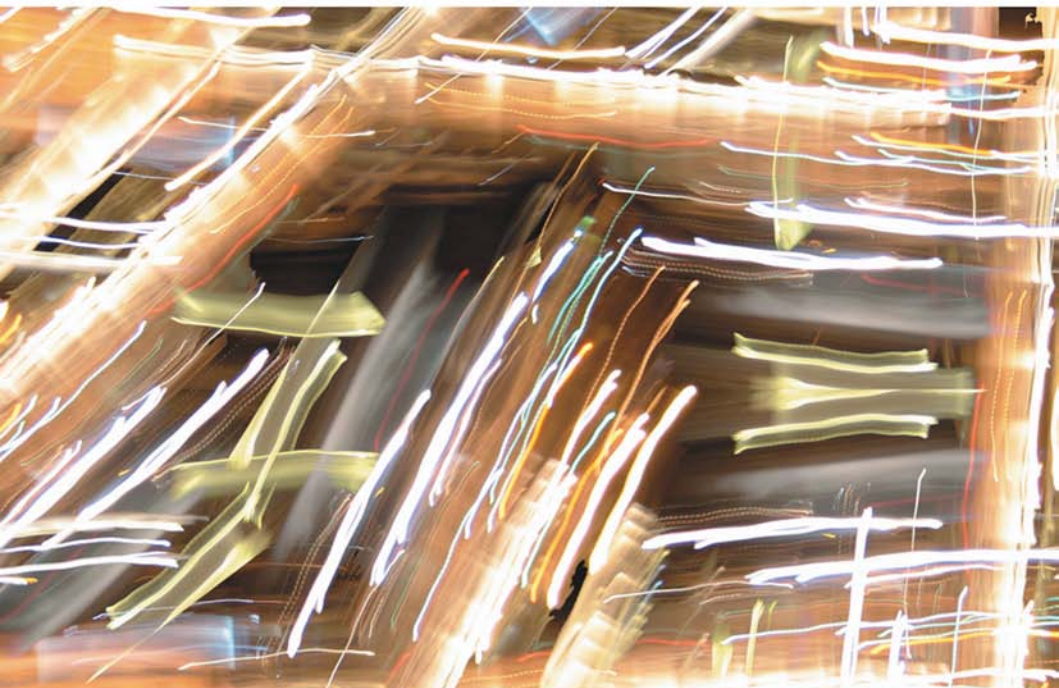


# ИСКУССТВО ПРОГРАММИРОВАНИЯ ИГР НА C++

МИХАИЛ ФЛЕНОВ



**Проблемы создания движка 3D-игры  
Скелетная и вершинная анимация  
Алгоритмы проверки столкновений  
DirectInput и перемещения в 3D-мире  
3D-звук**



**Михаил Фленов**

**Искусство  
программирования  
ИГР на C++**

Санкт-Петербург

«БХВ-Петербург»

2006

УДК 681.3.06  
ББК 32.973.26-018.1  
Ф69

**Фленов М. Е.**

Ф69 Искусство программирования игр на C++. — СПб.: БХВ-Петербург, 2006. — 256 с.: ил.

ISBN 5-94157-832-6

Описаны современные технологии программирования 3D-игр, а также некоторые решения типичных проблем, с которыми может столкнуться программист при их разработке. В качестве практических примеров на протяжении всей книги рассматривается процесс создания простого движка игры, который использует все описываемые технологии: вершинные и пиксельные шейдеры, скелетную и вершинную анимацию, а также компоненты DirectMusic, DirectSound и DirectInput, входящие в библиотеку DirectX. Программный код, приведенный в книге, легко адаптировать и превратить в полноценную игру. Описываемый движок очень прост, но универсален и позволяет создавать игры любого жанра. На компакт-диске к книге содержатся листинги примеров и дополнительная информация по DirectX.

*Для программистов*

УДК 681.3.06  
ББК 32.973.26-018.1

### **Группа подготовки издания:**

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Юрий Рожко</i>
Компьютерная верстка	<i>Ольга Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн обложки	<i>Инны Тачиной</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.07.06.

Формат 70×100<sup>1/16</sup>. 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 5-94157-832-6

© Фленов М. Е., 2006  
© Оформление, издательство "БХВ-Петербург", 2006

# Оглавление

<b>Предисловие</b> .....	<b>6</b>
О чем эта книга .....	7
Что нужно знать .....	9
Что не вошло в книгу .....	9
Благодарности .....	9
Структура книги .....	10
<b>Глава 1. Введение в программирование игр</b> .....	<b>12</b>
1.1. Подготовка к созданию приложения .....	12
1.2. Скелет приложения .....	15
1.3. Инициализация Direct3D .....	20
1.4. Функция формирования сцены .....	23
1.5. Функция загрузки сетки .....	24
1.6. Пример загрузки сетки .....	26
1.7. Моделирование .....	29
1.8. Шейдеры .....	32
1.8.1. Простейший пример шейдеров .....	36
1.8.2. Использование вершинного шейдера .....	41
1.8.3. Пиксельный шейдер .....	46
<b>Глава 2. Разработка движка</b> .....	<b>52</b>
2.1. Структура движка .....	53
2.2. Двигатель объекта .....	54
2.3. Двигатель игры .....	62
2.4. Запуск движка .....	68
2.5. Тени .....	72
2.6. Множество источников освещения .....	83
<b>Глава 3. Скелетная анимация</b> .....	<b>86</b>
3.1. Что такое скелет .....	87
3.2. Формат хранения сетки .....	89
3.3. Основы программирования скелетов .....	95
3.4. Загрузка сетки и скелета из X-файла .....	98
3.5. Разбор объектов .....	103
3.6. Загрузка сетки .....	106
3.7. Загрузка дочерних элементов .....	109

3.8. Поиск фрейма.....	110
3.9. Обновление сетки.....	112
3.10. Обновление сетки.....	113
3.11. Анимация скелета .....	116
3.12. Анимация из X-файла .....	121
3.13. Загрузка анимации из X-файла .....	123
3.13.1. Необходимые классы .....	124
3.13.2. Анализ файла.....	125
3.13.3. Анализ блока анимации .....	126
3.13.4. Анализ ключа .....	127
3.14. Использование анимации .....	129
<b>Глава 4. Войдите.....</b>	<b>132</b>
4.1. Введение в DirectInput .....	132
4.2. Класс для входа .....	134
4.3. Реализация класса ввода.....	137
4.4. Перечисление устройств.....	141
4.5. Опрос состояния действий .....	145
4.6. Использование класса клавиатуры .....	149
<b>Глава 5. "Фейсом об тейбл" .....</b>	<b>154</b>
5.1. Алгоритмы обнаружения.....	154
5.1.1. Точка против плоскости.....	155
5.1.2. Точка против куба .....	155
5.1.3. Точка против сферы .....	156
5.1.4. Сфера против сферы.....	157
5.2. Столкновения в играх .....	157
5.3. Пример реализации коллизий .....	160
5.4. Ниже плинтуса .....	165
5.5. Напутствие.....	169
<b>Глава 6. Вершинная анимация.....</b>	<b>170</b>
6.1. Теория .....	170
6.2. Загрузка ключа .....	175
6.3. Отображение сетки .....	176
6.4. Использование шейдера .....	178
6.5. Множественность кадров .....	182
6.6. Практика использования .....	187
<b>Глава 7. Программирование звука.....</b>	<b>189</b>
7.1. Введение в звук .....	189
7.1.1. <i>IDirectMusicLoader</i> .....	191
7.1.2. <i>IDirectMusicPerformance</i> .....	193
7.1.3. <i>IDirectMusicSegment8</i> .....	194

