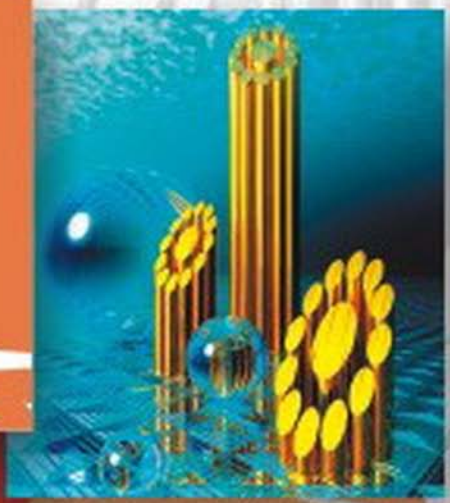
СТАНИСЛАВ ГОРНАКОВ

bhv

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПРОГРАММИРОВАНИЯ и ОТЛАДКИ ШЕЙДЕРОВ в DirectX и OpenGL



- ОСНОВЫ DirectX
- АССЕМБЛЕРНЫЙ ЯЗЫК ШЕЙДЕРОВ
- ПРОФАЙЛЕР PIX for Windows
- ATI RenderMonkey 1.5
- NVIDIA FX Composer 1.5

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Станислав Горнаков

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПРОГРАММИРОВАНИЯ И ОТЛАДКИ ШЕЙДЕРОВ В DirectX И OpenGL

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06
ББК 32.973.26-018.2
Г67

Горнаков С. Г.

Г67 Инструментальные средства программирования и отладки шейдеров в DirectX и OpenGL. — СПб.: БХВ-Петербург, 2005. — 256 с.: ил.

ISBN 5-94157-611-0

Рассматриваются основы DirectX, показаны приемы работы с фиксированным и программируемым графическими конвейерами, дана информация по применению профайлера PIX for Windows, необходимого для отладки программ в DirectX, подробно представлена информация об инструментариях ATI RenderMonkey 1.5 и NVIDIA FX Composer 1.5, позволяющих создавать и отлаживать шейдеры в DirectX и OpenGL.

Для программистов графики

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Рыбинский*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 26.05.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 20,64.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-611-0

Оглавление

Предисловие

Индустрия компьютерных развлечений находится на подъеме и пользователя уже не устраивает качество графики в прежних компьютерных играх. Никто больше не захочет играть в игры с рисованной и "квадратной" графикой, тем более что существующие компьютерные системы могут прорисовывать графику на достаточно высоком уровне. Новая линейка видеокарт от компаний ATI и NVIDIA улучшают ситуацию в игровой индустрии и позволяют разработчикам создавать действительно красивые трехмерные игры. В свою очередь разработчики постепенно отказываются от использования устаревшего фиксированного конвейера при визуализации сцен в компьютерных играх. Ему на смену пришла новая модель программируемого конвейера на основе вершинных и пиксельных шейдеров. На сегодняшний и даже завтрашний день использование шейдеров в программировании графики — это приоритетное направление, как для производителей программного обеспечения, так и для производителей аппаратного обеспечения. В целом же программирование графики с использованием вершинных и пиксельных шейдеров довольно сложная и трудоемкая задача для программиста. Большое количество различных инструкций в исходном коде ассемблерного языка программирования шейдеров никак не способствуют ее облегчению. Отчасти проблема со сложностью в программировании шейдеров решается с помощью высокоуровневого языка шейдеров. В DirectX это HLSL (High-Level Shader Language, высокоуровневый язык программирования шейдеров), а в OpenGL — язык GLSL (GL Shader Language, язык программирования шейдеров в OpenGL). Оба этих языка в чем-то схожи и решают одну и ту же задачу: повышение качества компьютерной графики при максимальном упрощении семантики языка на основе C-подобного синтаксиса исходного кода. Высокоуровневые языки программирования шейдеров — это будущее игровой индустрии компьютерных развлечений.

Для написания и редактирования программ шейдеров можно использовать различные средства, но гораздо лучше воспользоваться специализированными инструментальными средствами с набором мощных функциональных возможностей, в которых вам и поможет разобраться эта книга.

О чем эта книга

В связи с большим объемом предлагаемого материала книга разбита на несколько частей. *Часть I* книги знакомит читателя с основами программирования графики под DirectX 9.

Глава 1 — это введение в DirectX 9, где вы узнаете, из каких компонентов состоит DirectX 9, на чем он базируется и для чего необходим.

Глава 2 посвящена рассмотрению фиксированного графического конвейера с использованием матричных преобразований, работе с освещением и материалом на основе примера, исходный код которого вы найдете на компакт-диске.

Глава 3 объясняет работу с программируемым графическим конвейером на основе вершинных и пиксельных шейдеров. В частности, рассматриваются основы ассемблерного языка программирования шейдеров третьей версии с полной семантикой команд или инструкций по всем трем версиям шейдеров.

Часть II книги освящает работу с профайлером PIX for Windows, необходимым для отладки программ под DirectX.

Глава 4 содержит описание возможностей профайлера PIX for Windows, здесь также разбирается процесс инсталляции DirectX 9 Update (Summer 2004), в поставку которого входит PIX for Windows.

Глава 5 иллюстрирует работу с профайлером PIX for Windows. Здесь объясняется работа с параметрами сборки информации, с входным и выходным файлом сборки и анализа полученной информации.

Часть III книги познакомит вас с инструментальным средством разработки приложений RenderMonkey 1.5 компании ATI.

Глава 6 в общих чертах представляет инструментарий RenderMonkey 1.5 от процесса инсталляции до полного обзора интерфейса и возможностей.

Глава 7 на примере показывает возможности RenderMonkey 1.5 в программировании шейдеров и при работе с эффектами. Также рассматривается пакет SDK RenderMonkey 1.0, необходимый для написания подключаемых модулей (plug-in) в RenderMonkey 1.5.

Часть IV книги посвящена анализу интегрированной среды разработки приложений FX Composer 1.5 компании NVIDIA.

Глава 8 знакомит читателя со структурой FX Composer 1.5 и его возможностями.

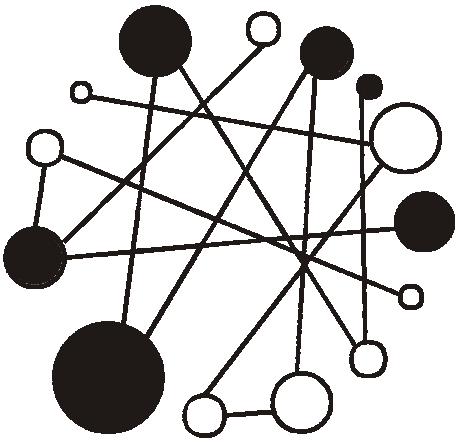
Глава 9 посвящена примеру, на основе которого будут показаны первостепенные принципы работы с FX Composer 1.5.

Что необходимо знать

Предполагается, что читатель знаком с основами языка программирования C++ и имеет представление о создании программ с использованием DirectX. Однако в первой части книги дается общая информация по DirectX 9, поэтому читать эту книгу может и начинающий программист, не знакомый с основами DirectX.

Благодарности

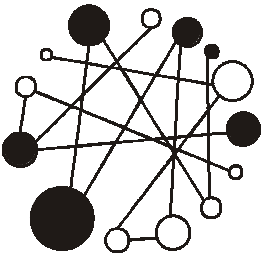
Хочу поблагодарить свою жену Светлану за всестороннюю помощь в создании книги. Особая благодарность Игорю Рыбинскому и издательству "БХВ-Петербург" за предоставление возможности написания этой книги.



ЧАСТЬ I

Основы программирования графики в DirectX 9

ГЛАВА 1



Введение в DirectX 9

"Мы связываем большие надежды
с новой платформой XNA".

Основатель корпорации Microsoft, Билл Гейтс

Большое количество компьютерных систем всего мира построено на комплектующих различных производителей. В свою очередь производители компьютерных комплектующих пользуются персональными наработками в технологическом процессе создания материнских плат, видеоадаптеров, звуковых и сетевых карт, используя при этом различные чипсеты, графические и звуковые процессоры, наборы микросхем и системной логики, постоянно улучшая, модернизируя и внедряя новые технологии в компьютерное производство. Это очевидным образом сказывается на развитии компьютерной индустрии в целом. В связи с чем компьютерный рынок изобилует множеством моделей компьютеров с разнообразными техническими характеристиками. С одной стороны, богатый выбор комплектующих подразумевает конкуренцию и, как следствие, конкурентные цены (что очень хорошо для простого обывателя), но с другой стороны — это должно составлять некоторые проблемы для программистов. Представьте себя эдаким матерым программистом (а может вы таковым и являетесь), задумавшим создать мощную трехмерную игру под операционную систему Windows. Отодвинем на задний план сложности создания графического движка игры, искусственного интеллекта, хорошей звуковой дорожки, а учтем лишь необходимость работы игры на любом аппаратном обеспечении компьютерной системы. В этой ситуации необходимо предусмотреть работу игры на всех имеющихся моделях видеоадаптеров, звуковых и сетевых картах. Как вы думаете, это реально? Боюсь, что такое можно предположить только в самых смелых мечтах. Совсем другое дело происходит при использовании технологии DirectX, разработанной корпорацией Microsoft.

Не вдаваясь в технологические подробности, можно сказать, что DirectX основывается на трех китах. Первый кит — это производители аппаратного

обеспечения, которые обеспечивают поддержку DirectX в изготовляемых устройствах. Второй кит — это операционная система Windows, поддерживающая DirectX. И третий кит — инструментальный пакет разработчика или DirectX SDK. Пакет разработчика DirectX SDK содержит большой набор готовых интерфейсов, классов, функций, макросов, структур и констант, значительно упрощающих разработку компьютерных игр для операционной системы Windows. На данный момент разработчикам доступен пакет DirectX 9.0 SDK Update (Summer 2004), включающий в себя ряд обновлений по сравнению с двумя предыдущими версиями DirectX 9b и DirectX 9.

Модель составных компонентов

DirectX 9 построен на независимой спецификации, декларирующей создание программных компонентов COM (Component Object Model). Архитектура COM-технологии очень четко разграничена на COM-объекты и COM-интерфейсы. В DirectX COM-объекты представлены компонентами Direct3D, DirectInput, DirectMusic, DirectSound, DirectPlay, DirectShow и DirectSetup. Для того чтобы получить доступ к COM-объекту, необходимо воспользоваться COM-интерфейсом, который в свою очередь состоит из массива указателей на функции. С помощью этих функций достигается доступ к COM-объектам. Названия COM-интерфейсов начинаются с английской заглавной буквы "`I`", например `IDirect3D`. Для того чтобы получить доступ к COM-объекту, необходимо создать переменную, в которой впоследствии будет храниться указатель на интерфейс.

Компоненты DirectX

DirectX 9 предоставляет возможность программисту разрабатывать программы, осуществляющие работу со звуком, сетью, трехмерной графикой, видео, и поэтому DirectX построен в виде набора *компонентов*. Это было хорошей идеей — не сбрасывать все в одну общую массу, а отделить, например, классы для работы с устройствами ввода-вывода от классов, обеспечивающих работу со звуком.

DirectX 9 имеет несколько компонентов.

- *DirectX Graphics* отвечает за возможность вывода на экран двухмерной и трехмерной графики с помощью *DirectDraw* и *Direct3D*. DirectDraw позволяет рисовать на экране двухмерную графику, однако этот компонент давно не обновлялся и находится как бы в замороженном состоянии, поэтому его текущая версия числится под номером семь, т. е. DirectDraw 7. Если вы все же желаете воспользоваться этим компонентом для рисования двухмерной графики, то сделать это возможно, но не обязательно и даже не желательно. Direct3D тоже обладает возможностью рисования

двухмерных фигур, и лучше пользоваться именно этим компонентом. В последней версии DirectX 9.0 SDK Update (Summer 2004) корпорация Microsoft рекомендует воздерживаться от использования устаревшего DirectDraw. Все это, в частности, связано с созданием новой платформы под названием XNA, о которой мы поговорим в этой главе в *разд. "Платформа XNA"*.

- *DirectInput* обеспечивает взаимодействие программы с устройствами ввода, а именно с клавиатурой, мышью, джойстиком и рулем. Вы, наверно, скажете: "Но платформа Windows имеет свои стандартные средства для работы с устройствами ввода". Дело в том, что скорость доступа в играх должна быть максимально быстрой, иначе любое замедление, "торможение" может сильно повлиять на течение игрового процесса. Стандартные средства Win32 не в силах обеспечить такую скорость, а DirectInput работает напрямую с аппаратным обеспечением компьютера в обход сервисов операционной системы, что дает максимально возможную скорость доступа в приложении. Компонент DirectInput тоже не обновлялся до девятой версии и представлен интерфейсом `IDirectInput8`. Корпорация Microsoft приняла решение не обновлять DirectInput, оттого что заложенного функционала еще вполне достаточно на несколько ближайших лет. Это и вполне логично, какие устройства ввода мы имеем на сегодняшний день? Клавиатура, мышь, руль с педалями, джойстик, шлем виртуальной реальности, еще ряд каких-то устройств с виброотдачей и обратной связью — вот, пожалуй, и весь список. Все перечисленные устройства имеют поддержку в `IDirectInput8`, поэтому обновлять этот компонент пока нет смысла, но с приходом новой платформы XNA, по всей видимости, концепция DirectInput может несколько изменится.
- *DirectMusic* и *DirectSound*, как видно из названий компонентов, отвечают за музыку и звук, делая возможным воспроизведение MIDI- и WAV-файлов. Два этих компонента, так же как и DirectInput, не обновлялись, и их текущая версия числится под номером восемь.
- *DirectShow* необходим для работы с видео- и аудиоданными. Имеется возможность воспроизведения следующих форматов:
  - MPEG (Motion Picture Experts Group);
  - MPEG Audio Layer-3 (MP3-формат);
  - AVI (Audio Video Interleaved).
- *DirectPlay* обеспечивает работу приложения с сетью и основан на своем протоколе. В DirectX 9.0 SDK Update (Summer 2004) появилась новая возможность работы по протоколу Bluetooth.
- *DirectSetup* позволяет произвести установку файлов DirectX на компьютер пользователя и отвечает за автоматический запуск диска с игрой или программой.

Каждый из перечисленных ранее компонентов предоставляет программисту инструменты для упрощения работы с графикой, звуком, видео и устройствами ввода. Для того чтобы воспользоваться в полной мере предлагаемыми средствами, необходимо подключить в создаваемое приложение заголовочные и библиотечные файлы.

Подсказка

Заголовочные файлы DirectX 9 — это, как правило, файлы с расширением h, подключаемые в приложение с помощью директивы `#include`. Например: `#include <d3d9.h>`. Библиотечные файлы DirectX 9 в программу можно подключить с помощью директивы `#pragma`, прописав эту строчку в исходном коде. Например: `#pragma comment( lib, "d3d9.lib")`. Либо явно добавить библиотеки с указанием пути к каталогу. В Visual C++ 6.0 это делается следующим образом: выберите в меню команды **Progect → Settings** и в появившемся диалоговом окне перейдите на вкладку **Link**. В строке **Object/library modules** пропишите подключаемую библиотеку, например: `d3d9.lib`. В среде программирования Visual C++ .NET для подключения библиотек необходимо во вкладке **Solution Explorer** инструментария Visual C++ .NET нажать правой кнопкой мыши на названии созданного проекта. В появившемся меню выделите поле **Properties**, откроется диалоговое окно. В левой части этого окна находится ряд папок, раскройте папку **Input** и в поле **Additional Dependencies** пропишите подключаемые библиотеки. Все библиотеки прописываются через пробел. После добавления библиотечных файлов нажмите кнопку **OK**.

Не забывайте производить операции по подключению заголовочных и библиотечных файлов DirectX 9 в каждое создаваемое приложение, иначе у вас возникнет множество необъяснимых ошибок во время компиляции исходных кодов.

Техника построения сцен

В DirectX 9 для представления графики используется компонент **Direct3D9**. Это очень мощный и, пожалуй, самый главный компонент, имеющий огромное количество интерфейсов для работы с вершинным и индексным буферами, текстурами, вершинными и пиксельными шейдерами и т. д. Direct3D9 позволяет программисту создавать трехмерную графику высокого качества, не затрачивая при этом особых усилий. Вместе с тем, Direct3D9 — сложный компонент, имеющий множество предопределенных функций, структур, макросов, типов, но достаточно разобраться один раз в принципе построения графики и пользоваться приобретенными навыками постоянно будет уже не так сложно.

Ключевыми понятиями в Direct3D9 являются рендеринг и сцена. *Сцена* в DirectX — это прямая аналогия любых театральных подмостков, на которых как в телевизоре вы видите статические и движущиеся объекты, совершающие различные действия. То есть все, что отображается на экране монитора — это, по сути, и есть сцена.

Процесс представления сцены на дисплее в DirectX носит название *рендеринг*, благодаря которому происходит визуализация или представление сцены на экране монитора. Механизм отображения сцены или рендеринг довольно сложный процесс, состоящий из ряда последовательных операций, напоминающий некий абстрактный конвейер, визуализирующий графику на экране. На рис. 1.1 показан механизм представления графики на экране монитора с помощью графического конвейера.

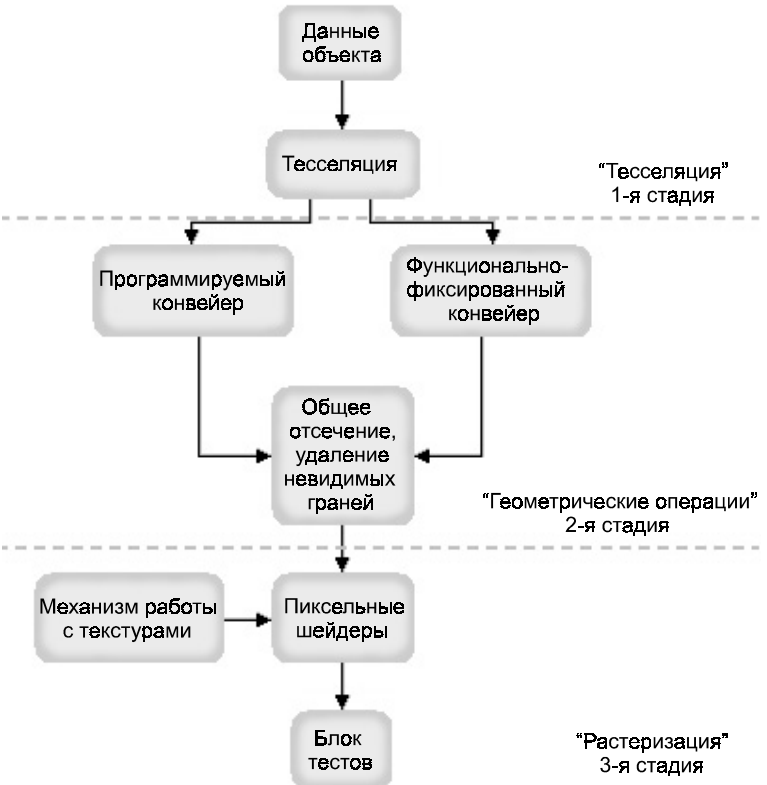


Рис. 1.1. Графический конвейер DirectX

Из рис. 1.1 видно, что *графический конвейер* разделен на три стадии: тесселяция, геометрические преобразования и растеризация. На *стадии тесселяции* происходит отбор граней объекта, т. е. разбиение модели на более мелкие части, такие как полигоны. Эта стадия выполняется программно.

Вторая *стадия геометрических преобразований* включает в себя большой комплекс по преобразованию вершин с помощью матриц, установке материала, расчета освещения объекта. Если вы еще раз посмотрите на рис. 1.1, то заметите, что на стадии геометрических преобразований имеется два пу-

ти. Это использование фиксированного и программируемого графического конвейера.

Фиксированный конвейер использует массу повторяющихся операций по преобразованию объекта, тогда как *программируемый конвейер* построен на работе с вершинными шейдерами. Более подробно работу обоих конвейеров мы рассмотрим в *главах 2* и *3*.

Третья стадия — это *стадия растеризации*, производящая представление объекта на пиксельном уровне. На этой стадии можно задействовать пиксельные шейдеры, о которых мы поговорим в *главе 3*.

После прохождения объектом всех трех стадий готовая модель помещается в буфер кадра и выводится на экран.

Представленная общая модель работы графического конвейера является довольно сложным процессом. Все действия по обработке объекта в конвейере происходят аппаратно и лишь на стадии тесселяции — программно. Как программисту, вам не придется следить конкретно за каждой стадией графического конвейера. Для того чтобы создать объект или построить сцену, а потом отобразить ее на экране монитора, достаточно выполнить ряд последовательных шагов. В итоге все ваши действия будут сводиться к созданию необходимых указателей на интерфейсы Direct3D и вызову соответствующих функций DirectX для построения и настройки сцены. В *главе 2* в общих чертах рассматриваются принципы построения сцены в DirectX с использованием фиксированного графического конвейера, а в *главе 3* объясняется суть работы программируемого конвейера.

Платформа XNA

Прежде чем вы перейдете к изучению материала последующих глав, хочется представить вам новшество для разработчиков программного обеспечения, которое готовится к выпуску компанией Microsoft — это *платформа XNA*. Она позиционируется корпорацией Microsoft как единая платформа для разработки программного обеспечения под Windows, XBox и мобильных телефонов, работающих на операционной системе Windows Mobile. По замыслу Microsoft платформа XNA должна создать общую среду, объединяющую в себе все лучшие наработки этой корпорации в игровой индустрии. Получив в свои руки такую среду, разработчики смогут значительно упростить и сократить сроки создания игр, которые окажутся одинаково работоспособными как на XBox, так и на компьютерных системах под управлением операционной системы Windows. Но самое главное это то, что программистам не придется тратить много времени на освоение новых технологий. Платформа XNA будет построена на инструментариях, применяемых программистами уже сейчас. В качестве основной среды программирования будет использоваться Visual Studio, по всей видимости, новой версии, выход

которой планируется в 2005 году. Что касается API, то платформа XNA будет базироваться на DirectX высокоуровневом языке программирования шейдеров (HLSL), PIX for Windows (о котором пойдет речь во второй части этой книги), Xaudio API и XACT. Конечно, вполне очевидно, что появятся обновления в том же DirectX, но это будут именно обновления, а не внедрение новой технологии. То есть все обновления будут опираться на базу, с которой работают большинство разработчиков всего мира, и переход к новой более мощной платформе не должен повлечь за собой новых затрат, как денежных, так и трудовых. XNA — это платформа нового поколения, вобравшая в себя все лучшее, что есть на сегодняшний день в игровой индустрии.

Прогнозировать какие-либо сроки выхода этой платформы еще рано — официальных заявлений от корпорации Microsoft пока не поступало. Процесс создания новой платформы даже на уже имеющейся базе весьма трудоемок, поэтому год-два, наверное, придется подождать. Скорее всего, выход новой операционной системы Windows, новой приставки XBox, новой среды программирования Visual Studio можно будет считать некой отправной точкой для платформы XNA. К слову сказать, новая версия DirectX 9.0 SDK Update (Summer 2004) уже содержит некоторые нововведения, связанные с платформой XNA, так что ждать осталось не долго.

Языки HLSL, GLSL и Cg

Нельзя сказать, что мощность компьютерных систем достигла предела, однако процессора с частотой 2500 МГц и памяти в 512 Мбайт уже вполне достаточно, чтобы играть в любую игру, смотреть фильмы, записывать и слушать музыку, не говоря уже о более простых задачах. В какой-то момент времени частота работы процессора просто перестала играть большую роль и совсем не экзотическая система с процессором 3000 МГц и 1024 Мбайт памяти на борту уже считается компьютером Hi-End класса. А при бурно развивающейся системе кредитования населения в нашей стране компьютерные системы с такими характеристиками в ближайшее время будут стоять в каждом доме. Но при этом пользователю хочется высококачественной графики, сравнимой с реальным миром. После просмотра очередного киношедевра с великолепными спецэффектами типа "Ночной дозор", человек включает компьютер и желает наблюдать и у себя нечто подобное. Но, запуская игру уровня Quake 3 с заметно квадратными полигонами, в глубине разочаровывается, возвращаясь из виртуальной реальности в невыразительную действительность. Поэтому сейчас основной задачей компьютерной индустрии развлечений является создание действительно мощных графических процессоров, и, в принципе, таковые уже имеются. Ведущие разработчики в этой области, компании NVIDIA и ATI, предлагают видеоадаптеры, способные справиться с отрисовкой скелетной анимации, реалистичной по-

верхности материала с большим количеством полигонов и источников света на основе работы с вершинными и пиксельными шейдерами.

Фиксированный графический конвейер, используемый видеоадаптерами вчерашнего дня, явно устарел, приведя отрасль в тупиковое состояние. Улучшение фиксированного конвейера или библиотечных функций DirectX и OpenGL ощутимых результатов не дает. Поэтому был разработан новый программируемый графический конвейер, работающий с вершинными и пиксельными шейдерами.

Общий принцип работы с шейдерами заключается в том, что программисту дается возможность работать с графическим процессором видеоадаптера напрямую. В DirectX такая возможность осуществляется с версии DirectX 8.1, а в OpenGL только с появлением нового языка GLSL (GL Shader Language, язык программирования шейдеров в OpenGL). Работая напрямую с графическим процессором посредством регистровых записей, с применением вершинных и пиксельных шейдеров, значительно улучшается качество и скорость визуализации графики. Гибкость, на основе которой построен программируемый конвейер при работе с преобразованиями и освещением, достигается за счет избавления от массы повторяющихся операций и выводит качество рисуемой графики на новый до этого времени не доступный уровень.

Работа с шейдерными программами в DirectX доступна и в восьмой и в девятой версии. Изначально, когда только разрабатывалась технология работы с шейдерами, был придуман и описан так называемый *ассемблерный язык программирования шейдеров*, основанный на построчном исполнении инструкций или команд. Так же как и в ассемблере, ассемблерный язык шейдеров за один машинный цикл выполняет одну команду, соответственно и написание программного кода шейдеров сводится для удобства восприятия к построчному перечислению команд. Такой подход к созданию шейдерных программ мог порадовать кого угодно, но только не программистов. Хорошо, если при использовании ассемблерного языка шейдеров пишется небольшая программка, а если это действительно серьезная компьютерная игра с длинным программным кодом и сотнями строчных инструкций шейдеров, тогда что? Да, безусловно, можно хорошо продумать структуру будущей программы и разбить ассемблерный код шейдеров на несколько файлов, но это ли решение проблемы?

Вот тут и была описана и создана концепция C-подобного языка программирования шейдеров: DirectX HLSL, OpenGL GLSL и язык Cg компании NVIDIA. Какой из них появился раньше, какой позже, спорить нет смысла, у всех трех языков примерно одинаковая направленность — это облегчение как написания, так и понимания шейдерных программ. Все три языка в чем-то сходны — это C-подобные языки с предопределенными типами данных, строящиеся на основе структурной реализации программы. При этом,

естественно, каждый язык обладает своим синтаксисом и семантикой, но повторюсь, некоторые сходства имеются. То есть если вы можете программировать на C++, то при желании разберетесь и с языком Java. Примерно также обстоят дела и с упомянутыми ранее языками программирования шейдеров. Все три языка призваны улучшить, облегчить и ускорить работу программиста.

Язык HLSL создан в недрах корпорации Microsoft и поддерживается всеми производителями аппаратного обеспечения, а факт распространения операционной системы Windows во всем мире, по разным оценкам достигающий 80—90%, делает этот язык программирования шейдеров безусловным лидером. Да и разработчики компьютерных игр под Windows понимают, что DirectX для этой операционной системы — стандарт, а значит, и язык HLSL тоже. Если рассматривать HLSL в ракурсе новой платформы XNA, то будущее языков GLSL и Cg кажется бесперспективным.

Язык программирования шейдеров GLSL, работающий только с OpenGL, при ближайшем рассмотрении кажется мощнее, чем HLSL, и даже его некоторая направленность на новое аппаратное обеспечение добавляет этому языку своих вистов. И то, что в свою очередь OpenGL — это фактически стандарт для другой операционной системы Linux, добавляет шансов языку GLSL не только на выживание, но и на благополучное процветание.

К слову сказать, операционная система Symbian, которой комплектуются большинство смартфонов и коммуникаторов, в своей седьмой версии поддерживает OpenGL ES — некую урезанную версию OpenGL для мобильных телефонов. Правда, и в этом сегменте рынка Microsoft имеет свою операционную систему — Windows Mobile, где снова присутствует DirectX. Конечно, телефонным системам думать о шейдерах пока еще рано, но широкая распространенность телефонов и мобильных игр свидетельствует о том, что все возможно и прогнозировать в этом направлении что-либо сложно.

Язык программирования Cg компании NVIDIA выглядит чуть похуже предшественников. Все-таки его узкая направленность на видеоадаптеры только от NVIDIA несколько ограничивает его распространение. Но, не умоляя достоинств Cg, можно сказать, что это действительно развитый и хорошо продуманный язык со своим "характером".

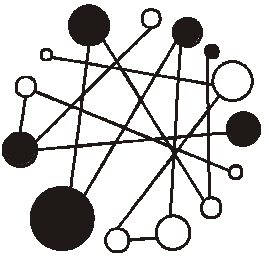
Какой из языков будет в будущем доминировать, предугадать сложно, а может, каждый из языков будет просто "ехать по своей колее". Конечно, в идеале программист может знать все эти языки и найти ему работу будет намного легче, хотя узконаправленный специалист — это лучший специалист. Остается надеяться, что компьютерные развлечения продолжат развиваться, а мы в связи с этим без работы не останемся.

Напоследок хочется сказать несколько слов об инструментальных средствах, предназначенных для программирования шейдеров. Сейчас на рынке имеются RenderMonkey и FX Composer — очень хорошие средства для работы

с шейдерами, но все же они направлены на просмотр и редакцию кода, на эксперименты с текстурами и материалом, т. е. скорее на улучшение программного кода, нежели на его написание. Безусловно, возможно и создание проекта с нуля. Эти средства позволяют как программисту, так и художнику просматривать результаты изменений в реальном времени, но все же, как бы хотелось иметь всего лишь один программный продукт, наделенный возможностями Visual C++ .NET, FX Composer, RenderMonkey и Pix for Windows! Одна программа, одно рабочее пространство, множество функций, динамический интерфейс, интерактивность в реальном времени, полное понимание HLSL, GLSL и Cg. Неважно даже какая компания сделает это и какое будет название. Мечты? К сожалению, скорее да, чем нет. Раз уж ветвь истории наделила нас шейдерами, наверное, стоит задуматься о таком программном продукте. Может корпорация Microsoft как раз и порадует программистов выходом платформы XNA, где будет предусмотрено нечто подобное на базе Visual Studio .NET под DirectX.

А пока, о лучшем, что есть на сегодняшний день, вы сможете узнать из этой книги, где разбираются общие вопросы, связанные с работой профайлера PIX for Windows, инструментариев RenderMonkey и FX Composer.

ГЛАВА 2



Фиксированный графический конвейер

Как уже упоминалось в предыдущей главе, для построения сцены в DirectX 9 необходимо выполнить последовательно шаг за шагом определенные действия. Набор этих операций можно разбить на несколько следующих этапов:

1. *Создание оконного приложения.* На этом этапе необходимо создать простое оконное приложение, в которое впоследствии можно встраивать исходный код создаваемой сцены. Оконное приложение создается на основе библиотечных функций Win32 API.
2. *Инициализация и настройка Direct3D9.* На этом этапе происходит инициализация Direct3D9, в том числе создание необходимых указателей на интерфейсы и создание устройства Direct3D9. Также необходимо произвести настройку частоты смены кадров и разрешение буфера глубины, т. е. задать необходимые параметры для представления графики на экране монитора.
3. *Создание сцены.* После инициализации и настройки Direct3D9 можно приступить к созданию или загрузке объектов и всей сцены в частности. При создании объектов используются вершинные и индексные буферы, где хранятся позиции точек, из которых состоит объект. Для загрузки готового объекта, созданного с помощью любого редактора трехмерной графики, в DirectX используются X-файлы. Конвертировав готовую модель в X-формат, вы сможете легко загрузить ее с помощью функций DirectX 9. Также есть возможность загрузки объектов и других форматов.
4. *Трансформация и освещение (T&L).* Этот этап, пожалуй, самый сложный и именно на этой стадии вы можете задействовать фиксированный либо программируемый графический конвейер. Для представления объекта на экране применяются матричные преобразования, а для того чтобы объект выглядел реалистично, используются материал и освещение.

5. *Рендеринг сцены*. После инициализации, настройки Direct3D и построения сцены можно производить вывод графики на экран монитора.

Все рассмотренные этапы создания и отрисовки трехмерной графики не включали в себя использование клавиатуры, мыши и звука. Для того чтобы задействовать перечисленные компоненты, необходимо выполнить еще ряд дополнительных действий. Эти процессы в книге не рассматриваются, но при необходимости вы сможете найти нужную информацию в книге "DirectX 9. Уроки программирования на C++" издательства "БХВ-Петербург".

Двойная буферизация

Смысл *двойной буферизации* чрезвычайно прост. Для вывода графики на экран используется *задний буфер* (back buffer), в который помещается очередной кадр. Задний буфер по своим параметрам, разрешению и цветности идентичен первичной поверхности экрана, и за счет постоянной смены буферов или переключения поверхностей достигается плавная анимация в графике. Например, вы вывели на экран некую модель и желаете переместить ее в другой конец экрана. Для этого необходимо стереть модель в месте ее первоначального вывода и нарисовать заново. Если производить эти действия на первичной поверхности, анимация может происходить с заметными глазу рывками. Куда проще нарисовать модель на новом месте в заднем буфере и произвести смену буферов. Этот процесс вначале может показаться сложным, но на самом деле программисту не приходится отслеживать задний буфер, эти действия выполняются программно средствами DirectX. Просто в определенных параметрах представления устройства выставляются необходимые флаги и количество создаваемых задних буферов, после чего процесс двойной буферизации будет осуществляться автоматически под контролем DirectX.

Теперь давайте рассмотрим подробнее стадии по созданию и отрисовке сцены на экране монитора.

Создание оконного приложения

Операционная система Windows работает на основе управляемых событий, поэтому, создавая *оконное приложение*, вы также должны позаботиться о создании обработчика событий. Создавая оконное приложение, вы вправе определить стиль, размер, цвет фона и режимы отображения оконного приложения. Разбор исходного кода, иллюстрирующего создание оконного приложения, к сожалению, выходит за рамки этой книги, но в конце данной главы и на компакт-диске к книге в папке \Code\Demo в файле Demo.cpp вы можете ознакомиться с кодом примера, создающим оконное

приложение, а также полным спектром действий по инициализации, настройке, созданию объекта и рендерингу с использованием Direct3D9. Исходный код несложен, и я думаю, вам не составит труда разобраться во всем самостоятельно.

Инициализация и настройка Direct3D9

Процесс *инициализации* и *настройки Direct3D9* сводится к последовательному вызову библиотечных функций и заполнению полей структуры параметров представления Direct3D9. Но первым делом необходимо создать *указатель* на главный интерфейс `IDirect3D9`.

```
LPDIRECT3D9 pd3d9 = NULL;
pd3d9 = Direct3DCreate9( D3D_SDK_VERSION);
```

После объявления переменной `pd3d9` (предварительно обнулив ее значение) нужно воспользоваться функцией `Direct3DCreate()` для создания указателя на интерфейс `IDirect3D9`, который будет находиться в переменной `pd3d9`. Функция `Direct3DCreate()` в качестве параметра всегда использует макрос `D3D_SDK_VERSION`, указывающий на текущую версию DirectX SDK. Затем необходимо определить формат поверхности дисплея или текущий режим визуального отображения дисплея. Для этого нужно воспользоваться функцией `IDirect3D9::GetAdapterDisplayMode` и структурой `DISPLAYMODE`.

```
D3DDISPLAYMODE Display;
pd3d9->GetAdapterDisplayMode(D3DADAPTER_DEFAULT, &Display);
```

Функция `GetAdapterDisplayMode()` возвращает установленный текущий формат поверхности дисплея, сохраняя полученный результат в переменной `Display`, а именно: разрешение экрана (т. е. ширину и высоту), цветность и частоту обновления экрана. Все полученные данные необходимы для создания заднего буфера, который должен быть идентичен исходной поверхности дисплея.

Примечание

При работе с библиотечными функциями DirectX 9 необходимо использовать отлаженную систему обработки ошибок. Для этого в DirectX предусмотрены два макроса: `FAILED` — проверяющий программный код на наличие ошибок и `SUCCEEDED` — проверяющий программный код на успешность выполнения. Оба макроса имеют соответствующие коды возврата ошибок, о которых можно узнать из справочной информации DirectX SDK. Например, для того чтобы определить текущий формат дисплея, можно воспользоваться следующей конструкцией кода:

```
if(FAILED(pd3d9->GetAdapterDisplayMode(D3DADAPTER_DEFAULT,
   ↳dis)))return E_FAIL;
```

Следующим шагом в процессе настройки Direct3D9 будет установка параметров представления Direct3D9 с помощью структуры `D3DPRESENT_PARAMETERS`. Это очень большая структура, с ее помощью создаются задний буфер, буфер глубины и определяется полноэкранный или оконный режим работы приложения.

```
D3DPRESENT_PARAMETERS d3d9pp;
// обнуление
ZeroMemory(&d3d9pp, sizeof(d3d9pp));
// задействуется полноэкранный режим
d3d9pp.Windowed = FALSE;
// подключается задний буфер
d3d9pp.SwapEffect = D3DSWAPEFFECT_DISCARD;
// формат поверхности заднего буфера
d3d9pp.BackBufferFormat = Display.Format;
// создается буфер глубины
d3d9pp.EnableAutoDepthStencil = TRUE;
// формат поверхности буфера глубины
d3d9pp.AutoDepthStencilFormat = D3DFMT_D16;
// ширина заднего буфера
d3d9pp.BackBufferWidth  = Display.Width;
// высота заднего буфера
d3d9pp.BackBufferHeight = Display.Height;
// количество задних буферов
d3d9pp.BackBufferCount = 2;
// частота обновления
d3d9pp.FullScreen_RefreshRateInHz = Display.RefreshRate;
```

Приведенный исходный код имеет комментарии, и в нем будет нетрудно разобраться. Но хочется добавить замечание по поводу заднего буфера и буфера глубины. Как вы уже знаете, задний буфер необходим для создания качественной анимации в компьютерных играх. В параметре

```
d3d9pp.BackBufferCount = 2;
```

было создано два задних буфера, но их число может быть и больше, главное чтобы ваш видеоадаптер поддерживал указанное количество задних буферов. Два-три задних буфера — это вполне стандартное значение.

А для чего же нужен *буфер глубины*? Экран монитора по своей сути представляет собой абсолютно плоскую двухмерную поверхность, и для того чтобы правильно отображать трехмерные модели, применяется буфер глубины, который иногда еще называют *z-буфер*. То есть это дополнительная ось Z, необходимая для правильного представления объемной трехмерной графики на экране монитора.

После создания z-буфера нужно созданный буфер подключить в приложение, а также задействовать режим отсечения невидимых граней объекта.

```
pDevice->SetRenderState(D3DRS_CULLMODE, D3DCULL_CW);
pDevice->SetRenderState(D3DRS_ZENABLE, D3DZB_TRUE);
```

Подключение опции *отсечения невидимых граней* объекта — это общепринятая практика в DirectX. Например, вы рисуете на экране куб, вращающийся вокруг своей оси. В какой-то промежуток времени одна из частей куба обязательно закроет какую-то другую часть куба. И поскольку вы все равно не увидите эту закрытую часть, то включается режим отсечения невидимых граней. Зачем же тратить ресурсы процессора и памяти на те части объекта, которые не видно. Сам процесс по отсечению происходит автоматически, главное только правильно построить сам объект.

Последний шаг в настройке и инициализации Direct3D9 заключается в создании устройства Direct3D9 с помощью функции `IDirect3DDevice9::CreateDevice`.

```
pd3d9 -> CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hwnd,
D3DCREATE_HARDWARE_VERTEXPROCESSING, &d3d9pp, &pDevice);
```

Инициализировав и настроив Direct3D9, можно переходить к этапу по созданию сцены.

Создание сцены

Построение *сцены* в Direct3D — это процедура трудоемкая и всецело зависит от задачи, которую вам необходимо решить. На этом этапе возможна загрузка готовых трехмерных моделей, созданных, например, в 3D Studio Max, или целого уровня, созданного в том же редакторе, или построение объекта сервисами Direct3D, которые мы и рассмотрим.

В качестве примера возьмем программу Demo, код которой находится в конце главы. В этом примере происходит построение куба на основе индексного и вершинного буфера. Концепция построения объектов в DirectX использует понятие *вершина*. Каждый объект, каким бы он не был сложным, состоит из полигонов. *Полигон* — это определенного размера площадь в пространстве, ограниченная точками. В Direct3D обычно используется треугольник, площадь которого ограничивается тремя углами. Каждый угол имеет свои координаты в пространстве по осям X, Y и Z. Точка, заданная в пространстве по трем осям X, Y и Z, в DirectX называется *вершиной*. Определив значения для трех вершин, можно построить простой треугольник, а соединив два треугольника по гипотенузе, получаем квадрат. Чем больше полигонов, тем сложнее фигура для построения модели.

В DirectX могут применяться *системы координат* двух видов, в зависимости от установленного формата вершин. Определение формата вершин зависит

от того, будет ли рисоваться двухмерная или трехмерная графика. При использовании двухмерных объектов применяется система координат, изображенная на рис. 2.1, и в этом случае при построении объекта используется *преобразованный формат вершин*.

Рис. 2.1. Двухмерная система координат

При создании трехмерного объекта применяется более сложная модель работы с вершинами. Вершины оставляют *не преобразованными*, а для визуализации вершин на экране монитора используются матричные преобразования. Поэтому применяется левосторонняя система координат, показанная на рис. 2.2.

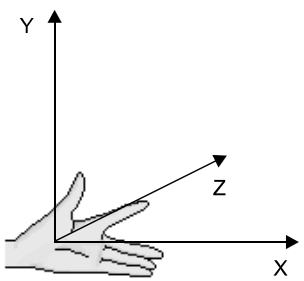


Рис. 2.2. Левосторонняя система координат

Но обо всем по порядку. Создаваемый объект состоит из набора полигонов, задающихся вершинами. Позиция одной вершины задается тремя координатами по осям X, Y и Z. Для построения квадрата необходимо два соприкасающихся гипотенузами треугольника. Для создания двух треугольников потребуется шесть вершин, а для того чтобы построить куб, понадобится

6 вершин * 6 сторон куба = 36 вершин. Для хранения такого количества вершин логично использовать структуру, например:

```
struct CUSTOMVERTEX
{
    FLOAT x,  y,  z;
    FLOAT nx, ny, nz;
};
```

Координаты `x`, `y` и `z` определяют позицию вершины в пространстве, а три других координаты определяют направление нормалей, необходимых для правильного расчета освещения сцены, о котором вы узнаете в этой главе из *разд. "Трансформация и освещение"*.

Одна сторона куба — это два соприкасающихся треугольника. Для построения одной стороны, т. е. квадрата, необходимо задать позиции для шести вершин, по три на каждый треугольник. При детальном рассмотрении выясняется, что две вершины каждого треугольника будут иметь одинаковые позиции. Поэтому можно значительно упростить процесс построения квадрата, и куба в частности, путем *индексации* каждой вершины, а полученные значения хранить в отдельном массиве данных. В конце главы и на компакт-диске в папке \Code\Demo\Demo.cpp вы можете ознакомиться с кодом, создающим индексированный куб с помощью структуры `Vertex` и массива индексов `Index`.

После того, как были заданы позиции для всех вершин, необходимо указать используемый формат вершин. Для этого применяется следующая конструкция программного кода:

```
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ|D3DFVF_NORMAL);
```

Макрос `D3DFVF_XYZ` указывает на не преобразованный формат вершин, поскольку все преобразования будут осуществляться с помощью матриц. Если вы желаете использовать формат преобразованных вершин, то нужно воспользоваться макросом `D3DFVF_XYZRHW`.

Последующие действия по построению куба сводятся к использованию индексного буфера и буфера вершин. Буфер вершин необходим для хранения общего количества вершин объекта, можно сказать, что это заданный массив данных, в котором содержатся сами вершины, а также цвет каждой вершины и нормали. Впоследствии при рендеринге буфер вершин используется для вывода объекта на экран. Индексный буфер содержит индексы для каждой стороны куба и также используется при рендеринге. Процесс работы с буфером вершин и индексным буфером сводится к нескольким последовательным шагам — это создание буфера, его блокировка для последующей записи данных и разблокировка буфера. В следующих строках кода показаны перечисленные операции для буфера вершин:

```
LPDIRECT3DVERTEXBUFFER9 pBV = NULL;
// создание буфера
pDevice->CreateVertexBuffer(36 * sizeof(CUSTOMVERTEX),
        0, D3DFVF_CUSTOMVERTEX,D3DPOOL_DEFAULT, &pBV, NULL);
VOID* copy;
// блокировка буфера
pBV->Lock(0, sizeof(Vertex), (void**)&copy, 0);
// запись данных в буфер
memcpy(copy, Vertex, sizeof(Vertex));
// разблокировка буфера
pBV->Unlock();
```

Для записи данных в буфер вершин необходимо обязательно заблокировать его, а каждая блокировка буфера должна сопровождаться его разблокировкой. Действия по работе с индексным буфером практически идентичны действиям с буфером вершин.

```
LPDIRECT3DINDEXBUFFER9 pBI = NULL;
// создание буфера
pDevice->CreateIndexBuffer(36 * sizeof(Index),
      0, D3DFMT_INDEX16, D3DPOOL_DEFAULT,&pBI, NULL);
// блокировка буфера
pBI->Lock(0, sizeof(Index), (void**)&copy, 0);
// запись данных в буфер
memcpy(copy, Index, sizeof(Index));
// разблокировка буфера
pBI->Unlock();
```

Таким образом, после создания объекта можно перейти к следующему этапу.

Трансформация и освещение

На этом этапе можно выбрать работу с фиксированным или программируемым графическим конвейером. Программируемый конвейер использует вершинные шейдеры и будет обсуждаться в следующей главе. Основная база, на которой основана работа с фиксируемым и программируемым графическими конвейерами, построена на использовании матричных преобразований и работе с материалом и освещением.

Матричные преобразования

Для представления трехмерной сцены на экране монитора в DirectX задействуются матричные преобразования. Матричные операции основаны на использовании *матриц* — это мировая матрица (World Matrix), матрица вида

(View Matrix) и матрица проекции (Projection Matrix). Все три матрицы имеют вид массивов данных с размерностью 4×4.

Для того чтобы правильно отобразить объект на экране, необходимо преобразовать каждую вершину с помощью последовательного использования трех матриц в следующем порядке: мировая матрица, матрица вида и матрица проекции. Создание каждой из матриц можно выполнить вручную, а можно воспользоваться достаточно большим набором функций из библиотеки утилит D3DX, входящей в состав Direct3D9.

> ***Примечание***
>
> Для того чтобы воспользоваться библиотекой D3DX, необходимо подключить заголовочный файл d3dx9.h и библиотечный файл d3dx9.lib в проект.

Мировая матрица

Мировая матрица — это первая матрица, на которую нужно умножить каждую вершину объекта. С помощью мировой матрицы производится мировое преобразование объекта, и в результате каждый из объектов получает свою локальную систему координат. На основании локальной системы координат происходит построение объекта. Также при использовании мирового преобразования доступны действия по вращению, трансляции и масштабированию объекта. Например, для того чтобы преобразовать объект в мировое пространство и осуществить вращение по осям X и Y, можно воспользоваться следующим программным кодом.

```
D3DXMATRIX MatrixWorld, MatrixWorldX, MatrixWorldY;
FLOAT Angel = (timeGetTime() % 3000) * (2.0f * D3DX_PI) / 3000.0f;
D3DXMatrixRotationX(&MatrixWorldX, Angel);
D3DXMatrixRotationY(&MatrixWorldY, Angel);
D3DXMatrixMultiply(&MatrixWorld,&MatrixWorldX,&MatrixWorldY);
```

С помощью функций `D3DXMatrixRotationX()` и `D3DXMatrixRotationY()` производится вращение объекта по осям X и Y на угол, равный значению в переменной `Angel`. Полученный результат вращения по оси X сохраняется в переменной `MatrixWorldX`, а по оси Y — в переменной `MatrixWorldY`. После этого два результата вращения по разным осям перемножаются между собой с помощью функции `D3DXMatrixMultiply()`, а итоговый результат помещается в переменную `MatrixWorld`. Для того чтобы мировое преобразование вступило в силу, необходимо воспользоваться функцией `IDirect3DDevice9::SetTransform`.

```
pDevice->SetTransform(D3DTS_WORLD,&MatrixWorld);
```

Матрица вида

С помощью *матрицы вида* происходит определение позиции просмотра сцены. Например, за позицию просмотра сцены можно взять местонахож-

дение ваших глаз в пространстве, а поскольку ваш взгляд может быть устремлен в любую точку экрана монитора, то должно определяться направление взгляда и середина области просмотра, т. е. то, на что вы смотрите в данный момент. Для преобразования объекта матрицей вида нужно воспользоваться функцией `D3DXMatrixLookAtLH()` из библиотеки утилит D3DX.

```
D3DXMATRIX MatrixView;
D3DXMatrixLookAtLH(&MatrixView, &D3DXVECTOR3 (0.0f,0.0f,-11.0f),
&D3DXVECTOR3 (0.0f, 0.0f, 0.0f), &D3DXVECTOR3 (0.0f, 1.0f, 0.0f));
```

Первый параметр `MatrixView` в функции `D3DXMatrixLookAtLH()` — это результат видовых преобразований. Три последующих параметра определяют соответственно точку просмотра сцены, на что конкретно направлен взгляд и что в итоге мы видим. Для определения позиции просмотра используется структура `D3DXVECTOR3`, представляющая точку в пространстве по осям X, Y и Z.

Для того чтобы видовые преобразования вступили в силу, нужно воспользоваться функцией `SetTransform()`.

```
pDevice->SetTransform(D3DTS_VIEW, &MatrixView);
```

Матрица проекции

После преобразования объекта мировой матрицей и матрицей вида производится последний этап преобразований с помощью *матрицы проекции*. Проекционные преобразования позволяют правильно представить объект на двухмерной плоскости монитора, масштабируя объект относительно позиции просмотра, передней и задней области отсечения.

```
D3DXMATRIX MatrixProjection;
D3DXMatrixPerspectiveFovLH(&MatrixProjection,D3DX_PI/4,1.0f,1.0f,100.0f);
```

Первый параметр в функции `D3DXMatrixPerspectiveFovLH()` является результатом преобразований. Следующий параметр определяет поле зрения по оси Y, обычно это значение равно `D3DX_PI/4`. Третий параметр — коэффициент сжатия для соотношения геометрических размеров. Два последних параметра задают передний и задний планы отсечения сцены. Чтобы назначенные преобразования вступили в силу, вызывается функция `SetTransform()`.

```
pDevice->SetTransform(D3DTS_PROJECTION, &MatrixProjection);
```

После матричных преобразований можно воспользоваться освещением объекта для реалистичной отрисовки его на экране.

Освещение

Применение освещения в построении трехмерной сцены значительно улучшает качество рисуемой графики. В реальной жизни нас окружают множе-

ство разных предметов, каждый из которых имеет свой цвет, а точнее поверхность с определенным цветовым свойством. При попадании лучей света на поверхность предмета происходит гармоничное сочетание свойств света и свойств поверхности предмета, что в итоге и предопределяет наши цветовые и световые ощущения, полученные от этого предмета. В Direct3D поверхность объекта с цветовыми свойствами называется *материалом*. При работе с освещением необходимо в обязательном порядке задавать свойства материала для объекта, иначе построение сцены с использованием источников света не будет иметь никакого эффекта.

Материал

Материал в Direct3D9 создается на основе структуры `D3DMATERIAL9`, имеющей пять параметров, задавая значения которых можно определить необходимые свойства материала объекта.

Параметры структуры `D3DMATERIAL9` определяют:

- `Diffuse` — диффузный цвет материала;
- `Ambient` — окружающий цвет;
- `Specular` — отражающий цвет;
- `Emissive` — эмиссионный цвет;
- `Power` — мощность отражаемого цвета.

Четыре вида цвета `Diffuse`, `Ambient`, `Specular` и `Emissive` задаются с помощью структуры `D3DCOLORVALUE`, которая содержит ARGB составляющую цвета. Задавать значения красного, синего, зеленого цветов и альфа канала можно в пределах от 0.0 до 1.0. Более темные тона можно получить, если использовать значение меньше 0.0, и наоборот, светлые тона получаются при использовании значения больше 1.0. В следующих строках кода создается материал желтого цвета.

```
D3DMATERIAL9 Material;
ZeroMemory(&Material, sizeof(D3DMATERIAL9));
Material.Diffuse.r = Material.Ambient.r = 1.0f;
Material.Diffuse.g = Material.Ambient.g = 1.0f;
Material.Diffuse.b = Material.Ambient.b = 0.0f;
Material.Diffuse.a = Material.Ambient.a = 1.0f;
pDevice->SetMaterial(&Material);
```

В последней строке кода используется функция `SetMaterial()` для установки назначенных свойств материала. После определения материала можно приступить к созданию источника света.

Свет

Как и в окружающем нас мире, в DirectX тоже предусмотрены источники света. Имеются три вида *источников света*:

- параллельный или направленный (directional) — этот тип не имеет определенного источника, но светит в определенном направлении;
- точечный (point-source) — имеет заданный источник света и светит во все направления;
- прожекторный (spotlight) — направленный источник света конусообразной формы, такой как, например, фонарик или прожектор.

Свет, излучаемый источником, имеет свои цветовые свойства, определяющие цвет излучаемого света. Свойства света задаются с помощью структуры `D3DLIGHT9`, например, следующим образом:

```
D3DLIGHT9 Light;
ZeroMemory(&Light, sizeof(D3DLIGHT9));
Light.Type = D3DLIGHT_DIRECTIONAL;
Light.Diffuse.r  = 1.0f;
Light.Diffuse.g  = 1.0f;
Light.Diffuse.b  = 1.0f;
Light.Range      = 1000.0f;
```

В первой строке программного кода задается значение для параметра `Type` структуры `D3DLIGHT9`. Этот параметр — член структуры `D3DLIGHTTYPE`, на основе которой происходит определение типа источника света. В конкретном случае используется макрос `D3DLIGHT_DIRECTIONAL`, назначающий параллельный или направленный источник света. Существуют еще два макроса для назначения источника света, это:

- `D3DLIGHT_POINT` — точечный источник света;
- `D3DLIGHT_SPOT` — прожекторный источник света.

В следующих трех строках исходного кода устанавливаются свойства цвета для источника излучения — это диффузный цвет. И в последней строке кода устанавливается дальность излучения источника света.

Для того чтобы свет начал работать в освещении сцены, необходимо назначить созданный источник света для устройства Direct3D, разрешить работу с освещением и установить дополнительные свойства света.

```
pDevice->SetLight(0, &Light);
pDevice->LightEnable(0, TRUE);
pDevice->SetRenderState(D3DRS_LIGHTING, TRUE);
pDevice->SetRenderState(D3DRS_AMBIENT, 0);
```

Нормализация

Последний этап в процессе освещения — это нормализация вершин рисуемого объекта. *Нормаль* — это вектор, расположенный перпендикулярно стороне или вершине примитива. Нормаль принимает непосредственное участие в расчете освещения сцены. Каждая вершина объекта должна быть нормализована по отношению к стороне примитива. Например, передняя часть куба в нашем примере имеет четыре вершины. Все вершины нормализованы, т. е. каждой вершине назначен вектор или нормаль, расположенная перпендикулярно стороне куба. Все последующие стороны куба должны быть нормализованы таким же образом. Для этого в структуре `CUSTOMVERTEX` и задаются нормали.

```
struct CUSTOMVERTEX
{
    FLOAT x,  y,  z;
    FLOAT nx, ny, nz; // нормали
};
```

Определяя формат вершин, необходимо задать и нормализацию.

```
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ|D3DFVF_NORMAL);
```

Значение нормали в Direct3D определяется единичным значением, поэтому в структуре `Vertex` назначается единица для всех нормалей вершин. В конце этой главы найдите функцию `TL()`, структуру `Vertex` и посмотрите, каким образом происходит нормализация сторон куба в проекте Demo.

Для того чтобы нормали участвовали в расчете освещения, необходимо их установить для используемого источника света.

```
D3DXVECTOR3 VectorDir;
VectorDir = D3DXVECTOR3(0.0f, 0.0f,1.0f),
D3DXVec3Normalize((D3DXVECTOR3*)&Light.Direction, &VectorDir);
```

На этом этап по трансформации и освещению закончен, можно переходить непосредственно к выводу объекта на экран монитора.

Рендеринг сцены

Операции по рендерингу сцены лучше разместить в одной функции. Перед тем как производить прорисовку сцены, необходимо очистить задний буфер цветовой составляющей, а поскольку функция рендеринга впоследствии будет помещена в цикл Windows, то очистка заднего буфера цветом будет осуществляться циклично. Для очистки заднего буфера цветом воспользуемся функцией `IDirect3DDevice9::Clear`.

```
pDevice->Clear( 0, NULL, D3DCLEAR_TARGET| D3DCLEAR_ZBUFFER,
D3DCOLOR_XRGB(60,100,150), 1.0f, 0);
```

После очистки заднего буфера можно приступать к прорисовке всей сцены. Для этого Direct3D дается соответствующая команда:

```
pDevice->BeginScene();
```

Это, как вы догадались по названию функции `BeginScene()`, начало сцены для визуализации объекта. Далее связываем поток данных с буфером вершин:

```
pDevice->SetStreamSource(0, pBV, 0, sizeof(CUSTOMVERTEX));
```

Задаем формат вершин:

```
pDevice->SetFVF(D3DFVF_CUSTOMVERTEX);
```

Производим установку индексов из буфера индексов для каждой вершины:

```
pDevice->SetIndices(pBI);
```

Затем осуществляем сборку куба:

```
pDevice->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 36, 0, 12);
```

И даем команду на окончание сцены:

```
pDevice->EndScene();
```

На каждый вызов функции начала сцены `BeginScene()` должен следовать вызов функции окончания сцены `EndScene()`. Между двумя этими функциями происходит построение сцены в Direct3D. Для представления сцены на экране используется функция `IDirect3DDevice9::Present`.

```
pDevice->Present(NULL, NULL, NULL, NULL);
```

И в конце, необходимо поместить функции по инициализации Direct3D, созданию объекта и рендеринга в цикл Windows.

```
if(SUCCEEDED(Initial(hwnd)))
{
   if(SUCCEEDED(Create()))
   {
      ShowWindow(hwnd, SW_SHOWDEFAULT);
      UpdateWindow(hwnd);
          ZeroMemory(&msg, sizeof(msg));
          while(msg.message!=WM_QUIT)
          {
             if(PeekMessage(&msg, NULL,0,0,PM_REMOVE))
             {
                TranslateMessage( &msg);
                DispatchMessage( &msg);
             }
```

```
            else
               Rendering();
         }
   }
}
```

На этом этап по созданию трехмерной сцены закончен, можно откомпилировать пример и посмотреть, что в итоге получилось. Единственное о чем еще необходимо помнить — это об *освобождении ресурсов*, задействованных Direct3D в процессе построения сцены. Когда вы отключаете работу программы, то необходимо проследить за выгрузкой COM-объектов из памяти. В рассмотренном примере были задействованы буфер вершин, индексный буфер, устройство Direct3D и главный объект Direct3D. Необходимо выгрузить все задействованные COM-объекты в порядке убывания. Освобождение захваченных ресурсов будет выглядеть следующим образом:

```
if(pBI  != NULL)
   pBI->Release();
if(pBV  != NULL)
   pBV->Release();
if(pDevice != NULL)
   pDevice->Release();
if(pd3d9 != NULL)
   pd3d9->Release();
```

В листинге 2.1 находится исходный код примера, рассмотренного в этой главе.

