Независимый электронный журнал о разработке компьютерных игр

http://xtreme3d.narod.ru

FPS

> Основы Blender:

> Эффект свечения

> Проекция изображения

> Размытие при движении

> Blender 2.5

#10

2010

Мировая арт-война <

Game Music Emu <

Первые шаги в PhysFS <

Парсер своими руками <

Размытие при помощи GLSL <

FPS

• Основы Blender
Эффект свечения.....3
Проекция изображения на "экран"...6
Размытие при движении.....10

• Blender 2.5
Симуляция дыма.....12

• Мировая арт-война
Dominance War.....14
Unearthly Challenge.....16

• Game Music Emu
Музыка из ретро-игр в C++.....18

• Виртуальная файловая система
Первые шаги в PhysFS.....22

• Парсер своими руками
Язык файлов конфигурации.....26

• Шейдеры на все случаи жизни
Размытие при помощи GLSL.....34

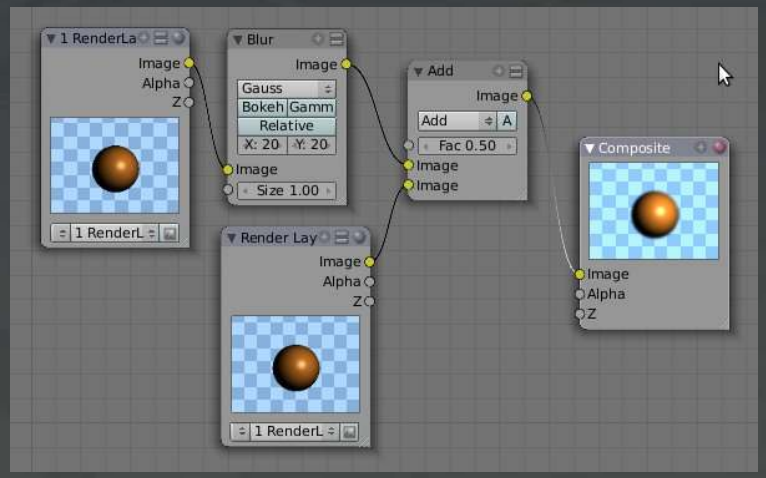
© 2008-2010 Clocktower Games. Все названия и логотипы являются собственностью их законных владельцев и не используются в качестве рекламы продуктов или услуг. Редакция не несет ответственности за корректность и достоверность информации в статьях и надежность всех упоминаемых URL-адресов. Материалы журнала распространяются согласно условиям лицензии Creative Commons.

Главный редактор: Тимур Гафаров
Дизайн и верстка: Тимур Гафаров

По вопросам сотрудничества обращаться по адресу clocktower89@mail.ru.
Официальный сайт журнала: http://xtreme3d.narod.ru/

Эффект свечения

ОПЫТНЫЕ пользователи Blender наверняка знают о такой полезной вещи, как редактор композитных узлов, позволяющий пропустить отрендеренное изображение через различные фильтры. Эффект свечения (glow) для всей сцены с его помощью делается за пять минут.



Я обычно использую два исходных буфера. Один размывается по Гауссу, затем оба смешиваются в режиме суммирования. Такой эффект хорош, когда нужно заполнить всю сцену мягким, «волшебным» светом.

Все бы ничего, но встает проблема: как применить свечение не ко всей сцене, а к отдельному объекту? Колдовать с узлами материала? Нет, мы поступим гораздо проще.

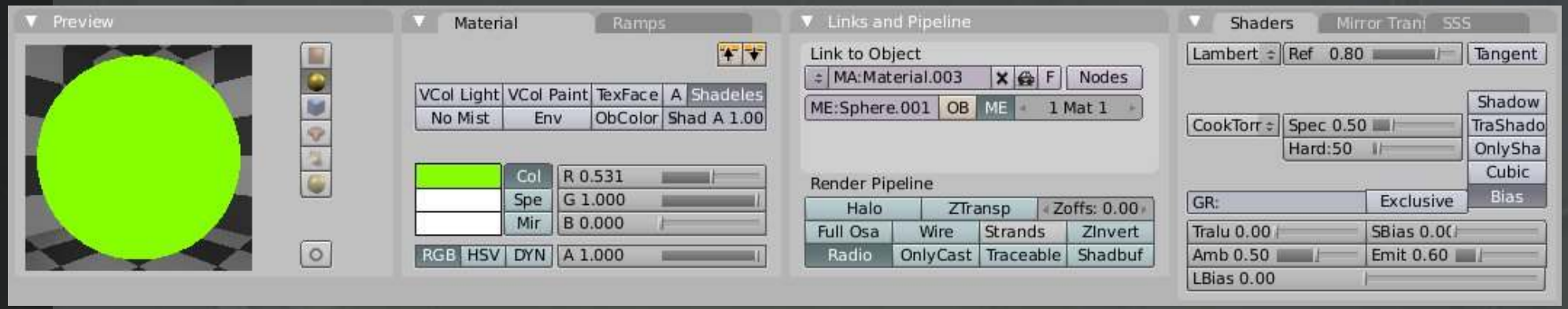
Графический конвейер Blender располагает удобной возможностью распределять объекты по так называемым слоям (scene layers). Они напоминают слои в Photoshop или в Gimp. На экране и при рендеринге отображаются только те объекты, которые принадлежат активному слою. Мы можем активировать несколько слоев сразу. Обычно слои используются, чтобы прятать ненужные в данный момент объекты для повышения быстродействия программы.

Кроме того, существуют также слои рендеринга (render layers). Это мощнейший инструмент, хотя и редко используемый. Слой рендеринга совершает отрисовку одного или нескольких слоев сцены, применяя собственные настройки визуализации. Но самое, на мой взгляд, главное, заключается в том, что отдельные слои рендеринга можно использовать одновременно в качестве исходных буферов при композитной обработке!

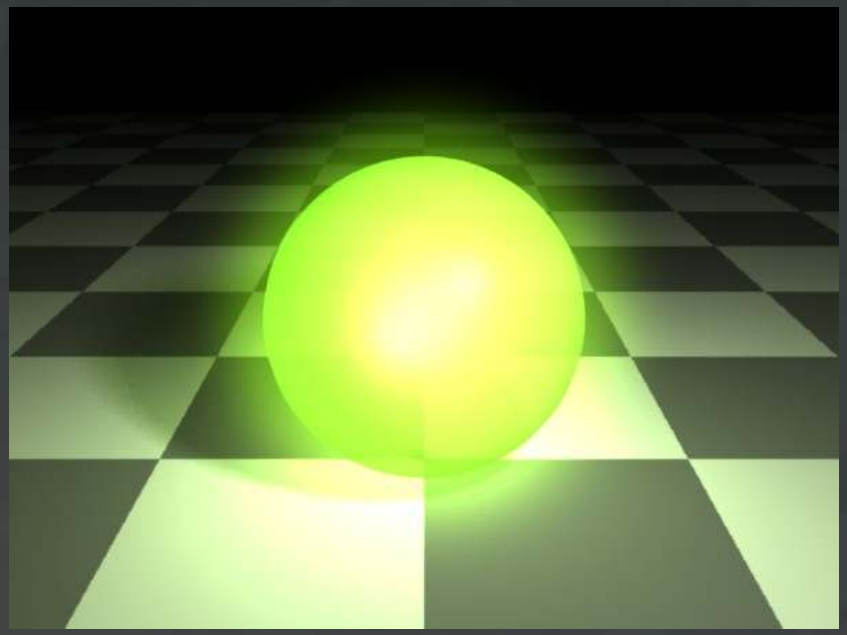
Вот что мы сделаем: на один слой сцены поместим светящийся объект, а на другой — его копию, но с другим материалом — таким, чтобы цвет свечения немного отличался от цвета самого объекта. Например, сам объект может быть светло-зеленым, почти белым, а свечение — более насыщенного зеленого цвета.

1. Создаем сам объект. Пусть это будет обычная сфера. Внутри нее можно создать источник света. Создаем сфере материал и выставляем настройки:

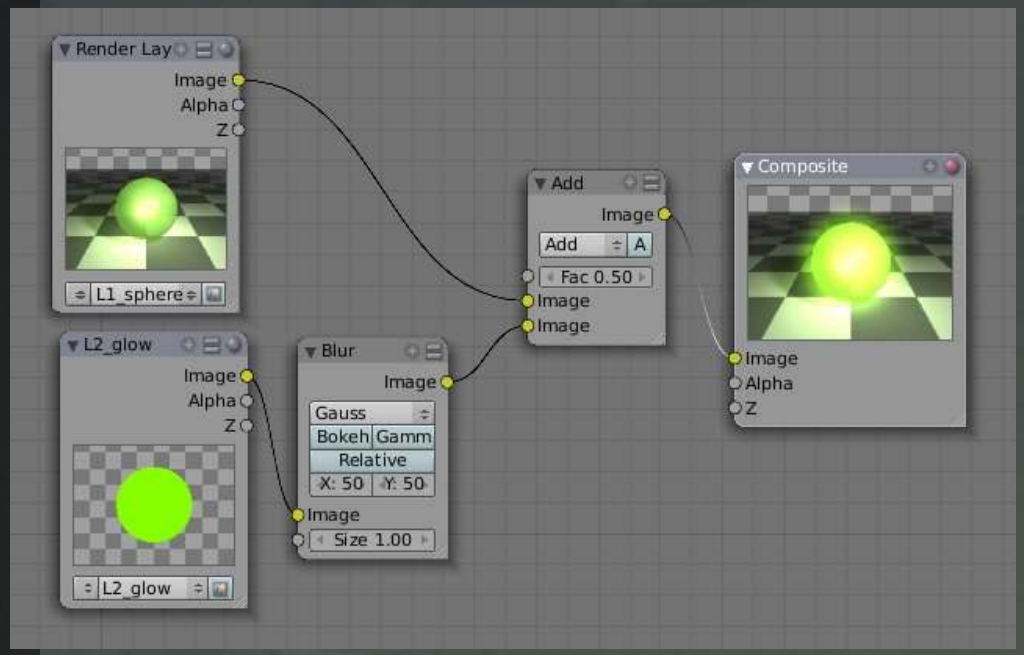
2. Копируем объект (Shift+D), немного увеличиваем копию и перемещаем ее на второй слой. Для этого открываем настройки объекта (F7) и на вкладке Draw нажимаем вторую кнопку слева в верхнем ряду в таблице слоев. Затем создаем материал и выставляем настройки.