7.2. Воспроизведение .....	194
7.3. Завершение воспроизведения .....	199
7.4. Погружение в 3D .....	199
7.5. Контроль над 3D .....	206
7.6. Мультизвук .....	207
7.7. Управление параметрами .....	208
7.8. Резюме .....	209
<b>Глава 8. Игровые эффекты .....</b>	<b>210</b>
8.1. Обман зрения .....	210
8.2. Видео .....	213
8.2.1. База для разработки фильтра .....	214
8.2.2. Разработка фильтра .....	216
8.2.3. Подтверждение типов медианных .....	219
8.2.4. Получение кадра .....	221
8.2.5. Загрузка видео и использование фильтра .....	223
8.2.6. Компиляция .....	227
8.3. Мелочи жизни .....	228
8.4. Не все золото, что блестит .....	230
8.5. Искусственный интеллект .....	235
8.6. Оптимизация графики .....	238
<b>Заключение.....</b>	<b>251</b>
<b>Приложение. Описание компакт-диска.....</b>	<b>253</b>
<b>Список литературы .....</b>	<b>254</b>
<b>Предметный указатель .....</b>	<b>255</b>

# Предисловие

С самого начала необходимо определиться, что именно будет рассматриваться в данной книге. Несмотря на то, что в названии присутствует слово "игра", полноценную игру мы писать не будем, потому что для этого понадобится достаточно большая книга или даже несколько книг объемом примерно в 1000 страниц. В книге на 300—400 страницах можно описать только самую простейшую игру определенного жанра и с простейшими графическими возможностями. Но не это наша цель. Наша цель рассмотреть используемые технологии, понять проблемы, с которыми может столкнуться разработчик движка игры<sup>1</sup> и заложить багаж знаний, который пригодится вам в будущем.

Я мог бы написать простую игру и рассказать в книге, как это было сделано, но от нее толку будет мало. Намного интереснее, если вы сами создадите что-то самостоятельно, а данная книга поможет вам сделать это. Мы рассмотрим основные проблемы, с которыми вы можете столкнуться, а превращение кода в полноценную игру будет делом техники, знаний и умений.

Если честно, то популярность игры зависит от графических возможностей не более чем на 30%. Можно привести массу примеров, когда игра становилась популярной, а использовала старинные графические движки и текстуры плохого качества. Недаром даже в те времена, когда на PC уже начал властвовать Wolfstain 3D, мы продолжали на работе раскладывать примитивный тетрис. Проходят годы и появляются более красивые игры, но даже во времена Half-Life мы с удовольствием раскладываем линии (Lines). Самое главное в игре — это сюжет и геймплей (*Геймплей* — это сущность игры — динамика,

---

<sup>1</sup> *Движок игры* — набор функций или классов, которые предоставляют разработчику возможность создавать игровой мир. Чаще всего в движке реализована возможность загрузки, хранения и отображения объектов игрового мира, искусственный интеллект, возможности сетевого обмена данными, поддержка музыки и звуковых эффектов и т. д.

управление, развитие, от которых зависит интерес к самой игре). Если в игру интересно играть, то она будет популярной и чаще всего это не зависит от графики, и этому есть множество подтверждений.

Сложность графики — это всего лишь вспомогательная составляющая, но зато какая приятная для игрока и интересная для программиста. Создавать красивую графику интересно, и смотреть на нее приятно.

Слишком сложная графика может наоборот отрицательно повлиять на популярность игры. Если для формирования сцены необходимо слишком много процессорных ресурсов и для ее запуска требуется компьютер, стоимостью в \$3000, то игра никогда не станет популярной, потому что такими компьютерами обладают единицы и не факт, что все они заплатят за ваш труд. А если сделать игру слишком простой, чтобы она запускалась даже на Pentium с частотой 100 МГц, то на такую графику невозможно будет смотреть и с вероятностью в 99% такую игру ждет полный провал, если нет суперуникального и супервеликолепного геймплея.

Великое искусство — найти идеальное сочетание производительности и качества изображения. Тут приходится применять все возможные алгоритмы оптимизации и нередко нужно обращаться к искусству хакеров. Нет, я не призываю вас писать игру на языке ассемблера или даже в машинных кодах, чтобы она была идеально быстрой, но над оптимизацией придется хорошенечко подумать.

На протяжении всей книги мы будем писать простой движок 3D-игры, а также учиться создавать различные эффекты, которые в сочетании с хорошим геймплеем могут сделать вашу игру шедевром. Да, наработки, которые мы напишем, могут пригодиться вам в будущих проектах, и все же, их нельзя назвать идеальными.

*В программировании нет предела совершенству, но к нему необходимо стремиться.* Если такие слова еще никто не говорил, то запишите их на мой счет ☺, потому что это реальность программиста.

## О чем эта книга

Итак, данная книга посвящена созданию графической части игр, а точнее, оптимизации, графическим эффектам и алгоритмам реализации классических задач, которые встречаются в играх. Для иллюстрации примеров мы будем использовать язык программирования C++ и графический пакет DirectX.

Какое отношение имеют хакеры к играм и графическим эффектам? Дело в том, что графические эффекты получили распространение именно благодаря хакерам. Именно они, с помощью *демо-роликов* — небольших программ, ко-



торые используют скрытые возможности компьютера для создания красивых графических эффектов. Некоторые программисты, попав в мир демо-сцен (demoscene), продолжают совершенствовать свои умения на общественных началах или просто уходят в другую область.

Если не иметь постоянного источника дохода, то такие программисты долго не держатся на сцене (не работают над созданием роликов demoscene), потому что создаваемые ролики не приносят дохода. Можно привести только единичные случаи, когда создатель эффектов и роликов получил за свою работу деньги, и к таким случаям относятся:

- победа на конкурсе;
- получение заказа на создание ролика от какой-нибудь фирмы, например, демо-ролики очень часто можно встретить у таких компаний, как Nvidia, ATI и др.

Получить доход от любимого дела сложно. А ведь нужно как-то кушать самому и кормить свою семью, если она существует. Себя прокормить на пособия по безработице в Европе и штатах можно, а вот семью проблематично. Работать где-то и параллельно заниматься сценой сложно, потому что графика отнимает слишком много времени. Именно поэтому мир демо-сцен (demoscene) принадлежит молодым, и чаще всего не работающим ребятам.

Чтобы остаться в мире графики и получать за свою работу зарплату или другого вида доход, нередко программисты уходят в мир создания компьютерных игр. Именно так можно продолжать оттачивать мастерство в сфере графических эффектов, заниматься любимым делом и получать за это деньги. Это мечта любого землянина — заниматься своим любимым делом за деньги.

Мое мнение, что хакеры, которые начинали свою практику с демо-роликов, очень сильно повлияли на мир компьютерных игр и благодаря им графика становится все более реалистичной. Конечно же, не хакерами едиными живет компьютерная графика, и есть программисты, которые никак не были связаны с демо-сценой, но я думаю, что большинство из них все же что-то использовали из мира хакеров.

