СЕРГЕЙ ЕСЕНИН

bhv®

DirectX и Delphi:

РАЗРАБОТКА ГРАФИЧЕСКИХ И МУЛЬТИМЕДИЙНЫХ ПРИЛОЖЕНИЙ

КОМПОНЕНТЫ DirectX: DirectX Graphics, DirectSound, DirectMusic, DirectInput, DirectShow

ШЕЙДЕРЫ И ЯЗЫК HLSL

ДВУМЕРНАЯ И ТРЕХМЕРНАЯ ГРАФИКА

ЗАХВАТ И ВОСПРОИЗВЕДЕНИЕ ИЗОБРАЖЕНИЯ И ЗВУКА

РАБОТА С УСТРОЙСТВАМИ ВВОДА

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Сергей Есенин

DirectX и Delphi:

РАЗРАБОТКА ГРАФИЧЕСКИХ И МУЛЬТИМЕДИЙНЫХ ПРИЛОЖЕНИЙ

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.06
ББК 32.973.26-018.2
Е82

Есенин С. А.

Е82 DirectX и Delphi: разработка графических и мультимедийных приложений. — СПб.: БХВ-Петербург, 2006. — 512 с.: ил.

ISBN 5-94157-867-9

Рассмотрена разработка приложений с использованием технологии DirectX в среде программирования Borland Delphi. Подробно описаны все основные компоненты, входящие в состав DirectX: DirectX Graphics, DirectShow, DirectInput, DirectSound и DirectMusic. Показано создание собственных наборов классов, облегчающих работу с различными компонентами DirectX. На практических примерах рассмотрена работа с двумерной и трехмерной графикой, шейдеры и язык HLSL, различные цветовые эффекты, работа с текстурой, освещением и т. д. Уделено внимание выводу изображения в оверлейном режиме, механизмам захвата изображения (на примере работы с web-камерой) и захвата звука. Представлены механизмы воспроизведения мультимедиаданных в различных форматах: AVI, MPEG, MP3 и др. Прилагаемый компакт-диск содержит исходные коды примеров, рассмотренных в книге, а также набор классов.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

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

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.05.06.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 41,28.
Тираж 2500 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 5-94157-867-9

Оглавление

Благодарности **1**

Введение **2**

На кого рассчитана книга 2
Как построена книга 3
Требования к компьютеру и программному обеспечению 5

Часть I. Общие сведения **7**

Глава 1. DirectX **9**

Состав DirectX 9
DirectX или OpenGL? 10

Глава 2. Среда разработки Borland Delphi **11**

Структура среды разработки 11
Первое приложение 12

Глава 3. Библиотека COM **14**

COM или DLL? 14
Объекты и интерфейсы 15
Интерфейс *IUnknown* 15
Уникальные идентификаторы 16
Результат *HResult* 17
Инициализация библиотеки COM и завершение работы с ней 19

Глава 4. Пишем COM-сервер **20**

Создание COM-сервера 20
Регистрация COM-сервера 28
Клиентская часть 29

ИТОГИ **33**

Часть II. DirectX Graphics 35

Глава 5. Direct3D 37

Инициализация 37
Очистка устройства 41
Прорисовка сцены 42
Первый пример 43
Полноэкранный режим 49
Потеря устройства 52
Примитивы 54
Буфер вершин 54
Точка 56
Цвет 61
Пример анимации 62
Линии и последовательность линий 65
Треугольник и последовательности треугольников 68
От треугольника к прямоугольнику 74
Построения в пространстве 74
Матрицы 75
Сложение матриц и умножение на число 75
Перемножение матриц 76
Единичная матрица 76
Матрицы переноса (сдвига) 76
Матрицы вращения 77
Матрица масштабирования 78
Матрицы отражения 78
Типы матриц Direct3D 79
Функции Direct3D для работы с матрицами 81
Нормали 84
От теории к практике 85
Куб 89
Буфер глубины 96
Правила построения объектов 97
Источники света 103
Материал 107
Работа с материалом и источниками света 107
Туман 113
Работа с текстурой 118
Фильтрация текстур 122
Mesh-объекты 126
Несколько объектов одновременно 127
Работа с текстом на плоскости и в пространстве 135

Подсчет числа кадров в секунду ... 142
Несколько текстур на одном объекте ... 145
Сферические текстурные координаты ... 147
Создаем туннель ... 150
Прозрачность ... 151
Мультитекстурирование ... 155
Motion Blur ... 157
Шейдеры ... 162
Основные сведения ... 162
Введение в HLSL ... 164
Вершинные шейдеры ... 170
Пиксельные шейдеры ... 179
Работа с текстурой ... 181

Глава 6. DirectDraw ... 184

Обзор библиотеки ... 184
Инициализация ... 185
Первый пример ... 185
Уровни взаимодействия ... 189
Полноэкранный режим работы ... 191
Поверхности ... 192
Рисуем на поверхности ... 194
Блиттинг ... 196
Переключение страниц ... 197
Потеря доступа к поверхности ... 198
Цветовые ключи ... 203
Палитра ... 205
Прямой доступ к поверхности ... 205
Огонь ... 206
Оконный режим работы ... 213
Оверлеи DirectX ... 218

ИТОГИ ... 235

Часть III. DirectSound ... 237

Глава 7. Общие сведения ... 239

Область применения ... 239
Достоинства ... 239
Недостатки ... 239
Принцип работы ... 240
Уровни взаимодействия ... 240

Глава 8. Вывод звука 242

Интерфейсы 242
Создание буферов 253
Потеря буферов 253
Звуковые эффекты 254
Классы *TdxSound* и *TdxSoundManager* 256
Пример использования классов *TdxSound* и *TdxSoundManager* 282

Глава 9. Захват звука 310

Интерфейсы 310
Буфер захвата 311
Захват аудио 313
Класс *TdxSoundCapture* 315
Пример использования класса *TdxSoundCapture* 321

ИТОГИ 326

Часть IV. DirectMusic 327

Глава 10. Работа с MIDI и WAV-файлами 329

Интерфейсы 329
Порядок работы 334
Классы *TdxMusicSegment* и *TdxMusicManager* 336
Пример использования классов 346

ИТОГИ 351

Часть V. DirectInput 353

Глава 11. Общие сведения 355

Режимы работы 355
Уровни взаимодействия 356

Глава 12. Работа с устройствами ввода 357

Интерфейсы 357
Общий алгоритм работы 361
Клавиатура 362
Мышь 364
Джойстик 366
Класс *TdxInputManager* 368
Пример использования класса *TdxInputManager* 377

ИТОГИ 385

Часть VI. DirectShow 387

Глава 13. Основные сведения 389

Область применения 389
Поддерживаемые форматы 389
Фильтры и граф фильтров 390
Типы фильтров 392
Менеджер графа фильтров 393

Глава 14. Работа с MP3, AVI, MPEG и другими мультимедиаформатами 394

Интерфейсы 394
Интерфейс управления фильтром 395
Интерфейс управления контактом 395
Интерфейс построения графа фильтров 396
Интерфейс управления графом фильтров 397
Интерфейс управления позиционированием в потоке 397
Интерфейс управления выводом звука 400
Интерфейс управления механизмом событий 401
Интерфейс управления выводом видеоданных 403
Интерфейс перехвата кадра из потока видео 405
Алгоритм работы 408
Класс *TdxMediaPlayer* 410
Пример работы с классом *TdxMediaPlayer* 428

Глава 15. Захват аудио и видео 437

Захват видео 437
Захват звука 437
Интерфейсы 438
Перечисление устройств определенного класса 443
Режимы захвата и предварительного просмотра 445
Запись видео со звуком 447
Сжатие потоков аудио и видео 449
Страницы свойств 451
Алгоритм работы 455
Класс *TdxCaptureManager* 457
Пример использования класса *TdxCaptureManager* 477

ИТОГИ 486

Заключение 487

Приложения 489

Приложение 1. Интернет-ресурсы 491

Приложение 2. Описание содержимого компакт-диска 493

Список литературы 494

Предметный указатель 495

Благодарности

Данная книга вышла в свет благодаря моральной поддержке и помощи со стороны моей жены Ольги и родителей, к которым я и приехал от суеты городской в деревню, дабы в спокойной обстановке закончить то, что начал.

Также благодарю за помощь своего товарища по работе, Хворова Василия. Его советы и замечания оказались очень ценными.

Конечно же, стоит отметить и роль сотрудника издательства "БХВ-Петербург" Шишигина Игоря, всеми правдами и неправдами заставившего меня проделать эту работу. За что ему отдельное спасибо!

Введение

Работая за компьютером с установленной операционной системой Microsoft Windows, мы, не задумываясь, можем запустить какую-либо трехмерную игру, в которой будем управлять, к примеру, самолетом при помощи джойстика или мыши и клавиатуры. Или решим послушать музыку, или просмотреть какой-то интересный фильм. Можем пообщаться по сети в режиме реального времени, да еще при этом получая изображения собеседника с Web-камеры или иного устройства захвата. И все это и даже больше нам помогает проделать система DirectX.

Свою книгу я решил посвятить описанию приемов разработки графических и мультимедиаприложений с использованием системы DirectX применительно к среде разработке Borland Delphi. Почему именно Delphi, а не Borland C++ Builder или, скажем, Microsoft Visual Studio? Да хотя бы потому, что книг по программированию с использованием DirectX в Delphi не так много. А отдельные компоненты DirectX, такие как DirectShow, вообще мало освещены.

В книге я постарался раскрыть такие аспекты разработки программного обеспечения с использованием DirectX, как работа с двумерной и трехмерной графикой, работа со звуком, устройствами ввода и мультимедиапотоками. Описание работы с графикой (подсистема DirectX Graphics) состоит из двух частей — описание подсистемы Direct3D и подсистемы DirectDraw. Несмотря на то, что подсистема DirectDraw считается несколько устаревшей, она все равно не утратила своей актуальности, и ее интерфейсы будут поддерживаться в DirectX и в дальнейшем. Работа со звуком будет изучена в главах, описывающих работу с DirectSound и DirectMusic; работа с устройствами ввода, такими как клавиатура, мышь и джойстик, будет рассмотрена в главе, посвященной DirectInput. А в последней части книги мы проанализируем работу с мультимедиапотоками: научимся воспроизводить такие мультимедиаформаты, как AVI, MPEG, MP3 и т. д. Научимся получать изображение и звук с устройств захвата и сохранять на диске.

На кого рассчитана книга

Книга в первую очередь рассчитана на людей, знакомых со средой разработки Delphi и имеющих представление о технологии COM, которые хотят изучить систему DirectX и такие ее возможности, как работа с графикой, устройствами ввода, работа с мультимедиаданными и т. д.

Предполагается, что читателю не нужно объяснять всех тонкостей работы в среде Delphi и всех тонкостей технологии COM. Тем не менее, в первой части книги среда разработки и возможности COM будут кратко описаны, и даже будет приведен пример разработки COM-сервера и клиентской части.

Как построена книга

Книга состоит из данного введения, шести частей, заключения, двух приложений, списка литературы и предметного указателя. В свою очередь шесть частей содержат пятнадцать глав.

Часть I книги является вводной. В ней представлены основные сведения о системе DirectX, расписаны компоненты, входящие в ее состав. Дается сравнение DirectX с OpenGL. Кратко описывается среда разработки Borland Delphi. Рассматриваются возможности библиотеки COM. Дается описание интерфейсов и COM-объектов, уникальных идентификаторов и результата вызова методов. В заключение приводится пример: пишем COM-сервер и клиентскую часть.

Часть II содержит описание графической подсистемы DirectX Graphics. Первая половина предлагает описание работы подсистемы Direct3D. Мы обсудим работу в оконном и полноэкранном режимах. Научимся рисовать различные примитивы, строить различные фигуры из примитивов на плоскости и в пространстве. Подробно изучим работу с матрицами и разберем, какие типы матриц используются в Direct3D, и для чего каждая из них предназначена. На практике научимся строить трехмерные объекты из примитивов на примере куба, изучим свойства освещения и материалов, научимся использовать туман и работать с текстурой. Изучим различные типы фильтрации текстур, такие как линейная фильтрация, анизотропная и многоуровневая фильтрации. Выясним возможности библиотеки утилит D3DX и рассмотрим mesh-объекты. Научимся работать с текстом на плоскости и в пространстве, накладывать на объект несколько текстур, а также изучим мультитекстурирование на примере. Обсудим возможность создания прозрачных объектов различными способами и научимся использовать эффект размытия при движении (Motion Blur). Разберем, что такое шейдеры, и научимся писать их на языке HLSL.

Вторая половина этой части вкратце описывает возможности и приемы работы с подсистемой DirectDraw. Будет проведен обзор библиотеки, ее возможностей, достоинств и недостатков. Мы изучим порядок работы с данной подсистемой, научимся работать с ней в полноэкранном и оконном режимах, рассмотрим различные типы поверхностей DirectDraw. Узнаем, что такое цветовой ключ и зачем он нужен, обсудим работу с палитрой, научимся

работать с поверхностью напрямую и разберемся, как работать с оверлейными поверхностями.

Основным отличием этой части от последующих является то, что изучение подсистем Direct3D и DirectDraw построено по принципу примеров, т. к. это наиболее простой и удобный способ изучения работы с графикой. В дальнейшем, в каждой части книги для описываемой подсистемы будут созданы собственные классы и примеры их использования.

Часть III описывает работу с подсистемой DirectSound. Мы рассмотрим область применения данной подсистемы, ее достоинства и недостатки, изучим принцип работы. Разберем, что такое уровни взаимодействия и потеря буферов. Научимся воспроизводить WAV-файлы и накладывать на звук различные эффекты. Будет представлен класс, упрощающий работу с DirectSound, под названием `TdxSoundManager`, и рассмотрен пример его использования. Затем мы изучим способы захвата звука и записи в WAV-файл. Для этого нами будет рассмотрен класс `TdxSoundCapture` вместе с примером.

Часть IV расскажет нам о подсистеме DirectMusic и ее отличиях от DirectSound. С помощью нее мы научимся воспроизводить MIDI- и WAV-файлы. Будут представлены классы `TdxMusicSegment` и `TdxMusicManager`, описывающие звуковой сегмент и менеджер воспроизведения соответственно.

Часть V описывает работу с устройствами ввода. Мы изучим режимы работы и уровни взаимодействия и разберем общий алгоритм работы. Рассмотрим класс `TdxInputManager`, упрощающий работу с клавиатурой, мышью и джойстиком, и пример его использования.

Часть VI содержит информацию о подсистеме DirectShow. Это архитектура, позволяющая управлять потоками мультимедиаданных. Сначала мы рассмотрим область применения данной архитектуры и поддерживаемые форматы потоков данных. Изучим такие понятия, как фильтр, граф фильтров и менеджер графа фильтров. Рассмотрим способы воспроизведения таких мультимедиаформатов, как AVI, MPEG, MP3 и др. Разберем работу интерфейсов управления фильтром, контактом, работу интерфейса графа фильтров и интерфейса управления графом фильтров. Рассмотрим интерфейсы управления позиционированием в потоке, управления выводом звука, управления механизмом событий, управления выводом видеоданных и интерфейс перехвата кадра из потока видео. Для работы с подсистемой DirectShow будет представлен класс `TdxMediaPlayer`, который фактически инкапсулирует набор свойств и методов, характерных для мультимедиапроигрывателя, а пример использования класса и будет тем самым проигрывателем.

Затем перейдем к изучению архитектуры захвата изображения и звука и разберем работу всех нужных нам интерфейсов. Научимся перечислять устройства определенных классов. Рассмотрим режимы предварительного просмотра и захвата потоков данных. Изучим возможность захвата изображения и звука одновременно, а также научимся сжимать полученные данные. Обсудим возможность настройки устройств с помощью страниц свойств и рассмотрим общую последовательность шагов, необходимых для получения данных с различных устройств захвата. Класс `TdxCaptureManager`, который будет рассмотрен нами в конце части, обеспечивает возможность захвата и предварительного просмотра видеопотока и потока аудио одновременно.

В *заключении* будет подведен краткий итог книги, а также представлена информация, как можно связаться с автором книги.

Приложения, которые представлены в конце книги, содержат информацию о наиболее интересных интернет-ресурсах, которыми, так или иначе, пользовался в свое время автор книги и пользуется сейчас, и описание прилагаемого к книге компакт-диска.

В книге также присутствует список литературы, который поможет читателю найти дополнительную информацию.

Ну и последнее — это предметный указатель. Он представляет собой наиболее удобный инструмент для поиска в книге по ключевым словам.

Требования к компьютеру и программному обеспечению

Для обеспечения корректной работы всех примеров, приведенных в книге, рекомендуется следующая конфигурация компьютера:

- процессор Intel Pentium III 1000 МГц и выше;
- видеокарта 32 Мбайт (1024×768) и более производительная, поддерживающая работу с DirectX 8.0 и выше;
- оперативная память 128 Мбайт и выше;
- жесткий диск объемом 10 Гбайт и более;
- CD/DVD-привод;
- операционная система Microsoft Windows 2000/XP/Server 2003;
- DirectX 9.0;
- установленная среда разработки Borland Delphi 7 и старше.

Не следует считать приведенную конфигурацию компьютера окончательной. Примеры будут работать и на компьютере с меньшей производительностью.

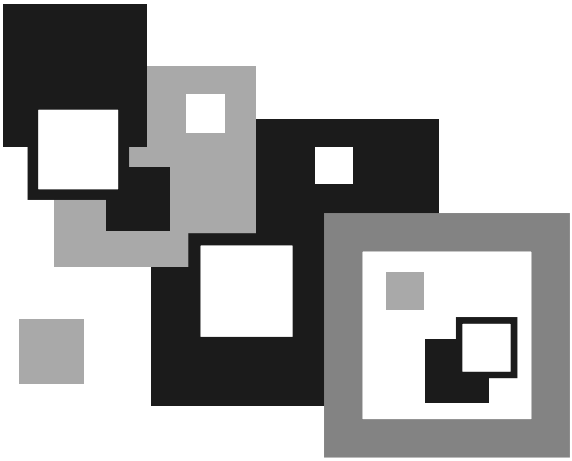
Однако требования к DirectX на компьютере не меняются — должна быть установлена версия не ниже версии DirectX 9.0.

На прилагаемом к книге компакт-диске *(см. приложение 2)* располагаются необходимые для сборки примеров заголовочные файлы DirectX, а также библиотеки, необходимые для работы с графической подсистемой Direct3D.

Все примеры были протестированы, как минимум, на трех компьютерах следующей конфигурации:

- Intel Pentium IV 2,4 ГГц\512 Мбайт DDR\128 NVidia FX 5200\120 Гбайт HDD Maxtor\DVD-RW Nec\Windows XP SP2\DirectX 9.0c;
- Intel Pentium IV 3,0 ГГц\1024 Мбайт DDR\128 ATI Radeon 9250\80 Гбайт HDD Seagate\DVD-Rom Toshiba\Windows XP SP2\DirectX 9.0c;
- ноутбук BLISS 507S: Intel Pentium M 1,73 ГГц\512 Мбайт DDR\128 ATI Radeon X700\60 Гбайт HDD\DVD-RW\Windows XP SP2\DirectX 9.0c.

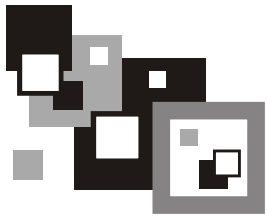
Для корректной работы примеров вы должны указать в настройках среды Borland Delphi путь к папке с заголовками DirectX, которые находятся в каталоге DirectX прилагаемого к книге компакт-диска. А в папке Lib хранятся библиотеки, необходимые для работы примеров с подсистемой Direct3D. Вам необходимо переписать библиотеки в такой каталог у себя на компьютере, к которому прописан путь в настройках Windows, например, Windows\System32.



ЧАСТЬ I

Общие сведения

Глава 1



DirectX

Система DirectX представляет собой большой набор API-функций низкого уровня, позволяющих разрабатывать различные высокопроизводительные графические и мультимедиаприложения. Имеются средства для работы со звуком, устройствами ввода и упрощена разработка сетевых приложений. Используется система зачастую в таких областях, как разработка компьютерных игр и систем безопасности.

Практически весь набор API-функций, так или иначе, базируется на технологии COM.

Состав DirectX

Свое знакомство с мощной системой, именуемой DirectX, мы начнем с изучения ее состава. Итак, вот основные компоненты (подсистемы), входящие в состав DirectX:

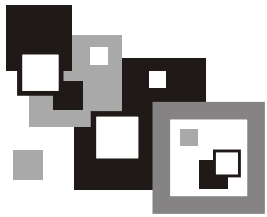
- DirectX Graphics — компонент, объединивший в себе две мощных графических подсистемы для работы с двумерной и трехмерной графикой — DirectDraw и Direct3D;
- DirectShow — архитектура, позволяющая управлять захватом и воспроизведением мультимедиапотоков;
- DirectInput — подсистема, используемая для работы с различными устройствами ввода, такими как клавиатура, мышь, джойстик, и другими игровыми устройствами (например, устройствами с обратной связью);
- DirectSound — компонент DirectX, обеспечивающий работу с оцифрованным звуковым потоком;

- DirectMusic — компонент DirectX, так же как и DirectSound обеспечивающий работу со звуковым потоком, только поддерживающий работу и с форматом MIDI;
- DirectPlay — подсистема, позволяющая разрабатывать многопользовательские приложения, ярким примером которых служат многопользовательские игры;
- DirectSetup — простой набор API-функций, позволяющий устанавливать компоненты DirectX одним вызовом;
- DirectX Media Objects (DMO) — базирующиеся на технологии COM компоненты поддержки потоковых объектов.

DirectX или OpenGL?

Как известно, помимо технологии Direct3D, для вывода трехмерной графики существует еще целый ряд технологий, одной из которых является OpenGL. Данная технология интересна своей поддержкой отличных от системы Microsoft Windows операционных систем, в то время как DirectX совместима только с ОС Windows. Собственно была даже своеобразная война между сторонниками этих систем.

Но на этом их конкурентоспособность и заканчивается: в системе OpenGL отсутствует поддержка работы со звуком, сетью, устройствами ввода и т. д. Зато отдельные компоненты DirectX прекрасно уживаются с OpenGL, что и восполняет все пробелы.



Глава 2

Среда разработки Borland Delphi

Изучать систему DirectX мы будем применительно к языку программирования Pascal, и работать будем в довольно популярной среде разработки Borland Delphi 7. Собственно версия особого значения не имеет, будь то более ранняя или поздняя версия — достаточно установить соответствующие версии заголовочных файлов DirectX и запустить наши примеры. При невозможности открытия проекта (несоответствие формата формы и т. п.) достаточно создать новый проект и перенести код в него.

Структура среды разработки

Среда разработки Borland Delphi 7 состоит из нескольких отдельно расположенных функциональных окон. К основным можно отнести следующие окна:

- палитра компонентов (**Component Palette**);
- дизайнер форм (**Form Designer**);
- дерево объектов (**Object TreeView**);
- инспектор объектов (**Object Inspector**);
- окно редактора кода (**Editor Window**).

Имеются также и различные окна отладки, настроек и т. д. Общий вид среды разработки Delphi после запуска представлен на рис. 2.1.

Сверху расположена палитра компонентов, под ней слева размещаются друг под другом дерево объектов и инспектор объектов. Справа от них находятся окно редактора кода и дизайнер форм.

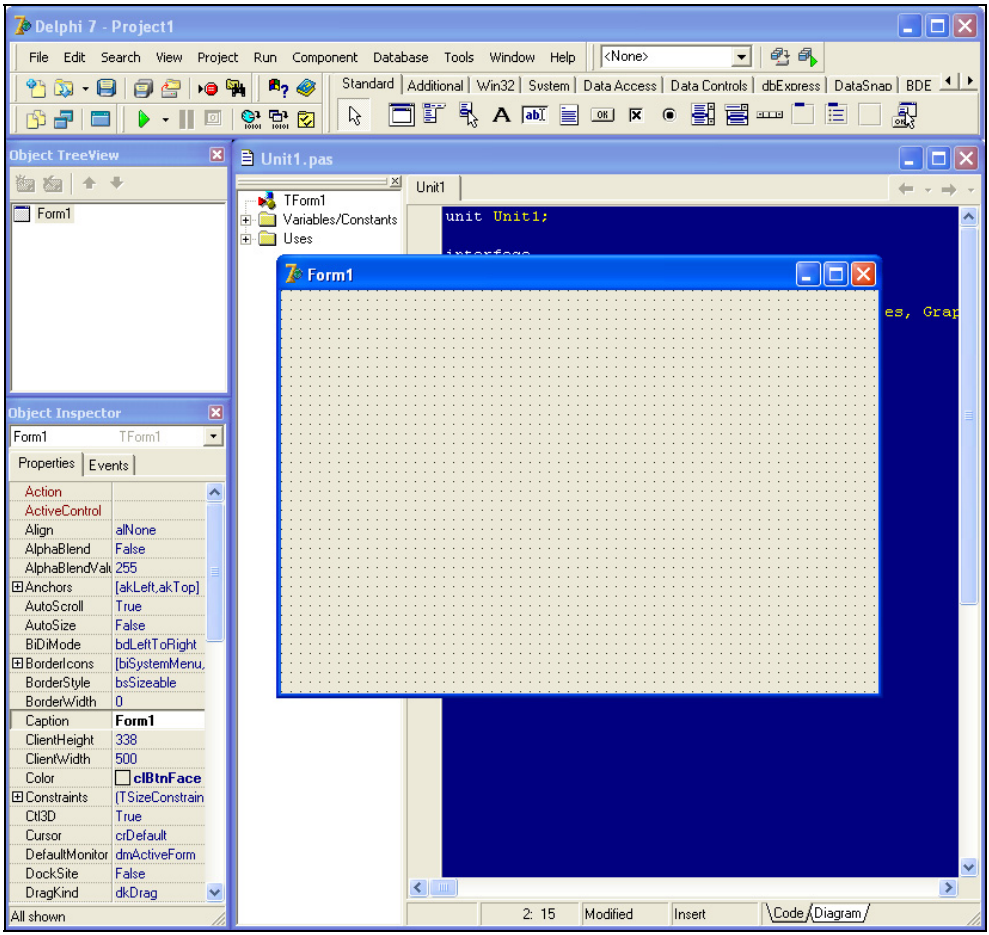


Рис. 2.1. Среда разработки Borland Delphi

Первое приложение

Создавать приложения в Delphi очень удобно и даже просто. Выберем пункт меню **File | New | Application**, и у нас появляется каркас приложения с готовым файлом проекта и одной формой (рис. 2.2). Сохранив проект на диск и нажав клавишу <F9>, мы откомпилируем и запустим наше приложение, состоящее из одной пустой формы. Как бы там ни было, это законченное приложение, пусть и абсолютно бесполезное с точки зрения функциональности, зато в дальнейшем такое простое создание окон сильно облегчит нашу жизнь.

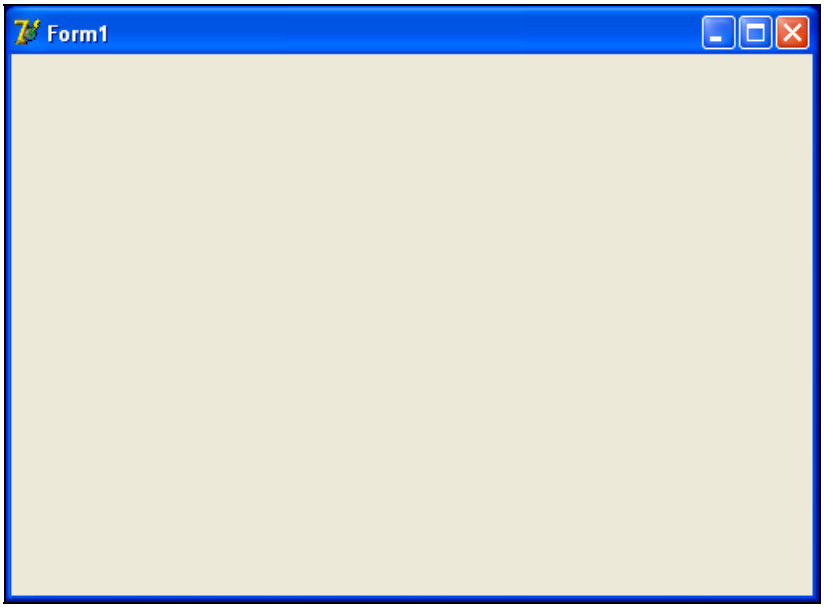
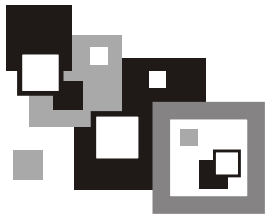


Рис. 2.2. Первое приложение

Однако не стоит думать, что работа с подсистемой DirectX настолько же проста, насколько и создание простейшего приложения в Delphi.

Глава 3



Библиотека COM

Так что же такое COM? Что это за библиотека и зачем она нужна? Почему DirectX базируется на COM?

Расшифровывается аббревиатура COM как Component Object Model — модель компонентных объектов. Модель не зависит от платформы, является распределенной объектно-ориентированной системой для создания бинарных интерактивных компонентов. И в связи с тем, что, как уже было сказано, DirectX базируется на COM, мы должны иметь хотя бы минимальное представление о данной библиотеке и уметь пользоваться ее возможностями. Несмотря на то, что библиотека COM покажется кому-то сложной и запутанной, в большинстве случаев ее использование в разработке приложений с помощью DirectX оказывается достаточно простым.

COM или DLL?

Библиотеки динамической компоновки (DLL, Dynamic Link Library) предоставляют схожую функциональность. Приложение подгружает библиотеку и может использовать ее функциональность в своих целях. Подобно DLL, объекты COM предоставляют методы, которые приложение может использовать по своему усмотрению. Взаимодействие с объектами COM происходит практически так же, как и работа с объектами Delphi. Но имеется и целый ряд отличий:

- объекты COM имеют более строгую инкапсуляцию. Мы не можем просто так создать COM-объект и использовать все его методы, т. к. все методы, так или иначе, сгруппированы по интерфейсам. Для вызова определенного метода нам может понадобиться создать COM-объект и запросить нужный интерфейс;

- объекты COM — это не объекты Delphi, и процесс их создания будет иным. Существует несколько способов создания COM-объектов, но все они используют методы библиотеки COM. API системы DirectX содержит ряд методов, которые упрощают создание некоторых объектов DirectX;
- для управления жизненным циклом объекта мы должны так же пользоваться методами библиотеки COM;
- COM-объекты не требуют явной загрузки. Обычно эти объекты COM так же располагаются в DLL, но нам не требуется загружать эту библиотеку или подключать ее статически для использования COM-объектов. Каждый COM-объект имеет свой уникальный идентификатор, который и используется для его создания. COM автоматически загружает нужную библиотеку DLL;
- поскольку модель COM представляет собой стандарт бинарной разработки для программных компонентов, то это означает независимость от языка разработки. Создаваемые объекты могут выполняться в одном процессе, в разных процессах и даже на другом компьютере.

Объекты и интерфейсы

Выше уже было упомянуто такое понятие, как *интерфейс*. Под интерфейсом понимается набор сгруппированных по определенным признакам методов. COM-объект — это реализация интерфейса (одного или нескольких одновременно). То есть фактически при вызове какого-либо метода интерфейса мы вызываем метод объекта. Отдельно следует упомянуть, что как один COM-объект может реализовывать произвольное количество интерфейсов, так и один интерфейс может реализовываться различными объектами COM.

В Delphi есть такое понятие, как *абстрактный метод*. Из таких методов строятся *абстрактные классы*. Интерфейсы и абстрактные классы очень схожи по своей сути, но имеют и ряд существенных отличий. Например, класс, являющийся производным, может реализовывать несколько интерфейсов, в то время как у него может быть только один базовый класс.