3. Переходим в настройки сцены (F7), создаем два слоя рендеринга и указываем для них слои сцены.

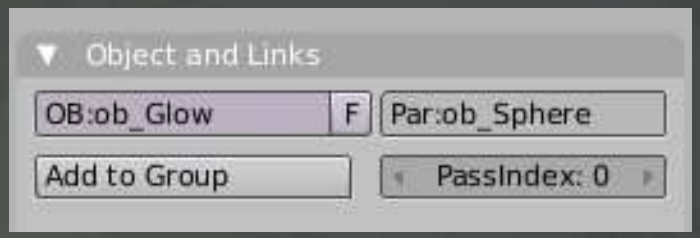


4. Переходим в редктор узлов и создаем систему как на рисунке. Активируем оба слоя (для этого достаточно включить обе кнопки на любой таблице слоев, удерживая Shift) и нажимаем F12, чтобы отрендерить.

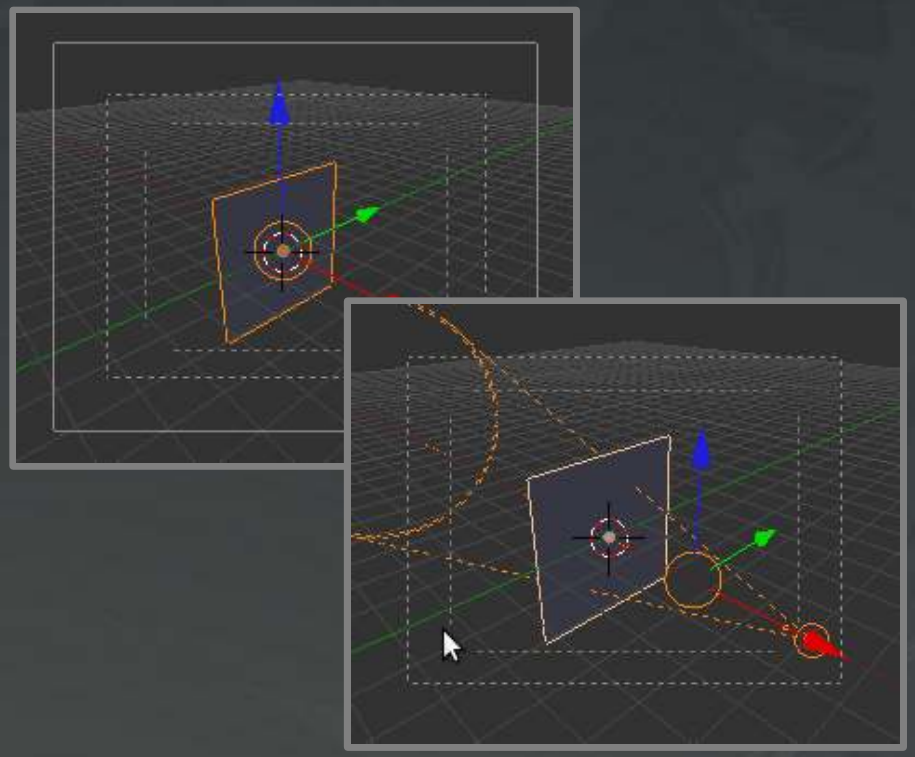


Если вы создаете анимацию, не лишним будет сделать объект свечения дочерним для главного объекта, чтобы при его перемещении происходила автоматическая синхронизация. Перейдите на второй слой, выделите объект и нажмите F7. На вкладке Object and Links в поле Par введите имя главного объекта.

Gecko
clocktower89@mail.ru



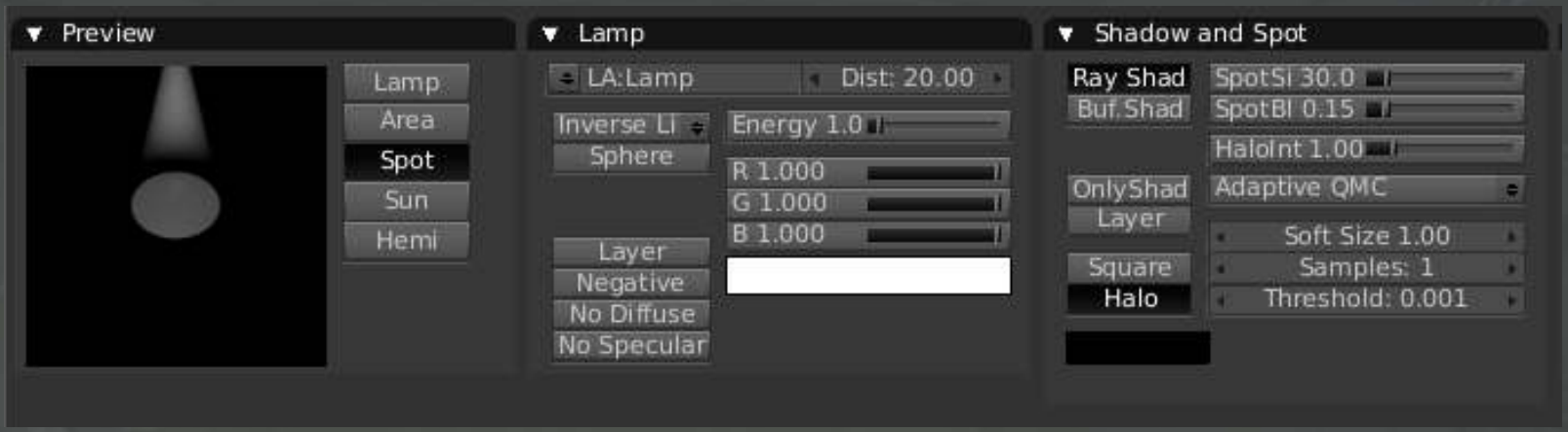
Проекция изображения на «экран»



БЛАГОДАРЯ поддержке прозрачных теней в Blender вы можете без особого труда добиться любопытнейших эффектов. Один из них - «кинопроектор»: прозрачное изображение просвечивается, и полученная тень отбрасывается на экран напротив.

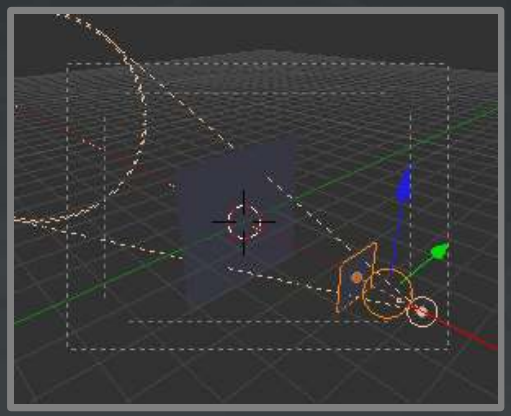
1. Создайте плоскость и поверните ее вертикально. Это у нас будет экран.

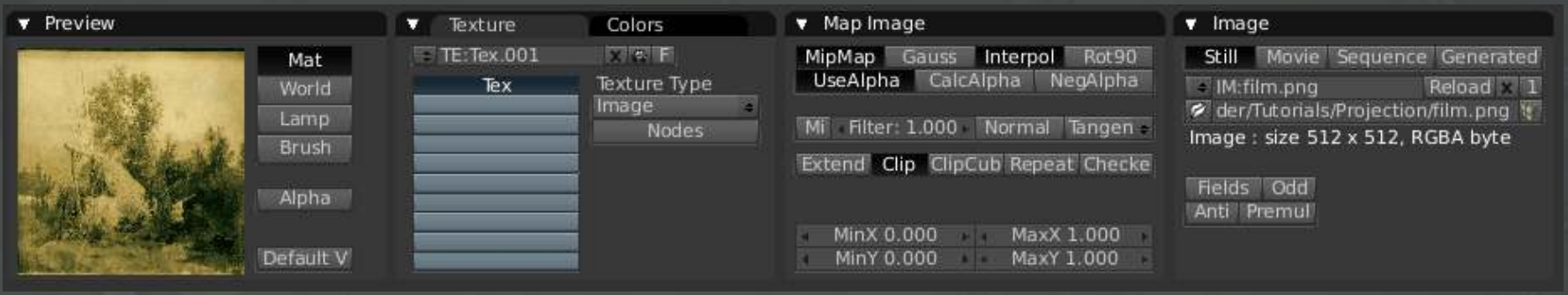
2. Создайте новую лампу и укажите ей тип Spot. Поместите лампу на нужном расстоянии от экрана и поверните на 90 градусов. Параметр SpotSi на вкладке Shadow and Spot измените на 30. Там же включите кнопки RayShad и Halo.



3. Создайте новую плоскость, поменьше, укажите ей в качестве родителя лампу и поместите на близком расстоянии от лампы. Если нужно - измените пропорции. Эта плоскость будет выступать в качестве слайда.

4. Создайте для слайда материал. На вкладке Links and Pipeline включите кнопку Ztransp. На вкладке Material уменьшите значение Alpha до 0.5. Добавьте новую текстуру типа Image и загрузите картинку. Я использовал для этой цели слегка отредактированную репродукцию картины И. И. Шишкина “Шалаш”.

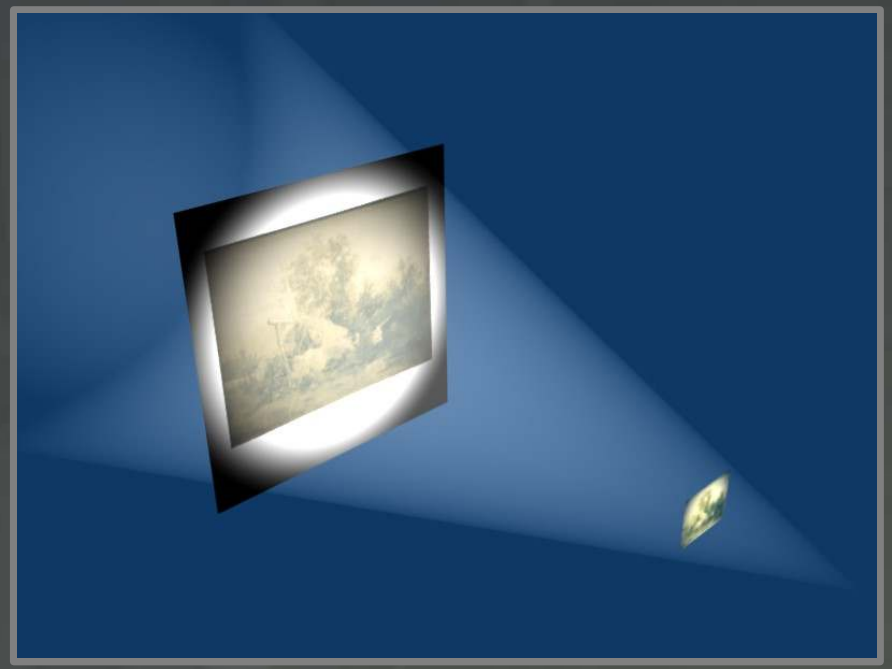




5. Выберите плоскость-экран и создайте ей новый материал. На вкладке Shaders включите кнопку TraShad.