Листинг 2.1. Иллюстрация работы фиксированного графического конвейера

```
//***************************************************************************
// Demo.cpp
// Пример, иллюстрирующий работу фиксированного графического конвейера
//***************************************************************************
#include <windows.h>          // подключаем заголовочный файл Windows
#include <d3d9.h>             // подключаем заголовочный файл DirectX 9
#include <d3dx9.h>            // подключаем библиотеку D3DX
#include <mmsystem.h>         // подключаем системную библиотеку
//***************************************************************************
// глобальные переменные
//***************************************************************************
LPDIRECT3D9                 pd3d9    = NULL;  // главный объект Direct3D
LPDIRECT3DDEVICE9           pDevice  = NULL;  // устройство Direct3D
LPDIRECT3DVERTEXBUFFER9     pBV      = NULL;  // буфер вершин
LPDIRECT3DINDEXBUFFER9      pBI      = NULL;  // индексный буфер
```

```
//
struct CUSTOMVERTEX
{
    FLOAT x,  y,  z;       // координаты
    FLOAT nx, ny, nz;      // нормали
};
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ|D3DFVF_NORMAL)  // формат вершин
//*************************************************************************
// функция Initial()
// стадия инициализации и настройки Direct3D
//*************************************************************************
HRESULT Initial( HWND hwnd)
{
     if( NULL == ( pd3d9 = Direct3DCreate9( D3D_SDK_VERSION)))
        return E_FAIL;

    D3DDISPLAYMODE Display;
    if( FAILED( pd3d9->GetAdapterDisplayMode( D3DADAPTER_DEFAULT,
       ↳&Display)))
        return E_FAIL;
    D3DPRESENT_PARAMETERS d3d9pp;
    ZeroMemory( &d3d9pp, sizeof(d3d9pp));
    d3d9pp.Windowed = FALSE;
    d3d9pp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3d9pp.BackBufferFormat = Display.Format;
    d3d9pp.EnableAutoDepthStencil = TRUE;
    d3d9pp.AutoDepthStencilFormat = D3DFMT_D16;
    d3d9pp.BackBufferWidth  = Display.Width;
    d3d9pp.BackBufferHeight = Display.Height;
    d3d9pp.BackBufferCount = 2;
    d3d9pp.FullScreen_RefreshRateInHz = Display.RefreshRate;
    if( FAILED( pd3d9 -> CreateDevice( D3DADAPTER_DEFAULT,
       ↳D3DDEVTYPE_HAL, hwnd, D3DCREATE_HARDWARE_VERTEXPROCESSING,
       ↳&d3d9pp, &pDevice)))
        return E_FAIL;
   // включаем отсечение Direct3D
   pDevice->SetRenderState( D3DRS_CULLMODE, D3DCULL_CW);
   // подключаем z-буфер
   pDevice->SetRenderState (D3DRS_ZENABLE, D3DZB_TRUE);
 return S_OK;
}
```

```
//-----------------------------------------------------------------------
// функция Create()
// стадия создания сцены
//-----------------------------------------------------------------------
HRESULT Create()
{
   CUSTOMVERTEX Vertex[] =
    {

       {  1.0f,-1.0f,-1.0f, 0.0f, 0.0f,-1.0f,}, //A
       {  1.0f, 1.0f,-1.0f, 0.0f, 0.0f,-1.0f,}, //B
       { -1.0f, 1.0f,-1.0f, 0.0f, 0.0f,-1.0f,}, //C
       { -1.0f,-1.0f,-1.0f, 0.0f, 0.0f,-1.0f,}, //D

       { -1.0f,-1.0f,-1.0f,-1.0f, 0.0f, 0.0f,}, //A2
       { -1.0f, 1.0f,-1.0f,-1.0f, 0.0f, 0.0f,}, //B2
       { -1.0f, 1.0f, 1.0f,-1.0f, 0.0f, 0.0f,}, //C2
       { -1.0f,-1.0f, 1.0f,-1.0f, 0.0f, 0.0f,}, //D2

       { -1.0f,-1.0f, 1.0f, 0.0f, 0.0f, 1.0f,}, //A3
       { -1.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f,}, //B3
       {  1.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f,}, //C3
       {  1.0f,-1.0f, 1.0f, 0.0f, 0.0f, 1.0f,}, //D3

       {  1.0f,-1.0f, 1.0f, 1.0f, 0.0f, 0.0f,}, //A4
       {  1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f,}, //B4
       {  1.0f, 1.0f,-1.0f, 1.0f, 0.0f, 0.0f,}, //C4
       {  1.0f,-1.0f,-1.0f, 1.0f, 0.0f, 0.0f,}, //D4

       {  1.0f,-1.0f,-1.0f, 0.0f,-1.0f, 0.0f,}, //A5
       { -1.0f,-1.0f,-1.0f, 0.0f,-1.0f, 0.0f,}, //B5
       { -1.0f,-1.0f, 1.0f, 0.0f,-1.0f, 0.0f,}, //C5
       {  1.0f,-1.0f, 1.0f, 0.0f,-1.0f, 0.0f,}, //D5

       {  1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 0.0f,}, //A6
       { -1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 0.0f,}, //B6
       { -1.0f, 1.0f,-1.0f, 0.0f, 1.0f, 0.0f,}, //C6
       {  1.0f, 1.0f,-1.0f, 0.0f, 1.0f, 0.0f,}, //D6

    };
    const unsigned short Index[]={
    0,1,2,      2,3,0,
    4,5,6,      6,7,4,
    8,9,10,     10,11,8,
```

```
    12,13,14,   14,15,12,
    16,17,18,   18,19,16,
    20,21,22,   22,23,20,
    };
    // создаем буфер вершин
    if( FAILED( pDevice->CreateVertexBuffer( 36 * sizeof(CUSTOMVERTEX),
        ↳0, D3DFVF_CUSTOMVERTEX,D3DPOOL_DEFAULT, &pBV, NULL)))
        return E_FAIL;
    // блокируем буфер вершин
    VOID* copy;
    if( FAILED( pBV->Lock( 0, sizeof(Vertex), (void**)&copy, 0)))
        return E_FAIL;
    // копируем вершины
    memcpy( copy, Vertex, sizeof(Vertex));
    // разблокируем буфер вершин
    pBV->Unlock();
    // создаем индексный буфер
    pDevice->CreateIndexBuffer( 36 * sizeof(Index),
       0, D3DFMT_INDEX16, D3DPOOL_DEFAULT,&pBI, NULL);
    // блокируем индексный буфер
    pBI->Lock( 0, sizeof(Index), (void**)&copy, 0);
    // копируем позиции вершин
    memcpy( copy, Index, sizeof(Index));
    // разблокируем индексный буфер
    pBI->Unlock();
  return S_OK;
}
//-----------------------------------------------------------------------
// TL()
// стадия трансформации и освещения (T&L)
//-----------------------------------------------------------------------
VOID TL()
{
    D3DXMATRIX MatrixWorld, MatrixWorldX, MatrixWorldY; // мировая
    D3DXMATRIX MatrixView;       // матрица вида
    D3DXMATRIX MatrixProjection; // матрица проекции
    // MatrixWorld
    UINT  Time  = timeGetTime() % 3000;
    FLOAT Angel = Time * (2.0f * D3DX_PI) / 3000.0f;
    D3DXMatrixRotationX( &MatrixWorldX,  Angel);
    D3DXMatrixRotationY( &MatrixWorldY,  Angel);
    D3DXMatrixMultiply(&MatrixWorld,&MatrixWorldX,&MatrixWorldY);
    pDevice->SetTransform( D3DTS_WORLD,&MatrixWorld);
```

```
    // MatrixView
    D3DXMatrixLookAtLH( &MatrixView,  &D3DXVECTOR3 ( 0.0f, 0.0f,-11.0f),
                                      &D3DXVECTOR3 ( 0.0f, 0.0f, 0.0f),
                                      &D3DXVECTOR3 ( 0.0f, 1.0f, 0.0f));
    pDevice->SetTransform( D3DTS_VIEW, &MatrixView);
    // MatrixProjection
    D3DXMatrixPerspectiveFovLH(&MatrixProjection,D3DX_PI/4,1.0f,1.0f,
     ↳100.0f);
    pDevice->SetTransform( D3DTS_PROJECTION, &MatrixProjection);

    D3DMATERIAL9  Material; // материал
    D3DLIGHT9     Light;    // свет

    // установим материал
    ZeroMemory( &Material, sizeof(D3DMATERIAL9));
    Material.Diffuse.r = Material.Ambient.r = 1.0f;
    Material.Diffuse.g = Material.Ambient.g = 1.0f;
    Material.Diffuse.b = Material.Ambient.b = 0.0f;
    Material.Diffuse.a = Material.Ambient.a = 1.0f;
    pDevice->SetMaterial( &Material);

    D3DXVECTOR3 VectorDir;
    // установим свет
    ZeroMemory( &Light, sizeof(D3DLIGHT9));
    Light.Type       = D3DLIGHT_DIRECTIONAL;
    Light.Diffuse.r  = 1.0f;
    Light.Diffuse.g  = 1.0f;
    Light.Diffuse.b  = 1.0f;
    Light.Range      = 1000.0f;
    // установим нормаль
    VectorDir = D3DXVECTOR3(0.0f, 0.0f,1.0f),
    D3DXVec3Normalize( (D3DXVECTOR3*)&Light.Direction, &VectorDir);
    pDevice->SetLight( 0, &Light);
    pDevice->LightEnable( 0, TRUE);
    pDevice->SetRenderState( D3DRS_LIGHTING, TRUE);
    pDevice->SetRenderState( D3DRS_AMBIENT, 0);
}
//**************************************************************************
//  Rendering()
//  стадия отрисовки сцены на экране
//**************************************************************************

VOID Rendering()
{
    if(pDevice == NULL)
        return;
```

```
    // очищаем задний буфер
    pDevice->Clear( 0, NULL, D3DCLEAR_TARGET| D3DCLEAR_ZBUFFER,
                             D3DCOLOR_XRGB(60,100,150), 1.0f, 0);
    // начало сцены
    pDevice->BeginScene();
    // стадия трансформации
    TL();
    pDevice->SetStreamSource( 0, pBV, 0, sizeof(CUSTOMVERTEX));
    // формат вершин
    pDevice->SetFVF( D3DFVF_CUSTOMVERTEX);
    // установка индексов
    pDevice->SetIndices(pBI);
    // вывод объекта
    pDevice->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 36, 0, 12);
    // конец сцены
    pDevice->EndScene();
    // представляем на экран
    pDevice->Present( NULL, NULL, NULL, NULL);
}
//************************************************************************
//  DeleteDirect3D()
//  освобождение захваченных ресурсов
//************************************************************************
VOID DeleteDirect3D()
{
     if( pBI  != NULL)
        pBI->Release();

    if( pBV  != NULL)
        pBV->Release();

    if( pDevice != NULL)
        pDevice->Release();

    if( pd3d9 != NULL)
        pd3d9->Release();
}
//************************************************************************
// MainWinProc()
// функция обработки сообщений Windows
//************************************************************************

LRESULT CALLBACK MainWinProc(HWND   hwnd,
                             UINT    msg,
                             WPARAM  wparam,
                             LPARAM  lparam)
```

```
{
switch(msg)
      {
     case WM_DESTROY:
              {
               DeleteDirect3D();
               PostQuitMessage(0);
               return(0);
               }
     case WM_KEYDOWN:
       {
        if(wparam==VK_ESCAPE)
        PostQuitMessage(0);
        return 0;
       }
       }
  return DefWindowProc(hwnd, msg, wparam, lparam);
}
//************************************************************************
// WinMain()
// входная точка приложения
//************************************************************************
int WINAPI WinMain( HINSTANCE hinstance,
                                    HINSTANCE hprevinstance,
                                    LPSTR lpcmdline,
                                    int ncmdshow)
{
WNDCLASSEX windowsclass;  // создаем класс
HWND       hwnd;          // создаем дескриптор окна
MSG        msg;           // идентификатор сообщения

// определим класс окна WNDCLASSEX
windowsclass.cbSize = sizeof(WNDCLASSEX);
windowsclass.style = CS_DBLCLKS | CS_OWNDC | CS_HREDRAW | CS_VREDRAW;
windowsclass.lpfnWndProc = MainWinProc;
windowsclass.cbClsExtra = 0;
windowsclass.cbWndExtra = 0;
windowsclass.hInstance = hinstance;
windowsclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
windowsclass.hCursor = LoadCursor(NULL, IDC_ARROW);
windowsclass.hbrBackground = (HBRUSH)GetStockObject(GRAY_BRUSH);
windowsclass.lpszMenuName = NULL;
windowsclass.lpszClassName = "WINDOWSCLASS";
windowsclass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
```

```
// регистрируем класс
if (!RegisterClassEx(&windowsclass))
    return(0);
// создаем окно
if (!(hwnd = CreateWindowEx(NULL,"WINDOWSCLASS",NULL,
              WS_OVERLAPPEDWINDOW | WS_VISIBLE,0,0,
              GetSystemMetrics(SM_CXSCREEN),
              GetSystemMetrics(SM_CYSCREEN),
              NULL, NULL, hinstance, NULL)))
return 0;

 if( SUCCEEDED( Initial( hwnd)))
  {
       if( SUCCEEDED( Create()))
       {
           ShowWindow( hwnd, SW_SHOWDEFAULT);
           UpdateWindow( hwnd);

              ZeroMemory( &msg, sizeof(msg));
              while( msg.message!=WM_QUIT)
                {
                  if(PeekMessage( &msg, NULL,0,0,PM_REMOVE))
                  {
                      TranslateMessage( &msg);
                      DispatchMessage( &msg);
                   }
                  else
                  Rendering();
                }
          }
  }
   return 0;
}
```

На рис. 2.3 изображена работа рассмотренной в этой главе программы Demo, исходный код которой вы также можете найти на компакт-диске в папке \Code\Demo.

Итак, в этой главе был продемонстрирован полный процесс по созданию трехмерной сцены. При построении сцены на этапе трансформации и освещения задействовался фиксированный конвейер. На первый взгляд изъянов у фиксированного конвейера нет, но постоянно повторяющийся блок операций по умножению каждой вершины на набор матриц и расчету освещения приводит к перегруженности фиксированного конвейера и, как

следствие, ограничению на вывод качественной трехмерной графики. В следующей главе вы познакомитесь с программируемым конвейером, имеющим более гибкую реализацию в процессе построения трехмерных сцен.

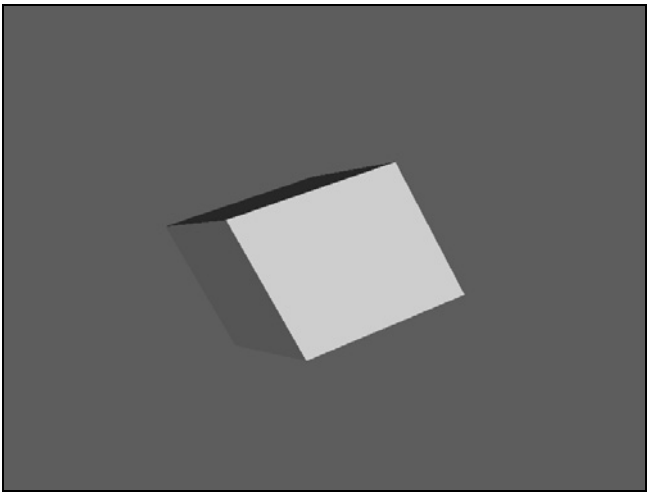
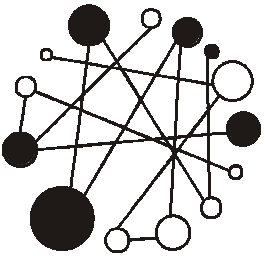


Рис. 2.3. Работа программы Demo

ГЛАВА 3



Программируемый графический конвейер

В предыдущей главе мы рассмотрели работу простого приложения Direct3D с применением фиксированного графического конвейера. К сожалению, фиксированный конвейер не обладает необходимой гибкостью, и если на заре появления трехмерных компьютерных игр он устраивал разработчиков, то сейчас механизм работы фиксированного конвейера явно устарел. В связи с этим в какой-то промежуток времени возник закономерный вопрос: что делать? Улучшение библиотечных функций Direct3D не могло привести к переходу на более качественный уровень графики. Тогда была разработана модель нового, программируемого конвейера. Соответственно поддержка такого конвейера осуществляется как со стороны DirectX, так и со стороны производителей видеоадаптеров. В *главе 1* на рис. 1.1 был представлен графический конвейер Direct3D9 и из него видно, что на стадии трансформации и освещения существует возможность использования нового программируемого конвейера. Этот конвейер работает с вершинами объекта с помощью специальных программ, написанных на языке вершинных шейдеров. Также на стадии растеризации можно использовать пиксельные шейдеры. Вершинные и пиксельные шейдеры находятся на различных стадиях общего графического конвейера, но два этих механизма можно отнести к программируемому графическому конвейеру. В чем же суть вершинных и пиксельных шейдеров?

Вершинные и *пиксельные шейдеры* — это написанные на специальном языке программы, с помощью которых можно аппаратно производить необходимые операции с вершиной и пикселом. Общая концепция вершинных и пиксельных шейдеров заключается в том, что программисту становится доступен набор определенных регистров на видеокарте, и с помощью языка вершинных и пиксельных шейдеров можно производить необходимые операции с вершиной и пикселом. Все операции производятся графически процессором видеоадаптера напрямую, т. е. аппаратно, что значительно улучшает трехмерную графику.

Что касается специального языка вершинных и пиксельных шейдеров, то тут имеется несколько вариантов. Во-первых, это язык, похожий на ассемблер, выполненный традиционно для низкоуровневых языков в виде инструкций. Версий ассемблерного языка вершинных и пиксельных шейдеров несколько, это:

- vs_1_1; ps_1_1; ps_1_3; ps_1_4;
- vs_2_0; vs_2_x; ps_2_x; ps_2_0;
- vs_3_0; ps_3_0.

Все версии усложняются в порядке возрастания и рассчитаны на различные категории видеоадаптеров. Сначала были маломощные видеокарты, которые поддерживали первую версию шейдеров, в дальнейшем появилась несколько усложненная версия шейдеров — вторая. Для этих версий шейдеров была адаптирована новая линейка более мощных видеокарт. И совсем недавно появилась третья версия шейдеров, рассчитанная на мощные и соответственно дорогие видеоадаптеры. Все версии ассемблерного языка вершинных и пиксельных шейдеров строятся на основе инструкций и усложняются по старшинству версий, сохраняя при этом обратную совместимость.

Но аналитики корпорации Microsoft естественно понимали, что разработчикам не нравится усложнение процесса программирования графики с помощью ассемблерного языка, поэтому был создан новый язык — высокоуровневый язык программирования шейдеров (High-Level Shader Language, HLSL). Как и все высокоуровневые языки, HLSL имеет очевидные преимущества в виде упрощенного синтаксиса, гибкости, мощности, и, самое главное, HLSL напоминает C-подобный язык программирования, что значительно упрощает его использование.

Высокоуровневый язык программирования шейдеров содержит в себе потенциал всех трех вариантов ассемблерного языка, но со своим синтаксисом. Корпорация Microsoft позиционирует HLSL как основной язык вершинных и пиксельных шейдеров в новой платформе XNA. По замыслу Microsoft язык HLSL будет использоваться как при создании компьютерных игр, так и игр для приставки XBox.

При работе с вершинными и пиксельными шейдерами общий принцип работы примерно одинаков, но реализация несколько разная, поэтому затронем каждую модель программирования шейдеров.

Вершинные шейдеры

Используя *вершинные шейдеры* в трехмерной графике, можно добиться потрясающе красивых и реалистичных эффектов при трансформации, деформации и анимации объектов. Вершинный шейдер не создает вершины объекта, он обрабатывает с помощью графического процессора видеокарты,

имеющиеся данные, значительно улучшая качество представляемой на экране монитора графики.

Применяя фиксированный конвейер при создании объекта, необходимо установить последовательно три матрицы, материал и свет. В программируемом конвейере задаются только те матрицы, которые необходимы и передаются все дальнейшие расчеты по трансформации и освещению программе вершинного шейдера, избегая тем самым массы повторяющихся операций, требующихся при пересчете освещения и позиции вершины. Вершинный шейдер за один машинный цикл обрабатывает одну вершину. Если вы используете программируемый конвейер, то фиксированный конвейер задействовать будет невозможно и наоборот.

Как уже упоминалось, программа вершинного шейдера может писаться на подобном ассемблеру языке или на HLSL, но принцип работы у обоих языков схож. При помощи декларации шейдера происходит связывание потока данных с регистрами вершинного шейдера, который аппаратно поддерживается видеоадаптером. После этого в работу вступает непосредственно программа шейдера, производящая операции по трансформации и расчету освещения. Графический процессор видеокарты на основании полученных данных производит соответствующие вычисления, выполняя построение объекта и расчет освещения.

Теперь о том, как работает вершинный шейдер. Связав поток данных с входными регистрами при помощи декларации, вы получаете доступ к регистрам. В константных регистрах хранятся матрицы трансформации и источники освещения, которые вы предварительно заносите в эти регистры. Результат перемножения вершины на матрицу заносится в выходной регистр или временный (если планируются еще какие-либо расчеты, связанные с этим значением) и на основании полученного результата производятся предопределенные операции с объектом. Все эти действия и выполняет вершинный шейдер. В теории это выглядит достаточно просто, но вот на практике все значительно сложнее.

Поскольку имеется несколько версий ассемблерного языка вершинных шейдеров и каждая версия по мере возрастания содержит возможности всех предыдущих версий, то имеет смысл познакомиться с последней третьей версией, учитывая при этом и более ранние версии шейдеров.

Итак, имеется определенное количество *регистров* для хранения значений. Для каждой из версий вершинных шейдеров доступны различные регистры, в табл. 3.1 приведены все регистры с учетом каждой версии шейдеров.

Входные регистры (Input Registers) выполнены в виде вектора (x, y, z, w) для загрузки вершин. Эти регистры предназначены только для чтения. Имеется всего шестнадцать входных регистров. Обозначаются они следующим образом: `v0`—`v15`.

Таблица 3.1. *Соответствие регистров и версий вершинных шейдеров*

Название регистров	Обозначение	Количество	Версии вершинных шейдеров			
			vs_1_1	vs_2_0	vs_2_x	vs_3_0
Входные регистры	v#	16	да	да	да	да
Временные регистры	r#	32	да	да	да	да
Адресный регистр	a0	1	да	да	да	да
Константные регистры для типа Float	c#	256	да	да	да	да
Константные регистры для типа Integer	i#	16	нет	да	да	да
Константные регистры для типа Boolean	b#	16	нет	да	да	да
Регистр счетчика циклов	aL	1	нет	да	да	да
Входные псевдорегистры вершинного шейдера	s#	4	нет	нет	нет	да
Выходной регистр булевого значения	p0	1	нет	нет	да	да
Выходные регистры	o#	12	да	да	да	да

Временные регистры (Temporary Registers) предназначены для хранения некой промежуточной информации. Каждый временный регистр доступен для однократной записи данных и трехразового доступа для чтения информации из этого регистра. Временных регистров 32 и обозначаются они как `r0`—`r31`.

Адресный регистр (Address Register) всего один, обозначается он как `a0` и служит для адресаций, связанных с константными регистрами.

Константные регистры для типа `Float` (Constant Float Registers) служат только для чтения. В этих регистрах могут содержаться матрицы и источники освещения. Информация, содержащаяся во всех константных регистрах действительна только на время выполнения шейдера. Количество константных регистров зависит от используемой версии шейдера, а количество доступных регистров можно узнать с помощью функции `MaxVertexShaderConst()`. Обозначаются регистры как `c0`—`c256`.

Константные регистры для типа `Integer` (Constant Integer Registers) используются для работы с циклами, их количество ограничено шестнадцатью и обозначаются они `i0`—`i15`.

Константные регистры для типа `Boolean` (Constant Boolean Registers) используются для логического управления процессом выполнения команд шейдера. Существует шестнадцать регистров, обозначенных как `b0`—`b15`.

Регистр счетчика циклов (Loop Counter Register) используется для подсчета циклов. Такой регистр всего один и обозначается он как `aL`.

Входные псевдорегистры вершинного шейдера (Sampler) необходимы для идентификации начальной стадии выборки текстур. Для работы нужно задействовать функцию `IDirect3DDevice9::SetSamplerState`.

Выходной регистр булевого значения (Predicate Register) необходим для содержания выходного булевого значения, обозначается этот регистр аббревиатурой `p0`.

Выходных регистров (Output Registers) двенадцать, они содержат исходный результат определенных операций, которые и участвуют в построении объекта. Выходные регистры обозначаются как `o0`—`o11`.

Итак, имеется набор регистров, для того чтобы связать необходимые регистры с потоком данных, т. е. чтобы получить возможность записывать данные в регистры, производятся вызовы библиотечных функций Direct3D, к которым мы вернемся позже в этом разделе. После того как вы разместили данные в соответствующих регистрах, управление программой передается в вершинный шейдер. Программа вершинного шейдера выполняет различные логические и математические операции с данными, находящимися в регистрах, а точнее, эта программа состоит из набора команд, которые выполняются процессором видеоадаптера.

Определенный набор команд или инструкций совершает различные операции с вершинами для расчета преобразований и освещения. Например, во входных регистрах у вас уже имеются загруженные вершины, а в константных регистрах результирующая матрица. Перемножив вершину на матрицу, полученный результат заносится в выходные регистры и уже на основании этих данных происходит построение объекта. В версии vs_3_0 вершинных шейдеров имеется большой набор инструкций, с помощью которых строится программа шейдера. В табл. 3.2 приведены все инструкции третьей версии, а также показана доступность тех или иных инструкций для всех имеющихся ранних версий вершинных шейдеров.

Для того чтобы воспользоваться набором имеющихся инструкций, в исходном коде программы шейдеров необходимо соблюдать следующий синтаксический шаблон:

```
op dest, src0, src1, src2, src3
```

Таблица 3.2. *Инструкции вершинных шейдеров*

Обозначение	Назначение	Слот	Версии вершинных шейдеров			
			vs_1_1	vs_2_0	vs_2_x	vs_3_0
abs	Вычисляет абсолютное значение	1	нет	да	да	да
add	Добавляет два вектора	1	да	да	да	да
break	Останавливает текущий цикл для loop и rep	1	нет	нет	да	да
breakp	Выходная точка для endloop и endrep	3	нет	нет	да	да
call	Производит запрос к отмеченной инструкции	2	нет	да	да	да
callnz bool	Производит запрос к отмеченной инструкции для константных регистров типа Boolean	3	нет	да	да	да
callnz pred	Производит запрос к отмеченной инструкции для выходного регистра булевого значения	3	нет	нет	да	да
crs	Вычисляет кросс-продукт	2	нет	да	да	да
dcl_usage outut register – vs	Декларация для выходных регистров	0	нет	нет	нет	да
dcl_samplerType	Декларация вершинного шейдера для samplerType	0	нет	нет	нет	да
def	Определяет константы для шейдера вершин	0	да	да	да	да
defb	Определяет константы шейдера вершин для загрузки Boolean-значения	0	нет	да	да	да
defi	Определяет константы шейдера вершин для загрузки Integer-значения	0	нет	да	да	да

Таблица 3.2 *(продолжение)*

Обозначение	Назначение	Слот	Версии вершинных шейдеров			
			vs_1_1	vs_2_0	vs_2_x	vs_3_0
dp3	Трехкомпонентный дот-продукт	1	да	да	да	да
dp4	Четырехкомпонентный дот-продукт	1	да	да	да	да
dst	Вычисляет вектор расстояния	1	да	да	да	да
else	Начало для блока else	1	нет	да	да	да
endif	Окончание для блока if bool...else	1	нет	да	да	да
endloop	Окончание для цикла loop	2	нет	да	да	да
endrep	Окончание блока rep...endrep	2	нет	да	да	да
exp	Экспоненциал, полная точность	1	да	да	да	да
expp	Экспоненциал, частичная точность	1	да	да	да	да
frc	Возвращает дробную часть входного компонента	1	да	да	да	да
if bool	Начало блока if...bool	3	нет	да	да	да
if_comp	Используется при пропуске блока кода, основанного на условиях	3	нет	нет	да	да
if pred	Начало блока if...pred	3	нет	нет	да	да
label	Маркирует следующую инструкцию	0	нет	да	да	да
lit	Вычисляет освещение	3	да	да	да	да
log	Полная точность $\log_2(x)$	1	да	да	да	да
logp	Частичная точность $\log_2(x)$	1	да	да	да	да
loop	Начало цикла loop	3	нет	да	да	да

Таблица 3.2 *(продолжение)*

Обозначение	Назначение	Слот	Версии вершинных шейдеров			
			vs_1_1	vs_2_0	vs_2_x	vs_3_0
lrp	Линейный интерполятор	2	нет	да	да	да
m3x2	Произведение вектора на матрицу размерностью 3×2	2	да	да	да	да
m3x3	Произведение вектора на матрицу размерностью 3×3	3	да	да	да	да
m3x4	Произведение вектора на матрицу размерностью 3×4	4	да	да	да	да
m4x3	Произведение вектора на матрицу размерностью 4×3	3	да	да	да	да
m4x4	Произведение вектора на матрицу размерностью 4×4	4	да	да	да	да
mad	Произведение и сложение	1	да	да	да	да
max	Максимум	1	да	да	да	да
min	Минимум	1	да	да	да	да
mov	Перемещение данных в регистры	1	да	да	да	да
mova	Перемещение данных с плавающей точкой в адресный регистр	1	нет	да	да	да
mul	Произведение	1	да	да	да	да
nop	Операция не производится	1	да	да	да	да
nrm	Нормализация 3D-вектора	3	нет	да	да	да
pow	Полная точность abs(src0)^{src1}	3	нет	да	да	да
rcp	Вычисляет обратное значение	1	да	да	да	да
rep	Начало блока rep	3	нет	да	да	да

Таблица 3.2 *(окончание)*

Обозначение	Назначение	Слот	Версии вершинных шейдеров			
			vs_1_1	vs_2_0	vs_2_x	vs_3_0
ret	Идентичен ключевому слову Return в C++	1	нет	да	да	да
rsq	Вычисляет квадратный корень	1	да	да	да	да
setp_comp	Объявление для начала работы с Predicate Register *(см. ранее)*	1	нет	нет	да	да
sge	Если значение больше или равно, то назначить	1	да	да	да	да
sgn	Если значение меньше, то назначить	3	нет	да	да	да
sincos	Вычисляет синус и косинус	8	нет	да	да	да
slt	Определяет меньшее из двух значений	1	да	да	да	да
sub	Вычитание	1	да	да	да	да
texldl	Производит определенную выборку текстур	5 или 2	нет	нет	нет	да
vs	Версия шейдера	0	да	да	да	да

- op — это инструкция или команда, с помощью которой производится назначенная операция;
- dest — указывает регистр для записи результата;
- src(0, 1, 2, 3, ...) — входные, временные или константные регистры, содержащие данные.

Общий смысл, я думаю, вам понятен: сначала идет инструкция, определяющая, что надо делать с данными во входных, временных или константных регистрах, а результат помещается в заданный регистр.

Но перед тем как производить операции с данными в регистрах, необходимо иметь эти данные в регистрах. Для чего нужно связать поток данных с регистрами, в Direct3D9 это достигается с помощью соответствующих деклараций, а именно структуры D3DVERTEXELEMENT9 с типизированным масси-

вом данных. Прототип структуры D3DVERTEXELEMENT9 выглядит следующим образом:

```
typedef struct _D3DVERTEXELEMENT9 {
   BYTE Stream;
   BYTE Offset;
   BYTE Type;
   BYTE Method;
   BYTE Usage;
   BYTE UsageIndex;
} D3DVERTEXELEMENT9;
```

Параметры:

- Stream — номер потока;
- Offset — первый такт для чтения;
- Type — тип входных данных;
- Method — здесь можно использовать несколько операторов, но значение по умолчанию D3DDECLMETHOD_DEFAULT является стандартным вариантом;
- Usage — определяет используемые данные, например вершины, нормали, текстуры;
- UsageIndex — специфический параметр, обычно ставится в 0.

С помощью указанной структуры определяется, каким образом происходит связывание потока данных и регистров, при этом установки, используемые в структуре D3DVERTEXELEMENT9, очевидным образом обязаны сочетаться с программным кодом шейдера.

После создания программы шейдера и объявления декларации необходимо воспользоваться функцией IDirect3DDevice9::CreateVertexDeclaration для вступления этой декларации в силу. Затем необходимо задействовать функцию D3DXAssembleShaderFromFile() для компиляции и загрузки программы вершинного шейдера. Далее для создания вершинного шейдера используется функция IDirect3DDevice9::CreateVertexShader. И в самом конце, во время рендеринга сцены, используется функция IDirect3DDevice9::SetVertexShader.

На этом все операции по созданию, загрузке и работе шейдера заканчиваются, теперь можно радоваться преимуществам третьей версии вершинных шейдеров, при условии, конечно, что эта версия шейдеров поддерживается аппаратно видеокартой.

Пиксельные шейдеры

По аналогии с вершинными шейдерами, которые работают с вершинами, *пиксельные шейдеры* выполняют операции на пиксельном уровне, что позволяет добиваться потрясающей детализации поверхности рисуемой графики. Процесс программирования пиксельных шейдеров по общей модели работы идентичен вершинным шейдерам, но адресация регистров несколько другая в силу иной архитектуры регистров. В табл. 3.3 представлены имеющиеся регистры с учетом доступности того или иного регистра в каждой из версий пиксельных шейдеров.

Входные цветовые регистры (Input Color Registers) содержат цвет вершины в виде RGBA значения, обозначаются они как `v#`. Эти регистры предназначены только для чтения. При использовании входных цветовых регистров применяется следующий вид записи:

```
dcl v#.M
```

- `dcl` — команда для объявления регистра;
- `v#` — входной регистр с номером;
- `M` — компонент RGBA.

Временных регистров (Temporary Registers) имеется 32, они предназначены для содержания различных промежуточных результатов, обозначаются они как `r#`.

Константные регистры для типа `Float` (Constant Float Registers) содержат 4-D константы. Функция `IDirect3DDevice9::SetPixelShaderConstantF` задействует константные регистры, обозначаются константные регистры для типа `Float`: `c#`.

Константные регистры для типа `Integer` (Constant Integer Registers) — это целочисленные регистры, использующиеся для работы с циклами `loop` и `rep`. Задействуются функцией `IDirect3DDevice9::SetPixelShaderConstantI`. Обозначаются константные регистры для типа `Integer`: `i#`.

Константные регистры для типа `Boolean` (Constant Boolean Register) используются для различных логических операций и обозначаются как `b#`. Устанавливаются с помощью функции `IDirect3DDevice9::SetPixelShaderConstantB`.

Входные псевдорегистры пиксельного шейдера (Sampler Register) необходимы для идентификации начальной стадии выборки текстур, обозначение: `s#`. Для работы нужно задействовать функцию `IDirect3DDevice9::SetSamplerState`.

Face-регистр или лицевой регистр используется только в третьей версии пиксельных шейдеров. Этот скалярный регистр типа `Float` может содержать лицевую область примитива, обозначается аббревиатурой `vFace`.

Таблица 3.3. *Соответствие регистров и версий пиксельных шейдеров*

Названия регистров	Обозначения	Кол-во	Версии пиксельного шейдера						
			ps_1_1	ps_1_2	ps_1_3	ps_1_4	ps_2_0	ps_2_x	ps_3_0
Входные цветовые регистры	v#	10	да	да	да	да	да	да	да
Временные регистры	r#	32	нет	нет	нет	нет	да	да	да
Константные регистры для типа Float	c#	224	да	да	да	да	да	да	да
Константные регистры для типа Integer	i#	16	нет	нет	нет	нет	нет	да	да
Константные регистры для типа Boolean	b#	16	нет	нет	нет	нет	нет	да	да
Входные псевдорегистры пиксельного шейдера	s#	16	нет	нет	нет	нет	да	да	да
Face-регистр	vFace	1	нет	нет	нет	нет	нет	нет	да
Регистр позиции	vPos	1	нет	нет	нет	нет	нет	нет	да
Регистр счетчика циклов	aL	1	нет	нет	нет	нет	да	да	да
Выходной регистр для булевого значения	p0	1	нет	нет	нет	нет	нет	да	да
Выходные цветовые регистры	oC#	Зависит от реализации	нет	нет	нет	нет	да	да	да
Выходные регистры глубины	oDepth	1	нет	нет	нет	нет	да	да	да

Регистр позиции (Position Register) содержит текущую позицию пиксела (x, y) соответствующих каналов, имеет обозначение `vPos`.

Регистр счетчика циклов (Loop Counter Register) используется для работы с циклом `loop`/`endloop` и обозначается `aL`.

Выходной регистр для булевого значения (Predicate Register) необходим для содержания булевого значения, носит обозначение `p0`.

Выходные цветовые регистры (Output Color Register) служат для вывода окончательной цветовой компоненты шейдера, имеют обозначение `oC#`.

Выходные регистры глубины (Output Depth Register) — это скалярные регистры, появившиеся еще в версии ps_2_0, они используются для тестирования буфера глубины, обозначены как `oDepth`.

Процесс работы пиксельного шейдера делится на два параллельных конвейера. На рис. 3.1 изображен пиксельный конвейер.

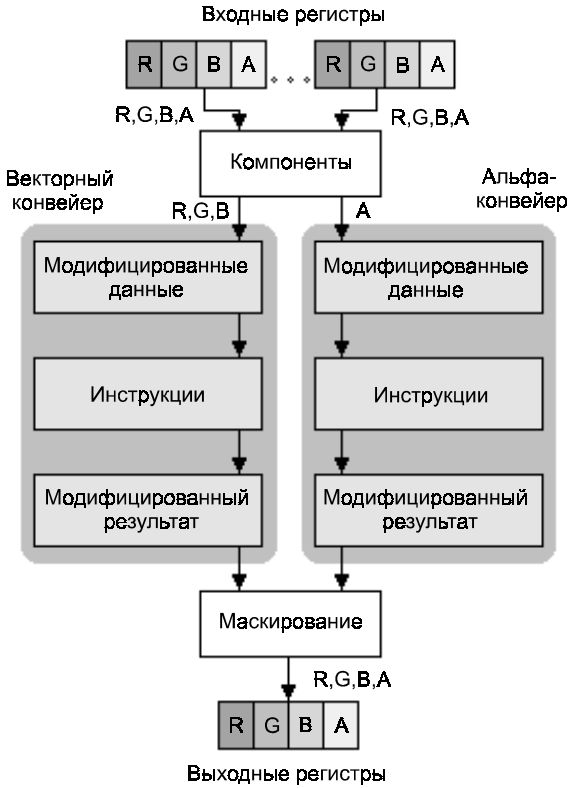


Рис. 3.1. Пиксельный конвейер

Таблица 3.4. *Инструкции пиксельных шейдеров*

Обозначения	Назначение	Слот	Версии пиксельных шейдеров						
			ps_1_1	ps_1_2	ps_1_3	ps_1_4	ps_2_0	ps_2_x	ps_3_0
`abs`	Вычисляет абсолютное значение	1	нет	нет	нет	нет	да	да	да
`add`	Добавляет два вектора	1	да	да	да	да	да	да	да
`break`	Выход из текущего цикла	1	нет	нет	нет	нет	нет	да	да
`break_comp`	Выход из текущего цикла	3	нет	нет	нет	нет	нет	да	да
`breakp`	Выход из текущего цикла	3	нет	нет	нет	нет	нет	да	да
`call`	Производит запрос к отмеченной инструкции	2	нет	нет	нет	нет	нет	да	да
`callnz bool`	Производит запрос к отмеченной инструкции для константных регистров типа `Boolean`	3	нет	нет	нет	нет	нет	да	да
`callnz pred`	Производит запрос к отмеченной инструкции для выходного регистра булевого значения	3	нет	нет	нет	нет	нет	да	да
`cmp`	Избирается `src1`, если `src0 >= 0`; иначе избирается `src2`	1	нет	да	да	да	да	да	да
`crs`	Вычисляет кросс-продукт	2	нет	нет	нет	нет	да	да	да
`dcl_samplerType`	Определяет тип данных	0	нет	нет	нет	нет	да	да	да
`dcl_usage`	Декларация использования пиксельного шейдера	0	нет	нет	нет	нет	нет	нет	да
`def`	Определяет константы для пиксельного шейдера	0	да	да	да	ад	да	да	да

Таблица 3.4 *(продолжение)*

Обозначения	Назначение	Слот	Версии пиксельных шейдеров						
			ps_1_1	ps_1_2	ps_1_3	ps_1_4	ps_2_0	ps_2_x	ps_3_0
defb	Определяет константы пиксельного шейдера для загрузки Boolean-значения	0	нет	нет	нет	нет	нет	да	да
defi	Определяет константы пиксельного шейдера для загрузки Integer-значения	0	нет	нет	нет	нет	нет	да	да
dp3	Трехкомпонентный дот-продукт	1	да	да	да	да	да	да	да
dp4	Четырехкомпонентный дот-продукт	1	нет	да	да	да	да	да	да
dsx	Вычисляет изменение состояния по X	2	нет	нет	нет	нет	нет	да	да
dsy	Вычисляет изменение состояния по Y	2	нет	нет	нет	нет	нет	да	да
else	Логический блок else	1	нет	нет	нет	нет	нет	да	да
endif	Маркирует конец блока if...else	1	нет	нет	нет	нет	нет	да	да
endloop	Окончание блока endloop	2	нет	нет	нет	нет	нет	нет	да
endrep	Окончание блока endrep	2	нет	нет	нет	нет	нет	да	да
exp	Экспоненциал, полная точность	1	нет	нет	нет	нет	да	да	да
frc	Фрактальный компонент	1	нет	нет	нет	нет	да	да	да
if bool	Начало блока if bool	3	нет	нет	нет	нет	нет	да	да
if_comp	Используется при пропуске блока кода, основанного на условиях	3	нет	нет	нет	нет	нет	да	да
if pred	Начало блока if pred	3	нет	нет	нет	нет	нет	да	да
label	Маркирует следующую инструкцию	0	нет	нет	нет	нет	нет	да	да

Таблица 3.4 *(продолжение)*

Обозначения	Назначение	Слот	Версии пиксельных шейдеров						
			ps_1_1	ps_1_2	ps_1_3	ps_1_4	ps_2_0	ps_2_x	ps_3_0
`log`	Полная точность $\log p_2(x)$	1	нет	нет	нет	нет	да	да	да
`loop`	Начало цикла `loop`	3	нет	нет	нет	нет	нет	нет	да
`lrp`	Линейный интерполятор	2	да	да	да	да	да	да	да
`m3x2`	Произведение вектора на матрицу размерностью 3×2	2	нет	нет	нет	нет	да	да	да
`m3x3`	Произведение вектора на матрицу размерностью 3×3	3	нет	нет	нет	нет	да	да	да
`m3x4`	Произведение вектора на матрицу размерностью 3×4	4	нет	нет	нет	нет	да	да	да
`m4x3`	Произведение вектора на матрицу размерностью 4×3	3	нет	нет	нет	нет	да	да	да
`m4x4`	Произведение вектора на матрицу размерностью 4×4	4	нет	нет	нет	нет	да	да	да
`mad`	Произведение и сложение	1	да	да	да	да	да	да	да
`max`	Максимум	1	нет	нет	нет	нет	да	да	да
`min`	Минимум	1	нет	нет	нет	нет	да	да	да
`mov`	Перемещение данных в регистры	1	да	да	да	да	да	да	да
`mul`	Произведение	1	да	да	да	да	да	да	да
`nop`	Операция не производится	1	да	да	да	да	да	да	да
`nrm`	Нормализация 3D-вектора	3	нет	нет	нет	нет	да	да	да
`pow`	Полная точность $\texttt{abs(src0)}^{\texttt{src1}}$	3	нет	нет	нет	нет	да	да	да

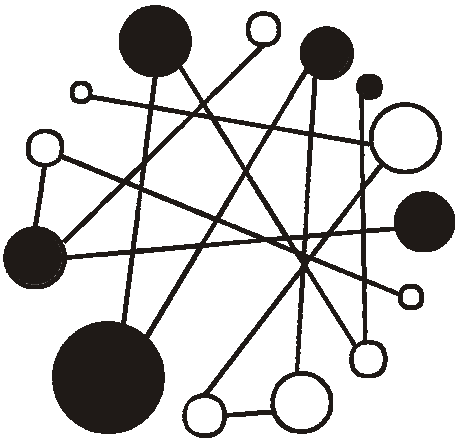
Таблица 3.4 *(окончание)*

Обозначения	Назначение	Слот	Версии пиксельных шейдеров						
			ps_1_1	ps_1_2	ps_1_3	ps_1_4	ps_2_0	ps_2_x	ps_3_0
ps	Версия пиксельного шейдера	0	да	да	да	да	да	да	да
rcp	Вычисляет обратное значение	1	нет	нет	нет	нет	да	да	да
rep	Начало блока rep	3	нет	нет	нет	нет	нет	да	да
ret	Идентичен ключевому слову Return в C++	1	нет	нет	нет	нет	нет	да	да
rsq	Вычисляет квадратный корень	1	нет	нет	нет	нет	нет	да	да
setp_comp	Объявление для начала работы с Predicate Register *(см. ранее)*	1	нет	нет	нет	нет	нет	да	да
sincos	Вычисляет синус и косинус	8	нет	нет	нет	нет	да	да	да
sub	Разность	1	да	да	да	да	да	да	да
texkill	Отменяет рендеринг текущего пиксела	2	да	да	да	да	да	да	да
texld	Производит выборку текстур	4 или 1	нет	нет	нет	нет	да	да	да
texldb	Производит выборку к текстурному уровню	6	нет	нет	нет	нет	да	да	да
texldl	Производит выборку текстур	5 или 2	нет	нет	нет	нет	нет	нет	да
texldd	Производит выборку текстур к пользовательским градиентам	3	нет	нет	нет	нет	нет	да	да
texldp	Производит выборку текстур	4 или 3	нет	нет	нет	нет	да	да	да

Одним из конвейеров является скалярный конвейер, предназначается он для работы с альфа-каналом. Второй конвейер называется векторным и работает с RGB-значением цвета. Та часть языка шейдеров, которая необходима для работы с пиксельными шейдерами, работает на основании математических операций и текстурных адресаций. Исходный результат размещается в выходных регистрах.

Так же как и в вершинных шейдерах, используется свой набор инструкций для работы с данными. В табл. 3.4 перечислены имеющиеся инструкции в соответствии с версиями пиксельного шейдера.

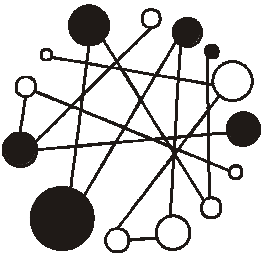
Создание и загрузка пиксельного шейдера почти идентична алгоритму, используемому при работе с вершинными шейдерами, за исключением декларации и установок пиксельного шейдера. Необходимую информацию вы сможете найти в документации по DirectX 9 SDK или в книге "DirectX 9. Уроки программирования на C++" издательства "БХВ-Петербург". В оставшихся частях книги мы подробно рассмотрим интегрированные средства разработки приложений от компаний ATI и NVIDIA, необходимые для отладки и программирования шейдеров, а также профайлер PIX for Windows.



ЧАСТЬ II

Инструментарий PIX for Windows

ГЛАВА 4



Первое знакомство с профайлером PIX for Windows

"Мы уверенны, что именно программное обеспечение будет определять будущее игровой индустрии".

Основатель корпорации Microsoft, Билл Гейтс

Из года в год корпорация Microsoft развивает и модернизирует выпускаемое программное обеспечение для платформ Windows, XBox и Windows Mobile, предлагая пользователям новый уровень программных продуктов. По словам руководителя и создателя корпорации Microsoft Билла Гейтса, в ближайшее десятилетие именно программное обеспечение выведет на новый и более качественный уровень игровую индустрию компьютерных развлечений. Одной из таких разработок как раз и является платформа XNA, представленная корпорацией Microsoft на ежегодной конференции разработчиков игр в городе Сан-Хосе штата Калифорния в 2004 году. Как уже упоминалось в *главе 1*, платформа XNA станет ключевой средой разработки приложений для Windows и XBox. Основу всей платформы будет составлять DirectX SDK, интегрированная среда разработки приложений Visual Studio, инструментарии PIX for Windows, XACT и высокоуровневый язык программирования шейдеров (HLSL). На данном этапе часть перечисленных средств уже доступна разработчикам программного обеспечения. Например, язык HLSL может использоваться только на компьютерных платформах, а инструмент PIX, до недавнего времени, применялся исключительно при создании игр для консольной приставки XBox. С выходом платформы XNA произойдет общая интеграция передовых средств корпорации Microsoft в единое целое, предоставляя разработчику мощнейший набор кроссплатформенных инструментов для создания игровых приложений. Одним из заметных средств новой платформы XNA должен стать профайлер PIX for Windows, анализирующий работу приложений. До сих пор инструментарий PIX был доступен только разработчикам игр для платформы XBox, а с выходом об-

новленной версии DirectX 9 SDK Update (Summer 2004) появилась новая версия под названием PIX for Windows, способная профилировать программы, написанные под операционную систему Windows. Что, безусловно, является показателем близости момента появления платформы XNA.

Профайлер PIX for Windows распространяется вместе с инструментальным средством разработчика DirectX 9 SDK Update (Summer 2004) в виде одной из утилит, поэтому вам будет необходима новая версия DirectX 9 SDK. При инсталляции DirectX 9 SDK некоторые устанавливаемые компоненты интегрируются в среду разработки приложений Visual Studio .NET, вследствие этого прежде установите Visual Studio .NET и только после этого инсталлируйте DirectX 9 SDK.

Установка DirectX 9 SDK (Summer 2004)

Для установки DirectX 9 SDK на жесткий диск вам понадобится около 520 Мбайт свободного места.

Двойной щелчок левой кнопки мыши на значке файла dxsdk_sum2004 приведет к распаковке архива во временную папку. После извлечения архива появится диалоговое окно **InstallShield Wizard**, уведомляющее пользователя об установке Microsoft DirectX 9 SDK на компьютер (рис. 4.1). Обратите внимание, что с левой стороны этого окна имеется красочно оформленная надпись **Microsoft DirectX & Microsoft XNA**, этот заголовок лишний раз подтверждает всю серьезность намерений корпорации Microsoft по продвижению платформы XNA.

Рис. 4.1. Диалоговое окно **InstallShield Wizard**

Нажав кнопку **Next** в окне **InstallShield Wizard**, вы попадете в окно с лицензионным соглашением от корпорации Microsoft, разъясняющим особенности использования DirectX 9 SDK. После изучения лицензионного соглашения и в случае согласия с предъявляемыми требованиями необходимо поставить галочку напротив поля **I accept the terms in the license** и нажать кнопку **Next** для продолжения процесса установки DirectX 9 SDK на ваш компьютер.

Следующее диалоговое окно носит название **Custom Setup** (рис. 4.2). Здесь имеется множество различных компонентов, которые предполагается инсталлировать на ваш компьютер. Все компоненты разбиты на шесть папок, рассмотрим назначение каждой из этих папок.

- Install DirectX Runtime — содержит Runtime-библиотеку DirectX 9.
- DirectX Documentation — документация по DirectX 9 SDK.
- DirectX Sample and Source Code — включает примеры и исходные коды для DirectX 9 SDK:
  - C++ Sample Source Code — исходные коды примеров для C++;
  - Managed Sample Source Code — исходные коды примеров для C#.
- DirectX Utilities — содержит утилиты, поставляемые в комплекте с DirectX 9 SDK:
  - DirectX extensions for Visual Studio .NET — средство отладки для вершинных и пиксельных шейдеров.

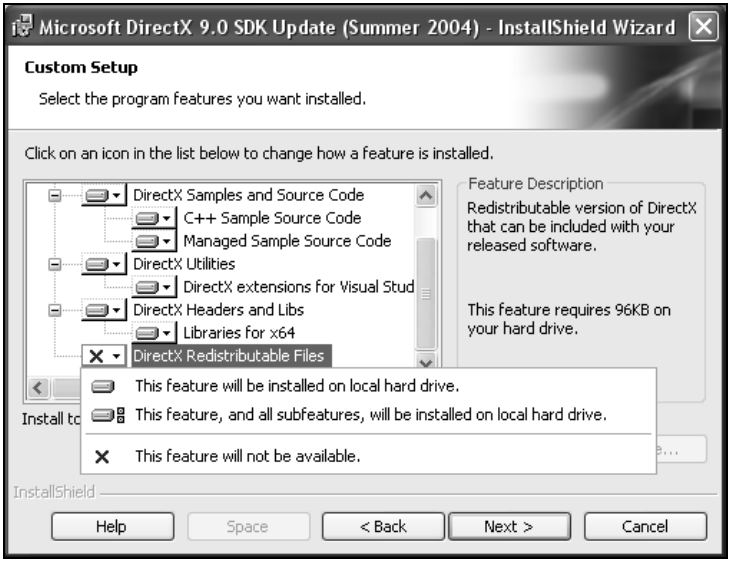


Рис. 4.2. Окно **Custom Setup**

- DirectX Headers and Libs — набор библиотечных и заголовочных файлов:
  - Libraries for x64 — библиотека для 64-битных процессоров.
- DirectX Redistributable Files — необходим для создания окончательной версии программы.

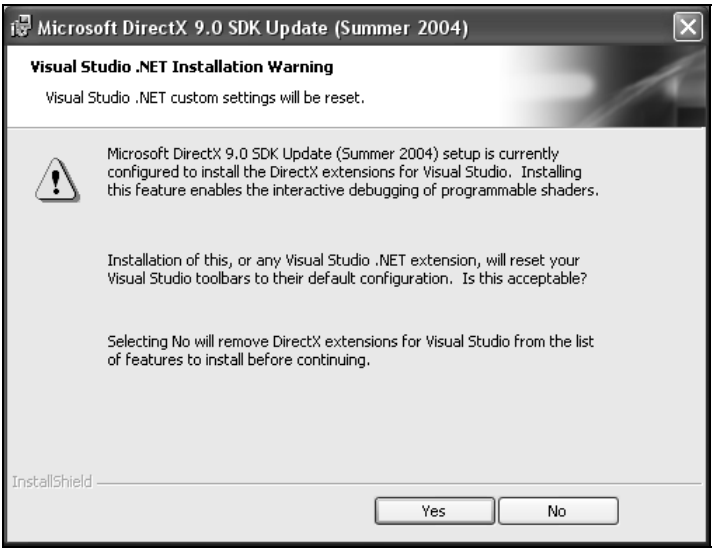


Рис. 4.3. Диалоговое окно **Visual Studio .NET Installation Warning**

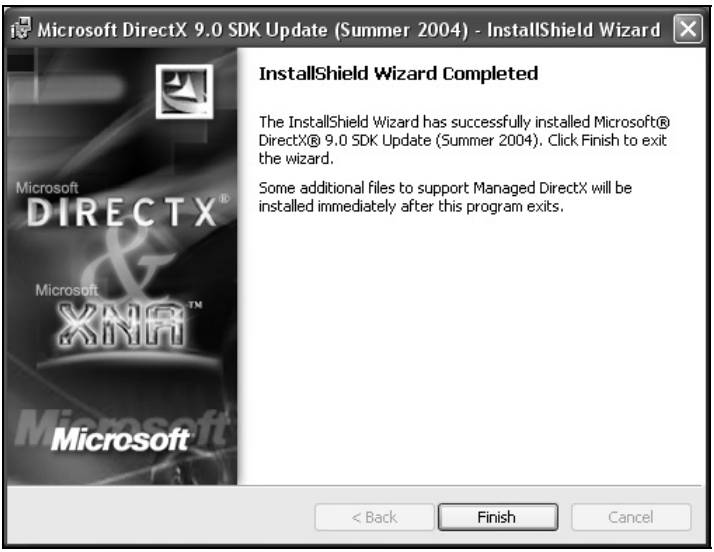


Рис. 4.4. Окно **InstallShield Wizard Completed**

Выбрав нужные компоненты в окне **Custom Setup** и нажав кнопку **Next**, вы попадете в окно **Visual Studio .NET Installation Warning** (рис. 4.3), предупреждающее о сбросе текущих настроек в Visual Studio .NET в связи с установкой компонента DirectX extensions for Visual Studio .NET (если таковой был выбран). Для продолжения установки нажмите кнопку **Next**, после чего произойдет непосредственная установка выбранных компонентов на компьютер.

После установки DirectX 9 SDK появится последние диалоговое окно **InstallShield Wizard Completed** (рис. 4.4), уведомляющее об окончании инсталляции и необходимости перезагрузки компьютера. Нажмите кнопку **Finish** для окончания процесса установки.

Профайлер PIX for Windows

При тестировании программы может выявиться ряд обстоятельств, по которым созданное приложение будет работать несколько медленнее, чем это предполагалось. Вместо того, чтобы переделывать программу заново или оптимизировать программный код, что несомненно приведет к большим временным затратам, можно воспользоваться так называемым профайлером (profiler). Этот вид программных продуктов необходим для выявления причин неадекватной работы приложения, путем сбора определенной информации во время работы программы. Одним из таких средств является PIX for Windows.

Инструментарий PIX for Windows — это профайлер, необходимый для детальной сборки информации об эффективности работы приложения. С помощью PIX for Windows программист может собрать полную информацию о работе тестируемого приложения и на основе полученной информации выявить "узкие" места в программе. Работа профайлера PIX for Windows основана на анализе каждого фрейма тестируемой программы, где возможно определение следующих типов данных:

- сравнение медленно выполняющихся фреймов с более быстрыми фреймами;
- идентификация совершающихся вызовов функций Direct3D;
- идентификация изменения состояний.

Используя PIX for Windows, вы параллельно производите запуск тестируемой программы, скажем на три-четыре секунды, и за этот небольшой промежуток времени PIX for Windows производит анализ работы приложения, показывая полученный результат в виде покадрового графика. Графическая раскадровка выбранного отрезка времени значительно упростит поиск причин, из-за которых тестируемое приложение теряет в производительности.

Для запуска профайлера PIX for Windows выполните команду **Start | All programs | Microsoft DirectX 9.0 Update (Summer 2004) | DirectX Utilities |**

PIX for Windows (Пуск | Все программы | Microsoft DirectX 9.0 Update (Summer 2004) | DirectX Utilities | PIX for Windows), после чего произойдет запуск профайлера и на экране монитора появится рабочая поверхность PIX for Windows, изображенная на рис. 4.5.

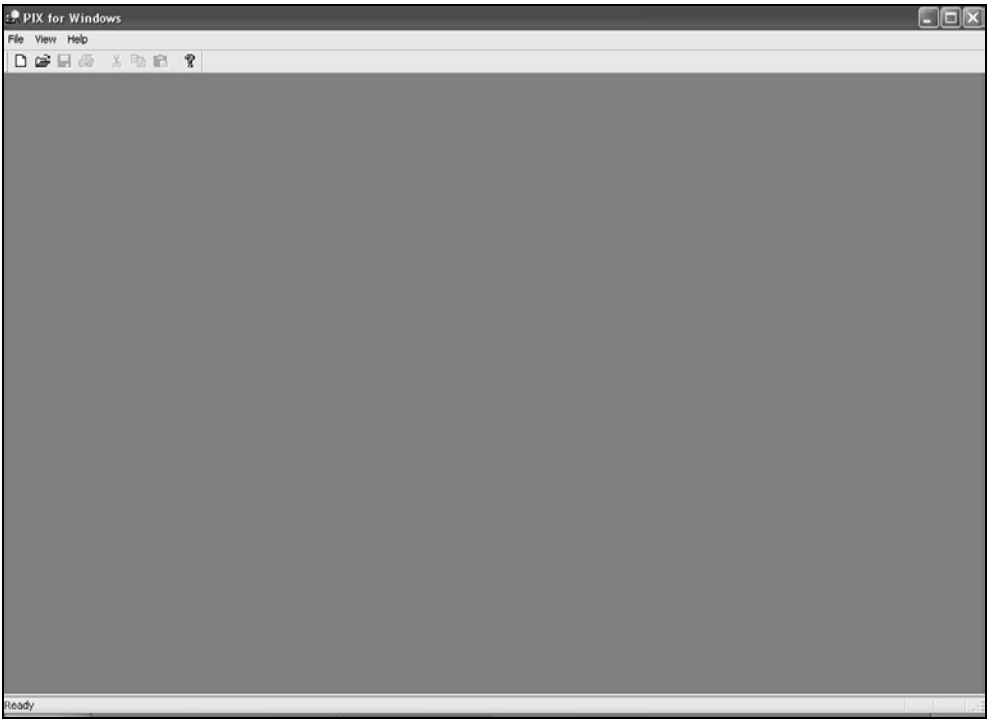


Рис. 4.5. Окно профайлера PIX for Windows

Главное меню PIX for Windows содержит три пункта: **File** (Файл), **View** (Вид) и **Help** (Помощь). Меню **File** (Файл) имеет четыре вложенные команды (приводятся с "горячими" клавишами в скобках):

- **New Experiment** (<Ctrl>+<N>) — открывает новый файл эксперимента;
- **Open** (<Ctrl>+<O>) — открывает необходимый файл эксперимента;
- **Close** — закрывает все открытые файлы;
- **Exit** — осуществляет выход из программы.

Меню **View** (Вид) содержит три вложенные команды:

- **Toolbar** — инструментальная панель;
- **Status Bar** — строка состояния;
- **Options** — определяет путь к DLL.

Меню **Help** (Помощь) имеет две вложенные команды:

- **Help Topics** — справочная система;
- **About PIX for Windows** — информация о версии и правообладателе программы PIX for Windows.

Инструментальная панель PIX for Windows выполнена в виде небольшого набора кнопок быстрого доступа. Всего имеется восемь кнопок:

- **New Experiment** — новый эксперимент;
- **Open** — открыть;
- **Save** — сохранить файл;
- **Print** — печать;
- **Cut** — удаление;
- **Copy** — копирование;
- **Paste** — вставка;
- **Help** — помощь.

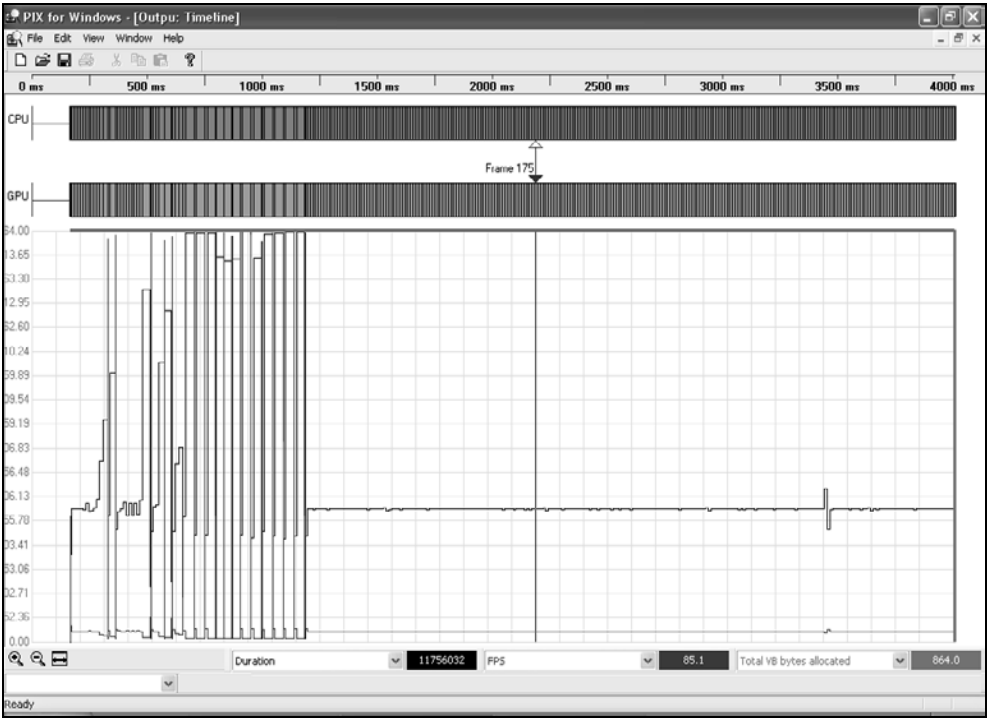


Рис. 4.6. Результат тестирования программы Demo

Думаю, что при первом ознакомлении с профайлером PIX for Windows этот инструмент не сильно впечатлит вас. Но как говорят: "Встречают по одежке, а провожают по уму". На рис. 4.6 изображен результат тестирования программы Demo, создание которой было рассмотрено в *главе 2*.

За крайне скудным интерфейсом скрыты весьма мощные функции, благодаря которым можно получить максимальную информацию о работе тестируемого приложения. Впоследствии, как мне кажется, PIX for Windows перерастет в отдельный мощный программный продукт с большим количеством конструктивных возможностей. Например, уже сейчас существует способ улучшения функционала PIX for Windows с помощью сторонних производителей, на основе подключаемых дополнительных модулей, выполненных в виде DLL-файлов.

Секрет

На момент написания книги, по слухам, компании ATI и NVIDIA как раз занимались разработкой соответствующих модулей для своих видеокарт.

Специфические функции PIX for Windows

В Microsoft DirectX 9.0 SDK Update (Summer 2004) имеется несколько функций, которые можно использовать в создаваемом Direct3D-приложении, интегрируя эти функции в свой исходный код. Но самое интересное, что некоторые функции не поддерживаются текущей версией PIX for Windows и созданы с заделом на будущее, что уже само по себе интересно. Давайте рассмотрим подробно прототипы существующих функций.

Функция `D3DPERF_BeginEvent()` определяет начало событий, прототип этой функции выглядит следующим образом:

```
int WINAPI D3DPERF_BeginEvent(
   D3DCOLOR col,
   LPCWSTR wszName)
```

Параметры функции `D3DPERF_BeginEvent()`:

- `col` — цвет, в который будет окрашено графическое отображение заданного события;
- `wszName` — исследуемое событие.

Функция `D3DPERF_EndEvent()` определяет окончание событий, прототип этой функции такой:

```
int WINAPI D3DPERF_EndEvent(VOID)
```

Функция `D3DPERF_SetMarker()` устанавливает маркер для событий, прототип этой функции выглядит следующим образом:

```
void WINAPI D3DPERF_SetMarker(
   D3DCOLOR col,
   LPCWSTR wszName)
```

Параметры функции `D3DPERF_SetMarker()`:

- `col` — цвет, в который будет окрашено графическое отображение заданного события;
- `wszName` — исследуемое событие.

Функция `D3DPERF_SetRegion()` помечает необходимый регион заданным цветом, но в настоящий момент она не поддерживается, прототип этой функции такой:

```
void WINAPI D3DPERF_SetRegion(
   D3DCOLOR col,
   LPCWSTR wszName)
```

Параметры функции `D3DPERF_SetRegion()`:

- `col` — цвет, в который будет окрашено графическое отображение заданного события;
- `wszName` — исследуемое событие.