Интерфейс *IUnknown*

Так же как и класс Delphi имеет базовый класс `TObject`, так и для интерфейсов определен базовый интерфейс — `IUnknown`. По правде говоря, в Delphi, начиная с 6 версии, этот интерфейс именуется `IInterface`, что в принципе не меняет его сути. Для интерфейсов, в отличие от объектов, наследование не может означать повторного использования кода, т. к. интерфейс и его реализация — две абсолютно разные вещи. Также нужно ска-

зать, что наследование интерфейсов не может быть выборочным, т. е. производный интерфейс наследует все методы базового интерфейса.

Интерфейс `IUnknown` содержит всего три виртуальных метода. Первый — это получение указателя на интерфейс COM-объекта:

```
function QueryInterface(
  const IID: TGUID;
  out Obj):
HResult; stdcall;
```

Здесь:

- `IID` — уникальный идентификатор запрашиваемого интерфейса;
- `Obj` — переменная, в которую будет занесен запрашиваемый интерфейс. Если объект не поддерживает запрашиваемый интерфейс, то в переменную будет записано нулевое (`NIL`) значение.

Оставшиеся два метода управляют подсчетом ссылок. Увеличение числа ссылок на единицу:

```
function _AddRef: Integer; stdcall;
```

И уменьшение числа ссылок на объект на единицу:

```
function _Release: Integer; stdcall;
```

Методы управления подсчетом ссылок не требуется вызывать в явном виде — Delphi сделает это автоматически. Это означает, что при создании COM-объекта будет автоматически вызван метод `_AddRef`, а при присваивании указателю на интерфейс значения `NIL` (или когда объект выйдет за область видимости) автоматически будет вызван метод `_Release`.

Уникальные идентификаторы

Как вы уже наверно успели заметить, в методе `QueryInterface` первым параметром мы передаем некий уникальный идентификатор запрашиваемого интерфейса, имеющий тип данных `TGUID`.

Глобальные идентификаторы являются ключевой составляющей библиотеки COM. Если просто посмотреть на этот идентификатор, то это обычная структура (запись), состоящая из 128 битов. При создании идентификатора гарантируется его уникальность. COM широко использует эти идентификаторы для следующих целей:

- для уникальной идентификации COM-объекта. Такие идентификаторы называются *идентификаторами класса* (Class Identifier, CLSID). Они будут использоваться нами для создания конкретного COM-объекта;

- для идентификации определенного интерфейса. Значение GUID, которое определяет некоторый интерфейс, будет называться *идентификатором интерфейса* (Interface Identifier, IID).

Несмотря на то, что уникальный идентификатор представляет собой структуру, его нередко записывают в виде строки. Ее формат — это пять шестнадцатеричных целых чисел в формате `8-4-4-4-12` или `"{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}"`. Например, идентификатор интерфейса `IDirect3D9`:

```
{81BDCBCA-64D4-426d-AE8D-AD0147F4275C}
```

Идентификатор запомнить достаточно проблематично, и при его написании легко допустить ошибку. Гораздо проще использовать его эквивалентное имя. К примеру, это имя мы можем использовать для создания COM-объекта. По принятым соглашениям мы должны добавлять префиксы `IID_` или `CLSID_` к имени интерфейса или объекта. Например, для интерфейса `IDirect3D9` идентификатором будет выступать `IID_IDirect3D9`.

Результат *HResult*

Методы COM-объектов возвращают 32-битное целочисленное значение типа `HResult`. В большинстве случаев тип `HResult` представляет собой структуру, содержащую две основные информативные части:

- корректно ли отработал метод или произошла ошибка;
- более детальная информация о результате операции.

Можно использовать в качестве значения константы, описанные в модуле Windows.pas, такие как `S_OK`, `E_FAIL`, `E_UNEXPECTED`, `E_NOTIMPL` и т. д. Но можно использовать и собственные значения. Результаты вызовов методов COM-объекта обычно описываются в документации к ним.

Существует соглашение, по которому коды успешного завершения метода начинаются с префикса `S_`, а коды завершения с ошибкой — с префикса `E_`, например, `S_OK` и `E_FAIL`.

То, что методы могут возвращать различные варианты успеха или неудачи, означает, что нам нужно быть внимательными при анализе результата. К примеру, метод возвращает `S_OK` при успешном завершении работы и `E_FAIL` при ошибке. Тогда код обработки результата может выглядеть следующим образом:

```
if Result = E_FAIL then
  begin
    // Произошла ошибка, обрабатываем
  end
```

```
else begin
    // Ошибок нет
end;
```

А теперь допустим, что код ошибки может быть равен E_FAIL, E_UNEXPECTED, E_NOTIMPL и т. п. А у нас анализируется только E_FAIL, и все остальные ошибочные результаты будут обработаны так же, как и успешные. Это означает, что нам понадобится более детальный анализ всех возможных результатов.

Для облегчения нашей печальной участи в модуле Windows.pas определены 2 метода, которые тестируют результат HResult на предмет успеха или ошибки. Первый метод — Succeeded. Он проверяет, является ли результат успешным или нет:

```
function Succeeded(
  Status: HRESULT):
BOOL;
```

Здесь Status — тестируемое значение.

Результатом вызова этого метода будет TRUE, если тестируемое значение является успешным результатом выполнения метода, и FALSE — в противном случае.

Второй метод — Failed, который полностью противоположен первому — он проверяет, является ли результат ошибочным или нет:

```
function Failed(
  Status: HRESULT):
BOOL;
```

Результатом вызова этого метода будет TRUE, если тестируемое значение является ошибочным, и FALSE — в противном случае.

В своей работе мы достаточно часто будем применять эти два метода. Скажем, тот пример, который мы приводили ранее, должен быть исправлен следующим образом:

```
if FAILED(Result) then
begin
  // Произошла ошибка, обрабатываем
end
else begin
  // Ошибок нет
end;
```

Но не следует думать, что все методы COM-объектов возвращают тип HResult. Так, например, методы IUnknown._AddRef и IUnknown._Release возвращают текущее количество ссылок на объект.

Инициализация библиотеки COM и завершение работы с ней

Начинать работу с библиотекой COM необходимо с инициализации. Для этого имеется метод CoInitializeEx, описанный в модуле ActiveX.pas:

```
function CoInitializeEx(
  pvReserved: Pointer;
  coInit: Longint):
HResult; stdcall;
```

Здесь:

- pvReserved — зарезервировано. Должно использоваться нулевое значение;
- coInit — флаг, определяющий потоковую модель:
  - COINIT_MULTITHREADED — многопоточная модель — объекты могут вызываться из разных потоков;
  - COINIT_APARTMENTTHREADED — раздельное адресное пространство у потоков;
  - COINIT_DISABLE_OLE1DDE — отключение поддержки DDE для OLE1;
  - COINIT_SPEED_OVER_MEMORY — использование большего объема памяти для увеличения быстродействия.

Этот метод производит инициализацию статических и загружаемых библиотек COM и устанавливает текущую потоковую модель. Он должен быть вызван перед началом использования функций COM API (кроме функции CoGetMalloc и функций распределения памяти).

Для завершения работы с COM необходимо вызвать метод CoUninitialize, который освобождает ресурсы загруженных библиотек:

```
procedure CoUninitialize; stdcall;
```

Вызов метода будет иметь успех только тогда, когда перед ним был произведен вызов метода инициализации библиотеки COM CoInitializeEx. И наоборот, если был произведен вызов CoInitializeEx, то вызов CoUninitialize обязателен.

Глава 4

Пишем COM-сервер

Создание COM-сервера

В этой главе мы с вами создадим первый COM-сервер, научимся его регистрировать и использовать все его возможности в своем приложении. В роли COM-сервера будет выступать так называемый In-Process COM Server, реализованный в виде DLL.

Для создания такой библиотеки в Delphi необходимо сформировать библиотеку ActiveX. Выберем пункт меню **File | New | Other...**, и у нас на экране появится диалог выбора создаваемого объекта. Перейдем на вкладку **ActiveX** и выберем пункт **ActiveX Library** (рис. 4.1).

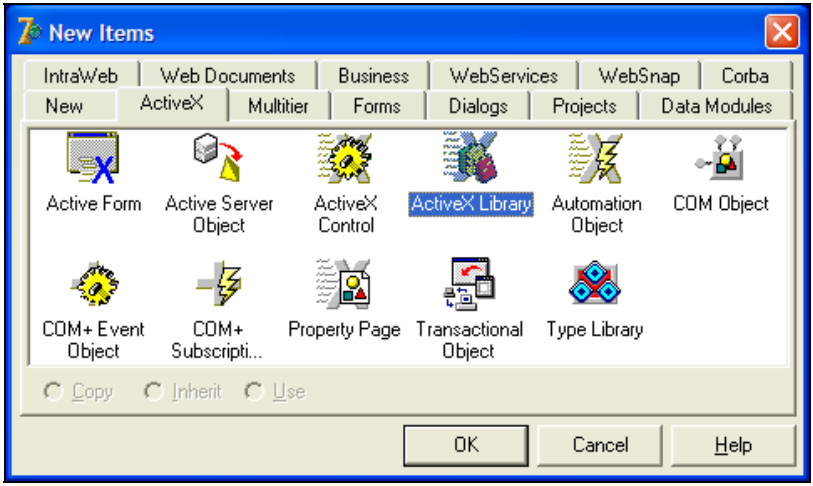


Рис. 4.1. Диалог создания нового объекта

Сохраним проект под именем COM_Server. Весь текст нашего модуля состоит всего из нескольких строк (листинг 4.1).

Листинг 4.1. Текст модуля COM_Server.dpr

```
library COM_Server;

uses
  ComServ;

exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.RES}

begin
end.
```

Как мы видим, в модуле присутствуют четыре экспортируемые функции:

- `DllGetClassObject` — получение класса объекта;
- `DllCanUnloadNow` — проверка возможности выгрузки COM-сервера из памяти;
- `DllRegisterServer` — регистрация COM-сервера в системном реестре;
- `DllUnregisterServer` — удаление из реестра информации о COM-сервере.

Для нас нет необходимости в реализации данных функций, т. к. они уже и так реализованы в модуле ComServ.pas.

Теперь необходимо собственно создать код COM-сервера. Для примера я хотел бы реализовать следующую идею: мы создадим объект `TSimpleObject`, который будет реализовывать два интерфейса — интерфейс конфигурирования `ISimpleConfigurator` и интерфейс рисования `ISimpleDrawing`. Так же как и для создания ActiveX-библиотеки, нам нужно выбрать пункт меню **File | New | Other...** и перейти на вкладку **ActiveX**. Далее следует выбрать объект **COM Object** и нажать кнопку **OK**. У нас на экране появится мастер создания COM-объектов (COM Object Wizard). Зададим параметры COM-объекта,

как это показано на рис. 4.2. Заметьте, что, несмотря на имя объекта `SimpleObject`, в качестве имени реализуемого интерфейса мы указываем `ISimpleConfigurator`. Флажок **Include Type Library** оставьте отмеченным, тогда для нашего объекта будет автоматически сгенерирована библиотека типов, которая нам пригодится при создании клиентского приложения.

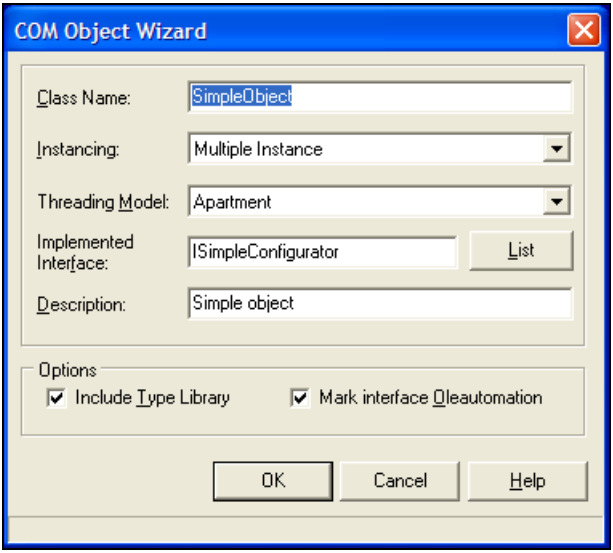


Рис. 4.2. Диалог создания COM-объекта

Сохраним созданный модуль под именем UnitMain.pas. Обратите внимание на то, что теперь в модуле COM_Server.dpr в список используемых модулей добавились еще 2 — библиотека типов и модуль с нашим классом:

```
uses
  ComServ,
  COM_Server_TLB in 'COM_Server_TLB.pas',
  UnitMain in 'UnitMain.pas' {SimpleObject: CoClass};
```

Выберите пункт меню **View | Type Library**. На экране появится окно редактирования библиотеки типов. Здесь мы можем выполнять такие действия, как добавление и удаление интерфейсов, добавление и удаление методов интерфейсов и типов данных и т. д. Наш интерфейс `ISimpleConfigurator` не содержит пока ни одного метода. Давайте добавим в него следующие методы: метод установки окна рисования, метод установки цвета границы и цвета заполнения, метод установки размера объекта.

Метод установки окна рисования:

```
function SetWindow(
  Handle: Integer):
HResult; stdcall;
```

Здесь `Handle` — дескриптор окна рисования.

Установка цветов:

```
function SetColors(
  BorderColor,
  FillColor: Integer):
HResult; stdcall;
```

Здесь:

- `BorderColor` — цвет границы;
- `FillColor` — цвет заливки.

Установка размера объекта:

```
function SetObjectSize(
  ObjectSize: Integer):
HResult; stdcall;
```

Здесь `ObjectSize` — новый размер объекта.

После этого добавим в список второй интерфейс — `ISimpleDrawing` и установим в качестве его родительского интерфейса интерфейс `IUnknown`. Перейдем на вкладку **Flags** данного интерфейса и снимем флажок с параметра **Dual**. Должен остаться только один флажок — **Ole Automation**. Выберем в дереве объектов библиотеки типов объект `SimpleObject`, перейдем на его вкладку **Implements** и добавим в список созданный интерфейс `ISimpleDrawing`.

В интерфейсе `ISimpleDrawing` мы объявим всего один метод — рисование некоторого объекта:

```
function DrawSimpleObject(
  X,
  Y: Integer):
HResult; stdcall;
```

Здесь `X`, `Y` — координаты объекта.

Рисовать в качестве объекта можно все что угодно — мы же будем рисовать закрашенный квадрат.

Внеся все эти изменения, необходимо нажать кнопку обновления на панели библиотеки типов, после чего все изменения, сделанные в этом редакторе, появятся и в редакторе Delphi.

Если вы сделали все правильно, то у вас должно получиться то же самое, что представлено на рис. 4.3.

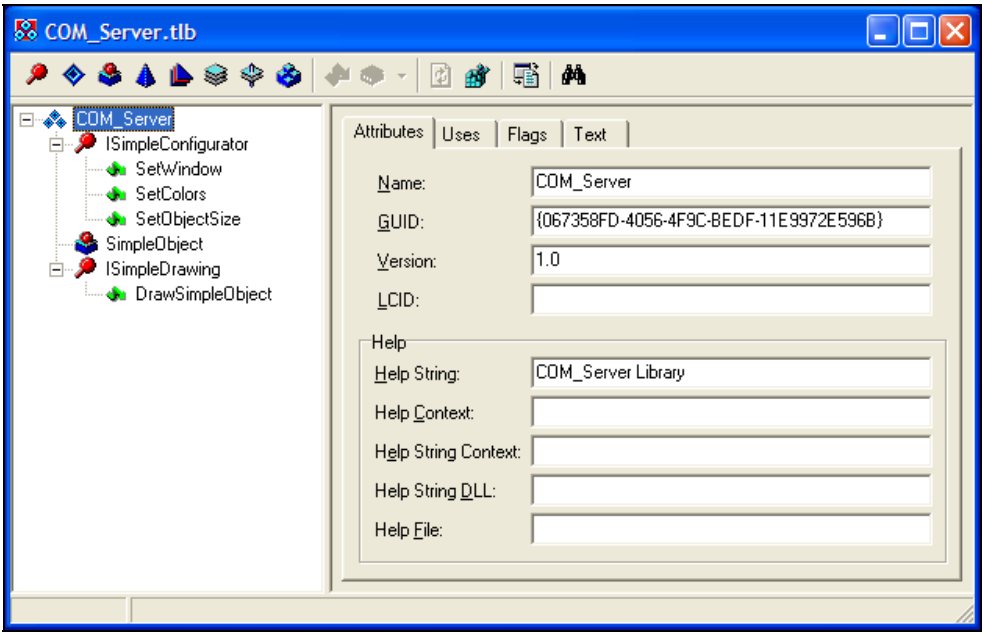


Рис. 4.3. Редактор библиотеки типов

Теперь остается только написать реализацию перечисленных выше 4-х методов. Наибольший интерес для нас представляет собственно единственный метод рисования `ISimpleDrawing.DrawSimpleObject`. Для рисования в чужом окне мы будем использовать внутри COM-сервера объект типа `TCanvas`. Было бы крайне нежелательно создавать и удалять этот объект каждый раз в процедуре рисования, и поэтому мы переопределим методы инициализации и удаления COM-объекта: `Initialize` и `Finalize`. В метод инициализации мы добавим создание объекта типа `TCanvas` и проинициализируем начальные значения размера объекта и цветов границы и заливки. Метод удаления COM-объекта будет содержать код удаления объекта `TCanvas`. Текст модуля приведен в листинге 4.2.

Листинг 4.2. Текст модуля UnitMain.pas проекта COM_Server

```
unit UnitMain;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, Messages, SysUtils, Graphics, ActiveX, Classes, ComObj,
  COM_Server_TLB, StdVcl;

type
  TSimpleObject = class(TTypedComObject, ISimpleConfigurator,
    ISimpleDrawing)
  private
    FCanvas: TCanvas;
    FHandle: THandle;
    FObjectSize: integer;
    FBorderColor,
    FFillColor: integer;

  protected
    function SetObjectSize(ObjectSize: Integer): HResult; stdcall;
    function SetColors(BorderColor, FillColor: Integer): HResult;
      stdcall;
    function SetWindow(Handle: Integer): HResult; stdcall;

    function DrawSimpleObject(X, Y: Integer): HResult; stdcall;
  public
    procedure Initialize; override;
    destructor Destroy; override;
  end;

implementation
```

```
uses ComServ;

function TSimpleObject.SetObjectSize(ObjectSize: Integer): HResult;
begin
  FObjectSize := ObjectSize;

  Result := S_OK;
end;

function TSimpleObject.SetColors(BorderColor, FillColor: Integer):
HResult;
begin
  FBorderColor := BorderColor;
  FFillColor := FillColor;

  Result := S_OK;
end;

function TSimpleObject.SetWindow(Handle: Integer): HResult;
begin
  FHandle := Handle;

  Result := S_OK;
end;

function TSimpleObject.DrawSimpleObject(X, Y: Integer): HResult;
var
  DC: HDC;
begin
  Result := E_FAIL;
  if FHandle = 0 then EXIT;

  DC := GetDC(FHandle);
  try
    FCanvas.Handle := DC;
    FCanvas.Pen.Color := FBorderColor;
```

```
    FCanvas.Brush.Color := FFillColor;
    FCanvas.Rectangle(X, Y, X + FObjectSize, Y + FObjectSize);
  finally
    FCanvas.Handle := 0;
    ReleaseDC(FHandle, DC);
  end;

  Result := S_OK;
end;

destructor TSimpleObject.Destroy;
begin
  if Assigned(FCanvas) then
  begin
    FreeAndNil(FCanvas);
  end;

  inherited;
end;

procedure TSimpleObject.Initialize;
begin
  inherited;

  FCanvas := TCanvas.Create;

  FObjectSize  := 5;
  FBorderColor := clBlack;
  FFillColor   := clWhite;
end;

initialization
  TTypedComObjectFactory.Create(ComServer, TSimpleObject,
    Class_SimpleObject, ciMultiInstance, tmApartment);
end.
```

Регистрация COM-сервера

После компиляции мы получим библиотеку COM_Server.dll. Это и есть наш COM-сервер. Для его использования нам остается только его зарегистрировать. Это можно проделать одним из следующих способов:

- выполнив команду меню **Run | Register ActiveX Server**;
- запустив из командной строки специальную программу regsvr32.exe и передав ей в качестве первого параметра путь к COM-серверу;
- в комплекте с Delphi идет специальная утилита для регистрации COM-серверов под названием TRegSvr.exe — можно воспользоваться ее возможностями для регистрации нашего сервера.

Естественно, это неполный список возможных путей регистрации сервера. Можно, к примеру, написать свой инсталлятор, который будет автоматически его регистрировать, но нам это не нужно.

В каталоге Examples\COM\Server\ на прилагаемом к книге компакт-диске находится описанный выше пример COM-сервера. Также в этом каталоге располагается и файл REG.CMD — этот командный файл производит регистрацию COM-сервера. Вся регистрация осуществляется одной строкой:

```
regsvr32 COM_Server.dll
```

После регистрации COM-сервера информация о нем заносится в реестр Windows. Мы можем просмотреть эту информацию из редактора реестра Windows (рис. 4.4).

Ну, и соответственно после регистрации сервера мы можем со спокойной совестью приступать к написанию клиентской части.

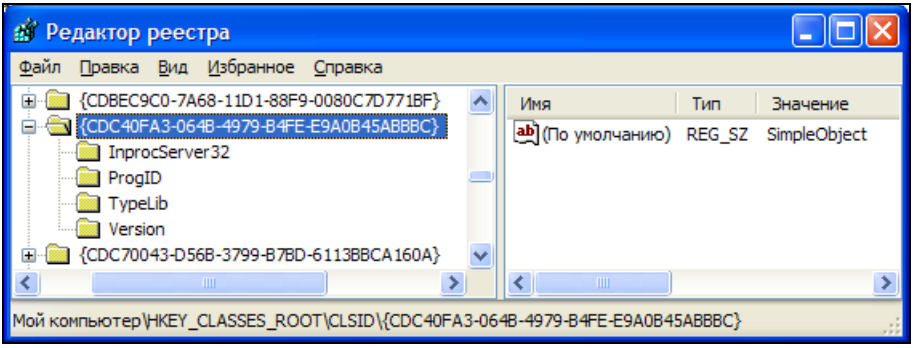


Рис. 4.4. Зарегистрированный в реестре COM-сервер

Клиентская часть

Создадим обычное приложение и подключим к нему модуль библиотеки типов COM_Server_TLB.pas из проекта COM-сервера. Теперь мы можем использовать возможности COM-сервера в своем приложении.

Для большей наглядности я решил привести два примера создания COM-объектов. В первом случае мы будем вызывать метод `CreateComObject` из модуля ComObj.pas, а во втором — функцию API `CoCreateInstance`. Как бы то ни было, первый метод гораздо проще — у него всего один параметр, но все равно он неявно использует вызов API:

```
function CreateComObject(const ClassID: TGUID): IUnknown;
begin
  OleCheck(CoCreateInstance(ClassID, nil, CLSCTX_INPROC_SERVER or
    CLSCTX_LOCAL_SERVER, IUnknown, Result));
end;
```

Метод `CoCreateInstance` создает экземпляр объекта заданного класса:

```
function CoCreateInstance(
  const clsid: TCLSID;
  unkOuter: IUnknown;
  dwClsContext: Longint;
  const iid: TIID;
  out pv):
HResult; stdcall;
```

Здесь:

- `clsid` — идентификатор класса создаваемого объекта (CLSID);
- `unkOuter` — указывает, используется ли объект в агрегации. Нулевое (`NIL`) значение говорит о том, что объект не является частью агрегата;
- `dwClsContext` — контекст, в котором объект должен быть создан. Мы будем использовать значение `CLSCTX_INPROC`, которое, в свою очередь, является комбинацией двух следующих значений:
  - `CLSCTX_INPROC_SERVER` — внутренний сервер;
  - `CLSCTX_INPROC_HANDLER` — код сервера будет загружен в адресное пространство приложения (суррогатный сервер);
- `iid` — идентификатор нужного интерфейса;
- `pv` — переменная, в которую будет записан указатель на нужный интерфейс.

При создании основной формы приложения мы создаем два экземпляра нашего COM-объекта разными способами и настраиваем различные пара-

метры. Для первого экземпляра COM-объекта мы только назначим окно для рисования, а остальные параметры оставим без изменения, а для другого экземпляра устанавливаем все параметры. Затем запрашиваем у экземпляра каждого COM-объекта второй интерфейс — интерфейс рисования. Рисовать мы будем в обработчике события `OnMouseMove` у формы: при нажатии левой кнопки мыши будет вызываться метод рисования первого COM-объекта, а при нажатии правой кнопки мыши — второго.

Посмотрите на пример, расположенный в каталоге Examples\COM\Client на компакт-диске. Этот пример представляет собой клиентскую часть, использующую возможности описанного выше COM-сервера. В листинге 4.3 приводится текст примера.

Листинг 4.3. Текст модуля FormMain.pas проекта COM_Client

```
unit FormMain;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, ActiveX, ComObj, COM_Server_TLB, StdCtrls;

type
  TMainForm = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
  private
    FSimpleObject1:  ISimpleConfigurator;
    FSimpleDrawing1: ISimpleDrawing;

    FSimpleObject2:  ISimpleConfigurator;
    FSimpleDrawing2: ISimpleDrawing;
  public
  end;
var
  MainForm: TMainForm;
```

```
implementation

{$R *.dfm}

procedure TMainForm.FormCreate(Sender: TObject);
var
  hr: HResult;
begin
  FSimpleObject1 := CreateComObject(CLASS_SimpleObject) as
    ISimpleConfigurator;
  if FSimpleObject1 <> NIL then
  begin
    FSimpleObject1.SetWindow(Handle);
    FSimpleObject1.QueryInterface(IID_ISimpleDrawing, FSimpleDrawing1);
  end;

  hr := CoCreateInstance(CLASS_SimpleObject, NIL, CLSCTX_INPROC,
    IID_ISimpleConfigurator, FSimpleObject2);
  if (SUCCEEDED(hr)) then
  begin
    FSimpleObject2.SetWindow(Handle);
    FSimpleObject2.SetObjectSize(10);
    FSimpleObject2.SetColors(clRed, clYellow);
    FSimpleObject2.QueryInterface(IID_ISimpleDrawing, FSimpleDrawing2);
  end;
end;

procedure TMainForm.FormDestroy(Sender: TObject);
begin
  if FSimpleDrawing2 <> NIL then FSimpleDrawing2 := NIL;
  if FSimpleObject2 <> NIL then FSimpleObject2 := NIL;
  if FSimpleDrawing1 <> NIL then FSimpleDrawing1 := NIL;
  if FSimpleObject1 <> NIL then FSimpleObject1 := NIL;
end;

procedure TMainForm.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
```

```
begin
  if ssLeft in Shift then
  begin
    if FSimpleDrawing1 <> NIL then
    begin
      FSimpleDrawing1.DrawSimpleObject(X, Y);
    end;
  end else if ssRight in Shift  then
  begin
    if FSimpleDrawing2 <> NIL then
    begin
      FSimpleDrawing2.DrawSimpleObject(X, Y);
    end;
  end;
end;

end.
```

Поэкспериментировав с примером, мы можем получить следующий результат (рис. 4.5).

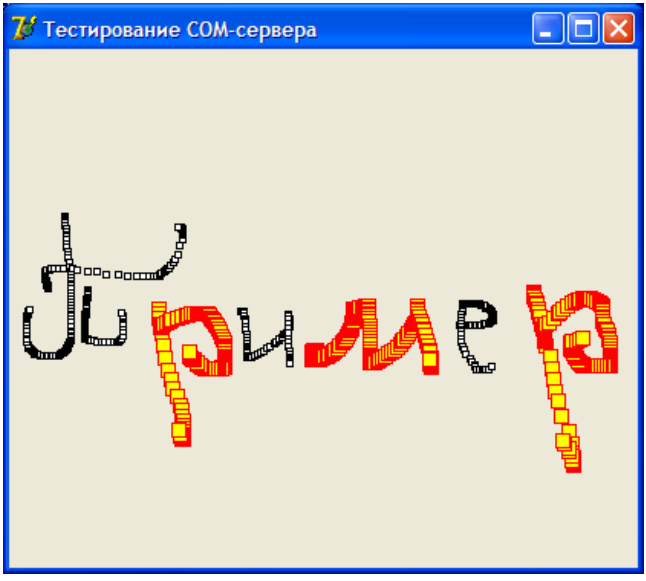
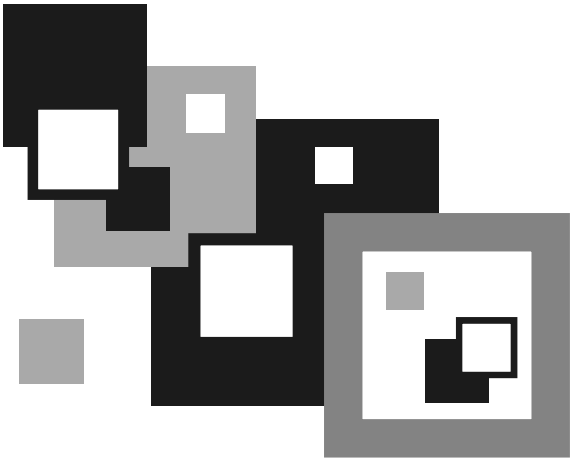


Рис. 4.5. Использование COM-сервера

Итоги

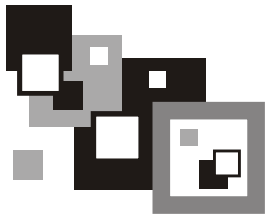
В этой вводной части мы с вами перечислили все компоненты, входящие в состав DirectX, определили, в чем же отличие DirectX от OpenGL, кратко описали среду разработки и научились работать с библиотекой COM на примере. Он является лишь простейшим примером работы с библиотекой COM и не показывает всех возможностей этой технологии, но в любом случае будет полезен тем, кто не имел опыта работы с данной библиотекой.



ЧАСТЬ II

DIRECTX GRAPHICS

Глава 5



Direct3D

Начнем знакомство с графикой с изучения подсистемы Direct3D. Все примеры, которые будут нами рассмотрены, располагаются на прилагаемом к книге компакт-диске в каталоге Examples\DirectX Graphics\Direct3D\Example_*xx*, где *xx* — порядковый номер примера. Первые построения будут сделаны на плоскости — мы начнем изучение с инициализации подсистемы и вывода на экран точки. Доберемся до рисования прямоугольника на плоскости, а затем и в пространстве, и перейдем к изучению пространственных построений. Рассмотрим различные матричные преобразования, работу с текстом на плоскости и в пространстве, изучим работу с материалом и источниками света и т. д.

Инициализация

Под инициализацией данной подсистемы нужно понимать создание главного объекта с интерфейсом `IDirect3D9` и создание объекта устройства `IDirect3DDevice9`. Рассмотрим это подробнее.

Первым делом нам необходимо создать объект с интерфейсом `IDirect3D9`. Это делается при помощи вызова метода `Direct3DCreate9`:

```
function Direct3DCreate9(
  SDKVersion: LongWord):
IDirect3D9;
```

Здесь `SDKVersion` — константа, необходимая для проверки того, что программа была скомпилирована с нужными версиями заголовочных файлов и библиотек. Должно использоваться значение `D3D_SDK_VERSION`.

