Независимый электронный журнал о разработке компьютерных игр

FPS

№13
2011

Blender
История 3D-графики
Язык D: шаблоны
OpenCV

Трассировщик лучей
Эра скриптовых языков
Модели освещения
...и многое другое!

Нас читают:

gcup.ru
Все для начинающего и профессионального разработчика игр

xtreme3d.narod.ru
Сайт движка Xtreme3D

make-games.ru
Портал создания игр

gamer-club.ucoz.com
Все для создания игр без программирования и не только

mizzystic.ru
Крупнейший информационный Game Maker портал

xbobr.at.ua
Создание игр, программ, музыки

portalgame.net.ru
Игровой портал

gamesfpscreator.at.ua
Сайт для помещений игр и обсуждаловок

dapf.us
Design And Programming Forum

gameshaker.ukoz.ru
Все для редактирования и создания игр

esate.ru
Мультимедиа-сообщество

dogames.ru
Мастерская игр

www.mobkiosk.com
"Мобильный киоск"

gamecreate.ru
Создай свою игру!

igrostroenie.net.ru
Игровые движки и ресурсы

rus.game-maker.ru
Русское сообщество Game Maker

Главный редактор: Тимур Гафаров
Дизайн и верстка: Тимур Гафаров
По вопросам сотрудничества обращаться по адресу clocktower89@mail.ru или gecko0307@gmail.com.
Официальный сайт журнала: http://fps-magazine.narod.ru

Электронный журнал о разработке компьютерных игр

№13 '11

В этом выпуске:

Операторы

В предыдущем номере журнала мы рассмотрели основу модифицирования интерфейса Blender — расширение класса панелей (bpy.types.Panel) и научились выводить на панель нужную нам информацию. Однако реальное расширение интерфейса Blender невозможно без использования операторов. Все действия и команды Blender реализованы как операторы: например, оператор удаления объекта — bpy.ops.object.delete().

Оператор — это класс, наследуемый от bpy.types.Operator и в обязательном порядке содержащий метод execute (который, собственно, и выполняется при вызове оператора). Оператор также имеет имя, под которым он доступен для вызова.

Вот простейший оператор, печатающий в консоль строку «Hello, world!»:

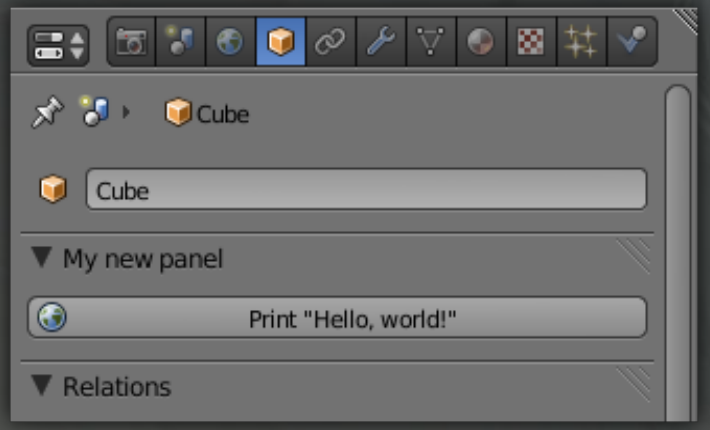
```
import bpy;

class CustomOperator(bpy.types.Operator):
    bl_label = "Print \"Hello, world!\""
    bl_idname = "hello"
    def execute(self, context):
        print("Hello, world!")
        return {'FINISHED'}
```

Так как в метод execute передается контекст, который дает доступ к данным текущего проекта, оператор можно использовать для любых действий — включая манипуляции с объектами, геометрией объектов, материалами, настройками Blender и т.д.

Графически оператор представлен в виде кнопки, которую можно расположить на панели. Мы создадим новую панель и добавим на нее наш оператор:

```
class ObjectButtonsPanel(bpy.types.Panel):
    bl_space_type = "PROPERTIES"
    bl_region_type = "WINDOW"
    bl_context = "object"
    bl_label = "My new panel"
    def draw(self, context):
        layout = self.layout
        col = layout.column()
        col.operator("hello", icon='WORLD_DATA')
```

Опциональный аргумент icon при вызове оператора (col.operator) указывает имя значка, который нужно отобразить на кнопке слева от текста. Полный список всех доступных значков Blender вы можете найти в Интернете.



Blender. Настольная книга

Трехмерная графика — прекрасная отрасль для развития навыков художника, дизайнера и программиста. Современные условия сделали ее гораздо более доступной, чем когда-либо. Для того, чтобы попробовать свои силы в этой сфере, уже не обязательно обладать специализированными рабочими станциями, суперкомпьютерами, сложным и дорогим программным обеспечением. Благодаря стремительному развитию свободного ПО и формированию глобального информационного сообщества, любой желающий сегодня может овладеть основами 3D-графики и анимации. В значительной степени это заслуга разработчиков свободного пакета трехмерного моделирования Blender, который смело можно ставить в один ряд с 3D Studio Max, Maya и прочими индустриальными гигантами.

В течение минувшего года пользователи Blender во всем мире громкими апплодисментами встречали каждое обновление новой ветки 2.5. Кому-то понравился новый интерфейс, кого-то впечатлили нововведения, кто-то оценил обновленный Python API. Порталы и блоги, посвященные Blender, начали переделывать контент под Blender 2.5. Наш журнал также не остался в стороне — весь год мы публиковали статьи и уроки, раскрывая хитрости работы в новой версии пакета. Настало время подняться на ступень выше: редакция «FPS» рада сообщить о начале работы над крупным проектом — полноценной электронной книгой, посвященной Blender 2.5!

Издание будет озаглавлено «Blender. Настольная книга». Целевая аудитория — начинающие пользователи Blender 2.5 (как перешедшие со старых версий, так и начинающие знакомство с программой «с нуля»). Книга будет оформлена в виде сборника статей, охватывающих различные аспекты работы в Blender и скомпонованных по принципу «от простого к сложному». Планируется 10 больших глав по 3-6 статей в каждой.

Ниже приведено ориентировочное содержание книги:

Глава 5. Материалы и спецэффекты

- Симуляция атмосферы.
- Подповерхностное рассеивание.
- Многослойные текстуры.
- Редактор узлов. Композитинг.
- Слои. Многослойный рендеринг.

Глава 6. Основы анимации

- Временная шкала. Интерполяционные кривые.
- Скелетная анимация. Арматура, кости, развесовка.
- Рендеринг видеоклипа.
- Монтаж фильма.

Глава 7. Физика

- Система частиц.
- Силовые поля.
- Мягкие тела и ткань.
- Симуляция жидкости.
- Симуляция дыма.

Глава 8. Скриптовый язык Python

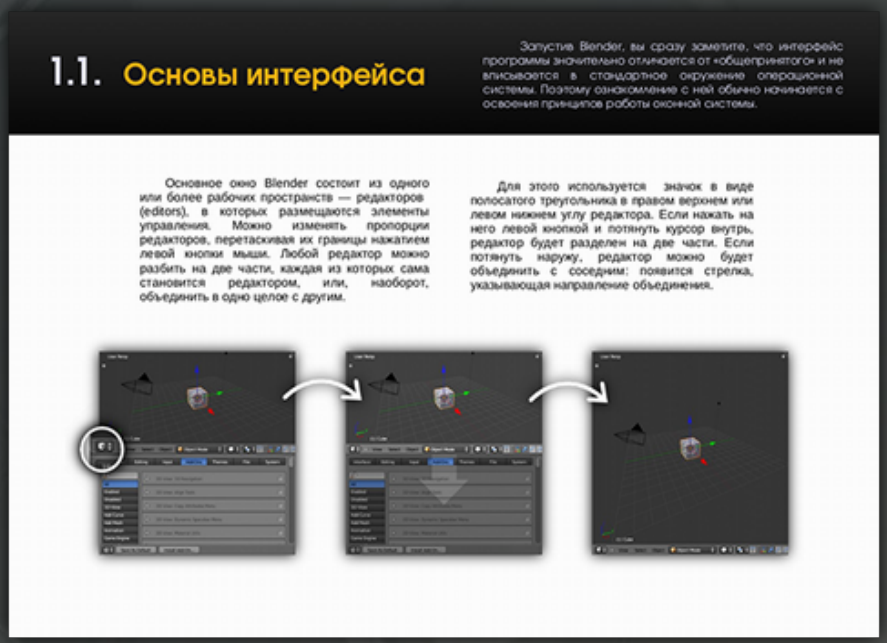
- Основы Python.
- Система расширений Blender.
- Экспорт и импорт данных.
- Модификация интерфейса Blender.

Глава 9. Игровой движок Blender

- Редактор логики. Игровые свойства.
- Карты освещения, нормалей, AO.
- Анимация персонажей.
- Игровая физика.
- Шейдеры.
- Использование Python в игровом движке.

Глава 10. Советы по работе над проектом

- Внешние и вложенные данные, ссылки, связывание.
- Обмен данными.
- Подключаемые движки рендеринга. Сетевой рендеринг.


