades in, partially or totally through black
- Bumps and Wipes - one cut bumps the other one out of frame, or wipes over it (like from the top left corner down)

**Cut Splicing using Nodes** In the example noodle below, we have two pieces of footage that we want to cut splice together.

- Magic Monkey - named 0001.png through 0030.png
- Credits - named credits0001.png through credits0030.png

The editor has reviewed the Credits and thought the first two frames could be thrown away (onto the cutting room floor, as they say) along with the last 8, leaving 20 frames from the total shot. Not shown in this image, but crucial, is that in the Output panel, we set our render output filename to “Monkey-Credits-”, and our Animation start and end frames to 1 and 50 (30 from the Monkey, 20 from the credits). Notice the Time node; it tells the Mix node to use the top image until frame 30, and then, at frame 31, changes the Mix factor to 1, which means to use the bottom set of images.

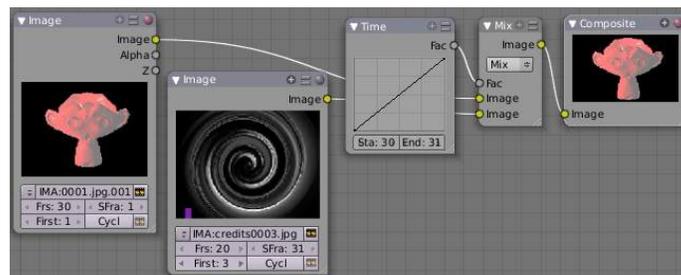


Fig. 2.2473: Cut Splice using Nodes

Upon pressing the ANIM button, Blender will composite the animation. If you specified an image format for output, for example, PNG, Blender will create 50 files, named “Monkey-Credits-0001.png” through “Monkey-Credits-0050.png”. If you specified a movie format as output, such as AVI-JPEG, then Blender will create only one file, “Monkey-Credits-.avi”, containing all 50 frames.

Use cut scenes for rapid-fire transition, conveying a sense of energy and excitement, and to pack in a lot of action in a short time. Try to avoid cutting from a dark scene to a light one, because it’s hard on the eyes. It is very emotionally contrasting, and sometimes humorous and ironic, to cut from a very active actor in one scene to a very still actor in another scene, a la old Road Runner and Coyote scenes.

**Fade Splicing using Nodes** In the previous topic, we saw how to cut from one sequence to another. To fade in or out, we simply replace one set of images with a flat color, and expand the Time frame for the splice. In the image below, beginning at frame 20, we start fading **out** to cyan:

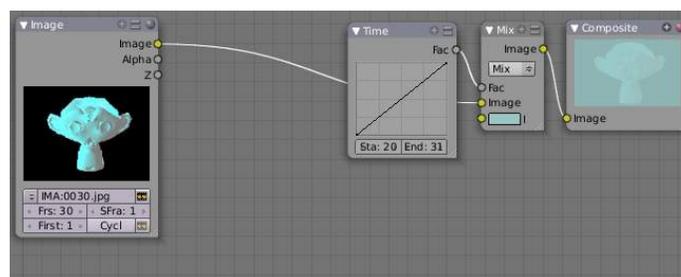


Fig. 2.2474: Fading Out using Nodes

Cyan was chosen because that is the color of the Monkey at that time, but you can just as easily choose any color. The image below shows frame 30, when we have almost faded completely.

To fade **in**, change the Mix node and plug the image sequence into the bottom socket, and specify a flat color for the top socket.

**Mix Splice using Nodes** To mix, or crossover, from one scene to the next, start feeding the second scene in while the first is mixing out. The noodle below shows frame 25 of a mix crossover special effect to transition from one scene to the next, beginning at frame 20 with the transition completed by frame 30. Action continues in the first scene as it fades out and is mixed with action that starts in the second scene.

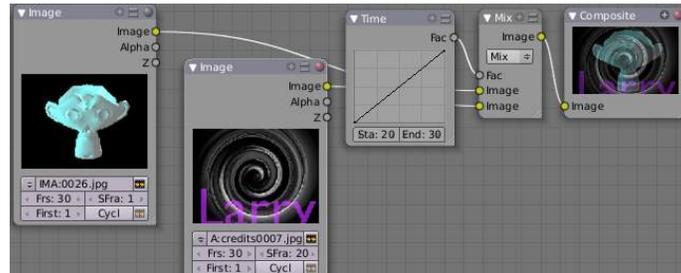


Fig. 2.2475: Mix Splice using Nodes

Use this effect to convey similarities between the two scenes. For example, Scene 1 is the robber walking down the street, ending with the camera focusing in on his feet. Scene 2 is a cop walking down the street after him, starting with his feet and working its way up to reveal that the cop is following the robber.

**Wink Splice using Nodes** A Wink is just like blinking your eyes; one scene fades to black and the other fades in. To use Blender to get this effect, build on the Cut and Fade splices discussed above to yield:

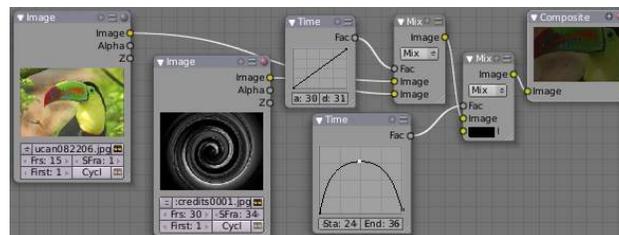


Fig. 2.2476: A Wink using Nodes

In the above example, showing frame 27, we have adjusted some parameters to show you the power of Blender and how to use its Nodes to achieve just the blended crossover effect you desire:

- **Postfeed:** Even though there were only 15 frames of animation in the Toucan strip, the cutover (top Time node) does not occur until frame 30. Blender continues to put out the last frame of an animation, *automatically extending it for you*, for frames out of the strip's range.
- **Prefeed:** Even though the swirl does not start playing until frame 34, Blender supplies the first frame of it for Frames 31 through 33. In fact, it supplies this image all the way back to frame 1.
- **Partial Fade:** Notice the second 'wink' Time node. Like a real wink, it does not totally fade to black; only about 75%. When transitioning between scenes where you want some visual carryover, use this effect because there is not a break in perceptual sequence.

---

#### Note: Multiple Feeds

The above examples call out two feeds, but by replicating the Input, Time and Mix nodes, you can have multiple feeds at any one time; just set the Time node to tell the Mixer when to cut over to using it.

---

## Movie Clip

TODO - see: <https://developer.blender.org/T43469>

## Mask

TODO - see: <https://developer.blender.org/T43469>

## RGB node

---

## Reference

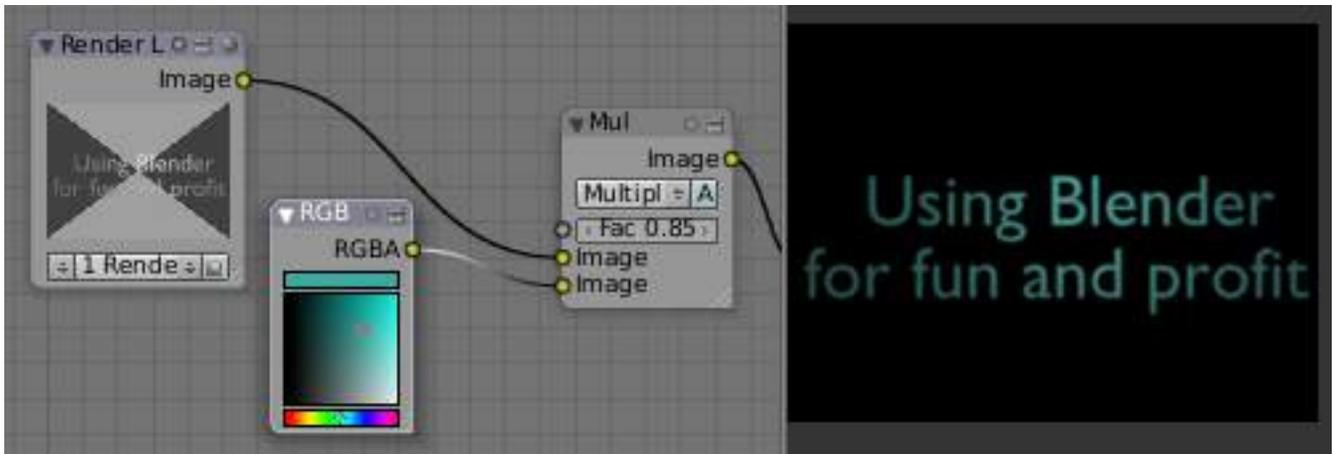
Panel: Node Editor → Node Composition

Menu: Shift-A → Input → RGB

---

The RGB node has no inputs. It just outputs the Color currently selected in its controls section; a sample of it is shown in the top box. In the example to the right, a gray color with a tinge of red is selected.

To change the brightness and saturation of the color, LMB click anywhere within the square gradient. The current saturation is shown as a little circle within the gradient. To change the color itself, click anywhere along the rainbow Color Ramp.



**Example** In this example, our corporate color is teal, but the bozo who made the presentation forgot. So, we multiply his lame black and white image with our corporate color to save him from embarrassment in front of the boss when he gives his boring presentation.

## Value node

---

## Reference

Panel: Node Editor → Node Composition

Menu: Shift-A -> Input -> Value

The Value node has no inputs; it just outputs a numerical value (floating point spanning 0.00 to 1.00) currently entered in the NumButton displayed in its controls selection.

Use this node to supply a constant, fixed value to other nodes' value or factor input sockets.

## Texture Node

### Reference

Panel: Node Editor -> Node Composition

Menu: Shift-A -> Input -> Texture

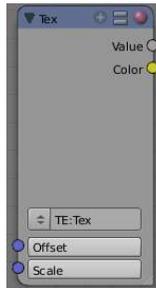


Fig. 2.2477: Texture node

The *Texture* node makes 3D textures available to the compositor.

The Texture node makes 3D textures available to the compositor. A texture, from the list of textures available in the current blend file, is selected and introduced through the value and/or color socket.

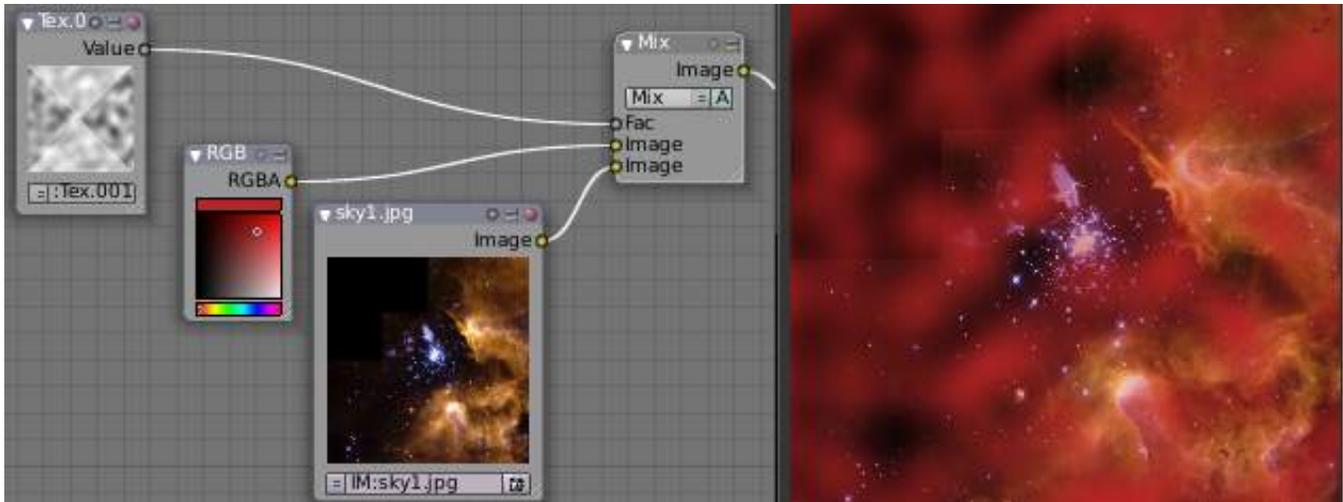
**Note:** Please read up on the Blender Library system for help on importing and linking to textures in other blender files.

**Note:** You cannot edit the textures themselves in the node window. To use this node, create and edit the texture in the normal texture buttons, then select the texture from the menu button on the node.

You can change the *Offset* and a *Scale* (which is called Offs XYZ and Size XYZ in the Materials Texture Map Input panel) for the texture by clicking on the label and setting the sliders, thus affecting how the texture is applied to the image. For animation, note that this is a vector input socket, because the XYZ values are needed.

Texture nodes can output a straight black-and-white *Value* image (don't mistake this for alpha) and an image (*Color*).

**Example** In the example above, we want to simulate some red plasma gas out there in space. So, we fog up an image taken from the Hubble telescope of Orion and take the ever-so-useful Cloud texture and use it to mix in red with the image.



### Bokeh Image

Bokeh Image generates a special input image for use with the [Bokeh Blur](#) filter node.

Bokeh Image is designed to create a reference image which simulates optical parameters such as aperture shape and lens distortions which have important impacts on bokeh in real cameras.

The first three settings simulate the aperture of the camera. Flaps sets an integer number of blades for the cameras iris diaphragm. Angle gives these blades an angular offset relative to the image plane and Rounding sets the curvature of the blades with a 0 being straight and 1 bringing them to a perfect circle.

Catadioptric provides a type of distortion found in mirror lenses and some telescopes. This can be useful to produce a ‘busy’ bokeh.

Lens Shift introduces chromatic aberration into the blur such as would be caused by a tilt-shift lens.

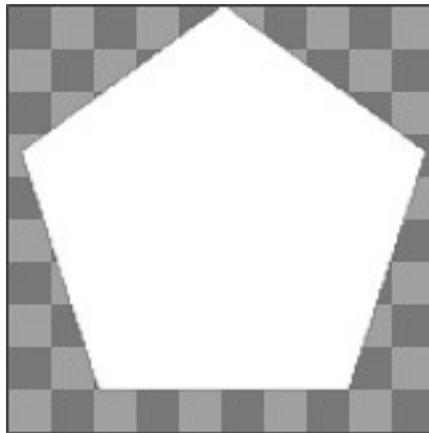


Fig. 2.2478: Example of a bokeh image with 5 flaps.

### Time node

### Reference

Panel: Node Editor -> Node Composition

Menu: Shift-A -> Input -> Time

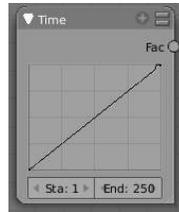


Fig. 2.2479: Time node

The Time node generates a *fac* tor value (from 0.00 to 1.00) (that changes according to the curve drawn) as time progresses through your movie (frames).

The *Start* and *End* NumButtons specify the range of time the values should be output along, and this range becomes the X-axis of the graph. The curve defines the Y-value and hence the factor that is output. In the example to the right, since the timespan is 250 frames and the line is straight from corner to corner, 0.50 would be output at frame 125, and 0.75 will be output at frame 187.

**Note:** Note on output values

The [Map Value](#) node can be used to map the output to a more appropriate value. With some time curves, it is possible that the Time node may output a number larger than one or less than zero. To be safe, use the Min/Max clamping function of the Map Value node to limit output.

You can reverse time (unfortunately, only in Blender and not in the real world) by specifying a Start frame greater than the End frame. The net effect of doing so is to flip the curve around. Warning: doing so is easily overlooked in your node map and can be very confusing (like meeting your mother when she was/is your age in “Back to the Future”).

**Note:** Time is Relative

In Blender, time is measured in frames. The actual duration of a time span depends on how fast those frames whiz by (frame rate). You set the frame rate in your animation settings ([Scene Context](#)). Common settings range from 5 seconds per frame for slideshows (0.2 fps), to 30 fps for US movies.

**Time Node Examples** In the picture below, over the course of a second of time (30 frames), the following time controls are made:

Common uses for this include a “fade to black”, wherein the accelerate time curve (typically exponentially-shaped) feeds a mix value that mixes a constant black color in, so that the blackness accelerates and eventually darkens the image to total black. Other good uses include an increasing soften (blur-out or -in) effect, or *fade-in* a background or foreground, instead of just jumping things into or out of the scene.

You can even imagine hooking up one blur to a background renderlayer, another inverted blur to a foreground renderlayer, and time-feeding both. This node group would simulate someone focusing the camera lens.

**Examples and suggestions** As your imagination runs wild, consider a few ideas that came to me just now on my couch: mixing a clouds texture with a time input to fog up a piece of glass or show spray paint building up on a wall. Consider mixing red and the soften with time (decreasing output) to show what someone sees when waking up from a hard hit on the head. Mix HSV input with a starfield image with time (decreasing output) to show what we might see someday as we accelerate our starship and experience red-shift.

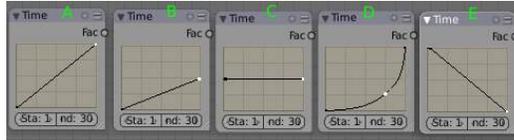


Fig. 2.2480: See:

1. No Effect
2. Slow Down
3. Freeze
4. Accelerate
5. Reverse

As a user, you should know that we have arrived at the point where there are many ways to do the same thing in Blender. For example, an old way to make a slide show using Blender, you created multiple image textures, one image for each slide, and assigned them as texture channels to the material for the screen, then created a screen (plane) that filled the camera view. Using a material ipo, you would adjust the Color influence of each channel at different frames, fading one in as the previous slide faded out. Whew! Rearranging slide and changing the timing was clunky but doable by moving the IPO keys. The *Node* way is to create an image input, one for each slide image. Using the Image input and Time nodes connected to an AlphaOver mixer is much simpler, clearer, and easier to maintain.

### Track Position

TODO - see: <https://developer.blender.org/T43469>

### Output Nodes

These nodes are used to output the composited result in some way.

### Composite

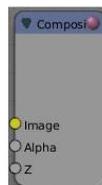


Fig. 2.2481: Composite node

The Composite node is where the actual output from the compositor is connected to the renderer. Connecting a node to the *Composite* node will output the result of that node's full tree to the Renderer; leaving this node unconnected will result in a blank image. This node is updated after each render, but also if you change things in your node-tree (provided at least one finished input node is connected).

You can connect three channels: the actual RGBA image, the Alpha image, and the Z (depth) image. You should only have one Composite node in your map so that only one final image is rendered when the *Compositing* button is pressed on the Render Options Post-Processing panel. Otherwise, unpredictable results may occur.

**Note:** If multiple Composite nodes are added, only the active one (last selected, indicated with a slightly darker header) will be used.

**Saving your Composite Image** The RENDER button renders a single frame or image. Save your image using F3 or the *File*→*Save Image* menu. The image will be saved using the image format settings on the Render panel.

To save a sequence of images, for example, if you input a movie clip or used a Time node with each frame in its own file, use the ANIM button and its settings. If you might want to later overlay them, be sure to use an image format that supports an Alpha channel (such as PNG). If you might want to later arrange them front to back or create a depth of field effect, use a format that supports a Z-depth channel (such as EXR).

To save a composition as a movie clip (all frames in a single file), use an AVI or Quicktime format, and use the ANIM button and its settings.

## Viewer



Fig. 2.2482: Viewer node

The *Viewer* node is a temporary, in-process viewer. Plug it in wherever you would like to see an image or value-map in your node-tree.

LMB click on the image to update it, if it wasn't done automatically. You can use as many of these as you would like. It is possible to automatically plug a Viewer node to any other node by pressing Shift-Ctrl-LMB on it.

**Note:** It is possible to add multiple Viewer nodes, though only the active one (last selected, indicated with a slightly darker header) will be shown on the backdrop or in the UV/Image editor.

**Border Compositing** A border for the viewer node can be defined using Ctrl-B and selecting a rectangular area.

This border is used to define the area of interest of the viewer node which restricts compositing to this area. Used for faster previews by skipping compositing outside of the defined area of interest. This is only a preview option, final compositing during a render ignores this border.

Use Ctrl-Alt-B to discard the defined border and see a full preview.

**Tile order** The tile order can be defined for the backdrop image, using the *Tile order* field in the properties of the viewer node (*Properties* panel in *Properties* sidebar, with the viewer node selected):

**Rule of thirds** Calculates tiles around each of the 9 zones defined by the **rule of thirds** (see [Rule of Thirds](#) for more information).

**Bottom up** Tiles are calculated from the bottom up.

**Random** Calculates tiles in a non-specific order.

**Center** Calculates the tiles around a specific center, defined by X and Y fields.

**Using the UV/Image Editor Window** The viewer node allows results to be displayed in the UV/Image Editor. The image is facilitated by selecting *Viewer Node* on the window's header linked image selector. The UV/Image Editor will display the image from the currently selected viewer node.

To save the image being viewed, use *Image* → *Save As Image* (F3) to save the image in a file.

The UV/Image Editor also has three additional options in its header to view Images with or without Alpha, or to view the Alpha or Z itself. Holding LMB in the Image display allows you to sample the values.

### SplitViewer Node

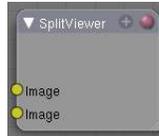


Fig. 2.2483: SplitViewer node

The *SplitViewer* node takes two images and displays one half of each on each side (top socket on the right half, bottom socket input on the left). Use this node for making side-by-side comparisons of two renderings/images, perhaps from different renderlayers or from different scenes. When transitioning between scenes, you want to be sure the stop action is seamless; use this node to compare the end of one scene with the beginning of another to ensure they align.

### File Output Node



Fig. 2.2484: File Output node

This node puts out an RGBA image, in the format selected, for each frame range specified, to the filename entered, as part of a frameset sequence. This means that the name of the file will be the name you enter plus a numeric frame number, plus the filename extension (based on format). Based on the format you choose, various quality/compression options may be shown.

To support subsequent arrangement and layering of images, the node can supply a Z-depth map. However, please note that only the OpenEXR image formats save the Z information.

The image is saved whenever Blender feels like it. Just kidding; whenever you press the Render button, the current frame image is saved. When you press the Anim button, the frameset sequence (specified in the Start and End frame) is saved.

This node saves you from doing (or forgetting to do) the Save Image after a render; the image is saved automatically for you. In addition, since this node can be hooked in anywhere in the noodle, you can save intermediate images automatically. Neat, huh?

---

#### **Note:** Filespecs

As with all filename entries, use // at the beginning of the field to shorthand reference the current directory of the .blend file. You can also use the .. breadcrumb to go up a directory.

---

## Levels Node

The Levels Node takes an image as an input, and can output a 1D value based on the levels of an image. It can read the input's *Combined RGB*, *Red*, *Green*, *Blue*, or *Luminance* channels.

It can output a *Mean* value, or average of values, or a *Standard deviation*, which measures the diversity of values.

## Color Nodes

These nodes adjust the image's colors, for example increasing the contrast, making it warmer, overlaying another image, etc.

## Mix Node



This node mixes a base image (threaded to the top socket) together with a second image (bottom socket) by working on the individual and corresponding pixels in the two images or surfaces. The way the output image is produced is selected in the drop-down menu. The size (output resolution) of the image produced by the mix node is the size of the base image. The alpha and Z channels are mixed as well.

### See also:

*Color Blend Modes* for details on each blending mode.

### Note: Color Channels

There are two ways to express the channels that are combined to result in a color: RGB or HSV. RGB stands for the Red/Green/Blue pixel format, and HSV stands for the Hue/Saturation/Value pixel format.

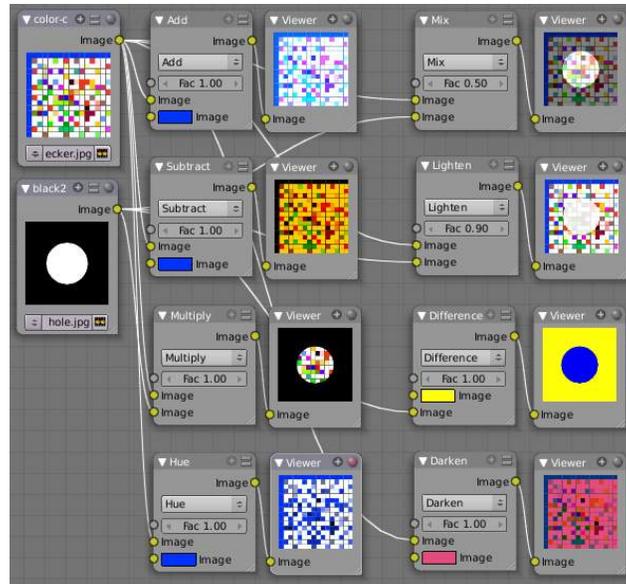
**Alpha** Click the *Alpha* button to make the mix node use the Alpha (transparency) values of the second (bottom) node. If enabled, the resulting image will have an Alpha channel that reflects both images' channels. Otherwise, (when not enabled, light green) the output image will mix the colors by considering what effect the Alpha channel has of the base (top input socket) image. The Alpha channel of the output image is not affected.

**Fac** The amount of mixing of the bottom socket is selected by the Factor input field (*Fac*:). A factor of zero does not use the bottom socket, whereas a value of 1.0 makes full use. In Mix mode, 0.5 is an even mix between the two, but in Add mode, 0.5 means that only half of the second socket's influence will be applied.

**Examples** Below are samples of common mix modes and uses, mixing a color or checker with a mask.

Some explanation of the mixing methods above might help you use the Mix node effectively:

- *Add* - adding blue to blue keeps it blue, but adding blue to red makes purple. White already has a full amount of blue, so it stays white. Use this to shift a color of an image. Adding a blue tinge makes the image feel colder.



- *Subtract* : Taking Blue away from white leaves Red and Green, which combined make Yellow (and you never thought you'd need a color wheel again, eh?). Taking Blue away from Purple leaves Red. Use this to de-saturate an image. Taking away yellow makes an image bluer and more depressing.
- *Multiply* : Black (0.00) times anything leaves black. Anything times White (1.00) is itself. Use this to mask out garbage, or to colorize a black-and-white image.
- *Hue* : Shows you how much of a color is in an image, ignoring all colors except what is selected: makes a monochrome picture (style 'Black & Hue').
- *Mix* : Combines the two images, averaging the two.
- *Lighten* : Like bleach, makes your whites whiter. Use with a mask to lighten up a little.
- *Difference* : Kinda cute in that it takes out a color. The color needed to turn Yellow into White is Blue. Use this to compare two verrry similar images to see what had been done to one to make it the other; sorta like a change log for images. You can use this to see a watermark (see [Using Mix to Watermark images](#)) you have placed in an image for theft detection.
- *Darken*, with the colors set here, is like looking at the world through rose-colored glasses (sorry, I just couldn't resist).

**Contrast Enhancement using Mix** Here is a small map showing the effects of two other common uses for the RGB Curve: **Darken** and **Contrast Enhancement**. You can see the effect each curve has independently, and the combined effect when they are **mixed** equally.

As you can hopefully see, our original magic monkey was overexposed by too much light. To cure an overexposure, you must both darken the image and enhance the contrast. Other paint programs usually provide a slider type of control, but Blender, ah the fantastic Blender, provides a user-definable curve to provide precise control.

In the top RGB curve, *Darken*, only the right side of the curve was lowered; thus, any X input along the bottom results in a geometrically less Y output. The *Enhance Contrast* RGB 'S' curve scales the output such that middle values of X change dramatically; namely, the middle brightness scale is expanded, and thus whiter whites and blacker blacks are output. To make this curve, simply click on the curve and a new control point is added. Drag the point around to bend the curve as you wish. The Mix node combines these two effects equally, and Suzanne feels much better. And NOBODY wants a cranky monkey on their hands.

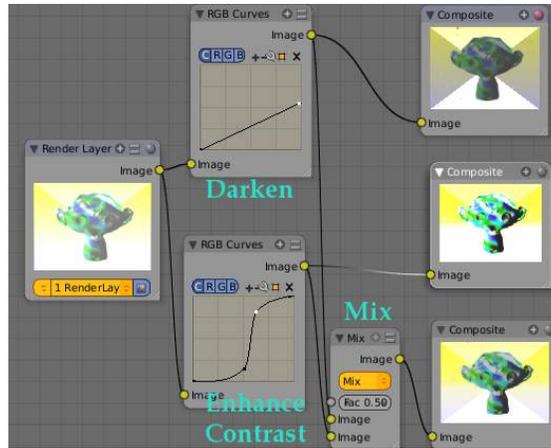


Fig. 2.2485: Example node setup showing “Darken”, “Enhance Contrast” and “Mix” nodes for composition.

**Using Mix to Watermark images** In the old days, a pattern was pressed into the paper mush as it dried, creating a mark that identified who made the paper and where it came from. The mark was barely perceptible except in just the right light. Probably the first form of subliminal advertising. Nowadays, people watermark their images to identify them as personal intellectual property, for subliminal advertising of the author or hosting service, or simply to track their image’s proliferation throughout the web. Blender provides a complete set of tools for you to both encode your watermark and to tell if an image has your watermark.

**Encoding Your Watermark in an Image** First, construct your own personal watermark. You can use your name, a word, or a shape or image not easily replicated. While neutral gray works best using the encoding method suggested, you are free to use other colors or patterns. It can be a single pixel or a whole gradient; it’s up to you. In the example below, we are encoding the watermark in a specific location in the image using the Translate node; this helps later because we only have to look in a specific location for the mark. We then use the RGB to BW node to convert the image to numbers that the Map Value node can use to make the image subliminal. In this case, it reduces the mark to one-tenth of its original intensity. The Add node adds the corresponding pixels, make the ones containing the mark ever-so-slightly brighter.



Fig. 2.2486: Embedding your mark in an Image using a Mark and Specific Position

Of course, if you *want* people to notice your mark, don’t scale it so much, or make it a contrasting color. There are also many other ways, using other mix settings and fancier rigs. Feel free to experiment!

**Note:** Additional uses

You can also use this technique, using settings that result in visible effects, in title sequences to make the words appear to be cast on the water’s surface, or as a special effect to make words appear on the possessed girl’s forearm. yuk.

**Decoding an Image for your Watermark** When you see an image that you think might be yours, use the node map below to compare it to your stock image (pre-watermarked original). In this map, the Mix node is set to Difference, and the Map Value

node amplifies any difference. The result is routed to a viewer, and you can see how the original mark stands out, clear as a bell:

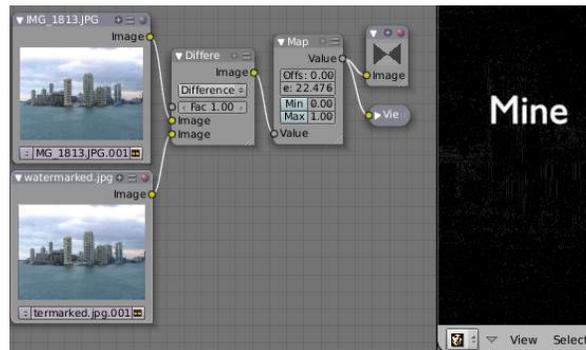


Fig. 2.2487: Checking an image for your watermark

Various image compression algorithms lose some of the original; the difference shows as noise. Experiment with different compression settings and marks to see which works best for you by having the encoding map in one scene, and the decoding map in another. Use them while changing Blender's image format settings, reloading the watermarked image after saving, to get an acceptable result. In the example above, the mark was clearly visible all the way up to JPEG compression of 50%.

**Using Dodge and Burn (History Lesson)** Use the dodge and burn mix methods in combination with a mask to affect only certain areas of the image. In the old darkroom days, when, yes, I actually spent hours in a small stinky room bathed in soft red light, I used a circle cutout taped to a straw to dodge areas of the photo as the exposure was made, casting a shadow on the plate and thus limiting the light to a certain area.

To do the opposite, I would burn in an image by holding a mask over the image. The mask had a hole in it, letting light through and thus 'burning' in the image onto the paper. The same equivalent can be used here by mixing an alpha mask image with your image using a dodge mixer to lighten an area of your photo. Remember that black is zero (no) effect, and white is one (full) effect. And by the way, ya grew to like the smell of the fixer, and with a little soft music in the background and the sound of the running water, it was very relaxing. I kinda miss those dayz.

### AlphaOver Node



Fig. 2.2488: AlphaOver node

Use this node to layer images on top of one another. This node takes two images as input, combines them by a factor, and outputs the image. Connect the Background image to the top input, and the foreground image to the lower input. Where the foreground image pixels have an alpha greater than 0 (namely, have some visibility), the background image will be overlaid.

Use the *Factor* slider to 'merge' the two pictures. A factor less than 1.00 will make the foreground more transparent, allowing the background to bleed through.

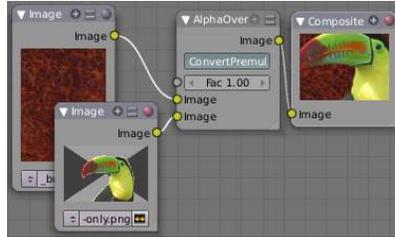


Fig. 2.2489: Assembling a composite Image using AlphaOver

**Examples** In this example, an image of a Toucan is superimposed over a wooden background. Use the PreMultiply button when the foreground image and background images have a combined Alpha that is greater than 1.00; otherwise you will see an unwanted halo effect. The resulting image is a composite of the two source images.

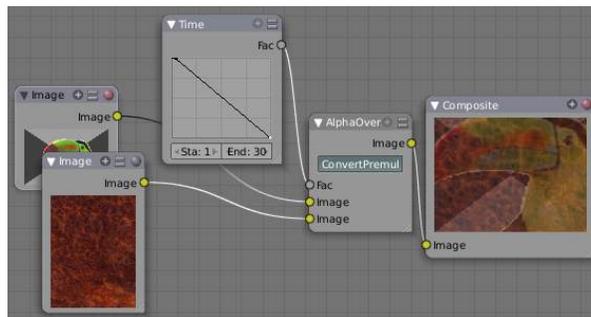


Fig. 2.2490: Animated See-Through/Sheer SFX using AlphaOver - Frame 11

In this example, we use the Factor control to make a sheer cloth or onion-skin effect. You can animate this effect, allowing the observer to 'see-through' walls (or any foreground object) by hooking up a Time node to feed the Factor socket as shown below. In this example, over the course of 30 frames, the Time node makes the AlphaOver node produce a picture that starts with the background wood image, and slowly bleeds through the Toucan. This example shows frame 11 just as the Toucan starts to be revealed.

AlphaOver does not work on the colors of an image, and will not output any image when one of the sockets is unconnected.

**Strange Halos or Outlines** To clarify the pre-multiplied-alpha button: An alpha channel has a value of between 0 and 1. When you make an image transparent (to composite it over another one), you are really multiplying the RGB pixel values by the alpha values (making the image transparent (0) where the alpha is black (0), and opaque (1) where it is white (1)).

So, to composite image A over image B, you get the alpha of image A and multiply it by image A, thus making the image part of A opaque and the rest transparent. You then inverse the alphas of A and multiply image B by it, thus making image B transparent where A is opaque and vice versa. You then add the resultant images and get the final composite.

A pre-multiplied alpha is when the image (RGB) pixels are already multiplied by the alpha channel, therefore the above compositing op doesn't work too well, and you have to hit 'convert pre-mult'. This is only an issue in semi transparent area, and edges usually. The issue normally occurs in Nodes when you have combined, with alpha, two images, and then wish to combine that image with yet another image. The previously combined image was previously multiplied (pre-mult) and needs to be converted as such (hence, *Convert PreMul*).

If you don't pay attention and multiply twice, you will get a white or clear halo around your image where they meet, since your alpha value is being squared or cubed. It also depends on whether or not you have rendered your image as a pre-mult, or straight RGBA image.

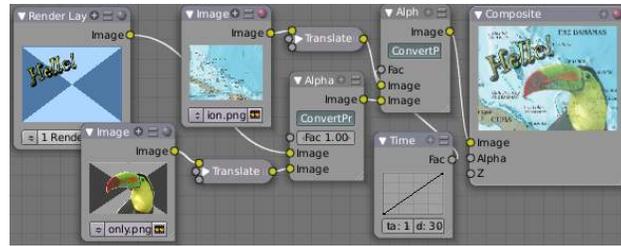
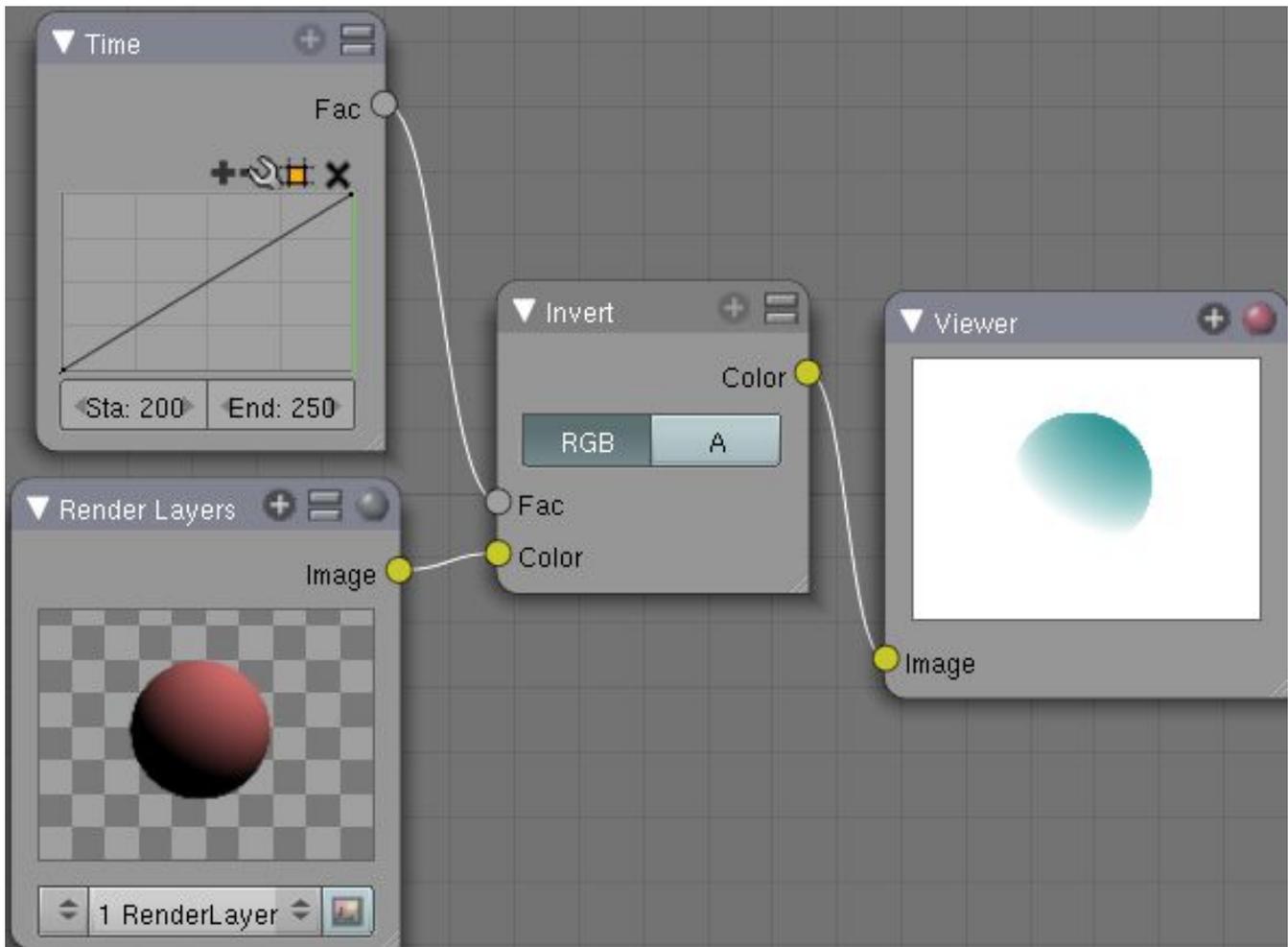


Fig. 2.2491: Layering Images using AlphaOver Premul

## Invert



This handy node inverts the colors in the input image, producing a negative.

### Options

**Factor** Controls the amount of influence the node exerts on the output image

**Color** The input image. In this case, a red sphere on a black transparent background

**RGB** Invert the colors from white. In this example, red inverted is cyan (teal).

A Invert the alpha (transparency) channel as well. Handy for masking.

## RGB Curves Node

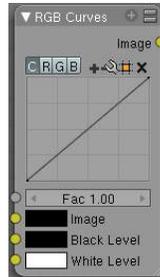


Fig. 2.2492: RGB Curves node

For each color component channel (RGB) or the composite (C), this node allows you to define a bezier curve that varies the input (x-axis) to produce an output value (y-axis). Clicking on one of the *C R G B* components displays the curve for that channel.

### See also:

- Read more about using the *Curve Widget*.

Here are some common curves you can use to achieve desired effects:

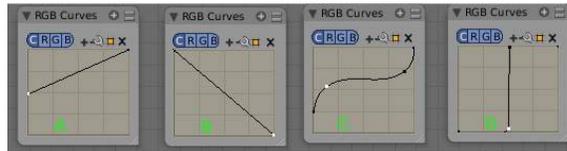


Fig. 2.2493: Identifiers: A) Lighten B) Negative C) Decrease Contrast D) Posterize

## Options

**Fac** How much the node should factor in its settings and affect the output.

**Black Level** Defines the input color that is mapped to black. Default is black, which does not change the image.

**White Level** Defines the input color that is mapped to white. Default is white, which does not change the image.

The levels work exactly like the ones in the image viewer. Input colors are scaled linearly to match black/white levels.

To define the levels, either use LMB on the color patch to bring up the color selection widget or connect some RGBA input to the sockets.

To only affect the value/contrast (not hue) of the output, set the levels to shades of gray. This is equivalent to setting a linear curve for C.

If you set any level to a color with a saturation greater than 0, the output colors will change accordingly, allowing for basic color correction or effects. This is equivalent to setting linear curves for R, G and B.

## Examples

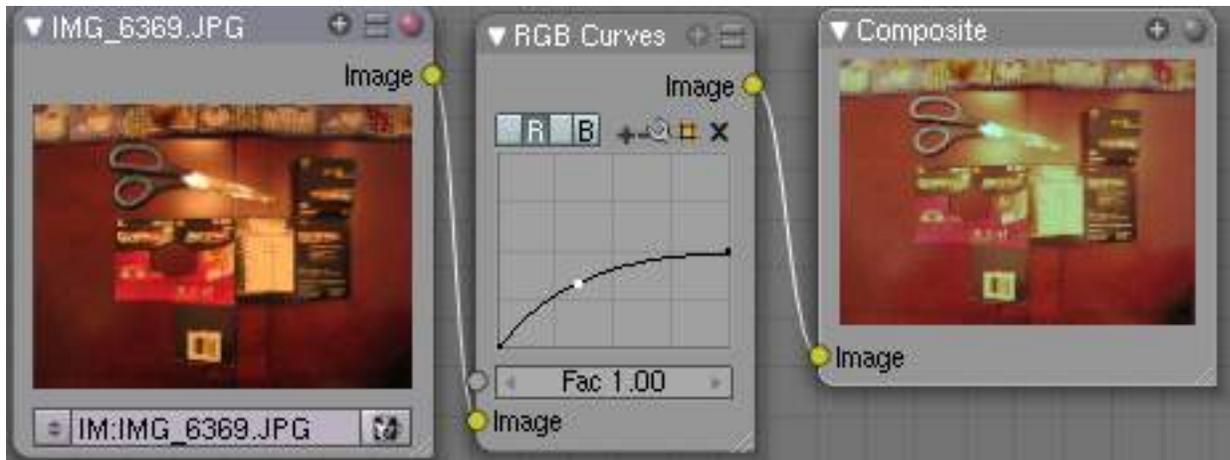


Fig. 2.2494: Color correction with curves

**Color correction using Curves** In this example, the image has way too much red in it, so we run it through an RGB node and reduce the Red channel by about half.

We added a middle dot so we could make the line into a sideways exponential curve. This kind of curve evens out the amount of a color in an image as it reaches saturation. Also, read on for examples of the Darken and Contrast Enhancement curves.

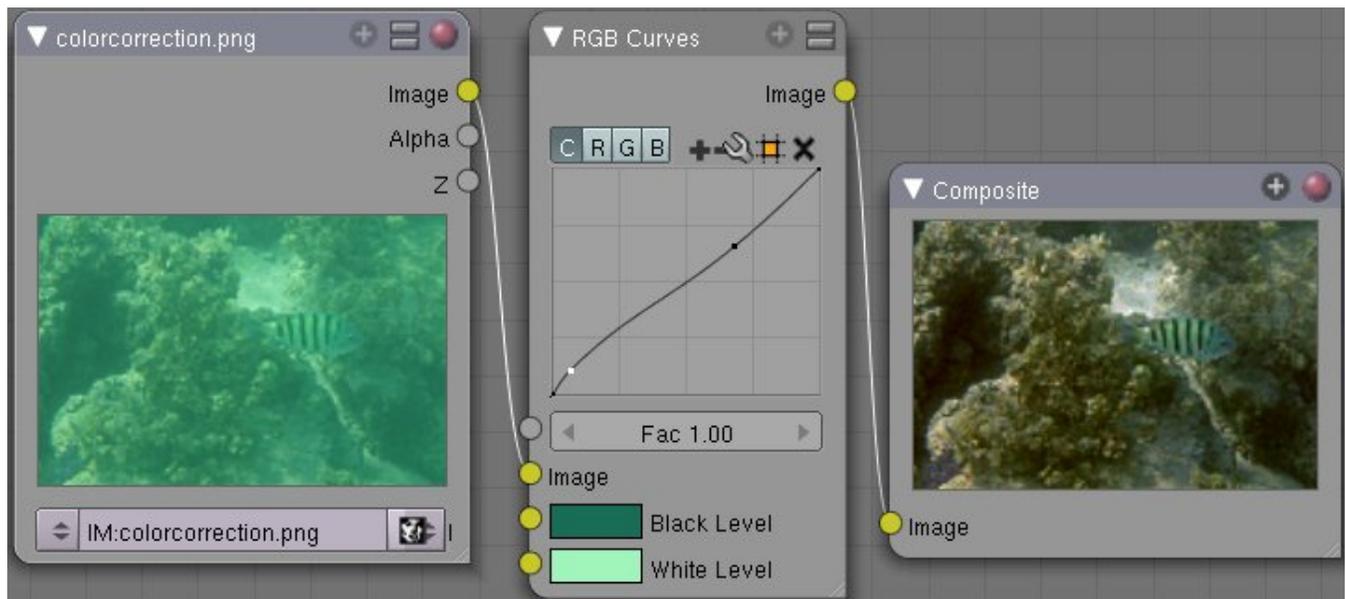


Fig. 2.2495: Color correction with Black/White Levels

**Color correction using Black/White Levels** Manually adjusting the RGB curves for color correction can be difficult. Another option for color correction is to use the Black and White Levels instead, which really might be their main purpose.

In this example, the White Level is set to the color of a bright spot of the sand in the background, and the Black Level to the color in the center of the fish's eye. To do this efficiently it's best to bring up an image viewer window showing the original input image. You can then use the levels' color picker to easily choose the appropriate colors from the input image, zooming in to pixel level if necessary. The result can be fine-tuned with the R,G, and B curves like in the previous example.

The curve for C is used to compensate for the increased contrast that is a side-effect of setting Black and White Levels.

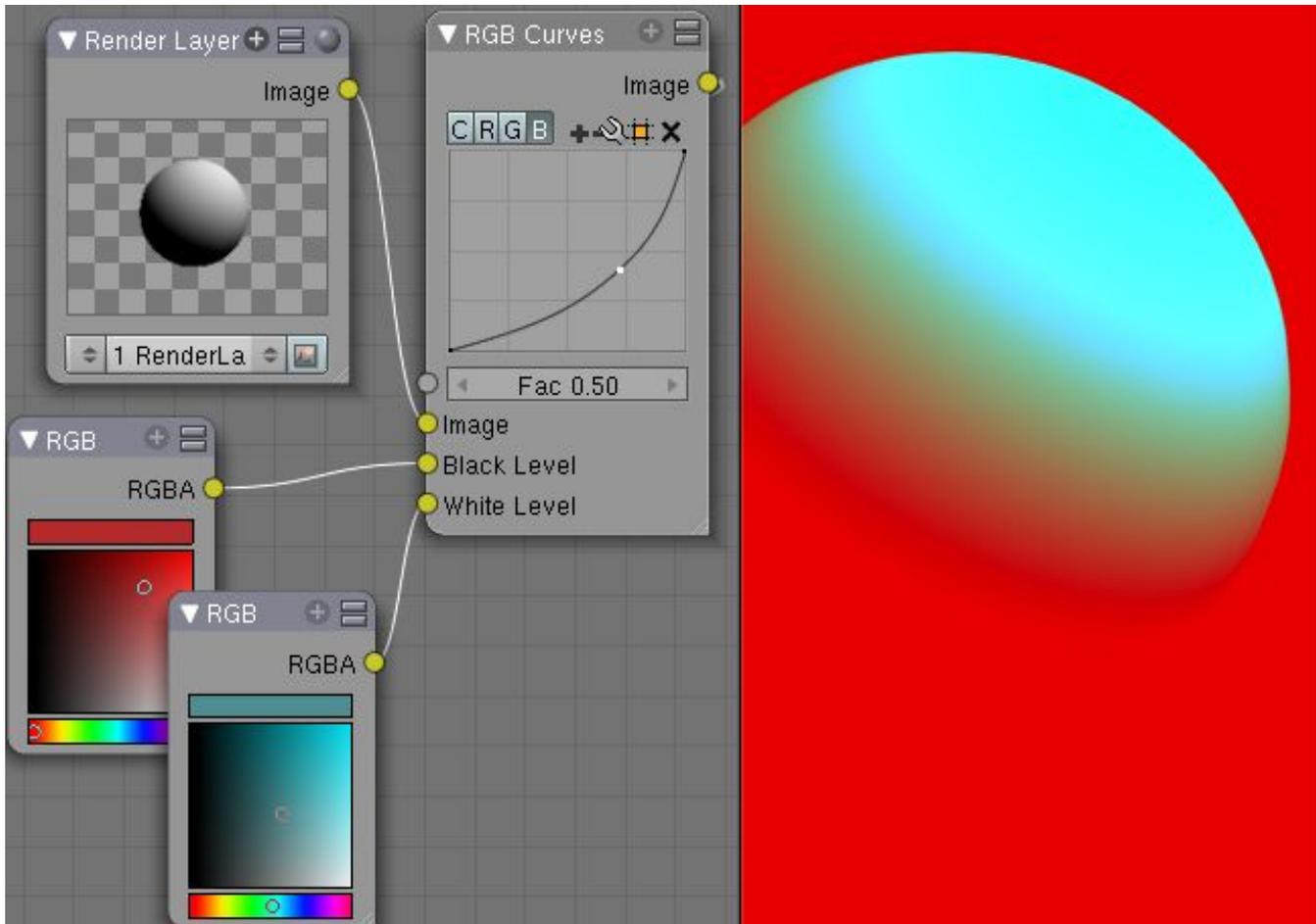


Fig. 2.2496: Changing colors

**Effects** Curves and Black/White Levels can also be used to completely change the colors of an image.

Note that e.g. setting Black Level to red and White Level to blue does not simply substitute black with red and white with blue as the example image might suggest. Levels do color scaling, not substitution, but depending on the settings they can result in the described color substitution.

(What really happens when setting Black Level to pure red and White Level to pure blue is that the red channel gets inverted, green gets reduced to zero and blue remains unchanged.)

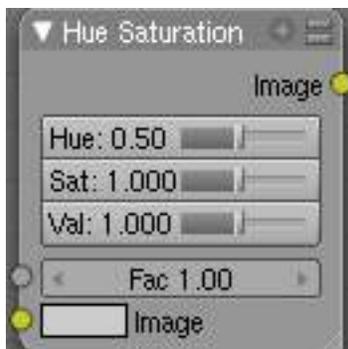
Because of this the results of setting arbitrary Black/White Levels or RGB curves is hard to predict, but can be fun to play with.

### Hue Saturation Node

As an alternative to RGB editing, color can be thought of as a mix of Hues, namely a normalized value along the visible spectrum from infra-red to ultraviolet (the rainbow, remember “Roy G. Biv”). The amount of the color added depends on the saturation of that color; the higher the saturation, the more of that pigment is added. Use the saturation slider of this node to “bring out” the colors of a washed-out image.

This node takes an input image and runs the color of the image (and the light it reflects and radiates) ‘up’ through a factor (0.0-1.0) and applies a saturation of color effect of a hue to the image:

**Hue:** The **Hue** slider specifies how much to shift the hue of the image. Hue 0.5 (in the middle) does not shift the hue or affect the color of the image. As Hue shifts left, the colors shift as more cyan is added; a blue image goes bluer, then greener,



then yellow. A red image goes violet, then purple, blue, and finally teal. Shifting right (increasing Hue from 0.5 to 1.0) introduces reds and greens. A blue image goes purple, plum, red, orange, and then yellow. A red image goes golden, olive, green, and cyan.

**Sat:** **Saturation** affects the amount of pigment in the image. A saturation of 0 actually *removes* hues from the color, resulting in a black-and-white grayscale image. A saturation of 1.0 blends in the hue, and 2.0 doubles the amount of pigment and brings out the colors.

**Val:** **Value** affects the overall amount of the color in the image. Increasing values make an image lighter; decreasing values shift an image darker.

**Fac:** **Factor** determines how much this node affects the image. A factor of 0 means that the input image is not affected by the Hue and Saturation settings. A factor of 1 means they rule, with .5 being a mix.

**Hue/Saturation tips** Some things to keep in mind that might help you use this node better:

**Hues are vice versa.** A blue image, with a Hue setting at either end of the spectrum (0 or 1), is output as yellow (recall that white, minus blue, equals yellow). A yellow image, with a Hue setting at 0 or 1, is blue.

**Hue and Saturation work together.** So, a Hue of .5 keeps the blues the same shade of blue, but the saturation slider can deepen or lighten the intensity of that color.

**Gray & White are neutral hues.** A gray image, where the RGB values are equal, has no hue. Therefore, this node can only affect it with the *Val* slider. This applies for all shades of gray, from black to white; wherever the values are equal.

**Changing the effect over time.** The Hue and Saturation values are set in the node by the slider, but you can feed a Time input into the Factor to bring up (or down) the effect change over time.

---

**Note:** Tinge

This HSV node simply shifts hues that are already there. To colorize a gray image, or to ADD color to an image, use a mix node to add in a static color from an RGB input node with your image.

---

**HSV Example** Here, the image taken by a cheap digital camera in poor lighting at night using a flash (can we do it any worse, eh?) is adjusted by decreasing the Hue (decreasing reds and revealing more blues and greens), decreasing Saturation (common in digital cameras, and evens out contrast) and increasing Value (making it all lighter).

## Color Balance

The Color Balance node can adjust the color and values of an image using two different correction formulas.

The *Lift*, *Gamma*, *Gain* formula uses *Lift*, *Gamma*, and *Gain* calculations to adjust an image. *Lift* increases the value of dark colors, *Gamma* will adjust midtones, and *Gain* adjusts highlights.



The *Offset, Power, Slope* formula uses *Offset, Power, and Slope*:  $out = (i * s + o) ^ p$

where:

- out** The color graded pixel code value.
- i** The input pixel code value (0=black, 1=white).
- s** Slope (any number 0 or greater, nominal value is 1.0).
- o** Offset (any number, nominal value is 0).
- p** Power (any number greater than 0, nominal value is 1.0).

**Factor** Controls the amount of influence the node exerts on the output image

### Hue Correct

The Hue Correct node is able to adjust the Hue, Saturation, and Value of an image, with an input curve.

By default, the curve is a straight line, meaning there is no change. The spectrum allows you to raise or lower HSV levels for each range of pixel colors. To change a H, S, or V level, move the curve points up or down. Pixels with hue values each point in the horizontal position of the graph will be changed depending on the shape of the curve.

### Bright/Contrast

**Bright** A multiplier-type factor by which to increase the overall brightness of the image. Use a negative number to darken an image.

**Contrast** A scaling type factor by which to make brighter pixels brighter but keeping the darker pixels dark. Higher values make details stand out. Use a negative number to decrease the overall contrast in the image.

**Notes** It is possible that this node will put out a value set that has values beyond normal range, i. e. values  $> 1$  or  $< 0$ . If you will be using the output to mix with other images in the normal range, you should clamp the values using the Map Value node (with the Min and Max enabled), or put through a ColorRamp node (with all normal defaults).

Either of these nodes will scale the values back to normal range. In the example image, we want to amp up the specular pass. The bottom thread shows what happens if we do not clamp the values; the specular pass has values much less than 1 in the dark areas; when added to the medium gray, it makes black. Passing the brightened image through either the Map Value or the ColorRamp produces the desired effect.

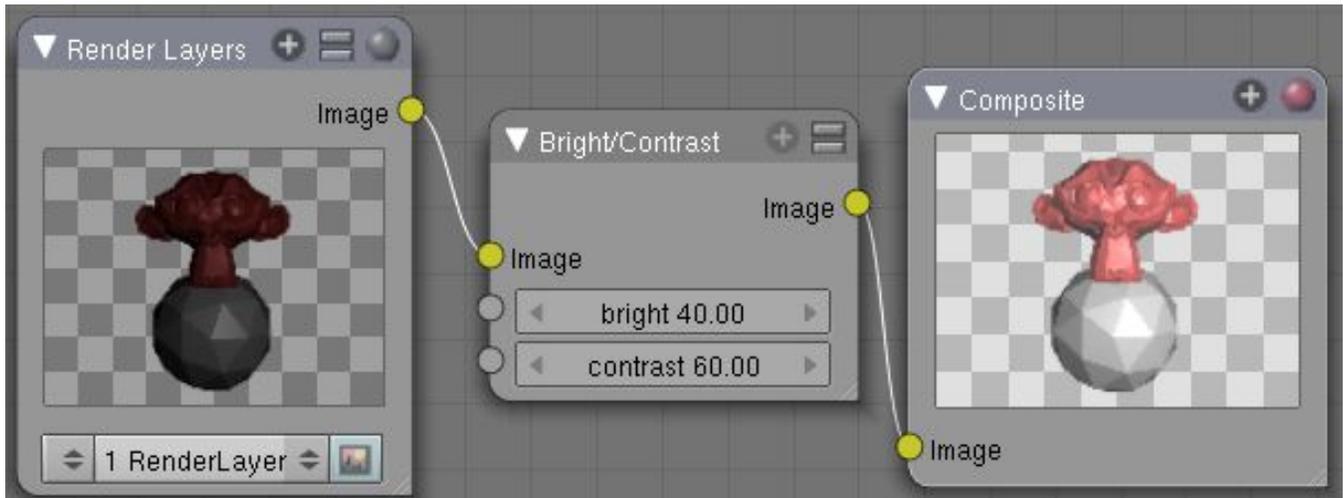
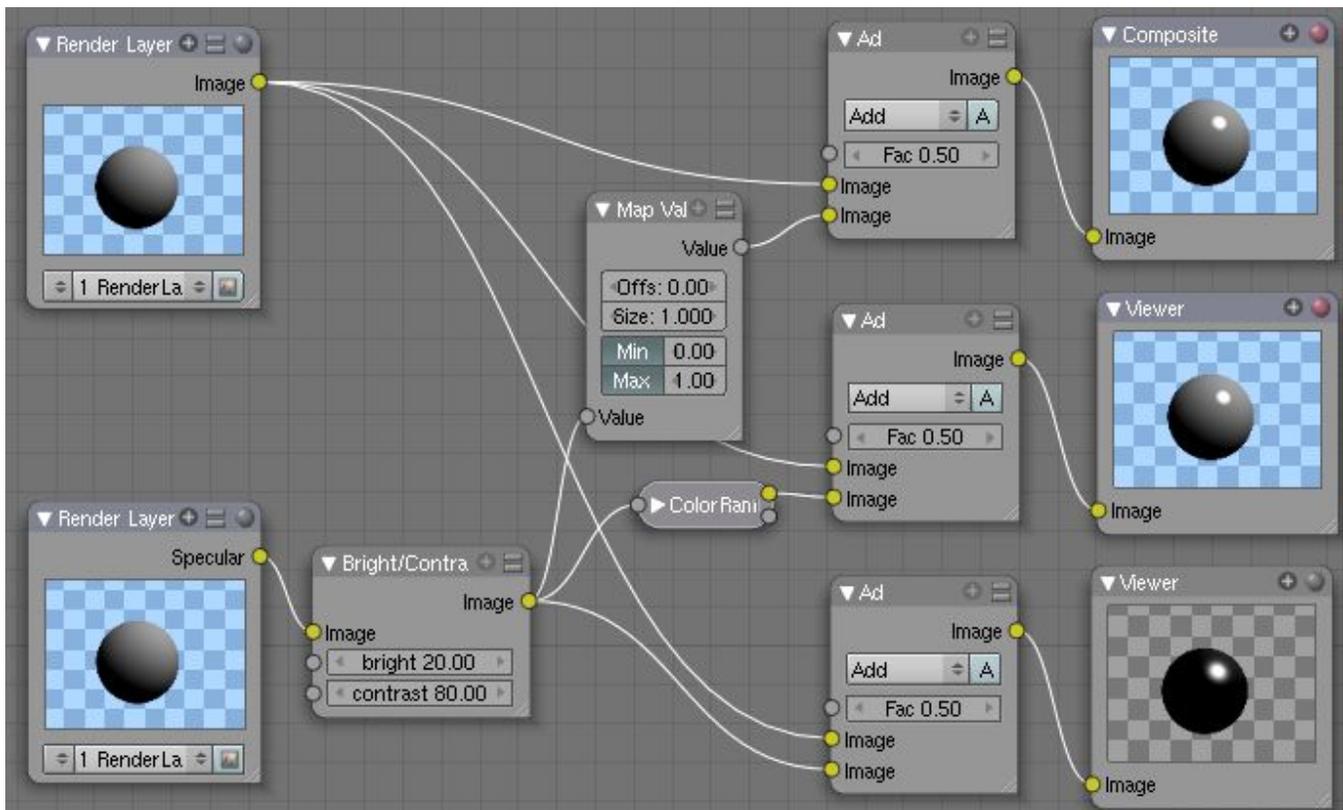
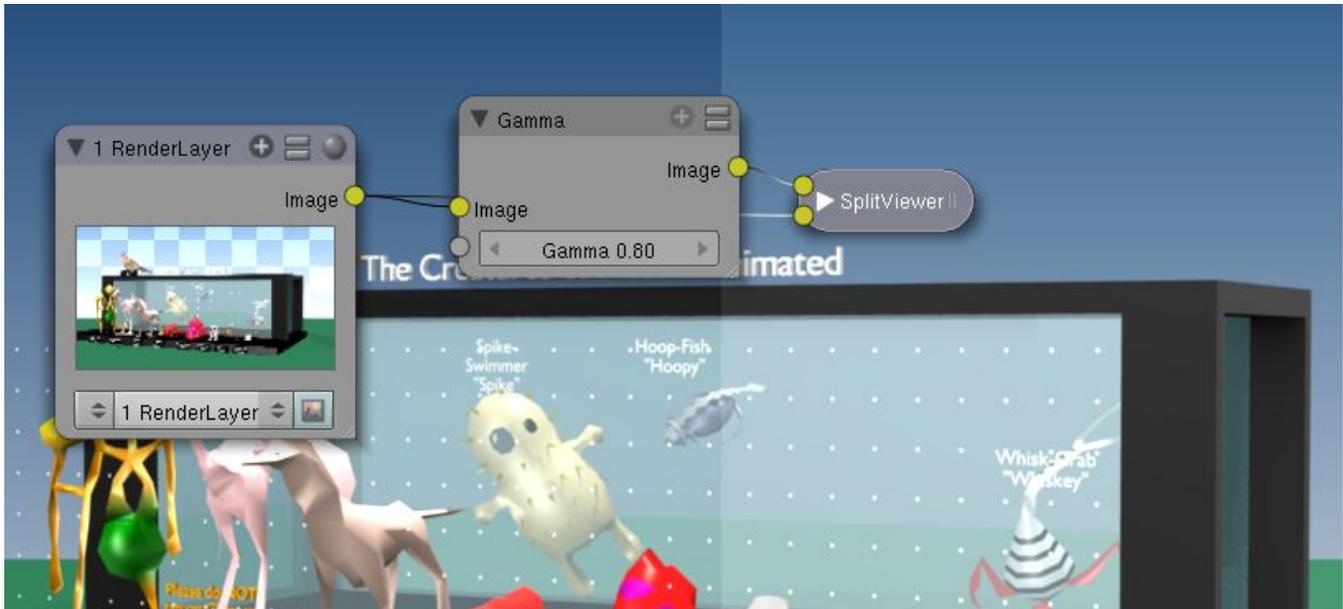


Fig. 2.2497: A basic example





## Gamma

A reason for applying gamma correction to the final render is to correct lighting issues. Lighting issues that can be corrected by a gamma correction node are light attenuation with distance, light falloff at terminators, and light and shadow superpositions. Simply think about the renderer as a virtual camera. By applying a gamma correction to your render, you are just replicating what digital camera do with photos. Digital cameras gamma correct their photos, so you do the same thing. The gamma correction is, indeed, 0.45, not 2.2.

But reverse gamma correction on textures and colors have another very important consequence when you are using rendering techniques such as radiosity or GI. When doing the GI calculations, all textures and colors are taken to mean reflectance. If you do not reverse gamma correct your textures and colors, then the GI render will look way too bright because the reflected colors are all way too high and thus a lot more light is bouncing around than it should.

Gamma correction in Blender enters in a few places. The first is in this section with the nodes, both this node and the Tonemap node, and the second is in calculating Radiosity. In the noodle to the left, the split viewer shows the before and after effect of applying a gamma correction.

## Color Correction

TODO - see: <https://developer.blender.org/T43469>

## Tone Map

Tone mapping is a technique used in image processing and computer graphics to map one set of colors to another in order to approximate the appearance of high dynamic range images in a medium that has a more limited dynamic range.

Essentially, tone mapping addresses the problem of strong contrast reduction from the scene values (radiance) to the displayable range while preserving the image details and color appearance important to appreciate the original scene content.

The Tone Map node has two methods of calculation:

### Rh Simple

**Key** The value the average luminance is mapped to.

**Offset** Normally always 1, but can be used as an extra control to alter the brightness curve

**Gamma** If not used, set to 1

### R/D Photoreceptor

**Intensity** If less than zero, darkens image; otherwise, makes it brighter

**Contrast** Set to 0 to use estimate from input image

**Adaptation** If 0, global; if 1, based on pixel intensity

**Color Correction** If 0, same for all channels; if 1, each independent

### Z-Combine Node

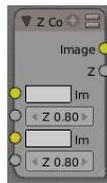


Fig. 2.2498: Z Combine node

The Z-Combine node takes two images and two Z-value sets as input. It overlays the images using the provided Z values to detect which parts of one image are in front of the other. If both Z values are equal, it uses the top image. It puts out the combined image, with the combined Z-depth map, allowing you to thread multiple Z-combines together.

Z-Combine chooses whichever Z-value is less when deciding which image pixel to use. Normally, objects are in front of the camera and have a positive Z value. If one Z-value is negative, and the other positive, Z-Combine will use the image corresponding to the negative value. You can think of a negative Z value as being behind the camera. When choosing between two negative Z-values, Z-Combine will use whichever is more negative.

Alpha values carry over from the input images. Not only is the image pixel chosen, but also its alpha channel value. So, if a pixel is partially or totally transparent, the result of the Z-Combine will also be partially transparent; in which case the background image will show through the foreground (chosen) pixel. Where there are sharp edges or contrast, the alpha map will automatically be anti-aliased to smooth out any artifacts.

However, you can obtain this by making an AlphaOver of two Z-Combine, one normal, the other having inverted (reversed?) Z-values as inputs, obtained using for each of them a *MapValue* node with a *Size* field set to -1.0:

**Examples** In the example to the right, render output from two scenes are mixed using the Z-Offset node, one from a sphere of size 1.30, and the other a cube of size 1.00. The sphere and square are located at the same place. The cube is tipped forward, so the corner in the center is closer to the camera than the sphere surface; so Z-Offset chooses to use the cube's pixels. But the sphere is slightly larger (a size of 1.30 versus 1.00), so it does not fit totally 'inside' the cube. At some point, as the cube's sides recede back away from the camera, the sphere's sides are closer. When this happens, Z-offset uses the sphere's pixels to form the resulting picture.

This node can be used to combine a foreground with a background matte painting. Walt Disney pioneered the use of multi-plane mattes, where three or four partial mattes were painted on glass and placed on the left and right at different Z positions; minimal camera moves to the right created the illusion of depth as Bambi moved through the forest.

**Note:** Valid Input

Z Input Sockets do not accept fixed values; they must get a vector set (see Map Value node). Image Input Sockets will not accept a color, since it does not have UV coordinates.

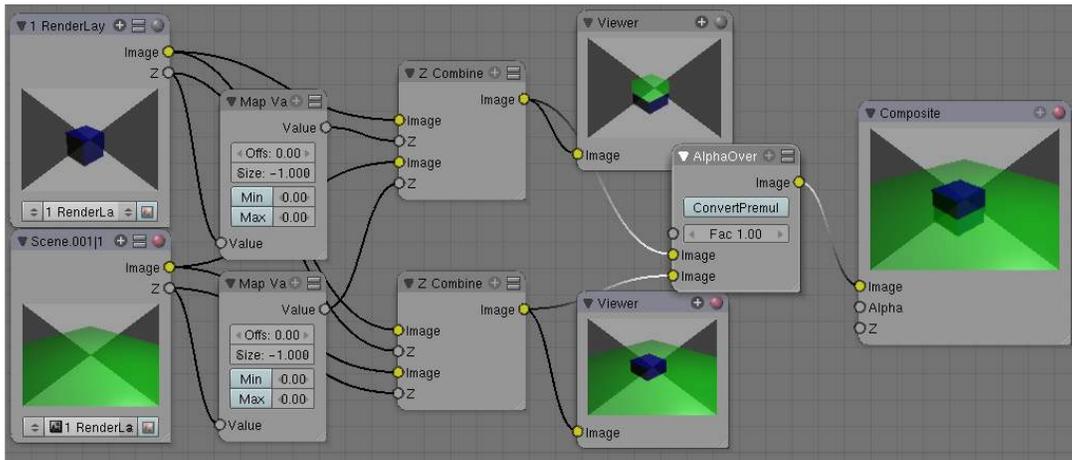


Fig. 2.2499: Alpha and Z-Combine node.

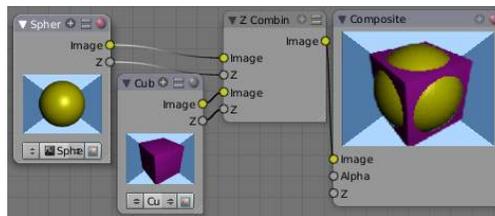


Fig. 2.2500: Choosing closest pixels

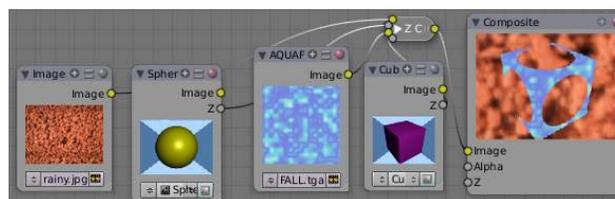


Fig. 2.2501: Mix and Match Images

You can use Z-Combine to merge two images as well, using the Z-values put out by two renderlayers. Using the Z-values from the sphere and cube scenes above, but threading different images, yields the example to the right.

