СТАНИСЛАВ ГОРНАКОВ

bhv

DirectX 9

УРОКИ ПРОГРАММИРОВАНИЯ НА C++

ПРОГРАММИРОВАНИЕ ТРЕХМЕРНОЙ ГРАФИКИ

КОМПОНЕНТЫ Direct3D, DirectInput, DirectMusic И DirectSound

МАТРИЧНЫЕ ПРЕОБРАЗОВАНИЯ

ПРОГРАММИРОВАНИЕ ВЕРШИННЫХ И ПИКСЕЛЬНЫХ ШЕЙДЕРОВ

СПРАВОЧНИК ПО DirectX 9 SDK

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Станислав Горнаков

DirectX 9

УРОКИ ПРОГРАММИРОВАНИЯ НА C++

Санкт-Петербург
«БХВ-Петербург»
2004

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1
Г67

Горнаков С. Г.

Г67 DirectX 9: Уроки программирования на C++. — СПб.: БХВ-Петербург, 2004. — 400 с.: ил.

ISBN 5-94157-482-7

Рассмотрено профессиональное программирование трехмерной графики под Windows на языке C++ с использованием библиотеки DirectX 9. Раскрыты возможности компонента Direct3D по выводу трехмерной графики, текстурированию объектов, работе с освещением, вершинными и пиксельными шейдерами и др. Описаны также компоненты DirectInput, DirectMusic и DirectSound. Материал изложен в виде уроков и поможет читателю самостоятельно изучить технологию DirectX, на основе которой создаются профессиональные компьютерные игры. Прилагаемый компакт-диск содержит примеры, рассмотренные в книге.

Для программистов

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Яковлева*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.10.04.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 32,25.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

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

ISBN 5-94157-482-7

Содержание

Введение

Технология DirectX от корпорации Microsoft сегодня является стандартом программирования игр под платформу Windows. Более 90% компьютерных игр пишутся на языке C++ с использованием библиотеки DirectX. Со времен появления первой графической, и действительно мощной операционной системы Windows 95, корпорация Microsoft пытается выпускать мультимедийные библиотеки, посредством которых можно было бы заниматься программированием компьютерных игр. Самый первый пакет назывался Game SDK. Он был выпущен примерно в тот период, когда Windows 95 завоевывала весь мировой рынок. Game SDK был полностью основан на растровой графике, но это и вполне понятно, в тот момент просто не было мощных 3D-адаптеров, также имелась поддержка звука и устройств ввода. Год спустя выходят одна за другой версии DirectX 2 и DirectX 3, в их составе появляется Direct3D. После выхода Windows 98 библиотека DirectX пополнилась пятой и шестой версией. И только в 1999 году появилась DirectX 7, ознаменовавшая новую эпоху трехмерных игр. Через год выходит DirectX 8, после чего все сомнения по поводу того, на чем и с чем писать компьютерные игры, разваливаются как карточный домик. В середине 2003 года выпускается новая версия DirectX 9 и на данный момент это самая последняя версия, о которой и пойдет речь в этой книге.

О чем эта книга

Вся книга построена на уроках по DirectX 9 с использованием языка программирования C++. Уроки необходимо изучать в хронологическом порядке; без понимания предыдущего урока последующий урок будет практически недоступен для освоения.

На *первом уроке* раскрываются основы программирования под Windows, где в частности будет создано оконное приложение, в которое в дальнейшем будет интегрироваться исходный код каждого урока.

Второй урок даст вам ответ, что же такое DirectX, из каких компонентов состоит рассматриваемая библиотека, на основе какой из технологий строится DirectX. Соответственно будут получены необходимые сведения по COM и большой вводный курс в Direct3D, также будет изучена общая концепция построения сцены в Direct3D, а в конце всего урока подробно рассмотрена инсталляция DirectX 9 SDK на ваш компьютер.

На *третьем уроке* будет рассмотрена самая ключевая информация начального этапа работы с Direct3D — это инициализация Direct3D, создание устройства Direct3D и настройка аппаратной части компьютера, необходимой для работы. К слову сказать, все уроки до тринадцатого включительно будут посвящены Direct3D.

На *четвертом уроке* мы построим и выведем на экран простой треугольник с помощью буфера вершин и конвейера рендеринга, получая при этом основную информацию о геометрии сцены в Direct3D.

Матрицам посвящен *пятый урок*. Если вы забыли либо не знали, что такое матрицы, как сложить и перемножить их между собой, этот урок для вас. Матрицы в Direct3D имеют свое специфическое значение, об этом и многом другом весь пятый урок.

Шестой урок будет ознаменован выводом на экран полноценного трехмерного объекта с помощью концепции индексации вершин.

Буфер глубины — это тема *седьмого урока*. Сортировка точек объекта в зависимости от расстояния до глаз — это то, для чего необходим буфер глубины, имеющий еще название Z-буфер.

Восьмой урок посвящен материалу и свету. Затрагивается тема модели освещения, дается представление об источниках света, раскрывается общий принцип взаимодействия материала с освещением объекта, на основе которого строится общее восприятие цвета глазом человека. Нормали, необходимые для расчета освещения, тоже обсуждаются на этом уроке.

