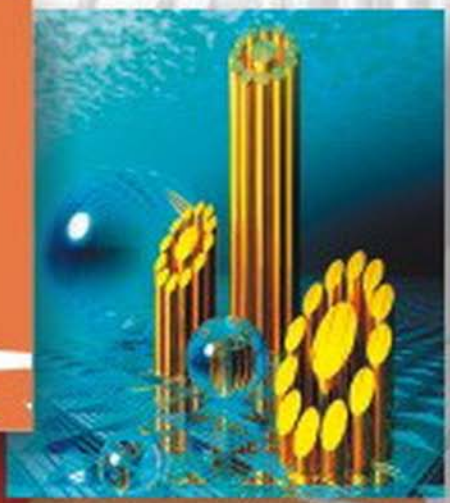
СТАНИСЛАВ ГОРНАКОВ

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.

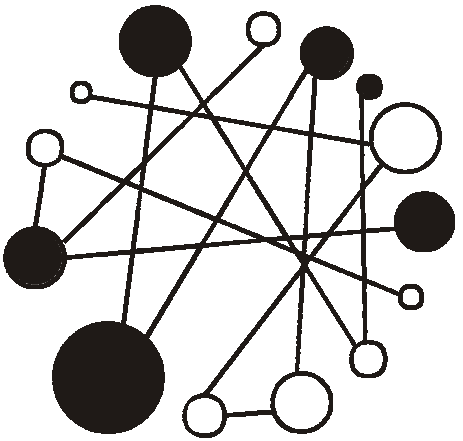
Глава 9 посвящена примеру, на основе которого будут показаны первостепенные принципы работы с FX Composer 1.5.

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

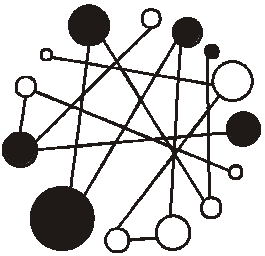
Хочу поблагодарить свою жену Светлану за всестороннюю помощь в создании книги. Особая благодарность Игорю Рыбинскому и издательству "БХВ-Петербург" за предоставление возможности написания этой книги.



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.

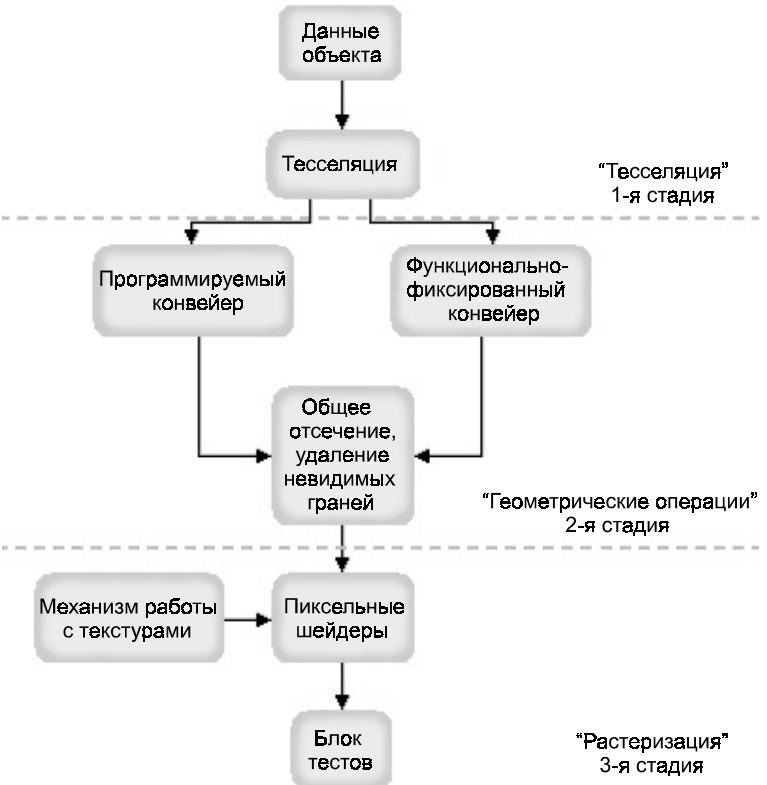


Рис. 1.1. Графический конвейер DirectX

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".

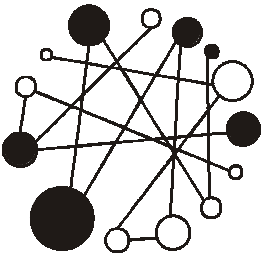
Какой из языков будет в будущем доминировать, предугадать сложно, а может, каждый из языков будет просто "ехать по своей колее". Конечно, в идеале программист может знать все эти языки и найти ему работу будет намного легче, хотя узконаправленный специалист — это лучший специалист. Остается надеяться, 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.

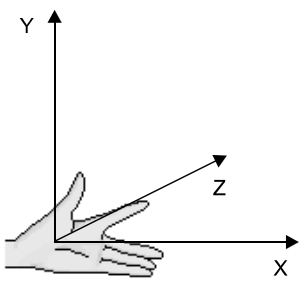


Рис. 2.2. Левосторонняя система координат

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-