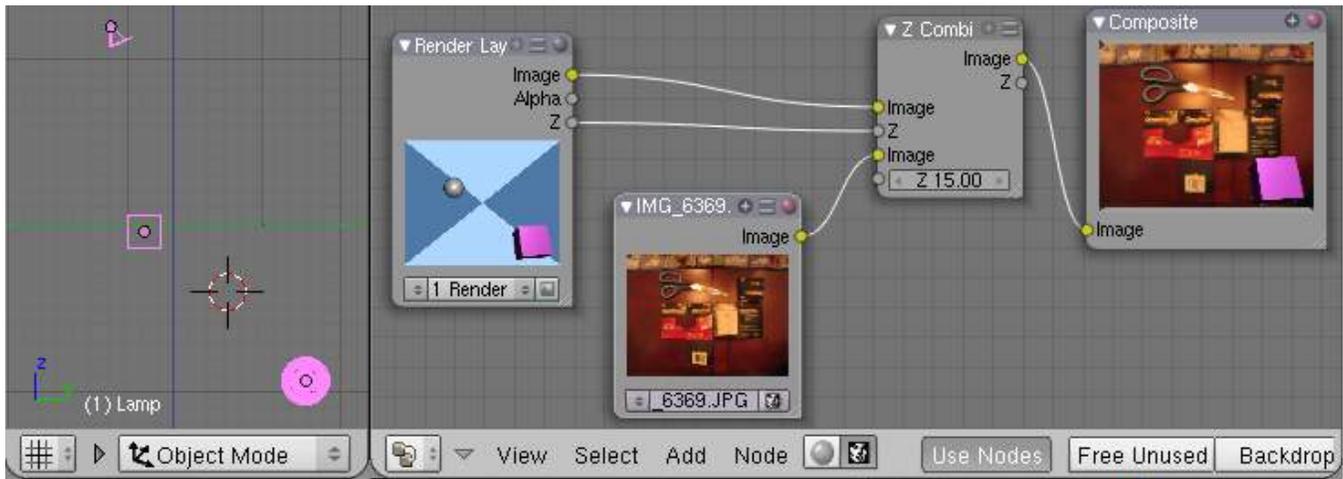


Fig. 2.2502: Z-Combine in action

In this noodle (you may click the little expand-o-matic icon in the bottom right to view it to full size), we mix a render scene with a flat image. In the side view of the scene, the purple cube is 10 units away from camera, and the gray ball is 20. The 3D cursor is about 15 units away from camera. We Z-in the image at a location of 15, thus inserting it in-between the cube and the ball. The resulting image appears to have the cube on the table.

---

#### Note: Invisible Man Effect

If you choose a foreground image which has a higher Alpha than the background, and then mix the Z-combine with a slightly magnified background, the outline of the transparent area will distort the background, enough to make it look like you are seeing part of the background through an invisible yet Fresnel-lens object.

---

## Converter Nodes

As the name implies, these nodes convert the colors or other properties of various data (e.g. transparency) in some way.

They also split out or re-combine the different color channels that make up an image, allowing you to work on each channel independently. Various color channel arrangements are supported, including traditional RGB, HSV and High Definition Media Interface (HDMI) formats.

### Math Node

This node performs the selected math operation on an image or buffer. All common math functions are supported. If only an image is fed to one Value socket, the math function will apply the other Value consistently to every pixel in producing the output Value. Select the math function by clicking the up-down selector where the “Add” selection is shown.

The trig functions of Sine, Cosine, Tangent use only the top socket and accept values in radians between 0 and  $2\pi$  for one complete cycle.

### Examples

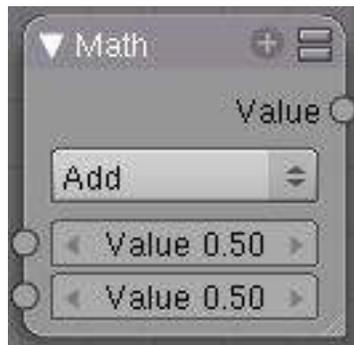


Fig. 2.2503: Math node

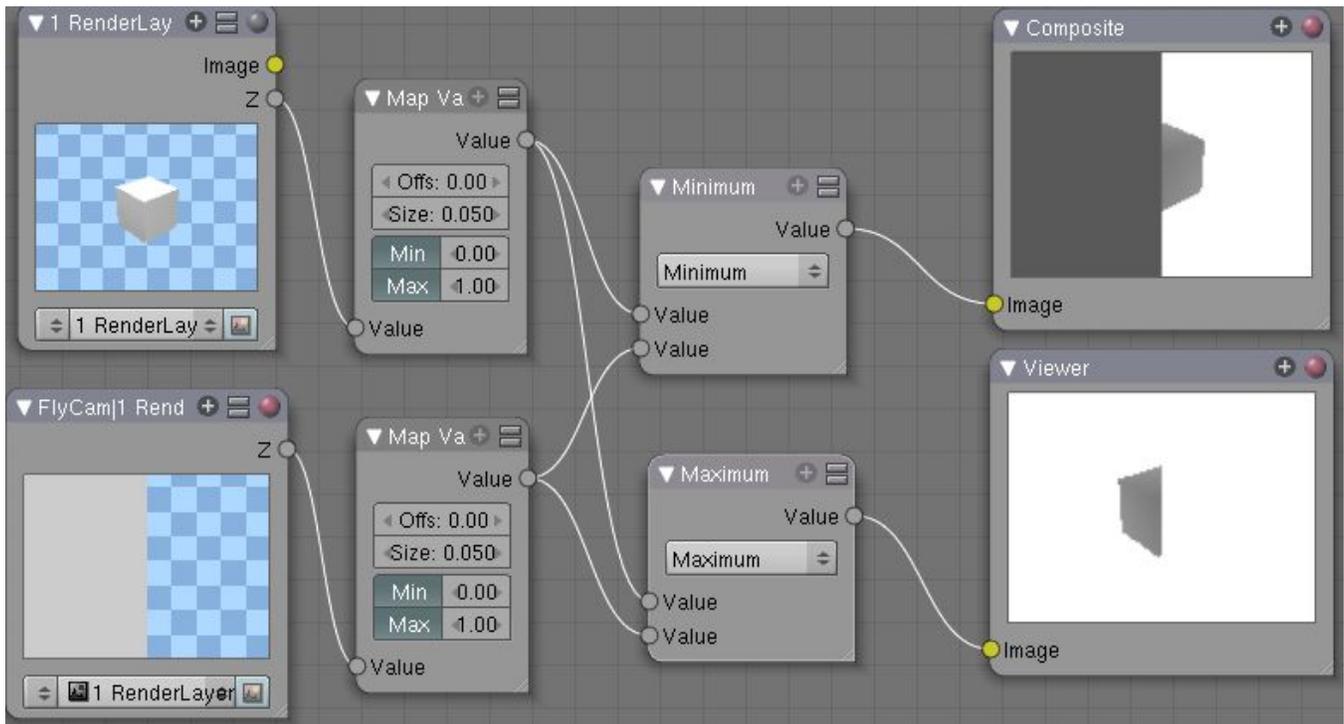
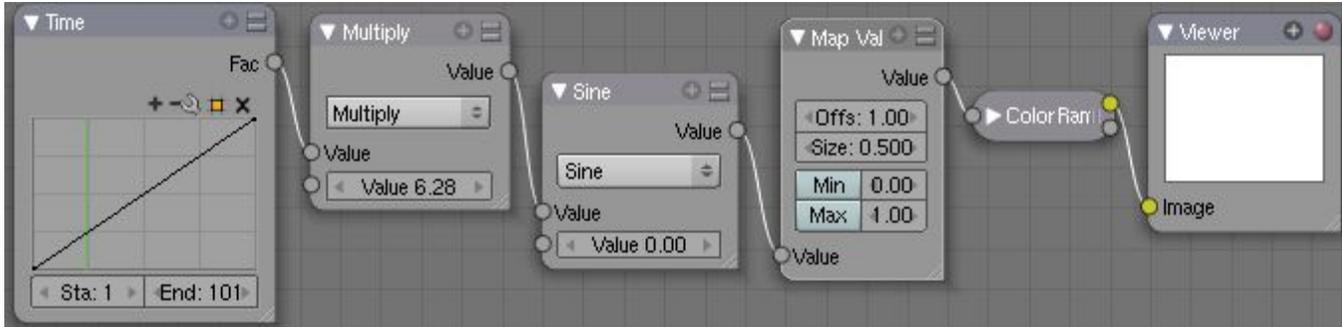


Fig. 2.2504: Example

**Manual Z-Mask** This example has one scene input by the top RenderLayer node, which has a cube that is about 10 BU from the camera. The bottom RenderLayer node inputs a scene (FlyCam) with a plane that covers the left half of the view and is 7 BU from the camera. Both are fed through their respective Map Value nodes to divide the Z buffer by 20 (multiply by .05, as shown in the Size field) and clamped to be a Min/Max of 0.0/1.0 respectively.

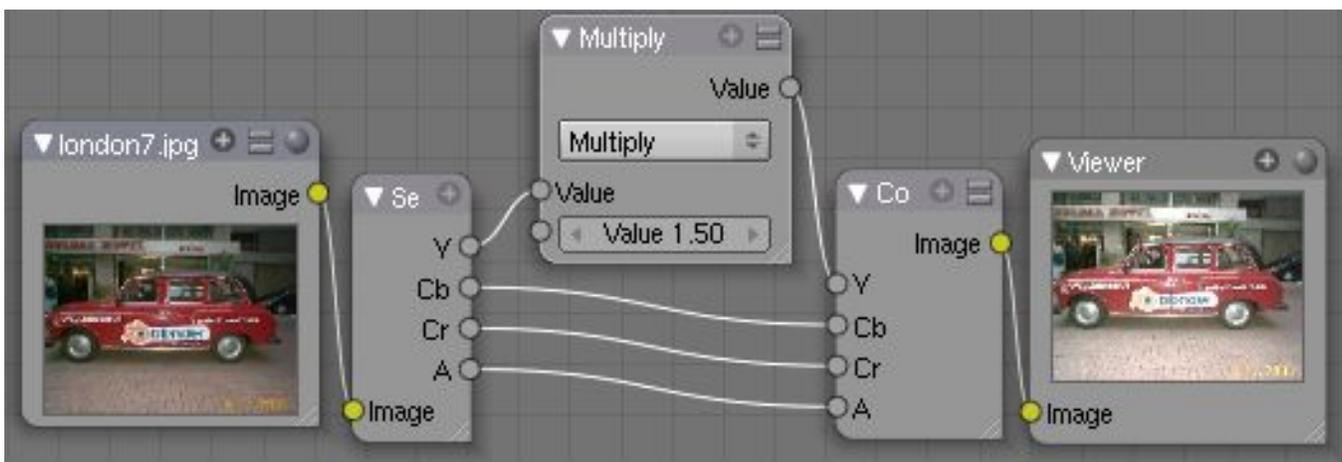
For the Minimum function, the node selects those Z values where the corresponding pixel is closer to the camera; so it chooses the Z values for the plane and part of the cube. The background has an infinite Z value, so it is clamped to 1.0 (shown as white). In the maximum example, the Z values of the cube are greater than the plane, so they are chosen for the left side, but the plane (FlyCam) Renderlayer's Z are infinite (mapped to 1.0) for the right side, so they are chosen.



**Using Sine Function to Pulsate** This example has a Time node putting out a linear sequence from 0 to 1 over the course of 101 frames. The green vertical line in the curve widget shows that frame 25 is being put out, or a value of .25. That value is multiplied by  $2\pi$  and converted to 1.0 by the Sine function, since we all know that  $\text{Sine}(2\pi/4)=\text{Sine}(\pi/2)=+1.0$ .

Since the Sine function can put out values between -1.0 and 1.0, the Map Value node scales that to 0.0 to 1.0 by taking the input (-1 to 1), adding 1 (making 0 to 2), and multiplying the result by one half (thus scaling the output between 0 and 1). The default ColorRamp converts those values to a grayscale. Thus, medium gray corresponds to a 0.0 output by the sine, black to -1.0, and white to 1.0. As you can see,  $\text{Sine}(\pi/2)=1.0$ . Like having your own visual color calculator! Animating this noodle provides a smooth cyclic sequence through the range of grays.

Use this function to vary, for example, the alpha channel of an image to produce a fading in/out effect. Alter the Z channel to move an scene in/out of focus. Alter a color channel value to make a color “pulse”.



**Brightening/Scaling a Channel** This example has a Multiply node increasing the luminance channel (Y) of the image to make it brighter. Note that you should use a Map Value node with Min() and Max () enabled to clamp the output to valid values. With this approach you could use a logarithmic function to make a high-dynamic range image. For this particular example, there is also a Brighten/Contrast node that might give simpler control over brightness.

**Quantize/Restrict Color Selection** In this example, we want to restrict the color output to only 256 possible values. Possible use of this is to see what the image will look like on an 8-bit cell phone display. To do this, we want to restrict the R, G and B values of any pixel to be one of a certain value, such that when they are combined, will not result in more than 256 possible values. The number of possible values of an output is the number of channel values multiplied by each other, or  $Q = R * G * B$ .

Since there are 3 channels and 256 values, we have some flexibility how to quantize each channel, since there are a lot of combinations of  $R * G * B$  that would equal 256. For example, if  $\{R, G, B\} = \{4, 4, 16\}$ , then  $4 * 4 * 16 = 256$ . Also,  $\{6, 6, 7\}$  would give 252 possible values. The difference in appearance between  $\{4, 4, 16\}$  and  $\{6, 6, 7\}$  is that the first set  $\{4, 4, 16\}$  would have fewer shades of red and green, but lots of shades of blue. The set  $\{6, 6, 7\}$  would have a more even distribution of colors. To get better image quality with fewer color values, give more possible values to the predominant colors in the image.

### Theory Two Approaches to Quantizing to 6 values

To accomplish this quantization of an image to 256 possible values, lets use the set  $\{6, 6, 7\}$ . To split up a continuous range of values between 0 and 1 (the full Red spectrum) into 6 values, we need to construct an algorithm or function that takes any input value but only puts out 6 possible values, as illustrated by the image to the right. We want to include 0 as true black, with five other colors in between. The approach shown produces  $\{0, .2, .4, .6, .8, 1\}$ . Dividing 1.0 by 5 equals .2, which tells us how far apart each quantified value is from the other.

So, to get good even shading, we want to take values that are 0.16 or less and map them to 0.0; values between 0.16 and 0.33 get fixed to 0.2; colorband values between 0.33 and 0.5 get quantized to 0.4, and so on up to values between 0.83 and 1.0 get mapped to 1.0.

---

#### Note: Function $f(x)$

An algebraic function is made up of primitive mathematical operations (add, subtract, multiply, sine, cosine, etc) that operate on an input value to provide a desired output value.

---

#### Spreadsheet showing a function

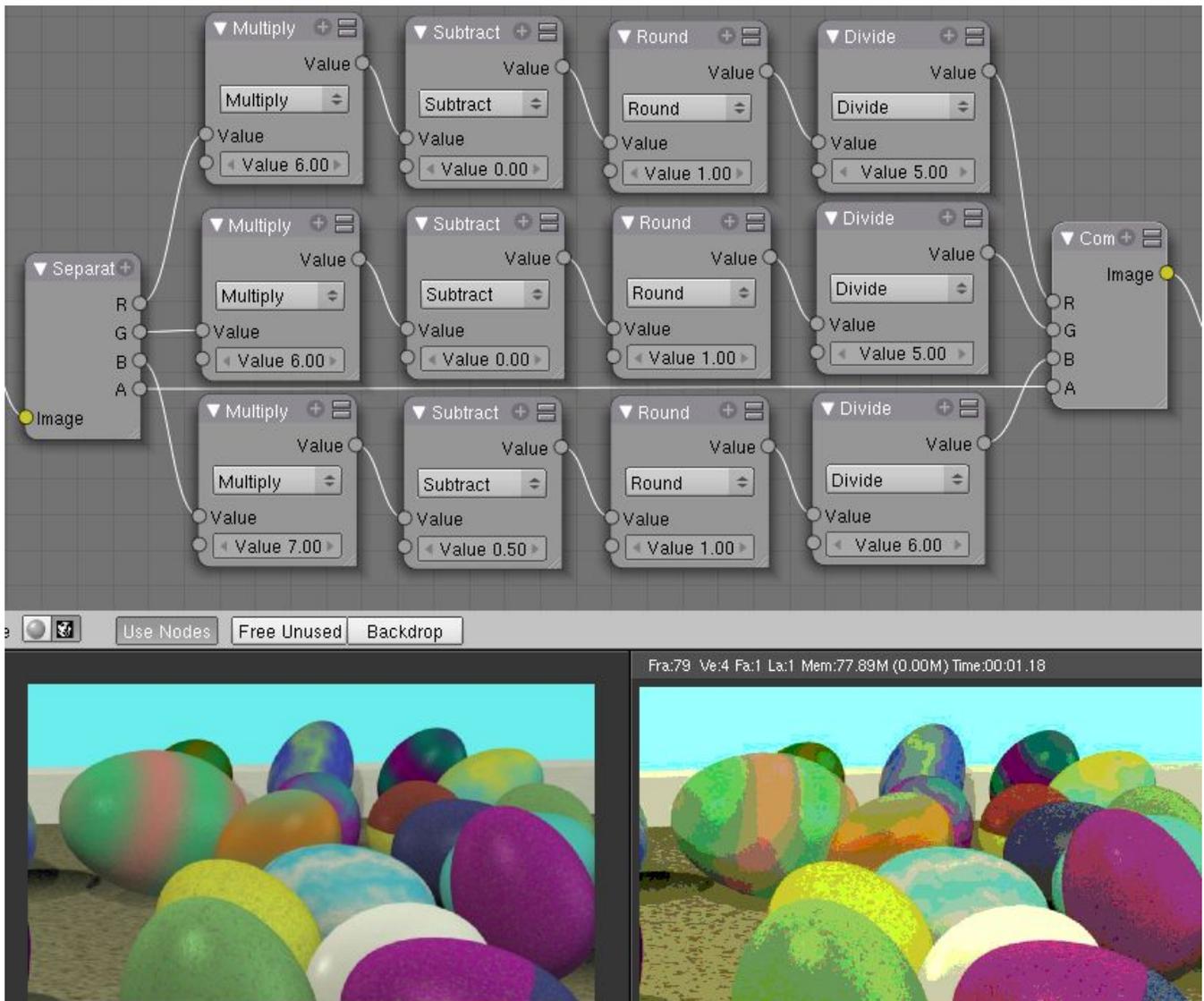
The theory behind this function is scaled truncation. Let us suppose we want a math function that takes in a range of values between 0 and 1, such as .552, but only outputs a value of 0.0, 0.2, 0.4, etc. We can imagine then that we need to get that range 0 to 1 powered up to something 0 to 6 so that we can chop off and make it a whole number. So, with six divisions, how can we do that? The answer is we multiply the range by 6. The output of that first math multiply node is a range of values between 0 and 6. To get even divisions, because we are using the rounding function (see documentation above), we want any number plus or minus around a whole number will get rounded to that number. So, we subtract a half, which shifts everything over. The Round() function then makes that range 0 to 5. We then divide by 5 to get back a range of numbers between 0 and 1 which can then be combined back with the other color channels. Thus, you get the function

$$f(x, n) = \text{round}[x * n - 1/2] / (n - 1)$$

where  $n$  is the number of possible output values, and  $x$  is the input pixel color and  $f(x, n)$  is the output value. There's only one slight problem, and that is for the value exactly equal to 1, the formula result is 1.2, which is an invalid value. This is because the round function is actually a roundup function, and exactly 5.5 is rounded up to 6. So, by subtracting .501, we compensate and thus 5.499 is rounded to 5. At the other end of the spectrum, pure black, or 0, when .501 subtracted, rounds up to 0 since the Round() function does not return a negative number.

Sometimes using a spreadsheet can help you figure out how to put these nodes together to get the result that you want. Stepping you through the formula for  $n=6$  and  $x=0.70$ , locate the line on the spreadsheet that has the 8-bit value 179 and R value 0.7. Multiplying by 6 gives 4.2. Subtracting 1/2 gives 3.7, which rounds up to 4. 4 divided by 5 = .8. Thus,  $f(0.7, 6) = 0.8$  or an 8-bit value of 204. You can see that this same 8-bit value is output for a range of input values. Yeah! Geeks Rule! This is how you program Blender to do compositing based on Algebra. Thank a Teacher if you understand this.

**Reality** To implement this function in Blender, consider the noodle above. First, feed the image to the Separate RGB node. For the Red channel, we string the math nodes into a function that takes each red color, multiplies (scales) it up by the desired number of divisions (6), offsets it by 0.5, rounds the value to the nearest whole number, and then divides the image pixel color



by 5. So, the transformation is  $\{0..1\}$  becomes  $\{0..6\}$ , subtracting centers the medians to  $\{-0.5..5.5\}$  and the rounding to the nearest whole number produces  $\{0,1,2,3,4, 5\}$  since the function rounds down, and then dividing by five results in six values  $\{0.0,0.2,0.4,0.6,0.8,1.0\}$ .

The result is that the output value can only be one of a certain set of values, stair-stepped because of the rounding function of the math node noodle. Copying this one channel to operate on Green and Blue gives the noodle below. To get the 6:6:7, we set the three multiply nodes to  $\{6,6,7\}$  and the divide nodes to  $\{5,5,6\}$ .

If you make this into a node group, you can easily re-use this setup from project to project. When you do, consider using a math node to drive the different values that you would have to otherwise set manually, just to error-proof your work.

**Summary** Normally, an output render consists of 32- or 24-bit color depth, and each pixel can be one of millions of possible colors. This noodle example takes each of the Red, Green and Blue channels and normalizes them to one of a few values. When all three channels are combined back together, each color can only be one of 256 possible values.

While this example uses the Separate/Combine RGB to create distinct colors, other Separate/Combine nodes can be used as well. If using the YUV values, remember that U and V vary between -0.5 and +0.5, so you will have to first add on a half to bring the range between 0 and 1, and then after dividing, subtract a half to bring in back into standard range.

The JPG or PNG image format will store each of the colors according to their image standard for color depth (e.g. JPG is 24-bit), but the image will be very very small, since reducing color depth and quantizing colors is essentially what the JPEG compression algorithm accomplishes.

You do not have to reduce the color depth of each channel evenly. For example, if blue was the dominant color in an image, to preserve image quality, you could reduce Red to 2 values, Green to 4, and let the blue take on  $256/(2*4)$  or 32 values. If using the HSV, you could reduce the Saturation and Value to 2 values (0 or 1.0) by Multiply by 2 and Divide by 2, and restrict the Hue to 64 possible values.

You can use this noodle to quantize any channel; alpha, speed (vector), z-values, and so forth.

## ColorRamp Node

The ColorRamp Node is used for mapping values to colors with the use of a gradient. It works exactly the same way as a [Colorband for textures and materials](#), using the Factor value as a slider or index to the color ramp shown, and outputting a color value and an alpha value from the output sockets.

By default, the ColorRamp is added to the node map with two colors at opposite ends of the spectrum. A completely black black is on the left (Black as shown in the swatch with an Alpha value of 1.00) and a whitewash white is on the right. To select a color, LMB click on the thin vertical line/band within the colorband. The example picture shows the black color selected, as it is highlighted white. The settings for the color are shown above the colorband as (left to right): color swatch, Alpha setting, and interpolation type.

To change the hue of the selected color in the colorband, LMB click on the swatch, and use the pop-up color picker control to select a new color. Press Return to set that color.

To add colors, hold Ctrl down and Ctrl-LMB click inside the gradient. Edit colors by clicking on the rectangular color swatch, which pops up a color-editing dialog. Drag the gray slider to edit Alpha values. Note that you can use textures for masks (or to simulate the old “Emit” functionality) by connecting the alpha output to the factor input of an RGB mixer.

To delete a color from the colorband, select it and press the Delete button.

When using multiple colors, you can control how they transition from one to another through an interpolation mixer. Use the interpolation buttons to control how the colors should band together: Ease, Cardinal, Linear, or Spline.

Use the A: button to define the Alpha value of the selected color for each color in the range.

**Using ColorRamp to create an Alpha Mask** A powerful but often overlooked feature of the ColorRamp is to create an Alpha Mask, or a mask that is overlaid on top of another image, and, like a mask, allows some of the background to show through. The example map below shows how to use the Color Ramp node to do this:

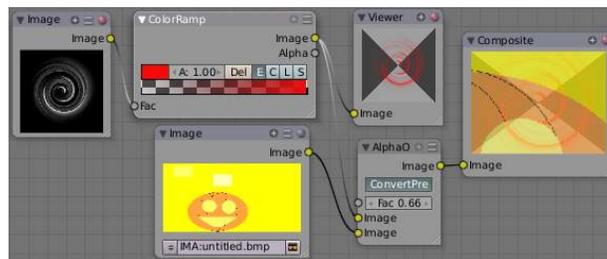


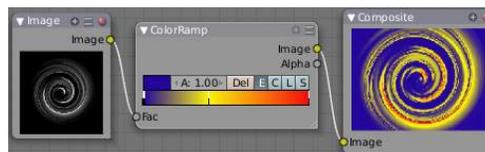
Fig. 2.2505: Using the ColorRamp node to create an alpha mask

In the map above, a black and white swirl image, which is lacking an alpha channel, is fed into the ColorRamp node as a *Factor*. (Technically, we should have converted the image to a value using the RGB-to-BW node, but hey, this works just as well since we are using a BW image as input.)

We have set the ColorRamp node to a purely transparent color on the left end of the spectrum, and a fully Red color on the right. As seen in the viewer, the ColorRamp node puts out a mask that is fully transparent where the image is black. Black is zero, so ColorRamp uses the ‘color’ at the left end of the spectrum, which we have set to transparent. The ColorRamp image is fully red and opaque where the image is white (1.00).

We verify that the output image mask is indeed transparent by overlaying it on top of a pumpkin image. For fun, we made that AlphaOver output image 0.66 transparent so that we can, in the future, overlay the image on a flashing white background to simulate a scary scene with lighting flashes.

**Using ColorRamp to Colorize an Image** The real power of ColorRamp is that multiple colors can be added to the color spectrum. This example compositing map takes a boring BW image and makes it a flaming swirl!



In this example, we have mapped the shades of gray in the input image to three colors, blue, yellow, and red, all fully opaque (Alpha of 1.00). Where the image is black, ColorRamp substitutes blue, the currently selected color. Where it is some shade of gray, ColorRamp chooses a corresponding color from the spectrum (bluish, yellow, to reddish). Where the image is fully white, ColorRamp chooses red.

### Set Alpha Node



Fig. 2.2506: Set Alpha node

This node adds an alpha channel to a picture. Some image formats, such as JPEG, do not support an alpha channel. In order to overlay a JPEG image on top of a background, you must add an alpha channel to it using this node.

The *Image* input socket is optional. If an input image is not supplied, the base color shown in the swatch will be used. To change the color, LMB click the swatch and use the color-picker control to choose or specify a color you want.

The amount of *Alpha* (1.00 being totally opaque and 0.00 being totally transparent) can be set for the whole picture using the input field. Additionally, the Alpha factor can be set by feeding its socket.

**Note:** This is not, and is not intended to be, a general-purpose solution to the problem of compositing an image that doesn't contain Alpha information. You might wish to use "Chroma Keying" or "Difference Keying" (as discussed elsewhere) if you can. This node is most often used (with a suitable input being provided by means of the socket) in those troublesome cases when you *can't*, for some reason, use those techniques directly.

**Using SetAlpha to Fade to Black** To transition the audience from one scene or shot to another, a common technique is to "fade to black". As its name implies, the scene fades to a black screen. You can also "fade to white" or whatever color you wish, but black is a good neutral color that is easy on the eyes and intellectually "resets" the viewer's mind. The node map below shows how to do this using the Set Alpha node.

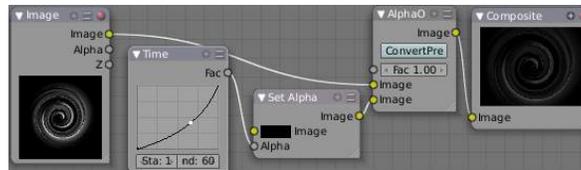


Fig. 2.2507: Fade To Black

In the example above, the alpha channel of the swirl image is ignored. Instead, a *time node* introduces a factor from 0.00 to 1.00 over 60 frames, or about 2 seconds, to the Set Alpha node. Note that the time curve is exponentially-shaped, so that the overall blackness will fade in slowly and then accelerate toward the end. The Set Alpha node does not need an input image; instead the flat (shadeless) black color is used. The Set Alpha Node uses the input factor and color to create a black image that has an alpha set which goes from 0.00 to 1.00 over 60 frames, or completely transparent to completely opaque. Think of alpha as a multiplier for how vivid you can see that pixel. These two images are combined by our trusty AlphaOver node completely (a *Factor* of 1.00) to produce the composite image. The SetAlpha node will thus, depending on the frame being rendered, produce a black image that has some degree of transparency. Set up and Animate, and you have an image sequence that fades to black over a 2-second period.

**Note:** No Scene information used

This example node map does not use the RenderLayer. To produce this 2 second animation, no blender scene information was used. This is an example of using Blender's powerful compositing abilities separate from its modeling and animation capabilities. (A Render Layer could be substituted for the Image layer, and the "fade-network" effect will still produce the same effect)

**Using SetAlpha to Fade In a Title** To introduce your animation, you will want to present the title of your animation over a background. You can have the title fly in, or fade it in. To fade it in, use the SetAlpha node with the Time node as shown below.

In the above example, a Time curve provides the Alpha value to the input socket. The current RenderLayer, which has the title in view, provides the image. As before, the trusty AlphaOver node mixes (using the alpha values) the background swirl and the alphaed title to produce the composite image. Notice the *ConvertPre* -Multiply button is NOT enabled; this produces a composite where the title lets the background image show through where even the background image is transparent, allowing you to layer images on top of one another.

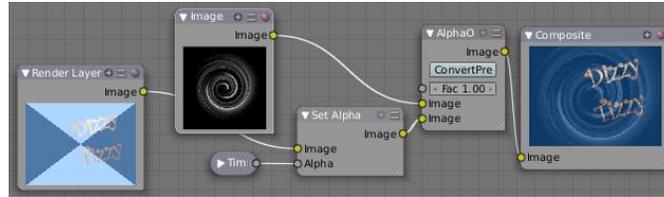


Fig. 2.2508: Using Set Alpha to Fade in a Title



Fig. 2.2509: Using Set Alpha to Colorize an Image

**Using SetAlpha to Colorize a BW Image** In the example above, notice how the blue tinge of the render input colors the swirl. You can use the Set Alpha node's color swatch with this kind of node map to add a consistent color to a BW image.

In the example map to the right, use the *Alpha* value of the SetAlpha node to give a desired degree of colorization. Thread the input image and the Set Alpha node into an AlphaOver node to colorize any black and white image in this manner. Note the *ConvertPre* -Multiply button is enabled, which tells the AlphaOver node not to multiply the alpha values of the two images together.

### Alpha Convert

This node converts the alpha channel interpretation of an image from pre-multiplied to straight or the reverse.

For details on the difference between both kinds of alpha channels see [Alpha Channel](#).

### ID Mask Node



Fig. 2.2510: ID Mask node

This node will use the Object Index pass (see RenderLayers) to produce an anti-aliased alpha mask for the object index specified. The mask is opaque where the object is, and transparent where the object isn't. If the object is partially transparent, the alpha mask matches the object's transparency. This post-process function fills in the jaggies with interpolated values.

---

#### Note: Object Index

Object indices are only output from a RenderLayers node or stored in a multilayer OpenEXR format image.

---

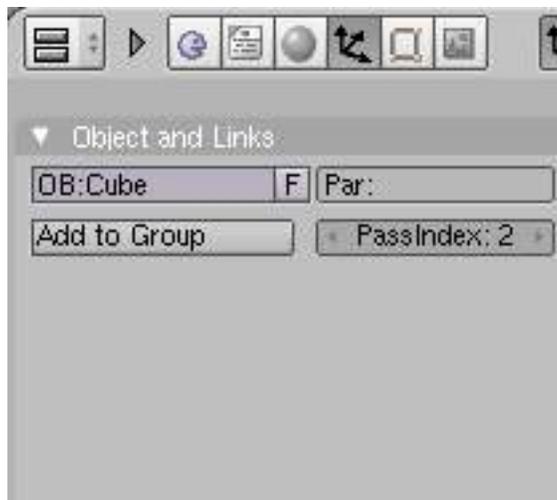


Fig. 2.2511: Setting an Object Index

You can specify, for any of the objects in your scene, an Object Index as shown the right (the currently select object has an index of 2). When rendered, if Object Index passes are enabled, its index will be 2, and setting the ID Mask node to 2 will show where that object is in the scene.

This node is extremely well suited to removing the aliases shown as output from the Defocus node or DOF noodles caused by some objects being close to camera against objects far away.

**Example** In this example, the left rear red cube is assigned PassIndex 1, and the right cube PassIndex 2. Where the two cubes intersect, there is going to be noticeable pixelation (jaggies) because they come together at a sharp angle and are different colors. Using the mask from object 1, which is smoothed (anti-aliased) at the edges, we use a Mix node set on Multiply to multiply the smoothed edges against the image, thus removing those nasty (Mick) Jaggies. Thus, being smoothed out, the Rolling Stones gather no moss. (I really hope you get that obscure reference :)

Note that the mask returns white where the object is fully visible to the camera (not behind anything else) and black for the part of the object that is partially or totally obscured by a fully or partially opaque object in front of it. If something else is in front of it, even if that thing is partially transparent and you can see the object in a render, the mask will not reflect that partially obscured part.

### RGB to BW Node

This node converts an RGB input and outputs a greyscale image.

### Combine/Separate Nodes

All of these node do essentially the same thing: they split out an image into (or recombine an image from) its composite color channels. Each format supports the Alpha (transparency) channel. The standard way of representing color in an image is called a *color space*. There are several color spaces supported:

**RGB** Red-Green-Blue traditional primary colors, also broadcast directly to most computer monitors

**HSV** Three values, often considered as more intuitive than the RGB system (nearly only used on computers):

**Hue** the **Hue** of the color (in some way, choose a 'color' of the rainbow);

**Saturation** the **quantity** of hue in the color (from desaturate - shade of gray - to saturate - brighter colors)

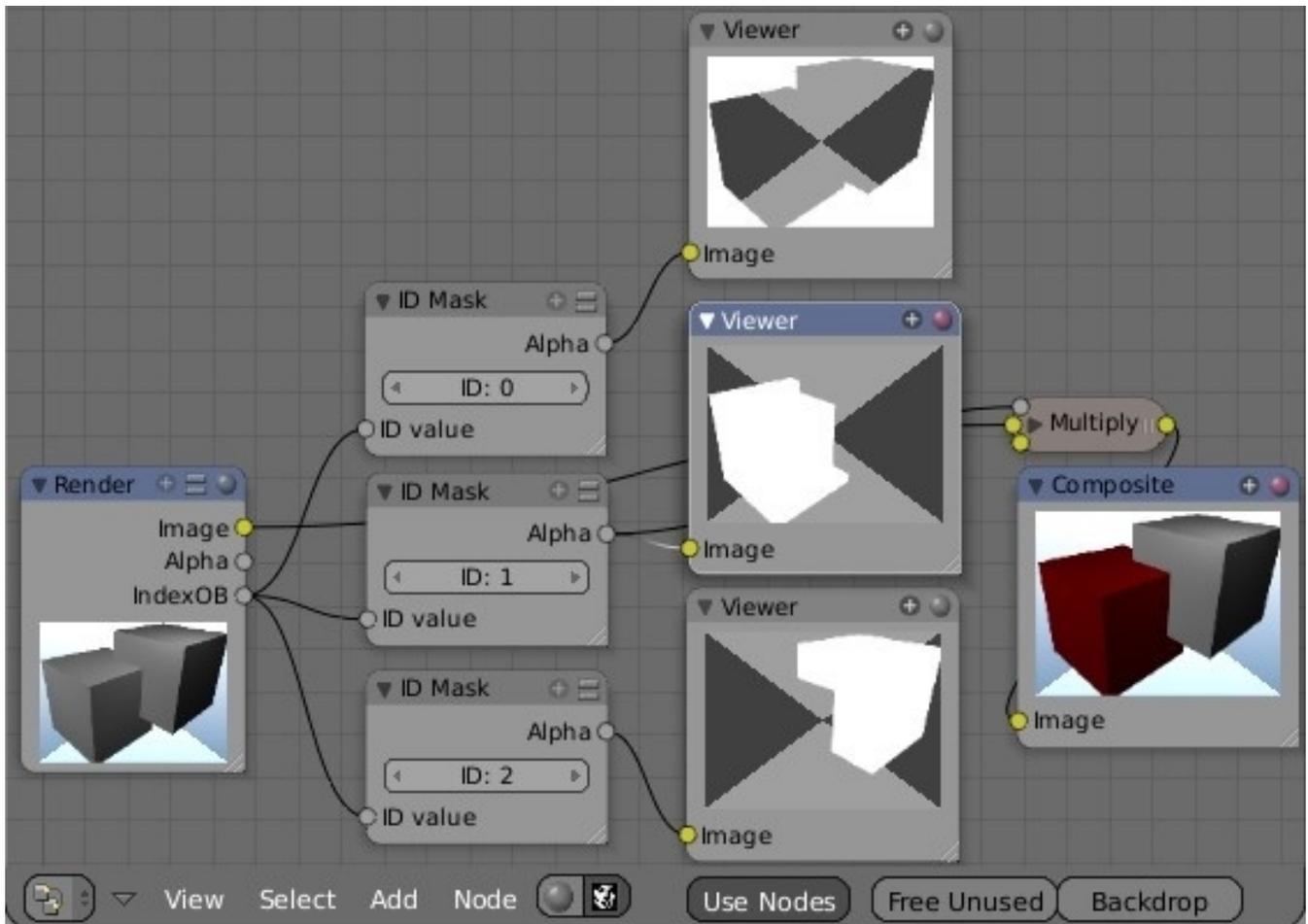


Fig. 2.2512: Example

**Value: the luminosity of the color** (from ‘no light’ - black - to ‘full light’ - ‘full’ color, or white if Saturation is 0.0).

**YUV** Luminance-Chrominance standard used in broadcasting analog PAL (European) video.

**YCbCr** Luminance-ChannelBlue-ChannelRed Component video for digital broadcast use, whose standards have been updated for HDTV and commonly referred to as the HDMI format for component video.

See the global wikipedia for more information on color spaces.

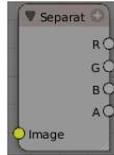


Fig. 2.2513: Separate RGBA node

**Separate/Combine RGBA Node** This node separates an image into its red, green, blue and alpha channels. There's a socket for each channel on the right.

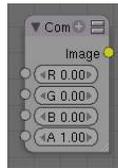
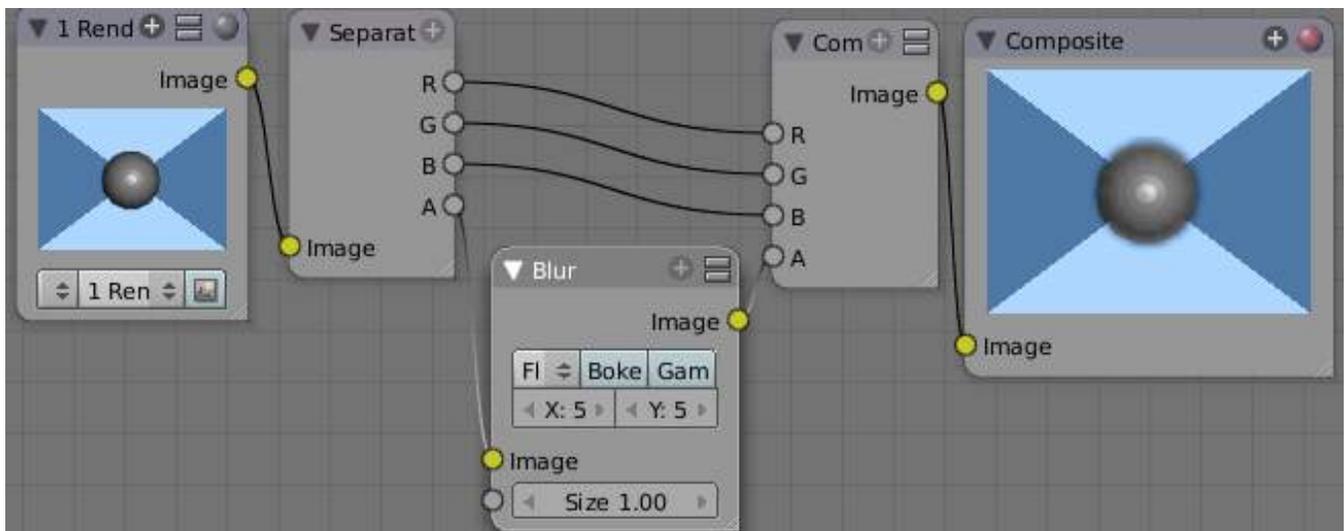
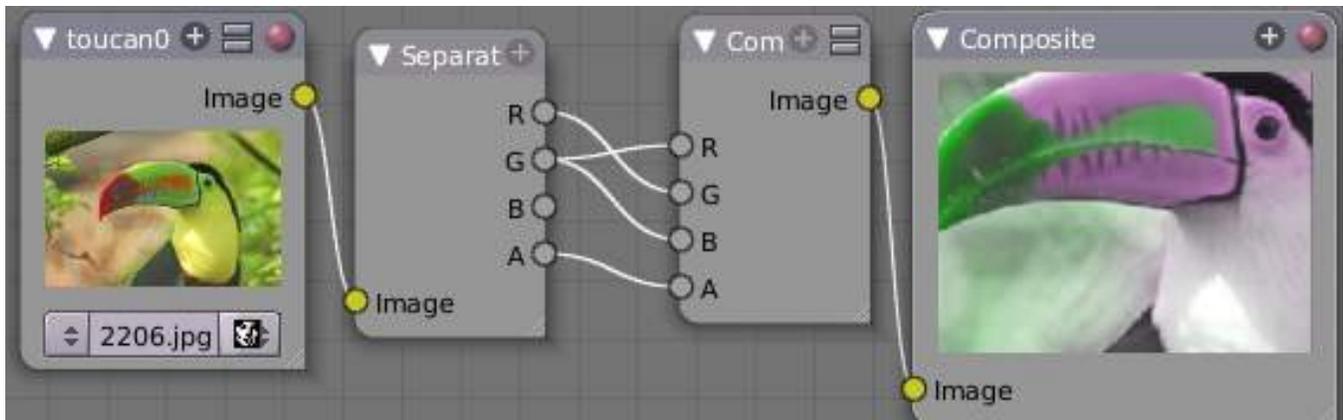


Fig. 2.2514: Combine RGBAnode

This node combines separate input images as each color and alpha channel, producing a composite image. You use this node combine the channels after working on each color channel separately.



**Examples** In this first example, we take the Alpha channel and blur it, and then combine it back with the colors. When placed in a scene, the edges of it will blend in, instead of having a hard edge. This is almost like anti-aliasing, but in a three-dimensional sense. Use this noodle when adding CG elements to live action to remove any hard edges. Animating this effect over a broader scale will make the object appear to “phase” in and out, as a “out-of-phase” time-traveling sync effect.



In this fun little noodle we make all the reds become green, and all the green both Red and Blue, and remove Blue from the image completely. Very cute. Very fun.



Fig. 2.2515: Separate HSVA node

**Separate/Combine HSVA Nodes** This node separates an image into image maps for the hue, saturation, value and alpha channels.

Use and manipulate the separated channels for different purposes; i.e. to achieve some compositing/color adjustment result. For example, you could expand the Value channel (by using the multiply node) to make all the colors brighter. You could make an image more relaxed by diminishing (via the divide or map value node) the Saturation channel. You could isolate a specific range of colors (by clipping the Hue channel via the Colorramp node) and change their color (by the Add/Subtract mix node).



Fig. 2.2516: Separate YUVA node

**Separate/Combine YUVA Node** This node converts an RGBA image to YUVA color space, then splits each channel out to its own output so that they can be manipulated independently. Note that U and V values range from -0.5 to +0.5.

Combines the channels back into a composite image. If you do not connect any input socket, you can set a default value for the whole image for that channel using the numeric controls shown.

**Separate/Combine YCbCrA Node** This node converts an RGBA image to YCbCrA color space, then splits each channel out to its own output so that they can be manipulated independently:

- Y: Luminance, 0=black, 1=white
- Cb: Chrominance Blue, 0=Blue, 1=Yellow



Fig. 2.2517: Combine YUVA node

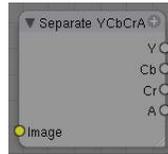


Fig. 2.2518: Separate YCbCrA node

- Cr: Chrominance Red, 0=Red, 1=Yellow

---

**Note:** If running these channels through a ColorRamp to adjust value, use the Cardinal scale for accurate representation. Using the Exponential scale on the luminance channel gives high-contrast effect.

---



Fig. 2.2519: Combine YCbCrA node

So, I kinda think you get the idea, and I was trying to think of some other creative way to write down the same thing, but I can't. So, you'll have to figure this node out on your own.

## Filter Nodes

Filters process the pixels of an image to highlight additional details or perform some sort of post-processing effect on the image.

### Blur Node

The Blur node blurs an image, using one of seven blur modes (set using the upper-left pop-up button), and a radius defined by the X and Y number buttons. By default these are set to zero, so to enable the node you must set one or both to a value greater than 0. You can optionally connect a value image to the Size input node, to control the blur radius with a mask. The values must be mapped between 0-1 for best effect, as they will be multiplied with the X and Y number button values.

**Options** The X and Y values are the number of pixels over which to spread the blur effect.

The Bokeh button (only visible as Bok or Bo on some screen setups) will force the blur node to use a circular blur filter. This gives higher quality results, but is slower than using a normal filter. The Gam button (for "gamma") makes the Blur node gamma-correct the image before blurring it.

The difference between them is how they handle sharp edges and smooth gradients and preserve the highs and the lows. In particular (and you may have to closely examine the full-resolution picture to see this):

**Flat** Simply blurs everything uniformly



Fig. 2.2520: Blur node

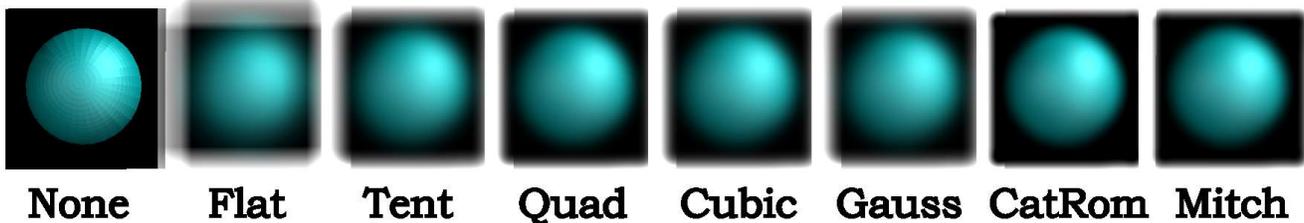


Fig. 2.2521: Blur node blur modes using 15% of image size as XY, no Bokeh/Gamma. Click expand to see details

**Tent** Preserves the high and the lows better making a linear falloff

**Quadratic** CatRom keeps sharp-contrast edges crisp.

**Cubic, Mitch** Preserve the highs but give almost a out-of-focus blur while smoothing sharp edges

**Example** An example blend file, in fact the one used to create the image above, is [available here](#). The .blend file takes one image from the RenderLayer “Blurs” and blurs it while offsetting it (Translate) and then combining it (AlphaOver) to build up the progressive sequence of blurs. Play with the Value and Multiply nodes to change the amount of blurring that each algorithm does.

### Bilateral Blur Node

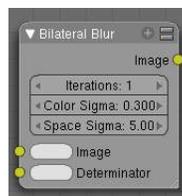


Fig. 2.2522: Blur node

The bilateral blur node performs a high quality adaptive blur on the source image. It can be used for various purposes like: smoothing results from blenders raytraced ambient occlusion smoothing results from various unbiased renderers, to fake some performance-heavy processes, like blurry refractions/reflections, soft shadows, to make non-photorealistic compositing effects.

### Inputs

**Bilateral blur has 2 inputs:** *Image*, for the image to be blurred. *Determinator*, which is non-obligatory, and is used only if connected.

if only 1st input is connected, the node blurs the image depending on the edges present in the source image. If the Determinator is connected, it serves as the source for defining edges/borders for the blur in the image. This has great advantage in case the source image is too noisy, but normals in combination with zbuffer can still define exact borders/edges of objects.

## Options

**Iterations** Defines how many times the filter should perform the operation on the image. It practically defines the radius of blur.

**Color Sigma** Defines the threshold for which color differences in the image should be taken as edges.

**Space sigma** A fine-tuning variable for blur radius.

## Examples

### Dilate/Erode Node

This node blurs individual color channels. The color channel (or a black and white image) is connected to the *Mask* input socket, and the *Distance* is set manually (by clicking on the arrows or the value) or automatically from a value node or a time-and-map-value noodle. A positive value of *Distance* expands the influence of a pixel on its surrounding pixels, thus blurring that color outward. A negative value erodes its influence, thus increases the contrast of that pixel relative to its surrounding pixels, thus sharpening it relative to surrounding pixels of the same color.

**Example** In the above example image, we wanted to take the rather boring array of ball bearings and spruce it up; make it hot, baby. So, we dilated the red and eroded the green, leaving the blue alone. If we had dilated both red and green...(hint: red and green make yellow). The amount of influence is increased by increasing the *Distance* values. [Blend file available here.](#)

### Despeckle

TODO - see: <https://developer.blender.org/T43469>

### Filter Node

The Filter node implements various common image enhancement filters. The supported filters are, if not obvious, named after the mathematical genius who came up with them:

**Soften** Slightly blurs the image.

**Sharpen** Increases the contrast, especially at edges

**Laplace** Softens around edges

**Sobel** Creates a negative image that highlights edges

**Prewitt** Tries to do Sobel one better.

**Kirsch** Improves on the work done by those other two flunkies, giving a better blending as you approach an edge.

**Shadow** Performs a relief emboss/bumpmap effect, darkening outside edges.

The *Soften*, *Laplace*, *Sobel*, *Prewitt* and *Kirsch* all perform edge-detection (in slightly different ways) based on vector calculus and set theory equations that would fill six blackboards with gobbledy gook. Recommended reading for insomniacs.

### Bokeh Blur

The Bokeh Blur node generates a bokeh type blur similar to Defocus. Unlike defocus an in-focus region is defined in the compositor. There is also more flexibility in the type of blur applied through the [Bokeh Image](#) node.

Several performance optimizations are also available such as OpenCL support, calculation area restriction and masking.

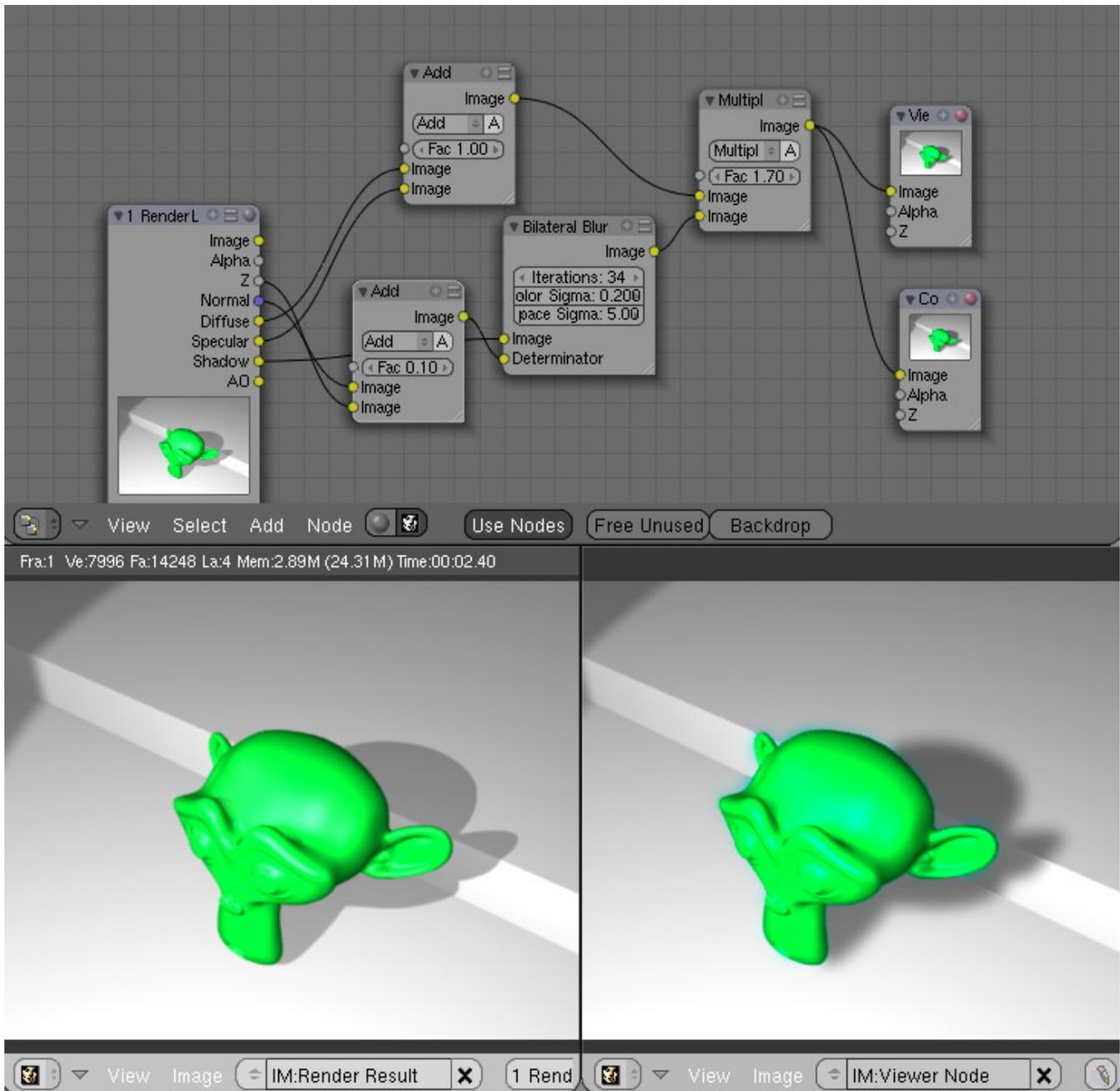


Fig. 2.2523: Bilateral smoothed buffered shadow

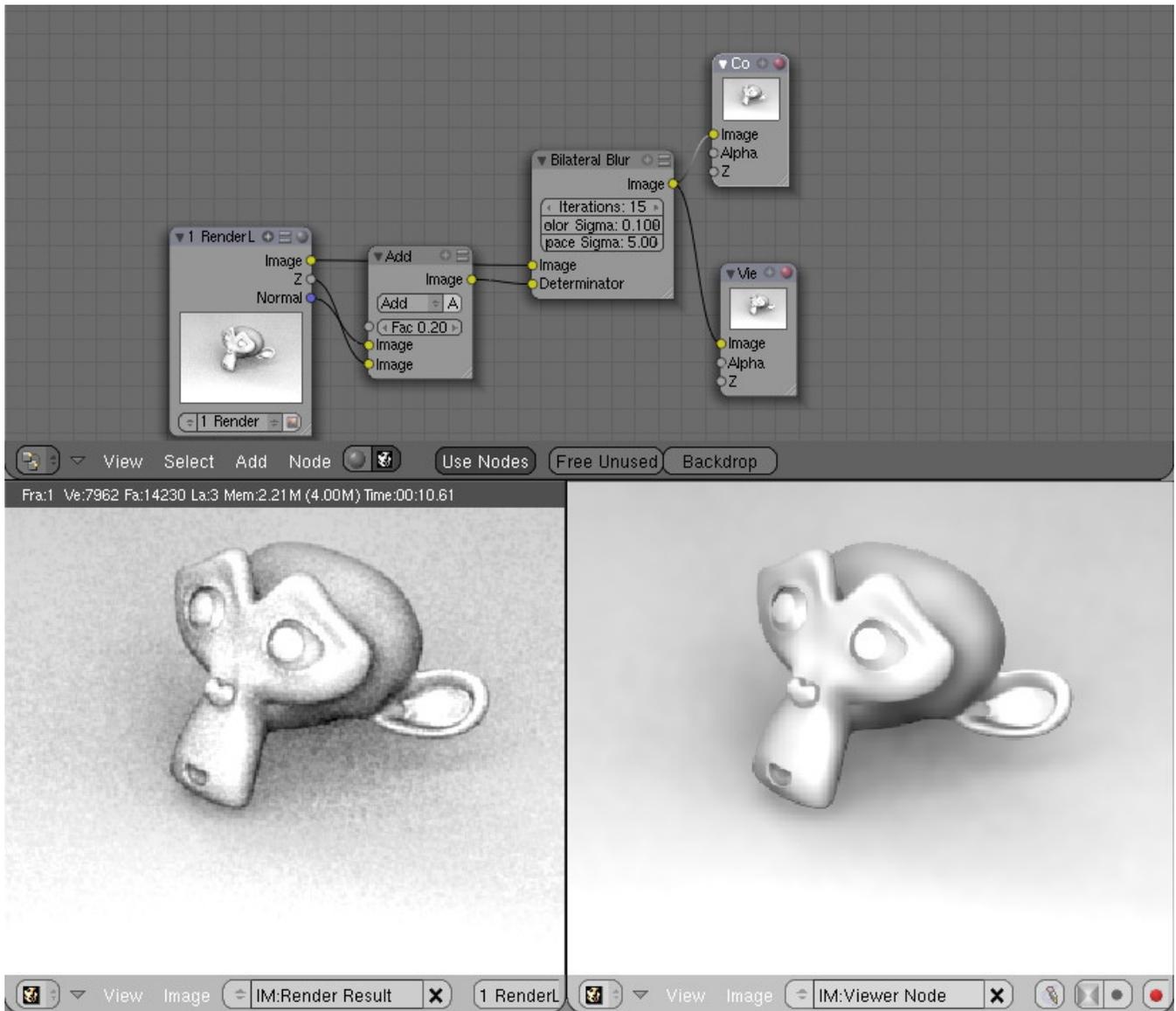


Fig. 2.2524: Bilateral smoothed AO

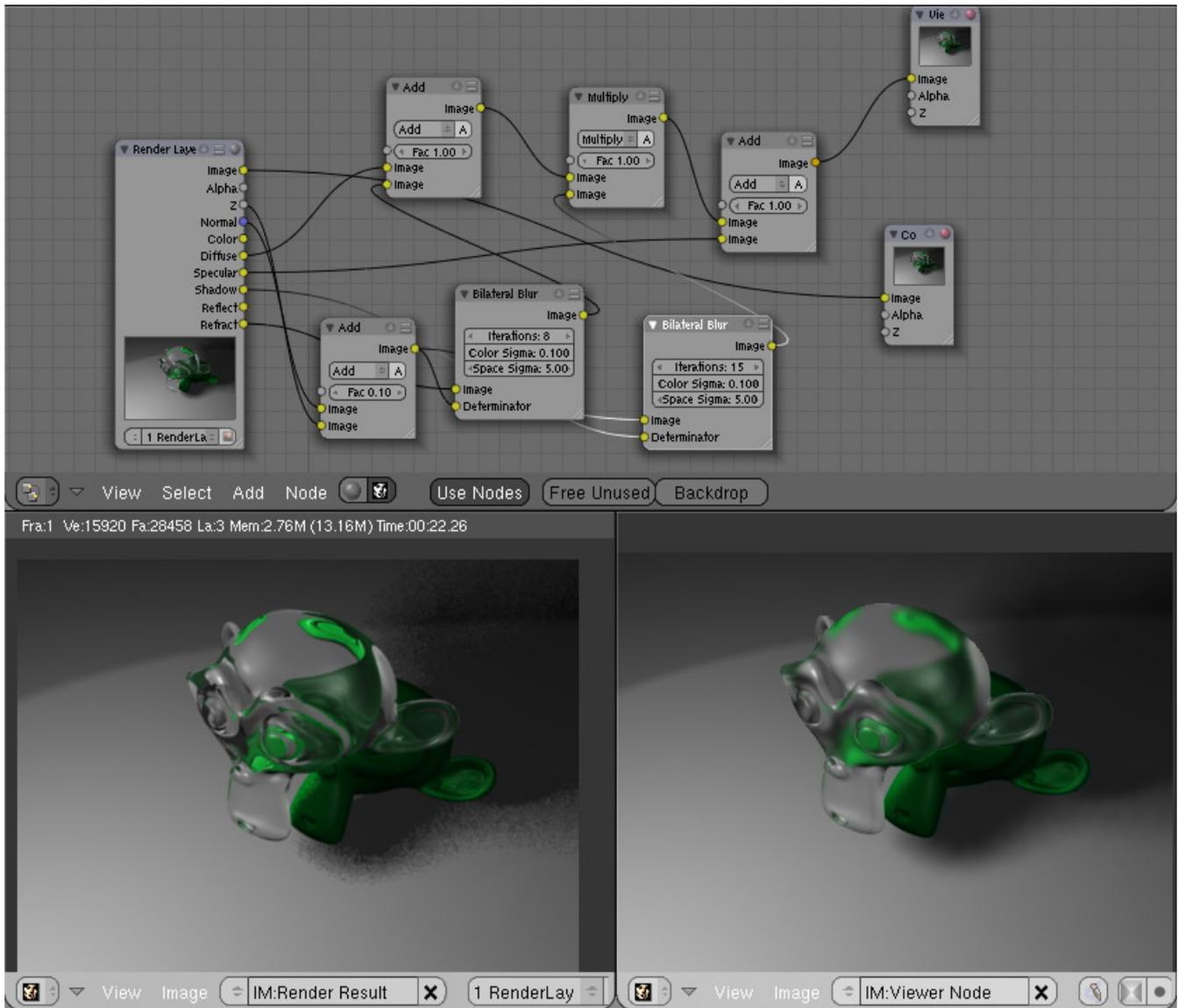


Fig. 2.2525: Bilateral faked blurry refraction+smoothed retracted soft shadow

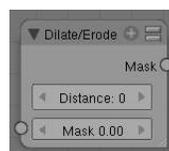


Fig. 2.2526: Dilate/Erode node

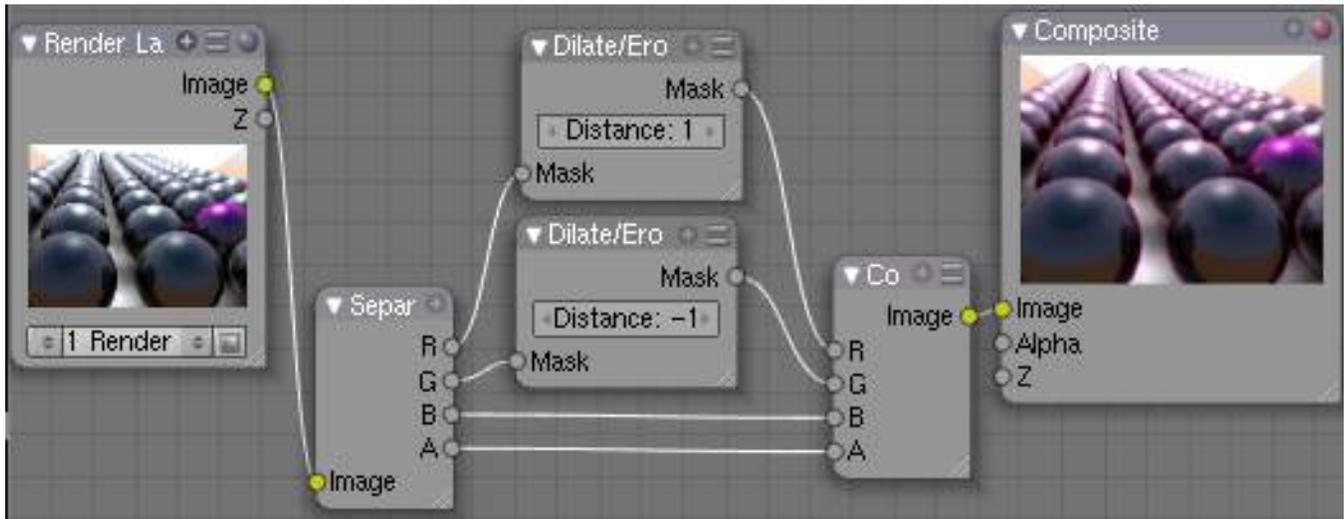


Fig. 2.2527: Magenta tinge



Fig. 2.2528: Filter node

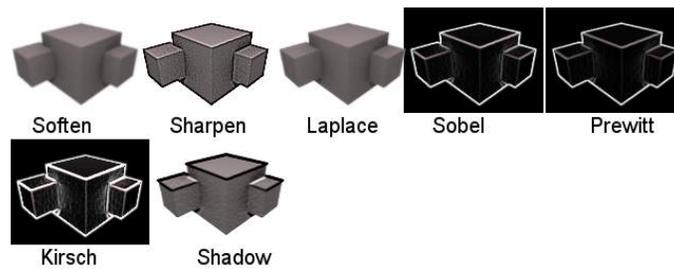


Fig. 2.2529: The Filter node has seven modes, shown here.

## Sockets

**Max blur** Max blur is intended to act as an optimization tool by limiting the number of pixels across which the blur is calculated.

**Bokeh** This is an input for the [Bokeh Image](#) node.

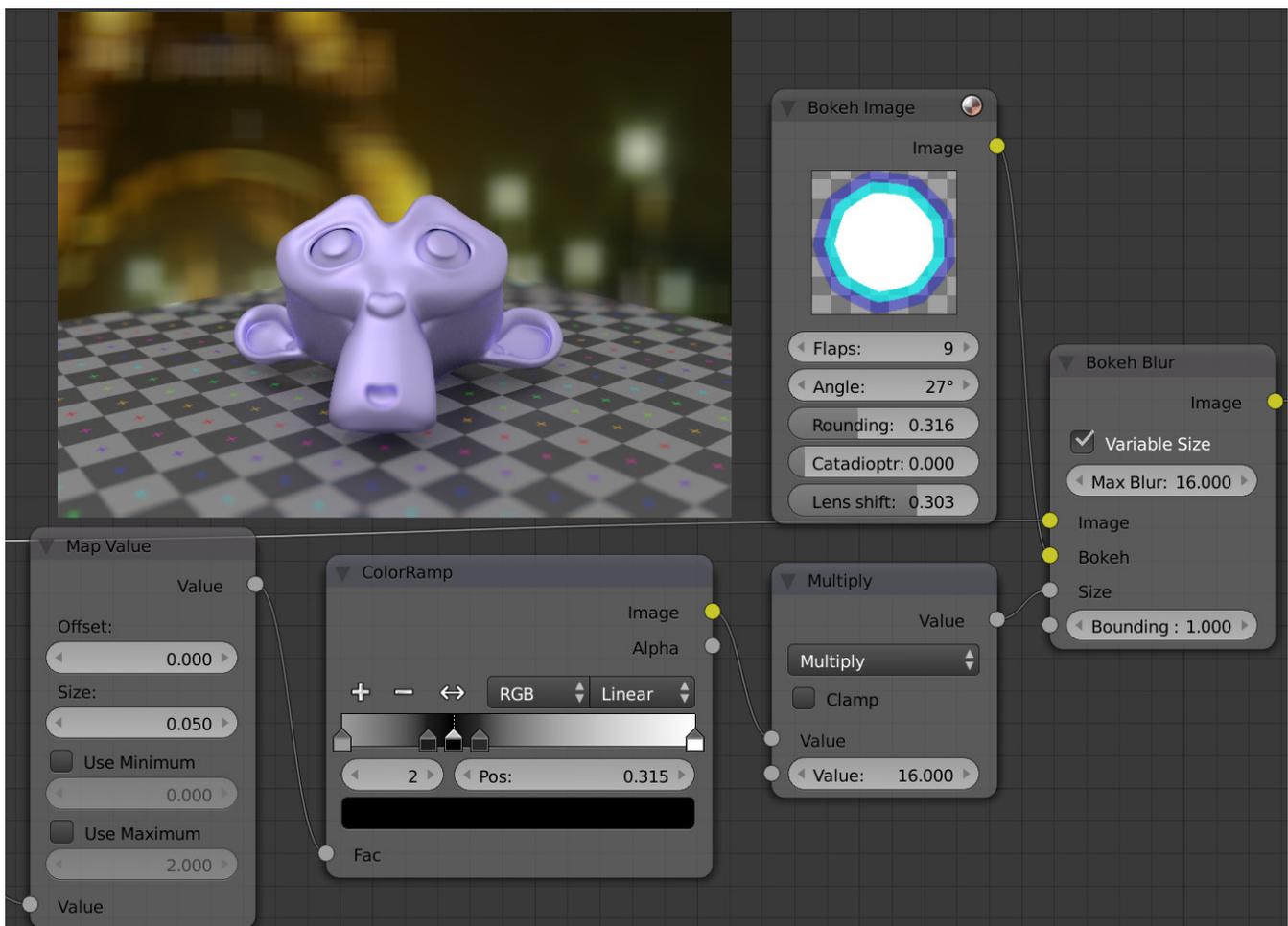
**Size** Size controls the amount of blur. Size can either be a single value across the entire image or a variable value controlled by an input image. In order to use the latter the Variable Size option must be selected. See the examples section below for more on how to use this.

**Bounding Box** This can be used with a [Box Mask](#) matte node or with a [Mask](#) input node to restrict the area of the image the blur is applied to. This could be helpful, for example, when developing a node system by allowing only a small area of the image to be filtered thus saving composite time each time adjustments are made.

**Examples** Three examples of how the size input may be used follow.

An [ID masked](#) alpha image can be used so that a background is blurred while foreground objects remain in focus. To prevent strange edges the [Dilate](#) node should be used.

The Z pass can be visualized using a [Map Value](#) node and [ColorRamp](#) node as described in [Render Layers](#) . A [multiply](#) Math node can be used following the color-ramp so that a blur value greater than 1 is used for objects outside the focal range.



A manually created greyscale image can be used to define the sharp and blurry areas of a pre existing image. Again, a [multiply](#) node can be used so that a blur value greater than 1 is used.



### Vector (Motion) Blur Node

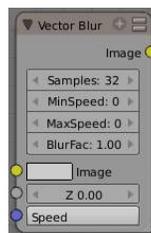


Fig. 2.2530: Vector Blur node

Motion blur is the effect of objects moving so fast they blur. Because CG animations work by rendering individual frames, they have no real knowledge of what was where in the last frame, and where it is now.

In Blender, there are two ways to produce motion blur. The first method (which produces the most correct results) works by rendering a single frame up to 16 times with slight time offsets, then accumulating these images together; this is called Motion Blur and is activated on the Render panel. The second (and much faster) method is the Compositor node Vector Blur.

To use, connect the appropriate passes from a Render Result node.

**Note:** Make sure to enable the Speed (called Vec) pass in the Render Layers panel for the render layer you wish to perform motion blur on.

**Maximum Speed:** Because of the way vector blur works, it can produce streaks, lines and other artifacts. These mostly come from pixels moving too fast; to combat these problems, the filter has minimum and maximum speed settings, which can be used to limit which pixels get blurred (e.g. if a pixel is moving really, really fast but you have maximum speed set to a moderate amount, it won't get blurred).

**Minimum Speed:** Especially when the camera itself moves, the mask created by the vectorblur node can become the entire image. A very simple solution is to introduce a small threshold for moving pixels, which can efficiently separate the hardly-moving pixels from the moving ones, and thus create nice looking masks. You can find this new option as 'min speed'. This minimum speed is in pixel units. A value of just 3 will already clearly separate the background from foreground.

**Hint:** You can make vector blur results a little smoother by passing the Speed pass through a blur node (but note that this can make strange results, so it's only really appropriate for still images with lots of motion blur).

**Examples** An in-depth look at how to use the Vector Blur node can be found [here](#).

As far as we know, this node represents a [new approach to calculating motion blur](#). Use vector blur in compositing with confidence instead of motion blur. In face, when compositing images, it is necessary to use vector blur since there isn't "real" motion. In this [example blend file](#), you will find a rigged hand reaching down to pick up a ball. Based on how the hand is moving (those vectors), the image is blurred in that direction. The fingers closest to the camera (the least Z value) are blurred more, and those farther away (the forearm) is blurred the least.