Функция `QueryRepeatFrame()` представляет текущий кадр для повторного анализа PIX for Windows, и в данный момент не поддерживается. Прототип этой функции выглядит следующим образом:

```
bool WINAPI D3DPERF_QueryRepeatFrame(VOID)
```

Функция `D3DPERF_SetOptions()` определяет опции для работы текущей программы, и пока тоже не поддерживается. Прототип функции `D3DPERF_SetOptions()`:

```
void WINAPI D3DPERF_SetOptions(
   DWORD dwOptions)
```

Параметры функции `D3DPERF_SetOptions()`:

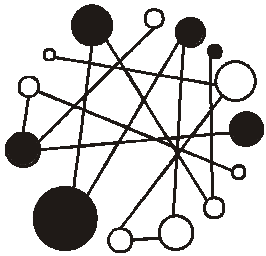
- `dwOptions` — устанавливает опции.

Функция `D3DPERF_GetStatus(VOID)` определяет текущий статус профилировщика. В данный момент не поддерживается, прототип этой функции выглядит так:

```
DWORD WINAPI D3DPERF_GetStatus(VOID)
```

В следующей главе мы подробнее познакомимся с функциональными возможностями профайлера PIX for Windows.

ГЛАВА 5



Работа с профайлером PIX for Windows

Работа с профайлером PIX for Windows основана на базе экспериментов с тестируемым приложением. Запуская PIX for Windows, вы задаете необходимые параметры для сборки информации *(параметры сборки)* и затем производите запуск программы на назначенный промежуток времени. За отведенный временной интервал работы программы профайлер PIX for Windows собирает информацию о параметрах, которые были заданы, сохранив полученные значения в отдельном файле, носящем название *выходной файл эксперимента*. Как правило, это файл с расширением PIXRun. Также еще имеется *файл эксперимента*, сохраняющий скомпонованные параметры сборки с расширением PIXExp. Открыв выходной файл эксперимента с помощью PIX for Windows, можно проанализировать полученные характеристики работы программы и выявить допущенные ошибки в программном коде.

Профайлер PIX for Windows производит замеры по установленным параметрам сборки. Эти параметры определены в PIX for Windows в виде списка, из которого вы вправе выбирать любой необходимый набор параметров сборки, компонуя все в отдельный файл эксперимента. Некоторые из параметров сборки в профайлере PIX for Windows заданы жестко и вне зависимости от вашего выбора замеры по этим параметрам будут производиться всегда, это:

- загруженность центрального процессора;
- загруженность графического процессора;
- продолжительность каждого фрейма;
- частота кадров (FPS).

Остальные параметры сборки добавляются опционально. Думается, что общий принцип работы с профайлером PIX for Windows вам ясен, поэтому давайте перейдем к более подробному анализу работы PIX for Windows.

Компоновка параметров сборки эксперимента

Первое, что необходимо сделать при тестировании программы с помощью PIX for Windows, — это произвести компоновку параметров для будущего эксперимента. Откройте профайлер PIX for Windows, выполнив команду в меню **Start | All programs | Microsoft DirectX 9.0 Update (Summer 2004) | DirectX Utilities | PIX for Windows** (Пуск | Все программы | Microsoft DirectX 9.0 Update (Summer 2004) | DirectX Utilities | PIX for Windows). После запуска профайлера PIX for Windows для создания нового файла эксперимента выберите в меню команду **File | New Experiment** (Файл | Новый эксперимент), или нажмите комбинацию клавиш <Ctrl>+<N>, или воспользуйтесь кнопкой **New Experiment** (Новый эксперимент) на панели инструментов PIX for Windows. После этих действий вам откроется диалоговое окно нового эксперимента **Experiment1**, изображенное на рис. 5.1, где и происходит компоновка файла эксперимента.

Диалоговое окно нового эксперимента **Experiment1** разделено на пять статических областей, каждая из которых имеет свой набор элементов управления. С помощью имеющихся элементов управления задаются необходимые опции, на основе которых впоследствии происходит компоновка файла эксперимента. В нижней части диалогового окна **Experiment1** под статическими областями располагается кнопка **Start Experiment** (Начало эксперимента), нажав эту кнопку вы запустите процесс выполнения эксперимента. Давайте рассмотрим сборку файла эксперимента.

Настройки запуска программы

Первая статическая область **Target startup options** (Опции запуска), показанная на рис. 5.2 диалогового окна **Experiment1**, имеет три следующих текстовых поля:

- **Program path** — указывается путь к тестируемой программе;
- **Startup folder** — задается путь к каталогу всей тестируемой программы;
- **Command-line arguments** — применяется при использовании командной строки.

Произведя соответствующие установки в текстовых полях, нужно сохранить будущий файл эксперимента, выбрав в меню команду **File | Save As** (Файл | Сохранить как). Действия по сохранению файла эксперимента обязательны, но если вы вдруг забудете это сделать, на этапе запуска процесса эксперимента PIX for Windows вам обязательно напомнит об этом, показав диалоговое окно, изображенное на рис. 5.3.

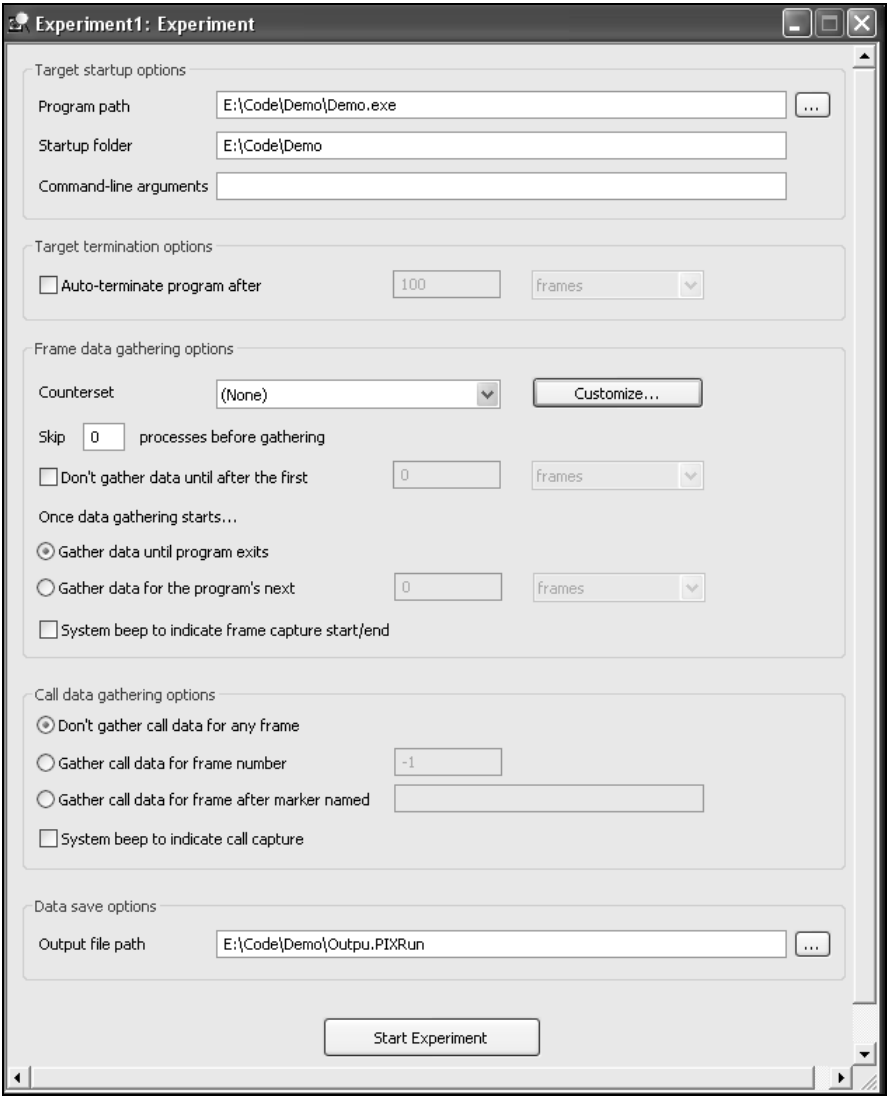


Рис. 5.1. Диалоговое окно эксперимента **Experiment1**

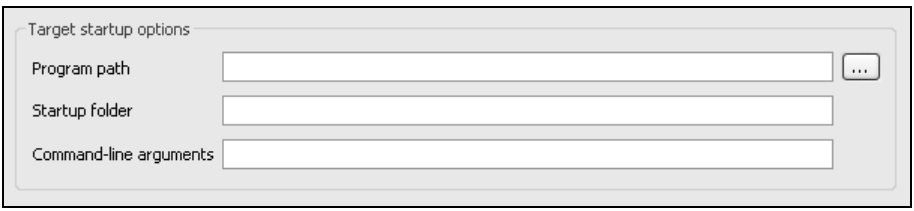


Рис. 5.2. Область **Target startup options**

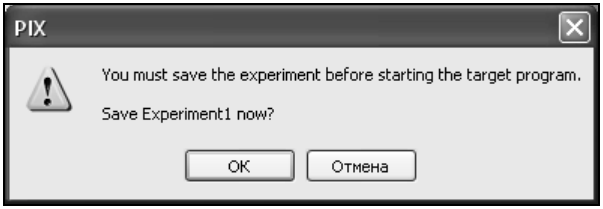


Рис. 5.3. Окно, предупреждающее о необходимости сохранения файла эксперимента

Указав путь и папку к тестируемой программе, в текстовых полях области **Target startup options** (Опции запуска) можно перейти к настройке параметров окончания эксперимента.

Настройки параметров окончания работы программы

В статической области **Target terminaten options** (Опции завершения) (рис. 5.4) диалогового окна **Experiment1** имеется элемент управления **Auto-terminaten program after** (Автоматическое завершение программы), выполненный в виде переключателя (Radio button).

Если в поле **Target terminaten options** (Опции завершения) не выбран переключатель **Auto-terminaten program after** (Автоматическое завершение программы), то временной промежуток, отведенный для процесса тестирования программы, должен регулироваться вами. В том случае, если переключатель **Auto-terminaten program after** (Автоматическое завершение программы) будет выбран, завершение процесса тестирования произойдет автоматически. При выборе переключателя **Auto-terminaten program after** (Автоматическое завершение программы) с правой стороны от названия переключателя станут доступными текстовое поле и выпадающий список, где и происходит определение значений по окончанию работы тестируемой программы. В выпадающем списке имеется три элемента на выбор:

- **frames** — количество фреймов, после которых происходит остановка процесса тестирования программы;
- **mc** — время в миллисекундах, на которое производится запуск программы;
- **seconds** — время в секундах, на которое производится запуск программы.

Рис. 5.4. Область **Target terminaten options**

Выбрав нужный элемент из выпадающего списка, в текстовом поле рядом с ним необходимо указать числовое значение, по которому будет определяться окончание работы программы, при этом, естественно, учитывая выбранный элемент списка. То есть если вы выбрали исчисление времени работы тестируемой программы в миллисекундах, то запись в текстовом поле должна реализовываться соответствующим образом. Определившись с интервалом, необходимым для работы тестируемой программы, можно перейти к этапу задания самих параметров сборки.

Определение параметров сборки

Область **Frame data gathering options** (Опции сборки данных для каждого фрейма) в диалоговом окне **Experiment1** профайлера PIX for Windows показана на рис. 5.5 и содержит свой набор элементов управления.

Первый элемент управления **Counterset** (Комплект сборки) области **Frame data gathering options** (Опции сборки данных для каждого фрейма) позволяет определить комплект параметров, которые вы стремитесь собрать в отношении работы каждого фрейма тестируемой программы. Элемент управления **Counterset** (Комплект сборки) является ключевым в определении параметров сборки, поэтому давайте прежде рассмотрим оставшиеся элементы управления в статической области **Frame data gathering options** (Опции сборки данных для каждого фрейма), а потом вернемся к элементу управления **Counterset** (Комплект сборки).

- **Skip processes before gathering** — определяет количество пропущенных процессов до начала сборки информации.
- **Don't gather data until after the first** — это флажок, указывающий на временной интервал либо на количество фреймов, по прошествии которых происходит процесс сборки параметров. Задается числом и исчисляется в секундах, миллисекундах или количеством фреймов.

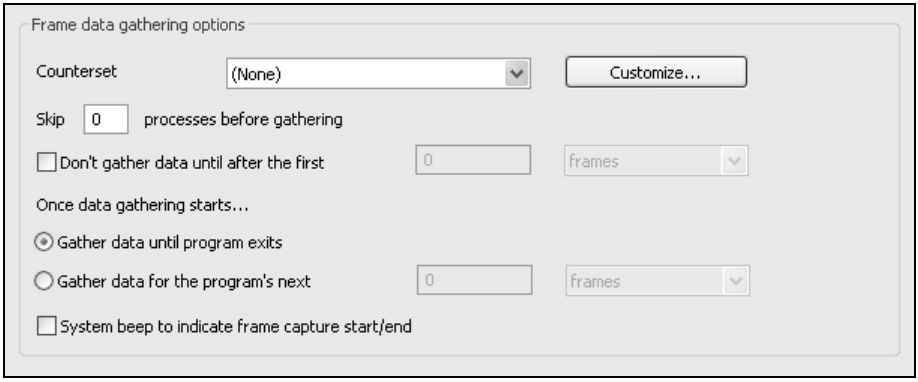


Рис. 5.5. Область **Frame data gathering options**

- **Gather data until program exits** — переключатель, задающий сборку информации до выхода из программы.
- **Gather data for the program's next** — выбор этого переключателя определит процесс сборки информации в течение работы всей программы.
- **System beep to indicate frame capture start/end** — системный гудок начала или окончания процесса сборки.

Теперь вернемся к элементу управления **Counterset** (Комплект сборки) и его значению в настройках параметров сборки. С правой стороны от элемента управления **Counterset** (Комплект сборки) располагается выпадающий текстовый список и кнопка **Customize** (Выбор). В выпадающем текстовом списке присутствуют три следующих варианта выбора параметров сборки:

- **None** — не выбран ни один параметр для сборки;
- **Counterset** — шаблонный набор параметров сборки;
- **Custom** — выборочный набор параметров сборки.

Вариант **None** (Никакой) в текстовом списке **Counterset** (Комплект сборки) не устанавливает ни одного параметра для сборки информации, но тем не менее он производит ряд следующих замеров:

- загруженность центрального процессора;
- загруженность графического процессора;
- продолжительность каждого фрейма;
- частота кадров (FPS).

Это тот самый стандартный набор замеров, который автоматически также будет доступен в любом из выбранных вами вариантов сборки.

Следующий вариант сборки **Counterset** (Комплект сборки) представляет собой предопределенный комплект из пяти параметров:

- **Instantaneous FPS** — количество выводимых изображений в секунду;
- **Number of Calls** — количество запросов;
- **Number of Draw calls** — количество запросов функций DirectX, связанных с отрисовкой примитива;
- **Time spent in Draw calls** — потраченное время на запрос функций DirectX, связанных с отрисовкой примитива;
- **Total resource objects allocated** — показывает распределение ресурсов.

Последний вариант сборки **Custom** (Выбор) в текстовом списке **Counterset** (Комплект сборки) области **Frame data gathering options** (Опции сборки данных для каждого фрейма) предоставляет возможность выбора параметров сборки из их огромного количества. Выбрав вариант **Custom** (Выбор), нажмите на кнопку **Customize** (Выбрать), и вам откроется новое диалоговое

окно **PIX Counters** (Сборщики параметров профайлера PIX), изображенное на рис. 5.6.

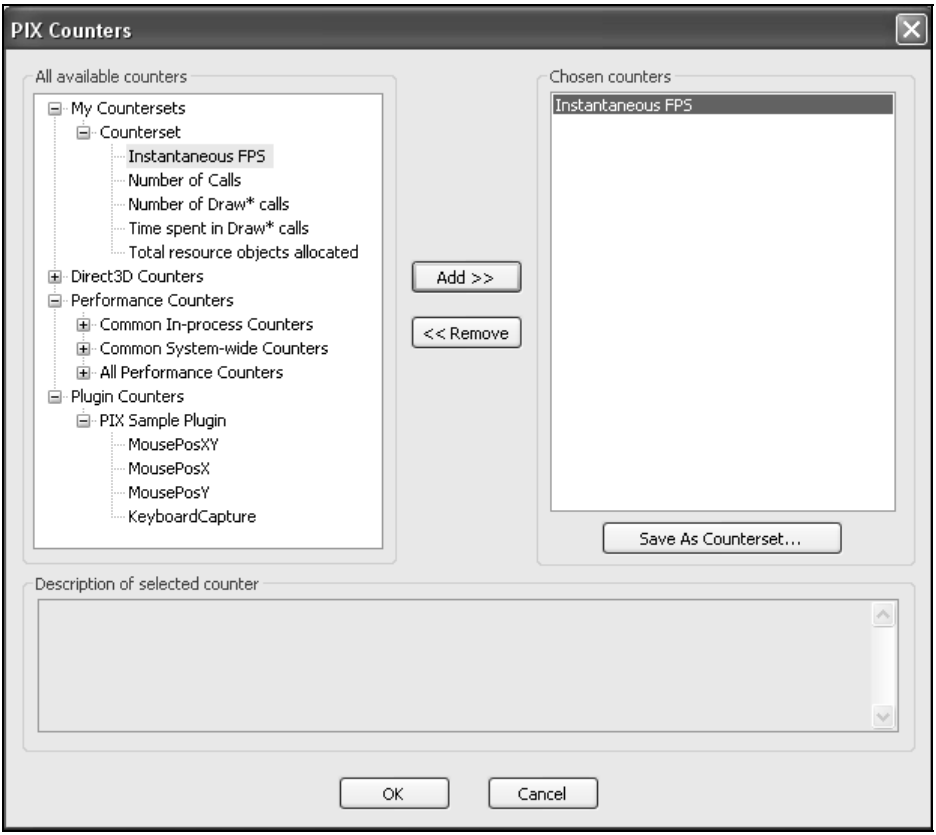


Рис. 5.6. Диалоговое окно **PIX Counters**

Диалоговое окно **PIX Counters** (Сборщики параметров профайлера PIX) разделено на две большие рабочие области в виде нередактируемых текстовых полей. С левой стороны окна **PIX Counters** (Сборщики параметров профайлера PIX) имеется область **All available counters** (Все доступные параметры сборки), а с правой стороны область **Chosen counters** (Выбранные параметры сборки). В области **All available counters** находятся все доступные параметры сборки, разделенные на четыре категории и реализованные в виде древовидной структуры. Опускаясь по иерархии каждой из категорий, можно производить выбор одного или нескольких параметров. Для этого на среднем уровне между двух рабочих областей диалогового окна **PIX Counters** (Сборщики параметров профайлера PIX) расположены две кнопки **Add** (Добавить) и **Remove** (Удалить). Выделив щелчком левой кнопки мыши необходимый параметр из категории, нажмите на кнопку **Add**, и выбранный вами

параметр переместится в левую рабочую область **Chosen counters** (Выбранные параметры сборки). Все доступные параметры сборки именно из левой области **Chosen counters** (Выбранные параметры сборки) будут задействованы в процессе тестирования программы.

Также можно создать свой шаблон параметров сборки для последующих процессов сборки информации. Для этого поместите в левую область **Chosen counters** (Выбранные параметры сборки) необходимое количество параметров и нажмите на кнопку **Save As Counterset**, которая расположена под областью **Chosen counters** (Выбранные параметры сборки). После нажатия этой кнопки появится небольшое диалоговое окно **Choose a name for this counterset** (Выберите название для параметров сборки), показанное на рис. 5.7. В появившемся окне нужно задать имя создаваемому вами шаблону параметров и нажать кнопку **OK**, после чего шаблон автоматически добавится в иерархию категории параметров сборки **My Countersets** (Мои параметры сборки).

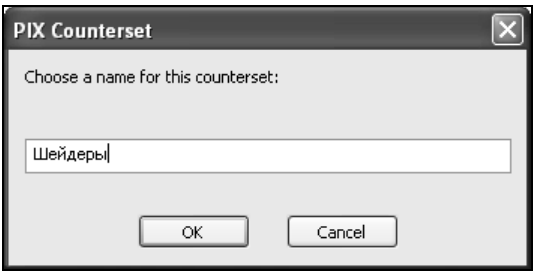


Рис. 5.7. Окно выбора имени пользовательского шаблона

Если вы хотите удалить один из добавленных вами параметров сборки в категории **My Countersets** (Мои параметры сборки), то необходимо раскрыть иерархию этой категории, щелкнуть правой кнопкой мыши на ненужном параметре сборки и в появившемся контекстном меню выбрать команду **Delete** (Удалить).

Как уже упоминалось, все имеющиеся параметры сборки разделены на четыре категории: **My Countersets** (Мои параметры сборки), **Direct3D Counters** (Параметры сборки Direct3D), **Performance Counters** (Сборщик рабочих характеристик) и **Plugin Counters** (Дополнительно подключаемые модули). Каждая из категорий имеет свой набор параметров, поэтому рассмотрим подробно каждую категорию.

Категория *My Countersets*

В категории **My Countersets** (Мои параметры сборки) изначально определен один вложенный список параметров сборки под названием **Counterset** (Параметры сборки). Чтобы увидеть этот вложенный список параметров, на-

жмите на пиктограмму, выполненную в виде квадратика с плюсом, расположенную слева от названия категории, и раскроется вся иерархия категории **My Countersets** (Мои параметры сборки). Иерархия каждой категории выполнена традиционным образом в виде древовидной структуры с набором списков всевозможных параметров. Список **Counterset** включает в себя следующие пять параметров:

- **Instantaneous FPS** — количество выводимых изображений в секунду;
- **Number of Calls** — количество запросов;
- **Number of Draw* calls** — количество запросов функций DirectX, связанных с отрисовкой примитива;
- **Time spent in Draw* calls** — потраченное время на запрос функций DirectX, связанных с отрисовкой примитива;
- **Total resource objects allocated** — показывает распределение ресурсов.

Как вы, наверное, заметили, перечисленный список абсолютно идентичен варианту **Counterset** из выпадающего текстового списка области **Frame data gathering options** (Опции сборки данных для каждого фрейма) диалогового окна **Experiment1**. На самом деле это один и тот же список параметров сборки, а его местонахождение определено как раз в категории **My Countersets** (Мои параметры сборки). Вдобавок к уже имеющемуся списку параметров, как мы выяснили, вы вправе добавлять свои дополнительные шаблоны, необходимые вам для работы.

Категория *Direct3D Counters*

Категория **Direct3D Counters** (Параметры сборки Direct3D) содержит множество вложенных друг в друга списков с различными параметрами сборки. С помощью списков собирается информация о произведенных запросах библиотечных функций Direct3D. Раскрыв иерархию категории **Direct3D Counters** (Параметры сборки Direct3D), вы обнаружите пять основных списков параметров, включающих в себя еще ряд дополнительных вложений. Основные пять списков имеют следующие названия:

- **General** — основные параметры сборки;
- **Primitives** — определяет количество вызовов библиотечных функций, связанных с прорисовкой примитива;
- **Resources** — наблюдает за распределением памяти;
- **State Changes** — следит за изменением состояний;
- **Shader Constant Changes** — просчитывает число констант шейдера.

Все перечисленные пять списков содержат множественные вложения параметров для сборки информации. Список **General** (Основные) собирает основную метрику Direct3D программы. Раскрыв иерархию списка **General**

(Основные), вы обнаружите два параметра сборки, с помощью которых можно собрать следующую информацию:

- **Frames per second** — количество кадров в секунду;
- **Calls per frame** — количество вызовов Direct3D, произведенных в пределах одного кадра. Библиотечные функции D3DX не используются в этом параметре сборки.

Выделив необходимый сборщик в списке **General** (Основные), перенесите его в область **Chosen counters** (Выбранные параметры сборки) и произведите необходимые операции по тестированию программы.

Следующий список **Primitives** (Примитивы) категории **Direct3D Counters** (Параметры сборки Direct3D) производит мониторинг количества вызовов библиотечных функций, связанных с построением примитива. Список **Primitives** (Примитивы) содержит пять вложенных списков:

- **Draw calls** — общее количество вызовов;
- **DrawPrimitive calls** — сборщик количества вызовов для Direct3D функции `IDirect3DDevice9::DrawPrimitive`;
- **DrawIndexedPrimitiv calls** — сборщик количества вызовов для Direct3D функции `IDirect3DDevice9::DrawIndexedPrimitive`;
- **DrawPrimitiveUP calls** — сборщик количества вызовов для Direct3D функции `IDirect3DDevice9::DrawPrimitiveUP`;
- **DrawIndexedPrimitiveUP calls** — сборщик количества вызовов для Direct3D функции `IDirect3DDevice9::DrawIndexedPrimitiveUP`.

Как вы заметили, четыре последних параметра сборки из списка **Primitives** (Примитивы) используются для наблюдения за запросами к библиотечным функциям Direct3D, на основе которых происходит построение примитива. Для каждого из пяти списков можно прослеживать два следующих параметра сборки информации, которые доступны при раскрытии иерархии этих списков:

- **Number of calls** — число произведенных вызовов;
- **Time spent in calls** — затраченное время на каждый вызов.

Определяя таким образом параметры сборки для списка **Primitives** (Примитивы), вы сможете проследить за качеством операций по построению объекта.

Список **Resources** (Ресурсы) категории **Direct3D Counters** (Параметры сборки Direct3D) позволяет собирать информацию, связанную с распределением памяти ресурсов. Можно установить сборщики параметров для вершинного и индексного буфера, текстур, а также собрать обобщенную информацию о распределении памяти ресурсов. Список **Resources** (Ресурсы) имеет четыре вложенных списка:

- **All resources** — общая информация о распределении памяти ресурсов;
- **Vertex buffers** — информация, связанная с буфером вершин;
- **Index buffers** — информация, связанная с индексным буфером;
- **Textures** — информация, связанная с текстурами.

Каждый из перечисленных списков можно раскрыть нажатием на пиктограмму, выполненную в виде изображения квадратика с плюсом. В открывшейся древовидной иерархии каждого списка находится еще по пять одинаковых вложенных списков, определяющих класс памяти для содержания буфера ресурса. Под *буфером ресурса* в Direct3D подразумевается буфер вершин, индексный буфер и текстурные координаты. Буфер ресурса в Direct3D задается перечисляемым типом `D3DPOOL`. Следующие пять вложенных списков определяют параметры сборки для соответствующих форматов хранения буфера ресурса.

- **All pool** — все форматы;
- **Default pool** — форматы, заданные по умолчанию. Соответствует флагу `D3DPOOL_DEFAULT` типа `D3DPOOL`;
- **Managed pool** — форматы определяются автоматически. Соответствует флагу `D3DPOOL_MANAGED` типа `D3DPOOL`;
- **System pool** — форматы, использующие системную память. Соответствует флагу `D3DPOOL_SYSTEMMEM` типа `D3DPOOL`;
- **Scratch pool** — формат, использующий системную память, но не обновляющийся при потере ресурса. Соответствует флагу `D3DPOOL_SCRATCH` типа `D3DPOOL`.

Все пять вложенных списков можно также раскрыть, нажав на пиктограмму в виде квадратика с плюсом, после чего станут доступны параметры сборки информации. Каждый вложенный список имеет одинаковый набор следующих параметров сборки информации:

- **Total bytes allocated as of this frame** — суммарное количество распределенных байт одного фрейма;
- **New bytes allocated this frame** — новые распределенные байты одного фрейма;
- **Total objects allocated as of this frame** — суммарное количество объектов, распределенных для одного фрейма;
- **New objects allocated this frame** — новые распределенные объекты для одного фрейма;
- **Number of locks of this frame** — количество блокировок, производимых в течение одного фрейма;
- **Number of unlocks of this frame** — количество разблокировок, производимых в течение одного фрейма.

Выбрав необходимый параметр сборки из списка **Resources** (Ресурсы), вы сможете проследить за распределением памяти ресурсов вашей программы.

Предпоследний список **State Changes** (Изменения состояния) в категории параметров **Direct3D Counters** (Параметры сборки Direct3D) позволяет определить количество вызовов библиотечных функций, связанных с устройством Direct3D, с помощью которых происходит изменение состояния. Список **State Changes** (Изменения состояния) содержит семь параметров сборки:

- **SetRenderState per frame** — фиксирует количество вызовов функции `IDirect3DDevice9::SetRenderState` в фрейме;
- **SetTextureStageState per frame** — фиксирует количество вызовов функции `IDirect3DDevice9::SetTextureStageState` в фрейме;
- **Misc FF state sets per frame** — определяет общее количество вызовов функции изменения состояния в фрейме;
- **SetRenderTarget calls this frame** — фиксирует количество вызовов функции `IDirect3DDevice9::SetRenderTarget` в текущем фрейме;
- **SetVertexShader calls this frame** — фиксирует количество вызовов функции `IDirect3DDevice9::SetVertexShader` в фрейме;
- **SetPixelShader calls this frame** — фиксирует количество вызовов функции `IDirect3DDevice9::SetPixelShader` в текущем фрейме;
- **SetTexture calls this frame** — фиксирует количество вызовов функции `IDirect3DDevice9::SetTexture` в текущем фрейме.

Последний список параметров сборки **Shader Constant Changes** (Изменения констант шейдера) категории параметров сборки **Direct3D Counters** (Параметры сборки Direct3D) следит за числом установленных констант вершинных и пиксельных шейдеров. Список **Shader Constant Changes** (Изменения констант шейдера) очень полезен при тестировании приложений, построенных на основе работы с шейдерами. В этом списке имеются следующие параметры сборки:

- **SetVertexShaderConstant<T> calls per frame** — фиксирует число установленных констант вершинного шейдера для следующих функций:
  - **IDirect3DDevice9::SetVertexShaderConstantB** — служит для установки булевой константы вершинного шейдера;
  - **IDirect3DDevice9::SetVertexShaderConstantF** — служит для установки дробного значения константы вершинного шейдера;
  - **IDirect3DDevice9::SetVertexShaderConstantI** — служит для установки целочисленного значения константы вершинного шейдера.

- **SetPixelShaderConstant<T> calls per frame** — фиксирует число установленных констант пиксельного шейдера для следующих функций:
  - **IDirect3DDevice9::SetVertexPixelConstantB** — служит для установки булевой константы пиксельного шейдера;
  - **IDirect3DDevice9::SetVertexPixelConstantF** — служит для установки дробного значения константы пиксельного шейдера;
  - **IDirect3DDevice9::SetVertexPixelConstantI** — служит для установки целочисленного значения константы пиксельного шейдера.

Категория *Performance Counters*

Категория **Performance Counters** (Эксплуатационные показатели сборки) собирает общие эксплуатационные характеристики тестируемой программы. Категория **Performance Counters** (Эксплуатационные показатели сборки) включает в себя три списка.

Список **Common In-process counters** (Общие характеристики процесса) необходим для сборки информации в ходе процесса работы программы. Этот список содержит шесть следующих параметров сборки:

- **% Processor Time** — процент загруженности процессора;
- **Virtual Bytes Allocated** — количество виртуально распределяемых байт;
- **Thread Count** — счетчик потоков;
- **Handle Count** — счетчик дескрипторов;
- **IO Read Bytes/sec** — количество прочтенных байт в секунду;
- **IO Write Bytes/sec** — количество записанных байт в секунду.

Список **Common System-wide counters** (Общие системные параметры сборки) измеряет общую системную метрику тестируемой программы и включает в себя пять параметров сборки:

- **% Processor Time** — процент загруженности процессора;
- **Page Faults/sec** — количество дефектов;
- **Available Bytes** — доступные байты;
- **Total Threads** — системные потоки;
- **% Disk Time** — временной процент обращения к физическому диску.

Третий и последний список **All Performance Counters** (Все эксплуатационные показатели) содержит самое большое количество параметров сборки, предоставляющих полную информацию обо всех эксплуатационных характеристиках тестируемой программы. Самое интересное заключается в том, что с помощью списка **All Performance Counters** (Все эксплуатационные показатели) в PIX for Windows вы можете профилировать различные виды программ, а не только DirectX-приложения. На рис. 5.8 вы можете увидеть небольшую

часть из общего количества параметров сборки списка **All Performance Counters** (Все эксплуатационные показатели).

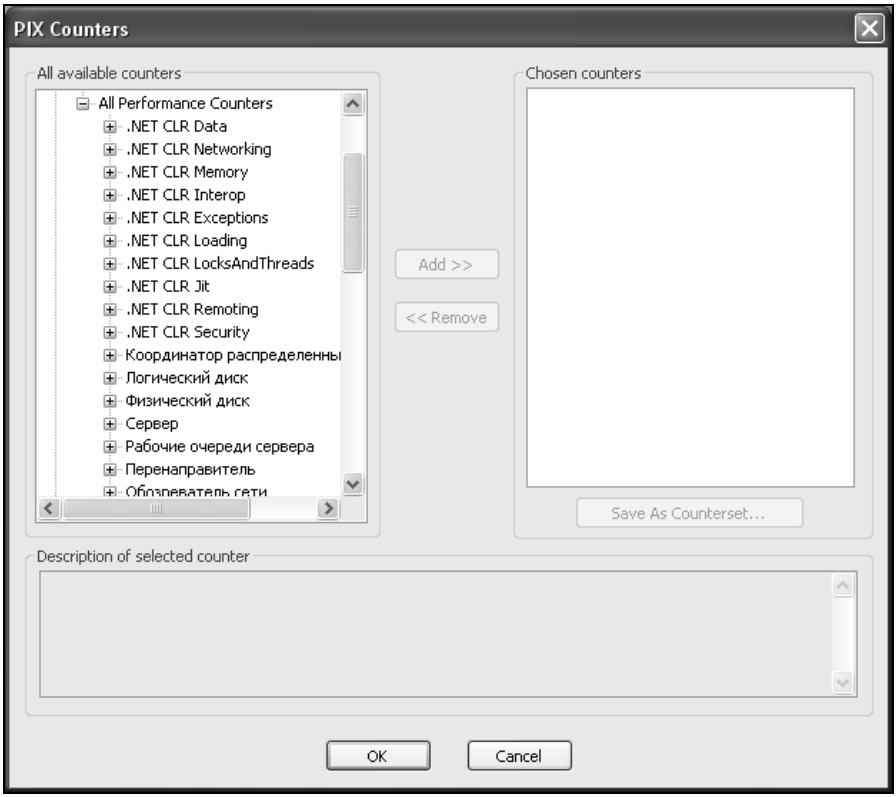


Рис. 5.8. Список **All Performance Counters**

Список **All Performance Counters** (Все эксплуатационные показатели) содержит ряд вложенных в себя списков, каждый из которых имеет в своем распоряжении множество разнообразнейших параметров сборки, как говорится, на все случаи жизни. Давайте, для удобства рассмотрения, определим каждый вложенный список под заголовком, а имеющиеся параметры сборки для каждого списка выполним в виде перечислений с исчерпывающим объяснением каждого параметра.

NET CLR Data

- **SqlClient: Curent # pooled and nonpooled connections** — текущее число всех подключений, как задействованных, так и незадействованных;
- **SqlClient: Curent # pooled connections** — текущее число подключений, связанных с процессом;

- **SqlClient: Curent # connections pools** — текущее число пулов, связанных с процессом;
- **SqlClient: Peak # pooled connections** — наибольшее число подключений во всех пулах;
- **SqlClient: Total # failed connects** — число неудачных подключений;
- **SqlClient: Total # failed commands** — число неудачных команд.

NET CLR Networking

- **Connections Established** — общее количество подключений всего процесса;
- **Bytes Received** — общее количество байт, полученных в процессе подключения;
- **Bytes Sent** — общее количество байт, посланных в процессе подключения;
- **Datagram Received** — общее количество датаграмм, полученных в процессе подключения;
- **Datagram Sent** — общее количество датаграмм, посланных в процессе подключения.

NET CLR Exceptions

- **# of Exceps Throw** — общее количество исключений;
- **# of Exceps Throw/sec** — количество исключений в секунду;
- **# of Filters/sec** — число выполненных фильтров исключений в секунду;
- **# of Finallys/sec** — количество конечных блоков в секунду;
- **Throw To Catch Depth/sec** — значение счетчика обработчика особых ситуаций.

NET CLR Loading

- **Current Classes Loaded** — количество загруженных на данный момент классов;
- **Total Classes Loaded** — общее количество загруженных классов;
- **Rate of Classes Loaded** — количество загруженных классов в секунду;
- **Current appdomains** — номер прикладной области в приложении;
- **Total Appdomains** — наибольший номер прикладной области в приложении;
- **Rate of appdomains** — номер загруженной прикладной области в секунду;
- **Current Assemblies** — количество трансляций, загруженных в текущем приложении;

- **Total Assemblies** — общее количество трансляций, загруженных в текущем приложении;
- **Rate of Assemblies** — количество трансляций, загруженных в секунду;
- **% Time Loading** — параметр зарезервирован для будущего использования;
- **Assembly Search Length** — параметр зарезервирован для будущего использования;
- **Total # of Load Failures** — максимальное количество незагруженных классов;
- **Rate of Load Failures** — количество незагруженных классов в секунду;
- **Bytes in Loader Heap** — текущий размер физической памяти в байтах;
- **Total appdomains unloaded** — общее количество выгруженных прикладных областей;
- **Rate of appdomains unloaded** — количество выгруженных прикладных областей в секунду.

NET CLR LocksAndThreads

- **Total # of Contentions** — общее количество незаблокированных потоков;
- **Contention Rate/sec** — скорость неудачного блокирования потоков;
- **Current Queue Length** — количество текущих ожидающих блокировки потоков;
- **Queue Length Pea** — общее количество ожидающих блокировки потоков;
- **Queue Length/sec** — количество текущих ожидающих блокировки потоков в секунду;
- **# of current logical Threads** — количество текущих объектов потока;
- **# of current physical Threads** — количество созданных потоков;
- **# of current recognized threads** — текущее количество потоков с объектами;
- **# of total recognized threads** — общее количество потоков с объектами;
- **rate of recognized threads/sec** — количество потоков с объектами в секунду.

NET CLR Jit

- **# of Methods Jitted** — общее количество откомпилированных методов;
- **# of IL Bytes Jitted** — количество откомпилированных байтов;
- **Total # of Bytes Jitted** — количество откомпилированных байтов;
- **IL Bytes Jitted/sec** — скорость компилирования байтов в секунду;

- **Standart Jit Failures** — максимальное число неоткомпилированных методов;
- **% Time in Jit** — процент времени компиляции.

.NET CLR Remoting

- **Remote Calls/sec** — количество запросов на объект в секунду;
- **Total Remote Calls** — общее количество запросов на объект;
- **Channels** — общее количество каналов сообщений;
- **Context Proxies** — общее количество созданных объектов;
- **Context-Bound Classes Loaded** — общее число загруженных классов;
- **Context-Bound Objects Alloc/sec** — количество классов, распределенных в секунду;
- **Contexts** — количество контекстов в приложении.

.NET CLR Security

- **Total Runtime Checks** — количество проверок кода доступа;
- **% Time Sig. Authenticating** — параметр зарезервирован для будущего использования;
- **# Link Time Checks** — общее количество проверок времени в приложении;
- **% Time in RT checks** — процент времени, потраченный на проверки кода доступа;
- **Stack Walk Depth** — глубина стека в последнюю проверку кода доступа.

Координатор распределенных транзакций

- **Активные транзакции** — активное число транзакций на данный момент;
- **Завершенные транзакции** — количество уже завершенных транзакций;
- **Прерванные транзакции** — количество всех прерванных транзакций;
- **Транзакции под сомнением** — количество транзакций, подвергающихся сомнению;
- **Максимум активных транзакций** — число одновременно запущенных транзакций за все время;
- **Принудительно завершенные транзакции** — количество транзакций, принудительно завершенных системным администратором;
- **Принудительно прерванные транзакции** — количество транзакций, принудительно прерванных системным администратором;
- **Время отклика (минимум)** — минимальный промежуток времени между начальной и конечной транзакцией;

- **Время отклика (среднее)** — средний промежуток времени между начальной и конечной транзакцией;
- **Время отклика (максимум)** — максимальный промежуток времени между начальной и конечной транзакцией;
- **Транзакций/с** — количество выполненных транзакций за одну секунду;
- **Завершено транзакций/с** — количество завершенных транзакций за одну секунду;
- **Прервано транзакций/с** — количество прерванных транзакций за одну секунду.

Логический диск

- **% свободного места** — свободное место на логическом диске, отраженное в процентах по отношению ко всему объему логического диска;
- **Свободно мегабайт** — количество свободных мегабайтов на логическом диске;
- **Текущая длина очереди диска** — текущая длина очереди к диску;
- **% активности диска** — показывает процентно-временную активность диска;
- **Средняя длина очереди диска** — среднее значение всей длины очереди к диску;
- **% активности диска при чтении** — процент потраченного времени на обработку запросов для чтения;
- **Средняя длина очереди чтения диска** — среднее значение имеющихся запросов на чтение;
- **% активности диска при записи** — процент потраченного времени на обработку запросов для записи;
- **Средняя длина очереди записи на диск** — среднее значение имеющихся запросов на запись;
- **Среднее время обращения к диску (сек)** — среднее время в секундах, необходимое на один обмен данных с диском;
- **Среднее время чтения с диска (сек)** — среднее время в секундах, необходимое для одной операции чтения данных с диска;
- **Среднее время записи на диск (сек)** — среднее время в секундах, необходимое для одной операции записи данных на диск;
- **Обращений к диску/сек** — время обращения к диску в секундах как для записи, так и для чтения данных;
- **Обращений чтения с диска/сек** — время обращения к диску в секундах для чтения данных;

- **Обращений записи на диск/сек** — время обращения к диску в секундах для записи данных;
- **Скорость обмена с диском (байт/сек)** — скорость чтения и записи данных с диска;
- **Скорость чтения с диска (байт/сек)** — скорость чтения данных с диска;
- **Скорость записи на диск (байт/сек)** — скорость записи данных на диск;
- **Средний размер одного обмена с диском (байт)** — средний размер в байтах, полученных при операции чтения и записи данных;
- **Средний размер одного чтения с диска (байт)** — средний размер в байтах, полученных при операции чтения данных с диска;
- **Средний размер одной записи на диск (байт)** — средний размер в байтах, полученных при операции записи данных на диск;
- **Процент времени бездействия** — временной процент бездействия диска;
- **Расщеплений ввода-вывода/сек** — счетчик расщеплений данных.

Физический диск

- **Текущая длина очереди диска** — текущая длина очереди к диску;
- **% активности диска** — показывает процентно-временную активность диска;
- **Средняя длина очереди диска** — среднее значение всей длины очереди к диску;
- **% активности диска при чтении** — процент потраченного времени на обработку запросов для чтения;
- **Средняя длина очереди чтения диска** — среднее значение имеющихся запросов на чтение;
- **% активности диска при записи** — процент потраченного времени на обработку запросов для записи;
- **Средняя длина очереди записи на диск** — среднее значение имеющихся запросов на запись;
- **Среднее время обращения к диску (сек)** — среднее время в секундах, необходимое на один обмен данных с диском;
- **Среднее время чтения с диска (сек)** — среднее время в секундах, необходимое для одной операции чтения данных с диска;
- **Среднее время записи на диск (сек)** — среднее время в секундах, необходимое для одной операции записи данных на диск;
- **Обращений к диску/сек** — время обращения к диску в секундах как для записи, так и для чтения данных;

- **Обращений чтения с диска/сек** — время обращения к диску в секундах для чтения данных;
- **Обращений записи на диск/сек** — время обращения к диску в секундах для записи данных;
- **Скорость обмена с диском (байт/сек)** — скорость чтения и записи данных с диска;
- **Скорость чтения с диска (байт/сек)** — скорость чтения данных с диска;
- **Скорость записи на диск (байт/сек)** — скорость записи данных на диск;
- **Средний размер одного обмена с диском (байт)** — средний размер в байтах, полученных при операции чтения и записи данных;
- **Средний размер одного чтения с диска (байт)** — средний размер в байтах, полученных при операции чтения данных с диска;
- **Средний размер одной записи на диск (байт)** — средний размер в байтах, полученных при операции записи данных на диск;
- **Процент времени бездействия** — временной процент бездействия диска;
- **Расщеплений ввода-вывода/сек** — счетчик расщеплений данных.

Сервер

- **Всего байт/сек** — общее количество полученных и отправленных байтов сервером через сеть;
- **Получено байт/сек** — общее количество полученных байтов сервером через сеть;
- **Передано байт/сек** — общее количество отправленных байтов сервером через сеть;
- **Сеансов, закрытых по тайм-ауту** — количество сеансов сервера, закрытых автоматически на основании времени бездействия, превышающего заданный тайм-аут;
- **Сеансов, закрытых из-за возникновения ошибки** — количество сеансов сервера, закрытых автоматически на основании возникновения ошибок или времени бездействия, превышающего заданный тайм-аут;
- **Сеансов, завершенных нормально** — определяет количество качественно завершенных сеансов;
- **Сеансов, завершенных принудительно** — определяет количество принудительно завершенных сеансов;
- **Ошибок входа** — показывает количество ошибок входа на сервер;
- **Ошибок отсутствия права доступа** — определяет количество ошибок входа на сервер на основании отказа в правах доступа;

- **Ошибок предоставленного доступа** — количество попыток несанкционированного доступа на сервер;
- **Ошибок системы** — общее количество ошибок на сервере;
- **Отвергнуто запросов блокирования** — количество нереализованных запросов блокировки;
- **Нехваток рабочих элементов** — показывает количество непринятых данных в период индикации;
- **Всего операций открытия файлов** — общее количество открытых файлов;
- **Открытых файлов** — количество текущих открытых файлов;
- **Сеансов сервера** — показывает текущие открытые сеансы сервера;
- **Операций поиска файлов** — текущее количество операций по поиску файлов;
- **Байт в невыгружаемом страничном пуле** — общее количество невыгруженных байтов из памяти;
- **Отказов невыгружаемого страничного пула** — определяет малое количество памяти системы;
- **Невыгружаемый пул (пик)** — определяет количество необходимой памяти в системе;
- **Байт в выгружаемом страничном пуле** — общее количество выгружаемой памяти;
- **Отказов выгружаемого страничного пула** — число отказов для выделения памяти из выгружаемого страничного пула;
- **Выгружаемый пул (пик)** — максимальное число байтов, выгруженных из пула;
- **Поставленных в очередь контекстных блоков/сек** — частота попадания в очередь контекстных блоков;
- **Операций входа (Logon)/сек** — частота операций по входу на сервер;
- **Всего операций входа (Logon)** — общее количество операций входа на сервер.

Рабочие очереди сервера

- **Длина очереди** — длина очереди сервера;
- **Активных потоков** — число текущих выполняющихся потоков;
- **Доступных потоков** — число доступных потоков для процессора;
- **Доступных рабочих элементов** — число доступных запросов сервера;
- **Заимствованных рабочих элементов** — в том случае, если процессор уже использовал все доступные рабочие элементы, он может позаимствовать свободные элементы другого процессора;

- **Нехваток рабочих элементов** — если на протяжении длительного времени счетчик больше 0, то следует увеличить значение `MaxWorkItems` в системном реестре для службы сервера;
- **Текущих клиентов** — счетчик количества клиентов, обслуживаемых на данный момент;
- **Получено байт/сек** — количество полученных байтов, определяет степень загрузки сервера;
- **Отправлено байт/сек** — количество отправленных байтов;
- **Передано байт/сек** — количество отправленных и полученных байтов;
- **Операций чтения/сек** — частота, с которой сервер осуществляет операции чтения файлов по запросу;
- **Прочитано байт/сек** — количество прочитанных байтов;
- **Операций записи/сек** — частота операций записи;
- **Записано байт/сек** — количество записанных байтов;
- **Всего байт/сек** — скорость считывания или записи данных в файлы;
- **Всего файловых операций/сек** — частота работы сервера по чтению или записи в файлы по запросу клиентов для данного процессора;
- **Поставленных в очередь контекстных блоков/сек** — частота попадания контекстных блоков в очередь FSP данного сервера.

Перенаправитель

- **Всего байт/сек** — скорость обработки данных;
- **Файловых операций с данными/сек** — частота обработки операций с данными;
- **Пакетов/сек** — частота обработки пакетов данных;
- **Получено байт/сек** — скорость поступления данных;
- **Получено пакетов/сек** — частота получения пакетов;
- **Прочитано страничных байт/сек** — скорость чтения данных после ошибок;
- **Прочитано не страничных байт/сек** — скорость считывания данных приложений;
- **Прочитано из кэша байт/сек** — скорость получения приложениями данных из кэш-памяти;
- **Прочитано сетевых байт/сек** — скорость чтения данных из сети;
- **Передано байт/сек** — скорость передачи байтов в сеть;
- **Передано пакетов/сек** — скорость передачи пакетов в сеть;
- **Записано страничных байт/сек** — скорость записи данных;

- **Записано не страничных байт/сек** — скорость записи данных приложений при перенаправлении;
- **Записано в кэш байт/сек** — скорость записи данных в кэш-память;
- **Записано сетевых байт/сек** — скорость записи данных через сеть;
- **Операций чтения файлов/сек** — частота запроса данных;
- **Операций чтения прямого доступа/сек** — частота чтения файла;
- **Чтений пакетов/сек** — частота чтения пакетов;
- **Длинных чтений/сек** — частота чтения данных бóльших, чем буфер сервера;
- **Чтений коротких пакетов/сек** — частота чтения данных меньших, чем буфер сервера;
- **Операций записи файлов/сек** — частота отправки данных;
- **Операций записи прямого доступа/сек** — частота записи файла;
- **Операций записи пакетов/сек** — частота записи пакетов данных через сеть;
- **Длинных операций записи/сек** — частота записи данных бóльших, чем буфер сервера;
- **Операций записи коротких пакетов/сек** — частота записи данных меньших, чем буфер сервера;
- **Отказов на операцию чтения/сек** — частота отказов операций непосредственного чтения;
- **Отказов на операцию записи/сек** — частота отказов операций непосредственной записи;
- **Сетевых ошибок/сек** — частота возникновения ошибок;
- **Сеансов сервера** — количество сеансов сервера;
- **Повторных подключений к серверу** — количество повторных подключений к серверу;
- **Подключений к серверам базового уровня** — подсчитывает количество активных подключений к серверам, использующим базовый протокол MS-Net SMB, включая собственно серверы MS-Net, а также серверы Xenix и Vax;
- **Подключений к Lan Manager 2.0** — подсчитывает количество активных подключений к серверам Lan Manager 2.0, включая серверы LMX;
- **Подключений к Lan Manager 2.1** — подсчитывает количество активных подключений к серверам Lan Manager 2.1, включая серверы LMX;
- **Подключений к Windows NT** — подсчитывает количество активных подключений к компьютерам с Windows 2000 или ранее;

- **Отключений от сервера** — количество отключений от сервера;
- **Зависаний сеансов сервера** — количество прерванных сеансов из-за зависаний сервера;
- **Активных команд** — количество ждущих запросов.

Обозреватель сети

- **Объявлений сервера/сек** — частота объявлений сервера;
- **Объявлений домена/сек** — частота объявлений домена;
- **Всего объявлений/сек** — сумма значений объявлений сервера и объявлений домена;
- **Избирательных пакетов/сек** — частота получения избирательных пакетов;
- **Операций записи в почтовый слот/сек** — частота приема сообщений через почтовый слот;
- **Запросов на список серверов/сек** — частота запросов на список серверов;
- **Нумераций сервером/сек** — частота обработки запросов;
- **Нумераций доменом/сек** — частота обработки запросов;
- **Прочих нумераций/сек** — частота обработки запросов;
- **Всего нумераций/сек** — частота обработки запросов обозревателей;
- **Промахов объявлений серверов** — количество сброшенных объявлений серверов;
- **Промахов датаграмм почтовых слотов** — количество отклоненных датаграмм почтовых слотов;
- **Промахов запросов на список серверов** — количество необработанных запросов на список серверов;
- **Ошибок распределения объявлений сервера/сек** — частота, с которой серверы не обрабатываются в связи с недостатком памяти;
- **Ошибок распределения почтовых слотов** — количество неудавшихся попыток выделить буфер;
- **Ошибок приема в почтовый слот** — количество непринятых сообщений;
- **Ошибок записи в почтовый слот** — количество незаписанных сообщений;
- **Ошибок открытия почтового слота/сек** — частота ошибок подключения к почтовым слотам;
- **Дублирование объявлений гл. обозревателя** — количество выявлений другого обозревателя;
- **Неправильных датаграмм/сек** — частота получения неправильных датаграмм.

Кэш

- **Отображений данных/сек** — частота отображения страниц в кэш;
- **Синхронных отображений данных/сек** — частота отображения страниц в кэш и ожидание их получения;
- **Асинхронных отображений данных/сек** — частота отображения страниц в кэш при не ожидании их получения;
- **% попаданий при отображении данных** — процент отображения страниц в кэш;
- **Фиксаций при отображении данных/сек** — частота отображений страниц в кэш с фиксацией;
- **Фиксаций при чтении/сек** — частота чтения данных в кэш с последующей фиксацией;
- **Синхронных фиксаций при чтении/сек** — частота чтения данных в кэш для подготовки записи на диск с последующей фиксацией;
- **Асинхронных фиксаций при чтении/сек** — частота чтения данных в кэш для подготовки записи на диск с последующей фиксацией;
- **% попаданий фиксации при чтении** — процент фиксаций страниц в кэш;
- **Чтений с копированием/сек** — частота чтения данных из кэш-памяти;
- **Синхронных чтений с копированием/сек** — частота синхронного чтения данных из кэш-памяти;
- **Асинхронных чтений с копированием/сек** — частота асинхронного чтения данных из кэш-памяти;
- **% попаданий при чтении с копированием** — процент операций чтения с копированием из кэш-памяти в буфер;
- **Чтений MDL/сек** — частота чтения данных страниц кэш-памяти с применением дескрипторов памяти;
- **Синхронных чтений MDL/сек** — частота чтения данных страниц кэш-памяти с применением дескрипторов памяти;
- **Асинхронных чтений MDL/сек** — частота чтения асинхронных данных страниц кэш-памяти с применением дескрипторов памяти;
- **% попаданий чтений MDL** — процент запросов на чтение страниц кэш-памяти с применением дескрипторов памяти;
- **Упреждающих чтений/сек** — частота операций чтения с нахождением последовательного доступа к файлам;
- **Быстрых чтений/сек** — частота операций чтения непосредственно из кэш-памяти в обход файловой системы;
- **Синхронных быстрых чтений/сек** — частота операций чтения непосредственно из кэш-памяти в обход файловой системы;

- **Асинхронных быстрых чтений/сек** — частота асинхронных операций чтения непосредственно из кэш-памяти в обход файловой системы;
- **Промахов ресурса быстрого чтения/сек** — частота отказов быстрого чтения;
- **Не осуществленных быстрых чтений/сек** — частота чтения данных с вызовами функций API;
- **Сбросов "ленивой" записи/сек** — частота задействования "ленивой" записи данных на диск;
- **Страниц "ленивой" записи/сек** — частота выполнения "ленивой" записи данных на диск;
- **Сбросов данных/сек** — частота сбросов данных на диск;
- **Страниц сброса данных/сек** — частота сбросов страниц на диск.

Процессор

- **% загруженности процессора** — время процессора для обработки команд;
- **% работы в пользовательском режиме** — процент работы процессора в пользовательском режиме;
- **% работы в привилегированном режиме** — процент работы процессора в привилегированном режиме;
- **Прерываний/сек** — скорость обслуживания аппаратных прерываний;
- **% времени DPC** — процент времени обработки вызовов отложенных процедур;
- **% времени прерываний** — процент времени обслуживания аппаратных прерываний;
- **Поставлено в очередь DPC/сек** — скорость установки отложенных вызовов процедур в очередь;
- **Скорость DPC** — скорость установки отложенных вызовов процедур в очередь между прерываниями таймера системы;
- **Процент времени бездействия** — процент времени простоя процессора;
- **% времени C1** — процент времени простоя процессора, при котором используется питание C1;
- **% времени C2** — процент времени простоя процессора, при котором используется питание C2;
- **% времени C3** — процент времени простоя процессора, при котором используется питание C3;
- **C1 переходов/сек** — скорость перехода процессора в режим простоя, при котором используется питание C1;

- **C2 переходов/сек** — скорость перехода процессора в режим простоя, при котором используется питание C2;
- **C3 переходов/сек** — скорость перехода процессора в режим простоя, при котором используется питание C3.

Память

- **Ошибок страницы/сек** — количество ошибок ссылок на страницы;
- **Доступно байт** — объем памяти в байтах;
- **Байт выделенной виртуальной памяти** — количество выделенной виртуальной памяти;
- **Предел выделенной виртуальной памяти** — рубеж выделенной виртуальной памяти;
- **Запись копий страниц/сек** — число ошибок страницы;
- **Ошибок транзита/сек** — число ошибок в состоянии транзита;
- **Ошибок запроса обнуления/сек** — ошибки страницы, при решении которых требуется страница без каких-либо данных;
- **Обмен страниц в сек** — количество читаемых или записываемых на диск страниц в секунду;
- **Ввод страниц/сек** — число считанных страниц;
- **Чтений страниц/сек** — количество чтений с диска;
- **Вывод страниц/сек** — число измененных записываемых на диск страниц;
- **Байт в выгружаемом страничном пуле** — объем в байтах системной области памяти;
- **Байт в невыгружаемом страничном пуле** — объем в байтах специальной системной области памяти;
- **Операций вывода страниц/сек** — счетчик измененных записываемых на диск страниц;
- **Распределений в выгружаемом страничном пуле** — количество запросов выделения пространства в системной области памяти;
- **Распределений в невыгружаемом страничном пуле** — количество запросов выделения пространства в системной области памяти;
- **Свободных элементов таблицы страниц** — число неиспользуемых компонентов таблицы страниц;
- **Байт кэш-памяти** — общее число байт кэш-памяти;
- **Байт кэш-памяти (пик)** — максимальное число байт кэш-памяти;
- **Байт в резидентном страничном пуле** — количество байтов в страницах страничного пула;

- **Всего байт системного кода** — количество байтов кода операционной системы;
- **Резидентных байт системного кода** — количество байтов общего кода операционной системы;
- **Всего байт системных драйверов** — количество байтов в загружаемых страницах системных драйверов;
- **Резидентных байт системных драйверов** — количество байтов общего кода системных драйверов;
- **Резидентных байт системного кэша** — число байтов системного кэша;
- **% использования выделенной памяти** — процентное соотношение выделенной памяти (Committed Bytes) к границе выделенной памяти (Commit Limit);
- **Доступно КБ** — количество килобайтов доступной памяти;
- **Доступно МБ** — количество мегабайтов доступной памяти.

Объекты

- **Счетчик процессов** — число процессов в момент сбора информации;
- **Счетчик потоков** — число потоков в момент сбора информации;
- **Счетчик событий** — число событий в момент сбора информации;
- **Счетчик семафоров** — число семафоров в момент сбора информации;
- **Счетчик мьютексов** — число мьютексов в момент сбора информации;
- **Счетчик секций** — число секций в момент сбора информации.

Файл подкачки

- **% использования** — процент применения файла подкачки;
- **% использования (пик)** — максимальный процент применения файла подкачки.

Система

- **Операций чтения файлов/сек** — средняя величина всех операций чтения файловой системы;
- **Операций записи файлов/сек** — средняя величина всех операций записи файловой системы;
- **Операций управления файлами/сек** — средняя величина прочих операций файловой системы;
- **Байт чтения файлов/сек** — среднее количество байтов всех операций чтения файловой системы;
- **Байт записи файлов/сек** — среднее количество байтов всех операций записи файловой системы;

- **Байт управления файлами/сек** — среднее количество байтов прочих операций файловой системы;
- **Контекстных переключений/сек** — количество контекстных переключений;
- **Системных вызовов/сек** — частота системных вызовов;
- **Файловых операций с данными/сек** — частота запуска операций чтения и записи;
- **Время работы системы** — количество секунд работы системы;
- **Длина очереди процессора** — число ожидающих потоков;
- **Счетчик процессов** — число процессов в момент сбора информации;
- **Счетчик потоков** — число потоков в момент сбора информации;
- **Исправлений выравнивания/сек** — частота ошибок выравнивания, исправляемых системой;
- **Обработка особых ситуаций/сек** — частота обработки особых ситуаций;
- **Эмуляция плавающей точки/сек** — частота эмуляции плавающей точки, выполняемая системой;
- **% использования квоты реестра** — процент использования системой общей квоты реестра.

Процесс

- **% загруженности процессора** — процент затраченного времени процессором на выполнение действий;
- **% работы в пользовательском режиме** — процент времени, проведенного в пользовательском режиме;
- **% работы в привилегированном режиме** — процент времени, проведенного в привилегированном защищенном режиме;
- **Байт виртуальной памяти (пик)** — максимальное количество байтов виртуальной памяти, использованной за все время процессом;
- **Байт виртуальной памяти** — количество байтов виртуальной памяти, использующейся в настоящее время процессом;
- **Ошибок страницы/сек** — число ошибок ссылок потока на страницы в секунду;
- **Рабочее множество (пик)** — максимальное количество байтов страниц памяти, использованных за все время процессом;
- **Рабочее множество** — количество байтов страниц памяти, использующихся в настоящее время;
- **Байт файла подкачки (пик)** — максимальное количество байтов файла подкачки;

- **Байт файла подкачки** — количество байтов файла подкачки, использующегося в настоящее время;
- **Байт исключительного пользования** — количество выделенных байтов памяти, использующихся персонально;
- **Счетчик потоков** — количество активных потоков;
- **Базовый приоритет** — базовый приоритет данного процесса относительно общего базового приоритета;
- **Прошло времени (сек)** — количество секунд выполнения процесса;
- **Идентификатор процесса** — численный идентификатор процесса;
- **Код (ID) создавшего процесса** — код процесса, создающего данный процесс;
- **Байт в выгружаемом страничном пуле** — запрашиваемое количество байтов памяти для страниц выгружаемого страничного пула;
- **Байт в невыгружаемом страничном пуле** — запрашиваемое количество байтов памяти для страниц невыгружаемого страничного пула;
- **Счетчик дескрипторов** — количество открытых в настоящее время дескрипторов;
- **I/O-операций чтения в сек** — скорость чтения при вводе-выводе;
- **I/O-операций записи в сек** — скорость записи при вводе-выводе;
- **I/O-операций с данными в сек** — скорость чтения и записи при вводе-выводе;
- **I/O-прочих операций в сек** — скорость прочих операций при вводе-выводе;
- **I/O-чтение байт в сек** — скорость чтения байтов при вводе-выводе;
- **I/O-запись байт в сек** — скорость записи байтов при вводе-выводе;
- **I/O-обмен данными, байт в сек** — скорость чтения и записи при вводе-выводе;
- **I/O-прочих байт в сек** — скорость передачи прочих байтов при вводе-выводе.

Поток

- **% загруженности процессора** — процент затраченного времени процессором на выполнение действий;
- **% работы в пользовательском режиме** — процент времени, проведенного в пользовательском режиме;
- **% работы в привилегированном режиме** — процент времени, проведенного в привилегированном защищенном режиме;

- **Контекстных переключений/сек** — количество контекстных переключений;
- **Прошло времени (сек)** — количество секунд выполнения потока;
- **Текущий приоритет** — текущий динамический приоритет потока;
- **Базовый приоритет** — текущий базовый приоритет потока;
- **Начальный адрес** — начальный адрес виртуальной памяти;
- **Состояние потока** — числовое значение текущего состояния потока;
- **Причина состояния ожидания для потока** — числовое значение состояния ожидания потока;
- **Идентификатор процесса** — численный идентификатор процесса;
- **Идентификатор потока** — численный идентификатор потока.

