Независимый электронный журнал о разработке компьютерных игр

9

2010

FPS

> Графика > Обзоры

> Кодинг > Уроки

Главный редактор: Гафаров Т. А .
Дизайн и верстка: Гафаров Т. А .

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

Деформация объектов? Легко!

Как по-вашему удобнее всего моделировать плавные сгибающиеся объекты? Генерацией по кривым? А вот и нет! Представим себе такую ситуацию: вы смоделировали какой-нибудь гибкий объект, например, трубку, змею или стебель растения. И теперь необходимо деформировать его: согнуть, скрутить, раздуть и т.п. От кривых тут толку маловато. Скелет создавать тоже не очень-то хочется — да и смысла в этом нет, если вы не делаете анимацию. Однако есть весьма простой выход: использование модификатора ***MeshDeform****.*

Сей инструмент в целом напоминает «модификатор» из программы Anim8tor (который, кстати, был моим первым трехмерным редактором). Суть его такова: мы создаем дополнительный меш в виде многоуровневой коробки, окружающей модель. Объект будет «привязан» к этой коробке таким образом, что, изменяя положение вершин коробки, мы соответственно трансформируем и нашу модель.

Рассмотрим действие MeshDeform на примере простого цилиндра (рис. 1). Сначала мы должны объединить все вершины модели в группу. В режиме редактирования на вкладке Link and Materials рядом со списком Vertex Groups нажмите New. Можете ввести группе новое имя, например, DeformGroup. Теперь нажмите A, а затем — кнопку Assign.

Теперь выйдите из режима редактирования, разместите 3D-курсор по цетру цилиндра и добавьте куб (<пробел> → Add → Mesh → Cube). Экструдируйте его, пока он не станет многоуровневым, как на рис. 2. Получившаяся коробка обязательно должна полностью заключать в себе цилиндр.

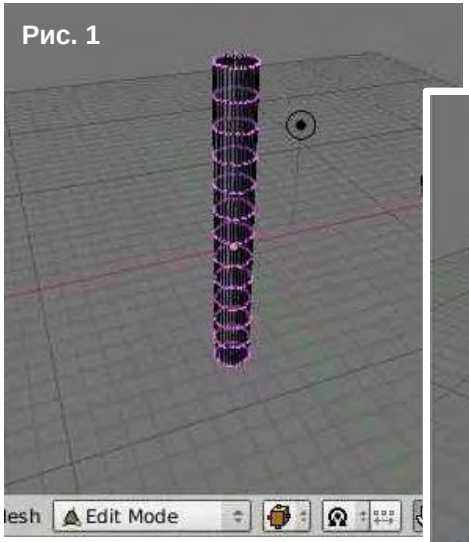


Рис. 1

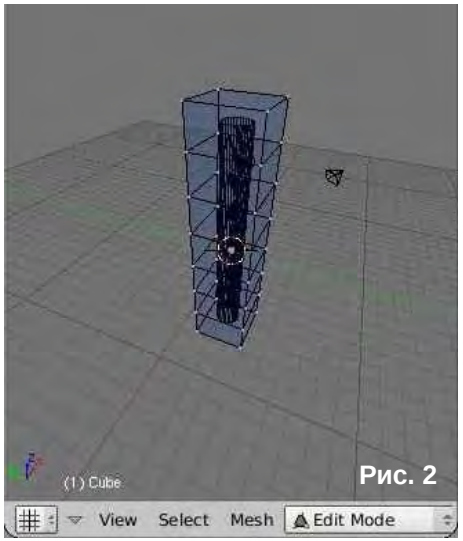


Рис. 2

При этом удобно переключиться на каркасный режим отображения. Также введите коробке какое-нибудь имя, например, DeformCube.

Теперь вернемся к цилиндру. Выделите его и, находясь в объектном режиме, на вкладке Modifiers нажмите Add Modifier и выберите MeshDeform. В графе Ob введите имя коробки (DeformCube), а в графе VGroup — имя группы вершин (DeformGroup). Нажмите кнопку Bind. Процесс привязки может занять несколько секунд (в зависимости от сложности модели и деформационного меша).

Теперь дело за вами. Выделите коробку, перейдите в режим редактирования и перемещайте вершины коробки, наблюдая за тем, как «реагирует» на это цилиндр. Можно просто согнуть его (рис. 3), а можно сделать воронку, увеличив масштаб необходимых частей куба (рис. 4). Если вы переместите коробку целиком, цилиндр тоже переместится. То же самое — в отношении поворота и масштабирования.

Теперь встает вопрос: как отрендерить модель, не удаляя деформационную коробку? Надо сделать коробку «невидимой». Чтобы она не заслоняла нам цилиндр, можно назначить ей каркасный режим индивидуально: Объектный режим → Вкладка Draw → Drawtype → Wire. А чтобы коробка не отображалась при рендеринге, создаем ей новый материал и делаем следующее:

1. На вкладке Material уменьшаем значение A до нуля;

2. На вкладке Links and Pipeline включаем Ztransp и отключаем Traceabl и Shadbuf;

3. На вкладке Shaders уменьшаем значение Spec до нуля и отключаем Shadows.

Вот такой нехитрый метод. С его помощью вы без труда сможете деформировать любой объект.

Рис. 3

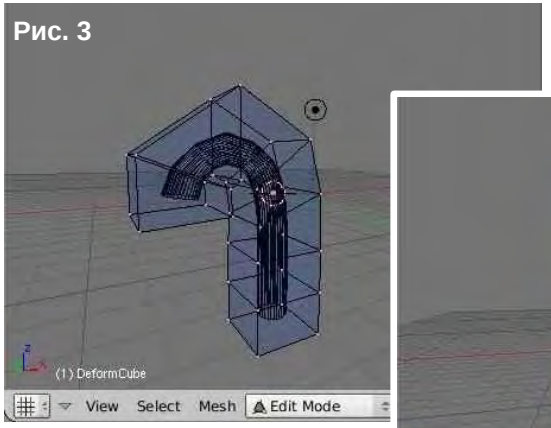


Рис. 4

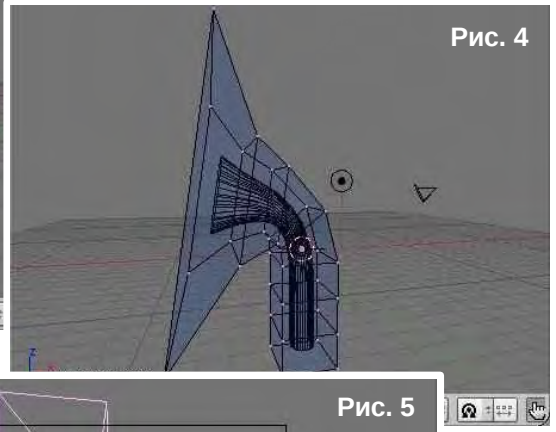


Рис. 5

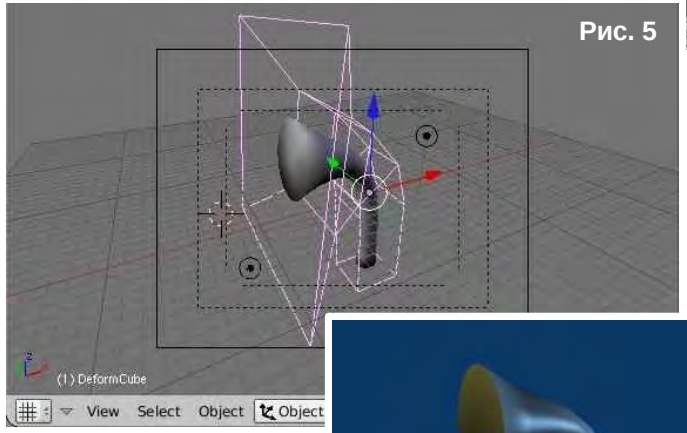
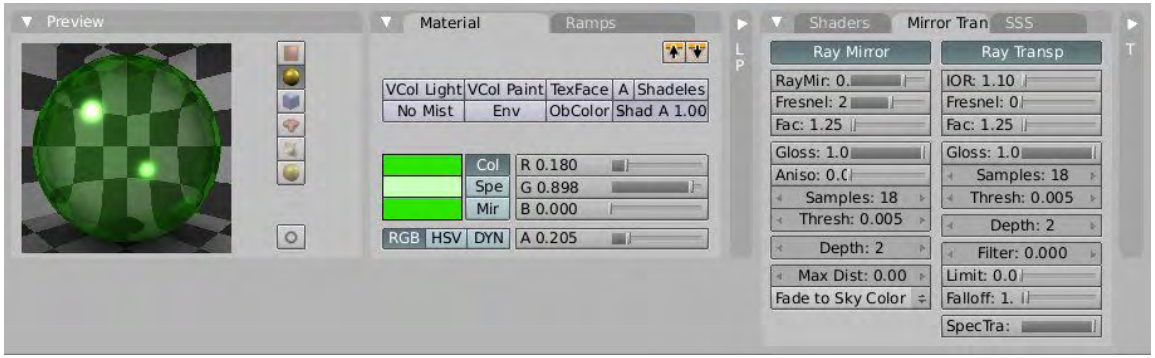