**Known Bugs** FIXME(Template Unsupported: Version;{{Version|2.44}}) Does not work when reading from a multilayer OpenEXR sequence set

## Defocus

This single node can be used to emulate depth of field using a postprocessing method. It can also be used to blur the image in other ways, not necessarily based on 'depth' by connecting something other than a Zbuffer. In essence, this node blurs areas of an image based on the input zbuffer map/mask.



Fig. 2.2531: DofDist setting for the camera.

**Camera Settings** The *Defocus* node uses the actual camera data in your scene if supplied by a *RenderLayer* node.

To set the point of focus, the camera now has a *Distance* parameter, which is shorthand for Depth of Field Distance. Use this camera parameter to set the focal plane of the camera (objects Depth of Field Distance away from the camera are in focus). Set *Distance* in the main *Camera* edit panel; the button is right below the *Depth of Field*.

To make the focal point visible, enable the camera *Limits* option, the focal point is then visible as a yellow cross along the view direction of the camera.

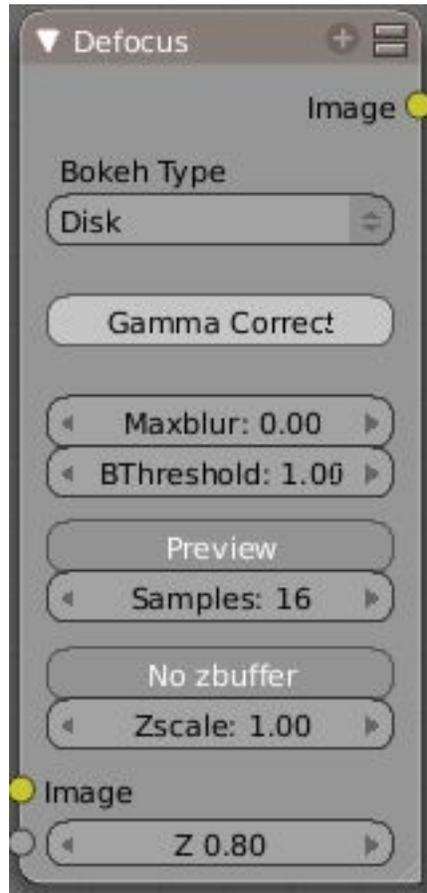


Fig. 2.2532: Defocus node

**Node Inputs** The node requires two inputs, an image and a zbuffer, the latter does not need to be an actual zbuffer, but can also be another (grayscale) image used as mask, or a single value input, for instance from a time node, to vary the effect over time.

**Node Setting** The settings for this node are:

**Bokeh Type menu** Here you set the number of iris blades of the virtual camera's diaphragm. It can be set to emulate a perfect circle (*Disk*) or it can be set to have 3 (*Triangle*), 4 (*Square*), 5 (*Pentagon*), 6 (*Hexagon*), 7 (*Heptagon*) or 8 blades (*Octagon*). The reason it does not go any higher than 8 is that from that point on the result tends to be indistinguishable from a *Disk* shape anyway.

**Rotate** This button is not visible if the *Bokeh Type* is set to *Disk*. It can be used to add an additional rotation offset to the Bokeh shape. The value is the angle in degrees.

**Gamma Correct** Exactly the same as the *Gamma* option in Blender's general *Blur* node (see [Blur Node](#)). It can be useful to further brighten out of focus parts in the image, accentuating the Bokeh effect.



Fig. 2.2533: Defocus node using Z-Buffer

**fStop** This is the most important parameter to control the amount of focal blur: it simulates the aperture  $f$  of a real lens (‘iris’) - without modifying the luminosity of the picture, however! As in a real camera, the *smaller* this number is, the more-open the lens iris is, and the *shallower* the depth-of-field will be. The default value 128 is assumed to be infinity: everything is in perfect focus. Half the value will double the amount of blur. This button is not available if *No zbuffer* is enabled.

**Maxblur** Use this to limit the amount of blur of the most out of focus parts of the image. The value is the maximum blur radius allowed. This can be useful since the actual blur process can sometimes be very slow. (The more blur, the slower it gets.) So, setting this value can help bring down processing times, like for instance when the world background is visible, which in general tends to be the point of maximum blur (not always true, objects very close to the lens might be blurred even more). The default value of 0 means there is no limit to the maximum blur amount.

**BThreshold** The defocus node is not perfect: some artifacts may occur. One such example is in-focus objects against a blurred background, which have a tendency to bleed into the edges of the sharp object. The worst-case scenario is an object in-focus against the very distant world background: the differences in distance are very large and the result can look quite bad. The node tries to prevent this from occurring by testing that the blur difference between pixels is not too large, the

value set here controls how large that blur difference may be to consider it ‘safe.’ This is all probably quite confusing, and fortunately, in general, there is no need to change the default setting of 1. Only try changing it if you experience problems around any in-focus object.

**Preview** As already mentioned, processing can take a long time. So to help make editing parameters somewhat ‘interactive’, there is a preview mode which you can enable with this button. Preview mode will render the result using a limited amount of (quasi)random samples, which is a *lot* faster than the ‘perfect’ mode used otherwise. The sampling mode also tends to produce grainy, noisy pictures (though the more samples you use, the less noisy the result). This option is on by default. Play around with the other parameters until you are happy with the results, and only then disable the preview mode for the final render.

**Samples** Only visible when *Preview* is set. Sets the amount of samples to use to sample the image. The higher, the smoother the image, but also the longer the processing time. For preview, the default of 16 samples should be sufficient and is also the fastest.

**No zbuffer** Sometimes you might want to have more control to blur the image. For instance, you may want to only blur one object while leaving everything else alone (or the other way around), or you want to blur the whole image uniformly all at once. The node therefore allows you to use something other than an actual zbuffer as the Z input. For instance, you could connect an image node and use a grayscale image where the color designates how much to blur the image at that point, where white is maximum blur and black is no blur. Or, you could use a Time node to uniformly blur the image, where the time value controls the maximum blur for that frame. It may also be used to obtain a possibly slightly-better DoF blur, by using a fake depth shaded image instead of a zbuffer. (A typical method to create the fake depth shaded image is by using a linear blend texture for all objects in the scene or by using the ‘fog/mist’ fake depth shading method.) This also has the advantage that the fake depth image can have anti-aliasing, which is not possible with a real zbuffer. *No zbuffer* will be enabled automatically whenever you connect a node that is not image based (e.g. time node/value node/etc).

**Zscale** Only visible when *No zbuffer* enabled. When *No zbuffer* is used, the input is used directly to control the blur radius. And since usually the value of a texture is only in the numeric range 0.0 to 1.0, its range is too narrow to control the blur properly. This parameter can be used to expand the range of the input (or for that matter, narrow it as well, by setting it to a value less than one). So for *No zbuffer*, this parameter therefore then becomes the main blur control (similar to *fStop* when you *do* use a zbuffer).

**Examples** In this [blend file example](#), the ball array image is blurred as if it was taken by a camera with a f-stop of 2.8 resulting in a fairly narrow depth of field centered on 7.5 blender units from the camera. As the balls recede into the distance, they get blurrier.

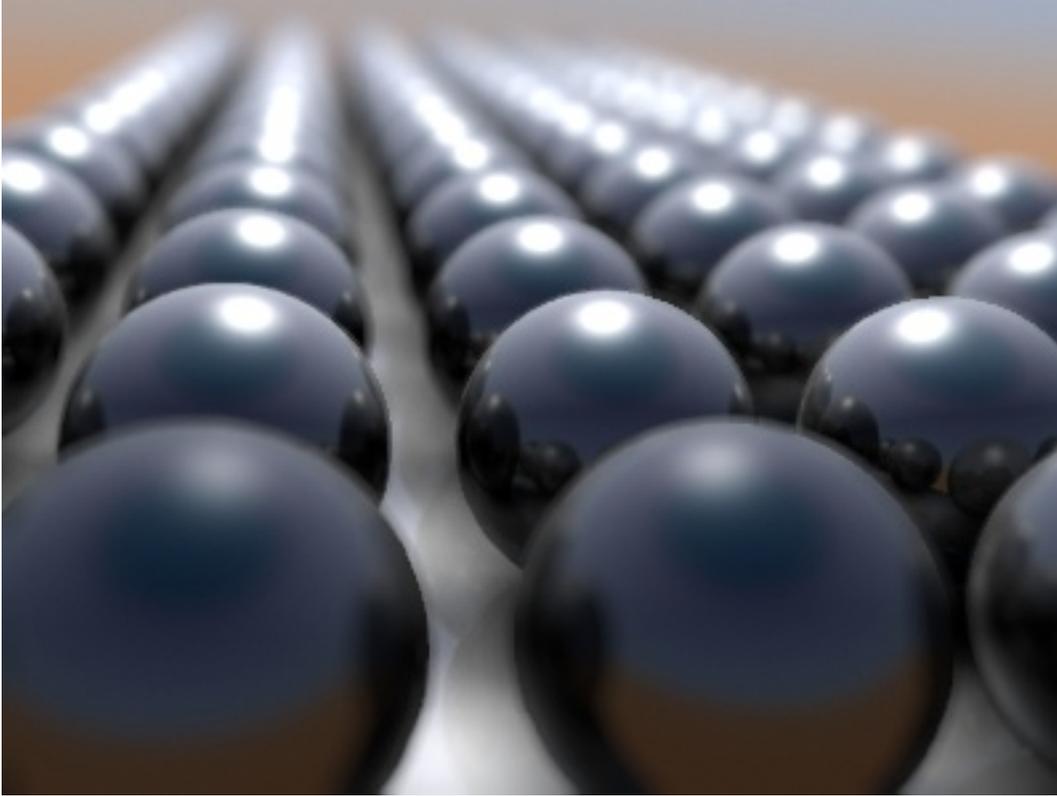
## Hints

**Preview** In general, use preview mode, change parameters to your liking, only then disable preview mode for the final render. This node is compute intensive, so watch your console window, and it will give you status as it computes each render scan line.

**Edge Artifacts** For minimum artifacts, try to setup your scene such that differences in distances between two objects that may visibly overlap at some point are not too large.

**“Focus Pull”** Keep in mind that this is not ‘real’ DoF, only a post-processing simulation. Some things cannot be done which would be no problem for real DoF at all. A typical example is a scene with some object very close to the camera, and the camera focusing on some point far behind it. In the real world, using shallow depth of field, it is not impossible for nearby objects to become completely invisible, in effect allowing the camera to see ‘behind’ it. Hollywood cinematographers use this visual characteristic to good effect to achieve the popular “focus pull” effect, where the focus shifts from a nearby to a distant object, such that the “other” object all but disappears. Well, this is simply not possible to do with the current post-processing method in a single pass. If you really want to achieve this effect, quite satisfactorily, here’s how:

- Split up your scene into “nearby” and “far” objects, and render them in two passes.
- Now, combine the two the two results, each with their own “defocus” nodes driven by the same Time node, but with one of them inverted. (e.g. using a “Map Value” node with a Size of -1.) As the defocus of one increases, the



defocus on the other decreases at the same rate, creating a smooth transition.

**Aliasing at Low f-Stop Values** At very low values, less than 5, the node will start to remove any oversampling and bring the objects at DoFDist very sharply into focus. If the object is against a contrasting background, this may lead to visible stairstepping (aliasing) which OSA is designed to avoid. If you run into this problem:

- Do your own OSA by rendering at twice the intended size and then scaling down, so that adjacent pixels are blurred together
- Use the blur node with a setting of 2 for x and y
- Set DoFDist off by a little, so that the object in focus is blurred by the tiniest bit.
- Use a higher f-Stop, which will start the blur, and then use the Z socket to a Map Value to a Blur node to enhance the blur effect.
- Rearrange the objects in your scene to use a lower-contrast background

**No ZBuffer** A final word of warning, since there is no way to detect if an actual zbuffer is connected to the node, be VERY careful with the *No ZBuffer* switch. If the *Zscale* value happens to be large, and you forget to set it back to some low value, the values may suddenly be interpreted as huge blur-radius values that will cause processing times to explode.

### Glare

TODO - see: <https://developer.blender.org/T43469>

### Inpaint

TODO - see: <https://developer.blender.org/T43469>

## Directional Blur Node

Blurs an image in a specified direction and magnitude. Can be used to fake motion blur.

### Options

**Iterations** Controls how many times the image is duplicated to create the blur effect. Higher values give smoother results.

**Wrap** Wraps the image on the X and Y axis to fill in areas that become transparent from the blur effect.

**Center** Sets the position where the blur center is. This makes a difference if the angle, spin, and/or zoom are used.

**Distance** How large the blur effect is.

**Angle** Image is blurred at this angle from the center

**Spin** Rotates the image each iteration to create a spin effect, from the center point.

**Zoom** Scales the image each iteration, creating the effect of a zoom.

### Pixelate

Add this node in front of a [scale](#) node to get a pixelated (non smoothed) image from the resultant up scaled image.

**Example** In the node editor, set the node tree to compositing in the menu bar and check the ‘Use Nodes’ checkbox. Add an input Image node and an output Viewer node. Connect the Input node to the viewer node and check the ‘Backdrop’ checkbox in the menu bar. Open an image you would like to pixelate using the open button on the image node. This image should now appear in the backdrop. Now add two scale nodes between the input and output (Add>Distort>Scale). Change the values of X and Y to 0.2 in the first scale box and to 5 in the second. The background image will be unchanged.

Now add a Pixelate node between the two scale nodes.

(note: you can use alt-v and v to zoom the backdrop in and out respectively if needed)

## Sun Beams

Sun Beams is a 2D effect for simulating the effect of bright light getting scattered in a medium ([Crepuscular Rays](#)). This phenomenon can be created by renderers, but full volumetric lighting is a rather arduous approach and takes a lot of render time. Also when working with 2D images only the volumetric data may not be available. In these cases the “Sun Beams” node provides a computationally cheap way of creating a convincing effect based on image brightness alone.

**Usage** Usually the first step is to define the area from which rays are cast. Any diffuse reflected light from surfaces is not going to contribute to such scattering in the real world, so should be excluded from the input data. Possible ways to achieve this are

- entirely separate image as a light source
- brightness/contrast tweaking to leave only the brightest areas
- muting shadow and midtone colors, which is a bit more flexible
- masking for ultimate control

After generating the sun beams from such a light source image they can then be overlayed on the original image. Usually a simple “Add” mix node is sufficient, and physically correct because the scattered light adds to the final result.





## Vector Nodes

These nodes can be used to manipulate various types of vectors, such as surface normals and speed vectors.

### Normal Node

The Normal node generates a normal vector and a dot product. Click and Drag on the sphere to set the direction of the normal.

This node can be used to input a new normal vector into the mix. For example, use this node as an input to a Color Mix node. Use an Image input as the other input to the Mixer. The resulting colored output can be easily varied by moving the light source (click and dragging the sphere).

### Map Value Node

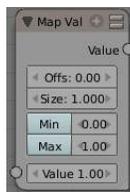


Fig. 2.2534: Map Value node

Map Value node is used to scale, offset and clamp values (value refers to each vector in the set). The formula for how this node works is:

**Offs** will add a number to the input value

**Size** will scale (multiply) that value by a number

**Min/Max** you can set the minimum and maximum numbers to clamp (cut off) the value too. Min and Max must be individually enabled by LMB clicking on the label for them to clamp. **Shift-LMB** on the value to change it.

- If Min is enabled and the value is less than Min, set the output value to Min
- If Max is enabled and the input value is greater than Max, set the output value to Max

This is particularly useful in achieving a depth-of-field effect, where you can use the Map Value node to map a Z value (which can be 20 or 30 or even 500 depending on the scene) to range between 0-1, suitable for connecting to a Blur node.

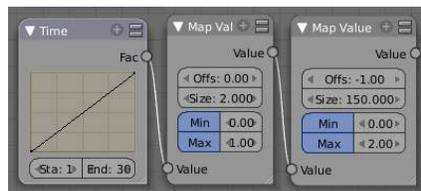


Fig. 2.2535: Using Map Value to multiply

**Using Map Value to Multiply values** You can also use the map value node to multiply values to achieve an output number that you desire. In the mini-map to the right, the Time node outputs a value between 0.00 and 1.00 evenly scaled over 30 frames. The *first* Map Value node multiplies the input by 2, resulting in an output value that scales from 0.00 to 2.00 over 30 frames. The *second* Map Value node subtracts 1 from the input, giving working values between -1.00 and 1.00, and multiplies that by 150, resulting in an output value between -150 and 150 over a 30-frame sequence.

## Map Range

TODO - see: <https://developer.blender.org/T43469>

## Normalize

Normalizing a vector scales its magnitude, or length, to a value of 1, but keeps its direction intact.

## Vector Curves Node



The Vector Curves node maps an input vector image's x, y, and z components to a diagonal curve. The three channels are accessed via the X, Y, and Z buttons at the top of the node. Add points to the curve by clicking on it.

Note that dragging a point across another will switch the order of the two points (e.g. if point A is dragged across point B, then point B will become point A and point A will become point B).

Use this curve to slow things down or speed them up from the original scene.

## Matte Nodes

These nodes give you the essential tools for working with blue-screen or green-screen footage, where live action is shot in front of a blue or green backdrop for replacement by a matte painting or virtual background.

In general, hook up these nodes to a viewer, set your UV/Image Editor to show the viewer node, and play with the sliders in real-time using a sample image from the footage, to get the settings right. In some cases, small adjustments can eliminate artifacts or foreground image degradation. For example, taking out too much green can result in foreground actors looking 'flat' or blueish/purplish.

You can and should chain these nodes together, refining your color correction in successive refinements, using each node's strengths to operate on the previous node's output. There is no "one stop shopping" or one "does-it-all" node; they work best in combination.

Usually, green screen is shot in a stage with consistent lighting from shot to shot, so the same settings will work across multiple shots of raw footage. Footage shot outside under varying lighting conditions (and wind blowing the background) will complicate matters and mandate lower falloff values.

---

### Note: Garbage Matte

Garbage matte is not a node, but a technique where the foreground is outlined using a closed curve (bezier or nurbs). Only the area within the curve is processed using these matte nodes; everything else is garbage and thus discarded.

## Keying

TODO - see: <https://developer.blender.org/T43469>

## Keying Screen

TODO - see: <https://developer.blender.org/T43469>

## Channel Key Node



Fig. 2.2536: Channel Key node

The *Channel Key* node determines background objects from foreground objects by the difference in the selected channel's levels. For example in YUV color space, this is useful when compositing stock footage of explosions (very bright) which are normally shot against a solid, dark background.

There is one input to this node, the *Image* that is to be keyed.

Control this node using:

**Color Space** buttons selects what color space the channels will represent.

**Channel** buttons selects the channel to use to determine the matte.

**High** value selector determines the lowest values that are considered foreground. (which is supposed to be - relatively - height values: from this value to 1.0).

**Low** value selector determines the highest values that are considered to be background objects. (which is supposed to be - relatively - low values: from 0.0 to this value).

It is possible to have a separation between the two values to allow for a gradient of transparency between foreground and background objects.

The outputs of this node are the *Image* with an alpha channel adjusted for the keyed selection and a black and white *Matte* (i.e the alpha mask).

### Color Spill Node



Fig. 2.2537: Color Spill node

The *Color Spill* node reduces one of the RGB channels so that it is not greater than any of the others. This is common when compositing images that were shot in front of a green or blue screen. In some cases, if the foreground object is reflective, it will show the green or blue color; that color has “spilled” onto the foreground object. If there is light from the side or back, and the foreground actor is wearing white, it is possible to get “spill” green (or blue) light from the background onto the foreground objects, coloring them with a tinge of green or blue. To remove the green (or blue) light, you use this fancy node.

There is one input to this node, the *Image* to be processed.

The *Enhance* slider allows you to reduce the selected channel’s input to the image greater than the color spill algorithm normally allows. This is useful for exceptionally high amounts of color spill.

The outputs of this node are the image with the corrected channels.

### Box Mask

TODO - see: <https://developer.blender.org/T43469>

## Ellipse Mask

TODO - see: <https://developer.blender.org/T43469>

## Luminance Key Node

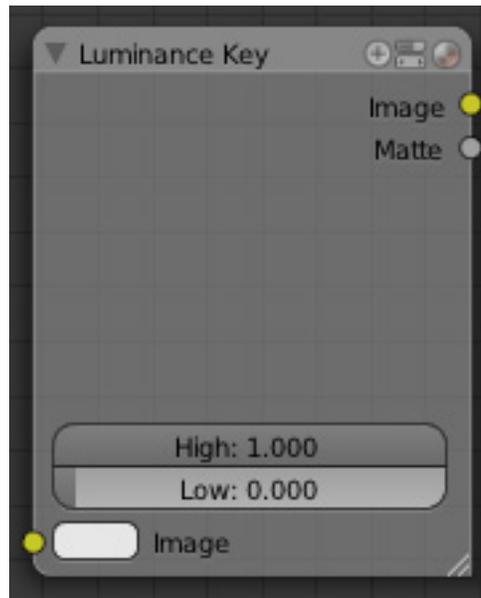


Fig. 2.2538: Luminance Key node

The *Luminance Key* node determines background objects from foreground objects by the difference in the luminance (brightness) levels. For example, this is useful when compositing stock footage of explosions (very bright) which are normally shot against a solid, dark background.

There is one input to this node, the *Image* that is to be keyed.

Control this node using:

- The *High* value selector determines the lowest values that are considered foreground. (which is supposed to be - relatively - light: from this value to 1.0).
- The *Low* value selector determines the highest values that are considered to be background objects. (which is supposed to be - relatively - dark: from 0.0 to this value).

It is possible to have a separation between the two values to allow for a gradient of transparency between foreground and background objects.

The outputs of this node are the *Image* with an alpha channel adjusted for the keyed selection and a black and white *Matte* (i.e. the alpha mask).

**Example** For this example, let's throw you a ringer. Here, the model was shot against a *white* background. Using the Luminance Key node, we get a matte out where the background is white, and the model is black; the opposite of what we want. If we wanted to use the matte, we have to switch the white and the black. How to do this? ColorRamp to the rescue - we set the left color White Alpha 1.0, and the right color to be Black Alpha 0.0. Thus, when the Colorramp gets in black, it spits out white, and vice versa. The reversed mask is shown; her white outline is useable as an alpha mask now.

Now to mix, we don't really need the AlphaOver node; we can just use the mask as our Factor input. In this kinda weird case, we can use the matte directly; we just switch the input nodes. As you can see, since the matte is white (1.0) where we don't

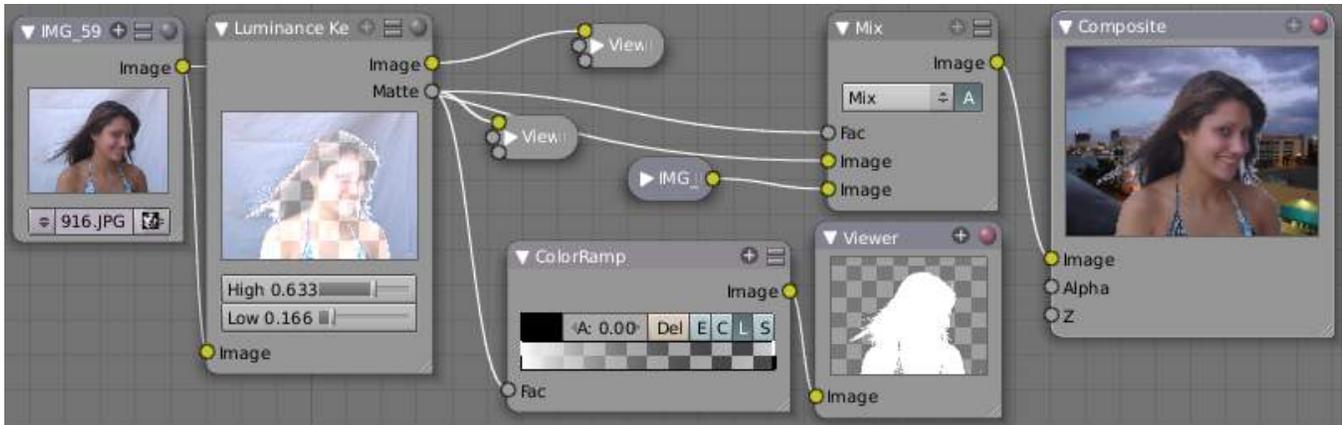


Fig. 2.2539: Using Luma Key...with a twist

want to use the model picture, we feed the background photo to the bottom socket (recall the mix node uses the top socket where the factor is 0.0, and the bottom socket where the factor is 1.0). Feeding our original photo into the top socket means it will be used where the Luminance Key node has spit out Black. Voila, our model is teleported from Atlanta to aboard a cruise ship docked in Miami.

### Difference Key Node

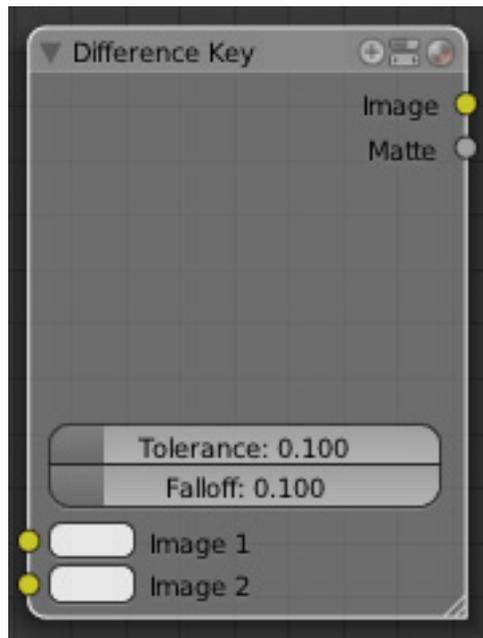


Fig. 2.2540: Difference Key node

The difference key node determines how different each channel is from the given key in the selected color space. If the differences are below a user defined threshold then the pixel is considered transparent. Difference matting does not rely on a certain background color, but can have less than optimal results if there is a significant amount of background color in the foreground object.

There are two inputs to this node.

- The first is an input *Image* that is to be keyed.
- The *Key Color* can be input as an RGB value or selected using the color picker by clicking on the *Key Color* box to bring up the color dialog, then clicking on the eye dropper tool and selecting a color.

The selectable color spaces are *RGB* (default), *HSV*, *YUV*, and *YCbCr*.

You can adjust the tolerance of each color in the colorspace individually so that you can have more red variance or blue variance in what you would allow to be transparent. I find that about 0.15 (or 15%) is plenty of variance if the background is evenly lit. Any more unevenness and you risk cutting into the foreground image.

When the *Falloff* value is high, pixels that are close to the *Key Color* are more transparent than pixels that are not as close to the *Key Color* (but still considered close enough to be keyed). When the *Falloff* value is low, it does not matter how close the pixel color (*Image*) is to the *Key Color*, it is transparent.

The outputs of this node are the *Image* with an alpha channel adjusted for the keyed selection and a black and white *Matte* (i.e. the alpha mask).

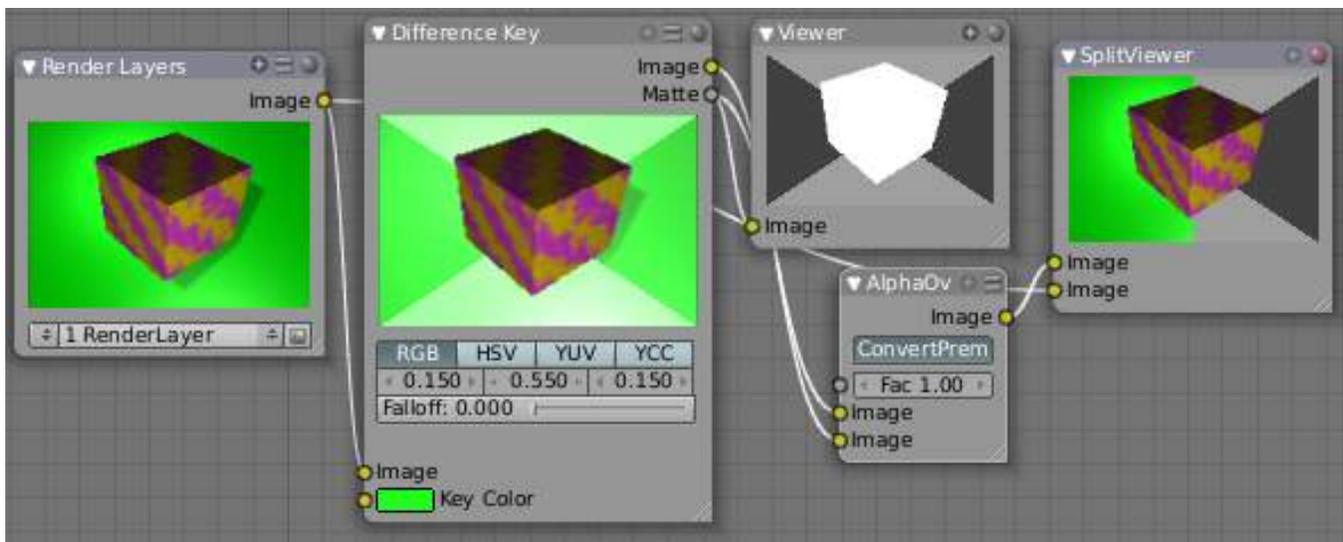


Fig. 2.2541: Using the Difference Key Node

**Simple Example** In the example to the right (click to expand), we have a purple cube with yellow marbling in front of a very unevenly lit green screen. We start building our noodle by threading the image to a difference key, and using the eyedropper, pick a key color very close to the edge of the cube, around where the halo is at the corner on the left-hand side; a fairly bright green. We thread two viewers from the output sockets so we can see what (if anything) the node is doing. We add an AlphaOver node, threading the Matte to the **TOP** socket and the image to the **BOTTOM** socket. Very Important, because 0 time blue is not the same as blue times zero. You always want your mask to go to the top socket of the AlphaOver. Premultiply is set and a full multiply is on so that we completely remove the green. In this example, we thread the output of the alphaover to a SplitViewer node so we can compare our results; the original is threaded to the bottom input of the SplitViewer, so that original is on the left, processed is on the right.

We set our variance to .15, and see what we get. What we get (not shown) is a matte that masks around the cube, but not on the right and around the edges where the green is darker; that shade it is too far away from our key color. So, since it is the green that is varying that we want to remove, we increase the Green variation to 1.00 (not shown). Whoa! All the Green disappears (all green within a 100% variation of our green key color is *all* the green), along with the top of the box! Not good. So, we start decreasing the green until we settle on 55% (shown).

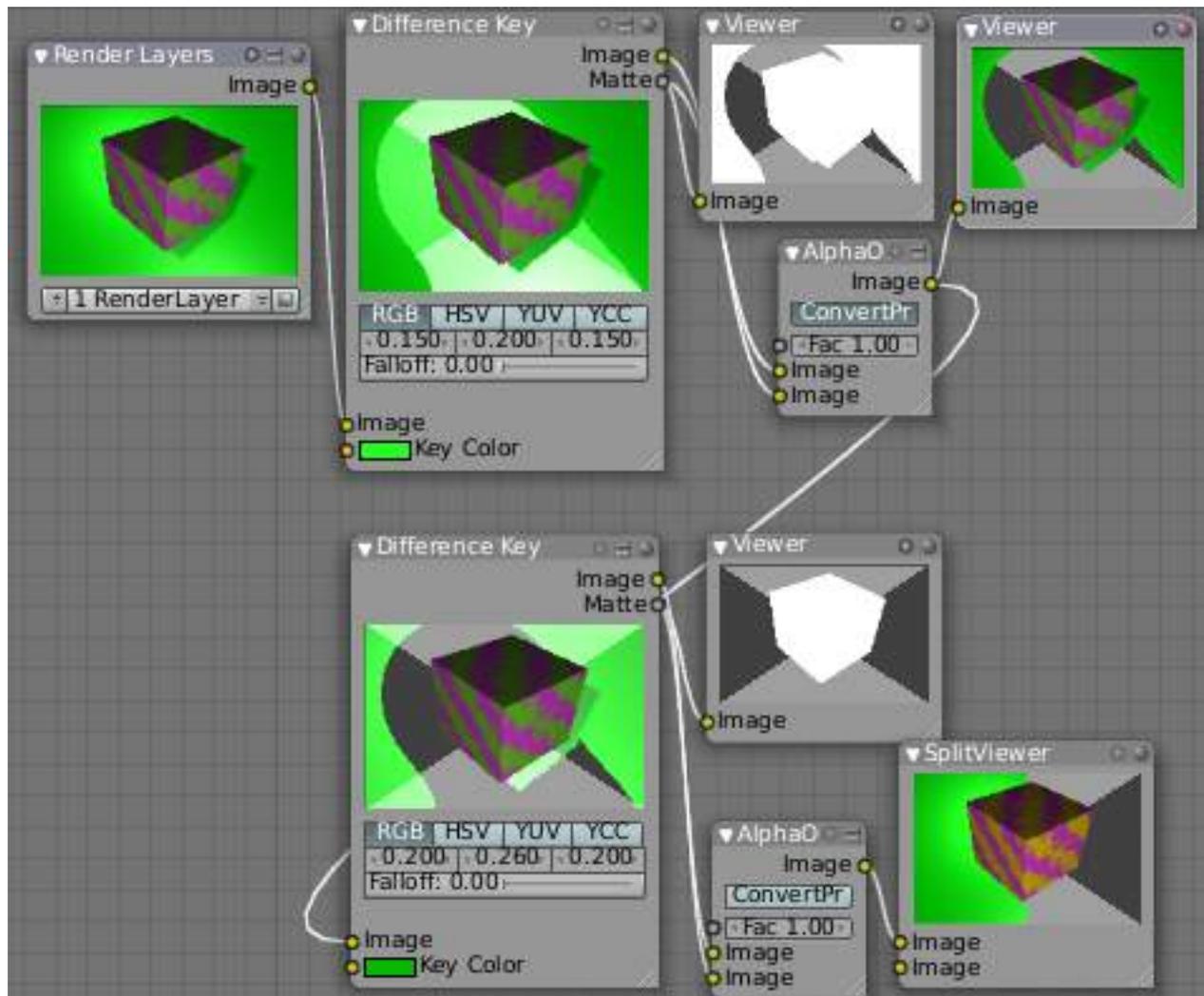


Fig. 2.2542: Chaining Difference Key Nodes

**Chaining Example** We pay out the wazoo for our highly talented (and egotistical I might add) Mr. Cube to come into the studio and do a few takes. We told him NOT to wear a green tie, but when we look at our footage, lo and behold, there he is with a green striped tie on. When we use our simple noodle, the green stripes on his tie go alpha, and the beach background shows through. So, we call him up and, too late, he's on his way back to Santa Monica and it wasn't in his contract and it wasn't his fault, after all, we're supposed to have all this fancy postpro software yada yada and he hangs up. Geez, these actors.

So, we chain two Difference Key nodes as shown to the right, and problem solved. What we did was lower the variation percentage on the first to remove some of the green, then threaded that to a second (lower) difference key, where we sampled the green more toward the shadow side and outside edge. By keeping both variations low, none of the green in his tie is affected; that shade is outside the key's +/- variation tolerances.

## Distance Key

TODO - see: <https://developer.blender.org/T43469>

## Chroma Key Node

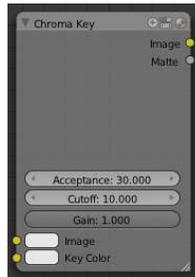


Fig. 2.2543: Chroma Key node

The *Chroma Key* node determines if a pixel is foreground or background (and thereby should be transparent) based on its chroma values. This is useful for compositing images that have been shot in front of a green or blue screen.

There is one input to this node, the *Image* that is to be keyed.

Control this node using:

**Green / Blue buttons** Basic selection of what color the background is supposed to be.

**Cb Slope and Cr Slope (chroma channel) sliders** Determines how quickly the processed pixel values go from background to foreground, much like falloff.

**Cb Pos and Cr Pos sliders** Determines where the processing transition point for foreground and background is in the respective channel.

**Threshold** Determines if additional detail is added to the pixel if it is transparent. This is useful for pulling shadows from an image even if they are in the green screen area.

**Alpha threshold** The setting that determines the tolerance of pixels that should be considered transparent after they have been processed. A low value means that only pixels that are considered totally transparent will be transparent, a high value means that pixels that are mostly transparent will be considered transparent.

The outputs of this node are the *Image* with an alpha channel adjusted for the keyed selection and a black and white *Matte* (i.e the alpha mask).

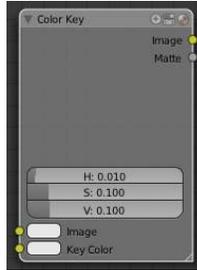


Fig. 2.2544: Color Key node

### Color Key

The color key node creates a matte based on a specified color of the input image. The sliders represent threshold values for *Hue*, *Saturation*, and *Value*. Higher values in this node's context mean a wider range of colors from the specified will be added to the matte.

### Double Edge Mask

TODO - see: <https://developer.blender.org/T43469>

### Distort Nodes

These nodes distort the image in some fashion, operating either uniformly on the image, or by using a mask to vary the effect over the image.

### Scale Node



Fig. 2.2545: Scale node

This node scales the size of an image. Scaling can be either absolute or relative. If Absolute toggle is on, you can define the size of an image by using real pixel values. In relative mode percents are used.

For instance X: 0.5 and Y: 0.5 would produce image which width and height would be half of what they used to be. When expanding an image greatly, you might want to blur it somewhat to remove the square corners that might result. Unless of course you want that effect; in which case, ignore what I just said.

Use this node to match image sizes. Most nodes produce an image that is the same size as the image input into their top image socket. So, if you want to uniformly combine two images of different size, you must scale the second to match the resolution of the first.

### Lens Distortion

Use this node to simulate distortions that real camera lenses produce.

**Distort** This creates a bulging or pinching effect from the center of the image.

**Dispersion** This simulates chromatic aberration, where different wavelengths of light refract slightly differently, creating a rainbow colored fringe.

**Projector** Enable or disable slider projection mode. When on, distortion is only applied horizontally. Disables *Jitter* and *Fit*.

**Jitter** Adds jitter to the distortion. Faster, but noisier.

**Fit** Scales image so black areas are not visible. Only works for positive distortion.

### Movie Distortion

TODO - see: <https://developer.blender.org/T43469>

### Translate Node

The translate node translates (moves) an image by the specified amounts in the X and Y directions. X and Y are in pixels, and can be positive or negative. To shift an image up and to the left, for example, you would specify a negative X offset and a positive Y.

**Usage** This node can be used for:

- Movie credits.
- Moving a matte.
- Camera shake.

### Rotate Node

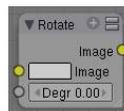


Fig. 2.2546: Rotate node

This node rotates an image. Positive values rotate clockwise and negative ones counterclockwise.

### Flip Node



Fig. 2.2547: Flip node

This node flips an image at defined axis that can be either X or Y. Also flipping can be done on both X and Y axis' simultaneously.

You can use this node to just flip or use it as a part of mirror setting. Mix half of the image to be mirrored with its flipped version to produce mirrored image.

## Crop Node

The Crop Node takes an input image and crops it to a selected region.

**Crop Image Size** When enabled, the image size is cropped to the specified region. When disabled, image remains the same size, and uncropped areas become transparent pixels.

**Relative** When enabled, crop dimensions are a percentage of the image's width and height. When disabled, the range of the sliders are the width and height of the image in pixels.

**Crop Region Values** These sliders define the lower, upper, left, and right borders if the crop region.

## Displace Node

Ever look down the road on a hot summer day? See how the image is distorted by the hot air? That's because the light is being bent by the air; the air itself is acting like a lens. This fancy little node does the same thing; it moves an input image's pixels based on an input vector mask (the vector mask mimics the effect of the hot air).

This can be useful for a lot of things, like hot air distortion, quick-and-dirty refraction, compositing live footage behind refracting objects like looking through bent glass or glass blocks, and more! Remember what HAL saw in 2001:Space Odyssey; that distorted wide-angle lens? Yup, this node can take a flat image and apply a mask to produce that image.

The amount of displacement in the X and Y directions is determined by

- The value of the mask's channels:
- The scaling of the mask's channels

The (red) channel 1's value determines displacement along the positive or negative X axis. The (green) channel 2's value determines displacement along the positive or negative Y axis.

If both the channels' values are equal (i.e. a greyscale image), the input image will be displaced equally in both X and Y directions, and also according to the X scale and Y scale buttons. These scale button act as multipliers to increase or decrease the strength of the displacement along their respective axes. They need to be set to non-zero values for the node to have any effect.

Because of this, you can use the displace node in two ways, with a greyscale mask (easy to paint, or take from a procedural texture), or with a vector channel or RGB image, such as a normal pass, which will displace the pixels based on the normal direction.

**Example** In this example, she's singing about dreams of the future. So, to represent this, we use a moving clouds texture (shot just by rendering the cloud texture on a moving plane) as the displacement map. Now, the colors in a black and white image go from zero (black) to one (white), which, if fed directly without scaling would only shift the pixels one position. So, we scale their effect in the X and Y direction.

Upon reviewing it, sometimes stretching in both the X and Y direction made her face look fat, and we all can guess her reaction to looking fat on camera. SO, we scale it only half as much in the X so her face looks longer and thinner. Now, a single image does not do justice to the animation effect as the cloud moves, and this simple noodle does not reflect using blur and overlays to enhance (and complicate) the effect, but this is the core.

Photos courtesy of Becca, no rights reserved. See also some movies of this node in action, made by the wizard programmer himself, by following this [external link](#)

## Map UV Node

So, I think we all agree that the problem is...we just don't know what we want. The same is true for directors. Despite our best job texturing our models, in post production, inevitably the director changes their mind. "Man, I really wish he looked more

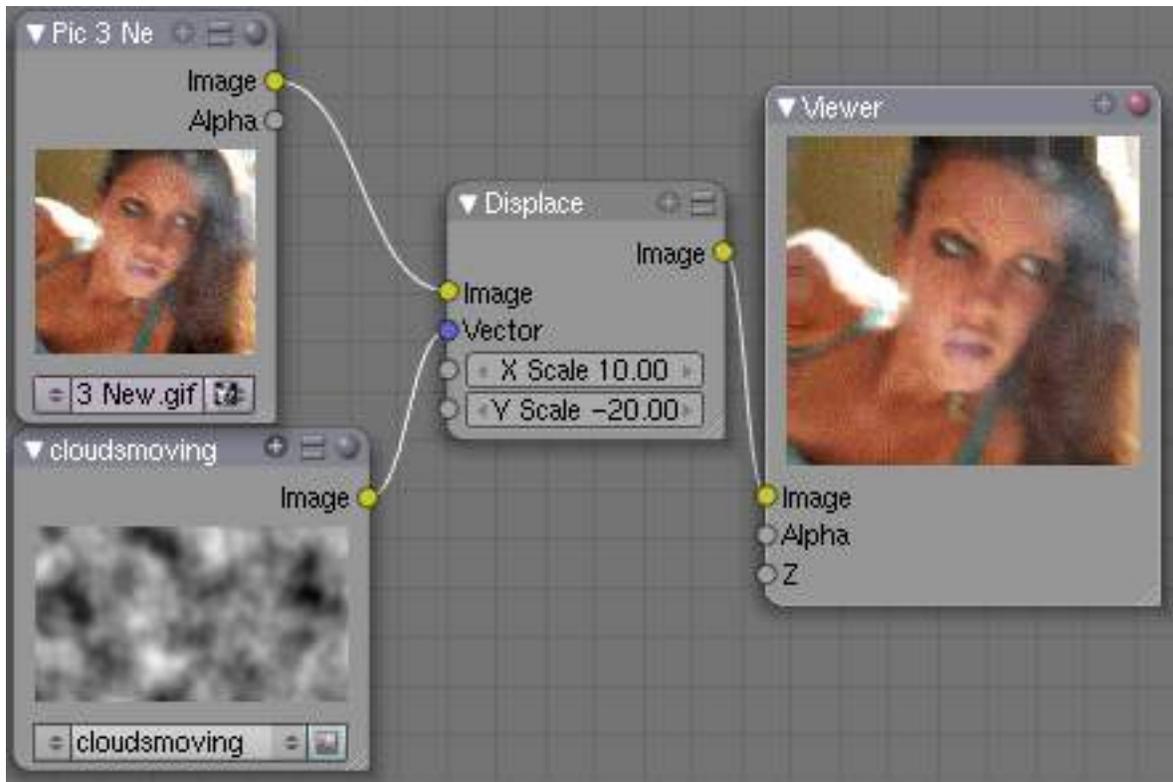
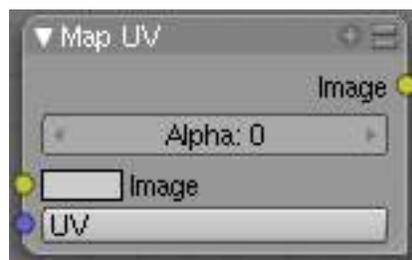


Fig. 2.2548: Music Video Distortion Example Using Displace



ragged. Who did makeup, anyway?” comes the remark. While you can do quite a bit of coloring in post production, there are limits. Well, now this little node comes along and you have the power to **re-texture your objects** *after they have been rendered*. Yes, you read that right; it’s not a typo and I’m not crazy. At least, not today.

Using this node (and having saved the UV map in a multilayer OpenEXR format image sequence), you can apply new flat image textures to all objects (or individual objects if you used the very cool [ID Mask Node](#) to enumerate your objects) in the scene.

Thread the new UV Texture to the Image socket, and the UV Map from the rendered scene to the UV input socket. The resulting image is the input image texture distorted to match the UV coordinates. That image can then be overlay mixed with the original image to paint the texture on top of the original. Adjust alpha and the mix factor to control how much the new texture overlays the old.

Of course, when painting the new texture, it helps to have the UV maps for the original objects in the scene, so keep those UV texture outlines around even after all shooting is done.

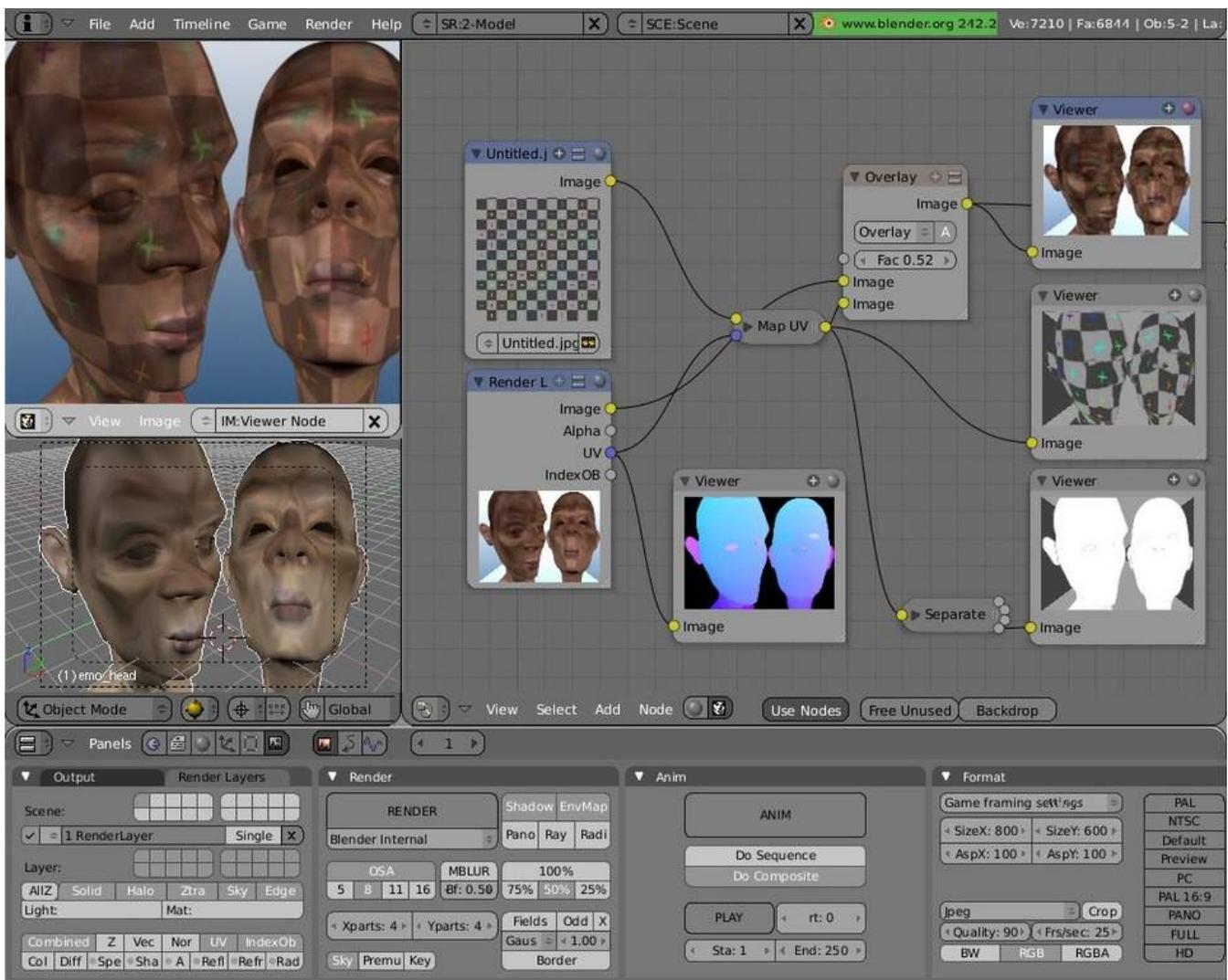


Fig. 2.2549: Adding a Grid UV Textures for Motion Tracking

**Examples** In the example to the right, we have overlaid a grid pattern on top of the two Emo heads after they have been rendered. During rendering, we enabled the UV layer in the RenderLayer tab (Buttons window, Render Context, RenderLayer

tab). Using a mix node, we mix that new UV Texture over the original face. We can use this grid texture to help in any motion tracking that we need to do.

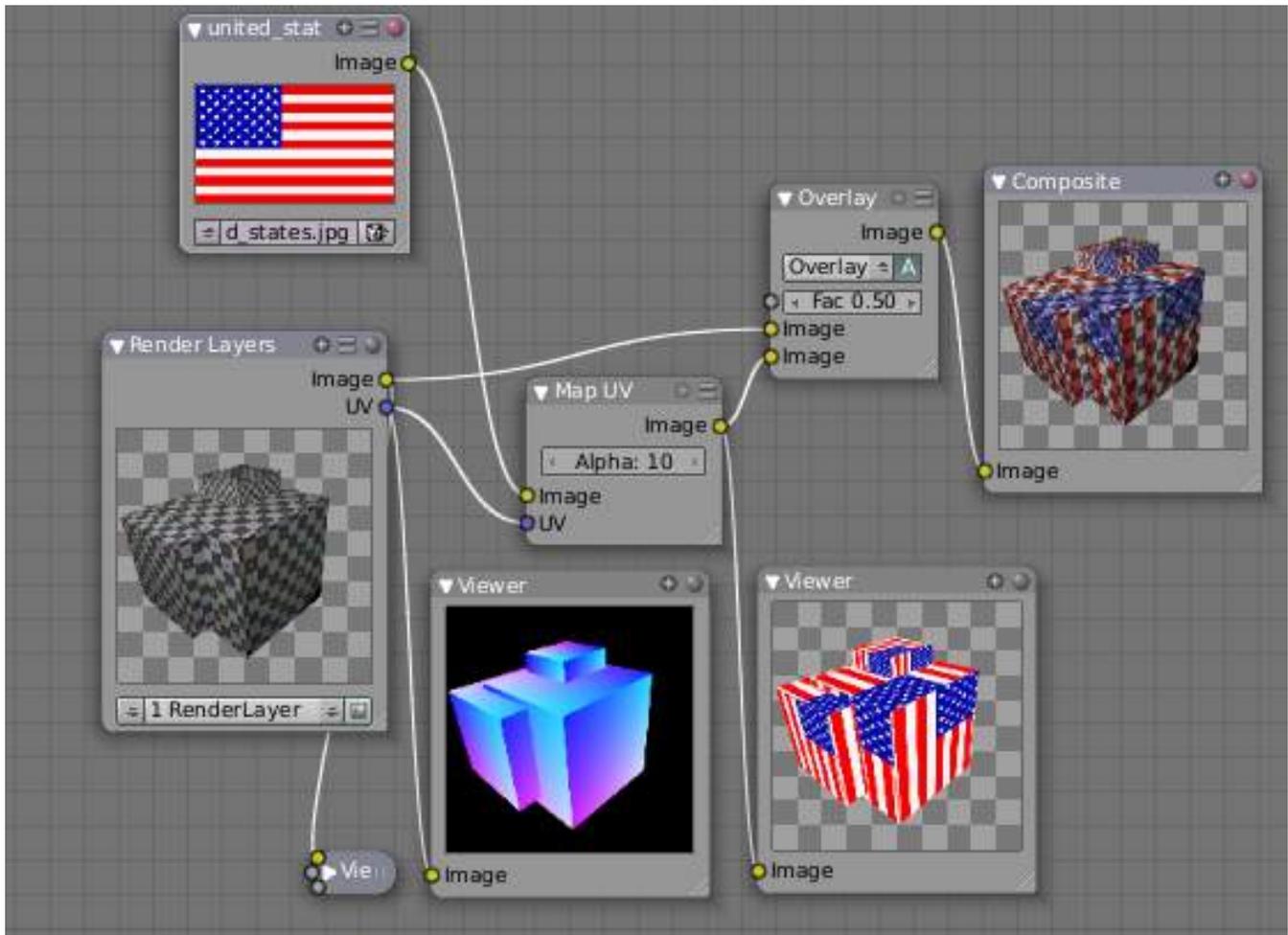


Fig. 2.2550: Adding UV Textures in Post-Production

In this example, we overlay a flag on top of a cubie-type thing, and we ensure that we Enable the Alpha pre-multiply button on the Mix node. The flag is used as additional UV Texture on top of the grid. Other examples include the possibility that we used an unauthorized product box during our initial animation, and we need to substitute in a different product sponsor after rendering.

Of course, this node does NOT give directors the power to rush pre-production rendering under the guise of “we’ll fix it later”, so maybe you don’t want to tell them about this node. Let’s keep it to ourselves for now.

### Transform Node

This node combines the functionality of three other nodes: [Scale](#), [translate](#), and [rotate](#) nodes.

**X, Y** Used to move the input image horizontally and vertically.

**Angle** Used to rotate an image around its center. Positive values rotate counter-clockwise and negative ones clockwise.

**Scale** Used to resize the image. The scaling is relative, meaning a value of 0.5 gives half the size and a value of 2.0 gives twice the size of the original image.

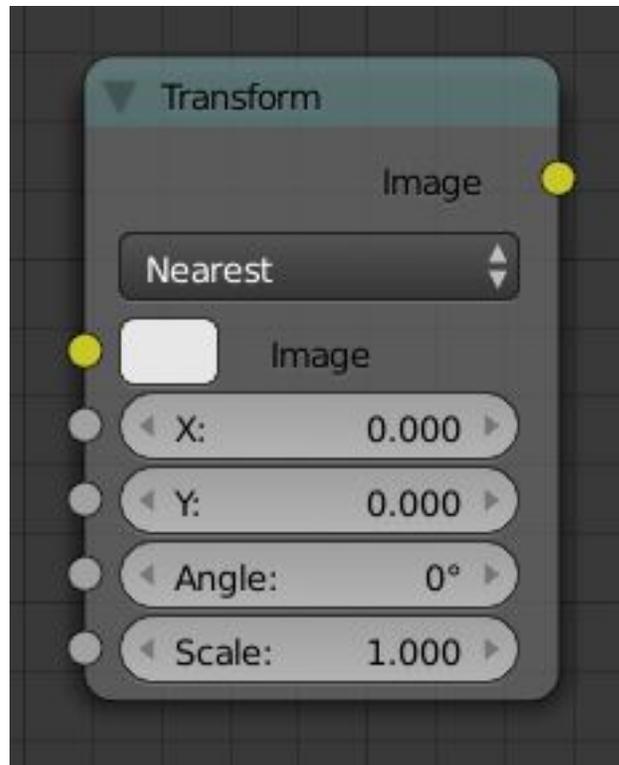


Fig. 2.2551: Transform node

**Stabilize 2D**

TODO - see: <https://developer.blender.org/T43469>

**Plane Track Deform**

TODO - see: <https://developer.blender.org/T43469>

**Corner Pin**

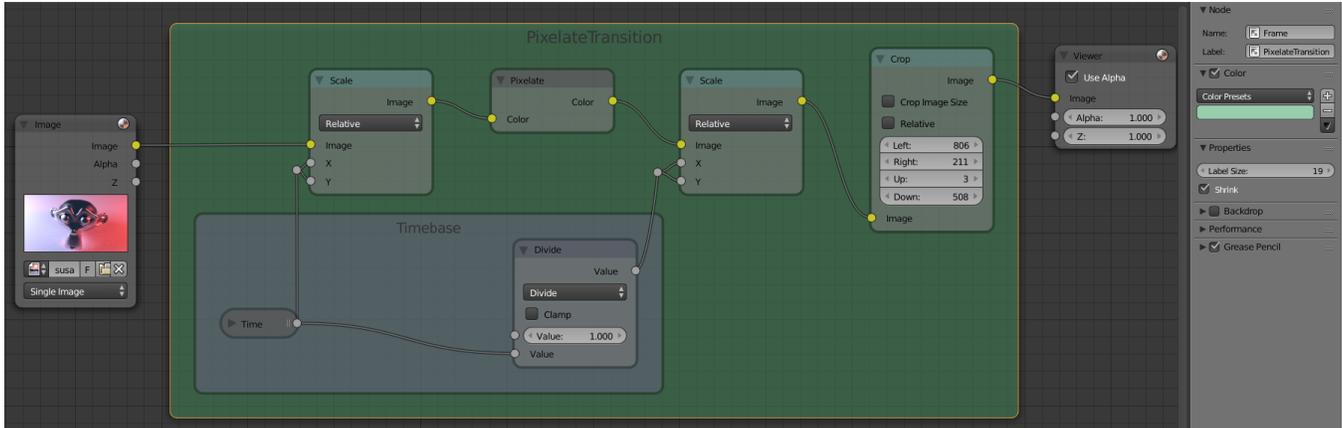
TODO - see: <https://developer.blender.org/T43469>

**Layout Nodes**

These are nodes which help you control the layout and connectivity of nodes within the Compositor.

**Frame Node**

The Frame node is a useful tool for organizing nodes by collecting related nodes together in a common area. Frames are useful when a node setup becomes large and confusing yet the re-usability of a Node Group is not required.



**Adding and Removing Nodes** Once a Frame node is placed in the editor, nodes can be added by simply dropping them onto the frame or by selecting the node(s) then the frame and using `Ctrl-P`.

To remove them select the node(s) and use the `Alt-P` shortcut. This uses the same default keyboard bindings as Parenting and can be thought of as a similar concept.

**Resizing Frame** When the Frame node is first placed in the node editor workspace it may be resized by dragging one of the edges.

Once a node is placed in the Frame, the Frame shrinks around it so as to remove wasted space. At this point it is no longer possible to grab the edge of the Frame to resize it, instead resizing occurs automatically when nodes within the Frame are rearranged.

This behavior can be changed by disabling the *Shrink* option in the Properties tab of the Properties region (N).

**Label and Color** Frame Nodes can be given a title by modifying the Label field in the properties panel. Label size can be changed as well so that, for example, subordinate Frames have smaller titles.

Frame Node colors can be applied from the properties panel which can be used to provide a powerful visual cue.

Once a satisfactory colour is found it may be saved as a preset for re-use in other Frame nodes. To do this press the `+` button next to the Color Presets drop down in the properties panel and add a name for the preset. To delete a preset first choose that preset for the active Frame and press the `-` button in the properties panel.

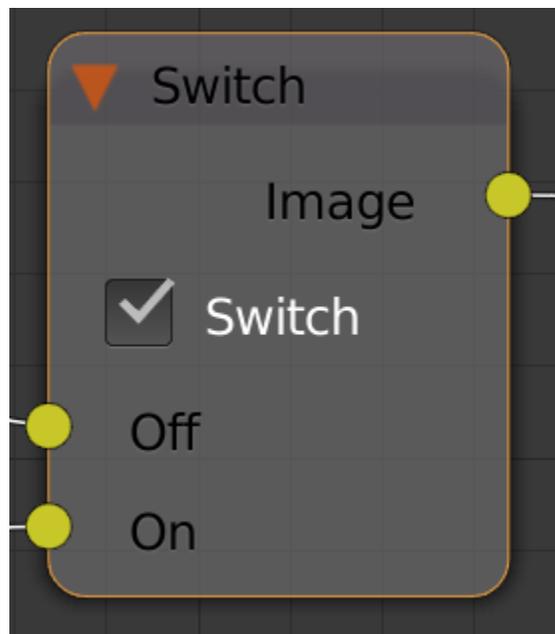
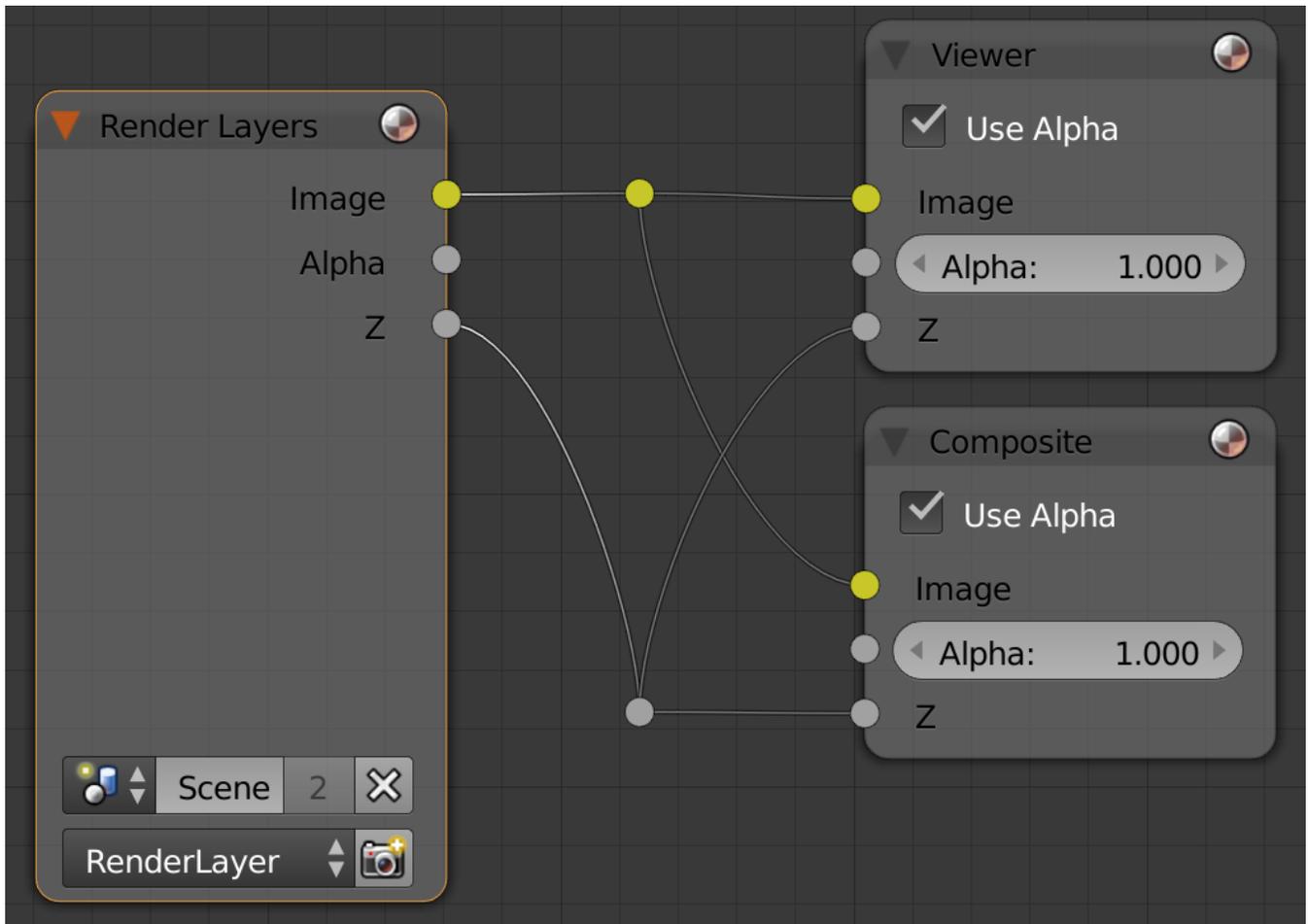
## Reroute Node

A node used primarily for organization. Reroute looks and behaves much like a socket on other nodes in that it supports one input connection while allowing multiple output connections.

To quickly add a Reroute node into an existing connection, hold `Shift` and `LMB` while sweeping across the link to ‘cut’ in a new node.

## Switch Node

Switch between two images using a checkbox. When the checkbox is checked, the ‘On’ input is output. When it is unchecked the ‘Off’ input is output instead. Switch state may be animated by adding a [keyframe](#) This makes the Switch node useful for bypassing nodes which are not wanted during part of a sequence.



## 2.10 Motion Tracking

### 2.10.1 Introduction

Blender's motion tracker supports a couple of very powerful tools for 2D tracking and 3D motion tracking, including camera tracking and object tracking, as well as some special features like the plane track for compositing. Tracks can also be used to move and deform masks for rotoscoping in the Mask Editor, which is available as a special mode in the Movie Clip Editor.

It's ready to be used in production, as validated for example by the open movie "[Tears of Steel](#)" by the Blender Foundation. Since then it has been improved a lot and the tracker is now fast, accurate and versatile.

### 2.10.2 Getting started

Here's brief descriptions of motion tracking tools currently available in Blender

#### Supervised 2D tracking

There's no common algorithm which can be used for all kinds of footage, feature points and their motions. Such algorithms can be constructed, but they'll be really slow and they can still fail, so the only way to perform 2D tracking is to manually choose the tracking algorithm and its settings. Current defaults work nicely for general footage which isn't very blurry and where feature points aren't getting highly deformed by perspective.

If you aren't sure about algorithms and settings and don't want to read this document, you can just play with settings and find one which works for you.

#### Manual lens calibration using grease pencil and/or grid

All cameras record distorted video. Nothing can be done about this because of the manner in which optical lenses work. For accurate camera motion, the exact value of the focal length and the "strength" of distortion are needed.

Currently, focal length can be automatically obtained only from the camera's settings or from the EXIF information. There are some tools which can help to find approximate values to compensate for distortion. There are also fully manual tools where you can use a grid which is getting affected by distortion model and deformed cells defines straight lines in the footage.

You can also use the grease pencil for this - just draw a line which should be straight on the footage using poly line brush and adjust the distortion values to make the grease pencil match lines on the footage.

To calibrate your camera more accurately, use the grid calibration tool from OpenCV. OpenCV is using the same distortion model, so it shouldn't be a problem.

#### Camera and object motion solving

Blender not only supports the solving of camera motion, including tripod shots, but also the solving of object motion in relation to the motion of the camera. In addition to that there is the Plane Track, which solves the motion of all markers on one plane.

There are also plans to add more tools in the future, for example more automatic tracking and solving, multi-camera solving and constrained solutions.

## Basic tools for scene orientation and stabilization

After solve, you need to orient the real scene in the 3D scene for more convenient compositing. There are tools to define the floor, the scene origin, and the X/Y axes to perform scene orientation.

If something is needed to stabilize video from the camera to make the final result looks nicer, 2D stabilization is available to help. It stabilizes video from the camera, which can compensate for camera jumps and tilt.

## Basic nodes for compositing scene into real footage

Some new nodes were added to the Compositor to composite scene into footage in easier way. So there are nodes for 2D stabilization, distortion and undistortion which are easy to use.

## Not implemented tools

Some tools aren't available in Blender yet, but they are in our TODO list. So there's currently no support for such things as rolling shutter filtering or motion capturing. But you can try to hack this stuff using currently implemented things.

## 2.10.3 Manual

### Movie Clip Editor

Almost all motion tracking tools are concentrated in the Movie Clip Editor. Currently it doesn't have any tools which aren't related to motion tracking, but in the future it may be expanded to be used for masking, so that's why it's given this more abstract name, rather than motion tracking.

This editor can be found in the list of editor types.

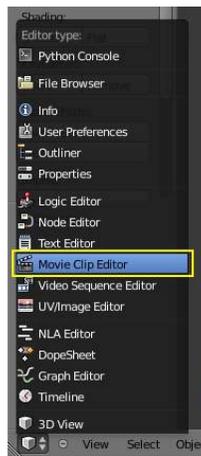


Fig. 2.2552: Editor type menu

When you switch to Movie Clip Editor window, the interface changes in the following way.

The next logical step to open a new video clip to start working with. There are several ways to do this:

- Use *Open* button from movie editor header
- Use *Clip* → *Open* menu
- Use `Alt-O` shortcut

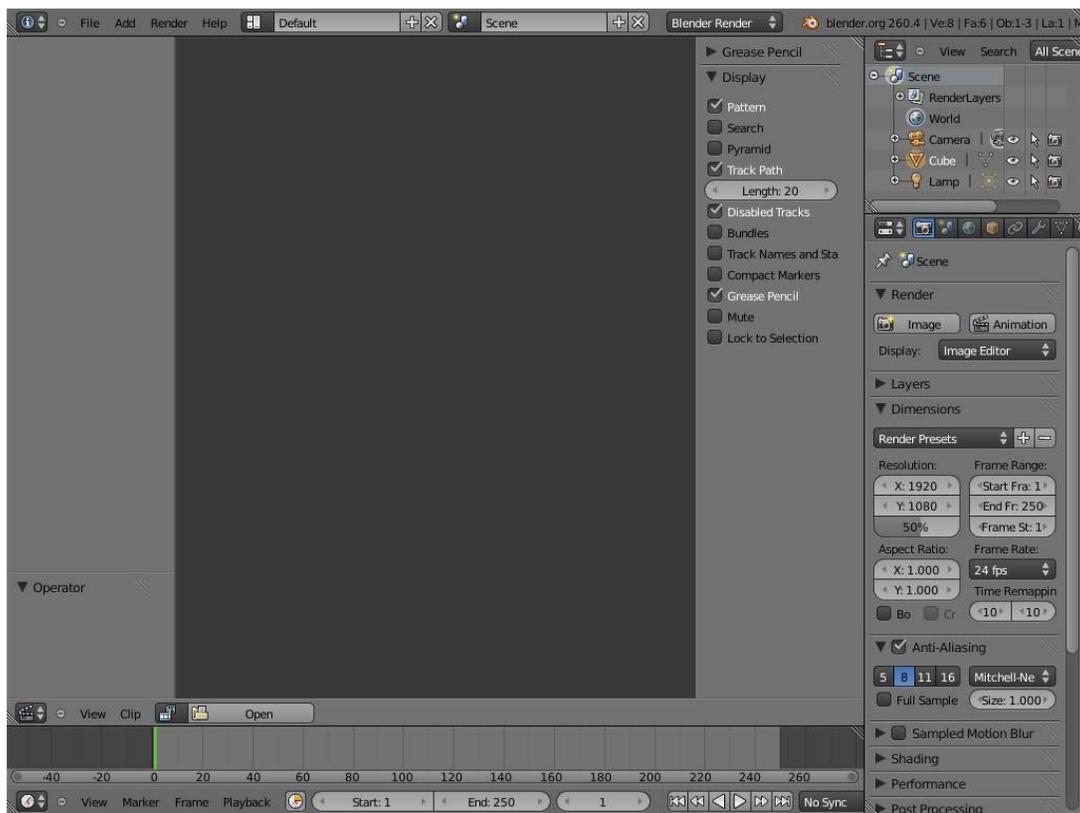


Fig. 2.2553: Movie Clip Editor interface

Both movie files and image sequences can be used in the clip editor. If you're using an image sequence there's one limitation on naming of files: the numbers at the end of the image name should be increasing continuously.