1.1. Основы интерфейса

Запустив Blender, вы сразу заметите, что интерфейс программы значительно отличается от «общепринятого» и не вписывается в стандартное окружение операционной системы. Поэтому ознакомление с ней обычно начинается с освоения принципов работы оконной системы.

Основное окно Blender состоит из одного или более рабочих пространств — редакторов (editors), в которых размещаются элементы управления. Можно изменять пропорции редакторов, перетаскивая их границы нажатием левой кнопки мыши. Любой редактор можно разбить на две части, каждая из которых сама становится редактором, или, наоборот, объединить в одно целое с другим.

Для этого используется значок в виде полосатого треугольника в правом верхнем или левом нижнем углу редактора. Если нажать на него левой кнопкой и потянуть курсор внутрь, редактор будет разделен на две части. Если потянуть наружу, редактор можно будет объединить с соседним: появится стрелка, указывающая направление объединения.

Издание будет распространятся в формате PDF по лицензии Creative Commons BY SA. На подготовку текста книги мы отводим год, затем несколько месяцев на подготовку иллюстраций, корректировку и верстку.

К работе над книгой приглашаются все желающие! На почтовый ящик редакции (clocktower89@mail.ru или gecko0307@gmail.com) принимаются статьи и материалы по любой из вышеперечисленных тем, а также общие советы и предложения.

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 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, для рекламных листов — 1000x700. Формат — JPG или PNG. 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 на рекламу принимаются на почтовый ящик редакции: clocktower89@mail.ru или лично главному редактору: gecko0307@gmail.com (просьба в качестве темы указывать «сотрудничество с FPS», а не просто «реклама», так как письмо может отсеять спам-фильтр).

Прикрепленные материалы (рекламный лист, информация и пр.) могут быть как прикреплены к письму, так и загружены на какой-либо надежный сервер (убедительная просьба **не использовать** RapidShare, DepositFiles и другие подобные файлохранилища — загружайте файлы на свой сайт или ftp-сервер и присылайте статические ссылки). Все материалы желательно архивировать в формате zip, rar, 7z, tar.gz, tar.bz2 или tar.lzma.

Форум АНИМАТОРОВ и ХУДОЖНИКОВ
vanimatori.ucoz.ru
animator.forum2x2.ru
Создан для того, чтобы поделиться опытом с пользователями – у нас много статей и разделов. У нас можно отдохнуть, поболтать, найти друзей. Здесь Вас полностью обучат рисованию и анимации, дадут дельные советы. Также можно задать вопрос на любую тему. Есть форумная валюта (можно купить только форумные товары; снять деньги, к сожалению, нельзя). Вы можете поместить свою работу, фотографию или просто картинку в галерею.
Заходите, мы всегда будем Вам рады!

РЕСУРСЫ для игр, программ, форумов и т.д.
zacaz.zbord.ru
Принимаются заказы на спрайты, рисунки, баннеры, музыку, SAMP-сервера и многое другое.
***Цены на рисунки** вычисляются исходя из следующего тарифа:*
изображение 1x1 – 1 руб; 1x2 – 1,50 руб.
***Музыка**: 1 минута – 300 руб.*
***SAMP-сервер**: один сервер – 100 руб.*

История 3D-графики в играх

Давным-давно, когда воздух был чистым, деревья - высокими, а небезызвестная Корпорация еще не монополизировала рынок операционных систем, появились первые компьютерные игры...

Естественно, тогда трехмерной графикой даже и не пахло: первые игрушки были абсолютно плоскими и создавались в основном для игровых автоматов. Затем появилась изометрия, которой, кстати, до сих пор находят применение в играх. Но о трехмерной графике, с которой знакомы мы с вами, задумались где-то в 80-х годах XX века, хотя, говорят, уже в 70-х некоторые энтузиасты работали с wireframe-каркасами. Самой первой трехмерной игрой, использующей каркасную графику, была **Battlezone** (Atari, 1980 г.). В дальнейшем же появились реализации, работающие с затенением граней, правда, по скорости они здорово уступали спрайтам, и их использование могло быть оправдано только в случае, когда полигональных моделей было совсем немного. Прародителем современных 3D-игр считается **I, Robot** от той же Atari (1983 г.). В середине 80-х появилась первая игра линейки **Microsoft Flight Simulator**, что вполне закономерно: рисовать надо только несколько самолетиков и плоскую землю. Зато вовсю используется третье измерение. Нужно ли говорить, что игра пользовалась большой популярностью? Чисто полигональные движки тогда практически не использовались, наряду с полигональными моделями широко применялись спрайты, так что "полного 3D" еще не существовало. Наиболее ярким примером сочетания полигональных и спрайтовых объектов является игра **Alone in the Dark** (это уже 1992 год). Эта технология применяется и сегодня, хотя уже достаточно редко.

В том же 1992 году стараниями небезызвестной компании id Software свет увидел первый шутер от первого лица (FPS) - **Wolfenshtein 3D**. Хотя на самом деле Wolfenshtein'у предшествовала игра **Catacomb Abyss**, но почему-то она оказалась забыта игроками. Впервые была применена перспектива, благодаря чему объекты уменьшались, отдаляясь от игрока. Впервые (в отличие от Microsoft Flight Simulator или F-19) в этих играх применяется текстурирование - вместо однотонных граней используются масштабируемые растровые изображения.

Сердцем тогдашних компьютеров был 386-й процессор, а видеокарты отвечали только за вывод на экран монитора непосредственно кадра, не проводя никаких дополнительных вычислений. Через год происходит очередной прорыв: появляется **Doom**. Теперь игрок перемещается по полностью трехмерному уровню, правда при этом монстры по-прежнему остаются спрайтовыми, но все-таки эффект присутствия становится достаточно ощутимым, игра на самом деле затягивает.

Настоящий переворот в области игровой трехмерной графики происходит в 1996 году - выходит **Quake** от все той же id Software. Теперь полностью полигональный движок становится реальностью, появляется множество новых технологий, например, Z-буфер, а текстурирование уже подразумевается как нечто само собой разумеющееся. Правда, впервые Z-буфер был опробован еще в Doom, но теперь он становится стандартом. Кстати говоря, незадолго до этого (1995 г.), свои первые неуверенные шаги делает Game SDK, разработанный Microsoft для Windows 95. По сути, Game SDK - первенец в линейке DirectX. В это же время получает распространение среди программистов графический стандарт OpenGL (разработчик - Silicon Graphics совместно с Sun Microsystems), изначально разработанный для CAD-систем еще в 1992 году.

Тогда же (точнее, немного позднее - в 1997 году) появляются первые видеокарты с 3D-ускорением - Voodoo Graphics от 3Dfx (представитель первого поколения видеокарт). Наряду с DirectX (в 1996 г. выходит уже DirectX 2) для написания игр с аппаратным ускорением трехмерной графики используется интерфейс Glide все той же 3Dfx. Надо ли говорить, какой популярностью среди игроков (и разработчиков игр) пользовался 3D-ускоритель: адаптер мог обрабатывать до 1 миллиона треугольников и 45 миллионов пикселей в секунду, полностью брал на себя вычисления, относящиеся к 3D-сцене. Достаточно интересным было подключение Voodoo к компьютеру: так как 2D-часть на акселераторе полностью отсутствовала, для работы была необходима еще и обычная видеокарта, которая подключалась к ускорителю.

Совместное использование 3D- и 2D-карт было существенным недостатком, поэтому сразу после Voodoo Graphics выходит Voodoo Rush, сочетающая в себе оба модуля. Впрочем, качество этой карты оставляло желать лучшего, поэтому она не смогла завоевать признание игроков.

Вскоре (1998 год) выходит игра **Unreal** (Epic Megagames). В играх используются все более и более изощренные технологии (например, микрофактурные текстуры и прозрачность), рынок видеокарт с 3D-ускорением резко расширяется: на арену выходят NVIDIA, Maxtor, S3, а через некоторое время и ATI. Например, еще в конце 1997-го nVidia выпускает первый видеоадаптер, сочетающий в себе 2D- и 3D-ускорители, и при этом превосходящий по производительности Voodoo - NVIDIA Riva128.

В конце того же 1998 года выходит DirectX 6, позволивший разработчикам использовать новые технологии: stencil, W-буфер, мультитекстурирование и многое другое. 3D-технологии развиваются с невообразимой скоростью, революция следует за революцией. В конце 1999 года выходит DirectX 7 - очередной прорыв. На этот раз появляется аппаратный T&L (Transforms and Lighting) - модуль, отвечающий за преобразование геометрии и аппаратный рассчет освещения. За ускорение 3D-графики отвечают видеокарты третьего поколения. Постепенно оттесняя 3Dfx, лидером в производстве 3D-ускорителей становится NVIDIA. С появлением RivaTNT2 в 1999 году nVidia отказывается от поддержки API Glide.