Текст, выводимый на экран, имеет огромное значение в играх, об этом весь *девятый урок*. В конце урока будет осуществлен переход из оконного режима в полноэкранный.

Десятый урок раскрывает основы текстурирования. Рассматривается стандартный способ наложения текстур на объект.

Одиннадцатый урок является продолжением темы о текстурировании. Будет изучено мультитекстурирование.

На *двенадцатом уроке* в приложение будет загружена модель, сделанная сторонним 3D-редактором и конвертированная в X-формат для загрузки.

Тринадцатый урок посвящен вводному курсу в вершинные шейдеры. Этот большой теоретический урок раскрывает общую концепцию работы вершинных шейдеров, затрагивая вершинные шейдеры первой версии.

Три последующие урока — *четырнадцатый*, *пятнадцатый* и *шестнадцатый* — рассказывают об устройствах ввода, представленных интерфейсом DirectInput. Рассматривается работа с клавиатурой и мышью.

Уроки семнадцатый и *восемнадцатый* посвящены звуку. Для этих целей в DirectX существуют интерфейсы DirectMusic и DirectSound.

И самый последний *урок* — *девятнадцатый* — это итоговый урок по работе с Direct3D, DirectInput, DirectMusic. Будет создан ряд отдельных классов, более реально отражающих технику создания и компоновки компьютерных игр.

В конце всей книги вас ждет приятный сюрприз, материализованный в виде *приложения 1* — справочника по основным компонентам DirectX 9 SDK, где описаны наиболее часто используемые интерфейсы, структуры, функции, макросы.

В *приложении 2* приведены интересные интернет-ссылки. В содержании компакт-диска поможет разобраться *приложение 3*. А список использованных источников информации можно увидеть в *приложении 4*.

Что вы должны знать

Книга рассчитана на начинающего программиста, но предполагается, что вы уже знаете в минимальном объеме язык программирования C++ и можете создать проект в среде Visual C++ .NET, написать несколько строк кода, откомпилировать его и запустить полученный exe-файл. Все уроки и примеры, приведенные в данной книге, описываются в очень доступной форме. Однако, эта книга не о языке программирования C++ или о среде Visual C++ .NET.

Какое программное обеспечение мы будем использовать

При написании этой книги использовались: Windows XP, Visual C++ .NET, DirectX 9 SDK. Весь написанный код должен компилироваться и в более ранних версиях Visual C++, но будем исходить из того, что раз все вышеперечисленные программные продукты уже доступны в последних версиях, то соответственно они и использовались. Что касается других языков и библиотек программирования, то не вступая в полемику с людьми, пользующимися чем-то другим, хочу сказать только одно — 90% игр пишутся на C++ и DirectX. Если хотите заработать на своих знаниях — учите то, на что сейчас делается ставка. Удачи вам!

Благодарности

Эту книгу я хочу посвятить любви всей моей жизни, которая постоянно меня подбадривала и оказывала большую моральную помощь в создании книги. Без моей жены этой книги не было бы вообще. Света, это все для тебя!

Конечно, хочу вспомнить своих родителей, без которых трудно себе представить появление этой книги и меня, как человека.

И еще хотелось бы поблагодарить издательство "БХВ-Петербург", обеспечившее самые благоприятные условия для написания этой книги, и лично отметить Анатолия Адаменко, Игоря Шишигина и Вадима Александровича Сергеева. Особая благодарность редактору книги Яковлевой Елене Сергеевне за большую проделанную работу.

Урок 1

Основы программирования под Windows

Прежде чем мы начнем программирование графики с помощью DirectX 9, нам необходимо создать каркас программы, являющийся своего рода рабочей поверхностью для трехмерного приложения. Ни для кого не секрет, что Windows многозадачная система, и в целом все то, что вы видите на экране монитора — это обыкновенное оконное приложение. Создав каркас программы, мы можем перейти к программированию графики, рисуя любые объекты в созданном приложении. Поэтому первоочередной нашей задачей будет написание кода окна, на основе которого и будут строиться все дальнейшие примеры к урокам. Что касается размеров оконного приложения, то они тоже играют роль. Имеются два режима — полноэкранный и оконный. В оконном режиме вы можете задавать размеры создаваемого окна, определять его стили и место вывода на экране. В полноэкранном режиме указываются разрешение и частота обновления экрана. На первых уроках мы будем использовать оконный режим, а в дальнейшем перейдем к полноэкранному, переход между двумя режимами сам по себе несложен.

Создаем проект в Visual C++ .NET

Весь этот урок будет посвящен созданию оконного приложения. Для этого в Windows, а точнее в среде программирования, которую мы используем, существуют стандартные средства. Начнем с того, что откроем среду программирования Visual C++ .NET и создадим проект. Я уверен, что вы знаете, как это делается, но на всякий случай повторим. Для создания проекта необходимо осуществить следующие действия.

1. Выполните команду **File | New | Project**. Появится диалоговое окно **New Project**.
2. В поле **Project Types** выберите папку **Visual C++ Projects**, а в поле **Templates — Win 32 Project**.