Создав главный объект, переходим к созданию объекта устройства, который создается при помощи метода `IDirect3D9.CreateDevice`:

```
function CreateDevice(
  Adapter: LongWord;
  DeviceType: TD3DDevType;
  hFocusWindow: HWND;
  BehaviorFlags: DWord;
  pPresentationParameters: PD3DPresentParameters;
  out ppReturnedDeviceInterface: IDirect3DDevice9):
HResult; stdcall;
```

Здесь:

- `Adapter` — порядковый номер адаптера. Основной адаптер всегда имеет номер `D3DADAPTER_DEFAULT`;
- `DeviceType` — тип устройства. Может принимать одно из следующих значений:
  - `D3DDEVTYPE_HAL` — аппаратная растеризация;
  - `D3DDEVTYPE_REF` — программная растеризация;
  - `D3DDEVTYPE_SW` — сменное программное устройство, зарегистрированное при помощи метода `IDirect3D9.RegisterSoftwareDevice`;
- `hFocusWindow` — дескриптор окна, которое будет использовано подсистемой Direct3D для вывода изображения. Для полноэкранного режима должно указываться окно верхнего уровня;
- `BehaviorFlags` — комбинация одного или нескольких флагов:
  - `D3DCREATE_ADAPTERGROUP_DEVICE` — текущий адаптер становится основным (по отношению к остальным);
  - `D3DCREATE_DISABLE_DRIVER_MANAGEMENT` — подсистема Direct3D будет сама управлять ресурсами вместо драйвера;
  - `D3DCREATE_FPU_PRESERVE` — двойная точность при операциях с плавающей точкой. Как следствие, снижается производительность;
  - `D3DCREATE_HARDWARE_VERTEXPROCESSING` — аппаратная обработка вершин;
  - `D3DCREATE_MIXED_VERTEXPROCESSING` — смешанная (программная и аппаратная) обработка вершин;
  - `D3DCREATE_MULTITHREADED` — указывает на то, что приложение Direct3D должно быть многопоточным. Создается глобальная критическая секция, которая снижает производительность;

- D3DCREATE_PUREDEVICE — не поддерживать Get-вызовы и отключить поддержку сервисов эмуляции при обработке вершин;
- D3DCREATE_SOFTWARE_VERTEXPROCESSING — программная обработка вершин.
- Флаги D3DCREATE_HARDWARE_VERTEXPROCESSING, D3DCREATE_MIXED_VERTEXPROCESSING и D3DCREATE_SOFTWARE_VERTEXPROCESSING являются взаимоисключающими;

- pPresentationParameters — указатель на структуру TD3DPresentParameters, описывающую параметры устройства;
- ppReturnedDeviceInterface — адрес указателя, в который будет занесен интерфейс IDirect3DDevice9 созданного объекта.

Структура TD3DPresentParameters, описывающая параметры создаваемого устройства, имеет следующие поля:

- BackBufferWidth — ширина заднего буфера;
- BackBufferHeight — высота заднего буфера;
- BackBufferFormat — формат заднего буфера. Может использоваться текущий формат, полученный с помощью метода IDirect3DDevice9.GetDisplayMode;
- BackBufferCount — число задних буферов (0, 1, 2 или 3, причем значение 0 в данном случае эквивалентно 1). При невозможности создать указанное число буферов в данный параметр будет занесено реальное число созданных буферов;
- MultiSampleType — тип мультисэмплинга (полноэкранного сглаживания). Данный параметр используется только при установленном режиме обмена D3DSWAPEFFECT_DISCARD;
- MultiSampleQuality — качество полноэкранного сглаживания;
- SwapEffect — эффект обмена буферов;
- hDeviceWindow — дескриптор окна вывода (для полноэкранных приложений);
- Windowed — флаг, определяющий режим работы (оконный или полноэкранный);
- EnableAutoDepthStencil — управление буфером глубины;
- AutoDepthStencilFormat — формат буфера глубины;
- Flags — флаг, задающий возможность блокирования заднего буфера;
- FullScreen_RefreshRateInHz — частота обновления в полноэкранном режиме;
- PresentationInterval — синхронизация частоты адаптера с частотой вывода на экран заднего буфера.

Все эти действия будут выполняться в процедуре `InitD3D` наших примеров (листинг 5.1).

Листинг 5.1. Процедура `InitD3D`

```
{*****************************************************************}
{** Инициализация подсистемы Direct3D                           **}
{*****************************************************************}
function TMainForm.InitD3D: HResult;
var
  d3dDisplayInfo: TD3DDisplayMode;
  d3dParams: TD3DPresentParameters;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Создаем объект с интерфейсом IDirect3D9
  FD3D := Direct3DCreate9(D3D_SDK_VERSION);

  // Завершаем работу процедуры при ошибке
  if FD3D = NIL then EXIT;

  // Получаем установки текущего режима адаптера
  Result := FD3D.GetAdapterDisplayMode(D3DADAPTER_DEFAULT,
    d3dDisplayInfo);

  // Завершаем работу в случае сбоя
  if FAILED(Result) then EXIT;

  // Задаем параметры устройства:
  ZeroMemory(@d3dParams, SizeOf(d3dParams));
  //   использовать оконный режим
  d3dParams.Windowed := TRUE;
  //   режим переключения между буферами
  d3dParams.SwapEffect := D3DSWAPEFFECT_DISCARD;
  //   формат буфера не определен
  d3dParams.BackBufferFormat := d3dDisplayInfo.Format;
```

```
  // Создаем объект устройства
  Result := FD3D.CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
    Handle, D3DCREATE_SOFTWARE_VERTEXPROCESSING, @d3dParams,
    FD3DDevice);
end;
```

Очистка устройства

Очистка устройства производится методом `IDirect3DDevice9.Clear`. Данный метод позволяет очищать все пространство вывода либо набор прямоугольных областей и заливать фон указанным цветом. Так же метод очищает буфер глубины и буфер шаблона:

```
function Clear(
  Count: DWord;
  pRects: PD3DRect;
  Flags: DWord;
  Color: TD3DColor;
  Z: Single;
  Stencil: DWord):
HResult; stdcall;
```

Здесь:

- `Count` — число прямоугольных областей в массиве `pRects`. Если `pRects` имеет нулевое значение, то и данный параметр должен быть нулевым;
- `pRects` — массив прямоугольных областей, которые должны быть очищены. Допустимо использовать нулевое значение для очистки всей области целиком;
- `Flags` — параметр указывает, какие поверхности должны быть очищены:
  - `D3DCLEAR_STENCIL` — очищает буфер шаблона и задает ему значение, указанное в параметре `Stencil`;
  - `D3DCLEAR_TARGET` — устанавливает цвет очищаемой области в значение, указанное в параметре `Color`;
  - `D3DCLEAR_ZBUFFER` — очищает буфер глубины до значения, указанного в параметре `Z`;
- `Color` — цвет области в формате ARGB;
- `Z` — значение, которое будет использоваться при очистке буфера глубины. Может принимать значение от `0.0` до `1.0`;

- `Stencil` — новое значение буфера шаблона. Может принимать значение от 0 до $2^n - 1$, где n является разрядностью буфера шаблона.

В примерах очистка буфера будет производиться в методе `ClearDevice` (листинг 5.2).

Листинг 5.2. Метод `ClearDevice`

```
{*****************************************************************}
{** Очищаем устройство                                          **}
{*****************************************************************}
function TMainForm.ClearDevice: HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если устройство не создано, то завершаем работу
  if FD3DDevice = NIL then EXIT;

  // Чистим устройство
  Result := FD3DDevice.Clear(0, NIL, D3DCLEAR_TARGET,
    D3DCOLOR_XRGB($FF, 0, $FF), 1, 0);
end;
```

Прорисовка сцены

Прорисовка будет осуществляться в методе `RenderScene` (листинг 5.3). Первым делом мы очищаем устройство, а затем переключаем буферы.

Листинг 5.3. Метод `RenderScene`

```
{*****************************************************************}
{** Прорисовка сцены                                            **}
{*****************************************************************}
function TMainForm.RenderScene: HResult;
begin
  // Чистим устройство
  Result := ClearDevice;
```

```
  // Если произошла ошибка, то завершаем работу
  if FAILED(Result) then EXIT;

  // Переключение буферов
  FD3DDevice.Present(NIL, NIL, 0, NIL);
end;
```

Этот метод в дальнейшем будет усложнен по мере изучения нового материала.

Первый пример

Все, что было сказано выше, относится к следующему примеру. Это простейший пример, располагающийся в каталоге Example_01, в котором производится инициализация подсистемы Direct3D и окно вывода закрашивается цветом. На рис. 5.1 мы можем увидеть результат работы данного примера.

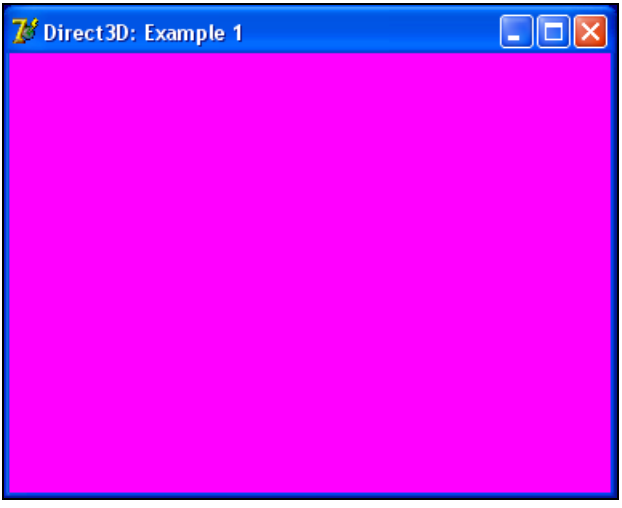


Рис. 5.1. Первый пример работы с Direct3D

В листинге 5.4 текст основного модуля приведен полностью со всеми комментариями. В дальнейшем я не вижу необходимости в представлении полных листингов всех примеров — достаточно будет приводить только те места кода, которые претерпевают какие-либо изменения.

Листинг 5.4. Текст модуля FormMain.pas примера Direct3D\Example_01

```
UNIT FormMain;
{*****************************************************************}
{** Direct3D: Example_01                                        **}
{** Автор: Есенин Сергей Анатольевич                            **}
{*****************************************************************}

{**} INTERFACE {**************************************************}

{**} USES {*******************************************************}
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, Direct3D9, ExtCtrls, AppEvnts;

{**} TYPE {*******************************************************}
  TMainForm = class(TForm)
    applicationEventsMain: TApplicationEvents;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure applicationEventsMainIdle(Sender: TObject;
      var Done: Boolean);
    procedure applicationEventsMainMinimize(Sender: TObject);
    procedure applicationEventsMainRestore(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  PRIVATE
    FD3D: IDirect3D9;
    FD3DDevice: IDirect3DDevice9;

    FIsActive: boolean;

    function InitD3D: HResult;
    procedure FreeD3D;

    function ClearDevice: HResult;
```

```
    function RenderScene: HResult;
  PUBLIC
  END;

{**} VAR {***************************************************}
  MainForm: TMainForm;

{**} IMPLEMENTATION {*********************************************}

{$R *.dfm}

{**} { TMainForm } {*********************************************}

{*****************************************************************}
{** Инициализация подсистемы Direct3D                           **}
{*****************************************************************}
function TMainForm.InitD3D: HResult;
var
  d3dDisplayInfo: TD3DDisplayMode;
  d3dParams: TD3DPresentParameters;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Создаем объект с интерфейсом IDirect3D9
  FD3D := Direct3DCreate9(D3D_SDK_VERSION);

  // Завершаем работу процедуры при ошибке
  if FD3D = NIL then EXIT;

  // Получаем установки текущего режима адаптера
  Result := FD3D.GetAdapterDisplayMode(D3DADAPTER_DEFAULT,
    d3dDisplayInfo);

  // Завершаем работу в случае сбоя
  if FAILED(Result) then EXIT;
```

```
  // Задаем параметры устройства:
  ZeroMemory(@d3dParams, SizeOf(d3dParams));
  //   использовать оконный режим
  d3dParams.Windowed := TRUE;
  //   режим переключения между буферами
  d3dParams.SwapEffect := D3DSWAPEFFECT_DISCARD;
  //   формат буфера не определен
  d3dParams.BackBufferFormat := d3dDisplayInfo.Format;

  // Создаем объект устройства
  Result := FD3D.CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
    Handle, D3DCREATE_SOFTWARE_VERTEXPROCESSING, @d3dParams,
    FD3DDevice);
end;

{*********************************************************************}
{** Освобождаем ресурсы                                             **}
{*********************************************************************}
procedure TMainForm.FreeD3D;
begin
  FD3DDevice := NIL;
  FD3D := NIL;
end;

{*********************************************************************}
{** Инициализация подсистемы при создании формы                     **}
{*********************************************************************}
procedure TMainForm.FormCreate(Sender: TObject);
begin
  if FAILED(InitD3D) then
  begin
    ShowMessage('Error initializing Direct3D...');
    Halt;
  end;
end;
```

```
{*********************************************************************}
{** Освобождаем ресурсы при завершении работы программы             **}
{*********************************************************************}
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  FreeD3D;
end;

{*********************************************************************}
{** Очищаем устройство                                              **}
{*********************************************************************}
function TMainForm.ClearDevice: HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если устройство не создано, то завершаем работу
  if FD3DDevice = NIL then EXIT;

  // Чистим устройство
  Result := FD3DDevice.Clear(0, NIL, D3DCLEAR_TARGET,
    D3DCOLOR_XRGB($FF, 0, $FF), 1, 0);
end;

{*********************************************************************}
{** Прорисовка сцены                                                **}
{*********************************************************************}
function TMainForm.RenderScene: HResult;
begin
  // Чистим устройство
  Result := ClearDevice;

  // Если произошла ошибка, то завершаем работу
  if FAILED(Result) then EXIT;
```

```
  // Переключение буферов
  FD3DDevice.Present(NIL, NIL, 0, NIL);
end;

{*****************************************************************}
{** Различные действия                                          **}
{*****************************************************************}
procedure TMainForm.applicationEventsMainIdle(Sender: TObject;
  var Done: Boolean);
begin
  if FIsActive then
  begin
    RenderScene;
  end;

  Done := FALSE;
end;

{*****************************************************************}
{** Сворачиваем приложение                                      **}
{*****************************************************************}
procedure TMainForm.applicationEventsMainMinimize(Sender: TObject);
begin
  FIsActive := FALSE;
end;

{*****************************************************************}
{** Восстанавливаем приложение                                  **}
{*****************************************************************}
procedure TMainForm.applicationEventsMainRestore(Sender: TObject);
begin
  FIsActive := TRUE;
end;

{*****************************************************************}
{** Приложение активно                                          **}
```

```
{*****************************************************************}
procedure TMainForm.FormActivate(Sender: TObject);
begin
  FIsActive := TRUE;
end;

END.
```

Полноэкранный режим

Помимо оконного режима, в Direct3D имеется и полноэкранный режим. Следующий пример из каталога Example_02 наглядно демонстрирует нам его использование. На самом деле программа этого примера делает даже больше — она позволяет переключаться между оконным и полноэкранным режимом работы (при помощи комбинации клавиш <Alt>+<Enter>).

У нас в примере появляется дополнительный флаг:

```
FWindowed: boolean;
```

Этот флаг и определяет режим работы приложения — оконный или полноэкранный. В связи с появлением новых возможностей изменится и ряд процедур.

В процедуре создания формы мы устанавливаем значение данного флага в FALSE, чтобы использовать изначально оконный режим работы (листинг 5.5).

Листинг 5.5. Процедура создания формы

```
{*****************************************************************}
{** Инициализаця подсистемы при создании формы                  **}
{*****************************************************************}
procedure TMainForm.FormCreate(Sender: TObject);
begin
  // По умолчанию наше приложение оконное
  FWindowed := TRUE;

  if FAILED(InitD3D) then
  begin
    ShowMessage('Error initializing Direct3D...');
```

```
    Halt;
  end;
end;
```

Изменилась процедура инициализации подсистемы Direct3D — появилась дополнительная функциональность, связанная с указанием текущего режима работы приложения (листинг 5.6).

Листинг 5.6. Процедура инициализации подсистемы Direct3D

```
{*********************************************************************}
{** Инициализация подсистемы Direct3D                                **}
{*********************************************************************}
function TMainForm.InitD3D: HResult;
var
  d3dDisplayInfo: TD3DDisplayMode;
  d3dParams: TD3DPresentParameters;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Создаем объект с интерфейсом IDirect3D9
  FD3D := Direct3DCreate9(D3D_SDK_VERSION);

  // Завершаем работу процедуры при ошибке
  if FD3D = NIL then EXIT;

  // Получаем установки текущего режима адаптера
  Result := FD3D.GetAdapterDisplayMode(D3DADAPTER_DEFAULT,
    d3dDisplayInfo);

  // Завершаем работу в случае сбоя
  if FAILED(Result) then EXIT;

  // Задаем параметры устройства:
  ZeroMemory(@d3dParams, SizeOf(d3dParams));
```

```
  //   какой режим использовать
  d3dParams.Windowed := FWindowed;
  //   режим переключения между буферами
  d3dParams.SwapEffect := D3DSWAPEFFECT_DISCARD;
  //   формат буфера не определен
  d3dParams.BackBufferFormat := d3dDisplayInfo.Format;

  // Если приложение полноэкранное, то...
  if not FWindowed then
  begin
    // ... используем текущие параметры режима адаптера
    d3dParams.FullScreen_RefreshRateInHz := d3dDisplayInfo.RefreshRate;
    d3dParams.BackBufferWidth  := d3dDisplayInfo.Width;
    d3dParams.BackBufferHeight := d3dDisplayInfo.Height;

    // Убираем заголовок окна и линии границ
    BorderStyle := bsNone;
  end

  // Если приложение оконное, то...
  else begin

    // Задаем обычный стиль окна
    BorderStyle := bsSizeable;
  end;

  // Создаем объект устройства
  Result := FD3D.CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
    Handle, D3DCREATE_SOFTWARE_VERTEXPROCESSING, @d3dParams,
    FD3DDevice);
end;
```

Появилась процедура обработки нажатия клавиш (листинг 5.7).

Листинг 5.7. Процедура обработки нажатия клавиш

```
{*****************************************************************}
{** Обработка нажатия клавиш                                    **}
{*****************************************************************}
procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  // Нажатие <ALT>+<Enter> приводит к смене режима
  if (ssAlt in Shift) and (Key = VK_RETURN) then
  begin
    FWindowed := not FWindowed;
    FreeD3D;
    InitD3D;
  end;
end;
```

Фактически для работы в полноэкранном режиме мы всего лишь изменили несколько параметров в структуре `TD3DPresentParameters`, а точнее указали, что не будем использовать оконное приложение, задали частоту обновления экрана в полноэкранном режиме, указали ширину и высоту нужного нам режима и создали соответствующее устройство. Изменяется также и стиль границ окна нашего приложения: в полноэкранном режиме мы убираем заголовок формы и линии границ, а в оконном приложении восстанавливаем обычный стиль.

Начиная с этого примера, все последующие будут иметь возможность переключения между оконным и полноэкранным режимами работы.

Потеря устройства

Поэкспериментируйте со вторым примером. Попереключайте режимы работы. А затем попробуйте включить спящий режим работы и выйти из него, либо просто переключитесь на другое приложение из полноэкранного режима работы. Устройство будет потеряно и воспроизведение станет невозможным.

Проверить текущий статус устройства можно с помощью метода `IDirect3DDevice9.TestCooperativeLevel`:

```
function TestCooperativeLevel: HResult; stdcall;
```

В случае успеха данный метод вернет значение D3D_OK, а в случае неудачи — одно из следующих значений:

- D3DERR_DEVICELOST — устройство потеряно, и его нельзя восстановить в данный момент. Воспроизведение невозможно;
- D3DERR_DEVICENOTRESET — устройство потеряно, но может быть восстановлено.

Соответственно, в примере из каталога Example_03 реализована такая проверка и восстановление устройства. Фактически под восстановлением устройства можно понимать полную повторную инициализацию подсистемы. Цикл ожидания теперь выглядит так, как представлено в листинге 5.8.

Листинг 5.8. Процедура цикла ожидания

```
{*********************************************************************}
{** Различные действия                                              **}
{*********************************************************************}
procedure TMainForm.applicationEventsMainIdle(Sender: TObject;
  var Done: Boolean);
var
  hr: HResult;
begin
  if FIsActive then
  begin
    // Получаем статус устройства
    hr := FD3DDevice.TestCooperativeLevel;

    // Если устройство не готово, то завершаем работу
    if hr = D3DERR_DEVICELOST then EXIT;

    // Если устройство можно восстановить, то восстанавливаем
    if hr = D3DERR_DEVICENOTRESET then
    begin
      FreeD3D;
      InitD3D;
    end;
```

```
    // Прорисовываем сцену
    RenderScene;
  end;

  Done := FALSE;
end;
```

Примитивы

Вот и добрались мы, наконец-то, до самого процесса рисования. В Direct3D оно происходит при помощи примитивов (как и в DirectDraw, здесь нет каких-либо заготовок, точнее, они имеются в пакете специальных утилит D3DX, но об этом будет рассказано позже). К основным примитивам можно отнести точки, линии и треугольники. А полный список примитивов таков:

- точки (Point Lists);
- линии (Line Lists);
- последовательность линий (Line Strips);
- треугольники (Triangle List);
- последовательность треугольников (Triangle Strip);
- последовательность треугольников с общей вершиной (Triangle Fan).

Буфер вершин

Каждый примитив состоит из набора вершин: из одной вершины состоит точка, из двух вершин состоит отрезок и из трех треугольник. Список вершин должен храниться в специальном буфере — *буфере вершин* (Vertex Buffer).

Работа с буфером вершин осуществляется при помощи интерфейса `IDirect3DVertexBuffer9`. Создается буфер вершин методом `IDirect3DDevice9.CreateVertexBuffer`:

```
function CreateVertexBuffer(
  Length: LongWord;
  Usage,
  FVF: DWord;
  Pool: TD3DPool;
```

```
  out ppVertexBuffer: IDirect3DVertexBuffer9;
  pSharedHandle: PHandle):
HResult; stdcall;
```

Здесь:

- `Length` — размер буфера в байтах;
- `Usage` — указывает на способ использования ресурсов. Может принимать нулевое значение;
- `FVF` — комбинация одного или нескольких флагов:
  - `D3DFVF_DIFFUSE` — формат вершин содержит диффузную цветовую составляющую;
  - `D3DFVF_NORMAL` — в формате вершин присутствует вектор нормали. Флаг не может использоваться совместно с флагом `D3DFVF_XYZRHW`;
  - `D3DFVF_PSIZE` — размер точки присутствует в формате вершины;
  - `D3DFVF_SPECULAR` — формат вершин содержит отражающую цветовую составляющую;
  - `D3DFVF_XYZ` — формат вершины включает позицию нетрансформированной вершины;
  - `D3DFVF_XYZRHW` — формат вершины включает позицию трансформированной вершины. Флаг не может использоваться совместно с флагами `D3DFVF_NORMAL` и `D3DFVF_XYZ`;
  - `D3DFVF_XYZB1` через `D3DFVF_XYZB5` — формат включает позицию и вес для использования в операциях смешивания;
  - `D3DFVF_XYZW` — формат вершин содержит *x*-, *y*-, *z*-, *w*-координаты. Данная константа используется только в программируемом конвейере вершин;
- `Pool` — класс памяти для ресурсов:
  - `D3DPOOL_DEFAULT` — ресурсы будут размещаться в локальной видеопамяти либо в AGP;
  - `D3DPOOL_MANAGED` — подсистема Direct3D управляет размещением ресурса между системной памятью и видеопамятью;
  - `D3DPOOL_SYSTEMMEM` — ресурс размещается в системной памяти;
  - `D3DPOOL_SCRATCH` — ресурс размещается в системной памяти и не будет доступен Direct3D;
- `ppVertexBuffer` — адрес переменной, в которую будет передан интерфейс созданного буфера вершин;

- `pSharedHandle` — зарезервировано. Должно использоваться нулевое значение.

Для работы с буфером вершин нам потребуется два метода. Первый метод — это блокировка доступа к вершинам и получение указателя на область памяти буфера `IDirect3DVertexBuffer9.Lock`:

```
function Lock(
  OffsetToLock,
  SizeToLock: LongWord;
  out ppbData: Pointer;
  Flags: DWord):
HResult; stdcall;
```

Здесь:

- `OffsetToLock` — смещение от начала буфера в байтах;
- `SizeToLock` — размер блока данных в байтах;
- `ppbData` — указатель, который будет содержать ссылку на область памяти данных буфера;
- `Flags` — способ доступа к данным в буфере. Может принимать нулевое значение.

Второй метод позволяет выполнять разблокировку буфера вершин `IDirect3DVertexBuffer9.Unlock`:

```
function Unlock: HResult; stdcall;
```

Точка

Четвертый пример, из каталога Examples_04, выводит точку в центр экрана. Фон закрашивается черным цветом. Рассмотрим подробнее наши действия.

Любая точка, так или иначе, представляет собой вершину, и мы должны задать ее тип:

```
TCustomVertex = packed record
  x, y, z, rwh: Single;
end;
```

Данная структура описывает положение точки в пространстве, а также указывает на то, что мы используем преобразованные координаты и точка является освещенной.

Соответственно, константа, описывающая формат точек, будет иметь следующий вид:

```
D3DFVF_CUSTOMVERTEX = D3DFVF_XYZRHW;
```

Первым делом после инициализации Direct3D мы должны заполнить буфер вершин. Для этого мы добавляем процедуру `PrepareVertices` (листинг 5.9).

Листинг 5.9. Заполнение буфера вершин

```
{*********************************************************************}
{** Подготовка набора вершин                                        **}
{*********************************************************************}
function TMainForm.PrepareVertices: HResult;
var
  Vertices: TCustomVertex;
  pVertices: pointer;
  d3dViewport: TD3DViewport9;
begin
  // Создаем буфер вершин
  Result := FD3DDevice.CreateVertexBuffer(SizeOf(Vertices), 0,
    D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT, FD3DVertexBuffer, NIL);

  // При невозможности создания завершаем работу
  if FAILED(Result) then EXIT;

  // Получаем параметры окна вывода
  FD3DDevice.GetViewport(d3dViewport);

  // Установка параметров вершины
  ZeroMemory(@Vertices, SizeOf(Vertices));
  Vertices.x := d3dViewport.Width  div 2;
  Vertices.y := d3dViewport.Height div 2;
  Vertices.rwh := 1;

  // Блокируем доступ к буферу вершин
  Result := FD3DVertexBuffer.Lock(0, SizeOf(Vertices),
    pVertices, 0);
  if FAILED(Result) then EXIT;
```

```
  try
    // Копируем в буфер данные
    CopyMemory(pVertices, @Vertices, SizeOf(Vertices));
  finally
    // Разблокировка буфера
    FD3DVertexBuffer.Unlock;
  end;
end;
```

Вершину мы располагаем по центру окна вывода, параметры которого получаем методом `IDirect3DDevice9.GetViewport`:

```
function GetViewport(
  out pViewport: TD3DViewport9):
HResult; stdcall;
```

Здесь `pViewport` — структура `TD3DViewport9`, в которую будут записаны текущие параметры.

Структура `TD3DViewport9` содержит следующие поля:

- `X`, `Y` — координаты верхнего левого угла окна вывода;
- `Width`, `Height` — ширина и высота окна вывода;
- `MinZ`, `MaxZ` — параметры глубины.

Процедура прорисовки сцены также претерпела определенные изменения (листинг 5.10).

Листинг 5.10. Процедура прорисовки сцены

```
{*********************************************************************}
{** Прорисовка сцены                                                **}
{*********************************************************************}
function TMainForm.RenderScene: HResult;
begin
  // Чистим устройство
  Result := ClearDevice;

  // Если произошла ошибка, то завершаем работу
  if FAILED(Result) then EXIT;
```

```
  // Начало сцены
  FD3DDevice.BeginScene;
  try
    // Связываем буфер вершин с потоком данных устройства
    FD3DDevice.SetStreamSource(0, FD3DVertexBuffer, 0,
      SizeOf(TCustomVertex));

    // Устанавливаем формат вершин
    FD3DDevice.SetFVF(D3DFVF_CUSTOMVERTEX);

    // Рисуем примитив
    FD3DDevice.DrawPrimitive(D3DPT_POINTLIST, 0, 1);
  finally

    // Завершаем сцену
    FD3DDevice.EndScene;
  end;

  // Переключение буферов
  FD3DDevice.Present(NIL, NIL, 0, NIL);
end;
```

Прорисовка сцены начинается с вызова `IDirect3DDevice9.BeginScene`:

```
function BeginScene: HResult; stdcall;
```

Фактически данный метод обозначает начало блока прорисовки сцены. В конце блока необходимо вызвать метод `IDirect3DDevice9.EndScene`:

```
function EndScene: HResult; stdcall;
```

Объявив начало сцены, первым делом мы связываем буфер вершин с потоком данных устройства методом `IDirect3DDevice9.SetStreamSource`:

```
function SetStreamSource(
  StreamNumber: LongWord;
  pStreamData: IDirect3DVertexBuffer9;
  OffsetInBytes,
  Stride: LongWord):
HResult; stdcall;
```

Здесь:

- `StreamNumber` — номер потока, начиная с нулевого;
- `pStreamData` — указатель на интерфейс буфера вершин;
- `OffsetInBytes` — смещение в байтах от начала потока;
- `Stride` — шаг в байтах.

И после этого устанавливаем формат вершин при помощи метода `IDirect3DDevice9.SetFVF`:

```
function SetFVF(
  FVF: DWORD):
HResult; stdcall;
```

Здесь `FVF` — битовая комбинация флагов (флаги представлены ранее в описании параметра `FVF` метода `IDirect3DDevice9.CreateVertexBuffer`).

Для рисования примитивов предназначен метод `IDirect3DDevice9.DrawPrimitive`:

```
function DrawPrimitive(
  PrimitiveType: TD3DPrimitiveType;
  StartVertex,
  PrimitiveCount: LongWord):
HResult; stdcall;
```

Здесь:

- `PrimitiveType` — тип примитивов:
  - `D3DPT_POINTLIST` — точки;
  - `D3DPT_LINELIST` — линии;
  - `D3DPT_LINESTRIP` — последовательность линий;
  - `D3DPT_TRIANGLELIST` — треугольники;
  - `D3DPT_TRIANGLESTRIP` — последовательность треугольников;
  - `D3DPT_TRIANGLEFAN` — последовательность треугольников с общей вершиной;
- `StartVertex` — индекс первой загружаемой вершины;
- `PrimitiveCount` — число примитивов.

После запуска появится окно, в центре которого будет светиться точка (рис. 5.2).

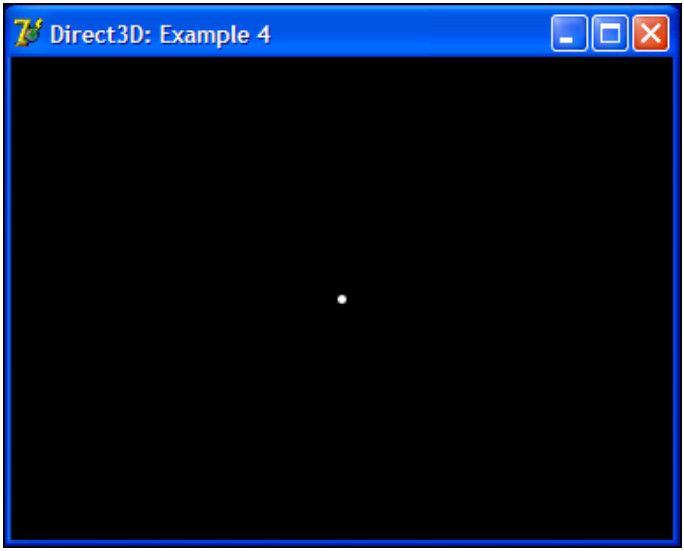


Рис. 5.2. Рисуем точку

Цвет

Точку мы нарисовали, теперь давайте ее раскрасим. Для закраски необходимо кое-что изменить в наших структурах. В структуру `TCustomVertex` добавляется новый параметр `color`:

```
TCustomVertex = packed record
  x, y, z, rwh: Single;
  color: DWORD;
end;
```

А в константу, описывающую формат наших вершин, включаем параметр, указывающий на поддержку диффузного освещения:

```
D3DFVF_CUSTOMVERTEX = D3DFVF_XYZRHW or D3DFVF_DIFFUSE;
```

И последнее, что мы делаем, — это задаем цвет вершины:

```
Vertices.color := $000000FF;
```

Теперь у нас на экране появится синяя точка. Пример находится в папке Example_05.