Начало нового тысячелетия оказалось очень символичным: на рубеже веков были практически переломлены основные принципы построения трехмерной графики в видеокарте. На смену фиксированному конвейеру приходит программируемый, широкое распространение приобретают шейдеры. В конце 2000 года исчезает легендарная 3Dfx (компания была куплена ее главным конкурентом - NVIDIA).

Итак, для чего же были разработаны шейдеры? Дело в том, что в предыдущих поколениях за поддержку того или иного эффекта (например, тумана) отвечало ядро видеокарты, а точнее его TCL-блок (Transforms Clipping Lighting - трансформация, отсечение и освещение), поэтому введение новых эффектов происходило очень медленно - было необходимо ждать очередного поколения видеокарт, в которых этот эффект поддерживается. Кроме того, нужно было постоянно расширять сам графический чип и переписывать драйвера, что добавляло хлопот производителям "железа". Вполне логичным шагом была замена фиксированного TCL-блока программируемым. Если точнее, то TCL заменяется вершинным шейдером, а кроме него существуют также пиксельные шейдеры, которые изменяют процесс обработки текстур. Это значит, что каждый разработчик может контролировать процесс обработки видеокартой данных о 3D-сцене и изменять его, чтобы добавить какой-нибудь новый визуальный эффект.

Именно версия поддерживаемых шейдеров становится основным отличием между видеокартами следующих поколений: шейдеры появляются в 3D-ускорителях четвертого поколения, например NVIDIA GeForce 4 или ATI Radeon 8500, и поддерживаются в DirectX 8. Следующее поколение - вершинный и пиксельный шейдеры версии 2.0 и DirectX 9. Примерами видеокарт этого поколения являются ATI Radeon 9600 или nVidia GeForce FX 5800. Бурное развитие языка программирования шейдеров высокого уровня для Direct3D и OpenGL 2.0 привело к практически полному отказу от использования ассемблера в описании шейдеров. Были разработаны специальные языки программирования шейдеров: Cg от NVIDIA, HLSL от Microsoft (впоследствии ставший частью DirectX 9) и GLSL от SGI, являющийся аналогом HLSL, разработанным для OpenGL.

В эпоху DirectX 9 для соответствия спецификациям требовалась лишь поддержка шейдеров 2.0. Сами разработчики за три года «додумали» столько, что DirectX 9.0c и 9.0a отличались такими «мелочами», как HDR, новые типы фильтрации, использование геометрических шаблонов и т.д. И все это кардинальным образом смогло изменить реализм трехмерной графики. Однако отсутствие жесткой стандартизации на уровне API привело к тому, что многие возможности пришлось снова поддерживать на уровне приложений, а это вылилось в проблемы с совместимостью, потери производительности и др.

Пожалуй, главное новшество, положенное в основу DirectX 10, – максимальное недопущение «вольностей» со стороны производителей GPU из-за специфики структуры нового набора шейдеров 4.0. В результате многочисленные надстройки, придуманные производителями, ушли в прошлое. Предусматривается использование полностью программируемых универсальных шейдеров при отсутствии разделения на вершинные и пиксельные, как это было раньше. Также добавлен новый тип шейдеров – геометрический, промежуточный между вершинным и пиксельным. Он дает возможность производить манипуляции над уже определившимся массивом треугольников после окончания работы вершинного шейдера и, самое главное, допускает произвольное изменение их геометрии.

:: AR-новости ::

ARToolworks Inc. и Esperient Ltd. выпустили СОВМЕСТНЫЙ ПРОЕКТ на базе своих технологий ARToolkit и Esperient Creator. Система Creator AR, по словам разработчиков, объединяет простоту использования среды Esperient Creator и мощь технологий дополненной реальности "под капотом" библиотеки ARToolkit.

Creator AR работает в качестве плагина к Esperient Creator, позволяющий использовать конструктор для разработки интерактивных приложений с применением дополненной реальности. Для создания простейших приложений даже не требуется программирование - Creator AR полностью интегрируется в визуальный интерфейс Esperient Creator.

Бен Воган (Ben Vaughan), администратор ARToolworks: «Я очень доволен, что мы смогли объединить наши передовые технологии, чтобы создать новый продукт, который будет полезен для наших клиентов. А простой в освоении инструментарий сделает технологию дополненной реальности доступнее и позволит разработчикам создавать весьма интересные приложения».

Для свободного скачивания доступна демонстрационная версия Creator AR, полная версия для коммерческого использования стоит $1500.

Корпорация Qualcomm объявила ПОБЕДИТЕЛЕЙ КОНКУРСА Augmented Reality Developer Challenge 2010. Первый приз ($125000) получила команда разработчиков из Литвы с игрой Paparazzi - симулятором папарацци, где, как ясно из названия, перед игроком стоит задача скрытой съемки звезд шоу-бизнеса. Второе место ($50,000) заняла фирма Defiant Development Pty Ltd. (Австралия), представившая Inch High Stunt Guy - игру, использующую дополненную реальность для управления трюковым мотоциклом. На третьем месте ($25,000) - разработчики из США с симулятором спасательного вертолета Danger Copter.

Что общего у легендарной Mike Tyson's Punch-Out на NES и сверхсовременного игрового контроллера Microsoft Kinect? Группа хакеров kinecthacks.net объединили две, казалось бы, совершенно несовместимые вещи: недавно программисты ПРЕДСТАВИЛИ ВИДЕОРОЛИК, который демонстрирует возможность использования Kinect для распознавания ударов во всеми любимом боксерском симуляторе!

Напомним, что Kinect - это «контроллер без контроллера» для Xbox 360, разработанный компанией Microsoft. Kinect позволяет пользователю взаимодействовать с приставкой через устные команды, позы тела и показываемые объекты или рисунки. Усилиями энтузиастов Kinect "подружили" с ПК, в результате чего стало появляться множество оригнальных проектов с дополненной реальностью.

OpenCV

OpenCV — одна из наиболее популярных библиотек для работы с дополненной реальностью. Это библиотека компьютерного зрения (computer vision) — набор алгоритмов для эффективной обработки и распознавания графической информации в реальном времени. OpenCV разработана корпорацией Intel и активно использует специфичные возможности интеловских процессоров (в частности, IPP — Integrated Performance Primitives) для оптимизации собственной работы. Библиотека кроссплатформенна, распространяется по лицензии BSD.

OpenCV находит широчайшее применение в самых разных сферах информационных технологий, включая трехмерные редакторы, системы распознавания лиц и жестов, HCI, компьютерные игры, робототехнику, стереоскопию и многое другое. Библиотека располагает множеством средств для работы с изображениями (включая базовую обработку, преобразование цветовых режимов и т.д.) и потоком видеоданных (ввод из файла или с камеры, вывод в файл). В арсенале OpenCV — мощные средства структурного анализа, калибровки, отслеживания движений и распознавания объектов. Кроме того, библиотека позволяет вывести изображение на экран и нарисовать простые геометрические фигуры для демонстрации результатов вычислений. В OpenCV даже предусмотрены базовые функции пользовательского интерфейса (полосы прокрутки, поддержка клавиатуры и мыши).

OpenCV состоит из следующих модулей:
cv — основные функции;
cvaux — вспомогательные и экспериментальные функции;
cxcore — структуры данных и линейная алгебра;
highgui — функции GUI.

OpenCV написана в основном на C, но имеет интерфейсы и для других языков (C++, C#, Python, Ruby, Java). Мы рассмотрим работу с OpenCV на примере замечательного языка D — биндинг библиотеки к D можно скачать с сайта журнала (fps-magazine.narod.ru) в разделе «Файловый архив».

Начнем с простого примера — вывода изображения с веб-камеры на экран. Объявляем основной модуль и импортируем функции OpenCV:

```
module main;

import std.stdio;
import std.string;

import opencv.cv;
import opencv.cvtypes;
import opencv.cxcore;
import opencv.highgui;
```

Объявляем основную функцию:

```
int main()
{
```

Объявляем служебные указатели и переменные:

```
CvCapture *capture = null;
IplImage  *frame = null;
int key = 0;
```

Инициализируем веб-камеру:

```
capture = cvCreateCameraCapture( 0 );
if ( !capture )
{
   writeln( "Cannot initialize webcam!\n" );
   return 1;
}
```

Создаем окно для вывода видео:

```
cvNamedWindow( "result", CV_WINDOW_AUTOSIZE );
```

Стартуем бесконечный цикл, который завершится, если пользователь нажмет клавишу **Q** (не лучший вариант, но для демонстрационных целей сойдет). В теле цикла принимаем с камеры очередной кадр изображения:

```
while( true )
{
   frame = cvQueryFrame( capture );
```

Если по каким-то причинам это не удается, вызываем аварийное завершение цикла (и, соответственно, всей программы):

```
   if (!frame) break;
```

Если кадр передан удачно, отображаем его в окне:

```
   cvShowImage("result", frame);
```

Проверяем нажатие клавиши и закрываем цикл:

```
   key = cvWaitKey( 10 );
   if (key == 'q') break;
}
```

Закрываем окно, выключаем камеру и завершаем работу приложения:

```
cvDestroyWindow( "result" );
cvReleaseCapture( &capture );
return 0;
}
```

В следующем номере журнала вы узнаете, как «подружить» OpenCV с OpenGL — такой тандем будет неплохой основой для разработки игр и других интерактивных приложений с дополненной реальностью.

```
module main;

import std.stdio;
import std.string;
import opencv.cv;
import opencv.cvtypes;
import opencv.cxcore;
import opencv.highgui;
int main()
{
   CvCapture *capture = null;
   IplImage  *frame = null;
   int key = 0;

   capture = cvCreateCameraCapture( 0 );
   if ( !capture )
   {
      writeln( "Cannot initialize webcam!\n" );
      return 1;
   }

   cvNamedWindow( "result", CV_WINDOW_AUTOSIZE );

   while( true )
   {
      frame = cvQueryFrame( capture );
      if( !frame ) break;

      cvShowImage( "result", frame );

      key = cvWaitKey( 10 );
      if (key == 'q') break;
   }

   cvDestroyWindow( "result" );
   cvReleaseCapture( &capture );
   return 0;
}
```

Язык D. Шаблоны

Одно из неоспоримых преимуществ языка D — мощнейшая система шаблонов. Шаблоны в D реализованы лучше, чем в C++. В этой статье я хочу показать несколько наиболее ярких примеров обобщенного программирования на D.