3. В поле **Name** укажите имя всего проекта. Чтобы не путаться, предлагаю назвать его `Urok1`. В поле **Locations** задайте директорию, в которой будут храниться ваши исходные коды, например `C:\Code`. Нажмите кнопку **OK**.

Появится новое диалоговое окно **Win 32 Application Wizard - Urok1** (рис. 1.1), в котором нужно перейти на вкладку **Application Settings** и установить в области **Additional options** флажок **Empty project**. В области **Application type** необходимо выбрать переключатель **Windows application**.

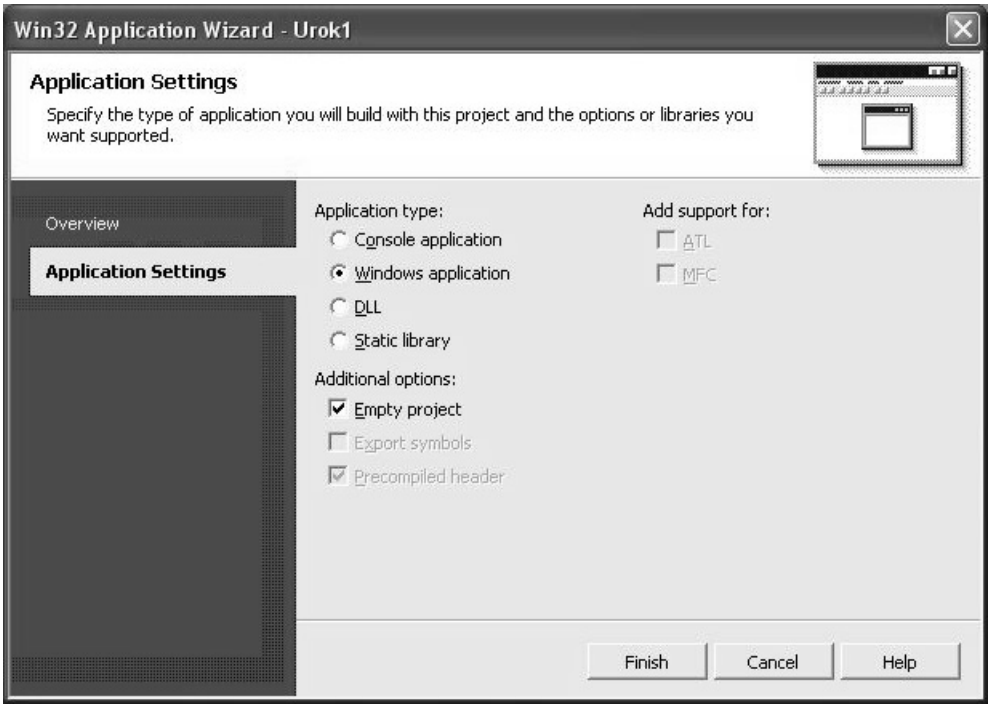

Рис. 1.1. Диалоговое окно **Win 32 Application Wizard - Urok1**

После чего нажмите кнопку **Finish**. С помощью этих операций мы создали обыкновенное пустое приложение, в котором нет ни строчки кода. С ним и будем работать. С левой стороны экрана в Visual C++. NET находится панель, содержащая три вкладки: **Solution Explorer**, **Resourse View** и **Class View**. Выбрав вкладку **Solution Explorer**, вы увидите дерево проекта Urok1. При нажатии на знак "+" рядом с названием проекта откроется ветвь, в которой будут находиться папки: **Source File** (файлы кода имеют, как правило, расширения с и cpp) и **Header File** (заголовочные файлы, обычно с расширением h). Добавим пустой файл в папку **Source File**. В нем мы будем писать код

нашей программы. Для добавления пустого файла в папку **Source File** необходимо выполнить следующие действия.

1. Нажмите правой кнопкой мыши на папке **Source File**, в появившемся меню выберите пункт **Add**. Перейдите по стрелке в выпадающее меню, нажмите на пункте **Add New Item**. На экране появится диалоговое окно **Add New Item - Urok1**, показанное на рис. 1.2.
2. В поле **Templates** укажите пункт **C++ File (.cpp)**. При этом будет создан пустой файл, в котором и пишется код программы.
3. В поле **Name** дадим название созданному файлу `WindowsBazis`. Нажмите кнопку **Open**. После чего у вас во вкладке **Solution Explorer - Urok1** в папке **Sourse File** появится пустой файл WindowsBazis.cpp.