6. Нажмите F12 и посмотрите результат.

Gecko
clocktower89@mail.ru



Вы разрабатываете перспективный проект? Открыли интересный сайт? Хотите «раскрутить» свою команду или студию? Мы Вам поможем!

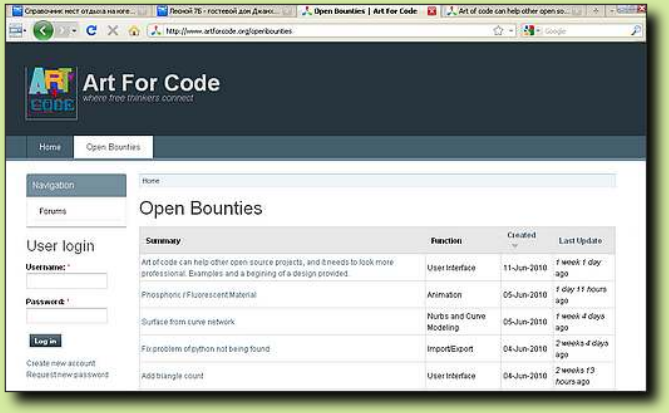
Спецпредложение от «FPS»!

«FPS» предлагает уникальную возможность: совершенно БЕСПЛАТНО разместить на страницах журнала рекламу Вашего проекта!! При этом от Вас требуется минимум:

- **Соответствие рекламируемого общей тематике журнала**. Это может быть игра, программное обеспечение для разработчиков, какой-либо движок и/или SDK, а также любой другой ресурс в рамках игростроя (включая сайты по программированию, графике, звуку и т.д.). Заявки, не отвечающие этому требованию, рассматриваться не будут.
- **Готовый баннер или рекламный лист**. Для баннеров приемлемое разрешение: 800x200 (формат JPG, сжатие 100%). Для рекламных листов: 1000x700 (формат JPG, сжатие 90%). Содержание — произвольное, но не выходящее за рамки общепринятого и соответствующее грамматическим нормам. Совет: к созданию рекламного листа рекомендуем отнестись ответственно. Если не можете сами качественно оформить рекламу, найдите подходящего художника. !! «Голый» текст без графики и оформления не принимается.
- Краткое описание Вашего проекта и — обязательно — **ссылка на соответствующий сайт** (рекламу без ссылки не публикуем).
- Заявки со включенными **дополнительными материалами для журнала** (статьи, обзоры и т.д.) не только приветствуются, но даже более приоритетны.

Заявки на рекламу принимаются на почтовый ящик редакции: clocktower89@mail.ru (тема: «Сотрудничество с FPS», а не просто «Реклама», так как ее может отсеять спам-фильтр).

Прикрепленные материалы (рекламный лист, информация, статьи и пр.) могут быть как прикреплены к письму, так и загружены на какой-либо надежный сервер или файлохранилище (но не RapidShare или DepositFiles!). Все материалы желательно архивировать в формате ZIP, RAR или 7Z.



http://www.artforcode.org/

НЕДАВНО открылся интересный проект Art for Code ("арт в обмен на код"), главная цель которого — дать возможность художникам и программистам (в первую очередь, разработчикам игр) взаимно помочь друг другу.

Идея заключается в следующем: художники составляют список улучшений, которые бы они хотели видеть в 3D-редакторе Blender (например, поддержка импорта/экспорта в определенный формат файла, улучшения пользовательского интерфейса и т.д.). Любой желающий, которому под силу справиться с указанной задачей, может получить в качестве вознаграждения за её решение игровой контент — 3D-модель или текстуру (это заранее оговаривается с художником).

Сайт проекта выступает в качестве посредника между программистами и художниками; на нем вы можете найти список актуальных задач для решения.

Размытие при движении

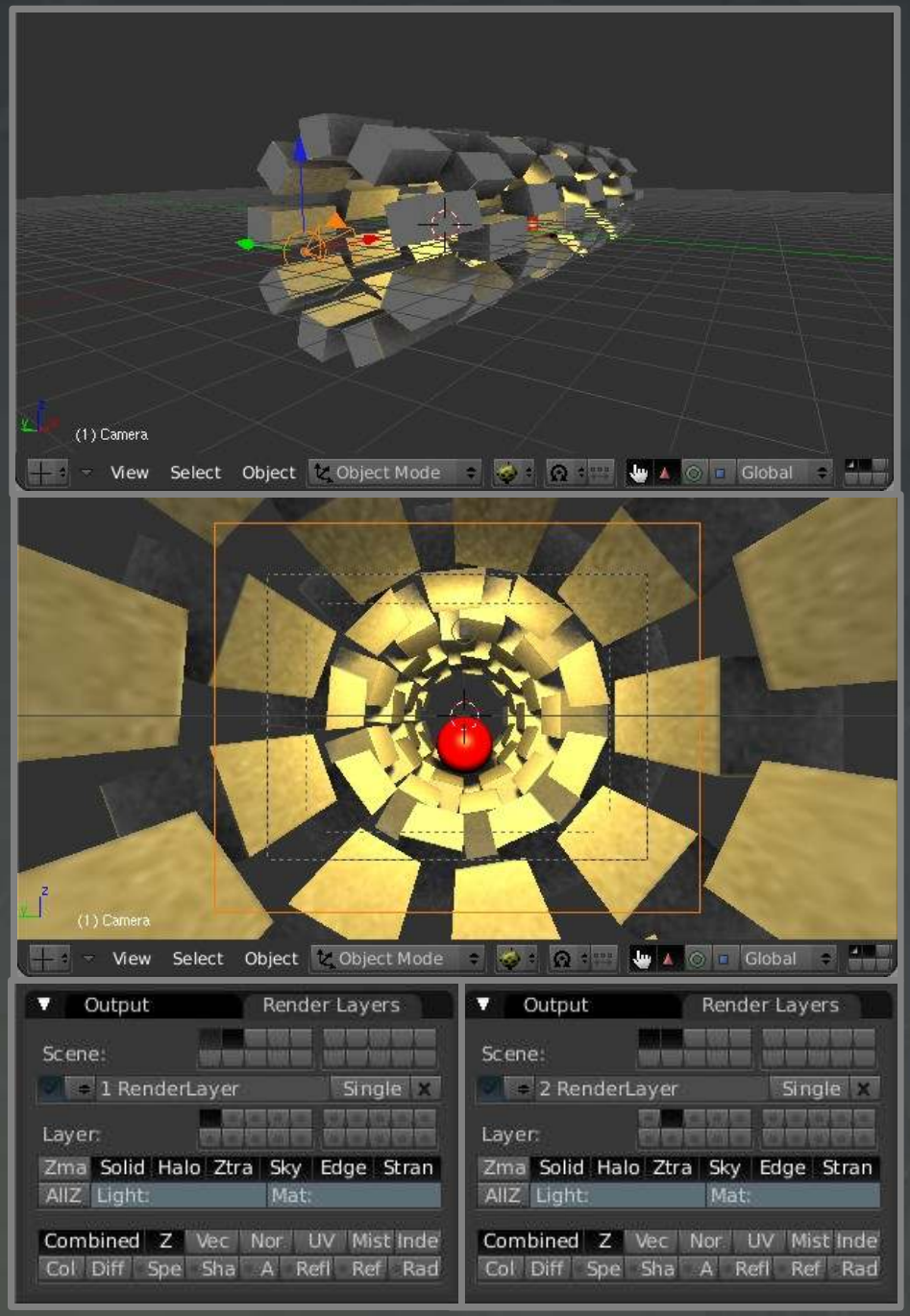
ЭФФЕКТ размытия при движении (motion blur) можно передавать различными способами. Сегодня мы рассмотрим тот его случай, когда размывается окружающее пространство, тогда как сам движущийся объект остается четким. Так происходит, если камера движется с той же скоростью, что и объект. Я рассматриваю эффект только для статичного изображения, но ничто не мешает вам адаптировать его для анимации.