Простой шаблон объявляется следующим образом:

```
template Foo(T)
{
   void print(T t)
   {
      writeln(t);
   }
}
```

Пример использования:

```
Foo!int.print(10); // выводит 10
```

Параметром шаблона может быть не только тип, но и любое значение (через псевдоним):

```
template Foo(alias s)
{
   void print()
   {
      writeln("hello, ", s, "!");
   }
}
```

Пример использования:

```
Foo!"world".print(); // выводит строку «hello, world!»
```

Благодаря псевдонимам, шаблон может вычислять значение во время компиляции. Следующий шаблон возвращает квадрат числа:

```
template sqr(alias n)
{
   enum { sqr = n * n }
}
```

Пример использования:

```
int i = sqr!16; // переменной i присваивается значение 256
```

Другая категория шаблонов D — шаблоны функций:

```
void Foo(T)(T t)
{
   writeln(t);
}
```

Пример использования:

```
Foo!int(10); // выводит 10
```

Аргументом шаблона функции может быть так называемый кортеж (tuple) — особая абстракция, обладающая свойствами структуры и массива. Как и структура, кортеж может содержать элементы разных типов. Подобно массиву, кортеж индексирует элементы — к ним можно обращаться привычным оператором квадратных скобок.

В следующем примере шаблон функции принимает аргумент типа auto ref. В нее можно передать любое количество аргументов любого типа — они будут интерпретированы как кортеж. Сама функция выводит сначала количество элементов кортежа, затем первый элемент, а затем перебирает все циклом foreach.

```
void Foo(T...)(auto ref T x)
{
   writeln(x.length);
   writeln(x[0]);
   foreach(v; x) writeln(v);
}
```

Пример использования:

```
Foo( "hello!", 10, 0.567f );
```

Самое интересное — можно объявлять литералы кортежей:

```
template Tuple(E...)
{
   alias E Tuple;
}
alias Tuple!(3, 7, 'c') t;
Foo(t);
```

Кортеж однотипных элементов можно использовать для инициализации массива:

```
alias Tuple!(4, 2, 10) t;
int[] a = [t]; // все равно, что [4, 2, 10]
```

Такой кортеж называется кортежем выражений (expression tuple). Есть также кортеж типов (type tuple):

```
alias Tuple!(int, float) TF;
```

Его можно использовать как тип аргументов обычной функции:

```
void Foo(TF tf)
{
   writeln(tf);
}
```

Пример использования:

```
Foo(1, 0.5);  // выводит 10.5
```

Наконец, кортеж можно генерировать из элементов структуры при помощи свойства tupleof:

```
void setPosition(float x, float y, float z)
{
   writeln(x, " ", y, " ", z);
}

struct Point
{
   float x;
   float y;
   float z;
}

Point p = { 1.0, 0.0, 2.0 };
setPosition(p.tupleof);
```

Кортеж — это не настоящий тип данных, его следует рассматривать часть парадигмы обобщенного программирования. Проще говоря, кортеж как конструкция существует только во время компиляции программы. Для работы с динамическими данными в рантайме он, разумеется, не годится — на это есть другие средства. Например, для нахождения суммы чисел можно использовать такой шаблон:

```
T sum(T)(T[] array...)
{
   T result = 0;
   foreach(v; array) result+=v;
   return result;
}
```

Пример использования:

```
r = sum( 10, 20, 90, 30 );
```

Функция также принимает массив (как статический, так и динамический):

```
int[] a = [10, 20, 90, 30];
r = sum(a);
```

Другая важная концепция обобщенного программирования в D — примесь (mixin). Примеси отчасти заменяют в D препроцессор. Есть два вида примесей. Первый — это шаблон примеси, при помощи которого можно вставлять (подмешивать) заранее написанный код в текущий контекст:

```
mixin template Foo()
{
   int x = 5;
}
```

Теперь

```
struct Bar
{
   mixin Foo;
}
```

все равно, что

```
struct Bar
{
   int x = 5;
}
```

Шаблон примеси может быть параметризован:

```
mixin template Foo(T)
{
   T x = 5;
}

mixin Foo!(int);
```

Можно также подмешивать виртуальные функции в классы:

```
mixin template Foo()
{
   void func() { writefln("Foo.func()"); }
}

class Bar
{
   mixin Foo;
}
```

Другой тип примесей — примеси строк — используются для компиляции строк как обычного кода D. Следующий пример генерирует структуру по заданным параметрам:

```
template GenStruct(string Name, string M1)
{
   const char[] GenStruct =
      "struct " ~ Name ~
      "{ int " ~ M1 ~ "; }";
}

mixin(GenStruct!("Foo", "bar"));
```

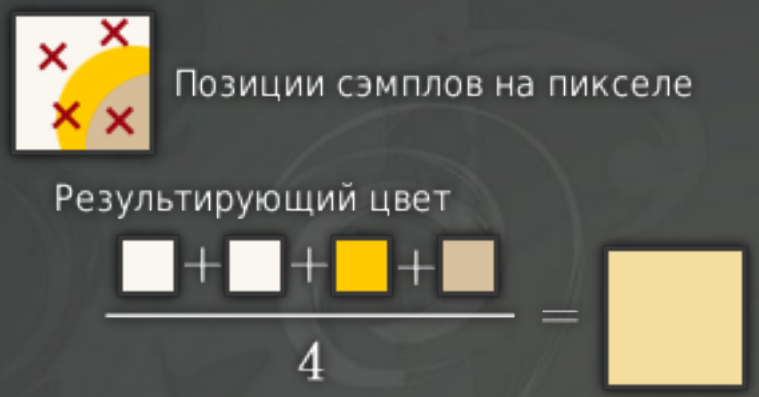
Генерирует:

```
struct Foo
{
   int bar;
}
```

Трассировщик лучей на D. Суперсэмплинг

Мы продолжаем разговор о трассировщиках лучей, начатый в двенадцатом номере журнала. Специфика цифрового изображения — деление на дискретные однородные точки — при рендеринге примитивов сказывается в виде так называемого алиасинга — эффекта ступенчатости линий и краев объектов. Способы решения этой проблемы имеют общее название антиалиасинг. Один из методов антиалиасинга, широко используемый при оффлайн-рендеринге — суперсэмплинг.

Сэмплом в компьютерной графике называют цветовую единицу изображения. В большинстве случаев это то же самое, что пиксель, видимый на экране — однако, например, при суперсэмплинге на экран выводятся не все вычисленные сэмплы. Суть этого метода проста: для каждого пикселя вычисляется не один, а несколько сэмплов, а в результат записывается их среднее арифметическое. Вы можете представить себе это так: происходит рендеринг картинки в более высоком разрешении, которая затем уменьшается до нужных размеров (так называемый даунсэмплинг) — при этом «лишние» пиксели используются в качестве дополнительных данных для усреднения цвета.



Суперсэмплинг — достаточно ресурсоемкая техника, требующая в несколько раз больше памяти и процессорного времени, чем при простом дискретном рендеринге. Поэтому сейчас все чаще используется адаптивный суперсэмплинг: дополнительные сэмплы вычисляются только для тех пикселей, которые находятся на границах резких переходов цвета. Для каждого пикселя выбирается два-три сэмпла — если цветовая разница между ними невысока, результат вычисляется только по ним; в противном случае вычисляются дополнительные сэмплы.