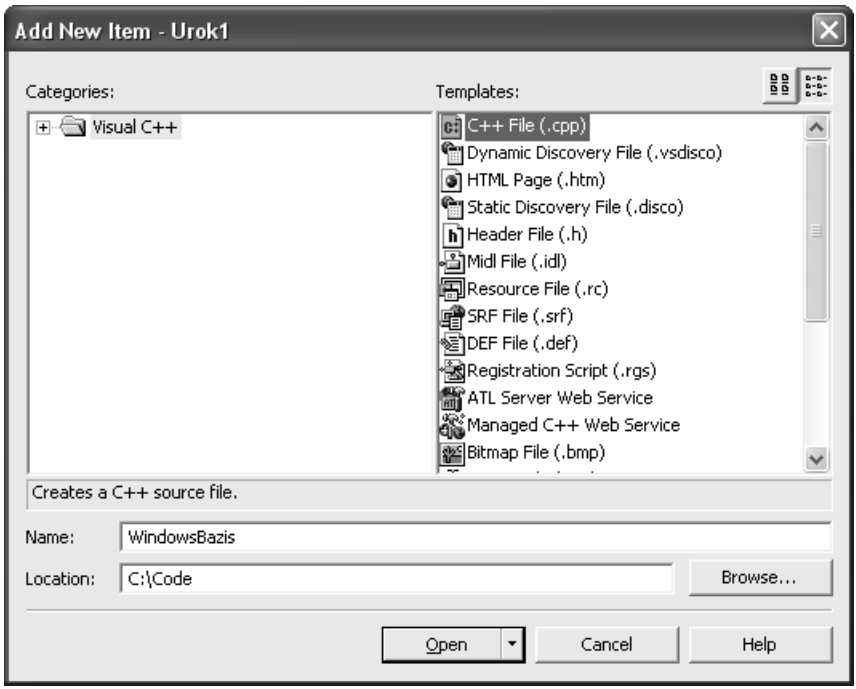

Рис. 1.2. Диалоговое окно **Add New Item - Urok1**

Общая модель сообщений в Windows

Windows очень интересная операционная система и все общение с приложениями строится на модели событий. Любое оконное приложение в момент своего существования общается с Windows посредством сообщений.

Таких сообщений может быть сколько угодно. Чтобы систематизировать их, существует т. н. очередь событий, реализованная в виде обыкновенного цикла. Каждое из поступивших сообщений ждет своей очереди для обработки, после чего изымаются операционной системой Windows. А поскольку Windows очень большая и многозадачная операционная система, то приложений в какой-то определенный промежуток времени может существовать много, а значит, и поступаемых сообщений тоже. Чтобы обработать все сообщения в Windows имеется обработчик событий для всех приложений. Схема взаимодействия приложений с сообщениями представлена на рис. 1.3.

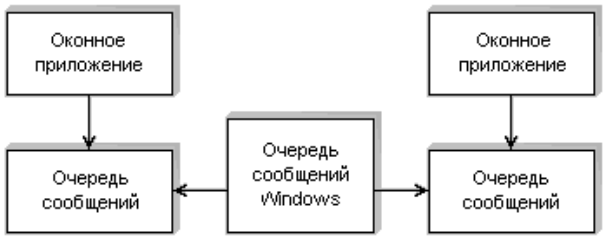

Рис. 1.3. Модель событий Windows

Исходя из модели событий, предлагаемой Windows, получается, что нам необходимо создать окно и запустить обработчик сообщений. Отправной точкой всех приложений Windows является функция `WinMain()`, в которой происходит создание окна и обработка событий. Поэтому вначале мы создадим функцию `WinMain()`, в которой опишем класс `windowsclass`, в котором укажем размеры, стиль, цвет, фон предполагаемого окна. Потом мы создадим окно при помощи функции `CreateWindowEx()`. Далее зарегистрируем окно (печать в налоговой о том, что мы есть и готовы платить налоги), и в конце создадим обработчик событий, куда будем посылать всякие нехорошие письма.

Если в начале этого урока вы не создали проект для работы, сделайте это, и приступим к программированию в файле WindowsBazis.cpp.

Главная функция *WinMain*

Первое, что надо сделать, — это включить в код главный заголовочный файл windows.h, содержащий объявление классов, необходимых для программирования в среде Windows:

```
#include <windows.h>
```

Далее идет главная входная функция всех приложений Windows: `WinMain()`, прототип которой выглядит следующим образом:

```
int WINAPI WinMain(
   HINSTANCE     hinstance,
   HINSTANCE     hprevinstanse,
   LPSTR         LpCmdLine,
   int           nCmdShow)
```

Функция `WinMain()` имеет следующие параметры:

- `hinstance` — это дескриптор создаваемого приложения, т. н. "хендл" (handle). Генерируется операционной системой Windows для дальнейшего отслеживания его использования. Можно сказать, что это своего рода метка создаваемого вами окна;
- `hprevinstance` — этот параметр уже не используется. Он был актуален при использовании Windows 3.1, оставлен просто для совместимости и всегда равен 0;
- `lpCmdLine` — указатель на командную строку, которая идет сразу за именем запускаемой команды;
- `nCmdShow` — это значение указывает, как создаваемое окно будет отображаться на экране монитора при создании. Данный параметр имеет большое количество всевозможных флагов, определяющих вид создаваемого окна. Не вижу большого смысла в разборе этих флагов, нам важнее программирование графики с помощью DirectX 9, нежели системное программирование. Но если вам это все же интересно, то можно найти нужную информацию в справочном материале Win32 API.