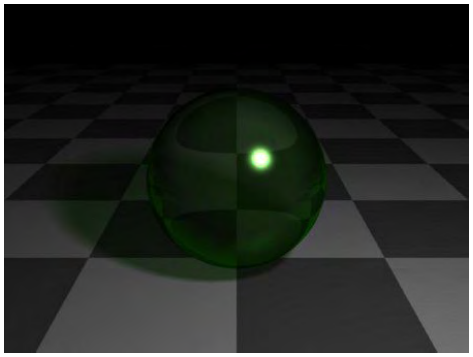
Рис. 6

Gecko
clocktower89@mail.ru

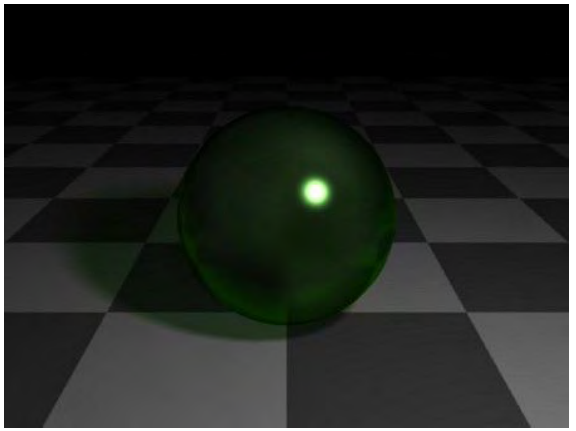
Рендеринг стекла

Рендеринг стекла в Blender упрощает поддержка прозрачности и отражения при помощи трассировки лучей. Ниже приведен простой пример зеленого стекла. Сегодня мы рассмотрим несколько простых, но эффектных приемов, дополняющих эту технологию.

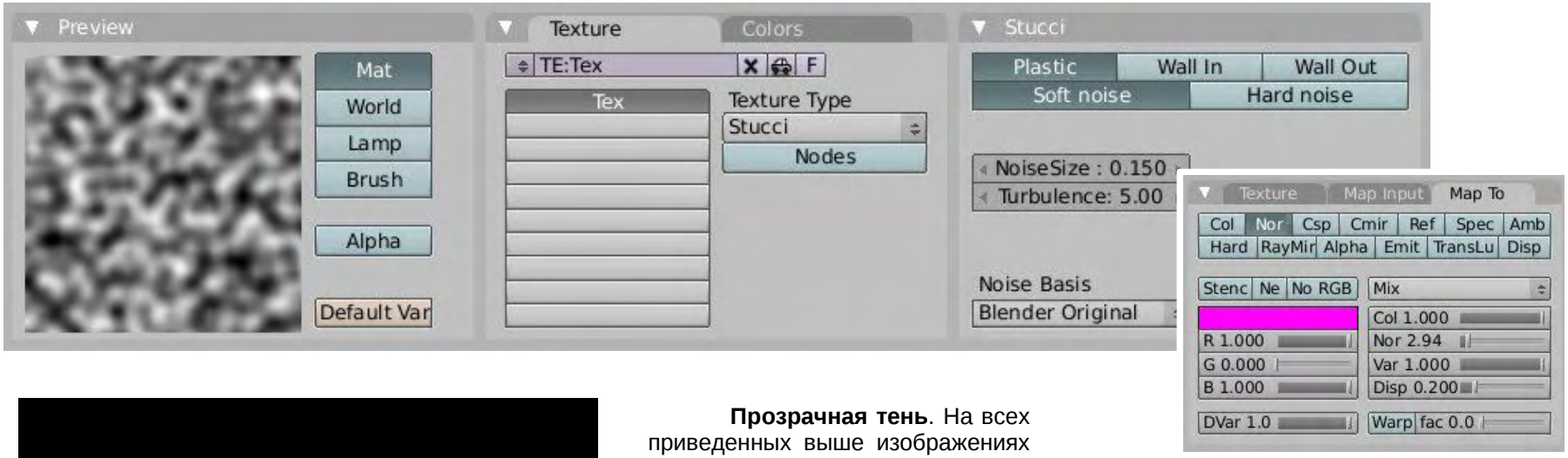


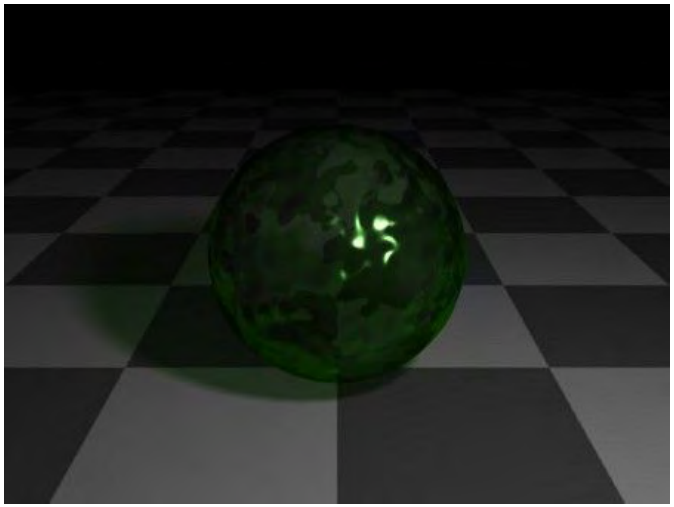


Мутное стекло. Данный метод напомниает подповерхностное рассеивание (Sub-Surface Scattering, SSS) и во многих ситуациях позволяет достичь похожего результата при гораздо меньших ресурсозатратах. Лучи света, проходя сквозь поверхность объекта, рассеиваются. В нашем случае, вместо полной прозрачности, мы получаем эффект размытия, который можно наблюдать в мутном стекле, промасленной бумаге и других подобных материалах. Для управления степенью размытия прозрачности используется параметр Gloss (значение 0 соответствует максимальному размытию, 1 — отсутствию размытия). Аналогичный параметр есть и для отражения.



Рельефное стекло. Вы же не собираетесь моделировать этот рельеф вручную из полигонов, когда есть normal mapping? Достаточно только добавить нужную текстуру, а затем в настройках Map To включить кнопку Nor и увеличить параметр Nor до нужного значения. Ниже приведен пример рельефного стекла с текстурой stucci.





Прозрачная тень. На всех приведенных выше изображениях видна прозрачная зеленоватая тень от стеклянной сферы. А по умолчанию тень не зависит от цвета объекта. Чтобы другие предметы могли принимать тени от прозрачных объектов, включите кнопку TraShad на вкладке Shaders.

Gecko
clocktower89@mail.ru

Blender 2.5
Руководство по выживанию

Если не принимать в расчет коммерческий кинематограф, пакет трехмерного моделирования Ble der был и остается самым очевидным решением в широчайшей области задач. Его можно смело рекомендовать образовательным учреждениям, разработчикам игр, независимым художественным студиям, малым предприятиям, да и просто любому, кто желает без особых затрат познакомиться с компьютерной графикой. Наконец, для пользователей альтернативных операционных систем это вообще чуть ли не единственный доступный инструмент в сфере 3D-графики...