Основная проблема суперсэмплинга — вопрос количества и позиций дополнительных сэмплов. Ведь заранее неизвестно, в каком месте пикселя произойдет резкая смена цвета. Следовательно, нужен «умный» способ выбрать сэмплы. Существуют несколько подходов к решению этого вопроса:

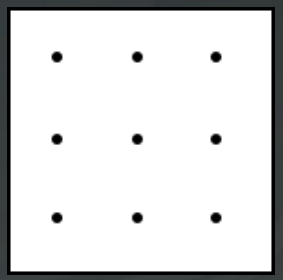
- **Сетка.** Простейший алгоритм. Пиксель однородно разбивается на несколько субпикселей, и сэмпл выбирается из центра каждого. Основной недостаток алгоритма сетки — для достижения полного антиалиасинга необходимо достаточно большое количество сэмплов (сетка 3x3 или выше).

- **Случайная выборка.** Также известна как стохастический сэмплинг. Выбор сэмплов происходит в случайных местах — что, в принципе, позволяет уменьшить количество необходимых сэмплов, но может вызывать нежелательные артефакты вследствие нерегулярности.

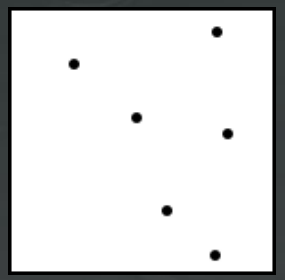
- **Диск Пуассона.** Сэмплы выбираются тоже случайно — но с проверкой расстояний между точками. В результате получается полное покрытие обширной дискообразной области при сравнительно небольшом количестве сэмплов. К сожалению, данный алгоритм малопригоден для использования в графике реального времени.

- **Джиттер.** Модификацированный алгоритм сетки, аппроксимирующий диск Пуассона. Пиксель разбивается на несколько субпикселей, но сэмпл выбирается не из центра, а из случайного места внутри каждого из них.

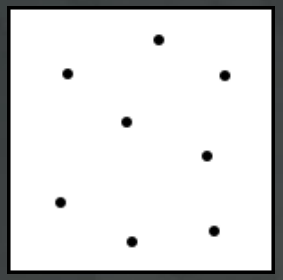
● **Повернутая сетка.** Используется сетка 2x2, повернутая на заданный угол. Сэмплы, таким образом, не привязаны к вертикальной и горизонтальной осям, что позволяет покрыть большее пространство при небольшом количестве сэмплов.



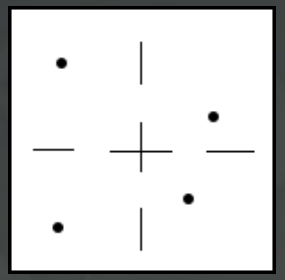
Сетка



Случайная выборка



Диск Пуассона



Джиттер

Повернутая сетка

Приведу простую реализацию суперсэмплинга с алгоритмом сетки (на языке D):

```
int samples = 4;
for (int x=0; x<buffer.width; x++)
for (int y=0; y<buffer.height; y++)
{
   color[] supersamples;
   for (float sx=0.0f; sx<1.0f; sx+=1.0f/samples)
   for (float sy=0.0f; sy<1.0f; sy+=1.0f/samples)
   {
      ray cameraRay = new ray(
         cameraPosition +
         vector3f(x-buffer.width/2+sx, y-buffer.height/2+sy,0),
         cameraPosition +
         vector3f(x-buffer.width/2+sx, y-buffer.height/2+sy,1000)
      );
      color ssColor = ... // здесь вычисляется цвет сэмпла
      supersamples ~= ssColor;
   }
   color resultColor = average(supersamples);
   buffer.setPixel(x,y,resultColor);
}
```

Функция average находит усредненный цвет на основании заданного массива цветов. Ее реализация зависит от способа кодирования цвета, но, в общем случае, вычисляется нахождением среднего арифметического для всех каналов (то есть, необходимо найти сумму каналов и разделить на общее количество элементов массива). Все вычисления производятся для чисел с плавающей запятой. В псевдокоде это можно представить следующим образом (для простоты предположим, что в массиве 4 элемента):

```
color result;
result.r = (col[0].r + col[1].r + col[2].r + col[3].r)/4;
result.g = (col[0].g + col[1].g + col[2].g + col[3].g)/4;
result.b = (col[0].b + col[1].b + col[2].b + col[3].b)/4;
result.a = (col[0].a + col[1].a + col[2].a + col[3].a)/4;
```

Neko и haXe. Эра скриптовых языков

Молодое поколение программистов, взращенное на Java и C#, в наши дни трудно удивить очередным скриптовым языком (а консерваторов «сишников» — и подавно). Но факт остается фактом: компилируемые языки постепенно переходят в категорию «для профессионалов» и «для хакеров», а для решения прикладных задач все чаще используют Python.

На волне этой тенденции возникают качественно новые идеи. К их числу можно отнести haXe (http://haxe.org) — метаязык сверхвысокого уровня абстракции. Программы на нем не выполняются напрямую, а транслируются в код для других языков. На момент написания статьи haXe поддерживает трансляцию в C++, Flash, JavaScript (HTML5), PHP и Neko (в разработке — поддержка Java). Впечатляющий потенциал такого подхода к программированию заключается в том, что можно выбрать наиболее подходящую платформу для эффективной эксплуатации программы без необходимости полного портирования кода.

Основной платформой haXe является необычайно быстрый и гибкий скриптовый язык Neko. В отличие от haXe, Neko является языком с динамической типизацией. Он спроектирован не столько для удобства написания кода программистом, сколько для автоматической генерации, в том числе – трансляции с других языков. Поэтому, как правило, компилятор Neko используется не сам по себе, а в качестве бэкенда и виртуальной машины для других языков. Поэтому haXe и Neko рассматриваются вместе, как единый набор интрументов.

Возможности Neko обеспечиваются тремя "китами":

- **стандартный язык Neko**
- **NXML**
- **NekoML**

Стандартный язык Neko чем-то напоминает Lua. Его можно использовать для разработки прототипов собственных модулей. Neko обеспечивает простое и быстрое тестирование новых модулей, поскольку не нужно применять объектно-ориентированные структуры и методы.

Два других языка – это XML-подобный язык разметки NXML и функциональный язык NekoML. NXML спроектирован для автоматической генерации компиляторами. Причина такого решения в том, что, в то время как людям проще читать стандартные языки, структуры XML гораздо проще для автоматического разбора и навигации. NXML также обеспечивает более простой способ включать отладочную информацию.

А вот NekoML – совсем другая история. Он следует стилю функциональных языков семейства ML и схож с языком Objective Caml. NekoML отлично подходит для создания компиляторов. Компилятор Neko, изначально написанный на Objective Caml, теперь тоже написан на NekoML и собирает сам себя. Этот «замкнутый круг» называется bootstrapping.

Но вернемся к haXe. «Hello, World!» на этом языке будет выглядеть следующим образом:

```
// MyProgram.hx:
class MyProgram
{
   static function main()
   {
      trace("Hello, World!");
   }
}
```

Проект компилируется одной командой:

```
haxe -main MyProgram -neko MyProgram.n
```

Полученный код можно выполнить на виртуальной машине Neko:

```
neko MyProgram.n
```

...или собрать исполняемый файл, скомпоновав байт-код Neko с виртуальной машиной:

```
nekotools boot MyProgram.n
```

haXe — объектно-ориентированный язык. В отличие от того же Lua, ООП в нем реализуется не в виде прототипов, а в виде классов. Это роднит haXe с C++. Классы haXe элегантно связаны с концепцией модулей этого языка — каждый модуль (файл исходного кода с расширением *.hx) объявляет одноименный класс, который можно импортировать из других модулей. Модули объединяются в пакеты, которым соответствуют каталоги с файлами *.hx.

```
// math/Vector.hx:
package math;

class Vector
{
   public function new() { }
}

// MyProgram.hx:
import math.Vector;
...
var vec: Vector = new Vector();
```

Любопытен подход haXe к типизации: можно объявить переменную, но не указывать ее тип — компилятор автоматически определит тип при первом присваивании значения. Нечто подобное есть в языке D (тип auto — но он работает только при объявлении с одновременным присваиванием).

```
var x; // значение не присвоено, тип переменной x — Unknown
x = 3; // присвоено значение типа Int, тип переменной x — Int
```

Как и в других современных языках, в haXe есть динамические массивы:

```
// Массив со статической инициализацией
var a: Array<Int> = [5,10,100];
trace(a[1]);

// Массив с динамической инициализацией
var a: Array<Int> = [];
a.push(5);
a.push(10);
a.push(100);
trace(a[1]);
```

Благодаря встроенным методам push и pop, динамический массив можно использовать в качестве стека:

```
// Программа выводит 5:
var a: Array<Int> = [];
a.push(5);
a.push(10);
a.pop();
trace(a[a.length-1]);
```

Итерация элементов массива осуществляется при помощи операторов for и in:

```
for( i in 0...a.length )
{
   trace(a[i]);
}

for( i in a )
{
   trace(i);
}
```

Кроме того, haXe поддерживает таблицы (анонимные объекты):

```
var abonent = { name: "Alex", number: 2980035 };
trace(abonent);
```

И ассоциативные массивы (хэши):

```
var dict: Hash<String> = new Hash<String>();
dict.set("object", "sphere");
dict.set("color", "blue");
for( i in dict.iterator() )
{
   trace(i);
}
```