Разбираем класс *windowsclass*

Далее нам нужно создать класс `windowsclass`. *Класс* — это то, что позволяет различать вид и назначения окон. Обычно говорят, что окна отличаются друг от друга классами. Windows уже имеет большое количество заранее предопределенных классов, и все, что надо сделать, это сохранить информацию о создаваемом классе `windowsclass` в структуре `WINDCLASSEX`. Давайте создадим класс `windowsclass`, отвечающий нашим запросам:

```
WINDCLASSEX windowsclass;    //  наш класс
```

А теперь рассмотрим прототип структуры `WINDCLASSEX`, необходимый для создания класса:

```
typedef struct_WINDCLASSEX {
   UINT          cbSize;
   UINT          style;
```

```
    WINDPOC       LpfnWndProc;
    int           cbClsExtra;
    int           cbWndExtra;
    HANDLE        hInstance;
    HICON         hIcon;
    HCURSOR       hCursor;
    HBRUSH        hbrBackgroung;
    LPCTSTR       LpszMenuName;
    LPCTSTR       LpszClassName;
    HICON         hIconSm;
} WINDCLASSEX;
```

Структура имеет следующие параметры:

- `cbSize` — это размер создаваемой структуры. Не будем терять время и сразу при рассмотрении параметров будем заполнять поля структуры `WINDCLASSEX`: `windowsclass.cbSize = sizeof(WINDCLASSEX)`. Здесь мы указываем размер создаваемой структуры;
- `style` — содержит множество различных флагов стилей окна, которые можно комбинировать с помощью операции побитового ИЛИ "|". Заполняем поле `style`:

  ```
  windowsclass.style = CS_VREDRAW|CS_HREDRAW|CS_OWNDC|CS_DBLCLKS,
  ```

  комбинируя различные флаги. Рассмотрим использующиеся флаги. Если у вас появится желание узнать о них больше, то вы можете обратиться к справочной системе Win 32:

  - `CS_VREDRAW` — если размер высоты окна меняется или окно было перемещено, то требуется перерисовка всего окна;
  - `CS_HREDRAW` — если ширина окна меняется или окно было перемещено, то требуется перерисовка всего окна;
  - `CS_OWNDC` — для каждого окна данного класса дается свой контекст устройств;
  - `CS_DBLCLKS` — при совершенном в окне двойном щелчке мышью окну посылается информация о двойном щелчке;
- `LpfnWndProc` — это указатель на функцию обратного вызова. В любом приложении Windows существует цикл обработки событий, при работе которого и происходит вызов `MainWinProc`, поэтому необходимо записать следующее: `windowsclass.LpfnWndProc = MainWinProc`;
- два поля, `cbCLsExtra` и `cbWndExtra`, предназначены для хранения дополнительной информации. Однако ими редко кто пользуется, по крайней мере, нам в программировании DirectX они не понадобятся. Поэтому напишем: `windowsclass.cbCLsExtra = 0` и `windowsclass.cbWndExtra = 0`;

- `hInstance` — параметр, отвечающий за экземпляр создаваемого приложения, передаваемый функцией `WinMain()`: `windowsclass.hInctance = hinstance`;
- `hIcon` — служит для определения пиктограммы вашего приложения. С помощью функции `LoadIcon()` можно загрузить собственную или системную пиктограмму. Прототип функции `LoadIcon()` выглядит следующим образом: `HICON LoadIcon(HINSTANCE hInstance, LPCTSTR LpIconName)`. Параметры прототипа такие:
  - `hInstance` — экземпляр приложения. Чтобы загрузить стандартную пиктограмму, используется значение `NULL`;
  - `LpIconName` — это идентификатор загружаемого ресурса пиктограммы. Здесь мы используем `IDI_APPLICATION` — пиктограмму по умолчанию. Вообще существует много всевозможных идентификаторов для загрузки разных пиктограмм, но сейчас задействуем этот. В дальнейшем, при программировании своей игры вы захотите создать свою пиктограмму для замены системной. На основании прототипа функции `LoadIcon()`, можно записать: `windowsclass.hIcon = LoadIcon(NULL, IDI_APPLICATION)`;
- `hCursor` — этот параметр отвечает за курсор и, по сути, практически идентичен предыдущему параметру `hIcon`, за исключением названия функции, используемой для загрузки курсора. Записывается следующим образом: `windowsclass.hCursor = LoadCursor(NULL, IDC_ARROW)`. Стандартный идентификатор стрелки `IDC_ARROW`. Также, как и в случае с пиктограммами, существует более десятка различных идентификаторов для курсора, но для игр обычно рисуются свои;
- `hbrBacKground` — это поле структуры, отвечающей за цвет фона окна. Чтобы закрасить фон определенным цветом, нужно использовать функцию `GetStockObject()`, принимающую один параметр в виде флага, определяющего цвет кисти. Для закраски фона серым цветом запишем следующее:

  ```
  indowsclass.hbrBackground = (HBRUSH)GetStockObject(GRAY_BRUSH).
  ```

  Впоследствии, когда мы завершим написание кода окна, можно попробовать закрасить фон кистями других цветов, но в DirectX это не будет иметь никакого значения, т. к. для этих целей определены свои средства. Ниже приведены некоторые кисти различных цветов:
  - `GRAY_BRUSH` — это серый цвет, которым мы закрашиваем фон окна (правильнее называть — серая кисть);
  - `BLACK_BRUSH` — черная кисть;
  - `WHITE_BRUSH` — белая кисть;

- LTGRAY_BRUSH — светло-серая кисть;
- DKGRAY_BRUSH — темно-серая кисть;
- HOLLOW_BRUSH — пустая кисть;