So, when a movie clip is loaded into the clip editor, extra panels are displayed in the interface.

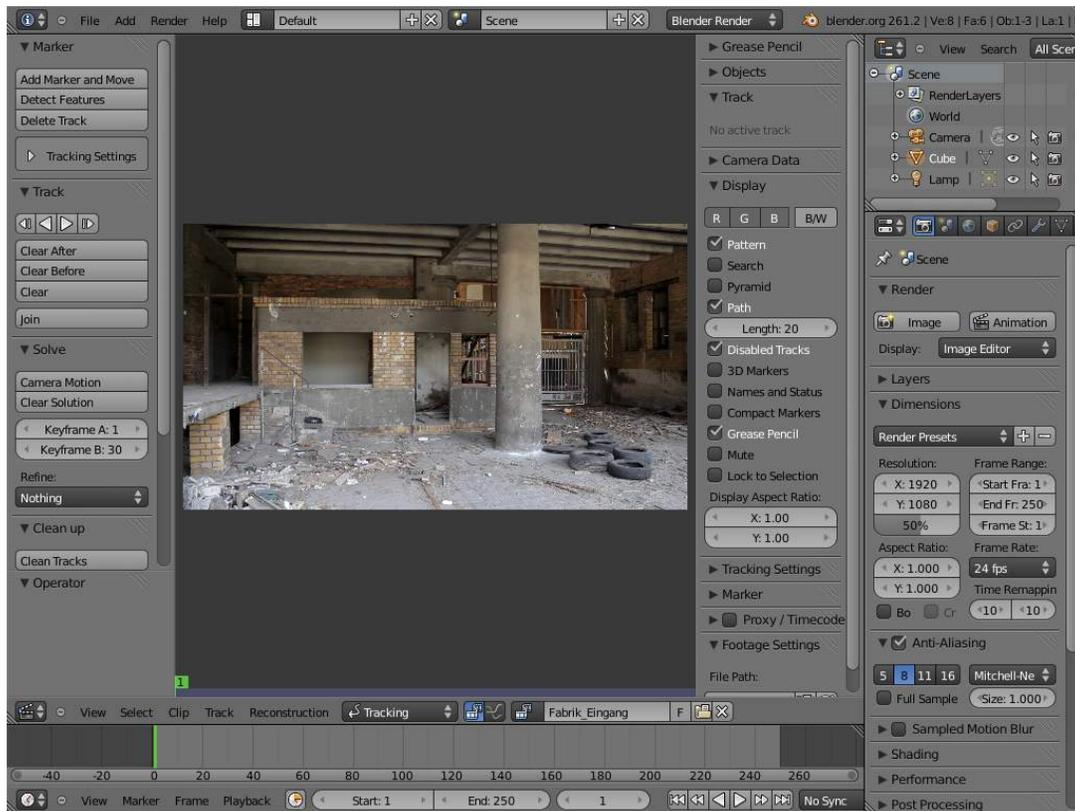


Fig. 2.2554: Movie Clip Editor with opened clip

There are plenty of new tools on the screen and here's short description of all of them.

First of all, it should be mentioned that the camera solver consists of three quite separate steps:

- 2D tracking of footage
- Camera intrinsics (focal length, distortion coefficients) specification/estimation/calibration
- Solving camera, scene orientation, scene reconstruction

Tools in the clip editor are split depending on which step they're used in, so the interface isn't cluttered up with scene orientation tools when only 2D tracking can be done. The currently displayed tool category can be changed using the Mode menu which is in the editor header.

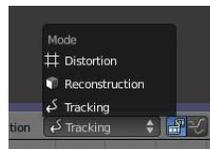


Fig. 2.2555: Movie Clip Editor mode menu

But almost all operators can be called from menus, so it's not necessary to change the mode every time you want to use a tool

which is associated with a different editor mode.

In tracking mode only tools which are related to tracking and camera solving are displayed. Camera solving tools are included here because it's after solving you'll most probably want to re-track existing tracks or place new tracks to make solving more accurate.

## Tools available in tracking mode

### Marker panel

- The **Add Marker and Move** operator places a new marker at the position of the mouse (which is under the button in this case, not ideal but it's just how things work) and then it can be moved to the needed location. When it's moved to the desired position, `FIXME(Template Unsupported: Shortcut/Mouse; {{Shortcut/MouseLmb}})` can be used to finish placing the new marker. Also, `Return` and `Spacebar` can be used to finish placing the marker. But it's faster to use `Ctrl-LMB` to place markers directly on the footage. This shortcut will place the marker in the place you've clicked. One more feature here: until you've released the mouse button, you can adjust the marker position by moving the mouse and using the track preview widget to control how accurately the marker is placed.
- The **Detect Features** operator detects all possible features on the current frame and places markers at these features. This operator doesn't take into account other frames, so it can place markers on features which belong to moving objects, and if camera is turning away from this shot, no markers would be placed on frames after the camera moved away.

There are several properties for this operator:

**Placement** is used to control where to place markers. By default, they'll be added through the whole frame, but you can also outline some areas with interesting features with grease pencil and place markers only inside the outlined area. That's how the "Inside Grease Pencil" placement variant works. You can also outline areas of no interest (like trees, humans and so) and place markers outside of these areas. That's how the "Outside Grease Pencil" placement variant works.

**Margin** controls the distance from the image boundary for created markers. If markers are placed too close to the image boundary, they'll fail to track really quickly and they should be deleted manually. To reduce the amount of manual clean-up, this parameter can be used.

**Trackability** limits minimal trackability for placing markers. This value comes from the feature detection algorithm and basically it means: low values means most probably this feature would fail to track very soon, high value means it's not much such track. Amount of markers to be added can be controlled with this value.

**Distance** defines the minimal distance between placed markers. It's needed to prevent markers from being placed too close to each other (such placement can confuse the camera solver).

- **Delete Track** is a quite self-explaining operator which deletes all selected tracks.

### Track panel

- The first row of buttons is used to perform tracking of selected tracks (i.e. following the selected feature from frame to frame). Tracking can happen (in order of buttons):
  - Backward one frame
  - Backward along the sequence
  - Forward along the whole sequence
  - Forward one frame

This operator depends on settings from the Tracking Settings panel, which will be described later. If during sequence tracking the algorithm fails to track some markers, they'll be disabled and tracking will continue for the rest of the markers. If the algorithm fails when tracking frame-by-frame, the marker is not disabled, and the most likely position of the feature on the next frame is used.

**Clear After** deletes all tracked and keyframed markers after the current frame for all selected tracks.

**Clear Before** deletes all tracked and keyframed markers before the current frame for all selected tracks.

**Clear** clears all markers except the current one from all selected tracks.

**Join** operator joins all selected tracks into one. Selected tracks shouldn't have common tracked or keyframed markers at the same frame.

### Solve panel

**Camera Motion** operator solves the motion of camera using all tracks placed on the footage and two keyframes specified on this panel. There are some requirements:

- There should be at least 8 common tracks on the both of the selected keyframes.
- There should be noticeable parallax effects between these two keyframes.

If everything goes smoothly during the solve, the average reprojection error is reported to the information space and to the clip editor header. Reprojection error means the average distance between reconstructed 3D position of tracks projected back to footage and original position of tracks. Basically, reprojection error below 0.3 means accurate reprojection, 0.3-3.0 means quite nice solving which still can be used. Values above 3 means some tracks should be tracked more accurately, or that values for focal length or distortion coefficients were set incorrectly.

The **Refine** option specifies which parameters should be refined during solve. Such refining is useful when you aren't sure about some camera intrinsics, and solver should try to find the best parameter for those intrinsics. But you still have to know approximate initial values - it'll fail to find correct values if they were set completely incorrectly initially.

### Cleanup Panel

This panel contains a single operator and its settings. This operator cleans up bad tracks: tracks which aren't tracked long enough or which failed to reconstruct accurately. Threshold values can be specified from sliders below the button. Also, several actions can be performed for bad tracks:

- They can simply be selected
- Bad segments of tracked sequence can be removed
- The whole track can be deleted

### Clip Panel

This panel currently contains the single operator *Set as background* which sets the clip currently being edited as the camera background for all visible 3D viewports. If there's no visible 3D viewports or the clip editor is open in full screen, nothing will happen.

## Properties available in tracking mode

### Grease Pencil Panel

It's a standard grease pencil panel where new grease pencil layers and frames can be controlled. There's one difference in the behavior of the grease pencil from other areas - when a new layer is created "on-demand" (when making a stroke without adding a layer before this) the default color for the layer is set to pink. This makes the stroke easy to notice on all kinds of movies.

### Objects Panel



Fig. 2.2556: Objects Panel in clip editor

This panel contains a list of all objects which can be used for tracking, camera or object solving. By default there's only one object in this list which is used for camera solving. It can't be deleted and other objects can't be used for camera solving; all added objects are used for object tracking and solving only. These objects can be referenced from Follow Track and Object Solver constraints. Follow Track uses the camera object by default.

New objects can be added using `Plus` and the active object can be deleted with the `Minus` button. Text entry at the bottom of this panel is used to rename the active object.

If some tracks were added and tracked to the wrong object, they can be copied to another object using `Track → Copy Tracks` and `Track → Paste Tracks`.

The usage for all kind of objects (used for camera and object tracking) is the same: track features, set camera data, solve motion. Camera data is sharing between all objects and refining of camera intrinsics happens when solving camera motion only.

### Track Panel



Fig. 2.2557: Track Panel in clip editor

First of all, track name can be changed in this panel. Track names are used for linking tracking data to other areas, like a Follow Track constraint.

The next thing which can be controlled here is the marker's enabled flag (using the button with the eye icon). If a marker is disabled, its position isn't used either by solver nor by constraints.

The button with the lock icon to the right of the button with the eye controls whether the track is locked. Locked tracks can't be edited at all. This helps to prevent accidental changes to tracks which are "finished" (tracked accurately along the whole footage).

The next widget in this panel is called "Track Preview" and it displays the content of the pattern area. This helps to check how accurately the feature is being tracked (controlling that there's no sliding off original position) and also helps to move the track back to the correct position. The track can be moved directly using this widget by mouse dragging.

If an anchor is used (the position in the image which is tracking is different from the position which is used for parenting), a preview widget will display the area around the anchor position. This configuration helps in masking some things when there's no good feature at position where the mask corner should be placed. Details of this technique will be written later.

There's small area below the preview widget which can be used to enlarge the vertical size of preview widget (the area is highlighted with two horizontal lines).

The next setting is channels control. Tracking happens in gray-scale space, so a high contrast between the feature and its background yields more accurate tracking. In such cases disabling some color channels can help.

The last thing is custom color, and the preset for it. This setting overrides the default marker color used in the clip editor and 3D viewport, and it helps to distinguish different type of features (for example, features in the background vs. foreground and so on). Color also can be used for "grouping" tracks so a whole group of tracks can be selected by color using the Select Grouped operator.

---

**Tip:** To select good points for tracking, use points in the middle of the footage timeline and track backwards and forwards from there. This will provide a greater chance of the marker and point staying in the camera shot.

---

## Camera Data Panel

This panel contains all settings of the camera used for filming the movie which is currently being edited in the clip editor.

First of all, predefined settings can be used here. New presets can be added or unused presets can be deleted. But such settings as distortion coefficients and principal point aren't included into presets and should be filled in even if camera presets are used.

**Focal Length** is self-explanatory; it's the focal length with which the movie was shot. It can be set in millimeters or pixels. In most cases focal length is given in millimeters, but sometimes (for example in some tutorials on the Internet) it's given in pixels. In such cases it's possible to set it directly in the known unit.

**Sensor Width** is the width of the CCD sensor in the camera. This value can be found in camera specifications.

**Pixel Aspect Ratio** is the pixel aspect of the CCD sensor. This value can be found in camera specifications, but can also be guessed. For example, you know that the footage should be 1920x1080, but the images themselves are 1280x1080. In this case, the pixel aspect is:  $1920 / 1280 = 1.5$

**Optical Center** is the optical center of the lens used in the camera. In most cases it's equal to the image center, but it can be different in some special cases. Check camera/lens specifications in such cases. To set the optical center to the center of image, there's a `Return` button below the sliders.

**Undistortion K1, K2 and K3** are coefficients used to compensate for lens distortion when the movie was shot. Currently these values can be tweaked by hand only (there are no calibration tools yet) using tools available in Distortion mode. Basically, just tweak K1 until solving is most accurate for the known focal length (but also take grid and grease pencil into account to prevent "impossible" distortion).

## Display Panel

This panel contains all settings which control things displayed in the clip editor.

**R, G, B** and **B/W** buttons at the top of this panel are used to control color channels used for frame preview and to make the whole frame gray scale. It's needed because the tracking algorithm works with gray-scale images and it's not always obvious to see which channels disabled will increase contrast of feature points and reduce noise.

**Pattern** can be used to disable displaying of rectangles which correspond to pattern areas of tracks. In some cases it helps to make the clip view cleaner to check how good tracking is.

**Search** can be used to disable displaying of rectangles which correspond to search areas of tracks. In some cases it helps to make the clip view cleaner to check how good tracking is. Only search areas for selected tracks will be displayed.

**Pyramid** makes the highest pyramid level be visible. Pyramids are defined later in the Tracking Settings panel section, but basically it helps to determine how much a track is allowed to move from one frame to another.

**Track Path** and **Length** control displaying of the paths of tracks. The ways tracks are moving can be visible looking at only one frame. It helps to determine if a track jumps from its position or not.

**Disabled Tracks** makes it possible to hide all tracks which are disabled on the current frame. This helps to make view more clear, to see if tracking is happening accurately enough.

**Bundles** makes sense after solving the movie clip, and it works in the following way: the solved position of each track gets projected back to the movie clip and displayed as a small point. The color of the point depends on the distance between the projected coordinate and the original coordinate: if they are close enough, the point is green, otherwise it'll be red. This helps to find tracks which weren't solved nicely and need to be tweaked.

**Track Names and Status** displays information such as track name and status of the track (if it's keyframed, disabled, tracked or estimated). Names and status for selected tracks are displayed.

**Compact Markers** The way in which markers are displayed (black outline and yellow foreground color) makes tracks visible on all kind of footage (both dark and light). But sometimes it can be annoying and this option will make the marker display more compactly - the outline is replaced by dashed black lines drawn on top of the foreground, so that marker areas are only 1px thick.

**Grease pencil** controls if grease pencil strokes are allowed to be displayed and made.

**Mute** changes displaying on movie frame itself with black square, It helps to find tracks which are tracked inaccurately or which weren't tracked at all.

**Grid** (available in distortion mode only) displays a grid which is originally orthographic, but is affected by the distortion model. This grid can be used for manual calibration - distorted lines of grids are equal to straight lines in the footage.

**Manual Calibration** (available in distortion mode only) applies the distortion model for grease pencil strokes. This option also helps to perform manual calibration. A more detailed description of this process will be added later.

**Stable** (available in reconstruction mode only). This option makes the displayed frame be affected by the 2D stabilization settings. It's only a preview option, which doesn't actually change the footage itself.

**Lock to Selection** makes the editor display selected tracks at the same screen position along the whole footage during playback or tracking. This option helps to control the tracking process and stop it when the track is starting to slide off or when it jumped.

**Display Aspect Ratio** changes the aspect ratio for displaying only. It does not affect the tracking or solving process.

## Tracking Settings Panel

**Common options** This panel contains all settings for the 2D tracking algorithms. Depending on which algorithm is used, different settings are displayed, but there are a few that are common for all tracker settings:

**Adjust Frames** controls which patterns get tracked; to be more precise, the pattern from which frame is getting tracked. Here's an example which should make things clearer.

The tracker algorithm receives two images inside the search area and the position of a point to be tracked in the first image. The tracker tries to find the position of that point from the first image in the second image.

Now, this is how tracking of the sequence happens. The second image is always from a frame at which the position of marker isn't known (next tracking frame). But a different first image (instead of the one that immediately precedes the second image in the footage) can be sent to the tracker. Most commonly used combinations:

- An image created from a frame on which the track was keyframed. This configuration prevents sliding from the original position (because the position which best corresponds to the original pattern is returned by the tracker), but it can lead to small jumps and can lead to failures when the feature point is deformed due to camera motion (perspective transformation, for example). Such a configuration is used if **Adjust Frames** is set to 0.
- An image created from the current frame is sent as first image to the tracker. In this configuration the pattern is tracking between two neighboring frames. It allows dealing with cases of large transformations of the feature point but can lead to sliding from the original position, so it should be controlled. Such a configuration is used if **Adjust Frames** is set to 1.

If **Adjust Frames** is greater than 1, the behavior of tracker is: keyframes for tracks are creating every **Adjust Frames** frames, and tracking between keyframed image and next image is used.

**Speed** can be used to control the speed of sequence tracking. This option doesn't affect the quality of tracking; it just helps to control if tracking happens accurately. In most cases tracking happens much faster than real time, and it's difficult to notice when a track began to slide out of position. In such cases **Speed** can be set to Double or Half to add some delay between tracking two frames, so slide-off would be noticed earlier and the tracking process can be cancelled to adjust positions of tracks.

**Frames Limit** controls how many frames can be tracked when the Track Sequence operator is called. So, each Track Sequence operation would track maximum **Frames Limit** frames. This also helps to notice slide-off of tracks and correct them.

**Margin** can be used disable tracks when they become too close to the image boundary. This slider sets "too close" in pixels.

**KLT tracker options** The KLT tracker is the algorithm used by default. It allows tracking most kinds of feature points and their motion. It uses pyramid tracking which works in the following way. The algorithm tracks an image larger than the defined pattern first to find the general direction of motion. Then it tracks a slightly smaller image to refine the position from the first step and make the final position more accurate. This iterates several times. The number of steps of such tracking is equal to the **Pyramid Level** option and we tell that on first step tracking happens for highest pyramid level. So Pyramid Level=1 is equal to pattern itself, and each next level doubles tracking image by 2.

The search area should be larger than the highest pyramid level and the "free space" between the search area and highest pyramid level defines how much the feature can move from one frame to another and still be tracked.

Default settings should work in most general cases, but sometimes the pyramid level should be changed. For example, when footage is blurry, adding extra pyramid levels helps to track them.

This algorithm can fail in situations where a feature point is moving in one direction and the texture around that feature point is moving in another direction.

**SAD tracker options** On each step, the SAD tracker reviews the whole search area and finds the pattern on the second image which is most like the pattern which is getting tracking. This works pretty quickly, but can fail in several cases. For example, when there's another feature point which looks like the tracking feature point in the search area. In this case, SAD will tend to jump off track from one feature to another.

**Correlation** defines the threshold value for correlation between two patterns which is still considered successful tracking. 0 means there's no correlation at all, 1 means correlation is full.

There's one limitation: currently: it works for features of size 16x16 pixels only.

### Marker Panel

This panel contains numerical settings for marker position, pattern and search area dimensions, and offset of anchor point from pattern center. All sliders are self-explanatory.

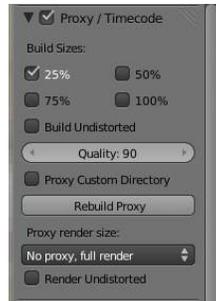


Fig. 2.2558: Proxy / Timecode Panel in clip editor

### Proxy / Timecode Panel

This panel contains options used for image proxies and timecodes for movies.

Proxy allows displaying images with lower resolution in the clip editor. This can be helpful in cases when tracking of 4K footage is happening on a machine with a small amount of RAM.

The first four options are used to define which resolutions of proxy images should be built. Currently it's possible to build images 25%, 50%, 75% and 100% of the original image size. Proxy size of 100% can be used for movies which contain broken frames which can't be decoded.

**Build Undistorted** means that the proxy builder also creates images from undistorted original images for the sizes set above. This helps provide faster playback of undistorted footage.

Generated proxy images are encoding using JPEG, and the quality of the JPEG codec is controlled with the **Quality** slider.

By default, all generated proxy images are storing to the <path of original footage>/BL\_proxy/<clip name> folder, but this location can be set by hand using the **Proxy Custom Directory** option.

*Rebuild Proxy* will regenerate proxy images for all sizes set above and regenerate all timecodes which can be used later.

**Use Timecode Index** can (and better be used) for movie files. Basically, timecode makes frame search faster and more accurate. Depending on your camera and codec, different timecodes can give better result.

**Proxy Render Size** defines which proxy image resolution is used for display. If **Render Undistorted** is set, then images created from undistorted frames are used. If there's no generated proxies, render size is set to "No proxy, full render", and render undistorted is enabled, undistortion will happen automatically on frame draw.

### Tools available in reconstruction mode

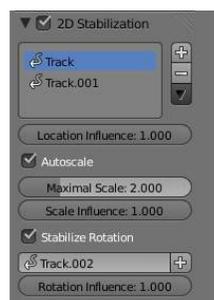


Fig. 2.2559: Proxy / 2D Stabilization Panel in clip editor

There's one extra panel which is available in reconstruction mode - 2D stabilization panel.

This panel is used to define data used for 2D stabilization of the shot. Several options are available in this panel.

First of all is the list of tracks to be used to compensate for camera jumps, or location. It works in the following way: it gets tracks from the list of tracks used for location stabilization and finds the median point of all these tracks on the first frame. On each frame, the algorithm makes this point have the same position in screen coordinates by moving the whole frame. In some cases it's not necessary to fully compensate camera jumps and **Location Influence** can be used in such cases.

The camera can also have rotated a bit, adding some tilt to the footage. There's the **Stabilize Rotation** option to compensate for this tilt. A single extra track needs to be set for this, and it works in the following way. On first frame of the movie, this track is connected with the median point of the tracks from list above and angle between horizon and this segment is kept constant through the whole footage. The amount of rotation applied to the footage can be controlled by **Rotation Influence**.

If the camera jumps a lot, there'll be noticeable black areas near image boundaries. To get rid of these black holes, there's the **Autoscale** option, which finds smallest scale factor which, when applied to the footage, would eliminate all the black holes near the image boundaries. There's an option to control the maximal scale factor, (**Maximal Scale**), and the amount of scale applied to the footage (**Scale Influence**).

## 2.11 Grease Pencil

### 2.11.1 Introduction

Use the *Grease Pencil* tool to draw freehand sketches and annotations in the 3D View, UV/Image Editor, Node Editor, or Movie Clip Editor. The sketches are saved with the blend file. Planning animation poses and motion curves, sketching out model topology, and use as director's shot review tool are just a few of the applications.



Fig. 2.2560: The Grease Pencil in action.

The next few pages explain how to use this tool:

- Drawing sketches.
- Layers and Animation.
- Converting sketches to geometry.

### 2.11.2 Drawing With Grease Pencil

- Enable the *Grease Pencil* by clicking *Draw*, *Line*, *Poly* or *Erase* from the Toolshelf (T). A new layer will be automatically added for you to draw on.
- A new layer can be added from the *Grease Pencil Properties panel*. This panel can also be used to customize the color, opacity and thickness of the pencil lines. Changes to these settings will affect all strokes on the current layer.

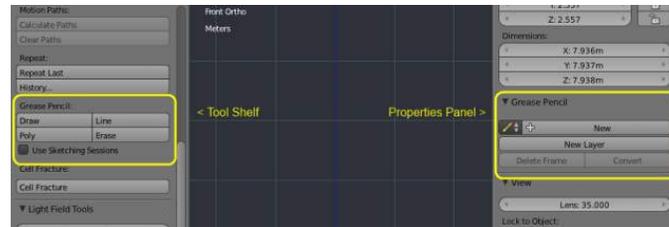


Fig. 2.2561: Grease Pencil Tool Shelf and Properties Panel.

*Grease Pencil* sketches can be converted to editable geometry and used to aid the animation process.

- [Read more about Layers and Animation](#)
- [Read more about Converting sketches to geometry](#)

## Drawing

The Toolshelf provides a number of options for drawing with the *Grease Pencil* which are detailed below. The Toolshelf can be seen in the screenshot *Grease Pencil Tool Shelf and Properties Panel* above.

### Grease Pencil Mode and Shortcut Summary

**Draw D-LMB** Draw a new stroke (multiple short, connected lines). The stroke will finish when you release the mouse button.

**Line Ctrl-D-LMB** Draw a new line in rubber band mode. The line will finish when you release the mouse button.

**Poly Ctrl-D-RMB** Draw connected lines by clicking at various points. Lines will be automatically added to connect the two points.

**Erase D-RMB** Erases segments of strokes that fall within the radius of the eraser “brush”. The erasing will continue until the mouse button is released. If begun with *Erase*, either RMB or LMB will erase strokes. The size of the eraser “brush” can be controlled with *Wheel* or (*NumpadPlus*, *NumpadMinus*) keys (while still holding RMB).

### Sketching Sessions

A Sketching Session allows for rapid sketching with the *Grease Pencil* when multiple strokes are desired. With this option set, a sketching session starts when a *Grease Pencil* stroke is made. The type of session (Draw, Line, Poly, Erase) is determined by the first stroke made which can be done via hotkeys or the Toolshelf. Use *Esc* or *Return* to exit the sketching session. Note that in a Erase Sketching Session both LMB or RMB can be used once the session has started.

### Appearance Settings

Set the color, line width and other aspects of the grease pencil’s appearance in the *Grease Pencil Panel* of the *Properties* shelf (N) shown here.

There are separate settings for each layer with those of the active layer shown in the panel. All the strokes on a layer (not just those made after a particular change) are affected by that layer’s grease pencil properties.

**Stroke** Sets the line color and opacity.

**Fill** Sets the color of the interior space enclosed by the strokes. Increase the opacity from zero to make the fill visible. Fill works best on convex shapes.

**Thickness** Width of the line strokes.

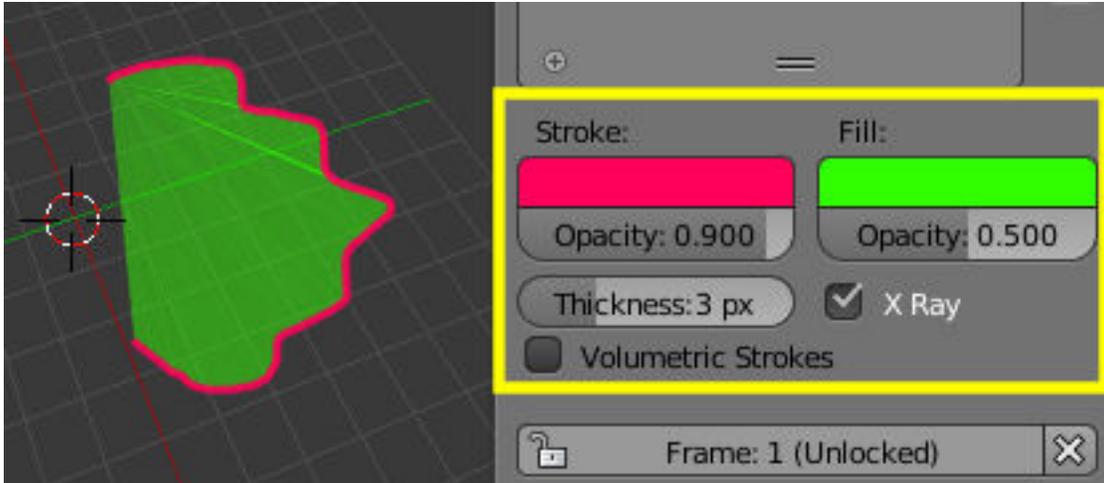


Fig. 2.2562: Grease pencil properties

**X-Ray** Makes the lines visible when they pass behind other objects in the scene.

**Volumetric Strokes** Draw strokes as a series of filled spheres, resulting in an interesting volumetric effect. Get best results with partial opacity and large stroke widths.

## Drawing Settings

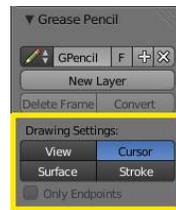


Fig. 2.2563: Grease Pencil Drawing Settings.

In the *Grease Pencil Panel* of the *Tool shelf* (T) there are several choices for *Drawing Settings*.

**View** New strokes are locked to the view.

**Cursor (3D view only)** New strokes are drawn in 3D-space, with position determined by the 3D cursor and the view rotation at the time of drawing. *Cursor* is available as an option in the *UV/Image Editor* but it functions identically to the *View* option.

**Surface (3D view only)** New strokes are drawn in 3D-space, with their position projected onto the first visible surface.

**Stroke (3D view only)** New strokes are drawn in 3D-space, with their position projected onto existing visible strokes. Note that strokes created with *View* are not in 3D-space and are not considered for this projection.

Enabling the *Only Endpoints* setting applies the drawing setting only to the endpoints of the stroke. The part of the stroke between the endpoints is adjusted to lie on a plane passing through the endpoints.

## Editing

These tools let you move and reshape grease pencil strokes after they have been drawn.



Fig. 2.2564: The effect of different Drawing Settings on Grease Pencil strokes.

Open the Grease Pencil tab on the Toolshelf (T). Look for the tools in the Edit Strokes panel shown here:

The basic steps are:

- enter the grease pencil edit mode
- select some strokes
- move and reshape them

### Edit Mode

**Enable Editing (D-Tab)** Enters or exits the edit mode.

While in the grease pencil editing mode, Blender redirects the common editing keys to operate on the grease pencil layer instead of the 3D scene components.

### Select

Grease pencil strokes are formed from a series of connected vertex points. To make changes, first select points on the strokes that you want to edit. You can only select points on the active layer. The selected points are highlighted as in the image above.

---

**Hint:** Set the layer's *Stroke Thickness* to 1 to make the points more visible.

---

Use the mouse to select the points, or one of the selection buttons in the panel as detailed in [Basic Selection](#).

Various selection functions similar to those available when editing meshes can be used:

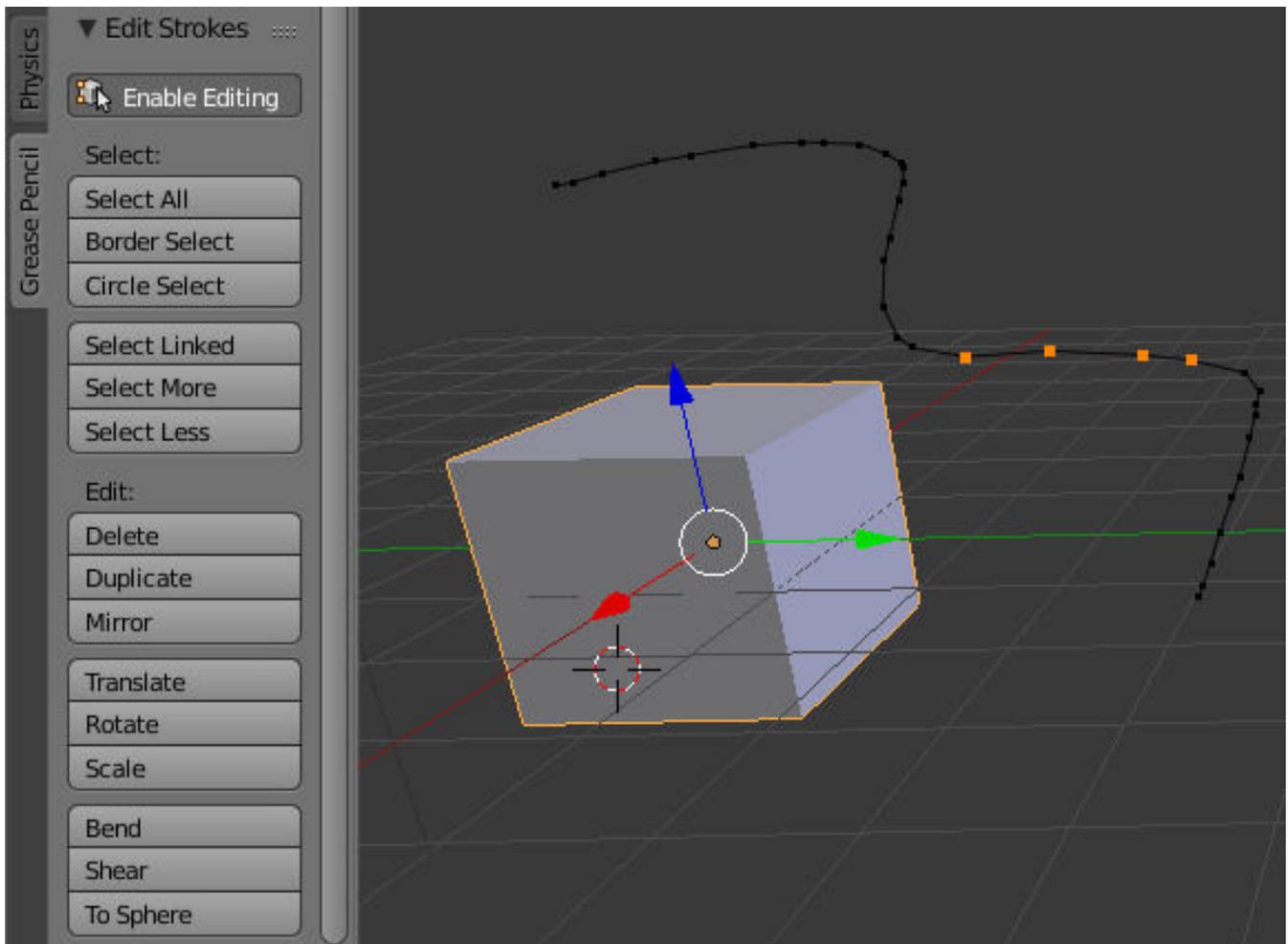


Fig. 2.2565: Edit panel with grease pencil strokes.

Select All	A
Border Select	B
Circle Select	C
Select Linked	Ctrl-L
Select More	Ctrl-NumpadPlus
Select Less	Ctrl-NumpadMinus
Select Stroke	Alt-LMB

## Edit

**Delete (X)** Choose from:

- Points - delete the selected points, leaving a gap in the stroke
- Dissolve - reconnect the ends so there is no gap in the stroke
- Strokes - delete the entire stroke containing any selected points
- Frame - delete a frame when doing [Animation of the Sketches](#).

**Duplicate (Shift-D)** Make a copy of the selected points at the same location. Use the mouse to *Translate* them into position. LMB places them at their new position. RMB cancels and removes the duplicates.

**Translate (G) Rotate (R) Scale (S)** Move the selected points with the mouse. LMB places them at their new position. Refine these operations with *Pivot Center*, *View* or *Global* transform orientations, snap to *Increment* and *Proportional Editing* detailed in the general [Transformations Instructions](#).

**Mirror (Ctrl-M) Bend (Shift-W) Shear (Shift-Ctrl-Alt-S) To Sphere (Shift-Alt-S)** These are similar to the equivalent mesh operations detailed in [Deforming Instructions](#).

## Sensitivity When Drawing

The default settings for the sensitivity of mouse/stylus movement when drawing have been set to reduce jitter while still allowing fine movement. However, if these are not appropriate they can be altered in *User Preferences window* → *Editing* → *Grease Pencil*.

**Manhattan Distance** The minimum number of pixels the mouse should have moved either horizontally or vertically before the movement is recorded. Decreasing this should work better for curvy lines.

**Euclidean Distance** The minimum distance that the mouse should have traveled before movement is recorded.

**Eraser Radius** The size of the eraser “brush”.

**Smooth Stroke** This turns on the post-processing step of smoothing the stroke to remove jitter. It is only relevant when not drawing straight lines. By default this is enabled. It should be noted that it can often cause “shrinking” of drawings, and may be turned off if the results are not desirable.

**Simplify Stroke** This turns on the post-processing step of simplifying the stroke to remove about half of current points in it. It is only relevant when not drawing straight lines. By default this is disabled. As with *Smooth Stroke*, it can often cause “shrinking” of drawings, and loss of precision, accuracy and smoothness.

## Additional Notes For Tablet Users

- The thickness of a stroke at a particular point is affected by the pressure used when drawing that part of the stroke.
- The “eraser” end of the stylus can be used to erase strokes.

### 2.11.3 Layers

Grease Pencil sketches are organized in layers, much like the image layers in the GIMP or Photoshop. These layers are not related to any of the other layer systems in Blender.

The layers' main purpose is to gather sketches that are related in some meaningful way (i.e. “blocking notes”, “director’s comments on blocking”, or “guidelines”). For this reason, all the strokes on a layer (not just those made after a particular change) are affected by that layer’s color, opacity, and stroke thickness settings.

Layers are managed in the *Grease Pencil Panel* of the *Properties* region (N) shown here.

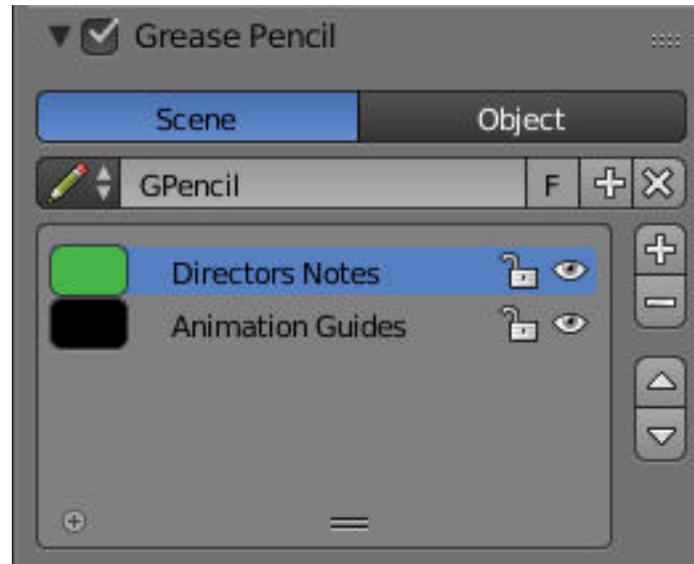


Fig. 2.2566: Grease Pencil Panel

Use the adjacent controls to Add, Remove or adjust the position of a layer in the list. Each layer has a visibility icon, and a lock icon to protect it from further changes. Double click on a layer name to rename it.

There is a list of layers attached to each scene and a list of layers associated with each object. The buttons above the list box control its contents, showing either the layers associated with the active scene or the list of layers associated with the active object.

By default, most operations occur only on the *active* layer highlighted in the list.

#### Animation of the Sketches

Use the Grease Pencil to do basic pencil tests (i.e. 2D animation in flipbook style). Sketches are stored on the frame that they were drawn on, as a separate drawing (only on the layer that they exist on). Each drawing is visible until the next drawing for that layer is encountered. The only exception to this is the first drawing for a layer, which will also be visible before the frame it was drawn on.

Therefore, it is simple to make a pencil-test/series of animated sketches:

- Go to first relevant frame. Draw.
- Jump to next relevant frame. Draw some more.
- Keep repeating process, and drawing until satisfied. Voila! Animated sketches.

## Onion Skinning

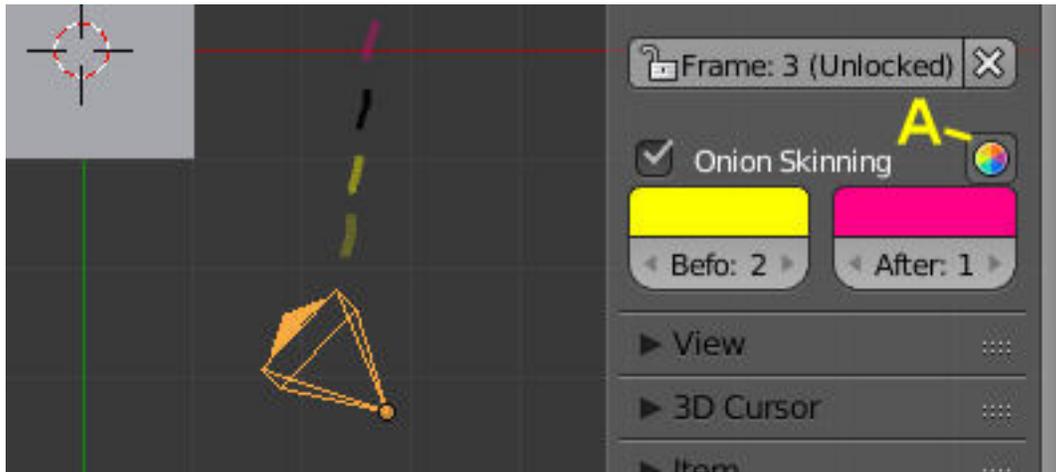


Fig. 2.2567: Grease Pencil Onion Skinning

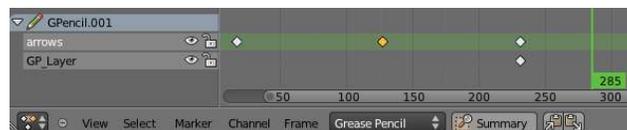
Onion-skinning, also known as ghosting, helps an animator by displaying the neighboring frames as a faded trail. Enable the option with the *Onion Skin* button in the grease pencil properties panel (shown above).

Use *Before* and *After* to set the number of ghost frames drawn on either side of the current frame. When *Use Custom Colors* (Marked **A**) is enabled, you can also use the *Before* and *After* controls to change the color of the ghosted frames.

## Adjusting Timing of Sketches

It is possible to set a Grease-Pencil block to be loaded up in the *DopeSheet* for editing of the timings of the drawings. This is especially useful for animators blocking out shots, where the ability to re-time blocking is one of the main purposes of the whole exercise.

- In an *Dope Sheet* window, change the mode selector (found beside the menus) to *Grease Pencil* (by default, it should be set to *DopeSheet*).
- At this point, the *DopeSheet* should now display a few “channels” with some “keyframes” on them. These “channels” are the layers, and the “keyframes” are the frames at which the layer has a sketch defined. They can be manipulated like any other data in the *DopeSheet* can be.



All the available Grease-Pencil blocks for the current screen layout will be shown. The Area/Grease-Pencil datablocks are drawn as green channels, and are named with relevant info from the views. They are also labeled with the area (i.e. window) index (which is currently not shown anywhere else though).

## Copying Sketches

It is possible to copy sketches from a layer/layers to other layers in the *Action Editor*, using the “Copy”/“Paste” buttons in the header. This works in a similar way as the copy/paste tools for keyframes in the *Action Editor*.

Sketches can also be copied from one screen (or view) to another using these tools. It is important to keep in mind that keyframes will only be pasted into selected layers, so layers will need to be created for the destination areas too.

### 2.11.4 Converting Sketches to Objects

In the 3D view, sketches on the active layer can be converted to geometry, based on the current view settings, by transforming the points recorded when drawing (which make up the strokes) into 3D-space. Currently, all points will be used, so it may be necessary to simplify or subdivide parts of the created geometry for standard use.

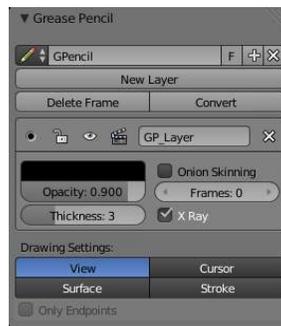


Fig. 2.2568: Grease Pencil panel in 3D View.

Sketches can currently be converted into curves, as proposed by the *Convert Grease Pencil* menu popped-up by the *Convert* button in the grease pencil properties

**Path** Create NURBS 3D curves of order 2 (i.e. behaving like polylines).

**Bezier Curve** Create Bezier curves, with free “aligned” handles (i.e. also behaving like polylines).

---

**Note:** Why “polyline-like” curves?

To get by default curves following exactly the grease pencil strokes. If you want a smoothed curve, just edit it to get auto handles (for Bezier), or raise its order (for NURBS).

---

**Note:** Converting to Mesh

If you want to convert your sketch to a mesh, simply choose first *NURBS*, and then convert the created curve to a mesh.

---

#### General Options

Stroke’s width will be used to set the curve’s control points’ radii and weights (**not** NURBS weights, but those used e.g. as goal by the softbody simulation...). The default behavior is to get strokes’ width (as defined in its settings - and which might have been modulated by the pen pressure), to multiply it by a given constant (0.1), and to assign it directly to weights. Radii get the same value scaled by the *Radius Fac* factor (e.g. with a **10.0** factor, a stroke width of **3** will give radii of **3.0** ...).

*Normalize Weight* (enabled by default) will scale weights value so that they tightly fit into the  $[0.0, 1.0]$  range.

All this means that with a pressure tablet, you can directly control the radius and weight of the created curve, which can affect e.g. the width of an extrusion, or the size of an object through a *Follow Path* constraint or *Curve* modifier!

*Link Strokes* (enabled by default) will create a single spline (i.e. curve element) from all strokes in active grease pencil layer. This especially useful if you want to use the curve as a path. All the strokes are linked in the curve by “zero weights/radii” sections.

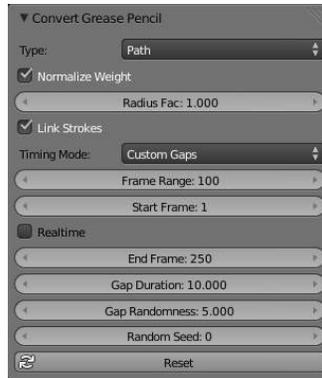


Fig. 2.2569: The Convert to Curve options.

## Timing

Grease pencil now stores “dynamic” data, i.e. how fast they were drawn. When converting to curve, those data can be used to create an *Evaluate Time* F-Curve (in other words, a path animation), that can be used e.g. to control another object’s position along that curve (*Follow Path* constraint, or, through a driver, *Curve* modifier). So this allows you to reproduce your drawing movements.

**Warning:** All those “timing” options need *Link Stroke* to be enabled - else they would not make much sense!

**Warning:** Please note that if you use this tool with older grease pencil’s strokes (i.e. some without any timing data), you will only have a subset of those options available (namely, only linear progression along the curve over a specified range of frames).

**Timing Mode** This control let you choose how timing data are used.

**No Timing** Just create the curve, without any animation data (hence all following options will be hidden)...

**Linear** The path animation will be a linear one.

**Original** The path animation will reflect to original timing, including for the “gaps” (i.e. time between strokes drawing).

**Custom Gaps** The path animation will reflect to original timing, but the “gaps” will get custom values. This is especially useful if you have very large pauses between some of your strokes, and would rather like to have “reasonable” ones!

**Frame Range** The “length” of the created path animation, in frames. In other words, the highest value of *Evaluation Time*.

**Start Frame** The starting frame of the path animation.

**Realtime** When enabled, the path animation will last exactly the same duration it took you do draw the strokes.

**End Frame** When *Realtime* is disabled, this defines the end frame of the path animation. This means that the drawing timing will be scaled up or down to fit into the specified range.

**Gap Duration** *Custom Gaps* only. The average duration (in frames) of each gap between actual strokes. Please note that the value entered here will only be exact if *Realtime* is enabled, else it will be scaled, exactly as the actual strokes’ timing is!

**Gap Randomness** Only when *Gap Duration* is non-null. The number of frames actual gap duration can vary of. This allows the creation of gaps having an average well defined duration, yet keeping some random variations to avoid an “always the same” effect.

**Random Seed** The seed fed to the random generator managing gaps duration variations. Change it to get another set of gaps duration in the path animation.

## Example

Here is a simple “hand writing” video created with curves converted from sketch data:

And the blend file: [File:ManGreasePencilConvertToCurveDynamicExample.blend](#)

## 2.11.5 Ruler and Protractor

The ruler can be accessed from the toolshelf, once activated you can use the ruler to measure lengths and angles in the scene.

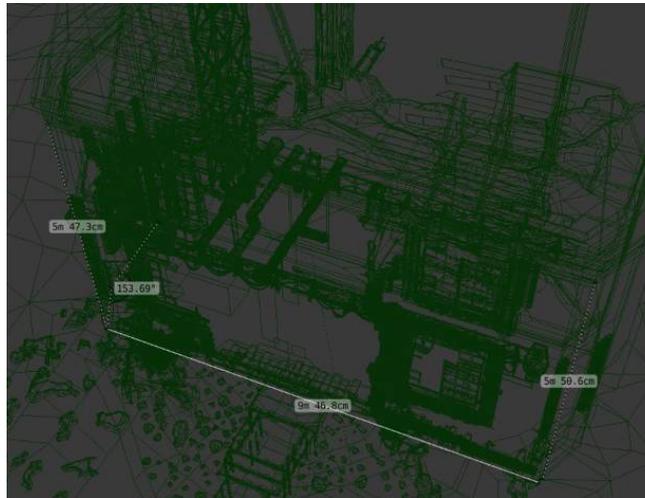


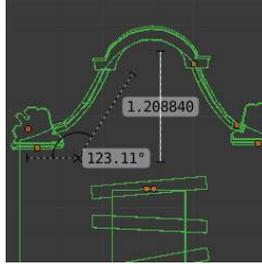
Fig. 2.2570: Example of the ruler and protractor.



Fig. 2.2571: Example using the ruler to measure thickness.

## Usage

Here are common steps for using the ruler.



- Activate the Ruler from the toolshelf.
- Click and drag in the view-port to define the initial start/end point for the ruler.
- Orbit the view and click on either end of the ruler to re-position it. Holding `Ctrl` enables snap to elements.
- Click on the middle to measure angles.
- Press `Return` to store the ruler for later use or `Escape` to cancel.

---

**Note:** Editing operations can be used while the ruler is running, however tools like the knife can't be used at the same time.

---



---

**Note:** Unit settings and scale from the scene are used for displaying dimensions.

---

## Shortcuts

- `Ctrl-LMB` Adds new ruler.
- `LMB` Drag end-points to place them, Hold `Ctrl` to snap, Hold `Shift` to measure thickness.
- `LMB` Drag center-point to measure angles, drag out of the view to convert back to a ruler.
- `Delete` Deletes the ruler.
- `Ctrl-C` Copies the rulers value to the clipboard.
- `Esc` Exits
- `Return` Saves the rulers for the next time the tool is activated.

## 2.12 Game Engine

### 2.12.1 Introduction to Game Engine

The Blender Game Engine (BGE) is Blender's tool for real time projects, from architectural visualizations and simulations to games.

A word of warning, before you start any big or serious project with the Blender Game Engine, you should note that it is currently not very supported and that there are plans for its retargeting and refactoring that, in the very least, will break compatibility. For further information, you should get in touch with the developers via mailing list or IRC and read the [development roadmap](#).

data-conversion → input | animation, physics and logic | rendering

different renderer, logic is done via logic bricks or python press 'P' to play in preview mode in the embedded player

use cases and sample games

Blender has its own built in Game Engine that allows you to create interactive 3D applications or simulations. The major difference between Game Engine and the conventional Blender system is in the rendering process. In the normal Blender engine, images and animations are built off-line - once rendered they cannot be modified. Conversely, the Blender Game Engine renders scenes continuously in real-time, and incorporates facilities for user interaction during the rendering process.



Fig. 2.2572: Screenshot from “Yo Frankie”, produced with Blender Game Engine

The Blender Game Engine oversees a game loop, which processes logic, sound, physics and rendering simulations in sequential order. The engine is written in C++.

By default, the user has access to a powerful, high level, Event Driven [Logic Editor](#) which is comprised of a series of specialised components called “Logic Bricks”. The [Logic Editor](#) provides deep interaction with the simulation, and its functionality can be extended through Python scripting. It is designed to abstract the complex engine features into a simple user interface, which does not require experience with Programming. An overview of the [Logic Editor](#) can be found in the [Game Logic Screen Layout](#)

The Game Engine is closely integrated with the existing code base of Blender, which permits quick transitions between the traditional modeling featureset and game-specific functionality provided by the program. In this sense, the Game Engine can be efficiently used in all areas of game design, from prototyping to final release.

The Game Engine can simulate content within Blender, however it also includes the ability to export a binary run-time to Windows, Linux and MacOS. There is also basic support for mobile platforms with the Android Blender Player GSOE 2012 project.

There are a number of powerful libraries included in the 2.5 / 2.6 releases of Blender, including:

- Recast - a state of the art navigation mesh construction toolset for games.
- Detour - a path-finding and spatial reasoning toolkit.
- Bullet - a physics engine featuring 3D collision detection, soft body dynamics, and rigid body dynamics
- Audaspace - a sound library for control of audio. Uses OpenAL or SDL

When creating a game or simulation in the BGE, there are four essential steps:

- Create visual elements that can be rendered. This could be 3D models or images.
- Enable interaction within the scene using logic bricks to script custom behaviour and determine how it is invoked (using the appropriate “sensors” such as keyboards or joysticks).
- Create one (or more) camera to give a frustum from which to render the scene, and modify the parameters to support the environment in which the game will be displayed, such as Stereo rendering.

- Launch the game, using the internal player or exporting a runtime to the appropriate platform.

## 2.12.2 Game Logic Screen Layout

The design, construction, debugging and running of a game utilises a wide range of Blender functions. To help with the process, Blender incorporates a suggested screen layout for setting up BGE games. This includes many already-familiar panels but also a new **Logic Editor** panel (4) concerned solely with the BGE.

The diagram below shows this default Game Logic screen layout, together with the appropriate options for game setup/debug/running (these should be set up in the order shown).

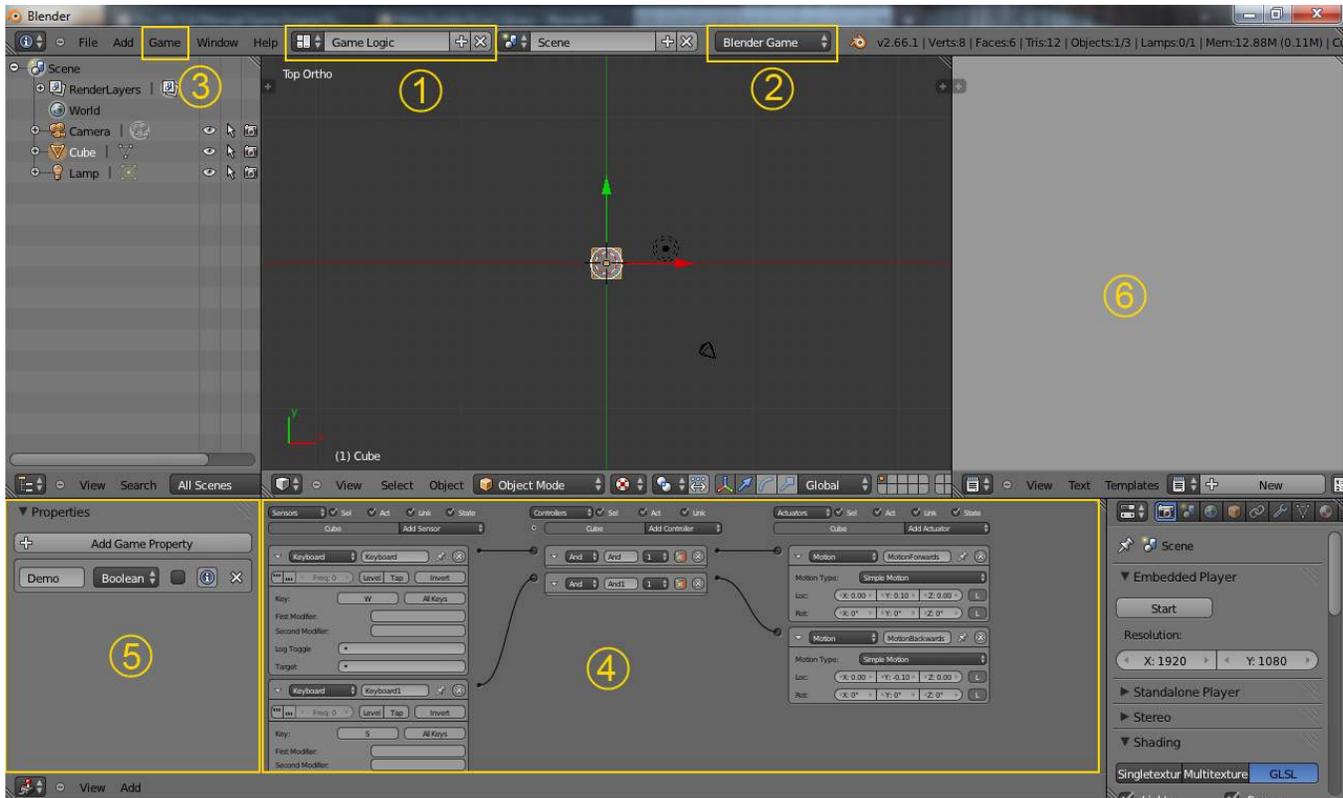


Fig. 2.2573: Game Logic Screen Layout

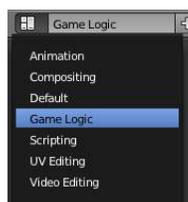


Fig. 2.2574: Game Logic Menu

1) **Game Logic** Selected from the list of screen layouts for various applications. This includes many already-familiar panels Information, 3D view, Properties but also a new Logic Editor panel concerned solely with the BGE.

2) **Blender Game** Selected from the render engine menu. This specifies that all output will be output by the real-time Blender



Fig. 2.2575: Render Engine Menu

Game Engine renderer. It also opens various other menu options such as the Game options (see below) and a range of Properties for the BGE renderer properties (see below)

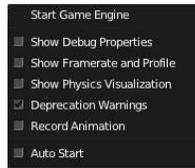


Fig. 2.2576: Game Options

**3) Game** This menu gives various options for conditions for running the Game Engine. Note that this menu is only available when the render engine is set to Blender Game.

**Start Game** Run game in Game Engine (shortcut `p` or `Shift-P` when the mouse cursor is over the 3D View window).

**Show Debug Properties** Show properties marked for debugging while game runs

**Show framerate and profile** :Show framerate and profiling information while game runs

**Show Physics visualization** show a vizualisation of physics bounds and interactions

**Deprecation warnings** Print warnings when using deprecated features in the python API

**Record animation** Record animation to F-curves

**Auto Start** Automatically start game at load time

**4) Logic Editor panel** The [Logic Editor](#) is where the [logic](#), [properties](#) and [states](#) are set up to control the behaviour of the objects in the game. (The Logic Editor panel can also be displayed by selecting Logic Editor in the Display Editor menu, by pressing `Shift-F2`, or by pressing `Ctrl-Right`).

## 5) Properties

---

**Tip:** Two Meanings for the Same Word

Note that the name “Property” has two different uses in Blender terminology - firstly in the wider use of the Property Display Panel as described here, and secondly as the term used for specific Game Engine logic variables which are also called “properties”.

---

The Property panel of the screen is selected as usual from the main Information menu. However note that several sections of the Property panel are changed when the render engine (2) is changed from Blender Render to Blender Game.

See following sections for details of the content of [Physics Properties](#) panels.

## 2.12.3 Logic

### Introduction

Game Logic is the default scripting layer in the game engine. Each `GameObject` in the game may store a collection of logical components (Logic Bricks) which control its behavior within the scene. Logic bricks can be combined to perform user-defined

actions that determine the progression of the simulation.

## Logic Bricks

The main part of game logic can be set up through a graphical interface the **Logic Editor**, and therefore does not require detailed programming knowledge. Logic is set up as blocks (or “bricks”) which represent preprogrammed functions; these can be tweaked and combined to create the game/application. There are three types of logic brick: **Sensors**, **Controllers** and **Actuators**. Sensors are primitive event listeners, which are triggered by specific events, such as a collision, a key press or mouse movement. Controllers carry out logic operations on sensor output, and trigger connected actuators when their operating conditions are met. Actuators interact with the simulation directly, and are the only components in the game which are able to do so (other than the Python controller, and other simulation components such as Physics)

## Properties

**Properties** are like variables in other programming languages. They are used to save and access data values either for the whole game (eg. scores), or for particular objects/players (e.g. names). However, in the Blender Game Engine, a property is associated with an object. Properties can be of different types, and are set up in a special area of the **Logic Editor**.

## States

Another useful feature is object **States**. At any time while the simulation is running, the object will process any logic which belongs to the current state of the object. States can be used to define groups of behavior - eg. an actor object may be “sleeping”, “awake” or “dead”, and its logic behavior may be different in each of these three states. The states of an object are set up, displayed and edited in the Controller logic bricks for the object.

## Logic Editor

The Logic Editor provides the main method of setting up and editing the game logic for the various actors (i.e. objects) that make up the game. The logic for the objects which are currently selected in the associated 3D panel are displayed as logic bricks, which are shown as a table with three columns, showing sensors, controllers, and actuators, respectively. The links joining the logic bricks conduct the pulses between sensor-controller and controller-actuator.

To give you a better understanding of the Logic Editor panel, the image below shows a typical panel content in which the major components have been labeled. We will look at each one individually.

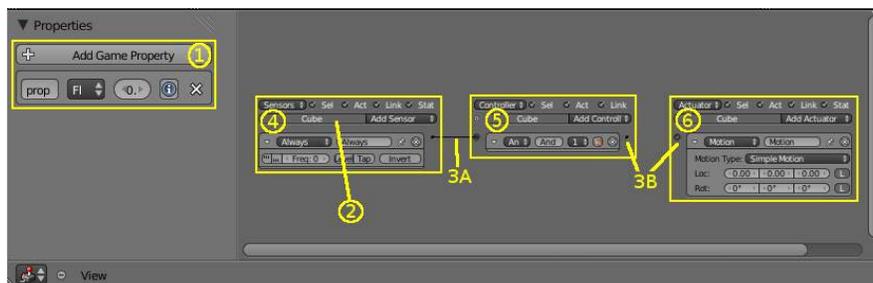


Fig. 2.2577: The different parts of the Logic Panel.

1. Game Property Area Game properties are like variables in other programming languages. They are used to save and access data associated with an object. Several types of properties are available. Properties are declared by clicking the Add Game Property button in this area. For a more in-depth look at the content, layout and available operations in this area, see [Properties](#).

2. **Object Name** This box shows the name of the object which owns the logic bricks below.
3. **Links** Links (3A) indicate the direction of logical flow between objects. Link lines are drawn by LMB dragging from one Link node (3B) to another. Links can only be drawn from Sensors to Controllers, or from Controllers to Actuators. You cannot directly link Sensors to Actuators; likewise, Actuators cannot be linked back to Sensors (however special actuator and sensor types are available to provide these connections).

Sending nodes (the black circles found on the right-hand side of Sensors and Controllers) can send to multiple Reception nodes (the white circles found on the left-hand side of Controllers and Actuators). Reception nodes can likewise receive multiple links.

Links can be created between logic bricks belonging to different objects.

To delete a link between two nodes, LMB drag between the two nodes.

4. **Sensor Area** This column contains a list of all sensors owned by the active object (and any other selected objects). New sensors for the active object are created using the “Add Sensor” button. For a more in-depth look at the content, layout and available operations in this area, see [Sensors](#).
5. **Controller Area** This column contains a list of all controllers owned by the active object (and any other selected objects). New controllers for the active object are created using the “Add Controller” button, together with the creation of states for the active object. For a more in-depth look at the content, layout, and available operations in this area, see [Controllers](#).
6. **Actuator Area** This column contains a list of all actuators owned by the active object (and any other selected objects). New actuators for the active object are created using the “Add Actuator” button. For a more in-depth look at the content, layout, and available operations in this area, see [Actuators](#).

## Sensors

### Introduction

Sensors are the logic bricks that cause the logic to do anything. Sensors give an output when something happens, e.g. a trigger event such as a collision between two objects, a key pressed on the keyboard, or a timer for a timed event going off. When a sensor is triggered, a positive pulse is sent to all controllers that are linked to it.

The logic blocks for all types of sensor may be constructed and changed using the [Logic Editor](#) details of this process are given in the [Sensor Editing](#) page.

The following types of sensor are currently available:

**Actuator** Detects when a particular actuator receives an activation pulse.

**Always** Gives a continuous output signal at regular intervals.

**Collision** Detects collisions between objects or materials.

**Delay** Delays output by a specified number of logic ticks.

**Joystick** Detects movement of specified joystick controls.

**Keyboard** Detects keyboard input.

**Message** Detects either text messages or property values

**Mouse** Detects mouse events.

**Near** Detects objects that move to within a specific distance of themselves.

**Property** Detects changes in the properties of its owner object.

**Radar** Detects objects that move to within a specific distance of themselves, within an angle from an axis.

**Random** Generates random pulses.

**Ray** Shoots a ray in the direction of an axis and detects hits.

**Touch** Detects when the object is in contact with another object.

### Sensor Editing



Fig. 2.2578: Sensor Column with Typical Sensor

Blender sensors can be set up and edited in the left-hand column of the Logic Panel. This page describes the general column controls, and also those parameters which are common to all individual sensor types.

The image shows a typical sensor column with a single example sensor. At the top of this column, the column heading includes menus and buttons to control which of all the sensors in the current Game Logic are displayed.



Fig. 2.2579: Sensor Column Heading

**Column Heading** The column headings contain controls to set which sensors, and the level of detail given, in the sensor column. This is very useful for hiding unnecessary sensors so that the necessary ones are visible and easier to reach. Both these can be controlled individually.

#### Sensors:

**Show Objects** Expands all objects.

**Hide Objects** Collapses all objects to just a bar with their name.

**Show Sensors** Expands all sensors.

**Hide Sensors** Collapses all sensors to bars with their names.

It is also possible to filter which sensors are viewed using the four heading buttons:

**Sel** Shows all sensors for selected objects.

**Act** Shows only sensors belonging to the active object.

**Link** Shows sensors which have a link to a controller.

**State** Only sensors connected to a controller with active states are shown.

**Object Heading** In the column list, sensors are grouped by object. By default, sensors for every selected object appear in the list, but this may be modified by the column heading filters.

At the head of each displayed object sensor list, two entries appear:

**Name** The name of the object.

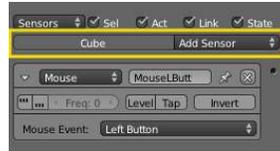


Fig. 2.2580: Sensor Object Heading

**Add Sensor** When clicked, a menu appears with the available sensor types. Selecting an entry adds a new sensor to the object. See [Sensors](#) for a list of available sensor types.

### Sensor Common Options

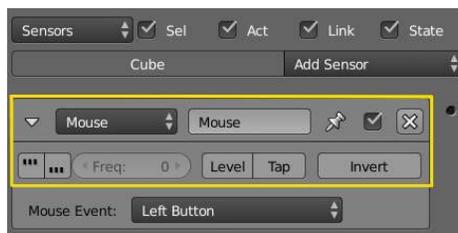


Fig. 2.2581: Common Sensor Options

All sensors have a set of common buttons, fields and menus. They are organized as follows:

**Triangle button** Collapses the sensor information to a single line (toggle).

**Sensor type menu** Specifies the type of the sensor.

**Sensor name** The name of the sensor. This can be selected by the user. It is used to access sensors with Python; it needs to be unique among the selected objects.

**Pin button** Display the sensor even when it is not linked to a visible states controller.

**Checkbox button** Sets active state of the sensor

**X button** Deletes the sensor.

---

**Note:** Note about triggers

If a controller does not get trigger by any connected sensor (regardless of the sensors' state) it will not be activated at all.

A sensor triggers the connected controllers on state change. When the sensor changes its state from negative to positive or positive to negative, the sensor triggers the connected controllers. A sensor triggers a connected controller as well when the sensor changes from deactivation to activation.

---

The following parameters specifies how the sensor triggers connected controllers:



Fig. 2.2582: True level triggering. If this is set, the connected controllers will be triggered as long as the sensor's state is positive. The sensor will trigger with the delay (see parameter: frequency) of the sensor.

**Freq** Despite its name "Frequency", this parameter sets the delay between repeated triggers, measured in frames (also known as logic ticks). The default value is 0 and it means no delay. It is only used at least one of the level triggering parameters are enabled.



Fig. 2.2583: False level triggering. If this is set, the connected controllers will be triggered as long as the sensor's state is negative. The sensor will trigger with the delay (see parameter: frequency) of the sensor.

Raising the value of *freq* is a good way for saving performance costs by avoiding to execute controllers or activate actuators more often than necessary.

Examples: (Assuming the default frame rate with a frequency of 60 Hz (60 frames per second)).

freq	meaning	frames with trigger	frames without trigger	period in frames	frequency in frames/sec
0	The sensor triggers the next frame.	1	0	1	60
1	The sensor triggers at one frame and waits another one until it triggers again. It results in half speed.	1	1	2	30
29	The sensor triggers one frame and waits 29 frames until it triggers again.	1	29	30	2
59	The sensor triggers one frame and waits 59 frames until it triggers again.	1	59	30	1

**Level Button** Triggers connected controllers when state (of the build-in state machine) changes. (For more information see [States](#)).

The following parameters specifies how the sensor's status gets evaluated:

**Tap Button** Changes the sensor's state to to negative one frame after changing to positive even if the sensor evaluation remains positive. As this is a state change it triggers the connected controllers as well. Only one of *Tap* or *Level* can be activated. If the *TRUE level triggering* is set, the sensor state will consecutive change from True to False until the sensor evaluates False. The *FALSE level triggering* will be ignored when the *Tap* parameter is set.

**Invert Button** This inverts the sensor output. If this is set, the sensor's state will be inverted. This means the sensors's state changes to positive when evaluating False and changes to False when evaluating True. If the *Tap* parameter is set, the sensor triggers the controller based on the inverted sensor state.

## Actuator sensor



Fig. 2.2584: Actuator sensor

The Actuator sensor detects when a particular actuator receives an activation pulse.

The *Actuator* sensor sends a TRUE pulse when the specified actuator is activated.

The sensor also sends a FALSE pulse when the specified actuator is deactivated.

See [Sensor Common Options](#) for common options.

Special Options:

**Actuator** Name of actuator (NB This must be owned by the same object).



Fig. 2.2585: Always sensor

### Always Sensor

The *Always* sensor is used for things that need to be done every logic tick, or at every  $x$  logic tick (with non-null  $f$ ), or at start-up (with *Tap*).

See [Sensor Common Options](#) for common options.

This sensor doesn't have any special options.

### Collision sensor



Fig. 2.2586: Collision sensor

A *Collision* sensor works like a *Touch* sensor but can also filter by property or material. Only objects with the property/material with that name will generate a positive pulse upon collision. Leave blank for collision with any object.

See [Sensor Common Options](#) for common options.

Special Options:

**Pulse button** Makes it sensible to other collisions even if it is still in touch with the object that triggered the last positive pulse.

**M/P button** Toggles between material and property filtering.

---

**Note:** Note about soft bodies

The *Collision* sensor can not detect collisions with soft bodies. This is a limitation in Bullet, the physics library used by the Game Engine.

---

### Delay sensor



Fig. 2.2587: Delay sensor

The *Delay* sensor is designed for delaying reactions a number of logic ticks. This is useful if an other action has to be done first or to time events.

See [Sensor Common Options](#) for common options. Special Options:

**Delay** The number of logic ticks the sensor waits before sending a positive pulse.

**Duration** The number of logic ticks the sensor waits before sending the negative pulse.

**Repeat Button** Makes the sensor restart after the delay and duration time is up.

### Joystick sensor



Fig. 2.2588: Joystick sensor

The *Joystick* sensor triggers whenever the joystick moves. It also detects events on a range of ancilliary controls on the joystick device (hat, buttons, etc.). More than one joystick may be used (see “Index”). The exact layout of the joystick controls will depend on the make and model of joystick used.

See [Sensor Common Options](#) for common options.