Стандартная библиотека haXe реализует рефлексию — способность объектов получать метаданные о собственных свойствах и методах (а также добавлять новые свойства и методы):

```
var object: MyClass = new MyClass();

Reflect.setField(object,"foo",function(text: String)
{
   trace(text);
});

Reflect.callMethod(object,
   Reflect.field(object,"foo"),
   ["hello, world!"]);
```

Чтение и запись файлов — также не проблема:

```
import neko.io.File;
import neko.Lib;
...
var fileContents = File.getContent("file.txt");
Lib.println(fileContents);
```

Благодаря поддержке динамических библиотек C, на haXe/Neko теоретически можно писать полноценные программы с графическим интерфейсом, ненамного уступающие в плане производительности скомпилированным на C или C++. Для haXe уже существует ряд врапперов к популярным библиотекам, которые можно найти на lib.haxe.org. Конечно, haXe — в значительной степени экспериментальная система, и говорить о полном отказе от Python в пользу NekoVM для решения прикладных задач пока рановато. Однако haXe имеет все шансы стать основной веб-платформой будущего — этот язык можно использовать, например, для стандартизации веб-приложений и создания единого для всех интернет-сервисов сетевого API, дискуссии о котором идут уже не первый год. Очевидны также преимущества haXe при разработке онлайн-игр, в том числе — с использованием новейших возможностей современных браузеров (HTML5, WebGL и др.)

«Кошмар» программиста

Человеку свойственно ошибаться. Поэтому частенько даже в самом безошибочном, на первый взгляд, программном коде могут встретиться так называемые семантические ошибки. В отличие от синтаксических, они не отлавливаются компилятором — код может быть скомпилирован, но во время работы вылетать с неожиданным «segmentation fault». Еще хуже, если программа вызывает утечку памяти или даже сбой операционной системы. В этой статье рассмотрены наиболее распространенные ошибки, с которыми сталкиваются программисты.

Ошибка сегментации (англ. segmentation fault или сокращённо segfault) — ошибка программного обеспечения, возникающая при попытке обращения к недоступным для записи участкам памяти либо при попытке изменения памяти запрещённым способом.

Сегментная адресация памяти является одним из подходов к управлению и защите памяти в операционной системе. Для большинства целей она была вытеснена страничной памятью, однако в документациях по традиции используют термин «Ошибка сегментации». Некоторые операционные системы до сих пор используют сегментацию на некоторых логических уровнях, а страничная память используется в качестве основной политики управления памятью.

В UNIX-подобных операционных системах процесс, обращающийся к недействительным участкам памяти, получает сигнал SIGSEGV. В MS Windows такой процесс создаёт исключение STATUS_ACCESS_VIOLATION, и, как правило, запускает программу Dr. Watson, которая показывает пользователю окно с предложением отправить отчёт об ошибке в Microsoft.

Вот пример кода ANSI C, который приводит к ошибке сегментации на платформах с защитой памяти:

```
char *s = "hello world";
*s = 'H';
```

Когда программа, содержащая этот код, скомпилирована, строка «hello world» размещается в секции программы с бинарной пометкой «только для чтения». При запуске операционная система помещает её с другими строками и константами в сегмент памяти, предназначенный только для чтения. После запуска переменная s указывает на адрес строки, а попытка присвоить значение символьной константы H через переменную в памяти приводит к ошибке сегментации.

Компиляция и запуск таких программ на OpenBSD 4.0 вызывает следующую ошибку выполнения:

```
$ gcc segfault.c -g -o segfault
$ ./segfault
Segmentation fault
```

В отличие от этого, gcc 4.1.1 на Linux возвращает ошибку ещё во время компиляции:

```
$ gcc segfault.c -g -o segfault
segfault.c: In function ‘main’:
segfault.c:4: error: assignment of read-only location
```

Этот пример кода создаёт нулевой указатель и пытается присвоить значение по несуществующему адресу. Это вызывает ошибки сегментации во время выполнения программы на многих системах.

```
int* ptr = (int*)0;
*ptr = 1;
```

Ещё один способ вызвать ошибку сегментации заключается в том, чтобы вызвать функцию main рекурсивно, что приведёт к переполнению стека:

```
int main()
{
   main();
}
```

Утечка памяти (англ. memory leak) — процесс неконтролируемого уменьшения объёма свободной оперативной памяти, связанный с ошибками в работающих программах, вовремя не освобождающих ненужные уже участки памяти, или с ошибками системных служб контроля памяти.

Рассмотрим следующий фрагмент кода на C++:

```
/*1*/  char* pointer = 0;
/*2*/  for( int i = 0; i < 10; i++ ) {
/*3*/     pointer = new char[100];
/*4*/  }
/*5*/  delete [] pointer;
```

В этом примере на 3-й строке создается объект в динамической памяти. Код на 3-й строке выполняется 10 раз, причём каждый следующий раз адрес нового объекта перезаписывает значение, хранящееся в указателе pointer. На 5-й строке выполняется удаление объекта, созданного на последней итерации цикла. Однако первые 9 объектов остаются в динамической памяти, и одновременно в программе не остаётся переменных, которые бы хранили адреса этих объектов. Т.е. в 5-й строке невозможно ни получить доступ к первым 9 объектам, ни удалить их.

Динамическая память является ограниченным ресурсом. Управление динамической памятью программы обычно осуществляется библиотекой языка программирования, которая сама работает поверх динамической памяти, предоставляемой операционной системой. Утечки памяти приводят к тому, что потребление памяти программой неконтролируемо возрастает, в результате рано или поздно вступают в действие архитектурные ограничения среды исполнения (операционной системы, виртуальной машины, ЭВМ), и тогда новое выделение памяти становится невозможным. В этой ситуации в программе, которая запрашивает память, обычно происходит аварийная остановка. Это может по стечению обстоятельств произойти и совсем с другой программой после того, как программа, подверженная утечкам, потребит всю память ЭВМ.

Существуют различные способы предотвращения утечек памяти:

- **Отказ от динамической памяти.** Например, FORTRAN-77 полностью отказывается от применения механизмов динамического распределения памяти, что исключает подобные ошибки, но существенно ограничивает функциональность программ.

- **Владеющие указатели.** Владеющие указатели позволяют в той или иной мере согласовать время жизни указателя и время жизни объекта, на который он ссылается. Тем не менее, использование владеющих указателей не помогает в случае циклических ссылок между объектами.

- **Сборка мусора.** Некоторые языки программирования (например, Oberon, Java, D, языки платформы .NET) предоставляют средства, позволяющие автоматически освобождать неиспользуемую память (так называемые сборщики мусора, англ. garbage collectors). Сборщики мусора решают также и проблему циклических ссылок, но сборка мусора является ресурсоемкой операцией. За использование подобных средств приходится расплачиваться быстродействием системы.

  Сборка мусора была изобретена Джоном Маккарти в 1959 году при разработке языка программирования Lisp, структура которого делает ручное управление памятью крайне затруднительным.

- **Перезапуск программы.** В тех случаях, когда устранить утечки памяти не представляется возможным, например, при использовании кода, поставляемого в виде программных модулей и изготовленного сторонними разработчиками, применяют своеобразный способ игнорирования утечек. Код, подверженный утечкам, размещают в отдельной программе, а эту программу с нужной периодичностью перезапускают. Запуски и перезапуски программы выполняются внешней программой, которая также подаёт исходные данные и забирает результаты. Поскольку при завершении программы вся память, затребованная ей у операционной системы, возвращается операционной системе, такой метод не позволяет утечкам приобрести катастрофический характер.

Переполнение буфера (buffer overflow) — явление, возникающее, когда компьютерная программа записывает данные за пределами выделенного в памяти буфера. Переполнение буфера обычно возникает из-за неправильной работы с данными, полученными извне, и памятью, при отсутствии жесткой защиты со стороны подсистемы программирования (компилятор или интерпретатор) и операционной системы. В результате переполнения могут быть испорчены данные, расположенные следом за буфером или перед ним.

Переполнение буфера является наиболее популярным способом взлома компьютерных систем, так как большинство языков высокого уровня используют технологию стекового кадра — размещение данных в стеке процесса, смешивая данные программы с управляющими данными (в том числе адреса начала стекового кадра и адреса возврата из исполняемой функции).

Переполнение буфера может вызывать аварийное завершение или зависание программы, ведущее к отказу обслуживания (denial of service, DoS). Отдельные виды переполнений, например переполнение в стековом кадре, позволяют злоумышленнику загрузить и выполнить произвольный машинный код от имени программы и с правами учетной записи, от которой она выполняется.

Рассмотрим следующую программу на C. Ее можно использовать для генерации ошибок переполнения буфера. Первый аргумент командной строки программа принимает как текст, которым заполняется буфер.