Более подробно о демо-сцене и о графических эффектах в роликах вы можете узнать из книг [4] или [6] ("DirectX и C++. Искусство программирования" и "DirectX и Delphi. Искусство программирования". — СПб.: БХВ-Петербург, 2006). Если вы их не читали и мало знакомы с DirectX и ее 9-й версией, то рекомендую прочитать сначала именно одну из этих книг, в зависимости от знакомого вам языка программирования — C++ или Delphi. Дело в том, что книга, которую вы держите сейчас в руках, является своего рода продолжением этих книг, и более глубоко погружает нас в мир компьютерной графики и создания эффектов.

Я буду очень часто ссылаться на книги [4] и [6], которые описывают одни и те же примеры, только для иллюстрации используется свой язык программирования — C++ или Delphi. Нет смысла покупать сразу две книги, возьмите только одну, использующую тот язык программирования, который вы знаете.

## Что нужно знать

Для понимания данной книги вам потребуются знания языков C++ и DirectX. Причем глубоких знаний языка программирования C++ не нужно, потому что в работе с графикой мы практически не будем использовать MFC. В основном нам понадобится математика, циклы и условные операторы.

А вот DirectX желательно знать хорошо, в том числе и работу с шейдерами, благодаря которым можно получить наиболее реалистичные эффекты. Все, что необходимо знать из пакета DirectX, описано в книгах [4] и [6], и если вы их читали, то с пониманием этой книги вопросов не должно возникнуть. Те вопросы, которые не рассматривались в моей первой книге по графике, здесь будут освещены очень подробно. Получается, что лучше все же купить обе книги. Данная работа является как бы продолжением моей первой работы по графике — "DirectX и C++. Искусство программирования" и "DirectX и Delphi. Искусство программирования".

## Что не вошло в книгу

Для создания полноценной игры вам необходимо:

- разработать сценарий;
- создать игровые уровни;
- нарисовать текстуры объектов;
- смоделировать 3D-объекты;
- написать музыку и звуковые эффекты.

Это не полный цикл создания игры, но это все и то, что упущено в списке, рассматриваться не будет. Мы будем разбирать только графическую часть игры, эффекты и оптимизацию, т. е. то, что можно позаимствовать из мира хакеров и использовать в играх.

## Благодарности

Единственное, что читают жена и родители в моих книгах — благодарности. Да, родители больше ничего особо и не понимают в книгах, а жена не особо пытается. Это хорошо, потому что еще один программист в семье будет сер-

езной проблемой. Я даже не могу себе представить этот процесс. Я хочу поблагодарить их за помощь и поддержку, даже несмотря на то, что компьютер отнимает у меня слишком много времени.

Благодарю всех, кто помогает мне в работе: сотрудников издательства "БХВ-Петербург", редакторов, друзей, сотрудников журнала "Хакер" (особенно Лозовского Александра, рецензии которого находятся на задней обложке к большинству моих книг), всех друзей по сайту [www.vr-online.ru](http://www.vr-online.ru) и просто всех, кто меня знает и при этом хотя бы не ненавидит ☺.

Если у вас возникли вопросы по книге или просто пожелания, то я всегда открыт к общению. Пишите мне на адрес издательства [mail@bhv.ru](mailto:mail@bhv.ru) или лучше заходите на форум сайта [www.vr-online.ru](http://www.vr-online.ru). Отвечать на почту мне удастся далеко не всегда, а на форум я захожу каждый день, и по возможности стараюсь помочь всем, у кого возникли сложности или вопросы.

Если вы готовы погрузиться в мир компьютерной графики и игровых эффектов, то переворачивайте страницу, мы будем постепенно погружаться в этот мир. Надеюсь, что он вам понравится.

## Структура книги

Я постарался сделать рассказ максимально последовательным. На протяжении следующих 8 глав вы будете последовательно писать движок игры, постепенно наращивая его возможности. Я настоятельно рекомендую, чтобы вы именно самостоятельно писали, тогда вы:

- ☐ лучше сможете понять код, лучше ориентироваться в нем или даже оптимизировать, а в некоторых главах есть что оптимизировать;
- ☐ быстро сможете адаптировать движок под собственные возможности. Движок хоть и построен специально для обучения, но может использоваться и в реальных боевых условиях.

Теперь посмотрим, что представляют собой 8 глав книги.

- ☐ **Глава 1. Введение в программирование игр.** В этой главе мы поговорим об основах программирования. Большую часть посвятим созданию базового приложения, которое будет использоваться в дальнейшем. Базовая часть будет уметь инициализировать Direct3D, обрабатывать события, отображать сцену и загружать сетки Mesh.
- ☐ **Глава 2. Разработка движка.** В этой главе наше базовое приложение будет обрастать возможностями. Сначала мы напишем класс, с помощью которого можно будет управлять объектами сцены, и научим наш движок отображать эти объекты с учетом освещения и даже теней. Да, о тенях нужно заботиться заранее, а не в самом конце, как делают некоторые разработчики.

- **Глава 3. Скелетная анимация.** Здесь мы поговорим о современном способе создания анимации персонажей игры — людей, животных, птиц и монстров. Все эти персонажи состоят из костей и при их перемещении, движении конечностями, двигаются и кости. В играх тоже можно использовать кости для создания анимации движения. В этой главе вы узнаете, как все это реализуется просто и эффективно.
- **Глава 4. Войдите.** Опрос устройств ввода. Наш движок игры уже "населен" объектами, и вы можете заставить объекты перемещаться, но для этого необходимо только научиться опрашивать устройства ввода и реагировать на них, т. е. перемещать камеру по миру в ответ на нажатия стрелок на клавиатуре.
- **Глава 5. "Фейсом об тейбл".** После *главы 4* вы сможете двигаться по виртуальному миру, но при этом стены на данный момент только отображаются, но не выполняют своей основной функции — не являются препятствием. Двигаться сквозь стены — это нарушение даже примитивных законов физики, поэтому в данной главе мы поговорим о различных алгоритмах определения столкновений.
- **Глава 6. Вершинная анимация.** В *главе 3* мы познакомимся со скелетной анимацией, но она может использоваться далеко не всегда. Например, в кубах нет костей и их таким способом анимировать невозможно, а если вы попытаетесь что-то подобное повернуть, то движения будут нереальными. Одежда, вода и другие мягкие объекты сцены тоже нельзя анимировать через скелеты и в таких случаях приходится прибегать к старому, но проверенному временем методу — вершинной анимации.
- **Глава 7. Программирование звука.** Времена немого кино прошли уже достаточно давно, и без звука уже никуда не деться. Любая игра, даже с простейшими графическими возможностями обязана иметь звуковое и музыкальное сопровождение. В этой главе мы познакомимся со звуковыми возможностями библиотеки DirectX.
- **Глава 8. Игровые эффекты.** В этой главе я собрал небольшие рекомендации и эффекты, которые не вошли в предыдущие главы, а именно — работа с видеороликами, которые вы можете использовать для заставок между уровнями, работа с мелкими объектами, создания различных эффектов и т. д. Эта глава подскажет вам, как сделать игру лучше и красивее.