Пример анимации

Усложним задачу и заставим наши вершины двигаться. Пример из каталога Example_06 нам это наглядно демонстрирует (рис. 5.3).

Рис. 5.3. Пример анимации

Вершины передвигаются слева направо. Максимальное количество вершин задается в виде константы:

```
MAX_VERTICES = 500;
```

Процедура заполнения вершин теперь рассчитана на массив вершин (листинг 5.11), а не на одну, как в прошлом примере.

Листинг 5.11. Заполнение буфера вершин для анимации точек

```
{*********************************************************************}
{** Подготовка набора вершин                                        **}
{*********************************************************************}
function TMainForm.PrepareVertices: HResult;
var
  Vertices: array[0..MAX_VERTICES - 1] of TCustomVertex;
  pVertices: pointer;
```

```
  d3dViewport: TD3DViewport9;
  I: integer;
begin
  // Создаем буфер вершин
  Result := FD3DDevice.CreateVertexBuffer(SizeOf(Vertices), 0,
    D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT, FD3DVertexBuffer, NIL);

  // При невозможности создания завершаем работу
  if FAILED(Result) then EXIT;

  // Получаем параметры окна вывода
  FD3DDevice.GetViewport(d3dViewport);

  Randomize;

  // Установка параметров вершин
  for I := 0 to MAX_VERTICES - 1 do
  begin
    Vertices[I].x := Random(d3dViewport.Width);
    Vertices[I].y := Random(d3dViewport.Height);
    Vertices[I].rwh := 1;
    Vertices[I].color := Random($00FFFFFF);
  end;

  // Блокируем доступ к буферу вершин
  Result := FD3DVertexBuffer.Lock(0, SizeOf(Vertices),
    pVertices, 0);
  if FAILED(Result) then EXIT;

  try
    // Копируем в буфер данные
    CopyMemory(pVertices, @Vertices, SizeOf(Vertices));
  finally
    // Разблокировка буфера
    FD3DVertexBuffer.Unlock;
  end;
end;
```

Перемещение вершин происходит в методе `MoveVertices` (листинг 5.12).

Листинг 5.12. Метод `MoveVertices`

```
{*********************************************************************}
{** Изменение положения вершин                                      **}
{*********************************************************************}
function TMainForm.MoveVertices: HResult;
var
  Vertices: array[0..MAX_VERTICES - 1] of TCustomVertex;
  pVertices: pointer;
  d3dViewport: TD3DViewport9;
  I: integer;
begin
  // Получаем параметры окна вывода
  FD3DDevice.GetViewport(d3dViewport);

  // Блокируем доступ к буферу вершин
  Result := FD3DVertexBuffer.Lock(0, SizeOf(Vertices),
    pVertices, 0);
  if FAILED(Result) then EXIT;

  try
    ZeroMemory(@Vertices, SizeOf(Vertices));

    // Копируем вершины во временный буфер
    CopyMemory(@Vertices, pVertices, SizeOf(Vertices));

    // Меняем положение вершин
    for I := 0 to MAX_VERTICES - 1 do
    begin
      Vertices[I].x := Vertices[I].x + 1;
      if Vertices[I].x > d3dViewport.Width then
      begin
        Vertices[I].x := -1;
        Vertices[I].y := Random(d3dViewport.Height);
```

```
        Vertices[I].color := Random($00FFFFFF);
      end;
    end;

    // Копируем вершины в буфер
    CopyMemory(pVertices, @Vertices, SizeOf(Vertices));
  finally

    // Разблокировка буфера
    FD3DVertexBuffer.Unlock;
  end;
end;
```

Принцип работы этой процедуры очень прост: копируем содержимое буфера вершин во временный буфер, меняем координату *x* каждой вершины и записываем данные обратно в буфер. При достижении вершиной правой границы мы ее обнуляем и задаем вершине новое положение высоты и новое значение цвета. За счет этого достигается плавность и целостность нашей картины.

Линии и последовательность линий

Примеры из каталогов Example_07 и Example_08 демонстрируют нам работу с линиями и последовательностями линий. Для большей наглядности я решил сделать примеры анимированными, как и прошлый пример. Концы линий в примерах отскакивают от границ экрана.

Значения смещений по осям *x* и *y* для каждой вершины хранятся в массиве `FDeltaPoints`:

```
FDeltaPoints: array[0..MAX_VERTICES] of TPoint;
```

Смещения задаются один раз в процедуре подготовки набора вершин:

```
// Заполняем массив смещений для наших вершин
ZeroMemory(@FDeltaPoints, SizeOf(FDeltaPoints));
for I := 0 to MAX_VERTICES - 1 do
begin
  FDeltaPoints[I].X := Random(5) + 1;
  FDeltaPoints[I].Y := Random(5) + 1;
```

```
  // Случайным образом указываем направление
  if Random(100) mod 2 = 0 then
     FDeltaPoints[I].X := -FDeltaPoints[I].X;
  if Random(100) mod 2 = 0 then
     FDeltaPoints[I].Y := -FDeltaPoints[I].Y;
end;
```

Положение вершин меняется следующим образом:

```
// Меняем положение вершин
for I := 0 to MAX_VERTICES - 1 do
begin
  Vertices[I].x := Vertices[I].x + FDeltaPoints[I].X;
  Vertices[I].y := Vertices[I].y + FDeltaPoints[I].Y;

  if (Vertices[I].x < 0) or (Vertices[I].x > d3dViewport.Width) then
  begin
    FDeltaPoints[I].X := -FDeltaPoints[I].X;
    Vertices[I].x := Vertices[I].x + FDeltaPoints[I].X;
  end;

  if (Vertices[I].y < 0) or (Vertices[I].y > d3dViewport.Height) then
  begin
    FDeltaPoints[I].Y := -FDeltaPoints[I].Y;
    Vertices[I].y := Vertices[I].y + FDeltaPoints[I].Y;
  end;
end;
```

Соответственно изменился и механизм прорисовки примитивов — для рисования линий мы используем метод `IDirect3DDevice9.DrawPrimitive` с параметром `D3DPT_LINELIST` и количеством примитивов в два раза меньшим общего числа вершин:

```
FD3DDevice.DrawPrimitive(D3DPT_LINELIST, 0, MAX_VERTICES div 2);
```

На рис. 5.4 показана работа первого примера с линиями.

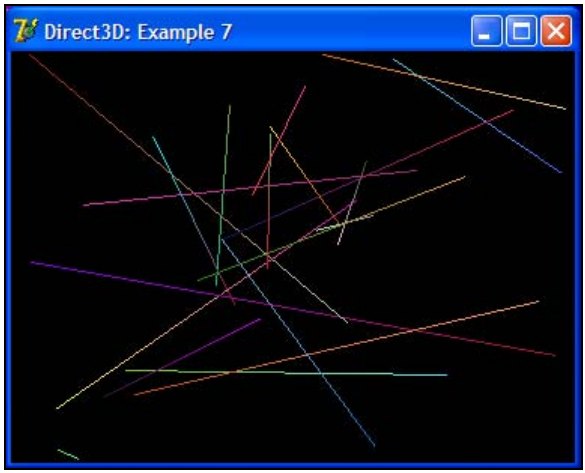


Рис. 5.4. Рисуем линии

Если же нам нужно нарисовать не отдельно взятые линии, а целую последовательность, то достаточно изменить параметр `D3DPT_LINELIST` в методе вывода примитивов на `D3DPT_LINESTRIP` и указать число линий на единицу, меньшую общего числа вершин:

```
FD3DDevice.DrawPrimitive(D3DPT_LINESTRIP, 0, MAX_VERTICES - 1);
```

В итоге мы увидим на экране то, что представлено на рис. 5.5.

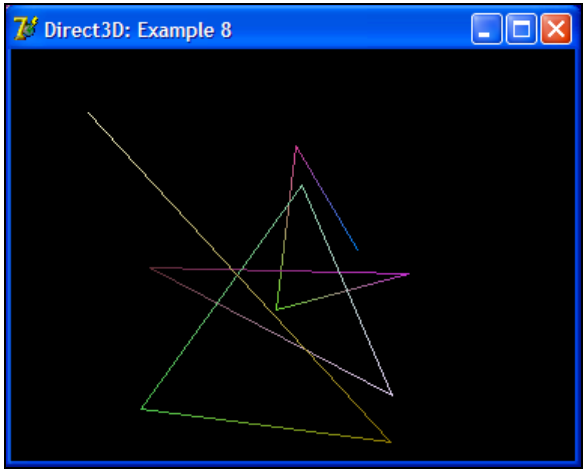


Рис. 5.5. Рисуем последовательность линий

Треугольник и последовательности треугольников

От рисования линий перейдем к рисованию треугольников. Процесс их вывода не сильно отличается от процесса вывода таких примитивов, как точка и линия. Основное отличие — это число вершин.

Для вывода треугольника мы зададим его вершины в виде констант в процедуре подготовки вершин (листинг 5.13).

Листинг 5.13. Заполнение буфера вершин для рисования треугольников

```
{*********************************************************************}
{** Подготовка набора вершин                                        **}
{*********************************************************************}
function TMainForm.PrepareVertices: HResult;
const
  Vertices: array[0..2] of TCustomVertex = (
    (x:  50; y:  50; z: 0.5; rwh: 1; color: $000000FF),
    (x: 250; y: 250; z: 0.5; rwh: 1; color: $0000FF00),
    (x:  50; y: 250; z: 0.5; rwh: 1; color: $00FF0000)
  );
var
  pVertices: pointer;
begin
  // Создаем буфер вершин
  Result := FD3DDevice.CreateVertexBuffer(SizeOf(Vertices), 0,
    D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT, FD3DVertexBuffer, NIL);

  // При невозможности создания завершаем работу
  if FAILED(Result) then EXIT;

  // Блокируем доступ к буферу вершин
  Result := FD3DVertexBuffer.Lock(0, SizeOf(Vertices),
    pVertices, 0);
  if FAILED(Result) then EXIT;
```

```
  try
    // Копируем в буфер данные
    CopyMemory(pVertices, @Vertices, SizeOf(Vertices));
  finally
    // Разблокировка буфера
    FD3DVertexBuffer.Unlock;
  end;
end;
```

Метод `IDirect3DDevice9.DrawPrimitive` должен быть вызван с параметром `D3DPT_TRIANGLELIST` и количеством примитивов равным одному, т. к. будет рисоваться один треугольник:

```
FD3DDevice.DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);
```

Результат работы примера из каталога Example_09 приведен на рис. 5.6.

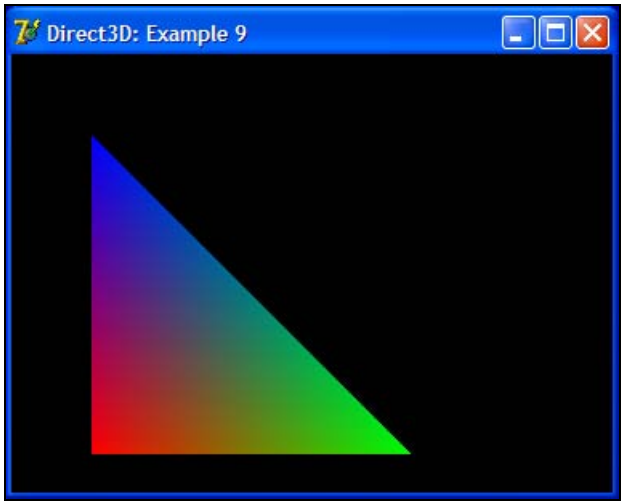


Рис. 5.6. Треугольник

Последовательность треугольников видна на рис. 5.7.

В этом примере (из каталога Examples_10) мы видим последовательность смежных треугольников, имеющих общую грань. Для их рисования использовался режим вывода примитивов `D3DPT_TRIANGLESTRIP`:

```
FD3DDevice.DrawPrimitive(D3DPT_TRIANGLESTRIP, 0, 6);
```

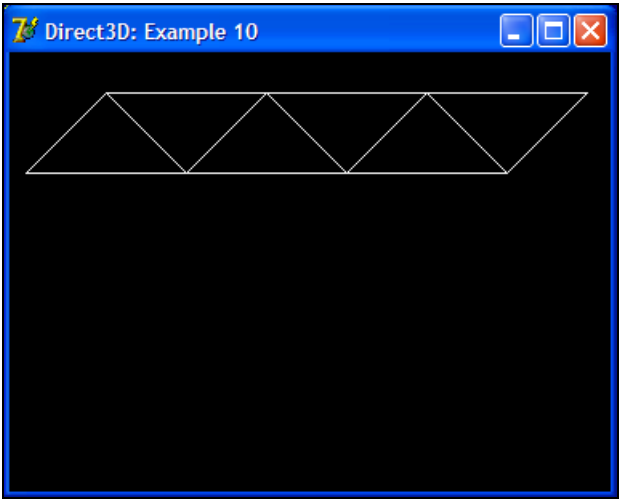


Рис. 5.7. Последовательность треугольников

Как вы уже наверно успели заметить, треугольники в данном режиме не закрашены, и мы отчетливо можем увидеть каркас фигуры, ими составленной. Это достигается путем включения режима прорисовки каркаса в процессе вывода сцены:

```
FD3DDevice.SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);
```

А сами вершины задаются в процедуре `PrepareVertices` (листинг 5.14).

Листинг 5.14. Заполнение буфера вершин для рисования последовательности треугольников

```
{*********************************************************************}
{** Подготовка набора вершин                                        **}
{*********************************************************************}
function TMainForm.PrepareVertices: HResult;
var
  Vertices: array[0..7] of TCustomVertex;
  pVertices: pointer;
  I: integer;
begin
  // Создаем буфер вершин
```

```
  Result := FD3DDevice.CreateVertexBuffer(SizeOf(Vertices), 0,
    D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT, FD3DVertexBuffer, NIL);

  // При невозможности создания завершаем работу
  if FAILED(Result) then EXIT;

  Randomize;

  // Установка параметров вершин
  ZeroMemory(@Vertices, SizeOf(Vertices));
  for I := 0 to 7 do
  begin
    Vertices[I].x := I * 50 + 10;
    Vertices[I].y := ((I + 1) mod 2) * 50 + 25;
    Vertices[I].rwh := 1;
    Vertices[I].color := $00FFFFFF;
  end;

  // Блокируем доступ к буферу вершин
  Result := FD3DVertexBuffer.Lock(0, SizeOf(Vertices),
    pVertices, 0);
  if FAILED(Result) then EXIT;

  try
    // Копируем в буфер данные
    CopyMemory(pVertices, @Vertices, SizeOf(Vertices));
  finally
    // Разблокировка буфера
    FD3DVertexBuffer.Unlock;
  end;
end;
```

Нами остался нерассмотренным последний режим вывода примитивов — `D3DPT_TRIANGLEFAN`. Этот режим позволяет рисовать последовательность треугольников с одним общим центром.

Пример из каталога Example_11 наглядно нам это демонстрирует (рис. 5.8).

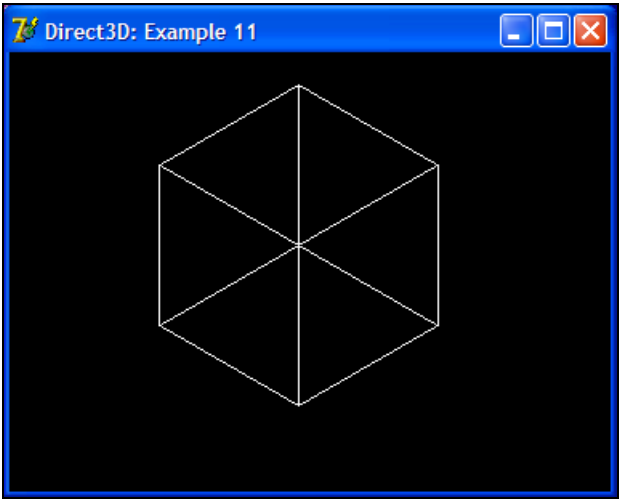


Рис. 5.8. Последовательность треугольников с общим центром

Первая вершина располагается по центру экрана, а остальные идут по часовой стрелке вокруг нее (листинг 5.15).

Листинг 5.15. Заполнение буфера вершин для рисования последовательности треугольников с общим центром

```
{*********************************************************************}
{** Подготовка набора вершин                                        **}
{*********************************************************************}
function TMainForm.PrepareVertices: HResult;
var
  Vertices: array[0..7] of TCustomVertex;
  pVertices: pointer;
  I: integer;
begin
  // Создаем буфер вершин
  Result := FD3DDevice.CreateVertexBuffer(SizeOf(Vertices), 0,
    D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT, FD3DVertexBuffer, NIL);

  // При невозможности создания завершаем работу
  if FAILED(Result) then EXIT;
```

```
  Randomize;

  // Установка параметров вершин
  ZeroMemory(@Vertices, SizeOf(Vertices));
  Vertices[0].x := 180;
  Vertices[0].y := 120;
  Vertices[0].rwh := 1;
  Vertices[0].color := $00FFFFFF;

  for I := 0 to 5 do
  begin
    Vertices[I + 1].x := sin(I * 60 * PI / 180) * 100 + 180;
    Vertices[I + 1].y := -cos(I * 60 * PI / 180) * 100 + 120;
    Vertices[I + 1].rwh := 1;
    Vertices[I + 1].color := $00FFFFFF;
  end;

  Vertices[7] := Vertices[1];

  // Блокируем доступ к буферу вершин
  Result := FD3DVertexBuffer.Lock(0, SizeOf(Vertices),
    pVertices, 0);
  if FAILED(Result) then EXIT;

  try
    // Копируем в буфер данные
    CopyMemory(pVertices, @Vertices, SizeOf(Vertices));
  finally
    // Разблокировка буфера
    FD3DVertexBuffer.Unlock;
  end;
end;
```

От треугольника к прямоугольнику

Научившись оперировать основными типами примитивов, давайте научимся строить из них прямоугольник. А точнее, построим прямоугольник из двух треугольников (рис. 5.9).

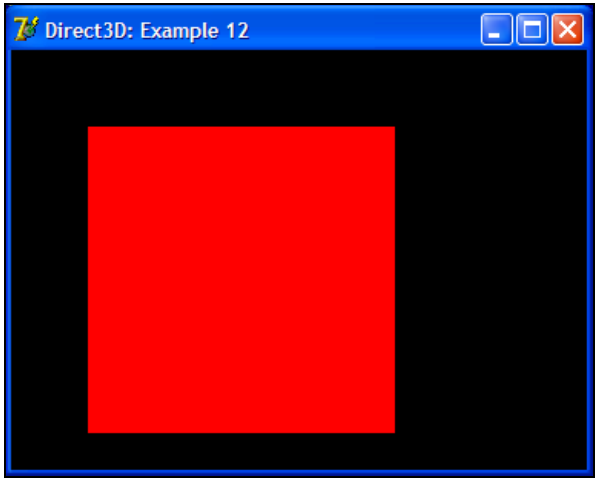


Рис. 5.9. Квадрат

Пример из каталога Example_12 показывает построение прямоугольника (квадрата) из двух треугольников. Залив треугольники одинаковым цветом, мы не видим их границы, и у нас создается впечатление одной целостной фигуры.

Построения в пространстве

Все наши построения до сих пор проводились на плоскости, а не в пространстве. И как бы красиво не выглядела наша сцена, она все равно нарисована на плоскости. В изученных нами примерах мы использовали значение `FVF`, равное `D3DFVF_XYZRHW`, при создании устройства. Это означало использование преобразованных координат. В работе с трехмерной графикой мы будем использовать непреобразованные координаты (флаг `D3DFVF_XYZ`), и координата *z*, до сих пор нами игнорируемая, окажется очень даже используемой.

Так или иначе, все операции с объектами сцены и самой сценой базируются на понятии *"матрица"*. Вращение объекта по различным осям, масштабиро-

вание, перемещение и любые другие операции построены именно на них. Так что давайте начнем изучение пространственных построений с матриц.

Матрицы

Любая точка пространства задается тремя координатами (*x*, *y*, *z*). Мы же будем использовать запись не из трех составляющих, а из четырех. Соответственно, координата в пространстве будет определяться четверкой чисел (*x*, *y*, *z*, *w*), где *w* — некий весовой коэффициент, на который должны перемножаться координаты точки при проецировании.

Матрицей размера *m*×*n* называется прямоугольная таблица чисел, которая содержит *m* строк и *n* столбцов. Частным случаем матрицы можно считать отдельную строку чисел или столбец. Например, вектор является частным случаем матрицы.

Матрицы можно умножать на число, перемножать между собой и транспонировать, можно находить определитель матрицы и матрицу, обратную исходной.

Сложение матриц и умножение на число

Складывать матрицы можно только тогда, когда их размеры одинаковы. Пусть у нас есть две матрицы — **A** и **B** размера *m*×*n*. Матрица **C**, являющаяся суммой этих матриц, будет вычисляться по формуле $c_{ij} = a_{ij} + b_{ij}$, где $i = 1, 2, ..., m$; $j = 1, 2, ..., n$:

$$\begin{pmatrix} 2 & 1 & 9 & 6 \\ 0 & 5 & 6 & 0 \\ 9 & 3 & 2 & 6 \\ 2 & 5 & 3 & 1 \end{pmatrix} + \begin{pmatrix} 5 & 1 & 0 & 5 \\ 7 & 0 & 1 & 4 \\ 3 & 3 & 8 & 2 \\ 7 & 1 & 9 & 1 \end{pmatrix} = \begin{pmatrix} 7 & 2 & 9 & 11 \\ 7 & 5 & 7 & 4 \\ 12 & 6 & 10 & 8 \\ 9 & 6 & 12 & 2 \end{pmatrix}.$$

Матрица **C**, являющаяся произведением матрицы **A** размера *m*×*n* на число λ, будет вычисляться следующим образом: $c_{ij} = \lambda \times a_{ij}$, где $i = 1, 2, ..., m$; $j = 1, 2, ..., n$. Пример:

$$7 \times \begin{pmatrix} 7 & 2 & 9 & 11 \\ 7 & 5 & 7 & 4 \\ 12 & 6 & 10 & 8 \\ 9 & 6 & 12 & 2 \end{pmatrix} = \begin{pmatrix} 49 & 14 & 63 & 77 \\ 49 & 35 & 49 & 28 \\ 84 & 42 & 70 & 56 \\ 63 & 42 & 84 & 14 \end{pmatrix}.$$

Перемножение матриц

Матрицы можно перемножать между собой только в том случае, если число столбцов первой матрицы совпадает с числом строк второй. Это означает, что можно перемножать лишь матрицы размером $m \times n$ на матрицы размером $n \times k$. Результатом такого умножения будет матрица размером $m \times k$:

$$c_{ij} = \sum_{s=1}^{n} a_{is} \times b_{sj} \text{ , где } i = 1, 2, ..., m; j = 1, 2, ..., k.$$

Пример:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \times \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \end{pmatrix} = \begin{pmatrix} 1 \times 9 + 2 \times 6 & 1 \times 8 + 2 \times 5 & 1 \times 7 + 2 \times 4 \\ 3 \times 9 + 4 \times 6 & 3 \times 8 + 4 \times 5 & 3 \times 7 + 4 \times 4 \\ 5 \times 9 + 6 \times 6 & 5 \times 8 + 6 \times 5 & 5 \times 7 + 6 \times 4 \end{pmatrix} = \begin{pmatrix} 21 & 18 & 15 \\ 51 & 44 & 37 \\ 81 & 70 & 59 \end{pmatrix}.$$

Умножение матриц будет использовано нами в дальнейшем при объединении различных эффектов. В качестве примера можно привести одновременное вращение объекта по нескольким осям и его масштабирование.

Единичная матрица

Матрица, все элементы которой, кроме элементов главной диагонали (которые являются единицами), равны нулю, называется *единичной матрицей*. Пример такой матрицы:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Другое название этой матрицы — *матрица идентичности*. Она будет применяться нами в дальнейшем для расчета различных преобразований.

Матрицы переноса (сдвига)

Для переноса по осям x, y и z мы, само собой, будем также пользоваться матрицами. Матрица переноса по оси x имеет следующий вид:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ A & 0 & 0 & 1 \end{pmatrix}.$$

Матрица переноса по оси *y* выглядит следующим образом:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & B & 0 & 1 \end{pmatrix}.$$

И, наконец, матрица переноса по оси *z* следующая:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & C & 1 \end{pmatrix}.$$

Можно также привести пример и матрицы сдвига по всем трем осям одновременно:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ A & B & C & 1 \end{pmatrix}.$$

Таким образом, перенос точки (x, y, z, w) в новое положение можно выразить следующим образом: $(x_1, y_1, z_1, w_1) = (x + A \times w, y + B \times w, z + C \times w, w)$. Данный пример показывает, что умножение вектора координат на матрицу сдвига приводит к перемещению объекта в новое положение в пространстве.

Матрицы вращения

По аналогии с предыдущим примером давайте рассмотрим и матрицы вращения. Матрица вращения по оси *x* на угол α задается следующим образом:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Для поворота на угол β вокруг оси *y* нам потребуется следующая матрица:

$$\begin{pmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

И, наконец, поворот на угол γ вокруг оси z можно осуществить при помощи этой матрицы:

$$\begin{pmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Матрица масштабирования

Изменение масштаба производится при помощи следующей матрицы:

$$\begin{pmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Здесь α, β и γ — коэффициенты сжатия вдоль осей x, y и z соответственно.

Матрицы отражения

И последний вид матриц преобразования, который мы рассмотрим, — это матрицы отражения. Матрица отражения от плоскости xy:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Отражение от плоскости yz:

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

И матрица отражения от плоскости zx:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Типы матриц Direct3D

Обычно приложения, создающие трехмерные сцены, используют одну из двух типов декартовых систем координат — левостороннюю (рис. 5.10) или правостороннюю (рис. 5.11).

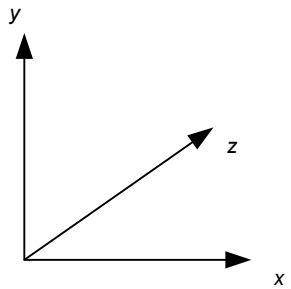


Рис. 5.10. Левосторонняя система координат

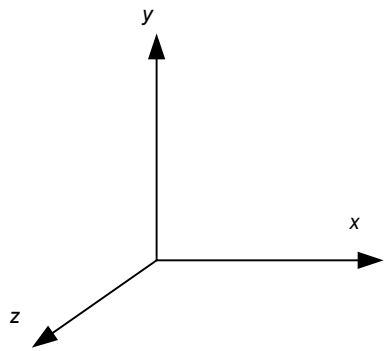


Рис. 5.11. Правосторонняя система координат

Ось *x* в этих системах направлена вправо, ось *y* вверх, а направление оси *z* и определяет тип системы координат. В Direct3D используется левосторонняя система координат.

Для работы с трехмерной сценой в подсистеме определены три основные типа матриц:

- мировая матрица (World Matrix);
- матрица вида (View Matrix);
- матрица проекции (Projection Matrix).

Мировая матрица описывает локальную для объекта систему координат. Все операции с объектом, такие как масштабирование, перемещение и вращение, будут фактически производиться с мировой матрицей объекта.

На рис. 5.12 наглядно представлены отношения между мировыми координатами и локальными для конкретного объекта:

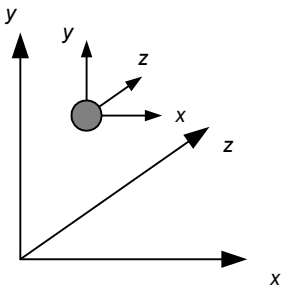


Рис. 5.12. Локальная координатная модель

Матрица вида соответствует взгляду наблюдателя на сцену, а это означает, что она задает позицию и направление просмотра (рис. 5.13). Эту матрицу можно трансформировать точно так же, как и мировую матрицу — двигать, вращать и т. д.

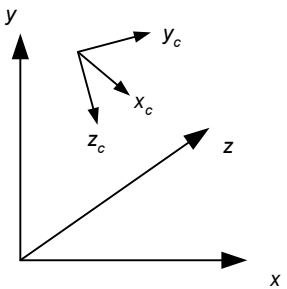


Рис. 5.13. Положение камеры в пространстве

Матрица проекции предназначена для построения проекции трехмерной сцены на экране (рис. 5.14). Это означает, что она задает правила проецирования, такие как местоположение передней и задней плоскости отсечения и перспектива.

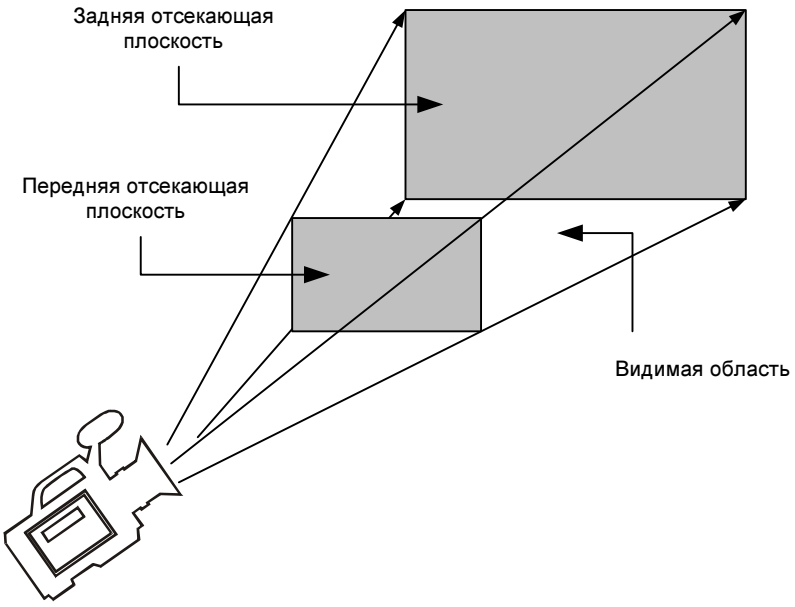


Рис. 5.14. Объекты попадают на экран, только находясь в видимой области

Функции Direct3D для работы с матрицами

Ранее мы с вами уже говорили, для чего предназначены матрицы, и какие их типы бывают. Теперь пришло время поговорить о функциях для работы с матрицами. Пакет библиотек D3DX содержит набор функций вида `D3DXMatrix`*XXX*, которые сильно упрощают жизнь программисту. Мы рассмотрим некоторые из этих функций более детально.

Итак, мы уже упоминали о следующих операциях:

- сдвиг;
- вращение;
- масштабирование;
- отражение.

Для каждой из этих операций имеются соответствующие функции.

Операция сдвига:

```
function D3DXMatrixTranslation(
  out mOut: TD3DXMatrix;
  x,
  y,
  z: Single):
PD3DXMatrix; stdcall; external d3dx9mathDLL;
```

Здесь:

- `mOut` — результирующая матрица;
- `x`, `y`, `z` — соответствующие координаты смещения.

Вращение по осям *x*, *y* и *z* осуществляется при помощи ряда методов.

Вращение по оси *x*:

```
function D3DXMatrixRotationX(
  out mOut: TD3DXMatrix;
  angle: Single):
PD3DXMatrix; stdcall; external d3dx9mathDLL;
```

Здесь:

- `mOut` — результирующая матрица;
- `angle` — угол поворота.