- LpszMenuName — поле, предназначенное для подключения стандартного меню к окну. У нас стоит значение NULL, потому что мы не будем использовать меню в наших проектах: windowsclass.LpszMenuName = NULL;
- LpszClassName — задается название вашего класса окна для создаваемого приложения. Могут существовать (что и происходит) сразу несколько приложений, при создании которых использовался класс windowsclass, а ваша операционная система должна их каким-то образом различать, именно для этого и служит это поле. Здесь вы можете дать любое название, главное потом в них не запутаться: windowsclass.LpszClassName = "WINDOWSCLASS";
- hIconSm — это дескриптор малой пиктограммы, которая выводится на панель задач Windows в полосе заголовка вашего окна. Воспользуемся стандартной пиктограммой. Для этого необходимо записать следующее: windowsclass.hIconSm = LoadIcon(NULL, IDI_APPLICATION).

Закончив заполнение всех полей структуры, посмотрим, что получилось:

```
WINDCLASSEX windowsclass;
windowsclass.cbSize = sizeof(WINDCLASSEX);
windowsclass.style = CS_VREDRAW|CS_HREDRAW|CS_OWNDC|CS_DBLCKLICKS;
windowsclass.LpfnWndProc = WindowProc;
windowsclass.cbExtra = 0;
windowsclass.cbWndExtra = 0;
windowsclass.hInstance = hinstanse;
windowsclass.LoadIcon(NULL,IDI_APPLICATION);
windowsclass.LoadCursor(NULL,IDI_ARROW);
windowsclass.hbrBackground = (HBRUSH)GetStockObject(GRAY_BRUSH);
windowsclass.LpszMenuName = NULL;
windowsclass.LpszClassName = "WINDOWSCLASS";
windowsclass.hIconSm = LoadIcon(NULL,IDI_APPLICATION);
```

После того как вы заполнили и сохранили в переменной windowsclass все значения, класс windowsclass нужно зарегистрировать. Делается это с помощью функции RegisterClassEx(). Принимаемый параметр и есть регистрируемый класс. Класс для каждого создаваемого приложения может быть зарегистрирован только один раз:

```
RegisterClassEx(& windowsclass);    //  регистрируем созданный класс
```

Создаем окно

После того как класс зарегистрирован, можно создавать окно. Это делается с помощью функции `CreatWindowEx()`. Общий вид этой функции следующий:

```
HWND CreateWindowEx(
   DWORD      dwExStyle,
   LPCTSTR    LpClassName,
   LPCTSTR    LpWindowName,
   DWORD      dwStyle,
   int        X,
   int        Y,
   int        nWidth,
   int        nHeight,
   HWND       hWindParent,
   HWND       hMenu,
   HINSTANCE  hInstance,
   LPVOID     LpParam);
```

Параметры функции `CreatWindowEx()` такие:

- `dwExStyle` — флаг стилей окна. Используется редко, в основном с флагом `WS_EX_TOPMOST`, для того, чтобы ваше окно появлялось поверх других окон. Либо просто значение `NULL`, чтобы игнорировать данный параметр;
- `LpClassName` — это имя класса создаваемого окна. В нашем случае `WINDOWSCLASS`;
- `LpWindowName` — это заголовок окна. Здесь можно написать все, что угодно, будем использовать название нашей цели, а именно "Базовое Окно для DirectX". Это название будет меняться каждый раз по смыслу последующего урока;
- `dwStyle` — это флаг, описывающий стиль и поведение создаваемого окна. Имеется небольшое количество разных флагов, вот некоторые из них:
  - `WS_OVERLAPPED` — окно с полосой заголовка и рамкой;
  - `WS_VISIBLE` — изначально видимое окно;
  - `WS_CAPTION` — окно с полосой заголовка (включает в себя стиль `WS_BORDER`);
  - `WS_BORDER` — окно в тонкой рамке;
  - `WS_ICONIC` — окно изначально минимизировано;
  - `WS_OVERLAPPEDWINDOW` — перекрывающееся окно (включает в себя следующие шесть стилей: `WS_OVERLAPPED`, `WS_CAPTION`, `WS_SYSMENU`,

WS_THICKFRAME, WS_MINIMIZEBOX и WS_MAXIMIZEBOX — довольно функциональный флаг, его мы и будем использовать;

- WS_MINIMIZE — изначально минимизированное окно;
- WS_MAXIMIZE — изначально максимизированное окно;
- WS_MAXIMIZEBOX — окно с кнопкой Maximize (обязательно используется со стилем WS_SYSMENU);
- WS_MINIMIZEBOX — окно с кнопкой Minimize (обязательно используется со стилем WS_SYSMENU);

- X, Y — это позиция верхнего левого угла окна в пикселах. Можно указать координаты или воспользоваться флагом CW_USEDEFAULT, оставив значения по умолчанию, укажем X, равное 300, а Y, равное 150, установив тем самым координаты левого верхнего угла в пикселах;
- nWidth и nHeight — ширина и высота окна в пикселах, также можно использовать флаг CW_USEDEFAULT или указать необходимые размеры, например, 500 на 400 пикселов. Именно этот размер задан в листинге 1.1;
- hWndParent — дескриптор родительского окна. Если родительского окна нет, используется значение NULL;
- hMenu — дескриптор меню, т. е. если у вас есть меню, вы можете указать его дескриптор, и оно будет присоединено к вашему окну. Если меню нет, то устанавливается значение NULL;
- hInstance — экземпляр вашего приложения. Использует значение hinstance функции WinMain().