Желаю вам приятного чтения. Надеюсь, что книга получилась хорошей, по крайней мере, мы все старались (вместе с редакторами, корректорами, дизайнерами).

# ГЛАВА 1



## Введение в программирование игр

Для начала нам необходимо определиться с базой, которая будет использоваться на протяжении всей книги. Напоминаю, если вы читали книгу "DirectX и C++. Искусство программирования" [4], то никаких проблем не возникнет. В некоторых случаях наши примеры будут пересекаться с данной книгой, и основа также будет схожа. Большая часть этой главы делает краткий экскурс по функциям, которые были написаны в книге, на которую я устал давать ссылку (а что поделаешь, если это продолжение), поэтому, если вы уже ознакомились с этой книгой, можно только просмотреть данную главу.

Игры, как и Демо-ролики, требуют серьезной оптимизации. Чем меньше ресурсов потребляет игра и при этом формирует качественную картинку, тем большее число пользователей сможет запустить игру. Если не обращать внимания на оптимизацию, то при максимальных возможностях и большом разрешении минимальными требованиями для игры может стать, например, Pentium 8, который может появиться лет через 5. Такую игру ожидает полный провал.

Итак, давайте потихоньку начнем погружаться в мир программирования. Для этого у вас уже должен быть установлен DirectX SDK и пути на заголовочные файлы и библиотеки должны быть уже прописаны в среде разработки. Надеюсь, вы знаете, как это делается.

### 1.1. Подготовка к созданию приложения

Оптимизировать игры сложнее. Если в Демо-роликах используется только графика и звук и весь код можно реализовать на чистом языке C, то в играх без объектов обойтись сложно. Объекты необходимы, но это не значит, что вы должны использовать библиотеку MFC. Из возможностей этой библиотеки и WinAPI в приложении, применяющей DirectX, необходимо только соз-

дание окна. Больше ничего из API Windows нам не поможет, и будут использоваться только интерфейсы DirectX или функции языка C. Поэтому скелет программы не должен использовать объектов. Давайте создадим скелет будущего игрового приложения.

Итак, создайте пустое приложение **File | New | Project** (Файл | Новый | Проект) и в появившемся окне выберите в дереве **Project Type** (Тип проекта) пункт **Visual C++ Projects | Win32** (Проекты Visual C++ | Win32). Чтобы наши программы были небольшими и быстрыми, мы не будем использовать MFC, поэтому выбираем пункт **Win32 Project** (Проект Win32) (рис. 1.1). В поле **Name** (Имя) укажите имя проекта, а в раскрывающемся списке **Location** (Месторасположение) — путь, по которому будет сохранен проект.

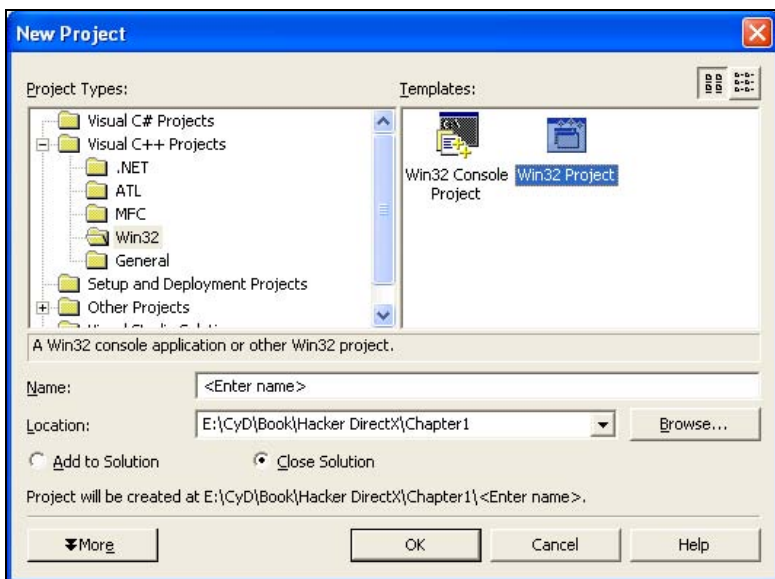


Рис. 1.1. Окно создания нового проекта

После нажатия кнопки **OK** перед нами открывается мастер создания приложения. В разделе **Application Settings** (Настройки приложения) (рис. 1.2) установите тип приложения, выбрав переключатель **Windows application** (Приложение Windows) в группе переключателей **Application type** (Тип приложения), а все прочие флажки должны остаться по умолчанию.

Теперь наше базовое приложение готово. Не забудьте в свойствах проекта подключить библиотеки Direct3D. Для этого выберите меню **Project | Properties** (Проект | Свойства). В дереве свойств выбираем раздел **Configuration Properties | Linker | Input** (Свойства конфигурации | Сборщик | Входящие). Перед вами откроется окно, как на рис. 1.3.