Вращение по оси *y*:

```
function D3DXMatrixRotationY(
  out mOut: TD3DXMatrix;
  angle: Single):
PD3DXMatrix; stdcall; external d3dx9mathDLL;
```

Вращение по оси *z*:

```
function D3DXMatrixRotationZ(
  out mOut: TD3DXMatrix;
  angle: Single):
PD3DXMatrix; stdcall; external d3dx9mathDLL;
```

Эти три функции имеют одинаковые параметры — результирующую матрицу и угол поворота.

Для масштабирования предназначен следующий метод:

```
function D3DXMatrixScaling(
  out mOut: TD3DXMatrix;
```

```
  sx,
  sy,
  sz: Single):
PD3DXMatrix; stdcall; external d3dx9mathDLL;
```

Здесь:

- `mOut` — результирующая матрица;
- `sx`, `sy`, `sz` — коэффициенты сжатия по соответствующим осям.

Отражение осуществляется функцией:

```
function D3DXMatrixReflect(
  out mOut: TD3DXMatrix;
  const Plane: TD3DXPlane):
PD3DXMatrix; stdcall; external d3dx9mathDLL;
```

Здесь:

- `mOut` — результирующая матрица;
- `Plane` — структура `TD3DXPlane`, описывающая план.

Структура имеет следующий вид:

```
TD3DXPlane = record
  a, b, c, d: Single;
end;
```

Параметры `a`, `b`, `c` и `d` этой структуры составляют уравнение плоскости:

$$ax + by + cz + dw = 0.$$

И последняя операция с матрицами, которую мы сейчас рассмотрим, — это перемножение матриц:

```
function D3DXMatrixMultiply(
  out mOut: TD3DXMatrix;
  const m1, m2: TD3DXMatrix):
PD3DXMatrix; stdcall; external d3dx9mathDLL;
```

Здесь:

- `mOut` — результирующая матрица;
- `m1`, `m2` — перемножаемые матрицы.

Еще несколько методов для работы с матрицами будет рассмотрено далее в процессе изучения материала.

Нормали

Каждая поверхность в Direct3D имеет перпендикулярный вектор, который называется *нормалью* (рис. 5.15). Направление нормали определяется ориентацией системы координат (левосторонняя система координат или правосторонняя) и направлением обхода вершин (по часовой стрелке либо против нее).

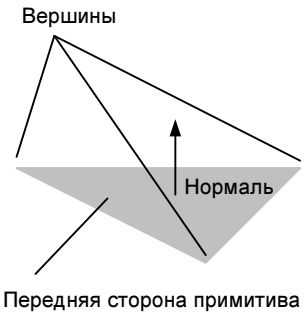


Рис. 5.15. Нормаль

По умолчанию на сцене прорисовывается только передняя сторона примитива. Фактически та, чьи вершины заданы в порядке обхода по часовой стрелке.

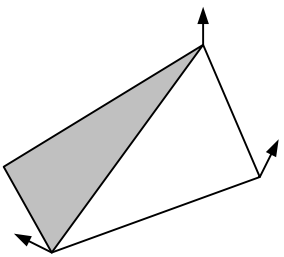


Рис. 5.16. Вершинные нормали

Но мы можем и сами управлять режимом отображения сторон примитива, задав режим `D3DRS_CULLMODE` методом `IDirect3DDevice9.SetRenderState`:

- `D3DCULL_NONE` — не обрабатывать заднюю грань. Это означает, что задняя сторона примитива будет всегда отображаться на экране;

- D3DCULL_CW — задние грани идут по часовой стрелке;
- D3DCULL_CCW — задние грани идут против часовой стрелки.

Нормали используются также при затенении примитивов по методу Гуро, расчета освещенности и наложения текстуры. Для этого уже применяются вершинные нормали (рис. 5.16).

От теории к практике

Вспомним последний разобранный нами пример. Там мы рисовали прямоугольник на экране. Давайте теперь нарисуем тот же самый прямоугольник, только в пространстве.

Для использования нетрансформированных вершин мы должны применить следующий их формат:

```
D3DFVF_CUSTOMVERTEX = D3DFVF_XYZ or D3DFVF_DIFFUSE;
```

При инициализации подсистемы мы задаем режим обработки задних граней и отключаем освещение сцены:

```
// Включаем режим обработки граней
FD3DDevice.SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);


// Отключаем освещение сцены
FD3DDevice.SetRenderState(D3DRS_LIGHTING, 0);
```

Следующим шагом мы должны настроить способ просмотра сцены, т. е. настроить матрицу вида и матрицу проекции. Мировая матрица нам пока не понадобится. Все необходимые настройки мы будем производить в методе SetupCamera (листинг 5.16).

Листинг 5.16. Метод SetupCamera

```
{*********************************************************************}
{** Настройка вида сцены                                            **}
{*********************************************************************}
function TMainForm.SetupCamera: HResult;
var
  ViewMatrix: TD3DMatrix;
  ProjectionMatrix: TD3DMatrix;
  Eye: TD3DVector;
```

```
  At: TD3DVector;
  Up: TD3DVector;
begin
  // Вектор, определяющий положение глаз наблюдателя
  Eye.x := 0; Eye.y := 0; Eye.z := -3;
  // Направление камеры
  At.x := 0;  At.y := 0;  At.z := 0;
  // Вектор, определяющий направление верха мировых координат.
  // Обычно имеет значение (0, 1, 0).
  Up.x := 0;  Up.y := 1;  Up.z := 0;

  // Строим левостороннюю матрицу вида
  D3DXMatrixLookAtLH(ViewMatrix, Eye, At, Up);
  // Устанавливаем матрицу вида
  FD3DDevice.SetTransform(D3DTS_VIEW, ViewMatrix);

  // Строим левостороннюю матрицу проекции
  D3DXMatrixPerspectiveFovLH(ProjectionMatrix, D3DX_PI / 4, 1, 1, 100);
  // Устанавливаем матрицу проекции
  Result := FD3DDevice.SetTransform(D3DTS_PROJECTION, ProjectionMatrix);
end;
```

Методы данной процедуры заслуживают отдельного рассмотрения. Первый метод — это построение левосторонней матрицы вида:

```
function D3DXMatrixLookAtLH(
  out mOut: TD3DXMatrix;
  const Eye, At, Up: TD3DXVector3):
PD3DXMatrix; stdcall; external d3dx9mathDLL;
```

Здесь:

- `mOut` — результирующая матрица;
- `Eye` — вектор, определяющий положение глаз наблюдателя;
- `At` — направление камеры;
- `Up` — вектор, определяющий направление верха мировых координат. Обычно имеет значение (0, 1, 0).

После построения данной матрицы мы должны ее установить методом IDirect3DDevice9.SetTransform:

```
function SetTransform(
  State: TD3DTransformStateType;
  const pMatrix: TD3DMatrix):
HResult; stdcall;
```

Здесь:

- State — переменная, определяющая объект изменения. Может принимать одно из следующих значений:
  - D3DTS_VIEW — матрица вида;
  - D3DTS_PROJECTION — матрица проекции;
  - D3DTS_TEXTURE0, ..., D3DTS_TEXTURE7 — идентифицирует матрицу для текстуры.

  Либо используется макрос D3DTS_WORLDMATRIX;
- pMatrix — структура, описывающая текущее изменение.

Матрица проекции строится методом D3DXMatrixPerspectiveFovLH:

```
function D3DXMatrixPerspectiveFovLH(
  out mOut: TD3DXMatrix;
  flovy,
  aspect,
  zn,
  zf: Single):
PD3DXMatrix; stdcall; external d3dx9mathDLL;
```

Здесь:

- mOut — результирующая матрица;
- flovy — угол обзора в радианах;
- aspect — отношение длины к высоте;
- zn — отсечение по оси z на переднем плане;
- zf — отсечение по оси z на заднем плане.

После настройки параметров просмотра сцены мы должны определить набор вершин (листинг 5.17).

Листинг 5.17. Подготовка набора вершин

```
{*********************************************************************}
{** Подготовка набора вершин                                        **}
{*********************************************************************}
function TMainForm.PrepareVertices: HResult;
const
  Vertices: array[0..5] of TCustomVertex = (
    (x: -1; y:  1; z: 0; color: $00FF0000),
    (x:  1; y:  1; z: 0; color: $00FF0000),
    (x:  1; y: -1; z: 0; color: $00FF0000),

    (x:  1; y: -1; z: 0; color: $00FF0000),
    (x: -1; y: -1; z: 0; color: $00FF0000),
    (x: -1; y:  1; z: 0; color: $00FF0000)
  );
var
  pVertices: pointer;
begin
  // Создаем буфер вершин
  Result := FD3DDevice.CreateVertexBuffer(SizeOf(Vertices), 0,
    D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT, FD3DVertexBuffer, NIL);

  // При невозможности создания завершаем работу
  if FAILED(Result) then EXIT;

  // Блокируем доступ к буферу вершин
  Result := FD3DVertexBuffer.Lock(0, SizeOf(Vertices),
    pVertices, 0);
  if FAILED(Result) then EXIT;

  try
    // Копируем в буфер данные
    CopyMemory(pVertices, @Vertices, SizeOf(Vertices));
  finally
```

```
    // Разблокировка буфера
    FD3DVertexBuffer.Unlock;
  end;
end;
```

Наш прямоугольник будет по-прежнему состоять из двух треугольников. Обратите внимание, какие изменения произошли в координатах вершин — у нас появились отрицательные координаты, т. к. центр осей координат находится в центре экрана.

Пример из каталога Example_13 наглядно нам все демонстрирует (рис. 5.17).

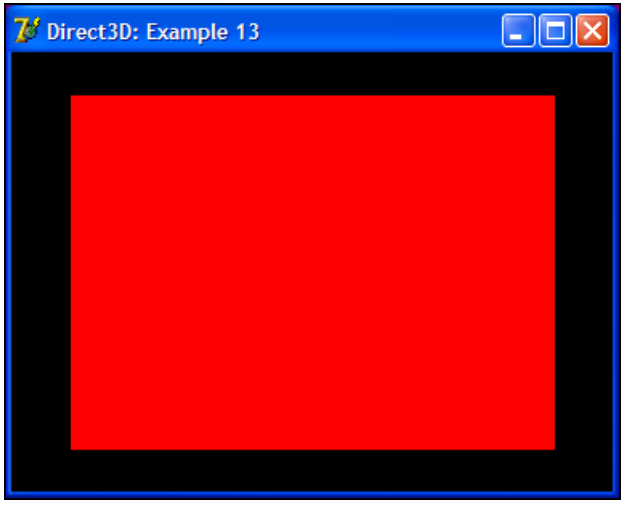


Рис. 5.17. Прямоугольник в пространстве

Куб

Научились строить квадрат — давайте усложним задачу и построим куб. Как известно, куб состоит из 6 граней, каждая из которых будет состоять у нас, в свою очередь, из двух треугольников. Соответственно, у нас получается 12 примитивов. Процедура подготовки набора вершин будет такой, как представлено в листинге 5.18.

Листинг 5.18. Процедура подготовки набора вершин для постоения куба

```
{*********************************************************************}
{** Подготовка набора вершин                                        **}
{*********************************************************************}
function TMainForm.PrepareVertices: HResult;
const
  Vertices: array[0..35] of TCustomVertex = (
    (x: -1; y:  1; z: -1; color: $00FF00FF),
    (x:  1; y:  1; z: -1; color: $00FF00FF),
    (x:  1; y: -1; z: -1; color: $00FF00FF),
    (x:  1; y: -1; z: -1; color: $00FF00FF),
    (x: -1; y: -1; z: -1; color: $00FF00FF),
    (x: -1; y:  1; z: -1; color: $00FF00FF),

    (x: -1; y:  1; z:  1; color: $00FF00FF),
    (x: -1; y:  1; z: -1; color: $00FF00FF),
    (x: -1; y: -1; z: -1; color: $00FF00FF),
    (x: -1; y: -1; z: -1; color: $00FF00FF),
    (x: -1; y: -1; z:  1; color: $00FF00FF),
    (x: -1; y:  1; z:  1; color: $00FF00FF),

    (x:  1; y:  1; z:  1; color: $00FF00FF),
    (x: -1; y:  1; z:  1; color: $00FF00FF),
    (x: -1; y: -1; z:  1; color: $00FF00FF),
    (x: -1; y: -1; z:  1; color: $00FF00FF),
    (x:  1; y: -1; z:  1; color: $00FF00FF),
    (x:  1; y:  1; z:  1; color: $00FF00FF),

    (x:  1; y:  1; z: -1; color: $00FF00FF),
    (x:  1; y:  1; z:  1; color: $00FF00FF),
    (x:  1; y: -1; z:  1; color: $00FF00FF),
    (x:  1; y: -1; z:  1; color: $00FF00FF),
    (x:  1; y: -1; z: -1; color: $00FF00FF),
    (x:  1; y:  1; z: -1; color: $00FF00FF),
```

```
    (x: -1; y:  1; z:  1; color: $00FF00FF),
    (x:  1; y:  1; z:  1; color: $00FF00FF),
    (x:  1; y:  1; z: -1; color: $00FF00FF),
    (x:  1; y:  1; z: -1; color: $00FF00FF),
    (x: -1; y:  1; z: -1; color: $00FF00FF),
    (x: -1; y:  1; z:  1; color: $00FF00FF),

    (x: -1; y: -1; z:  1; color: $00FF00FF),
    (x: -1; y: -1; z: -1; color: $00FF00FF),
    (x:  1; y: -1; z: -1; color: $00FF00FF),
    (x:  1; y: -1; z: -1; color: $00FF00FF),
    (x:  1; y: -1; z:  1; color: $00FF00FF),
    (x: -1; y: -1; z:  1; color: $00FF00FF)
  );
var
  pVertices: pointer;
begin
  // Создаем буфер вершин
  Result := FD3DDevice.CreateVertexBuffer(SizeOf(Vertices), 0,
    D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT, FD3DVertexBuffer, NIL);

  // При невозможности создания завершаем работу
  if FAILED(Result) then EXIT;

  // Блокируем доступ к буферу вершин
  Result := FD3DVertexBuffer.Lock(0, SizeOf(Vertices),
    pVertices, 0);
  if FAILED(Result) then EXIT;

  try
    // Копируем в буфер данные
    CopyMemory(pVertices, @Vertices, SizeOf(Vertices));
  finally
    // Разблокировка буфера
    FD3DVertexBuffer.Unlock;
  end;
end;
```

Если мы просто нарисуем куб на экране, то получим проекцию — квадрат. А это никак не может служить наглядным пособием. Поэтому для придания картинке большей реалистичности мы будем поворачивать наш куб по всем трем осям *x*, *y* и *z* одновременно. Ранее мы уже говорили о функциях, которые упрощают нашу работу по подготовке матриц поворота, а также обсуждали способ наложения нескольких модификаций на сцену одновременно — это перемножение матриц трансформации. В результате процедура прорисовки сцены будет такой, как представлено в листинге 5.19.

Листинг 5.19. Процедура прорисовки сцены

```
{*********************************************************************}
{** Прорисовка сцены                                                **}
{*********************************************************************}
function TMainForm.RenderScene: HResult;
var
  WorldMatrix:  TD3DMatrix;
  WorldMatrixX: TD3DMatrix;
  WorldMatrixY: TD3DMatrix;
  WorldMatrixZ: TD3DMatrix;
begin
  // Чистим устройство
  Result := ClearDevice;
  // Если произошла ошибка, то завершаем работу
  if FAILED(Result) then EXIT;

  // Начало сцены
  FD3DDevice.BeginScene;
  try
    // Связываем буфер вершин с потоком данных устройства
    FD3DDevice.SetStreamSource(0, FD3DVertexBuffer, 0,
      SizeOf(TCustomVertex));

    // Устанавливаем формат вершин
    FD3DDevice.SetFVF(D3DFVF_CUSTOMVERTEX);

    // Увеличиваем угол поворота
    inc(FRotAngle, 1);
```

```
    // Вращаем по всем осям x, y и z
    D3DXMatrixRotationX(WorldMatrixX, GradToRad(FRotAngle));
    D3DXMatrixRotationY(WorldMatrixY, GradToRad(FRotAngle));
    D3DXMatrixRotationZ(WorldMatrixZ, GradToRad(FRotAngle));

    // Накладываем все модификации на мировую матрицу
    D3DXMatrixMultiply(WorldMatrix, WorldMatrixX, WorldMatrixY);
    D3DXMatrixMultiply(WorldMatrix, WorldMatrix,  WorldMatrixZ);

    // Устанавливаем мировую матрицу для куба
    FD3DDevice.SetTransform(D3DTS_WORLD, WorldMatrix);

    // Рисуем примитивы
    FD3DDevice.DrawPrimitive(D3DPT_TRIANGLELIST, 0, 12);
  finally

    // Завершаем сцену
    FD3DDevice.EndScene;
  end;

  // Переключение буферов
  FD3DDevice.Present(NIL, NIL, 0, NIL);
end;
```

Вот вроде бы и все — все параметры настроены, куб вращается, но что-то не то (рис. 5.18).

Может, нужно просто-напросто окрасить грани куба в разный цвет? Давайте попробуем:

```
Vertices: array[0..35] of TCustomVertex = (
  (x: -1; y:  1; z: -1; color: $0000FF00),
  (x:  1; y:  1; z: -1; color: $0000FF00),
  (x:  1; y: -1; z: -1; color: $0000FF00),
  (x:  1; y: -1; z: -1; color: $0000FF00),
  (x: -1; y: -1; z: -1; color: $0000FF00),
  (x: -1; y:  1; z: -1; color: $0000FF00),
```

```
  (x: -1; y:  1; z:  1; color: $00FF00FF),
  (x: -1; y:  1; z: -1; color: $00FF00FF),
  (x: -1; y: -1; z: -1; color: $00FF00FF),
  (x: -1; y: -1; z: -1; color: $00FF00FF),
  (x: -1; y: -1; z:  1; color: $00FF00FF),
  (x: -1; y:  1; z:  1; color: $00FF00FF),

  (x:  1; y:  1; z:  1; color: $00F0FF0F),
  (x: -1; y:  1; z:  1; color: $00F0FF0F),
  (x: -1; y: -1; z:  1; color: $00F0FF0F),
  (x: -1; y: -1; z:  1; color: $00F0FF0F),
  (x:  1; y: -1; z:  1; color: $00F0FF0F),
  (x:  1; y:  1; z:  1; color: $00F0FF0F),

  (x:  1; y:  1; z: -1; color: $000000FF),
  (x:  1; y:  1; z:  1; color: $000000FF),
  (x:  1; y: -1; z:  1; color: $000000FF),
  (x:  1; y: -1; z:  1; color: $000000FF),
  (x:  1; y: -1; z: -1; color: $000000FF),
  (x:  1; y:  1; z: -1; color: $000000FF),

  (x: -1; y:  1; z:  1; color: $00FF0000),
  (x:  1; y:  1; z:  1; color: $00FF0000),
  (x:  1; y:  1; z: -1; color: $00FF0000),
  (x:  1; y:  1; z: -1; color: $00FF0000),
  (x: -1; y:  1; z: -1; color: $00FF0000),
  (x: -1; y:  1; z:  1; color: $00FF0000),

  (x: -1; y: -1; z:  1; color: $00F0F0F0),
  (x: -1; y: -1; z: -1; color: $00F0F0F0),
  (x:  1; y: -1; z: -1; color: $00F0F0F0),
  (x:  1; y: -1; z: -1; color: $00F0F0F0),
  (x:  1; y: -1; z:  1; color: $00F0F0F0),
  (x: -1; y: -1; z:  1; color: $00F0F0F0)
);
```

И снова на экране мы видим нечто непонятное (рис. 5.19).

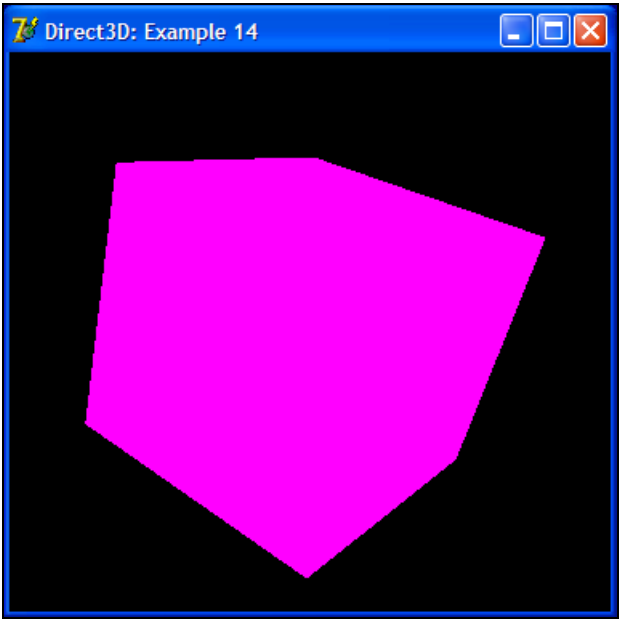


Рис. 5.18. Странный куб

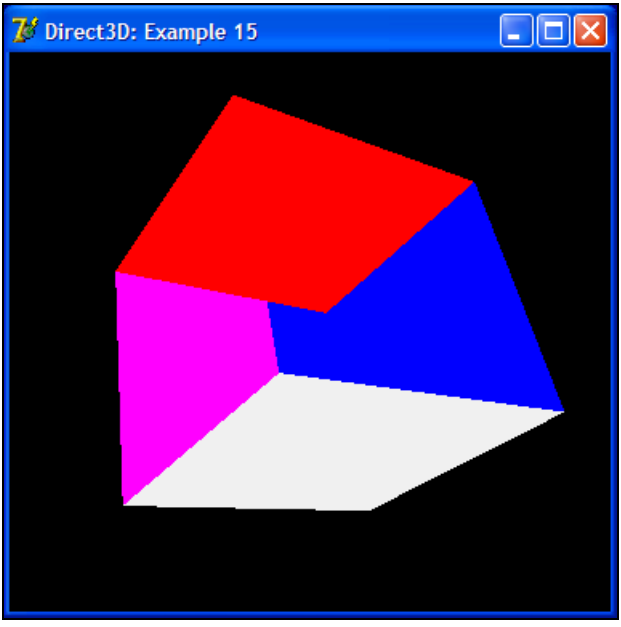


Рис. 5.19. Странный куб с окрашенными гранями

Так в чем же проблема? Почему на экране у нас нечто непонятное вместо куба?

Буфер глубины

Оказывается, нам просто не хватает буфера глубины. *Буфером глубины* называется свойство устройства хранить информацию о глубине сцены. Это означает, что данный буфер отвечает за порядок вывода примитивов на экран и корректное наложение их друг на друга. Некоторые называют этот буфер *z*-буфером.

Давайте изменим наш последний пример и добавим в него поддержку *z*-буфера:

```
// использовать буфер глубины
d3dParams.EnableAutoDepthStencil := TRUE;
// формат буфера глубины
d3dParams.AutoDepthStencilFormat := D3DFMT_D16;
```

Таким образом, мы указываем на поддержку 16-битного формата буфера глубины. После создания устройства включаем поддержку им буфера глубины:

```
FD3DDevice.SetRenderState(D3DRS_ZENABLE, D3DZB_TRUE);
```

И последнее, что нам осталось сделать, — это переписать метод очистки устройства и добавить туда флаг очистки буфера глубины (листинг 5.20).

Листинг 5.20. Метод очистки устройства

```
{*********************************************************************}
{** Очищаем устройство                                              **}
{*********************************************************************}
function TMainForm.ClearDevice: HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если устройство не создано, то завершаем работу
  if FD3DDevice = NIL then EXIT;
```

```
  // Чистим устройство
  Result := FD3DDevice.Clear(0, NIL, D3DCLEAR_TARGET or D3DCLEAR_ZBUFFER,
    D3DCOLOR_XRGB(0, 0, 0), 1, 0);
end;
```

Результат представлен на рис. 5.20.

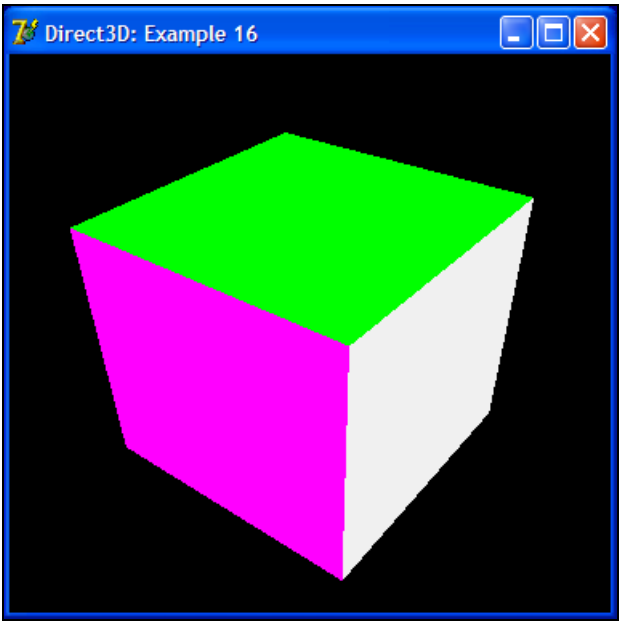


Рис. 5.20. Долгожданный куб

Нарисовано все корректно. Куб выглядит вполне естественно.

Правила построения объектов

Вот мы и научились строить куб из примитивов. Всего нам понадобилось 12 треугольников для построения куба. Куб имеет 6 граней, каждая из которых, в свою очередь, состоит из двух треугольников. Важен порядок задания вершин каждого треугольника — вершины задаются последовательно по часовой стрелке. На рис. 5.21 показана одна из граней нашего куба, состоящая

из двух треугольников. Из рисунка хорошо видно, что это треугольники *ABC* и *CDA*. Остальные грани куба строятся по точно такому же принципу.

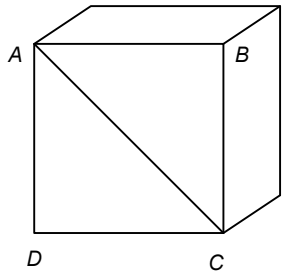


Рис. 5.21. Порядок задания вершин примитивов

Мы видим, что у двух треугольников две вершины общие. Таким образом, умножив две вершины на общее число граней куба, получим, что 12 вершин у нас лишние. И для построения куба хватило бы не 36 вершин, а всего лишь 24. Так каким же образом можно отказаться от этих лишних вершин?

Оказывается, есть и другой способ задания объектов, при котором мы задаем список вершин и список индексов, описывающий порядок использования наших вершин при построении объекта.

Рассмотрим все это на примере. Пусть для простоты вершины *A*, *B*, *C* и *D* имеют индексы 0, 1, 2 и 3. Тогда у нас получается следующий порядок описания индексов одной грани куба — это 0, 1, 2 и 2, 3, 0. То есть мы задаем все те же два треугольника, только иным способом.

Для оперирования *буфером индексов вершин* (Index Buffer) имеется специальный интерфейс `IDirect3DIndexBuffer9`. Для создания объекта буфера существует особый метод `IDirect3DDevice9.CreateIndexBuffer`:

```
function CreateIndexBuffer(
  Length: LongWord;
  Usage: DWord;
  Format: TD3DFormat;
  Pool: TD3DPool;
  out ppIndexBuffer: IDirect3DIndexBuffer9;
  pSharedHandle: PHandle):
HResult; stdcall;
```

Здесь:

- `Length` — размер буфера в байтах;
- `Usage` — указывает на способ использования ресурсов. Может принимать нулевое значение;
- `Format` — формат буфера индексов. Допустимыми значениями могут являться следующие:
  - `D3DFMT_INDEX16` — 16 битов на индекс;
  - `D3DFMT_INDEX32` — 32 бита на индекс;
- `Pool` — класс памяти для ресурсов (см. ранее описание аналогичного параметра в методе `IDirect3DDevice9.CreateVertexBuffer`);
- `ppIndexBuffer` — адрес переменной, в которую будет передан интерфейс созданного буфера индексов;
- `pSharedHandle` — зарезервировано. Должно использоваться нулевое значение.

И точно так же, как и для буфера вершин, нам для работы с буфером индексов потребуются два метода: `IDirect3DIndexBuffer9.Lock` и `IDirect3DIndexBuffer9.Unlock`.

Первый метод — блокировка доступа к содержимому буфера и получение указателя на область памяти буфера:

```
function Lock(
  OffsetToLock,
  SizeToLock: DWord;
  out ppbData: Pointer;
  Flags: DWord):
HResult; stdcall;
```

Здесь:

- `OffsetToLock` — смещение от начала буфера в байтах;
- `SizeToLock` — размер блока данных в байтах;
- `ppbData` — указатель, который будет содержать ссылку на область памяти данных буфера;
- `Flags` — способ доступа к данным в буфере. Может принимать нулевое значение.

Второй метод позволяет выполнить разблокировку буфера вершин `IDirect3DIndexBuffer9.Unlock`:

```
function Unlock: HResult; stdcall;
```

Пример из каталога Example_17 строит куб с использованием буфера индексов. Первым делом мы задаем список вершин и список индексов (листинг 5.21).

Листинг 5.21. Задание списка вершин и списка индексов

```
{*********************************************************************}
{** Подготовка набора вершин                                        **}
{*********************************************************************}
function TMainForm.PrepareVertices: HResult;
const
  Vertices: array[0..23] of TCustomVertex = (
    (x: -1; y:  1; z: -1; color: $000000FF),
    (x:  1; y:  1; z: -1; color: $000000FF),
    (x:  1; y: -1; z: -1; color: $000000FF),
    (x: -1; y: -1; z: -1; color: $000000FF),

    (x: -1; y:  1; z:  1; color: $00F00FF0),
    (x: -1; y:  1; z: -1; color: $00F00FF0),
    (x: -1; y: -1; z: -1; color: $00F00FF0),
    (x: -1; y: -1; z:  1; color: $00F00FF0),

    (x:  1; y:  1; z:  1; color: $0000FF00),
    (x: -1; y:  1; z:  1; color: $0000FF00),
    (x: -1; y: -1; z:  1; color: $0000FF00),
    (x:  1; y: -1; z:  1; color: $0000FF00),

    (x:  1; y:  1; z: -1; color: $000FF000),
    (x:  1; y:  1; z:  1; color: $000FF000),
    (x:  1; y: -1; z:  1; color: $000FF000),
    (x:  1; y: -1; z: -1; color: $000FF000),

    (x: -1; y:  1; z:  1; color: $00FF0000),
    (x:  1; y:  1; z:  1; color: $00FF0000),
    (x:  1; y:  1; z: -1; color: $00FF0000),
    (x: -1; y:  1; z: -1; color: $00FF0000),
```

```
    (x:  1; y: -1; z: -1; color: $00FFFFFF),
    (x:  1; y: -1; z:  1; color: $00FFFFFF),
    (x: -1; y: -1; z:  1; color: $00FFFFFF),
    (x: -1; y: -1; z: -1; color: $00FFFFFF)
  );

  Indexes: array[0..35] of WORD = (
     0,  1,  2,  2,  3,  0,
     4,  5,  6,  6,  7,  4,
     8,  9, 10, 10, 11,  8,
    12, 13, 14, 14, 15, 12,
    16, 17, 18, 18, 19, 16,
    20, 21, 22, 22, 23, 20
  );
var
  pVertices: pointer;
begin
  // Создаем буфер вершин
  Result := FD3DDevice.CreateVertexBuffer(SizeOf(Vertices), 0,
    D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT, FD3DVertexBuffer, NIL);

  // При невозможности создания завершаем работу
  if FAILED(Result) then EXIT;

  // Блокируем доступ к буферу вершин
  Result := FD3DVertexBuffer.Lock(0, SizeOf(Vertices),
    pVertices, 0);
  if FAILED(Result) then EXIT;

  try
    // Копируем в буфер данные
    CopyMemory(pVertices, @Vertices, SizeOf(Vertices));
  finally
    // Разблокировка буфера
    FD3DVertexBuffer.Unlock;
  end;
```

```
  // Создаем буфер индексов
  Result := FD3DDevice.CreateIndexBuffer(SizeOf(Indexes),
    D3DUSAGE_WRITEONLY, D3DFMT_INDEX16, D3DPOOL_MANAGED,
    FD3DIndexBuffer, NIL);

  // При невозможности создания завершаем работу
  if FAILED(Result) then EXIT;

  // Блокируем доступ к буферу индексов
  Result := FD3DIndexBuffer.Lock(0, SizeOf(Indexes), pVertices, 0);
  if FAILED(Result) then EXIT;

  try
    // Копируем в буфер данные
    CopyMemory(pVertices, @Indexes, SizeOf(Indexes));
  finally
    // Разблокировка буфера
    FD3DIndexBuffer.Unlock;
  end;
end;
```

Массив `Indexes` содержит индексы вершин для построения куба. Они будут переписаны в буфер индексов `FD3DIndexBuffer`. Для рисования куба в процедуре прорисовки сцены нами будет использоваться следующая последовательность команд:

```
// Устанавливаем индексы
FD3DDevice.SetIndices(FD3DIndexBuffer);

// Рисуем проиндексированные примитивы
FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 36, 0, 12);
```

Сначала устанавливаем набор индексов, а затем рисуем проиндексированные примитивы методом `IDirect3DDevice9.DrawIndexedPrimitive`:

```
function DrawIndexedPrimitive(
  _Type: TD3DPrimitiveType;
  BaseVertexIndex: Integer;
  MinVertexIndex,
```

```
  NumVertices,
  startIndex,
  primCount: LongWord):
HResult; stdcall;
```

Здесь:

- `_Type` — тип примитивов (см. ранее описание метода `IDirect3DDevice9.DrawPrimitive`);
- `BaseVertexIndex` — смещение от начала буфера до первого индекса вершины;
- `MinVertexIndex` — минимальный индекс вершины для использования в контексте данного вызова;
- `NumVertices` — число вершин, которые будут использованы в контексте данного вызова начиная с индекса `BaseVertexIndex` + `MinVertexIndex`;
- `startIndex` — позиция в массиве индексов, с которой начинается чтение вершин;
- `primCount` — число примитивов.