На данный момент нам известны все значения параметров функции. Создадим окно. Объявим дескриптор окна и вызовем функцию CreateWindowEx():

```
HWND hwnd;    //  дескриптор окна
if  (!(hwnd = CreateWindowEx (NULL, "Windowsclass",
     "Базовое Окно для DirectX", WS_OVERLAPPEDWINDOW|WS_VISIBLE,
     0, 0, 500, 400, NULL, NULL, hinstance, NULL)))
return (0);
```

Кстати, обратите внимание на конструкцию ifelse, с помощью которой происходит обработка ошибок.

После того как окно создано, его нужно явно вывести на экран и обновить. Делается это двумя функциями:

```
ShowWindow(hwnd, ncmdshow);    //  выводим окно
UpdateWindow(hwnd);            //  обновляем окно
```

Обрабатываем события в Windows

В Windows необходимые события обрабатываются с помощью созданного вами обработчика событий. На каждый класс можно определить свой обработчик событий, чем больше их будет, тем лучше функциональность вашей программы. В процессе работы любого приложения, в т. ч. и вашего, генерируется масса разнородных сообщений, которые передаются в т. н. очередь. События, происходящие с вашим окном, попадают в раздел очереди именно вашего окна. После чего главный обработчик событий изымает их и передает в функцию `MainWinProc()` для дальнейшей обработки. Событиями, которым не назначен обработчик, будет заниматься сама система Windows.

Разберем прототип функции `MainWinProc()`:

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

Параметры функции `MainWinProc()` следующие:

- `hwnd` — дескриптор окна. Требуется для определения, от какого окна получено сообщение, если сразу открыто несколько окон, принадлежащих одному и тому же классу;
- `msg` — идентификатор события, которое должно быть передано в функцию `MainWinProc()` для обработки;
- `wparam` и `lparam` — являются дополнительными параметрами обрабатываемого события. Для написания обработчика событий обычно используется конструкция `switch(msg)`, в ней, на основе идентификатора сообщений `msg`, используются дополнительные параметры `wparam` и `lparam`. Рассмотрим некоторые сообщения для данных параметров, которые могут пригодиться для разработки игры. Но если вы пишете приложение Win32 без использования DirectX, тогда нужно гораздо больше информации:
  - `WM_PAINT` — данное сообщение посылается при необходимости перерисовки всего окна, которое могло быть перемещено, увеличено в размере и т. д.;
  - `WM_DESTROY` — это сообщение посылается Windows, когда окно должно быть закрыто. Можно сказать, что это такой идентификатор, который сообщает о том, что нужно освободить все захваченные приложением ресурсы;
  - `WM_QUIT` — после освобождения ресурсов вызывается данное сообщение, которое и закрывает приложение.

Объединив все вместе, мы получим функцию WinProc:

```
LRESULT CALLBACK MainWinProc(HWND hwnd, UINT msg,
                             WPARAM wparam, LPARAM lparam)
   {
     switch(msg)
     case WM_PAINT:
     break;
     case WM_DESTROY:
     {
        PostQuitMessage(0);
        return 0;
     }  break;
     return (DefWindowProc(hwnd, msg, wparam, lparam));
   }
```

Функция PostQuitMessage(0) ставит в очередь сообщение WM_OUIT, которое и закроет ваше приложение. WM_OUIT применяется непосредственно в главном обработчике событий. Функция DefWindowProc() обрабатывает по умолчанию те сообщения, которые вы не используете.

Главный обработчик событий

Рассмотрим главный обработчик событий, применяемый в следующем примере:

```
while (GetMessage(&msg, NULL, 0, 0))
{
   TranslateMessage(&msg);  //  преобразование клавиатурного ввода
   DispatchMassage(&msg);   //  обработка и пересылка сообщений в WinProc
}
```

Прототип функции GetMessage() выглядит следующим образом:

```
BOOL    GetMessage(
LPMSG   LpMsg,
HWND    hWnd,
UINT    wMsgFiltrenMin,
UINT    wMsgFiltrenMax);
```

Параметры функции GetMessage() такие:

- LpMsg — указатель на структуру, отвечающую за сообщения;
- hWnd — дескриптор создаваемого окна;

- `wMsgFiltrenMin` — это первое сообщение, пришедшее на обработку;
- `wMsgFiltrenMax` — это последнее обрабатываемое сообщение.

Цикл `while` выполняется до тех пор, пока функция `GetMessage()` возвращает отличное от нуля значение. Работа самого цикла выглядит довольно просто: при поступлении сообщения из очереди оно обрабатывается функцией `TranslateMessage()` и передается функции `DispatchMassage()`, которая вызывает для обработки `MainWinProc()`, передавая ей всю необходимую информацию.