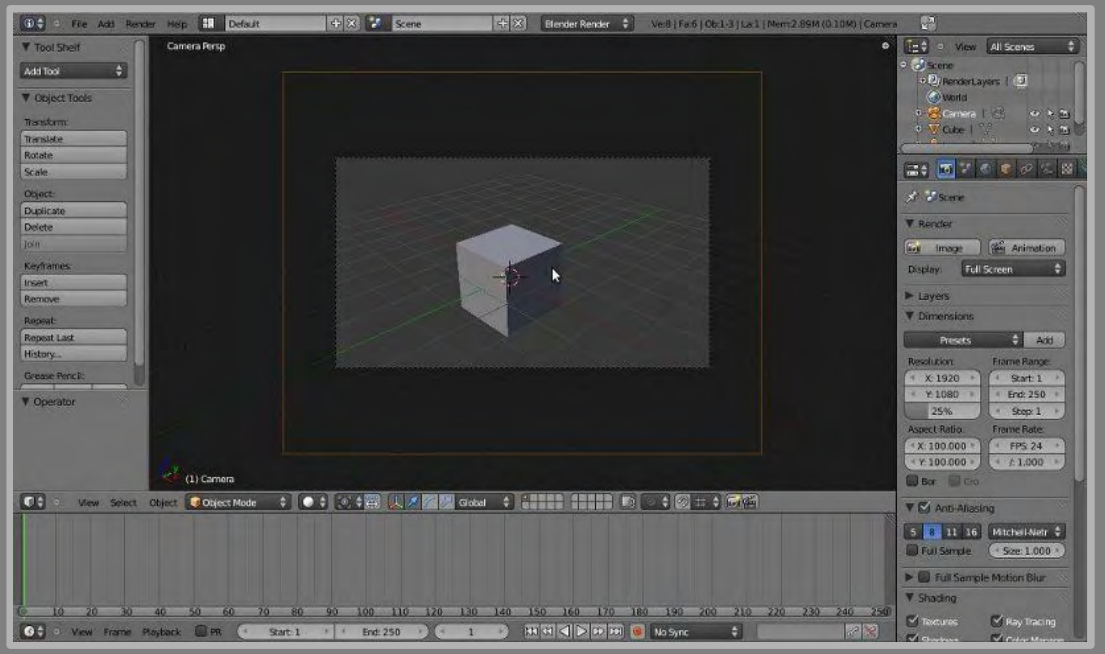
За годы существования под свободной лицензией, пакет трехмерного моделирования Blender приобрел стойкую репутацию крайне сложной в освоении программы. Интерфейс Blender — явление оригинальное и самобытное, и мнения о нем весьма расхожи. Одни считают его простым и логичным, другие — постигают с огромным трудом, третьи и вовсе боятся. Особенно же его «не любят» профессионалы в области графики и анимации, прикипевшие к своим привычным инструментам. В связи с этим нет и однозначной оценки, подходит ли Blender для применения в крупных коммерческих проектах — попросту некому оценивать. Это, конечно, не значит, что Blender используется исключительно в любительской среде. Однако такой миф существует, и, в определенном смысле, не дает Blender стать стандартом в психологическом смысле — в то время как технически его уже давно ставят в один ряд с Maya, 3DS Max и пр.

Но все же остается вопрос интерфейса. Если, к примеру, Gimp сейчас довольно популярен, то только потому что на него сравнительно легко перейти с Photoshop или другой аналогичной программы. Blender же, напротив, требует довольно длительного процесса обучения и привыкания. Однако это себя оправдывает: любой опытный пользователь Blender скажет вам, что нет интерфейса удобнее. Все бы хорошо, но разработчики время от времени подливают масла в огонь, выпуская очередную версию с очередными изменениями. Изменения, казалось бы, незначительны — но порой приводят к затруднительным ситуациям, ставя в тупик пользователя, привыкшего к определенной конфигурации и расположению тех или иных кнопок в предыдущей версии программы. К выходу версии 2.49 нагромождение этих «незначительных» изменений достигло апогея — и нам преподносят очередную революцию...

Ветку 2.5 я начал отслеживать уже с выходом нулевой альфы. Запустив ее в первый раз и по привычке нажав пробел, я сразу понял, что придется переучиваться. На момент написания этих строк у меня под рукой вторая альфа — и, поверьте, возвращаться к 2.49 мне уже не хочется. Хотите ощутить абсолютный контроль на кончиках пальцев? Забудьте все, что вы знали о Blender. Станьте частью революции.

Первое, что бросается в глаза — новая, чрезвычайно красивая и стильная тема оформления. Значки (между прочим, в стиле tango) и элементы интерфейса просто радуют глаз. Лично я в этом отношении неприхотлив, но кто же не любит работать в эстетически привлекательном окружении?

При запуске Blender выводит меню, в котором можно открыть один из предыдущих проектов, восстановить последнюю сессию или почитать руководства или отправиться на сайт программы.



На скриншоте показан интерфейс Blender в «заводской» конфигурации. У вас он может несколько отличаться, в зависимости от версии и разрешения экрана. Редактор свойств теперь имеет вертикальную ориентацию и располагается по умолчанию в правой части окна. Если кнопки переключения секций не помещаются на экране (как в моем случае), их можно прокручивать колесиком мыши. Кстати, к моей великой радости, столь любимые всеми вкладки, на которых располагаются кнопки свойств, никуда не исчезли, а просто видоизменились.

На мой взгляд, наиболее значительные изменения коснулись области просмотра сцены (3D-окна). Появилось две новинки. Во-первых, так называемая полка с инструментами (Tool Shelf) в левой части окна. Все операции над вершинами и полигонами теперь сгруппированы именно здесь. Сюда же можно добавлять и новые нужные вам инструменты. Tool Shelf можно прятать и показывать клавишей **T**. Во-вторых, чрезвычайно удобная панель свойств в правой части (клавиша **N**), в которой можно указать точные значения различных атрибутов выбранного объекта: координаты, угол поворота (можно задавать в углах Эйлера или кватернионах), масштаб и т.д. Содержимое панели свойств может меняться в зависимости от текущего режима. Полку инструментов и панель свойств также можно вызывать при помощи маленьких плюсиков в левом и правом верхних углах соответственно.

На панели инструментов трехмерного окна, наряду с меню режимов, инструментами трансформации и палитрой слоев, наконец-то появились кнопки переключения режима выделения (вершина, ребро, полигон). Содержимое всех меню в целом осталось неизменным.

Однако встает вопрос: как создать новое окно? Никакого контекстного меню при нажатии правой кнопкой на границе двух окон не появляется. Меня поначалу это поставило в тупик. На самом деле все просто: в правом верхнем углу каждого окна есть полосатый треугольничек. Достаточно оттянуть его вовнутрь, и появится новое окно. Соответственно, если оттянуть треугольник наружу, можно уничтожить и «поглотить» одно из соседних окон (эта операция возможна только в том случае, если оба соседних окна имеют одинаковую ширину или высоту — в зависимости от того, на какой границе окна происходит «поглощение»).

Кстати, вот еще одна полезная опция, которая в большинстве случаев избавляет от необходимости создавать несколько 3D-окон: Toggle Quad View в меню View (или **Ctrl+Alt+Q**). Она разбивает 3D-окно на четыре части для каждой проекции (перспективная и три ортогональных: Top, Front и Right). Правда, такой способ имеет определенные недостатки. Например, нельзя индивидуально настроить режим отображения для каждого вида, равно как и изменить их пропорции. Возможно, в будущем разработчики уделят этому внимание.

Иногда бывает так, что Blender с самого начала работает слишком медленно. В этом случае попробуйте зайти в настройки (User Prefences) и в секции System переключить опцию Window Draw Method с Full на Overlap или Overlap Flip.

По правде говоря, конфигурация GUI и расположение всевозможных кнопок не имеет особого значения, если вы знаете клавиатурные комбинации Blender. В большинстве своем они остались такими же, как и в предыдущих версиях, но некоторые клавиши все же изменили свою функцию. Для начала я перечислю только самые, на мой взгляд, распространенные. Более полный список можно легко найти в Интернете.

Пробел. Теперь он вызывает не меню добавления примитива, а строку поиска, в которой можно ввести название инструмента или команды, которую вы не можете найти среди кнопок. Blender попробует угадать, что именно вы ищете и предложит варианты.