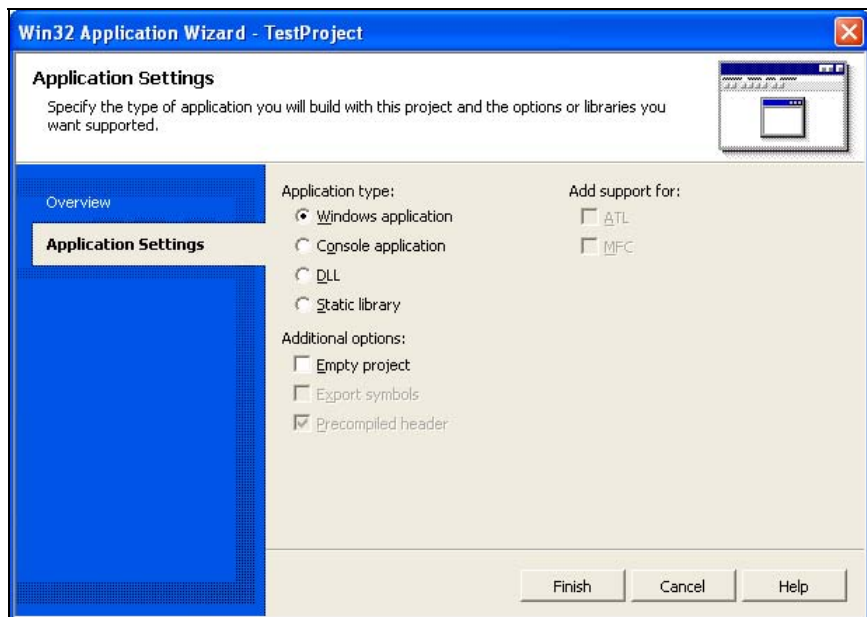


Рис. 1.2. Мастер создания приложения

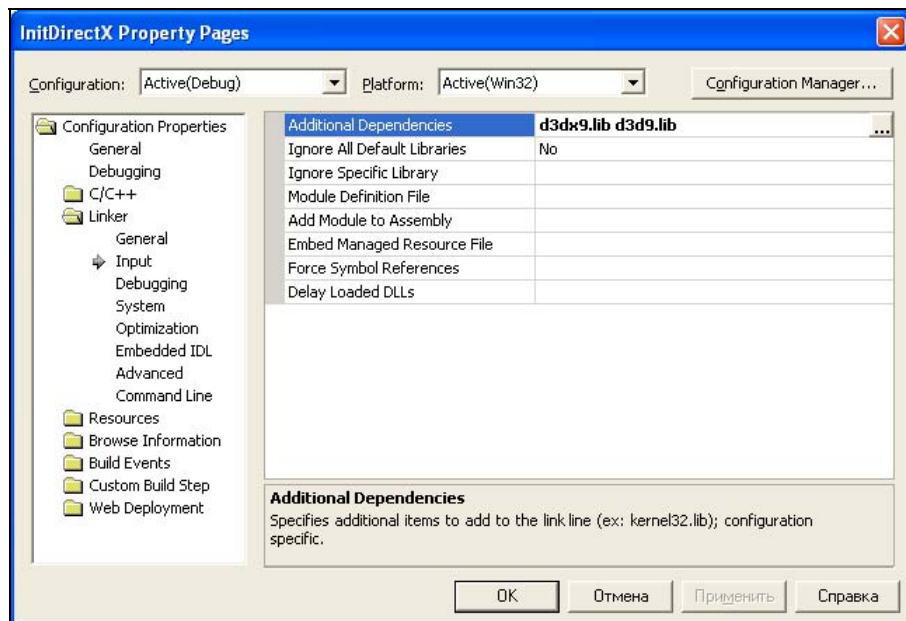


Рис. 1.3. Окно настройки свойств проекта в Visual Studio .NET

В строке **Additional Dependencies** (Дополнительные зависимости) необходимо указать библиотеки, которые нужно подключить во время сборки проекта. Выделите эту строку и щелкните по кнопке с изображением трех точек. Перед вами появится окно, в котором можно указать дополнительные библиотеки (рис. 1.4).

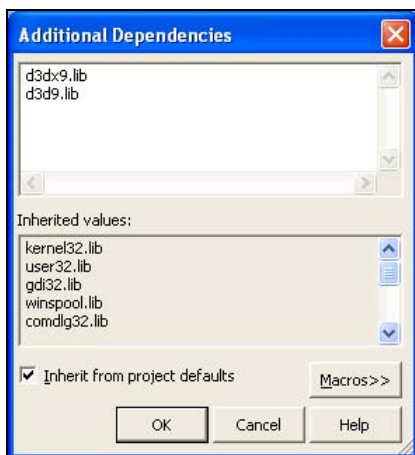


Рис. 1.4. Окно добавления библиотек

Для большинства проектов из данной книги необходимо как минимум указать библиотеки `d3dx9.lib` и `d3d9.lib`. Укажите их каждую в отдельной строке и нажмите **ОК** для сохранения изменений.

Добавить библиотеки нужно для обеих конфигураций: **Release** и **Debug**. Последовательно выберите в выпадающем меню **Configuration** (Конфигурация) оба пункта и добавьте модули.

Помимо этого, в разделе **Configuration Properties | C/C++ | Precompiled Header** (Свойства конфигурации | C/C++ | Предварительно скомпилированные заголовочные файлы) для обеих конфигураций установите в параметре **Create/Use precompiled header** (Создавать/Использовать предварительно скомпилированные заголовочные файлы) параметр **Automatically Generate (/YX)** (Автоматически генерировать).

## 1.2. Скелет приложения

Даже для простого приложения Win32 мастер создает слишком много лишнего. Например, меню и диалоговое окно с информацией о программе. Все это ненужно, если игра будет работать только в полноэкранном режиме. В большинстве игр меню и диалоговые окна Windows не используются, поэтому и



мы не будем этого делать. Удаляем содержимое файла, который сгенерировал мастер, и заменяем его кодом из листинга 1.1.

### Листинг 1.1. Скелет игрового приложения

```
#include <windows.h>
#include "d3d9.h"
#include "d3dx9.h"
#include "..\..\common\dxfunc.h"

// Глобальные переменные
char szWindowClass[] = "Direct3DTemplateProj";
char szTitle[] = "Direct3D Demo by Michael Flenov";

// Объекты Direct3D
IDirect3D9 *pD3D = NULL;
IDirect3DDevice9 *pD3DDevice = NULL;

int iWidth=800;
int iHeight=600;

// Объявление функций
int PASCAL WinMain(HINSTANCE hInst, HINSTANCE hPrev,
    LPSTR szCmdLine, int nCmdShow);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
bool Init(HWND hWnd);
void GraphEngine();

// Главная функция WinMain
int PASCAL WinMain(HINSTANCE hInst, HINSTANCE hPrev,
    LPSTR szCmdLine, int nCmdShow)
{
    WNDCLASSEX wcx;
    MSG msg;
    HWND hWnd;

    CoInitialize(NULL);

    // Регистрируем класс окна
    wcx.cbSize = sizeof(wcx);
    wcx.style = CS_CLASSDC;
    wcx.lpfnWndProc = (WNDPROC)WndProc;
    wcx.cbClsExtra = 0;
```

```
wcex.cbWndExtra = 0;
wcex.hInstance = hInst;
wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
wcex.hbrBackground = NULL;
wcex.lpszMenuName = NULL;
wcex.lpszClassName = szWindowClass;
wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
if(!RegisterClassEx(&wcex))
    return FALSE;

// Создаем окно
hWnd = CreateWindow(szWindowClass, szTitle,
    WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX, CW_USEDEFAULT,
    CW_USEDEFAULT, iWidth, iHeight, NULL, NULL, hInst, NULL);
if(!hWnd)
    return FALSE;
// Отображаем окно
ShowWindow(hWnd, SW_NORMAL);
UpdateWindow(hWnd);

// Инициализация
if(Init(hWnd) == TRUE)
{
    while (true)
    {
        if (PeekMessage(&msg, NULL, NULL, NULL, PM_REMOVE))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
            if (msg.message == WM_QUIT) break;
        }
        GraphEngine();
    }
}

// Очистка выделенных ресурсов
if (pD3DDevice) {pD3DDevice= NULL; pD3DDevice=NULL;}
if (pD3D) {pD3D= NULL; pD3D=NULL;}

CoUninitialize();

return 0;
}
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
    WPARAM wParam, LPARAM lParam)
{
    switch(message) {
        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

bool Init(HWND hWnd)
{
    if (DX3DInitZ(&pD3D, &pD3DDevice, hWnd, iWidth, iHeight, FALSE)!=S_OK)
    {
        MessageBox(hWnd, "DirectX Initialize Error", "Error", MB_OK);
        return FALSE;
    }
    return TRUE;
}

void GraphEngine()
{
    pD3DDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
        D3DCOLOR_XRGB(0,0,0), 1.0f, 0);
    if (SUCCEEDED(pD3DDevice->BeginScene()))
    {
        pD3DDevice->EndScene();
    }

    pD3DDevice->Present(NULL, NULL, NULL, NULL);
}
```

Давайте быстренько пробежимся по этому коду. Из него убраны все функции, которые использовались для инициализации, а осталась только одна WinMain. Регистрация класса, создание окна и отображение окна происходит именно здесь. Чтобы сэкономить несколько десятков байт на ресурсах, я убрал загрузку строк из ресурсов (имя класса и заголовок окна), а прописал эти строки константами. Убрана загрузка акселераторов, потому что не будет меню и визуального интерфейса.