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]) {
   char buffer[10];
   if (argc < 2) {
      fprintf(stderr, "ИСПОЛЬЗОВАНИЕ: %s строка\n", argv[0]);
      return 1;
   }
   strcpy(buffer, argv[1]);
   return 0;
}
```

Программу можно опробовать с несколькими разными строками. Строки размером в 9 или меньше символов не будут вызывать переполнение буфера. Строки в 10 и более символов будут вызывать переполнение, хотя это может и не приводить к ошибке сегментации.

По материалам Википедии

Модели освещения

Орен-Найар

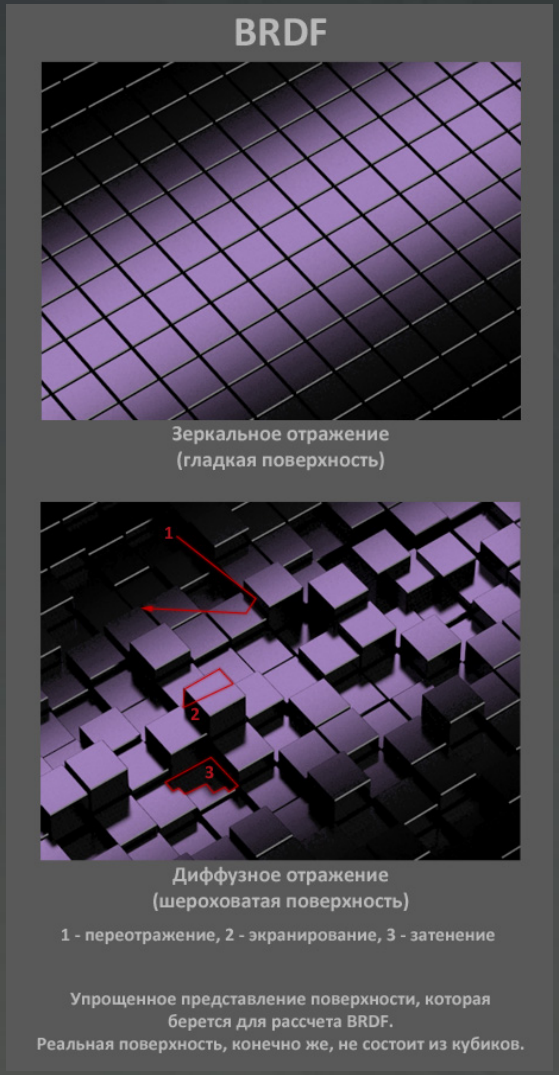
BRDF (Bidirectional reflectance distribution function — двунаправленная функция распределения отражений) описывает, как свет отражается или поглощается поверхностью в зависимости от разных углов падения.

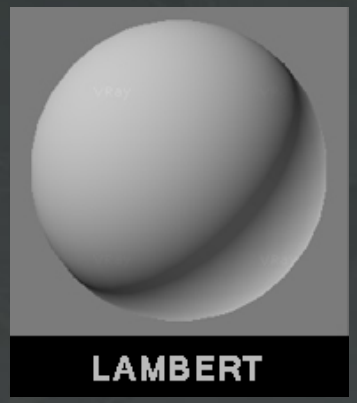
Существует три вида BRDF:
- ***Упрощенная** (без учета трассировки лучей)*
- ***Гибридная** (с трассировкой лучей)*
- ***Измеренная** (комплексная, основанная на реальных измерениях).*

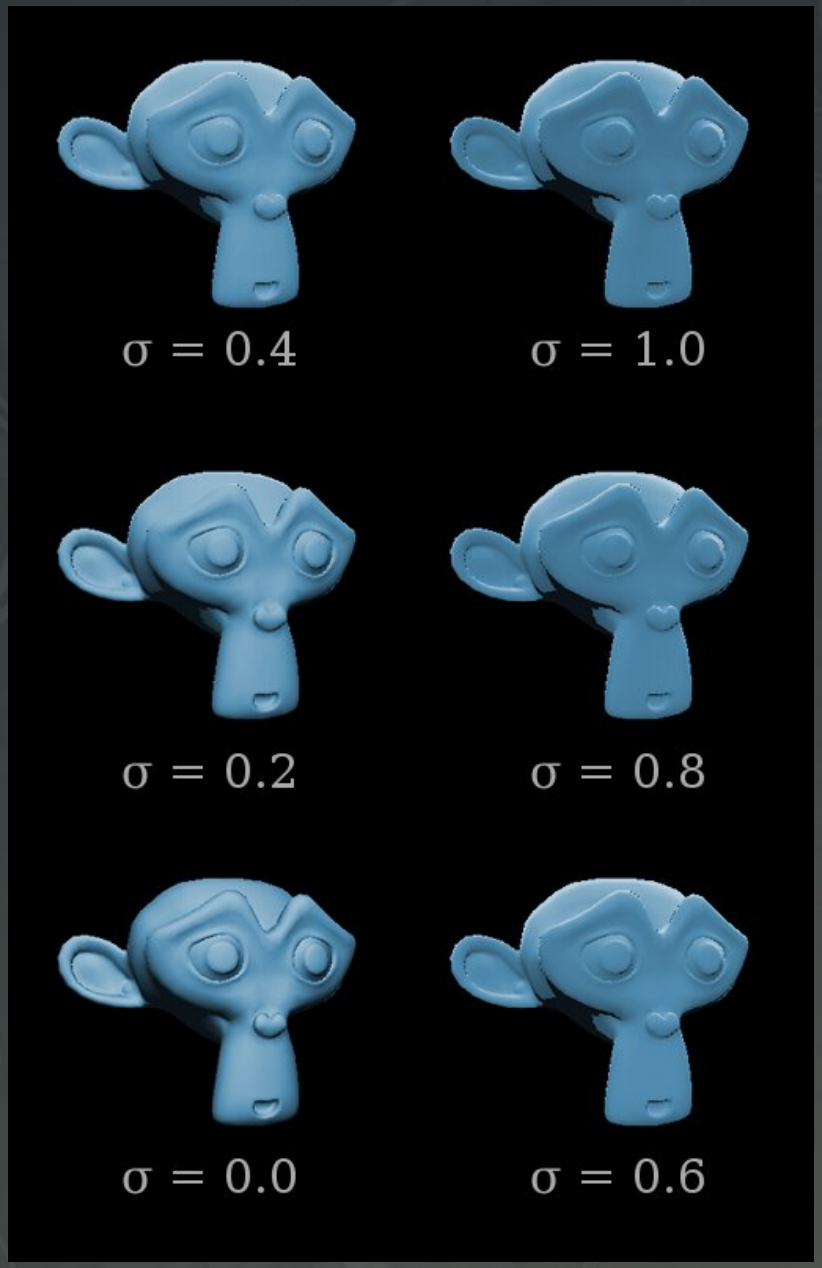
Измеренная и гибридная BRDF, как правило, обеспечивают более реалистичный результат, чем упрощенная.

Наиболее распространенные BRDF для моделирования диффузного освещения — Ламберт и Орен-Найар (Oren-Nayar). BRDF по Ламберту хорошо работает только для сравнительно гладких поверхностей. В отличие от нее, модель Орена-Найара (авторы — Майкл Орен и Шри К. Найар) основана на предположении, что поверхность состоит из множества микрограней, освещение каждой из которых описывается моделью Ламберта. Модель учитывает взаимное перекрытие (экранирование), затенение и переотражение между микрогранями.

Орен-Найар имеет параметр для контроля шероховатости поверхности (roughness). Этот параметр определяет, сколько света отразится назад в направлении источника света, что является характеристикой шероховатой, бархатистой или запыленной поверхности. Чем чем выше шероховатость, тем менее отчетливым становится диффузное отражение.







Формула Орена-Найара имеет вид:

$$I=I_L \cdot k_d \cdot cos(\theta_L) \cdot (A + B \cdot max(0, cos(\theta_v - \theta_L)) \cdot sin(\alpha) \cdot tan(\beta))$$

где I — интенсивность отраженного света, I_L — интенсивность точечного источника света, k_d – коэффициент диффузного отражения, θ_L - угол между направлением света и нормалью к поверхности, θ_V – угол между нормалью и направлением на наблюдателя. Параметры модели определяются по следующим формулам:

$$A=1-0.5 \cdot \frac{\sigma^2}{\sigma^2 + 0.33} \qquad B=0.45 \cdot \frac{\sigma^2}{\sigma^2 + 0.09}$$

$$\alpha = min(\theta_L, \theta_V) \qquad \beta = max(\theta_L, \theta_V)$$

$$cos\theta_L = (N, L) \qquad cos\theta_V = (N, V)$$

где N — нормаль, L — вектор направления на источник света, V — вектор наблюдателя. Параметр σ (задается в диапазоне [0, 1]) отвечает за шероховатость поверхности: чем он больше, тем более шероховатой является поверхность. Если σ = 0 (все микрограни расположены в одной плоскости), то A = 1, B = 0, и, следовательно, формула Орена-Найара упрощается до модели Ламберта:

$$I=I_L \cdot k_d \cdot cos(\theta_L)$$

Реализация на GLSL*

Вершинная программа:

```
varying vec4 V_eye;
varying vec4 L_eye;
varying vec4 N_eye;
void main(void)
{
   gl_Position = ftransform();