Объект "Задание"

- **Текущий процент времени процессора** — процент времени, затраченный на выполнение кода программы;
- **Текущий % времени в польз. режиме** — процент времени, затраченный на выполнение кода программы в пользовательском режиме;
- **Текущий % времени в режиме ядра** — процент времени, затраченный на выполнение кода программы в режиме ядра;
- **За данный период мсек процессора** — время в миллисекундах, затраченное на выполнение всех процессов;
- **За данный период мсек польз. режима** — время в миллисекундах, затраченное на выполнение всех процессов в пользовательском режиме;
- **За данный период мсек режима ядра** — время в миллисекундах, затраченное на выполнение всех процессов в режиме ядра;
- **Всего мсек процессора** — время в миллисекундах, затраченное на выполнение всех процессов;
- **Всего мсек польз. режима** — время в миллисекундах, затраченное на выполнение всех процессов в пользовательском режиме;
- **Всего мсек режима ядра** — время в миллисекундах, затраченное на выполнение всех процессов в режиме ядра;
- **Обмен страниц в сек** — скорость происхождения ошибок страниц в секунду;
- **Общий счетчик процессов** — общее количество активных и завершенных процессов;
- **Счетчик активных процессов** — количество активных процессов;
- **Счетчик завершенных процессов** — количество завершенных процессов.

Подробно об объекте "Задание"

- **% загруженности процессора** — процент затраченного времени процессором на выполнение действий;
- **% работы в пользовательском режиме** — процент времени, проведенного в пользовательском режиме;
- **% работы в привилегированном режиме** — процент времени, проведенного в привилегированном защищенном режиме;
- **Байт виртуальной памяти (пик)** — максимальное количество байтов виртуальной памяти, использованной за все время процессом;
- **Байт виртуальной памяти** — количество байтов виртуальной памяти, использующейся в настоящее время процессом;
- **Ошибок страницы/сек** — число ошибок ссылок потока на страницы в секунду;
- **Рабочее множество (пик)** — максимальное количество байтов страниц памяти, использованных за все время процессом;
- **Рабочее множество** — количество байтов страниц памяти, использующихся в настоящее время;
- **Байт файла подкачки (пик)** — максимальное количество байтов файла подкачки;
- **Байт файла подкачки** — количество байтов файла подкачки, использующегося в настоящее время;
- **Байт исключительного пользования** — количество выделенных байтов памяти, использующихся персонально;
- **Счетчик потоков** — количество активных потоков;
- **Базовый приоритет** — базовый приоритет данного процесса относительно общего базового приоритета;
- **Прошло времени (сек)** — количество секунд выполнения процесса;
- **Идентификатор процесса** — численный идентификатор процесса;
- **Код (ID) создавшего процесса** — код процесса, создавшего данный процесс;
- **Байт в выгружаемом страничном пуле** — запрашиваемое количество байтов памяти для страниц выгружаемого страничного пула;
- **Байт в невыгружаемом страничном пуле** — запрашиваемое количество байтов памяти для страниц невыгружаемого страничного пула;
- **Счетчик дескрипторов** — количество открытых в настоящее время дескрипторов;
- **I/O-операций чтения в сек** — скорость чтения при вводе-выводе;

- **I/O-операций записи в сек** — скорость записи при вводе-выводе;
- **I/O-операций с данными в сек** — скорость чтения и записи при вводе-выводе;
- **I/O-прочих операций в сек** — скорость прочих операций при вводе-выводе;
- **I/O-чтение байт в сек** — скорость чтения байтов при вводе-выводе;
- **I/O-запись байт в сек** — скорость записи байтов при вводе-выводе;
- **I/O-обмен данными, байт в сек** — скорость чтения и записи при вводе-выводе;
- **I/O-прочих байт в сек** — скорость передачи прочих байтов при вводе-выводе.

Порт RAS

- **Передано байт** — количество переданных байтов;
- **Получено байт** — количество полученных байтов;
- **Кадров передано** — количество переданных кадров;
- **Получено кадров** — количество полученных кадров;
- **Процент сжатия на выходе** — процент сжатия передаваемой информации;
- **Процент сжатия на входе** — процент сжатия получаемой информации;
- **Ошибок CRC** — число ошибок контрольных сумм;
- **Ошибок тайм-аута** — число ошибок тайм-аута;
- **Ошибок переполнения послед. порта** — число ошибок переполнения последовательного порта;
- **Ошибок выравнивания** — число ошибок выравнивания;
- **Ошибок переполнения буфера** — число ошибок переполнения буфера;
- **Всего ошибок** — всего происходящих ошибок;
- **Передано байт/сек** — скорость передаваемых данных в секунду;
- **Получено байт/сек** — скорость принимаемых данных в секунду;
- **Передано кадров/сек** — частота передаваемых кадров в секунду;
- **Получено кадров/сек** — частота получаемых кадров в секунду;
- **Всего ошибок/сек** — частота всех происходящих ошибок.

Всего RAS

- **Передано байт** — количество переданных байтов;
- **Получено байт** — количество полученных байтов;

- **Кадров передано** — количество переданных кадров;
- **Получено кадров** — количество полученных кадров;
- **Процент сжатия на выходе** — процент сжатия передаваемой информации;
- **Процент сжатия на входе** — процент сжатия получаемой информации;
- **Ошибок CRC** — число ошибок контрольных сумм;
- **Ошибок тайм-аута** — число ошибок тайм-аута;
- **Ошибок переполнения послед. порта** — число ошибок переполнения последовательного порта;
- **Ошибок выравнивания** — число ошибок выравнивания;
- **Ошибок переполнения буфера** — число ошибок переполнения буфера;
- **Всего ошибок** — всего происходящих ошибок;
- **Передано байт/сек** — скорость передаваемых данных в секунду;
- **Получено байт/сек** — скорость принимаемых данных в секунду;
- **Передано кадров/сек** — частота передаваемых кадров в секунду;
- **Получено кадров/сек** — частота получаемых кадров в секунду;
- **Всего ошибок/сек** — частота всех происходящих ошибок;
- **Всего подключений** — число подключений удаленного доступа.

Служба RSVP

- **Сетевых интерфейсов** — количество использующихся сетевых интерфейсов;
- **Сетевых сокетов** — количество использующихся сокетов;
- **Таймеров** — количество установленных событий таймера;
- **Сеансов RSVP** — активных сеансов RSVP;
- **Клиенты QoS** — активных приложений с QoS;
- **Отправителей с действующей QoS** — число сообщений, отправленных для отправителей с действующей QoS;
- **Получателей с действующей QoS** — число сообщений, отправленных для получателей с действующей QoS;
- **Ошибок запросов QoS** — количество ошибок запросов приложений с QoS;
- **Ошибок отправки QoS** — количество ошибочных уведомлений;
- **Уведомлений QoS** — количество доставленных уведомлений;
- **Байтов в уведомлениях QoS** — количество байтов в доставленных уведомлениях.

Очередь печати

- **Всего заданий напечатано** — количество напечатанных заданий;
- **Печатаемых байт/сек** — количество печатаемых байтов в секунду;
- **Всего напечатано страниц** — общее количество напечатанных страниц;
- **Заданий** — количество заданий в очереди печати;
- **Ссылок** — количество ссылок на принтер;
- **Максимум ссылок** — максимальное количество ссылок на принтер;
- **Заданий, обрабатываемых диспетчером очереди** — количество заданий, обрабатываемых диспетчером очереди;
- **Максимум заданий, обрабатываемых диспетчером** — максимальное количество заданий, обрабатываемых диспетчером очереди;
- **Ошибок "Отсутствует бумага"** — число ошибок из-за отсутствия бумаги;
- **Ошибок "Принтер не готов"** — число ошибок с пометкой "Принтер не готов";
- **Ошибок заданий** — общее число ошибок;
- **Вызовов нумерации сетевого принтера** — общее число вызовов сетевого принтера;
- **Вызовов добавления сетевого принтера** — общее число остальных вызовов сетевого принтера.

Телефония

- **Линий** — количество линий для обслуживания компьютером;
- **Телефонных устройств** — количество телефонных устройств для обслуживания компьютером;
- **Активных линий** — количество активных линий для обслуживания компьютером;
- **Активных телефонов** — количество активных телефонных устройств для обслуживания компьютером;
- **Исходящих звонков/сек** — частота исходящих звонков в секунду;
- **Входящих звонков/сек** — частота входящих отвеченных звонков в секунду;
- **Приложений-клиентов** — количество используемых приложений;
- **Текущих исходящих звонков** — исходящих звонков в секунду;
- **Текущих входящих звонков** — входящих звонков в секунду.

Сетевой интерфейс

- **Всего байт/сек** — скорость приема и посылки байтов;
- **Пакетов/сек** — частота приема и посылки байтов в секунду;

- **Получено пакетов/сек** — частота получения пакетов в секунду;
- **Отправлено пакетов/сек** — частота отправки пакетов в секунду;
- **Текущая пропускная способность** — текущая пропускная способность битов в секунду;
- **Получено байт/сек** — скорость получения байтов в секунду;
- **Получено одноадресных пакетов/сек** — частота передачи одноадресных пакетов в секунду;
- **Получено неодноадресных пакетов/сек** — частота передачи неодноадресных пакетов в секунду;
- **Отброшено полученных пакетов** — количество полученных отброшенных пакетов;
- **Получено пакетов с ошибками** — количество полученных пакетов с ошибками;
- **Неопознано полученных пакетов** — количество полученных отброшенных неопознанных пакетов;
- **Отправлено байт/сек** — скорость отправки байтов в секунду;
- **Отправлено одноадресных пакетов/сек** — частота отправления одноадресных пакетов в секунду;
- **Отправлено неодноадресных пакетов/сек** — частота отправления неодноадресных пакетов в секунду;
- **Исходящих пакетов отброшено** — количество отброшенных исходящих пакетов;
- **Исходящих пакетов с ошибками** — количество неотправленных пакетов с ошибками;
- **Длина очереди вывода** — количество исходящих пакетов в очереди вывода.

IP

- **Датаграмм/сек** — частота приема и отправки датаграмм;
- **Получено датаграмм/сек** — частота приема датаграмм;
- **Получено датаграмм с ошибками заголовка** — количество полученных отброшенных датаграмм из-за ошибок в заголовке;
- **Получено датаграмм с ошибками адреса** — количество полученных отброшенных датаграмм из-за ошибок в адресе;
- **Переслано датаграмм/сек** — частота получения датаграмм для их дальнейшей пересылки;
- **Получено датаграмм неопознанного потока** — количество полученных отброшенных датаграмм из-за ошибок протокола;

- **Отброшено полученных датаграмм** — количество полученных отброшенных датаграмм;
- **Доставлено полученных датаграмм/сек** — частота доставки датаграмм протоколам;
- **Отправлено датаграмм/сек** — частота отправленных датаграмм в секунду;
- **Исходящих датаграмм отброшено** — количество исходящих отброшенных датаграмм;
- **Исходящих датаграмм с ошибкой "Нет маршрута"** — количество исходящих отброшенных датаграмм из-за ошибки "Нет маршрута";
- **Получено фрагментов/сек** — частота получения фрагментов в секунду для сборки;
- **Собрано фрагментов/сек** — частота сборки фрагментов;
- **Ошибок при сборке фрагментов** — число возникших при сборке ошибок;
- **Фрагментировано датаграмм/сек** — частота фрагментирования датаграмм в секунду;
- **Ошибок при фрагментации** — количество отброшенных датаграмм, подлежащих фрагментации;
- **Создано фрагментов/сек** — частота создания фрагментов в секунду.

ICMP

- **Сообщений/сек** — частота приема-отправки сообщений в секунду;
- **Получено сообщений/сек** — частота приема сообщений в секунду;
- **Ошибок при получении сообщений** — количество ошибок при получении сообщений;
- **Получено сообщений "Dest. Unreachable"** — количество полученных сообщений о недостижимости местонахождения;
- **Получено сообщений "Time Exceeded"** — количество полученных сообщений об истечении времени;
- **Получено сообщений "Parameter Problem"** — количество полученных сообщений об ошибочных параметрах;
- **Получено сообщений "Source Quench"** — количество полученных сообщений о нехватке времени обработки;
- **Получено сообщений о перенаправлении/сек** — частота получения сообщений о перенаправлении;
- **Получено эхо-сообщений/сек** — частота приема эхо-сообщений;
- **Получено эхо-ответов/сек** — частота приема эхо-ответов;
- **Получено запросов на штамп времени/сек** — частота приема сообщений с запросом на штамп времени;

- **Получено ответов штампа времени/сек** — частота приема сообщений с ответом на штамп времени;
- **Получено запросов на маску адреса** — количество принятых сообщений с запросом маски адреса;
- **Получено ответов с маской адреса** — количество принятых сообщений с ответом на запрос маски адреса;
- **Отправлено сообщений/сек** — частота отправки сообщений;
- **Ошибок исходящих сообщений** — количество неотправленных из-за ошибок сообщений;
- **Отправлено сообщений "Destination Unreachable"** — количество отправленных сообщений о недостижимости местонахождения;
- **Отправлено сообщений "Time Exceeded"** — количество отправленных сообщений об истечении времени;
- **Отправлено сообщений "Parameter Problem"** — количество отправленных сообщений об ошибочных параметрах;
- **Отправлено сообщений "Source Quench"** — количество отправленных сообщений о нехватке времени обработки;
- **Отправлено сообщений о перенаправлении** — частота отправки сообщений о перенаправлении;
- **Отправлено эхо-сообщений/сек** — частота отправки эхо-сообщений;
- **Отправлено эхо-ответов/сек** — частота отправки эхо-ответов;
- **Отправлено запросов на штамп времени/сек** — частота отправки сообщений с запросом на штамп времени;
- **Отправлено ответов штампа времени/сек** — частота отправки сообщений с ответом на штамп времени;
- **Отправлено запросов на маску адреса** — количество отправленных сообщений с запросом маски адреса;
- **Отправлено ответов с маской адреса** — количество отправленных сообщений с ответом на запрос маски адреса.

TCP

- **Сегментов/сек** — частота приема и отправления сегментов;
- **Установлено подключений** — число подключений с состоянием ESTABLISHED или CLOSE-WAIT;
- **Активных подключений** — число подключений в состоянии SYN-SENT;
- **Пассивных подключений** — число подключений в состоянии SYN-RCVD;
- **Отказов подключений** — число подключений в состоянии CLOSED и LISTEN;

- **Сбросов подключений** — число переходов подключений в состояние CLOSED;
- **Получено сегментов/сек** — частота получения сегментов;
- **Отправлено сегментов/сек** — частота отправления сегментов;
- **Переотправлено сегментов/сек** — частота переотправления отправленных ранее сегментов.

UDP

- **Датаграмм/сек** — частота отправления или приема датаграмм;
- **Получено датаграмм/сек** — частота доставки датаграмм пользователю протокола UDP;
- **Датаграмм "No Port"/сек** — частота получения датаграмм с отсутствием приложения в местонахождении;
- **Получено датаграмм с другими ошибками** — частота получения недоставленных датаграмм;
- **Отправлено датаграмм/сек** — частота отправки датаграмм.

Сеанс служб терминалов

- **% загруженности процессора** — процент затраченного времени процессором на выполнение действий;
- **% работы в пользовательском режиме** — процент времени, проведенного в пользовательском режиме;
- **% работы в привилегированном режиме** — процент времени, проведенного в привилегированном защищенном режиме;
- **Байт виртуальной памяти (пик)** — максимальное количество байтов виртуальной памяти, использованной за все время процессом;
- **Байт виртуальной памяти** — количество байтов виртуальной памяти, использующейся в настоящее время процессом;
- **Ошибок страницы/сек** — число ошибок ссылок потока на страницы в секунду;
- **Рабочее множество (пик)** — максимальное количество байтов страниц памяти, использованных за все время процессом;
- **Рабочее множество** — количество байтов страниц памяти, использующихся в настоящее время;
- **Байт файла подкачки (пик)** — максимальное количество байтов файла подкачки;
- **Байт файла подкачки** — количество байтов файла подкачки, использующегося в настоящее время;

- **Байт исключительного пользования** — количество выделенных байтов памяти, использующиеся персонально;
- **Счетчик потоков** — количество активных потоков;
- **Байт в выгружаемом страничном пуле** — запрашиваемое количество байтов памяти для страниц выгружаемого страничного пула;
- **Байт в невыгружаемом страничном пуле** — запрашиваемое количество байтов памяти для страниц невыгружаемого страничного пула;
- **Счетчик дескрипторов** — количество открытых дескрипторов;
- **Ввод: Wd-байт** — количество введенных байтов;
- **Ввод: Wd-кадры** — количество введенных кадров;
- **Ввод: WaitForOutBuf** — количество ожиданий доступности буфера отправки;
- **Ввод: кадры** — количество введенных кадров;
- **Ввод: байт** — общее количество введенных байтов;
- **Ввод: байт после сжатия** — количество введенных байтов после сжатия;
- **Ввод: очистка словаря сжатия** — количество очисток словаря для сжатия данных;
- **Ввод: ошибки** — количество ошибок при вводе;
- **Ввод: тайм-ауты** — число тайм-аутов на линии;
- **Ввод: асинхронные ошибки кадров** — число ошибок асинхронной передачи кадров;
- **Ввод: асинхронные перегрузки** — число ошибок асинхронных перегрузок;
- **Ввод: асинхронные переполнения** — число ошибок асинхронного переполнения;
- **Ввод: асинхронные ошибки четности** — количество ошибок четности;
- **Ввод: транспортные ошибки** — количество ошибок транспортного уровня;
- **Вывод: Wd-байт** — количество выведенных байтов;
- **Вывод: Wd-кадры** — количество выведенных кадров;
- **Вывод: WaitForOutBuf** — количество ожиданий доступности буфера отправки;
- **Вывод: кадры** — количество выведенных кадров;
- **Вывод: байт** — общее количество выведенных байтов;
- **Вывод: байт после сжатия** — количество выведенных байтов после сжатия;
- **Вывод: очистка словаря сжатия** — количество очисток словаря для сжатия данных;

- **Выходные ошибки** — количество ошибок при выводе;
- **Вывод: тайм-ауты** — число тайм-аутов на линии;
- **Вывод: асинхронные ошибки кадров** — число ошибок асинхронной передачи кадров;
- **Вывод: асинхронные перегрузки** — число ошибок асинхронных перегрузок;
- **Вывод: асинхронные переполнения** — число ошибок асинхронного переполнения;
- **Вывод: асинхронные ошибки четности** — количество ошибок четности при выводе;
- **Вывод: транспортные ошибки** — количество ошибок транспортного уровня;
- **Всего: Wd-байт** — общее количество переданных байтов;
- **Всего: Wd-кадры** — общее количество переданных кадров;
- **Всего: WaitForOutBuf** — общее количество ожиданий доступности буфера отправки;
- **Всего: кадры** — общее количество переданных кадров;
- **Всего: байт** — общее количество выведенных байтов за весь сеанс;
- **Всего: байт после сжатия** — общее количество переданных байтов после сжатия;
- **Всего: очистка словаря сжатия** — общее количество очисток словаря для сжатия данных;
- **Всего: ошибок** — общее количество всех произошедших ошибок;
- **Всего: тайм-ауты** — общее число тайм-аутов на линии;
- **Всего: асинхронные ошибки кадров** — общее число ошибок асинхронной передачи кадров;
- **Всего: асинхронные перегрузки** — общее число ошибок асинхронных перегрузок;
- **Всего: асинхронные переполнения** — общее число ошибок асинхронного переполнения;
- **Всего: асинхронные ошибки четности** — общее количество ошибок четности при передаче;
- **Всего: транспортные ошибки** — общее количество ошибок транспортного уровня;
- **Всего: считывание из кэш-памяти протоколов** — общее число считываний из кэш-памяти протоколов;

- **Всего: найдено в кэш-памяти протоколов** — общее число объектов в кэш-памяти протоколов;
- **Всего: эффективность кэш-памяти протоколов** — общая эффективность кэш-памяти;
- **Считывание кэш-памяти протоколов (рисунки)** — количество ссылок на кэш-память с рисунками;
- **Найдено в кэш-памяти протоколов (рисунки)** — количество найденных в кэш-памяти рисунков;
- **Эффективность работы кэш-памяти протоколов (рисунки)** — результативность нахождения рисунков в кэш-памяти;
- **Считывание кэш-памяти протоколов (образы)** — количество ссылок на кэш-память с образами;
- **Найдено в кэш-памяти протоколов (образы)** — количество найденных в кэш-памяти образов;
- **Эффективность работы кэш-памяти протоколов (образы)** — результативность нахождения образов в кэш-памяти;
- **Считывание кэш-памяти протоколов (кисти)** — количество ссылок на кэш-память с кистями;
- **Найдено в кэш-памяти протоколов (кисти)** — количество найденных в кэш-памяти кистей;
- **Эффективность работы кэш-памяти протоколов (кисти)** — результативность нахождения кистей в кэш-памяти;
- **Считывание кэш-памяти протоколов (экраны)** — количество ссылок на кэш-память с сохраненными экранами;
- **Найдено в кэш-памяти протоколов (экраны)** — количество найденных в кэш-памяти экранов;
- **Эффективность работы кэш-памяти протоколов (экраны)** — результативность нахождения сохраненных экранов в кэш-памяти;
- **Степень сжатия при входе** — эффективность сжатия входных данных;
- **Эффективность сжатия выводимых данных** — эффективность сжатия выходных данных;
- **Общий коэффициент сжатия** — общая эффективность сжатия данных.

Службы терминалов

- **Всего сеансов** — общее число сеансов;
- **Активных сеансов** — число активных сеансов;
- **Неактивные сеансы** — число неактивных сеансов.

Объекты WMI

- **Классы HiPerf** — высокопроизводительные классы;
- **Действующие HiPerf** — действующие высокопроизводительные классы.

Категория *Plugin Counters*

Профайлер PIX for Windows поддерживает дополнительно подключаемые модули (plug-in), которые могут быть реализованы сторонними разработчиками. Дополнительные модули выполняются в виде скомпонованной библиотеки, обычно это так называемые DLL-файлы, которые подгружаются в программу для обновления или улучшения функциональности программы. Такие модули можно создавать самостоятельно для своих собственных нужд, а можно воспользоваться уже готовыми модулями сторонних производителей. В комплекте DirectX 9 SDK, с которым поставляется PIX for Windows, имеется один подключаемый модуль под названием PIX Sample Plugin (Пример простого подключаемого модуля), созданный корпорацией Microsoft. С помощью этого модуля можно протестировать работу клавиатуры и мыши. Раскройте иерархию категории **Plugin Counters** (Комплект сборки подключаемых модулей) и вам станет доступен подключаемый модуль PIX Sample. Модуль PIX Sample имеет четыре параметра с понятными названиями, надеюсь, что с ними вы разберетесь самостоятельно.

Опции сбора данных

Следующая область **Call data gathering options** (Опции сбора данных) в диалоговом окне **Experiment1** профайлера PIX for Windows, изображенная на рис. 5.9, содержит несколько элементов управления, с помощью которых можно фиксировать данные для заданного количества фреймов.

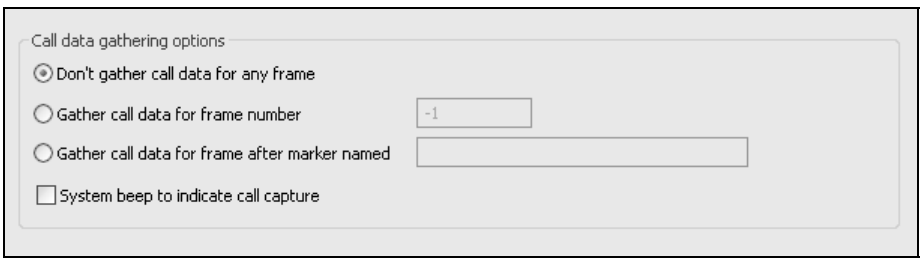


Рис. 5.9. Область **Call data gathering options**

Область **Call data gathering options** (Опции сбора данных) имеет связанную группу из трех переключателей, выбор одного переключателя автоматически исключает выбор двух других.

- **Don't gather call data for any frame** — сбор данных будет происходить для всех фреймов;

- **Gather call data for frame number** — сбор данных произойдет для заданного по номеру фрейма;
- **Gather call data for frame after marker named** — сбор данных произойдет по заданному маркеру.

С помощью набора этих переключателей вы можете задавать определенное количество фреймов или один конкретный фрейм для произведения необходимых замеров. В области **Call data gathering options** (Опции сбора данных) в диалоговом окне **Experiment1** профайлера PIX for Windows есть еще один полезный флажок **System beep to indicate call capture** (Системный гудок) для подачи системного гудка, который означает завершение тестирования программы.

Сохранение файла эксперимента

И последняя область **Data save options** (Опции сохранения данных) в диалоговом окне **Experiment1**, изображенная на рис. 5.10, необходима для указания папки, где будут сохранены итоговые результаты эксперимента.

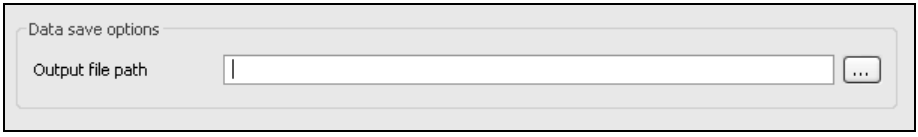


Рис. 5.10. Область **Data save options**

После того как вы произвели нужные настройки для сбора данных в диалоговом окне **Experiment1**, нажмите на кнопку **Start Experiment** (Начало эксперимента), которая находится в самом низу рабочей области диалогового окна **Experiment1**. Если вы не сохраняли файл эксперимента ранее, то на экране появится диалоговое окно, показанное на рис. 5.3, предупреждающие о необходимости сохранения результатов эксперимента в заданном каталоге. Как только будут произведены действия по сохранению файла эксперимента, произойдет запуск тестируемой программы на промежуток времени, заданный вами в области **Target terminaten options** *(см. разд. "Настройки параметров окончания работы программы" этой главы)*. Если временной интервал для сборки данных задан не был, то необходимо произвести самостоятельный выход из цикла работы программы, после чего можно переходить к анализу полученных данных.

Анализ результатов эксперимента

По завершению процесса сборки данных в каталогах, определенных для сохранения результатов эксперимента, появятся два файла: Experiment1.PIXExp и Outpu.PIXRun. Естественно, что в начале тестирования программы вы

можете задавать свои названия файлов, в книге же используются стандартные наименования файлов, предлагаемые PIX for Windows.

Файл Experiment1.PIXExp включает в себя все текущие параметры сборки данных для тестируемой программы, которые были назначены в процессе компоновки файла эксперимента. Открыв файл Experiment1.PIXExp, вы автоматически откроете PIX for Windows и диалоговое окно **Experiment1**. Открыть файл эксперимента из папки можно с помощью двойного щелчка правой кнопки мыши на названии файла, или же непосредственно из профайлера PIX for Windows, выбрав команду **File** | **New** (Файл | Новый).

Второй файл Outpu.PIXRun можно открыть точно таким же способом, как и файл Experiment1.PIXExp. Файл Outpu.PIXRun содержит итоговый результат всего эксперимента, представленный в виде набора графических диаграмм и числовых значений для определенных параметров сборки. На рис. 5.11 изображен профайлер PIX for Windows с открытым файлом Outpu.PIXRun.

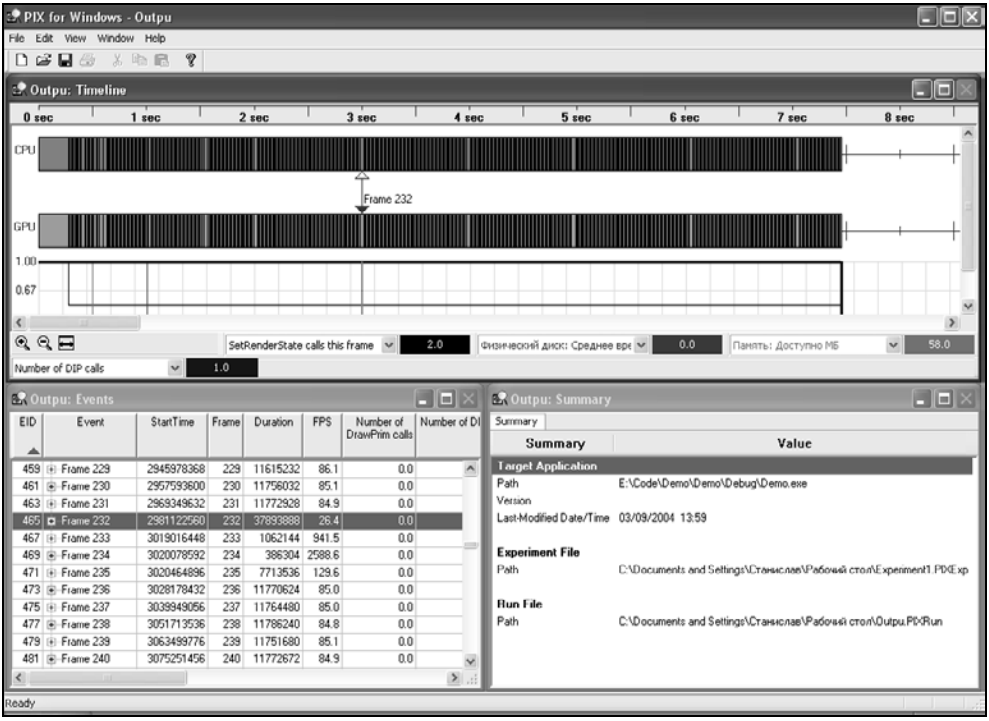


Рис. 5.11. Файл Outpu.PIXRun

Файл Outpu.PIXRun, как видно из рис. 5.11, разделен на три независимых друг от друга окна, это **Timeline** (Отрезок времени), **Events** (События) и **Summary** (Сумма). Все три окна аккуратно делят между собой рабочее про-

странство поверхности PIX for Windows. Рассмотрим каждое окно файл Outpu.PIXRun по отдельности.

Окно *Timeline*

Раскройте окно **Timeline** на всю площадь поверхности PIX for Windows, как показано на рис. 5.12, нажав на кнопку развернуть в верхнем углу окна **Timeline**.

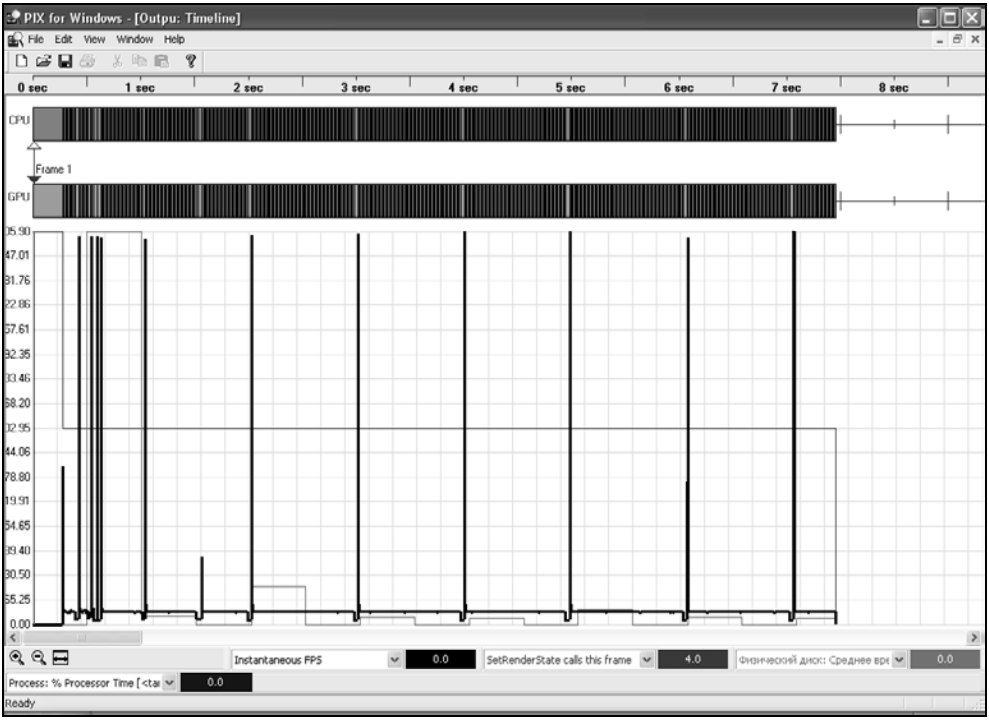


Рис. 5.12. Окно **Timeline**

Окно **Timeline** (Отрезок времени) содержит графические диаграммы замеров для выбранных параметров сборки. Первые две диаграммы вверху окна **Timeline** выполнены в виде двух параллельных друг другу прямоугольных отрезков черного и зеленого цветов. Первый сверху черный прямоугольный отрезок отображает загрузку центрального процессора (CPU), второй зеленый прямоугольный отрезок показывает загрузку графического процессора видеоадаптера (GPU). По своей длине два прямоугольных отрезка одинаковы и равны времени, затраченному на период тестирования программы. В линейке меню PIX for Windows по всей длине прямоугольных отрезков отражается временной интервал работы программы. Из рис. 5.12 можно

увидеть, что на эксперимент было затрачено около семи с половиной секунд времени, но временной цикл тестирования программы может задаваться и менее одной секунды, тогда значение времени будет отображаться в миллисекундах. Между двумя графиками загрузки CPU и GPU находится маркер, выполненный в виде нарисованной двухсторонней стрелки. Этим маркером можно выбирать любой по счету фрейм в пределах тестируемого цикла времени. Для этого нажмите правой кнопкой мыши на маркере и, не отпуская кнопки мыши, перетащите стрелку вдоль двух прямоугольных отрезков, отжав кнопку мыши на необходимом вам месте. В то же мгновение около маркера появится надпись, указывающая на выбранный по счету фрейм, например надпись **Frame 8**. Таким образом можно перемещаться по имеющимся графикам.

Под двумя диаграммами загрузки CPU и GPU находится большая область окна **Timeline** (Отрезок времени), выполненная в виде тетрадного листа в клеточку. В этой области окна **Timeline** графически отображаются некоторые выбранные вами значения параметров сборки. За один раз можно отразить до четырех сборщиков. Четыре необходимых параметра для графического отображения результатов выбираются с помощью четырех раскрывающихся списков, находящихся на панели инструментов в самой нижней части окна **Timeline**, показанной на рис. 5.13.

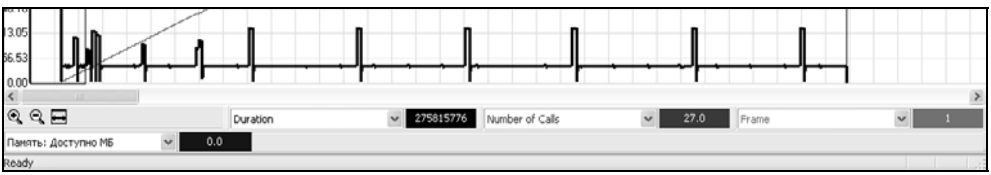


Рис. 5.13. Раскрывающиеся списки окна **Timeline**

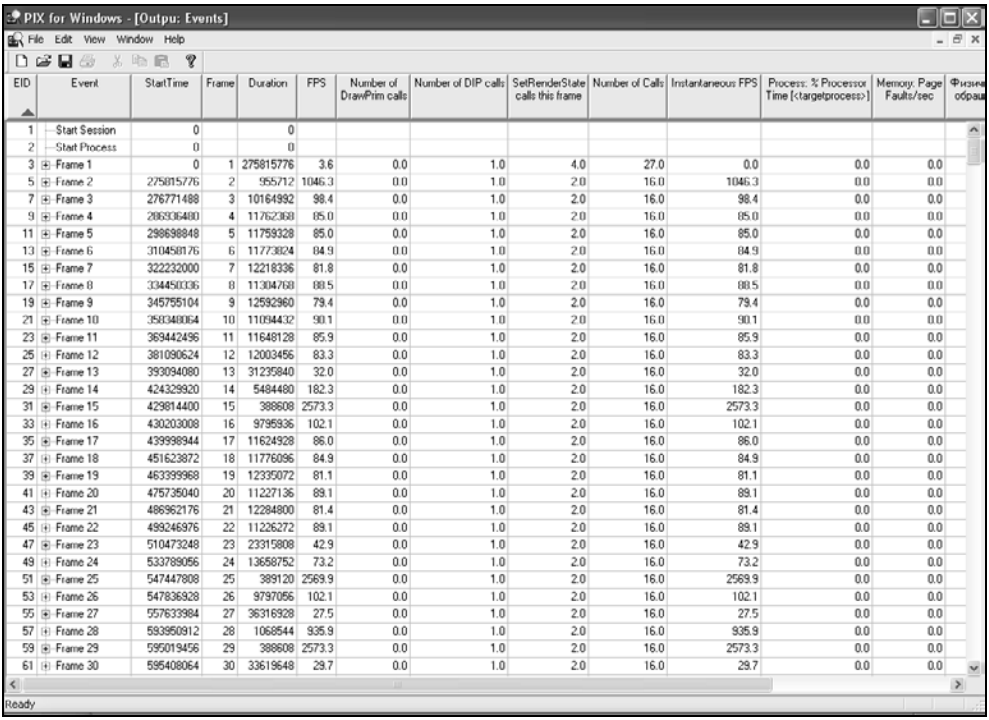
Во всех раскрывающихся списках вам будет доступен набор скомпонованных параметров сборки, который был произведен на этапе конфигурирования файла эксперимента *(см. разд. "Компоновка параметров сборки эксперимента" данной главы)*. Выбирая из раскрывающегося списка параметры, вы автоматически получите графические диаграммы результатов эксперимента.

Также на панели инструментов, где находятся раскрывающиеся списки, имеются три дополнительные пиктограммы. Две первые пиктограммы выполнены в виде лупы со значками плюс и минус, с помощью этих пиктограмм можно соответственно увеличивать или уменьшать графические диаграммы окна **Timeline** (Отрезок времени). Используя третью пиктограмму можно вернуть увеличенный или уменьшенный вид диаграмм к их первоначальному размеру.

Для изучения полученных результатов, выраженных в цифровых значениях, необходимо перейти к окну **Events** (События) файла Outpu.PIXRun.

Окно *Events*

Окно **Events** охватывает все значения результатов произведенных замеров, т. е. содержит список произошедших событий, выполненный в виде таблицы, напоминающей стандартный файл программы Excel. На рис. 5.14 показано окно **Events** (События), развернутое на всю рабочую площадь поверхности PIX for Windows.



EID	Event	StartTime	Frame	Duration	FPS	Number of DrawPrim calls	Number of DIP calls	SetRenderState calls this frame	Number of Calls	Instantaneous FPS	Process: % Processor Time (<targetprocess>)	Memory: Page Faults/sec	Физиче обраш
1	Start Session	0		0									
2	Start Process	0		0									
3	Frame 1	0	1	275815776	3.6	0.0	1.0	4.0	27.0	0.0	0.0	0.0	
5	Frame 2	275815776	2	955712	1046.3	0.0	1.0	2.0	16.0	1046.3	0.0	0.0	
7	Frame 3	276771488	3	10164992	98.4	0.0	1.0	2.0	16.0	98.4	0.0	0.0	
9	Frame 4	286936480	4	11762368	85.0	0.0	1.0	2.0	16.0	85.0	0.0	0.0	
11	Frame 5	298698848	5	11759328	85.0	0.0	1.0	2.0	16.0	85.0	0.0	0.0	
13	Frame 6	310458176	6	11773824	84.9	0.0	1.0	2.0	16.0	84.9	0.0	0.0	
15	Frame 7	322232000	7	12218336	81.8	0.0	1.0	2.0	16.0	81.8	0.0	0.0	
17	Frame 8	334450336	8	11304768	88.5	0.0	1.0	2.0	16.0	88.5	0.0	0.0	
19	Frame 9	345755104	9	12592960	79.4	0.0	1.0	2.0	16.0	79.4	0.0	0.0	
21	Frame 10	358348064	10	11094432	90.1	0.0	1.0	2.0	16.0	90.1	0.0	0.0	
23	Frame 11	369442496	11	11648128	85.9	0.0	1.0	2.0	16.0	85.9	0.0	0.0	
25	Frame 12	381090624	12	12003456	83.3	0.0	1.0	2.0	16.0	83.3	0.0	0.0	
27	Frame 13	393094080	13	31235840	32.0	0.0	1.0	2.0	16.0	32.0	0.0	0.0	
29	Frame 14	424329920	14	5484480	182.3	0.0	1.0	2.0	16.0	182.3	0.0	0.0	
31	Frame 15	429814400	15	388608	2573.3	0.0	1.0	2.0	16.0	2573.3	0.0	0.0	
33	Frame 16	430203008	16	9795936	102.1	0.0	1.0	2.0	16.0	102.1	0.0	0.0	
35	Frame 17	439998944	17	11624928	86.0	0.0	1.0	2.0	16.0	86.0	0.0	0.0	
37	Frame 18	451623872	18	11776096	84.9	0.0	1.0	2.0	16.0	84.9	0.0	0.0	
39	Frame 19	463399968	19	12335072	81.1	0.0	1.0	2.0	16.0	81.1	0.0	0.0	
41	Frame 20	475735040	20	11227136	89.1	0.0	1.0	2.0	16.0	89.1	0.0	0.0	
43	Frame 21	486962176	21	12284800	81.4	0.0	1.0	2.0	16.0	81.4	0.0	0.0	
45	Frame 22	499246976	22	11226272	89.1	0.0	1.0	2.0	16.0	89.1	0.0	0.0	
47	Frame 23	510473248	23	23315808	42.9	0.0	1.0	2.0	16.0	42.9	0.0	0.0	
49	Frame 24	533789056	24	13658752	73.2	0.0	1.0	2.0	16.0	73.2	0.0	0.0	
51	Frame 25	547447808	25	389120	2569.9	0.0	1.0	2.0	16.0	2569.9	0.0	0.0	
53	Frame 26	547836928	26	9797056	102.1	0.0	1.0	2.0	16.0	102.1	0.0	0.0	
55	Frame 27	557633984	27	36316928	27.5	0.0	1.0	2.0	16.0	27.5	0.0	0.0	
57	Frame 28	593950912	28	1068544	935.9	0.0	1.0	2.0	16.0	935.9	0.0	0.0	
59	Frame 29	595019456	29	388608	2573.3	0.0	1.0	2.0	16.0	2573.3	0.0	0.0	
61	Frame 30	595408064	30	33619648	29.7	0.0	1.0	2.0	16.0	29.7	0.0	0.0	

Рис. 5.14. Окно **Events**

В верхней части окна **Events** (События) находится шапка таблицы, где перечислены назначенные вами на этапе компоновки файла эксперимента текущие параметры сборки. Первые шесть колонок таблицы в окне **Events** (События) генерируются PIX for Windows автоматически, вне зависимости от всех последующих параметров сборки, выбранных вами:

- **EID** — код процесса;
- **Event** — события, в том числе старт процесса тестирования, его окончание и количество полученных фреймов. Причем события отображают значения для начала и конца каждого фрейма;
- **StartTime** — начальное и конечное время работы для каждого фрейма;

- **Frame** — последовательное числовое перечисление всех имеющихся фреймов, созданных в процессе тестирования программы. Отсчет начинается с единицы;
- **Duration** — продолжительность каждого фрейма;
- **FPS** — частота кадров.

Все следующие колонки таблицы окна **Events** (События) соответствуют заданным вами параметрам сборки на этапе компоновки файла эксперимента, например, на рис. 5.14 можно увидеть замеры для функций `IDirect3DDevice9::DrawPrimitive` и `IDirect3DDevice9::SetRenderState` и все полученные значения вызовов этих функций даются по каждому фрейму.

В колонке **Event** (События) по возрастающей в каждой из строк показаны все фреймы, созданные во время работы программы. Любая строка содержит порядковый номер отдельно взятого фрейма, например, с надписью **Frame 3**. С правой стороны от надписи находится пиктограмма в виде квадратика с плюсом, щелкнув по ней левой кнопкой мыши, можно открыть дополнительную строку в таблице, которая доступна для каждого фрейма и показывает набор значений сборки для окончания процесса работы выбранного фрейма. Например, если раскрыть фрейм с надписью **Frame 3** в PIX for Windows, то появится дополнительная строка с надписью **End Frame 3**. Переходить по значениям таблицы окна **Events** (События) можно с помощью мыши, щелкнув в районе необходимой строки, или с помощью клавиш <Up> (вверх) и <Down> (вниз), при этом выбранная строка моментально подсвечивается. При переходе по фреймам, т. е. по строкам таблицы в окне **Events** (События), происходит синхронное перемещение маркера в окне **Timeline** (Отрезок времени) на позицию выбранного фрейма в строке таблицы окна **Events** (События). И наоборот, если вы перемещаете маркер в окне **Timeline** (Отрезок времени), происходит автоматический переход по строкам таблицы окна **Events** (События).

Окно *Summary*

Окно **Summary** (Сумма), изображенное на рис. 5.15, содержит суммарную информацию о произведенных настройках по всему эксперименту.

В окне **Summary** (Сумма) имеется три группы со своим небольшим набором параметров:

- **Target Application** — информация о тестируемой программе с помощью следующих параметров:
  - **Path** — путь к тестируемому приложению;
  - **Version** — версия программы;
  - **Last-Modified Date/Time** — последняя дата работы с тестируемой программой.

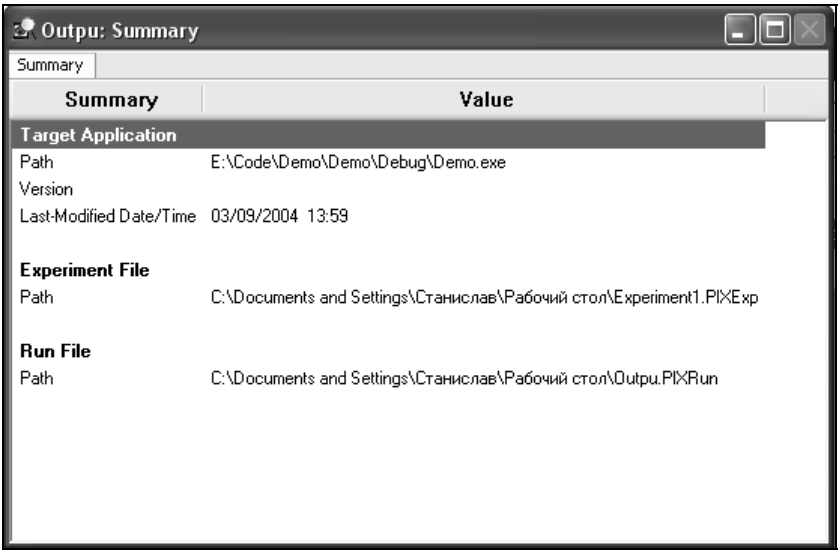
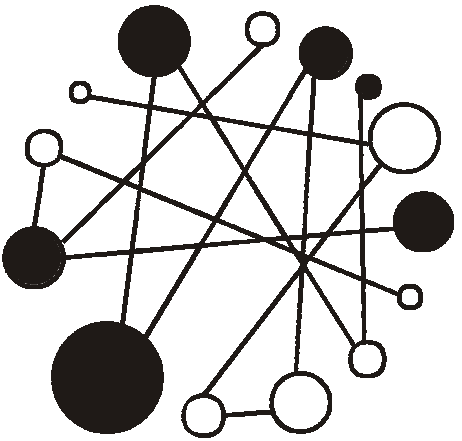


Рис. 5.15. Окно **Summary**

- **Experiment File** — настройки для файла эксперимента с помощью параметра:
  - **Path** — путь к файлу эксперимента.
- **Run File** — настройки для выходного файла эксперимента с помощью параметра:
  - **Path** — путь к выходному файлу эксперимента.

Резюмируя эту главу, можно сказать, что профайлер PIX for Windows отображает как графические, так цифровые значения итогов тестирования программы, значительно упрощающие общее восприятие полученных результатов, и дает возможность целенаправленно изучить имеющиеся ошибки в работе программы. На основе всех полученных результатов для параметров сборки информации можно получить общую картину происходящих процессов внутри тестируемой программы, что значительно ускорит и упростит отладку разрабатываемого программного продукта. Конечно, PIX for Windows — это не панацея от всех бед, и он не в силах сделать за вас всю работу, но со своими обязанностями PIX for Windows справляется отлично!

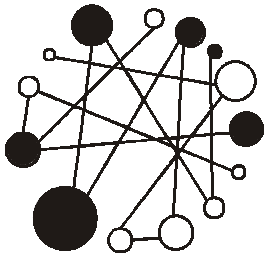
В следующих частях книги вы познакомитесь с инструментальными средствами компаний ATI и NVIDIA, необходимыми для отладки и программирования шейдеров.



ЧАСТЬ III

Инструментарий ATI RenderMonkey

ГЛАВА 6



Знакомство с инструментарием RenderMonkey

"Самый лучший интерфейс — наш интерфейс!"
Наталья Татарчук, лидер-программист проекта RenderMonkey компании ATI Technologies Inc.

Создание приложений с применением вершинных и пиксельных шейдеров является довольно трудоемкой задачей, и если при этом использовать для написания шейдеров, скажем, стандартный блокнот операционной системы Windows, то радости от этого будет немного. Программирование шейдеров в среде Visual Studio .NET отчасти решает проблемы с подсветкой синтаксиса и отладкой шейдеров, но значительно лучше использовать специализированные инструменты, предназначенные именно для работы с программным кодом шейдеров. До недавнего времени такие инструментарии отсутствовали на рынке, и разработчикам приходилось оставаться наедине со своими проблемами. Это и немудрено, потому что вершинные и пиксельные шейдеры не использовались в играх в таких масштабах, как сейчас. Но события изменяются с калейдоскопической быстротой, и уже ни одна более-менее приличная игра не строится без использования шейдеров. Осознав этот факт, компании ATI и NVIDIA выпустили свои абсолютно разные средства для отладки, редактирования и написания программ шейдеров. Эта и последующая главы целиком посвящены инструментарию RenderMonkey 1.5 компании ATI. В них отражена полная информация, начиная с правильной установки и заканчивая созданием потрясающих по мощности спецэффектов. А в *главах 8 и 9* будет рассмотрен IDE FX Composer 1.5 от компании NVIDIA.

Инструментарий RenderMonkey пережил уже несколько релизов и сейчас доступен в версии RenderMonkey 1.5. Также с дистрибутивом поставляется RenderMonkey SDK 1.0, о котором мы поговорим в конце этой главы в разделе RenderMonkey SDK. Отрадно осознавать, что во главе проекта

RenderMonkey стоит наша соотечественница Наталья Татарчук, являющаяся ведущим программистом этого проекта.

По словам разработчиков RenderMonkey, на рынке программных средств, предназначенных для работы с вершинными и пиксельными шейдерами зияют огромные пустоты, и RenderMonkey был создан с целью облегчения работы программистов. Использование пиксельных и вершинных шейдеров в программировании трехмерной графики в ближайшее время станет одной из основных ветвей развития игровой индустрии. Поэтому очень важно иметь специализированные программные средства для работы с шейдерами. Одним из таких средств как раз и является инструментарий RenderMonkey компании ATI.

Инструментарий RenderMonkey дает возможность разработчику при написании и отладке программ шейдеров увидеть результат работы сразу без перекомпиляции всего проекта. Версия инструментария RenderMonkey 1.5 поддерживает работу с высокоуровневым языком программирования шейдеров (HLSL) и языком шейдеров, специально предназначенного для OpenGL (GLSL).

Примечание

Если вы работаете с OpenGL, то при использовании бинормалей и тангенса в вершинном построении необходимо производить следующую запись:

```
attribute vec3 rm_Binormal;
attribute vec3 rm_Tangent;
```

Также необходимо помнить, что в DirectX используется левосторонняя система координат, и такая установка прописана в RenderMonkey по умолчанию, но вы можете изменить параметры в каждом новом проекте через окно свойств (**RenderMonkey Preferences**), адаптируя их под OpenGL.

Интуитивно понятный интерфейс RenderMonkey позволяет без особого "раскачивания" начинать работу над проектом любой сложности, вне зависимости от количества программистов и дизайнеров.

Более того, RenderMonkey впервые позволяет дизайнеру работать с шейдерами самостоятельно. Так как в RenderMonkey предусмотрена возможность совместной работы программиста и дизайнера, то программист может не опасаться, что дизайнер испортит программный код, а дизайнер может самостоятельно создавать качественные эффекты. Впервые программисты и дизайнеры имеют такое общее мощное средство для программирования и отладки шейдеров.

Версия RenderMonkey 1.5 поддерживает видеоадаптеры компании NVIDIA, что, признаться, весьма специфично для конкурирующих компаний. С другой стороны, если компания ATI действительно желает занять доминирующее положение в этом сегменте рынка, то это достаточно обдуманный и объяснимый ход. Так или иначе, разработчики только выигрывают.

В этой и следующей главах вы узнаете обо всех нюансах использования RenderMonkey, а начнем мы с установки дистрибутива RenderMonkey 1.5 на вашу компьютерную систему.

Установка RenderMonkey

На компакт-диске, поставляемом в месте с книгой, в папке \ATI находится установочный файл инструментария RenderMonkey с названием RenderMonkey 2004-08-03-v1.5.424. Это последняя версия продукта RenderMonkey на момент написания книги.

Двойной щелчок левой кнопкой мыши на установочном файле RenderMonkey вызовет мастер установки. Первое диалоговое окно **Welcome to the RenderMonkey 1.5 Installation Wizard**, изображенное на рис. 6.1, встретит вас радостным приветствием.

Рис. 6.1. Окно **Welcome to the RenderMonkey 1.5 Installation Wizard**

В диалоговом окне **Welcome to the RenderMonkey 1.5 Installation Wizard** вы также найдете напоминания о необходимости установки последней версии DirectX 9 SDK Update (Summer 2004) перед инсталляцией инструментария RenderMonkey 1.5, а также вам понадобится порядка 170 Мбайт свободного места на жестком диске. Нажав кнопку **Next** в диалоговом окне **Welcome to the RenderMonkey 1.5 Installation Wizard**, вы перейдете на следующий этап инсталляции RenderMonkey. Появившееся диалоговое окно **License Agreement**

содержит лицензионное соглашение, если вы согласны с условиями лицензии, то выберите переключатель **I accept license agreement** и нажмите на кнопку **Next**. В появившемся новом диалоговом окне **Destination Folder** (Место расположения папки), показанном на рис. 6.2, предлагается выбрать каталог для установки RenderMonkey.

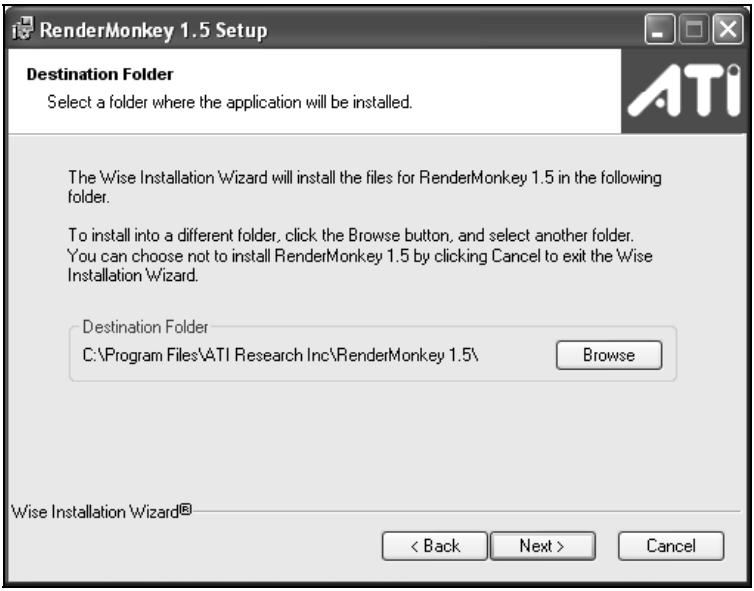


Рис. 6.2. Диалоговое окно **Destination Folder**

Оставьте каталог, предлагаемый мастером установки по умолчанию, либо воспользуйтесь кнопкой **Browse** (Обзор) в окне **Destination Folder** для выбора своего каталога, после чего нажмите кнопку **Next** для продолжения установки RenderMonkey. В новом появившемся диалоговом окне **Select Installation Type** (Выбор типа инсталляции), представленном на рис. 6.3, вам будет предложено на выбор три типа последующей установки программы.

Этот стандартный набор типов, как правило, доступен при установке почти любой программы. Тип установки **Typical** (Типичный) производит инсталляцию типового набора компонентов, необходимых для работы. Следующий вариант установки **Complete** (Комплект) включает в себя полную инсталляцию всех имеющихся в комплекте установки дополнений, обычно лучше придерживаться этого варианта комплектации. И последний тип установки **Custom** (Выборочный) дает возможность произвести выборочную установку поставляемых компонентов инструментария RenderMonkey. Не знаю как вы, а я всегда не могу удержаться от соблазна заглянуть в набор комплектации любой программы. Вот и сейчас давайте изберем тип установки **Custom** (Выборочный) воспользовавшись переключателем, находящимся напротив

соответствующей надписи, и нажмем на кнопку **Next**. В новом появившемся окне **Select Features**, изображенном на рис. 6.4, можно ознакомиться с полным комплектом поставки RenderMonkey.

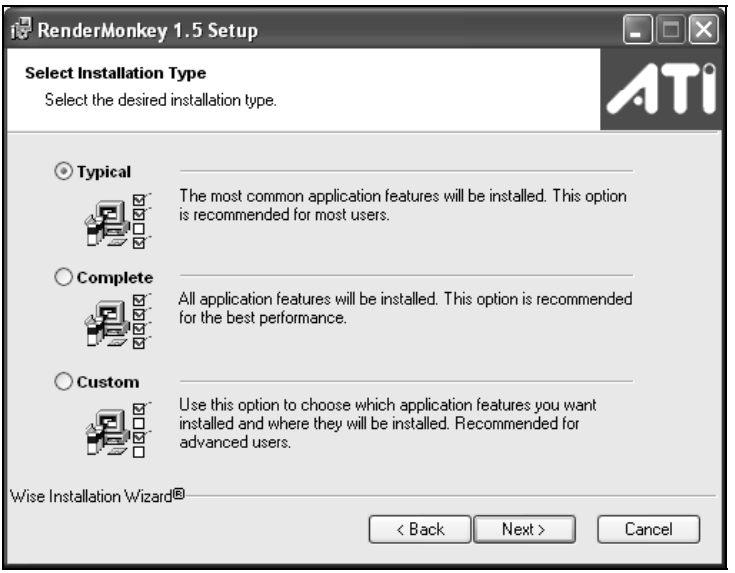


Рис. 6.3. Диалоговое окно **Select Installation Type**

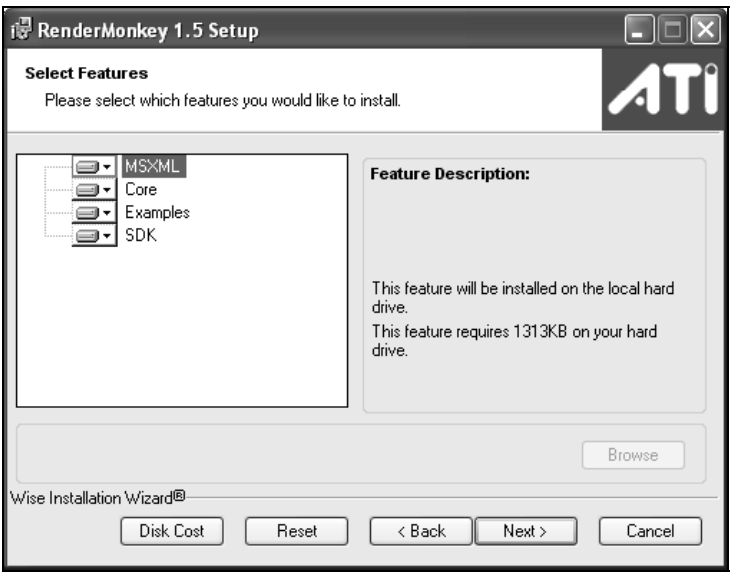


Рис. 6.4. Диалоговое окно **Select Features**

Нажав на кнопку **Next** в диалоговом окне **Select Features**, вы запустите процесс инсталляции RenderMonkey на ваш компьютер. По окончании установки появится последнее диалоговое окно, оповещающие о завершении инсталляции RenderMonkey. В этом окне нажмите на кнопку **Finish**, и таким образом инструментарий RenderMonkey будет благополучно установлен в вашей системе.

Изучаем RenderMonkey

После установки в меню **Start | All programs** (Пуск | Все программы) появится сформированное меню с общим названием **ATI Research Inc**, в котором вы найдете следующие команды:

- **DX9 Sample Workspaces** — открывает доступ к примерам для DirectX 9;
- **GL2 Sample Workspaces** — открывает доступ к примерам для OpenGL 2;
- **Release Note** — информация о релизе RenderMonkey 1.5;
- **RenderMonkey 1.5** — запускает интегрированную среду разработки приложений RenderMonkey;
- **RenderMonkey Documentation** — документация по RenderMonkey в формате PDF;
- **RenderMonkey SDK Documentation** — документация по SDK RenderMonkey в формате PDF;
- **RenderMonkey SDK Help** — контекстная справка, выполненная в виде HTML-страниц;
- **Uninstall RenderMonkey 1.5** — удаление RenderMonkey 1.5.

Для того чтобы запустить инструментарий RenderMonkey, выберите в меню **Start | All programs | ATI Research Inc | RenderMonkey 1.5 | RenderMonkey 1.5** (ПУСК | Все программы | ATI Research Inc | RenderMonkey 1.5 | RenderMonkey 1.5) или воспользуйтесь пиктограммой, автоматически помещенной на рабочем столе Windows при инсталляции и выполненной в виде лица симпатичной обезьянки желтого цвета с надписью RenderMonkey 1.5. После запуска программы вам откроется рабочее окно RenderMonkey, представленное на рис. 6.5.

На рис. 6.5 видно, что инструментарий RenderMonkey выполнен в лучших традициях подобных визуальных средств программирования с интуитивно понятным интерфейсом. Рабочая площадь RenderMonkey разделена на три различных по размеру окна, это **Workspace** (Рабочее пространство), **Output** (Окно вывода выходной информации) и **Preview** (Окно предварительного просмотра). При этом окно **Preview** при начальном старте RenderMonkey не открывается, а пространство, отведенное для этого окна, остается пустым. Также еще имеется ряд дополнительно появляющихся окон, которые стано-

вятся доступны по мере работы с инструментарием. В верхней части RenderMonkey расположена инструментальная панель с кнопками быстрого доступа и линейкой меню с набором стандартных команд **File** (Файл), **Edit** (Редактировать), **View** (Вид), **Window** (Окно) и **Help** (Помощь). Давайте рассмотрим меню и инструментальную панель RenderMonkey.