Источники света

Для того чтобы куб выглядел еще естественнее, нам необходимо использовать материал и освещение. Начнем изучение данной темы с источников света. Источники света используются для освещения объектов сцены. Если освещение задействовано, то подсистема Direct3D рассматривает цвет каждой вершины как совокупность следующих составляющих:

- цвет текущего материала;
- диффузная и отражающая цветовая составляющая вершины;
- цвет и интенсивность источника света, и уровень света окружающей среды.

Источники цвета бывают нескольких типов:

- направленный или параллельный (Directional);
- точечный или всенаправленный (Point Light);
- прожекторный или конусный (Spotlight).

Направленные источники света имеют только цвет и направление, у них нет позиции. Они излучают параллельные лучи света (рис. 5.22). Это означает, что все сгенерированные источником лучи будут проходить сквозь всю сцену в одном направлении. Направленный источник можно представить в ви-

де бесконечно удаленного источника, излучающего свет. Простой пример — солнце. Интенсивность излучения таких источников не ослабевает ни от времени свечения, ни от расстояния. Таким образом, направление и цвет источника, которые мы задаем, и будут являться единственной характеристикой, которую будет использовать подсистема Direct3D при расчете цветов вершин.

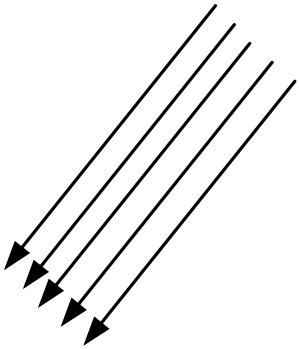


Рис. 5.22. Направленный источник света

Точечный источник света определяется позицией в пространстве и цветом, но у него нет какого-то конкретного направления лучей. Он одинаково излучает свет во всех направлениях, как это показано на рис. 5.23.

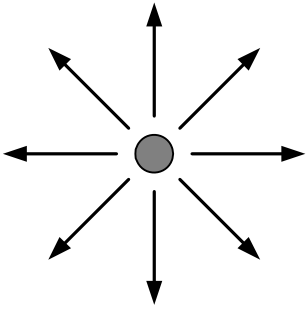


Рис. 5.23. Точечный источник света

Пример такого источника света напрашивается сам собой — это лампочка. Интенсивность света от такого источника ослабляется с расстоянием. Пока источник света существует, подсистема Direct3D использует положение источника в пространстве и координаты освещаемой вершины для определе-

ния вектора направления света и расстояния, которое должен преодолеть луч света. Все это в совокупности с вершинной нормалью необходимо для расчета уровня освещенности поверхности.

Прожекторный источник света имеет цвет, положение в пространстве и направление освещения. Свет, исходящий от источника, состоит из внутреннего яркого конуса и конуса внешнего, освещение которого уменьшается от центра к краям как показано на рис. 5.24.

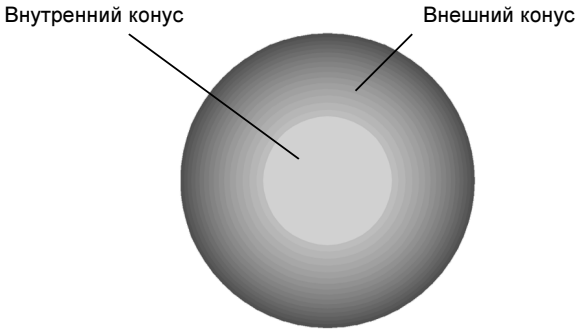


Рис. 5.24. Внутренний и внешний конусы

Интенсивность источника также меняется с расстоянием.

Для описания источников света в Direct3D имеется специальная структура `TD3DLight9`. Она содержит следующие поля:

- `_Type` — тип источника света:
  - `D3DLIGHT_POINT` — точечный источник;
  - `D3DLIGHT_SPOT` — прожекторный источник;
  - `D3DLIGHT_DIRECTIONAL` — направленный источник света;
- `Diffuse` — диффузный (рассеиваемый) цвет;
- `Specular` — зеркальный цвет;
- `Ambient` — цвет окружающей среды;
- `Position` — позиция источника света;
- `Direction` — направление излучения;
- `Range` — дистанция, на которой действует источник света;
- `Falloff` — уменьшение в освещении между внутренним конусом (угол θ) и краем внешнего конуса (угол φ);
- `Attenuation0`, `Attenuation1`, `Attenuation2` — параметры затухания;

- `Theta` — угол в радианах, определяющий внутренний конус прожекторного источника света — наиболее освещенный конус. Величина угла задается от `0` до `Phi`;
- `Phi` — угол в радианах, определяющий внешнюю границу внешнего конуса прожекторного источника света. Точки за пределами этой границы освещены не будут. Величина угла задается от `0` до `PI`.

Как видно из описания, три параметра структуры относятся исключительно к прожекторным источникам света. Это параметры `Falloff`, `Theta` и `Phi`. Общая схема, иллюстрирующая работу прожекторного источника света, приведена на рис. 5.25.

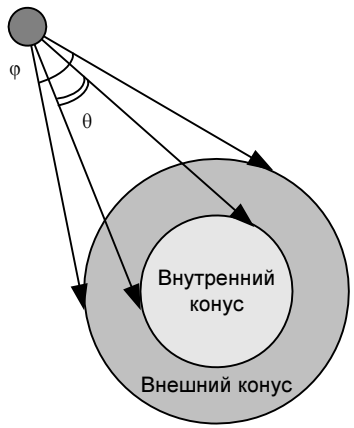


Рис. 5.25. Прожекторный источник света

Свет в Direct3D излучает 3 цвета, которые будут использоваться в вычислениях: диффузный цвет, цвет окружающей среды и зеркальный цвет. Каждый из этих цветов будет взаимодействовать с аналогичным цветом в свойствах материала, т. е. диффузный цвет источника будет взаимодействовать с диффузным цветом материала, зеркальный цвет источника — с зеркальным цветом материала и т. д.

Наиболее сильно влияющий на сцену свет — диффузный. Обычно диффузный цвет задается белым, но никто не мешает нам задавать и другие цвета, например, красный цвет для освещения сцены пожара.

Обычно характеристики цвета освещения задаются в диапазоне от `0.0` до `1.0`, но это не жесткие условия. Например, мы можем использовать значение `2.0` для задания цвета "более яркого, чем белый цвет".

Несмотря на то, что в Direct3D используется формат цвета RGBA, альфа-составляющая цвета не задействована.

Материал

В подсистеме Direct3D материал описывает способ взаимодействия примитива (полигона) со светом. По своей сути материал является набором свойств, от которых зависит способ обработки.

Для описания свойств материала имеется структура `TD3DMaterial9`, которая состоит из следующих полей:

- `Diffuse` — диффузный цвет;
- `Ambient` — цвет окружающей среды;
- `Specular` — зеркальный цвет;
- `Emissive` — излучаемый цвет;
- `Power` — величина, определяющая резкость. Чем больше значение, тем больше резкость.

Диффузный цвет определяет матовые поверхности, а зеркальный цвет задает блики.

Работа с материалом и источниками света

Изучив теорию, перейдем к практике. Первым делом из процедуры инициализации подсистемы удалим строку, отключающую освещение сцены:

```
FD3DDevice.SetRenderState(D3DRS_LIGHTING, 0);
```

Эта строка нам больше не потребуется. В процедуру прорисовки сцены добавляем методы установки материала и источника света. Вот что у нас получится в итоге (листинг 5.22).

Листинг 5.22. Прорисовка сцены

```
{*************************************************************************}
{** Прорисовка сцены                                                    **}
{*************************************************************************}
function TMainForm.RenderScene: HResult;
var
  WorldMatrix:  TD3DMatrix;
  WorldMatrixX: TD3DMatrix;
  WorldMatrixY: TD3DMatrix;
  WorldMatrixZ: TD3DMatrix;
```

```
  d3dMaterial: TD3DMaterial9;
  d3dVector: TD3DVector;
  d3dLight: TD3DLight9;
begin
  // Чистим устройство
  Result := ClearDevice;
  // Если произошла ошибка, то завершаем работу
  if FAILED(Result) then EXIT;

  // Начало сцены
  FD3DDevice.BeginScene;
  try
    // Связываем буфер вершин с потоком данных устройства
    FD3DDevice.SetStreamSource(0, FD3DVertexBuffer, 0,
      SizeOf(TCustomVertex));

    // Устанавливаем формат вершин
    FD3DDevice.SetFVF(D3DFVF_CUSTOMVERTEX);

    // Увеличиваем угол поворота
    inc(FRotAngle, 1);

    // Вращаем по всем осям x, y и z
    D3DXMatrixRotationX(WorldMatrixX, GradToRad(FRotAngle));
    D3DXMatrixRotationY(WorldMatrixY, GradToRad(FRotAngle));
    D3DXMatrixRotationZ(WorldMatrixZ, GradToRad(FRotAngle));

    // Накладываем все модификации на мировую матрицу
    D3DXMatrixMultiply(WorldMatrix, WorldMatrixX, WorldMatrixY);
    D3DXMatrixMultiply(WorldMatrix, WorldMatrix,  WorldMatrixZ);

    // Устанавливаем мировую матрицу для куба
    FD3DDevice.SetTransform(D3DTS_WORLD, WorldMatrix);

    // Задаем свойства материала
    ZeroMemory(@d3dMaterial, SizeOf(d3dMaterial));
```

```
d3dMaterial.Diffuse.r := 1;
d3dMaterial.Diffuse.g := 1;
d3dMaterial.Diffuse.b := 0;
d3dMaterial.Diffuse.a := 1;
d3dMaterial.Ambient := d3dMaterial.Diffuse;

// Устанавливаем материал
FD3DDevice.SetMaterial(d3dMaterial);

// Задаем параметры источника света
ZeroMemory(@d3dLight, SizeOf(d3dLight));
d3dLight._Type := D3DLIGHT_DIRECTIONAL;
d3dLight.Diffuse.r := 1;
d3dLight.Diffuse.g := 1;
d3dLight.Diffuse.b := 1;
d3dLight.Range := 20;

// Направление источника
d3dVector.x := 0;
d3dVector.y := 0;
d3dVector.z := 5;
D3DXVec3Normalize(d3dLight.Direction, d3dVector);

// Устанавливаем параметры первого источника света сцены
FD3DDevice.SetLight(0, d3dLight);

// Включаем источник
FD3DDevice.LightEnable(0, TRUE);

// Включаем режим обработки освещения
FD3DDevice.SetRenderState(D3DRS_LIGHTING, 1);

// Задаем освещение окружающей среды
FD3DDevice.SetRenderState(D3DRS_AMBIENT, $00202020);

// Устанавливаем индексы
FD3DDevice.SetIndices(FD3DIndexBuffer);
```

```
    // Рисуем проиндексированные примитивы
    FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 36, 0, 12);
  finally

    // Завершаем сцену
    FD3DDevice.EndScene;
  end;

  // Переключение буферов
  FD3DDevice.Present(NIL, NIL, 0, NIL);
end;
```

Как видно из примера, первым делом мы задаем свойства материала и устанавливаем его методом `IDirect3DDevice9.SetMaterial`:

```
function SetMaterial(
  const pMaterial: TD3DMaterial9):
HResult; stdcall;
```

Здесь `pMaterial` — структура `TD3DMaterial9`, описывающая свойства материала.

Затем задаем параметры источника света. Мы будем использовать направленный источник. Источник в сцене будет один, и его параметры устанавливаются методом `IDirect3DDevice9.SetLight`:

```
function SetLight(
  Index: DWord;
  const pLight: TD3DLight9):
HResult; stdcall;
```

Здесь:

- `Index` — индекс источника света, начиная с `0`;
- `pLight` — структура `TD3DLight9`, описывающая свойства источника света.

Установив параметры источника света, включаем его методом `IDirect3DDevice9.LightEnable`:

```
function LightEnable(
  Index: DWord;
  Enable: BOOL):
HResult; stdcall;
```

Здесь:

- `Index` — индекс источника света, начиная с `0`;
- `Enable` — режим работы источника света (`FALSE` — выключен, `TRUE` — включен).

Обработка диффузного цвета начинается после вызова метода `IDirect3DDevice9.SetRenderState` с включенным параметром `D3DRS_LIGHTING`:

```
FD3DDevice.SetRenderState(D3DRS_LIGHTING, 1);
```

Цвет окружающего освещения включается следующей командой:

```
FD3DDevice.SetRenderState(D3DRS_AMBIENT, $00202020);
```

Теперь запускаем пример на выполнение. Он располагается в каталоге Example_18. На рис. 5.26 представлен пример его работы.

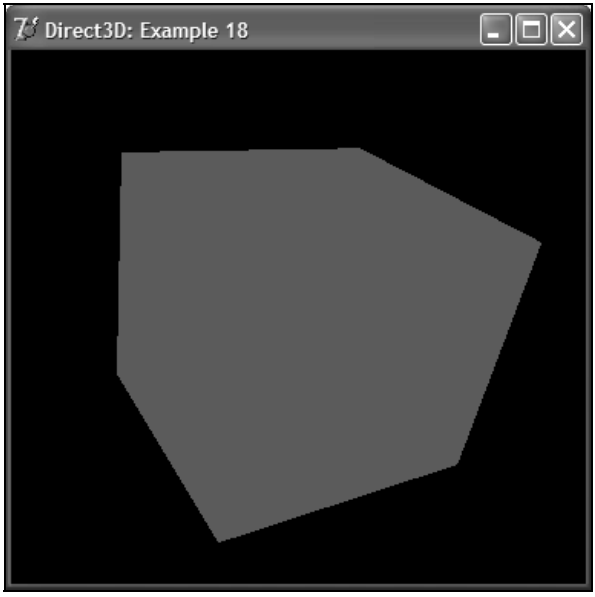


Рис. 5.26. Куб представляет собой нечто непонятное

Оказывается, мы не задали вершинные нормали, и освещение работает некорректно. Для их задания в структуру `TCustomVertex` необходимо добавить три параметра — вектор (`nx`, `ny`, `nz`), задающий вершинную нормаль:

```
TCustomVertex = packed record
  x, y, z: Single;
```

```
  nx, ny, nz: Single;
  color: DWORD;
end;
```

А в описание константы, задающей формат точек, мы добавим флаг D3DFVF_NORMAL:

```
D3DFVF_CUSTOMVERTEX = D3DFVF_XYZ or D3DFVF_DIFFUSE or D3DFVF_NORMAL;
```

Соответственно, и в массив вершин нашего куба будут добавлены векторы нормалей:

```
const
  Vertices: array[0..23] of TCustomVertex = (
    (x: -1; y:  1; z: -1; nx:  0; ny:  0; nz: -1; color: $000000FF),
    (x:  1; y:  1; z: -1; nx:  0; ny:  0; nz: -1; color: $000000FF),
    (x:  1; y: -1; z: -1; nx:  0; ny:  0; nz: -1; color: $000000FF),
    (x: -1; y: -1; z: -1; nx:  0; ny:  0; nz: -1; color: $000000FF),

    (x: -1; y:  1; z:  1; nx: -1; ny:  0; nz:  0; color: $00F00FF0),
    (x: -1; y:  1; z: -1; nx: -1; ny:  0; nz:  0; color: $00F00FF0),
    (x: -1; y: -1; z: -1; nx: -1; ny:  0; nz:  0; color: $00F00FF0),
    (x: -1; y: -1; z:  1; nx: -1; ny:  0; nz:  0; color: $00F00FF0),

    (x:  1; y:  1; z:  1; nx:  0; ny:  0; nz:  1; color: $0000FF00),
    (x: -1; y:  1; z:  1; nx:  0; ny:  0; nz:  1; color: $0000FF00),
    (x: -1; y: -1; z:  1; nx:  0; ny:  0; nz:  1; color: $0000FF00),
    (x:  1; y: -1; z:  1; nx:  0; ny:  0; nz:  1; color: $0000FF00),

    (x:  1; y:  1; z: -1; nx:  1; ny:  0; nz:  0; color: $000FF000),
    (x:  1; y:  1; z:  1; nx:  1; ny:  0; nz:  0; color: $000FF000),
    (x:  1; y: -1; z:  1; nx:  1; ny:  0; nz:  0; color: $000FF000),
    (x:  1; y: -1; z: -1; nx:  1; ny:  0; nz:  0; color: $000FF000),

    (x: -1; y:  1; z:  1; nx:  0; ny:  1; nz:  0; color: $00FF0000),
    (x:  1; y:  1; z:  1; nx:  0; ny:  1; nz:  0; color: $00FF0000),
    (x:  1; y:  1; z: -1; nx:  0; ny:  1; nz:  0; color: $00FF0000),
    (x: -1; y:  1; z: -1; nx:  0; ny:  1; nz:  0; color: $00FF0000),
```

```
    (x:  1; y: -1; z: -1; nx:  0; ny: -1; nz:  0; color: $00FFFFFF),
    (x:  1; y: -1; z:  1; nx:  0; ny: -1; nz:  0; color: $00FFFFFF),
    (x: -1; y: -1; z:  1; nx:  0; ny: -1; nz:  0; color: $00FFFFFF),
    (x: -1; y: -1; z: -1; nx:  0; ny: -1; nz:  0; color: $00FFFFFF)
  );
```

Запустите пример из каталога Example_19 и вы снова увидите наш куб, грани которого будут отражать свет.

Туман

В подсистеме Direct3D имеются 2 типа тумана — пиксельный туман и вершинный. Каждый из них обладает своими возможностями и интерфейсом управления.

Так зачем же мы используем туман? Оказывается, добавление тумана может придать сцене еще больше реалистичности. Также туман уменьшает нагрузку на графическую подсистему, что увеличивает производительность. Это достигается за счет того, что объекты, находящиеся за границей тумана, могут не обсчитываться в сцене вовсе.

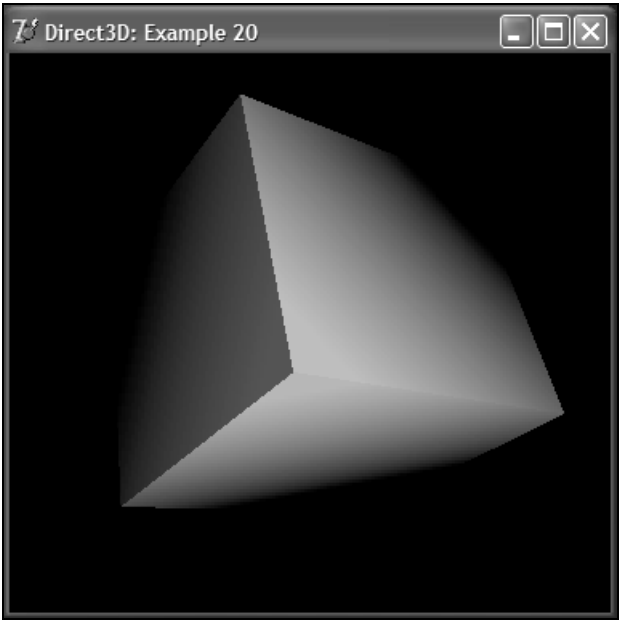


Рис. 5.27. Использование тумана

По своей сути туман является смесью цвета объекта с заданным цветом тумана, зависящим от глубины сцены или удаленности от точки просмотра. Простейший пример — это затенение объектов в зависимости от расстояния. Чем дальше объект уходит вглубь, тем больше он смешивается с цветом тумана. Посмотрите на рис. 5.27 — это наш пример из каталога Example_20, в котором куб "тонет" в черном тумане.

Пиксельный туман, называющийся также табличным туманом, реализуется драйвером устройства, а вершинный туман реализован в архитектуре освещения подсистемы Direct3D.

Режимы работы туманов обоих типов одинаковы. Это линейный туман (`D3DFOG_LINEAR`) либо один из экспоненциальных режимов тумана (`D3DFOG_EXP` и `D3DFOG_EXP2`).

Для линейного тумана характеристиками являются начальное и конечное значения границ тумана. Вообще, функция линейного тумана имеет следующий вид:

$$f = \frac{end - d}{end - start},$$

где:

- *start* — дистанция, на которой начинает действовать эффект тумана;
- *end* — дистанция, на которой эффект тумана больше не увеличивается;
- *d* — глубина или расстояние от наблюдателя.

Первый экспоненциальный режим тумана задается следующей функцией:

$$f = \frac{1}{e^{d \times density}},$$

где:

- *e* — основание натурального логарифма (приблизительно `2.71828182846`);
- *density* — плотность тумана. Значение лежит в диапазоне от `0.0` до `1.0`;
- *d* — глубина или расстояние от наблюдателя.

Для второго экспоненциального режима тумана характерна следующая формула:

$$f = \frac{1}{e^{(d \times density)^2}}.$$

Давайте перейдем от теории к практике и вернемся к нашему примеру. В примере используется табличный туман. Посмотрим, какие изменения произошли с методом прорисовки сцены (листинг 5.23).

Листинг 5.23. Метод RenderScene

```
{*********************************************************************}
{** Прорисовка сцены                                                **}
{*********************************************************************}
function TMainForm.RenderScene: HResult;
var
  WorldMatrix:  TD3DMatrix;
  WorldMatrixX: TD3DMatrix;
  WorldMatrixY: TD3DMatrix;
  WorldMatrixZ: TD3DMatrix;

  d3dMaterial: TD3DMaterial9;
  d3dVector: TD3DVector;
  d3dLight: TD3DLight9;

  FogStart, FogEnd: single;
begin
  // Чистим устройство
  Result := ClearDevice;
  // Если произошла ошибка, то завершаем работу
  if FAILED(Result) then EXIT;

  // Начало сцены
  FD3DDevice.BeginScene;
  try
    // Связываем буфер вершин с потоком данных устройства
    FD3DDevice.SetStreamSource(0, FD3DVertexBuffer, 0,
      SizeOf(TCustomVertex));

    // Устанавливаем формат вершин
    FD3DDevice.SetFVF(D3DFVF_CUSTOMVERTEX);

    // Увеличиваем угол поворота
    inc(FRotAngle, 1);
```

```
// Вращаем по всем осям x, y и z
D3DXMatrixRotationX(WorldMatrixX, GradToRad(FRotAngle));
D3DXMatrixRotationY(WorldMatrixY, GradToRad(FRotAngle));
D3DXMatrixRotationZ(WorldMatrixZ, GradToRad(FRotAngle));

// Накладываем все модификации на мировую матрицу
D3DXMatrixMultiply(WorldMatrix, WorldMatrixX, WorldMatrixY);
D3DXMatrixMultiply(WorldMatrix, WorldMatrix,  WorldMatrixZ);

// Устанавливаем мировую матрицу для куба
FD3DDevice.SetTransform(D3DTS_WORLD, WorldMatrix);

// Задаем свойства материала
ZeroMemory(@d3dMaterial, SizeOf(d3dMaterial));
d3dMaterial.Diffuse.r := 1;
d3dMaterial.Diffuse.g := 1;
d3dMaterial.Diffuse.b := 0;
d3dMaterial.Diffuse.a := 1;
d3dMaterial.Ambient := d3dMaterial.Diffuse;

// Устанавливаем материал
FD3DDevice.SetMaterial(d3dMaterial);

// Задаем параметры источника света
ZeroMemory(@d3dLight, SizeOf(d3dLight));
d3dLight._Type := D3DLIGHT_DIRECTIONAL;
d3dLight.Diffuse.r := 1;
d3dLight.Diffuse.g := 1;
d3dLight.Diffuse.b := 1;
d3dLight.Range := 20;

// Направление источника
d3dVector.x := 0;
d3dVector.y := 0;
d3dVector.z := 5;
D3DXVec3Normalize(d3dLight.Direction, d3dVector);
```

```
  // Устанавливаем параметры первого источника света сцены
  FD3DDevice.SetLight(0, d3dLight);

  // Включаем источник
  FD3DDevice.LightEnable(0, TRUE);

  // Включаем режим обработки освещения
  FD3DDevice.SetRenderState(D3DRS_LIGHTING, 1);

  // Задаем освещение окружающей среды
  FD3DDevice.SetRenderState(D3DRS_AMBIENT, $00202020);

  // Включаем поддержку тумана
  FD3DDevice.SetRenderState(D3DRS_FOGENABLE, 1);

  // Задаем режим тумана
  FD3DDevice.SetRenderState(D3DRS_FOGTABLEMODE, D3DFOG_LINEAR);

  // Устанавливаем цвет тумана
  FD3DDevice.SetRenderState(D3DRS_FOGCOLOR, $00);

  // Начало и конец полосы тумана
  FogStart := 3.5; FogEnd := 5.0;
  FD3DDevice.SetRenderState(D3DRS_FOGSTART, PDWORD(@FogStart)^);
  FD3DDevice.SetRenderState(D3DRS_FOGEND, PDWORD(@FogEnd)^);

  // Устанавливаем индексы
  FD3DDevice.SetIndices(FD3DIndexBuffer);

  // Рисуем проиндексированные примитивы
  FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 36, 0, 12);
finally

  // Завершаем сцену
  FD3DDevice.EndScene;
end;
```

```
  // Переключение буферов
  FD3DDevice.Present(NIL, NIL, 0, NIL);
end;
```

Итак, у нас появились две новые переменные, задающие границы тумана. Поддержка тумана включается методом `IDirect3DDevice9.SetRenderState` с параметром `D3DRS_FOGENABLE`, имеющим значение `1`, а поддержка табличного тумана включается тем же самым методом, только с параметром `D3DRS_FOGTABLEMODE`, имеющим значение `1`. Если вы захотите использовать не табличный туман, а вершинный, то достаточно будет изменить первый параметр с `D3DRS_FOGTABLEMODE` на `D3DRS_FOGVERTEXMODE`:

```
FD3DDevice.SetRenderState(D3DRS_FOGVERTEXMODE, D3DFOG_LINEAR);
```

Границы полосы линейного тумана устанавливаются также методом `IDirect3DDevice9.SetRenderState` с параметрами `D3DRS_FOGSTART` и `D3DRS_FOGEND`:

```
FD3DDevice.SetRenderState(D3DRS_FOGSTART, PDWORD(@FogStart)^);
FD3DDevice.SetRenderState(D3DRS_FOGEND, PDWORD(@FogEnd)^);
```

Вот собственно и все, что нам требовалось для работы с туманом.

Работа с текстурой

Довольно сложно представить себе хоть одну трехмерную компьютерную игру, в которой не используются текстуры. Именно благодаря текстурам мы можем наблюдать абсолютно естественную трехмерную картинку. К примеру, часто в играх мы можем видеть различные рельефные объекты, которые на самом деле таковыми не являются, просто на них наложена соответствующая текстура.

Текстура представляет собой обыкновенную двумерную картинку, которая накладывается на примитив, правда за одним исключением — имеются и кубические текстуры. Каждое индивидуальное цветовое значение в текстуре называется *текселем* (**Tex**ture **El**ement). Каждый тексель имеет свой уникальный адрес в текстуре. Под адресом можно понимать столбец и номер строки, которые обозначаются символами *u* и *v* соответственно.

Текстурные координаты располагаются в пространстве самой текстуры. То есть началом можно считать точку (0, 0). Когда текстура накладывается на примитив в трехмерном пространстве, то адреса всех текселов текстуры пе-

реводятся в объектные координаты. Затем они должны быть переведены в экранные координаты или позиции точек.

На рис. 5.28 наглядно показано, каким образом координаты текстуры будут соответствовать примитиву.

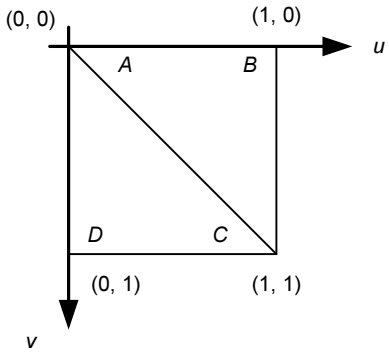


Рис. 5.28. Соответствие текстурных координат вершинам примитива

Для привязки вершин примитива к текстурным координатам нам необходимо дополнить описание вершины текстурными координатами:

```
TCustomVertex = packed record
  x,  y,  z:  Single;
  nx, ny, nz: Single;
  tu, tv:     Single;
end;
```

Мы убрали из структуры цвет, т. к. на каждую грань куба будет наложена текстура. Константа, описывающая формат точек, будет иметь следующий вид:

```
D3DFVF_CUSTOMVERTEX = D3DFVF_XYZ or D3DFVF_NORMAL or D3DFVF_TEX1;
```

Массив вершин куба дополнился координатами текстуры:

```
Vertices: array[0..23] of TCustomVertex = (
    (x: -1; y:  1; z: -1; nx:  0; ny:  0; nz: -1; tu: 0; tv: 0),
    (x:  1; y:  1; z: -1; nx:  0; ny:  0; nz: -1; tu: 1; tv: 0),
    (x:  1; y: -1; z: -1; nx:  0; ny:  0; nz: -1; tu: 1; tv: 1),
    (x: -1; y: -1; z: -1; nx:  0; ny:  0; nz: -1; tu: 0; tv: 1),
```

```
    (x: -1; y:  1; z:  1; nx: -1; ny:  0; nz:  0; tu: 0; tv: 0),
    (x: -1; y:  1; z: -1; nx: -1; ny:  0; nz:  0; tu: 1; tv: 0),
    (x: -1; y: -1; z: -1; nx: -1; ny:  0; nz:  0; tu: 1; tv: 1),
    (x: -1; y: -1; z:  1; nx: -1; ny:  0; nz:  0; tu: 0; tv: 1),

    (x:  1; y:  1; z:  1; nx:  0; ny:  0; nz:  1; tu: 0; tv: 0),
    (x: -1; y:  1; z:  1; nx:  0; ny:  0; nz:  1; tu: 1; tv: 0),
    (x: -1; y: -1; z:  1; nx:  0; ny:  0; nz:  1; tu: 1; tv: 1),
    (x:  1; y: -1; z:  1; nx:  0; ny:  0; nz:  1; tu: 0; tv: 1),

    (x:  1; y:  1; z: -1; nx:  1; ny:  0; nz:  0; tu: 0; tv: 0),
    (x:  1; y:  1; z:  1; nx:  1; ny:  0; nz:  0; tu: 1; tv: 0),
    (x:  1; y: -1; z:  1; nx:  1; ny:  0; nz:  0; tu: 1; tv: 1),
    (x:  1; y: -1; z: -1; nx:  1; ny:  0; nz:  0; tu: 0; tv: 1),

    (x: -1; y:  1; z:  1; nx:  0; ny:  1; nz:  0; tu: 0; tv: 0),
    (x:  1; y:  1; z:  1; nx:  0; ny:  1; nz:  0; tu: 1; tv: 0),
    (x:  1; y:  1; z: -1; nx:  0; ny:  1; nz:  0; tu: 1; tv: 1),
    (x: -1; y:  1; z: -1; nx:  0; ny:  1; nz:  0; tu: 0; tv: 1),

    (x:  1; y: -1; z: -1; nx:  0; ny: -1; nz:  0; tu: 0; tv: 0),
    (x:  1; y: -1; z:  1; nx:  0; ny: -1; nz:  0; tu: 1; tv: 0),
    (x: -1; y: -1; z:  1; nx:  0; ny: -1; nz:  0; tu: 1; tv: 1),
    (x: -1; y: -1; z: -1; nx:  0; ny: -1; nz:  0; tu: 0; tv: 1)
);
```

Легко заметить, что во всех гранях куба последовательность координат текстуры одна и та же.

