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

+ CD

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	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Юрий Рожко*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Инны Тачиной*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.07.06.
Формат $70\times100^{1}/_{16}$. 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 5-94157-832-6

Оглавление

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 Demo-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 Demo-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.

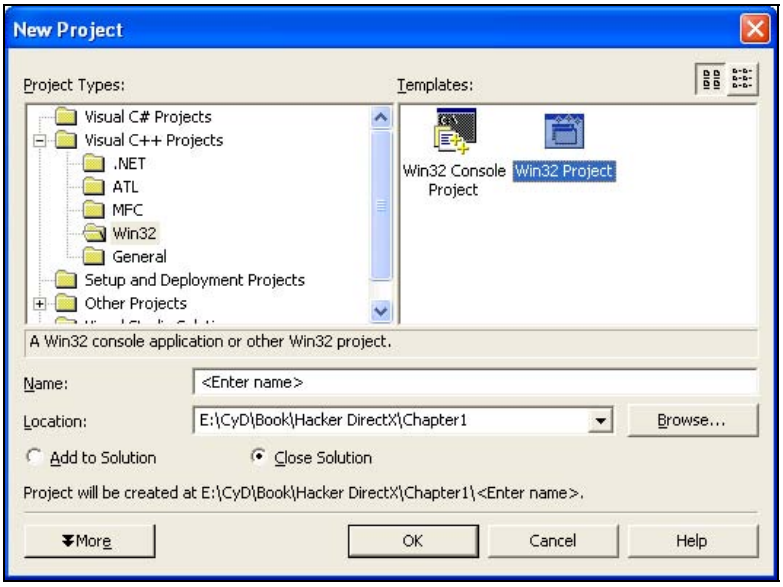

Рис. 1.1. Окно создания нового проекта

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.

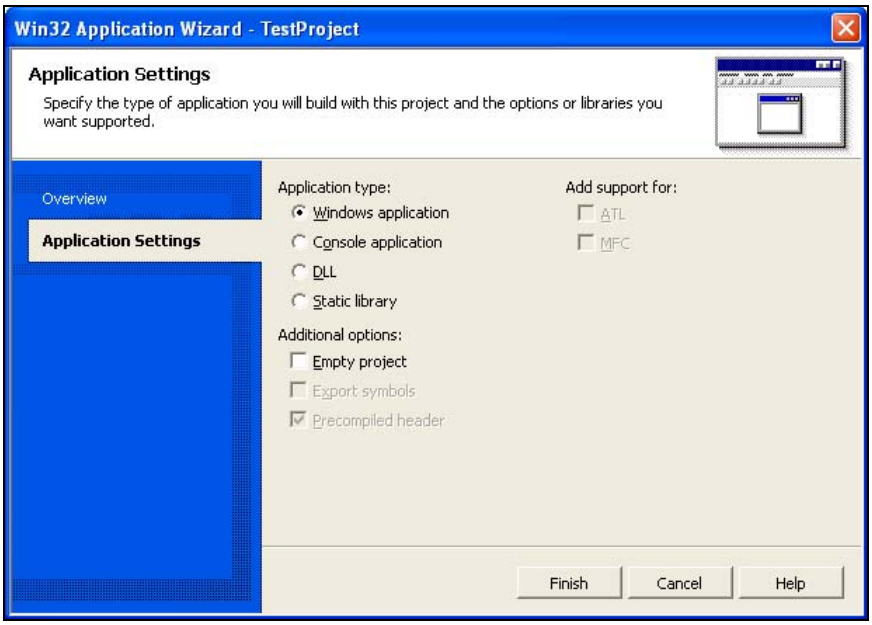

Рис. 1.2. Мастер создания приложения

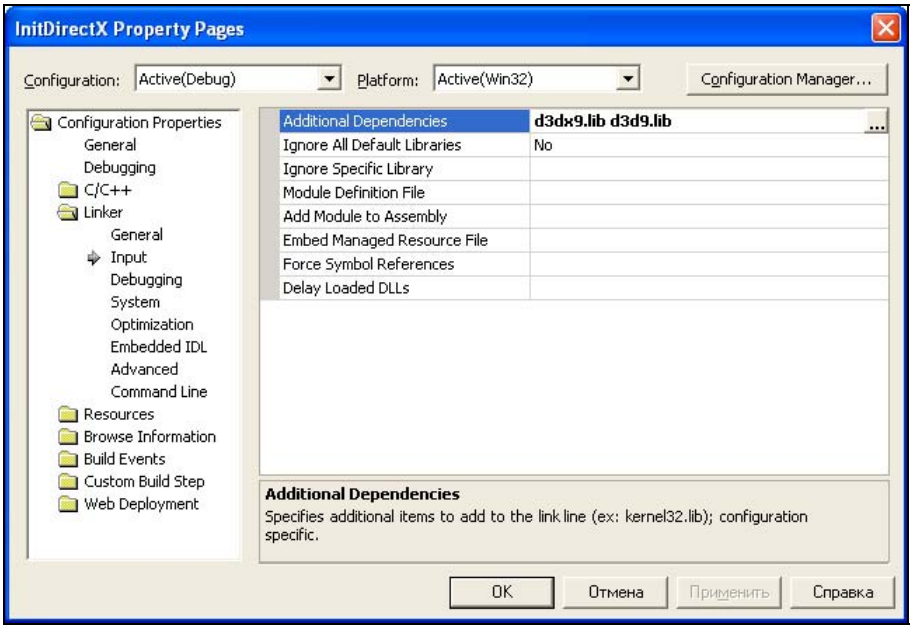

Рис. 1.3. Окно настройки свойств проекта в Visual Studio .NET

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).

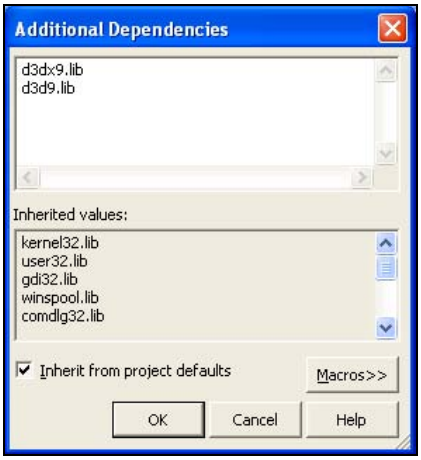

Рис. 1.4. Окно добавления библиотек

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 wcex;
 MSG        msg;
 HWND       hWnd;

 CoInitialize(NULL);

 // Регистрируем класс окна
 wcex.cbSize = sizeof(wcex);
 wcex.style = CS_CLASSDC;
 wcex.lpfnWndProc = (WNDPROC)WndProc;
 wcex.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 просмотрим 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
```