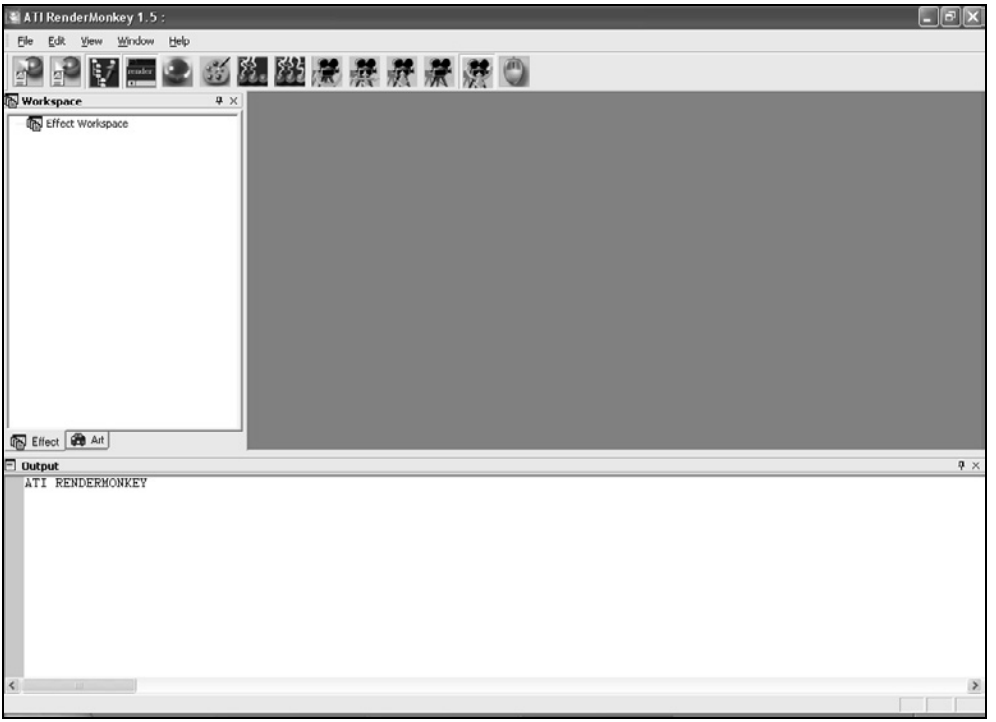


Рис. 6.5. Рабочее окно инструментария RenderMonkey

Меню *File*

В меню **File**, его можно увидеть на рис. 6.6, содержится пять стандартных команд по открытию, закрытию или сохранению рабочих файлов.

Меню **File** содержит следующие команды:

- **New** (<Ctrl>+<N>) — открывает новое рабочее пространство, в том числе окно **Workspace** с новым **Effect Workspace** (Рабочее пространство эффекта). По умолчанию это пустое окно. Также при открытии нового рабочего пространства либо при закрытии RenderMonkey появится диалоговое окно с предложением сохранить текущий проект;
- **Open** (<Ctrl>+<O>) — эта команда открывает существующий файл **Effect Workspace**;

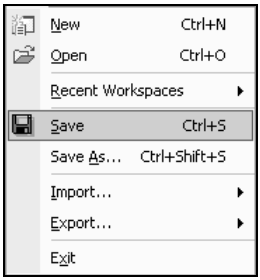


Рис. 6.6. Меню File

- **Recent Workspaces** — эта команда имеет свое выпадающие меню, где содержится пять последних проектов;
- **Save** (<Ctrl>+<S>) — сохраняет текущее открытое окно **Workspace**;
- **Save As** (<Ctrl>+<Shift>+<S>) — сохраняет текущее окно **Workspace** в заданной директории;
- **Import** — эта команда позволяет импортировать разработчикам дополнительные модули своего собственного формата;
- **Export** — с помощью команды экспорта разработчику предоставляется возможность произвести экспорт файла в свой формат данных, в частности есть возможность экспортировать файл в FX-формат, используемый в DirectX 9;
- **Exit** — выход из RenderMonkey.

Меню *Edit*

Меню **Edit** содержит восемь команд, оно изображено на рис. 6.7. С помощью команд меню **Edit** можно производить различные операции по редактированию или правке имеющихся данных.

Меню **Edit** содержит следующие команды:

- **Undo** (<Ctrl>+<Z>) — совершает отмену последней произведенной операции;
- **Redo** (<Ctrl>+<Y>) — возврат к предыдущей операции;

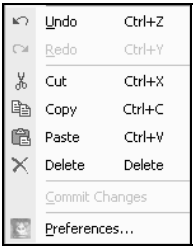


Рис. 6.7. Меню Edit

- **Cut** (<Ctrl>+<X>) — служит для вырезания выделенного узла из **Effect Workspace**;<
- **Copy** (<Ctrl>+<C>) — копирует узел из **Effect Workspace**;
- **Paste** (<Ctrl>+<V>) — вставка узла в **Effect Workspace**;
- **Delete** (Delete) — удаляет узел в **Effect Workspace**;
- **Commit Changes** (<F7>) — компилирует активный шейдер;
- **Preferences** — с помощью этой команды можно открыть диалоговое окно свойств (RenderMonkey Preferences), изображенное на рис. 6.8.

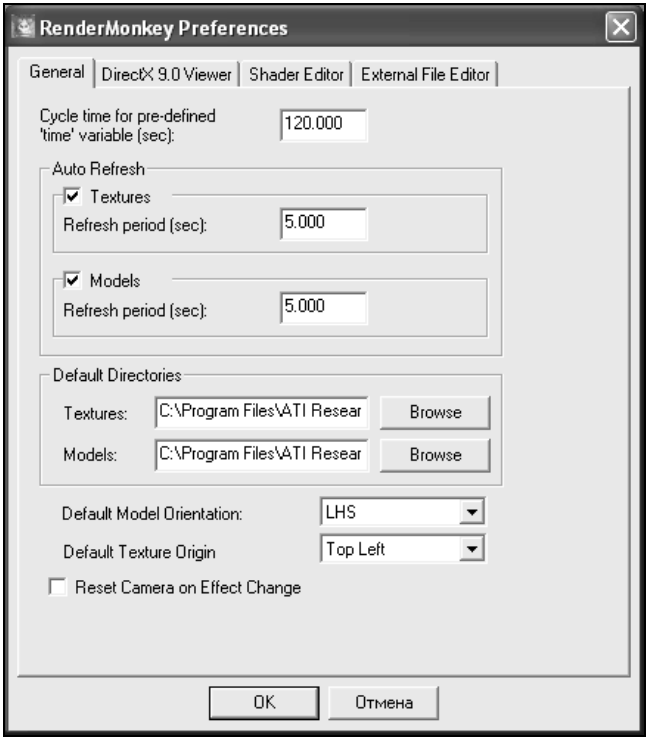


Рис. 6.8. Диалоговое окно **RenderMonkey Preferences**

В диалоговом окне **RenderMonkey Preferences** находится четыре вкладки: **General** (Общие), **DirectX 9.0 Viewer** (Параметры для DirectX 9.0), **Shader Editor** (Редактор шейдеров) и **External File Editor** (Внешний редактор файлов). Каждая вкладка имеет свой набор элементов управления, с помощью которых можно задавать различные параметры настройки работы в среде RenderMonkey, рассмотрим подробно каждую вкладку диалогового окна **RenderMonkey Preferences**.

Вкладка *General*

Вкладка **General** (Основные) диалогового окна **RenderMonkey Preferences**, изображенная на рис. 6.8, позволяет задавать основные опции для работы с RenderMonkey. В верхней части вкладки **General** располагается текстовое поле **Cycle time for pre-defined 'time' variable** (Время цикла для предопределенной переменной) с установленным по умолчанию значением 120 секунд. С помощью этого текстового поля можно задавать время цикла для предопределенных переменных, необходимых для удобства работы с шейдерами.

Следующий набор элементов управления ограничен статической областью под названием **Auto Refresh** (Автоматическая регенерация). В этой области находится два флажка: **Textures** (Текстуры) и **Models** (Модели). Напротив каждого из флажков имеется текстовое поле для задания цифровых значений. С помощью этих опций можно задавать время для автоматического обновления изменившихся ресурсов (время по умолчанию задано значением 5 секунд).

Ниже статической области **Auto Refresh** вкладки **General** диалогового окна **RenderMonkey Preferences** расположена еще одна статическая область **Default Directories** (Директория по умолчанию) с двумя текстовыми полями, указывающими на расположение каталога с ресурсами, а именно текстурами и моделями, используемыми в работе RenderMonkey.

Далее на вкладке **General** располагаются два списка: **Default Model Orientation** (Система координат для модели по умолчанию) и **Default Texture Origin** (Текстурные координаты по умолчанию). В списке **Default Model Orientation** можно выбирать между левосторонней системой координат (**LHS**), используемой в DirectX, и правосторонней системой координат (**RHS**) для представления модели на экране монитора при работе с OpenGL. Список **Default Texture Origin** предназначен для определения текстурных координат.

И последняя опция на вкладке **General** — флажок **Reset Camera on Effect Change** (Сброс камеры по изменению эффекта), служащий для автоматического сброса позиции камеры в момент изменения текущего эффекта.

Вкладка *DirectX 9.0 Viewer*

Вкладка **DirectX 9.0 Viewer**, изображенная на рис. 6.9, позволяет настраивать параметры, связанные непосредственно с использованием свойств DirectX 9. На этой вкладке существует множество различных элементов управления.

Первое текстовое поле **HLSL Includes** (Включает HLSL) позволяет определить каталог, где могут располагаться файлы шейдера. Каталогов может быть несколько, тогда название всех каталогов прописывается в текстовом поле **HLSL Includes** через точку с запятой.

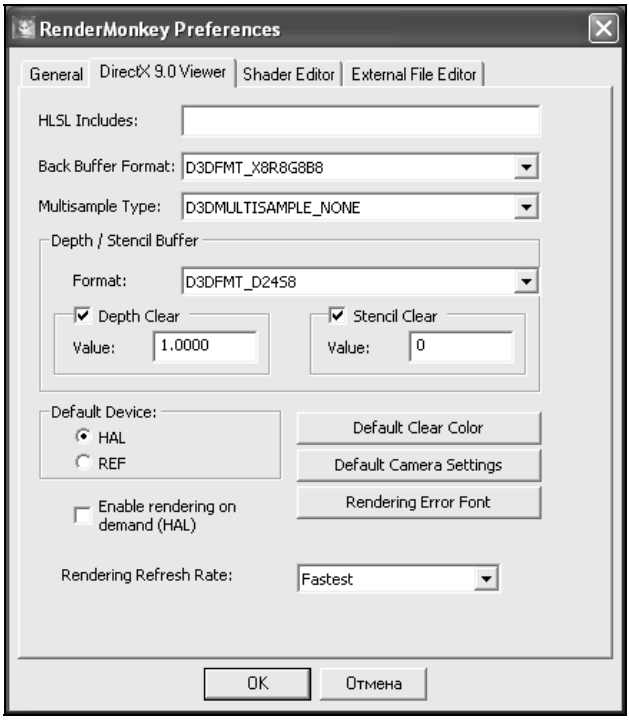


Рис. 6.9. Вкладка **DirectX 9.0 Viewer**

Затем на вкладке **DirectX 9.0 Viewer** диалогового окна **RenderMonkey Preferences** располагается список **Back Buffer Format** (Формат заднего буфера), необходимый для выбора формата заднего буфера. Здесь перечислены все имеющиеся в DirectX форматы заднего буфера, и вы можете использовать любой необходимый вам формат.

Ниже списка **Back Buffer Format** находится еще один список **Multisample Type** (Тип мультисемплинга), используемый для выбора мультисемплинга. По умолчанию выставлено значение **D3DMULTISAMPLE_NONE**.

Вслед за списком **Back Buffer Format** на вкладке **DirectX 9.0 Viewer** диалогового окна **RenderMonkey Preferences** (см. рис. 6.9) располагается статическая область **Depth / Stencil Buffer** (Буфер глубины и буфер трафарета), включающая в себя небольшой набор разнородных элементов управления.

С помощью списка **Format** (Формат) в статической области **Depth / Stencil Buffer** можно устанавливать необходимый формат для буфера глубины и буфера трафарета. Два флажка: **Depth Clear** (Очистить буфер глубины) и **Stencil Clear** (Очистить буфер трафарета), если они выбраны, очищают соответственно буфер глубины и буфер трафарета. Под двумя флажками **Depth Clear** и **Stencil Clear** находятся два идентичных по размеру текстовых поля с

одноименным названием **Value** (Значение), в этих полях задаются значения, на основе которых будет происходить очистка буфера глубины и буфера трафарета.

Следом за статической областью **Depth / Stencil Buffer Settings** (Буфер глубины и буфер трафарета) на вкладке **DirectX 9.0 Viewer** располагается статическая область **Default Device** (Устройство по умолчанию) с двумя переключателями: **HAL** (Аппаратный уровень абстракции) и **REF** (Программный уровень абстракции). С помощью этих двух переключателей можно задавать аппаратную или программную обработку данных.

С правой стороны от статической области **Default Device** (Устройство по умолчанию) на вкладке **DirectX 9.0 Viewer** находятся три кнопки: **Default Clear Color** (Цвет по умолчанию), **Default Camera Settings** (Настройки камеры по умолчанию) и **Rendering Error Font** (Шрифт ошибок).

После нажатия кнопки **Default Clear Color** появится небольшое диалоговое окно с одноименным названием, показанное на рис. 6.10, где можно задавать цвет для очистки заднего буфера.

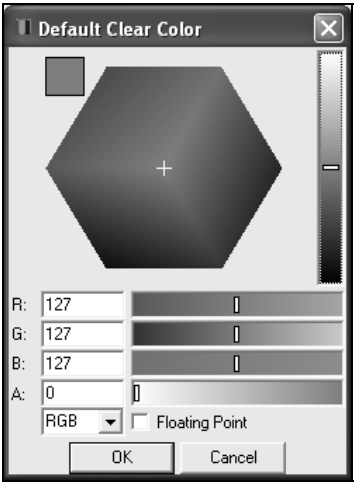


Рис. 6.10. Диалоговое окно **Default Clear Color**

Кнопка **Default Camera Settings** (Настройки камеры по умолчанию) позволяет устанавливать позицию камеры в окне предварительного просмотра. Изначально позиция камеры установлена по умолчанию, если вы желаете назначить новую позицию, то нажмите на кнопку **Default Camera Settings** и в появившемся диалоговом окне **Default DirectX preview camera node**, показанном на рис. 6.11, установите необходимые значения.

Третья и последняя кнопка **Rendering Error Font** (Шрифт ошибок) на вкладке **DirectX 9.0 Viewer** диалогового окна **RenderMonkey Preferences** ведет к

окну задания цвета текста, предназначенного для вывода информации об имеющихся ошибках. Нажмите на кнопку **Rendering Error Font**, и появится диалоговое окно **Шрифт**, изображенное на рис. 6.12, где производится установка шрифта.

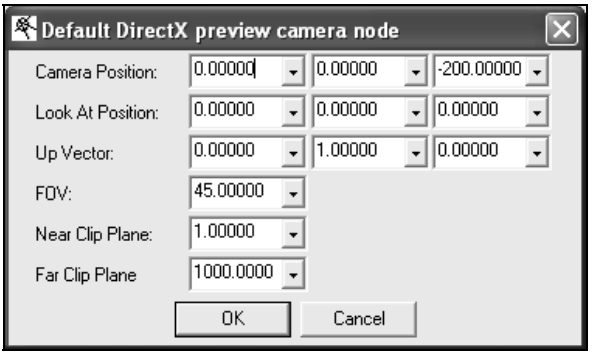


Рис. 6.11. Диалоговое окно **Default DirectX preview camera node**

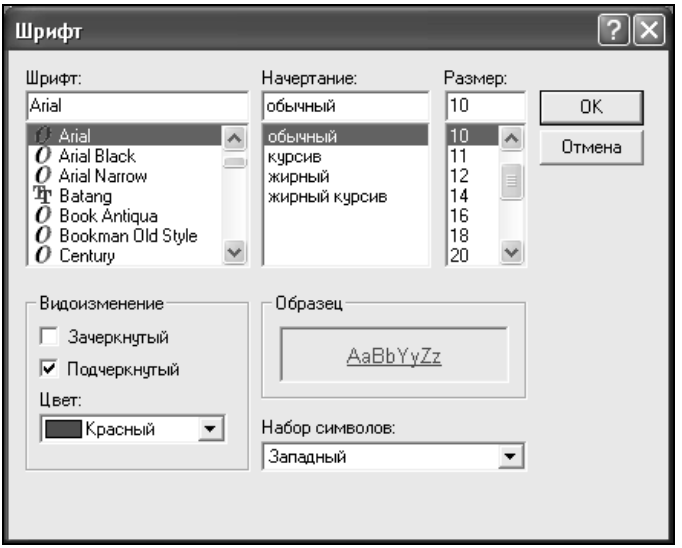


Рис. 6.12. Диалоговое окно **Шрифт**

В нижней части вкладки **DirectX 9.0 Viewer** диалогового окна **RenderMonkey Preferences** находится список **Rendering Refresh Rate** (Частота обновления), используя который вы можете управлять частотой обновления текущего эффекта в окне **Preview**.

Вкладка *Shader Editor*

Вкладка **Shader Editor** (Редактор шейдеров), изображенная на рис. 6.13, необходима для настройки текстового редактора шейдеров.

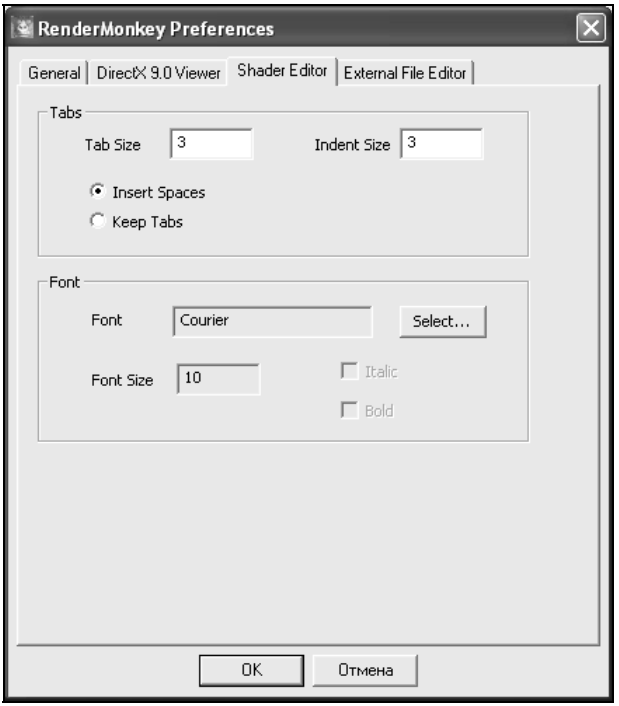


Рис. 6.13. Вкладка **Shader Editor**

Вкладка **Shader Editor** разделена на две области: **Tabs** (Табуляция) и **Font** (Шрифт), каждая из которых содержит небольшой набор элементов управления. В области **Tabs** задаются параметры для размеров отступа и позиций табуляции, а в области **Font** с помощью текстовых полей и кнопки **Select** (Выбор) можно задавать различные виды шрифтов для текстового редактора шейдеров.

Вкладка *External File Editor*

Вкладка **External File Editor** (Внешний файловый редактор), показанная на рис. 6.14, позволяет выбирать внешние подключаемые программы для изменения текстур или моделей.

На вкладке имеется два текстовых поля: **Textures** (Текстуры) и **Models** (Модели), в которых необходимо указать местонахождение подключаемых в проект текстур и моделей.

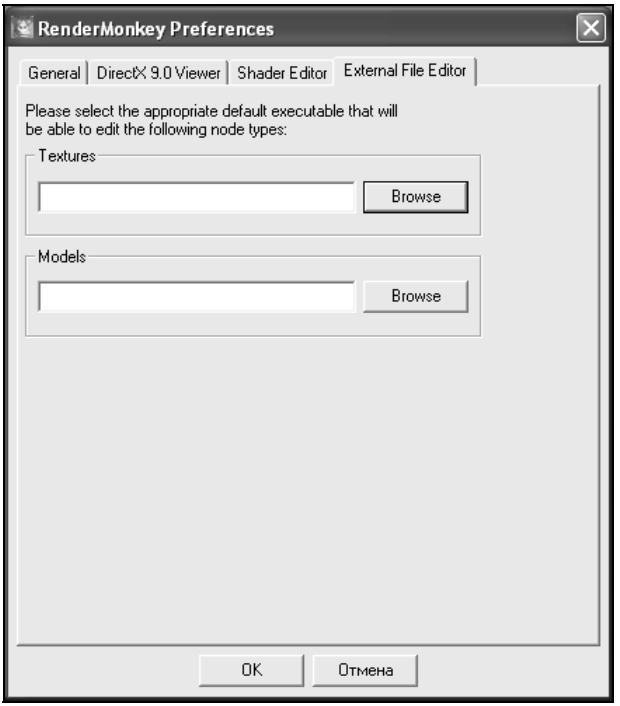


Рис. 6.14. Вкладка **External File Editor**

Меню *View*

Меню **View**, расположенное на инструментальной панели **RenderMonkey**, имеет четыре команды: **Workspace**, **Output**, **Preview** и **Artist Editor**. Каждая из команд соответствует названию рабочих окон и может открыть необходимую рабочую область. Например, на рис. 6.15 показана область **Artist Editor** (Художественный редактор).

Меню *Window*

Меню **Window**, изображенное на рис. 6.16, содержит семь команд для управления окнами инструментария RenderMonkey, также по мере работы с RenderMonkey в это меню могут быть автоматически добавлены и другие команды, связанные с открытыми окнами.

Меню **Window** содержит следующие команды:

- **Close** — закрывает окно с фокусом;
- **Close All** — закрывает все открытые окна;
- **Cascade** — выстраивает открытые окна каскадом;

Рис. 6.15. Окно **Artist Editor**

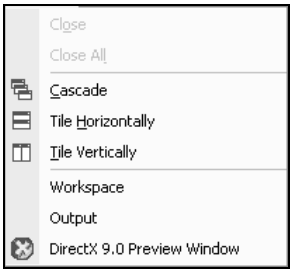


Рис. 6.16. Меню **Window**

- **Tile Horizontally** — расставляет открытые окна горизонтально по ширине всей рабочей области;
- **Tile Vertically** — располагает вертикально открытые окна в рабочей области;
- **Workspace** — открывает окно **Workspace**;
- **Output** — открывает окно **Output**.

Меню *Help*

В меню **Help** имеются две команды: **PlugIn Wizard** (Мастер подключаемых модулей) и **About** (Информация). Команда **PlugIn Wizard** разрешает пользователю генерировать проекты для Microsoft Visual Studio и программный код для дополнительно подключаемых в RenderMonkey модулей. Команда **About** содержит общую информацию об инструментарии RenderMonkey (рис. 6.17).

Рис. 6.17. Информационное окно **About RenderMonkey**

Панель инструментов *RenderMonkey*

Панель инструментов **RenderMonkey**, изображенная на рис. 6.18, содержит набор красочно оформленных кнопок для быстрой работы с системой (кнопки быстрого доступа).

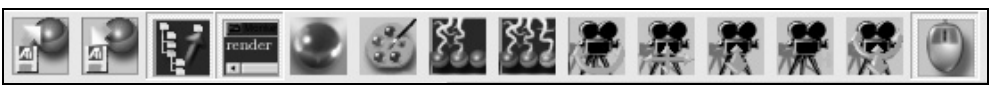


Рис. 6.18. Панель инструментов **RenderMonkey**

Из рис. 6.18 видно, что на панели инструментов располагаются четырнадцать интерактивных кнопок. Давайте рассмотрим каждую из них последовательно слева на право:

- **Open Workspace** — открывает новое рабочее пространство;
- **Save Workspace** — сохраняет открытое рабочее пространство;
- **Workspace Window** — открывает или закрывает окно **Workspace**;
- **Output Window** — открывает или закрывает окно **Output**;
- **Preview Window** — открывает или закрывает окно **Preview**;
- **Artist Editor** — открывает или закрывает окно **Artist Editor**;

- **Compile Active Effect** — компилирует активные эффекты, открытые в окне предварительного просмотра, также можно воспользоваться клавишей <F7>;
- **Compile All Shaders in the Workspace** — компилирует все шейдеры;
- **Rotate Camera** — задействует возможность вращения камеры с помощью левой кнопки мыши;
- **Move Camera** — дает возможность панорамирования камеры левой кнопкой мыши;
- **Zoom Camera** — изменение масштаба изображения с помощью левой кнопки мыши;
- **Camera Home** — возвращает камеру в исходное положение;
- **Overloaded Camera Mode** — режим перегрузки камеры. В этом случае левой кнопкой мыши можно вращать камеру, правой кнопкой мыши панорамировать камеру, а средней кнопкой или колесом производить масштабирование;
- **Mouse input Mode** — режим работы с мышью.

Окно *Workspace*

Окно **Workspace** (Рабочее пространство) находится с левой стороны рабочей поверхности инструментария RenderMonkey и содержит древовидную структуру текущего **Effect Workspace**. Посмотрите на рис. 6.19, где показана древовидная структура проекта Order — Independed Transparency из каталога ATI Research Inc\RenderMonkey 1.5\Example\DX9.

В окне **Workspace** отражается структура текущего проекта, выполненная в виде отдельных элементов или так называемых *узлов*, сгруппированных по общим атрибутам. Окно рабочего пространства **Workspace** имеет две вкладки: **Effect** (Эффект) и **Art** (Художественный), расположенные в нижней части окна **Workspace**. Вкладка **Effect** отображает весь комплект доступных узлов, где вы можете переходить по всей древовидной структуре, а также производить редактирование необходимых узлов. Для этого щелкните правой кнопкой мыши на нужном узле, и в ответ появится контекстное меню с набором доступных операций редактирования. Вторая вкладка **Art**, изображенная на рис. 6.20, отображает узлы, которые доступны для редактирования художником, программист же может не волноваться, что художник изменит один из эффектов, поскольку на вкладке **Art** доступны узлы, не изменяющие общего содержания программного кода. Более подробно работу с узлами окна **Workspace** мы рассмотрим в следующей главе на примерах.

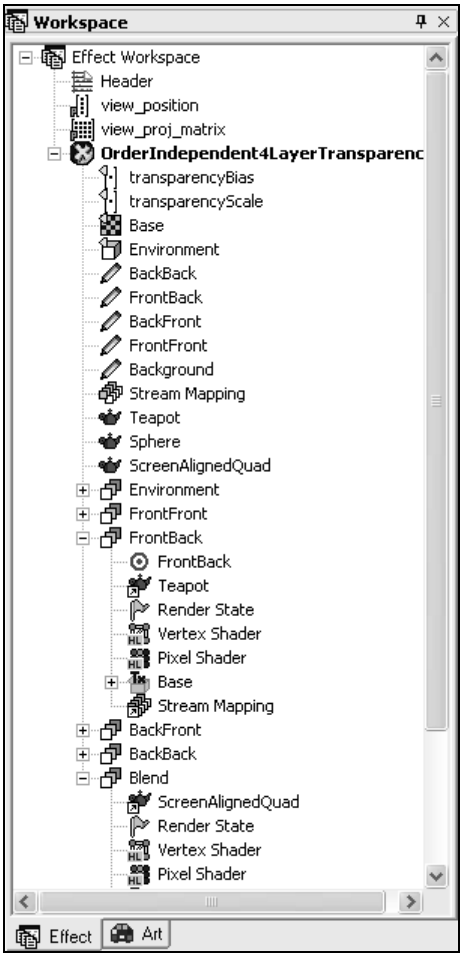


Рис. 6.19. Окно **Workspace**

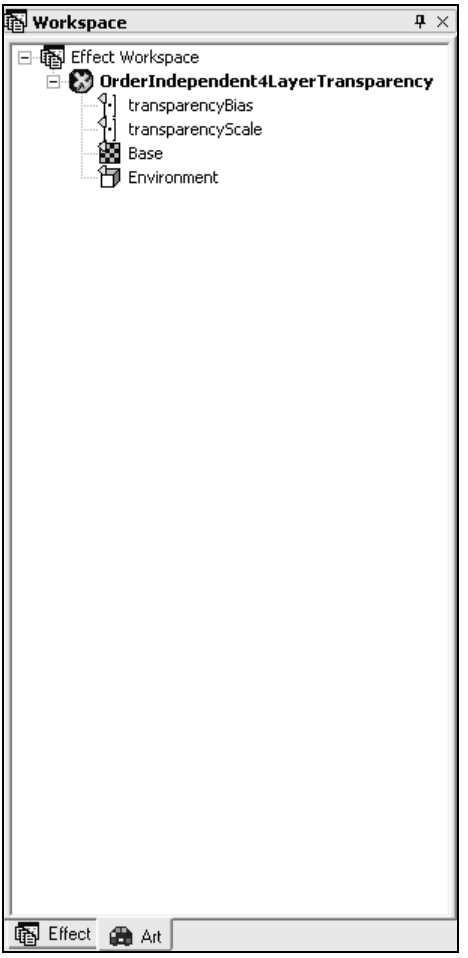


Рис. 6.20. Вкладка **Art**

Окно *Output*

Окно **Output** (Окно вывода выходной информации), изображенное на рис. 6.21, отображает выходную текстовую информацию о происходящих процессах в работе RenderMonkey.

Окно **Output** интерактивно, при каждом изменяющемся состоянии в работе с моделью происходит вывод соответствующей информации. При работе с текущим проектом с помощью окна **Output** можно воспользоваться справкой по "горячим" клавишам RenderMonkey, для этого нажмите на клавиатуре клавишу <H> и в окне **Output** будет перечислен список доступных "горячих" клавиш.

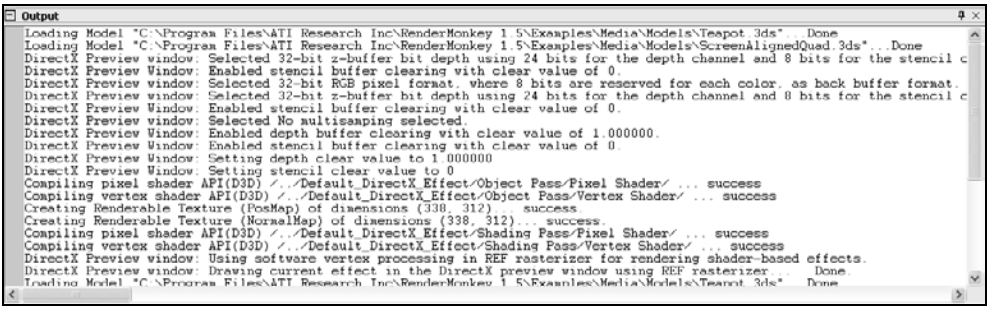


Рис. 6.21. Окно **Output**

В следующей главе мы более подробно рассмотрим работу RenderMonkey, а сейчас хочется продемонстрировать еще одну отличительную особенность инструментария RenderMonkey.

RenderMonkey SDK

В дистрибутиве RenderMonkey 1.5 есть прекрасная возможность создавать свои дополнительно подключаемые модули с помощью RenderMonkey SDK 1.0. Инструментальный пакет разработчика RenderMonkey SDK 1.0 содержит набор различных по своим возможностям библиотек, применяя которые можно создавать подключаемые модули. Инструментарий RenderMonkey написан на языке программирования C++ с использованием функционала RenderMonkey SDK 1.0. Такая гибкая архитектура позволяет разработчикам дополнять и улучшать RenderMonkey. Сама компания ATI поддерживает структуру дополнений, что вполне логично и рационально, ведь намного проще создавать дополнения, улучшающие работу программного обеспечения, чем создавать всю программу заново. И конечно, подобная структура подключения новых модулей дает возможность компании ATI и сторонним разработчикам создавать новые программные продукты с дальнейшей интеграцией в RenderMonkey.

С использованием RenderMonkey SDK вы будете создавать Win32-приложения в средах Visual C++ 6 или Visual C++ .NET, также можно задействовать MFC (Microsoft Foundation Classes, базовые классы Microsoft).

RenderMonkey SDK 1.0 идет в поставке основного дистрибутива RenderMonkey и специальной инсталляции не требует, поэтому SDK становится сразу доступным после установки RenderMonkey.

Библиотека RenderMonkey SDK

Библиотечные файлы RenderMonkey SDK с расширением lib содержат небольшой, но достаточно функциональный набор дополнительных утилит.

Давайте рассмотрим имеющиеся библиотечные файлы:

- RmCore — это основная библиотека, которую необходимо подключать при любом использовании RenderMonkey SDK. Она содержит механизмы управления интерфейсами и классами;
- RmUtilites — это STL-подобная библиотека, необходимая для работы с классами, списками и массивами строк;
- RmMFCUtilites — как видно из названия, эта библиотека основана на взаимодействии с MFC и предназначена для создания диалоговых окон, меню и других графических объектов.
- RmGrfxUtil — с помощью этой библиотеки можно работать с текстурами, в том числе производить преобразования форматов, изменение размера и т. д.

Библиотечные файлы находятся в том каталоге, в который был установлен инструментарий RenderMonkey, в папке SDK\Lib. Их необходимо явно подключать в рабочий проект. Как это делается, вы узнаете в конце этой главы, в *разд. "Подключение SDK в Visual C++ 6"* и *"Подключение SDK в Visual C++ .NET"*.

Заголовочные файлы RenderMonkey

Файлы заголовков RenderMonkey SDK с расширением h содержат описания интерфейсов, классов и функций, с помощью которых вы значительно упростите работу по созданию дополнительных модулей. Заголовочных файлов не так много, поэтому давайте рассмотрим их все:

- RmApplication.h — содержит определение основного интерфейса `IrmApplication`, доступ к которому можно получить через функцию `getRmApp()`;
- RmArray.h — предназначен для работы с массивами данных;
- RmCore.h — этот заголовочный файл содержит в себе все заголовки для упрощения процедуры подключения заголовочных файлов в проект;
- RmDefines.h — включает ряд необходимых макросов, подключаемых с помощью директивы `#defines`;
- RmEffect.h — предназначен для работы с эффектами;
- RmFile.h — содержит файлы помощи для функций открытия и закрытия файла;
- RmLinkedList.h — включает в себя описание класса шаблона `RmLinkedList`;
- RmMath.h — содержит математические классы;
- RmMatrix.h — в этом файле содержится все необходимое для работы с матрицами;
- RmMesh.h — предназначен для работы с мэшами в DirectX;

- RmPlugin.h — этот заголовочный файл декларирует объявление дополнительно подключаемых модулей;
- RmRegistryManager.h — своего рода менеджер состояния, определяет различные текстурные состояния;
- RmString.h — содержит интерфейс шаблона для работы со строками;
- RmStringToPtr.h — шаблон, упрощающий работу с указателями на строки;
- RmTypes.h — включает в себя набор структур и перечисляемых типов;
- RmUndo.h — как видно из названия этого файла, он обеспечивает возможность операций по отмене и восстановлению произведенных действий;
- RmVariableManager.h — декларирует описание интерфейса `IVariableManager` для управления списками;
- RmXMLManager.h — в этом файле находится определение интерфейса `IRmXMLManager`, необходимого для операций загрузки и сохранения файлов RenderMonkey с расширением rfx.

Все перечисленные заголовочные файлы находятся в директории RenderMonkey в папке SDK\lib.

> ***Примечание***
>
> При создании своих дополнительных модулей в исходном коде необходимо явное подключение используемого заголовочного файла.

Для создания новых подключаемых модулей с помощью RenderMonkey SDK можно воспользоваться средствами разработки приложений от компании Microsoft: Visual C++ 6 или Visual C++ .NET. Схема подключения SDK несколько различна для этих сред программирования, поэтому стоит рассмотреть процессы подключения SDK более внимательно.

Подключение SDK в Visual C++ 6

Перед началом работы убедитесь, что у вас установлен сервис-пакет № 5 (MSVC 6.0 Service Pack 5) для Visual C++ 6. При подключении RenderMonkey SDK в проект среды программирования Visual C++ 6 откройте Visual C++ 6 и создайте новый проект с помощью команд **File | New | Project** (Файл | Новый | Проект). В появившемся диалоговом окне **New**, изображенном на рис. 6.22, выберите шаблон создаваемого приложения, это может быть как Win32 Dynamic-Link Library, так и MFC DLL, а в поле **Project name** (Имя проекта) задайте имя будущему проекту и нажмите кнопку **OK**.

В только что созданном проекте вам необходимо произвести ряд настроек, связанных с подключением RenderMonkey SDK в ваш проект, для этого

лучше избрать форму последовательного перечисления шагов в процессе настроек.

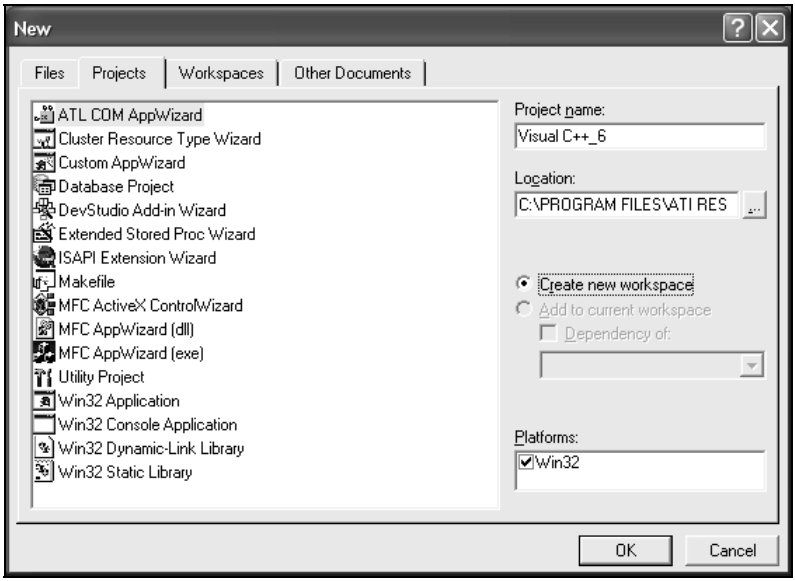


Рис. 6.22. Диалоговое окно **New**

Примечание

При создании нового проекта в среде программирования Visual C++ 6 использовалось название DemoSDK, в дальнейшем именно это название будет применяться при описании настроек параметров проекта.

1. После создания проекта DemoSDK откройте диалоговое окно **Settings** (Установки), воспользовавшись командами меню **Project | Settings**, и в появившемся диалоговом окне **Project Settings** (Настройки проекта) произведите настройки необходимых параметров. Диалоговое окно **Project Settings**, показанное на рис. 6.23, разделено на две части, с левой стороны располагается иерархия рабочего проекта DemoSDK, а в правой части окна расположен набор вкладок, перемещаясь по которым можно настраивать необходимые параметры. Если вы планируете использовать MFS, то на вкладке **General** (Основные) в списке **Microsoft Foundation Classes** выберите значение **Use MFC in a Shared DLL**.
2. Затем в диалоговом окне **Project Settings** перейдите на вкладку **Debug** (Отладка), изображенную на рис. 6.24, для назначения каталогов, используемых при отладке. В поле **Executable for debug session** (Программа для сессии отладки) вам нужно указать путь к исполняемому файлу RenderMonkey.exe, который находится в каталоге инструментария

RenderMonkey. Также в поле **Working directory** (Рабочая директория) пропишите путь к каталогу, где находится исполняемый файл RenderMonkey.exe.

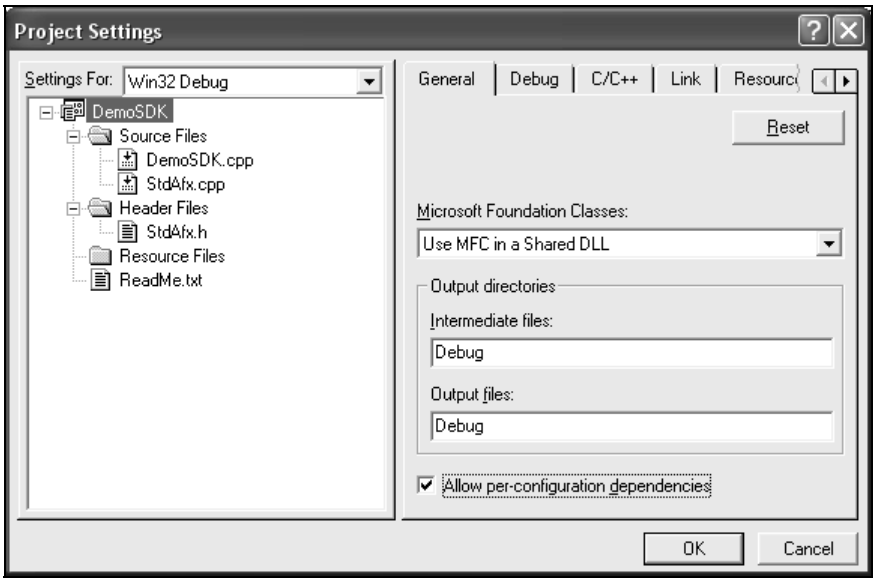


Рис. 6.23. Вкладка **General** окна **Project Settings**

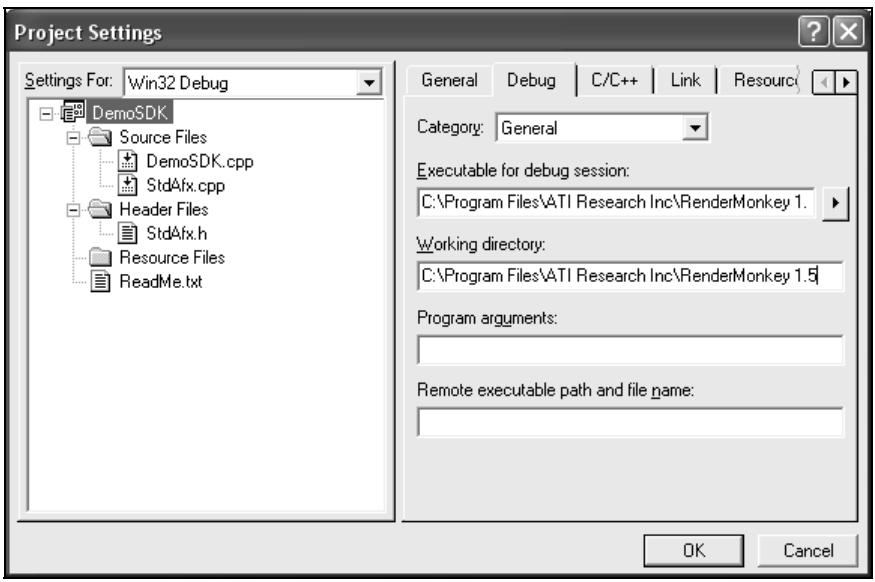


Рис. 6.24. Вкладка **Debug** окна **Project Settings**

3. Дальше перейдите на вкладку **C/C++** в диалоговом окне **Project Settings** (Настройки проекта). Вид вкладки **C/C++** изменяется в зависимости от выбранной категории в списке **Category** (Категория). Вначале изберите категорию **General**, как показано на рис. 6.25, и в поле **Preprocessor definitions** (Определение препроцессора) через запятую к уже имеющимся значениям добавьте макрос _UNICODE, для того чтобы RenderMonkey мог использовать уникод.

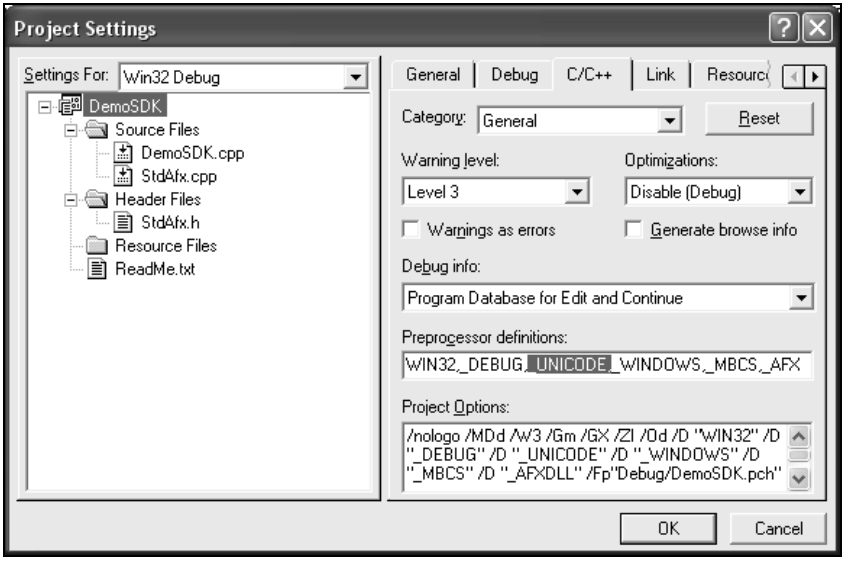


Рис. 6.25. Категория **General** вкладки **C/C++** окна **Project Settings**

4. Затем на этой же вкладке **C/C++**, но уже в категории **Code Generation** (Генерация кода), которую можно выбрать из списка **Category**, необходимо в списке **Use run-time library** выбрать значение **Multithreaded DLL**, как показано на рис. 6.26.
5. Далее на этой же вкладке **C/C++** выберите категорию **C++ Language** из списка **Category**, изображенную на рис. 6.27. Здесь вам необходимо выбрать флажок **Enable Run-Time Type Information (RTTI)** и перейти в категорию **Preprocessor** (Препроцессор).
6. В категории **Preprocessor** вкладки **C/C++** в текстовом поле **Additional include directories** (Дополнительно подключаемые директории) необходимо прописать путь к каталогу SDK\Include инструментария RenderMonkey, как показано на рис. 6.28. На этом настройки на вкладке **C/C++** заканчиваются.
7. Теперь перейдите на вкладку **Link** (Связывание) в диалоговом окне **Project Settings** и в списке **Category** выберите категорию **General**. В поле

Output file name (Имя файла) нужно прописать путь к каталогу PlugIns, который находится в папке RenderMonkey, например C:\Program Files \ATI Research Inc\RenderMonkey 1.5\PlugIns, как показано на рис. 6.29.

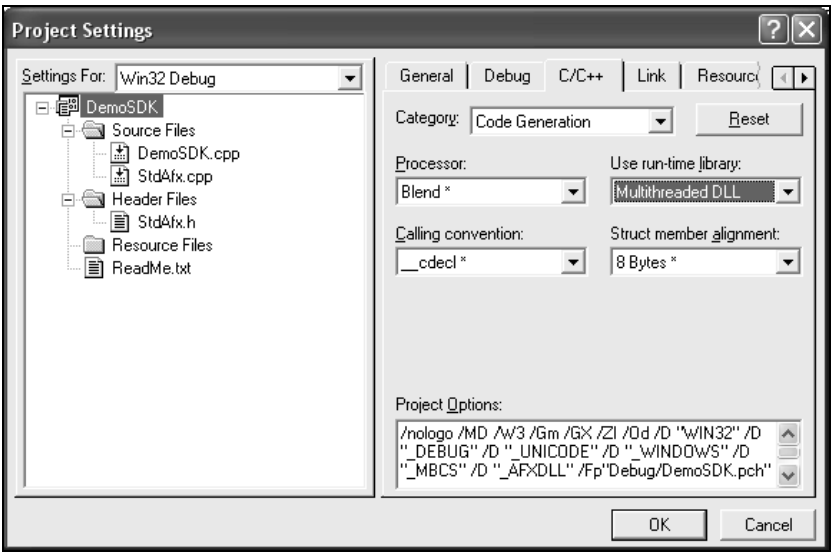


Рис. 6.26. Категория **Code Generation** вкладки **C/C++** окна **Project Settings**

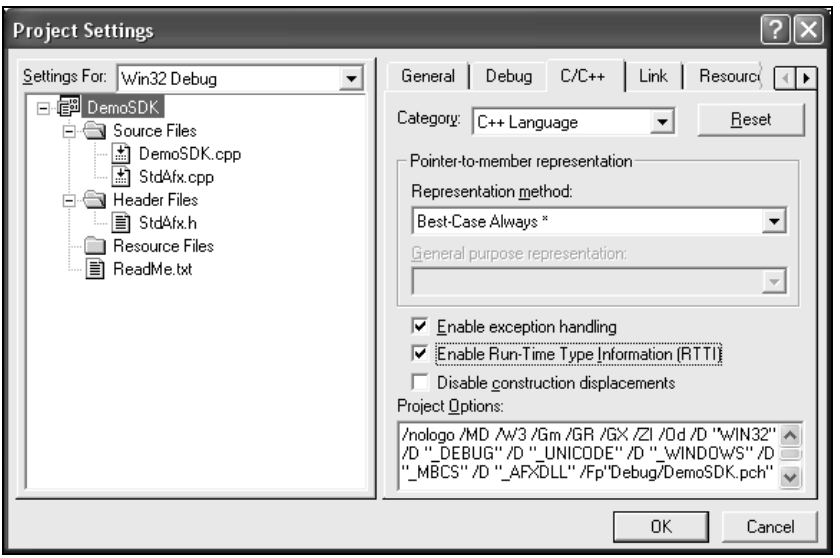


Рис. 6.27. Категория **C++ Language** вкладки **C/C++**

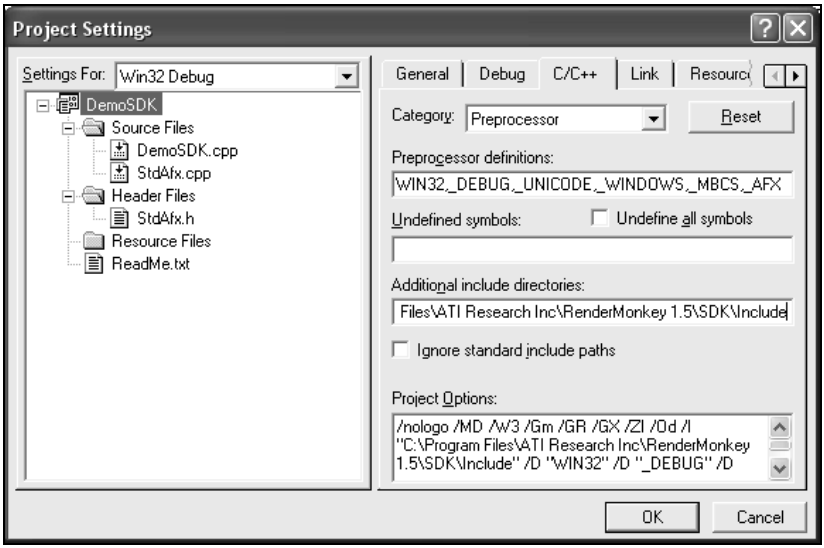


Рис. 6.28. Категория **Preprocessor** вкладки **C/C++**

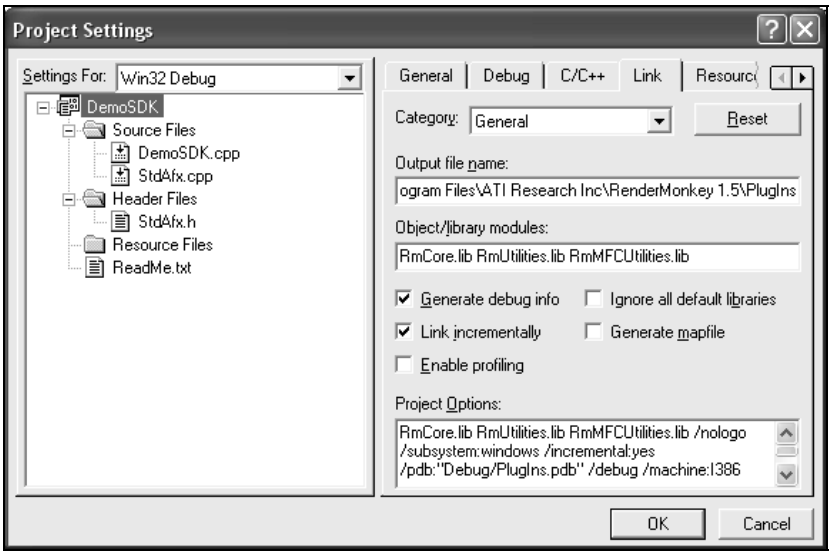


Рис. 6.29. Категория **General** вкладки **Link** окна **Project Settings**

Затем в поле **Object/library modules** нужно написать через пробел следующие библиотеки: RmCore.lib, RmUtilities.lib, и в том случае, если вы используете MFS, библиотеку RmMFCUtilities.lib.

8. Последний завершающий шаг — это указание каталога, где находятся библиотечные файлы RenderMonkey. Для этого на вкладке **Link** в списке

Category выберите категорию **Input** (Ввод) и в поле **Additional library path** (Дополнительный путь к библиотекам) укажите путь к каталогу SDK\Lib инструментария RenderMonkey. На этом подключение RenderMonkey SDK в проект Visual C++ 6 будет закончено, и вы сможете приступить к работе по написанию необходимого модуля.

Подключение SDK в Visual C++ .NET

Откройте среду программирования Visual C++ .NET и создайте новый проект с помощью команд **File | New | Project** (Файл | Новый | Проект). В появившемся диалоговом окне **New Project** (Новый проект), изображенном на рис. 6.30, выберите в поле **Templates** шаблон создаваемого приложения: MFS DLL в том случае, если вы собираетесь использовать компоненты MFS, либо Win32 Project для создания простого Windows-приложения.

Примечание

В качестве имени создаваемого проекта использовалась аббревиатура Demo for Visual C++_NET, которая в дальнейшем будет применяться в рассмотрении процесса подключения RenderMonkey SDK в проекте Visual C++ .NET.

Выбрав определенный шаблон, нажмите кнопку **OK** в диалоговом окне **New Project**, и Visual C++ .NET сгенерирует проект, после чего необходимо выполнить ряд последовательных действий для подключения RenderMonkey SDK.

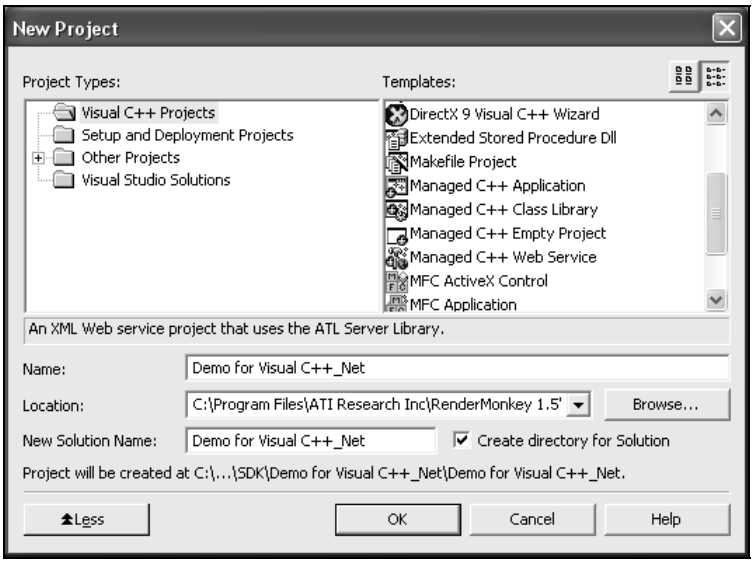


Рис. 6.30. Диалоговое окно **New Project**

1. В окне **Solution Explorer** среды программирования Visual C++ .NET щелкните правой кнопкой мыши на названии созданного проекта и в появившемся контекстном меню выберите команду **Properties** (Свойства). В ответ на выполненные действия появится диалоговое окно **Demo for Visual C++_Net Property Pages**, изображенное на рис. 6.31. С левой стороны этого окна в области древовидной структуры каталогов выберите папку **Configuration Properties** (Свойства конфигураций) и страницу свойств **General** (Основные). Страница свойств **General** имеет ряд настраиваемых параметров, выполненных в виде набора полей. Каждое поле содержит свой настраиваемый список свойств, щелкните левой кнопкой мыши на одном из полей и вам станет доступно определенное количество свойств. На странице свойств **General** необходимо изменить три параметра. В поле **Configuration Type** (Типы конфигурации) выберите **Dynamic Library (.dll)**, в поле **Character Set** (Установка кодировки) определите свойства как **Use Unicode Character Set** (Использовать уникод) и в поле **Use of MFC** (Использовать MFC), в том случае, если вы используете MFC, выберите **Use MFC in a Shared DLL**.

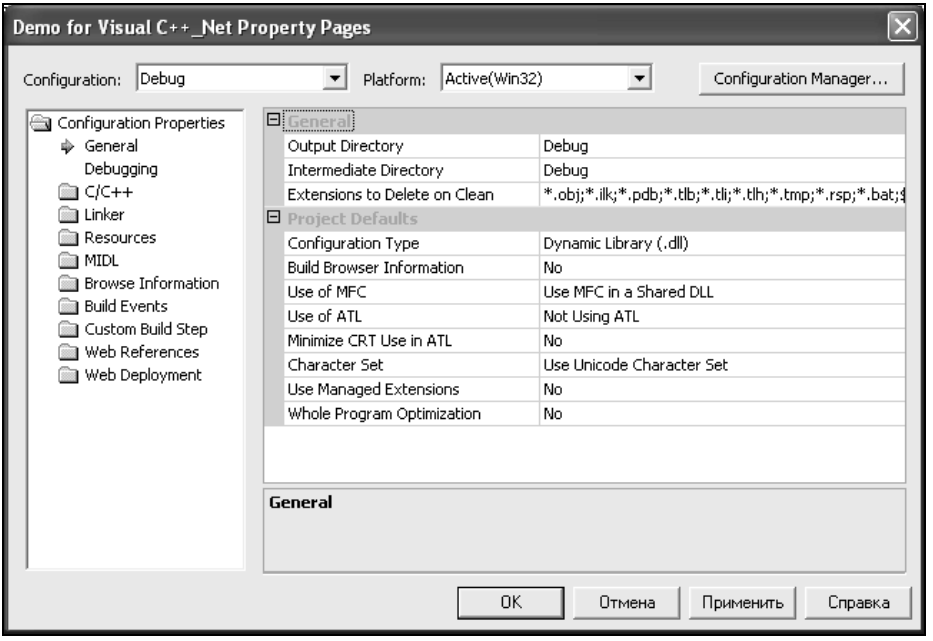


Рис. 6.31. Диалоговое окно **Demo for Visual C++_Net Property Pages**

2. Затем в диалоговом окне **Demo for Visual C++_Net Property Pages** необходимо перейти в папку C/C++ на страницу свойств **General**, представленную на рис. 6.32, и в поле **Additional Include Directories** (Дополнитель-

но подключаемые директории) указать полный путь к папке SDK\Include, находящейся в каталоге установки инструментария RenderMonkey.

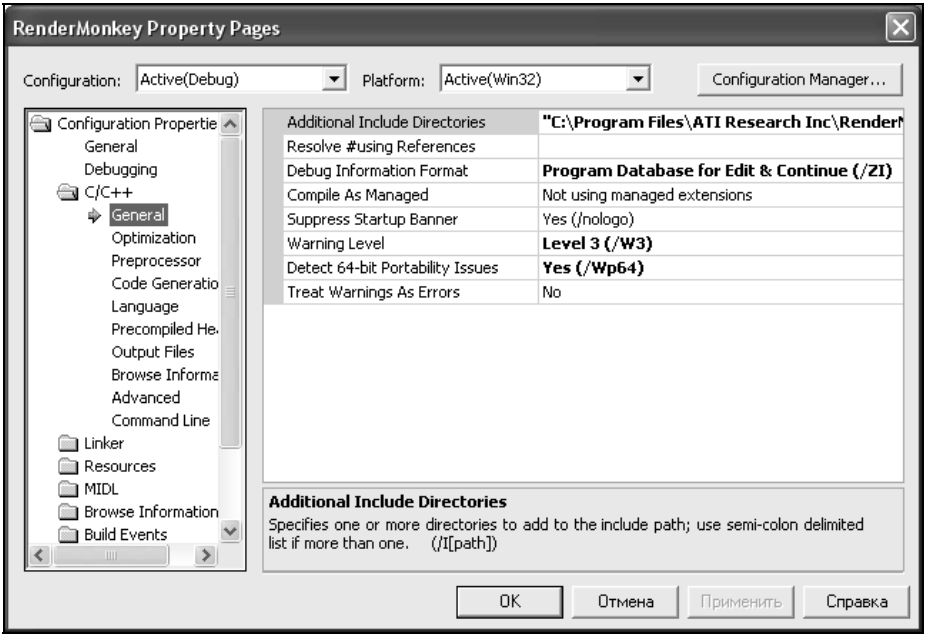


Рис. 6.32. Страница свойств **General** папки **C/C++**

3. Дальше в той же папке **C/C++** диалогового окна **Demo for Visual C++_Net Property Pages**, но уже на странице свойств **Code Generation** в поле **Runtime Library** должна быть выбрана строка **Mutli-threaded DLL (/MD)**, как показано на рис. 6.33.
4. Потом в папке **C/C++** диалогового окна **Demo for Visual C++_Net Property Pages** перейдите на страницу свойств **Language** (Язык), представленную на рис. 6.34, где вам необходимо изменить параметры для двух полей. В поле **Treat wchar_t as Built-in Type** выберите из списка значение **No**, а в поле **Enable Run-Time Type Info** выберите **Yes (/GR)**.
5. Затем перейдите по древовидной структуре каталогов диалогового окна **Demo for Visual C++_Net Property Pages** к папке **Linker** и выберите в этой папке страницу свойств **General**, изображенную на рис. 6.35. На странице свойств **General** нужно изменить значения в двух полях. В поле **Output File** вы должны указать каталог, куда будет сохранен создаваемый модуль, обычно это каталог RenderMonkey/SDK/Lib, и в поле **Additional Library Directories** (Дополнительно подключаемые директории) указать путь к подключаемым библиотекам — это каталог RenderMonkey/SDK /Lib.