После отображения окна вызывается функция `Init`. Эта функция добавлена для удобства, и в ней будет осуществлена инициализация `Direct3D`. Если результат ее выполнения положителен, то инициализация прошла успешно и можно начинать основной цикл приложения. Иначе, приложение завершит работу.

Теперь посмотрим на цикл обработки сообщений:

```
while (true)
{
    // Если есть сообщение в очереди
    if (PeekMessage(&msg, NULL, NULL, NULL, PM_REMOVE))
    {
        // Обработать сообщение
        TranslateMessage(&msg);
        DispatchMessage(&msg);
        if (msg.message == WM_QUIT) break;
    }
    // Вызвать функцию движка игры
    GraphEngine();
}
```

В данном случае для обработки сообщений (в полноэкранном режиме их будет минимум) мы запускаем бесконечный цикл. Внутри цикла сначала проверяем, есть ли для нас сообщения. Если да, то обрабатываем их. После этого проверяем, если сообщение равно `WM_QUIT`, то необходимо завершить работу программы, поэтому дальнейшее выполнение цикла не имеет смысла, и мы прерываем его с помощью оператора `break`.

Если не было сообщения `WM_QUIT`, то цикл продолжит выполнение, а здесь вызывается функция `GraphEngine`. Эта функция добавлена для удобства, чтобы вынести движок игры в виде отдельной функции, и с ней удобнее было работать.

Не обойтись и без функции `WndProc`, которая необходима для обработки сообщений. Так как у нас не будет никаких пунктов меню, то обрабатываем только событие `WM_DESTROY`, чтобы сделать корректный выход из программы.

Теперь рассмотрим функцию `Init`, которая инициализирует `Direct3D` и задает параметры отображения по умолчанию. В шаблонном приложении она выглядит следующим образом:

```
bool Init(HWND hWnd)
{
    // Инициализация Direct3D
    if (DX3DInitZ(&pD3D, &pD3DDevice, hWnd,
                iWidth, iHeight, FALSE)!=S_OK)
```

```

{
    MessageBox(hWnd, "DirectX Initialize Error", "Error", MB_OK);
    return FALSE;
}

return TRUE;
}

```

Основа функции `Init` — это вызов функции `DX3DInitZ`. Эта функция была написана в книге "DirectX и C++. Искусство программирования" [4] и для удобства вынесена в отдельный модуль `dxfunc.cpp`.

### Примечание

Файл `dxfunc.cpp` можно найти на компакт-диске в каталоге `Common`.

## 1.3. Инициализация Direct3D

Функция `DX3DInitZ` универсальна и удобна. Достаточно подключить модуль к любому проекту и проинициализировать `Direct3D` вызовом одной функции `DX3DInitZ`. Эту функцию вы можете увидеть в листинге 1.2.

### Листинг 1.2. Функция `DX3DInitZ` для инициализации `Direct3D`

```

HRESULT DX3DInitZ(IDirect3D9 **ppiD3D9,
    IDirect3DDevice9 **ppiD3DDevice9, HWND hWnd,
    DWORD iWidth, DWORD iHeight, BOOL bFullScreen)
{
    // Инициализация Direct3D 9-й версии
    if ((*ppiD3D9 = Direct3DCreate9(D3D_SDK_VERSION)) == NULL)
        return E_FAIL;

    // Заполняем основные параметры представления
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));
    d3dpp.BackBufferWidth = iWidth;
    d3dpp.BackBufferHeight = iHeight;
    d3dpp.AutoDepthStencilFormat = D3DFMT_D16;
    d3dpp.EnableAutoDepthStencil = TRUE;

    // Запрос на отображение в полноэкранном режиме
    int iRes;
    if (!bFullScreen)
        iRes=MessageBox(hWnd, "Use fullscreen mode?", "Screen",
            MB_YESNO | MB_ICONQUESTION);
}

```

```
else
    iRes = IDYES;

if(iRes == IDYES)
{
    // Полноэкранный режим
    // Установка параметров полноэкранного режима
    d3dpp.BackBufferFormat = D3DFMT_R5G6B5;
    d3dpp.SwapEffect = D3DSWAPEFFECT_FLIP;
    d3dpp.Windowed = FALSE;
    d3dpp.FullScreen_RefreshRateInHz = D3DPRESENT_RATE_DEFAULT;
    d3dpp.PresentationInterval = D3DPRESENT_INTERVAL_DEFAULT;
}
else
{
    // Оконный режим
    RECT wndRect;
    RECT clientRect;

    // Корректируем размер окна, чтобы клиентская область была
    // четко указанного размера
    GetWindowRect(hWnd, &wndRect);
    GetClientRect(hWnd, &clientRect);
    iWidth = iWidth + (wndRect.right-wndRect.left) -
        (clientRect.right-clientRect.left);
    iHeight = iHeight + (wndRect.bottom-wndRect.top) -
        (clientRect.bottom-clientRect.top);

    // Устанавливаем размеры окна
    MoveWindow(hWnd, wndRect.left, wndRect.top, iWidth, iHeight, TRUE);

    // Получить формат пиксела
    D3DDISPLAYMODE d3ddm;
    (*ppiD3D9)->GetAdapterDisplayMode(D3DADAPTER_DEFAULT, &d3ddm);

    // Установка параметров
    d3dpp.BackBufferFormat = d3ddm.Format;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.Windowed = TRUE;
}

DWORD Flags= D3DCREATE_MIXED_VERTEXPROCESSING;

// Создать 3D-устройство
HRESULT hRes;
```

```

if(FAILED(hRes = (*ppiD3D9)->CreateDevice(D3DADAPTER_DEFAULT,
    D3DDEVTYPE_HAL, hWnd, Flags, &d3dpp, ppiD3DDevice9)))
    return hRes;

// Установить перспективу
float Aspect = (float)d3dpp.BackBufferWidth /
    (float)d3dpp.BackBufferHeight;
D3DXMATRIX matProjection;
D3DXMatrixPerspectiveFovLH(&matProjection, D3DX_PI/4.0f, Aspect,
    2.0f, 1000.0f);
(*ppiD3DDevice9)->SetTransform(D3DTS_PROJECTION, &matProjection);

// По умолчанию освещение будет отключено
(*ppiD3DDevice9)->SetRenderState(D3DRS_LIGHTING, FALSE);

return S_OK;
}

```

Бегло посмотрим и эту функцию. В качестве параметров она получает:

- переменную для хранения интерфейса Direct3D;
- переменную для хранения интерфейса устройства Direct3D;
- идентификатор окна;
- ширину окна;
- высоту окна.

Кроме того, надо учитывать, что если последний параметр равен `true`, то необходимо использовать полноэкранный режим, иначе оконный.

В самом начале функции инициализируется интерфейс Direct3D. Затем заполняются параметры представления (структура `D3DPRESENT_PARAMETERS`), которая необходима для инициализации устройства Direct3D. Значения некоторых параметров этой структуры будут отличаться в зависимости от оконного или полноэкранного режима. Одинаковыми будут только ширина и высота заднего буфера. Заполнив эти параметры, отображаем на экране запрос — нужно ли использовать полноэкранный режим.

Далее идет уже разделение кода на две части. Сначала заполняются параметры для полноэкранного режима, а затем для оконного.