   V_eye = gl_ModelViewMatrix * gl_Vertex;
   L_eye = gl_LightSource[0].position - V_eye;
   N_eye = vec4(gl_NormalMatrix * gl_Normal, 1.0);

   V_eye = -V_eye;
}
```

Фрагментная программа:

```
varying vec4 V_eye;
varying vec4 L_eye;
varying vec4 N_eye;

float roughness = 0.85;

void main()
{
   vec4 Ca = gl_FrontMaterial.ambient;
   vec4 Cd = gl_FrontMaterial.diffuse;

   float rs = roughness * roughness;
   float A = 1 - 0.5 * rs / (rs + 0.33);
   float B = 0.45 * rs / (rs + 0.09);
   vec3 V = normalize(vec3(V_eye));
   vec3 L = normalize(vec3(L_eye));
   vec3 N = normalize(vec3(N_eye));

   float NL = dot ( N, L );
   float NV = dot ( N, V );

   vec3 LProj = normalize ( L - N * NL );
   vec3 VProj = normalize ( V - N * NV );
   float cx = max ( dot ( LProj, VProj ), 0.0 );

   float cosAlpha = NL > NV ? NL : NV;
   float cosBeta  = NL > NV ? NV : NL;
   float dx = sqrt ( ( 1.0 - cosAlpha * cosAlpha )
                   * ( 1.0 - cosBeta * cosBeta ) ) / cosBeta;

   gl_FragColor = max ( 0.0, NL ) * Cd * (A + B * cx * dx);
   gl_FragColor.a = 1.0;
}
```

** - предполагается, что в приложении уже включены и настроены соответствующие параметры OpenGL (позиция источника света, свойства материала и др.) Приведенная реализация в целях упрощения не учитывает интенсивность источника света. Исходный код предоставлен как общественное достояние (Public Domain) и может быть использован безо всяких ограничений.*

Как собрать пакет Debian?

Debian — один из старейших фундаментальных дистрибутивов Linux, который часто берется за основу для создания других операционных систем. Если вы используете Ubuntu или Mint, то ваша система тоже является частью большого семейства Debian. Несмотря на косметические различия, все дистрибутивы этого семейства имеют одно сходство — они совместимы с Debian. В первую очередь это касается системы управления пакетами. В Ubuntu можно устанавливать пакеты из репозитория Debian, и наоборот. Поэтому само собой разумеется, что подход к созданию пакетов Debian будет справедлив для всех основанных на нем дистрибутивов.

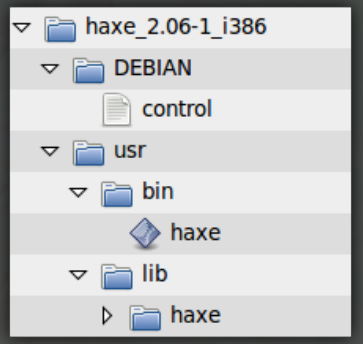
Однако что делать, если вы не нашли нужных deb-пакетов ни в одном репозитории? Большинство пользователей привыкло в такой ситуации собирать программы из исходников, осуществляя установку штатными средствами сопутствующего make-файла (sudo ./configure && make && make install). Однако линуксоид со стажем прекрасно понимает — таким образом происходит «засорение» системы бесхозными файлами. Это, собственно, и есть проблема Windows №2 (первая проблема, как известно, вирусы), которую в Linux решают менеджеры пакетов. В отличие от них, вы со временем можете забыть, что уже установили, а что — нет. А винчестер все-таки не резиновый. Поэтому я призываю не торопиться с командой make install и попробовать оформить свежесобранную программу в аккуратный пакет.

В этой статье я привожу только основные моменты: создание пакета для помещения в официальные репозитории — это отдельная тема, там к мэйнтейнеру выдвигаются очень строгие требования, необходимо соблюдать множество правил. А вот для сборки личного пакета для «домашнего» использования большинство из этих правил знать необязательно.

Простейший пакет собирается в четыре этапа:

1. Создайте новый каталог и назовите его mypackage_1.0-1_i386, где «myprogram» — название пакета, «1.0» — версия, «1» — номер ревизии (то есть, вашей сборки пакета) и «i386» — архитектура CPU, под которую скомпилированы бинарные файлы в пакете. Если пакет содержит файлы для разработчиков (заголовочные файлы, статические библиотеки, документацию), к названию пакета прибавляется «-dev». Конечно, придерживаться этих соглашений вас никто не заставляет, но так будет удобнее и для менеджера пакетов, и для вас самих. Например, недавно я собрал собственные пакеты для языка haXe: haxe_2.06-1_i386, libneko_1.8.1-1_i386, libneko-dev_1.8.1-1_i386, neko_1.8.1-1_i386.

2. Поместите в этот каталог файлы программы, повторяя структуру файловой системы Linux. То есть, если файл libsomething.so должен установиться в каталог /usr/lib, то необходимо создать в нашем рабочем каталоге папку usr, а в ней — lib, и скопировать в нее libsomething.so. На этом этапе необходимо быть очень осторожным, чтобы файлы не конфликтовали с уже существующими в системе (поскольку установка пакета запускается с правами root, она, и глазом не моргнув, заменит существующие файлы новыми — а это, ясное дело, чревато фатальными последствиями).



3. Создайте в каталоге еще одну папку и назовите ее DEBIAN. В ней создайте файл control следующего содержания (привожу пример из своего пакета):

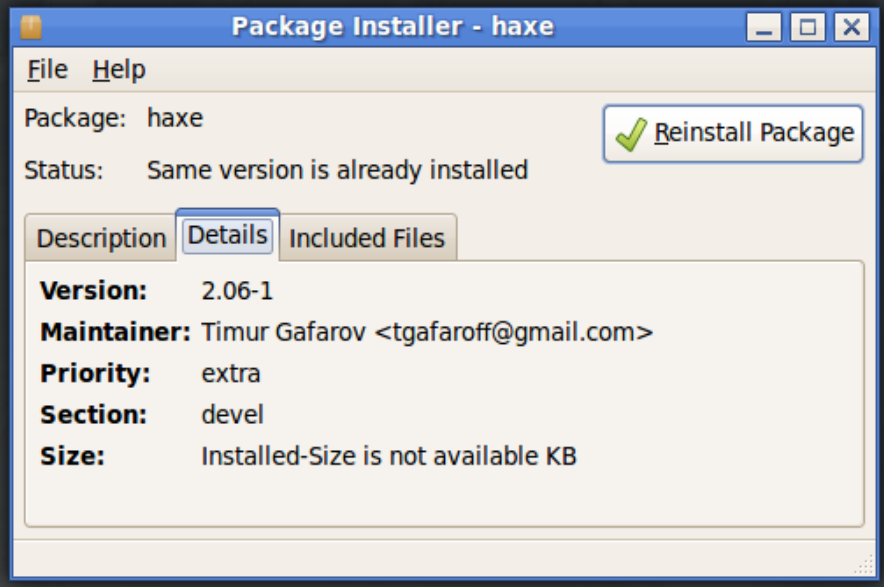
Package: haxe
Version: 2.06-1
Priority: extra
Section: devel
Maintainer: Timur Gafarov <tgafaroff@gmail.com>
Architecture: i386
Depends: neko
Provides: haxe
Description: haXe language.
Website: http://haxe.org

Как видите, ничего сложного. Эта информация используется системой для индексирования и каталогизации пакетов. Она же выводится на экран при установке пакета графическим инсталлятором (например, GDebi).

4. Теперь перейдите в каталог двумя уровнями выше (то есть, в которой находится папка mypackage_1.0-1_i386) и введите следующую команду:

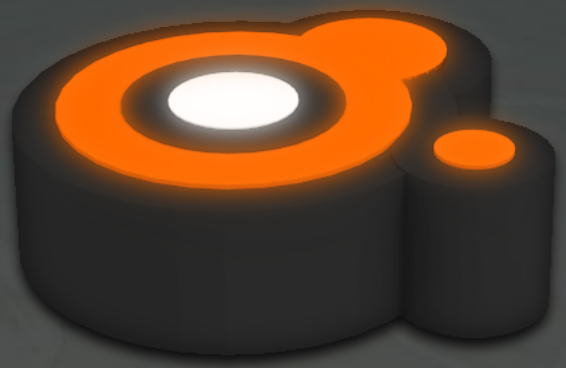
```
dpkg-deb --build mypackage_1.0-1_i386
```

Будет собран пакет mypackage_1.0-1_i386.deb.



Это все!

Надеемся, номер вышел интересным. Если так, поддержите FPS! Отправляйте статьи, обзоры, интервью и прочее на любые темы, касающиеся игр, графики, звука, программирования и т.д. на clocktower89@mail.ru или gecko0307@mail.ru.



Сборка от 1.3.2011.
python 2.5.2 (r252:60911, Oct 5 2008, 19:24:49) [GCC 4.3.2];
reportlab 2.5; svglib 0.6.3; svgmath 0.3.3; pygments 1.3.1.