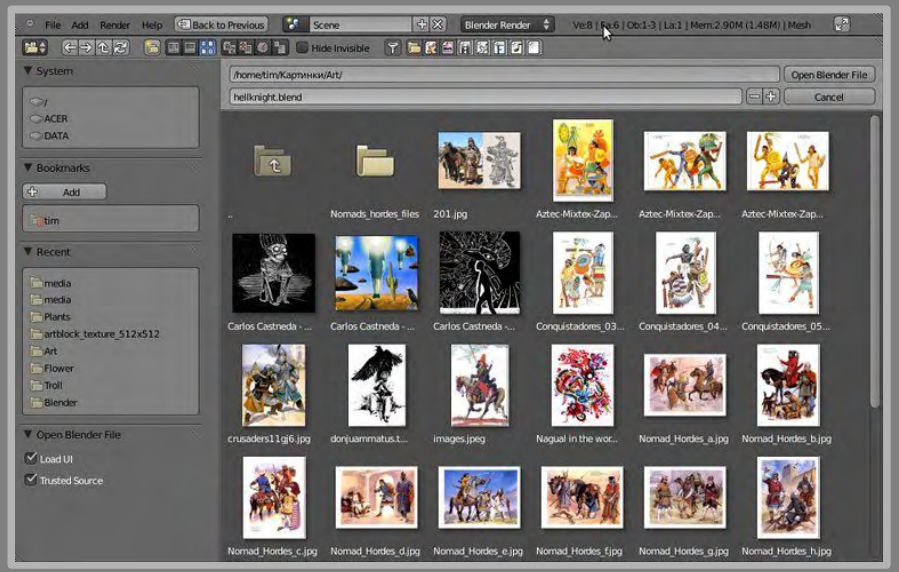
B. В режиме редактирования она, как и прежде, включает режим выделения прямоугольником, но вторичное нажатие уже не приводит к переключению в режим выделения окружностью. Окружность включается клавишей **C**.

G. В режиме редактирования позволяет свободно переместить выбранные вершины (или другие элементы). Вращение производится клавишей **R**, масштабирование — клавишей **S**. К сожалению, в Blender 2.5 нет (или пока нет) привычного интерпретатора жестов мышью, который позволяет перемещать вершины, нарисовав прямую линию, и вращать их, нарисовав эллипс.

Ходят слухи, что в будущем в Blender появится возможность выбирать профили горячих клавиш — как в 2.5 или 2.4x. Поживем – увидим.

Отдельного упоминания заслуживает новый удобнейший файловый браузер. Еще бы — ведь наконец-то появились полноценные миниатюры графических файлов (ранее доступные только в специальном браузере изображений). Есть фильтр, позволяющий выводить только файлы определенного типа, закладки, список часто посещаемых директорий, список системных дисков, различные режимы просмотра (список, подробный список, миниатюры). Можно отключить показ скрытых файлов, отсортировать файлы по имени, типу, размеру и дате изменения. При загрузке blend-файла можно не загружать конфигурацию интерфейса и отключить автозапуск скриптов. При сохранении изображения можно выбрать формат — раньше это делалось исключительно в предварительных настройках экспорта.

Продолжение следует...



clocktower89@mail.ru

Gecko

Начинаем работать с GMOgre

Это тема довольно обширная, поэтому пока рассмотрим базовые классы GMOgre и научимся с ними работать. Для начала мы рассмотрим всего три класса. Это SceneManager (Менеджер сцены), Entity (Объект сцены) и SceneNode (Узел сцены). Эти три класса являются основными "строителями" игровой сцены в GMOgre. Поэтому перво-наперво мы должны научится работать с ними.

SceneManager

Как понятно из перевода – это менеджер сцены. А кто такой менеджер? Правильно, управляющий. Соответственно SceneManager управляет всеми созданными объектами. В GMOgre менеджеры сцены могут быть нескольких типов. Это сделано для того, чтобы было проще управлять трехмерным миром. Давайте рассмотрим их:

- ST_GENERIC — общий менеджер сцены.
- ST_EXTERIOR_CLOSE — менеджер ландшафта
- ST_EXTERIOR_FAR — менеджер природы
- ST_EXTERIOR_REAL_FAR —
- ST_INTERIOR — менеджер BSP.

Пока это все, что вам нужно знать о SceneManager'e на первых порах изучения GMOgre. Позже мы остановимся на нем более подробно.

Entity

Entity – это объект GMOgre. Под ним понимается 3D модель формата *.mesh. У объектовэтого класса существуют специальные функции для управления ими, но о них мы поговорим позже.

К вышесказанному можно добавить, что объектами класса Entity являются именно 3D модели, созданные в редакторах трехмерной графики. А вот свет, полигон, камеры,созданные через функции GMOgre,уже не будут относится к этому классу, так как, по сути, не являются трехмерными объектами.

Нужно отметить, что GMOgre не загружает модели в сцену напрямую, так как не может самостоятельно определить положение и ориентацию загружаемого объекта в пространстве. Для задания этих свойств и существует так называемый узел сцены (SceneNode), в котором мы сами можем задать эти параметры.

SceneNode

Итак, SceneNode — это структурный класс, необходимый для размещения объектов на сцене, создания иерархий узлов, содержащих модели, дабы в последствии манипулировать ими было проще. Узел сцены хранит информацию о положении и ориентации связанного с ним объекта. Сам по себе SceneNode не является никаким объектом в GMOgre и не отображается на экране.

Нужно также отметить, что первый созданный вами узел привязан к главному (корневому) узлу, содержащемуся в менеджере сцены. Позиция каждого последующего узла всегда вычисляется относительно его родителя, а каждый новый SceneManagerсодержит в себе новый корневой узел, к которому принадлежат все другие узлы.

Пишем основу программы на GMOgre

Как известно, к любой библиотеке для GM (*.dll) прилагается ряд скриптов, в которых инициализируются ее функции. Библиотеки должны хранится вместе с файлом проекта в одной папке, иначе будет сбой при инициализации. Также можно изменить путь к библиотеке и расположить её в другом доступном месте, но для этого нужно прописать в скриптах путь к ней. В принципе, это не так важно, но знать необходимо.

Итак, перейдем от «лирического отступления» к подготовке проекта и непосредственно к практике.

Большинство игр создается на основе специального интерфейса — обычно либо DirectX, либо OpenGL. Если коротко, они, в целом, служат для вывода «картинки» на экран. На преимуществах и недостатках этих двух систем мы останавливаться не будем —в интернете достаточно рассуждений на эту тему.

В GMOgre доступен выбор между ними:

- RENDER_DX9 — DirectX 9
- RENDER_GL — OpenGL

Создайте объект obj_engine и разместите его в комнате.В событии Create необходимо разместить следующий код:

```
InitializeOgre3D();                    //инициализируем библиотеку
StartOgre3DEngine(global.RENDER_DX);   //выбираем интерфейс
```

Теперь нам необходимо указать папки с внешними ресурсами, чтобы GMOgre знал, где и что искать. Ogre поддерживает ZIP-архивы, что дает возможность хранить ресурсы в сжатом виде. Добавим места, где хранятся все наши ресурсы:

```
AddResourceLocation("./media/packs/OgreCore.zip",global.LOC_ZIP);
AddResourceLocation("./media/materials/programs",global.LOC_FILESYSTEM);
AddResourceLocation("./media/materials/scripts", global.LOC_FILESYSTEM);
AddResourceLocation("./media/materials/textures",global.LOC_FILESYSTEM);
AddResourceLocation("./media/models",            global.LOC_FILESYSTEM);
```

Параметр **global**.LOC_ZIPуказывает GMOgre о том, что данный ресурс является ZIP-архивом. Aglobal.LOC_FILESYSTEM,соответственно, —папкой, которая находится в локальной файловой системе.

Теперь, когда мы добавили пути к ресурсам, нам необходимо инициализировать все их группы. Для этого вызываем следующую функцию:

```
InitializeAllResourceGroups();
```

С ресурсами разобрались, теперь перейдем к созданию сцены, камеры и прочих атрибутов игры.

Создание своей сцены

Итак, мы запустили GMOgre, добавили папки с ресурсами, одним словом — подготовили GM для последующей работы с GMOgre. Но, все же, этого недостаточно для того, чтобы увидеть сцену…

Ранее мы уже говорили о SceneManager и вы знаете, для чего он нужен. Теперь нам необходимо создать его. Ниже представлен код создания менеджера сцены. В качестве параметра (типа менеджера) зададим ST_GENERIC:

```
CreateSceneManager(global.ST_GENERIC);
```

Теперь можно идти далее. Чтобы мы могли увидеть сцену, нам нужно создать вид и камеру к ней. Виды (viewports)очень напоминают виды в самом GM, поэтому, я думаю, в пояснении не нуждаются.