Special Options:

**Index** Specifies which joystick to use.

**All Events** Sensor triggers for all events on this joystick’s current type



Fig. 2.2589: Joystick Events

**Event Type** A menu to select which joystick event to use



Fig. 2.2590: Joystick Single Axis

**Single Axis** Detect movement in a single joystick Axis.

**Axis Number** 1 = Horizontal axis (left/right) 2 = Vertical axis (forward/back) 3 = Paddle axis up/down 4 = Joystick axis twist left/right



Fig. 2.2591: Joystick Hat

**Axis Threshold** Threshold at which joystick fires (Range 0 - 32768)

**Hat** Detect movement of a specific hat control on the joystick.

**Hat number** Specifies which hat to use (max. 2)

**Hat Direction** Specifies the direction to use: up, down, left, right, up/right, up/left, down/right, down/left.

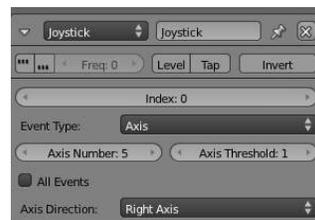


Fig. 2.2592: Joystick Axis

**Axis**

**Axis Number** Specifies the axis (1 or 2)

**Axis Threshold** Threshold at which joystick fires (Range 0 - 32768)

**Axis Direction** Specifies the direction to use:

(Axis Number = 1) Joystick Left, Right, Up, Down (Axis Number = 2) Paddle upper (Left); paddle Lower (Right); Joystick twist left (Up) Joystick twist right (Down)



Fig. 2.2593: Joystick Button

**Button** Specify the *button number* to use.



Fig. 2.2594: Keyboard sensor

## Keyboard Sensor

The *Keyboard* sensor is for detecting keyboard input. It can also save keyboard input to a `String` property.

See [Sensor Common Options](#) for common options.

Special Options:

**Key** This field detects presses on a named key. Press the button with no label and a key to assign that key to the sensor. This is the active key, which will trigger the TRUE pulse. Click the button and then click outside of the button to deassign the key.

A FALSE pulse is given when the key is released.

**All keys button** Sends a TRUE pulse when any key is pressed. This is useful for custom key maps with a `Python` controller.

**First Modifier, Second Modifier** Specifies additional key(s), all of which must be held down while the active key is pressed in order for the sensor to give a TRUE pulse. These are selected in the same way as **Key**. This is useful if you wish to use key combinations, for example `Ctrl-R` or `Shift-Alt-Esc` to do a specific action.

**LogToggle** Assigns a `Bool` property which determines if the keystroke will or will not be logged in the target `String`. This property needs to be TRUE if you wish to log your keystrokes.

**Target** The name of property to which the keystrokes are saved. This property must be of type `String`. Together with a `Property` sensor this can be used for example to enter passwords.

## Message Sensor



Fig. 2.2595: Message sensor

The *Message* sensor can be used to detect either text messages or property values. The sensor sends a positive pulse once an appropriate message is sent from anywhere in the engine. It can be set up to only send a pulse upon a message with a specific subject.

See [Sensor Common Options](#) for common options.

Special Options:

**Subject** Specifies the message that must be received to trigger the sensor (this can be left blank).

..note:

See `:doc:`Message Actuator </game_engine/logic/actuators/message>`` for how to send messages.

## Mouse sensor



Fig. 2.2596: Mouse sensor

The *Mouse* sensor is for detecting mouse events.

See [Sensor Common Options](#) for common options.

Special Options: The controller consist only of a list of types of mouse events. These are:

**Mouse over any** gives a TRUE pulse if the mouse moves over any game object.

**Mouse over** gives a TRUE pulse if the mouse moves over the owner object.

**Movement** any movement with the mouse causes a stream of TRUE pulses.

**Wheel Down** causes a stream of TRUE pulses as the scroll wheel of the mouse moves down.

**Wheel Up** causes a stream of TRUE pulses as the scroll wheel of the mouse moves up.

**Right button** gives a TRUE pulse.

**Middle button** gives a TRUE pulse.

**Left button** gives a TRUE pulse.

A FALSE pulse is given when any of the above conditions ends.

There is no logic brick for specific mouse movement and reactions (such as first person camera), these have to be coded in python.

## Near sensor



Fig. 2.2597: Near sensor

A *Near* sensor detects objects that move to within a specific distance of themselves. It can filter objects with properties, like the *Collision* sensor.

See [Sensor Common Options](#) for common options.

Special Options:

**Property** This field can be used to limit the sensor to look for only those objects with this property.

**Distance** The number of blender units it will detect objects within.

**Reset** The distance the object needs to be to reset the sensor (send a FALSE pulse).

Notes: 1) The Near sensor can detect objects “through” other objects (walls etc). 2) Objects must have “Actor” enabled to be detected.

**Note:** Note about soft bodies

The *Near* sensor can not detect soft bodies. This is a limitation in Bullet, the physics library used by the Game Engine.

## Property Sensor



Fig. 2.2598: Property sensor

The *Property* sensor detects changes in the properties of its owner object.

See [Sensor Common Options](#) for common options.

Special Options:



Fig. 2.2599: Property Evaluation

**Evaluation Type** Specifies how the property will be evaluated against the value(s).

**Greater Than** Sends a TRUE pulse when the property value is greater than the *Value* in the sensor.

**Less Than** Sends a TRUE pulse when the property value is less than the *Value* in the sensor.

**Changed** Sends a TRUE pulse as soon as the property value changes.

**Interval** Sends a TRUE pulse when the *Value* of the property is between the *Min* and *Max* values of the sensor.

**Not Equal** Sends a TRUE pulse when the property value differs from the *Value* in the sensor.

**Equal** Sends a TRUE pulse when the property value matches the *Value* in the sensor.

Note the names of other properties can also be entered to compare properties.

## Radar Sensor

The *Radar* sensor works much like a *Near* sensor, but only within an angle from an axis, forming an invisible cone with the top in the objects’ center and base at a distance on an axis.

See [Sensor Common Options](#) for common options.

Special Options:

**Property** This field can be used to limit the sensor to look for only those objects with this property.



Fig. 2.2600: Radar sensor

Notes: 1) The Radar sensor can detect objects “through” other objects (walls etc). 2) Objects must have “Actor” enabled to be detected.

**Axis** This menu determines the direction of the radar cone. The  $\pm$  signs is whether it is on the axis direction (+), or the opposite (-).

**Angle** Determines the angle of the cone. (Range: 0.00 to 179.9 degrees).

**Distance** Determines the length of the cone. (Blender units).

This sensor is useful for giving bots sight only in front of them, for example.

---

**Note:** Note about soft bodies

The *Radar* sensor can not detect soft bodies. This is a limitation in Bullet, the physics library used by the Game Engine.

---

### Random sensor



Fig. 2.2601: Random sensor

The *Random* sensor generates random pulses.

See [Sensor Common Options](#) for common options.

Special Options:

**Seed** This field to enter the initial seed for the random number algorithm. (Range 0-1000).

---

**Note:** 0 is not random, but is useful for testing and debugging purposes.

---

**Note:** If you run several times with the same Seed, the sequence of intervals you get will be the same in each run, although the intervals will be randomly distributed.

---

### Ray Sensor

The *Ray* sensor shoots a ray in the direction of an axis and sends a positive pulse once it hits something. It can be filtered to only detect objects with a given material or property.

See [Sensor Common Options](#) for common options.

Special Options: It shares a lot of buttons and fields with *Radar* sensor.



Fig. 2.2602: Ray sensor

**Property** This field can be used to limit the sensor to look for only those objects with this property.

---

**Note:**

1. Unless the Property field is set, the Ray sensor can detect objects “through” other objects (walls etc).
2. Objects must have “Actor” enabled to be detected.

---

**Axis** This menu determines the direction of the ray. The  $\pm$  signs is whether it is on the axis direction (+), or the opposite (-).

**Range** Determines the length of the ray. (Blender units).

**X-Ray Mode button** Makes it x-ray, so that it sees through objects that don't have the property or material specified in the filter field.

### Touch sensor



Fig. 2.2603: Touch sensor

The *Touch* sensor sends a positive pulse when the object is in contact with another object.

See [Sensor Common Options](#) for common options.

Special Options:

**Material** This field is for filtering materials. Only contact with the material in this field will generate a positive pulse. Leave blank for touch with any object.

A TRUE pulse is sent on collision and the FALSE pulse is sent once the objects are no longer in contact.

---

**Note:** Touch sensor has been removed in 2.69

The *Touch* sensor is no longer available in v2.69 or later. The [Collision Sensor](#) now provides the same functionality.

---

**Note:** Note about soft bodies

The *Touch* sensor can not detect collisions with soft bodies. This is a limitation in Bullet, the physics library used by the Game Engine.

## Controllers

### Introduction

The controllers are the bricks that collect data sent by the sensors, and also specify the state for which they operate. After performing the specified logic operations, they send out pulse signals to drive the actuators to which they are connected.

When a sensor is activated, it sends out a positive pulse, and when it is deactivated, it sends out a negative pulse. The controllers' job is to check and combine these pulses to trigger the proper response.

The logic blocks for all types of controller may be constructed and changed using the [Logic Editor](#); details of this process are given in the [Controller Editing](#) page.

**Controller Types** There are eight types of controller logic brick to carry out the logic process on the input signal(s): these are described in the separate pages shown below:

- [AND](#)
- [OR](#)
- [XOR](#)
- [NAND](#)
- [NOR](#)
- [XNOR](#)
- [Expression](#)
- [Python](#)

This table gives a quick overview of the logic operations performed by the logical controller types. The first column, input, represents the number of positive pulses sent from the connected sensors. The following columns represent each controller's response to those pulses. True means the conditions of the controller are fulfilled, and the actuators it is connected to will be activated; false means the controller's conditions are not met and nothing will happen. Please consult the individual controller pages for a more detailed description of each controller.

**Note:** It is assumed that more than one sensor is connected to the controller. For only one sensor, consult the "All" line.

Positive sensors	Controllers					
	<a href="#">AND</a>	<a href="#">OR</a>	<a href="#">XOR</a>	<a href="#">NAND</a>	<a href="#">NOR</a>	<a href="#">XNOR</a>
None	False	False	False	True	True	True
One	False	True	True	True	False	False
Multiple, not all	False	True	False	True	False	True
All	True	True	False	False	False	True

### Controller Editing

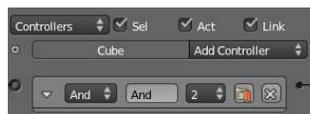


Fig. 2.2604: Controller Column with Typical Sensor

Blender controllers can be set up and edited in the central column of the Logic Panel. This page describes the general column controls, those parameters which are common to all individual controller types, and how different states for the objects in the logic system can be set up and edited.

The image shows a typical controller column with a single controller. At the top of this column, and for sensors and actuators, the column heading includes menus and buttons to control which of all the controllers in the current Game Logic are displayed.



Fig. 2.2605: Controller Column Headings

**Column Heading** The column headings contain controls to set which controllers appear, and the level of detail given, in the controller column. This is very useful for hiding unnecessary controllers so that the necessary ones are visible and easier to reach. Both these can be controlled individually.

#### Controllers:

**Show Objects** Expands all objects.

**Hide Objects** Collapses all objects to just a bar with their name.

**Show Controllers** Expands all Controllers.

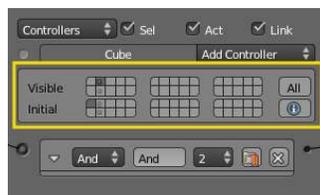
**Hide Controllers** Collapses all Controllers to bars with their names.

It is also possible to filter which controllers are viewed using the three heading buttons:

**Sel** Shows all controllers for selected objects.

**Act** Shows only controllers belonging to the active object.

**Link** Shows controllers which have a link to actuators/sensors.



**Object Heading** In the column list, controllers are grouped by object. By default, controllers for every selected object appear in the list, but this may be modified by the column heading filters.

**At the head of each displayed object controller list, three entries appear:** **(Used States Button)** Shows which states are in use for the object. Detailed description of the marked panel is given in [States](#).

**Name** The name of the object.

**Add Controller** When clicked, a menu appears with the available controller types. Selecting an entry adds a new controller to the object. See [Controllers](#) for a list of available controller types.

## AND Controller

This controller gives a positive (TRUE) output when All its inputs are TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options:



Fig. 2.2606: AND Controller

**Controller Type menu** Specifies the type of the controller.

**Controller Name** The name of the controller. This can be selected by the user. It is used to access controllers with python; it needs to be unique among the selected objects.

**State Index** Sets the designated state for which this controller will operate.

**Preference Button** If on, this controller will operate before all other non-preference controllers (useful for start-up scripts).

**X Button** Deletes the sensor.

## OR Controller

This controller gives a positive (TRUE) output when Any one or more of its inputs are TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options:



Fig. 2.2607: OR Controller

**Controller Type menu** Specifies the type of the controller.

**Controller Name** The name of the controller. This can be selected by the user. It is used to access controllers with python; it needs to be unique among the selected objects.

**State Index** Sets the designated state for which this controller will operate.

**Preference Button** If on, this controller will operate before all other non-preference controllers (useful for start-up scripts).

**X Button** Deletes the sensor.

## NAND Controller

This controller **activates** all connected actuators if

- the game object is in the designated state
- at least one connected sensor triggers the controller
- at least one connected sensor evaluated False

This controller **deactivates** all connected actuators if

- the game object is in the designated state
- at least one connected sensor triggers the controller
- ALL connected sensor evaluated True

Options:



Fig. 2.2608: NAND Controller

**Controller Type menu** Specifies the type of the controller.

**Controller Name** The name of the controller. This can be selected by the user. It is used to access controllers with python; it needs to be unique among the selected objects.

**State Index** Sets the designated state for which this controller will operate.

**Preference Button** If enabled, this controller will operate before all other non-preference controllers (useful for start-up scripts).

**X Button** Deletes the sensor.

### NOR Controller

This controller gives a positive (TRUE) output when None of its inputs are TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options:



Fig. 2.2609: NOR Controller

**Controller Type menu** Specifies the type of the controller.

**Controller Name** The name of the controller. This can be selected by the user. It is used to access controllers with python; it needs to be unique among the selected objects.

**State Index** Sets the designated state for which this controller will operate.

**Preference Button** If on, this controller will operate before all other non-preference controllers (useful for start-up scripts).

**X Button** Deletes the sensor.

### XOR Controller

This controller gives a positive (TRUE) output when One (and only one) of its inputs are TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options:



Fig. 2.2610: XOR Controller

**Controller Type menu** Specifies the type of the controller.

**Controller Name** The name of the controller. This can be selected by the user. It is used to access controllers with python; it needs to be unique among the selected objects.

**State Index** Sets the designated state for which this controller will operate.

**Preference Button** If on, this controller will operate before all other non-preference controllers (useful for start-up scripts).

**X Button** Deletes the sensor.

### XNOR Controller

This controller gives a positive (TRUE) output when One (and only one) of its inputs are FALSE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.

Options:



Fig. 2.2611: XNOR Controller

**Controller Type menu** Specifies the type of the controller.

**Controller Name** The name of the controller. This can be selected by the user. It is used to access controllers with python; it needs to be unique among the selected objects.

**State Index** Sets the designated state for which this controller will operate.

**Preference Button** If on, this controller will operate before all other non-preference controllers (useful for start-up scripts).

**X Button** Deletes the sensor.

### Expression Controller

This controller evaluates a user written expression, and gives a positive (TRUE) output when The result of the expression is TRUE, and The object is in the designated State. For all other conditions the controller gives a negative (FALSE) output.



Fig. 2.2612: Expression Controller

**Expression** The expression, which is written in the box, can consist of variables, constants and operators. These must follow the rules laid out below.

**Variables** You can use:

- **sensors names**,
- **properties** : assign a game property to an object and use it in a controller expression.

These cannot contain blank spaces.

## Operations

**Mathematical operations** Operators: \*, /, +, -

Returns: a number

Examples: 3 + 2, 35 / 5

## Logical operations

- Comparison operators: <, >, >=, <=, ==, !=
- Booleans operators: AND, OR, NOT

Returns: True or False.

Examples: 3 > 2 (True), 1 AND 0 (False)

**Conditional statement (if)** Use:

```
if( expression, pulse_if_expression_is_true, pulse_if_expression_is_false )
```

If the controller evaluates `expression` to True:

- if `pulse_if_expression_is_true` is True, the controller sends a positive pulse to the connected actuators.
- if `pulse_if_expression_is_true` is False, the controller sends a negative pulse to the connected actuators.

If the controller evaluates `expression` to False:

- if `pulse_if_expression_is_false` is True, the controller sends a positive pulse to the connected actuators.
- if `pulse_if_expression_is_false` is False, the controller sends a negative pulse to the connected actuators.

**Examples** Given the object has a property `coins` equal to 30:

```
coins > 20
```

returns True (the controller sends a positive pulse to the connected actuators).

Given the object has:

- a sensor called `Key_Inserted` equal to True,
- a property named `Fuel` equal to False,

```
Key_Inserted AND Fuel
```

returns False (the controller sends a negative pulse to the connected actuators).

This is the same as doing:

```
if (Key_Inserted AND Fuel, True, False)
```

Instead, you could do:

```
if (Key_Inserted AND Fuel, False, True)
```

to return a positive pulse when `Key_Inserted AND Fuel` returns `False`.

You can also do:

```
if ((Key_Inserted AND Fuel) OR (coins > 20), True, False)
```

This expression returns `True`, hence in this case the controller sends a positive pulse to the connected actuators.

## Python Controller

### Actuators

#### Introduction

Actuators perform actions, such as move, create objects, play a sound. The actuators initiate their functions when they get a positive pulse from one (or more) of their controllers.

The logic blocks for all types of actuator may be constructed and changed using the [Logic Editor](#); details of this process are given in the [Actuator Editing](#) page.

The following types of actuator are currently available:

**Action** Handles armature actions. This is only visible if an armature is selected.

**Camera** Has options to follow objects smoothly, primarily for camera objects, but any object can use this.

**Constraint** Constraints are used to limit object's locations, distance, or rotation. These are useful for controlling the physics of the object in game.

**Edit Object** Edits the object's mesh, adds objects, or destroys them. It can also change the mesh of an object (and soon also recreate the collision mesh).

**Filter 2D** Filters for special effects like sepia colors or blur.

**Game** Handles the entire game and can do things as restart, quit, load, and save.

**Message** Sends messages, which can be received by other objects to activate them.

**Motion** Sets object into motion and/or rotation. There are different options, from "teleporting" to physically push rotate objects.

**Parent** Can set a parent to the object, or unparent it.

**Property** Manipulates the object's properties, like assigning, adding, or copying.

**Random** Creates random values which can be stored in properties.

**Scene** Manage the scenes in your .blend file. These can be used as levels or for UI and background.

**Sound** Used to play sounds in the game.

**State** Changes states of the object.

**Steering** Provides pathfinding options for the object.

**Visibility** Changes visibility of the object.

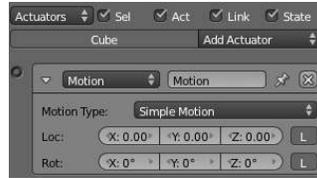


Fig. 2.2613: Actuator Column with Typical Actuator

## Actuator Editing

Blender actuators can be set up and edited in the right-hand column of the Logic Panel. This page describes the general column controls, and also those parameters which are common to all individual actuator types.

The image shows a typical actuator column with a single example actuator. At the top of this column, the column heading includes menus and buttons to control which of all the actuators in the current Game Logic are displayed.

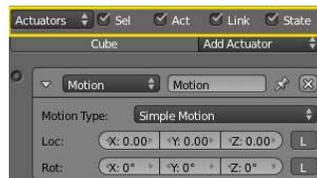


Fig. 2.2614: Actuator Column Heading

**Column Heading** The column headings contain controls to set which actuators, and the level of detail given, in the actuator column. This is very useful for hiding unnecessary actuators so that the necessary ones are visible and easier to reach. Both these can be controlled individually.

### Actuators

**Show Objects** Expands all objects.

**Hide Objects** Collapses all objects to just a bar with their name.

**Show Actuators** Expands all actuators.

**Hide Actuators** Collapses all actuators to bars with their names.

It is also possible to filter which actuators are viewed using the four heading buttons:

**Sel** Shows all actuators for selected objects.

**Act** Shows only actuators belonging to the active object.

**Link** Shows actuators which have a link to a controller.

**State** Only actuators connected to a controller with active states are shown.

**Object Heading** In the column list, actuators are grouped by object. By default, actuators for every selected object appear in the list, but this may be modified by the column heading filters.

At the head of each displayed object sensor list, two entries appear:

**Name** The name of the object.

**Add** When clicked, a menu appears with the available actuator types. Selecting an entry adds a new actuator to the object. See `Actuators` for list of available actuator types.

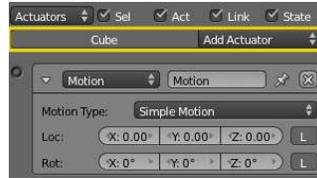


Fig. 2.2615: Actuator Object Heading

### Actuator Common Options

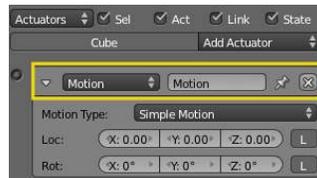


Fig. 2.2616: Common Actuator Options

All actuators have a set of common buttons, fields and menus. They are organized as follows:

**Triangle button** Collapses the sensor information to a single line (toggle).

**Actuator type menu** Specifies the type of the sensor.

**Actuator name** The name of the actuator. This can be selected by the user. It is used to access actuators with python; it needs to be unique among the selected objects.

**X Button** Deletes the actuator.

### Filter 2D Actuator

*2D Filter* s are image filtering actuators, that apply on final render of objects.



Fig. 2.2617: Edit Object actuator

**Filter 2D Type** Select the type of 2D Filter required.

*Custom Filter Invert Sepia Gray Scale Prewitt Sobel Laplacian Erosion Dilation Sharpen Blur Motion Blur Remove Filter Disable Filter Enable Filter*

Only one parameter is required for all filters:

**Pass Number** The pass number for which this filter is to be used.

Details of the filters are given in the descriptive text below.

**Motion Blur** *Motion Blur* is a *2D Filter* that needs previous rendering information to produce motion effect on objects. Below you can see *Motion Blur* filter in Blender window, along with its logic bricks:

You can enable Motion Blur filter using a *Python* controller: from bge import render render.enableMotionBlur(0.85)



Fig. 2.2618: 2D Filters: Motion Blur.

And disable it: `from bge import render render.disableMotionBlur()`

**Note:** Your graphic hardware and OpenGL driver must support accumulation buffer (`glAccum` function).

**Built-In 2D Filters** All 2D filters you can see in *2D Filter* actuator have the same architecture, all built-in filters use fragment shader to produce final render view, so your hardware must support shaders.



Fig. 2.2619: 2D Filters: Motion Blur.

*Blur*, *Sharpen*, *Dilation*, *Erosion*, *Laplacian*, *Sobel*, *Prewitt*, *Gray Scale*, *Sepia* and *Invert* are built-in filters. These filters can be set to be available in some passes.

To use a filter you should:

- Create appropriate sensor(s) and controller(s).
- Create a *2D Filter* actuator.
- Select your filter, for example *Blur*.
- Set the pass number that the filter will be applied.

To remove a filter on a specific pass:

- Create appropriate sensor(s) and controller(s).



Fig. 2.2620: 2D Filters: Sepia.

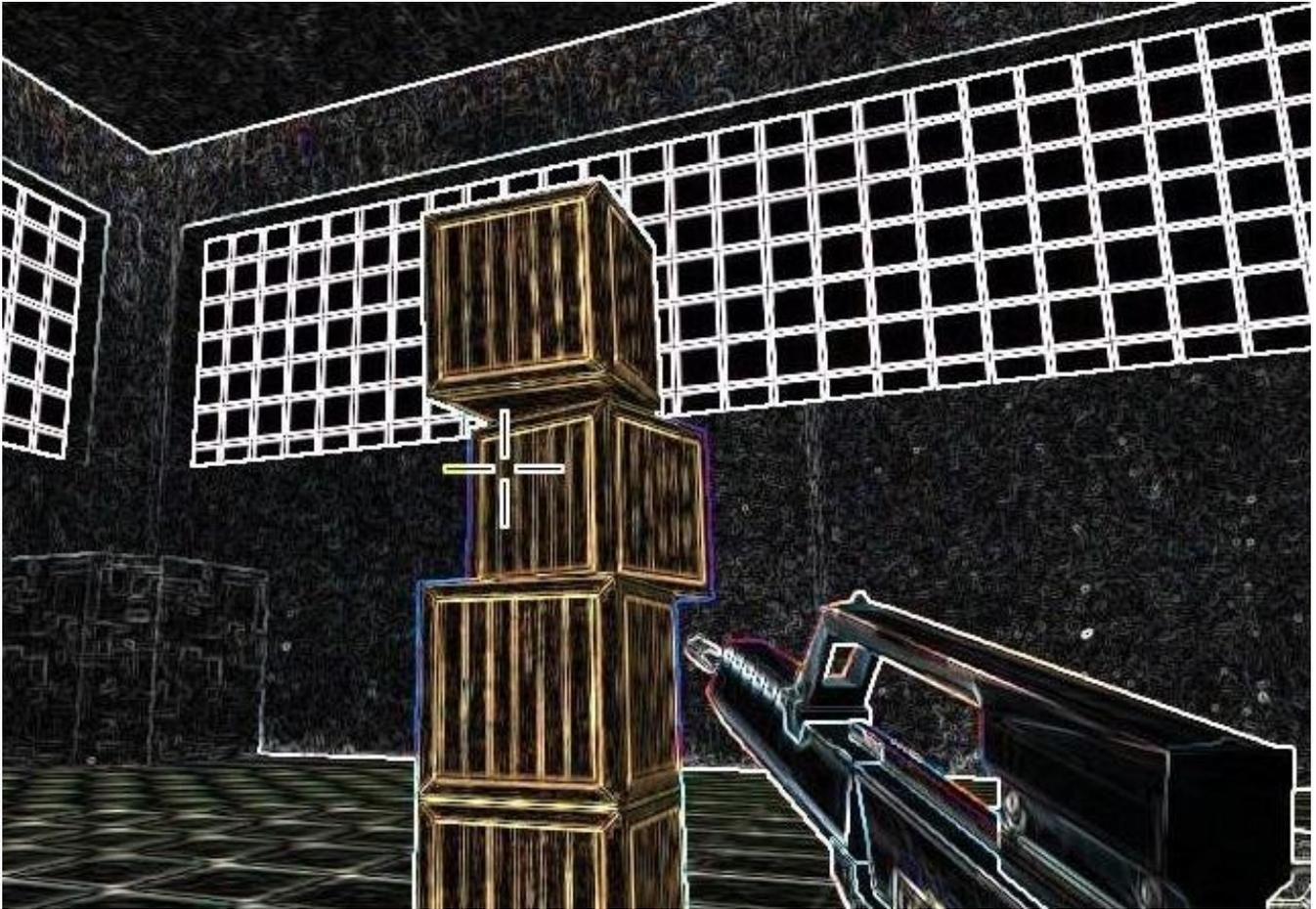


Fig. 2.2621: 2D Filters: Sobel.

- Create a *2D Filter* actuator.
- Select *Remove Filter*.
- Set the pass number you want to remove the filter from it.

To disable a filter on a specific pass:

- Create appropriate sensor(s) and controller(s).
- Create a *2D Filter* actuator.
- Select *Disable Filter*.
- Set the pass number you want to disable the filter on it.

To enable a filter on a specific pass:

- Create appropriate sensor(s) and controller(s)
- Create a *2D Filter* actuator.
- Select *Enable Filter*.
- Set the pass number you want to enable the filter on it.

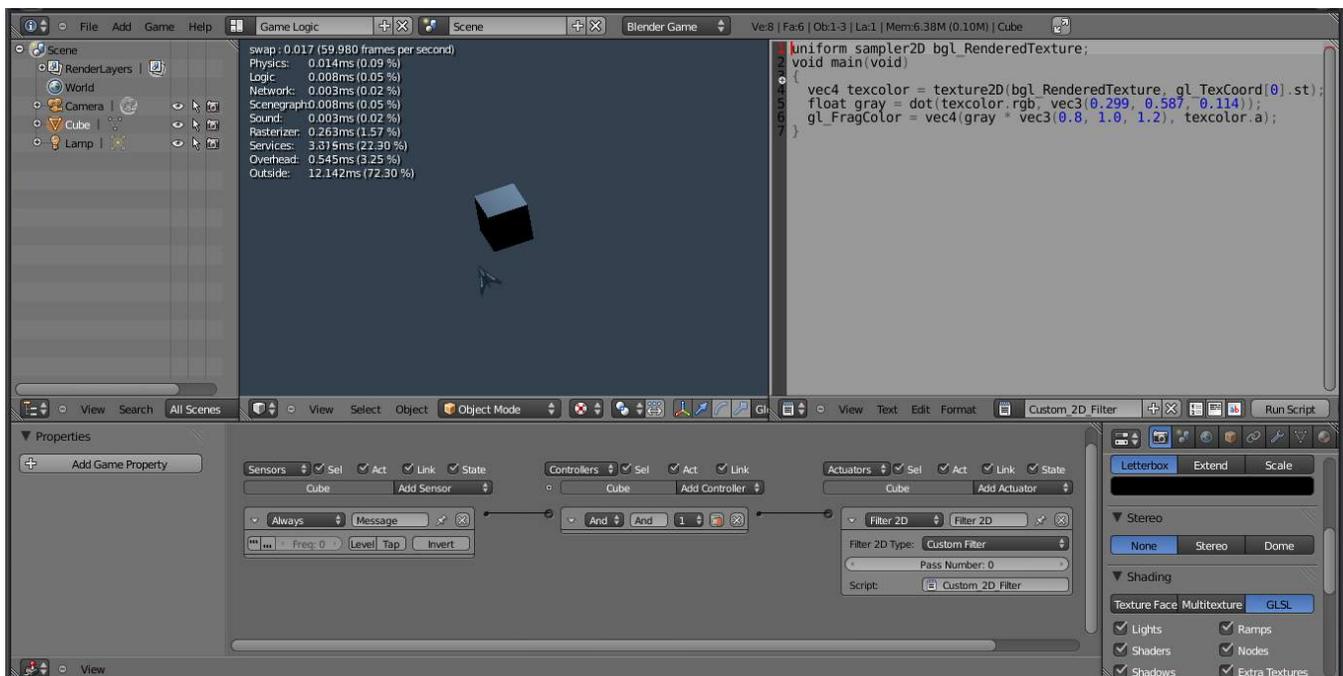


Fig. 2.2622: 2D Filters: Custom Filter.

**Custom Filters** Custom filters give you the ability to define your own 2D filter using GLSL. Its usage is the same as built-in filters, but you must select *Custom Filter* in *2D Filter* actuator, then write shader program into the Text Editor, and then place shader script name on actuator.

Blue Sepia Example:

```

uniform sampler2D bgl_RenderedTexture;
void main(void)
{
    vec4 texcolor = texture2D(bgl_RenderedTexture, gl_TexCoord[0].st);
}

```

```
float gray = dot(texcolor.rgb, vec3(0.299, 0.587, 0.114));
gl_FragColor = vec4(gray * vec3(0.8, 1.0, 1.2), texcolor.a);
}
```

## Action Actuator



Fig. 2.2623: Action Actuator

Actuates armature actions, and sets the playback method. The Action actuator is only visible when an armature is selected, because actions are stored in the armature.

See [Actuator Common Options](#) for common options.

Special Options: **Action Playback Type**

**Play** Play ipo once from start to end when a TRUE pulse is received.

**Ping Pong** Play ipo once from start to end when a TRUE pulse is received. When the end is reached play ipo once from end to start when a TRUE pulse is received.

**Flipper** Play ipo once from start to end when a TRUE pulse is received. (Plays backwards when a FALSE pulse is received).

**Loop End** Play ipo continuously from end to start when a TRUE pulse is received.

**Loop Start** Play ipo continuously from start to end when a TRUE pulse is received.

**Property** Uses a property to define what frame is displayed.

**Action** Select the action to use

**Continue** Restore last frame when switching on/off, otherwise play from the start each time.

**Start Frame** Set the start frame of the action.

**End Frame** Set the end frame of the action.

**Child Button** Update action on all children objects as well.

**Blendin** Number of frames of motion blending.

**Priority** Execution priority - lower numbers will override actions with higher numbers. With 2 or more actions at once, the overriding channels must be lower in the stack.

**Frame Property** Assign the action's current frame number to this property.

**Property** Use this property to define the Action position. Only for Property playback type.

**Layer** The animation layer to play the action on.

**Layer Weight** How much of the previous layer to blend into this one.

## Camera Actuator

Makes the camera follow or track an object.

See [Actuator Common Options](#) for common options.

Special Options:

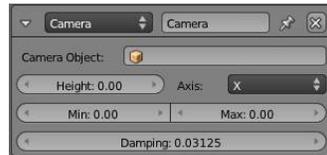


Fig. 2.2624: Camera Actuator

**Camera Object** Name of the Game Object that the camera follows/tracks.

**Height** Height the camera tries to stay above the Game Object's object center

**Axis** Axis in which the Camera follows (X or Y)

**Min** Minimum distance for the camera to follow the Game Object

**Max** Maximum distance for the camera to follow the Game Object

**Damping** Strength of the constraint that drives the camera behind the target. Range: 0 to 10. The higher the parameter, the quicker the camera will adjust to be inside the constrained range (of min, max and height).

## Constraints Actuator

Adds a constraint to the location, orientation

See [Actuator Common Options](#) for common options.

Special Options: **Constraint Mode** Menu specifying type of constraint required.

- Force Field Constraint
- Orientation Constraint
- Distance Constraint
- Location Constraint

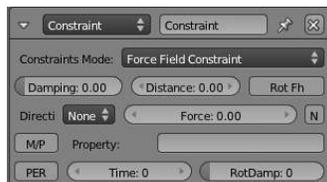


Fig. 2.2625: Constraint actuator - Force Field

**Force Field Constraint** Create a force field buffer zone along one axis of the object.

**Damping** Damping factor of the Fh spring force (Range 0.0 - 1.0)

**Distance** Height of Fh area

**Rot Fh** Make game object axis parallel to the normal of trigger object.

**Direction** Axis in which to create force field (can be + or -, or None)

**Force** Force value to be used.

**N** When on, use a horizontal spring force on slopes

**M/P** Trigger on another Object will be either Material (M) or Property (P)

**Property** Property/Material that triggers the Force Field constraint (blank for ALL Properties/Materials)

**Per** Persistence button When on, force field constraint always looks at Property/Material; when off, turns itself off if it can't find the Property/Material.

**Time** Number of frames for which constraint remains active

**RotDamp** Damping factor for rotation

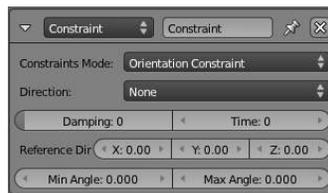


Fig. 2.2626: Constraint Actuator - Orientation

**Orientation Constraint** Constrain the specified axis in the Game to a specified direction in the World axis.

**Direction** Game axis to be modified (X, Y, Z or none)

**Damping** Delay (frames) of the constraint response (0 - 100)

**Time** Time (frames) for the constraint to remain active (0 - 100)

**ReferenceDir** Reference direction (global coordinates) for the specified game axis.

**MinAngle** Minimum angle for the axis modification;

**MaxAngle** Maximum angle for the axis modification;

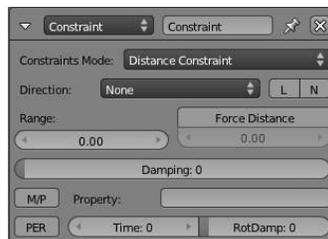


Fig. 2.2627: Constraint actuator - Distance

**Distance Constraint** Maintain the distance the Game Object has to be from a surface

**Direction** Axis Direction (X, Y, Z, -X, -Y, -Z, or None)

**L** If on, use local axis (otherwise use World axis)

**N** If on, orient the Game Object axis with the mesh normal.

**Range** Maximum length of ray used to check for Material/Property on another game object (0 - 2000 Blender Units)

**Force Distance** •Distance to be maintained between object and the Material/Property that triggers the Distance Constraint(-2000 to +2000 Blender Units).

**Damping** Delay (frames) of the constraint response (0 - 100)

**M/P** Trigger on another Object will be either Material (M) or Property (P)

**Property** Property/Material that triggers the Force Field constraint (blank for ALL Properties/Materials)

**Per** Persistence button: When on, force field constraint always looks at Property/Material; when off, turns itself off if it can't find the Property/Material.

**Time** Number of frames for which constraint remains active

**RotDamp** Damping factor for rotation

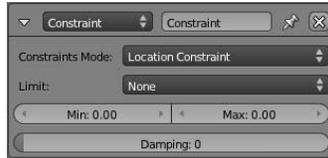


Fig. 2.2628: Constraint actuator - Location

**Location Constraint** Limit the position of the Game Object within one World Axis direction. To limit movement within an area or volume, use two or three constraints.

**Limit** Axis in which to apply limits (LocX, LocY, LocZ or none)

**Min** Minimum limit in specified axis (Blender Units)

**Max** Maximum limit in specified axis (Blender Units)

**Damping** Delay (frames) of the constraint response (0 - 100)

### Edit Object Actuator

The Edit Object actuator allows the user to edit settings of objects in game

See [Actuator Common Options](#) for common options.

Special Options:



Fig. 2.2629: Edit Object actuator

**Edit Object** Menu of options for Edit Object actuator

*Dynamics Track To Replace Mesh End Object Add Object*



Fig. 2.2630: Edit Object actuator - Dynamics

**Dynamics** Provides a menu of *Dynamic Operations* to set up dynamics options for object.

**Set Mass** Enables the user to set the mass of the current object for Physics (Range 0 - 10,000).

**Disable Rigid Body** Disables the Rigid Body state of the object - disables collision.

**Enable Rigid Body** Disables the Rigid Body state of the object - enables collision.

**Suspend Dynamics** Suspends the object dynamics (object velocity).

**Restore Dynamics** Resumes the object dynamics (object velocity).



Fig. 2.2631: Edit Object actuator - Track to

**Track To** Makes the object “look at” another object, in 2D or 3D. The Y-axis is considered the front of the object.

**Object** Object to follow.

**Time** No. of frames it will take to turn towards the target object (Range 0-2000).

**3D Button (toggle).** Enable 2D (X,Y) or 3D (X,Y,Z) tracking.

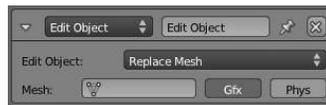


Fig. 2.2632: Edit Object actuator - Replace Mesh

**Replace Mesh** Replace mesh with another. Both the mesh and/or its physics can be replaced, together or independently.

**Mesh** name of mesh to replace the current mesh.

**Gfx Button** replace visible mesh.

**Phys Button** replace physics mesh (not compound shapes)



Fig. 2.2633: Edit Object actuator - End Object

**End Object** Destroy the current object (Note, debug properties will display error Zombie Object in console)

### Add Object

Adds an object at the centre of the current object.

**The object that is added needs to be on another, hidden, layer.**

**Object** The name of the object that is going to be added.:

**Time** The time (in frames) the object stays alive before it disappears. Zero makes it stay forever.

**Linear Velocity** Linear Velocity, works like in the motion actuator but on the created object instead of the object itself. Useful for shooting objects, create them with an initial speed.

**Angular Velocity** Angular velocity, works like in the motion actuator but on the created object instead of the object itself.



Fig. 2.2634: Edit Object actuator - Add Object

## Game Actuator

The Game actuator allows the user to perform Game-specific functions, such as Restart Game, Quit Game and Load Game. See [Actuator Common Options](#) for common options.

Special Options:



Fig. 2.2635: Game actuator

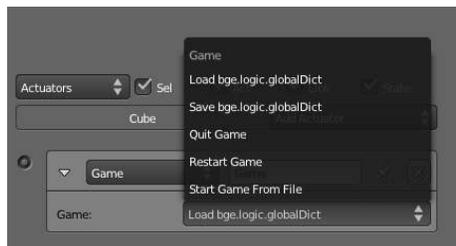


Fig. 2.2636: Game

## Game

**Load bge.logic.globalDict** Load *bge.logic.globalDict* from *.bgeconf*.

**Save bge.logic.globalDict** Save *bge.logic.globalDict* to *.bgeconf*.

**Quit Game** Once the actuator is activated, the blenderplayer exits the runtime.

**Restart Game** Once the actuator is activated, the blenderplayer restarts the game (reloads from file).

**Start Game From File** Once the actuator is activated, the blenderplayer starts the *.blend* file from the path specified.

**File** Path to the *.blend* file to load.

**Notes** If you use the keyboard sensor as a hook for the `ESC` key, in the event that the quit game actuator fails, such as an error in a python file, the game will be unable to close. Data may be recovered from *quit.blend File* → *Recover Last Session*

## Message Actuator

The Message actuator allows the user to send data across a scene, and between scenes themselves.

See [Actuator Common Options](#) for common options.

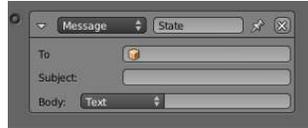


Fig. 2.2637: Message actuator

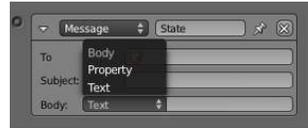


Fig. 2.2638: Message actuator Options

Special Options:

**To** Object to broadcast to. Leave blank if broadcast to all (or sending to another scene).

**Subject** Subject of message. Useful if sending certain types of message, such as “end-game”, to a message sensor listening for “end game”->AND->Quit Game actuator

**Body** Body of message sent (only read by Python\*).

**Text** User specified text in body.

**Property** User specified property.

**Usage Notes** You can use the Message Actuator to send data, such as scores to other objects, or even across scenes! (alternatively use `bge.logic.globalDict`).

## Motion Actuator

The Motion actuator sets an object into motion. There are two modes of operation, Simple or Servo, in which the object can either teleport & rotate, or dynamically move.

See [Actuator Common Options](#) for common options.

Special Options: **Motion Type**, which determines the type of motion:

**Simple Motion** Applies a change in location and/or rotation directly.

**Servo Control** Sets a target speed, and also how quickly it reaches that speed.

The *Simple Motion* actuator gives control over position and velocity, but does this as an instant displacement; the object never passes any of the coordinates between the start and end positions. This can interfere with the physical simulation of other objects, and can cause an object to go through another object. The *Servo Control* actuator does not suffer from this, since it produces physically correct velocities, and leaves updating the position to the physics simulation.



Fig. 2.2639: Motion actuator for Simple Motion

## Simple Motion

**Loc** The object jumps the number of blender units entered, each time a pulse is received.

**Rot** The object rotates by the specified amount, each time a pulse is received.

**L** Coordinates specified are Global (gray) or Local (White).

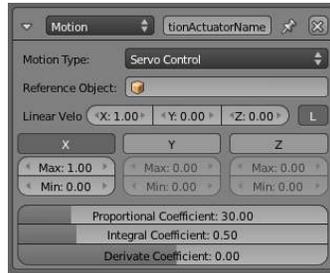


Fig. 2.2640: Motion actuator set to *Servo Control*

**Servo Control** The Servo Control actuator influences the velocity of a game object by applying forces, resulting in correct behavior when colliding with other objects controlled by the physics simulation. The amount of force necessary is determined by a **PID controller**, a type of controller that is often used in control systems. Only the positional velocity is influenced by this actuator; it does not control rotation at all, and it controls position only indirectly.

Controlling the position is not necessary in that respect; that is left to a player moving the object via direction-type controls (such as the WSAD keys in a first person shooter). In such a scenario, each direction-key sensor should be attached to a different Servo Control actuator setting a different target velocity.

**Tip:** To use the Servo Control actuator, it is necessary to set the object's Physics Type to "Dynamic" or "Rigid Body", and to mark the object as "Actor" in the same panel. This actuator does not work with the Character physics type.

**Reference Object** Specifies the object which the actuator uses as a reference for the velocity. When set, it will use a velocity relative to that object instead of absolute (i.e. world-relative) velocity. Use this for a player object standing on a moving platform.

**Linear Velocity** The target linear velocity for the object.

**L** Determines whether the Linear Velocity specified are in Local (button depressed) or Global (button released) coordinates.

**X, Y, Z force limits** Sets minimum and maximum limits for the force applied to the object. If disabled (i.e. X, Y or Z buttons are depressed) the force applied is unlimited.

The following three coefficients determine the response to the *velocity error*, which is the difference between the target velocity and the object's actual velocity.

**Proportional Coefficient** This controls the reaction proportional to the velocity error. Small values cause smooth (but possibly too slow) changes in velocity. Higher values cause rapid changes, but may cause overshooting.

**Integral Coefficient** This controls the reaction to the sum of errors so far. Using only the Proportional component results in a systematic velocity error if there is friction: some velocity delta is necessary to produce the force that compensates the friction. Using the Integral component suppresses this effect (the target velocity is achieved on average) but can create oscillations; the control will speed to compensate the initial velocity error. To avoid the oscillation, the Proportional component must be used with the Integral component (the Proportional component damps the control) This is why the GUI sets the Proportional Coefficient systematically when you change the Integral Coefficient.

**Derivative Coefficient** Set the Derivative Coefficient. This dampens the acceleration when the target velocity is almost reached.

## Parent Actuator

Enables you to change the parent relationships of the current object.

See [Actuator Common Options](#) for common options.

Special Options: **Scene** Menu for parenting operation required.



Fig. 2.2641: Parent Actuator

**Set Parent** Make this object to be current object's parent

**Parent Object** Name of parent object

**Compound'** Add this object shape to the parent shape (only if the parent shape is already compound)

**Ghost'** Make this object ghost while parented

**Remove Parent** Remove all parents of current object

**Parent Object** Name of parent object

## Property Actuator

Using the Property actuator you can change the value of a given property once the actuator itself is activated.

See [Actuator Common Options](#) for common options.

Special Options:

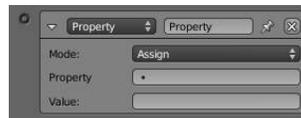


Fig. 2.2642: Property actuator

### Mode

**Assign** the *Property* target property will become equal to the set *Value* once the actuator is activated

**Add** adds *Value* to the value of the property *Property* once the actuator is activated (enter a negative value to decrease). For *Bool*, a value other than 0 (also negative) is counted as True.

**Copy** copies a property from another object to a property of the actuator owner once the actuator is activated.

**Toggle** switches 0 to 1 and any other number than 0 to 0 once the actuator is activated. Useful for on/off switches.

**Property** The target property that this actuator will change

**Value** The value to be used to change the property

**Example** You have a character, it has a property called “hp” (hit points) to determine when he has taken enough damage to die. hp is an int with the start value of 100.

You set up two *Collision* sensors, one for enemy bullets, and one for picking up more health. The first one is connected (through an *AND* controller) to an *Add Property* actuator with the property hp and the value -10. Every time the player is hit by an enemy bullet he loses 10 hp. The other sensor is connected (through an *AND* controller) to an other *Add Property* actuator, this one with the value 50. So every time the player collides with a health item the hp increases by 50. Next you set up a *Property* sensor for an interval, greater than 100. This is connected (through an *AND* controller) to an *Assign Property* actuator which is set to 100. So if the players hp increases over 100 it is set to 100.

## Random Actuator

Sets a random value into a property of the object

See [Actuator Common Options](#) for common options.

Special Options:



Fig. 2.2643: Camera Actuator

**Seed** Starting seed for random generator (range 1 - 1000)

**Distribution** Menu of distributions from which to select the random value. The default entry of Boolean Constant gives either True or False, which is useful for test purposes.

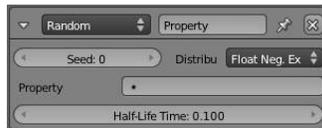


Fig. 2.2644: Float Neg. Exp.

**Float Neg. Exp.** Values drop off exponentially with the specified half-life time.

**Property** Float property to receive value

**Half-Life Time** Half-life time (Range 0.00 -10000.00)



Fig. 2.2645: Float Normal

**Float normal** Random numbers from a normal distribution.

**Property** Float property to receive value

**Mean** Mean of normal distribution (Range -10000.00 to +10000.00)

**SD** Standard deviation of normal distribution (Range 0.00 to +10000.00)



Fig. 2.2646: Float Uniform

**Float uniform** Random values selected uniformly between maximum and minimum.

**Property** Float property to receive value

**Min** Minimum value (Range -10000.00 to +10000.00)

**Max** Maximum value (Range -10000.00 to +10000.00)

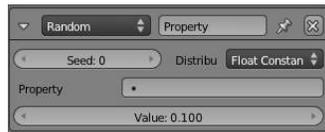


Fig. 2.2647: Float Constant

**Float constant** Returns a constant value.

**Property** Float property to receive value

**Value** Value (Range 0.00 to +1.00)



Fig. 2.2648: Random Integer Poisson

**Int Poisson** Random numbers from a Poisson distribution.

**Property** Integer property to receive value

**Mean** Mean of Poisson distribution (Range 0.01 to +100.00)

**Int uniform** Random values selected uniformly between maximum and minimum.

**Property** Integer property to receive value

**Min** Minimum value (Range -1000 to +1000)

**Max** Maximum value (Range -1000 to +1000)

**Int constant** Returns a constant value.

**Property** Integer property to receive value

**Value** Value (Range 0.00 to +1.00)

**Bool Bernoulli** Returns a random distribution with specified ratio of TRUE pulses.

**Property** Boolean property to receive value

**Chance** Proportion of TRUE responses required.

**Bool uniform** A 50/50 chance of obtaining True/False.



Fig. 2.2649: Random Integer Uniform



Fig. 2.2650: Random Integer Constant

**Property** Boolean property to receive value

**Bool constant** Returns a constant value.

**Property** Boolean property to receive value

**Value** Value (True or False)

## Scene Actuator

The *Scene* actuator manages the scenes in your .blend file, these can be used as levels or for UI and background.

See [Actuator Common Options](#) for common options.

Special Options: The actuator has eight modes:

**Restart** Restarts the current scene, everything in the scene is reset

**Set Scene** Changes scene to selected one

**Set Camera** Changes which camera is used

**Add OverlayScene** This adds an other scene, and draws it on top of the current scene. It is good for interfacing: keeping the health bar, ammo meter, speed meter in an overlay scene makes them always visible.

**Add BackgroundScene** This is the opposite of an overlay scene, it is drawn behind the current scene

**Remove Scene** Removes a scene.

**Suspend Scene** Pauses a scene

**Resume Scene** Resumes a paused scene.

---

**Note:** A scene that it is paused can not resume itself. You need an active scene to resume other scene that it is paused.

---



Fig. 2.2651: Random Bool Bernoulli



Fig. 2.2652: Random Bool Uniform



Fig. 2.2653: Random Bool Constant

### Sound Actuator

Select a sound file from the list or make a new one.

See [Actuator Common Options](#) for common options.

Special Options:

**Music File title** Select music file from the list presented.

### State Actuator

The State actuator allows the user to create complex logic, whilst retaining a clear user interface. It does this by having different states, and performing operations upon them

See [Actuator Common Options](#) for common options.

Special Options:

#### Operation

Menu to select the state operation required.

**Change State** Change from the current state to the state specified.

**Remove State** Removes the specified states from the active states (deactivates them).

**Add State** Adds the specified states to the active states (activates them).

**Set State** Moves from the current state to the state specified, deactivating other added states.

**Usage Notes** With the state actuator, you can create tiers of logic, without the need for hundreds of properties. Use it well, and you benefit greatly, but often problems may be circumvented by python.



Fig. 2.2654: Scene actuator

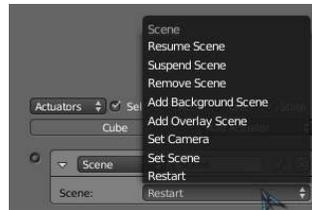


Fig. 2.2655: Scene actuator options



Fig. 2.2656: Sound Actuator

### Steering Actuator

The steering actuator moves an object towards a target object, with options to seek, flee, or follow a path. This actuator will not actually try to avoid obstacles by deviating the objects course.

See [Actuator Common Options](#) for common options.

#### Options

**Behavior:** Seek, Flee or Path following

**Target Object:** The game object to seek.

**Navigation Mesh Object:** The name of the navigation mesh object used by the Steering Actuator when in Path following behavior. The game object will use the Navigation Mesh to create a path to follow the Target Object.

**Tip:** You can create your own mesh to use for navigation and make it a Navigation Mesh in:

- Properties Window → Physics context → Physics panel → choosing Physics Type: Navigation Mesh

Or you can let Blender create a Navigation Mesh, then select a mesh. (Floor or ground or etc.)

- Properties Window → Scene context → Navigation mesh object panel → Build navigation mesh

**Dist** The maximum distance for the game object approach the Target Object.

**Velocity** The velocity used to seek the Target Object.

**Acceleration** The maximum acceleration to use when seeking the Target Object.

**Turn Speed** The maximum turning speed to use when seeking the Target Object.

**Facing:** Set a game object axis that always faces the Target Object.

**Axis** The game object axis that always faces the Target Object. Options are: Positive (X, Y, Z) and Negative (- X, - Y, -Z).



Fig. 2.2657: State actuator

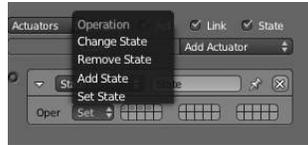


Fig. 2.2658: State actuator options

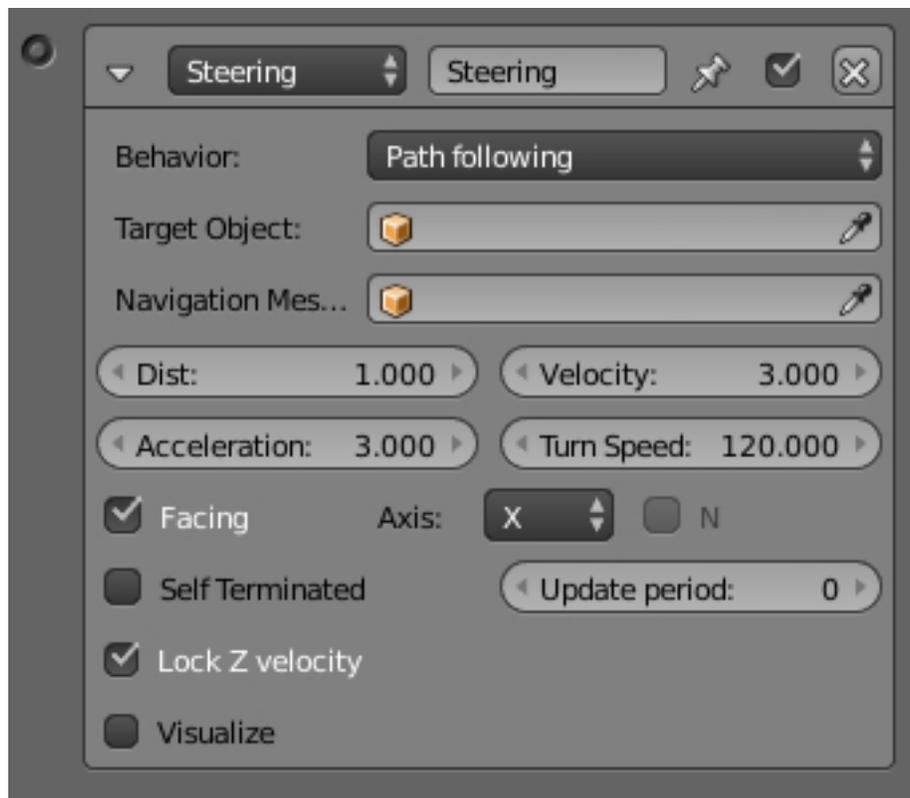


Fig. 2.2659: Steering Actuator Panel

**Axis N** Use the Normal of the Navigation Mesh to align the up vector of the game object.

Self Terminated:

**Disabled** Stops moving toward the Target Object once it reaches the maximum distance to approach the Target Object. Will follow the Target Object if it moves further away than the maximum distance.

**Enabled** Stops moving toward the Target Object once it reaches the maximum distance to approach the Target Object. Won't follow even if the Target Object moves further away than the maximum distance.

**Visualize** This checkbox let the user specify whether to show or not the debug informations of the actuator. It is also necessary to enable Debug Properties in the Display menu of the render context.

## Visibility Actuator

The Visibility actuator allows the user to change the visibility of objects during runtime.



Fig. 2.2660: Visibility actuator

See [Actuator Common Options](#) for common options.

Special Options:

**Visible** Toggle checkbox to toggle visibility

**Occlusion** Toggle checkbox to toggle occlusion. Must be initialised from the Physics tab.

**Children** Toggle checkbox to toggle recursive setting - will set visibility / occlusion state to all child objects, children of children (recursively)

**Usage Notes** Using the visibility actuator will save on Rasterizer usage, however not Physics, and so is limited in terms of Level of Detail (LOD). For LOD look at replace mesh, but be aware that the logic required can negate the effect of the LOD.

## Properties

Properties are the game logic equivalent to variables. They are stored with the object, and can be used to represent things about them such as ammo, health, name, and so on.

### Property Types

There are five types of properties:

**Timer** Starts at the property value and counts upwards as long as the object exists. It can for example be used if you want to know how long time it takes the player to complete a level.

**Float** Uses decimal numbers as values, can range from -10000.000 to 10000.000. It is useful for precision values.

**Integer** Uses integers (whole numbers) as values, between -10000 and 10000. Useful for counting things such as ammunition, where decimals are unnecessary.

**String** Takes text as value. Can store 128 characters.

**Boolean** Boolean variable, has two values: true or false. This is useful for things that have only two modes, like a light switch.

**Using Properties** When a game is running, values of properties are set, manipulated, and evaluated using the [Property Sensor](#) and the [Property Actuator](#).

Logic Properties are created and edited using the panel on the left of the Logic Editor Panel. The top menu provides a list of the available property types.



Fig. 2.2661: Properties Panel of the Logic Editor

**Add Game Property button** This button adds a new property to the list, default is a *Float* property named `prop`, followed by a number if there already is one with this name.

**Name field** Where you give your property its name, this is how you are going to access it through python or expressions. The way to do so in python is by dictionary style lookup (`GameObject["propname"]`). The name is case sensitive.

**Type menu** This menu determines which type of property it is. The available options are in [Property Types](#).

**Value field** Sets the initial value of the property.

**i information button** Display property value in debug information. If debugging is turned on, the value of the property is given in the top left-hand corner of the screen while the game is running. To turn debugging on, tick *Show Debug Properties* in the *Game* menu. All properties with debugging activated will then be presented with their object name, property name and value during gameplay. This is useful if you suspect something with your properties is causing problems.

## States

In the BGE, an object can have different “states”. At any time while the game is playing, the current state of the object defines its behavior. For instance, a character in your game may have states representing awake, sleeping or dead. At any moment their behavior in response to a loud bang will be dependent on their current state; they may crouch down (awake); wake up (asleep) or do nothing (dead).

### How States Operate

States are set up and used through controllers: note that only controllers, not actuators and sensors, are directly controlled by the state system. Each object has a number of states (up to 30; default = 1), and can only be in one state at any particular time. A controller must always specify the state for which it will operate - it will only give an output pulse if a) its logic conditions are met, and b) the object is currently in the specified State. States are set up and edited in the object’s Controller settings (for details see below).

---

**Tip:** State settings are automatic in simple games. By default, the number of states for each object is 1, and all controllers are set to use State 1. So, if a game does not need multiple states, everything will work without explicitly setting states - you do not need to bother about states at all.

---

One of the actuators, the State actuator, can set or unset the object’s State bits, and so allow the object’s reaction to a sensor signal to depend on its current state. So, in the above example, the actor will have a number of controllers connected to the “loud bang” sensor, for each of the “awake”, “asleep” or “dead” states. These will operate different actuators depending on the current state of the actor, and some of these actuators may switch the actor’s state under appropriate conditions.

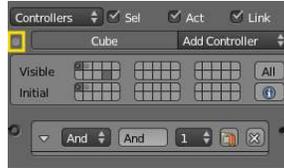


Fig. 2.2662: State Panel Button

## Editing States

States are set up and edited using the Controller (center) column of the Game Logic Panel. To see the State panel, click on the State Panel Button shown. The panel shows two areas for each of the 30 available states; these show Visible states, and Initial states (see below). Setting up the State system for a game is performed by choosing the appropriate state for each controller in the object's logic.

The display of an object's state logic, and other housekeeping, is carried out using the State Panel for the object, which is switched on and off using the button shown. The panel is divided into two halves, Visible and Initial.

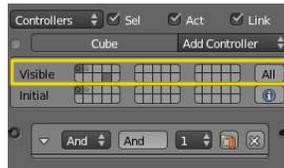


Fig. 2.2663: State Panel Visible

## Visible States

In the Visible area, each of the 30 available states is represented by a light-gray square. This panel shows what logic is visible for the logic brick displayed for the object. At the right is the All button; if clicked, then all the object's logic bricks are displayed (this is a toggle), and all State Panel squares are light-gray. Otherwise, individual states can be clicked to make their logic visible. (Note that you can click more than one square). Clicking the square again unselects the state.

States for the object that are in use (i.e. the object has controllers which operate in that state) have dots in them, and squares are dark-gray if these controllers are shown in the Game Logic display. The display of their connected sensors and actuators can also be controlled if the State buttons at the head of their columns are ticked.

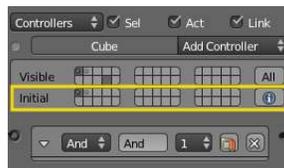


Fig. 2.2664: State Panel Initial

## Initial State

In the Initial area, each of the 30 available states is again represented by a light-gray square. One of these states may be clicked as the state in which the object starts when the game is run.

At the right is the I (Information) button; if clicked, and the (Game) Show Debug Properties menu entry is clicked, the current state of the object is shown in the top left-hand corner of the display while the game is running.

## 2.12.4 Camera

### Camera

The Game Engine camera is in many ways similar to the Camera in the normal Blender Render system, and is created, parameterized and manipulated in similar ways. However because of its use as a real-time device, the Game Engine camera has a number of additional features - it may be used as not only as a static camera, but also as a moving device with its default characteristics (ie. with its own programmed moves), or it may track another object in the game. Furthermore, any game object may be used as a camera; the view is taken from the object's origin point. Lastly, it may be given special capabilities such as Stereo vision, Dome visualisation etc. which have special relevance to game technology.

When you start the Game Engine, the initial camera view is taken from the latest 3D View. This may be either a selected camera object or the default camera (see below). Thus to start the game with a particular camera, you must select the camera and press `Numpad0` before starting the Game Engine.

---

**Tip:** To avoid camera distortion

Always zoom the view in until the camera object fills the entire viewport.

---

### Default Camera

The default camera view is taken from the latest 3D viewport view, at a distance equivalent to the viewer. This means that if the normal 3D view is active the scene does not change when the Game Engine is started.

### Camera Object

The Camera object in the Game Engine follows much the same structure as the conventional Blender camera - see [Camera](#) for details of how to set up, manipulate and select a camera. The following sections show some of the special facilities available in BGE cameras.

### Parent Camera to Object

The camera will follow the object. First select the camera and then select the object. Next `Ctrl-P` → *Make Parent*.

Note that if your object has any rotations then the camera will also have those rotations. To avoid this use *Parent to Vertex*.

### Parent to Vertex

The easiest way to accomplish this is to select your object and `Tab` to *Edit mode*. Now select the vertex and `Tab` back to *Object mode*.

Next, without any objects selected, select the camera and, holding the `Shift` key, select the object. `Tab` into *Edit mode*, and `Ctrl-P` and choose *Make vertex parent*.

Now the camera will follow the object and it will maintain its rotation, while the object rotates.

### Object as a Camera

Any object may also become a camera with whatever properties are set for the object.

To make an object the camera, in *Object mode* select the object and press `Ctrl-Numpad0` on the numpad.

To reverse it, just select the camera and `Ctrl-Numpad0` again.

### Camera Lens Shift

In the Blender interface, there is an option to shift the camera view on the  $x/y$  plane of the view. It is comparable to lens shift in video projectors that usually shift the image up along the  $Y$  axis. So for example, when you put the beamer on a table it does not project half of the image on the table.

Unfortunately, this parameter is not taken in account by the Game Engine.

To manipulate the projection we can then directly modify the camera projection matrix in Python.

```
import bge
scene = bge.logic.getCurrentScene()
cam = scene.active_camera
# get projection matrix
camatrix = cam.projection_matrix
# modifying the camera projection matrix by modifying the x and y terms
# of the 3rd row to obtain a shift of the rendered area
camatrix[2][0] = 2*shiftx
camatrix[2][1] = 2*shifty
cam.projection_matrix = camatrix
```

Here in field of view units are `shiftx` and `shifty`. So for example, for shifting the view up half a screen `shifty` is set to 0.5.

Note that a camera's `projection_matrix` attribute may not be set until after initialization scripts are executed and running this code immediately after the game starts will mess up the projection matrix.

### Camera Editing

The camera (or cameras) used in a Blender game have a wide-ranging effect on the way in which the game is rendered and displayed. Mostly this is controlled using the Properties panel of the camera(s) used in the game.

---

#### Tip: Render Engine

Make sure that the render engine is set to Blender Game when attempting to set these controls - otherwise this description will not tally with what you see!

---

In the Camera Properties area, there are six panels available, as shown. Each can be expanded or contracted using the usual triangle button. The features in each panel will be described in detail below.

#### Embedded Player

**Start** button - Start the Game Engine. Shortcut `P`.

#### Standalone Player

This panel provides information for the Standalone Game Player which allows games to be run without Blender. See [Standalone Player](#) for further details.

**Fullscreen** - Off - opens standalone game as a new window. On - opens standalone game in full screen.

#### Resolution

**X** Sets the  $X$  size of the viewport for full-screen display.

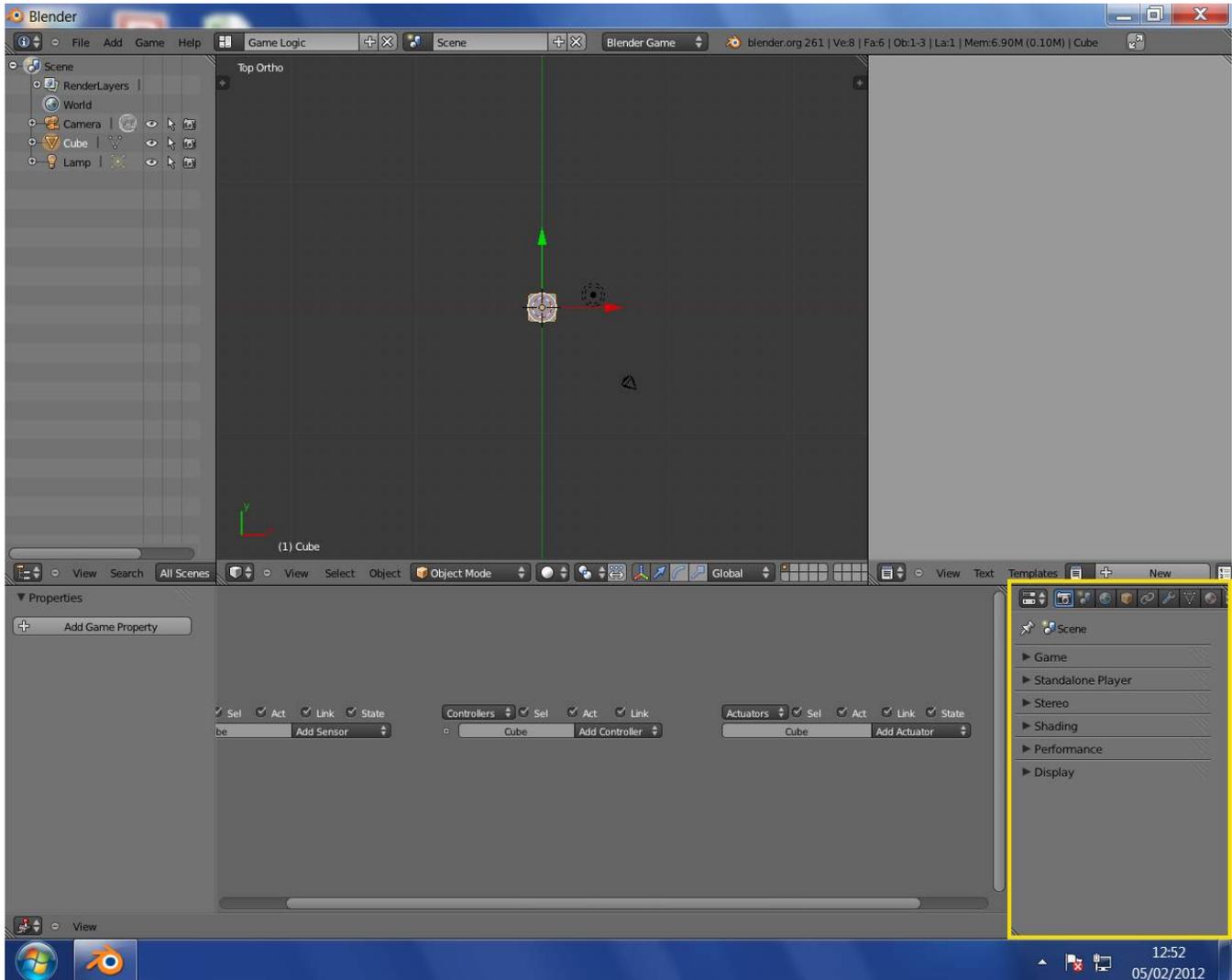


Fig. 2.2665: Camera Properties



Fig. 2.2666: Game Panel

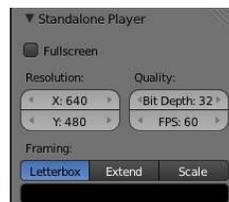


Fig. 2.2667: Standalone Panel

**Y** Sets the Y size of the viewport for full-screen display.

### Quality

**Bit Depth** Number of bits used to represent color of each pixel in full-screen display.

**FPS** Number of frames per second of full-screen display.

**Framing** Shows how the display is to be fitted in to the viewport.

**Letterbox** Show the entire viewport in the display window, and fill the remainder with the “bar” color.

**Extend** Show the whole display in the viewport, and fill the remainder with bars.

**Scale** Scale the display in X and Y to exactly fill the entire viewport.

**Bar Color** Select a color to use as the color of bars around the viewport (default black). To use this, select a color mode (RGB, HSV or Hex), then use the color slider and color wheel to choose a bar color.

### Stereo



Fig. 2.2668: Stereo Panel

Select a stereo mode that will be used to capture stereo images of the game (and also, by implication, that stereo displays will use to render images in the standalone player).

**None** Render single images with no stereo.

**Stereo** Render dual images for stereo viewing using appropriate equipment. See [Stereo Camera](#) for full details of available options.

**Dome** Provides facilities for an immersive dome environment in which to view the game. See [Dome Camera](#) for full details of available options.

### Shading



Fig. 2.2669: Shading Panel

Specifies the shading mode to be used in rendering the game. The shading facilities available in Blender for use in [Materials](#) and [Textures](#) are essentially the same in the Blender Game Engine. However the constraints of real-time display mean that only some of the facilities are available.

**Single Texture** Use single texture facilities.

**Multitexture** Use Multitexture shading.

**GLSL** Use GLSL shading. GLSL should be used whenever possible for real-time image rendering.



Fig. 2.2670: Performance Panel

## Performance

**Use Frame Rate** Respect the frame rate rather than rendering as many frames as possible.

**Display Lists** Use display lists to speed up rendering by keeping geometry on the GPU.

**Restrict Animation Updates** Restrict number of animation updates to the animation FPS (this is better for performance but can cause issues with smooth playback).

## Display

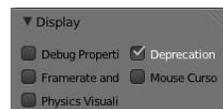


Fig. 2.2671: Display Panel

Gives various display options when running the Game Engine. Under the...

**Debug Properties** Show properties marked for debugging while game runs. Note that debug properties to be shown must be requested at source (eg. `i-button` in state tables). Only available when game is run within Blender - not in standalone player version.

**Framerate and Profile** Show framerate and profiling information while game runs. Only available when game is run within Blender - not in standalone player version.

**Physics Visualization** Show physics bounds and interactions while game runs (available in both Blender and standalone versions).

**Deprecation Warnings** Print warnings when using deprecated features in the python API. Only available when game is run within Blender - not in standalone player version.