Для управления текстурными ресурсами служит интерфейс `IDirect3DTexture9`. В примере из каталога Exacmple_21 мы будем создавать текстуру из файла Texture.bmp. Для этого у нас появился метод `CreateTextureFromFile` (листинг 5.24).

Листинг 5.24. Создание текстуры из файла

```
{*********************************************************************}
{** Создание текстуры из файла                                      **}
{*********************************************************************}
```

```
function TMainForm.CreateTextureFromFile(FileName: string): HResult;
begin
  Result := D3DXCreateTextureFromFile(FD3DDevice, PAnsiChar(FileName),
    FD3Dtexture);
end;
```

Собственно для создания текстуры из файла мы используем метод `D3DXCreateTextureFromFile` библиотеки D3DX:

```
function D3DXCreateTextureFromFile(
  Device: IDirect3DDevice9;
  pSrcFile: PChar;
  out ppTexture: IDirect3DTexture9):
HResult; stdcall; external d3dx9texDLL name 'D3DXCreateTextureFromFileA';
```

Здесь:

- `Device` — объект с интерфейсом `IDirect3DDevice9`;
- `pSrcFile` — путь к файлу;
- `ppTexture` — адрес указателя, в который будет занесен интерфейс созданного объекта текстуры.

В процедуру прорисовки сцены мы добавим всего пару строк для включения режима наложения текстуры:

```
// Установка текстуры
FD3DDevice.SetTexture(0, FD3Dtexture);

// Отключаем обработку света
FD3DDevice.SetTextureStageState(0, D3DTSS_COLOROP, D3DTA_TEXTURE);
```

Помимо наложения текстуры, мы также отключили поддержку освещенности на поверхности, покрытой текстурой. Если же нам в дальнейшем потребуется обработка света, то мы можем включить его вызовом метода `IDirect3DDevice9.SetTextureStageState` с параметром `D3DTOP_MODULATE`:

```
FD3DDevice.SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_MODULATE);
```

После запуска нашего примера мы увидим следующую картину (рис. 5.29).

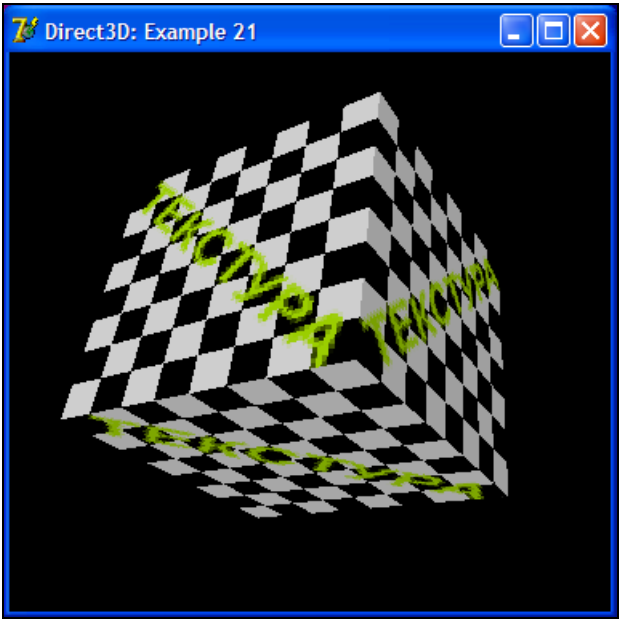


Рис. 5.29. Текстура

Фильтрация текстур

В процессе обработки трехмерного примитива подсистема Direct3D строит его двумерную проекцию на экране. Если на примитив наложена текстура, то Direct3D использует текстуру для каждой точки примитива при построении двумерного изображения. То есть для каждой точки изображения на экране подсистема должна получить значение цвета из текстуры. Этот процесс называется *процессом фильтрации текстуры.*

Предположим, что у нас на сцене обрабатывается объект, на который наложена текстура. При приближении объекта к наблюдателю мы можем заметить ухудшение качества изображения текстуры. Появляются различные изломы и т. п. Наоборот, при удалении объекта от наблюдателя качество изображения также ухудшается, оно становится неясным. Это происходит потому, что когда выполняется операция фильтрации текстуры, текстура может быть увеличена или уменьшена. Другими словами, она отображается на примитив, который может быть больше или меньше по размерам самой текстуры. Соответственно, в одном случае некоторому набору точек примитива будет соответствовать всего один тексель, а в другом случае набор тек-

селей должен быть размещен на одной точке. Собственно предыдущий пример нам все наглядно демонстрирует — мы можем наблюдать, что происходит при увеличении размера текстуры.

Подсистема Direct3D упрощает сложный процесс фильтрации текстур. Всего существуют три типа фильтрации текстур:

- ❒ линейная фильтрация;
- ❒ анизотропная фильтрация;
- ❒ многоуровневая (Mipmap) фильтрация.

Если тип фильтрации явно не указан, то используется так называемый тип фильтрации "ближайшая точка". Мы можем задать такие установки подсистемы, что при вычислении адреса текселя, не являющегося целым числом, будет скопирован цвет ближайшего текселя с целочисленным адресом. Данный метод наиболее эффективен при использовании текстуры примерно одинакового с примитивом размера. В случае неравных размеров размер текстуры лучше подогнать под размер примитива путем увеличения или уменьшения, иначе изображение может быть нечетким или на нем может появиться некая зубчатость.

В Direct3D используется так называемая билинейная фильтрация, которая является одной из форм линейной фильтрации. Подобно точечной фильтрации, при билинейной фильтрации сначала вычисляется адрес текселя текстуры, который не является целым числом, а затем находится его ближайший целочисленный сосед. Далее вычисляется среднее значение цветов текселей, находящихся слева и справа от текущего, а также сверху и снизу от него.

Искажение, которое мы видим на текселах трехмерного объекта, чья поверхность повернута под некоторым углом к экрану, называется *анизотропией*. Когда точка анизотропного примитива отображается на тексель, ее форма меняется. Подсистема Direct3D рассматривает анизотропию как некоторое удлинение экранной точки — отношение длины к ширине точки, отображенной обратно в пространство текстуры. Анизотропная фильтрация может совместно использоваться с линейной и многоуровневой фильтрациями.

Многоуровневая (Mipmap, от лат. *much in little* — многое в малом) фильтрация — фактически это набор текстур с одним и тем же изображением только разного разрешения. А нужно это для того, чтобы при увеличении объекта на него накладывалась одна текстура, а при уменьшении — другая. Если рассматривать набор текстур в сторону уменьшения разрешения, то можно заметить, что каждая последующая картинка уменьшается в размере на степень двойки. Если, к примеру, наибольший размер текстуры 256×256, то

размер следующего изображения будет 128×128, а затем 64×64. Текстура, кстати, вовсе не обязательно должна быть квадратом.

Давайте на примере посмотрим, какие преимущества дает нам использование фильтрации текстуры в своем приложении.

Для того чтобы задействовать режим фильтрации, нам необходимо вызвать метод `IDirect3DDevice9.SetSamplerState`:

```
function SetSamplerState(
  Sampler: DWORD;
  _Type: TD3DSamplerStateType;
  Value: DWORD):
HResult; stdcall;
```

Здесь:

- `Sampler` — стадия мультитекстурирования;
- `_Type` — для манипулирования режимами фильтрации должна использоваться одна из констант:
  - `D3DSAMP_MAGFILTER` — тип фильтрации при увеличении;
  - `D3DSAMP_MINFILTER` — тип фильтрации при уменьшении;
  - `D3DSAMP_MIPFILTER` — Mipmap-фильтр;
- `Value` — значение.

Последним параметром в этом методе идет значение, которое собственно и задает тип фильтрации текстуры:

- `D3DTEXF_POINT` — точечная фильтрация;
- `D3DTEXF_LINEAR` — линейная фильтрация;
- `D3DTEXF_ANISOTROPIC` — анизотропная фильтрация.

В примере из каталога Example_22 я ввел флаг текущего фильтра `FLinerFilter`, в зависимости от значения которого будет применяться режим фильтрации в процедуре прорисовки сцены (рис. 5.30):

```
// Задаем режим фильтрации текстуры
if FLinerFilter then
begin
  // Линейная фильтрация
  FD3DDevice.SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
  FD3DDevice.SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
```

```
  FD3DDevice.SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);
end
else begin
  // Точечная фильтрация
  FD3DDevice.SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_POINT);
  FD3DDevice.SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_POINT);
  FD3DDevice.SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_POINT);
end;
```

Значение флага меняется по нажатию клавиши <F5> в процедуре обработки нажатия клавиш:

```
// Меняем тип фильтрации
if Key = VK_F5 then
begin
  FLinerFilter := not FLinerFilter;
end;
```

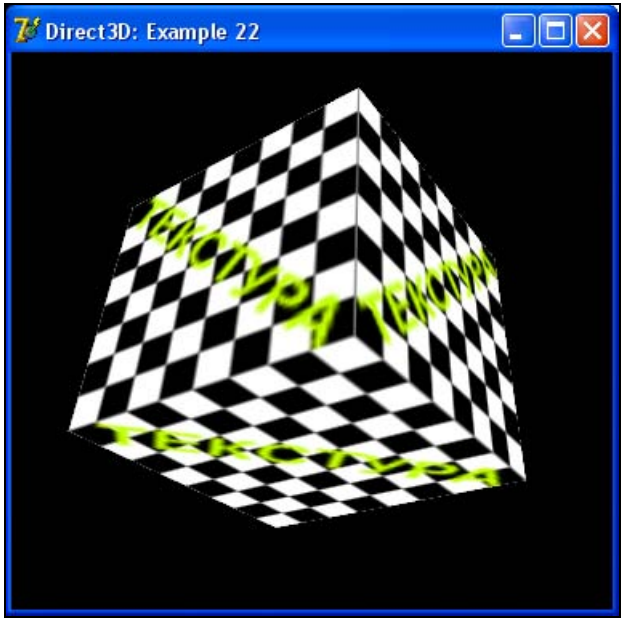


Рис. 5.30. Линейная фильтрация текстуры

Mesh-объекты

До сих пор мы с вами строили куб вручную — задавали набор вершин и индексов, описывающих построение куба в пространстве. В принципе, куб — достаточно простой объект, координаты вершин которого легко рассчитать вручную. Также мы можем просчитать координаты пирамиды и других простейших объектов. А вот когда дело дойдет до сложных объектов, например, сферы или тора — тут мы уже не на шутку призадумаемся об удобстве задания координат вручную.

Но не все так плохо, как может показаться на первый взгляд. В Direct3D есть специальный объект — так называемый mesh-объект (сеточный объект), который позволяет оперировать набором вершин и индексов. Для управления mesh-объектом предусмотрен интерфейс `ID3DXMesh` библиотеки D3DX.

Модифицируем наш пример — удалим из него всю работу с вершинным буфером, буфером индексов и текстурой. Добавим процедуру создания mesh-объектов `PrepareMeshes` (листинг 5.25).

Листинг 5.25. Создание mesh-объектов

```
{*************************************************************************}
{** Создание mesh-объектов                                               **}
{*************************************************************************}
function TMainForm.PrepareMeshes: HResult;
begin
  Result := D3DXCreateBox(FD3DDevice, 2, 2, 2, FD3DCube, NIL);
end;
```

Вывод объекта в процедуре прорисовки будет тоже достаточно прост:

```
// Выводим mesh-объект
FD3DCube.DrawSubset(0);
```

Запустите пример из каталога Example_23. На экране вы увидите вращающийся куб. Вот так, оказывается, все просто. Не нужна больше процедура заполнения вершин и задания индексов. Куб будет создан вызовом функции `D3DXCreateBox`:

```
function D3DXCreateBox(
  ppDevice: IDirect3DDevice9;
  Width,
  Height,
```

```
  Depth: Single;
  out ppMesh: ID3DXMesh;
  ppAdjacency: PID3DXBuffer):
HResult; stdcall; external d3dx9shapesDLL;
```

Здесь:

- `ppDevice` — устройство с интерфейсом `IDirect3DDevice9`;
- `Width`, `Height`, `Depth` — ширина, высота и глубина куба;
- `ppMesh` — переменная, в которую будет занесен интерфейс `ID3DXMesh` созданного объекта куба;
- `ppAdjacency` — указатель на массив смежных вершин. Можно использовать нулевое значение.

Наверно, сейчас кто-то из вас задастся вопросом — раз все так просто, так зачем же мы строили куб вручную? Может быть, нам и сферу стоило вручную построить заодно? Отвечаю — сферу нам строить вручную вовсе необязательно. Мы научимся создавать и ее, и другие объекты в следующем примере при помощи все той же библиотеки D3DX, а вот знать азы построения трехмерных объектов просто необходимо.

Несколько объектов одновременно

Библиотека D3DX содержит целый ряд функций для создания mesh-объектов. Одну из них мы уже рассмотрели — это создание куба `D3DXCreateBox`. Давайте рассмотрим еще несколько функций, которые позволяют создавать полигон, чайник, сферу, цилиндр и тор.

Создание полигона:

```
function D3DXCreatePolygon(
  ppDevice: IDirect3DDevice9;
  Length: Single;
  Sides: LongWord;
  out ppMesh: ID3DXMesh;
  ppAdjacency: PID3DXBuffer):
HResult; stdcall; external d3dx9shapesDLL;
```

Здесь:

- `ppDevice` — устройство с интерфейсом `IDirect3DDevice9`;
- `Length` — длина каждой стороны;

- `Sides` — число сторон. Значение не может быть меньше `3`;
- `ppMesh` — переменная, в которую будет занесен интерфейс `ID3DXMesh` созданного полигона;
- `ppAdjacency` — указатель на массив смежных вершин. Можно использовать нулевое значение.

Создание цилиндра:

```
function D3DXCreateCylinder(
  ppDevice: IDirect3DDevice9;
  Radius1,
  Radius2,
  Length: Single;
  Slices,
  Stacks: LongWord;
  out ppMesh: ID3DXMesh;
  ppAdjacency: PID3DXBuffer):
HResult; stdcall; external d3dx9shapesDLL;
```

Здесь:

- `ppDevice` — устройство с интерфейсом `IDirect3DDevice9`;
- `Radius1` — радиус конца цилиндра в отрицательном направлении оси *z*. Значение должно быть больше или равно `0.0`;
- `Radius2` — радиус конца цилиндра в положительном направлении оси *z*. Значение должно быть больше или равно `0.0`;
- `Length` — длина цилиндра вдоль оси *z*;
- `Slices` — число секций вдоль основной оси;
- `Stacks` — число колец вдоль основной оси;
- `ppMesh` — переменная, в которую будет занесен интерфейс `ID3DXMesh` созданного цилиндра;
- `ppAdjacency` — указатель на массив смежных вершин. Можно использовать нулевое значение.

Создание сферы:

```
function D3DXCreateSphere(
  ppDevice: IDirect3DDevice9;
  Radius: Single;
  Slices,
  Stacks: LongWord;
```

```
  out ppMesh: ID3DXMesh;
  ppAdjacency: PID3DXBuffer):
HResult; stdcall; external d3dx9shapesDLL;
```

Здесь:

- `ppDevice` — устройство с интерфейсом `IDirect3DDevice9`;
- `Radius` — радиус сферы. Значение должно быть больше или равно `0.0`;
- `Slices` — число секций вдоль основной оси;
- `Stacks` — число колец вдоль основной оси;
- `ppMesh` — переменная, в которую будет занесен интерфейс `ID3DXMesh` созданной сферы;
- `ppAdjacency` — указатель на массив смежных вершин. Можно использовать нулевое значение.

Создание тора:

```
function D3DXCreateTorus(
  ppDevice: IDirect3DDevice9;
  InnerRadius,
  OuterRadius: Single;
  Sides,
  Rings: LongWord;
  out ppMesh: ID3DXMesh;
  ppAdjacency: PID3DXBuffer):
HResult; stdcall; external d3dx9shapesDLL;
```

Здесь:

- `ppDevice` — устройство с интерфейсом `IDirect3DDevice9`;
- `InnerRadius` — внутренний радиус тора. Значение радиуса должно быть больше или равно `0.0`;
- `OuterRadius` — внешний радиус тора. Значение радиуса должно быть больше или равно `0.0`;
- `Sides` — количество сторон в поперечном сечении. Значение не может быть меньше `3`;
- `Rings` — количество колец, образующих тор. Значение не может быть меньше `3`;
- `ppMesh` — переменная, в которую будет занесен интерфейс `ID3DXMesh` созданного тора;
- `ppAdjacency` — указатель на массив смежных вершин. Можно использовать нулевое значение.

Создание чайника:

```
function D3DXCreateTeapot(
  ppDevice: IDirect3DDevice9;
  out ppMesh: ID3DXMesh;
  ppAdjacency: PID3DXBuffer):
HResult; stdcall; external d3dx9shapesDLL;
```

Здесь:

- `ppDevice` — устройство с интерфейсом `IDirect3DDevice9`;
- `ppMesh` — переменная, в которую будет занесен интерфейс `ID3DXMesh` созданного чайника;
- `ppAdjacency` — указатель на массив смежных вершин. Можно использовать нулевое значение.

Как мы можем заметить, для создания чайника не указано никаких размеров. Это означает, что для изменения его размеров нужно применять масштабирование.

Когда я рассказывал о матричных построениях, то упомянул мировую матрицу — матрицу, локальную для объекта сцены. До сих пор мы выводили всего один объект на сцене и особой надобности в использовании возможностей данной матрицы не испытывали. Теперь же мы будем выводить все шесть mesh-объектов. Процедура их создания будет выглядеть так, как показано в листинге 5.26.

Листинг 5.26. Создание шести mesh-объектов

```
{*************************************************************************}
{** Создание mesh-объектов                                              **}
{*************************************************************************}
function TMainForm.PrepareMeshes: HResult;
begin
  // Куб
  Result := D3DXCreateBox(FD3DDevice, 1, 1, 1, FD3DCube, NIL);
  if FAILED(Result) then EXIT;

  // Top
  Result := D3DXCreateTorus(FD3DDevice, 0.2, 0.3, 30, 30, FD3DTorus,
    NIL);
  if FAILED(Result) then EXIT;
```

```
  // Цилиндр
  Result := D3DXCreateCylinder(FD3DDevice, 0.15, 0.3, 2, 20, 20,
    FD3DCylinder, NIL);
  if FAILED(Result) then EXIT;

  // Чайник
  Result := D3DXCreateTeapot(FD3DDevice, FD3DTeaport, NIL);
  if FAILED(Result) then EXIT;

  // Сфера
  Result := D3DXCreateSphere(FD3DDevice, 0.5, 30, 30, FD3DSphere, NIL);
  if FAILED(Result) then EXIT;

  // Полигон
  Result := D3DXCreatePolygon(FD3DDevice, 0.5, 5, FD3DPolygon, NIL);
  if FAILED(Result) then EXIT;
end;
```

Блок, устанавливающий параметры света и окружающей среды, будет перенесен прямо в процедуру инициализации подсистемы.

Если бы мы оставили вывод каждого mesh-объекта прямо в процедуре прорисовки сцены, то ее код разросся бы до неимоверных размеров. Поэтому для установки материала mesh-объекта и его прорисовки я написал отдельный метод (листинг 5.27).

Листинг 5.27. Вывод mesh-объекта

```
{*************************************************************************}
{** Вывод mesh-объекта                                                  **}
{*************************************************************************}
function TMainForm.DrawMeshObject(Mesh: ID3DXMesh; Angle: Single;
  Material: TD3DColorValue): HResult;
var
  WorldMatrix:  TD3DMatrix;
  WorldMatrixX: TD3DMatrix;
  WorldMatrixY: TD3DMatrix;
  WorldMatrixZ: TD3DMatrix;
```

```
  d3dMaterial: TD3DMaterial9;
begin
  // Задаем свойства материала
  ZeroMemory(@d3dMaterial, SizeOf(d3dMaterial));
  d3dMaterial.Diffuse := Material;
  d3dMaterial.Ambient := d3dMaterial.Diffuse;

  // Устанавливаем материал
  FD3DDevice.SetMaterial(d3dMaterial);

  // Получение единичной матрицы
  D3DXMatrixIdentity(WorldMatrix);

  // Установка положения
  D3DXMatrixTranslation(WorldMatrix, -
    Sin(GradToRad(FRotAngle + Angle)) * 2,
    Cos(GradToRad(FRotAngle + Angle)) * 2, 0);

  // Поворот по всем трем осям x, y и z
  D3DXMatrixRotationX(WorldMatrixX, GradToRad(FRotAngle));
  D3DXMatrixRotationY(WorldMatrixY, GradToRad(FRotAngle));
  D3DXMatrixRotationZ(WorldMatrixZ, GradToRad(FRotAngle));

  // Накладываем все модификации на мировую матрицу
  D3DXMatrixMultiply(WorldMatrix, WorldMatrix, WorldMatrixX);
  D3DXMatrixMultiply(WorldMatrix, WorldMatrix, WorldMatrixY);
  D3DXMatrixMultiply(WorldMatrix, WorldMatrix, WorldMatrixZ);

  // Устанавливаем мировую матрицу для куба
  FD3DDevice.SetTransform(D3DTS_WORLD, WorldMatrix);

  // Выводим mesh-объект
  Result := Mesh.DrawSubset(0);
end;
```

После всего этого процедура прорисовки сцены сильно упростилась (листинг 5.28).

Листинг 5.28. Измененная процедура прорисовки сцены

```
{*********************************************************************}
{** Прорисовка сцены                                                **}
{*********************************************************************}
function TMainForm.RenderScene: HResult;
begin
  // Чистим устройство
  Result := ClearDevice;
  // Если произошла ошибка, то завершаем работу
  if FAILED(Result) then EXIT;

  // Начало сцены
  FD3DDevice.BeginScene;
  try
    // Увеличиваем угол поворота
    inc(FRotAngle, 1);

    // Рисуем mesh-объекты:
    //   тор
    DrawMeshObject(FD3DTorus,      0, D3DXColor(  1,    1,    1,  1));

    //   цилиндр
    DrawMeshObject(FD3DCylinder,  60, D3DXColor(  1,  0.1,  0.4,  1));

    //   чайник
    DrawMeshObject(FD3DTeaport,  120, D3DXColor(0.2,  0.4,  0.1,  1));

    //   полигон
    DrawMeshObject(FD3DPolygon,  180, D3DXColor(  2,    5,    5,  1));

    //   сфера
    DrawMeshObject(FD3DSphere,   240, D3DXColor(0.1,  0.2,  0.9,  1));
```

```
    //   куб
    DrawMeshObject(FD3DCube,     300, D3DXColor(0.8,    1,  0.1,  1));
  finally

    // Завершаем сцену
    FD3DDevice.EndScene;
  end;

  // Переключение буферов
  FD3DDevice.Present(NIL, NIL, 0, NIL);
end;
```

Алгоритм работы достаточно прост:

1. Инициализация Direct3D.
2. Создание mesh-объектов.
3. Прорисовка сцены.
4. Освобождение ресурсов и завершение работы приложения.

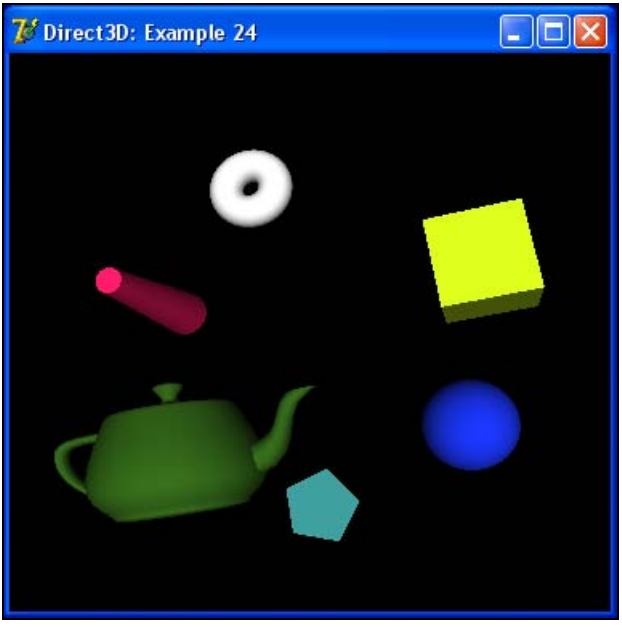


Рис. 5.31. Вывод нескольких объектов

Все объекты на сцене располагаются на одинаковом расстоянии друг от друга по кругу. Вращение объектов производится также по кругу, и в то же время каждый из объектов вращается самостоятельно в пространстве по осям *x*, *y* и *z*.

Посмотрите пример из каталога Example_24. В результате его работы на экране вы увидите подобное представленному на рис. 5.31.

Работа с текстом на плоскости и в пространстве

Начнем изучение работы с текстом с построений на плоскости. Для вывода текста мы будем использовать интерфейс `ID3DXFont`. Этот интерфейс позволяет выводить текст на текстурах и ресурсах, которые нуждаются в специфичном шрифте.

Для чего нам может потребоваться вывод текста? Да элементарно — для любой справочной информации, для общения между участниками сетевой игры и т. д.

Создание объекта шрифта производится при помощи метода `D3DXCreateFont`:

```
function D3DXCreateFont(
  pDevice: IDirect3DDevice9;
  Height: Integer;
  Width: Longint;
  Weight: LongWord;
  MipLevels: LongWord;
  Italic: BOOL;
  CharSet: DWORD;
  OutputPrecision: DWORD;
  Quality: DWORD;
  PitchAndFamily: DWORD;
  pFaceName: PChar;
  out ppFont: ID3DXFont):
HResult; stdcall; external d3dx9coreDLL name 'D3DXCreateFontA';
```

Здесь:

- `pDevice` — устройство с интерфейсом `IDirect3DDevice9`;
- `Height` — высота шрифта в логических единицах;

- `Width` — ширина шрифта в логических единицах;
- `Weight` — так называемый "вес" шрифта;
- `MipLevels` — число уровней Mipmap;
- `Italic` — использовать курсив или нет;
- `CharSet` — набор символов шрифта (кодировка);
- `OutputPrecision` — точность отсечения;
- `Quality` — качество вывода;
- `PitchAndFamily` — семейство и шаг;
- `pFaceName` — название шрифта;
- `ppFont` — переменная, которая получит интерфейс `ID3DXFont` созданного объекта.

В примере из каталога Example_25 создание шрифта вынесено в отдельную функцию (листинг 5.29).

Листинг 5.29. Создание шрифта

```
{*********************************************************************}
{** Создание объекта для работы с текстом                           **}
{*********************************************************************}
function TMainForm.CreateText: HResult;
var
  DC: HDC;
begin
  DC := GetDC(0);
  try
    Result := D3DXCreateFont(FD3DDevice, -MulDiv(9, GetDeviceCaps(DC,
      LOGPIXELSY), 72), 0, FW_BOLD, 0, FALSE, DEFAULT_CHARSET,
      OUT_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH or FF_DONTCARE,
      'Arial', FD3DText);
  finally
    ReleaseDC(0, DC);
  end;
end;
```

Для вывода текста имеется следующий метод — `ID3DXFont.DrawTextA`:

```
function DrawTextA(
```

```
  pSprite: ID3DXSprite;
  pString: PAnsiChar;
  Count: Integer;
  pRect: PRect;
  Format: DWORD;
  Color: TD3DColor):
Integer; stdcall;
```

Здесь:

- `pSprite` — указатель на объект с интерфейсом `ID3DXSprite`. Может принимать нулевое значение;
- `pString` — указатель на строку текста. Если параметр `Count` равен `-1`, то строка должна заканчиваться нулевым символом;
- `Count` — число символов в строке;
- `pRect` — указатель на прямоугольник, заданный в логических координатах, в который должен быть выведен текст;
- `Format` — формат текста. Значение может быть комбинацией следующих констант:
  - `DT_BOTTOM` — выравнивание текста по нижней границе. Значение должно быть в паре с константой `DT_SINGLELINE`;
  - `DT_CALCRECT` — определение ширины и высоты нужной прямоугольной области;
  - `DT_CENTER` — выравнивание текста по горизонтали;
  - `DT_EXPANDTABS` — обработка символов табуляции;
  - `DT_LEFT` — выравнивание текста по левому краю;
  - `DT_NOCLIP` — вывод текса без обрезания;
  - `DT_RIGHT` — выравнивание текста по правой границе;
  - `DT_RTLREADING` — вывод текста справа налево;
  - `DT_SINGLELINE` — вывод текста в виде одной строки;
  - `DT_TOP` — выравнивание текста по верхней границе;
  - `DT_VCENTER` — выравнивание текста по вертикали (только для одной строки);
  - `DT_WORDBREAK` — автоматическое разбиение текста на строки при выходе за границы;
- `Color` — цвет выводимого текста.

Вывод текста у нас будет осуществляться в методе `TextOut` (листинг 5.30).

Листинг 5.30. Метод `TextOut`

```
{*********************************************************************}
{** Вывод текста в заданную область заданным цветом                  **}
{*********************************************************************}
procedure TMainForm.TextOut(S: string; R: PRect; TextColor: TD3DColor);
begin
  if (FD3DText <> NIL) then
  begin
    FD3DText.DrawTextA(NIL, PChar(S), Length(S), R, DT_WORDBREAK,
      TextColor);
  end;
end;
```

Ну и собственно из процедуры прорисовки сцены мы будем вызывать метод вывода текста на экран. А выводить в качестве примера мы станем довольно известное выражение Шекспира (листинг 5.31).