Итак, создаем вид:

```
view_id = CreateViewport(0,0,0,room_width,room_height);
```

Если с последними двумя аргументами все понятно (определяют ширину и высоту вида), то первые три нуждаются в пояснениях. Выстроим их по-порядку, начиная с первого:

- **zorder** – аргумент, определяющий числовой порядок вида в игре. Это как слои в фотошопе. Благодаря этому параметру один вид накладывается на другой в порядке убывания вследствие увеличения числа аргумента в положительную сторону относительно нуля. То есть, чем выше параметр, тем ниже в слое находится вид;
- **left** – x-координата левого верхнего угла вида в пикселях;
- **top** - y-координата левого верхнего угла вида в пикселях;
- **width** – ширина вида в пикселях;
- **height** – высота вида в пикселях.

Следующим нашим шагом будет создание камеры к нашему виду. Об устройстве камеры и способах их взаимодействия и управления мы поговорим на следующем уроке, а пока, не вдаваясь в тонкости, создадим камеру и присвоим ей вид.

Для этого создадим новый объект GM, назовем obj_camera и добавим его в комнату. В событие Create добавляем следующий код:

```
camera_id= CreateCamera(room_width/room_height,5,10000,45);
SetViewportCamera(view_id,camera_id);   //Присваиваем виду камеру
SetCameraPosition(camera_id,0,200,40);  //Задаем позицию камере
```

Рассмотрим аргументы функции CameraCreate:

- aspect - отношение высоты вида к ширине
- znear – расстояние до ближней плоскости отсечения
- zfar - расстояние до дальней плоскости отсечения
- fov – угол поля зрения.

Теперь, наконец, пришло время создать рабочую программу и посмотреть на то, что получилось. Поэтому, не откладывая, приступим к делу…

Первое, что мы сделаем — установим параметры окружения и освещения, иначе мы попросту не увидим на экране ничего, кроме черного пятна.

Освещение окружения задается функцией SetAmbientLigh(). В её аргументе задаем цвет:

```
SetAmbientLight(c_white);
```

Добавьте этот код в объект obj_engine.

Теперь загрузим и поместим 3D объект на сцену. Для этого необходимо сделать следующее. Создаем объект GM, присваиваем ему имя obj_entityи добавляем его в комнату. В событиеCreateэтого объекта помещаем следующий код:

```
ent_id = CreateEntity("Robot.mesh");
```

В качестве параметра этой функции передается имя файла модели в формате *.mesh. Движок будет искать этот файл в директории, указанной функцией AddResourceLocation.

Теперь, когда объект загружен, нам нужно создать узел для него. Так как у каждого менеджера сцены есть главный узел, мы будем создавать его потомка.

Потомок — это объект, изначально имеющий свойства в точности такие же, как и у родителя. То есть, например, если добавить потомка к кубу (например, шар) и двигать его, то и шар будет двигаться с ним. Все просто!

Далее запишем соответствующий код в событие Create нашего объекта:

```
node_id = CreateRootChildSceneNode();
```

Затем мы присвоим объект к узлу сцены, таким образом мы дадим роботу ориентацию (для отдельных лиц с особым чувством юмора, оговорюсь, что под ориентацией я подразумевал положение в пространстве :)

```
AttachEntityToSceneNode(ent_id,node_id);
```

Модель будет отображаться на сцене в центре экрана. Почему? Да потому что главный узел сцены, а соответственно и менеджер сцены, располагается в координатах {0,0,0}. Чтобы изменить положение объекта, достаточно ввести код:

```
SetSceneNodePosition (node_id,-355,130,55);
```

Заметьте, что мы присваиваем позицию не создаваемому объекту (ent_id), а созданному потомку сцены (node_id). Остальные три аргумента отвечают за координаты по осям X, Y и Z соответственно.

Мы почти закончили написание нашей первой рабочей программы на GMOgre и сейчас сделаем финальное действие: добавим нижеприведенную функцию в событие End Step объекта obj_engine, чтобы он смог, грубо говоря,«обновлять» показываемое изображение в конце каждого шага:

```
RenderFrame();
```

Координаты и Векторы

Прежде, чем мы перейдем далее, поговорим о экранных координатах и векторах. Как и GM, GMOgre использует три координатные оси – X, Y и Z, где Z – вертикальная ось.

Для хранения позиции и поворота GMOgreДиспользует векторы. Если вы их плохо знаете или не помните, что они из себя представляют, то советую повторить математику на данную тему — она очень поможет при дальнейшем использовании GMOgre (тем более если вы собрались писать серьезный проект).

Добавление другого Объекта

Теперь, когда вы знаете, как работает система координат, мы можем вернутся к нашему коду. Выше я привел пример, как назначит 3D объекту позицию. Рассмотрим еще один вариант, как это можно сделать.

У некоторых функции в GMOgre есть параметры по умолчанию: например, в CreateRootChildSceneNodeих шесть: три для начальной позиции узла, и три для начальной ориентации (вращения).Если вы оставите эти аргументы без изменений, то GMOgre автоматически определит позицию и ориентацию модели как {0,0,0}. Но на сей раз мы все же зададим параметры для нашего объекта таким способом. Для этого создадим новый узел сцены и в наш объект obj_entity добавим еще пару строк в событие Create:

```
ent2_id = CreateEntity ("Robot.mesh");
node2_id = CreateRootChildSceneNode (50, 0, 0);
AttachEntityToSceneNode (ent2_id);
```

Надеюсь, это выглядит для вас знакомо. Мы сделали почти тоже самое, что и выше, но с двумя различиями. Первое — мы создали новый узел сцены и загрузили новый объект, присвоив ему новое имя. Второе — ввели местоположение объекта, через функцию CreateRootChildSceneNode. В первый аргумент функции мы ввели значение 50. Это говорит о том, что теперь модель будет находится в 50 единицах расстояния от центра узла по координате X.

Подробнее о объектах

Есть множество функций для трансформации моделей и описывать их все было бы крайне нерентабельно. Поэтому я расскажу только о самых необходимых, а об остальных вы и сами прочтете в официальном руководстве.

Рассмотрим сначала функции ShowEntity и HideEntity.С их помощью вы можете показать или скрыть объект. Если вам нужно скрыть объект на небольшой период времени, то вызывайте соответствующую функцию, вместо того, чтобы уничтожать его и потом создавать снова.

Подробнее о узлах сцены

Узлы сцены довольно сложны и запутанны. С ними можно проделать множество разных вещей, но пока мы рассмотрим только самые главные. Например, можно получить позицию узла сцены путем вызова функции GetSceneNodePosition, атак же установить позицию для любого объекта GMOgre с помощью функции SetSceneNodePosition.

Мы также можем изменить позицию объекта относительно его предыдущей позиции. Для этого нам понадобится функция TranslateSceneNode. Узел сцены может производить любую трансформацию объекта. Например, мы можем вращать объект, задав угол вращения.Есть три функции для вращения: PitchSceneNode, YawSceneNodeи RollSceneNode. Можно также использовать функцию ResetSceneNodeOrientation, чтобы сбросить все предыдущие вращения, примененные к объекту. Мы можем также использовать SetSceneNodeOrientation, GetSceneNode-Orientation, и RotateSceneNodeдля вращения с более точными параметрами.