**Mouse Cursor** Show mouse cursor while game runs (available in both Blender and standalone versions).

## Stereo Camera

Stereo Cameras allow you to generate images that appear three dimensional when wearing special glasses. This is achieved by rendering two separate images from cameras that are a small distance apart from each other, simulating how our own eyes see. When viewing a stereo image, one eye is limited to seeing one of the images, and the other eye sees the second image. Our brain is able to merge these together, making it appear that we are looking at a 3d object rather than a flat image. See [Stereoscopy](#) for more information on different stereoscopic viewing methods.

## Stereo Settings

*Stereo Mode*



Fig. 2.2672: Set the type of stereo camera to use. Possible modes are detailed below.

**Eye Separation** This value is extremely important. It determines how far apart the two image-capturing cameras are, and thus how “deep” the scene appears. Too small a value and the image appears flat; too high a value can result in headaches and eye strain. The ideal value mimics the separation of the viewer’s two eyes.

### Stereo Modes

Specifies the way in which the left-eye image and the right-eye image pixels are put together during rendering. This must be selected according to the type of apparatus available to display the appropriate images to the viewer’s eyes.

**Anaglyph** One frame is displayed with both images color encoded with red-blue filters. This mode only requires [glasses with color filters](#), there are no special requirements for the display screen and GPU.

**Quad Buffer** Uses double buffering with a buffer for each eye, totaling four buffers (Left Front, Left Back, Right Front and Right Back), allowing to swap the buffers for both eyes in sync. See [Quad Buffering](#) for more information.

**Side by Side** Lines are displayed one after the other, so providing the two images in two frames side by side.

**Above-Below** Frames are displayed one after the other, so providing the two images in two frames, one above the other.

**Interlaced** One frame is displayed with the two images on alternate lines of the display.

**Vinterlaced** One frame is displayed with both images displayed on alternate columns of the display. This works with some ‘autostereo displays’.

**3D Tv Top-Bottom** One frame displays the left image above and the right image below. The images are squashed vertically to fit. This mode is designed for passive 3D TV.

### Dome Camera

This feature allows artists to visualize their interactive projects within an immersive dome environment. In order to make it an extensible tool, we are supporting Fulldome, Truncated domes (front and rear), Planetariums and domes with spherical mirrors.

The Dome camera uses a multipass texture algorithm as developed by Paul Bourke and was implemented by Dalai Felinto with sponsorship from **SAT** - Society for Arts and Technology within the **SAT Metalab immersion research program**, that involves rendering the scene 4 times and placing the subsequent images onto a mesh designed especially such that the result, when viewed with an orthographic camera, is a fisheye projection.

---

**Note:** Remember to use Blender in **fullscreen mode** to get the maximum out of your projector.

To accomplish that launch Blender with the command-line argument `-W`. Also to get away of the top menu on Blender try to join all windows (buttons, 3dview, text, ...) in a single one. Otherwise if you only maximize it (Ctrl+Up) you can’t get the whole screen free to run your game (the top bar menu takes about 20 pixels).

---

### Dome Camera Settings

**Dome Type** This menu allows you to select which type of dome camera to use. They are outlined below, along with their respective settings.



- *Fisheye Mode*
- *Front-Truncated Dome Mode*
- *Rear-Truncated Dome Mode*
- *Cube Map Mode*
- *Spherical Panoramic Mode*

Available camera settings change depending on the selected Dome Type:

**Resolution** Sets the resolution of the Buffer. Decreasing this value increases speed, but decreases quality.

**Tessellation** 4 is the default. This is the tessellation level of the mesh. (Not available in Cube Map mode).

**Angle** Sets the field of view of the dome in degrees, from 90 to 250. (Available in Fisheye and Truncated modes).

**Tilt** Set the camera rotation in the horizontal axis. Available in Fisheye and Truncated modes).

**Warp Data Mesh** Use a custom warp mesh data file.

**Fisheye Mode** An Orthogonal Fisheye view from 90° to 250° degrees.

- From 90° to 180° we are using 4 renders.
- From 181° to 250° we are using 5 renders.

**Front-Truncated Dome Mode** Designed for truncated domes, this mode aligns the fisheye image with the top of the window while touching the sides.

- The Field of view goes from 90° to 250° degrees.
- From 90° to 180° we are using 4 renders.
- From 181° to 250° we are using 5 renders.

**Rear-Truncated Dome Mode** Designed for truncated domes, this mode aligns the fisheye image with the bottom of the window while touching the sides.

- The Field of view goes from 90° to 250° degrees.
- From 90° to 180° we are using 4 renders.
- From 181° to 250° we are using 5 renders.

**Cube Map Mode** Cube Map mode can be used for pre-generate animated images for CubeMaps.

- We are using 6 renders for that. The order of the images follows Blender internal EnvMap file format: - first line: right, back, left - second line: bottom, top, front



Fig. 2.2673: Fisheye Mode

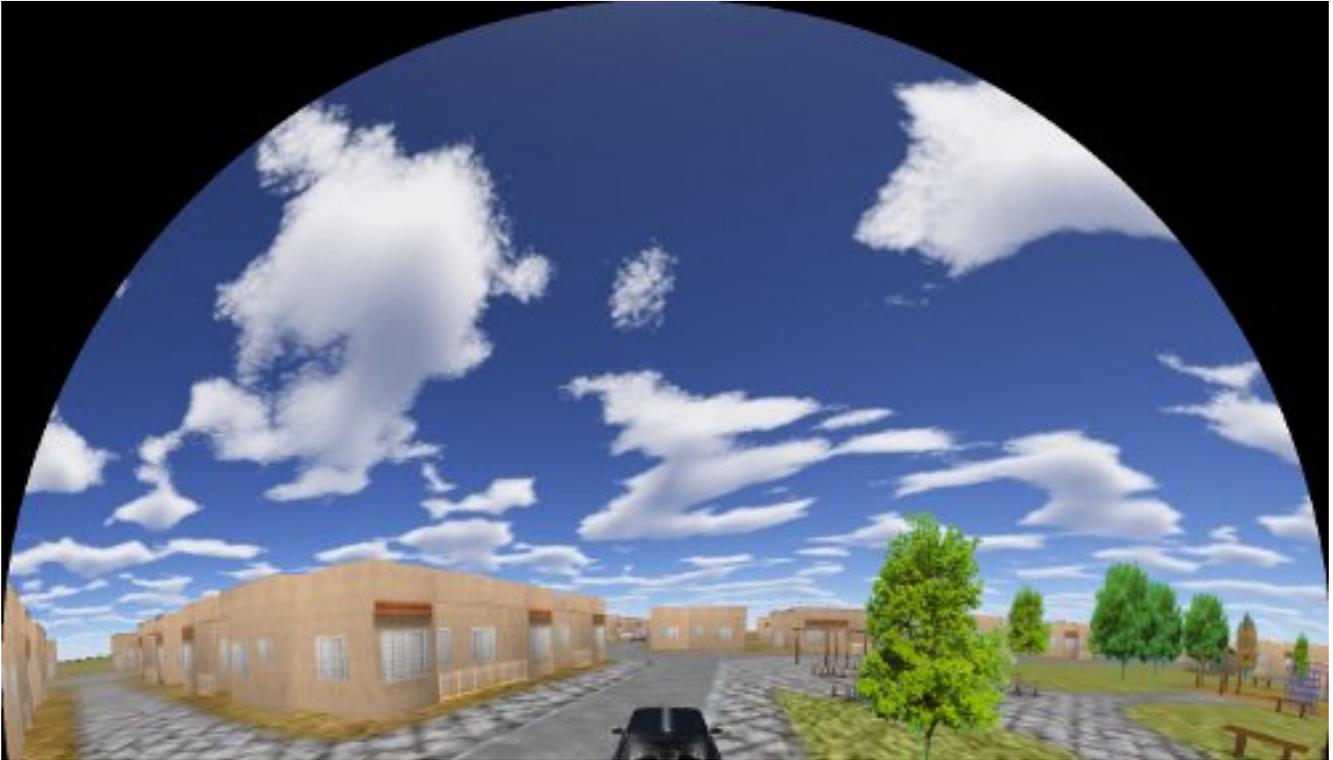


Fig. 2.2674: Front Truncated Dome Mode

**Spherical Panoramic Mode** A full spherical panoramic mode.

- We are using 6 cameras here.
- The bottom and top start to get precision with **Definition** set to 5 or more.

**Warp Data Mesh** Many projection environments require images that are not simple perspective projections that are the norm for flat screen displays. Examples include geometry correction for cylindrical displays and some new methods of projecting into planetarium domes or upright domes intended for VR.

For more information on the mesh format see [Paul Bourke's article](#).

In order to produce that images, we are using a specific file format.

File template:

```
mode
width height
n0_x n0_y n0_u n0_v n0_i
n1_x n1_y n1_u n1_v n1_i
n2_x n1_y n2_u n2_v n2_i
n3_x n3_y n3_u n3_v n3_i
(...)
```

First line is the image type the mesh is support to be applied to: **2 = rectangular**, **1 = radial** Next line has the mesh dimensions in pixels Rest of the lines are the nodes of the mesh.

Each line is compund of **x y u v i** (x,y) are the normalised screen coordinates (u,v) texture coordinates i a multiplicative intensity factor

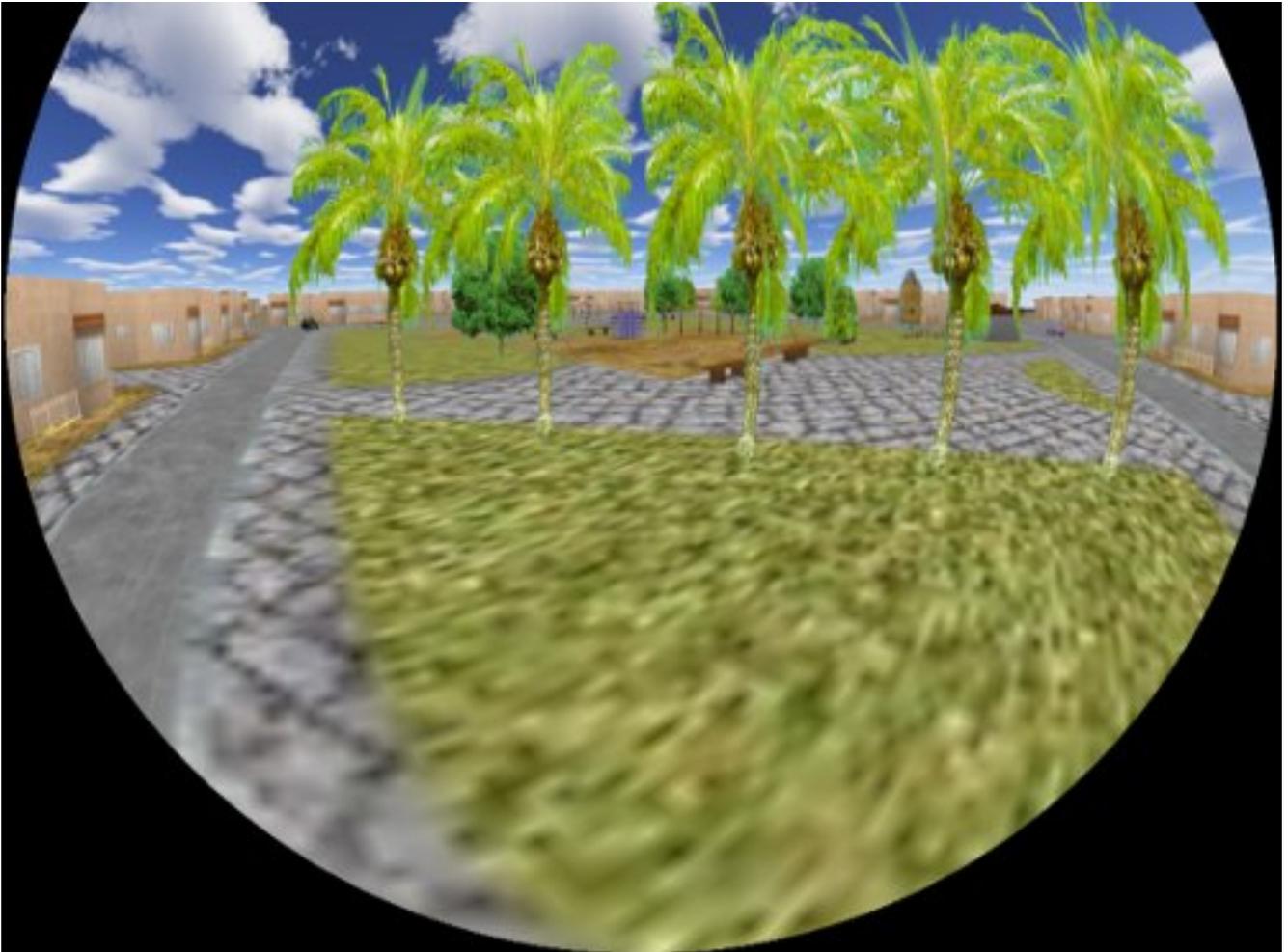


Fig. 2.2675: Rear Truncated Dome Mode

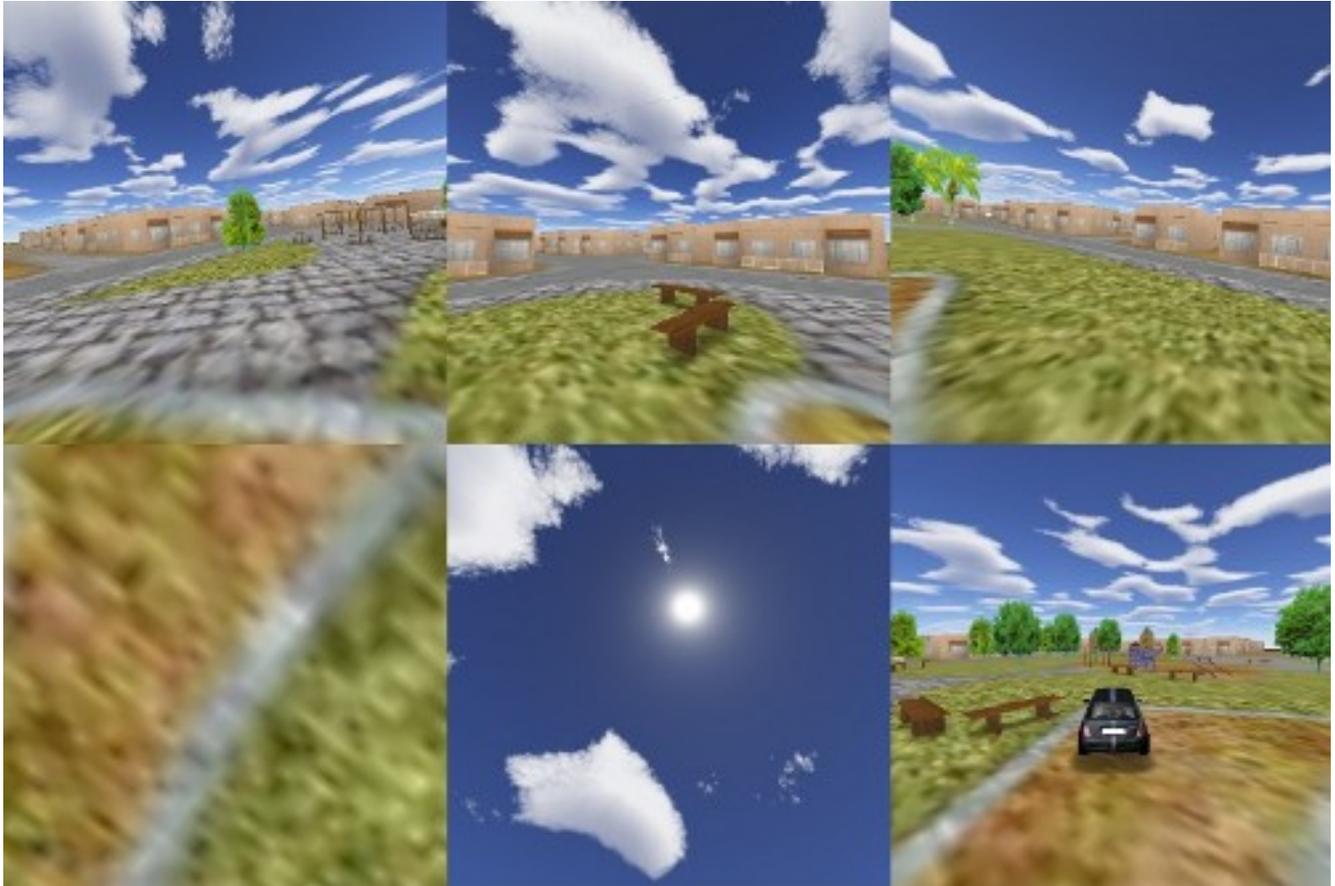


Fig. 2.2676: Environment Map Mode



Fig. 2.2677: Full Spherical Panoramic Mode



x varies from -screen aspect to screen aspecty varies from -1 to 1u and v vary from 0 to 1i ranges from 0 to 1, if negative don't draw that mesh node

- You need to create the file and add it to the Text Editor in order to select it as your Warp Mesh data file.
- Open the Text Editor (Window Types/Text Editor).
- Open your mesh data file(ie. myDome.data) in the text editor (Text/Open or Alt O on keyboard).
- Go to Game Framing Settings (Window Types/Buttons Window/Scene)
- Enable Dome Mode.
- Type filename in Warp Data field(ie. myDome.data).

To create your own Warp Meshes an interactive tool called meshmapper is available as part of [Paul Bourke's Warpplayer](#) software package(requires full version).

**Example files** [Spherical Mirror Dome 4x3](#), [Truncated Dome 4x3](#), [Sample Fullscreen File 4x3](#), [Sample Fullbuffer File 4x3](#).

**Note:** Important: the viewport is calculated using the ratio of canvas width by canvas height. Therefore different screen sizes will require different warp mesh files. Also in order to get the correct ratio of your projector you need to use Blender in Fullscreen mode.

## 2.12.5 Physics

### Blender Game Physics

Blender includes advanced physics simulation in the form of the Bullet Physics Engine ([BulletPhysics.org](#)). Most of your work will involve setting the right properties on the objects in your scene, then you can sit back and let the engine take over. The physics simulation can be used for Games, but also for Animation.

The Blender Game Engine (BGE) is based on Rigid-Body Physics, which differs significantly from the complementary set of tools available in the form of Soft Body Physics Simulations. Though the BGE does have a Soft Body type, it is not nearly as nuanced as the non-BGE Soft Body. The inverse is even more true: it is difficult to get the non-BGE physics to resemble anything like a stiff shape. Rigid Body Physics does not have, as an effect or a cause, any mesh deformations. For a discussion on how to partially overcome this, see: [Mesh Deformations](#).

### Global Options

The global Physics Engine settings can be found in the `World Properties`, which include the Gravity constant and some important engine performance tweaks.

### Object Physics



### Physics Type

**No Collision** Is not affected by the simulation nor affects other objects.

**Static** Participates in the simulation, affecting other objects, but is not affected by it.

**Dynamic** Object that can move besides colliding and being collided with.

**Rigid Body** Has rigid body dynamics.

**Soft Body** Soft body dynamics.

**Character Controller** Character controller.

**Vehicle Controller** Vehicle controller.

**Occluder** Prevents calculation of rendered objects (not their physics, though!).

**Sensor** Detects presence without restituting collisions.

**Navigation Mesh** To make pathfinding paths. Useful for Artificial Intelligence.

## Material Physics

Physics can be associated with a material on the material properties tab. These are settings that one would normally associate with a material, such as its friction and they are meant to be used in conjunction with the object physics settings, not replace it.

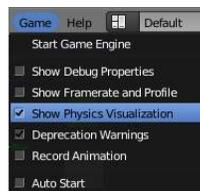
## Constraints

It is imperative to understand that the Blender Constraints generally don't work inside the BGE. This means interesting effects such as *Copy Rotation* are unavailable directly.

Your options include:

- **Parenting** - But not Vertex Parenting.
- **Rigid Body Joint** - This is the one Constraint that you can set up through the UI that works in the BGE. It has several options, and can be very powerful - see ITS page for a detailed description and demo .blend. Don't forget that you can loop through objects using `bpy` instead of clicking thousands of times to set up chains of these Constraints.
- **Rigid Body Joints on the Fly** - You can add/remove them after the BGE starts by using `bge.constraints.createConstraint()`. This can be good either to simply automate their setup, or to truly make them dynamic. A simple demo can be viewed in: [BGE-Physics-DynamicallyCreateConstraint.blend](#)
- **Python Controllers** - As always, in the BGE, you can get the most power when you drop into Python and start toying with the settings directly. For instance, the *Copy Rotation* mentioned above is not hard - All you have to do is something to the effect of `own.worldOrientation = bge.logic.getCurrentScene().objects['TheTargetObject'].worldOrientation`

## Visualizing Physics



Go to *Game* → *Show Physics Visualization* to show lines representing various attributes of the Bullet representation of your objects. Note that these might be easier to see when you turn on Wireframe Mode (Z) before you press P. Also note that you can see how the Bullet triangulation is working (it busts all your Quads to Tris at run-time, but the BGE meshes are still quads at run-time).

- **RGB/XYZ Widget** - Representing the object's Local Orientation and Origin.
- **Green** - “sleeping meshes” that are not moving, saving calculations until an external event “wakes” it.
- **White** - White lines represent active bounding meshes at are undergoing physics calculations, untill such calculations are so small that the object is put to rest. This is how you can see the effects of the *Collision Bounds*. - **Thick, or Many White Lines** - A compound collision mesh/meshes.
- **Violet** - Bounding meshes for Soft bodies.
- **Red** - The Bounding Box, the outer boundary of object. It is always aligned with global X Y and Z, and is used to optimize calculations. Also represents meshes that have been forced into “no sleep” status.
- **Yellow** - Normals.
- **Black** - When in wireframe, this is your mesh's visual appearance.

If you want finer-grained control over the display options, you can add this as a Python Controller and uncomment whichever pieces you want to see:

```
import bge
debugs = (
    bge.constraints.DBG_DRAWAABB,
)
for d in debugs:
    bge.constraints.setDebugMode(d)
```

For all debug modes, API docs for `bge.constraints`.

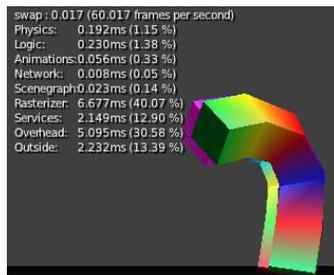


Fig. 2.2678: A shot of `Manual-BGE-Physics-DancingSticks.blend` with [Game → Show Framerate and Profile] enabled

**Show Framerate and Profile** If you enable *Game* → *Show Framerate and Profile*, it will put some statistics in the upper-left area of the game window.

These can be very informative, but also a bit cryptic. Moguri has elaborated on their meanings, for us: <http://mogurijin.wordpress.com/2012/01/03/bge-profile-stats-and-what-they-mean/>

## Mesh Deformations

As mentioned above, Rigid Body physics do not affect mesh deformations, nor do they account for them in the physics model. This leaves you with a few options:

**Soft Bodies** You can try using a *Soft Body*, but these are fairly hard to configure well.

**Actions** To use an [Action Actuator](#) to do the deformation, you have to make a choice. If you use Shapekeys in the Action, you will be fine as far as the overall collisions (but see below for the note on `reinstancePhysicsMesh()`). The mesh itself is both a display and a physics mesh, so there is not much to configure.

To use an Armature as the deformer will require a bit of extra thought and effort. Basically the Armature will only deform a mesh if the Armature is the parent of that mesh. But at that point, your mesh will lose its physics responsiveness, and only hang in the air (it's copying the location/rotation of the Armature). To somewhat fix this you can then parent the Armature to a collision mesh (perhaps a simple box or otherwise very-low-poly mesh). This "Deformation Mesh" will be the physics representative, being type: Dynamic or Rigid Body, but it will be set to Invisible. Then "Display Mesh" will be the opposite set to type: No Collision, but visible. This still leaves us with the problem mentioned in the previous paragraph.

When you deform a display mesh, it does not update the corresponding physics mesh. You can view this evidently when you enable physics visualization (*Visualizing Physics*) - the collision bounds will remain exactly as when they began. To fix this, you must call `own.reinstancePhysicsMesh()` in some form. Currently this only works on *Triangle Mesh* bounds, not *Convex Hull*. We have prepared a demonstration file in [Manual-BGE-Physics-DancingSticks.blend](#). Note that we had to increase the *World* → *Physics* → *Physics Steps* → *Substeps* to make the collisions work well. The more basic case is the case the Shapekeyed Action, which you can see in the back area of the scene. Since it is the only object involved, you can call `reinstancePhysicsMesh()` unadorned, and it will do the right thing.

The more complicated case is the *Collision Mesh* → *Armature* → *Display Mesh* cluster, which you can see in the front of the scene. What it does in the `.blend` is call `reinstancePhysicsMesh(viz)`, that is, passing in a reference to the visual mesh. If we tried to establish this relationship without the use of Python, we would find that Blender's dependency check system would reject it as a cyclic setup. This is an example of where Blender's checking is too coarsely-grained, as this circle is perfectly valid: the grandparent object (the Collision Mesh) controls the location/rotation, while the middle object (the Armature) receives the animated Action, where the child (the Display Mesh) receives the deformation, and passes that on up to the top, harmlessly. Something to note is that the Collision Mesh is merely a plane - that is all it requires for this, since it will be getting the mesh data from `viz`.

**Ragdolls** A third option is to create your items out of many sub-objects, connected together with Rigid Body Joints or similar. This can be quite a bit more work, but the results can be much more like a realistic response to collisions. For an Add-on that can help you out in the process, check out the [Blender Ragdoll Implementation Kit](#).

## Digging Deeper

Sometimes you will want to look at:

- The main Bullet Physics page - <http://bulletphysics.org/wordpress/>
- The Bullet Wiki - <http://www.bulletphysics.org/mediawiki-1.5.8/index.php?title=Documentation>
- The Bullet API Docs - <http://www.continuousphysics.com/Bullet/BulletFull/index.html>
- The Bullet Forums - <http://www.bulletphysics.org/Bullet/phpBB3/>

## Recording to Keyframes

Beyond gaming, sometimes you wish to render a complex scene that involves collisions, multiple forces, friction between multiple bodies, and air drag or even a simple setup that is just easier to achieve using the realtime physics.

Blender provides a way to "bake" or "record" a physics simulation into keyframes allowing it then to be played as an action either for animation or games. Keep in mind that the result of this method is a recording, no longer a simulation. This means that the result is completely deterministic (the same everytime it is run) and unable to interact with new objects that are added to the physics simulation after it was recorded. This may, or not, be desired according to the situation.

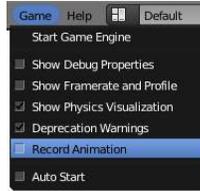


Fig. 2.2679: Menu to record Keyframes to the Dopesheet.

All you have to do to achieve this effect is go to the Info Editor (the bar at the top of the window) *Game* → *Record Animation*, and it will lock away your keyframes for use in *Blender Render* mode. You can go back to the 3D view and press **Alt-A** to play it back, or **Ctrl-F12** to render it out as an animation.

Note that you can also use Game Logic Bricks and scripting. Everything will be recorded.



Fig. 2.2680: Resulting recorded animation

**Keyframe Clean-up** *Record Animation* keys redundant data (data that was did not change relative to the last frame). Pressing **O** while in the *DopeSheet* will remove all superfluous keyframes. Unwanted channels can also be removed.

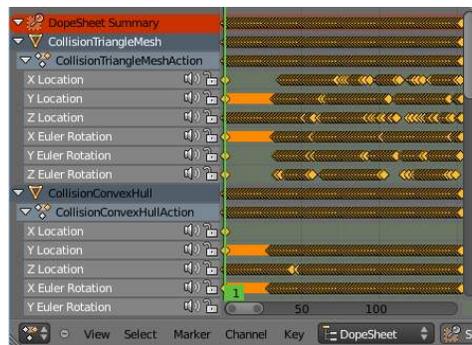


Fig. 2.2681: Cleaned up recording

## Exporting

**.bullet / Bullet compatible engines** You can snapshot the physics world at any time with the following code:

```
import bge
bge.constraints.exportBulletFile("test.bullet")
```

This will allow importing into other Bullet-based projects. See the [Bullet Wiki on Serialization](#) for more.

## World Physics

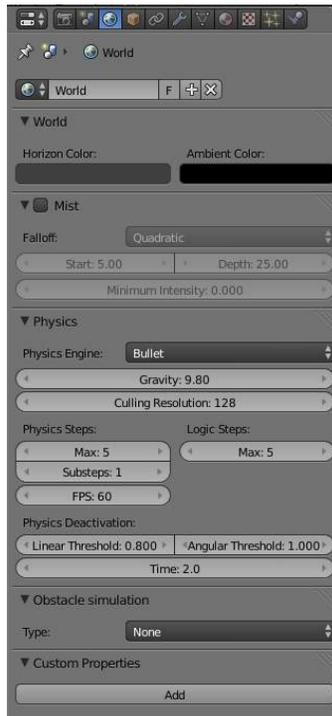


Fig. 2.2682: BGE World Panel (fully expanded)

World settings enable you to set some basic effects which affect all scenes throughout your game, so giving it a feeling of unity and continuity. These include ambient light, depth effects (mist) and global physics settings. These effects are a limited subset of the more extensive range of effects available with the Blender internal or cycles renderer.

**Tip:** While world settings offer a simple way of adding effects to a scene, [compositing nodes](#) are often preferred, though more complex to master, for the additional control and options they offer. For example, filtering the Z value (distance from camera) or normals (direction of surfaces) through compositing nodes can further increase the depth and spacial clarity of a scene.

## World

These two color settings allow you to set some general lighting effects for your game.

**Horizon Color** The RGB color at the horizon; i.e. the color and intensity of any areas in the scene which are not filled explicitly.

**Ambient Color** Ambient light mimics an overall background illumination obtained from diffusing surfaces (see [Ambient Light](#), [Exposure](#) and [Ambient Occlusion](#)). Its general color and intensity are set by these controls.

**Mist** Mist can greatly enhance the illusion of depth in your rendering. To create mist, Blender makes objects farther away more transparent (decreasing their Alpha value) so that they mix more of the background color with the object color. With Mist enabled, the further the object is away from the camera the less its alpha value will be. For full details, see [Mist](#).

**Mist** Toggles mist on and off

**Falloff** Sets the shape of the falloff of the mist.

**Start** The starting distance of the mist effect. No misting will take place for objects closer than this distance.

**Depth** The depth at which the opacity of objects falls to zero.

**Minimum intensity** Overall minimum intensity of the mist

## Game Physics

The Game Physics located in the World panel determine the type of physical rules that govern the game engine scene, and the gravity value to be used. Based on the physics engine selected, in physics simulations in the game engine, Blender will automatically move *Actors* in the downward (-Z) direction. After you arrange the actors and they move as you wish, you can then bake this computed motion into keyframes (see [Digging Deeper](#) for more info).

**Physics Engine** Set the type of physics engine to use.

**Bullet** The default physics engine, in active development. It handles movement and collision detection. The things that collide transfer momentum to the collided object.

**None** No physics in use. Things are not affected by gravity and can fly about in a virtual space. Objects in motion stay in that motion.

**Gravity** The gravitational acceleration, in units of meters per squared second ( $m \cdot s^{-2}$ ), of this world. Each object that is an actor has a mass and size slider. In conjunction with the frame rate (see [Render](#) section), Blender uses this info to calculate how fast the object should accelerate downward.

**Culling Resolution** The size of the occlusion culling buffer in pixel, use higher value for better precision (slower). The optimized Bullet DBVT for view frustum and occlusion culling is activated internally by default.

### Physics Steps

**Max** Sets the maximum number of physics steps per game frame if graphics slow down the game. higher value allows physics to keep up with realtime.

**Substeps** Sets the number of simulation substeps per physics timestep. Higher value give better physics precision.

**FPS** Set the nominal number of game frames per second. Physics fixed timestep =  $1/fps$ , independently of actual frame rate.

**Logic Steps** Sets the maximum number of logic frame per game frame if graphics slows down the game, higher value allows better synchronization with physics.

**Physics Deactivation** These settings control the threshold at which physics is deactivated. These settings help reducing the processing spent on Physics simulation during the game.

**Linear Threshold** The speed limit under which a rigid bodies will go to sleep (stop moving) if it stays below the limits for a time equal or longer than the deactivation time (sleeping is disabled when deactivation time is set to 0).

**Angular Threshold** Same as linear threshold, but for rotation limit (in rad/s)

**Time** The amount of time in which the object must have motion below the thresholds for physics to be disabled (0.0 disables physics deactivation).

## Obstacle Simulation

Simulation used for obstacle avoidance in the Game Engine, based on the RVO (Reciprocal Velocity Obstacles) principle. The aim is to prevent one or more actors colliding with obstacles.

See [Path finding and steering behaviors](#) for more details.

### Type

**None** obstacle simulation is disabled, actors aren't able to avoid obstacles

**RVO (cells)** obstacle simulation is based on the [RVO method](#) with cell sampling.

**RVO (rays)** obstacle simulation is based on the [RVO method](#) with ray sampling

**Level height** Max difference in heights of obstacles to enable their interaction. Used to define minimum margin between obstacles by height, when they are treated as those which are situated one above the other i.e. they doesn't influence to each other.

**Visualization** Enable debug visualization for obstacle simulation.

## Material

### Static Physics Object Type

Static objects in the [Blender Game Engine](#) do not automatically react to physics, including gravity and collisions. Even if hit by the force of a speeding 18-wheeler truck, it will remain unresponsive in terms of location, rotation, or deformation.

It will, however, give collision reactions. Objects will bounce off of Static Objects, and rotational inertia will transfer to objects capable of rotating (that is, Rigid Body Objects will spin in response, though Dynamic Objects will not).

Note that none of this prevents you from transforming the Static Objects with [Logic Bricks](#) or Python code. The visual objects will correctly move and their physics representation will update in the engine as well.

Another important note is that the default [Collision Bounds](#) is a Triangle Mesh, meaning it is higher in computational requirements but also in detail. This in turn means the "Radius" option has no effect by default.

In the example game demo, [Frijoles](#), the Static type is represented by the "Arena" object that holds all the moving bits.

For more documentation, see the [Top BGE Physics page](#).

### Options

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/PythonAccessToOptions;{{Doc:2.6/Manual/Game\_Engine/Physics

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/CommonOptions;{{Doc:2.6/Manual/Game\_Engine/Physics/Comm

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/RadiusOption;{{Doc:2.6/Manual/Game\_Engine/Physics/RadiusOp

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/AnisotropicFrictionOptions;{{Doc:2.6/Manual/Game\_Engine/Phys

### Collision Bounds

---

**Note:** The Static type differs from the others in that it defaults to a Triangle Mesh bounds, instead of a simple sphere.

---

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/Collision\_Bounds;{{Doc:2.6/Manual/Game\_Engine/Physics/Collis

## Create Obstacle

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/Create\_Obstacle;{{ Doc:2.6/Manual/Game\_Engine/Physics/Create\_

## All Types

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/AllTypes;{{ Doc:2.6/Manual/Game\_Engine/Physics/AllTypes}})

## No Collision Physics Object Type

“No Collision” objects in the [Game Engine](#) are completely unaffected by [Physics](#), and do cause physics reactions. They are useful as pure display objects, such as the child of a *Custom Collision Hull* (*Collision Bounds*).

In the example game demo, *Frijoles*, the No Collision type is represented by the “HUD” objects that display the health status.

For more documentation, see the [Top BGE Physics page](#).

## Options

The only option available on No Collision types is:

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/InvisibleOption; {{ Doc:2.6/Manual/Game\_Engine/Physics/InvisibleOption}} )

## All Types

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/AllTypes; {{ Doc:2.6/Manual/Game\_Engine/Physics/AllTypes}} )

## Dynamic Physics Object Type

Dynamic objects in the [Game Engine](#) give/receive collisions, but when they do so they themselves do not rotate in response. So, a Dynamic ball will hit a ramp and slide down, while a Rigid Body ball would begin rotating.

If you do not need the rotational response the Dynamic type can save the extra computation.

Note that these objects can still be rotated with [Logic Bricks](#) or Python code. Their physics meshes will update when you do these rotations - so collisions will be based on the new orientations.

In the example game demo, *Frijoles*, the Dynamic type is represented by the titular jumping beans. Though we want these characters to recoil back when they hit a Boulder or each other, having them torque in response to these collisions would result in their being impossible to control.

For more documentation, see the [Top BGE Physics page](#).

## Options

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/PythonAccessToOptions; {{ Doc:2.6/Manual/Game\_Engine/Physics/PythonAccessToOptions}} )

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/CommonOptions; {{ Doc:2.6/Manual/Game\_Engine/Physics/CommonOptions}} )

```
FIXME(Template      Unsupported:      Doc:2.6/Manual/Game_Engine/Physics/RightColumnOfOptions;
{{Doc:2.6/Manual/Game_Engine/Physics/RightColumnOfOptions}} )
```

```
FIXME(Template Unsupported: Doc:2.6/Manual/Game_Engine/Physics/MassOption; {{Doc:2.6/Manual/Game_Engine/Physics/MassOption}} )
```

```
FIXME(Template Unsupported: Doc:2.6/Manual/Game_Engine/Physics/RadiusOption; {{Doc:2.6/Manual/Game_Engine/Physics/RadiusOption}} )
```

```
FIXME(Template      Unsupported:      Doc:2.6/Manual/Game_Engine/Physics/FormFactorOption;
{{Doc:2.6/Manual/Game_Engine/Physics/FormFactorOption}} )
```

```
FIXME(Template      Unsupported:      Doc:2.6/Manual/Game_Engine/Physics/AnisotropicFrictionOptions;
{{Doc:2.6/Manual/Game_Engine/Physics/AnisotropicFrictionOptions}} )
```

```
FIXME(Template      Unsupported:      Doc:2.6/Manual/Game_Engine/Physics/TransformOptions;
{{Doc:2.6/Manual/Game_Engine/Physics/TransformOptions}} )
```

### Collision Bounds

```
FIXME(Template      Unsupported:      Doc:2.6/Manual/Game_Engine/Physics/Collision_Bounds;
{{Doc:2.6/Manual/Game_Engine/Physics/Collision_Bounds}} )
```

### Create Obstacle

```
FIXME(Template      Unsupported:      Doc:2.6/Manual/Game_Engine/Physics/Create_Obstacle;
{{Doc:2.6/Manual/Game_Engine/Physics/Create_Obstacle}} )
```

### All Types

```
FIXME(Template Unsupported: Doc:2.6/Manual/Game_Engine/Physics/AllTypes; {{Doc:2.6/Manual/Game_Engine/Physics/AllTypes}} )
```

## Rigid Body Physics Object Type

Probably the most common type of object in the [Game Engine](#). It will give/receive collisions and react with a change in its velocity and its rotation. A Rigid Body ball would begin rotating and roll down (where a [Dynamic](#) ball would only hit and slide down the ramp).

The idea behind Rigid Body dynamics is that the mesh does not deform. If you need deformation you will need to either go to [Soft Body](#) or else fake it with animated Actions.

In the example game demo, [Frijoles](#), the Rigid Body type is represented by the Boulders that spawn from the top of the level. Notice how they tumble and roll in response to the collisions with the Arena.

For more documentation, see the [Top BGE Physics page](#).

### Options

```
FIXME(Template      Unsupported:      Doc:2.6/Manual/Game_Engine/Physics/PythonAccessToOptions;
{{Doc:2.6/Manual/Game_Engine/Physics/PythonAccessToOptions}} )
```

```
FIXME(Template      Unsupported:      Doc:2.6/Manual/Game_Engine/Physics/CommonOptions;
{{Doc:2.6/Manual/Game_Engine/Physics/CommonOptions}} )
```

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/MassOption; {{Doc:2.6/Manual/Game\_Engine/Physics/MassOption}} )

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/RadiusOption; {{Doc:2.6/Manual/Game\_Engine/Physics/RadiusOption}} )

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/FormFactorOption; {{Doc:2.6/Manual/Game\_Engine/Physics/FormFactorOption}} )

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/AnisotropicFrictionOptions; {{Doc:2.6/Manual/Game\_Engine/Physics/AnisotropicFrictionOptions}} )

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/TransformOptions; {{Doc:2.6/Manual/Game\_Engine/Physics/TransformOptions}} )

### Collision Bounds

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/Collision\_Bounds; {{Doc:2.6/Manual/Game\_Engine/Physics/Collision\_Bounds}} )

### Create Obstacle

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/Create\_Obstacle; {{Doc:2.6/Manual/Game\_Engine/Physics/Create\_Obstacle}} )

### All Types

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/AllTypes; {{Doc:2.6/Manual/Game\_Engine/Physics/AllTypes}} )

## Soft Body Physics Object Type

The most advanced type of object in the [Game Engine](#). Also, it is the most finicky. If you are used to the fun experimentation that comes from playing around with the non-BGE Soft Body sims (such as Cloth), you will probably find a frustrating lack of options and exciting results. Do not despair, we are here to help you get some reasonable settings.

Your setup will involve making sure you have sufficient geometry in the Soft Body's mesh to support the deformation, as well as tweaking the options.

In the example game demo, [Frijoles](#), the Soft Body type is represented by the decorative checkered flag at the top of the level.

For more documentation, see the [Top BGE Physics page](#).

### Options

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/CommonOptions; {{Doc:2.6/Manual/Game\_Engine/Physics/CommonOptions}} )

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/MassOption; {{Doc:2.6/Manual/Game\_Engine/Physics/MassOption}} )

- *Shape Match* - Upon starting the Game Engine this will record the starting shape of the mesh as the “lowest energy” state. This means that the edges will have tension whenever they are flexed to some other form. This is set to on by default, and in this configuration turns the object into more of a thin sheet of metal rather than a cloth.
  - Demo: [BGE-Physics-Objects-SoftBodies\\_ShapeMatchAndLinearStiffness.blend](#)
  - Default: On.
  - Code effect: When on, it will call `btSoftBody::setPose(false,true)`
  - Python property: `obj.game.soft_body.use_shape_match`
  - Suboption: *Threshold* - Linearly scales the pose match
    - \* A threshold of 1.0 makes it behave like *Shape Match* on with a *Linear Stiffness* of 1.0.
    - \* A threshold of 0.0 makes it behave like *Shape Match* off with a *Linear Stiffness* of 0.0.
    - \* Range: 0-1.
    - \* Default: 0.5.
    - \* Code effect: Sets `btSoftBody::Config::kMT`.
    - \* Python property: `obj.game.soft_body.shape_threshold`
- *Welding* TODO.
- *Position Iteration* - Increase the accuracy at a linearly-increasing expense of time. The effect is visible especially with Soft Bodies that fall on sharp corners, though this can slow down even very simple scenes.
  - Demo: A situation where only the max setting of 10 works correctly: [BGE-Physics-Objects-SoftBodies\\_PositionIterations.blend](#).
  - Range: 0-10.
  - Default: 2.
  - Code effect: Represents the number of times `this loop` is run.
  - Python property: `obj.game.soft_body.location_iterations`
- *Linear Stiffness* - Linear stiffness of the soft body links. This is most evident when you have *Shape Match* off, but it is also evident with it on.
  - Demo: [BGE-Physics-Objects-SoftBodies\\_ShapeMatchAndLinearStiffness.blend](#)
  - Range: 0-1.
  - Default: 0.5.
  - Python property: `obj.game.soft_body.linear_stiffness`
- *Friction* - Dynamic friction coefficient. .. TODO: Learn/demo/explain.
  - Code effect: Sets `btSoftBody::Config::kMT`, which, for Soft Bodies, defines the minimum friction versus the Material Friction (which in turn defaults to 0.5).
  - Range: 0-1.
  - Default: 0.2.
  - Python property: `obj.game.soft_body.dynamic_friction`
- *Margin* - Small value makes the algorithm unstable. .. TODO: Learn/demo/explain.
  - Range: 0.01-1.
  - Default: 0.01.

- Python property: `obj.game.soft_body.collision_margin`
- *Bending Constraint* - Enable Bending Constraints .. TODO: Learn/demo/explain.
  - Default: On.
  - Python property: `obj.game.soft_body.use_bending_constraints`
- *Cluster Collision* - Affects Collision sensors as well as physics.
  - Demo: [BGE-Physics-Objects-SoftBodies\\_ClusterRigidToSoftBody.blend](#) for a demonstration of the effect on the [Collision Sensor](#). There you will observe the “Rigid to Soft Body” off, then on with Iterations of 1, 64, and 128. The Off and Iterations: 1 cases do not register collisions, and the other two do (though they send their poor Cubes flying into space).
  - Demo of badness: [Manual-BGE-Physics-SoftBody\\_BadClusterCollisions.blend](#) - four different ways of making misconfigured Soft Body objects.
  - Suboption: *Rigid to Soft Body* - Enable cluster collisions between Rigid and Soft Bodies.
    - \* Default: Off.
    - \* Python property: `obj.game.soft_body.use_cluster_rigid_to_softbody`
  - Suboption: *Soft to Soft Body* - Enable cluster collisions among Soft Bodies.
    - \* Default: Off.
    - \* Python property: `obj.game.soft_body.use_cluster_soft_to_softbody`
  - Suboption: *Iterations* - Number of cluster iterations.
    - \* Range: 1-128.
    - \* Default: 64.
    - \* Python property: `obj.game.soft_body.cluster_iterations`

## Hints

- A very important configurable in the case of Soft Body interactions is `World properties` → *Physics* → *Physics Steps* → *Substeps*. In the test `.blend` here: [Manual-BGE-Physics-SoftBody\\_PhysicsSteps.blend](#), you can see the behavior at various Substep levels: - The default level. The Grid object goes straight through the cube, hardly slowing down at all. - The Grid slows upon hitting the Cube’s top face, and stops fully on the bottom face. - The Grid stops at the top face, but two opposite Cube corners are visible. - ...no perceptible difference. - Finally a working sim. This is good, because it is the maximum step level.
- Surprisingly, the more vertices you have in your hit object, the less likely the Soft Body is to react with it. If you try letting it hit a Plane, it might stop, but a subdivided Grid might fail.

## Sensors

Soft bodies do not work with the Collision, Touch, Near, and Radar logic brick sensors.

## Goal Weights

## Force Fields

A common practice within the non-BGE Cloth simulator is to employ [Force Fields](#) to animate the cloth.

These do not work in the BGE, so you will have to figure out a way to use Python (or perhaps plain Logic Bricks) to apply forces to the Soft Body objects.

## All Types

FIXME(Template Unsupported: Doc:2.6/Manual/Game\_Engine/Physics/AllTypes; {{Doc:2.6/Manual/Game\_Engine/Physics/AllTypes}} )

## Vehicle Controller

### Introduction

The Vehicle Controller is a special *type of physics object* that the Physics Engine (bullet) recognizes.

It is composed of a **rigid body** representing the chassis and a set of wheels that are set to **no collision**. Emphasizing the distinction between a GameEngine, Logical or Render object and its representation for the Physics Engine is important.

To simulate a vehicle as a true rigid body, on top of also rigid body wheels, with a real suspension system made with joints, would be far too complicated and unstable. Cars and other vehicles are complicated mechanical devices and most often we do not want to simulate that, only that it ‘acts as expected’. The Vehicle Controller exists to provide a dedicated way of simulating a vehicle behavior without having to simulate all the physics that would actually happen in the real world. It abstracts the complexity away by providing a simple interface with tweakable parameters such as suspension force, damping and compression.

### How it works

Bullet’s approach to a vehicle controller is called a “Raycast Vehicle”. Collision detection for the wheels is approximated by ray casts and the tire friction is an anisotropic friction model.

A raycast vehicle works by casting a ray for each wheel. Using the ray’s intersection point, we can calculate the suspension length and hence the suspension force that is then applied to the chassis, keeping it from hitting the ground. In effect, the vehicle chassis ‘floats’ along on the rays.

The friction force is calculated for each wheel where the ray contacts the ground. This is applied as a sideways and forwards force.

You can check Kester Maddock’s approach to vehicle simulation [here](#). It includes some common problems, workarounds and tips and tricks.

### How to use

Currently the Vehicle Controller can only be used as a constraint via Python. There are plans to add it to the interface.

**Setup** You should have a body acting as the chassis, set it as a ‘Rigid Body’.

The wheels should be separate objects set to ‘No Collision’. The vehicle controller will calculate the collisions for you as rays so, if you set it to something else, it will calculate it twice in different ways and produce weird results.

**Collisions** A cylinder is typically a good collision shape for the wheels. For the chassis, the shape should be rough, like a box. If the vehicle is very complicated, you should split it into simpler objects and parent those (with their collision shapes) to the vehicle controller so that they will follow it. If your vehicle even has moving bits (weapons, wrecking balls, trolleys etc) they should also be simulated separately and connected to the vehicle as a joint.

## Python

**Assembling the Vehicle** The overall steps are:

- create a constraint for the vehicle and save its ID for future reference
- attach the wheels
- set wheel parameters: influence, stiffness, damping, compression and friction
- init variables

You can see an example in the file below.

**Controlling the Vehicle** This is done in 2 parts and it should be modeled according to the desired behavior. You should think of your gameplay and research appropriate functions for the input. For instance, can the vehicle reverse? jump? drift? does it turn slowly? How much time does it take to brake or get to full speed? The first part is **response to keys**. Whenever the player presses a key, you should set a value accordingly, such as increase acceleration. Example:

```
if key[0] == events.UPARROWKEY:
    logic.car["force"] = -15.0
elif key[0] == events.RIGHTARROWKEY:
    logic.car["steer"] -= 0.05
```

The second part is to **compute the movement** according to your functions.

```
## apply engine force ##
for i in range(0, totalWheels):
    vehicle.applyEngineForce(logic.car["force"], i)
...
## slowly ease off gas and center steering ##
logic.car["steer"] *= 0.6
logic.car["force"] *= 0.9
```

Both should be run each frame.

**Example** [demo\\_file.zip](#) (last update 9 September 2014)

## Occlude Object Type

If an Occlude type object is between the camera and another object, that other object will not be rasterized (calculated for rendering). It is culled because it is occluded.

There is a demo .blend file to exemplify some concepts: [BGE-Physics-Objects-Occluder.blend](#)

- A messed-up, subdivided Cube named “Cube”.
- Another one behind a “Physics Type: Occlude” plane, named “Cube.BG”.
- Another one outside the view Frustum, named “Cube.OffCamera”.

Now observe what happens to the profiling stats for each of the following (in order):

- Hit **P** as the scene is. It hums along at a fairly slow rate. On my system the Rasterizer step takes 130ms. The framerate will finally jump up once the “Cube” object has completely moved out of the view frustum. It’s as if the Occluder doesn’t do anything while the Cube is behind it.
- Delete the “Cube.OffCamera” object above, and notice that there is no improvement in speed. This is the view frustum culling working for you - it does not matter if that object exists or not.

- Hit Z to view wireframe. Notice that in the 3D Viewport you can see “Cube.BG”, but once you press P, it is not there.
- Make the “Occluder” object take up the whole camera’s view with S-X-5. You will see a huge leap in framerate, since almost nothing is being Rasterized. On my system the Rasterizer step drops to 5ms.
- Try a run with *World properties* → *Physics* → *Occlusion Culling* disabled. It will be slow again.
- Reenable *World properties* → *Physics* → *Occlusion Culling* and run it one more time to prove to yourself that your speed is back.
- Change the Occluder to “Physics Type: Static”. Notice that it is back to the original slowness.
- Change it back to “Physics Type: Occlude”.
- Now make the “Occluder” invisible. The framerate is back down to its original, slow rate.

## Details

As far as Physics is concerned, this type is equivalent to Rigid Object “No collision”. The reason why the Occluder mode is mutually exclusive with other physics mode is to emphasize the fact that occluders should be specifically designed for that purpose and not every mesh should be an occluder. However, you can enable the Occlusion capability on physics objects using Python and Logic bricks - see (Link- TODO)

When an occluder object enters the view frustum, the BGE builds a ZDepth buffer from the faces of that object. Whether the faces are one-side or two-side is important: only the front faces and two-side faces are used to build the ZDepth buffer. If multiple occluders are in the view frustum, the BGE combines them and keeps the most foreground faces.

The resolution of the ZDepth buffer is controllable in the World settings with the “Occlu Res” button:

By default the resolution is 128 pixels for the largest dimension of the viewport while the resolution of the other dimension is set proportionally. Although 128 is a very low resolution, it is sufficient for the purpose of culling. The resolution can be increased to maximum 1024 but at great CPU expense.

The BGE traverses the DBVT (Dynamic Bounding Volume Tree) and for each node checks if it is entirely hidden by the occluders and if so, culls the node (and all the objects it contains).

To further optimize the feature, the BGE builds and uses the ZDepth buffer only when at least one occluder is in the view frustum. Until then, there is no performance decrease compared to regular view frustum culling.

## Recommendations

Occlusion culling is most useful when the occluders are large objects (buildings, mountains, ...) that hide many complex objects in an unpredictable way. However, don’t be too concerned about performance: even if you use it inappropriately, the performance decrease will be limited due to the structure of the algorithm.

There are situations where occlusion culling will not bring any benefit:

- If the occluders are small and don’t hide many objects. - In that case, occlusion culling is just dragging your CPU down).
- If the occluders are large but hides simple objects. - In that case you’re better off sending the objects to the GPU).
- If the occluders are large and hides many complex objects but in a very predictable way.
  - Example: a house full of complex objects. Although occlusion culling will perform well in this case, you will get better performance by implementing a specific logic that hides/unhides the objects; for instance making the objects visible only when the camera enters the house).
- Occluders can be visible graphic objects but beware that too many faces will make the ZDepth buffer creation slow.

- For example, a terrain is not a good candidate for occlusion: too many faces and too many overlap. Occluder can be invisible objects placed inside more complex objects (ex: “in the walls” of a building with complex architecture). Occluders can have “holes” through which you will see objects.

## Sensor

The object detects static and dynamic objects but not other collisions sensors objects. The Sensor is similar to the physics objects that underlie the Near and Radar sensors. Like the Near and Radar object it is:

- static and ghost
- invisible by default
- always active to ensure correct collision detection
- capable of detecting both static and dynamic objects
- ignoring collision with their parent
- capable of broadphase filtering based on: - Actor option: the collisioning object must have the Actor flag set to be detected - property/material: as specified in the collision sensors attached to it.

Broadphase filtering is important for performance reason: the collision points will be computed only for the objects that pass the broadphase filter.

- automatically removed from the simulation when no collision sensor is active on it

Unlike the Near and Radar object it can:

- take any shape, including triangle mesh
- be made visible for debugging (just use the Visible actuator)
- have multiple collision sensors using it

Other than that, the sensor objects are ordinary objects. You can move them freely or parent them. When parented to a dynamic object, they can provide advanced collision control to this object.

The type of collision capability depends on the shape:

- box, sphere, cylinder, cone, convex hull provide volume detection.
- triangle mesh provides surface detection but you can give some volume to the surface by increasing the margin in the Advanced Settings panel. The margin applies on both sides of the surface.

Performance tip:

- Sensor objects perform better than Near and Radar: they do less synchronizations because of the Scenegraph optimizations and they can have multiple collision sensors on them (with different property filtering for example).
- Always prefer simple shape (box, sphere) to complex shape whenever possible.
- Always use broadphase filtering (avoid collision sensor with empty property/material)
- Use collision sensor only when you need them. When no collision sensor is active on the sensor object, it is removed from the simulation and consume no CPU.

Known limitations:

- When running Blender in debug mode, you will see one warning line of the console:
  - :: warning btCollisionDispatcher::needsCollision: static-static collision!” In release mode this message is not printed.
- Collision margin has no effect on sphere, cone and cylinder shape.

## Settings

**Invisible** [See Here](#)

## Collision Bounds

[See Here.](#)

## 2.12.6 Performance

### Introduction

When developing games, game engineers, software and hardware developers uses some tools to fine tune their games to specific platforms and operating systems, defining a basic usage scenario whereas the users would have the best possible experience with the game.

Most of these tools, are software tools available for the specific Game Engines whereas the games were being developed and will run.

Blender Game Engine also comes with some visual tools to fine tune the games being developed, so the game developers could test the best usage scenario and minimum software and hardware requirements to run the game.

In Blender, those tools are available at the *System* and *Display* tab of *Render Context* in the *Properties Window*. There are options for specific performance adjusts and measurements, ways to control the frame rate or the way the contents are rendered in Blender window (game viewport) while the game runs, as well as controls for maintainnig geometry allocated in graphic cards memory.

**Blender Game Engine rendering system controls:** [System](#) - Controls for Scene rendering while the game is running.

**Blender Game Engine Performance measurements:** [Display](#) - Controls for showing specific data about performance while the game is running.

### System

The *System* tab at the Render context of the Properties Window, let the game developer specify options about the system performance regarding to frame discards and restrictions about frame renderings, the key to stop the Blender Game Engine, and whether to maintain geometry in the internal memory of the Graphic card.

### Options



Fig. 2.2683: System tab at the Render Context

**Use Frame Rate** When checked, this will inform Blender whether to run freely without frame rate restrictions or not. The frame rate is specified at the *Display* tab of the *Render Context* of the *Properties Window*. For more information about frame rates, see the [Display](#) page.

**Display Lists** When checked, this will tell Blender to maintain the lists of the meshes geometry allocated at the GPU memory. This can help to speed up viewport rendering during the game if you have enough GPU memory to allocate geometry and textures.

**Restrict Animation Updates** When checked, this will force Blender game engine to discard frames (even at the middle of redrawing, sometimes causing *tearing* artifacts) if the rate of frame rendered by the GPU is greater than the specified at the **Display** Tab.

**Exit Key** Clicking at this button will ask the user to type a key to specify a key to stop the game engine from running.

## Display

The *Display* tab at the *Render* context of the *Properties* Window, let the game developer specify the maximum frame rate of the animations shown during the game execution, whether to see informations like framerate and profile, debug properties, physics geometry visualization, warnings, if the mouse cursor is shown during the game execution, and options to specify the framing style of the game to fit the window with the specified resolution.

## Options

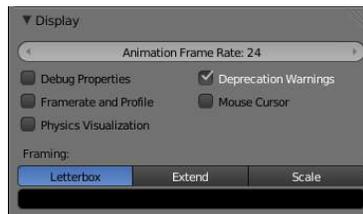


Fig. 2.2684: Fig. 1 - Display Tab at the Render Context

**Animation Frame Rate** This numeric field/slider specify the maximum frame rate at which the game will run. Minimum is 1, maximum is 120.

**Debug Properties** When checked, if a property was previously checked to be debugged during the game, the values of this property will be shown with the `Framerate` and `Profile` contents.

**Framerate and Profile** When checked, this will show values for each of the calculations Blender is doing while the game is running, plus the properties marked to be debugged.

**Physics visualization** Shows a visualization of physics bounds and interactions (like hulls and collision shapes), and their interaction.

**Deprecation Warnings** Every time when the game developer uses a deprecated functionality (which in some cases are outdated or crippled OpenGL Graphic cards functions), the system will emit warnings about the deprecated function.

**Mouse Cursor** Whether to show or not the mouse cursor when the game is running.

**Framing** There are three types of framing available:

**Letterbox** Show the entire viewport of the game in display window, using horizontal and/or vertical bars when needed.

**Extend** Show the entire viewport of the game in display window, viewing more horizontally or vertically.

**Scale** Stretch or Squeeze the viewport to fill the display window.

**Color Bar** This will let the game developer choose the bar colors when using the **Letterbox** Framing mode.

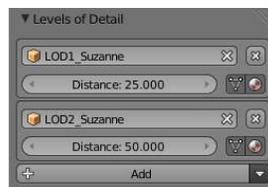
## Introduction

When creating visual assets it is often desirable to have a high amount of detail in the asset for up close viewing. However, this high amount of detail is wasted if the object is viewed from a distance, and brings down the scene's performance. To solve this, the asset can be swapped out at certain viewing distances. This is commonly referred to as a level of detail system. Each visual step of the asset is known as a level of detail. Levels of detail are most appropriate to use when you have a large scene where certain objects can be viewed both up close and from a distance.

## Settings

### Note: Modifiers on Level of Detail Objects

Any level of detail objects that have a modifier do not display correctly in the game engine. You will need to apply any modifiers for level of detail objects to appear correctly. A fix for this is being looked into.



Level of detail settings can be found in the Object settings when the renderer is set to Blender Game. In the Levels of Detail panel is a button to add a new level of detail to the current object. The settings for each level of detail is displayed in its own box. The exception to this is the base level of detail. This is automatically setup as the current object with a distance setting of 0. To remove a level of detail, click on the X button in the top right corner of the box of the level to be removed.

**Object** The object to use for this level of detail.

**Distance** The distance at which this level of detail becomes visible.

**Use Mesh** When this option is enabled, the mesh from the level of detail object is used until a lower level of detail overrides it.

**Use Material** When this option is enabled, the material from the level of detail object is used until a lower level of detail overrides it.

## Tools

Some tools for making levels of detail easier to manage and create can be found from the drop down menu next to the add button in the Levels of Detail panel.

**Set By Name** Searches the scene for specifically named objects and attempts to set them up as levels of detail on the currently selected object. The selected object must be the base level of detail (e.g. LOD0). This can be useful to quickly setup levels of detail on imported assets. In order to make use of this tool, your naming must be consistent, and each level must be prefixed or suffixed with "lodx" where x is the level that object is intended for. The case on "lod" must be consistent across all objects. Below are some example names that the tool will recognize.

- LOD0\_Box, LOD1\_Box, LOD2\_Box
- Box.lod0, Box.lod1, Box.lod2
- LoD0box, LoD1box, LoD2box



**Generate** This tool generates and sets up levels of details based on the selected object. Generation is done using the decimate modifier. Generation does not apply the modifier to allow further changing the settings. Generated objects are automatically named based on the level they are generated for. Below are some settings for the operator.

**Count** The number of levels desired after generation. This operator creates Count - 1 new objects.

**Target Size** The ratio setting for the decimate modifier on the last level of detail. The ratio settings for the other levels is determined by linear interpolation.

**Package into Group** With this setting enabled the operator performs some extra tasks to make the asset ready for easy linking into a new file. The base object and all of its levels of detail are placed into a group based on the base object's name. Levels other than the base are hidden for both the viewport and rendering. This simplifies the appearance of the system and does not affect the appearance of the base object. Finally, all levels are parented to the base object to remove clutter from the outliner.

**Clear All** Clears the level of detail settings from the current object.

## 2.12.7 Python API

### Introduction

This site is currently under development.

To see the full Python API please click on the following link: [Python API](#).

More informations:

- [Bullet physics](#)
- [Video Texture](#)

### Bullet physics Python API

Bullet Physics provides collision detection and rigid body dynamics for the Blender Game Engine. It takes some settings from Blender that previously were designed for the former collision detection system (called Sumo).

However, new features don't have an user interface yet, so Python can be used to fill the gap for now.

Features:

- Vehicle simulation.
- Rigid body constraints: hinge and point to point (ball socket).
- Access to internal physics settings, like deactivation time, debugging features.

Easiest is to look at the Bullet physics demos, how to use them. More information can be found [here](#).

Python script example:

```
import PhysicsConstraints
print dir(PhysicsConstraints)
```

**Note:** Note about parameter settings

Since this API is not well documented, it can be unclear what kind of values to use for setting parameters. In general, damping settings should be in the range of 0 to 1 and stiffness settings should not be much higher than about 10.

---

## The VideoTexture module: `bge.texture`

The `bge.texture` module allows you to manipulate textures during the game. Several sources for texture are possible: video files, image files, video capture, memory buffer, camera render or a mix of that. The video and image files can be loaded from the internet using an URL instead of a file name. In addition, you can apply filters on the images before sending them to the GPU, allowing video effect: blue screen, color band, gray, normal map. `bge.texture` uses FFmpeg to load images and videos. All the formats and codecs that FFmpeg supports are supported by `bge.texture`, including but not limited to:

- AVI
- Ogg
- Xvid
- Theora
- dv1394 camera
- video4linux capture card (this includes many webcams)
- videoForWindows capture card (this includes many webcams)
- JPG

### How it works

The principle is simple: first you identify an existing texture by object and name, then you create a new texture with dynamic content and swap the two textures in the GPU. The GE is not aware of the substitution and continues to display the object as always, except that you are now in control of the texture. At the end, the new texture is deleted and the old texture restored.

The present page is a guide to the `bge.texture` module with simple examples.

### Game preparation

Before you can use the thing `bge.texture` module, you must have objects with textures applied appropriately.

Imagine you want to have a television showing live broadcast programs in the game. You will create a television object and UV-apply a different texture at the place of the screen, for example `tv.png`. What this texture looks like is not important; probably you want to make it dark grey to simulate power-off state. When the television must be turned on, you create a dynamic texture from a video capture card and use it instead of `tv.png`: the TV screen will come to life.

You have two ways to define textures that `bge.texture` can grab:

- Simple UV texture.
- Blender material with image texture channel.

Because `bge.texture` works at texture level, it is compatible with all GE fancy texturing features: GLSL, multi-texture, custom shaders, etc.

### First example

Let's assume that we have a game object with one or more faces assigned to a material/image on which we want to display a video.

The first step is to create a `Texture` object. We will do it in a script that runs once. It can be at the start of the game, the video is only played when you refresh the texture; we'll come to that later. The script is normally attached to the object on which we want to display the video so that we can easily retrieve the object reference:

```
import bge.texture

contr = GameLogic.getCurrentController()
obj = contr.owner

if not hasattr(GameLogic, 'video'):
```

The check on `video` attribute is just a trick to make sure we create the texture only once.

### Find material

```
matID = bge.texture.materialID(obj, 'IMvideo.png')
```

`bge.texture.materialID()` is a handy function to retrieve the object material that is using `video.png` as texture. This method will work with Blender material and UV texture. In case of UV texture, it grabs the internal material corresponding to the faces that are assigned to this texture. In case of Blender material, it grabs the material that has an image texture channel matching the name as first channel.

The `IM` prefix indicates that we're searching for a texture name but we can also search for a material by giving the `MA` prefix. For example, if we want to find the material called `VideoMat` on this object, the code becomes:

```
matID = bge.texture.materialID(obj, 'MAVideoMat')
```

**Create texture** `bge.texture.Texture` is the class that creates the `Texture` object that loads the dynamic texture on the GPU. The constructor takes one mandatory and three optional arguments:

**gameObj** The game object.

**materialID** Material index as returned by `bge.texture.materialID()`, 0 = first material by default.

**textureID** Texture index in case of multi-texture channel, 0 = first channel by default. In case of UV texture, this parameter should always be 0.

**textureObj** Reference to another `Texture` object of which we want to reuse the texture. If we use this argument, we should not create any source on this texture and there is no need to refresh it either: the other `Texture` object will provide the texture for both materials/textures.

```
GameLogic.video = bge.texture.Texture(obj, matID)
```

**Make texture persistent** Note that we have assigned the object to a `GameLogic`, `video` attribute that we create for the occasion. The reason is that the `Texture` object must be persistent across the game scripts. A local variable would be deleted at the end of the script and the GPU texture deleted at the same time. `GameLogic` module object is a handy place to store persistent objects.

**Create a source** Now we have a `Texture` object but it can't do anything because it does not have any source. We must create a source object from one of the possible sources available in `bge.texture`:

**VideoFFmpeg** Moving pictures. Video file, video capture, video streaming.

**ImageFFmpeg** Still pictures. Image file, image on web.

**ImageBuff** Image from application memory. For computer generated images, drawing applications.

**ImageViewport** Part or whole of the viewport (=rendering of the active camera displayed on screen).

**ImageRender** Render of a non active camera.

**ImageMix** A mix of 2 or more of the above sources.

In this example we use a simple video file as source. The `VideoFFmpeg` constructor takes a file name as argument. To avoid any confusion with the location of the file, we will use `GameLogic.expandPath()` to build an absolute file name, assuming the video file is in the same directory as the blend file:

```
movie = GameLogic.expandPath('//trailer_400p.ogg')
GameLogic.video.source = bge.texture.VideoFFmpeg(movie)
```

We create the video source object and assign it to the `Texture` object `source` attribute to set the source and make it persistent: as the `Texture` object is persistent, the source object will also be persistent.

Note that we can change the `Texture` source at any time. Suppose we want to switch between two movies during the game. We can do the following:

```
GameLogic.mySources[0] = bge.texture.VideoFFmpeg('movie1.avi')
GameLogic.mySources[1] = bge.texture.VideoFFmpeg('movie2.avi')
```

And then assign (and reassign) the source during the game:

```
GameLogic.video.source = GameLogic.mySources[movieSel]
```

**Setup the source** The `VideoFFmpeg` source has several attributes to control the movie playback:

**range** [start,stop] (*floats*). Set the start and stop time of the video playback, expressed in seconds from beginning. By default the entire video.

**repeat** (*integer*). Number of video replay, -1 for infinite.

**framerate** (*float*). Relative frame rate, <1.0 for slow, >1.0 for fast.

**scale** (*bool*). Set to True to activate fast nearest neighbour scaling algorithm. Texture width and height must be a power of 2. If the video picture size is not a power of 2, rescaling is required. By default `bge.texture` uses the precise but slow `gluScaleImage()` function. Best is to rescale the video offline so that no scaling is necessary at runtime!

**flip** (*bool*). Set to True if the image must be vertically flipped. FFmpeg always delivers the image upside down, so this attribute is set to True by default.

**filter** Set additional filter on the video before sending to GPU. Assign to one of `bge.texture` filter object. By default the image is send unchanged to the GPU. If an alpha channel is present in the video, it is automatically loaded and sent to the GPU as well.

We will simply set the `scale` attribute to True because the `gluScaleImage()` is really too slow for real time video. In case the video dimensions are already a power of 2, it has no effect.

```
GameLogic.video.source.scale = True
```

**Play the video** We are now ready to play the video:

```
GameLogic.video.source.play()
```

Video playback is not a background process: it happens only when we refresh the texture. So we must have another script that runs on every frame and calls the `refresh()` method of the `Texture` object:

```
if hasattr(GameLogic, 'video'):
    GameLogic.video.refresh(True)
```

If the video source is stopped, `refresh()` has no effect. The argument of `refresh()` is a flag that indicates if the texture should be recalculated on next refresh. For video playback, you definitely want to set it to `True`.

**Checking video status** Video source classes (such as `VideoFFmpeg`) have an attribute `status`. If video is playing, its value is 2, if it's stopped, it's 3. So in our example:

```
if GameLogic.video.source.status == 3:
    #video has stopped
```

**Advanced work flow** `True` argument in `Texture.refresh()` method simply invalidates the image buffer after sending it to the GPU so that on next frame, a new image will be loaded from the source. It has the side effect of making the image unavailable to Python. You can also do it manually by calling the `refresh()` method of the source directly.

Here are some possible advanced work flow:

- Use the image buffer in python (doesn't effect the Texture):

```
GameLogic.video.refresh(False)
image = GameLogic.video.source.image
# image is a binary string buffer of row major RGBA pixels
# ... use image
# invalidates it for next frame
GameLogic.video.source.refresh()
```

- Load image from source for python processing without download to GPU:
- note that we don't even call refresh on the Texture
- we could also just create a source object without a Texture object

```
image = GameLogic.video.source.image
# ... use image
GameLogic.video.source.refresh()
```

- If you have more than 1 material on the mesh and you want to modify a texture of one particular material, get its ID

```
matID = bge.texture.materialID(gameobj, "MAMat.001")
```

GLSL material can have more than 1 texture channel, identify the texture by the texture slot where it is defined, here 2

```
tex=bge.texture.Texture(gameobj, matID, 2)
```

## Advanced demos

Here is a [demo](#) that demonstrates the use of two videos alternatively on the same texture. Note that it requires an additional video file which is the elephant dream teaser. You can replace with another other file that you want to run the demo.

Here is a [demo](#) that demonstrates the use of the `ImageMix` source. `ImageMix` is a source that needs sources, which can be any other `Texture` source, like `VideoFFmpeg`, `ImageFFmpeg` or `ImageRender`. You set them with `setSource()` and their relative weight with `setWeight()`. Pay attention that the weight is a short number between 0 and 255, and that the sum of all weights should be 255. `ImageMix` makes a mix of all the sources according to their weights. The sources must all have the same image size (after reduction to the nearest power of 2 dimension). If they don't, you get a Python error on the console.