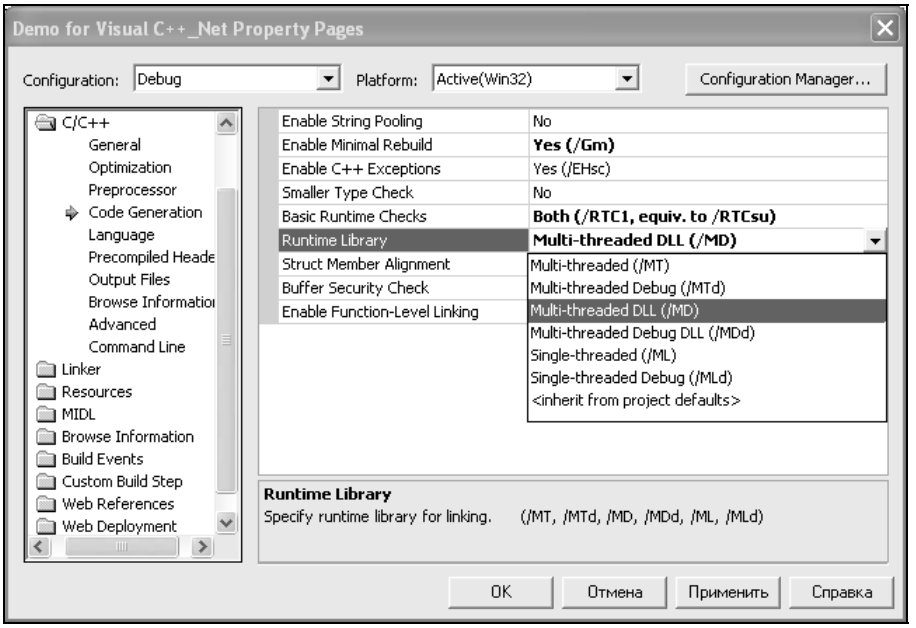


Рис. 6.33. Страница свойств **Code Generation** папки **C/C++**

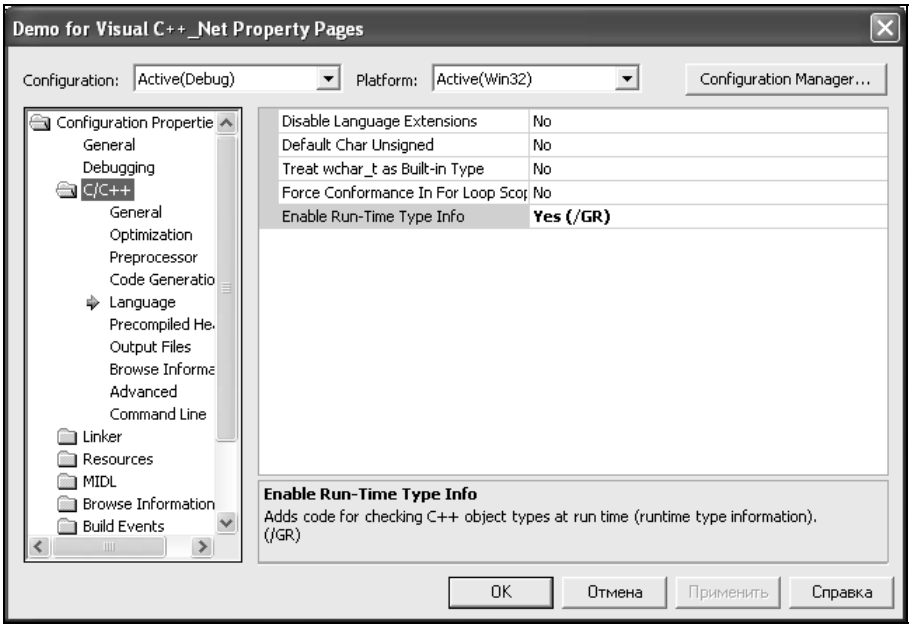


Рис. 6.34. Страница свойств **Language** папки **C/C++**

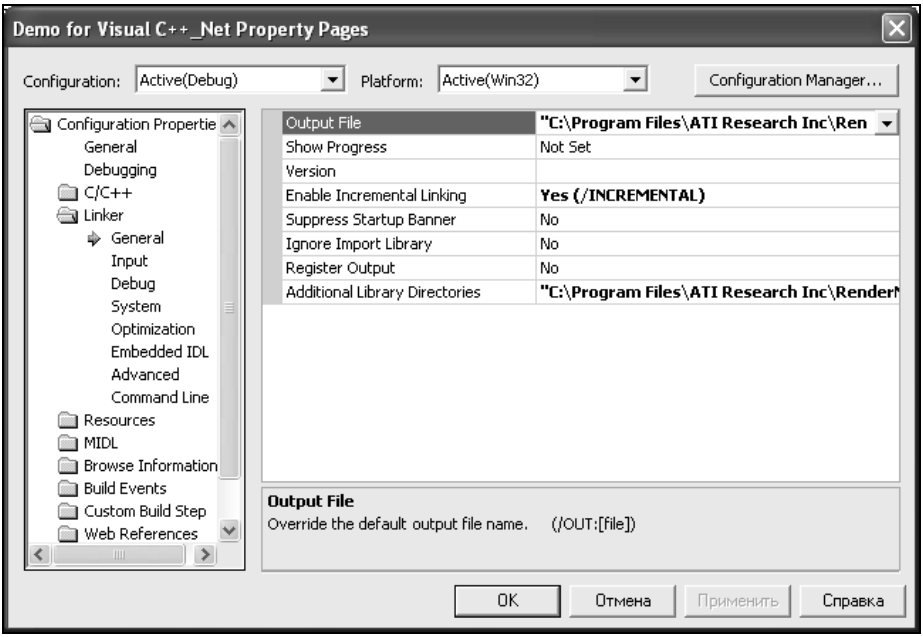


Рис. 6.35. Страница свойств **General** папки **Linker**

6. Далее в той же папке **Linker**, но уже на странице свойств **Input**, показанной на рис. 6.36, в поле **Additional Dependencies** (Дополнительная зависимость) необходимо прописать используемые библиотечные файлы. Это RmCore.lib и RmUtilities.lib, а в том случае, если вы применяете MFC, вы должны также подключить библиотечный файл RmMFCUtilities.lib.

7. И последний этап в процессе подключения SDK в проект Visual C++ .NET заключается в установках свойств, используемых в отладке. Для этого перейдите на страницу свойств **Debugging** (она находится в основной иерархии древовидной структуры каталогов диалогового окна **Demo for Visual C++_Net Property Pages** и изображена на рис. 6.37) для изменения двух значений. В поле **Command** вам необходимо задать путь к исполняемому файлу инструментария RenderMonkey.exe, а в поле **Working Directory** (Рабочая директория) укажите каталог инсталляции RenderMonkey. После чего нажмите последовательно кнопки **Применить** и **OK** в диалоговом окне **Demo for Visual C++_Net Property Pages** для вступления в силу всех произведенных изменений.

В следующей главе на примере мы рассмотрим техническую сторону работы инструментария RenderMonkey, что позволит вам убедиться в его абсолютно фантастических возможностях.

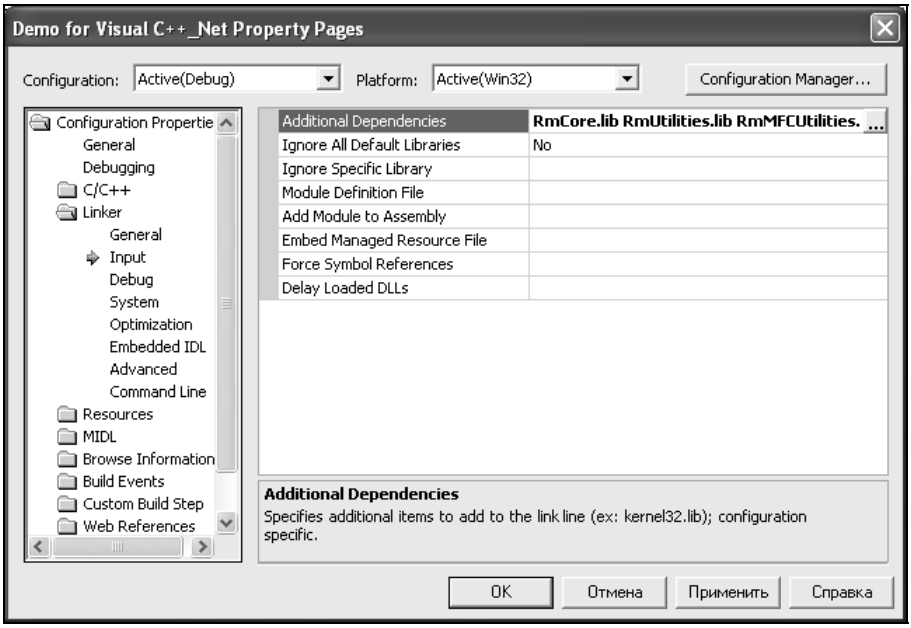


Рис. 6.36. Страница свойств **Input** папки **Linker**

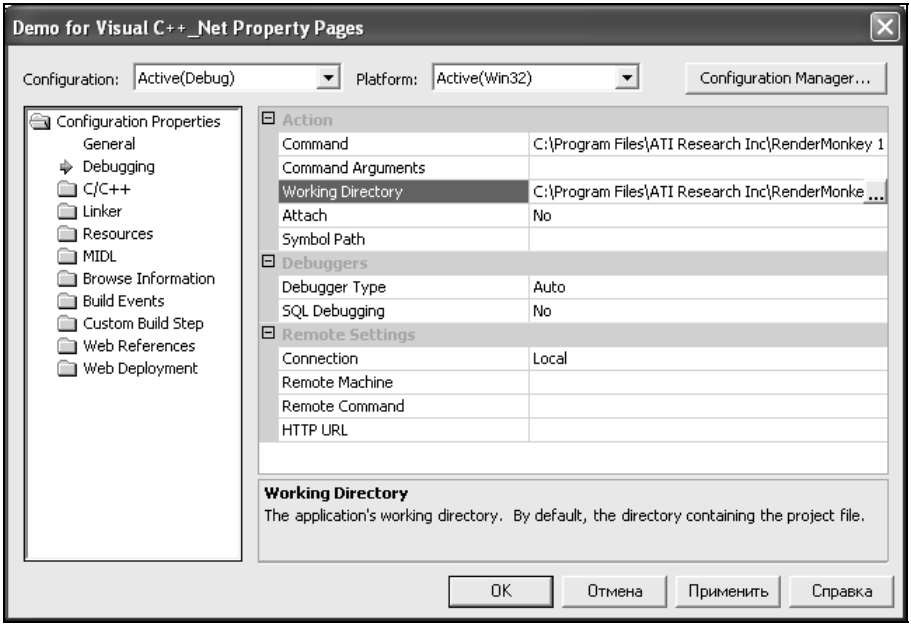
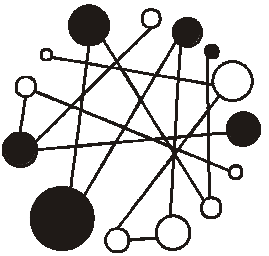


Рис. 6.37. Страница свойств **Debugging**

ГЛАВА 7



Работа с инструментарием RenderMonkey

В *главе 6* вы познакомились с интерфейсом RenderMonkey, теперь давайте перейдем к более детальному рассмотрению возможностей этого инструментария. После запуска RenderMonkey с левой стороны в его рабочей области будет находиться окно **Workspace**, где отображается иерархия текущего проекта, выполненная в виде древовидной структуры. При первоначальном открытии RenderMonkey в окне **Workspace** находится заданное по умолчанию рабочее пространство **Effect Workspace**, как показано на рис. 6.5, которое находится в каталоге установки инструментария RenderMonkey в папке ATI Research Inc\RenderMonkey 1.5\data\DefaultWorkspace.rfx. Файл с расширением rfx — это стандартный файл в формате XML, содержащий набор информации о рабочем эффекте, в том числе о коде шейдеров, установке состояния, текстурах, объектах и т. д. В данный момент формат XML является основным, и любой разработчик может создать конвертер или воспользоваться имеющимися на рынке средствами для преобразования в свой необходимый формат данных. Формат XML также используется и компанией NVIDIA в работе с инструментарием FX Composer.

При щелчке правой кнопкой мыши на рабочем пространстве **Effect Workspace** окна **Workspace** откроется контекстное меню, показанное на рис. 7.1, с перечнем доступных команд.

Контекстное меню рабочей области **Effect Workspace** имеет следующие команды:

- **Add Effect Group** — добавляет группу эффектов с помощью вложенного меню с тремя командами:
  - **Empty Effect Group** — добавляет пустую группу эффектов;
  - **Empty Effect Group w/DirectX Effect** — добавляет группу эффектов DirectX, которая находится в каталоге установки инструментария RenderMonkey в папке ATI Research Inc\RenderMonkey 1.5 \Examples \Media\Models;

- **Empty Effect Group w/OpenGL Effect** — добавляет группу эффектов OpenGL, которая находится в каталоге установки инструментария RenderMonkey в папке ATI Research Inc\RenderMonkey 1.5 \Examples \Media\Models.

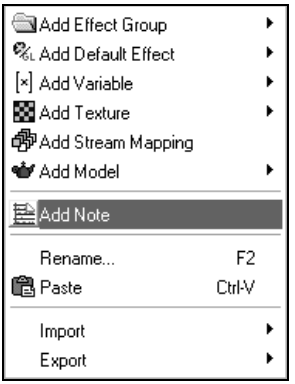


Рис. 7.1. Контекстное меню **Effect Workspace**

- **Add Default Effect** — добавляет эффект по умолчанию для DirectX и OpenGL;
- **Add Variable** — добавляет переменную типа `Integer`, `Boolean`, `Float`, `Matrix` и `Color`;
- **Add Texture** — добавляет текстуру из каталога установки инструментария RenderMonkey, находящуюся в папке ATI Research Inc\RenderMonkey 1.5 \Examples\Media\Texture;
- **Add Stream Mapping** — добавляет выборку вершинных потоков;
- **Add Model** — добавляет модель из каталога установки инструментария RenderMonkey, находящуюся в папке ATI Research Inc\RenderMonkey 1.5 \ Examples\Media\Models;
- **Add Note** — добавляет запись;
- **Rename** — переименовывает узел;
- **Paste** — вставляет узел;
- **Import** — импорт проекта в RenderMonkey в формате ZIP;
- **Export** — экспорт проекта RenderMonkey в форматах ZIP и FX.

С помощью команды **Add Effect Group** (Добавить группу эффектов) можно создавать пустую группу эффектов (**Empty Effect Group**) и добавлять в нее по необходимости свои элементы проекта для формирования связанной группы эффектов. Либо использовать набор имеющихся эффектов в стандартной поставке RenderMonkey с помощью команды **Empty Effect Group**

w/DirectX Effect из меню **Add Effect Group** для DirectX и команды **Empty Effect Group w/OpenGL Effect** для OpenGL.

Также имеется возможность в выборе различных эффектов, поставляемых RenderMonkey. Выберите команду **Add Default Effect** (Добавить эффект по умолчанию) щелчком правой кнопкой мыши на **Effect Workspace** в окне **Workspace**. После этих действий откроется вложенное меню с двумя пунктами: **DirectX** и **OpenGL**. Эти два пункта имеют свои независимые подменю с большим набором команд, каждая из которых представляет тот или иной вид эффекта. Выбирая любой из доступных эффектов для DirectX или OpenGL, вы сможете получить некий каркас для будущего проекта с возможностью дальнейшего его развития. Чтобы воспользоваться готовыми примерами стандартной поставки RenderMonkey, находящимися в папке ATI Research Inc\RenderMonkey 1.5 \Examples, необходимо на панели инструментов RenderMonkey нажать кнопку быстрого доступа **Open Workspace** (Открыть рабочее пространство) или воспользоваться командой меню **File | Open** (Файл | Открыть) (<Ctrl>+<O>). В появившемся диалоговом окне **Открыть** (Open), показанном на рис. 7.2, выберите необходимый вам пример из папок Dx9, GL2 или Media и нажмите кнопку **Открыть**.

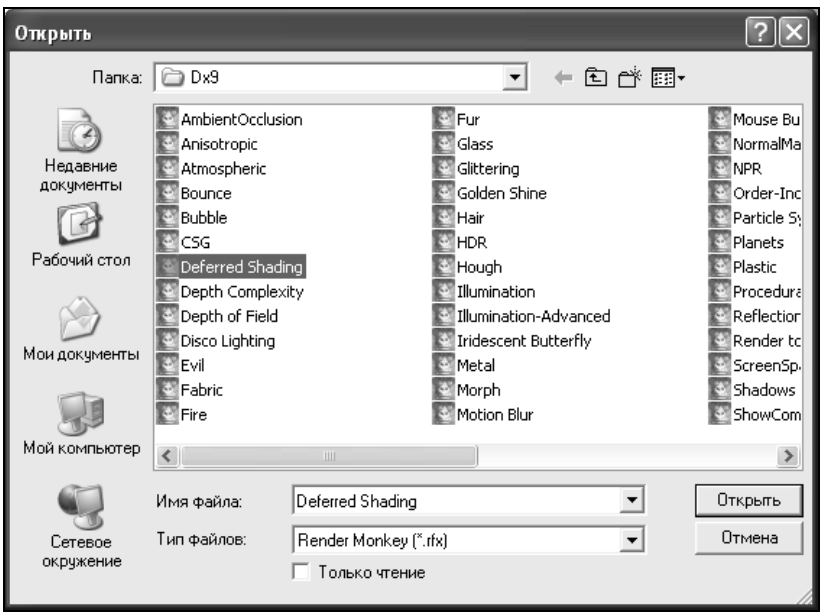


Рис. 7.2. Диалоговое окно **Открыть**

Выберем, например, проект Deferred Shading из папки Dx9, где применяется затенение объекта трехмерной сцены. В качестве объекта используется чайник желтого цвета в формате 3DS из каталога установки инструментария

RenderMonkey. Он находится в папке ATI Research Inc\RenderMonkey 1.5\Examples\Media\Models\Teapot.3ds. На рис. 7.3 показано рабочее окно инструментария RenderMonkey с открытым проектом Deferred Shading.

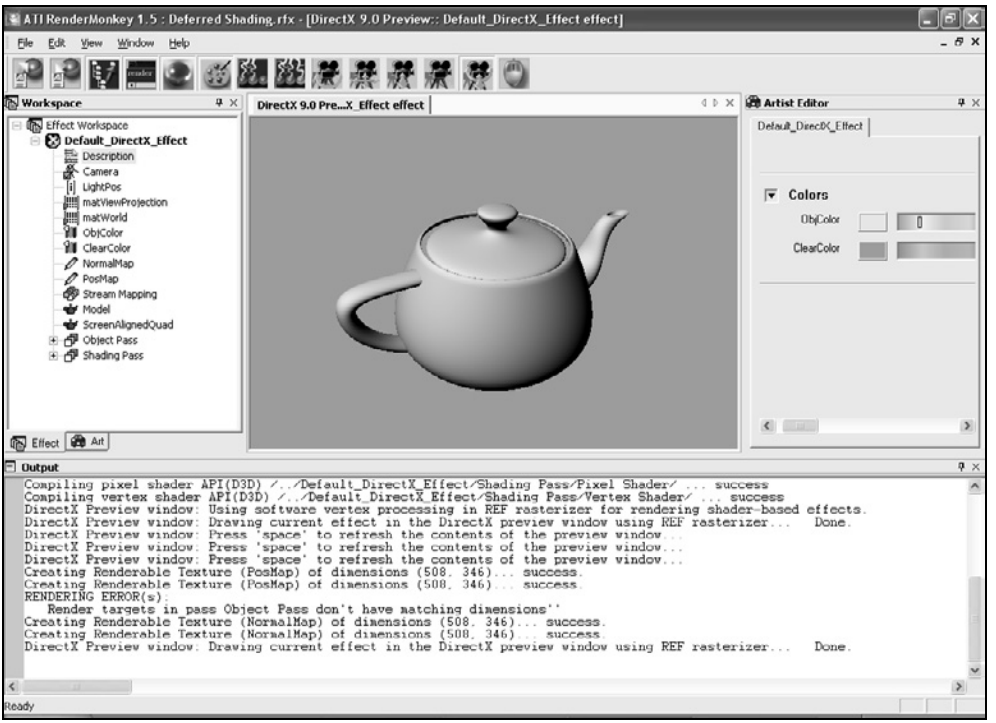


Рис. 7.3. Проект Deferred Shading

При открытии проекта Deferred Shading в окне **Workspace** инструментария RenderMonkey отображается полная иерархия проекта. Окно **Workspace** разделено на две активные вкладки: **Effect** (Эффект) и **Art** (Художник). Вкладка **Effect** (Эффект) содержит все необходимые атрибуты для работы программиста над текущим проектом, тогда как вкладка **Art** (Художник) предназначена для работы художника, и представленная структура проекта на этой вкладке не дает возможности художнику изменять исходный код проекта, что весьма удобно. На рис. 7.4 изображена структура проекта Deferred Shading в окне **Workspace**.

Все данные, как проекта Deferred Shading, так и любых других проектов, созданных в RenderMonkey, скомпонованы удобным образом. Проектные данные находятся в едином рабочем пространстве **Effect Workspace** и могут быть разделены на группы эффектов, отдельно взятые эффекты, проходы, установки состояний, переменные, вершинные и пиксельные шейдеры, объекты, текстуры и выборки вершинных потоков. **Effect Workspace** — это

корневой узел для всех проектов, его нельзя удалить, изменить, скопировать, переименовать или вырезать. Каждый тип перечисленных данных или узел (Node) содержит свой вид разнообразных пиктограмм для динамичной идентификации данных проекта. На рис. 7.4 хорошо виден набор узлов с графическими пиктограммами. Все данные организованы в виде древовидной структуры и могут быть сформированы в различные группы эффектов для вашего удобства. Сформированные группы эффектов позволяют объединять в группы необходимые типы данных, например, ничто не мешает вам сгруппировать шейдеры по профилю и языку программирования. Узлы строго подразделяются по своим типам и могут содержать одно и более значений. Говоря о типах данных, нужно понимать, что это не что иное, как

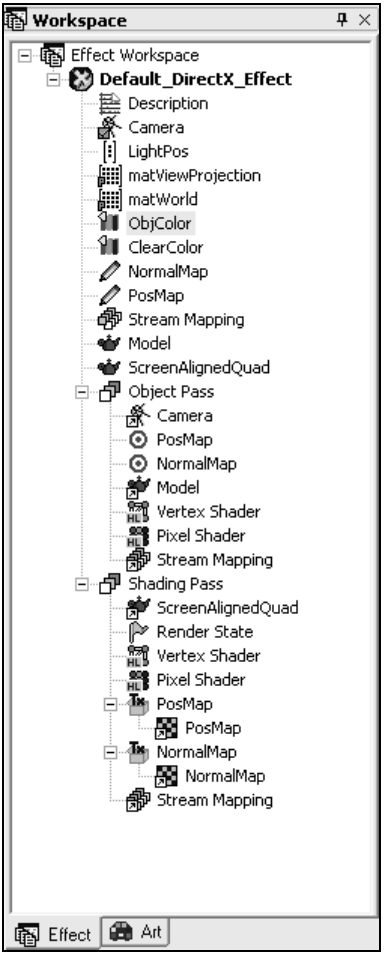


Рис. 7.4. Структура проекта Deferred Shading в окне **Workspace**

текстуры, модели, матрицы проекций, камеры, эффекты либо группы эффектов, используемые в сцене, шейдеры, переменные, источники освещения и т. д. Проект Deferred Shading, как и любой другой проект, наследует все свойства по умолчанию из заданного проекта Default Effect, что очень удобно для начального старта работы над проектом. Трудясь над проектом в DirectX или OpenGL, вы имеете полную свободу и возможность работы с любыми видами эффектов. Далее в этой главе будут рассматриваться все имеющиеся типы данных, которые можно задействовать в своем проекте RenderMonkey. На основе полученной информации вам будет очень легко формировать свои проекты.

Pass Node

В проектах RenderMonkey имеется один или несколько *проходов рисования* (Pass), которые наследуются от предыдущих проходов в эффекте или от Default Effect — эффекта по умолчанию. Каждый проход рисования включает в себя вершинный и пиксельный шейдер (обязательное условие), различные переменные, блоки установок (render state block), камеры, выборки вершинных объектов, текстурные и геометрические данные объекта. Также в проходах рисования могут использоваться всевозможные модели объектов. Посмотрите на рис. 7.5, там представлен проект Plastic из каталога Dx9 инструментария RenderMonkey, на основе которого мы рассмотрим модель работы проходов рисования. В качестве модели в проекте Plastic визуализирован слоненок синего цвета с эффектом пластика для материальных свойств объекта.

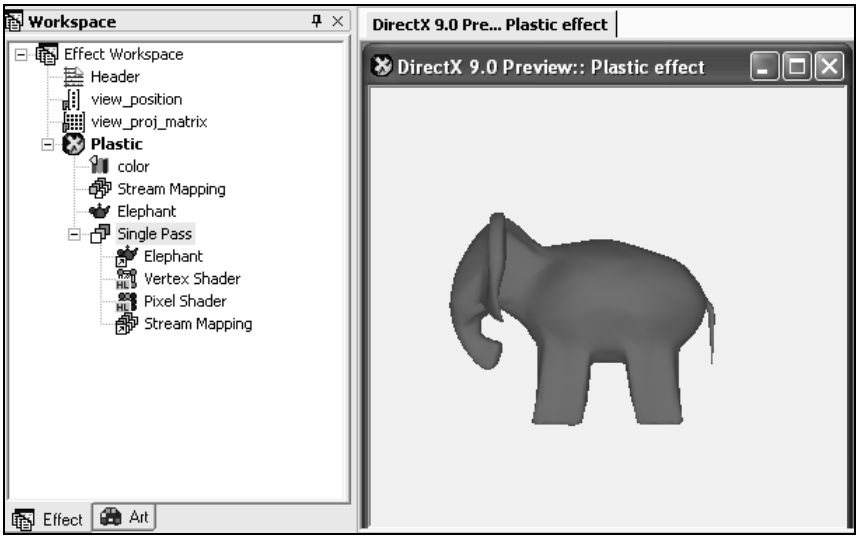


Рис. 7.5. Проект Plastic

В окне **Workspace** проекта Plastic (см. рис. 7.5) располагаются типы данных или узлы этого проекта. Узел **Single Pass** — это и есть простой одиночный проход рисования, включающий в себя набор данных. В проекте Plastic узел **Single Pass** состоит из вершинного шейдера (**Vertex Shader**), пиксельного шейдера (**Pixel Shader**), выборки вершинных потоков (**Stream Mapping**) и описания модели объекта (**Reference Node**). Проход рисования можно перемещать в пределах своего родительского эффекта вверх или вниз по древовидной структуре проекта. Также имеется возможность перестраивания, по своему усмотрению можно добавлять или удалять необходимые элементы в проходе рисования. Для этого необходимо щелкнуть правой кнопкой мыши на надписи прохода рисования, например в проекте Plastic Effect это **Single Pass**, и вам станет доступно контекстное меню для этого типа данных (рис. 7.6).

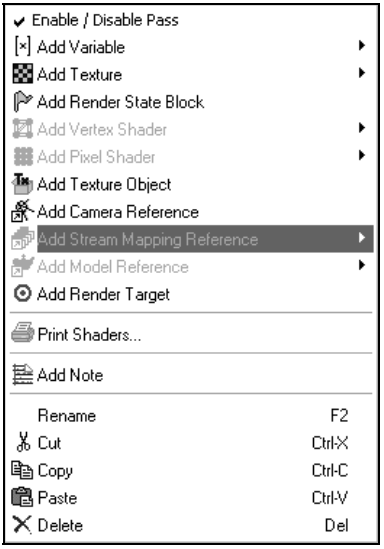


Рис. 7.6. Контекстное меню проекта Plastic

Контекстное меню прохода рисования стандартное и доступно в любом проекте RenderMonkey, рассмотрим его содержание:

- **Enable/Disable Pass** — включить или отключить проход рисования. При отключенном проходе рисования в окне **Workspace** появится красная косая черта, перечеркивающая пиктограмму прохода рисования;
- **Add Variable** — добавить переменную типа `Boolean`, `Integer`, `Float`, `Matrix` и `Color`;
- **Add Texture** — добавить текстуру;
- **Add Render State Block** — добавить блок установок;

- **Add Vertex Shader** — добавить вершинный шейдер;
- **Add Pixel Shader** — добавить пиксельный шейдер;
- **Add Texture Object** — добавить текстурный объект;
- **Add Camera Reference** — добавить описание свойства камеры;
- **Add Stream Mapping Reference** — добавить описание выборки вершинных потоков;
- **Add Model Reference** — добавить описание модели;
- **Add Render Target** — добавить адресата;
- **Print Shaders** — распечатка кода шейдеров;
- **Add Note** — добавить запись;
- **Rename** (<F2>) — переименовать;
- **Cut** (<Ctrl>+<X>) — вырезать;
- **Copy** (<Ctrl>+<C>) — скопировать в буфер обмена;
- **Paste** (<Ctrl>+<V>) — вставить из буфера обмена;
- **Delete** (<Del>) — удалить.

Обратите внимание, что не все пункты меню в проекте Plastic являются активными, это связано, прежде всего, с задействованными типами данных. По мере прибавления различных типов некоторые из пунктов меню будут становиться доступными. Для того чтобы добавить новый проход рисования, в проекте Plastic необходимо щелкнуть правой кнопкой мыши на названии проекта (**Plastic Effect**) и в появившемся меню выбрать пункт **Add Pass** (Добавить проход рисования). Вследствие чего в окне **Workspace** в иерархии рабочего проекта появится новый узел **Pass 1**, как показано на рис. 7.7, со стандартным набором данных (**Model**, **Vertex Shader**, **Pixel Shader** и **Stream Mapping**), который можно редактировать по своему усмотрению.

На этот момент узел **Model** в группе **Pass 1** не задействован, поэтому пиктограмма напротив надписи **Model** перечеркнута красной косой чертой, и если произвести компиляцию и запустить проект Plastic, то кроме сообщения об ошибке вы ничего не получите, это видно на рис. 7.8.

То есть каждый из проходов рисования должен быть привязан к конкретной модели, для этого щелкните правой кнопкой мыши в окне **Workspace** на узле **Model** в добавленном проходе рисования **Pass 1**. В появившемся меню выберите команду **Reference Node** и вы перейдете в подменю, где необходимо выбрать модель объекта. Но поскольку в одном проекте задействованных моделей может быть много, то и подменю отразит полный список используемых моделей в вашем проекте. Добавленный проход рисования вы вольны привязывать к любой из моделей, компонуя тем самым трехмерную сцену, исходя из ваших потребностей (рис. 7.9).

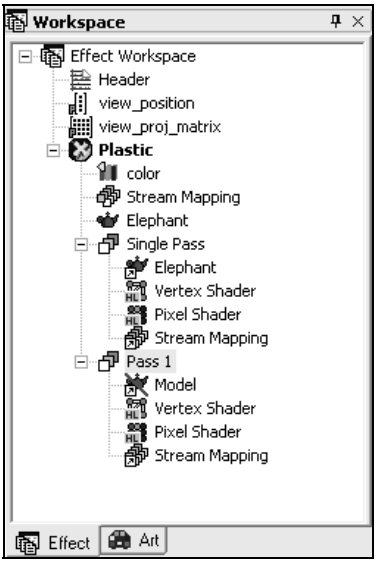


Рис. 7.7. Окно **Workspace** в проекте Plastic Effect с новым узлом **Pass 1**

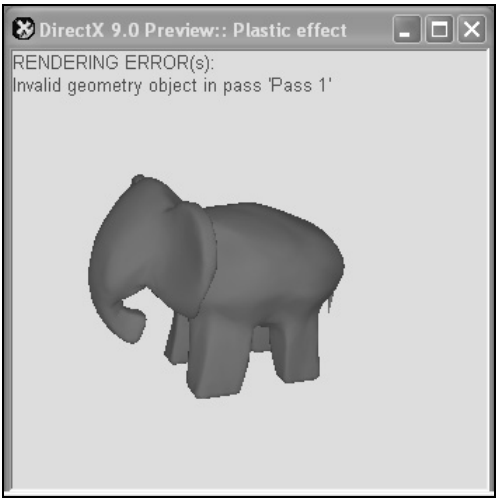


Рис. 7.8. Сообщение об ошибке при компиляции проекта Plastic

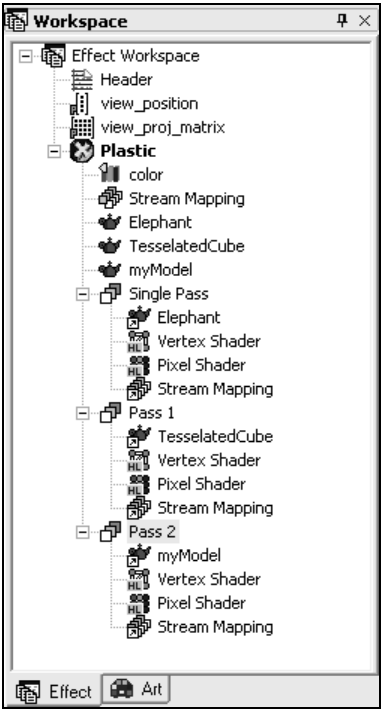


Рис. 7.9. Окно **Workspace** с несколькими проходами рисования

После привязывания моделей к проходам рисования необходимо откомпилировать проект. Для этого воспользуйтесь панелью инструментов **RenderMonkey** и нажмите кнопку **Compile All Shaders in the Workspace** (Откомпилировать все шейдеры на рабочем пространстве), а для того чтобы имеющиеся изменения вступили в силу, нажмите клавишу <Space> (Пробел) на клавиатуре компьютера. Все имеющиеся видоизменения трехмерной сцены моментально отразятся в окне **Preview Window** (Окно предварительного просмотра), где происходит прорисовка сцены. При добавлении новых эффектов, шейдеров, камер, текстур, моделей, переменных всегда необходимо явно откомпилировать проект и нажать клавишу <Space> для просмотра изменений в сцене в окне **Preview Window** (Окно предварительного просмотра). Но более простые модификации проекта, скажем изменение цвета фона или модели, выбор другой позиции просмотра сцены, не требуют перекомпиляции проекта, достаточно только нажатия клавиши <Space>.

Model Nodes

Модельные узлы (Model Nodes) представляют собой геометрические объекты или модели в формате 3DS, являющимся стандартом в игровой индустрии. Очень часто модели этого формата задействуются программистами в компьютерных играх. Модели в проекте RenderMonkey могут использоваться при исполнении прохода рисования. Если вы посмотрите на рис. 7.9, где был представлен проект Plastic, то заметите, что пиктограмма узла **Model Nodes** выполнена в виде маленькой изящной фигуры чайника красного цвета и располагается в родительской иерархии проекта Plastic. А уже в проходах рисования **Single Pass**, **Pass 1** и **Pass 2** графическое изображение пиктограммы выполнено в виде чайника со стрелкой. Это так называемая *модельная ссылка* (Model Reference), обозначающая, что модель объекта добавлена к проходу рисования и будет представлена в сцене, т. е. модель фактически связана через Model Reference с проходом рисования.

Добавление модели в проект можно произвести, нажав правой кнопкой мыши на названии рабочего проекта и в появившемся меню выбрав пункт **Add Model** (Добавить модель). В ответ на эти действия появится подменю с перечнем доступных моделей инструментария RenderMonkey (рис. 7.10).

Все модели объектов, перечисленные в списке, доступны из каталога RenderMonkey и находятся в папке ATI Research Inc\RenderMonkey 1.5 \Examples\Media\Models. Инструментарий RenderMonkey изначально имеет большой набор различных моделей, которые вы сможете использовать в сценах, но также ничто вам не мешает загружать и свои объекты. Для этого разместите ваши модели в каталоге RenderMonkey в папке ATI Research Inc\RenderMonkey 1.5\Examples\Media\Models либо укажите путь к каталогу с необходимыми моделями объектов. Воспользуйтесь командой меню

Edit | Preferences (Редактировать | Свойства) и в появившемся диалоговом окне **RenderMonkey Preferences**, показанном на рис. 7.11, на вкладке **General** в текстовом поле **Models** укажите путь к вашему каталогу объектов.

Рис. 7.10. Меню моделей RenderMonkey

Как уже упоминалось, модели объектов в RenderMonkey выполнены в формате 3DS, но при этом имеется полная поддержка форматов NMF, OBJ и самое главное осуществляется поддержка формата X-файлов DirectX корпорации Microsoft, и вы можете без проблем загружать модели этого формата. Более того, модель объекта в формате 3DS, загруженная в проект без особых усилий, конвертируется в формат X-файла. Для этого в текущем проекте в окне **Workspace** щелкните правой кнопкой мыши на модели и в появившемся контекстном меню выберите команды **Save | Geometry Saver** (Сохранить | Сохранить геометрию). В ответ на эти действия появится диалоговое окно **Сохранить как** (Save As), изображенное на рис. 7.12, где нужно задать имя и каталог для сохраняемой модели объекта в формате X-файла.

При щелчке правой кнопкой мыши в текущем проекте окна **Workspace** появляется контекстное меню с набором доступных команд редактирования и

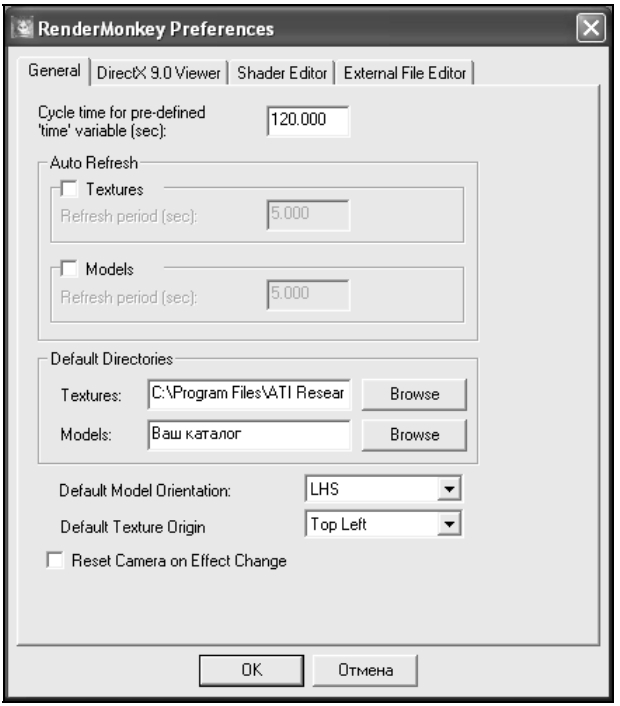


Рис. 7.11. Диалоговое окно **RenderMonkey Preferences**, новые модели

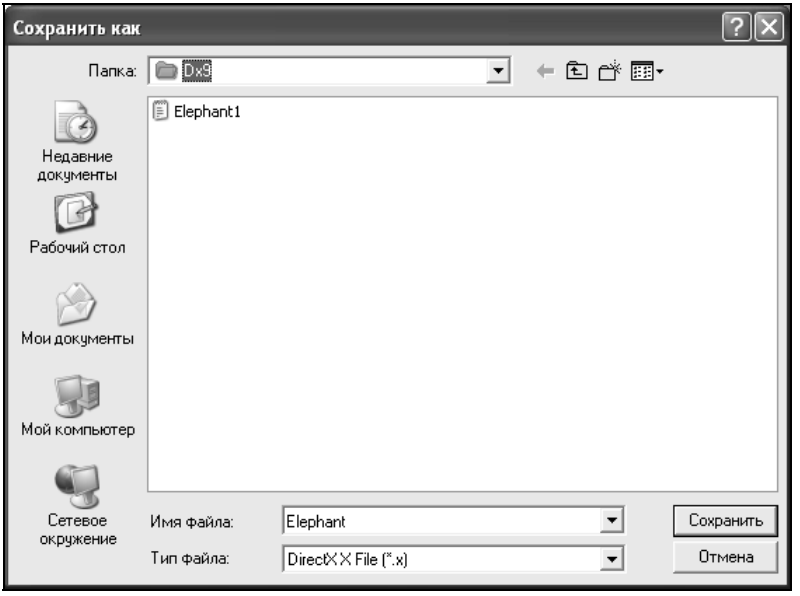


Рис. 7.12. Диалоговое окно **Сохранить как**

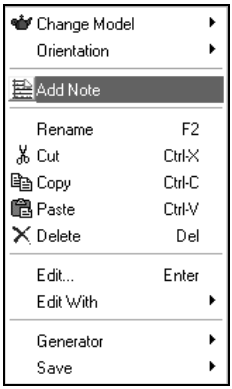


Рис. 7.13. Меню проекта Plastic и модели Elephant

задания различных дополнительных опций для текущей модели объекта, на рис. 7.13 представлено меню проекта Plastic для модели Elephant (Слон).

В контекстном меню, изображенном на рис. 7.13, присутствуют следующие команды:

- **Change Model** — выбор загружаемой модели из указанного каталога;
- **Orientation** — предоставляет возможность выбора ориентации координатной системы для текущей модели с помощью выпадающего меню и команд **RHS Coordinate System** (Правосторонняя система координат, применяемая в OpenGL) и **LHS Coordinate System** (Левосторонняя система координат, применяемая в DirectX);
- **Add Note** — добавляет запись или заметку для выбранной модели объекта;
- **Rename** — переименовывает выбранную модель;
- **Cut** — вырезает модель;
- **Copy** — копирует модель в буфер обмена;
- **Paste** — вставляет модель из буфера обмена;
- **Delete** — удаляет модель;
- **Edit** — редакция модели;
- **Edit With** — редакция модели с помощью следующих команд вторичного меню:
  - **Model Loader** — загружает модель в форматах 3DS, X, NMF и OBJ;
  - **External File Editor** — по умолчанию запускает утилиту MeshView из состава DirectX 9 SDK, необходимую при работе с форматом X-файлов, на рис. 7.14 показано рабочее окно утилиты MeshView для проекта Plastic и модели Elephant.

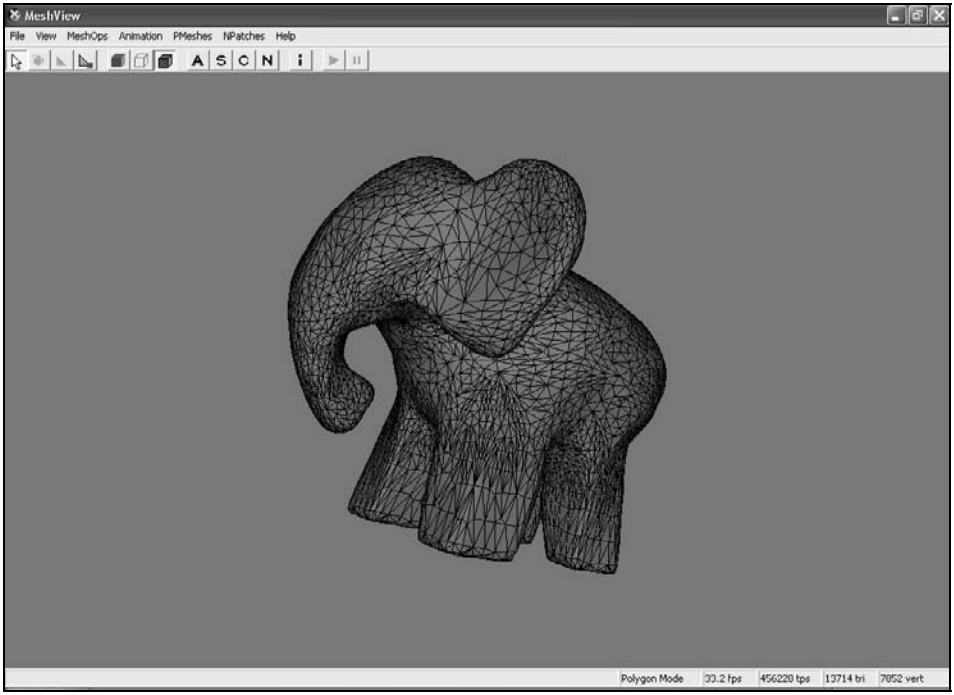


Рис. 7.14. Рабочее окно утилиты MeshView

- **Generator** — генерирует кубическую составляющую модели с помощью диалогового окна **Geometry Generator**, показанного на рис. 7.15;
- **Save** — сохраняет модель в формате X-файла.

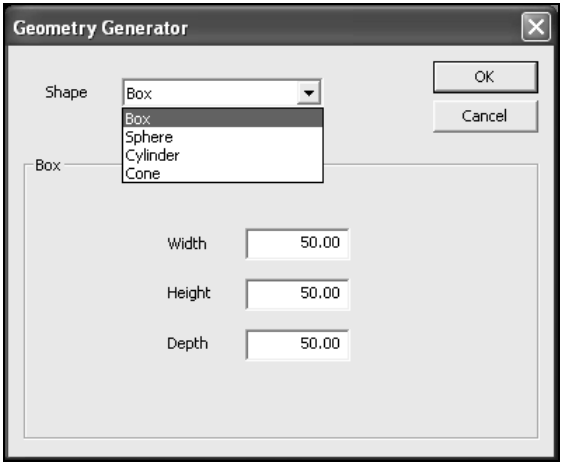


Рис. 7.15. Диалоговое окно **Geometry Generator**

При наведении курсора на любой узел текущего проекта в окне **Workspace** автоматически на несколько секунд появляется всплывающее меню с текстовой заметкой для данного узла. В зависимости от типа данных текст заметки содержит различные описания узла. Если вы поочередно наведете курсор на узлы любого проекта, то увидите появляющиеся заметки для каждого типа данных. Также имеется возможность редакции или добавления своих текстовых заметок для всех узлов проекта. Для этого нажмите правой кнопкой мыши на необходимом узле и в появившемся меню выберите команду **Add Note** (Добавить узел). В рабочем пространстве RenderMonkey появится небольшой по размеру текстовый редактор для добавления необходимых заметок по выбранному узлу, на рис. 7.16 изображен текстовый редактор в проекте Plastic для модели Elephant.

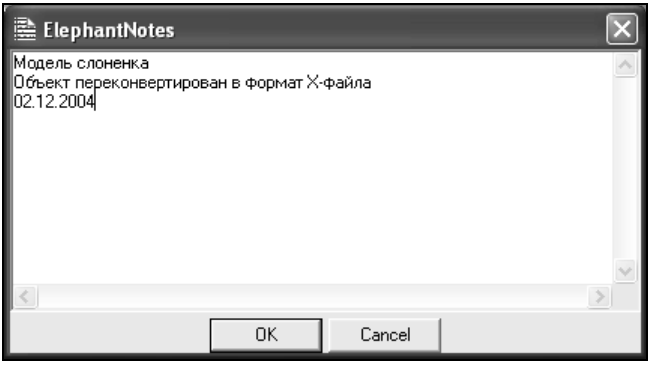


Рис. 7.16. Текстовые записи для модели Elephant

Camera Nodes

Узлы камер (Camera Nodes) необходимы для определения ориентации вида и играют ту же роль, что и камеры в простой трехмерной сцене. В одном проекте может быть несколько различных камер. Все камеры должны располагаться в родительской ветке иерархии проекта. Добавление камеры в текущий проект осуществляется следующим образом: нажмите на названии проекта в окне **Workspace** правую кнопку мыши и в появившемся меню выберите команду **Add Camera** (Добавить камеру). На рис. 7.17 изображено окно **Workspace** проекта TechnoPumpingSpeaker из стандартных примеров инструментария RenderMonkey с добавленной новой камерой под названием **MyCamera**.

В отдельный промежуток времени в проекте может быть активна только одна камера. Для того чтобы сделать камеру активной, нажмите правую кнопку мыши на названии необходимой камеры в окне **Workspace** и в появившемся контекстном меню выберите команду **Set Active Camera** (Акти-

визировать установленную камеру), после этого выбранная камера станет основной. При этом активная предыдущая камера будет сброшена. Камеру можно связывать с проходами рисования посредством узла **References Camera**. Для того чтобы добавить камеру в проход рисования, нажмите правую кнопку мыши на названии прохода рисования в окне **Workspace** и в появившемся меню выберите команду **Add Camera References** (Добавить ссылку на камеру), появится еще одно подменю, где перечисляются доступные для этого проекта камеры. На рис. 7.18 показано меню проекта TechnoPumpingSpeaker со списком доступных камер.

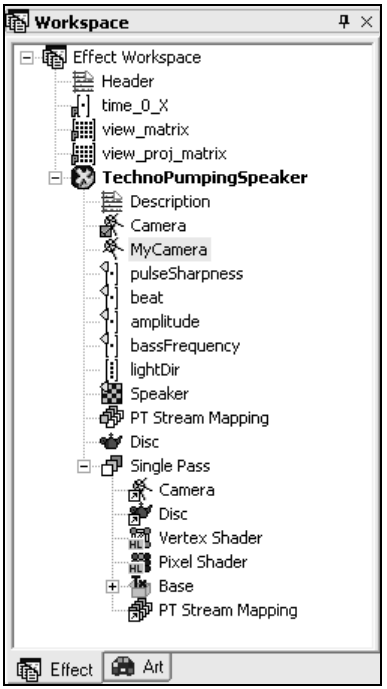


Рис. 7.17. Окно **Workspace** проекта TechnoPumpingSpeaker

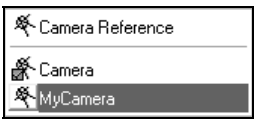


Рис. 7.18. Меню проекта TechnoPumpingSpeaker

Если в проходе рисования уже существует узел **References Camera** (Ссылка на камеру), связанный с одной из камер, то по необходимости вы можете связывать узел **References Camera** с любой другой камерой, имеющейся в текущем проекте. Чтобы связать узел **References Camera** с другой камерой, щелкните правой кнопкой мыши на названии узла **References Camera** в одном из проходов рисования в окне **Workspace**, а в появившемся меню выберите команду **References Node** (Ссылка на узел). Появится дополнительное меню со списком доступных камер. Таким образом происходит привязка камеры к одному из проходов рисования. Для всех узлов **Camera Nodes** и **References Camera** проекта доступны команды **Add Note** (Добавить примечание), **Rename** (Переименовать), **Cut** (Вырезать), **Copy** (Копировать), **Paste** (Вставить) и **Delete** (Удалить) через контекстное меню проекта.

Для коррекции параметров настройки имеющихся камер в проекте используется редактор камеры (Camera Editor). Двойной щелчок левой кнопки мыши на названии камеры в окне **Workspace** вызывает его. На рис. 7.19 изображен стандартный редактор свойств камеры.

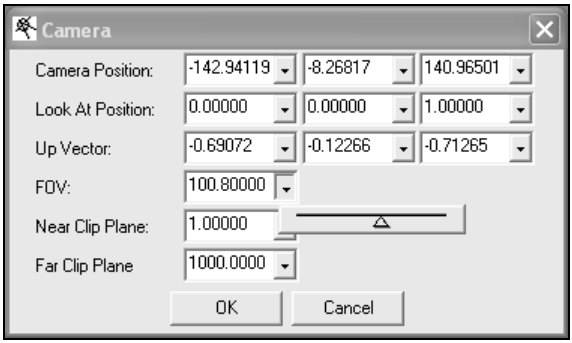


Рис. 7.19. Редактор камеры

Изменение свойств камеры доступно с помощью параметров редактора, выполненных в виде линейных регуляторов (slider или так называемый trackbar), где вы можете задавать различные значения параметрам. Редактор камеры имеет следующие параметры для настройки свойств камеры:

- **Camera Position** — определяет позицию камеры по осям X, Y и Z для объекта трехмерной сцены;
- **Look At Position** — задает направление просмотра по осям X, Y и Z для объекта трехмерной сцены;
- **Up Vector** — вектор просмотра, задается по осям X, Y и Z;
- **FOV** (Field Of View) — устанавливает поле зрения для всей сцены;
- **Near Clip Planes** — ближний план отсечения;
- **Far Clip Planes** — дальний план отсечения.

После изменения настроек параметров камеры нажмите кнопку **OK** в редакторе камеры, а для просмотра соответствующих изменений в сцене нажмите клавишу <Space>, тогда в окне просмотра (**Preview Window**) отобразятся произошедшие модификации проекта.

Stream Mapping Nodes

Выборка вершинных потоков (Stream Mapping Nodes) описывает характеристики модели, которая идет через проход рисования. Узел **Stream Mapping** может быть создан на любом из уровней древовидной иерархии текущего проекта. Для этого щелкните правой кнопкой мыши на названии проекта в

окне **Workspace** инструментария RenderMonkey и в появившемся меню выберите команду **Add Stream Mapping** (Добавить выборку вершинных потоков). Выборку вершинных потоков необходимо связать с проходом рисования с помощью ссылки **Stream Mapping References** для отображения выборки в одном из проходов рисования. Для связывания **Stream Mapping** с проходом рисования нажмите правую кнопку мыши над названием **Stream Mapping References** (Ссылка на выборку вершинных потоков) в необходимом проходе рисования и в появившемся меню выберите команду **Reference Node** (Ссылка на узел). В появившемся выпадающем меню будут представлены все имеющиеся в проекте выборки вершинных потоков, выберите нужную, выделив название выборки. Также можно просто перетащить мышью узел **Stream Mapping** из родительского эффекта на узел **Stream Mapping References** в проходе рисования окна **Workspace**. В этом случае произойдет автоматическое связывание выборки вершинных потоков с одним из проходов рисования.

Для изменения значений выборки вершинных потоков необходимо задействовать специальный редактор (Stream Mapping Editor), изображенный на рис. 7.20. Двойной щелчок левой кнопки мыши на названии выборки в проекте откроет окно редактора выборки вершинных потоков (Stream Mapping Editor).

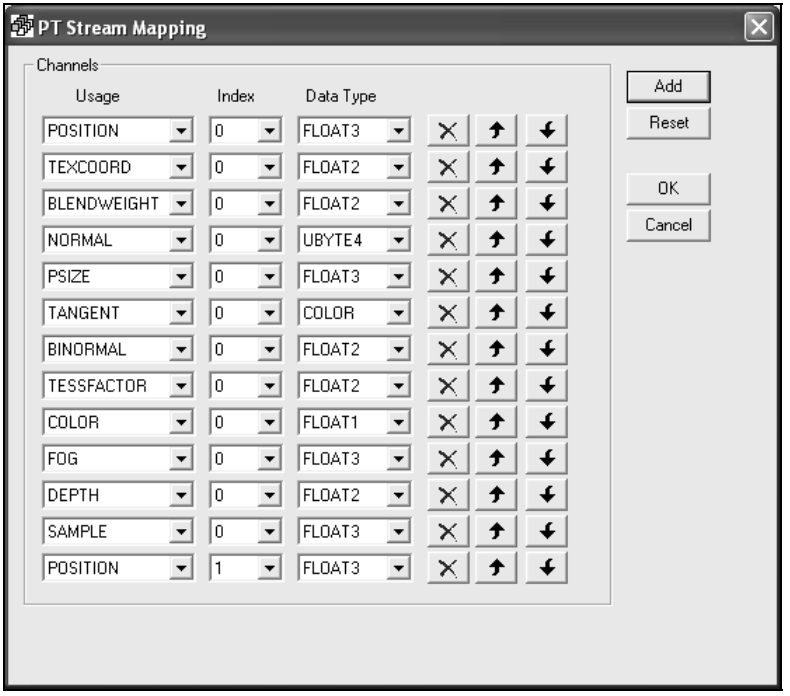


Рис. 7.20. Редактор выборки вершинных потоков

С помощью Stream Mapping Editor добавляются новые каналы потока или редактируются уже имеющиеся. Для добавления нового канала нажмите кнопку **Add** (Добавить) в редакторе, а для сброса значений к первоначальному состоянию нажмите кнопку **Reset** (Сброс). Каждый канал потока имеет три параметра — **Usage** (Используемые значения), **Index** (Индекс) и **Data Type** (Тип данных), — выполненные в виде списка с набором жестко заданных значений, как показано на рис. 7.20. При выборе нужных значений из списка осуществляется редакция выборки вершинных потоков. В списке параметра **Usage** содержится множество значений, приведем некоторые из них:

- **POSITION** — позиция;
- **NORMAL** — нормаль;
- **TEXCOORD** — текстурные координаты;
- **TANGENT** — пространство касательных;
- **BINORMAL** — бинормаль;
- **COLOR** — цвет;
- **FOG** — туман;
- **DEPTH** — глубина.

В следующем параметре **Index** задается индекс для канала выборки, а в параметре **Data Type** указывается применяемый тип данных. Напротив каждого канала в редакторе выборки вершинных потоков существуют дополнительные три кнопки с изображением крестика и двух стрелок вверх и вниз, как показано на рис. 7.20. С помощью двух кнопок с изображениями стрелок производится сортировка каналов выборки соответственно вверх и вниз, а кнопкой с графическим элементом крестик канал выборки удаляется. Для того чтобы все произведенные изменения вступили в силу, нажмите кнопку **OK** в окне редактора выборки вершинных потоков.

Texture Nodes

Узлы текстур (Texture Nodes) позволяют использовать текстуры в создаваемой сцене. RenderMonkey поддерживает следующие виды текстур:

- 2D Texture — двухмерные текстуры форматов BMP, IPG, PNG, JPEG, TGA;
- Cubemap Texture — кубические текстуры формата DDS;
- 3D Texture — объемные трехмерные текстуры формата DDS;
- Renderable Texture — динамические текстуры.

Для добавления текстуры в проект щелкните правой кнопкой мыши на названии проекта в окне **Workspace** и в появившемся меню выберите команду

Add Texture. Откроется дополнительное меню с перечисленными четырьмя типами доступных в RenderMonkey текстур, как показано на рис. 7.21.

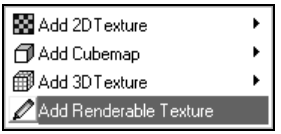


Рис. 7.21. Дополнительное меню **Add Texture**

В дополнительном меню **Add Texture** (Добавить текстуру) имеются четыре команды: **Add 2D Texture**, **Add Cubemap**, **Add 3D Texture** и **Add Renderable Texture**. При выборе первых трех команд открываются вторичные меню с набором текстур, поставляемых с RenderMonkey из каталога ATI Research Inc\RenderMonkey 1.5\ Examples\Media\Texture, где можно выбрать ряд текстур для своего проекта. Если вас не устроит набор имеющихся текстур, то всегда можно загрузить в каталог ATI Research Inc\RenderMonkey 1.5\ Examples\Media\Texture свои текстуры, либо в настройках RenderMonkey указать свой каталог для загрузки необходимых текстур. Для этого в меню RenderMonkey выберите команду **Edit | Preferences** (Редактировать | Свойства) и в появившемся диалоговом окне **RenderMonkey Preferences** перейдите на вкладку **General**, изображенную на рис. 7.22. В поле **Textures** укажите путь к каталогу с текстурами и при выборе команд **Add 2D Texture**, **Add Cubemap** и **Add 3D Texture** в меню **Add Texture** загрузка текстур будет осуществляться из необходимого каталога.

Все текстуры проекта могут использоваться с различными эффектами, также доступна возможность редакции текстуры по своему усмотрению с помощью контекстного меню, появляющегося при нажатии правой кнопки мыши на нужной текстуре. Контекстное меню редактирования текстур, показанное на рис. 7.23, содержит кроме рассмотренных ранее в этой главе команд **Add Note** (Добавить примечание), **Rename** (Переименовать), **Cut** (Вырезать), **Copy** (Копировать), **Paste** (Вставить), **Delete** (Удалить), **Edit** (Редактировать), **Edit With** (Редактировать с помощью) и **Save** (Сохранить) еще ряд специализированных команд для работы с текстурами.

Контекстное меню редактирования текстур содержит встроенную систему просмотра текстур **Preview**. В том случае если используется большое количество различных текстур, система просмотра **Preview** упрощает работу с ними.

Имеются следующие дополнительные текстурные команды в контекстном меню редактирования текстур:

- **Artist Variable** — выбор этой команды дает возможность текстурному художнику редактировать текстуру, но уже на вкладке **Art** окна **Workspace**. Избрав команду **Artist Variable**, вы подключаете текстуры на вкладку **Art**

окна **Workspace**, при этом в графическую пиктограмму на вкладке **Effect** окна **Workspace** добавляется небольшой по размеру желтый треугольник, сигнализирующий о подключении текстуры;

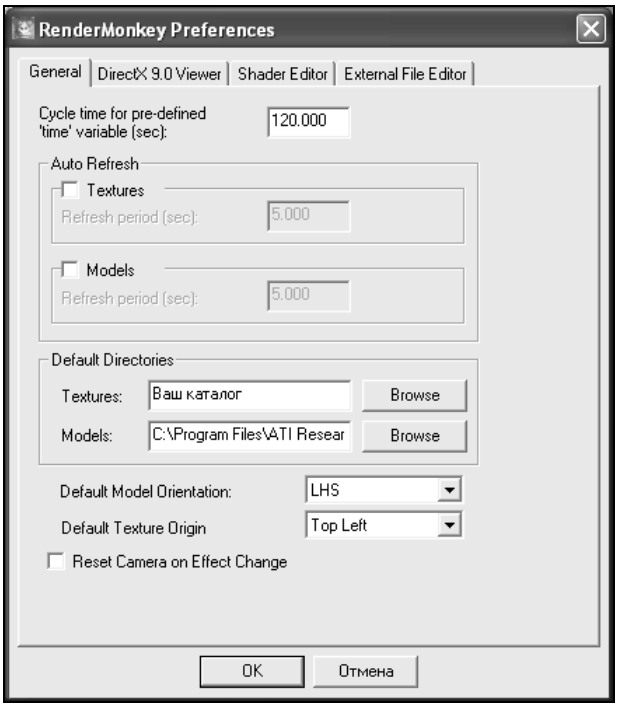


Рис. 7.22. Диалоговое окно **RenderMonkey Preferences**, новые текстуры

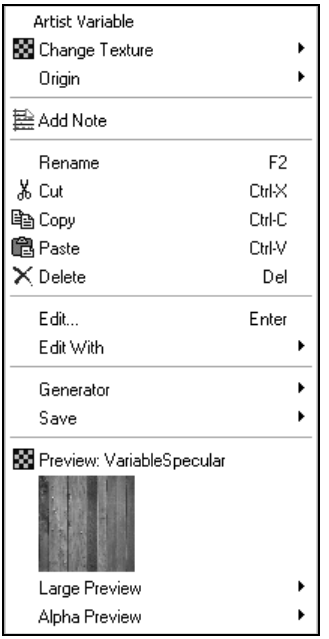


Рис. 7.23. Контекстное меню редактирования текстур

- **Change Texture** — выбор текстур из указанного каталога;
- **Origin** — выбор текстурной системы координат с помощью двух вторичных команд:
  - **Bottom Left Origin** — начало текстурных координат в левой нижней точке;
  - **Top Left Origin** — начало текстурных координат в правой верхней точке;
- **Generator** — генератор текстур, основанный на работе языка DirectX HLSL;
- **Large Preview** — это вид выбранной текстуры, отображенный в большой области просмотра, на рис. 7.24 представлена область просмотра **Large Preview** с текстурой VariableSpecular из каталога ATI Research Inc \RenderMonkey 1.5\ Examples\Media\Texture инструментария RenderMonkey.

В **Large Preview**, кроме просмотра самой текстуры, также дается полная информация о размере, месте нахождения и формате текстуры;

- **Alpha Preview** — просмотр альфа-составляющей текстуры.

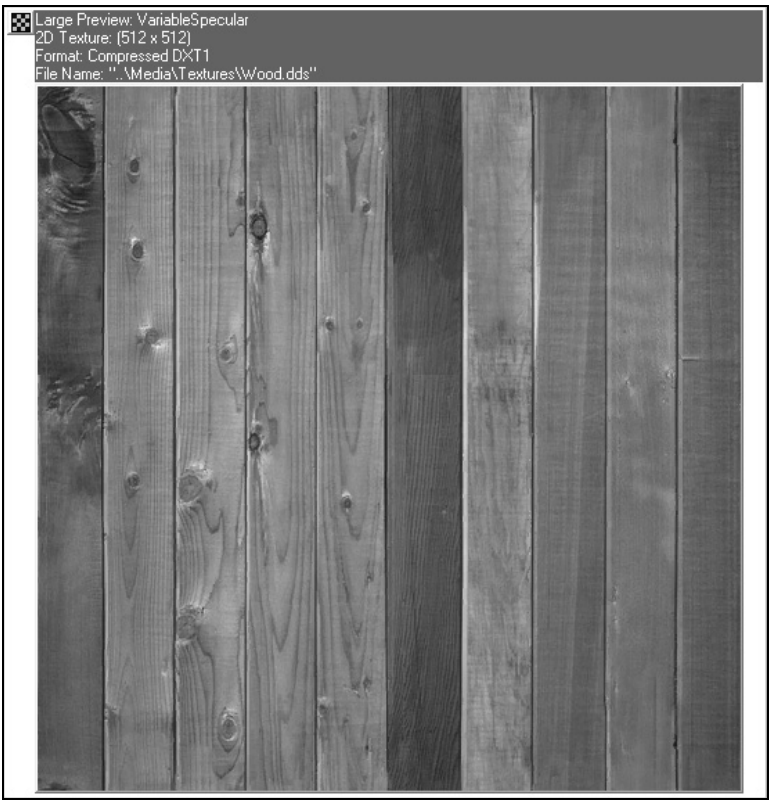


Рис. 7.24. Область просмотра **Large Preview**

Генератор текстур (**Generator**), доступный из контекстного меню редактирования текстур, позволяет программисту процедурно генерировать текстуры с использованием высокоуровневого языка шейдеров DirectX HLSL. Имеется возможность создания текстур типов 2D Texture, Cubemap Texture и 3D Texture. Для того чтобы сгенерировать текстуру процедурно, нажмите правой кнопкой мыши на названии текстуры в текущем проекте окна **Workspace** и в контекстном меню редактирования текстур выберите команду **Generator** (Генерация). Появится вторичное меню с командой **Procedural Texture generator**, выбрав которую вы откроете окно генератора текстур (**Procedural Texture generator**), показанное на рис. 7.25.