Когда структура `D3DPRESENT_PARAMETERS` готова, вызываем метод `CreateDevice` для создания устройства. Тут у нас указываются следующие параметры:

- адаптер будет по умолчанию;
- использовать аппаратные возможности видеокарты;

- ❑ в качестве окна будет использоваться идентификатор, который передан в качестве параметра;
- ❑ дополнительные флаги. Здесь передаем переменную `Flag`, которая содержит значение `D3DCREATE_MIXED_VERTEXPROCESSING`, т. е. используется смешанный режим обработки вершин;
- ❑ заполненная структура `D3DPRESENT_PARAMETERS`;
- ❑ переменная, в которую будет записан результат, т. е. указатель на устройство `Direct3D`.

Далее идет настройка матрицы проекции. В качестве соотношения сторон берем соотношение ширины и высоты экрана. Последняя строка кода отключает освещение. По умолчанию источники света отключены. Необходимо включать освещение только тогда, когда свет действительно нужен. Хотя в играх для улучшения реалистичности свет нужен всегда, я его отключил. Почему? Да потому что максимальную реалистичность может дать использование шейдеров, что мы и будем делать в примерах данной книги. Да, алгоритм освещения я буду использовать максимально простой, но вы легко сможете его заменить на что-то более качественное и подходящее для данной атмосферы.

## 1.4. Функция формирования сцены

Теперь посмотрим на функцию `GraphEngine`, где будет формироваться графика. Функция выглядит следующим образом:

```
void GraphEngine()
{
    // Здесь необходимо произвести предварительные расчеты
    ...

    // Очистить буфер
    pD3DDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
        D3DCOLOR_XRGB(0,0,0), 1.0f, 0);
    if (SUCCEEDED(pD3DDevice->BeginScene()))
    {
        // Здесь необходимо формировать сцену
        ...
        ...

        // Завершаем формирование сцены
        pD3DDevice->EndScene();
    }

    pD3DDevice->Present(NULL, NULL, NULL, NULL);
}
```



В самом начале функции необходимо произвести предварительные расчеты. Затем вызывается метод `Clear` интерфейса устройства `Direct3D` для очистки заднего буфера. Затем вызывается метод `BeginScene`, и если все прошло удачно, то можем формировать сцену. Формирование заканчивается вызовом метода `EndScene`. Теперь необходимо отобразить сцену на экране. Для этого вызываем метод `Present`.

### Примечание

Исходный код проекта можно найти на компакт-диске в каталоге `Chapter1/TemplateProj`.

## 1.5. Функция загрузки сетки

В книге "DirectX и C++. Искусство программирования" [4] была написана еще одна очень удобная функция, которую будем использовать и сейчас — `LoadMesh`. Эта функция удобна для загрузки сеток `Mesh` из `X`-файлов. Эту функцию вы можете также найти в файле `dxfunc.cpp` и в листинге 1.3.

### Листинг 1.3. Функция загрузки сеток

```

DWORD LoadMesh (char *filename, IDirect3DDevice9 *ppiD3DDevice9,
    ID3DXMesh **ppMesh, LPDIRECT3DTEXTURE9 **pMeshTextures,
    char *texturefilename, D3DMATERIAL9 **pMeshMaterials)
{
    LPD3DXBUFFER pD3DXMtrlBuffer;
    DWORD dwNumMaterials;

    // Загрузка сетки из файла
    D3DXLoadMeshFromX(filename, D3DXMESH_SYSTEMMEM, ppiD3DDevice9,
        NULL, &pD3DXMtrlBuffer, NULL, &dwNumMaterials, ppMesh);

    // Получаем указатель на материалы
    D3DXMATERIAL* d3dxMaterials =
        (D3DXMATERIAL*)pD3DXMtrlBuffer->GetBufferPointer();

    // Переменные для хранения массивов текстур и материалов
    (*pMeshTextures) = new LPDIRECT3DTEXTURE9[dwNumMaterials];
    (*pMeshMaterials) = new D3DMATERIAL9[dwNumMaterials];

    // Запускаем цикл заполнения массивов
    for( DWORD i=0; i<dwNumMaterials; i++ )

```

```
{
    // Копируем материал
    (*pMeshMaterials)[i] = d3dxMaterials[i].MatD3D;

    // Создаем текстуру
    if( FAILED(D3DXCreateTextureFromFile(ppiD3DDevice9,
        d3dxMaterials[i].pTextureFilename, &(*pMeshTextures)[i]))
        if( FAILED(D3DXCreateTextureFromFile(ppiD3DDevice9,
            texturefilename, &(*pMeshTextures)[i]))
            (*pMeshTextures)[i] = NULL;
}

// Возвращаем количество материалов в загруженной сетке
return dwNumMaterials;
}
```

Коротко "пробежимся" по этой функции. В самом начале загружаем сетку из X-файла с помощью функции `D3DXLoadMeshFromX`. Эта функция выглядит следующим образом:

```
HRESULT D3DXLoadMeshFromX(
    LPCTSTR pFilename,
    DWORD Options,
    LPDIRECT3DDEVICE9 pDevice,
    LPD3DXBUFFER* ppAdjacency,
    LPD3DXBUFFER* ppMaterials,
    LPD3DXBUFFER* ppEffectInstances,
    DWORD* pNumMaterials,
    LPD3DXMESH* ppMesh
);
```

Посмотрим параметры функции:

- `pFilename` — имя загружаемого файла;
- `Options` — опции, которые будут использоваться при загрузке. Опций достаточно много, но мы пока будем использовать только `D3DXMESH_SYSTEMMEM`, что означает, что загрузка должна происходить в системную память;
- `pDevice` — указатель на интерфейс `IDirect3DDevice9`;
- `ppAdjacency` — указатель на буфер для смежных граней;
- `ppMaterials` — указатель на буфер для материалов;
- `ppEffectInstances` — указатель на буфер для экземпляров эффектов;

- `pNumMaterials` — через этот параметр будет получено количество загруженных материалов;
- `ppMesh` — непосредственно указатель на объект сетки.

Функция `D3DXLoadMeshFromX` загружает только сетку, из которой состоит объект, материалы и имена текстур. Сами текстуры будут храниться в отдельном бинарном файле. Да и сами материалы находятся в не очень удобном виде и их лучше перебросить в отдельный массив типа `D3DXMATERIAL`.

Итак, после загрузки сетки создаем два массива: `pMeshTextures` и `pMeshMaterials`. В первый будем сохранять текстуры, а во второй материалы. Далее запускается цикл, в котором заполняются массивы. Для загрузки текстуры из файла используем функцию `D3DXCreateTextureFromFile`, которая выглядит следующим образом:

```
HRESULT D3DXCreateTextureFromFile(
    LPDIRECT3DDEVICE9 pDevice,
    LPCTSTR pSrcFile,
    LPDIRECT3DTEXTURE9 *ppTexture
);
```

Здесь у нас имеется три параметра:

- `pDevice` — указатель на интерфейс `IDirect3DDevice9`, с которым должна быть связана текстура;
- `pSrcFile` — имя файла;
- `ppTexture` — указатель на интерфейс типа `IDirect3DTexture9`, куда будет загружена текстура.

Если во время загрузки произошла ошибка, то пытаемся загрузить файл, который был указан в качестве параметра функции. Если и тут произошла ошибка, то текстуры не будет.