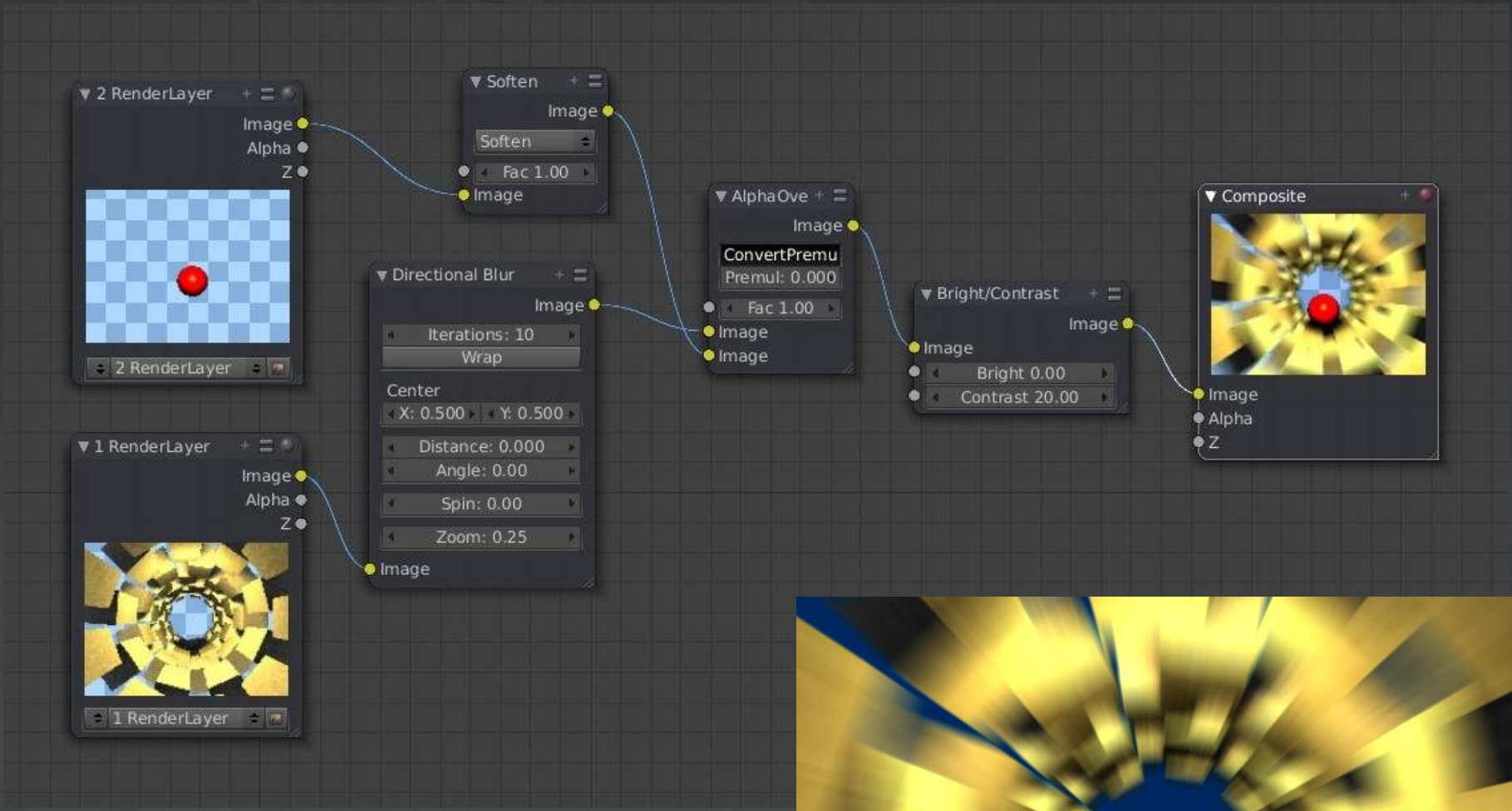
Урок частично основан на «Эффекте свечения» (стр. 3).

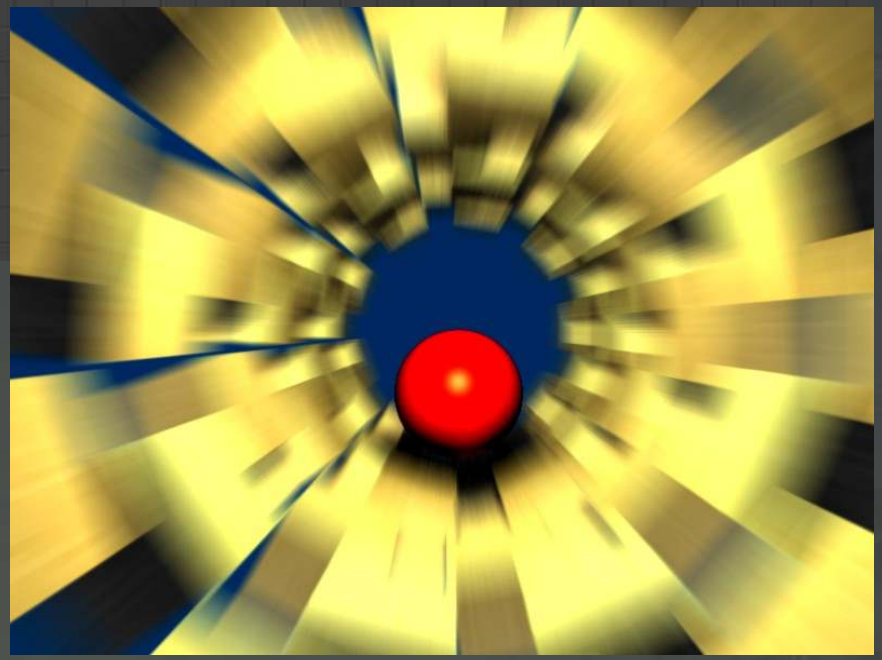
1. Для начала нам нужна какя-нибудь сцена. Для демонстрации эффекта лучше всего подойдет туннель с неровными стенками. Я смоделировал туннель из отдельных блоков, составленных в шахматном порядке. Внутри него находится «движущийся» объект — красный шар.

2. Поскольку размытие можно применять только к уже отрендеренному изображению, необходимо отделить туннель от шара. Разместите туннель на первом слое, а шар — на втором.

3. В настройках сцены на вкладке Render Layers создайте два слоя рендеринга и укажите для них слои сцены.





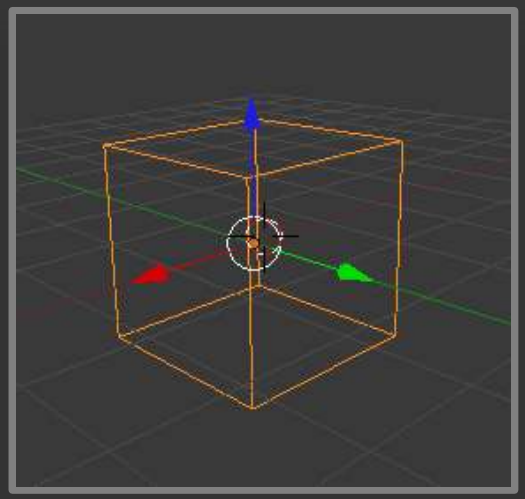


4. Самое интересное — серия операций над полученными изображениями в редакторе композитных узлов. Слой с туннелем размывается при помощи фильтра Directional Blur, затем два изображения соединяются в одно при помощи узла AlphaOver, а результат корректируется путем повышения контраста.

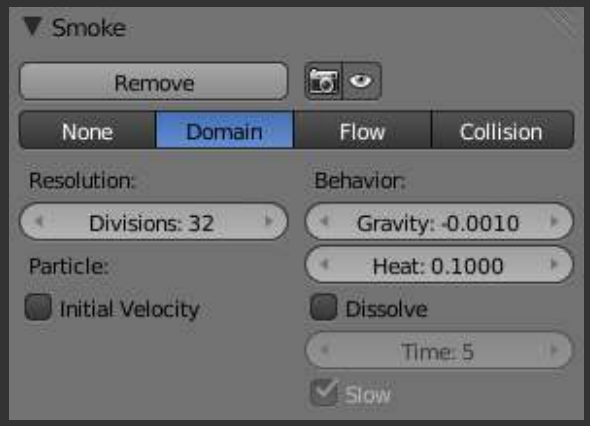
Gecko
clocktower89@mail.ru

Blender 2.5 Alpha 2 :: Симуляция дыма

СИМУЛЯЦИЯ дыма в Blender 2.5 уже успела стать притчей во языцех. Стоит ввести «Blender 2.5» в поисковую строку Google, и в предложенном списке похожих запросов вы обязательно найдете «Blender 2.5 smoke» :) Насколько можно судить по демонстрационным роликам, эффект действительно впечатляющий. Ранее нечто подобное можно было видеть в известном пакете Houdini — но он, к сожалению, коммерческий и потому доступен далеко не каждому. Однако теперь и пользователи Blender заполучили этот замечательный инструмент в свое распоряжение. Данный урок рассматривает организацию базовой симуляции с простейшими настройками. Более сложные и интересные результаты можно получить путем дальнейшего экспериментирования.

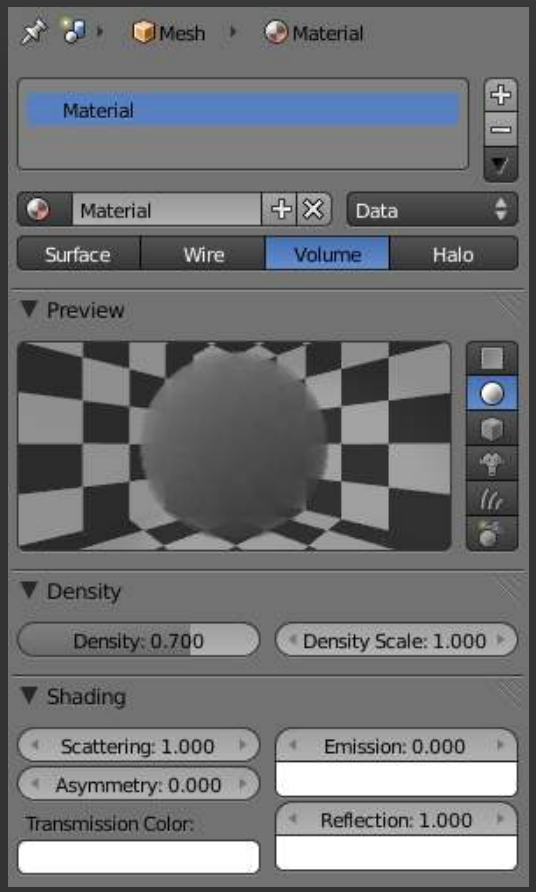


1. Создайте куб и сразу переключите его на каркасное отображение (Properties > Object > Display > Type > Wire).