Окно **Procedural texture generator** разделено на две области. Верхняя часть выполнена в виде статической области с набором различных элементов

управления для задания свойств создаваемой текстуры. Нижняя область — это текстовый редактор для написания кода шейдеров на языке HLSL, имеющий интерактивную подсветку синтаксиса языка HLSL, что значительно упрощает написание программного кода шейдеров.

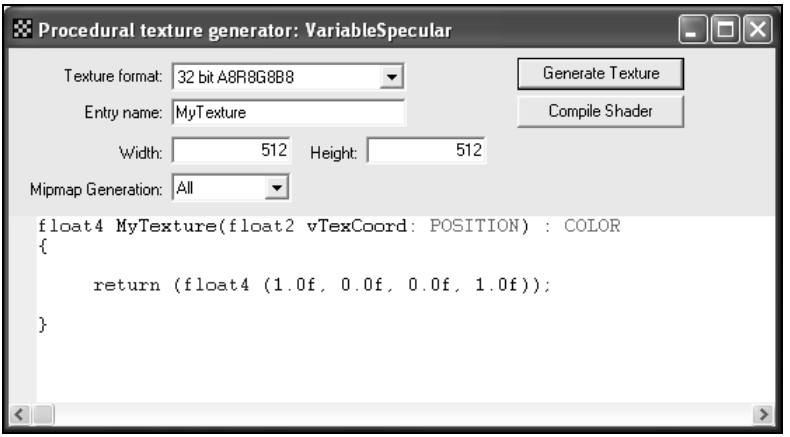


Рис. 7.25. Окно процедурного генератора текстур для 2D Texture

Для каждого типа процедурно сгенерированных текстур (а это 2D Texture, Cubemap Texture и 3D Texture) в статической области редактора **Procedural Texture Generator** изменяется набор элементов управления, рассмотрим имеющиеся элементы, подразделяя их для каждого типа создаваемых текстур:

- **Texture format** — список, содержащий перечень доступных форматов текстур и определяющий формат выходной текстуры. Доступен для всех типов текстур;
- **Entry name** — текстовое поле для задания имени основной функции генерации текстуры. Доступно для всех типов текстур;
- **Width** — текстовое поле для задания ширины размера всей текстуры. Доступно для типов 2D Texture и 3D Texture;
- **Height** — текстовое поле для задания высоты размера всей текстуры. Доступно для типов 2D Texture и 3D Texture;
- **Depth** — текстовое поле для задания глубины текстуры. Доступно в типе 3D Texture;
- **Edge size** — текстовое поле для задания размера кубической текстуры. Доступно в типе Cubemap Texture;
- **Mipmap Generation** — список генерации Mipmap-уровней. Доступен только для типов 2D Texture;

- **Generate Texture** — кнопка, дающая команду для генерации текстуры. Доступна для всех типов текстур;
- **Compile Shader** — кнопка компиляции исходного кода шейдеров, написанных на языке HLSL. Доступна для всех типов текстур.

Динамически созданные текстуры (Renderable Texture) добавляются непосредственно в древовидную структуру проекта и не могут быть созданы в проходах рисования через меню **Add Texture**. Для редакции этого вида текстур применяется специальный редактор — Renderable Texture Editor, показанный на рис. 7.26. В редакторе динамически созданных текстур производятся изменения их свойств, в том числе размера, формата текстуры и автогенерация Mipmap-уровней. Вызов редактора производится двойным нажатием левой кнопки мыши на названии динамически созданной текстуры в окне **Workspace**.

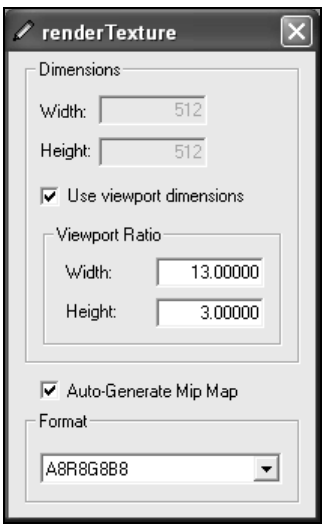


Рис. 7.26. Редактор динамически созданных текстур

Texture Object Nodes

Узлы текстурных объектов (Texture Object Nodes) добавляются к любому проходу рисования и включают установки фильтрации текстур для определения необходимой текстуры или состояния в проходе рисования. Каждый текстурный объект обязан содержать ссылку на текстурную переменную для сэмплинга, т. е. должен быть связан с проходом рисования посредством ссылки **Texture References**. Чтобы добавить в проект **Texture Object**, щелкните правой кнопкой мыши на названии одного из проходов рисования в ок-

не **Workspace** и в появившемся меню выберите команду **Add Texture Object** (Добавить текстурный объект). В структуре проекта в избранном проходе рисования появится новый текстурный объект. На рис. 7.27 изображено окно **Workspace** инструментария RenderMonkey с открытым проектом Render To Texture и с созданным новым текстурным объектом под названием NewTextureObject в проходе рисования Scrolling Render Target Result.

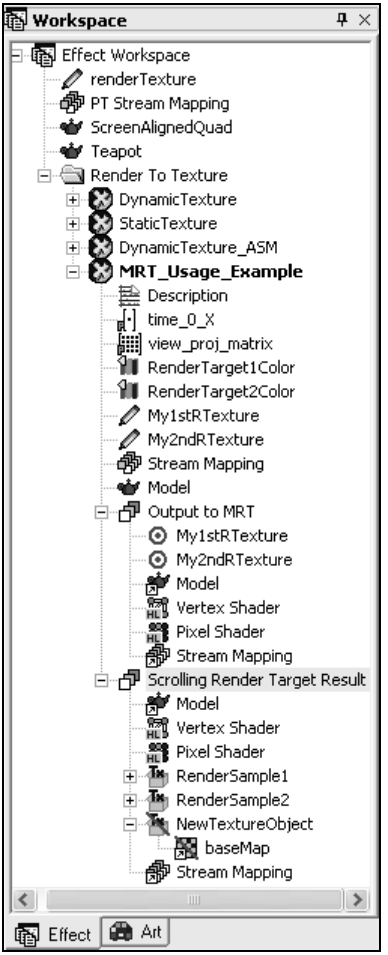


Рис. 7.27. Окно **Workspace** проекта Render To Texture

Изначально созданный текстурный объект не связан ни с одной из *текстурных ссылок* (**Texture References**), поэтому графическая иконка узла текстурного объекта перечеркнута красной косой чертой. В этой ситуации **Texture Object** наследует текстуры от высшего элемента в активном эффекте или от текстуры, заданной по умолчанию в RenderMonkey. Связывание

Texture Object можно производить со всеми доступными типами текстур в RenderMonkey: 2D Texture, Cubemap Texture и 3D Texture. Для связывания **Texture Object** с текстурой через ссылку **Texture References** необходимо раскрыть структуру созданного узла **Texture Object**, нажав слева от пиктограммы текстурного объекта на графический элемент, выполненный в виде квадратика с плюсом. В раскрывшейся структуре **Texture Object** будет содержаться ссылка **Texture References**, нажав правой кнопкой мыши на названии ссылки появится меню, выберите в этом меню команду **Reference Node** (Ссылка на узел). Откроется подменю с набором доступных текстур эффекта, на рис. 7.28 изображено вторичное меню проекта Render To Texture в проходе рисования Scrolling Render Target Result. Таким методом происходит связывание текстурных объектов с текстурами при помощи **Texture References**.

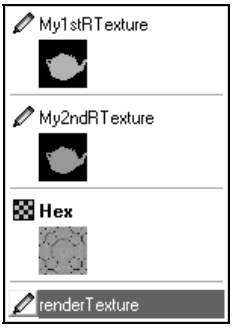


Рис. 7.28. Подменю в проекте Render To Texture

Для полноценной редакции текстурных узлов в RenderMonkey применяется редактор текстурных состояний (Texture State Editor), изображенный на рис. 7.29. Этот редактор отражает все текстурные состояния в пределах одного эффекта. Рабочее окно редактора разделено на две области по горизонтали, что хорошо видно на рис. 7.29.

В верхней области окна располагается система просмотра задействованных в проекте текстур, которые представлены в виде небольших образцов текстур. Нижняя область рабочего окна редактора имеет три столбца. В первом столбце **State** (Состояние) перечислены текстурные состояния. В следующем столбце **Value** (Переменная) происходит определение значения для текстурных состояний. Столбец **Incoming** (Входящий) устанавливает входящее значение, которое было определено в предыдущем проходе или в заданном по умолчанию эффекте. Изменение значений производится с помощью мыши. Щелчок левой кнопкой мыши в столбце **Value** напротив одного из значений столбца **State** откроет контекстное меню с набором доступных для выбранного текстурного состояния значений (см. рис. 7.29). Выбрав из меню необходимые параметры, нажмите клавишу <Enter>.

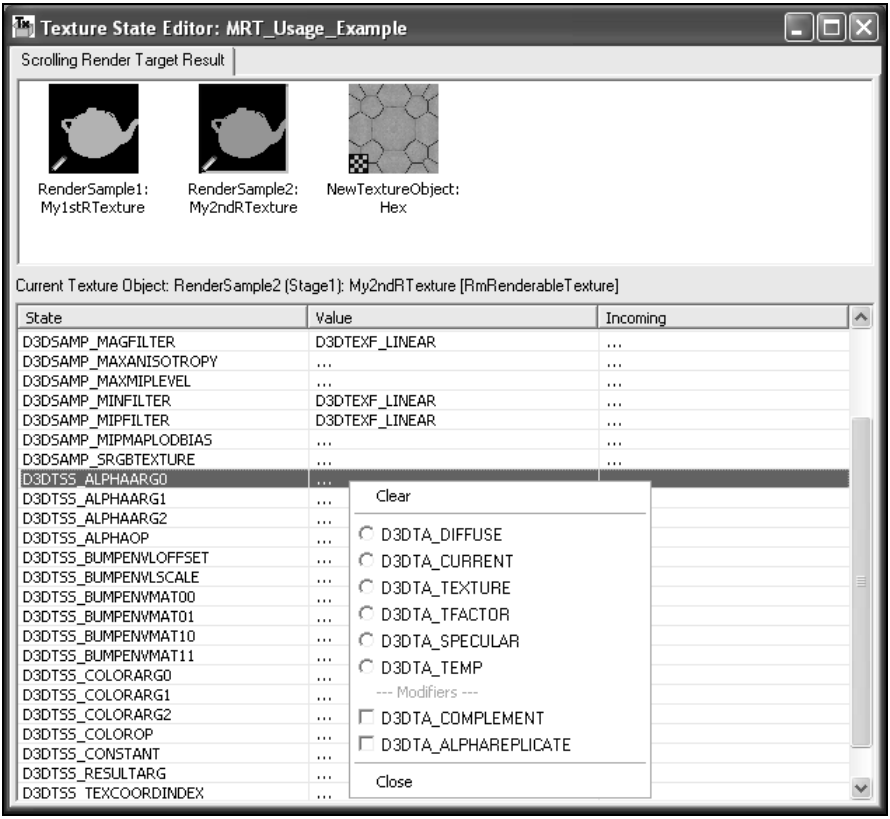


Рис. 7.29. Редактор текстурных состояний

Render State Block Nodes

Блок установок рисования (Render State Block) дает возможность задавать доступные в API установки состояния при рендеринге. Блок установок рисования должен добавляться к одному из проходов рисования, для этого нажмите правой кнопкой мыши на названии прохода рисования в текущем проекте окна **Workspace** и в появившемся меню выберите команду **Add Render State Block** (Добавить блок установок рисования). При создании **Render State Block** происходит наследование предыдущего блока в проходе рисования либо любого другого блока, найденного в активном эффекте. В том случае, если ни один из блоков установок рисования не был задан ранее, произойдет наследование от эффекта, заданного по умолчанию в RenderMonkey.

Для просмотра и изменения установок **Render State Block** воспользуйтесь редактором, доступным в RenderMonkey. Двойной щелчок правой кнопки

мыши на названии блока установок рисования в окне **Workspace** вашего проекта раскроет окно соответствующего редактора (Render State Editor), представленного на рис. 7.30.

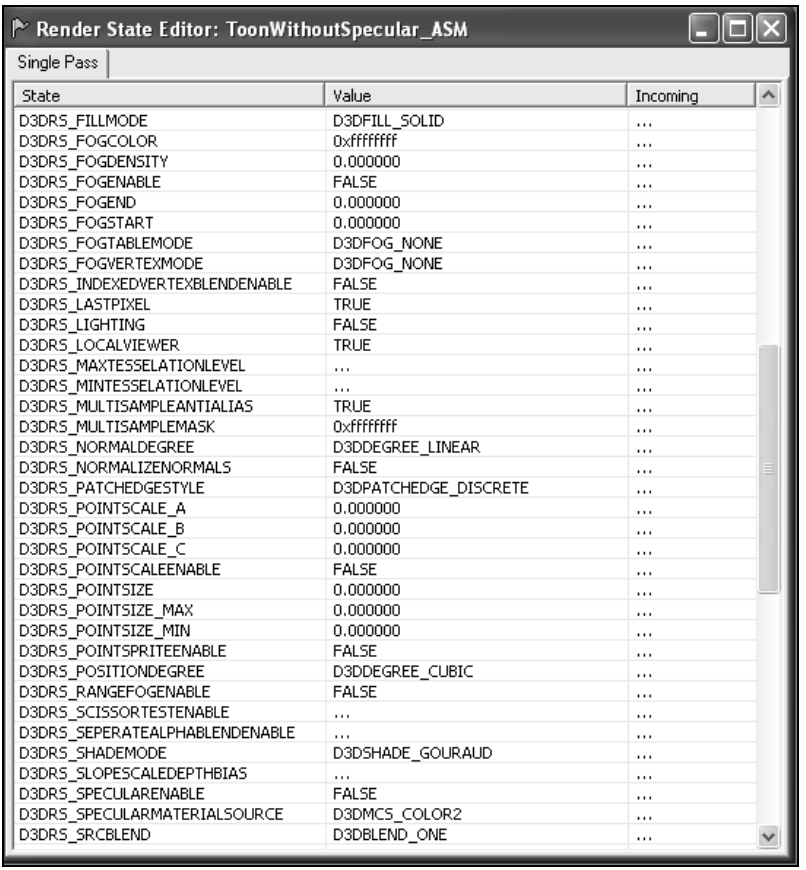


Рис. 7.30. Редактор установок рисования

Редактор отображает все существующие блоки установок рисования в пределах одного эффекта. Видоизменение значений в блоке установок рисования также может происходить в пределах одного эффекта.

В окне редактора содержится три столбца, их можно увидеть на рис. 7.30. Столбец **State** (Состояние) задает имя для каждого доступного состояния. Столбец **Value** (Переменная) определяет текущее значение состояний. Столбец **Incoming** (Входящий) определяет любое входящее значение, которое было установлено в предыдущем проходе или в заданном по умолчанию эффекте. Вы можете изменять порядок сортировки перечисленных значений с помощью мыши, перетаскивая значения в нужное место.

Двойной щелчок в поле **Value** напротив одного из значений столбца **State** автоматически инициализирует редактирование выбранного состояния. В том случае, если это числовое значение, вам будет дана возможность производить установки для сопутствующих числовых значений. Если же эти значения заданы в виде констант DirectX, определяющих состояния рендеринга, то их можно изменять с помощью появляющегося контекстного меню.

Render State Editor — достаточно мощное интерактивное средство, позволяющие напрямую устанавливать необходимые значения для блока установок рисования.

Render Target Nodes

С помощью *узла цель рисования* (Render Target Node) достигается переадресация заданной текстуры в один из проходов рисования, что позволяет выполнить рисование в динамически созданную текстуру и при этом не задействовать задний буфер (back buffer). Узел **Render Target** добавляется в любой из проходов рисования проекта. Для его создания нажмите правой кнопкой мыши на названии нужного прохода рисования в окне **Workspace** и в появившемся контекстном меню выберите команду **Add Render Target** (Добавить цель рисования). Появится подменю с перечислением доступных динамически созданных текстур проекта. На рис. 7.31 изображено подменю прохода рисования Object Pass в окне **Workspace** проекта Differed Shading.

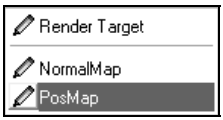


Рис. 7.31. Подменю **Render Target**

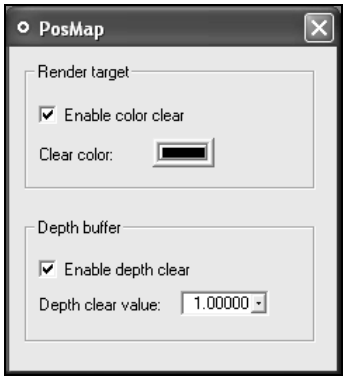


Рис. 7.32. Редактор цели рисования

Выбрав из вторичного меню название текстуры, вы совершите переадресацию текстуры в проход рисования Object Pass в окне **Workspace** проекта Differed Shading, обо всем остальном позаботится RenderMonkey. Для просмотра видоизменений проекта нажмите клавишу <Space>, и в окне **Preview** отобразится измененная сцена проекта.

Так же, как и все узлы в RenderMonkey, узел **Render Target** может быть отредактирован с помощью специального редактора — Render Target Editor, изображенного на рис. 7.32. Редактор содержит всего две опции: для очистки текстуры цветом (**color clear**) и для очистки буфера глубины (**depth clear**).

Шейдеры

Инструментарий RenderMonkey поддерживает все имеющиеся на день написания книги версии вершинных и пиксельных шейдеров. Более того, RenderMonkey осуществляет поддержку шейдерных языков программирования как для DirectX, так и для OpenGL, предоставляя мощные интеллектуальные текстовые редакторы для написания исходного кода шейдеров. Работа с шейдерами в RenderMonkey строится традиционным образом, т. е. на основе узлов. Могут быть задействованы следующие узлы:

- Vertex Shader Nodes;
- Pixel Shader Nodes.

Vertex Shader Nodes

Узел вершинного шейдера (Vertex Shader Node) в проекте представляет собой программный код вершинного шейдера. Узел **Vertex Shader** добавляется в проект в любой из необходимых проходов рисования. Чтобы добавить узел **Vertex Shader** в проект, щелкните правой кнопкой мыши на названии прохода рисования в окне **Workspace** рабочего проекта и в появившемся контекстном меню выберите одну из команд: **Add Vertex Shader** (Добавить вершинный шейдер) или **Add Vertex Program** (Добавить программу шейдера). В том случае, если вы работаете с DirectX, появится подменю с двумя пунктами: **DirectX ASM** (Ассемблерный код шейдеров в DirectX) и **DirectX HLSL** (Высокоуровневый язык программирования шейдеров в DirectX). Выбирая один из этих пунктов, вы предопределяете дальнейший ход работы либо с ассемблерным кодом вершинных шейдеров, либо с языком HLSL. Если вы работаете с OpenGL, то подменю команд **Add Vertex Shader** и **Add Vertex Program** будет содержать один пункт **OpenGL GLSL** (Язык программирования шейдеров в OpenGL). При выборе соответствующих команд из подменю в проекте в необходимый проход рисования будет добавлен узел для работы с вершинными шейдерами.

Созданный узел **Vertex Shader** может быть отредактирован с помощью стандартных команд **Rename** (Переименовать), **Cut** (Вырезать), **Copy** (Копировать), **Paste** (Вставить), **Delete** (Удалить), **Edit** (Редактировать), **Print Shader** (Напечатать шейдер) и **Add Note** (Добавить примечание), появляющихся в виде контекстного меню при щелчке правой кнопкой мыши на названии узла в окне **Workspace** текущего проекта.

При выборе команды **Edit** в контекстном меню узлов **Vertex Shader** автоматически открывается окно текстового редактора шейдеров — Shader Editor. Вид рабочего окна редактора шейдеров варьируется в зависимости от языка шейдеров, с которым вы работаете, т. е. DirectX ASM, DirectX HLSL или OpenGL GLSL. На рис. 7.33—7.35 показаны все три редактора шейдеров, имеющие заметные различия, зависящие от задействованной версии шейдеров.

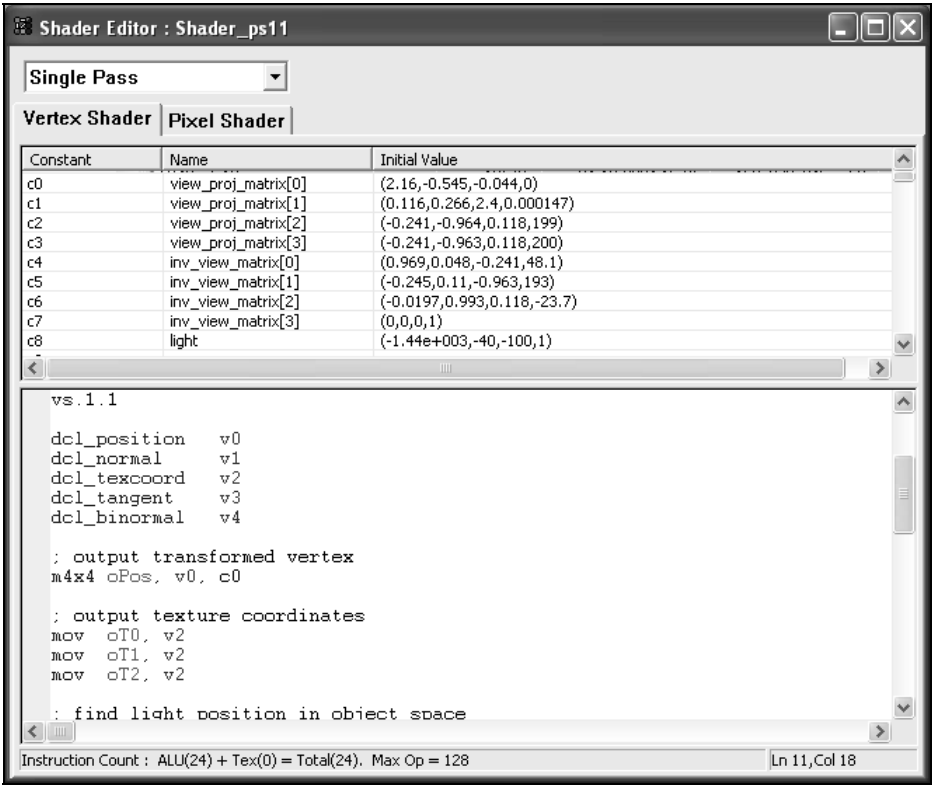


Рис. 7.33. Окно редактора шейдеров для DirectX ASM

Окно редактора ассемблерного кода шейдеров (ASM), изображенное на рис. 7.33, разделено пополам на две рабочие области. Верхняя часть окна редактора содержит константный редактор регистров программы шейдеров, выполненный в виде таблицы с тремя столбцами. Каждая строка таблицы показывает значения для одного из регистров шейдера.

Первый слева столбец **Constant** (Константа) в константном редакторе регистров содержит допустимый индекс одного из регистров шейдера. Во втором столбце **Name** (Имя) прописывается имя узла или переменной, с которой связан данный регистр. И в последнем столбце **Initial Value** (Значение

при инициализации) содержится начальное значение узла или переменной, связанной с регистром шейдеров. Таблица константного редактора интерактивна и связывание регистра с переменной или узлом происходит прямо в редакторе. Щелкнув левой кнопкой мыши в столбце **Name** (Имя) в строке напротив нужного значения индекса регистров, откроется контекстное меню с набором доступных для этого регистра значений. Выбрав в меню значение, вы тем самым свяжите регистр с одним из узлов или переменной. Также верхняя часть окна редактора ассемблерного кода шейдеров содержит список с перечислением всех проходов рисования рабочего проекта и две вкладки: **Vertex Shader** (Вершинный шейдер) и **Pixel Shader** (Пиксельный шейдер). С помощью этих вкладок вы переключаетесь между проходами рисования и текстовыми редакторами вершинных и пиксельных шейдеров.

Нижняя область окна редактора ассемблерного кода шейдеров — это текстовый редактор для написания исходного кода как вершинных, так и пиксельных шейдеров, обладающий системой распознавания синтаксиса и возможностью редакции кода.

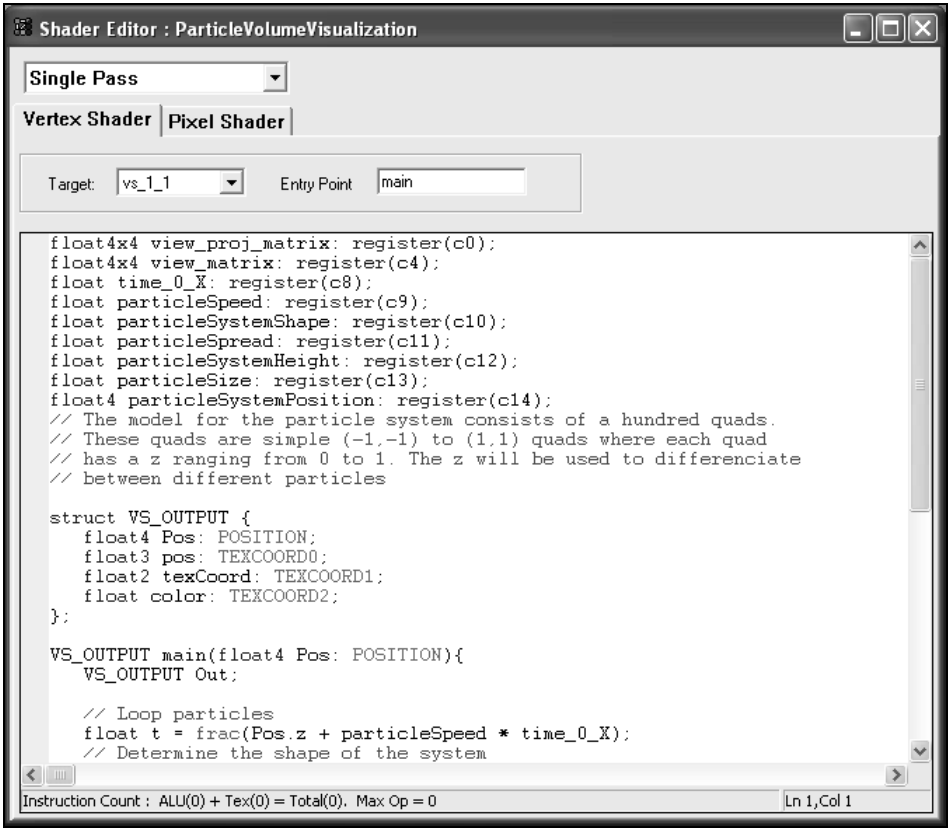


Рис. 7.34. Окно редактора шейдеров для DirectX HLSL

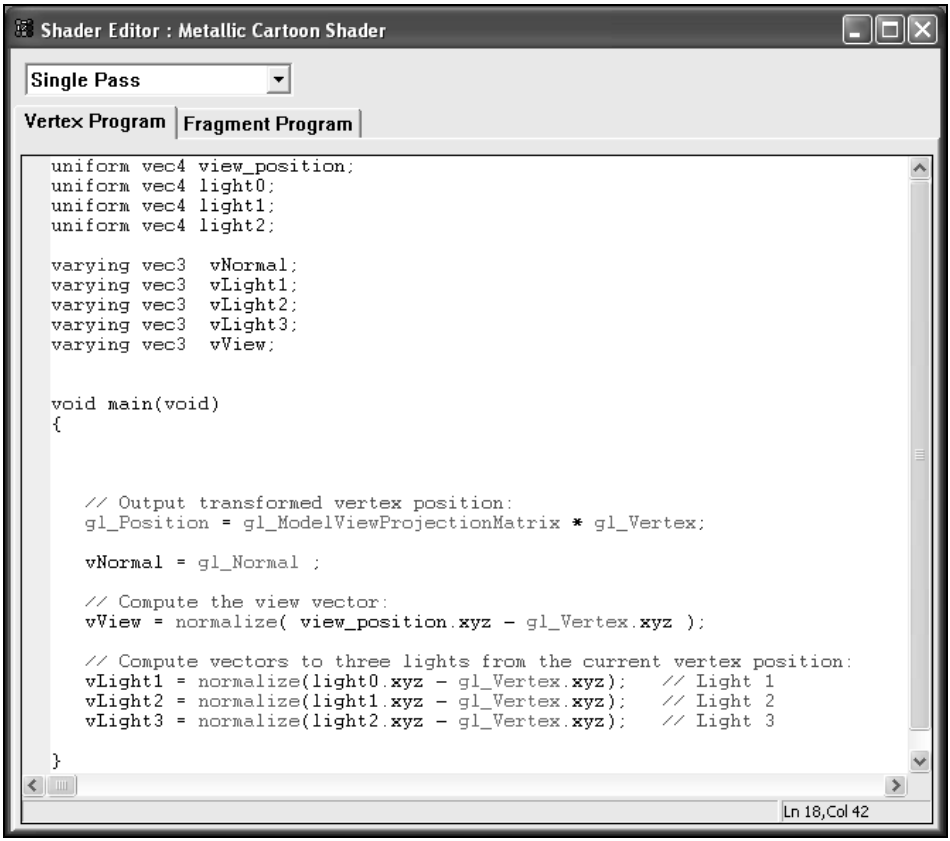


Рис. 7.35. Окно редактора шейдеров для OpenGL GLSL

Окно редактора DirectX HLSL, изображенное на рис. 7.34, предназначено для работы с высокоуровневым языком шейдеров (HLSL). Написание исходного кода происходит традиционным образом, как и в обыкновенном редакторе. Рабочее окно редактора DirectX HLSL также содержит вкладки **Vertex Shader** и **Pixel Shader** для переключения между кодом вершинного и пиксельного шейдера и включает в себя список с перечнем проходов рисования, именованный список **Target** (Цель) и текстовое поле **Entry Point** (Точка выхода). Список **Target** содержит объявление используемой версии шейдеров, а текстовое поле **Entry Point** задает начальную точку выхода для программы шейдера, т. е. в этом поле прописывается имя выходной функции в программе. Как правило, используется название `main`.

В RenderMonkey есть один приятный бонус при работе с исходным кодом языка HLSL — это операция **Disassembly** (Дизассемблирование). После успешной компиляции проекта в окне редактора DirectX HLSL щелкните правой кнопкой мыши на свободном пространстве и в появившемся кон-

текстном меню выберите команду **Show Disassembly Code** (Показать дизассемблированный код). Сразу в этом же окне код на языке HLSL будет ретранслирован в ассемблерный код шейдеров как для вершинных, так и для пиксельных шейдеров. И, конечно, редактор DirectX HLSL тоже имеет подсветку синтаксиса и поддерживает стандартные операции редактирования программного кода: **Copy** (Копировать), **Paste** (Вставить), **Save** (Сохранить), **Print** (Напечатать), **Find** (Найти) и **Delete** (Удалить).

Окно редактора OpenGL GLSL, показанное на рис. 7.35, предназначено для написания кода шейдеров на языке GLSL под OpenGL и имеет абсолютно аналогичные возможности, как и у редактора DirectX HLSL.

Для компиляции кода шейдеров в любом редакторе используются кнопки **Compile Active Effect** или **Compile All Shaders**, находящиеся на панели инструментов **RenderMonkey**.

Pixel Shader Nodes

Узлы пиксельных шейдеров (Pixel Shader Nodes), как вы наверно уже догадались, имеют идентичный функциональный набор, как и у узлов вершинного шейдера (Vertex Shader Nodes).

Но надо добавить еще несколько слов касательно редакторов шейдеров, рассмотренных в этом разделе. Все мы с вами люди, а люди способны делать ошибки, на которых и строится наша жизнь. При написании исходного кода шейдеров в любом из редакторов может случиться ошибка, и при компиляции проекта она соответственно всплывет. Все имеющиеся ошибки будут описаны в окне **Output**. При щелчке левой кнопкой мыши в окне **Output** на описании ошибки, в окне **Shader Editor** строка с ошибкой будет автоматически выделена синим цветом.

Variable Nodes

Переменные (Variable) предназначены для работы с данными внутри шейдеров, т. е. с регистрами, можно также задавать любые удобные имена для регистров. *Узлы переменных* (Variable Nodes) в RenderMonkey бывают нескольких типов.

- *Матрицы*. Поддерживаются следующие виды матриц:
  - `View projection` — видовая проекция;
  - `View` — видовая матрица;
  - `Inverse view` — инверсионный вид;
  - `Projection` — проекция.

- *Векторы*. Доступна поддержка следующих типов:
  - `View direction vector` — вектор направления обозревателя;
  - `View position vector` — вектор позиции камеры;
  - `Time` — время.
- *Значение цвета.*
- *Скалярные типы данных.*

И, конечно, RenderMonkey осуществляет полную поддержку значений `Boolean` (Булево значение), `Integer` (Целочисленное значение) и `Float` (Дробное значение) для типов данных.

Переменные могут добавляться в любое место структурной иерархии проекта. Чтобы создать новую переменную, нажмите правой кнопкой мыши на названии проекта и в появившемся контекстном меню выберите команду **Add Variable** (Добавить переменную). Появится подменю, изображенное на рис. 7.36, с перечнем команд, создающих необходимый тип переменных.

Рис. 7.36. Подменю команды **Add Variable**

Созданные переменные можно семантически (Semantic) связывать с данными шейдера. Для этого щелкните правой кнопкой мыши на созданной переменной и в появившемся меню выберите команду **Variable Semantic**. В появившемся подменю будет содержаться набор доступных данных для связывания. Также в этом меню имеется команда **Clear Semantic** (Очистить семантику) для очистки связанной переменной с данными шейдера.

Если вы работаете в группе с художниками и желаете, чтобы созданные переменные были доступны для редактирования художником, нажмите правой кнопкой мыши на необходимой переменной и в появившемся меню поставьте флажок напротив опции **Artist Variable** (Переменная для художника). В этом случае созданная переменная становится доступной для редактирования художником на вкладке **Art** (Художник) окна **Workspace**.

Для редакции созданных переменных в RenderMonkey используются удобные редакторы для каждого из типов данных:

- Matrix Editor (Редактор матриц);
- Vector Editor (Векторный редактор);

- Scalar Editor (Скалярный редактор);
- Color Editor (Редактор цвета).

А также редакторы для значений `Boolean` (Булево значение), `Integer` (Целочисленное значение) и `Float` (Дробное значение).

Чтобы открыть редактор для нужной переменной, щелкните два раза левой кнопкой мыши на названии переменной в окне **Workspace** текущего проекта. Рассмотрим доступные в RenderMonkey редакторы переменных.

Matrix Editor

Редактор матриц (Matrix Editor), изображенный на рис. 7.37, дает возможность устанавливать значение для матрицы 4×4 с плавающей точкой. Изначально значение матрицы устанавливается в диапазоне от −100.0 до 100.0, но вы можете задавать и любое другое значение вне этого диапазона. В рабочем окне редактора находятся три кнопки. Кнопка **OK** для подтверждения изменений, кнопка **Cancel** для отмены произведенных модификаций и кнопка **Set to Identity Matrix**, задающая для матрицы значения по умолчанию.

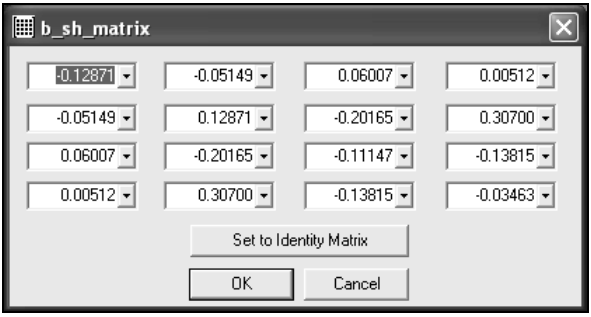


Рис. 7.37. Редактор матриц

Vector Editor

Векторный редактор (Vector Editor), изображенный на рис. 7.38, позволяет редактировать составляющую значения вектора с помощью текстовых списков для осей X, Y Z и W. Значение в текстовых списках можно переопределять как с помощью клавиатуры, так и выбирать из предлагаемого списка. Рабочее окно редактора имеет два флага. Флаг **Clamp from** (Диапазон установок) задает минимальное и максимальное значение дальности, а флаг **Keep** (Сохранить) сохраняет нормализованный вектор. Для вступления в силу установленных значений нажмите кнопку **OK** или **Cancel** — для отмены.

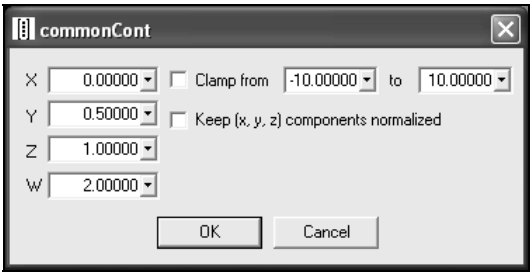


Рис. 7.38. Векторный редактор

Scalar Editor

Редактирование скалярных величин происходит с помощью текстового списка, находящегося в окне редактора, показанного на рис. 7.39. Флагом **Clamp from** задается диапазон значений. Для подтверждения действий служит кнопка **OK**, а кнопка **Cancel** — для отмены.

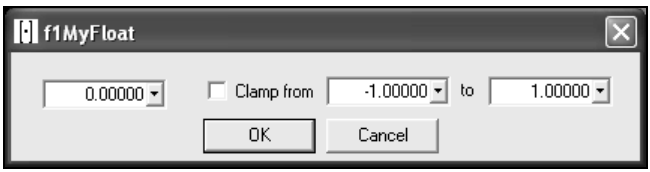


Рис. 7.39. Скалярный редактор

Color Editor

Редактор цвета (Color Editor) позволяет задавать цветовую составляющую для режимов RGBA и HSVA. Редакция значений происходит с помощью текстовых полей или цветовой модели, выполненной в виде шестиугольника с доступной цветовой гаммой. На рис. 7.40 изображены редакторы для режимов RGBA и HSVA.

Dynamic Variable Editor

Динамический редактор переменных (Dynamic Variable Editor) объединяет в себе несколько типов данных — это `Boolean` (рис. 7.41), `Float`, `Integer` и матрицы 2×2, 2×3 и 3×3.

Редактор типа `Float` (рис. 7.42) содержит три текстовых поля для значений по осям X, Y, Z и W, а также опциональный набор элементов управления для указания доступного диапазона значений.

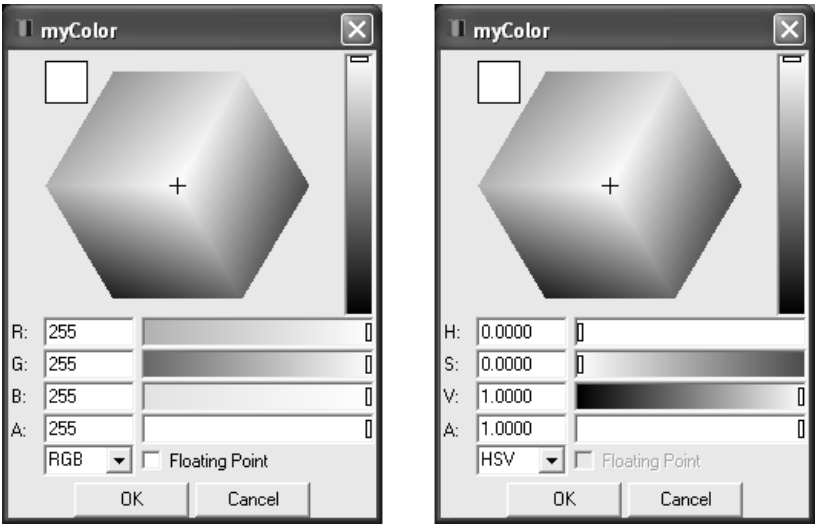


Рис. 7.40. Редактор цвета

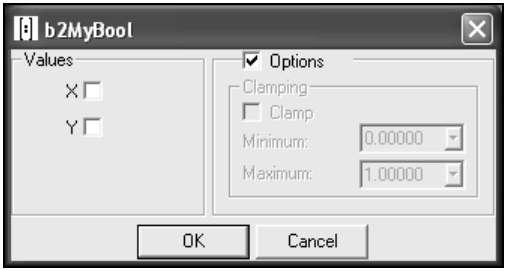


Рис. 7.41. Окно редактора для типа `Boolean`

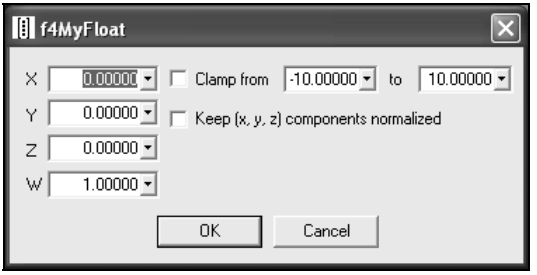


Рис. 7.42. Окно редактора для типа `Float`

В окне редактора типа `Integer` (рис. 7.43) доступно задание значения для целочисленной переменной с помощью текстовых полей по осям X, Y и Z. Также в окне редактора сформирован набор элементов управления для указания диапазона значений с помощью опции **Clamping** (Скрепить).

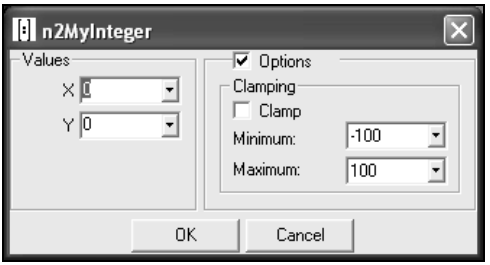


Рис. 7.43. Окно редактора для типа `Integer`

Матричный редактор (рис. 7.44) доступен для матриц размерностью 2×2, 2×3 и 3×3. В окне матричного редактора с помощью текстовых полей устанавливается значение матрицы с предопределенной размерностью. А с помощью группы элементов под названием **Clamping** формируется диапазон допустимых значений. Кнопки **OK** и **Cancel** в окне редактора служат для подтверждения или отмены устанавливаемых значений матрицы.

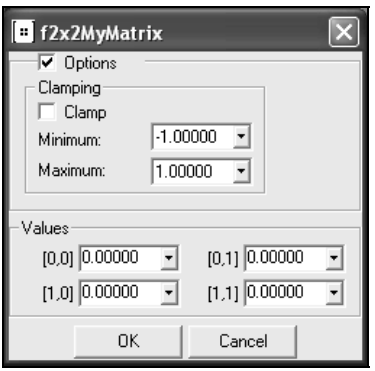


Рис. 7.44. Матричный редактор

Predefined Variable

Предопределенные переменные (Predefine Variable) — это набор предопределенных переменных (констант) для работы с данными шейдера. Рассмотрим имеющиеся предопределенные переменные для инструментария RenderMonkey.

Время (Time)

- `Time0_X` — эта переменная представляет значение с плавающей точкой, которое изменяется на основании временного цикла (Cycle time). Временной цикл по умолчанию задается значением в 120 секунд;
- `CosTime0_X` — косинус для значения `Time0_X`;
- `SinTime0_X` — синус для значения `Time0_X`;
- `TanTime0_X` — пространство касательных для значения `Time0_X`;
- `Time0_X_Packed` — упаковка упомянутых ранее значений для `Time0_X`;
- `Time0_1` — эта переменная представляет значение с плавающей точкой, которое изменяется в промежутке от 0 до 1 на основании временного цикла (Cycle time). Временной цикл по умолчанию задается значением в 120 секунд. По окончании временного цикла значение переменной возвращается к 0. Диапазон цикла задается в диалоговом окне **RenderMonkey Preferences**, доступном через команду меню **Edit | Preferences** (Редактировать | Свойства);
- `CosTime0_1` — косинус для значения `Time0_1`;
- `SinTime0_1` — синус для значения `Time0_1`;
- `TanTime0_1` — пространство касательных для значения `Time0_1`;
- `Time0_1_Packed` — упаковка упомянутых ранее значений для `Time0_1`;
- `Time0_2PI` — эта переменная представляет значение с плавающей точкой, которое изменяется в промежутке от 0 до 2PI на основании временного цикла (Cycle time). Временной цикл по умолчанию задается значением в 120 секунд. По окончании временного цикла значение переменной возвращается к 0. Диапазон цикла задается в диалоговом окне **RenderMonkey Preferences**, доступном через команду меню **Edit | Preferences** (Редактировать | Свойства);
- `CosTime0_2PI` — косинус для значения `Time0_2PI`;
- `SinTime0_2PI` — синус для значения `Time0_2PI`;
- `TanTime0_2PI` — пространство касательных для значения `Time0_2PI`;
- `Time0_2PI_Packed` — упаковка упомянутых ранее значений для `Time0_2PI`;
- `TimeCyclePeriod` — период цикла. По умолчанию цикл установлен в диапазоне от 0 до 120 секунд;
- `FPS` — частота кадров;
- `TimeElapsed` — время между кадрами.

Окно предварительного просмотра (Viewport)

- `ViewportWidth` — определяет ширину окна предварительного просмотра в пикселах;
- `ViewportHeight` — определяет высоту окна предварительного просмотра в пикселах;
- `ViewportWidthInverse` — возвращает значение 1.0 / `ViewportWidth`;
- `ViewportHeightInverse` — возвращает значение 1.0 / `ViewportHeight`.

Случайные значения (Random Values)

- `RandomFraction1PerPass` — первое случайное фрактальное значение для одного из проходов рисования в диапазоне [0 .. 1];
- `RandomFraction2PerPass` — второе случайное фрактальное значение для одного из проходов рисования в диапазоне [0 .. 1];
- `RandomFraction3PerPass` — третье случайное фрактальное значение для одного из проходов рисования в диапазоне [0 .. 1];
- `RandomFraction4PerPass` — четвертое случайное фрактальное значение для одного из проходов рисования в диапазоне [0 .. 1];
- `RandomFraction1PerEffect` — первое случайное фрактальное значение для одного из эффектов в диапазоне [0 .. 1];
- `RandomFraction2PerEffect` — второе случайное фрактальное значение для одного из эффектов в диапазоне [0 .. 1];
- `RandomFraction3PerEffect` — третье случайное фрактальное значение для одного из эффектов в диапазоне [0 .. 1];
- `RandomFraction4PerEffect` — четвертое случайное фрактальное значение для одного из эффектов в диапазоне [0 .. 1].

Проход рисования (Pass)

- `PassIndex` — индекс прохода рисования.

Параметры мыши (Mouse Parameters)

- `LeftMouseButton` — возвращает значение 1.0, если левая кнопка мыши нажата, или значение 0.0, если кнопка не нажата;
- `MiddleMouseButton` — возвращает значение 1.0, если средняя кнопка мыши нажата, или значение 0.0, если кнопка не нажата;
- `RightMouseButton` — возвращает значение 1.0, если правая кнопка мыши нажата, или значение 0.0, если кнопка не нажата;

- `MouseButtonsPacked` — упаковывает упомянутые ранее значения `LeftMouseButton`, `MiddleMouseButton` и `RightMouseButton`;
- `MouseCoordinateX` — возвращает позицию мыши по оси X;
- `MouseCoordinateY` — возвращает позицию мыши по оси Y;
- `MouseCoordinateXNDC` — возвращает значение, равное `MouseCoordinateX`/`ViewportWidth`;
- `MouseCoordinateYNDC` — возвращает значение, равное `MouseCoordinateY`/`ViewportHeight`;
- `MouseCoordsPacked` — упаковывает упомянутые ранее значения `MouseCoordinateX`, `MouseCoordinateY`, `MouseCoordinateXNDC` и `MouseCoordinateYNDC`;
- `MouseCoordinateXY` — возвращает значение `MouseCoordinateX` и `MouseCoordinateY` в виде двухкомпонентного вектора;
- `MouseCoordinateXYNDC` — возвращает значение `MouseCoordinateXNDC` и `MouseCoordinateYNDC` в виде двухкомпонентного вектора.

Параметры модели (Model Parameters)

- `ModelMoundingBoxTopLeftCorner` — координата левого верхнего угла модели;
- `ModelMoundingBoxBottomRightCorner` — координата правого нижнего угла модели;
- `ModelMoundingBoxCenter` — центр пространства ограничения модели;
- `ModelCentroid` — центр модели;
- `ModelBoundingSphereCenter` — центр сферы;
- `ModelBoundingSphereRadius` — радиус сферы.

Параметры вида (View Parameters)

- `ViewDirection` — вектор направления;
- `ViewPosition` — позиция вида;
- `ViewSideVector` — боковой вектор;
- `ViewUpVector` — вектор представления;
- `FOV` — поле зрения;
- `NearClipPlane` — передний план отсечения;
- `FarClipPlane` — задний план отсечения.

Видовые матрицы (View Matrices)

- `View` — матрица вида 4×4;
- `ViewTranspose` — видовая матрица трансляции 4×4;
- `ViewInverse` — видовая инверсная матрица 4×4;
- `ViewInverseTranspose` — видовая инверсная матрица трансляции 4×4;
- `Projection` — матрица проекции 4×4;
- `ProjectionTranspose` — проекционная матрица трансляции 4×4;
- `ProjectionInverse` — проекционная инверсная матрица 4×4;
- `ProjectionInverseTranspose` — проекционная инверсная матрица трансляции 4×4;
- `ViewProjection` — видовая проекционная матрица 4×4;
- `ViewProjectionTranspose` — видовая проекционная матрица трансляции 4×4;
- `ViewProjectionInverse` — видовая проекционная инверсная матрица 4×4;
- `ViewProjectionInverseTranspose` — видовая проекционная инверсная матрица трансляции 4×4;
- `World` — мировая матрица 4×4;
- `WorldTranspose` — мировая матрица трансляции 4×4;
- `WorldInverse` — мировая инверсная матрица 4×4;
- `WorldInverseTranspose` — мировая инверсная матрица трансляции 4×4;
- `WorldView` — мировая и видовая матрица 4×4;
- `WorldViewTranspose` — мировая видовая матрица трансляции 4×4;
- `WorldViewInverse` — мировая видовая инверсная матрица 4×4;
- `WorldViewInverseTranspose` — мировая видовая инверсная матрица трансляции 4×4;
- `WorldViewProjection` — мировая видовая и проекционная матрица 4×4;
- `WorldViewProjectionTranspose` — мировая видовая проекционная матрица трансляции 4×4;
- `WorldViewProjectionInverse` — мировая видовая проекционная инверсная матрица 4×4;
- `WorldViewProjectionInverseTranspose` — мировая видовая проекционная инверсная матрица трансляции 4×4.

Окно *Preview*

Окно предварительного просмотра (**Preview**) инструментария RenderMonkey интерактивно и обеспечивает просмотр создаваемой трехмерной сцены в реальном времени. Изначально камера в окне **Preview** установлена по центру экрана и скрыта для пользователя, но при щелчке левой кнопкой мыши в окне **Preview** в рабочем проекте вместо сцены будет представлена система координат конкретной модели, изображенная на рис. 7.45. В этот момент, если нажать и не отпускать левую кнопку мыши, можно разворачивать объект по осям X и Y. А с помощью колесика или средней кнопки мыши модель можно увеличивать или уменьшать.

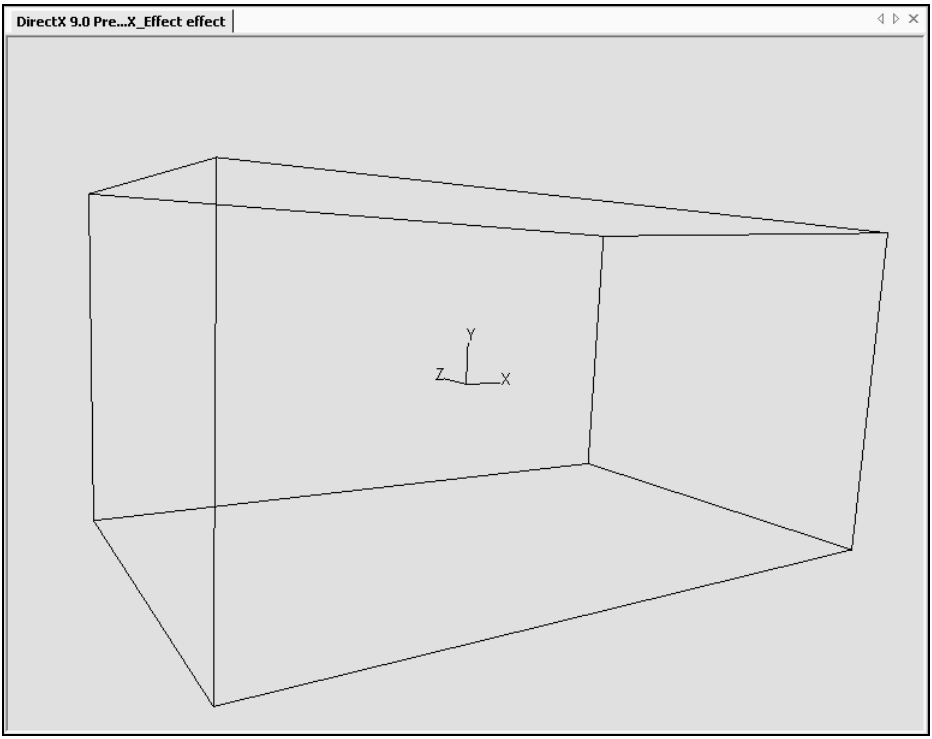


Рис. 7.45. Система координат модели

При нажатии правой кнопки мыши в окне **Preview** при работе с DirectX и OpenGL появляется контекстное меню с набором следующих команд:

- **HAL** — задействуются аппаратные возможности системы. Доступно только в DirectX;
- **REF** — задействуются программные возможности системы. Доступно только в DirectX;

- **Clear Color** — вызывает окно редактора цвета, задающего цветовую составляющую для очистки цветом заднего буфера;
- **Camera Setting** — дает возможность коррекции настроек камер;
- **Show Triad** — включает систему представления координат непосредственно с объектом сцены;
- **Show Bounding Box** — включает систему представления ограничительного куба для модели непосредственно с объектом сцены;
- **Fit Model to Screen** — изменяет активную камеру для более комфортного просмотра сцены;
- **Original View** — оригинальный вид;
- **Front View** — передняя, или фронтальная сторона, просмотра модели;
- **Back View** — задняя сторона просмотра модели;
- **Left View** — левая сторона просмотра модели;
- **Right View** — правая сторона просмотра модели;
- **Top View** — верхняя сторона просмотра модели;
- **Bottom View** — нижняя сторона просмотра модели.

Примечание

Работая с проектом и изменяя или дополняя его, не забывайте компилировать текущий эффект, используя кнопки на панели инструментов **Compile Active Effect** (Компилировать активный эффект) **Compile All Shader** (Компилировать все шейдеры) для просмотра в окне **Preview** текущих изменений.

В RenderMonkey при работе над проектом в окне предварительного просмотра доступен набор "горячих" клавиш, с помощью которых можно просматривать индивидуальные проходы прорисовки сцены. Нажав клавишу <H> на клавиатуре во время работы над проектом, в окне **Output** отобразятся все доступные "горячие" клавиши с подробным описанием их возможностей.

Редактор для художников

Одной из главных задач в команде разработчиков при работе над проектом является плотное взаимодействие программистов и художников. В RenderMonkey такая возможность предусмотрена на программном уровне и дает возможность экспериментировать художникам с параметрами создаваемой программы.

Как уже не раз упоминалось, программист при работе над проектом может определять ряд параметров, доступных для художника с помощью опции **Artist Variable** (Переменная художника) в контекстном меню, появляющем-

ся при нажатии правой кнопки мыши в структуре проекта окна **Workspace** (Рабочее пространство). Тем самым художнику дается доступ к определенным типам данных на вкладке **Art** (Художник) окна **Workspace**. Например, на рис. 7.46 изображена вкладка **Art** окна **Workspace**, где художнику доступны для редакции параметры проекта Wood из стандартной поставки RenderMonkey, который находится в каталоге ATI Research Inc\RenderMonkey 1.5 \Examples\Dx9

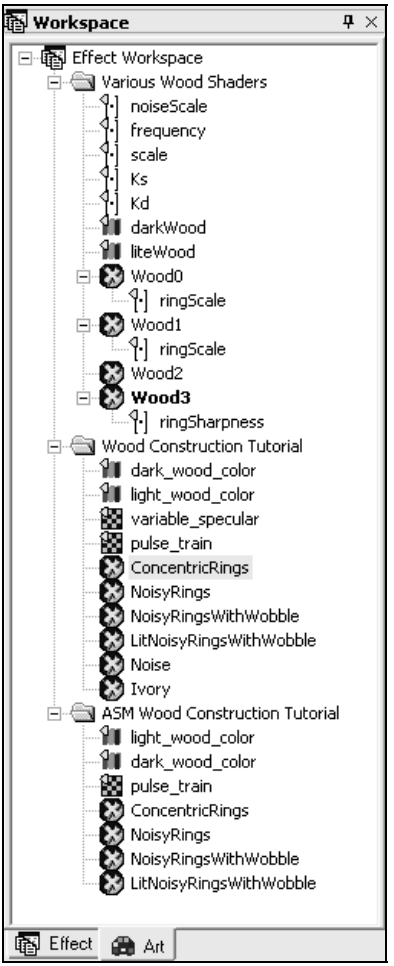


Рис. 7.46. Вкладка **Art** окна **Workspace** проекта Wood

Для художника в проекте Wood доступен ряд переменных, изменяя значение которых можно поэкспериментировать с графической составляющей проекта и при этом не вмешиваться в работу программиста. Все изменения

переменных осуществляются с помощью специализированного редактора, предназначенного для художников под названием (Artist Editor). Открытие Artist Editor в RenderMonkey осуществляется командой меню **View | Artist Editor** (Вид | Художественный редактор), на рис. 7.47 представлен Artist Editor для проекта Wood. Замечу, что от проекта к проекту Artist Editor содержит различное количество элементов управления, используя которые художник вправе редактировать доступные параметры проекта.

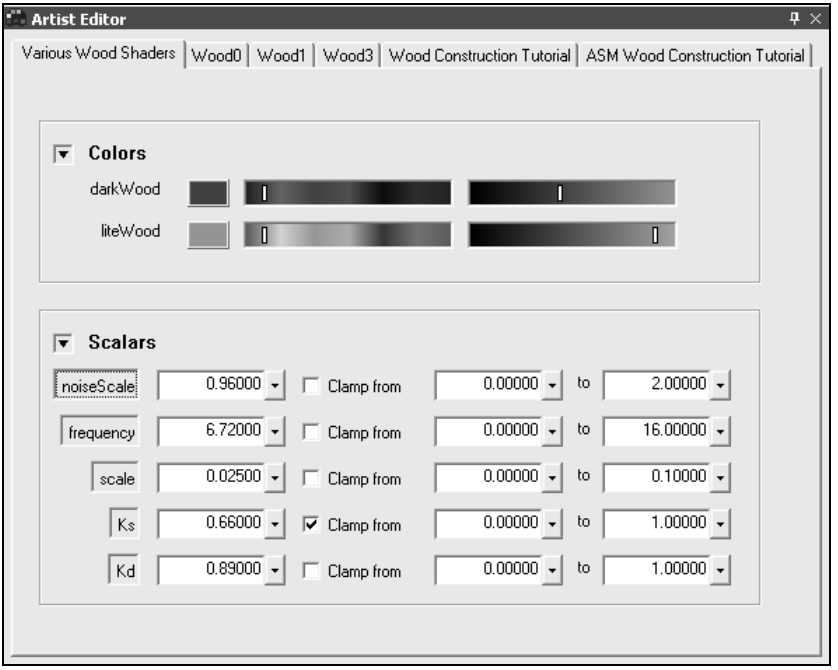
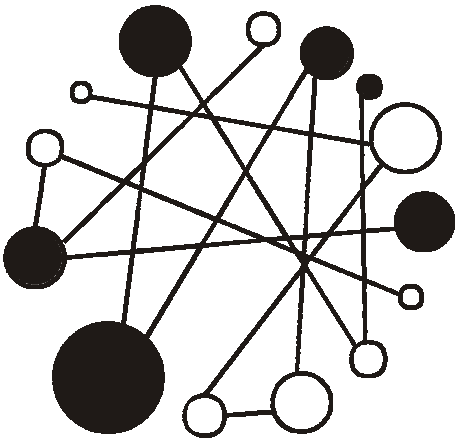


Рис. 7.47. Окно **Artist Editor** проекта Wood

Инструментарий RenderMonkey — это очень качественное программное средство, обладающее огромным потенциалом и способное удовлетворить нужды любой команды разработчиков. Несомненная гибкость и возможность в подключении дополнительных модулей выделяет RenderMonkey из ряда аналогичных средств, имеющихся на рынке.

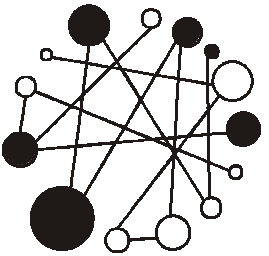
В следующих двух главах мы познакомимся с не менее интересным средством FX Composer 1.5, созданным компанией NVIDIA для отладки и программирования шейдеров.



ЧАСТЬ IV

Инструментарий NVIDIA FX Composer

ГЛАВА 8



Знакомство с FX Composer

Семейство видеоадаптеров от NVIDIA представлено многообразием моделей, ориентированных на потребителей различного уровня. В связи с этим кому как не компании NVIDIA создавать инструментальные средства для работы с графикой. Одним из существующих инструментариев и является интегрированное средство разработки приложений FX Composer 1.5 для работы с шейдерами.

FX Composer дает возможность создавать высокоэффективные программы шейдеров с работой в реальном времени и просмотром произошедших изменений в сцене. Кроме того, очень удобно писать исходный код в текстовом редакторе, обладающем интеллектуальной поддержкой всех версий шейдеров и языков HLSL. FX Composer поддерживает импорт/экспорт данных сцены, также можно просматривать эксплуатационные показатели программ. Поддержка осуществляется для всей линейки видеоадаптеров NVIDIA от GeForce FX и выше. Для работы с FX Composer 1.5 вам понадобится как минимум DirectX 9 SDK Update (Summer 2004).

Изучаем FX Composer

Установка FX Composer более чем элементарна: несколько чередующихся диалоговых окон предоставят вам возможность определиться с каталогом по установке, ознакомят с нюансами лицензионного соглашения и поздравят вас с удачной установкой выбранного программного обеспечения. По окончании установки FX Composer в меню **Start | All Programs** (Пуск | Все программы) будет сформирована группа с названием **NVIDIA Corporation** со следующим списком элементов:

- **FX Composer** — запускает работу FX Composer;
- **License** — условия лицензирования;
- **Release Note** — информационный документ о релизе FX Composer;

- **Uninstall NVIDIA FX Composer** — удаление FX Composer;
- **User Guide** — руководство пользователя в формате PDF.