## 2.12.8 Standalone Player

The standalone player allows a Blender game to be run without having to load the Blender system. This allows games to be distributed to other users, without their requiring a detailed knowledge of Blender (and also without the possibility of unauthorised modification). Note that the Game Engine Save as Runtime is an add-on facility which must be pre-loaded before use.

The following procedure will give a standalone version of a working game.

- **File - User Preferences - Add-ons: - Game Engine - Save as Game Engine Runtime - Install Add-on** (button).

(You can also **Save as Default** button, in which case the add-on will always be present whenever Blender is re-loaded).

- **File - Export - Save as Game Engine Runtime - (give appropriate directory/filename)- Save as Game Engine Runtime** (button).

The game can then be executed by running the appropriate .exe file. Note that all appropriate libraries are automatically loaded by the add-on.

If you are interested in licensing your game, read [Licensing](#) for a discussion of the issues involved.

---

**Tip:** Exporting...

If the game is to be exported to other computers, make a new empty directory for the game runtime and all its ancilliary libraries etc. Then make sure the **whole directory** is transferred to the target computer

---

## 2.12.9 Licensing of Blender Games

Blender and the Blender Game Engine (BGE) is licensed as GNU GPL, which means that your games (if they include Blender software) have to comply with that license as well. This only applies to the software, or the bundle if it has software in it, not to the artwork you make with Blender. All your Blender creations are your sole property.

GNU GPL - also called “Free Software” - is a license that aims at keeping the licensed software free, forever. GNU GPL does not allow you to add new restrictions or limitations on the software you received under that license. That works fine if you want your clients or your audience to have the same rights as you have (with Blender).

In summary, the software and source-code is bound to the GNU GPL, but the blend files, (models, textures, sounds) are not.

### Standalone Games

In case you save out your game as a single “Standalone” the .blend file gets included in the binary (the BGE player). That requires the .blend file to be compatible with the GNU GPL license.

In this case, you could decide to load and run another .blend file game (using the Game Actuator logic brick). That file then is not part of the binary, so you can apply any license you wish on it.

### More Information

More information you can find in the [blender.org FAQ](#).

## 2.13 Extensions

### 2.13.1 Extending Blender

Unlike many programs you may be familiar with, Blender is not monolithic and static. You can extend its functionality with [Python scripting](#) without having to modify the source and recompile

#### Add-ons

Add-ons are scripts you can enable to gain extra functionality within Blender, they can be enabled from the user preferences.

Outside of the Blender executable, there are literally hundreds of add-ons written by many people:

- Officially supported add-ons are bundled with Blender.
- Other **Testing** add-ons are included in development builds but not official releases, many of them work reliably and are very useful but are not ensured to be stable for release.

An Overview of all add-ons is available in this wiki in the [Scripts Catalog](#) and in the [Extensions tracker](#).

#### Scripts

Apart from add-ons there are also scripts you can use to extend Blenders functionality:

- Modules: Utility libraries for import into other scripts.
- Presets: Settings for Blender's tools and key configurations.
- Startup: These files are imported when starting Blender. They define most of Blender's UI, as well as some additional core operators.
- Custom scripts: In contrast to add-ons they are typically intended for one-time execution via the [text editor](#)

#### Saving your own scripts

##### File location

All scripts are loaded from the `scripts` folder of the [local](#), [system](#) and [user paths](#).

You can setup an additional search path for scripts in [User preferences](#) (*User Preferences* -> *File Paths*).

##### Installation

Add-ons are conveniently installed through Blender in the *User Preferences* -> *Add-ons* window. Click the *Install from File...* button and select the `.py` or `.zip` file.

To manually install scripts or add-ons place them in the `add-ons`, `modules`, `presets` or `startup` directory according to their type. See the description above.

You can also run scripts by loading them in the [text editor](#) window.

## 2.13.2 Blender, Python Manual

### Introduction

Welcome to the Blender Python Manual.

Python [www.Python.org](http://www.Python.org) is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax.

Python scripts are a powerful and versatile way to extend Blender functionality. Most areas of Blender can be scripted, including Animation, Rendering, Import and Export, Object Creation and the scripting of repetitive tasks.

To interact with Blender, scripts can make use of the tightly integrated API (Application Programming Interface).

### General information

Links that are useful while writing scripts.

- [Blender Python API](#)
  - Official API documentation. Use this for referencing while writing scripts.
- [API introduction](#)
  - A short introduction to get you started with the API. Contains examples.
- [CookBook](#)
  - A section of handy code snippets (yet to be written)
- [FAQ](#)
  - Frequently asked questions and their answers

Links that deal with distributing your scripts.

- [Sharing scripts](#)
  - Information on how to share your scripts and get them included in the official Blender distribution.
- [Creating Add-ons](#)
  - As of Blender 2.5 Alpha1, this is the new way to spread scripts for Blender.
- [Extensions project](#)
  - Project to maintain a central repository of extensions to Blender.

### Getting Started - Wiki tutorials

The following pages are located on this wiki and take you from the basics to the more advanced concepts of Python scripting for Blender.

- [Hello World](#)
- [Console](#)
- [Text editor](#)
- [Geometry](#)
- [Properties, ID-Properties and their differences](#)

## Getting Started - External links

The following pages are not located on this wiki, but contain a lot of good information to start learning how to write scripts for Blender.

- [Introductory tutorial by Satish Goda](#) - Takes you from the beginning and teaches how to do basic API manipulations.
- [Ira Krakow's video tutorials](#) - First video in a series of video tutorials.
- [Quickstart guide](#) - A quickstart guide for people who already have some familiarity with Python and Blender.
- [Examples thread](#) - A forum thread containing many short working script examples.
- [Introduction to Python](#) - A one hour video tutorial introducing Python and the Blender API.

### 2.13.3 Add-ons

*Add-on* is the general term for a script that extends Blender's functionality. They are found in the *Add-ons* tab of the *User Preferences* window. This tab allows to search, install, enable and disable Add-ons.

#### Searching

Blender comes with some useful Add-ons already, ready to be enabled, but you can also add your own, or any interesting ones you find on the web.

The [Scripts Catalog](#) provides an index of Add-ons that are included with Blender as well as listing a number of external Add-ons.

#### Enabling and Disabling

Enable and disable an add-on by checking or unchecking the box on the right of the add-on you chose, as shown on the figure.

The add-on functionality should be immediately available. If the Add-on does not activate when enabled, check the [Console window](#) for any errors that may have occurred.

You can click the arrow at the left of the add-on box to see more information, such as where it is located, a description and a link to the documentation. Here you can also find a button to report a bug specific of this add-on.

---

**Tip:** Saving Add-on Preferences

If you want an Add-on to be enabled every time you start Blender, you will need to *Save User Settings*.

---

#### Installation of a 3rd party Add-on

For add-ons that you found on the web or your own to show on the list, you have to install them first by clicking *Install from File...* and providing a *.zip* or *.py* file.

Alternatively you can manually install an Add-on, which is useful when developing your own add-ons. Move or link the files to `../scripts/addons` folder (where `..` is the path to your Blender configuration folder).

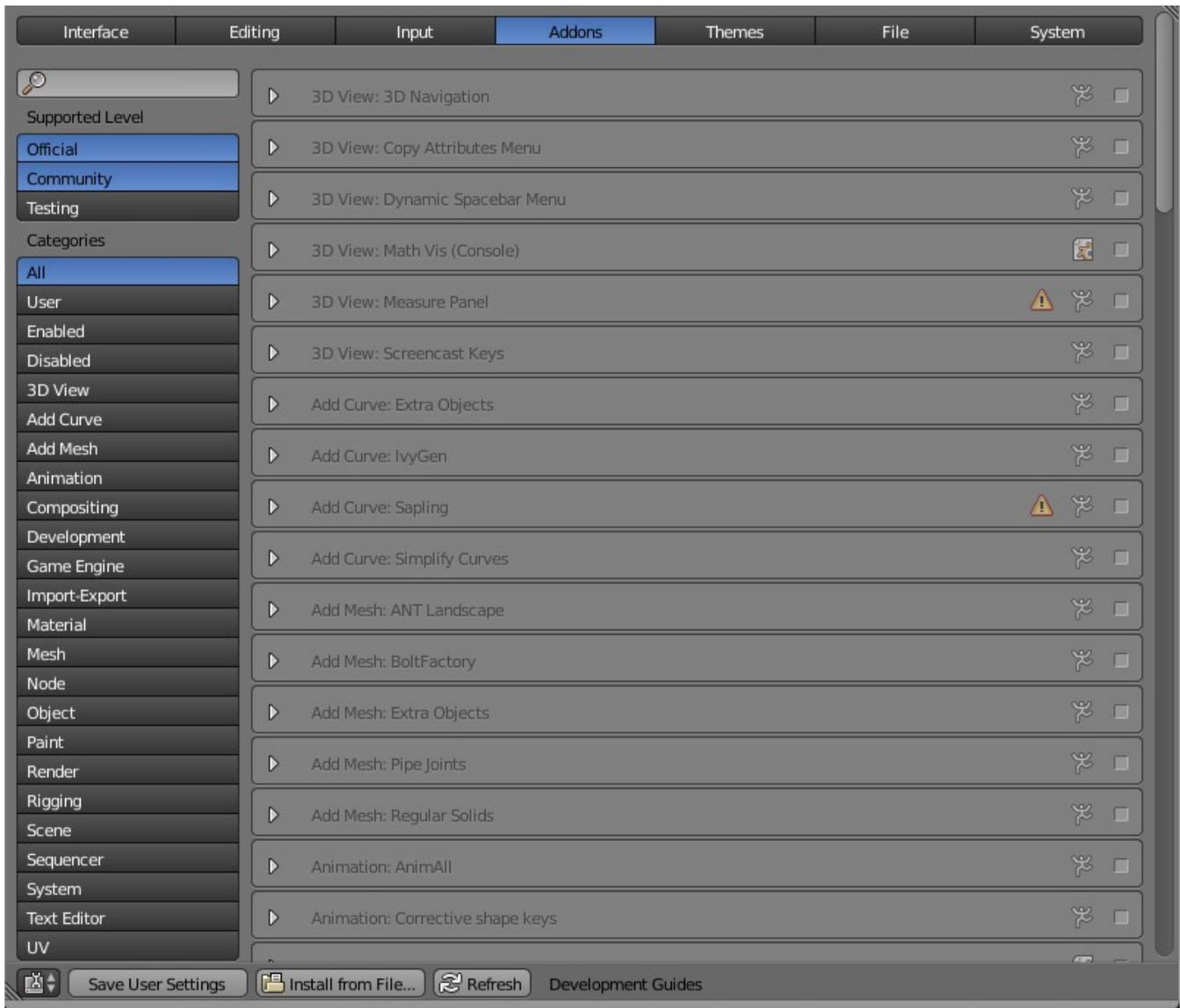


Fig. 2.2685: Add-ons tab in the User Preferences

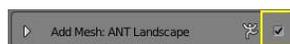


Fig. 2.2686: Enabling an Add-on

## File locations

For information on the location of blender directories see: [Configuration & Data Paths](#)

You can also create a personal folder containing new add-ons and configure your files' path in the *File* panel of the *User Preferences*. To create a personal script folder:

- Create an empty folder (i.e. 'script\_addon\_2-7x')
- Add one folder named 'addons'. It has to named like this for Blender to recognize it.
- Put your new add-ons in this 'addons' folder.
- open the *File* panel of the *User Preferences*.
- Fill the *Scripts* entry with the path to your script folder (i.e. 'script\_addon\_2-7x').

## Development guidelines

If you are a script developer, you may be interested in the [Add-ons development guidelines](#)

### 2.13.4 Why another Python tutorial?

This page exists for two reasons.

- To introduce Programming using Python 3.x quickly and efficiently.
- And most importantly teach inside of Blender's Console, so you can learn in context.

## What is Programming?

Programming in simple terms is nothing more than manipulating data. Operations on the data either modifies it, or create new data.

The simplest data is Numbers. Operations on Numbers are addition, Subtraction, multiplication etc., Its the simplest type of data imaginable.

But for solving real world problems, we need have compound data, that is built from simpler data like numbers. A good example is Vector data type, that is built from 3 numbers.

During the course of this tutorial, we will take a look at what data types that Python language provides and how you can create your own custom data for your programs and also write operations that work with that data.

## What is Python?

*Python* is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax.

Learning Python is also very easy, even if you have never programmed before.

## Python Interpreter

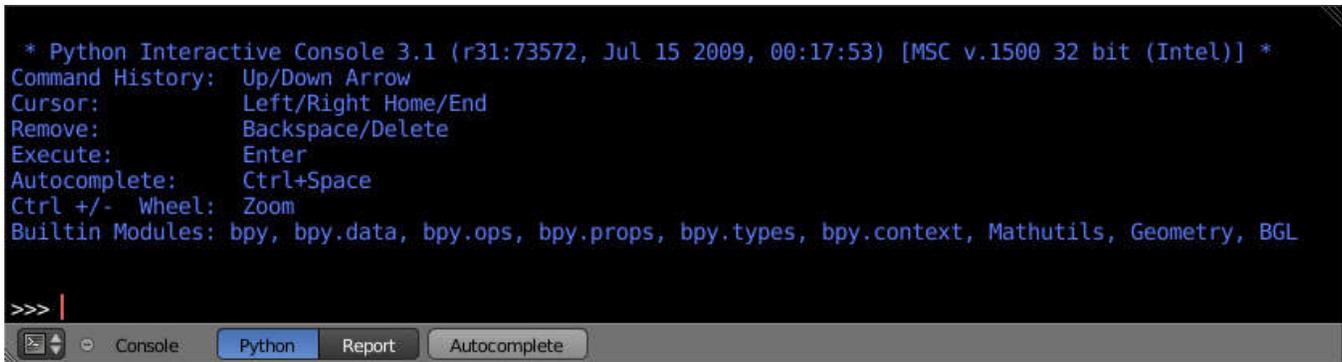
All the exercises in this tutorial will be using the built-in Console window type in Blender, which has a Python 3.x interpreter embedded in it.

Following is a video that shows how you can switch to the interpreter.

You can start typing Python commands, expressions and statements at the interpreter prompt: >>>

```
* Python Interactive Console 3.1 (r31:73572, Jul 15 2009, 00:17:53) [MSC v.1500 32 bit (Intel)] *
Command History: Up/Down Arrow
Cursor:         Left/Right Home/End
Remove:         Backspace/Delete
Execute:        Enter
Autocomplete:   Ctrl+Space
Ctrl +/- Wheel: Zoom
Builtin Modules: bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, Mathutils, Geometry, BGL

>>> |
```



## Hello World

Let's get started with the classical "Hello World" program.

Type the following print statement at the interpreter prompt and press Return key.

```
>>> print("Hello World")
Hello World
>>> |
```



Let's break down the above statement.

- "Hello World" is a string *literal* in Python. - A string is a sequence of characters (numbers, alphabets, special characters)
- `print()` is a built-in function in Python to print output.
- `print("Hello World")` outputs *Hello World* to the console.

## Exercise

Type the following commands and check the output

```
print('"Hello World"')
print('"Hello \n World"')
```

In Python, a string literal can be multiplied by a *number*. By doing so we are repeating the string by the count specified by *number*

- `number * string literal`
- `string literal * number`
- `*` is the multiplication operator in Python

```
>>> print("Hello World "*10)
Hello World Hello World Hello World Hello World Hello World Hello W
orld Hello World Hello World Hello World Hello World
>>> |
```



**Note:** Check out all the above examples in one place

## 2.13.5 The Console Editor Type

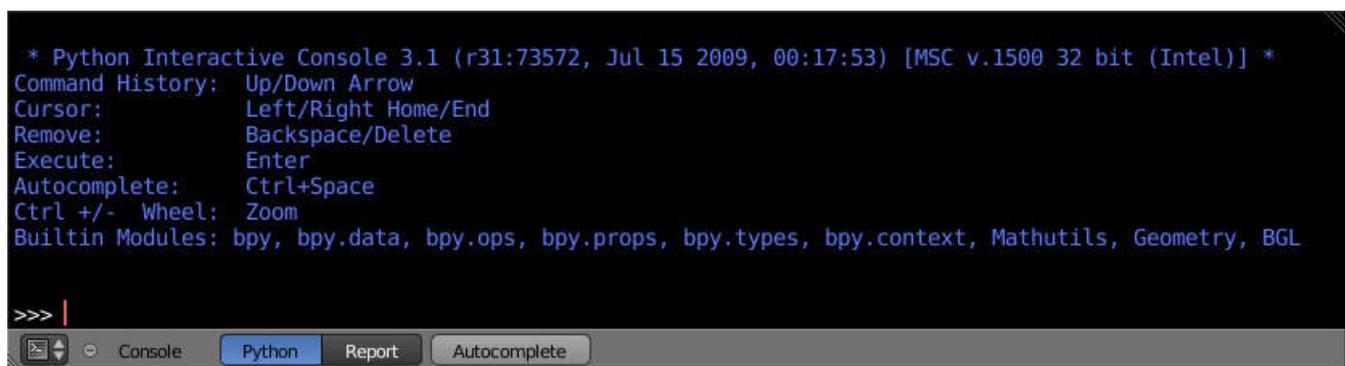
The interactive console in Blender 2.5 has been improved. Auto Complete, Python Reporting & more features have been added. Testing one-liners in the console is a good way to learn the Python API.

### Usage

#### Accessing Built-in Python Console

Launching the Console using mouse.

By pressing `Shift-F4` in any Blender Editor Type (3D View, Timeline etc..) you can change it to a Console Editor.



```
* Python Interactive Console 3.1 (r31:73572, Jul 15 2009, 00:17:53) [MSC v.1500 32 bit (Intel)] *
Command History: Up/Down Arrow
Cursor:          Left/Right Home/End
Remove:         Backspace/Delete
Execute:        Enter
Autocomplete:   Ctrl+Space
Ctrl +/- Wheel: Zoom
Builtin Modules: bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, Mathutils, Geometry, BGL

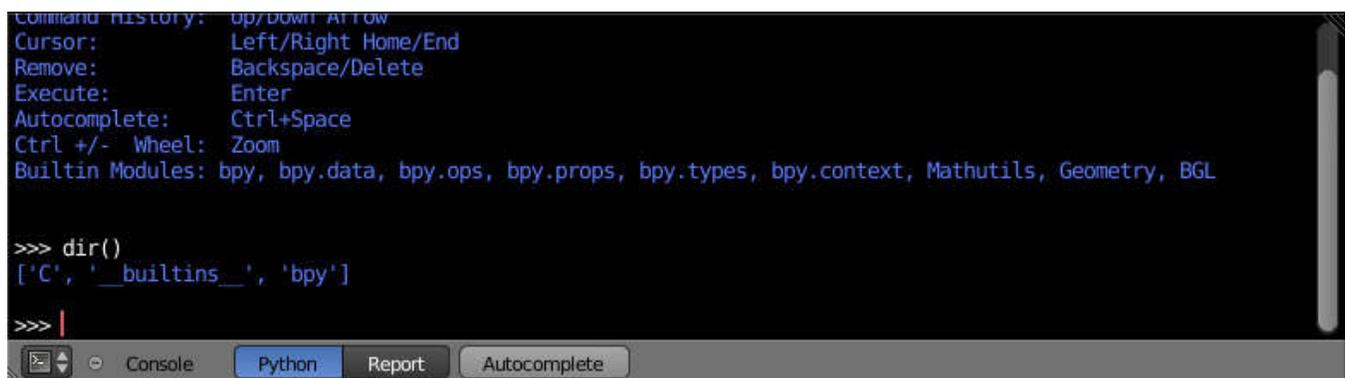
>>> |
```

From the screen shot above, you will notice that apart from the usual hot keys that are used to navigate, by pressing `Ctrl-Spacebar` you can enable Auto-complete feature.

Since Blender 2.5 uses Python 3.x, the interpreter is loaded and is ready to accept commands at the prompt `>>>`

#### First look at the Console Environment

To check what is loaded into the interpreter environment, type `dir()` at the prompt and execute it.



```
Command History: Up/Down Arrow
Cursor:          Left/Right Home/End
Remove:         Backspace/Delete
Execute:        Enter
Autocomplete:   Ctrl+Space
Ctrl +/- Wheel: Zoom
Builtin Modules: bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, Mathutils, Geometry, BGL

>>> dir()
['C', '__builtins__', 'bpy']

>>> |
```

Following is a quick overview of the output

- C Quick access to `bpy.context`
- D Quick access to `bpy.data`
- `__builtins__` Python Built-ins (Classes, functions, variables)
- `bpy` Top level Blender Python API module.

### Auto Completion at work

Now, type `bpy.` and then press `Ctrl-Spacebar` and you will see the Console auto-complete feature in action.

The screenshot shows the Blender Python console with the prompt `>>> bpy.` and a list of sub-modules: `app context data ops props types utils`. Below the console, there are buttons for `Console`, `Python`, `Report`, and `Autocomplete`.

You will notice that a list of sub-modules inside of `bpy` appear. These modules encapsulate all that we can do with Blender Python API and are very powerful tools.

Lets list all the contents of `bpy.app` module.

The screenshot shows the Blender Python console with the following output:

```
>>>
binary_path build date build_platform build_revision build_time build_type debug home n_fiel
ds n_sequence_fields n_unnamed_fields version version_string
>>> bpy.app.binary_path
'C:\DOCUMENT-1\furian\MYDOCU-1\DOWNLO-1\1225_R-1\Release\blender.exe'

version version_string
>>> bpy.app.version
(2, 50, 16)

>>> bpy.app.version_string
'2.50 (sub 16)'
```

Below the console, there are buttons for `Console`, `Python`, `Report`, and `Autocomplete`.

Notice the green output above the prompt where you enabled auto-completion. What you see is the result of auto completion listing. In the above listing all are module attribute names, but if you see any name end with `(')`, then that is a function.

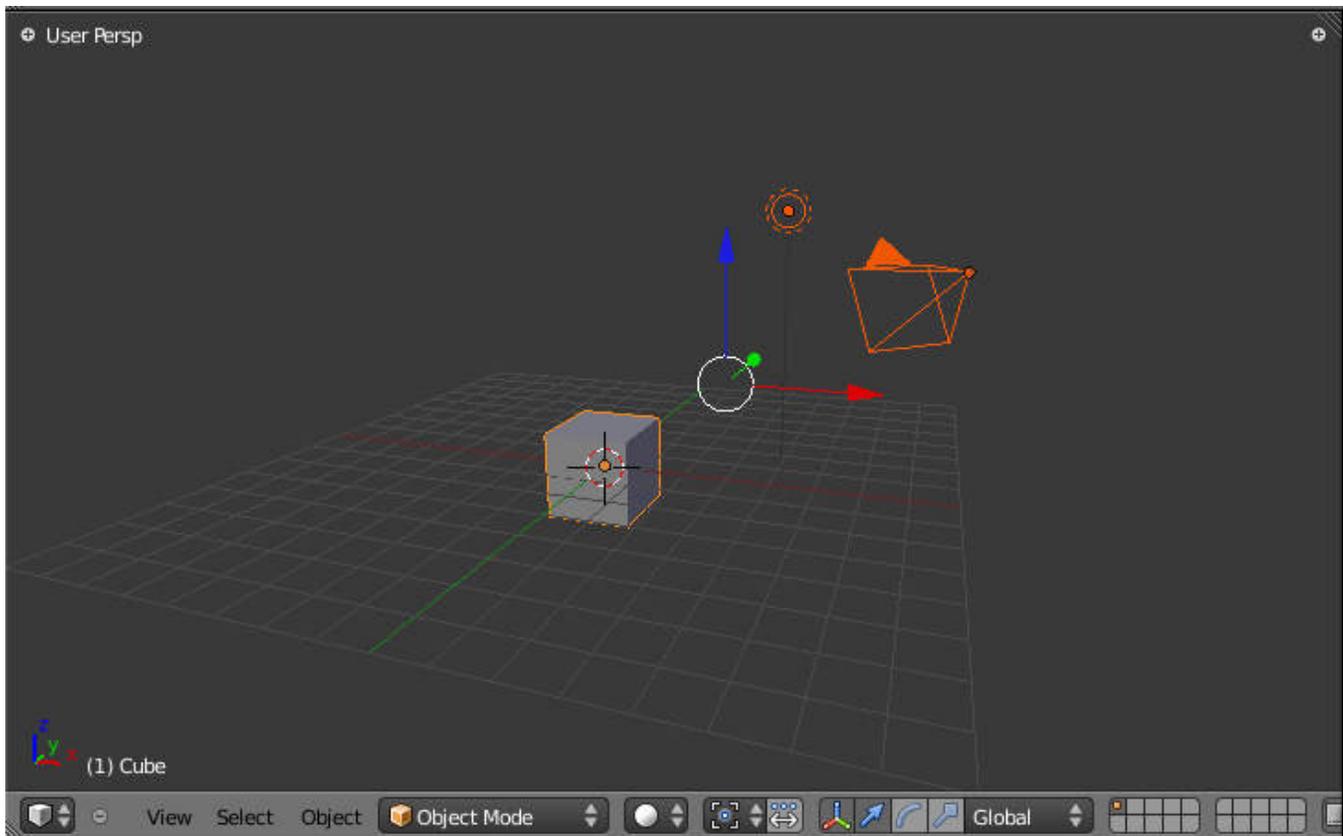
We will make use of this a lot to help our learning the API faster. Now that you got a hang of this, lets proceed to investigate some of modules in `bpy`.

### Before tinkering with the modules..

If you look at the 3D Viewport in the default Blender scene, you will notice 3 objects: Cube, Lamp and Camera.

- All objects exist in a context and there can be various modes under which they are operated upon.
- At any instance, only one object is active and there can be more than one selected objects.
- All objects are data in the Blender file.
- There are operators/functions that create and modify these objects.

For all the scenarios listed above (not all were listed, mind you..) the `bpy` module provides functionality to access and modify data.



## Examples

### bpy.context

**Note** For the commands below to show the proper output, make sure you have selected object(s) in the 3D view.

#### Try it out!

**bpy.context.mode** Will print the current 3D View mode (Object, Edit, Sculpt etc.,)

**bpy.context.object** or **bpy.context.active\_object** Will give access to the active object in the 3D View

```
>>> bpy.context.object.location.x = 1
```

Change x location to a value of 1

```
>>> bpy.context.object.location.x += 0.5
```

Move object from previous x location by 0.5 unit

```
>>> bpy.context.object.location = [1, 2, 3]
```

Changes x, y, z location

```
>>> bpy.context.object.location.xyz = [1, 2, 3]
```

Same as above

```

>>>
active_base active_bone active_object active_pose_bone area driver_add( edit_object editable_
bones_get( id_data is_property_hidden( is_property_set( items( keyframe_insert( keys( main
manager mode object particle_edit object_path_resolve( path_to_id( recast_type( region_rna_t
type scene screen sculpt_object selected_bases selected_bones selected_editable_bases selected
_editable_bones selected_editable_objects selected_objects selected_pose_bones space_data textu
re_paint_object tool_settings user_preferences values( vertex_paint_object visible_bones visib
le_pose_bones weight_paint_object window
>>> bpy.context.mode
'OBJECT'

>>> bpy.context.object
[BPy_StructRNA "Object" -> "Cube"]

active_base active_bone active_object active_pose_bone
>>> bpy.context.active_object
[BPy_StructRNA "Object" -> "Cube"]

selected_bases selected_bones selected_editable_bases selected_editable_bones selected_editable_
objects selected_objects selected_pose_bones
>>> bpy.context.selected_objects
[[BPy_StructRNA "Object" -> "Cube"], [BPy_StructRNA "Object" -> "Lamp"], [BPy_StructRNA "Object" ->
"Camera"]]

```

```
>>> type(bpy.context.object.location)
```

Data type of objects location

```
>>> dir(bpy.context.object.location)
```

Now that is a lot of data that you have access to

**bpy.context.selected\_objects** Will give access to a list of all selected objects.

```

>>> bpy.context.selected_objects then press {{Shortcut|Ctrl|Space}}
>>> bpy.context.selected_objects[0]

```

Prints out name of first object in the list

```
>>> [object for object in bpy.context.selected_objects if object != bpy.context.object]
```

Complex one... But this prints a list of objects not including the active object

### bpy.data

bpy.data has a bunch of functions and variables that give you access to all the data in the Blender file.

You can access following data in the current Blender file: objects, meshes, materials, textures, scenes, screens, sounds, scripts, texts, cameras, curves, lamps, brushes, armatures, images, lattices, libraries, worlds, groups, metaballs, particles, node\_groups

That's a lot of data.

**Try it out!**

```

BoolProperty( BoolVectorProperty( CollectionProperty( EnumProperty( FloatProperty( FloatVectorP
roperty( IntProperty( IntVectorProperty( PointerProperty( StringProperty( actions add_image(
armatures bl_rna brushes cameras curves driver_add( filename get( gpencil groups id_data
images is_property_hidden( is_property_set( items( keyframe_insert( keys( lamps lattices lib
raries materials meshes metaballs node_groups objects particles path_resolve( path_to_id( r
ecast_type( rna_type scenes screens scripts sounds texts textures values( vfonts window_ma
nagers worlds
>>> bpy.data
[BPy_StructRNA "Main"]

>>> bpy.data.objects
[BPy_PropertyRNA "Main" -> "objects"]

>>>
>>> for object in bpy.data.objects:
...     print(object.name + " is at location " + str(object.location))
...
Camera is at location [7.481132, -6.507640, 5.343665](vector)
Cube is at location [0.000000, 0.000000, 0.000000](vector)
Lamp is at location [4.076245, 1.005454, 5.903862](vector)

>>> |

```

### Exercise

```
>>> for object in bpy.data.scenes['Scene'].objects: print(object.name)
```

Return twice Prints the names of all objects belonging to the Blender scene with name "Scene"

```
>>> bpy.data.scenes['Scene'].objects.unlink(bpy.context.active_object)
```

Unlink the active object from the Blender scene named 'Scene'

```
>>> bpy.data.materials['Material'].shadows
>>> bpy.data.materials['Material'].shadows = False
```

### bpy.ops

The tool/action system in Blender 2.5 is built around the concept of operators. These operators can be called directly from console or can be executed by click of a button or packaged in a python script. Very powerful they are..

For a list of various operator categories, click [here](#)

Lets create a set of five Cubes in the 3D Viewport. First, delete the existing Cube object by selecting it and pressing X

**Try it out!** The following commands are used to specify that the objects are created in layer 1. So first we define an array variable for later reference:

```
>>> mylayers = [False]*20
>>> mylayers[0] = True
```

We create a reference to the operator that is used for creating a cube mesh primitive

```
>>> add_cube = bpy.ops.mesh.primitive_cube_add
```

Now in a for loop, we create the five objects like this (In the screenshot above, I used another method) Press ENTER-KEY twice after entering the command at the shell prompt.

```
>>> for index in range(0, 5):
...     add_cube(location=(index*3, 0, 0), layers=mylayers)
```

### 2.13.6 The Text Editor

Blender has a *Text Editor* among its windows types, accessible via the *Text Editor* button ( ) of the *Window type* menu, or via `Shift-F11`.

The newly opened Text window is grey and empty, with a very simple toolbar (*Text Toolbar*).

From left to right there are the standard *Window type* selection button and the window menus. Then there is the Text ID Block browse button followed by the New button for creating new Text files. Once you click it, you will find that the Toolbar has changed.. for good!

Now you find a textbox to change name of your text file, followed by + button to create new files. To remove the text block, click the **X** button.

The following three buttons toggle display of line numbers, word-wrap text and syntax highlighting respectively.

Typing on the keyboard produces text in the text buffer. As usual, pressing dragging and releasing LMB selects text.

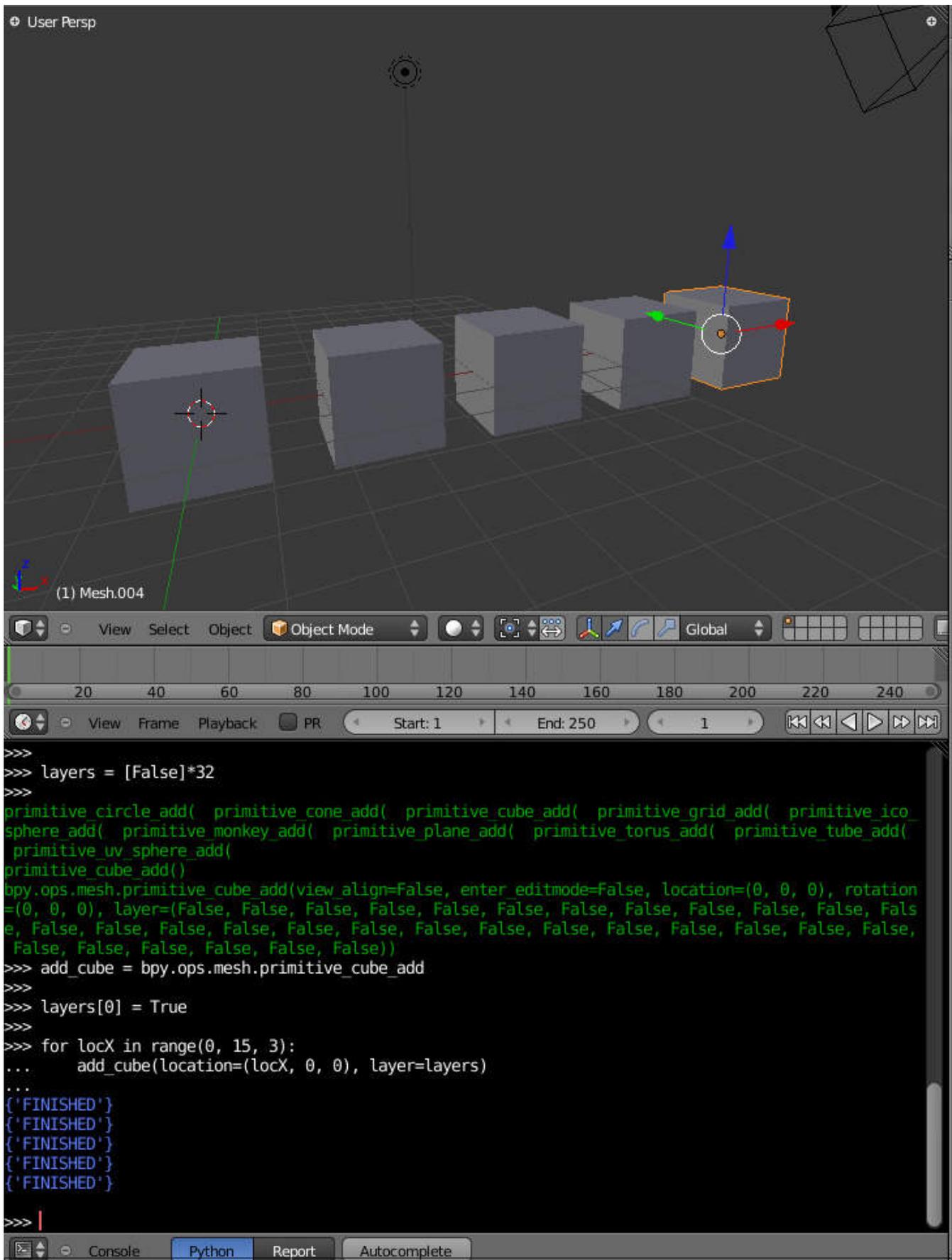
The following keyboard commands apply:

- `Ctrl-C` - Copies the marked text into the text clipboard.
- `Ctrl-X` - Cuts out the marked text into the text clipboard.
- `Ctrl-V` - Pastes the text from the clipboard at the cursor location in the Text window.
- `Shift-Ctrl-Alt-S` - Saves unsaved text as a text file, a *File Browser* window appears.
- `Alt-S` - Saves an already open file.
- `Alt-O` - Loads a text, a *File Browser* window appears.
- `Alt-P` - Executes the text as a Python script.
- `Ctrl-Z` - Undo.
- `Ctrl-Shift-Z` - Redo.
- `Alt-R` - Reopen (reloads) the current buffer (all non-saved modifications are lost).
- `Alt-M` - Converts the content of the text window into 3D text (max 100 chars).

To delete a text buffer just press the **X** button next to the buffer's name, just as you do for materials, etc.

The most notable keystroke is `Alt-P` which makes the content of the buffer being parsed by the internal Python interpreter built into Blender. The next page will present an example of Python scripting. Before going on it is worth noticing that Blender comes with a fully functional Python interpreter built in, and with a lots of Blender-specific modules, as described in the [API references](#).

The *Text Editor* has now also some dedicated Python scripts, which add some useful writing tools, like a class/function/variable browser, completion... You can access them through the *Text* -> *Text Plugins* menu entry.



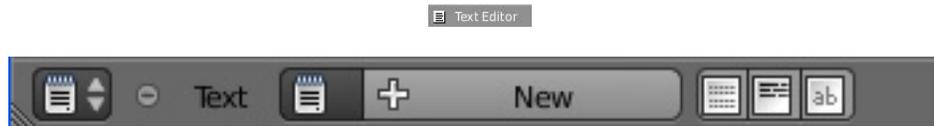


Fig. 2.2687: Text Toolbar.

### Other usages for the Text window

The text window is handy also when you want to share your `.blend` files with the community or with your friends. A Text window can be used to write in a README text explaining the contents of your blender file. Much more handy than having it on a separate application. Be sure to keep it visible when saving! If you are sharing the file with the community and you want to share it under some license you can write the license in a text window.

### Demonstration

#### Exercise

Copy the text below in the Text Editor.

```
import bpy
from math import radians, cos, sin

# An object can exist in 20 layers,
# so the following code determines on which layers you want it to be

# Get the cursor's location
cursor = bpy.context.scene.cursor_location

# Radius of the circle
radius = 5

# Space the cubes around the circle. Default is 36 degrees apart
# Get a list of angles converted to radians

anglesInRadians = [radians(degree) for degree in range(0, 360, 36)]

# Loop through the angles, determine x,y using polar coordinates
# and create object
for theta in anglesInRadians:
    x = cursor.x + radius * cos(theta)
    y = cursor.y + radius * sin(theta)
    z = cursor.z
    bpy.ops.mesh.primitive_cube_add(location=(x, y, z))
```

Execute the script with the *Run Script* button.

You can see the result of running the above script in this video.



Fig. 2.2688: Text Toolbar with a file open

## 2.13.7 Introduction

Since we are working with new and improving Python API, if you have something that needs to be answered, please add it here. We will find answers from dev's if we do not know them and provide an answer here.

### Geometry

#### How can I generate a mesh object using the API?

Download this code example `Script_GeneratePyramidMesh.py` and run it from the Text Window.

#### How do I apply a modifier using the API?

```
bpy.ops.object.convert(target='MESH', keep_original=False)
```

All the modifiers in the stack will be applied.

In case you just want to apply only the subsurf modifier and leave others alone, and create a new mesh (Old mesh will retain all its modifiers), the following code shows one way of doing it.

```
for modifier in bpy.context.object.modifiers:
    if modifier.type != 'SUBSURF':
        modifier.show_render=True
bpy.ops.object.convert(target='MESH', '''keep_original=True''')
```

#### How do I get the world coordinates of a control vertex of a BezierCurve?

```
wmtx = bpy.context.active_object.matrix_world

localCoord = bpy.context.active_object.data.splines[0].bezier_points[1].co

worldCoord = wmtx * localCoord
```

[More info...](#)

#### How do I select/deselect the control points of a Curve

##### Method 1

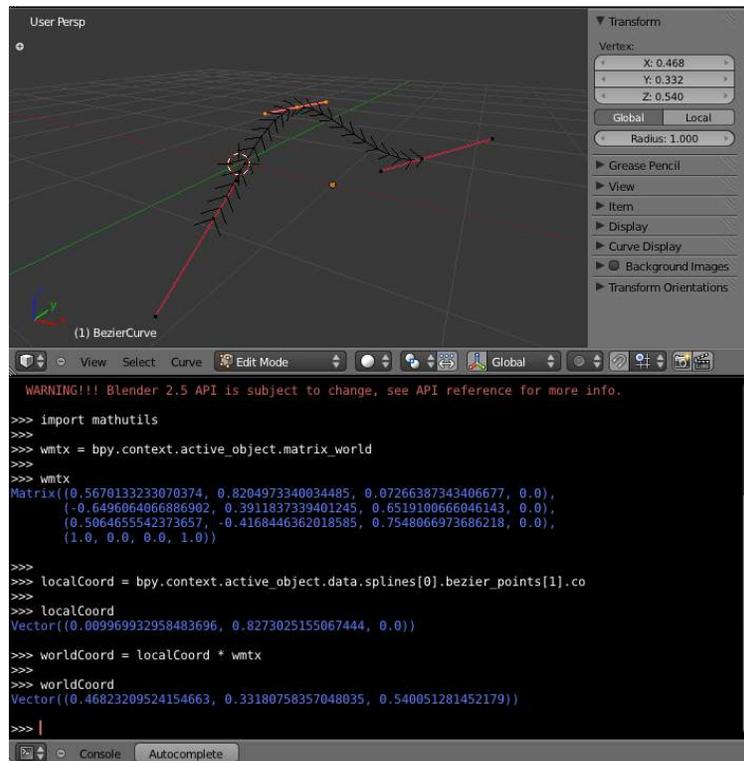
```
curve = bpy.context.selected_objects[0]

curve.data.splines[0].bezier_points[0].select_control_point = True
curve.data.splines[0].bezier_points[2].select_control_point = True
```

##### Method 2

```
bpy.context.active_object.data.splines[0].bezier_points[0].select_control_point = True
```

[More info...](#)



## Materials

### How to link a mesh/object to a material?

TODO

## Customization

### How do I automate custom hotkeys?

## 2.13.8 Scripting & Security

The ability to include Python scripts within blend files is valuable for advanced tasks such as rigging, automation and using the game-engine, however it poses a security risk since Python doesn't restrict what a script can do.

Therefore, you should only run scripts from sources you know and trust.

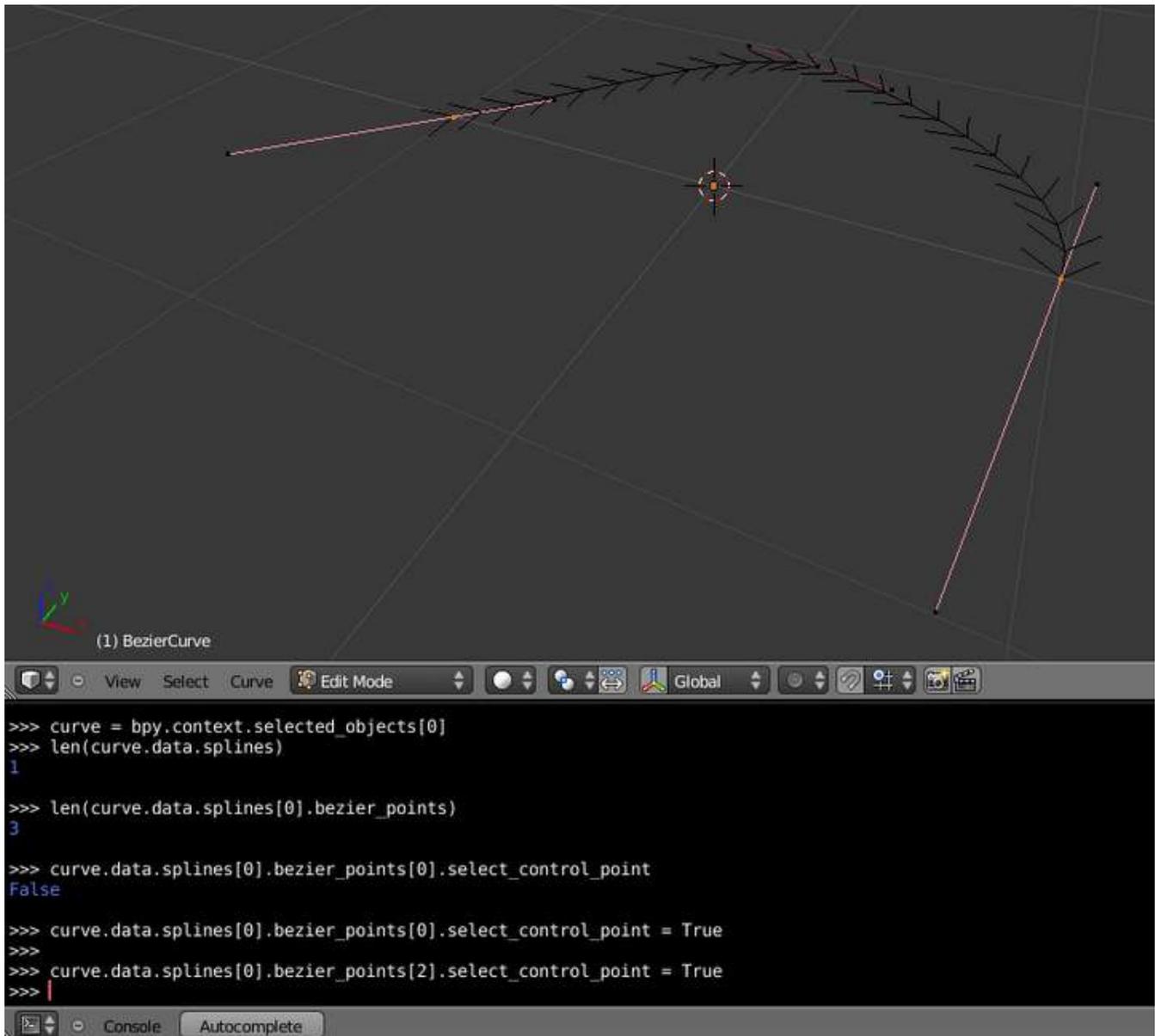
Automatic execution is disabled by default, however some blend files need this to function properly.

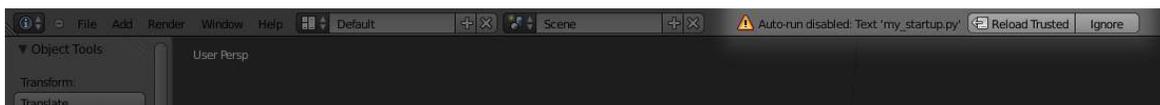
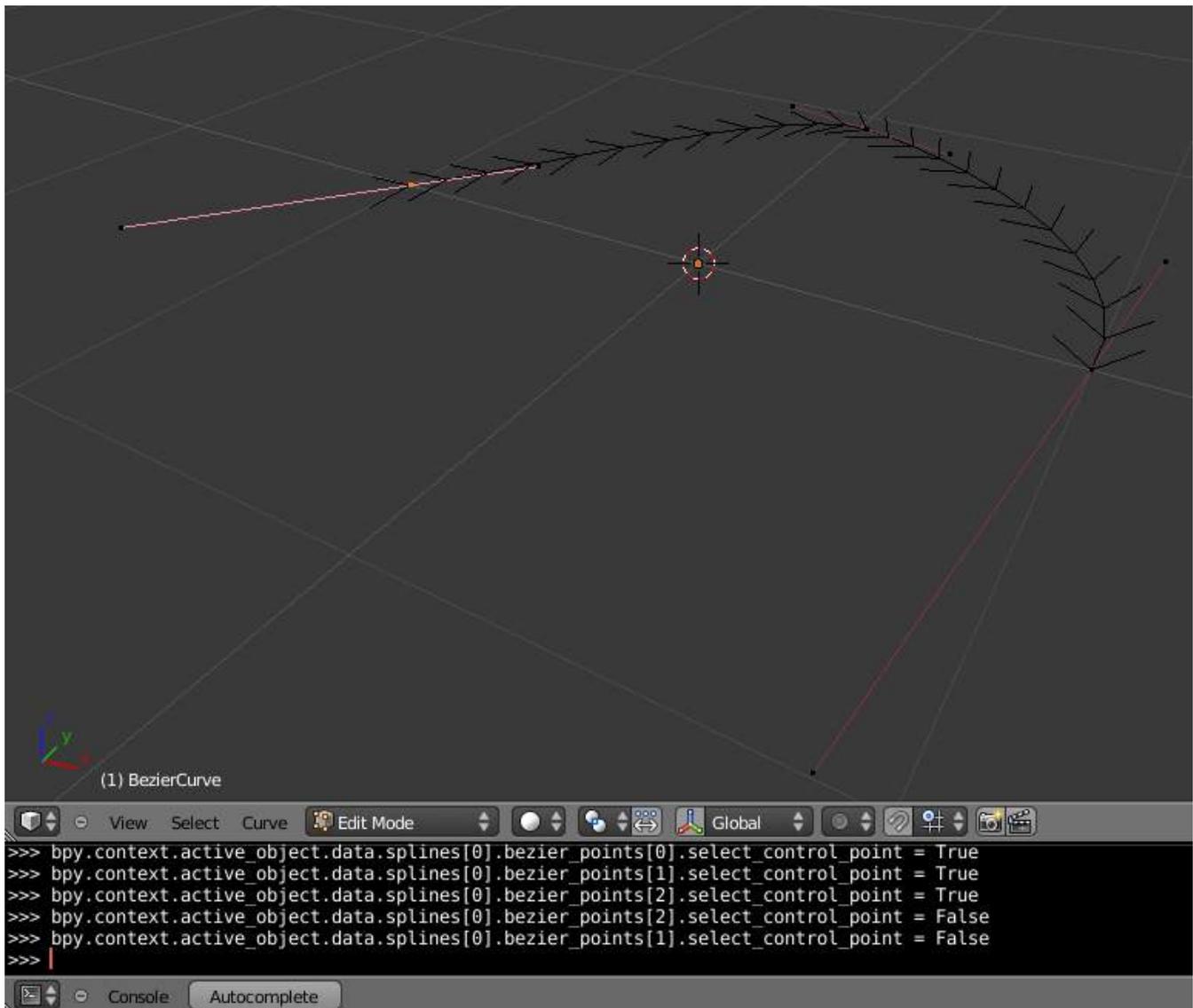
When a blend file tries to execute a script and is not allowed, a message will appear in the header with the option to **Reload Trusted** or **Ignore** the message.

## Scripts in Blend Files

### Auto Execution

Here are the different ways blend files may automatically run scripts.





- Registered Text-Blocks

A text block can have its **Register** option enabled which means it will load on start.

- Animation Drivers

Python expressions can be used to **drive** values and are often used in more advanced rigs and animations.

- Game Engine Auto-Start

scripts are often used for game logic, blend files can have auto-start enabled with runs the game on load.

## Manual Execution

There are other ways scripts in a `blend` file may execute that require user interaction (therefor will run even when auto-execution is off), but you should be aware that this is the case since it's not necessarily obvious.

- Running a script in the text editor (*ok, this is obvious!*).
- Rendering with FreeStyle - *FreeStyle uses scripts to control line styles*
- Running the Game-Engine.

## Controlling Script Execution

Blender provides a number of ways to control whether scripts from a blend file are allowed to automatically execute.

First of all, the file-selector has the option **Trusted Source** which you can use on a case-by-case basis to control auto-execution.

However you may forget to set this, or open a file without going through the file selector - so you can change the default (described next).

## Setting Defaults

In the **File** section of the user-preferences there is the toggle **Auto-Run Python Scripts**.

This means the **Trusted Source** option in the file-selector will be enabled by default, and scripts can run when blend files are loaded without using the file selector.

Once enabled you have the option to exclude certain directories, a typical configuration would be to trust all paths except for the download directory.

## Command Line

You may want to perform batch rendering or some other task from the command line - running Blender without an interface.

In this case the user-preferences are still used but you may want to override them.

- Enable with `-y` or `--enable-autoexec`
- Disable with `-Y` or `--disable-autoexec`

Example - rendering an animation in background mode, allowing drivers and other scripts to run:

```
blender --background --enable-autoexec my_movie.blend --render-anim
```

---

**Note:** These command line arguments can be used to start a regular blender instance and will still override the user-preferences.

---



### 2.13.9 Blender's Python API

The full Python API (Application Programmer Interface) of Blender is documented here:

Latest API - may be newer than current stable release!

Specific versions:

- 2.64 API
- 2.63 API
- 2.62 API
- 2.61 API
- 2.60a API
- 2.59 API
- 2.58 API
- 2.57 API
- 2.56 API

### Scripts

There are more than one hundred different scripts for Blender available on the net.

As with plugins, scripts are very dynamic, changing interface, functionalities and web location fairly quickly, so for an updated list and for a live link to them please refer to one of the two main Blender sites:

- [www.blender.org](http://www.blender.org)
- [www.blenderartists.org](http://www.blenderartists.org)
- Python extensions on this wiki.

## 2.14 Preferences

### 2.14.1 Introduction

This chapter explains how to change Blender's default configuration with the *User Preferences* editor.

The Blender *User Preferences* editor contains settings to control how Blender behaves.

#### Open User Preferences

To open a Blender *User Preferences* editor go to *File* → *User Preferences*.



#### Configure

Now that you have opened the User Preferences editor, you can configure Blender to your liking. At the top of the window, the available options are grouped into seven tabs:

**Interface** Change how UI elements are displayed and how they react.

**Editing** Control how several tools will interact with your input.

**Input** Customize how Blender reacts to the mouse and keyboard as well as define your own keymap.

**Add-ons** Manage Blender's *Add-ons*, allowing you to access features not built-in as well as install new features.

**Themes** Customize interface appearance and colors.

**File** Configure auto-save preferences and set default file paths for .blend files, rendered images, and more.

**System** Set resolution, scripting console preferences, sound, graphics cards, and internationalization.

### Save the new preferences

Once you have set your preferences, you will need to manually save them, otherwise the new configuration will be lost after a restart. Blender saves its preferences to *userpref.blend* in your user folder (see next section, “Load Factory Settings”, for details).

In the *User Preferences* window, click on the *Save User Settings* button in the bottom left. This will save all of the new preferences.

### Load Factory Settings

Go to *File* → *Load Factory Settings* then save the preferences via the *User Preferences* editor.

**Hint:** It can be valuable to make a backup of your preferences the event that you lose your configuration.

See the [directory layout](#) section to see where your preferences are stored.

## 2.14.2 Interface

Interface configuration lets you change how UI elements are displayed and how they react.



## Display

**Tooltips** When enabled, a tooltip will appear when your mouse pointer is over a control. This tip explains the function of what's under the pointer, gives the associated hotkey (if any) and the Python function that refers to it.

**Object Info** Display the active Object name and frame number at the bottom left of the 3D view.

**Large Cursors** Use large mouse cursors when available.

**View Name** Display the name and type of the current view in the top left corner of the 3D window. For example: *User Persp* or *Top Ortho*.

**Playback FPS** Show the frames per second screen refresh rate while an animation is played back. It appears in the viewport corner, displaying red if the frame rate set can't be reached.

**Global Scene** Forces the current scene to be displayed in all screens (a project can consist of more than one scene).

**Object Origin Size** Diameter of 3D Object centers in the view port (value in pixels from 4 to 10).

**Display Mini Axis** Show the mini axis at the bottom left of the viewport.

**Size** Size of the mini axis.

**Brightness** Adjust brightness of the mini axis.

## View manipulation

**Cursor Depth** Use the depth under the mouse when placing the cursor.

**Auto Depth** Use the depth under the mouse to improve view pan/rotate/zoom functionality.

**Zoom to Mouse Position** When enabled, the mouse pointer position becomes the focus point of zooming instead of the 2D window center. Helpful to avoid panning if you are frequently zooming in and out.

**Rotate Around Selection** The selected object becomes the rotation center of the viewport.

**Global Pivot** Lock the same rotation/scaling pivot in all 3D views.

**Auto Perspective** Automatically to perspective Top/Side/Front view after using User Orthographic. When disabled, Top/Side/Front views will retain Orthographic or Perspective view (whichever was active at the time of switching to that view).

**Smooth View** Length of time the animation takes when changing the view with the numpad (Top/Side/Front/Camera...). Reduce to zero to remove the animation.

**Rotation Angle** Rotation step size in degrees, when Numpad4, Numpad6, Numpad8, or Numpad2 are used to rotate the 3D view.

## 2D Viewports

**Minimum Grid Spacing** The minimum number of pixels between grid lines in a 2D (i.e. top orthographic) viewport.

**TimeCode Style** Format of Time Codes displayed when not displaying timing in terms of frames. The format uses '+' as separator for sub-second frame numbers, with left and right truncation of the timecode as necessary.

## Manipulator

Permits configuration of the 3D transform manipulator which is used to drag, rotate and resize objects (Size, Handle size).

## Menus

**Open on Mouse Over** Select this to have the menu open by placing the mouse pointer over the entry instead of clicking on it.

**Menu Open Delay** Time for the menu to open.

**Top Level** Time delay in 1/10 second before a menu opens (*Open on Mouse Over* needs to be enabled).

**Sub Level** Same as above for sub menus (for example: *File* → *Open Recent*).

## 2.14.3 Editing

These preferences control how several tools will interact with your input.



### Link Materials To



Fig. 2.2689: Example for a Mesh

To understand this option properly, you need to understand how Blender works with Objects. Almost everything in Blender is organized in a hierarchy of Datablocks. A Datablock can be thought of as containers for certain pieces of information. For example, the Object Datablock contains information about the Object's location while the Object Data (*ObData*) datablock contains information about the mesh.

A material may be linked in two different ways:



Fig. 2.2690: A material linked to ObData (left) and Object (right).

**ObData** Any created material will be created as part of the ObData datablock.

**Object** Any created material will be created as part of the Object datablock.

[Read more about Blender's Data System](#)

## New objects

**Enter Edit Mode** If selected, Edit Mode is automatically activated when you create a new object.

### Align To

**World** New objects align with world coordinates.

**View** New object align with view coordinates.

## Undo

**Global Undo** This enables Blender to save actions done when you are **not** in *Edit Mode*. For example, duplicating Objects, changing panel settings or switching between modes.

**Step** Number of Undo steps available.

**Memory Limit** Maximum memory usage in Mb (0 is unlimited).

[Read more about Undo and Redo options](#)

## Grease Pencil

Grease Pencil permits you to draw in the 3D viewport with a pencil-like tool.

**Manhattan Distance** The minimum number of pixels the mouse has to move horizontally or vertically before the movement is recorded.

**Euclidian Distance** The minimum distance that mouse has to travel before movement is recorded.

**Eraser Radius** The size of the eraser used with the grease pencil.

**Smooth Stroke** Smooths the pencil stroke after it's finished.

## Playback

**Allow Negative Frame** If set, negative framenumbers might be used.

## Keyframing

In many situations, animation is controlled by keyframes. The state of a value (e.g. location) is recorded in a keyframe and the animation between two keyframes is interpolated by Blender.

**Visual Keying** Use Visual keying automatically for constrained objects.

**Only Insert Needed** When enabled, new keyframes will be created only when needed.

**Auto Keyframing** Automatic keyframe insertion for Objects and Bones. Auto Keyframe is not enabled by default.

**Only Insert Available** Automatic keyframe insertion in available curves.

### New F-Curve Defaults

**Interpolation** This controls how the state between two keyframes is computed. Default interpolation for new keyframes is Bezier which provides smooth acceleration and de-acceleration whereas Linear or Constant is more abrupt.

**XYZ to RGB** Color for X, Y or Z animation curves (location, scale or rotation) are the same as the color for the X, Y and Z axis.

## Transform

**Release confirm** Dragging LMB on an object will move it. To confirm this (and other) transforms, a LMB is necessary by default. When this option is activated, the release of LMB acts as confirmation of the transform.

## Sculpt Overlay Color

This color selector allows the user to define a color to be used in the inner part of the brushes circle when in sculpt mode, and it is placed as an overlay to the brush, representing the focal point of the brush influence. The overlay color is visible only when the overlay visibility is selected (clicking at the *eye* to set its visibility), and the transparency of the overlay is controlled by the alpha slider located at the brush selector panel, located at the top of the tool shelf, when in sculpt mode.

## Duplicate Data

The ‘Duplicate Data’ check-boxes define what data is copied with a duplicated Object and what data remains linked. Any boxes that are checked will have their data copied along with the duplication of the Object. Any boxes that are not checked will instead have their data linked from the source Object that was duplicated.

For example, if you have Mesh checked, then a full copy of the mesh data is created with the new Object, and each mesh will behave independently of the duplicate. If you leave the mesh box unchecked then when you change the mesh of one object, the change will be mirrored in the duplicate Object.

The same rules apply to each of the check-boxes in the ‘Duplicate Data’ list.

## 2.14.4 Input

In the Input preferences, you can customize how Blender reacts to the mouse and keyboard as well as define your own keymap.

### Managing presets

Blender lets you define multiple *Preset* input configurations. Instead of deleting the default keymap to create yours, you can just add new *Presets* for both the mouse and keyboard. Mouse options can be found on the left hand side of the window and keyboard options to the right in the above picture.

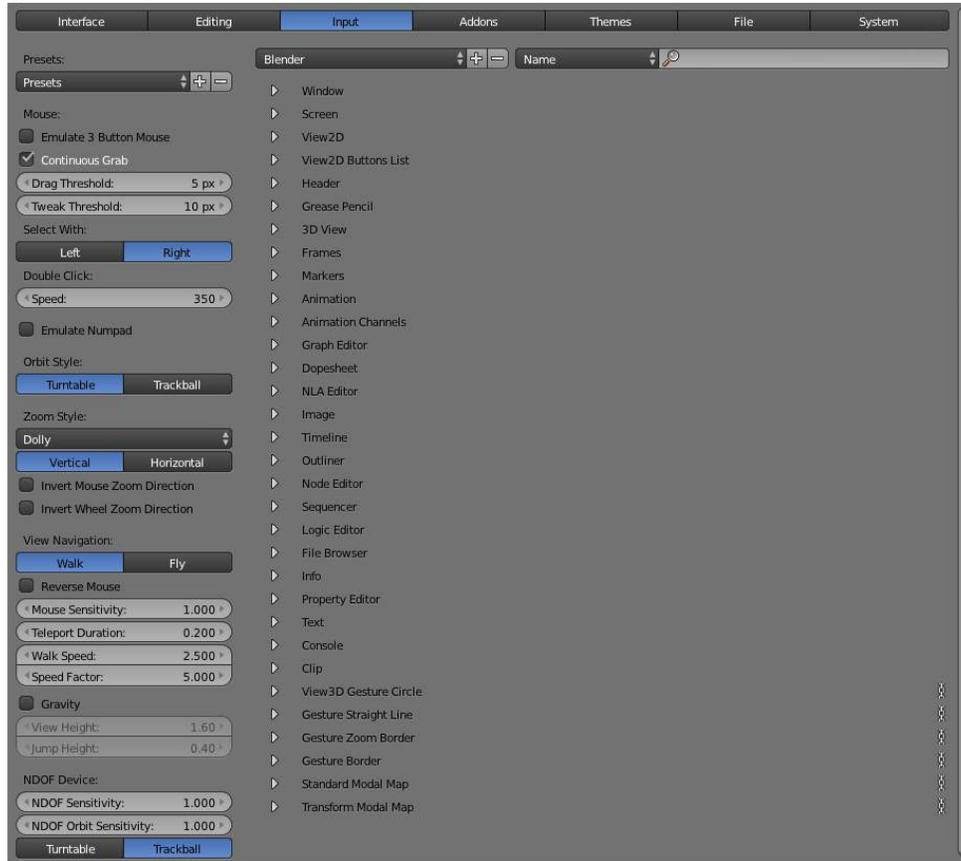
### Adding and deleting presets

Before changing anything in the default configuration, click on the “plus” symbol shown in the picture to add a new *Preset*. Blender will ask you to name your new preset after which you can select the *Preset* from the list to edit it. If you want to delete your *Preset*, select it from the list and then click the “minus” symbol.

### Selecting presets

You can change the preset you are using by doing one of the following:

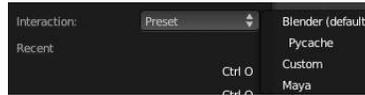
- Selecting the configuration from the *Interaction* menu of the splash screen at startup or by selecting *Help* → *Splash Screen*.



- Selecting the configuration from the *User Preferences Input* window.

**Note:** Note that either of the above options will only change the preset for the current file. If you select *File* → *New* or *File* → *Open*, the default preset will be re-loaded.

## Setting presets to default



Once you've configured your mouse and keyboard *Preset*s, you can make this the default configuration by:

- Opening the *User Preferences Input* editor and select your presets from the preset list or,
- Selecting your preset configuration from the splash screen.
- Saving your configuration using the *Save As Default* option from a *User Preferences* window or by pressing `Ctrl-U`.

## Export/Import key configuration

In some cases, you may need to save your configuration in an external file (e.g. if you need to install a new system or share your keymap configuration with the community). Simply **LMB** *Export Key Configuration* on the *Input* tab header and a file browser will open so that you can choose where to store the configuration. The *Import Key Configuration* button installs a keymap configuration that is on your computer but not in Blender.

## Mouse

**Emulate 3 Button Mouse** Blender can be configured to work with different mouse types (such as a two-button mouse, Apple single-button Mouse, or laptop touchpad). The functionality of the 3 mouse buttons will then be emulated with key/mousebutton combos as shown in the table below.

Table 2.72: Shortcuts for supported mouse hardware

3-button Mouse	2-button Mouse	Apple Mouse
LMB	LMB	LMB (mouse button)
MMB	Alt-LMB	Cmd-LMB (Option/Alt key + mouse button)
RMB	RMB	Cmd-LMB (Command/Apple key + mouse button)

All the Mouse/Keyboard combinations mentioned in the Manual can be expressed with the combinations shown in the table. For Example, `Shift-Alt-RMB` becomes `Shift-Alt-Cmd-LMB` on a single-button mouse.

**Continuous Grab** This feature is used to prevent the problem where an action such as grabbing or panning a view, is limited by your screen bounds.

This is done by warping the mouse within the view.

**Note:** Cursor warping is only supported by *relative* input devices (mouse, trackball, trackpad).

Graphics tablets however, typically use *absolute* positioning, this feature is disabled when a tablet is being used

This is detected for each action, so the presence of a tablet wont disable continuous-grab for mouse cursor input.

**Drag Threshold** The number of pixels that a User Interface element has to be moved before it is recognized by Blender.

**Select with** You can choose which button is used for selection (the other one is used to place the 3D cursor).

**Double Click** The time for a double click (in ms).

---

**Note:** The Mouse emulate option is only available if *Select With* is set to *Right*.

---

## Graphic Tablets

Graphic tablets can be used to provide a more traditional method of controlling the mouse cursor using a pen. This can help to provide a more familiar experience for artists who are used to painting and drawing with similar tools, as well as provide additional controls such as pressure sensitivity.

---

**Note:** If you are using a graphic tablet instead of a mouse and pressure sensitivity doesn't work properly, try to place the mouse pointer in the Blender window and then unplug/replug your graphic tablet. This might help.

---

## Numpad emulation

The Numpad keys are used quite often in Blender and are not the same keys as the regular number keys. If you have a keyboard without a Numpad (e.g. on a laptop), you can tell Blender to treat the standard number keys as Numpad keys. Just check *Emulate Numpad*.

## View manipulation

**Orbit Style** Select how Blender works when you rotate the 3D view (by default *MMB*). Two styles are available. If you come from Maya or Cinema 4D, you will prefer *Turntable*.

### Zoom Style

#### Choose your preferred style of zooming in and out with *Ctrl*-*MMB*

**Scale** *Scale* zooming depends on where you first click in the view. To zoom out, hold *Ctrl*-*MMB* while dragging from the edge of the screen towards the center. To zoom in, hold *Ctrl*-*MMB* while dragging from the center of the screen towards the edge.

**Continue** The *Continue* zooming option allows you to control the speed (and not the value) of zooming by moving away from the initial click-point with *Ctrl*-*MMB*. Moving up from the initial click-point or to the right will zoom out, moving down or to the left will zoom in. The further away you move, the faster the zoom movement will be. The directions can be altered by the *Vertical* and *Horizontal* radio buttons and the *Invert Zoom Direction* option.

**Dolly** *Dolly* zooming works similarly to *Continue* zooming except that zoom speed is constant.

**Vertical** Moving up zooms out and moving down zooms in.

**Horizontal** Moving left zooms in and moving right zooms out.

**Invert Zoom Direction** Inverts the Zoom direction for *Dolly* and *Continue* zooming.

**Invert Wheel Zoom Direction** Inverts the direction of the mouse wheel zoom.

**NDOF device** Set the sensitivity of a 3D mouse.

## Keymap editor

The Keymap editor lets you change the default Hotkeys. You can change keymaps for each window.

- Select the keymap you want to change and click on the white arrows to open up the keymap tree.



- Select which Input will control the function
  - Keyboard: Only hotkey or combo hotkey (E or Shift-E).
  - Mouse: Left/middle/right click. Can be combined with Alt, Shift, Ctrl, Cmd.
  - Tweak: Click and drag. Can also be combined with the 4 previous keys.
  - Text input: Use this function by entering a text
  - Timer: Used to control actions based on a time period. e.g. By default, Animation Step uses Timer 0, Smooth view uses Timer 1.
- Change hotkeys as you want. Just click on the shortcut input and enter the new shortcut.

If you want to restore the default settings for a keymap, just click on the *Restore* button at the top right of this keymap.

## 2.14.5 Add-ons

The Add-ons tab lets you manage secondary options which are not enabled in Blender by default. New features may be added with *Install Add-ons*. There will be a growing number of such Add-ons, generated by the Blender-community so look out for that one feature you were missing (or maybe simply create it yourself).

See the [Add-ons Page](#) for more on using Add-ons.

## 2.14.6 Themes

The *Themes* tab allows you to customize interface appearance and colors.

The colors for each editor can be set separately - simply select the editor you wish to change in the multi-choice list at the left, and adjust colors as required. Notice that changes appear in real-time on your screen. In addition, details such as the dot size in the *3D View* or the *Graph Editor* can also be changed.

Themes use blenders preset system, you can save a theme to an XML and install it on another system.

## 2.14.7 File Preferences

The *File Preferences* tab allows you to configure auto-save preferences and set default file paths for .blend files, rendered images, and more.