Теперь, когда вы знаете, как и что можно сделать с объектом, я добавлю еще, что мы можем создать потомка для объекта. У нас в настоящее время есть этот код в obj_entity, Измените выделенные строчки на приведенные ниже:

```
ent1_id = CreateEntity("Robot.mesh");
node1_id = CreateRootChildSceneNode();
AttachEntityToSceneNode(ent1_id,node1_id);

ent2_id = CreateEntity("Robot.mesh");
node2_id = CreateRootChildSceneNode(50,0,0);
AttachEntityToSceneNode(ent2_id,node2_id);
```

Мы изменили выделенную строку:

```
node2_id = CreateRootChildSceneNode(50,0,0);
```

На это:

```
node2_id = CreateChildSceneNode(node1_id,50,0,0);
```

Таким образом мы сделали node2_id потомком node1_id.Теперь потомок будет перемещаться за родителем. Например, мы можем добавить этот код к RobotNode:

```
TranslateSceneNode (node2_id, 10, 0, 10);
```

Посмотрим, как это все выглядит математически. Начнем с узла сцены. Его позиция всегда {0,0,0}. Позиция node1 будет выражена через {0,0,0} + {25, 0,0} = {25,0,0}.Теперь node2: {,0,0} + {25,0,0} + {60,0,10} ={85,0,10}. Таким образом нетрудно продемонстрировать вычисление абсолютных позиций узлов сцены, так сказать, «на пальцах». Однако в действительности все происходит немного по-другому. Ведь нужно учесть, что при вращении узла-родителя, координатная система его потомка тоже вращается. Вычисление абсолютных координат при этом осуществляется при помощи особых математических объектов — матриц. Одна матрица может хранить сразу позицию, вращение и масштаб объекта. Абсолютная трансформация узлов производится путем умножения этих матриц:

{0,0,0} * {матрица node1} * {матрица node2}

Чтобы задать масштаб для узла сцены, добавьте в код следующую функцию:

```
SetSceneNodeScale(node2_id,1,2,1);
```

Вращение осуществляется функциями YawSceneNode (вокруг оси X), PitchSceneNode (Y) и RollSceneNode (Z):

```
YawSceneNode(node1_id,-90);
PitchSceneNode(node2_id,-90);
RollSceneNode(node3_id,-90);
```

Заключение

Подведем итог. Мы рассмотрели основные принципы работы с GMOgre, в частности изучили три класса – узел сцены, менеджер сцены и объект. Более подробно вы сможете узнать о них из официальной справки — и, разумеется, из следующих уроков.

Rutraple
Pb_vrn_36@mail.ru

Язык AngelScript

В последние годы производительность компьютеров резко возросла. Как следствие, изменились и приоритеты при разработке ПО: на первом плане уже не оптимизация, а портируемость. На гребне волны этих тенденций поднялись такие кроссплатформенные продукты и технологии, как Python, Java, .Net/Mono, Web 2.0 и прочие. И разработка программ полностью на скриптовом языке — уже не баловство, а вполне адекватное решение. В конце концов, для обладателя Intel Core i7 не будет особой разницы, на чем написан, например, его медиаплеер — на чистом C или на Python. Так что игнорировать интерпретируемые языки в наши дни — по меньшей мере, снобизм.

Сегодня я хочу познакомить читателей журнала с одним из интереснейших представителей скриптовых языков — AngelScript. Его неоднозначная особенность заключается в том, что он имеет строгую типизацию (это роднит его с C/C++, Pascal и другими «столпами» программирования). С одной стороны, принято считать, что в скриптовых языках это совершенно излишне, так как скрипты пишут, в основном, «непрограммеры» (художники, левелдизайнеры — одним словом, больше пользователи, нежели разработчики), а их почему-то считают невероятно отсталыми и неспособными отличить int от float. С другой стороны, если переложить задачу определения типов с транслятора на пользователя, можно получить выйгрыш в производительности. Скорость AngelScript в большинстве случаев сопоставима с Lua (а в отдельных ситуациях — даже превосходит), в то время как регистрация функций и классов в AngelScript делается гораздо проще. Синтаксис AngelScript максимально приближен к C++, что, несомненно, для многих является огромным плюсом (ну не перевариваю я всякие там begin и end или табуляцию в Python — что я могу поделать?).

http://www.angelcode.com/angelscript/

Библиотека AngelScript полностью бесплатна и открыта, распространяется в виде исходных кодов под лицензией zlib. К ним прилагается большое число проектных файлов для разных компиляторов и IDE (есть и для Dev-C++). Можно скомпилировать AngelScript как статическую или динамическую библиотеку (лично я предпочитаю последнее). Как и полагается, в комплекте также идут примеры и руководство.

AngelScript вобрал в себя лучшее от C++ и других C-подобных языков. Однако в нем, например, нет любимых всеми пространств имен, шаблонов, спецификаторов доступа — словом, всего того, с чем, добровольно или вынужденно, приходится сталкиваться любому программисту на C++. Но скрипт — это не полноценная программа, поэтому даже тех аспектов ООП, что реализованы в AngelScript, более чем достаточно. Судите сами: наследование классов (все методы — виртуальные), перегрузка функций, методов, операторов и переменных, работа с указателями (правда, поддерживаются только указатели на объекты). При желании на уровне приложения можно зарегистрировать свои типы и классы. AngelScript компилирует скрипты в байт-код (на данный момент ведется работа над JIT-компилятором) и включает полноценную виртуальную машину с поддержкой многопоточности.

Рассмотрим выполнение простейшего скрипта «Hello World»:

```
void main()
{
   print("Hello world\n");
}
```

Кроме основного исходника, в проект необходимо включить файлы scriptstring.cpp, scripthelper.cpp, scriptbuilder.cpp из аддонов в официальном дистрибутиве AngelScript. Не забудьте указать библиотеку AngelScript в настройках компоновщика.

Включаем необходимые заголовки:

```
#include <cstdlib>
#include <iostream>
#include "angelscript.h"
#include "scriptstring.h"
#include "scripthelper.h"
#include "scriptbuilder.h"
using namespace std;
```

Объявляем функции:

```
void print(string &msg)
{
   printf("%s", msg.c_str());
}

void MessageCallback(const asSMessageInfo *msg, void* param)
{
   const char *type = "ERR ";
   if( msg->type == asMSGTYPE_WARNING ) type = "WARN";
   else if( msg->type == asMSGTYPE_INFORMATION )
      type = "INFO";
   printf("%s (%d, %d) : %s : %s\n", msg->section,
      msg->row, msg->col, type, msg->message);
}
```

Функция print будет зарегистрирована в AngelScript, чтобы скрипт мог ее вызывать.

В главной функции создаем скриптовый движок:

```
int main(int argc, char *argv[])
{
   asIScriptEngine *engine =
   asCreateScriptEngine(ANGELSCRIPT_VERSION);
   CScriptString *p_name = new CscriptString("string1");
   engine->SetMessageCallback(
      asFUNCTION(MessageCallback),0,asCALL_CDECL);
      RegisterScriptString(engine);
```

Затем зарегистрируем функцию print:

```
   engine->RegisterGlobalFunction(
      "void print(const string &in)",
      asFUNCTION(print), asCALL_CDECL);
```

Создаем новый модуль (насколько я понял, модуль в AngelScript — это нечто вроде объектного файла), загружаем в него скрипт и компилируем:

```
      CScriptBuilder builder;
      builder.StartNewModule(engine, "MyModule");
      builder.AddSectionFromFile("scripts/sample.as");
      builder.BuildModule();
```

Узнаем идентификатор главной функции в скрипте (main):

```
   asIScriptModule *mod = engine->GetModule("MyModule");
   int funcId = mod->GetFunctionIdByDecl("void main()");
```

После удачного завершения нашей функции можно уничтожить контекст и движок:

```
   ctx->Release();
   engine->Release();
```

А затем и завершить работу программы:

```
   return EXIT_SUCCESS;
}
```

Gecko
clocktower89@mail.ru

GLSL в OpenGL 1.4

Одно из преимуществ OpenGL — механизм расширений, благодаря которому программисты могут использовать возможности современных видеокарт даже со старыми спецификациями — лишь бы эти возможности поддерживались драйвером. Так, например, для использования шейдерного языка GLSL совсем необязательна полная поддержка OpenGL 2.0.

Для начала убедитесь, что драйвер поддерживает расширения ARB_vertex_shader и ARB_fragment_shader. Это можно сделать при помощи специальных утилит или бенчмарков, но, на мой взгляд, гораздо проще так:

```
printf("%s\n",glGetString(GL_EXTENSIONS));
```

Далее необходим способ загрузки расширений. Я рекомендую для этого библиотеку GLee (http://elf-stone.com).