## 1.6. Пример загрузки сетки

Теперь посмотрим, как можно использовать функцию `LoadMesh`. Загрузите шаблонный проект, на основе которого мы рассмотрим загрузку сетки. Для начала объявим необходимые глобальные переменные, а понадобится нам следующее:

```
DWORD dwNumMaterials; // Для хранения количества материалов
ID3DXMesh *pMesh; // Для хранения сетки
LPDIRECT3DTEXTURE9 *pMeshTextures; // Текстуры
D3DMATERIAL9 *pMeshMaterials; // Материалы
```

Теперь переходим в функцию `Init`, где происходит инициализация проекта, и после инициализации `Direct3D` добавляем следующий код:

```
dwNumMaterials = LoadMesh("tiny.x", pD3DDevice,
    &pMesh, &pMeshTextures, "texture.bmp", &pMeshMaterials);

// Для примера создадим источник света Direct3D
D3DLIGHT9 light;
ZeroMemory(&light, sizeof(D3DLIGHT9));
light.Type = D3DLIGHT_DIRECTIONAL;
light.Direction = D3DXVECTOR3(0.5f, 0.0f, 0.5f);
light.Diffuse.r = light.Diffuse.g = light.Diffuse.b = light.Diffuse.a =
1.0f;
pD3DDevice->SetLight(0, &light);
pD3DDevice->LightEnable(0, TRUE);
```

В этом примере загружается файл `tiny.x`, который идет в поставке `DirectX SDK` и специально предназначен для того, чтобы тестировать свои приложения. После загрузки сетки инициализируем освещение, чтобы загружаемый объект не был абсолютно черным, и вы увидели его во всей красе. В данном случае используется освещение `Direct3D`, дабы не усложнять сейчас пример, но в будущем мы перейдем на использование шейдеров.

Теперь посмотрим, как можно отобразить загруженную сетку. Конечно же, это необходимо делать в функции `GraphEngine`, код которой для данного примера вы можете увидеть в листинге 1.4.

#### Листинг 1.4. Пример отображения сетки

```
void GraphEngine()
{
    // Очистить буферы
    pD3DDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
        D3DCOLOR_XRGB(255,255,255), 1.0f, 0);

    if (SUCCEEDED(pD3DDevice->BeginScene()))
    {
        // Увеличиваем угол поворота сцены
        ViewAngle+=0.01f;
        float b = ViewAngle;

        // Рассчитываем мировую матрицу для определения положения объекта
        D3DMATRIX World = {
            cos(b)*cos(b), cos(b)*sin(b), sin(b), 0,
            -sin(b), cos(b), 0, 0,
```

```

    -sin(b)*cos(b), -sin(b)*sin(b), cos(b), 0,
    0, 0, 700, 1,
};
pD3DDevice->SetTransform(D3DTS_WORLD, &World);

// Включаем освещение
pD3DDevice->SetRenderState(D3DRS_LIGHTING, TRUE);

// Запускаем цикл отображения
for (DWORD i=0; i<dwNumMaterials; i++)
{
    // Устанавливаем материал
    pD3DDevice->SetMaterial(&pMeshMaterials[i]);

    // Устанавливаем текстуру
    if (pMeshTextures[i])
        pD3DDevice->SetTexture(0, pMeshTextures[i]);

    // Отображаем очередную часть сетки
    pMesh->DrawSubset(i);
}

pD3DDevice->EndScene();
}

// Отображаем сцену
pD3DDevice->Present(NULL, NULL, NULL, NULL);
}

```

Результат работы программы можно увидеть на рис. 1.5. Программирую графику уже давно, но до сих пор поражаюсь, как просто создать такую красоту. Конечно, красота зависит от содержимого X-файла, но и DirectX вносит свою красоту в эту сцену.

Теперь посмотрим на код отображения. После начала формирования сцены увеличиваем счетчик `ViewAngle`, который определяет угол поворота фигуры. Да, она будет вращаться, чтобы ее можно было разглядеть с разных сторон. После этого включаем освещение.

Далее идет самое интересное — цикл отображения сетки. Сетка может состоять из множества частей и каждая из них может иметь свою текстуру и материал. Количество шагов цикла — это количество загруженных материалов. В идеале, каждая фигура должна быть из одного материала/текстуры, чтобы цикл не выполнялся слишком долго и не создавал излишних накладных расходов.

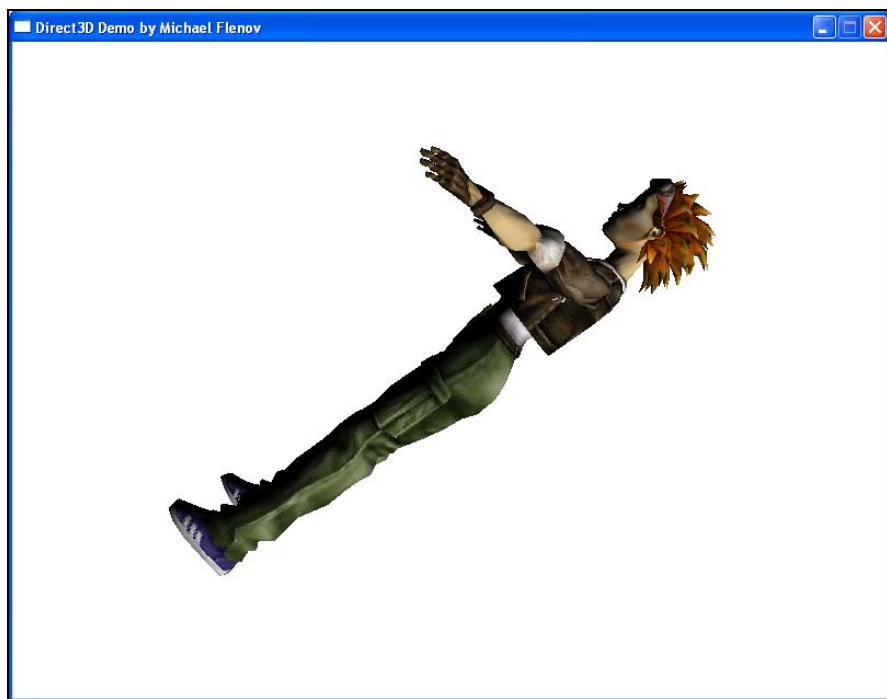


Рис. 1.5. Результат работы примера

На каждом этапе цикла устанавливаем материал, и если существует текстура, то устанавливаем и ее. После этого отображаем текущую сетку с помощью вызова метода `DrawSubset` интерфейса `ID3DXMesh`.

### Примечание

Исходный код проекта из этой главы можно найти на компакт-диске в каталоге `Chapter1/Mesh`.

## 1.7. Моделирование

В играх сложно обойтись без объектов сеток. Создавать объекты игры вручную из треугольников достаточно утомительная задача. Намного проще смоделировать объекты с помощью 3D-редактора и потом загрузить их в своей программе. Хороший 3D-редактор с помощью визуальных средств позволяет легко и быстро создавать достаточно сложные модели. Я предпочитаю использовать `3D Studio Max` (<http://www.discreet.com>). Да, это дорогой продукт, но универсальный и позволяет создать одну и ту же фигуру разными способами.