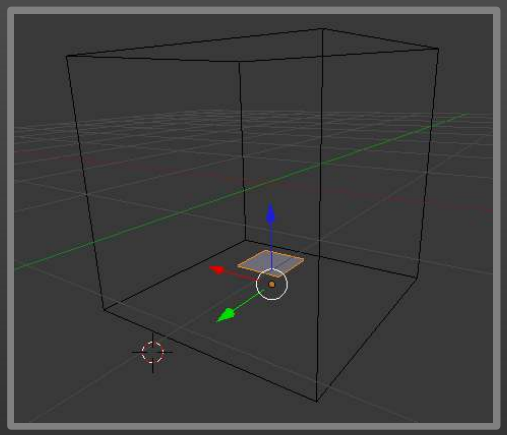


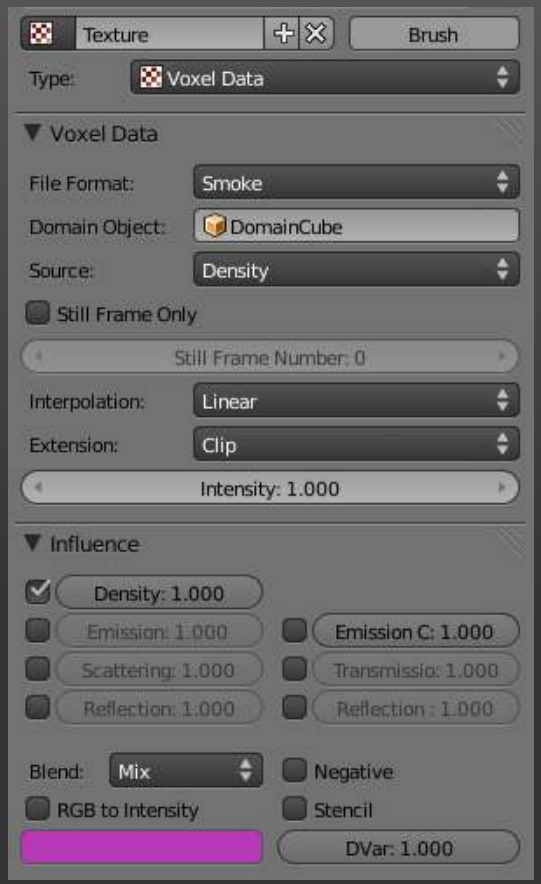
2. Перейдите в свойства физики и добавьте модификатор Smoke (Properties > Physics > Smoke > Add). Нажмите Domain.

3. Создайте кубу материал и выставьте настройки как на скриншоте.

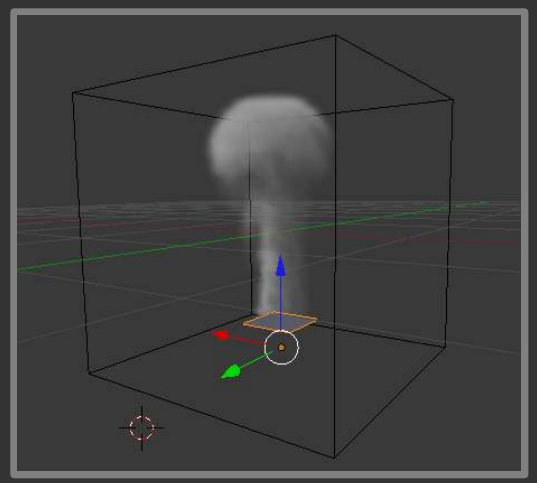


4. Добавьте новую текстуру, выберите тип Voxel Data. На вкладке Voxel Data укажите куб в свойстве Domain Object. На вкладке Influence снимите галочку с Emission C и поставьте на Density.





5. Создайте плоскость. Уменьшите ее и поместите внутрь куба-домена. В свойствах Particles добавьте ей систему частиц. На вкладке Physics нажмите No, на вкладке Display — None.



6. В свойствах физики добавьте Smoke. На этот раз нажмите Flow. В свойстве Particle System выберите созданную систему частиц.

7. Нажмите клавишу «вправо». Вы увидите, как с поверхности плоскости начнет подниматься дым. Чтобы дым шел быстрее, выделите куб-домен и повысьте свойство Heat на вкладке Smoke (например, до 4).

Gecko
clocktower89@mail.ru

DOMINANCE WAR IV

MASSIVE ART CHALLENGE ONLINE

КАЖДЫЙ год в течение шести недель художники со всего мира собираются для того, чтобы принять участие в дружеской "цифровой войне" между крупнейшими арт-коммьюнити всего мира. В течение всего конкурса у художников появляется возможность испытать себя, проверить свой уровень, познакомиться и завести новых друзей, достичь новых границ своего таланта, выиграть ценные призы.

ТЕМАТИКА конкурса - игровые персонажи. Каждый художник должен выбрать одно из нескольких представленных сообществ, которое он хотел бы представлять. Соревнование выигрывает то коммьюнити, у которого окажется лучшая работа по 3D или 2D-арту. Обе номинации независимые и судятся отдельно.

Dominance War впервые был проведен в 2006 г. при мощной поддержке ряда именитых игровых студий (Epic Games, Neversoft, Ensemble Studios, Iron Lore). Тогда, кстати, победу одержал Виталий Наймушин, сотрудник id Software. В начале 2008 года, объединив под свои знамена известнейшие интернет-ресурсы компьютерной графики (CGTalk.com, Polycount, 3DTotal) Dominance War II стал главным событием подобного рода в мире, а призовой фонд составил десятки тысяч долларов США.

В 2009 году прошла очередная мировая арт-война - DW IV. Что приятно, одним из главных золотых призеров стал наш соотечественник - Вадим Бахлычев. Его работа заняла первое место. А в номинации 2D еще один российский участник - Сергей Колесов - получил бронзу.

К сожалению, в этом году соревнование не планируется - Dominance War V перенесен на январь 2011 года.

http://www.dominancewar.com

DOMINANCE WAR
// JONATHAN FLETCHER //
// ARTPANDA.CO.UK //
V'HULCREM-KR

Unearthly Challenge

ЕСЛИ в Dominance War акцент ставится на художников по персонажам, то в этой битве участвуют пейзажисты, из-под пера и мышки которых выходят фантастические миры и локации - так называемые environment artists (художники по окружению).

Первый Unearthly Challenge был проведен в 2008 году. Его создатель - организатор Dominance War Фред Хульквист. В 2009 году помимо традиционных Polycount и GameArtisans в бой вступили команды LeewART (Китай) и CGTalk.ru.

Вам предлагается выбрать одну из сторон (светлую или темную) и создать соответствующий мир. Каким он будет - зависит только от вас, однако для победы требуется соблюдать ряд правил. Важно сделать не просто красивую картинку, но играбельную локацию. Другими словами, вы должны нарисовать или смоделировать игровой уровень, на котором было бы интересно играть.

Можно биться в одиночку (2D и 3D-арт) или объединяться в команды по трое (только 3D). При этом концепт-художники не ограничены в воображении, а вот 3D-шникам за основу придется взять один из предложенных организаторами концепт-артов.

http://www.unearthlychallenge.com

Unearthly Challenge '09 "End Game" | Full Metal Cookies (S_L_A_V_A, Druggon, Fatur) | Awaiting

WALLPAPER

Game Music Emu:

Воспроизведение музыки из приставочных игр

ГОВОРЯТ, что раньше и трава была зеленее, и вода мокрее. А еще у каждого раньше была своя любимая игровая консоль — NES, SNES, SMD, GameBoy и т.д. У кого-то даже — любительский 8-разрядный компьютер. И сегодня чуть ли не каждый третий как минимум интересуется эмуляцией этих консолей на современном компьютерном оборудовании. На почве этого интереса в среде геймдевелоперов возникла страсть к созданию римейков старых приставочных игр. Каждый фанат Марио или Соника считает своим долгом сделать собственную версию похождений любимых героев. С одной стороны, это, конечно, хорошо: старым персонажам и идеям дают новую жизнь, не давая им кануть в лету. Но это также свидетельствует и об отсутствии в любительском геймдеве чего-то совершенно нового. Люди приходят в геймдев безо всяких свежих идей. Однако я не буду судить их слишком строго: всякий, кто чем-то занят, достоин уважения. Недостоин его лишь тот, кто вместо того, чтобы действовать, убивает время, рассуждая о действиях других. Добротно сделанный клон порадует сердце геймера-ветерана не меньше, чем оригинал, запущенный в эмуляторе, не говоря уже о том, что дотошное копирование игр старой школы помогает овладеть определенными навыками и научиться создавать грамотные, сбалансированные во всех отношениях игры. Не зря многие молодые художники «набивают руку», копируя работы старых мастеров.

Если с графикой «в стиле ретро» все довольно просто (бери и срисовывай с оригинала), то с музыкой не все так однозначно. Не имея музыкального образования, лично мне, например, трудновато приходится со звуковой составляющей игры. Ну не придумали еще автоматического генератора мелодий, который заменил бы опытного композитора. Так же, как нет и графического редактора, который рисовал бы за вас спрайты и фоны. Поэтому, если у вас нет знакомого музыканта, приходится идти на ухищрения. Если требуется музыка определенной жанровой направленности, можно воспользоваться сервисами вроде Jamendo, где исполнители сами выкладывают свои композиции в свободный доступ (под лицензией Creative Commons, которая позволяет вам использовать их в некоммерческих целях). Но, по очевидным причинам, для римейков такая музыка не подходит. Можно вооружиться аудиоредактором и записать музыку из оригинала прямо во время эмуляции. Если же вы не хотите, чтобы архив с игрой весил в двадцать раз больше исполняемого файла, круг сужается до трех вариантов. Первые два — это MIDI и трекерные форматы. Если вы сумели найти саундтрек игры, скомпонованный в MIDI или MOD, вам повезло. В противном случае остается всего один беспроигрышный вариант — эмуляция звукового процессора нужной нам консоли.

Без паники :) Я, конечно, не стал бы предлагать такое, если бы не было готовой библиотеки. Она называется Game_Music_Emu (сокращенно GME) и найти ее можно на странице http://www.fly.net/~ant/libs/audio.html.

Это мультиформатный эмулятор, поддерживающий музыку ZX Spectrum, Amstrad CPC, Game Boy, Sega Mega Drive, TurboGrafx-16, PC Engine, MSX Home Computer, NES/Famicom, SNES, Atari, Sega Master System и BBC Micro. Распространяется в виде исходных кодов по лицензии GNU LGPL.

Сразу предупреждаю: придется немного попотеть со сборкой. Во-первых, ни к самой библиотеке, ни к сопутствующим демонстрационным программам make-файлов нет. Во-вторых, без ставших уже привычными танцев с бубном библиотека вряд ли соберется (хорошо хоть, дополнительных зависимостей не требует). Здесь уж кому как повезет — ведь у каждого своя версия GCC и других компонентов.

Будем думать, что вожделенный libgme.a вы таки заполучили. Попробуем написать простейший плеер на его основе. Нам понадобятся следующие заголовочные файлы из исходников GME: blargg_common.h, blargg_source.h, Gme_File.h, M3u_Playlist.h, blargg_config.h, Data_Reader.h, gme.h, Music_Emu.h. Для воспроизведения звука будем использовать SDL.