Создаем шейдер:

```
GLenum shader_vert = 0;
GLenum shader_frag = 0;
GLenum shader_prog = 0;
shader_prog = glCreateProgramObjectARB();
shader_vert = glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
shader_frag = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
```

Я не буду описывать процесс загрузки кода шейдеров из текстовых файлов, так как для этого у каждого есть свои средства. Допустим, что вершинная и фрагментная программы уже загружены:

```
char* shader_vert_source;
char* shader_frag_source;
```

Передаем исходники программ:

```
int len = strlen(shader_vert_source);
glShaderSourceARB(shader_vert, 1,
       (const GLcharARB**)&shader_vert_source, &len);
len = strlen(shader_frag_source);
glShaderSourceARB(shader_frag, 1,
       (const GLcharARB**)&shader_frag_source, &len);
```

Компилируем и компонуем шейдер:

```
glCompileShaderARB(shader_vert);
glCompileShaderARB(shader_frag);
glAttachObjectARB(shader_prog, shader_vert);
glAttachObjectARB(shader_prog, shader_frag);
glLinkProgramARB(shader_prog);
```

Проверяем, нет ли ошибок:

```
char infobuffer[1000];
int infobufferlen = 0;
glGetInfoLogARB(shader_vert, 999,
       &infobufferlen, infobuffer);
printf("Vertex shader: %s\n",infobuffer);
glGetInfoLogARB(shader_frag, 999,
       &infobufferlen, infobuffer);
printf("Fragment shader: %s\n",infobuffer);
```

Совершаем отрисовку:

```
glUseProgramObjectARB(shader_prog);
...
glUseProgramObjectARB(0);
```

Если фрагментная программа использует текстуру, перед glUseProgramObjectARB передаем соответствующий параметр:

```
GLint uniform_location =
glGetUniformLocationARB(shader_prog, "pTexture");
glUniform1iARB(uniform_location, 0);
glActiveTextureARB(GL_TEXTURE0_ARB);
glBindTexture(GL_TEXTURE_2D,glTexture);
```

Gecko
clocktower89@mail.ru

● Шейдеры на все случаи жизни ●

Освещение по Фонгу

В современных играх простое вершинное освещение используется уже довольно редко, почти полностью уступив место пиксельному. Сегодня мы рассмотрим шейдер, реализующий один из методов пиксельного освещения — по Фонгу — через расширения ARB_vertex_program и ARB_fragment_program. При желании его нетрудно переписать на GLSL или Cg. За основу я взял шейдер из GLScene, внеся в него несколько изменений (в частности, поддержку текстуры). Фрагментная программа теперь не требует никаких параметров и все необходимые данные читает из параметров состояния OpenGL.

1. Вершинная программа:

```
!!ARBvp1.0
OPTION ARB_position_invariant;

PARAM mvinv[4] = {state.matrix.modelview.inverse};
PARAM mvit[4] = {state.matrix.modelview.invtrans};
PARAM lightPos = state.light[0].position;
TEMP light, normal, eye;

ADD eye, mvit[3], -vertex.position;
MOV eye.w, 0.0;

DP4 light.x, mvinv[0], lightPos;
DP4 light.y, mvinv[1], lightPos;
DP4 light.z, mvinv[2], lightPos;
ADD light, light, -vertex.position;
MOV light.w, 0.0;

MOV result.texcoord[0], vertex.texcoord[0];
MOV result.texcoord[1], vertex.normal;
MOV result.texcoord[2], light;
MOV result.texcoord[3], eye;

END
```

2. Фрагментная программа:

```
!!ARBfp1.0

PARAM materialAmb = state.material.ambient;
PARAM materialDiff = state.material.diffuse;
PARAM materialSpec = state.material.specular;
PARAM shininess = state.material.shininess;
TEMP temp, light, normal, eye, R, amb, diff, spec, col;

DP3 temp, fragment.texcoord[1], fragment.texcoord[1];
RSQ temp, temp.x;
MUL normal, temp.x, fragment.texcoord[1];
DP3 temp, fragment.texcoord[2], fragment.texcoord[2];
RSQ temp, temp.x;
MUL light, temp.x, fragment.texcoord[2];
DP3 temp, fragment.texcoord[3], fragment.texcoord[3];
RSQ temp, temp.x;
MUL eye, temp.x, fragment.texcoord[3];

DP3_SAT diff, normal, light;
MUL diff, diff, materialDiff;
ADD diff, diff, materialAmb;

DP3 R, normal, light;
MUL R, R.x, normal;
MUL R, 2.0, R;
ADD R, R, -light;
DP3_SAT spec, R, eye;
POW spec, spec.x, shininess.x;
MUL spec, spec, materialSpec;

TEX col, fragment.texcoord[0], texture[0], 2D;

MUL col, col, diff;
ADD col, col, spec;

MOV result.color, col;
MOV result.color.w, 1.0;

END
```

Перед тем, как использовать шейдер, необходимо определить источник света, материал и текстуру:

```
glEnable (GL_LIGHTING);
glEnable (GL_LIGHT0);
GLfloat lightpos[] = { 5.0f, 10.0f, 0.0f, 1.0f };
glLightfv (GL_LIGHT0, GL_POSITION, lightpos);

glEnable(GL_COLOR_MATERIAL);
GLfloat ColorAmbient[4] = {0.2f, 0.2f, 0.2f, 1.0f };
GLfloat ColorDiffuse[4] = {1.0f, 1.0f, 1.0f, 1.0f };
GLfloat ColorSpecular[4]= {1.0f, 1.0f, 1.0f, 1.0f };
GLfloat HighShininess[4]= {15.0f,1.0f, 1.0f, 1.0f };
glMaterialfv(GL_FRONT, GL_AMBIENT,   ColorAmbient);
glMaterialfv(GL_FRONT, GL_DIFFUSE,   ColorDiffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,  ColorSpecular);
glMaterialfv(GL_FRONT, GL_SHININESS, HighShininess);

glBindTexture(GL_TEXTURE_2D,tex);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_TEXTURE_2D);
```

Gecko
clocktower89@mail.ru

Рис. 1

DevPak — это стандарт пакетов, изначально разработанный для IDE Dev-C++. Однако, например, Lazarus имеет собственный формат пакетов. Пакет может содержать исходники, заголовочные файлы, статические или динамические библиотеки, шаблоны, документацию и т.д. Пакеты содержатся в специальном сетевом хранилище — репозитории. Специальная программа — менеджер пакетов — позволяет соединится с репозиторием и установить в IDE любой пакет. Позднее она же поможет обновить или удалить пакеты. Такой подход был унаследован из Unix-подобных ОС, где мощные менеджеры пакетов (apt, rpm, autopackage, ebuild, portage и др.) позволяют устанавливать абсолютно любые компоненты системы, вплоть до ядра и его модулей.

Если вы не нашли в репозитории нужный вам DevPak, его совсем нетрудно собрать самостоятельно. Рассмотрим это на примере AngelScript — свободного кроссплатформенного скриптового языка. Такой DevPak, правда, уже есть (на devpaks.org), но мы соберем более свежую версию — 2.18.1.

1. Для начала вам, ясное дело, нужно собрать AngelScript или достать откомпилированный вариант.

2. Создайте директорию AngelScript-2.18.1-1xxx. Соответственно, после сборки пакета мы получим файл AngelScript-2.18.1-1xxx.DevPak. С именем файла пакета связано несколько правил. Их придерживаться необязательно, но желательно (особенно если вы потом захотите выложить свой DevPak в репозиторий). В первой части имени находится название устанавливаемого компонента (AngelScript). Во второй — версия (2.18.1). В третьей — ревизия пакета (1xxx). Ревизия состоит из номера сборки (1) и идентификатора (xxx), который может быть сокращением вашей фамилии или никнейма. Например, в моем случае имя пакета будет следующим: AngelScript-2.18.1-1gec.DevPak.

Как собрать DevPak?

3. Внутри нашей директории создайте еще одну и назовите ее AngelScript. В ней создайте директории docs, examples, include, lib и templates. В docs поместите файлы документации, в unclude — заголовочные файлы, в lib — статическую библиотеку libangelscript.a. Папки examples и templates можно пока оставить пустыми.