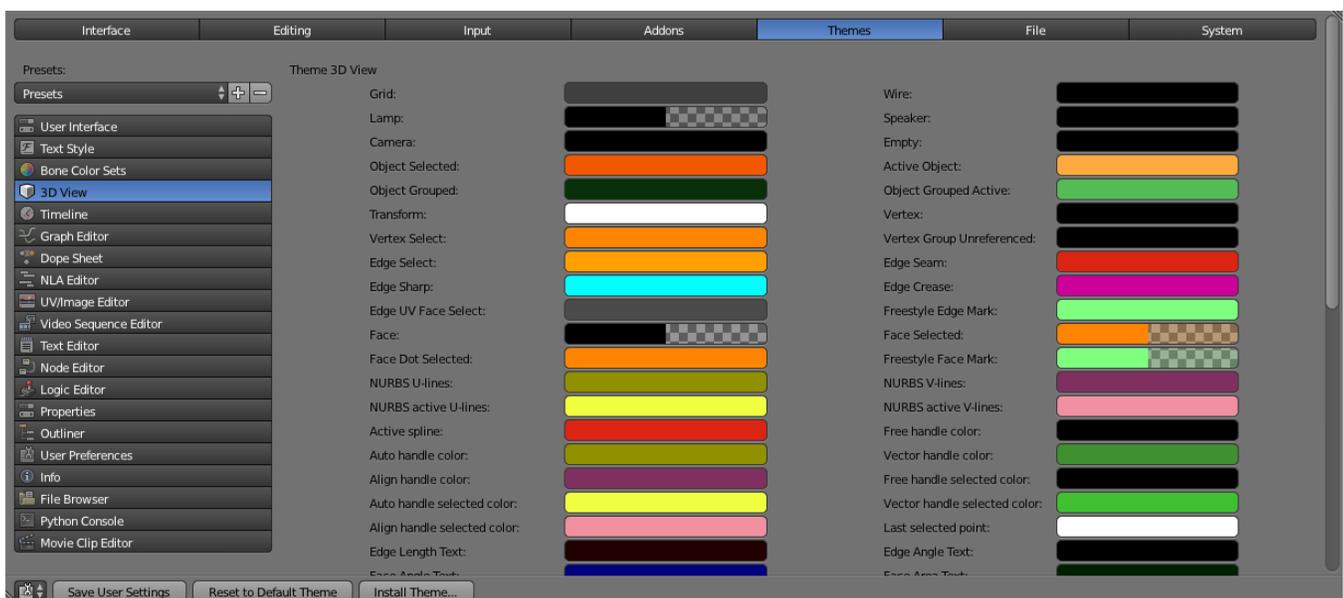
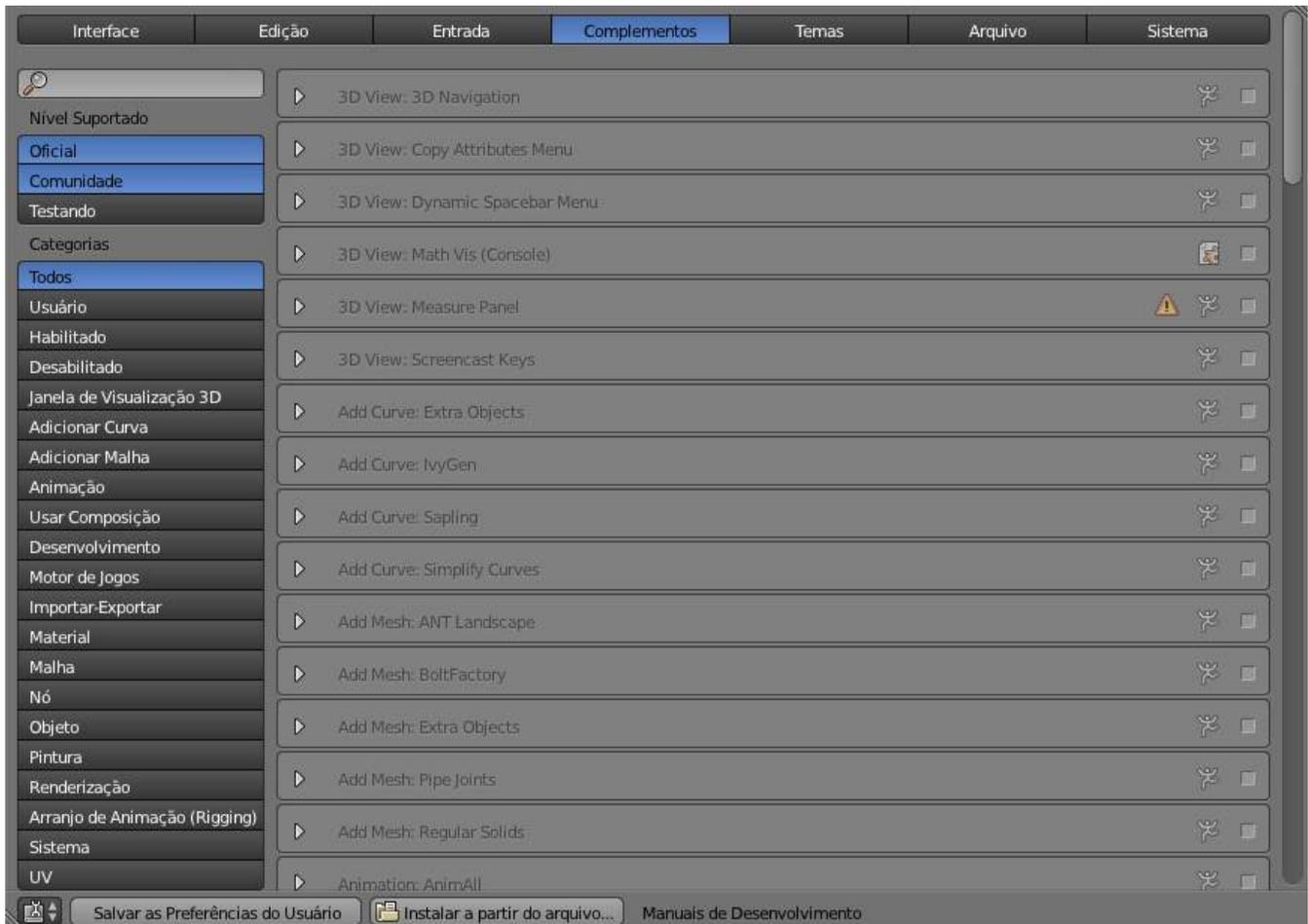
### File Paths

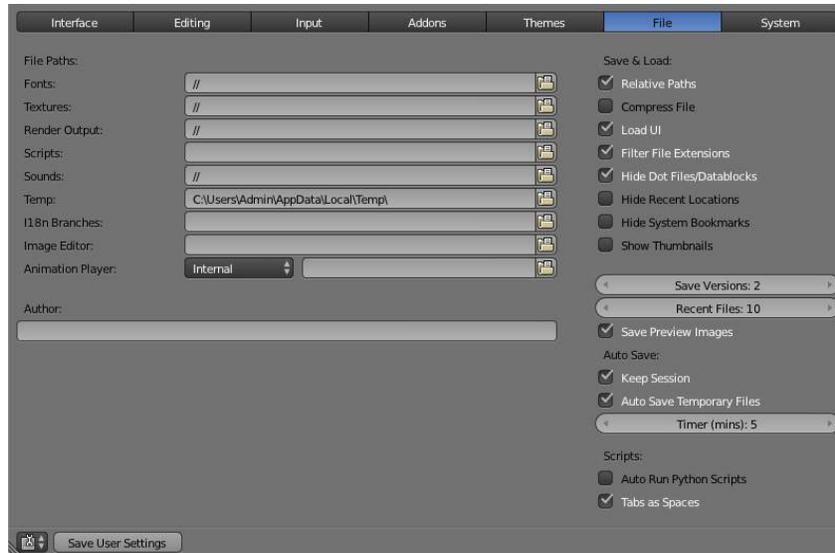
Locations for various external files can be set for the following options:

**Fonts** Default location when searching for font files.

**Textures** Default location when searching for image textures.

**Render Output** Where rendered images/videos are saved.





**Scripts** An additional location to search for Python scripts. See [Scripts Path](#) below.

**Sounds** Default location when searching for sound files.

**Temp** The location where temporary files are stored.

**Render Cache** The location where cached render images are stored.

**118n Branches** The path to the `/branches` directory of your local svn-translation copy, to allow translating from the UI.

**Image Editor** The path to an external program to use for image editing.

**Animation Player** The path to an external program to use for playback of rendered animations.

---

**Note:** If these folders do not exist, they will *not* be created automatically.

---

### Scripts Path

By default Blender looks in several directories (OS dependant) for scripts. By setting a user script path in the preferences an additional directory is looked in. This can be used to store certain scripts/templates/presets independently of the currently used Blender Version.

Inside the specified folder specific folders have to be created to tell Blender what to look for where. This folder structure has to mirror the structure of the scripts folder found in the installation directory of Blender:

- scripts
- add-ons
- modules
- presets
- camera
- cloth
- interface\_theme
- operator

- render
- ...
- startup
- templates Not all of the folders have to be present.

## Auto Execution

Python scripts (including driver expressions) are not executed by default for security reasons.

**Auto Run Python Scripts** You may choose to ignore these security issues and allow scripts to be executed automatically.

**Excluded Paths** Blend files in these folders will *not* automatically run Python scripts. This can be used to define where blend files from untrusted sources are kept.

**See also:**

[Scripting & Security](#)

## Save & Load

**Relative Paths** By default, external files use a [relative path](#).

**Compress File** Compress `.blend` file when saving.

**Load UI** Default setting is to load the Window layout (the [Screens](#)) of the saved file. This can be changed individually when loading a file from the *Open Blender File* panel of the *File Browser* window.

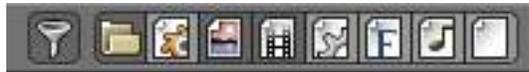


Fig. 2.2691: File extension filter

**Filter File Extensions** By activating this, file dialog windows will only show appropriate files (i.e. `.blend` files when loading a complete *Blender* setting). The selection of file types may be changed in the file dialog window.

**Hide Dot File/Datablocks** Hide file which start with `.` on file browsers (in Linux and Apple systems, `.` files are hidden).

**Hide Recent Locations** Hides the *Recent* panel of the *File Browser* window which displays recently accessed folders.

**Show Thumbnails** Displays a thumbnail of images and movies when using the *File Browser*.

## Auto Save

**Save Versions** Number of versions created for the same file (for backup).

**Recent Files** Number of files displayed in *File* → *Open Recent*.

**Save Preview Images** Previews of images and materials in the *File Browser* window are created on demand. To save these previews into your `.blend` file, enable this option (at the cost of increasing the size of your `.blend` file).

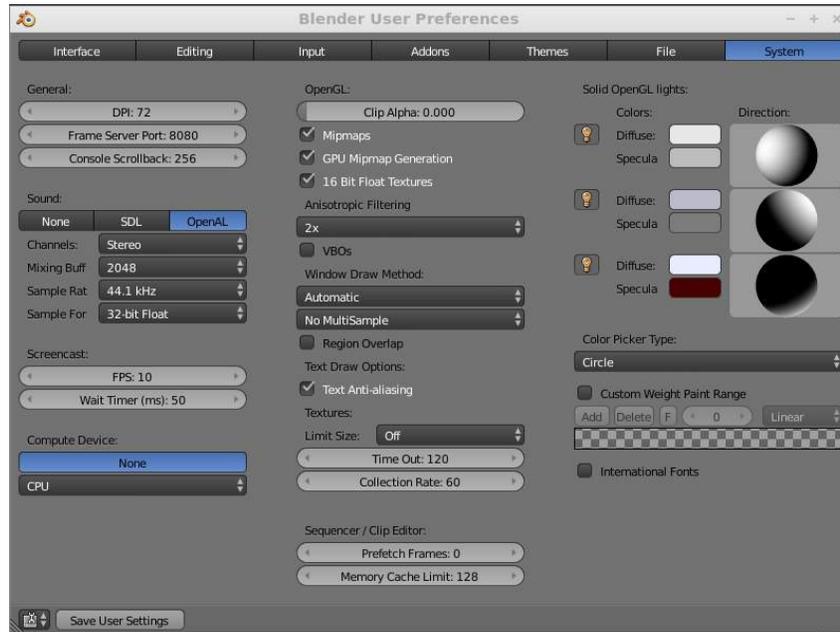
**Auto Save Temporary File** Enable Auto Save (create a temporary file).

**Timer** Time to wait between automatic saves.

Read more about Auto Save options

## 2.14.8 System preferences

The *System* tab allows you to set resolution, scripting console preferences, sound, graphics cards, and internationalization.



### General

**DPI** Value of the screen resolution which controls the size of Blender's interface fonts and internal icons shown. Useful for taking screen shots for book printing and use of high resolution monitors, (such as Retina Display and others) matching the screen DPI with Blender DPI. During normal usage, most Blender users might prefer to zoom screen elements pressing `Ctrl` and dragging `MMB` left and right over a panel to resize its contents, or use the `NumpadPlus` and `NumpadMinus` to zoom in and out the contents. Pressing `Home` (Show All) will reset the zooming at the screen/panel focused by the mouse pointer.

**Frame Server Port** TCP/IP port used in conjunction with the IP Address of the machine for frameserver rendering. Used when working with distributed rendering. Avoid changing this port value unless it is conflicting with already existing service ports used by your Operating System and/or softwares. Always consult your operating system documentation and services or consult your system administrator before changing this value.

**Console Scrollback** The number of lines, buffered in memory of the console window. Useful for debugging purposes and command line rendering.

### Sound

**Sound** Set the audio output device or no audio support. There are 3 Options:

**None** No Audio support (no audio output, audio strips can be loaded normally)

**SDL** Uses Simple Direct Media Layer API from [libsdl.org](http://libsdl.org) to render sounds directly to the sound device output. Very useful for sequencer strips editing.

**OpenAL** Uses OpenAL soft API for Linux and OpenAL from creative Labs for Windows. This API provides buffered sound rendering with 3D/spatial support. Useful for the BGE Games.

'Specific sound options' (With *SDL* or *OpenAL* enabled)

**Channels** Set the audio channel count. Available options are: *Stereo* (Default) , *4 Channels* , *5.1 Surround* , *7.1 Surround*

### Mixing Buffer

**Set the number of samples used by the audio mixing buffer. Available options are:** *512* , *1024* , *2048* (Default), *4096* , *8192*, *16384*, and *32768*

**Sample Rate** Set the audio sample rate. Available options are: *44.1 Khz* (Default), *48 Khz* , *96 Khz* and *192Khz*

**Sample Format** Set the audio sample format. Available options are: *32 bit float* (Default), *8 bit Unsigned* , *16 Bits Signed* , *24 Bits Signed* , *32 Bits Signed* , *32 Bits Float* and *64 Bits Float*

## Screencast

TODO

## Compute Device

The Options here will set the compute device used by the Cycles render engine.

**None** When set to *None* or the only option is *None*: your CPU will be used as a computing device for Cycles Render Engine

**CUDA** If the system has a compatible CUDA enabled graphics card and appropriate device drivers installed. When one or both of the options are available, the user will be able to choose whether to use CPU or other computing device for Cycles Rendering.

**OpenCL** Note that this currently has limited support unsupported, see: [Cycles Render engine page](#)

## Open GL

**Clip Alpha** Clip alpha below this threshold in the 3D viewport. Minimum: **0.000** (No Clip) , Maximum: **1.000** , Default **0.000** (No Clip)

**Mipmaps** Scale textures for 3D view using mipmap filtering. This increases display quality, but uses more memory.

**GPU MipMap Generation** Generate MipMaps on the GPU. Offloads the CPU Mimpap generation to the GPU.

**16 Bit Float Textures** Enables the use of 16 Bit per component Texture Images (Floating point Images).

**Anisotropic Filtering** Set the level of anisotropic filtering. Available Options are: *Off*” (*No Filtering*) , *2x* (Default) , *4x* , *8x* , *16x*

**VBOs** Use Vertex Buffer Objects, or vertex arrays if unsupported, for viewport rendering. Helps to speed up viewport rendering by allowing vertex array data to be stored in Graphics card memory.

## Window Draw Method

**Window Draw Method** Specifies the Window Draw Method used to display Blender Window(s).

**Automatic (Default)** Automatically set based on graphics card and driver.

**Triple Buffer** Use a third buffer for minimal redraws at the cost of more memory. If you have a capable GPU, this is the best and faster method of redraw.

**Overlap** Redraw all overlapping regions. Minimal memory usage, but more redraws. Recommended for some graphics cards and drivers combinations.

**Overlap Flip** Redraw all overlapping regions. Minimal memory usage, but more redraws (for graphics drivers that do flipping). Recommended for some graphic cards and drivers combinations.

**Full** Do a full redraw each time. Only use for reference, or when all else fails. Useful for certain cards with bad to no OpenGL acceleration at all.

**Multi-Sampling** This enables *FSAA* for smooth drawing, at the expense of some performance.

---

**Note:** This is known to cause selection issues on some configurations, see: *Invalid Selection*.

---

**Region Overlap** This checkbox will enable Blender to draw regions overlapping the 3D Window. It means that the Object Tools and Transform Properties Tab, which are opened by using the shortcuts T and N will be drawn overlapping the 3D View Window.

If you have a capable graphics card and drivers with *Triple Buffer* support, clicking the checkbox will enable the overlapping regions to be drawn using the *Triple Buffer* method, which will also enable them to be drawn using Alpha, showing the 3D View contents through the Object Tools and Transform Properties Tab.

## Text Draw Options

**Text Draw Options** Enable interface text anti-aliasing. When disabled, texts are drawn using text straight render (Filling only absolute Pixels). Default: Enabled.

## Textures

**Limit Size** Limit the maximum resolution for pictures used in textured display to save memory. The limit options are specified in a square of pixels, (e.g.: the option 256 means a texture of 256x256 pixels) This is useful for game engineers, whereas the texture limit matches paging blocks of the textures in the target graphic card memory. Available Options are: *Off* (No limit - Default) , *128*, *256*, *512*, *1024*, *2048*, *4096*, *8192*.

**Time Out** Time since last access of a GL texture in seconds, after which it is freed. Set to 0 to keep textures allocated. Minimum: **0** , Maximum: **3600** , Default: **120**

**Collection Rate** Number of seconds between each run of the GL texture garbage collector. Minimum: **0** , Maximum: **3600** , Default: **120**

## Sequencer/Clip Editor

**Prefetch Frames** Number of frames to render ahead during playback. Useful when the chosen video codec cannot sustain screen frame rates correctly using direct rendering from the disk to video. during video playbacks or editing operations. Minimum: **0** , Maximum: **500** , Default: **0** (No prefetch)

**Memory Cache Limit** Upper limit of the sequencer's memory cache (megabytes). For optimum clip editor and sequencer performance, high values are recommended. Minimum: **0** (No cache) , Maximum: **1024** (1 Gigabyte) , Default: **128**

## Solid OpenGL lights

*Solid OpenGL Lights* are used to light the 3D Window, mostly during *Solid view*. Lighting is constant and position “world” based. There are three virtual light sources, also called OpenGL auxiliary lamps, used to illuminate 3D View scenes, which will not display in renders.

The Lamp Icons allows the user to enable or disable OpenGL Lamps. At least one of the three auxiliary OpenGL Lamps must remain enabled for the 3D View. The lamps are equal, their difference is their positioning and colors. You can control the direction of the lamps, as well as their diffuse and specular colors. Available Options are:

**Direction** Clicking with LMB in the sphere and dragging the mouse cursor let's the user change the direction of the lamp by rotating the sphere. The direction of the lamp will be the same as shown at the sphere surface.

**Diffuse** This is the constant color of the lamp. Clicking on the color widget, opens the color picker mini window and allows the user to change colors using the color picker.

**Specular** This is the highlight color of the lamp Clicking on the color widget, opens the color picker mini window and allows the user to change colors using the color picker.

### Color Picker Type

Choose which type of color dialog you prefer - it will show when clicking LMB on any color field.

See the different color picker types at the [Extended Controls](#) page.

### Custom Weight Paint Range

*Mesh skin weighting* is used to control how much a bone deforms the mesh of a character. To visualize and paint these weights, Blender uses a color ramp (from blue to green, and from yellow to red). Enabling the checkbox will enable an alternate map using a ramp starting with an empty range. Now you can create your custom map using the common color ramp options. For detailed information about how to use color ramps, see: to the [Extended Controls](#) page.

### International Fonts

Blender supports a wide range of languages, enabling this check box will enable Blender to support International Fonts. International fonts can be loaded for the User Interface and used instead of Blender default bundled font.

This will also enable options for translating the User Interface through a list of languages and Tips for Blender tools which appears whenever the user hovers a mouse over Blender tools.

Blender supports I18N for internationalization. For more Information on how to load International fonts, see: [Editing Texts](#) page.

## 2.15 Advanced

This chapter covers advanced use (topics which may not be required for typical usage).

### 2.15.1 Command Line Arguments

Blender 2.75 Usage: blender [args ...] [file] [args ...]

#### Render Options

- b, --background** Run in background (often used for UI-less rendering)
- a, --render-anim** Render frames from start to end (inclusive)
- S, --scene <name>** Set the active scene <name> for rendering
- f, --render-frame <frame>** Render frame <frame> and save it. +<frame> start frame relative, -<frame> end frame relative.
- s, --frame-start <frame>** Set start to frame <frame> (use before the -a argument)

- e, --frame-end <frame>** Set end to frame <frame> (use before the -a argument)
- j, --frame-jump <frames>** Set number of frames to step forward after each rendered frame
- o, --render-output <path>** Set the render path and file name. Use // at the start of the path to render relative to the blend file.

The # characters are replaced by the frame number, and used to define zero padding. \* ani\_##\_test.png becomes ani\_01\_test.png \* test-#####.png becomes test-000001.png

When the filename does not contain #, The suffix #### is added to the filename.

The frame number will be added at the end of the filename, eg:

```
blender -b foobar.blend -o //render_ -F PNG -x 1 -a
```

//render\_ becomes //render\_####, writing frames as //render\_0001.png

- E, --engine <engine>** Specify the render engine use -E help to list available engines
- t, --threads <threads>** Use amount of <threads> for rendering and other operations [1-64], 0 for systems processor count.

## Format Options

- F, --render-format <format>**

**Set the render format, Valid options are...** TGA IRIS JPEG MOVIE IRIZ RAWTGA AVIRAW AVIJPEG PNG BMP FRAMESERVER

**(formats that can be compiled into blender, not available on all systems)** HDR TIFF EXR MULTILAYER MPEG AVICODEC QUICKTIME CINEON DPX DDS

- x, --use-extension <bool>** Set option to add the file extension to the end of the file

## Animation Playback Options

- a <options> <file(s)>**

**Playback <file(s)>, only operates this way when not running in background.** -p <sx> <sy> Open with lower left corner at <sx>, <sy> -m Read from disk (Don't buffer) -f <fps> <fps-base> Specify FPS to start with -j <frame> Set frame step to <frame> -s <frame> Play from <frame> -e <frame> Play until <frame>

## Window Options

- w, --window-border** Force opening with borders (default)
- W, --window-borderless** Force opening without borders
- p, --window-geometry <sx> <sy> <w> <h>** Open with lower left corner at <sx>, <sy> and width and height as <w>, <h>
- con, --start-console** Start with the console window open (ignored if -b is set), (Windows only)
- no-native-pixels** Do not use native pixel size, for high resolution displays (MacBook Retina)

## Game Engine Specific Options

### -g Game Engine specific options

<b>-g fixedtime</b>	Run on 50 hertz without dropping frames
<b>-g vertexarrays</b>	Use Vertex Arrays for rendering (usually faster)
<b>-g nomipmap</b>	No Texture Mipmapping
<b>-g linearmipmap</b>	Linear Texture Mipmapping instead of Nearest (default)

## Python Options

- y, --enable-autoexec** Enable automatic Python script execution
- Y, --disable-autoexec** Disable automatic Python script execution (pydrivers & startup scripts), (default).
- P, --python <filename>** Run the given Python script file
- python-text <name>** Run the given Python script text block
- python-expr <expression>** Run the given expression as a Python script
- python-console** Run blender with an interactive console
- addons** Comma separated list of addons (no spaces)

## Debug Options

- d, --debug** Turn debugging on
  - Prints every operator call and their arguments
  - Disables mouse grab (to interact with a debugger in some cases)
  - Keeps Python's `sys.stdin` rather than setting it to None
- debug-value <value>** Set debug value of <value> on startup
- debug-events** Enable debug messages for the event system
- debug-ffmpeg** Enable debug messages from FFMpeg library
- debug-handlers** Enable debug messages for event handling
- debug-libmv** Enable debug messages from libmv library
- debug-cycles** Enable debug messages from Cycles
- debug-memory** Enable fully guarded memory allocation and debugging
- debug-jobs** Enable time profiling for background jobs.
- debug-python** Enable debug messages for Python
- debug-depsgraph** Enable debug messages from dependency graph
- debug-depsgraph-no-threads** Switch dependency graph to a single threaded evaluation
- debug-gpumem** Enable GPU memory stats in status bar
- debug-wm** Enable debug messages for the window manager
- debug-all** Enable all debug messages (excludes libmv)

- debug-fpe** Enable floating point exceptions
- disable-crash-handler** Disable the crash handler

### Misc Options

- factory-startup** Skip reading the startup.blend in the users home directory
- env-system-datafiles** Set the BLENDER\_SYSTEM\_DATAFILES environment variable
- env-system-scripts** Set the BLENDER\_SYSTEM\_SCRIPTS environment variable
- env-system-python** Set the BLENDER\_SYSTEM\_PYTHON environment variable
- nojoystick** Disable joystick support
- noglsl** Disable GLSL shading
- noaudio** Force sound system to None
- setaudio** Force sound system to a specific device NULL SDL OPENAL JACK
- h, --help** Print this help text and exit
- R** Register .blend extension, then exit (Windows only)
- r** Silently register .blend extension, then exit (Windows only)
- v, --version** Print Blender version and exit
- Ends option processing, following arguments passed unchanged. Access via Python's `sys.argv`

### Other Options

- /?** Print this help text and exit (windows only)
- debug-freestyle** Enable debug/profiling messages from Freestyle rendering
- disable-abort-handler** Disable the abort handler
- enable-new-depsgraph** Use new dependency graph
- verbose <verbose>** Set logging verbosity level.

### Experimental features

- enable-new-depsgraph** Use new dependency graph

### Argument Parsing

Arguments must be separated by white space, eg:

```
blender -ba test.blend
```

...will ignore the a

```
blender -b test.blend -f8
```

...will ignore 8 because there is no space between the `-f` and the frame value

## Argument Order

Arguments are executed in the order they are given. eg:

```
blender --background test.blend --render-frame 1 --render-output '/tmp'
```

...will not render to /tmp because `--render-frame 1` renders before the output path is set

```
blender --background --render-output /tmp test.blend --render-frame 1
```

...will not render to /tmp because loading the blend file overwrites the render output that was set

```
blender --background test.blend --render-output /tmp --render-frame 1
```

...works as expected.

## Environment Variables

**BLENDER\_USER\_CONFIG** Directory for user configuration files.

**BLENDER\_USER\_SCRIPTS** Directory for user scripts.

**BLENDER\_SYSTEM\_SCRIPTS** Directory for system wide scripts.

**BLENDER\_USER\_DATAFILES** Directory for user data files (icons, translations, ..).

**BLENDER\_SYSTEM\_DATAFILES** Directory for system wide data files.

**BLENDER\_SYSTEM\_PYTHON** Directory for system python libraries.

**TEMP** Store temporary files here.

**TMP** or **\$TMPDIR** Store temporary files here.

**SDL\_AUDIODRIVER** LibSDL audio driver - also, esd, dma.

**PYTHONHOME** Path to the python directory, eg. /usr/lib/python.

## 2.16 Troubleshooting

### 2.16.1 Troubleshooting Startup

There are some common causes for problems when using Blender. If you can not find a solution for your problem here, try asking the community for help.

#### Crash on Startup

If blender crashes on startup there are a few things to check for:

- See if you computer meets the [minimum requirements](#)
- Confirm that you graphics card is supported and that the drivers support at least OpenGL 1.4
- Make sure you are using the correct Blender version (32 or 64 bit) for your architecture.

## 2.16.2 Troubleshooting the 3D View

---

### TODO

See: <https://developer.blender.org/T43810>

---

### Drawing

#### Depth Buffer Glitches

TODO, see: <http://blender.stackexchange.com/questions/1385>

### Performance

#### Slow Drawing

TODO.

#### Slow Selection

Blender uses OpenGL drawing for selection, some graphics card drivers are slow at performing this operation (*since its not in common used for games*).

This becomes especially problematic on dense geometry.

Possible Solutions:

**OpenGL Occlusion Queries (User Preference)** See *User Preferences* → *System* → *Selection*

This option defaults *Automatic*, try setting this to *OpenGL Occlusion Queries*, since there is a significant performance difference under some configurations.

**OpenGL Vertex Buffer Objects** See *User Preferences* → *System* → *VBOs*

This uses a more optimal drawing method which may speed up selection.

**Upgrade OpenGL Driver** In some cases slow selection is resolved by using updated drivers.

*It's generally good to use recent drivers when using 3D software.*

**Select Centers (Workaround)** In *Object Mode*, holding `Ctrl` while selecting uses the object center point. While this can be useful on its own, its has the side-effect of not relying on OpenGL selection.

**Change Draw Modes (Workaround)** Using *Wireframe* or even *Bounding Box* draw modes can used to more quickly select different objects.

---

**Note:** Obviously the workarounds listed here aren't long term solutions, but its handy to know if you're stuck using a system with poor OpenGL support.

Ultimately, if none of these options work out it may be worth upgrading your hardware.

---

## Navigation

### Lost in Space

When navigating your scene, you may accidentally navigate away from your scene and find yourself with a blank view-port TODO.

### Invisible Limit Zooming In

TODO, see: <http://blender.stackexchange.com/questions/644>

## Tools

### Invalid Selection

There are times when selection fails under some configurations, often this is noticeable in mesh *Edit Mode*, selecting vertices/edges/faces where random elements are selected.

Internally Blender uses *OpenGL* for selection, so the graphics card driver is relied on giving correct results.

Possible Solutions:

**Disable Anti-Aliasing (*FSAA*, *Multi-Sampling*)** This is by far the most common cause of selection issues.

There are known problems with some graphics cards when using FSAA/multi-sampling.

You can disable this option by:

- Turning FSAA/multi-sampling off in your graphics card driver options.
- Turning *Multi-Sampling* off in the *system preferences*.

**Change Anti-Aliasing Sample Settings** Depending on your OpenGL configuration, some specific sample configurations may work, while others fail.

Unfortunately finding working configuration involves trial & error testing.

**Upgrade OpenGL Driver** As with any OpenGL related issues, using recent drivers can resolve problems.

However it should be noted that this is a fairly common problem and remains unresolved with many drivers.

### 2.16.3 Troubleshooting Graphics Hardware

Blender makes use of OpenGL, which is typically hardware accelerated.

This means issues with the graphics card hardware and drivers can impact on Blender's behavior. This page lists some known issues using Blender on different graphics hardware and how to trouble-shoot them.

### 2.16.4 Troubleshooting Crashes

The most common causes of Blender crashes.

- Running out of Memory.
- Issues with Graphics Hardware/Drivers.
- Bugs in Blender's code.

Firstly, you may be able to recover your work with *File* → *Recover Last Session*.

To prevent the problem from happening again, you can check that the graphics drivers are up to date, upgrade your machine's hardware (the RAM or graphics card), and disable some options that are more memory intensive:

- On some computers, using the *VBO* option will result in instability. Find this preference at *User Preferences* → *System* → *VBO*
- Disable *Region Overlap* and *Triple buffering* at *User Preferences* → *System* → *Window Draw Method*.
- Using multisample, anti-aliasing also increase the memory usage and make display slower.
- On Linux, the Window Manager (KDE, Gnome, Unity) may be using hardware accelerated effects (eg. window shadows and transparency) that are using up the memory that Blender needs. Try disabling the desktop effects or switch to a light-weight Window Manager.

## 2.17 Glossary

**Active** Blender makes a distinction between selected and active. Only one Object or item can be active at any given time.

**Action Safe** Area of the screen visible on most devices. Place content inside it to ensure it doesn't get cut off.

**Actuator** A *logic brick* that acts like a muscle of a lifeform. It can move the object, or also make a sound.

**Aliasing** Rendering artifacts in the form of jagged lines.

**Alpha Channel** Additional channel in 2D image for transparency.

**Straight Alpha** This is the alpha type used by paint programs such as Photoshop or Gimp, and used in common file formats like PNG, BMP or Targa. So, image textures or output for the web are usually straight alpha. RGBA color are stored as (R, G, B, A) channels, with the RGB channels unaffected by the alpha channels.

**Premultiplied Alpha** Rendering will output premultiplied alpha images, and the OpenEXR file format uses this alpha type. So, intermediate files for rendering and compositing are often stored as premultiplied alpha. Compared to straight alpha, the colors could be considered to be stored as (R\*A, G\*A, B\*A, A), with the alpha multiplied into the RGB channel.

This is the natural output of render engines, with the RGB channels representing the amount of light that comes toward the viewer, and alpha representing how much of the light from the background is blocked.

**Conversion (Straight/Premultiplied) Alpha** Conversion between the two alpha types is not a simple operation and can involve data loss, as both alpha types can represent data that the other can not, though it is often subtle.

Straight alpha can be considered to be an RGB color image with a separate alpha mask. In areas where this mask is fully transparent, there can still be colors in the RGB channels. On conversion to premultiplied alpha this mask is 'applied' and the colors in such areas become black and are lost.

Premultiplied alpha on the other hand can represent renders that are both emitting light and letting through light from the background. For example a transparent fire render might be emitting light, but also letting through all light from objects behind it. On converting to straight alpha this effect is lost.

**Ambient Light** It's light that doesn't seem to come from a specific source, but is just there. In the real world, this is caused by stray photons bouncing around and occasionally ricocheting under the desk. Since the light doesn't come from anywhere, all sides of an object are illuminated equally, and it won't have any shading on it.

**Ambient Occlusion** A ratio of how much ambient light a surface point would be likely to receive. It simulates a huge dome light surrounding the entire scene. If a surface point is under a foot or table, it will end up much darker than the top of someone's head or the tabletop.

**Animation** Simulation of motion.

**Anti-aliasing** See *oversampling*.

**Armature** A single-member class of primitive object. It consists of *bones*. Its primary use is in development of animated, articulated objects.

**Automerge** Editing mode working with Snap-tool. It removes the doubles when you snap 2 vertices.

**Axis** A reference line depicting the direction of one of the coordinates in any coordinate system.

**Bevel** Operation to chamfer or bevel edges of an object.

**Bone** Each of the segments that make up an *armature*.

**Boolean** Operation that takes 2 meshes. There 3 types: union, intersection and difference. Order of meshes is important for difference operation. With A and B as the meshes:

- Difference (A - B): the inner part of B is subtracted from the inner part of A.
- Union: the inner parts of both meshes A and B are combined.
- Intersect: only inner parts both meshes have in common are kept.

These operations take care of inner and outer parts of meshes (given by normals).

**Bounding Box** Box that encloses the shape of an object. The box is aligned with the local space of the object.

**Bump Mapping** Is a technique where at each pixel, a perturbation to the surface normal of the object being rendered is looked up in a texture map and applied before the illumination calculation is done.

**Bézier** It's a computer graphics technique for generating and representing curves.

**Caustics** In optics is a bundle of light rays. For example a caustic effect may be seen when light refracts or reflects through some refractive or reflective material, to create a more focused, stronger light on the final location. Such amplification, especially of sunlight, can burn – hence the name. A common situation when caustics are visible is when some light points on glass. There is a shadow behind the glass, but also there is a stronger light spot. Nowadays, almost every advanced rendering system supports caustics. Some of them even support volumetric caustics. This is accomplished by raytracing the possible paths of the light beam through the glass, accounting for the refraction, reflection, etc.

**Collapse** Is a tool used to remove redundant edges from geometry.

**Color Blend Modes** Ways in which 2 colors can be blended in computer graphics.

See Wikipedia's [Blend Modes](#)

**Concave face** Face in which one vert is inside a triangle formed by other vertices of the face.

**Constraint** It is any factor that limits the performance of a system with respect to its goal.

**Control cage** Mesh used in subsurf modeling to control the shape of the mesh.

**Controller** A *logic brick* that acts like the brain of a lifeform. It makes decisions to activate muscles (*actuators*), either using simple logic or complex Python scripts.

**Convex face** Not *concave face*. Opposite of concave face.

**Coplanar** Items that are in the same plane in 3D space.

**Crease** It's used to define the sharpness of edges and faces of subsurfaced meshes.

**Curve** It's a class of objects. In Blender there are *Bézier* curves and *NURBS* curves.

**DOF, Depth Of Field** It's the distance in front of and behind the subject which appears to be in focus. For any given lens setting, there is only one distance at which a subject is precisely in focus, but focus falls off gradually on either side of that distance, so there is a region in which the blurring is tolerable. This region is greater behind the point of focus than it is in front, as the angle of the light rays change more rapidly; they approach being parallel with increasing distance.

**Diffuse Light** It's even, directed light coming off a surface. For most things, the diffuse light is the main lighting we see. Diffuse light comes from a specific direction or location, and creates shading. Surfaces facing towards the light source will be brighter, while surfaces facing away from the light source will be darker.

**Directional Light** Is a light that has a specific direction, but no location. It seems to come from an infinitely far away source, like the sun. Surfaces facing the light are illuminated more than surfaces facing away, but their location doesn't matter. A Directional Light illuminates all objects in the scene, no matter where they are.

**Displacement Mapping** Uses a greyscale heightmap, like *Bump Mapping*, but the image is used to physically move the vertices of the mesh at render time. This is of course only useful if the mesh has large amounts of vertices.

**Doppler Effect** The Doppler effect is the change in pitch that occurs when a sound has a velocity relative to the listener. When a sound moves towards the listener the pitch will rise. When going away from the listener the pitch will drop. A well known example is the sound of an car passing by.

**Double Buffer** Blender uses two buffers (images) to draw the interface in. The content of one buffer is displayed, while drawing occurs on the other buffer. When drawing is complete, the buffers are switched.

**Edge** Straight segment (line) that connects 2 *vertices*, and can be part of a *face*.

**Edge Loop** Chain of *edges* belonging to consecutive *quads*. An edge loop ends at a pole or a boundary. Otherwise it is cyclic.

**Edge Ring** Path of all *edges* along a *face loop* that share 2 faces belonging to that loop.

**Empty** Kind of object that cannot hold any geometry.

**Environment Map** Method of calculating reflections. It involves rendering images at strategic positions and applying them as textures to the mirror. Now in most cases obsoleted by Raytracing, which though slower is easier to use and more accurate.

**Extrude** Modeling tool used to extend and add geometry to a mesh.

**Face** Mesh element that defines a piece of surface. It consists of 3 or more *edges*.

**Face Loop** Chain of consecutive *quads*. A face loop stops at a *triangle* or *Ngon* (which don't belong to the loop), or at a boundary. Otherwise it's cyclic.

**Face Normal** The normalized vector perpendicular to the plane that a *face* lies in.

Each face has its own normal.

**FCurve** Curve that holds the animation values of a specific property.

**Field of View** The area in which objects are visible to the camera. Also see *Focal Length*

**Focal Length** Distance required by a lens to focus collimated light. Defines the magnification power of a lens. Also see *Field of View*

**FSAA, Full-Screen Anti-Aliasing** Also known as *Multi-Sampling*, is a way of enabling *Anti-aliasing* on the graphics card, so the entire image is displayed smooth.

On many graphics cards this can be enabled in the driver options. This can also be set in the *system preferences*.

**Gamma** TODO.

**Geometric Center** An object's geometric center coincides with the geometric center of its bounding box.

**Global Illumination** Is a superset of radiosity and ray tracing. The goal is to compute all possible light interactions in a given scene, and thus obtain a truly photo realistic image. All combinations of diffuse and specular reflections and transmissions must be accounted for. Effects such as colour bleeding and caustics must be included in a global illumination simulation.

**Gouraud Shading** Used to achieve smooth lighting on low-polygon surfaces without the heavy computational requirements of calculating lighting for each pixel. The technique was first presented by Henri Gouraud in 1971.

**HDRI, High Dynamic Range Image** HDRI is a set of techniques that allow a far greater dynamic range of exposures than normal digital imaging techniques. The intention is to accurately represent the wide range of intensity levels found in real scenes, ranging from direct sunlight to the deepest shadows. The use of high dynamic range imaging in computer graphics has been popularised by the work of Paul Debevec.

See Wikipedia's: [HDRI](#).

**IOR, Index Of Refraction** It's a property of transparent materials. When a light ray travels through the same volume it follows a straight path. However if it passes from one transparent volume to another, it bends. The angle by which the ray is bent can be determined by the IOR of the materials of both volumes.

**Interpolation** Method of calculating new data between points of known value, like *keyframes*.

**Inverse Kinematics** Is the process of determining the movement of interconnected segments of a body or model. Using ordinary Kinematics on a hierarchically structured object you can for example move the shoulder of a puppet. The upper and lower arm and hand will automatically follow that movement. IK will allow you to move the hand and let the lower and upper arm go along with the movement. Without IK the hand would come off the model and would move independently in space.

**Keyframe** It's a frame in an animated sequence drawn or otherwise constructed directly by the user. In classical animation, when all frames were drawn by animators, the senior artist would draw these frames, leaving the "in between" frames to an apprentice. Now, the animator creates only the first and last frames of a simple sequence (keyframes); the computer fills in the gap.

**Knife** Is a tool used to cut meshes to get more geometry.

**Lattice** A lattice consists of a non-renderable three-dimensional grid of vertices (see also the [Lattice Modifier](#)).

**Layer** A visibility flag for objects.

**Logic brick** A graphical representation of a functional unit in Blender's game logic. A Logic brick can be a *Sensor*, *Controller* or *Actuator*.

**Manifold** Manifold meshes, called also *water tight* meshes, define a **closed non-self-intersecting volume** (see also *non-manifold*). A manifold mesh is a mesh in which the structure of the connected faces in a closed volume will always point the normals (and their surfaces) to the outside or to the inside of the mesh without any overlaps. If you recalculate those normals, they will always point at a predictable direction (To the outside or to the inside of the volume). When working with non-closed volumes, a manifold mesh is a mesh in which the normals will always define two different and non-consecutive surfaces. A manifold mesh will always define an even number of non overlapped surfaces.

**Mesh** Type of object consisting of *vertices*, *edges* and *faces*.

**Motion Blur** It's the simulation of the phenomenon that occurs when we perceive a rapidly moving object. The object appears to be blurred because of our persistence of vision. Doing motion blur makes computer animation appear more realistic.

**Multi-sampling** See *FSAA*

**Ngon** It's a *face* that contains more than four vertices.

**Non-linear animation** Animation technique that allows the animator to edit motions as a whole, not just the individual keys. Nonlinear animation allows you to combine, mix, and blend different motions to create entirely new animations.

**Non-manifold** Non-Manifold meshes essentially define geometry which cannot exist in the real world. This kind of geometry is not suitable for several types of operations, specially those where knowing the volume (inside/outside) of the object is important (refraction, fluids, booleans, or 3D printing, to name a few). A non-manifold mesh is a mesh in which the structure of a non-overlapped surface (based on it's connected faces) won't determine the inside or the outside of a volume based on it's normals, defining a single surface for both sides, but ended with flipped normals. When working with non-closed volumes, a non-manifold mesh will always determine at least one discontinuity at the normal directions, either by an inversion of a connected loop, or by an odd number of surfaces. A non manifold mesh will always define an odd number of surfaces.

There are several types of non-manifold geometry:

- Some borders and holes (edges with only a single connected face), as faces have no thickness.
- Edges and vertices not belonging to any face (wire).
- Edges connected to 3 or more faces (interior faces).
- Vertices belonging to faces that are not adjoining (e.g. 2 cones sharing the vertex at the apex).

Use *3D View* → *Select* → *Non Manifold* to select these types of non-manifold geometry in a mesh.

**Normal** The normalized vector perpendicular to a surface.

Normals can be assigned to vertices, faces and modulated across a surface using *normal mapping*.

**Normal mapping** Is similar to *Bump mapping*, but instead of the image being a greyscale heightmap, the colours define in which direction the normal should be shifted, the 3 colour channels being mapped to the 3 directions X, Y and Z. This allows more detail and control over the effect.

**NURBS** Is a computer graphics technique for generating and representing **curves** and **surfaces**.

**Object center** Reference point of an object for positioning (translating), orienting (rotating), and scaling an it. In most cases, this center is at the geometric center of the object (geometric center of its bounding box). However, an object's center may be offset from the geometric center.

**OpenGL** The graphics system used by Blender (and many other graphics applications) for drawing 3D graphics, often taking advantage of hardware acceleration. See Wikipedia's: [OpenGL](#).

**Oversampling** Is the technique of minimizing *aliasing* when representing a high-resolution signal at a lower resolution.

Also called **Anti-Aliasing**.

**Overscan** Overscan is the term used to describe the situation when not all of a televised image is present on a viewing screen. See Wikipedia's: [Overscan](#)

**Particle system** It's a technique that simulate certain kinds of fuzzy phenomena, which are otherwise very hard to reproduce with conventional rendering techniques. Common examples include fire, explosions, smoke, sparks, falling leaves, clouds, fog, snow, dust, meteor tails, stars and galaxies, or abstract visual effects like glowing trails, magic spells. Also used for fur, grass or hair.

**Phong** Local illumination model that can produce a certain degree of realism in three-dimensional objects by combining three elements: diffuse, specular and ambient for each considered point on a surface. It has several assumptions - all lights are points, only surface geometry is considered, only local modelling of diffuse and specular, specular colour is the same as light colour, ambient is a global constant.

**Pivot Point** It's a reference point used by many mesh manipulation tools.

**Pixel** The smallest unit of information in a 2D raster image, representing a single color made up of red, green, and blue channels. If the image has an *alpha channel*, the pixel will contain a corresponding fourth channel.

**Pole** It's a vertex in which three or five or more edges are connected to. A vertex connected to one, two or four edges, is not a pole.

**Premultiplied Alpha** See [Alpha Channel](#)

**Primitive** Is a basic object that can be used as a basis for modeling more complicated objects.

**Procedural Texture** Computer generated (generic) textures. Procedural textures can be configured via parameters.

**Proportional Editing** Used to alter existing model in a more organic way. When elements are moved interactively, neighbouring elements are also moved depending on their distance and the defined parameters.

**Quad** It's a *face* that contains exactly four vertices.

**Radiosity** Is a global lighting method that calculates patterns of light and shadow for rendering graphics images from three-dimensional models. One of the many different tools which can simulate diffuse lighting in Blender.

See Wikipedia's [Radiosity \(computer graphics\)](#)

**Raytracing** Rendering technique that works by tracing the path taken by a ray of light through the scene, and calculating reflection, refraction, or absorption of the ray whenever it intersects an object in the world. More accurate than *scanline*, but much slower.

**Refraction** It's the change in direction of a wave due to a change in velocity. It happens when waves travel from a medium with a given *index of refraction* to a medium with another. At the boundary between the media, the wave changes direction; its wavelength increases or decreases but frequency remains constant.

**Relative Vertex Keys** Are part of a keyframe animation system that operates on vertex level objects. Each key is stored as a morph target such that several keys may be blended together to achieve complex mesh animation. Facial expressions, speech, and other detailed animated keyframed movements can be created within mesh-based models.

**Render** Process of generating an image out of a 3D model on a computer.

**Scanline** Rendering technique. Much faster than *raytracing*, but allows fewer effects, such as reflections, refractions, motion blur and focal blur.

**Sensor** A *logic brick* that acts like a sense of a lifeform. It reacts to touch, vision, collision etc.

**Shading** Process of altering the color of an object/surface in the 3D scene, based on its angle to lights and its distance from lights to create a photorealistic effect.

**Smoothing** Defines how *faces* are shaded. Face can be either solid (faces are rendered flat) or smooth (faces are smoothed by interpolating the normal on every point of the face).

**Specular light** Refers to the highlights on reflective objects.

**Straight Alpha** See *Alpha Channel*

**Sub surface scattering** Mechanism of light transport in which light penetrates the surface of a translucent object, is scattered by interacting with the material, and exits the surface at a different point. All non-metallic materials are translucent to some degree. In particular, materials such as marble, skin, and milk are extremely difficult to simulate realistically without taking subsurface scattering into account.

**Subdividing** It's used to add more geometry to a mesh. It creates new vertices on subdivided edges, new edges between subdivisions and new faces based on new edges. If new edges cross a new vertex is created on their crossing point.

**Subdivision surface** Is the tool which subdivides your model at render-time, without affecting your mesh at design-time.

Also called: **Subsurf**.

**Tessellation** TODO.

**Texture** A texture specifies visual patterns on surfaces and simulates physical surface structure.

**Title Safe** Area of the screen visible on all devices. Place text and graphics inside this area to make sure they don't get cut off.

**Topology** Arrangement of *Vertices*, *Edges*, and *Faces* which define the shape of a mesh. See *vertex*, *edge*, and *face*.

**Topology** TODO.

**Triangle** It's a *face* with exactly 3 *vertices*.

**UV map** A UV map defines a relation between the surface of a 3 dimensional mesh and a planar 2D texture. In detail, each face of the mesh is mapped to a corresponding face on the texture. It is possible and often common practice to map several faces of the mesh to the same or overlapping areas of the texture.

**Vertex** It's a point in 3D space containing a location. It may also have a defined color. Vertices are the terminating points of *edges*.

**VBO, Vertex Buffer Object** Term used for uploading geometry to the graphics cards memory for improved performance with *OpenGL* drawing.

See Wikipedia's [Vertex Buffer Object](#)

**Vertex Group** Vertices can be grouped together so that certain operations can work on specific groups.

Manual Index

## 2.18 Get Involved

This manual is maintained largely by volunteers.

Please consider to join the effort and [Contribute to this Manual](#)

### 2.18.1 About this Manual

#### About the Blender Manual

The Blender Manual is a community driven effort to which anyone can contribute. Either if you found a typo or if you want to improve the general quality of the documentation, there are several options for helping out. You can:

1. Fix problems, improve the documentation and write new sections - see how to [contribute](#).
2. [Report problems](#) in the documentation.
3. Get involved in discussions through the [mailing list](#) and [#blenderwiki IRC channel](#).

#### Organization and Process

The Blender manual maintainers are divided into administrators, section owners, section members and contributors. Before you can edit the documentation directly, you need to submit patches for review, similarly to how Blender's own development process works. Straight-forward patches are bound to be accepted very quickly and once you get accustomed to making changes and no longer need feedback, you will gain commit access.

Administrators are responsible for managing the whole project and infrastructures (such as this website). Section Owners are responsible for a given section, reviewing patches for it, giving feedback and keeping up to date with Blender development in that area.

Currently, the manual is still undergoing a transition process from the previous wiki format, and we are actively searching for contributors and documentation module owners.

#### Communication

Communication is a very important step of community development. Before you make any edits that are not simple and plainly justified (for example, moving folders around), you should verify with a manual maintainer that your contribution is along the community's vision for the manual. This ensures best use of your time and good will as it is otherwise possible that, for some reason, your changes will conflict and be rejected or need time-consuming review. For example, another person may be already working on the section you wish to change, the section may be scheduled for deletion or to be updated according to a planned change to Blender.

Manual maintainers and the general community can also point to areas that are in need of big or small changes. Communicating early and frequently is key to have a productive environment, to not waste people's effort and to attain a better Blender manual as a result.

Most communication happens on mailing lists, IRC chat and the bug and patch trackers.

---

If you are in doubt about functionality that you wish to document, you should pose your questions to the Blender developers responsible for that area or ask at the unofficial user support channel at [#blender](#).

Blender has its own system of module owners with developer and artist members who are responsible for the design and maintenance of the assigned Blender areas. See the [module owners page](#) for more information.

## License

Blender's Manual is free and published under [Creative Commons Zero](#).

This means that anyone is free to download, edit and share the manual. It also means that when contributing to the manual you don't hold exclusive copyright to your text. You are, of course, acknowledged and appreciated for your contribution. However others can change and improve your text in order to keep the manual consistent and up to date.

## Contribute

Whether you'd like to fix a tiny spelling mistake or rewrite an entire chapter, your help with the Blender manual is most welcome!

## How It Works

In plain English:

- The manual is written in the [reStructuredText](#) markup language and contained in a central repository.
- Writers can download the source files from this repository and edit them using a text editor.
- They can “convert” the source files into HTML web pages to see exactly what it'll look like on [blender.org/manual](http://blender.org/manual).
- When they're happy with their changes, they publish them to the central repository so that everyone else can work from them.

## Install and Build

The installation and building process is different for each operating system, instructions have been written for:

- [Linux](#)
- [OSX](#)
- [Windows](#)

## Make Your Changes

Firstly, make sure that your local copy of the manual is up to date with the online repository using:

```
svn update
```

You can now **edit** the documentation files, which are the `.rst` files inside the `manual` folder using a text editor of your choice.

For instructions on using different editors, see the [Editor Configuration](#) section of the community wiki.

Be sure to check the [Writing Style Guide](#) for conventions and [Markup Style Guide](#) to learn how to write in the reStructuredText markup language.

To **view** your changes, build the manual [as instructed](#). Keep in mind that you can also build only the chapter you just edited to view it quickly. Open the generated `.html` files inside the `build/html` folder using your web browser, or refresh the page if you have it open already.

When you are happy with your changes, you can **upload** them, so that they will be in the online manual. At first, this is done by submitting patches so that someone can review your changes and give you feedback. After, you can commit your changes directly.

## Submit Patches

The first few times you make changes to the manual, you'll need to submit them as patches for the section owner to review. This is just to make sure that we maintain a quality user manual, and that you don't accidentally break anything vital before you get used to the system.

In order to submit a patch, follow this process:

1. Make any changes that you want
2. Create a patch file by running:

```
svn diff > filename.diff
```

This creates a simple text file that shows what text was added, removed or changed between your working files and the central repository.

If you have created or deleted files, you will need run `svn add /path/to/file` or `svn rm /path/to/file` before creating the diff. To see a list of affected files, run `svn status`.

3. Upload the diff file [here](#). If you don't have an account already, you can [register for one](#).
4. After submitting the diff, you'll be asked to "Create a new Revision" before you can add a title and description of your changes.
5. If you know who the Section Owner (see [Documentation Team here](#)) of that chapter is, assign them as the *Reviewer* and they'll be notified of your patch. If you can't find out who that is (or there is no one), instead mail the [bf-docboard](#) mailing list, or tell someone in [#blenderwiki](#) on [IRC](#).
6. They will review your patch and let you know about any changes you could make, or commit the patch if it is accepted.

---

**Note:** If your patch includes changes to or additional images, simply attach them when you're creating the revision.

---

Once you have had a few patches accepted, we cut out the middle man and give you direct access to edit the manual!

## Commit Directly

Instead of creating a patch file, committing will submit the change directly to our central repository.

All you need to do now is run:

```
svn commit -m "This is what I did"
```

If you leave out `-m "message"`, you'll be prompted to type the message in a text editor.

Do not forget to always run `svn update` before committing.

Then you'll be asked for your username (from `developer.blender.org`) and password before the change is committed.

## Install

This section documents how to install the software used to generate the manual on your own system.

### Installation Guide for Editing the Blender Manual on Linux

This guide covers the following topics:

1. *Installing Dependencies*

2. *Downloading the Repository*
3. *Setting up the Build Environment*
4. *Building the HTML Files*

**Installing Dependencies** Below are listed the installation commands for popular Linux distributions.

For the appropriate system, run the command in a terminal:

#### Debian/Ubuntu

```
sudo apt-get install python python-pip subversion
```

#### Redhat/Fedora

```
sudo yum install python python-pip
```

#### Arch Linux

```
sudo pacman -S python python-pip subversion
```

**Downloading the Repository** Simply checkout the blender-manual repository using:

```
cd ~
svn checkout https://svn.blender.org/svnroot/bf-manual/trunk/blender_docs
```

The repository will now be downloaded which may take a few minutes depending on your internet connection.

**Setting up the Build Environment** In a terminal, enter the `blender_docs` folder which was just added by the SVN checkout:

```
cd ~/blender_docs
```

Inside that folder is a file called `requirements.txt` which contains a list of all the dependencies we need. To install these dependencies, we can use the `pip` command:

```
sudo pip install -r requirements.txt
```

**Building the HTML Files** We are now ready to convert all those `rst` files into pretty **html!**

Open a terminal to the folder `~/blender_docs` and simply run:

```
make
```

This is the command you will always use when building the docs. The building process may take several minutes the first time (or after any major changes), but the next time you build it should only take a few seconds.

Once the docs have been built, all the html files can be found inside `~/blender_docs/build/html`. Try opening `build/html/contents.html` in your web browser and read the manual.

```
xdg-open build/html/contents.html
```

Now that you are able to build the manual, please visit [blender.org/documentation](http://blender.org/documentation) for more information such as the style guide and how to submit patches and gain commit access.

**Building a Single Chapter** If you are working on a specific chapter of the manual, you can build it quickly using:

```
make <chapter name>
```

For example, to build only the documentation for the modifiers, use `make modifiers`. You can then view this quick build by opening `build/html/contents_quicky.html`.

This will build very quickly, but it will mean your next complete build of all the chapters will be slow.

## Installation Guide for Editing the Blender Manual on OSX

This guide covers the following topics:

1. *Installing Dependencies*
2. *Downloading the Repository*
3. *Setting up the Build Environment*
4. *Building the HTML Files*

---

**Note:** This guides relies heavily in command-line tools. It assumes you are the least familiar with the OSX Terminal application.

---

**Installing Dependencies** Install those packages or make sure you have them in your system.

- Python
- PIP
- Subversion

**Downloading the Repository** Simply checkout the blender-manual repository using:

```
cd ~
svn checkout https://svn.blender.org/svnroot/bf-manual/trunk/blender_docs
```

The repository will now be downloaded which may take a few minutes depending on your internet connection.

**Setting up the Build Environment** In a terminal, enter the `blender_docs` folder which was just added by the SVN checkout:

```
cd ~/blender_docs
```

Inside that folder is a file called `requirements.txt` which contains a list of all the dependencies we need. To install these dependencies, we can use the `pip` command:

```
sudo pip install -r requirements.txt
```

**Building the HTML Files** We are now ready to convert all those `rst` files into pretty `html`!

Open a terminal to the folder `~/blender_docs` and simply run:

```
make
```

This is the command you will always use when building the docs. The building process may take several minutes the first time (or after any major changes), but the next time you build it should only take a few seconds.

Once the docs have been built, all the html files can be found inside `~/blender_docs/build/html`. Try opening `build/html/contents.html` in your web browser and read the manual.

```
open build/html/contents.html
```

Now that you are able to build the manual, please visit [blender.org/documentation](http://blender.org/documentation) for more information such as the style guide and how to submit patches and gain commit access.

**Building a Single Chapter** If you are working on a specific chapter of the manual, you can build it quickly using:

```
make <chapter name>
```

For example, to build only the documentation for the modifiers, use `make modifiers`. You can then view this quick build by opening `build/html/contents_quicky.html`.

This will build very quickly, but it will mean your next complete build of all the chapters will be slow.

## Installation Guide for Editing the Blender Manual on MS-Windows

This guide covers the following topics:

1. *Installing Python* (used to “convert” the source files to HTML)
2. *Installing SVN and Downloading the Repository*
3. *Setting up the Build Environment*
4. *Building the HTML Files*

### Installing Python

1. Download the Python installation package for Windows from here: <https://www.python.org/downloads/>

*In this guide version 3.4 is used.*

2. Install Python with the installation wizard.

*In this guide the default settings are used. Python will be installed to C:\Python34*

**Installing SVN and Downloading the Repository** In this guide we’ll use TortoiseSVN, though any Subversion client will do.

1. Download TortoiseSVN for Windows from [here](#)
2. Install TortoiseSVN with the installation wizard. When choosing which features will be installed, it is recommended that you enable *command line client tools* to give you access to SVN from the command line (there is no harm in doing this, and it may be helpful if you ever run into any trouble).
3. Once the installation has finished, create a new folder that will contain everything related to the Blender Manual. In this guide we’ll use `C:\blender_docs`.
4. Open the new folder, right click and choose *SVN Checkout...* from the context menu.
5. In the *URL of repository* field, enter: `https://svn.blender.org/svnroot/bf-manual/trunk/blender_docs`.
6. In the *Checkout directory* field, enter: `C:\blender_docs`.
7. Click *OK* - the repository will now be downloaded which may take a few minutes depending on your internet connection.

## Setting up the Build Environment

- Open a command prompt and change to the repository folder using

```
cd C:\blender_docs
```

- Install all the requirements using Python's `pip` command

```
C:\Python34\Scripts\pip install -r requirements.txt
```

- If all goes well, you should see the following message when it's finished

```
Successfully installed Jinja2 MarkupSafe Pygments Sphinx docutils sphinx-rtd-theme Cleaning up...
```

During the setup some warnings may be shown, but don't worry about them. However if any errors occur, they may cause some problems.

## Building the HTML Files

We are now ready to convert all those `rst` files into pretty `html`!

- Open a command prompt and change to the repository with `cd C:\blender_docs`.
- Build using the following command

```
C:\Python34\Scripts\sphinx-build.exe -b html .\manual .\build\html
```

This is the command you will always use when building the docs. The building process may take several minutes the first time (or after any major changes), but the next time you build it should only take a few seconds.

- Once the docs have been built, all the html files can be found inside `C:\blender_docs\build\html`. Try opening `\build\html\contents.html` in your web browser and read the manual.

Now that you are able to build the manual, please visit [blender.org/documentation](http://blender.org/documentation) for more information such as the style guide and how to submit patches and gain commit access.

## Markup Style Guide

This page covers the conventions for writing and use of the reStructuredText (RST) markup syntax.

### Conventions

- 3 space indentation.
- Lines should be less than 120 characters long.
- Use italics for button/menu names.

### Headings

```
#####
Document Part
#####

*****
Document Chapter
*****

Document Section
```

```
=====
Document Subsection
-----
```

```
Document Subsubsection
^^^^^^^^^^^^^^^^^^^^^^^^
```

```
Document Paragraph
''''''''''''''''''''''
```

---

**Note:** *Parts* should only be used for contents or index pages.

---



---

**Note:** each `.rst` file should only have one chapter heading (\*) per file.

---

## Text Styling

See the [overview on ReStructured Text](#) for more information on how to style the various elements of the documentation and on how to add lists, tables, pictures and code blocks. The [sphinx reference](#) provides more insight additional constructs.

The following are useful markups for text styling:

```
*italic*
**bold**
``literal``
```

## Interface Elements

- `:kbd: `LMB`` - keyboard and mouse shortcuts.
- `*Mirror*` - interface labels.
- `:menuselection: `3D View --> Add --> Mesh --> Monkey`` - menus.

## Code Samples

There is support for syntax highlighting if the programming language is provided, and line numbers can be optionally shown with the `:linenos:` option.

```
.. code-block:: python
   :linenos:

import bpy
def some_function():
    ...
```

## Images

Figures should be used to place images:

```
.. figure:: /images/modifiers_subsurf_example.jpg
```

Image Caption

## Files

**No Caps, No Gaps** Lower case filenames, underscore between words.

**Sort Usefully** Order naming with specific identifiers at the end.

**Format** Use `.png` for images that have solid colors such as screenshots of the Blender interface, and `.jpg` for images with a lot of color variance, such as sample renders and photographs.

Do not use animated `.gif` files, these are hard to maintain, can be distracting and are usually large in file size. If a video is needed, use YouTube or Vimeo (see *Videos* below).

**Location** Place the image in the `manual/images` folder. Use no other subfolders.

**Naming** Image files should be named: `chapter_subsection_id.png`, eg:

- `render_cycles_lighting_example_01.jpg`
- `interface_intro_splash.jpg`
- `interface_ui_panel.jpg`

Do not use special characters or spaces

## Usage Guides

- Avoid specifying the resolution of the image or its alignment, so that the theme can handle the images consistently and provide the best layout across different screen sizes.
- When documenting a panel or section of the UI, it is better to use a single image that shows all of the relevant area (rather than multiple images for each icon or button) placed at the top of the section you are writing, and then explain the features in the order that they appear in the image.

---

**Note:** It's important that the manual can be maintained long term, UI and tool-options change so try to avoid having a lot of images (when they aren't especially necessary). Otherwise this becomes too much of a maintenance burden.

---

## Videos

Videos from YouTube and Vimeo can be embedded using:

```
.. youtube:: ID
```

```
.. vimeo:: ID
```

The ID is found in the video's URL, e.g:

- The ID for `https://www.youtube.com/watch?v=Ge2Kwy5EGE0` is `Ge2Kwy5EGE0`
- The ID for `http://vimeo.com/15837189` is `15837189`

## Usage Guides

- Avoid adding videos which rely on voice, as this is difficult to translate.
- Do not embed video tutorials as a means of explaining a feature, the writing itself should explain it adequately (though you may include a link to the video at the bottom of the page under the heading `Tutorials`).

## Useful Constructs

- `|BLENDER_VERSION|` - Resolves to the current Blender version.
- `:abbr: `SSAO (Screen Space Ambient Occlusion)`` - Abbreviations display the full text as a tooltip for the reader.
- `:term: `Manifold`` - Links to an entry in the [Glossary](#).

## Cross References and Linkage

You can link to another document in the manual with:

```
:doc: `The Title </section/path/to/file>`
```

To link to a specific section in another document (or the same one), explicit labels are available:

```
.. _sample-label:
[section or image to reference]
Some text :ref: `Optional Title <sample-label>`
```

Linking to a title in the same file.

```
Titles are Targets
=====
Body text.
Implicit references, like `Titles are Targets`_
```

Linking to the outside world:

```
`Blender Website <http://www.blender.org>`_
```

## Directory layout

Sections should be generally structured as follows:

- `directory_name/`
  - `index.rst` (contains links to internal files)
  - `introduction.rst`
  - `section_1.rst`
  - `section_2.rst`

For example:

- rendering/
  - index.rst
  - cycles/
    - \* index.rst
    - \* introduction.rst
  - \* materials/
    - index.rst
    - introduction.rst
    - volumes.rst

The idea is to enclose all the content of a section inside of a folder. Ideally every section should have an `index.rst` (containing the TOC for that section) and an `introduction.rst` (introducing) to the contents of the section.

**Table of Contents** By default a table of contents should show two levels of depth.

```
.. toctree::
   :maxdepth: 2

   introduction.rst
   perspective.rst
   depth_of_field.rst
```

## Further Reading

To learn more about reStructuredText, see:

**Sphinx RST Primer** Good basic introduction.

**Docutils reStructuredText reference** Links to reference and user documentation.

## Writing Style Guide

### Primary Goals

The main goals for this manual are as follows.

**User Focused** While some areas of computer graphics are highly technical, this manual shall be kept understandable by non-technical users.

**Complete** So there is a canonical source of truth for each of Blender's key areas. This doesn't mean we have to document every small detail, but users shouldn't have to rely on searching elsewhere to find the purpose of key features.

**Concise** Computer graphics is an incredibly interesting field, there are many rules, exceptions to the rules and interesting details. Expanding into details can add unnecessary content, so keep the text concise and relevant to the topic at hand.

**Maintainable** Keep in mind that Blender has frequent releases, so try to write content that won't have to be redone the moment some small change is made. This also helps a small documentor community to maintain the manual.

## Content Guidelines

In order to maintain a consistent writing style within the manual, please keep this page in mind and only deviate from it when you have a good reason to do so.

Rules of thumb:

- *Spell checking is strongly recommended.*
- Use American English (eg: modeling and not modelling, color and not colour).
- Take care about grammar, appropriate wording and use simple English.
- Keep Sentences short and clear, resulting in text that is easy to read, objective and to the point.
- Including why or how an option might be useful is a good idea.
- If you are unsure about how a feature works, ask someone else or find out who developed it and ask them.

To be avoided:

- Avoid to write in first person perspective, about yourself or your own opinions.
- Avoid **weasel words** and being unnecessarily vague, eg:

*“Reloading the file will probably fix the problem”*

*“Most people don’t use this option because ...”*

- Avoid including specific details such as:

*Blender has 23 different kinds of modifiers.*

*Enabling previews adds 65536 bytes to the size of each Blend file (unless its compressed).*

These details aren’t useful for users to memorize and they become quickly out-dated.

- Avoid documenting bugs.

Blender has often has 100’s of bugs fixed between releases, so its not realistic to reference even a fraction of them from the manual, while keeping this list up to date.

Issues which are known to the developers and aren’t going to be resolved before the next release can be documented as **Known Limitations**, in some cases it may be best to include them the in the **Troubleshooting** section.

- Avoid *Product Placement* - unnecessarily promoting software or hardware brands. Keep content vendor-neutral where possible.
- Avoid technical explanations about the mathematical/algorithmic implementation of a feature if there is a simpler way to explain it (e.g. explaining how mesh smoothing algorithms work is unnecessary, but the blending types of a mix node do need a mathematical explanation).
- Avoid repetition of large portions of text - simply explain it once, and from then on refer to that explanation.

In some cases you might also consider defining a `:term:` in the **glossary**.

- Avoid enumerating similar options, such as listing every preset or every frame-rate in a drop-down.

Their contents may be summarized or simply omitted.

*Such lists are only showing what is already obvious in the interface and end up being a lot of text to read & maintain.*

- Avoid documenting changes in Blender between releases, that's what the release notes are for. We only need to document the current state of Blender.
- Unless the unit a value is measured in is obscure and unpredictable, there is no need to mention it.
- Do not simply copy the tool-tips from Blender.

*People will come to the manual to learn more than is provided by the UI.*

As a last resort you can add comment (which is not shown in the html page, but useful for other editors):

```
.. TODO, how does this tool work? ask Joe Blogg's
```

## Translations

At the moment, this manual project is still a work in progress, having been recently migrated from the wiki.

The focus is on first to prove the English version and only then to move on to migrating and supporting other languages, preventing the contributors' efforts from being wasted in work that has to be redone. There are still big restructurings taking place as well as reviewing of entire sections, that would imply rewrites on the translated versions.

The manual **is** intended to have translations, and these are considered important, we just want to make sure that the structure and workflow for the English version are stable before investing work and good will on the translations.

If you want to translate, please wait until the manual is closer to being ready and we will update this page and announce on the [bf-docboard](#) mailing list.

---

**Note:** We have investigated translation workflow, See the [translation design task](#) for details on the proposed process.

---

## MediaWiki to Sphinx Migration

At the end of 2014 we migrated the manual from MediaWiki to Sphinx, which uses the reStructuredText markup language.

This is a somewhat controversial decision, so this sections explains some of the reasons why we felt Sphinx was worth moving to.

We realize that a change in technology alone won't solve all problems, at the end of the day it's really up to us to write a better manual, but there were some issues with `wiki.blender.org` which made it difficult to work with.

## Comparisons

Note that these are subjective points, more could be written on this. However for the purpose of maintaining a manual, here are some pros and cons for each system.

### MediaWiki

#### Pros

**Online editing** Only a web browser required.

**Quick Feedback** No need to *generate* docs locally before you can see the change on the web page.

**Low barrier of entry** Easy to get involved.

**Single Pages** Each page is an isolated document - this works well for Wikipedia, and Blender developer documents.

## Cons

**Poor version handling** With a wiki we can't easily document new features during the development process. The current wiki may include information which is valid for a nightly build, but not the latest stable release.

**Low quality *drive-by* edits** many pages would have incomplete edits, incorrect information or too much highly detailed text written on a topic. So while ease of contribution has its benefits, it proved to be problematic too.

**Poor Peer Review** It was hard to properly peer review edits, a lot of changes would be made with no feedback. Writers didn't really know if their work was considered good quality or not.

**Page Hierarchy** The hierarchy in Blender's wiki was supported with an extension to MediaWiki, but its something that MediaWiki doesn't support, managing this tree online is cumbersome.

**No Project Management** Without some project management, its difficult to keep track of who does what, assign tasks, report issues etc.

## Sphinx/reStructuredText

### Pros

**Release With Blender** We can release a version of the manual with each Blender release, make it available online as well as downloadable.

**Local Structure** More easily manage the overall structure of the manual, move pages and chapters around as regular files and folders.

**Automate Edits** Local files means we can more easily manipulate text, using text editors of choice, search/replace words and generally edit the manual without having to load up a web-page first. (`wiki.blender.org` access is slow in some countries).

Tasks such as running a spell-checker, on the entire manual wasn't really possible with MediaWiki.

**Project Management** While this isn't directly a feature of Sphinx, using version-control means we can integrate a [project management system](#) (Phabricator in this case).

This means we can have a central place to track issues, set goals for releases and assign tasks.

### Cons

**No online editing.** This isn't inherently a limitation of reStructuredText, and at some point we may investigate ways to support this.

**Must be built** Docs need to be compiled into HTML, which takes time.

**Higher *barrier of entry*** Installing SVN and Sphinx isn't so easy depending on your platform and experience.

### Barrier of Entry

Increasing the barrier of entry isn't something to be taken lightly, however its our opinion that the trade-off is worthwhile.

The short term benefit of quick & easy editing, has to be weighed against the long term benefits of using a system better suited to collaboratively writing a document.

We've also observed that drive-by edits often aren't such good quality, adding redundant text and even mis-information in some cases.

## Conclusion

Both systems have their strengths and weaknesses, it's yet to be seen if we can effectively maintain a manual with the new system that's been proposed.

But `wiki.blender.org` had some years to create the manual and while some areas were very high quality, it remained a mix of old docs and poor quality content for the most part.