Подключаем заголовки:

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <SDL/SDL.h>
#include "include/Music_Emu.h"
#include "include/blargg_source.h"
```

Объявляем эмулятор, а также структуру, которая содержит данные о проигрываемой музыкальной дорожке (звуковые файлы некоторых типов — например, NSF — могут содержать несколько дорожек):

```
Music_Emu* emu;
track_info_t track_info;
```

Следующие три функии нужны для управления воспроизведением звука:

```
static void sdl_callback( void* data,
                Uint8* out, int count )
{
     emu->play( count/2, (short*)out );
}

void sound_stop()
{
     SDL_PauseAudio( true );
     SDL_LockAudio();
     SDL_UnlockAudio();
}

void sound_play()
{
     SDL_PauseAudio( false );
}
```

Добавляем функцию для переключения дорожек:

```
void set_track( int track )
{
     sound_stop();
     emu->start_track( track );
     emu->track_info( &track_info );
     char str[256];
     sprintf( str, "%s: %d/%d",
          track_info.game,
          track,
          emu->track_count() );
     SDL_WM_SetCaption( str, str );
     sound_play();
}
```

Эта функция также выводит в заголовок окна название игры и номер дорожки.

Теперь можно заняться и главной функцией:

```
int main( int argc, char** argv )
{
```

Инициализируем SDL:

```
if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO)<0)
      return EXIT_FAILURE;
atexit( SDL_Quit );
SDL_EnableKeyRepeat( 500, 80 );
SDL_SetVideoMode( 480, 240, 0, 0 );
```

Определяем частоту дискретизации и стартовый номер дорожки:

```
int sample_rate = 44100;
int track = 0;
```

Включаем аудио:

```
SDL_AudioSpec as;
as.freq     = sample_rate;
as.format   = AUDIO_S16SYS;
as.channels = 2;
as.callback = sdl_callback;
as.samples  = 1024;
if ( SDL_OpenAudio(&as, 0) < 0 )
      return EXIT_FAILURE;
```

Загружаем файл и запускаем воспроизведение (путь к файлу задается первым параметром к программе):

```
gme_open_file( argv[argc-1], &emu, sample_rate );
set_track( track );
```

Организуем цикл, который будет повторяться, пока пользователь не закроет окно SDL или не нажмет клавишу Escape:

```
bool running = true;
bool pause = false;
while ( running )
{
      SDL_Delay(10);
      SDL_Event event;
      while ( SDL_PollEvent( &event ) )
      {
            switch ( event.type )
            {
                  case SDL_QUIT:
                        running = false;
                        break;

                  case SDL_KEYDOWN:
                        int key = event.key.keysym.sym;
                        switch ( key )
                        {

                  case SDLK_RIGHT:
                        if (track < emu->track_count())
                        set_track(++track);
                        break;

                  case SDLK_LEFT:
                        if (track > 0)
                        set_track(--track);
                        break;

                  case SDLK_SPACE:
                        if (pause) sound_play();
                        else sound_stop();
                        pause = !pause;
                        break;

                  case SDLK_ESCAPE:
                        running = false;
                        break;
                        }
            }
      }
}
```

В теле цикла также имеются проверки нажатия клавиш «вправо», «влево» и «пробел», которые вызывают, соответственно, переход к следующей/предыдущей дорожке и паузу/возобновление.

Наконец, по завершении цикла отключаем звук и завершаем работу программы:

```
    sound_stop();
    SDL_CloseAudio();
    return EXIT_SUCCESS;
}
```

Собираем командой

g++ -O3 -o player main.cpp -L"." libgme.a -lSDL

и запускаем:

./player musicfile.xxx

Нужно отметить, что Game_Music_Emu совершенно независима от каких-либо интерфейсов воспроизведения звука, будь то SDL или что-то еще. При желании ее можно «подружить» с DirectSound, OpenAL и т.д. В нашем примере мы использовали SDL, так как это наиболее простой вариант.

И напоследок — пару слов о приставочной музыке. Некоторые эмуляторы поддерживают извлечение музыкальной дорожки в файл. Но, как правило, необходимость в этом не возникает, так как почти из всех популярных игр музыку уже извлекли. К примеру, неплохую коллекцию саундтреков из игр NES, SNES, SMD можно найти на сайте http://www.zophar.net.

А желающим писать музыку самостоятельно можно посоветовать FamiTracker (http://famitracker.shoodot.net/) — пятиканальный трекер с поддержкой экспорта в NSF (формат музыки NES).

Gecko
clocktower89@mail.ru

Виртуальная файловая система
Первые шаги в PhysFS

ЧТЕНИЕ и запись файлов — очень важная область задач для игрового программиста. Для небольших, локальных проектов в этом отношении сгодятся и стандартные средства ввода-вывода C/C++, ставшие де-факто частью языка (хотя на деле это не совсем так — эти средства реализованы на основе системных библиотек). Однако, стоит только взяться за проект покрупнее, или же, выйдя за пределы родной Windows, попытаться «подружить» свое детище с другими ОС, становится ясно — необходимо решение уровнем повыше. Точно так же, как многие кроссплатформенные движки обеспечивают абстракцию поверх OpenGL и DirectX, предлагая унифицированный интерфейс и маскируя для программиста разницу между этими двумя API, существуют и абстракции над файловыми системами. Но для чего, спросите вы, они нужны, если стандартные средства ввода-вывода C/C++ и так везде одинаковые? Отвечу.

Во-первых, безопасность. Очень часто программисты, привыкшие в Windows всегда работать из-под учетной записи администратора (и, соответственно, писать программы, работающие только из-под нее), забывают, что, строго говоря, это не есть хороший тон. В NT-совместимых версиях Windows существует распределение прав доступа, которое все почему-то игнорировали. До недавнего времени — когда вышла Vista с ее более строгим отношением к правам доступа, все эти недочеты всплыли на поверхность. Я лично столкнулся с проблемой, когда Delphi 7 отказался запускаться из-под обычного пользователя. Собственно, конкретно тут Microsoft упрекнуть не в чем — они усложнили жизнь вирусописателям, а у Vista есть проблемы посерьезней, чем совместимость со старым прикладным ПО.

Но даже Vista в отношении безопасности и распределения прав доступа и рядом не стоит с UNIX-подобными системами. И причина здесь не только в ориентации *nix на сервера, и даже не в пресловутых недочетах «умельцев» из MS. Программисты, администраторы и пользователи Linux, BSD и др. систем сами весьма добросовестно относятся к вопросам безопасности, в то время как в Windows сложилась такая ситуация, что порой не отличишь полезную, безобидную программу от вредоносной. Задумайтесь: с позиции юниксоида, все эти так называемые «твикеры», «тюнеры», «оптимизаторы», «дефрагментаторы» и прочие столь популярные сейчас сомнительные поделки, вытворяющие под видом благих намерений невесть что в системных папках и реестре Windows (в святая святых!), да еще распространяющиеся под закрытой лицензией, не вызывают ни грамма доверия, а их бездумное использование (которое, между прочим, зачастую ошибочно принимается за признак опытности пользователя) выглядит как дикость и надругательство над собственной системой. Отсюда и репутация Windows как излюбленного полигона для компьютерных мошенников и злоумышленников — никто, от программистов до юзеров, не думает о правах доступа. Windows доверчиво позволяет программам, запущенным от имени администратора (а иначе их, спасибо программистам, и не запустишь) залезать в любые каталоги и делать все, что им заблагорассудится. Совсем иная ситуация при использовании виртуальной файловой системы, которая строго регламентирует возможности чтения и записи, ограничивая их только теми каталогами, которые укажет программист.

Во-вторых, независимость от особенностей физической файловой системы. В Linux каталоги разделяются обычной наклонной чертой, в Windows — обратной, в Mac OS – вообще двоеточием... А буквы дисков? А пути к дисководу и домашней папке пользователя? Голова идет кругом! Виртуальная ФС скрывает все эти индивидуальные различия, облегчая тем самым портирование игры на другие платформы.

В-третьих, данные (ресурсы) игры могут находится в самых разных местах (в рабочей директории, на оптическом диске, в домашней папке — если это какие-нибудь аддоны или патчи). Они могут иметь разные версии и разный приоритет при загрузке. Пользовательские файлы конфигурации, например, должны перекрывать дефолтные, и так далее. Наконец, любая уважающая себя крупная игра должна экономить место на винчестере и хранить данные в сжатых архивах — однако не всегда бывает возможность распаковывать их перед загрузкой. Виртуальная ФС и здесь окажется незаменимым помощником.

На мой взгляд, лучшая ВФС, полностью отвечающая всем эти требованиям, будучи легковесной, открытой и легкой в освоении — это PhysFS (http://icculus.org/physfs).

ВФС PhysFS состоит из каталогов, которые должны быть явным образом определены перед тем, как читать или записывать файлы. Эти каталоги представляют собой директории или архивы в реальной файловой системе. При этом доступ к содержимому архива осуществляется как если бы это была директория. PhysFS поддерживает архивы **ZIP**, **GRP** (Build Engine), **PAK** (Quake I и II), **HOG** (Descent I и II), **MVL** (Descent II), **WAD** (Doom). Архивы не обязаны иметь соответствующее расширение, его можно изменить на любое другое — PhysFS определяет тип архива по его содержимому.

Предположим, что ресурсы игры (текстуры, модели, скрипты и т.д.) могут храниться в различных местах с разным приоритетом: не только в рабочей директории игрового приложения, но и в домашней папке пользователя, а также, возможно, в архиве на компакт-диске с игрой. Соответственно, в PhysFS можно добавить следующие пути:

C:\Documents and Settings\user\game\data — папка data в директории игры в домашней папке;

.\data — папка data в рабочей директории игры;

D:\data.zip — архив data.zip на диске с игрой.

Эти пути называются путями поиска (search path). При чтении файла PhysFS будет искать его сначала в домашней папке, затем в рабочей директории, затем в архиве на диске. При этом путь к файлу указывается относительно пути поиска. Например, в данной ситуации файл C:\Documents and Settings\user\game\data\textures\image.jpg будет «виден» для PhysFS как textures/image.jpg. То же самое — в отношении архивов: D:\data.zip\textures\image.jpg = textures/image.jpg.

Из соображений безопасности, в PhysFS ограничены возможности записи. PhysFS может записывать только в одну директорию (хотя Вы можете ее переопределять всякий раз, когда это требуется). Директория записи не добавляется автоматически в список путей поиска.

Рекомендуемая очередность добавления путей поиска такова: директория записи, рабочая директория, компакт-диск, архивы.

PhysFS по умолчанию не распознает символьные ссылки (которые встречаются в некоторых архивах и файловых системах UNIX). Вы можете разрешить символьные ссылки функцией PHYSFS_permitSymbolicLinks, но имейте в виду, что ссылки могут указывать на файлы и директории вне области доступа PhysFS, что небезопасно.

Использовать PhysFS даже проще, чем стандартную библиотеку ввода/вывода. Рассмотрим пример чтения текстового файла из архива. Сначала подключим необходимые заголовки:

```
#include <stdio.h>
#include "physfs.h"
```

Дальнейший код помещается внутрь главной функции:

```
int main (int argc, char *argv[])
{

}
```

Инициализация PhysFS выглядит следующим образом:

```
PHYSFS_init(argv[0]);
```

В эту функцию мы передаем первый параметр приложения. Этого можно не делать, но рекомендуется.

Добавляем архив:

```
PHYSFS_addToSearchPath( "myarchive.zip", 1);
```

Значение 1 указывает, что данный путь должен иметь низший приоритет при поиске. Если изменить его на 0, приоритет будет высшим.

Определяем нужный нам файл и проверяем, существует ли он:

```
char* filename = "file.txt";
if ( !PHYSFS_exists(filename) )
{
      printf( "physfs error: no such file\n" );
      return 1;
}
```

Если файл не существует ни в одном из путей поиска, программа выдаст ошибку и завершит свою работу.

Открываем файл для чтения и узнаем его размер в байтах:

```
PHYSFS_file* myfile =
      PHYSFS_openRead( filename );
int length = PHYSFS_fileLength(myfile);
```

Объявляем массив байтов и выделяем под него нужный объем памяти:

```
char *buffer = new char[length];
```

Копируем данные из файла в массив:

```
PHYSFS_read(myfile, buffer, 1, length);
```

Массив должен оканчиваться нулем:

```
buffer[length-1] = 0;
```

Поскольку данные уже в памяти, файл нам уже не нужен:

```
PHYSFS_close(myfile);
```

Если мы завершили работу с PhysFS, ее необходимо выгрузить:

```
PHYSFS_deinit();
```

Выводим текст, удаляем его из памяти и завершаем работу программы:

```
printf("%s\n",buffer);
delete buffer;
return 0;
```

Вот полный листинг:

```
#include <stdio.h>
#include "physfs.h"

int main (int argc, char *argv[])
{
     PHYSFS_init(argv[0]);
     PHYSFS_addToSearchPath( "myarchive.zip", 1);
     char* filename = "file.txt";

     if ( !PHYSFS_exists(filename) )
     {
          printf( "physfs error: no such file\n" );
          return 1;
     }

     PHYSFS_file* myfile = PHYSFS_openRead( filename );
     int length = PHYSFS_fileLength(myfile);
     char *buffer = new char[length];
     PHYSFS_read(myfile, buffer, 1, length);
     buffer[length-1] = 0;
     PHYSFS_close(myfile);
     PHYSFS_deinit();

     printf("%s\n",buffer);

     delete buffer;
     return 0;
}
```

Gecko
clocktower89@mail.ru

Парсер своими руками

Создаем собственный язык файлов конфигурации на C++

ПРАКТИЧЕСКИ любой крупной программе необходим способ сохранять и загружать пользовательские настройки. Для этого придумано множество способов: кто-то предпочитает XML, в Windows часто используют ini-файлы и реестр, а кто-то создает собственный формат конфигов. Именно этим мы и займемся. В данной статье приведен один из способов анализа конфигурационных файлов с поддержкой типов данных, строковых и числовых значений, комментариев, а также многоуровневых ссылок. Вот пример такого файла:

```
# Configuration file example
string qw = "Hello World!"
string Word = qw
int One = 1
int Four = 4
float Value = 12.5
string WordLink = Word
int NumLink = Four
```

Анализатор (парсер) будет «превращать» эти данные в специальные объекты, хранящие имена, значения и типы прочитанных параметров.

```
#include <stdio.h>
#include <stdlib.h>
#include <cstdlib>
#include <sys/stat.h>
#include <string.h>

enum type { KEYWORD, TOKEN, DATA, NAME, END };

struct Token
{
      char* Name;
      type Expecting;
      type Type;
};

struct CharDef
{
      char Src;
      char Dest;
      char* Type;
};

class Object
{
      public:
      char* Type;
      char* Name;
      char* Data;
      Object();
      ~Object();
      char* GetType();
      char* GetName();
      char* GetDataString();
      int GetDataInt();
      float GetDataFloat();
};
```

```
class Object
{
      public:
      char* Type;
      char* Name;
      char* Data;
      Object();
      ~Object();
      char* GetType();
      char* GetName();
      char* GetDataString();
      int GetDataInt();
      float GetDataFloat();
};

Object::Object() {};

Object::~Object() {};

char* Object::GetType()
{
      return Type;
}

char* Object::GetName()
{
      return Name;
}

char* Object::GetDataString()
{
      return Data;
}

int Object::GetDataInt()
{
      return atoi(Data);
}

float Object::GetDataFloat()
{
      return atof(Data);
}
```

```
class CfgParser
{
      public:
      char* newlinestr;
      char newline;
      int MaxWordLength;
      char* buffer;
      int MaxKeywords;
      int Keywords;
      Token* kwdlist;
      int MaxCharDefs;
      int CharDefs;
      CharDef* chardeflist;
      int MaxObjects;
      Object* Database;
      int ObjectsCount;
      char CommentsChar;
      char StringChar;
      CfgParser();
      ~CfgParser();
      void Init();
      void DefineChar(char src, char dest, char* itype);
      void SetCommentsChar(char chr);
      void SetStringChar(char chr);
      void RegisterIdentifier(char* iname, type itype,
                              type iexpect);
      void RegisterType(char* iname);
      void RegisterAssignment(char* iname);
      void RegisterEnd(char* iname);
      Object* GetObject(char* name);
      Object* GetObject(int index);
      bool ObjectExists(char* name);
      int ParseFile(char* Source, int Length,
                    bool dbgmode);
};

Object::Object() {};

Object::~Object() {};

char* Object::GetType()
{
      return Type;
}
```

```
char* Object::GetName()
{
      return Name;
}

char* Object::GetDataString()
{
      return Data;
}

int Object::GetDataInt()
{
      return atoi(Data);
}

float Object::GetDataFloat()
{
      return atof(Data);
}

CfgParser::CfgParser() {}
CfgParser::~CfgParser() {}

void CfgParser::Init()
{
      newlinestr = (char*)"\n";
      newline = newlinestr[1];

      MaxWordLength = 128;
      MaxKeywords = 128;
      Keywords = 0;
      MaxCharDefs = 128;
      CharDefs = 0;
      MaxObjects = 128;
      ObjectsCount = 0;
      CommentsChar = '#';
      StringChar = '\047';

      buffer = new char[MaxWordLength];
      kwdlist = new Token[MaxKeywords];
      chardeflist = new CharDef[MaxCharDefs];
      Database = new Object[MaxObjects];
}
```

```
void CfgParser::DefineChar(char src, char dest, char* itype)
{
      chardeflist[CharDefs].Src = src;
      chardeflist[CharDefs].Dest = dest;
      chardeflist[CharDefs].Type = new char[MaxWordLength];
      strcpy(chardeflist[CharDefs].Type,itype);
      CharDefs++;
}

void CfgParser::SetCommentsChar(char chr)
{
      CommentsChar = chr;
}

void CfgParser::SetStringChar(char chr)
{
      StringChar = chr;
}

void CfgParser::RegisterIdentifier(char* iname, type itype,
                                   type iexpect)
{
      kwdlist[Keywords].Name = iname;
      kwdlist[Keywords].Type = itype;
      kwdlist[Keywords].Expecting = iexpect;
      Keywords++;
}

void CfgParser::RegisterType(char* iname)
{
      kwdlist[Keywords].Name = iname;
      kwdlist[Keywords].Type = KEYWORD;
      kwdlist[Keywords].Expecting = NAME;
      Keywords++;
}

void CfgParser::RegisterAssignment(char* iname)
{
      kwdlist[Keywords].Name = iname;
      kwdlist[Keywords].Type = TOKEN;
      kwdlist[Keywords].Expecting = DATA;
      Keywords++;
}
```

```
void CfgParser::RegisterEnd(char* iname)
{
      kwdlist[Keywords].Name = iname;
      kwdlist[Keywords].Type = KEYWORD;
      kwdlist[Keywords].Expecting = END;
      Keywords++;
}

Object* CfgParser::GetObject(char* name)
{
      for (int iterator_objects=0;
            iterator_objects<=ObjectsCount;
            iterator_objects++)
      {
            if (!strcmp(
            Database[iterator_objects].Name,
            name))
            {
                  return
                  &(Database[iterator_objects]);
            }
      }
      return NULL;
}

Object* CfgParser::GetObject(int index)
{
      if (index<=ObjectsCount)
            return &(Database[index]);
      else return NULL;
}

bool CfgParser::ObjectExists(char* name)
{
      for (int iterator_objects=0;
            iterator_objects<=ObjectsCount;
            iterator_objects++)
      {
            if (!strcmp(
            Database[iterator_objects].Name,
            name))
                  return true;
      }
      return false;
}
```

```
int CfgParser::ParseFile(char* Source, int Length,
                         bool dbgmode)
{
      int ii = 0;
      type expecting = KEYWORD;
      int line = 1;
      type ExpectAfterName = TOKEN;
      type ExpectAfterData = KEYWORD;
      int iterator_db = -1;
      int state = 0;
      int ret = 0;
      int comment = 0;
      int thestring = 0;
      printf("libcfgp: Parsing...\n");
      for (int i=0; i<=Length; i++)
      {
            if (Source[i]==StringChar and comment==0)
            {
                  if (thestring==0) thestring = 1;
                  else thestring = 0;
            }
            else if (Source[i]!=' '
            and Source[i]!='\012'
            and Source[i]!='\011')
            {
                  if (Source[i]==CommentsChar)
                  {
                        if (comment==0) comment=1;
                        else comment=0;
                  }
                  buffer[ii] = Source[i];
                  ii++;
            }
            else
            {
                  if (Source[i]=='\012')
                  //newline (end of comment)
                  {
                        if (comment==1) comment=0;
                  }
                  if (comment==1) continue;
```