Все что получилось в результате, приведено в листинге 1.1.

Листинг 1.1. Файл WindowsBazis.cpp

```
//-------------------------------------------------------------------------
//   WindowsBazis.cpp
//   данное приложение создает простейшее окно в Windows
//-------------------------------------------------------------------------
#include <windows.h>    //  подключаем заголовочный файл Windows
//-------------------------------------------------------------------------
//   функция
//   MainWinProc()
//   здесь происходит обработка сообщений
//-------------------------------------------------------------------------

LRESULT CALLBACK MainWinProc(HWND   hwnd,   //  дескриптор окна
                             UINT   msg,    //  идентификатор сообщения
                             WPARAM wparam, //  дополнительная информация
                             LPARAM lparam) //  дополнительная информация
{
   switch(msg)
   {
     case WM_PAINT:
     break;
     case WM_DESTROY:
     {
        PostQuitMessage(0);
        return(0);
     }  break;
   }
```

```
return (DefWindowProc(hwnd, msg, wparam, lparam));
}
//-------------------------------------------------------------------------
//   функция
//   WinMain
//   входная точка приложения
//-------------------------------------------------------------------------
int WINAPI WinMain(HINSTANCE hinstance,
                   HINSTANCE hprevinstance,
                   LPSTR     lpcmdline,
                   int       ncmdshow)
{
   WINDCLASSEX windowsclass;   //  создаем класс
   HWND        hwnd;           //  создаем дескриптор окна
   MSG         msg;            //  идентификатор сообщения
   //  определим класс окна WINDCLASSEX
   windowsclass.cbSize = sizeof(WINDCLASSEX);
   windowsclass.style = CS_DBLCLKS|CS_OWNDC|CS_HREDRAW|CS_VREDRAW;
   windowsclass.lpfnWndProc = MainWinProc;
   windowsclass.cbClsExtra = 0;
   windowsclass.cbWndExtra = 0;
   windowsclass.hInstance = hinstance;
   windowsclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
   windowsclass.hCursor = LoadCursor(NULL, IDC_ARROW);
   windowsclass.hbrBackground = (HBRUSH)GetStockObject(GRAY_BRUSH);
   windowsclass.lpszMenuName = NULL;
   windowsclass.lpszClassName= "WINDOWSCLASS";
   windowsclass.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
   //  зарегистрируем класс
   if(!RegisterClassEx(&windowsclass))
   return(0);
   //  можно создать окно
   if(!(hwnd = CreateWindowEx(NULL,     //  стиль окна
       "WINDOWSCLASS",                  //  класс
       "Базовое Окно для DirectX",      //  название окна
       WS_OVERLAPPEDWINDOW|WS_VISIBLE,
       0,0,                             //  левый верхний угол
       500,400,                         //  ширина и высота
```

```
        NULL,                          //  дескриптор родительского окна
        NULL,                          //  дескриптор меню
        hinstance,                     //  дескриптор приложения
        NULL)))                        //  указатель на данные окна
   return(0);
   ShowWindow(hwnd, SW_SHOWDEFAULT);   //  нарисуем окно
   UpdateWindow(hwnd);                 //  обновим окно
   while(GetMessage(&msg, NULL, 0, 0))
   {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
   return(msg.wParam);
}
```

Окончательный вариант листинга 1.1 также можно найти на прилагаемом компакт-диске в папке Code\Urok1\WindowsBazis.cpp.

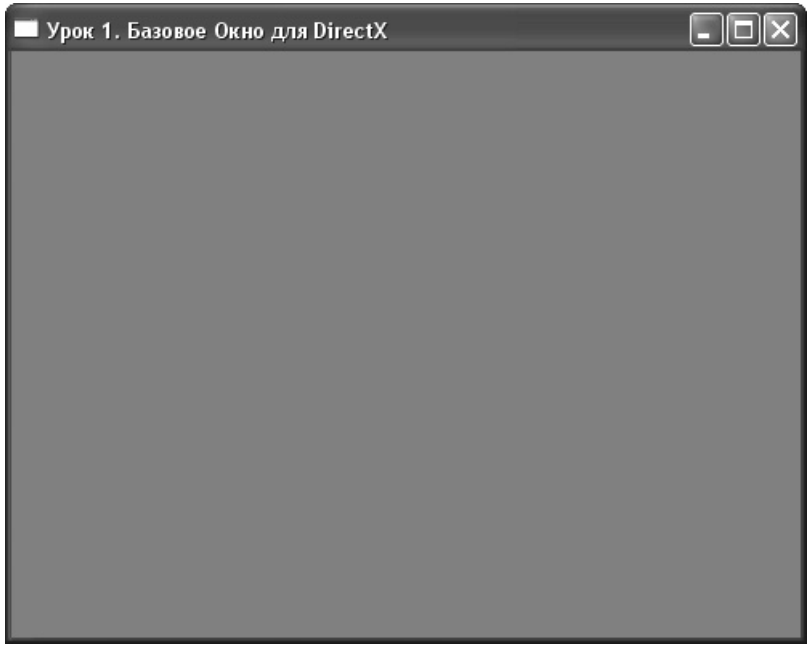

Рис. 1.4. Базовое окно для DirectX