Для запуска FX Composer 1.5 используйте команды меню **Start | All Programs | NVIDIA Corporation | NVIDIA FX Composer | FX Composer** (Пуск | Все программы | NVIDIA Corporation | NVIDIA FX Composer | FX Composer) и вам откроется рабочее окно FX Composer, представленное на рис. 8.1.

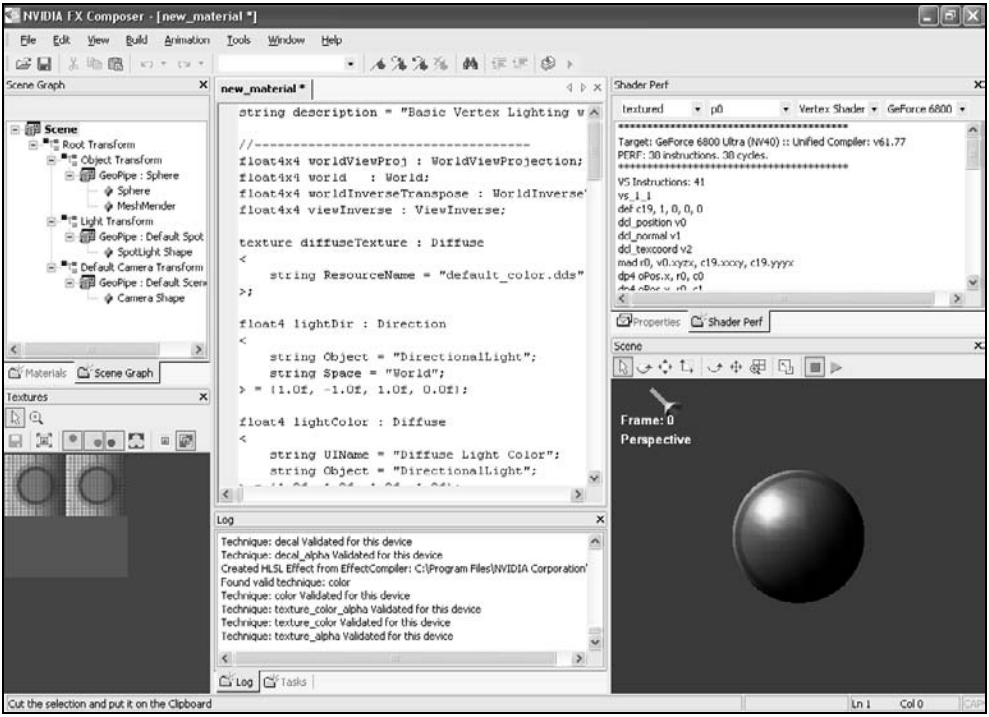


Рис. 8.1. Рабочее окно FX Composer

Средство разработки приложений FX Composer обладает приятным и главное интуитивно понятным интерфейсом, имеющим сходство с Visual C++ .NET. Рабочее окно FX Composer гармонично разделено на множество различных по размеру окон, или, как принято говорить при использовании FX Composer, панелей. Также имеется одно большое и самое главное по назначению окно текстового редактора, которое по мере открытия или закрытия всех дополнительных панелей соответственно уменьшается или увеличивается в своем размере. При первичном старте FX Composer открываются все панели, а окно текстового редактора находится в середине, как показано на рис. 8.1.

FX Composer обладает особой гибкостью в настройке его интерфейса, возможны любые комбинации при отображении соответствующих панелей.

Любую выбранную панель можно открыть, закрыть, удалить, переместить, развернуть и т. д., и при этом доступна возможность использования даже двух мониторов для настройки удобной вам конфигурации панелей. Кроме окна текстового редактора имеются восемь настраиваемых панелей: **Log** (Информационная панель), **Error** (Панель ошибок), **Properties** (Панель свойств), **Materials** (Панель со свойствами материала), **Textures** (Текстурная панель), **Shader Perf** (Панель шейдеров), **Scene** (Панель сцены) и **Scene Graph** (Структурная панель сцены). Открытие или закрытие необходимой панели доступно из меню **View | Panels** (Вид | Панели). Рассмотрим назначение панелей FX Composer.

Панель *Log*

Эта панель отображается, как правило, в нижней части FX Composer, располагаясь по всей ширине рабочей поверхности, но можно выбрать и любое другое место для нее. Панель **Log** является информационной, на ней регистрируются или отображаются все происходящие события при работе с FX Composer, такие как открытие и закрытие файлов, создание материала, назначение текстур, работа с шейдерами и т. д. На рис. 8.2 изображена панель **Log**, на которой присутствуют записи различных событий, произошедших за время работы с FX Composer.

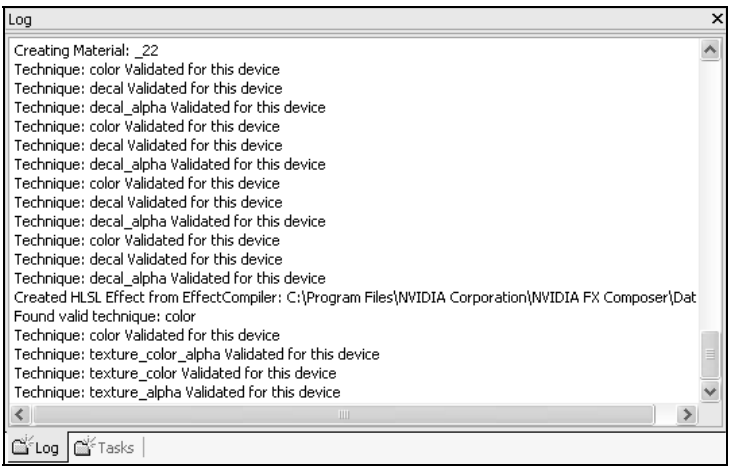


Рис. 8.2. Панель **Log**

Панель *Error*

Панель ошибок (**Error**) содержит перечисление текстовой информации о происходящих ошибках в работе над определенным проектом. Будь то ошибки компиляции или неправильно сформированные свойства материа-

ла, на панели **Error** обязательно будут выведены соответствующие записи, как это показано на рис. 8.3.

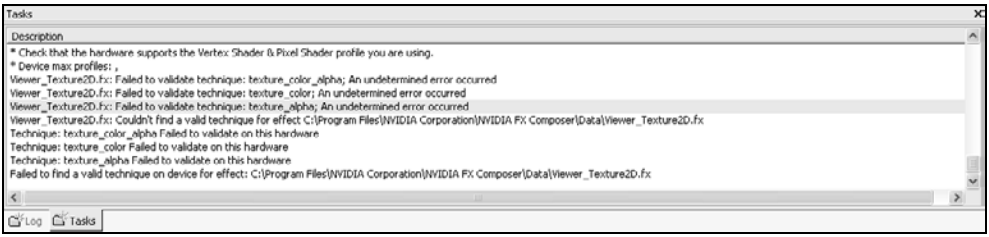


Рис. 8.3. Панель **Error**

Первоначально панель **Error** отображается в нижней части рабочего окна FX Composer и накладывается аккуратно поверх панели **Log**. В момент одновременного открытия этих двух панелей, они накладываются друг на друга, в связи с чем формируется дополнительная линейка с закладками для доступа к обеим панелям, посмотрите на рис. 8.3, где это очень хорошо видно. Обратите внимание, что если на закладке панели **Log** написано **Log**, то на закладке панели **Error** присутствует надпись **Tasks** (Задачи). Переключаясь с помощью двух закладок, можно получить доступ к информации на панелях **Error** и **Log**.

Панель *Properties*

Properties — это панель свойств объекта с набором изменяемых параметров (рис. 8.4). Панель **Properties** отображается с правой стороны FX Composer, занимая примерно третью часть всей рабочей поверхности, как это показано на рис. 8.1. Панель **Properties** используется для воссоздания свойств материала, цвета, текстур объекта, используемых значений матриц и векторов в виде перечисляемого набора параметров с изменяющимися значениями. Все имеющиеся параметры отображены в виде таблицы из двух столбцов: с левой стороны представлены параметры объекта, а с правой — значение для каждого параметра (см. рис. 8.4).

Панель *Materials*

С помощью панели **Materials** отображается или задается материал для поверхности объекта (рис. 8.5). На ней вы можете устанавливать свойства материала, загружать в проект свои образцы или использовать ресурсы FX Composer. Но главное это то, что можно отслеживать полученный результат в реальном времени. Панель **Materials** при запуске FX Composer находится с левой стороны рабочего окна FX Composer.

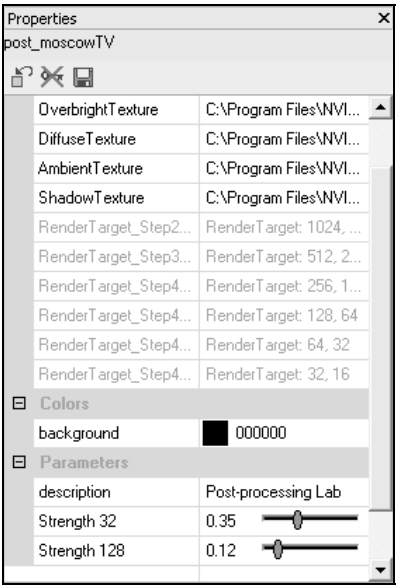


Рис. 8.4. Панель **Properties**

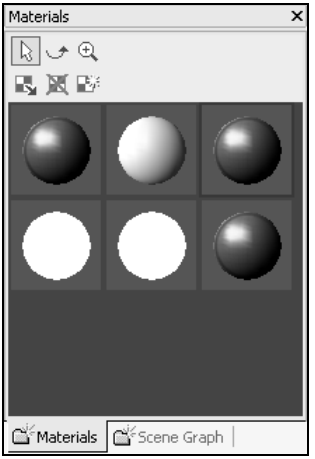


Рис. 8.5. Панель **Materials**

Панель *Textures*

На панели **Textures** можно назначать текстуру для материала (рис. 8.6). Существует возможность загрузки текстур из файлов ресурса или можно использовать уже имеющиеся в FX Composer текстуры. Панель **Textures** при работе находится с левой стороны FX Composer. Если также открыта панель

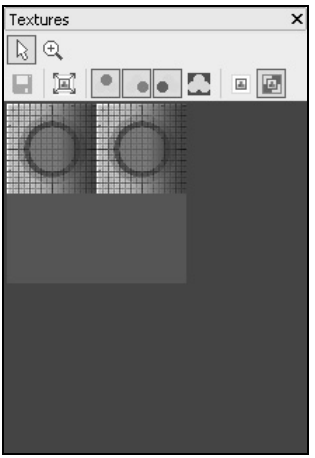


Рис. 8.6. Панель **Textures**

Materials, обе панели располагаются одна под другой, т. е. каждая из панелей просто растягивается на всю высоту рабочего окна FX Composer.

Панель *Shader Perf*

Панель **Shader Perf**, изображенная на рис. 8.7, служит для оценки эффективности создаваемой программы HLSL. Панель **Shader Perf** — это встроенный профилировщик, используемый в FX Composer для анализа эксплуатационных показателей программы шейдеров. Текстовая область **Shader Perf** содержит откомпилированный ассемблерный код программы HLSL, а с помощью встроенной линейки меню панели **Shader Perf** можно моделировать ситуации с различными графическими процессорами, что дает возможность продемонстрировать возможности создаваемой программы.

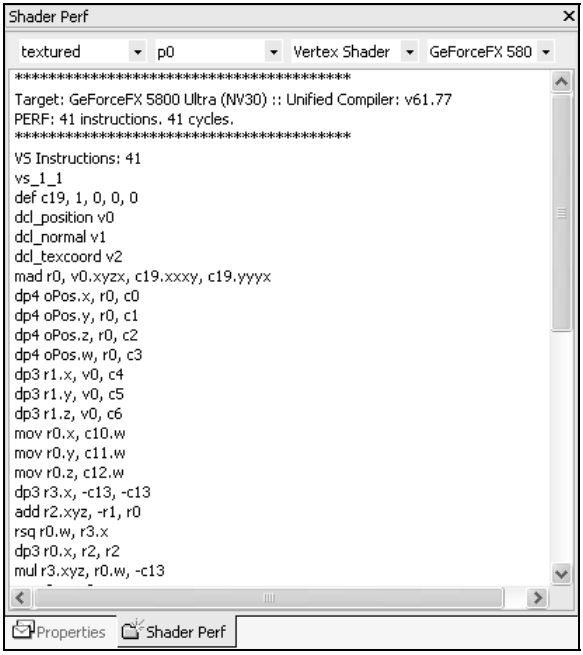


Рис. 8.7. Панель **Shader Perf**

Панель *Scene*

Эта панель необходима для представления всей сцены и позволяет в реальном времени наблюдать за всевозможными изменениями, связанными с назначением текстур, материала и использованием вершинных и пиксельных шейдеров. Имеется хорошо отлаженная система для импорта сцены в FX Composer. Панель **Scene** представлена на рис. 8.8 и располагается в ле-

вой нижней части рабочего пространства FX Composer. Если она закрыта, то панели **Shader Perf** и **Properties** растягиваются по всей высоте FX Composer.

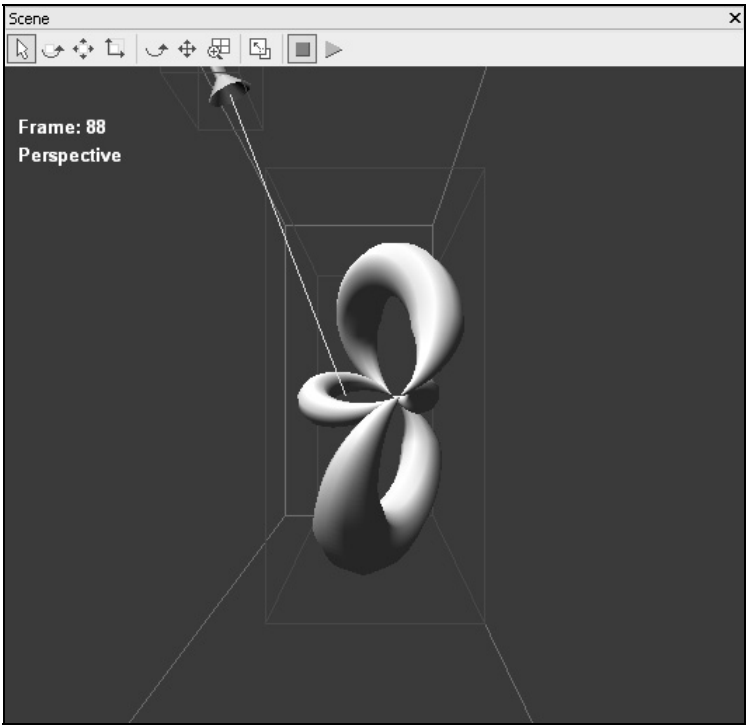


Рис. 8.8. Панель **Scene**

Панель *Scene Graph*

Панель **Scene Graph** необходима для отображения древовидной иерархии сцены, представленной в виде набора элементов, как это показано на рис. 8.9. Элемент в иерархии может представлять камеру, свет, трансформацию объекта в структурном виде, что достаточно сильно облегчает работу со сценой в целом.

Текстовый редактор

Окно текстового редактора, изображенное на рис. 8.10, при запуске FX Composer находится в центре экрана. Это самое большое окно из всех имеющихся, поскольку в редакторе происходит набор исходного кода программ шейдеров. Текстовый редактор FX Composer обладает интеллектуаль-

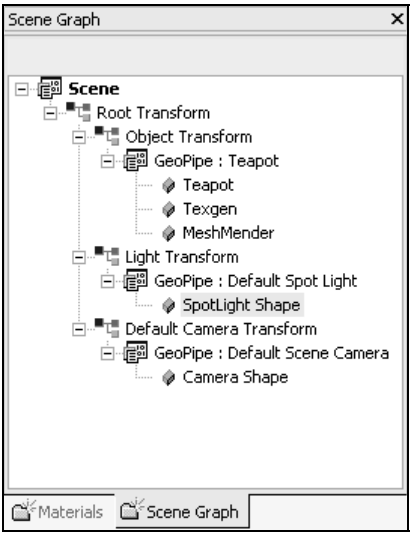


Рис. 8.9. Панель **Scene Graph**

ными возможностями. Подсветка синтаксиса языка шейдеров решена на высшем уровне, к примеру при наборе программного кода после оператора "точка" или "равно" может появляться информационный вспомогательный список с перечислением ключевых слов для быстрой работы с элементами программы.

PointShadowAim

```
////////////////////////////////////////////////////////////////////////////////
/// Vertex Shaders /////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

// from scene camera POV
AimShadowVertexOutput mainCamVS(AimShadowAppData IN,
                    uniform float4x4 ShadowXf)
{
    AimShadowVertexOutput OUT = (AimShadowVertexOutput)0;
    OUT.WNormal = mul(IN.Normal,WorldITXf).xyz; // world coords
    float4 Po = float4(IN.Position.xyz,(float)1.0); // "P" in object coordinates
    float4 Pw = mul(Po,WorldXf);             // "P" in world coordinates
    float4 Pl = mul(Pw,ShadowXf);            // "P" in light coords...
    OUT.LP = Pl;                             // ...for pixel-shader shadow calcs
    OUT.WView = normalize(ViewIXf[3].xyz - Pw.xyz); // world coords
    OUT.HPosition = mul(Po,WorldViewProjXf);    // screen clipspace coords
    OUT.UV = IN.UV.xy;                          // pass-thru
    OUT.LightVec = PointLightPos - Pw.xyz;      // world coords
    return OUT;
}
```

Рис. 8.10. Текстовый редактор FX Composer

Панель инструментов

Панель инструментов FX Composer содержит традиционный набор элементов меню и кнопок быстрого доступа, показанных на рис. 8.11. Организация меню очень напоминает меню среды Visual Studio .NET. Линейка меню FX Composer включает команды **File** (Файл), **Edit** (Редактирование), **View** (Вид), **Build** (Сборка и компиляция), **Animation** (Анимация), **Tools** (Инструменты), **Window** (Окно), **Help** (Помощь).

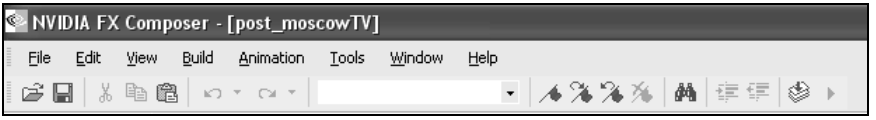


Рис. 8.11. Панель инструментов FX Composer

Меню *File*

Меню **File**, изображенное на рис. 8.12, содержит набор команд для открытия, закрытия, сохранения файлов проекта, импорта и экспорта сцен.

В меню **File** имеются следующие команды:

- **New** — загрузка нового материала;
- **Open** — содержит дочернее или вложенное меню команд со следующими командами:
  - **Project** — открывает проект FX Composer с расширениями fxcomposer или fxproj;
  - **Package** — открывает пакет FX Composer с расширениями fxcomposer или fxproj;
  - **Material** — открывает материал из файлов с расширениями fx или fxh;
  - **Scrip** — открывает файл NET Scripts с расширениями cs или vb;
  - **File** — открывает все файлы проектов FX Composer;
- **Import Scene** — производит импорт сцены из файлов с расширениями nvb, ply, obj и x;
- **Export Scene** — экспорт сцены в формате MDF (Material Dump File);
- **Save Project** — сохраняет проект;
- **Save Package** — сохраняет пакет;
- **Save** (<Ctrl>+<S>) — сохраняет текущий проект;
- **Save As** — сохраняет текущий проект в заданном каталоге;
- **Print** (<Ctrl>+<P>) — печать содержимого текстового редактора;

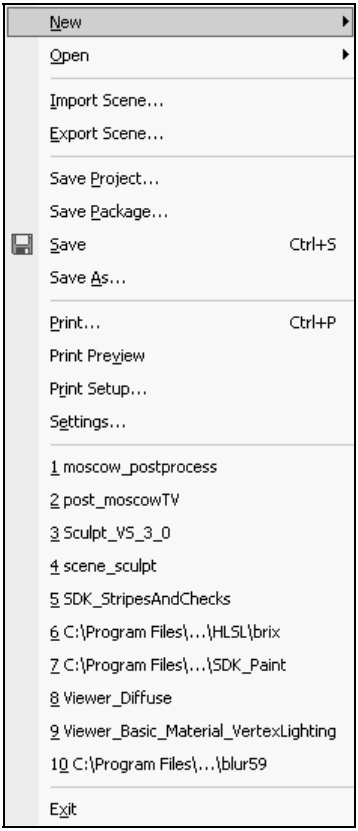


Рис. 8.12. Меню File

- **Print Preview** — предварительный просмотр перед печатью;
- **Print Setup** — установки печати;
- **Settings** — установки FX Composer. После этой команды в меню **File** идет перечисление последних десяти открытых файлов, как показано на рис. 8.12;
- **Exit** — выход из FX Composer.

Меню *Edit*

На рис. 8.13 изображено меню **Edit**, с помощью команд которого можно совершать операции редактирования в текстовом редакторе.

В меню **Edit** имеются следующие команды:

- **Undo** (<Alt>+<Backspace>) — на шаг назад или отменить операцию;
- **Redo** (<Ctrl>+<Y>) — возвратить;

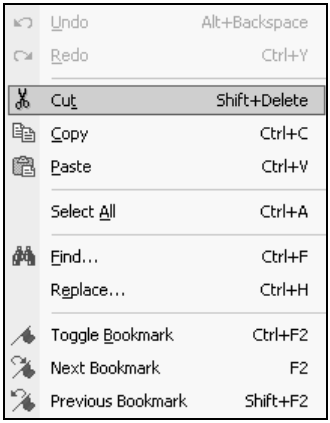


Рис. 8.13. Меню **Edit**

- **Cut** (<Shift>+<Delete>) — удаление;
- **Copy** (<Ctrl>+<C>) — копирование;
- **Paste** (<Ctrl>+<V>) — вставка;
- **Select All** (<Ctrl>+<A>) — выделить весь код в текстовом редакторе;
- **Find** (<Ctrl>+<F>) — найти слово в файле;
- **Replace** (<Ctrl>+<H>) — переместить;
- **Toggle Bookmark** (<Ctrl>+<F2>) — помечает текст в текстовом редакторе, делая закладку;
- **Next Bookmark** (<F2>) — переход к следующей закладке;
- **Previous Bookmark** (<Shift>+<F2>) — возврат к предыдущей закладке.

Меню *View*

В меню **View** содержатся всего две команды: **Panels** (Панели) и **Status Bar** (Полоса состояния). Команда **Panels** имеет вложенное меню с перечислением всех доступных панелей FX Composer: **Log Panel**, **Error Panel**, **Properties Panel**, **Materials Panel**, **Textures Panel**, **ShaderPerf Panel**, **Scene Panel** и **Scene Graph Panel**, как показано на рис. 8.14. Поставив или убрав флажок напротив выбранной панели в меню **View** | **Panels** (Вид | Панели), вы автоматически откроете или закроете соответствующую панель. Команда **Status Bar** откроет или закроет средства разработки приложений FX Composer.

Меню *Build*

В меню **Build** находится две команды **Compile** (Компиляция) и **Run** (Запустить). С помощью команды **Compile** или "горячих клавиш" <Ctrl>+<F7> производится компиляция текущего проекта с возможностью просмотра

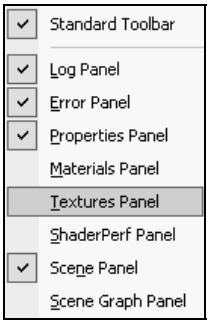


Рис. 8.14. Меню **View**, команда **Panels**

полученного результата в реальном времени, для этого выберите команду **Run**.

Меню *Animation*

Меню **Animation** содержит четыре команды, на рис. 8.15 это хорошо видно. Команды меню **Animation** связаны с панелью **Scene**, с их помощью можно влиять на анимацию в окне панели **Scene**.

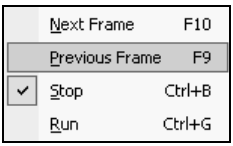


Рис. 8.15. Меню **Animation**

Меню **Animation** содержит четыре команды:

- **Next Frame** (<F10>) — осуществляет переход по последовательности имеющихся фреймов всей анимации;
- **Previous Frame** (<F9>) — предыдущий фрейм анимационной последовательности;
- **Stop** (<Ctrl>+<B>) — остановка анимации;
- **Run** (<Ctrl>+<G>) — начало анимации.

Меню *Tools*

В меню **Tools** содержится два пункта, каждый из которых имеет несколько вложенных команд. Давайте познакомимся с меню **Tools**:

- **Plugins** — подключаемые модули. Содержит две команды:
  - **Tools** — инструменты;
  - **List** — перечисление подключенных модулей в FX Composer;

- **Options** — настраивает опции с помощью команд:
  - **Draw Always** — всегда рисовать;
  - **Update Animated Materials** — обновлять после изменения материала.

Меню *Window*

Обычно в этом меню содержатся команды, с помощью которых можно настраивать необходимое расположение активных окон или панелей в средах программирования. Инструментарий FX Composer не исключение, он также имеет следующие стандартные команды:

- **New Window** — новое окно;
- **Cascade** — окна каскадом;
- **Tile** — открыть в отдельном окне;
- **Arrange Icons** — разместить иконки;
- **Windows** — открывает диалоговое окно со списком активных окон.

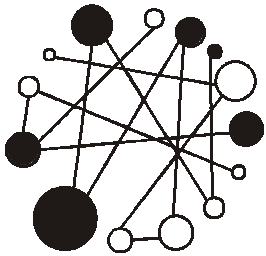
Меню *Help*

В меню **Help** расположены команды, связанные с контекстной справкой FX Composer:

- **About FX Composer** — рассказывает о FX Composer;
- **Semantics & Annotations** — семантика и аннотация, используемая при работе с FX Composer;
- **Show Tips** — справочные заметки на каждый день.

Интерфейс инструментария FX Composer достаточно сбалансирован и хорошо продуман, а главное интуитивно понятен. В следующей главе мы рассмотрим возможности FX Composer при работе с шейдерами.

ГЛАВА 9



Работа с инструментарием FX Composer

В FX Composer, кроме создания трехмерных сцен, имеется возможность импорта сцены, что и является основной направленностью данного инструментария. Для того чтобы загрузить сцену, выберите в линейке меню FX Composer команду **File | Import Scene** (Файл | Импорт сцены) и в появившемся диалоговом окне выберите необходимый файл. Инструментарий FX Composer поддерживает загрузку файлов с расширениями nvb, ply, obj, x. Загруженная трехмерная сцена в FX Composer будет отражена на панели **Scene** (Панель сцены), где вы сможете производить эксперименты над текущим проектом. Также инструментарий FX Composer содержит более 165 различных эффектов, доступных для использования в проектах. Экспорт созданной сцены можно произвести выбрав в линейке меню FX Composer команду **File | Export Scene** (Файл | Экспорт сцены).

FX Composer в своем составе имеет более 50 проектов. По умолчанию файлы проекта находятся в каталоге установки FX Composer в папке NVIDIA FX Composer/MEDIA/projects и сохраняются с расширениями fxcomposer или fxproj. Файл проекта FXCOMPOSER — это обыкновенный заархивированный файл, содержащий описание трехмерной сцены и состоящий из следующих файлов:

- SceneGraph.xml — этот файл охватывает описание сцены и материалов в формате XML;
- Binary.bin — бинарный компонент, включающий мэш и анимационные данные;
- *.fx — рабочие файлы и текстуры.

Основной проектный файл — это файл формата XML, содержащий описание всего проекта, а именно: версию файла для компоновки FX Composer, путь поиска имеющихся файлов, материалы, графические данные, специальные параметры сцены, объекты.

В дополнение к проектным файлам FX Composer включает несколько файлов конфигурации (Configuration files), находящихся в каталоге установки FX Composer в папке NVIDIA FX Composer\data, с помощи которых можно производить изменения в конфигурации FX Composer. Имеются следующие конфигурационные файлы:

- fxcomposer_config.xml — содержит список стандартных материалов, которые применяются в сцене, также имеется возможность добавлять в список свои материалы;
- fxedit.xml — в этом файле находятся синтаксические правила для текстового редактора FX Composer;
- fxmapping.xml — представляет набор семантики и аннотаций FX Composer;
- defaultscene.fx — содержит простейший список команд трехмерной сцены для очистки фона и прорисовки объектов в сцене;
- plugins.inf — содержит перечисление дополнений к сцене.

> ***Примечание***
>
> Компания NVIDIA работает над инструментальным пакетом разработчиков SDK FX Composer, направленным на возможность подключения дополнительных модулей к FX Composer, создаваемых сторонними производителями.

Откройте все перечисленные ранее файлы конфигурации FX Composer, например, с помощью браузера Internet Explorer или Opera и ознакомьтесь с содержанием этих файлов.

При работе над трехмерной сценой в FX Composer вы будете постоянно обращаться к различным панелям инструментария, необходимым для написания кода шейдеров, назначения материала или текстуры, загрузки нового объекта, изменения свойств камеры и т. д. Поэтому имеет смысл остановиться на работе каждой из панелей отдельно, рассмотрев множественные нюансы. Пожалуй, самые главные панели в FX Composer — это панель сцены и панель текстового редактора. С обзора возможностей этих панелей мы и начнем анализировать достоинства и потенциал FX Composer.

Панель сцены

Панель сцены (**Scene**) отражает текущую сцену проекта. Как уже говорилось в самом начале главы, можно как загружать сцену, так и создавать или использовать готовые сцены из FX Composer. Панель сцены инструментария FX Composer обладает богатейшим набором визуальных инструментов, что делает работу чрезвычайно комфортной. При работе с объектами сцены можно изменять или назначать новые текстуры и материал, применять разнообразные источники освещения, экспериментировать с набором всевоз-

можных камер, прослеживать анимационную последовательность фреймов сцены, перемещать и трансформировать любые объекты сцены. Все перечисленные действия реализовываются с помощью визуальных средств FX Composer без необходимости изменения исходного кода проекта в текстовом редакторе. Например, для того чтобы назначить новый материал объекту, выделите нужный объект на панели сцены, щелкнув на нем левой кнопкой мыши, а на панели материальных свойств подберите необходимый материал, выделив его с помощью левого щелчка мыши, и нажмите кнопку **Apply Material** (Применить материал) на той же панели материальных свойств. И незамедлительно в реальном времени на панели сцены выбранный объект примет измененные свойства материала.

На панели сцены имеется инструментальная линейка с комплектом кнопок быстрого доступа, изображенных на рис. 9.1, с помощью которых осуществляется работа с объектами сцены.

Рис. 9.1. Кнопки быстрого доступа панели сцены

Слева направо на рис. 9.1 располагаются следующие кнопки быстрого доступа:

- **Select Object** — режим выбора объекта;
- **Rotate Object** — вращение объекта;
- **Move Object** — перемещение объекта;
- **Scale Object** — приближение или удаление объекта;
- **Rotate Scene** — вращение сцены, включая все объекты и источники света;
- **Pan Scene** — панорамирование сцены, включая все объекты и источники света;
- **Dolly Scene** — приближение или удаление сцены, включая все объекты и источники света;
- **Zoom Extents** — реальный размер в пространстве;
- **Stop Animation** (<Ctrl>+<B>) — остановить анимацию;
- **Run Animation** (<Ctrl>+<G>) — включить анимацию.

Комплект кнопок быстрого доступа также продублирован командами из контекстного меню, появляющимися при нажатии правой кнопки мыши в области панели сцены и при выборе в линейки меню FX Composer команды **Animation** (Анимация). Команда меню **Animation**, кроме двух команд **Stop Animation** (Остановить анимацию) и **Run Animation** (Запустить анимацию),

содержит еще две дополнительные команды **Next Frame** (Следующий фрейм) и **Previous Frame** (Предыдущий фрейм), позволяющие перемещаться по имеющейся последовательности фреймов сцены, давая возможность отслеживать пошаговые изменения в сцене.

Для работы с объектами сцены, загруженными в FX Composer, очень удобно использовать контекстное меню, появляющееся при нажатии правой кнопкой мыши в области панели **Scene** (рис. 9.2).

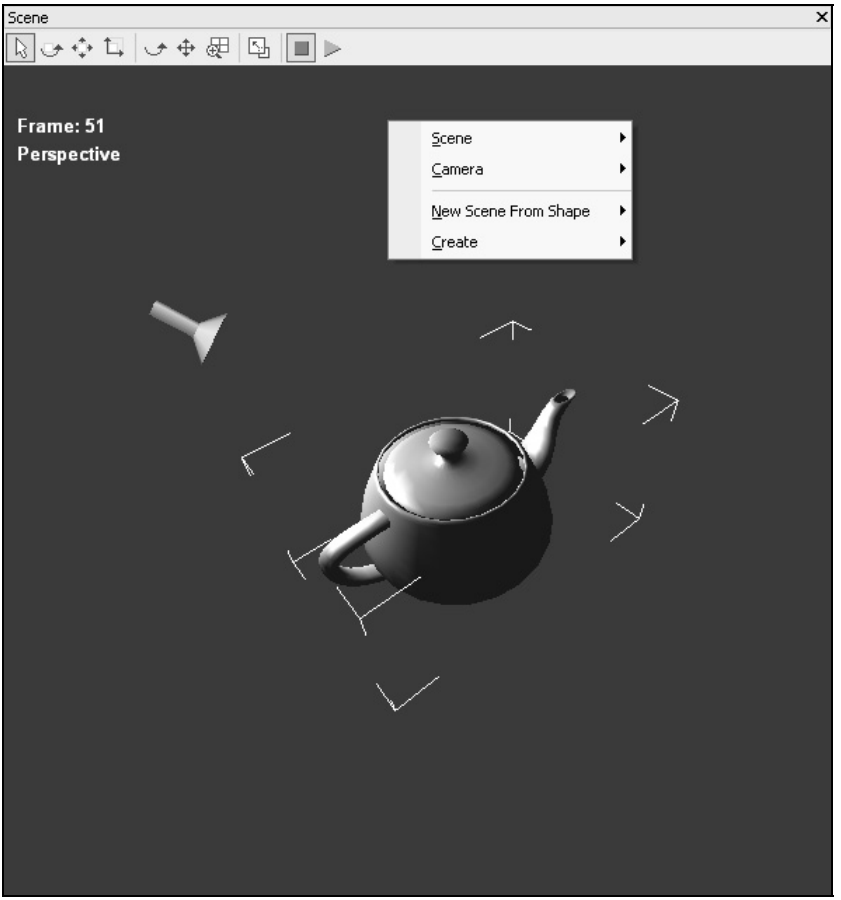


Рис. 9.2. Панель **Scene** с контекстным меню команд

Контекстное меню содержит четыре команды, каждая из которых имеет ряд вложенных команд, отображающихся через вторичные меню, а некоторые команды из вторичного меню включают в себя еще набор команд. Поэтому рассмотрим команды контекстного меню панели **Scene** в виде списка, соблюдая порядок вложенности команд.

- **Scene** — включает следующие команды для работы с объектами сцены:
  - **Render Options** — опции рендеринга. Содержит дополнительные команды:
    - **Wireframe** — каркасный вид объекта;
    - **No Cull** — без отсечения;
    - **Show Normals** — показать нормали объекта;
    - **Show Bounding Boxes** — показать ограничительную область объекта;
    - **Show Transform Hierarchy** — показать иерархию трансформации;
    - **Show Lights** — показать источник света;
    - **Show Camera** — показать камеру;
  - **Manipulate** — допустимые манипуляции над объектом сцены. Имеются следующие команды:
    - **Select Object** — выбрать объект;
    - **Rotate** — вращение объекта;
    - **Pan** — панорамирование сцены, включая все объекты и источники света;
    - **Dolly** — приближение или удаление сцены, включая все объекты и источники света;
    - **Zoom Extents** — реальный размер в пространстве;
    - **Move Object** — перемещение объекта;
    - **Scale Object** — приближение или удаление объекта;
    - **Rotate Scene** — вращение сцены, включая все объекты и источники света;
  - **Properties** — свойства выбранного объекта.
- **Camera** — выбор камеры для сцены. Есть только одна команда:
  - **Default Scene Camera** — задействовать камеру по умолчанию.
- **New Scene From Shape** — формировать новый вид сцены. Для этой опции меню доступны следующие команды:
  - **PlaneYX** — план по осям Y и X;
  - **PlaneXZ** — план по осям X и Z;
  - **PlaneZY** — план по осям Z и Y;
  - **Teapot** — чайник;
  - **Spiral** — спираль;
  - **3DText** — трехмерный текст;

- **Sphere** — сфера;
- **Cylinder** — цилиндр;
- **Cyclide** — циклоида;
- **Cube** — куб;
- **Torus** — полукруг.

❒ **Create** — создание нового вида источника света и объекта с помощью следующих команд:

- **Point Light** — точечный источник света;
- **Spot Light** — прожекторный источник света;
- **Directional Light** — параллельный или направленный источник света;
- **PlaneYX** — план по осям Y и X;
- **PlaneXZ** — план по осям X и Z;
- **PlaneZY** — план по осям Z и Y;
- **Teapot** — чайник;
- **Spiral** — спираль;
- **3DText** — трехмерный текст;
- **Sphere** — сфера;
- **Cylinder** — цилиндр;
- **Cyclide** — циклоида;
- **Cube** — куб;
- **Torus** — полукруг.

Выбирая необходимые команды из контекстного меню панели сцены, вы можете работать с объектами сцены в реальном времени, все произведенные операции моментально вступают в силу. Слежение за процессом изменения работы сцены без перекомпиляции проекта — очень хорошая возможность как для программистов, так и для художников. Взаимодействие объектов сцены проекта также напрямую связано с имеющимися панелями FX Composer, обсуждение функционала которых ждет вас далее в контексте этой главы.

Текстовый редактор

Интеллектуальные возможности текстового редактора FX Composer сравнимы с лучшими экземплярами выпущенных средств в области программирования. Обладая синтаксической подсветкой и удобной функцией предоставления ключевых слов для языка HLSL, текстовый редактор FX Composer

позволяет программисту работать гораздо эффективнее. Редактирование программного кода происходит с помощью "горячих" клавиш либо через меню **Edit**. К сожалению, функции работы с контекстным меню редактора FX Composer 1.5 недоступны. Но имеется возможность редакции значений для элементов сцены с помощью панели **Properties**, о которой вы узнаете из *разд. "Панель свойств" этой главы*. Во всем остальном работа с исходным кодом проекта программ шейдеров в текстовом редакторе мало чем отличается от работы в Visual Studio .NET.

Также в инструментарии FX Composer предусмотрена возможность работы с некоторыми командами языка XML, которые с помощью текстового редактора можно поместить в начало исходного кода в область комментариев. Схема работы команд заключается в использовании следующей конструкции кода:

```
<scene_commands>
// описание действий с помощью команд
</scene_commands>
```

В FX Composer предусмотрено небольшое количество команд, основные из них перечислены в табл. 9.1.

Таблица 9.1. *Команды FX Composer*

Команды	Назначение
`clear`	Очищает цветом область экрана, буфер глубины и трафарета
`draw`	Рисует буфер и объекты сцены
`settechnique`	Вводит в силу установленные изменения
`setcolortarget`	Рисует текстуру
`setdepthstenciltarget`	Изменяет буфер глубины

Подробную информацию о командах вы сможете найти в документации к FX Composer.

Панель свойств

Панель свойств (**Properties**) выполнена в виде таблицы из двух столбцов. Левый столбец на панели **Properties** включает построчный перечень свойств объекта проекта, а правый столбец — заданные значения для каждого из параметров. Например, на рис. 9.3 вы увидите панель **Properties** проекта BumpPlasticPerf из каталога установки FX Composer, находящегося в папке NVIDIA FX Composer/MEDIA/projects/tutorials.

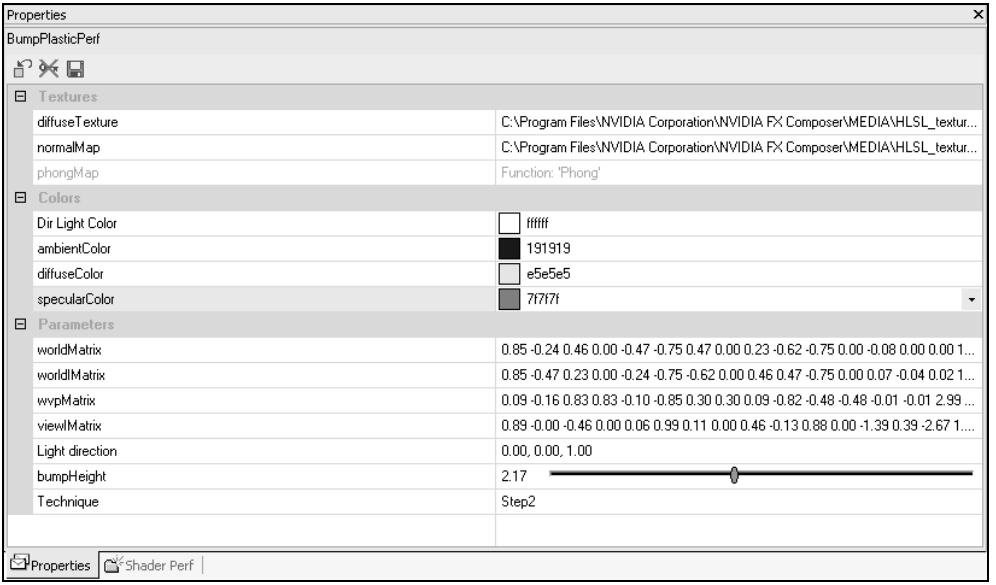


Рис. 9.3. Панель **Properties** проекта BumpPlasticPerf

В проекте BumpPlasticPerf на панели **Properties** имеющиеся свойства объекта отсортированы на группы: **Textures** (Текстуры), **Colors** (Цвета), **Parameters** (Параметры). Каждая группа содержит набор параметров, доступных для редактирования, а также имеет удобное свойство просмотра блока исходного кода для каждого параметра, который используется в коде шейдеров и доступен через текстовый редактор. Если вы щелкните правой кнопкой мыши в левом столбце на названии нужного вам параметра, то откроется небольшое диалоговое окно с программным блоком кода для выбранного параметра. Посмотрите на рис. 9.4, где изображено диалоговое окно параметра ambientColor группы **Colors** проекта BumpPlasticPerf.

Такое диалоговое окно свойств объекта доступно для каждого параметра. Оно функционирует в качестве информационной панели, давая возможность программисту не тратить время на поиск блоков исходного кода для каждого параметра в текстовом редакторе. То есть, просмотрев семантику кода в диалоговом окне для выбранного параметра, вы тут же можете приступить к редакции его значений.

Правый столбец в окне **Properties** содержит заданные значения параметров, которые доступны для редакции как в исходном коде программы шейдеров через текстовый редактор, так и через панель **Properties**. Нажав левой кнопкой мыши в правом столбце панели **Properties** напротив названия одного из параметров, вы сможете редактировать значения выбранного параметра. В зависимости от того, какое из свойств объекта было выбрано, откроются различные редакторы свойств. При изменении значений параметров в этих

редакторах происходит автоматическая замена в исходном коде, который доступен через текстовый редактор FX Compose. Например, в проекте BumpPlasticPerf существуют три группы: **Textures** (Текстуры), **Colors** (Цвета) и **Parameters** (Параметры). Каждая группа имеет свои индивидуальные редакторы для изменения значений параметров проекта.

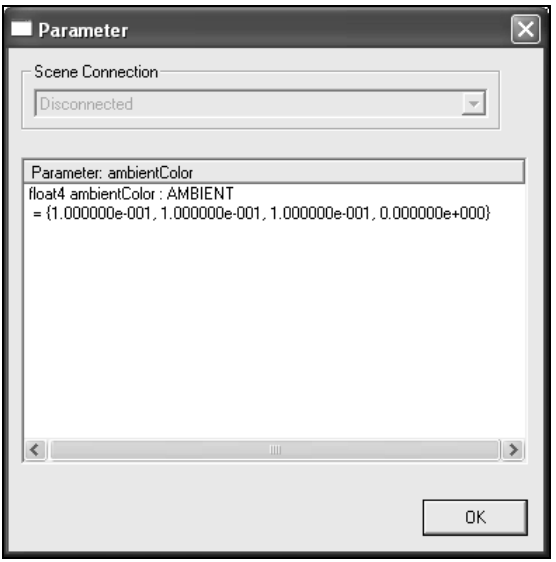


Рис. 9.4. Диалоговое окно для параметра ambientColor

Текстурная группа параметров служит для редакции, а точнее для простой загрузки текстуры через диалоговое окно **Открыть** (Open), которое становится доступным при щелчке левой кнопки мыши в окне **Properties** в правом столбце значений параметров. На рис. 9.5 изображено диалоговое окно **Открыть** для текстурной группы параметров.

Выбрав с помощью диалогового окна **Открыть** необходимый вид текстуры, вы загрузите ее в проект. Инструментарий FX Composer поддерживает текстуры с расширениями dds, png, ipg, bmp, tiff, tif и tga.

Группа **Colors** проекта BumpPlasticPerf содержит ряд переменных, задающих цветовую составляющую. Изменение этой группы элементов происходит с помощью редактора Colors, который становится доступен при нажатии левой кнопки мыши в окне **Properties** в правом столбце значений параметров для проекта BumpPlasticPerf.

Диалоговое окно редактора Colors состоит из двух вкладок (рис. 9.6): **Standard** (Стандарт) и **Custom** (Выборочный). На вкладке **Standard** выбор значения цвета происходит с помощью цветовой гаммы, щелкнув на которой левой кнопкой мыши вы назначите значение цвета для параметра из группы **Colors**. Вторая вкладка **Custom** окна **Colors** содержит также цветовую

гамму и шесть следующих дополнительных инкрементных регуляторов (up-down control):

- **Hue** — оттенки цвета;
- **Sat** — характеристики интенсивности цвета;
- **Lum** — яркость цвета;

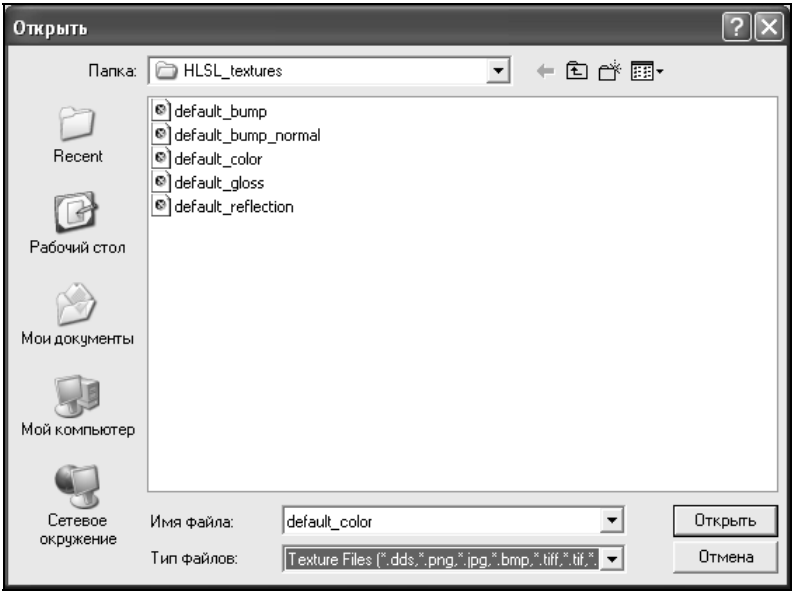
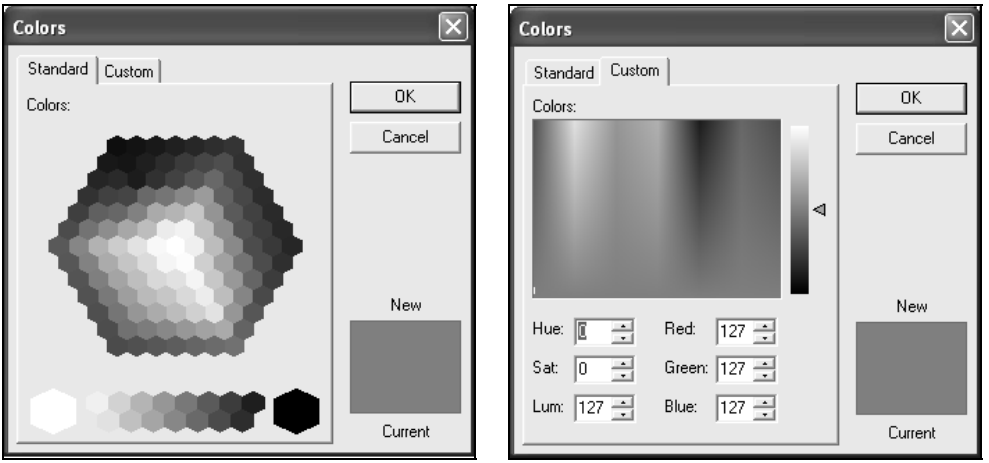


Рис. 9.5. Диалоговое окно **Открыть**



а) *б)*

Рис. 9.6. Окно редактора Colors: *а*) вкладка **Standart**; *б*) вкладка **Custom**

- **Red** — красный цвет;
- **Green** — зеленый цвет;
- **Blue** — голубой цвет.

Шесть инкрементных регуляторов содержат целочисленные значения от 0 до 255, задающие цветовую составляющую для выбранного параметра группы **Colors**.

Следующая группа **Parameters** на панели **Properties** проекта BumpPlasticPerf включает оставшиеся перечисления параметров проекта — это могут быть все виды матриц, векторы, булевы значения и т. д. Группа **Parameters** имеет в своем составе множество различных редакторов, текстовые поля, линейные регуляторы, списки, специализированные диалоговые окна, например на рис. 9.7 представлено диалоговое окно для матрицы вида viewIMatrix на панели **Properties** проекта BumpPlasticPerf.

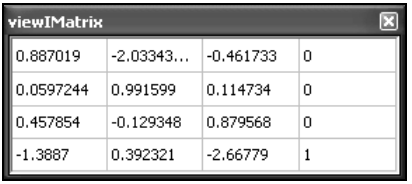


Рис. 9.7. Специализированное диалоговое окно для работы с матрицами

На панели свойств еще имеется своя линейка инструментальных средств с кнопками быстрого доступа, изображенными на рис. 9.8. Всего существует три кнопки: **Reset to defaults** (Сброс настроек), **Delete Keys** (Удалить ключи) и **Save** (Сохранить). С помощью кнопки **Reset to defaults** производится установка значений по умолчанию, кнопка **Delete Keys** служит для сброса измененных значений до первоначальных настроек и кнопка **Save** сохраняет текущие настройки.

Рис. 9.8. Кнопки быстрого доступа на панели **Properties**

При наведении курсора на кнопки быстрого доступа появляется контекстная подсказка, поэтому разобраться, какая из кнопок осуществляет ту или иную команду, будет просто.

Текстурная панель

На текстурной панели (**Textures**) графически представлены задействованные текстуры проекта. При щелчке правой кнопки мыши на одной из текстур

появляется контекстное меню с набором команд, изображенное на рис. 9.9. Команды контекстного меню содержат вложенные вторичные меню.

Рассмотрим содержимое контекстного меню текстурной панели:

- **Channels** — выбор каналов цвета. Эта команда содержит вторичное меню со следующими командами:
  - **Red** — красный;
  - **Green** — зеленый;
  - **Blue** — голубой;
  - **Alpha** — альфа-канал.
- **Source** — источник, в который рисуется текстура. Имеются вложенные команды:
  - **Scene** — сцена;
  - **Material** — материал.
- **Dimensions** — представление размера текстуры. Можно выбрать следующие команды:
  - **Zoom** — увеличение масштаба текстуры;
  - **Actual size** — полноценный размер.
- **Save** — сохраняет настройки.

Перечисленные команды, доступные из контекстного меню, также могут быть запущены с помощью кнопок быстрого доступа, которые имеются на текстурной панели (рис. 9.10).

Рис. 9.9. Контекстное меню текстурной панели

Рис. 9.10. Кнопки быстрого доступа текстурной панели

Структурная панель сцены

Структурная иерархия проекта отражена на панели **Scene Graph** — это некая информационная часть вашего проекта, где можно просматривать зависимость и расположение компонентов в проекте (например, доступных переменных, матриц, объектов, источников света, камер). На структурной панели имеется единственная команда **Properties** (Свойства), которая вызывается из контекстного меню, появляющегося при нажатии правой кнопки мыши

на названии элемента проекта на панели **Scene Graph**. При выборе команды **Properties** для одного из элементов проекта на панели **Properties** вам будут показаны свойства этого элемента, доступные для редакции.

Панель со свойствами материала

Панель **Materials** графически показывает задействованные материалы для объектов сцены. Для загрузки нового материала в проект необходимо воспользоваться командой **File | New | Materials** (Файл | Новый | Материалы). Работа с материалом через панель **Materials** происходит с помощью контекстного меню или кнопок быстрого доступа, расположенных на инструментальной линейке панели **Materials**. При нажатии правой кнопки мыши на одном из материалов откроется контекстное меню с набором команд, изображенное на рис. 9.11, также автоматически на панели текстового редактора и панели **Shader Perf** будет отображаться соответствующий программный код.

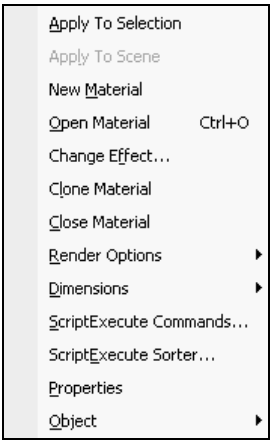


Рис. 9.11. Контекстное меню панели **Materials**

На панели **Materials** присутствуют следующие команды, некоторые из которых имеют вложенные вторичные меню:

- **Apply To Selection** — выбор этой команды делает материал основным в сцене;
- **Apply To Scene** — выбранный материал будет добавлен к объекту сцены;
- **New Material** — открывает диалоговое окно для выбора нового материала, загружаемого в сцену;
- **Open Material** (<Ctrl>+<O>) — открывает диалоговое окно для выбора материала, загружаемого в сцену;

- **Change Effect** — загрузка эффекта в сцену;
- **Clone Material** — клонирует или размножает выбранный материал в проекте;
- **Close Material** — закрывает материал, фактически удаляя его из проекта;
- **Render Options** — опции рендеринга. Имеется вложенная опция:
  - **Wireframe** — каркасный или проволочный метод отображения объекта в сцене;
- **Dimensions** — представление вида материала с помощью опции:
  - **Zoom** — увеличение масштаба материала;
- **ScriptExecute Commands** — открывает информационное диалоговое окно **Scene Commands** с командами сцены для выбранного материала;
- **ScriptExecute Sorter** — позволяет выполнить сортировку материалов проекта;
- **Properties** — при выборе этой команды свойства материала отображаются на панели **Properties**;
- **Object** — накладывает материал в соответствии с выбранным видом объекта, существуют следующие виды:
  - **PlaneYX** — план по осям Y и X;
  - **PlaneXZ** — план по осям X и Z;
  - **PlaneZY** — план по осям Z и Y;
  - **Teapot** — чайник;
  - **Spiral** — спираль;
  - **3DText** — трехмерный текст;
  - **Sphere** — сфера;
  - **Cylinder** — цилиндр;
  - **Cyclide** — циклоида;
  - **Cube** — куб;
  - **Torus** — полукруг.

Работая с контекстным меню панели **Materials**, вы без особых усилий визуальными средствами можете назначать выбранный материал для поверхности объекта, проводя эксперименты в реальном времени. Команды контекстного меню также доступны через кнопки быстрого доступа, изображенные на рис. 9.12, инструментальной линейки панели **Materials**.

Чтобы ознакомиться с кнопками быстрого доступа панели **Materials**, наведите курсор мыши на одну из кнопок, и FX Composer выдаст подсказку по ней.

Рис. 9.12. Кнопки быстрого доступа панели **Materials**

Панель шейдеров

Панель шейдеров (**Shader Perf**), изображенная на рис. 9.13, — это встроенный в FX Composer профайлер, дающий возможности профилировки программы шейдеров. С помощью панели **Shader Perf** можно проследить эффективность работы программы для различных аппаратных средств по эксплуатационным показателям. Инструментарий FX Composer предоставляет возможность эмулировать работу линейки видеоадаптеров серий NVIDIA GeForce FX и GeForce 6800.

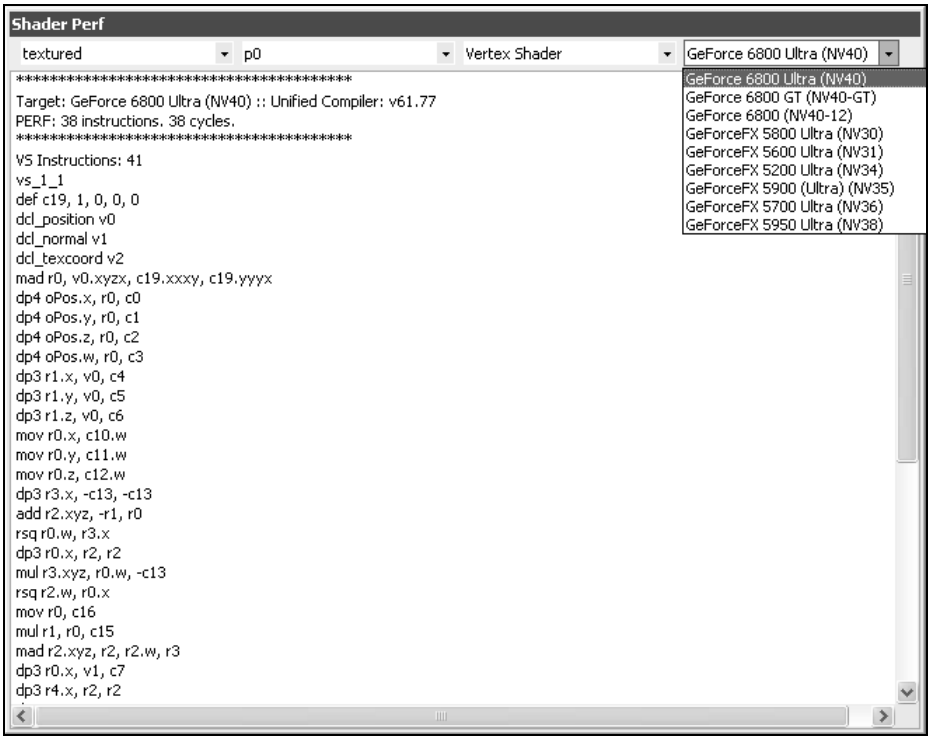


Рис. 9.13. Панель **Shader Perf**

Панель **Shader Perf** содержит большую текстовую область, отображающую ретранслированный ассемблерный исходный код программы шейдеров языка HLSL. Возможность редакции ассемблерного кода шейдеров не преду-

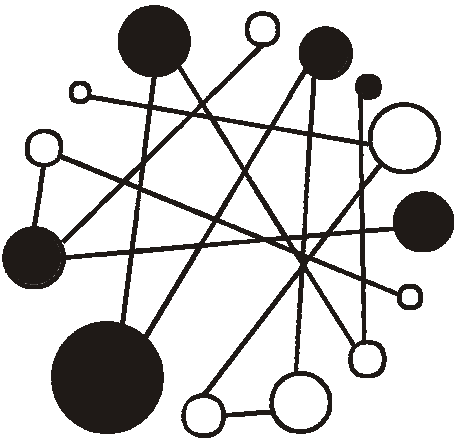
смотрена. При изменении исходного HLSL-кода проекта в текстовом редакторе FX Composer и в ретранслированном ассемблерном коде программы шейдеров после компиляции произойдут изменения согласно совершенным модификациям. Панель **Shader Perf** достаточно мощное средство, позволяющее выполнить оптимизацию шейдерных программ для различных графических процессоров.

Подводя итог возможностей FX Composer, стоит отметить, что данный инструментарий обладает потрясающим потенциалом в работе с шейдерными программами. Интерактивные возможности, продуманный динамический интерфейс, поддержка HLSL, загрузка собственных сцен — все это позволяет рекомендовать FX Composer широкому кругу программистов.

Заключение

Использование специализированных инструментариев значительно упрощает процесс программирования вершинных и пиксельных шейдеров в компьютерных играх. Рассмотренные в книге инструментальные средства обладают большим функционалом и помогут решить вам любую по сложности задачу, связанную с программированием шейдеров. Шейдеры — это будущее компьютерной индустрии. С течением времени программный код игр будет только усложняться, поэтому применение специальных средств становится неотъемлемой частью программирования. Каждый из инструментов достоин внимания, и, я надеюсь, эта книга помогла вам разобраться в многочисленных нюансах, возникающих при работе с рассмотренным в ней программным обеспечением.

Удачи вам!



ПРИЛОЖЕНИЯ

Приложение 1. Web-ресурсы

Приложение 2. Структура компакт-диска

ПРИЛОЖЕНИЕ 1

Web-ресурсы

www.bhv.ru — на сайте издательства "БХВ-Петербург" вы всегда можете узнать о всех выпускаемых новинках и ознакомиться с планом выхода отдельных книг из раздела сайта "Выходят из печати".

www.GameDev.ru — это ведущий русскоязычный сайт, объединяющий сотни программистов. На нем вы найдете множество статей по DirectX и OpenGL, массу исходных кодов, открытые проекты, конкурсы, предложения по работе и многое другое. Конечно, гордость этого сайта большой форум, где можно задать профессионалам вопрос любой сложности.

www.DTF.ru — сайт в большей степени ориентирован на профессионалов и содержит различные материалы в области программирования компьютерных игр с использованием DirectX и OpenGL. На сайте также можно найти большой тематический форум.

www.XDev.ru — этот сайт создан одним-единственным энтузиастом и содержит множество всевозможной информации по DirectX. Имеется неплохой форум, где можно просто задать вопросы или попросить помощи в случае неудачи при создании DirectX-программ.

www.microsoft.com — сайт корпорации Microsoft, где можно найти онлайновую документацию и скачать последнюю версию DirectX SDK.

www.GameDev.net — англоязычный ресурс с огромным количеством статей по физике, трехмерной графике, искусственному интеллекту, 3D-моделированию и многому другому.

www.nvidia.com — сайт компании NVIDIA.

www.ati.com — сайт компании ATI.

www.gDconf.com — англоязычная конференция, созданная для разработчиков компьютерных игр на базе DirectX и OpenGL.

www.Xgames3d.com — сайт компании Xtreme Games LLC, осуществляющей техническую поддержку разработчиков компьютерных игр, на сайте также находятся статьи по программированию игр.

ПРИЛОЖЕНИЕ 2

Структура компакт-диска

Папки	Описание	Главы
\Code	Исходные коды, рассматриваемые в книге	2
\ATI	Установочный файл инструментария RenderMonkey 1.5	6, 7
\NVIDIA	Инсталляционный пакет инструментария FX Composer 1.5	8, 9

Список используемых источников

1. Горнаков С. Г. DirectX 9. Уроки программирования на C++. — СПб.: БХВ-Петербург, 2004.
2. Онлайновая документация с сайта компании Microsoft: **http// www.microsoft.com**.
3. Документация к DirectX 9 SDK.
4. Документация с сайта компании ATI Technologies Inc.: **http// www.ati.com**.
5. Документация к IDE RenderMonkey 1.5 и SDK RenderMonkey 1.0.
6. Документация с сайта NVIDIA: **http// developer.nvidia.com**.
7. Документация к IDE FX Composer 1.5.

Предметный указатель

Б

В

Д

И

К

М

Н

О

П

Продолжение рубрики см. на с. 248