```
        if (thestring==0)
        {
                state = 0;
                ret=0;
                for (int j=ii; j<(MaxWordLength-ii); j++)
                        buffer[j]='\000';
                ii = 0;
                for (int k=0; k<=Keywords-1; k++)
                {
                        if ((!strcmp(buffer,kwdlist[k].Name)) and ret==0)
                        {
                                if (expecting!=kwdlist[k].Type)
                                {
                                        printf("%d: error: unexpected '%s'\n",
                                                line,buffer);
                                        state=3;
                                        break;
                                }
                        expecting=kwdlist[k].Expecting;
                        if (dbgmode)
                                printf("libcfgp: %d: Detected keyword: %s\n",
                                        line,buffer);
                        if (kwdlist[k].Type!=TOKEN)
                        {
                                iterator_db++;
                                Database[iterator_db].Type =
                                        new char[MaxWordLength];
                                strcpy(Database[iterator_db].Type,buffer);
                        }
                        state = 1;
                        ret = 1;
                        break;
                }
        }
}
else { expecting==DATA; buffer[ii] = Source[i]; ii++; }
```

```
if (state==0)
{
      if (expecting==NAME)
      {
            if (strcmp(buffer,""))
            {
                  expecting = ExpectAfterName;
                  if (dbgmode)
                  printf("libcfgp: %d: Detected name: %s\n",line,buffer);
                  Database[iterator_db].Name = new char[MaxWordLength];
                  strcpy(Database[iterator_db].Name,buffer);
            }
      }
      if (expecting==DATA)
      {
            if (strcmp(buffer,""))
            {
                  expecting = ExpectAfterData;
                  if (dbgmode)
                  printf("libcfgp: %d: Detected data: %s\n",line,buffer);
                  Database[iterator_db].Data = new char[MaxWordLength];
                  if (thestring==1)
                  {
                        sprintf(Database[iterator_db].Data,
                        "%s%s%c",
                        Database[iterator_db].Data,buffer,' ');
                  }
                  else strcpy(Database[iterator_db].Data,buffer);
                  //Process links
                  for (int iterator_objects=0; iterator_objects<iterator_db;
                  iterator_objects++)
                  {
                        if (!strcmp(Database[iterator_objects].Name,
                                    Database[iterator_db].Data))
                        {
                              if (dbgmode)
                              printf("libcfgp: %d: Detected link\n",line);
                              if (!strcmp(Database[iterator_objects].Type,
                                          Database[iterator_db].Type))
                                    Database[iterator_db].Data =
                                    Database[iterator_objects].Data;
                        }
                  }
```

```
                                    //Process character definitions
                                    for (int iterator_chars=0;
                                    iterator_chars<MaxWordLength; iterator_chars++)
                                    {
                                          for (int iterator_chardefs=0; iterator_chardefs<CharDefs;
                                                iterator_chardefs++)
                                          {
                                                if (!strcmp(Database[iterator_db].Type,
                                                            chardeflist[iterator_chardefs].Type))
                                                {
                                                      if (Database[iterator_db].Data[iterator_chars]==
                                                            chardeflist[iterator_chardefs].Src)
                                                      Database[iterator_db].Data[iterator_chars]=
                                                            chardeflist[iterator_chardefs].Dest;
                                                }
                                          }
                                    }
                              }
                        }
                  }
                  if (state==3) { break;}
                  //Newline
                  if (Source[i]=='\012') line++;
                  state = 2;
            }
      }
      delete buffer;
      delete kwdlist;

      if (state==2) { printf("libcfgp: Parse finished at line %d\n",line); }
      if (state==3) { printf("libcfgp: Parse failed at line %d\n",line); }

      if (state==3) return 1;
      ObjectsCount = iterator_db;
      return 0;
}
```

Вот пример использования парсера на конфигурационном файле, содержащем текст, приведенный в начале статьи:

```
int FileGetSize(char *filename)
{
      struct stat fileStats;
      if(stat(filename,&fileStats)!=0) return 0;
      return fileStats.st_size;
}

char* ReadTextFile(char *filename)
{
      FILE *file = fopen( filename, "r" );
      if (file == NULL) return 0;
      int size = FileGetSize(filename);
      if (size==0) return 0;
      char *buffer = new char[size];
      int bytes = fread(buffer,1,size,file);
      buffer[bytes] = 0;
      return buffer;
}

int main (int argc, char *argv[])
{
      CfgParser Config;

      Config.Init();
      Config.RegisterType((char*)"int");
      Config.RegisterType((char*)"float");
      Config.RegisterType((char*)"string");
      Config.RegisterAssignment((char*)"=");
      Config.SetStringChar('"');

      char* Source = ReadTextFile((char*)"test.txt");
      int Length = FileGetSize((char*)"test.txt");
      if (Config.ParseFile(Source,Length,false))
            return 1;

      Object* Word = Config.GetObject((char*)"Word");
      Object* One = Config.GetObject((char*)"One");
      Object* Four = Config.GetObject((char*)"Four");
      Object* Value =
            Config.GetObject((char*)"Value");
      Object* WordLink =
            Config.GetObject((char*)"WordLink");
      Object* NumLink =
            Config.GetObject((char*)"NumLink");

      printf("One+Four: %d\n",
            One->GetDataInt() + Four->GetDataInt() );
      printf("Value: %f\n",
            Value->GetDataFloat());
      printf("WordLink: %s\n",
            WordLink->GetDataString());
      printf("NumLink: %d\n",
            NumLink->GetDataInt());

      #ifdef WIN32
            system("PAUSE");
      #endif
      return 0;
}
```

На выходе мы получим:

```
libcfgp: Parsing...
libcfgp: Parse finished at line 11
One+Four: 5
Value: 12.500000
WordLink: Hello World!
NumLink: 4
```

Gecko
clocktower89@mail.ru

Размытие
при помощи GLSL

Размытие — не такой простой эффект, как кажется на первый взгляд. Чтобы добиться высокой скорости обработки, необходима грамотная оптимизация алгоритма. В приведенном шейдере для этого горизонтальное и вертикальное размытие помещены в две отдельные функции, результаты которых затем смешиваются при помощи mix().

Шейдер можно использовать, например, для реализации эффекта glow.

Вершинная программа:

```
varying vec2 texcoords;
void main()
{
      gl_Position = ftransform();
      texcoords = vec2(gl_MultiTexCoord0);
}
```

Фрагментная программа:

```
uniform sampler2D texture;
varying vec2 texcoords;

vec4 BlurHorizontal(float scale)
{
      vec4 two = vec4(2.0, 2.0, 2.0, 1.0);

      vec4 h1 = texture2D(texture, texcoords+vec2(scale , 0.0) );
      vec4 h2 = texture2D(texture, texcoords+vec2(2.0*scale, 0.0) );
      vec4 h3 = texture2D(texture, texcoords+vec2(3.0*scale, 0.0) );
      vec4 h4 = texture2D(texture, texcoords+vec2(4.0*scale, 0.0) );
      vec4 h5 = texture2D(texture, texcoords+vec2( -scale, 0.0) );
      vec4 h6 = texture2D(texture, texcoords+vec2( -2.0*scale, 0.0) );
      vec4 h7 = texture2D(texture, texcoords+vec2( -3.0*scale, 0.0) );
      vec4 h8 = texture2D(texture, texcoords+vec2( -4.0*scale, 0.0) );
      vec4 h9 = texture2D(texture, texcoords+vec2(1.5*scale, 1.5*scale) );
      vec4 h10 = texture2D(texture, texcoords+vec2(-1.5*scale, -1.5*scale) );
      vec4 h11 = texture2D(texture, texcoords+vec2(2.5*scale, 2.5*scale) );
      vec4 h12 = texture2D(texture, texcoords+vec2(-2.5*scale, -2.5*scale) );

      vec4 temp = (two*h1 + two*h2 + two*h3 + two*h4 + two*h5 + two*h6 +
                   two*h7 + two*h8 + two*h9 + two*h10 + two*h11 + two*h12)*0.05;
      temp.a = 1.0;
      return temp;
}
```

```
vec4 BlurVertical(float scale)
{
     vec4 two = vec4(2.0, 2.0, 2.0, 1.0);

     vec4 h1 = texture2D(texture, texcoords+vec2(0.0, scale) );
     vec4 h2 = texture2D(texture, texcoords+vec2(0.0, 2.0*scale) );
     vec4 h3 = texture2D(texture, texcoords+vec2(0.0, 3.0*scale) );
     vec4 h4 = texture2D(texture, texcoords+vec2(0.0, 4.0*scale) );

     vec4 h5 = texture2D(texture, texcoords+vec2(0.0, -scale) );
     vec4 h6 = texture2D(texture, texcoords+vec2(0.0, -2.0*scale) );
     vec4 h7 = texture2D(texture, texcoords+vec2(0.0, -3.0*scale) );
     vec4 h8 = texture2D(texture, texcoords+vec2(0.0, -4.0*scale) );

     vec4 h9 = texture2D(texture, texcoords+vec2(1.5*scale, 1.5*scale) );
     vec4 h10 = texture2D(texture, texcoords+vec2(-1.5*scale, -1.5*scale) );
     vec4 h11 = texture2D(texture, texcoords+vec2(2.5*scale, 2.5*scale) );
     vec4 h12 = texture2D(texture, texcoords+vec2(-2.5*scale, -2.5*scale) );

     vec4 temp = (two*h1 + two*h2 + two*h3 + two*h4 + two*h5 + two*h6 +
                  two*h7 + two*h8 + two*h9 + two*h10 + two*h11 + two*h12)*0.05;
     temp.a = 1.0;

     return temp;
}

void main()
{
     vec4 ver = BlurVertical(0.007);
     vec4 hor = BlurHorizontal(0.007);
     gl_FragColor.xyz = mix(ver.xyz, hor.xyz, 0.5);
     gl_FragColor.a = 1.0;
}
```

Gecko
clocktower89@mail.ru

Это все!

Надеемся, номер вышел интересным. Если так, поддержите FPS! Отправляйте статьи, обзоры, интервью и прочее на любые темы касательно игр, графики, звука, программирования и т.д. на ящик редакции: clocktower89@mail.ru.

Главный редактор журнала
Gecko