4. В директории AngelScript-2.18.1-1xxx создайте файл AngelScript.DevPackage следующего содержания:

```
[Setup]
Version=1
AppName=AngelScript
AppVersion=2.18.1
AppVerName=AngelScript 2.18.1
MenuName=AngelScript
Description=AngelScript is a library designed to allow applications to extend their
functionality through external scripts.
Url=http://www.angelcode.com/angelscript/
Dependencies=
License=COPYING.txt
Readme=README.txt

[Icons]
Website=http://www.angelcode.com/angelscript/
Online Documentation=http://www.angelcode.com/angelscript/sdk/docs/manual/index.html

[Files]
AngelScript=<app>\
```

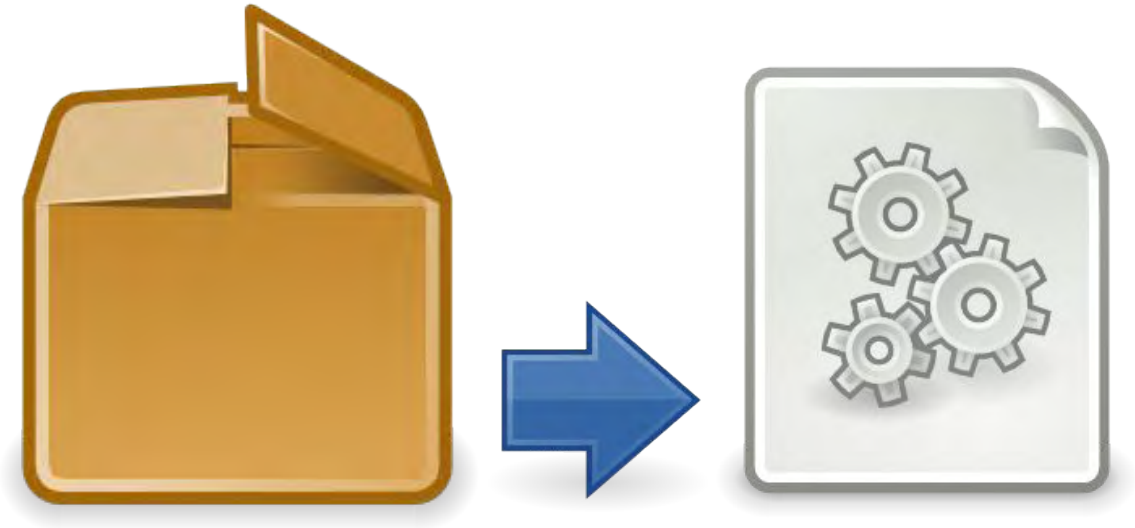
Этот файл конфигурации используется менеджером пакетов для получения информации о DevPak'е и его структуре. Вам понадобится указать имя и версию устанавливаемого компонента (AppName, AppVersion, AppVerName), краткое описание (Description), файлы лицензионного соглашения (License) и Readme (Readme), а также адреса официального сайта и онлайн-документации. Вся эта информация будет выведена при установке пакета.

5. Не забудьте поместить в директорию AngelScript-2.18.1-1xxx файлы COPYING.txt и README.txt. Ими обычно сопровождается любой архив с исходниками, и AngelScript — не исключение.

6. Осталось только создать из нашей директории архив в формате tar.bz2. В Windows это можно сделать, например, удобным архиватором PeaZip. Для тех, кто не в курсе — в Unix-подобных ОС вместо zip и rar обычно используются свои форматы, предназначенные для сжатия так называемого tarball (несжатого архива с расширением *.tar, история которого уходит корнями во времена ранних ЭВМ, которые записывали данные на магнитную ленту). Это tar.bz2 (bzip2), tar.gz (gzip) и tar.lzma.

7. У полученного архива смените расширение на *.DevPak и — готово! Можно запускать менеджер пакетов и тестировать.

Gecko
clocktower89@mail.ru



Оформляем Makefile

Большинство начинающих программистов обычно выбирают готовый «джентльменский набор» в виде компилятора, специализированного редактора (IDE), отладчика и других утилит. Не спорю, это удобно. Но такой подход чреват привыканием к конкретной среде. Такие программисты зачастую злоупотребляют различными излишествами (вроде отклоняющихся от стандартов директив и конструкций) и уже не мыслят работы без редактора форм и визуальных компонентов. Многие даже не умеют «общаться» с компилятором напрямую, без посредника в лице IDE. Однако среди Linux-програмистов вы таких не встретите. С чем же это связано?

По себе знаю: при переходе на Linux так и тянет к минимализму. Стремишься устанавливать и запускать только необходимый минимум приложений, чтобы сэкономить как можно больше ресурсов для решения своих задач. При этом возникает необходимость работать с командной строкой, писать различные скрипты автоматизации и т.д. Сборку программы тоже можно автоматизировать. Наверняка вы слышали о GNU Make. Даже далекие от программирования пользователи частенько предпочитают собирать программы из исходников, чтобы оптимизировать их под свои системные конфигурации – это дает ощутимый рост производительности (при желании можно даже всю систему собрать из исходников; для этого существуют специальные дистрибутивы – например, Gentoo или LFS). Так что команда ./configure && make && make install знакома почти каждому линуксоиду. Make – это утилита для автоматизации операций по преобразованию файлов. Чаще всего ее используют для генерации команд компилятору. Данная статья рассматривает работу make с компилятором GCC.

Программа make выполняет команды согласно правилам в специальном файле. Этот файл называется make-файл (makefile). Как правило, make-файл описывает, каким образом нужно компилировать и компоновать программу. Так что вся работа сводится к написанию правильного make-файла.

Научиться этому несложно: достаточно посмотреть, как он выглядит в исходниках простых демонстрационных програм (читать make-файлы крупных продуктов начинающим, а также слабонервным не рекомендую – чревато :) Гораздо труднее написать такой make-файл, который не придется переписывать по сто раз по мере увеличения проекта. Он должен быть компактным и читаемым. Он должен уметь автоматически выстраивать цепочки компиляции объектных файлов по заданным исходникам. В нем должно быть легко менять пути к исходникам и объектникам. Наконец, он должен быть таким, чтобы вы сами могли в нем разобраться по прошествии времени.

Я предлагаю такой вариант:

```
TARGET = mygame
SRCDIR = ./source
OBJDIR = ./source/obj
SRCNAMES = main.cpp Objects.cpp Sound.cpp
CXX = g++
CXXFLAGS = -O3
LIBS = -L"." -lGLU -lGL -lSDL libmystatic.a libmyshared.so
INCLUDE = -I"./source/include"
OBJ = $(addprefix $(OBJDIR)/, $(notdir $(SRCNAMES:.cpp=.o)))

all:   $(OBJ)
       $(CXX) $(CXXFLAGS) -o $(TARGET) $(OBJ) $(LIBS) -Wl,-rpath,.
       ./$(TARGET)

$(OBJDIR)/%.o: $(SRCDIR)/%.cpp
       $(CXX) $(CXXFLAGS) $(INCLUDE) -o $@ -c $<

clean:
       rm $(OBJDIR)/*.o

.PHONY: all clean
```

TARGET – целевой объект. Это имя исполняемого файла, который вы получите в результате сборки;

SRCDIR – директория с исходными файлами (*.cpp);

OBJDIR – директория с объектными файлами (*.o). В данном примере я решил отделить их от исходников, чтобы не мешались;

SRCNAMES – список исходных файлов. Сюда можно добавлять новые по мере их появления;

CXX – компилятор;

CXXFLAGS – флаги оптимизации. Набор доступных флагов зависит от используемого компилятора. Обычно здесь пишут -O1, -O2 или -O3, что соответствует трем уровням оптимизации. -O3 – самый высокий. Еще можно добавить, например, -mtune=pentium4 – это оптимизирует код под процессор Intel Pentium 4. А флаг -Os поможет уменьшить размер исполняемого файла. Более подробные сведения ищите в документации к вашей версии GCC;

LIBS – список библиотек, подключаемых при компоновке, а также опция -L ".", указывающая линкеру, что библиотеки следует искать и в текущей директории (относительно make-файла, разумеется);

INCLUDE – директории, в которых следует искать стандартные заголовочные файлы;

OBJ – список объектных файлов. Он автоматически генерируется из списка исходных файлов;

all – цель для сборки всего проекта и запуска целевого исполняемого файла. Вызывается, если подать команду make или make all. Обратите внимание, что при этом пересобираются не все объектные файлы, а только те, в чьих исходниках произошли изменения – таким образом, даже крупный проект будет собираться довольно быстро;

$(OBJDIR)/%.o – цель для сборки объектного файла. Вы можете, например, подать команду make Objects.o;

clean – цель для удаления всех объектных файлов. Ее можно использовать для пересборки проекта «с нуля», когда в проекте произошли кардинальные изменения, которые не прослеживаются автоматически: make clean && make all;

.PHONY – указывает абстрактные цели, которые не служат для сборки какого-то конкретного файла. К ним относятся all и clean. Если вы добавляете свои новые цели (например, для сборки проекта с различной конфигурацией или запуска с различными опциями), не забудьте указать их как .PHONY.

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