Листинг 5.31. Вывод текста

```
{*********************************************************************}
{** Вывод различной текстовой информации                             **}
{*********************************************************************}
procedure TMainForm.DrawTextMessage;
var
  rectMain,
  rectName: TRect;
  d3dViewport: TD3DViewport9;
begin
  // Получаем параметры окна вывода
  FD3DDevice.GetViewport(d3dViewport);

  // Области надписей
  rectMain := Rect(0, 0, 130, 90);
  rectName := Rect(0, 0,  80, 95);

  // Выводим текст
  OffsetRect(rectMain, d3dViewport.Width - 120, 10);
```

```
  TextOut(PChar('Быть или не быть - вот в чем вопрос...'), @rectMain,
    D3DCOLOR_XRGB(255, 0, 0));
  OffsetRect(rectMain, 2, 2);
  TextOut(PChar('Быть или не быть - вот в чем вопрос...'), @rectMain,
    D3DCOLOR_XRGB(50, 255, 50));

  OffsetRect(rectName, d3dViewport.Width - 80, 45);
  TextOut(PChar('(c) Шекспир'), @rectName, D3DCOLOR_XRGB(255, 0, 0));
  OffsetRect(rectName, 2, 2);
  TextOut(PChar('(c) Шекспир'), @rectName, D3DCOLOR_XRGB(50, 255, 50));
end;
```

На рис. 5.32 представлена работа нашего примера.

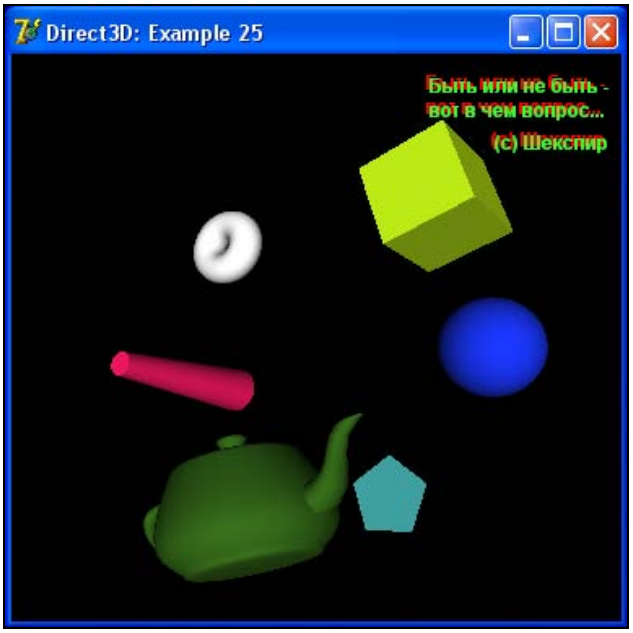


Рис. 5.32. Вывод двумерного текста

Из примера видно, что мы выводим каждый блок текста дважды — с небольшим смещением и изменением цвета. Это нужно для того, чтобы создать видимость трехмерного текста. И в некоторых случаях такое решение

вполне подходит. Но как же быть, когда нам действительно понадобится трехмерный текст?

Оказывается, в библиотеке D3DX имеется специальная функция для создания трехмерного текста:

```
function D3DXCreateText(
  ppDevice: IDirect3DDevice9;
  hDC: HDC;
  pText: PChar;
  Deviation: Single;
  Extrusion: Single;
  out ppMesh: ID3DXMesh;
  ppAdjacency: PID3DXBuffer;
  pGlyphMetrics: PGlyphMetricsFloat):
HResult; stdcall; external d3dx9shapesDLL name 'D3DXCreateTextA';
```

Здесь:

- `ppDevice` — устройство с интерфейсом `IDirect3DDevice9`;
- `hDC` — контекст устройства, содержащего нужный шрифт;
- `pText` — строка текста;
- `Deviation` — максимальное отклонение от шрифта;
- `Extrusion` — значение выдавливания по оси *z*;
- `ppMesh` — переменная, в которую будет занесен интерфейс `ID3DXMesh` созданного текста;
- `ppAdjacency` — указатель на массив смежных вершин. Можно использовать нулевое значение;
- `pGlyphMetrics` — массив структур, в который будут записаны параметры каждого символа строки. Может использоваться нулевое значение.

А точнее, данная функция создает исходя из параметров шрифта специальный mesh-объект, представляющий собой заданный текст в виде трехмерного объекта.

Рассмотрим пример из каталога Example_26. У нас имеется один mesh-объект, который представляет собой трехмерный текст (листинг 5.32).

Листинг 5.32. Создание трехмерного текста

```
{*********************************************************************}
{** Создание трехмерного текста                                      **}
{*********************************************************************}
```

```
function TMainForm.Create3DText: HResult;
var
  DC: HDC;
  nHeight: integer;
  hhFont: HFONT;
  hhFontOld: HFONT;
begin
  FD3DText3D := NIL;
  DC := CreateCompatibleDC(0);

  // Вычисляем высоту
  nHeight := -MulDiv(9, GetDeviceCaps(DC, LOGPIXELSY), 72);

  // Создаем шрифт
  hhFont := CreateFont(nHeight, 0, 0, 0, FW_NORMAL, 0, 0, 0,
    RUSSIAN_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
    DEFAULT_QUALITY, DEFAULT_PITCH or FF_DONTCARE, 'Times New Roman');

  hhFontOld := SelectObject(DC, hhFont);

  // Создаем трехмерный текст
  Result := D3DXCreateText(FD3DDevice, DC, 'Текст 3D',
    0.001, 0.4, FD3DText3d, NIL, NIL);

  SelectObject(DC, hhFontOld);
  DeleteObject(hhFont);
  DeleteDC(DC);
end;
```

Соответственно, его прорисовка не будет ничем отличаться от прорисовки обычного mesh-объекта:

```
DrawMeshObject(FD3DText3D, 300, D3DXColor(0.8, 0.7, 0.9, 1));
```

На рис. 5.33 показан результат работы нашего примера. В примере используются одновременно вывод двумерного текста и трехмерного.

Рис. 5.33. Вывод трехмерного текста

Подсчет числа кадров в секунду

А теперь давайте рассмотрим, каким образом ведется подсчет числа кадров в секунду в графическом приложении. Наверняка многим из вас доводилось видеть такую аббревиатуру, как FPS (Frames Per Second, число кадров в секунду). Многие игры и различные графические приложения выводят это значение специально, с целью показать производительность графической системы.

Рассчитать данное значение достаточно легко: в процедуру прорисовки сцены мы добавим счетчик кадров — прорисовали сцену, увеличили число кадров на единицу. Затем раз в секунду пересчитываем число прорисованных кадров и выводим результат на экран.

Соответственно, в процедуру прорисовки сцены добавится код увеличения числа прорисованных кадров:

```
// Увеличиваем счетчик кадров
inc(FFrameCount);
```

А процедура вывода текстовой информации претерпит некоторые изменения (листинг 5.33).

Листинг 5.33. Процедура вывода текста

```
{*********************************************************************}
{** Вывод различной текстовой информации                            **}
{*********************************************************************}
procedure TMainForm.DrawTextMessage;
var
  rectMain,
  rectName,
  rectFPS: TRect;
  d3dViewport: TD3DViewport9;
  FEndTime: DWORD;
begin
  // Получаем параметры окна вывода
  FD3DDevice.GetViewport(d3dViewport);
  // Области надписей
  rectMain := Rect(0, 0, 130, 90);
  rectName := Rect(0, 0,  80, 95);

  // Выводим текст
  OffsetRect(rectMain, d3dViewport.Width - 120, 10);
  TextOut(PChar('Быть или не быть - вот в чем вопрос...'), @rectMain,
    D3DCOLOR_XRGB(255, 0, 0));
  OffsetRect(rectMain, 2, 2);
  TextOut(PChar('Быть или не быть - вот в чем вопрос...'), @rectMain,
    D3DCOLOR_XRGB(50, 255, 50));

  OffsetRect(rectName, d3dViewport.Width - 80, 45);
  TextOut(PChar('(c) Шекспир'), @rectName, D3DCOLOR_XRGB(255, 0, 0));
  OffsetRect(rectName, 2, 2);
  TextOut(PChar('(c) Шекспир'), @rectName, D3DCOLOR_XRGB(50, 255, 50));

  // Считаем число кадров в секунду
  FEndTime := GetTickCount;
  if FEndTime - FStartTime > 1000 then
```

```
  begin
    FFPS := FFrameCount / (FEndTime - FStartTime) * 1000;
    FStartTime := FEndTime;
    FFrameCount := 0;
  end;

  // Выводим число кадров в секунду
  rectFPS  := Rect(0, 0,  70, 25);
  OffsetRect(rectFPS, 10, 5);
  TextOut(PChar(Format('FPS=%2.2f', [FFPS])), @rectFPS,
    D3DCOLOR_XRGB(255, 0, 50));
  OffsetRect(rectFPS, 2, 2);
  TextOut(PChar(Format('FPS=%2.2f', [FFPS])), @rectFPS, D3DCOLOR_XRGB(50,
    255, 50));
end;
```

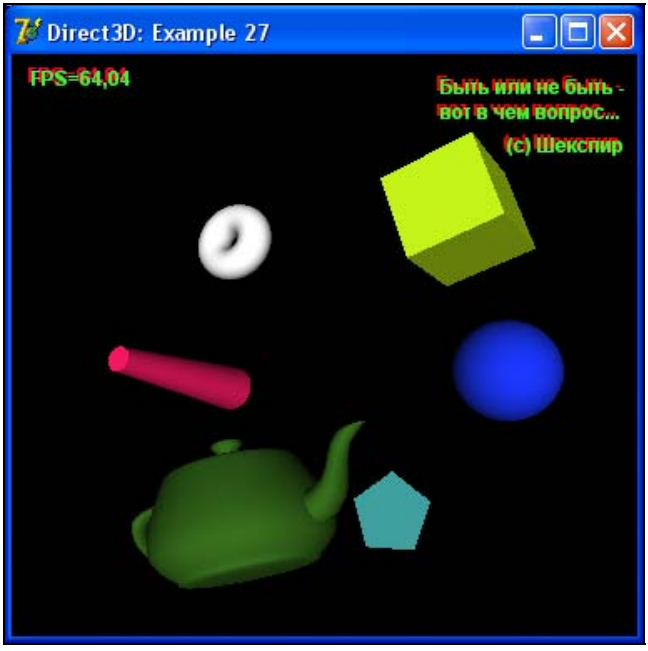


Рис. 5.34. Подсчет числа кадров в секунду

Посмотрите на последний пример из каталога Example_27. В левом верхнем углу вы увидите число просчитанных в секунду кадров (рис. 5.34). Однако стоит заметить, что на разных компьютерах с разной производительностью это число может быть совсем другим.

Несколько текстур на одном объекте

В одном из предыдущих примеров мы с вами накладывали текстуру на куб. У нас одна и та же текстура накладывалась на все грани. В целом получилось неплохо. Но иногда может потребоваться наложить разные текстуры на различные грани объекта. В качестве объекта будет выступать все тот же куб. Для него нам требуется подготовить и загрузить 6 текстур — по одной на каждую грань куба (листинг 5.34).

Листинг 5.34. Создание текстур

```
{*********************************************************************}
{** Создание текстур                                                 **}
{*********************************************************************}
function TMainForm.CreateTextures: HResult;
begin
  Result := D3DXCreateTextureFromFile(FD3DDevice, 'Texture1.bmp',
    FD3Dtexture1);
  if FAILED(Result) then EXIT;

  Result := D3DXCreateTextureFromFile(FD3DDevice, 'Texture2.bmp',
    FD3Dtexture2);
  if FAILED(Result) then EXIT;

  Result := D3DXCreateTextureFromFile(FD3DDevice, 'Texture3.bmp',
    FD3Dtexture3);
  if FAILED(Result) then EXIT;

  Result := D3DXCreateTextureFromFile(FD3DDevice, 'Texture4.bmp',
    FD3Dtexture4);
  if FAILED(Result) then EXIT;
```

```
  Result := D3DXCreateTextureFromFile(FD3DDevice, 'Texture5.bmp',
    FD3Dtexture5);
  if FAILED(Result) then EXIT;

  Result := D3DXCreateTextureFromFile(FD3DDevice, 'Texture6.bmp',
    FD3Dtexture6);
end;
```

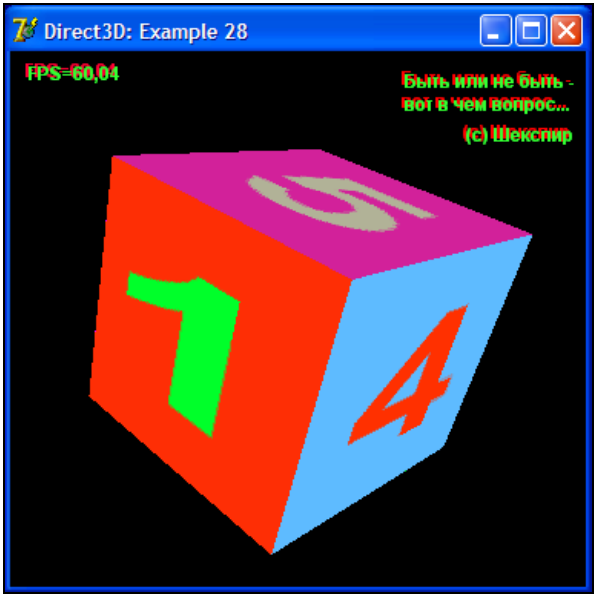


Рис. 5.35. Различные текстуры на гранях куба

Затем необходимо переписать блок прорисовки куба на сцене и рисовать каждую грань по отдельности, задавая каждый раз нужную текстуру:

```
// Установка текстуры и рисование куба по граням
FD3DDevice.SetTexture(0, FD3Dtexture1);
FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST,  0, 0, 4, 0, 2);

FD3DDevice.SetTexture(0, FD3Dtexture2);
FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST,  4, 0, 4, 0, 2);

FD3DDevice.SetTexture(0, FD3Dtexture3);
FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST,  8, 0, 4, 0, 2);
```

```
FD3DDevice.SetTexture(0, FD3Dtexture4);
FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 12, 0, 4, 0, 2);

FD3DDevice.SetTexture(0, FD3Dtexture5);
FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 16, 0, 4, 0, 2);

FD3DDevice.SetTexture(0, FD3Dtexture6);
FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 20, 0, 4, 0, 2);
```

Пример из каталога Example_28 (рис. 5.35) поможет вам лучше усвоить материал.

Сферические текстурные координаты

Итак, с кубом мы разобрались. Осталось теперь только изучить, как накладывать текстуру на mesh-объекты. Если кто-то думает, что это очень просто, то он заблуждается. Чтобы наложить текстуру на mesh-объект, нам необходимо вручную рассчитать текстурные координаты *u* и *v* для каждой вершины. И, если для куба более-менее понятно, как это сделать, то для таких сложных объектов, как например сфера, это сделать достаточно непросто.

Один из способов генерации подобных текстурных координат называется *сферическим преобразованием* или *сферической картой наложения* (Spherical Mapping). Координаты *u* и *v* рассчитываются исходя из координат *x* и *y* соответствующих нормалей по следующим формулам:

$$u = \frac{N_x}{2} + 0.5,$$

$$v = \frac{N_y}{2} + 0.5,$$

где *u* и *v* — текстурные координаты, а N_x и N_y — компоненты *x* и *y* соответствующей нормали. Несмотря на свою простоту, эта формула является очень эффективной.

Рассмотрим пример, располагающийся в папке Example_29. Я решил показать на примере чайника наложение текстуры на mesh-объект. Основной интерес для нас представляет процедура создания mesh-объектов. Первым делом мы создаем сам mesh-объект, чайник, и формируем объект текстуры на основе файла. Затем создаем клон чайника с нужным нам форматом вершин и освобождаем память, выделенную под объект первого чайника. Блокируем доступ к вершинам и меняем значения текстурных координат

всех вершин. Разблокируем буфер. Копируем mesh-объект и освобождаем память, выделенную под клонированный объект. Все эти действия будут производиться в методе `PrepareMeshes` (листинг 5.35).

Листинг 5.35. Метод `PrepareMeshes`

```
{*********************************************************************}
{** Создание mesh-объектов                                          **}
{*********************************************************************}
function TMainForm.PrepareMeshes: HResult;
type
  PCustomVertex = ^TCustomVertex;
  TCustomVertex = packed record
    position: TD3DXVector3;
    normal: TD3DXVector3;
    tu: Single;
    tv: Single;
  end;

const
  FVF_VERTEX = D3DFVF_XYZ or D3DFVF_NORMAL or D3DFVF_TEX1;

var
  Mesh: ID3DXMesh;
  pBuf: PCustomVertex;
  nVertex: integer;
  I: integer;
begin
  // Создаем чайник
  Result := D3DXCreateTeapot(FD3DDevice, FD3DTeaport, NIL);
  if FAILED(Result) then EXIT;

  // Создаем текстуру
  Result := D3DXCreateTextureFromFile(FD3DDevice, 'Texture.bmp',
    FD3DXTexture);
  if FAILED(Result) then EXIT;
```

```
  // Создаем клон чайника
  if FAILED(FD3DTeaport.CloneMeshFVF(D3DXMESH_SYSTEMMEM, FVF_VERTEX,
       FD3DDevice, Mesh)) then EXIT;
  // Освобождаем память
  FD3DTeaport := NIL;

  try
    // Блокируем доступ к буферу вершин
    Result := Mesh.LockVertexBuffer(0, pointer(pBuf));
    if FAILED(Result) then EXIT;

    // Получаем число вершин
    nVertex := Mesh.GetNumVertices;

    // Цикл по вершинам
    for I := 0 to nVertex - 1 do
    begin
      // Задаем значения u и v
      pBuf^.tu := pBuf^.normal.x / 2 + 0.5;
      pBuf^.tv := pBuf^.normal.y / 2 + 0.5;

      // Переходим к следующей вершине
      inc(pBuf);
    end;

  finally
    // Разблокируем буфер
    Mesh.UnlockVertexBuffer;

    // Копируем чайник
    FD3DTeaport := Mesh;

    // Освобождаем ресурсы
    Mesh := NIL;
  end;
end;
```

В процедуре прорисовки сцены добавится только код задания текстуры:

```
// Установка текстуры
FD3DDevice.SetTexture(0, FD3DXTexture);

// Выводим mesh-объект
FD3DTeaport.DrawSubset(0);
```

На рис. 5.36 представлена работа данного примера.

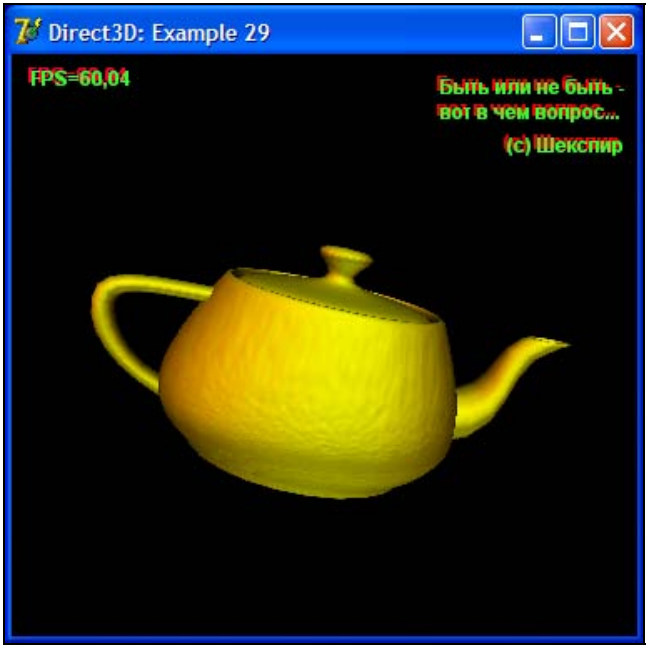


Рис. 5.36. Текстура на чайнике

Создаем туннель

Во многих играх или заставках к ним мы можем наблюдать такой интересный эффект, как полет по туннелю. Если определенное время смотреть в глубь туннеля, то создается впечатление, что мы куда-то летим вглубь. Я хотел бы привести свой пример реализации подобного туннеля на основе mesh-объекта, именуемого тором.

Посмотрите пример из каталога Example_30 (рис. 5.37). У нас на сцене будет присутствовать всего один тор, на который наложена текстура. Камера будет располагаться внутри тора, и в то же время тор будет вращаться по оси *z*, что и будет собственно выглядеть как путешествие по туннелю.

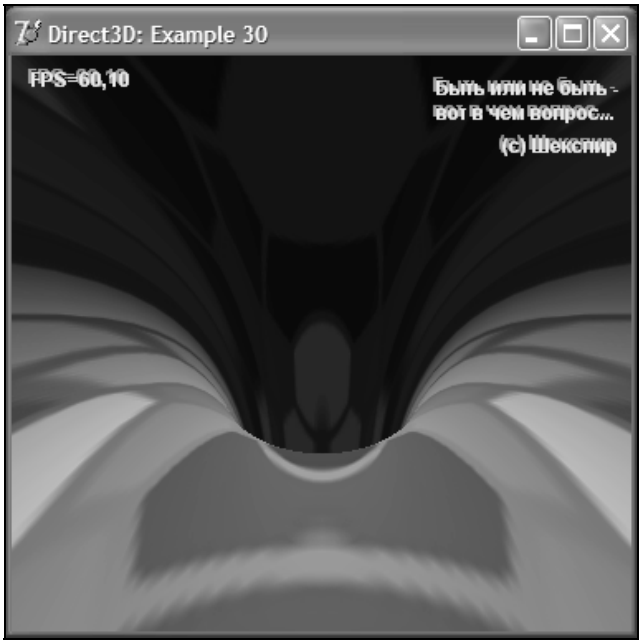


Рис. 5.37. Туннель

Прозрачность

В играх и прочих графических приложениях мы можем наблюдать такую интересную особенность при прорисовке сцены, как частичная прозрачность некоторых объектов, что добавляет сцене реализма, да и просто радует глаз.

Мы рассмотрим два способа задания прозрачности объектам сцены. В первом случае мы задействуем альфа-составляющую материала, а во втором случае — альфа-составляющую текстуры.

Рассмотрим пример из каталога Example_31 (рис. 5.38). Мы будем рисовать один куб внутри другого. Внутренний (малый) куб не имеет прозрачности. Для задания прозрачности внешнему (большому) кубу мы должны задать

параметр альфа-составляющей материала меньше единицы. Единичное значение соответствует отсутствию прозрачности, а нулевое задает полную прозрачность, т. е. объект не будет виден вовсе.

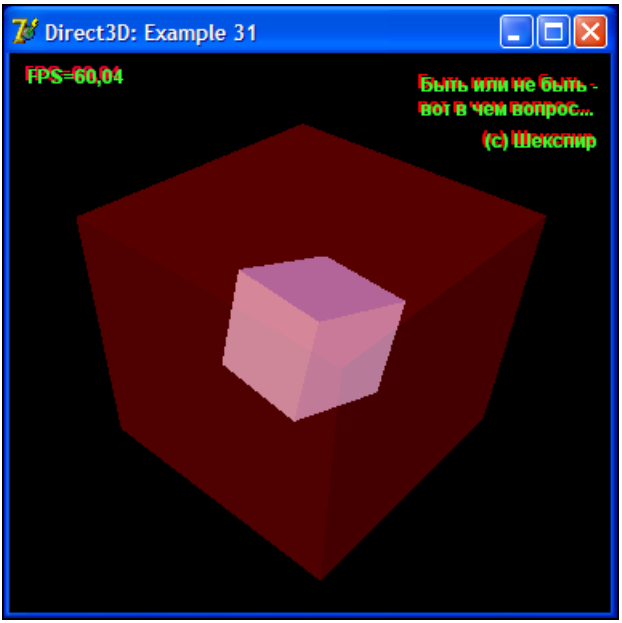


Рис. 5.38. Прозрачность
с использованием свойств материала

Необходимо еще учесть и то, что для прорисовки прозрачного объекта нам придется рисовать его грани дважды. Причем меняя направление режима отсечения на противоположный режим:

```
// Рисуем малый куб


// Вращаем по всем осям x, y и z
D3DXMatrixRotationX(WorldMatrixX, GradToRad(FRotAngle));
D3DXMatrixRotationY(WorldMatrixY, GradToRad(FRotAngle));
D3DXMatrixRotationZ(WorldMatrixZ, GradToRad(FRotAngle));

// Накладываем все модификации на мировую матрицу
D3DXMatrixMultiply(WorldMatrix, WorldMatrixX, WorldMatrixY);
D3DXMatrixMultiply(WorldMatrix, WorldMatrix,  WorldMatrixZ);
```

```
// Уменьшаем куб в размере
D3DXMatrixIdentity(TempMatrix);
D3DXMatrixScaling(TempMatrix, 0.4, 0.4, 0.4);
D3DXMatrixMultiply(WorldMatrix, WorldMatrix,  TempMatrix);

// Устанавливаем мировую матрицу для куба
FD3DDevice.SetTransform(D3DTS_WORLD, WorldMatrix);

// Задаем свойства материала
ZeroMemory(@d3dMaterial, SizeOf(d3dMaterial));
d3dMaterial.Diffuse.r := 0.5;
d3dMaterial.Diffuse.g := 0.5;
d3dMaterial.Diffuse.b := 1.0;
d3dMaterial.Diffuse.a := 1;
d3dMaterial.Ambient := d3dMaterial.Diffuse;

// Устанавливаем материал
FD3DDevice.SetMaterial(d3dMaterial);

// Устанавливаем режим отсечения
FD3DDevice.SetRenderState(D3DRS_CULLMODE, D3DCULL_CCW);

// Рисуем проиндексированные примитивы
FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 36, 0, 12);

// Включаем прозрачность
FD3DDevice.SetRenderState(D3DRS_ALPHABLENDENABLE, 1);
FD3DDevice.SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
FD3DDevice.SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA);

// Рисуем большой куб

// Вращаем по всем осям x, y и z в обратном направлении
D3DXMatrixRotationX(WorldMatrixX, -GradToRad(FRotAngle));
D3DXMatrixRotationY(WorldMatrixY, -GradToRad(FRotAngle));
D3DXMatrixRotationZ(WorldMatrixZ, -GradToRad(FRotAngle));
```

```
// Накладываем все модификации на мировую матрицу
D3DXMatrixMultiply(WorldMatrix, WorldMatrixX, WorldMatrixY);
D3DXMatrixMultiply(WorldMatrix, WorldMatrix,  WorldMatrixZ);

// Увеличиваем в размере до нормального
D3DXMatrixIdentity(TempMatrix);
D3DXMatrixScaling(TempMatrix, 1.0, 1.0, 1.0);
D3DXMatrixMultiply(WorldMatrix, WorldMatrix,  TempMatrix);

// Устанавливаем мировую матрицу для куба
FD3DDevice.SetTransform(D3DTS_WORLD, WorldMatrix);

// Задаем свойства материала
ZeroMemory(@d3dMaterial, SizeOf(d3dMaterial));
d3dMaterial.Diffuse.r := 1;
d3dMaterial.Diffuse.g := 0;
d3dMaterial.Diffuse.b := 0;
d3dMaterial.Diffuse.a := 0.4;
d3dMaterial.Ambient := d3dMaterial.Diffuse;

// Устанавливаем материал
FD3DDevice.SetMaterial(d3dMaterial);

// Устанавливаем режим отсечения
FD3DDevice.SetRenderState(D3DRS_CULLMODE, D3DCULL_CW);
// Рисуем проиндексированные примитивы
FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 36, 0, 12);

// Устанавливаем режим отсечения
FD3DDevice.SetRenderState(D3DRS_CULLMODE, D3DCULL_CCW);
// Рисуем проиндексированные примитивы
FD3DDevice.DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, 36, 0, 12);

// Выключаем прозрачность
FD3DDevice.SetRenderState(D3DRS_ALPHABLENDENABLE, 0);
```

Второй способ задания прозрачности основан на использовании альфа-канала текстуры. Если мы выбираем такой формат файла текстуры, как TGA, то в нем могут присутствовать не только слои R, G и B, но и альфа-канал. Пример из каталога Example_32 (рис. 5.39) показывает нам использование альфа-канала текстуры.

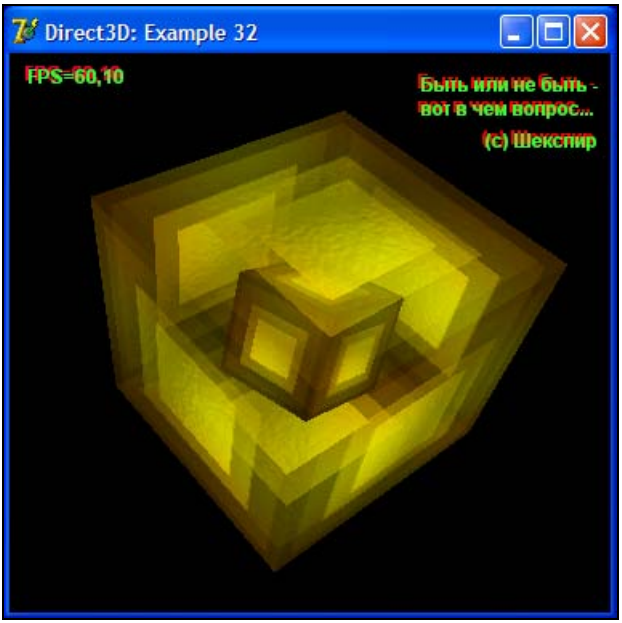


Рис. 5.39. Прозрачность с использованием альфа-канала текстуры

Нам достаточно загрузить текстуру из файла, а процедуре прорисовки сцены включить режим прозрачности. Свойства материала мы менять не будем. Обратите внимание на то, что нам так же, как и в прошлом примере, придется дважды прорисовывать грани прозрачного куба, меняя режим отсечения.

Мультитекстурирование

В некоторых приложениях нам может потребоваться такая возможность подсистемы Direct3D, как *мультитекстурирование*. Мультитекстурирование — это несколько стадий наложения текстуры на объект. Всего таких стадий может быть до 8.

Давайте представим себе трехмерную игру, в которой мы бегаем по лабиринту и стреляем в монстров. После выстрелов на предметах могут оставаться следы от пуль. Со временем эти следы исчезают.

Так как же и нам добиться подобного эффекта? Когда на одну текстуру будет наложена вторая?

Рассмотрим пример из каталога Example_33 (рис. 5.40). В примере будет вращаться куб, на каждую грань которого будет наложена текстура со словом "ТЕСТ", поверх которой, в сою очередь, будет располагаться вторая текстура, имитирующая пулевые отверстия. Причем я специально подготовил текстуру с девятью отверстиями, которые начинаются с нормального пулевого отверстия и постепенно меняют свою интенсивность.

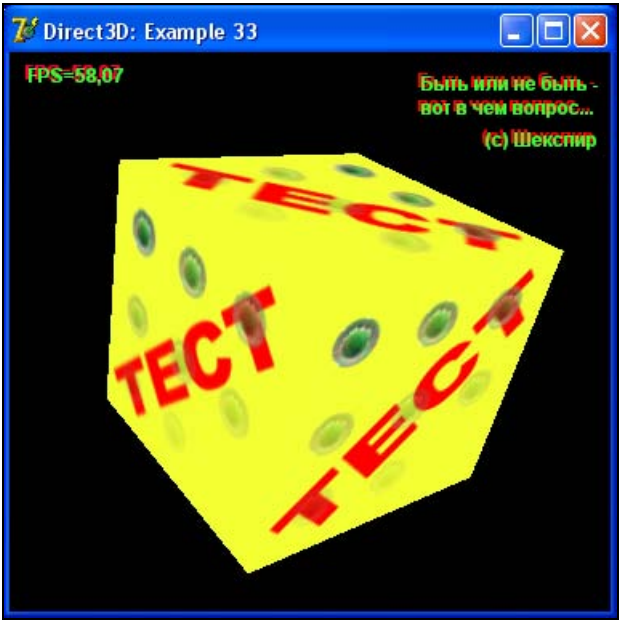


Рис. 5.40. Мультитекстурирование

Данный эффект достигается за счет все той же альфа-составляющей текстуры. Первая текстура не имеет альфа-канала, а во второй текстуре альфа-канал меняет свою интенсивность.

Процесс использования 2-х стадий наложения текстуры будет выглядеть следующим образом:

```
// Установка текстуры
FD3DDevice.SetTexture(0, FD3Dtexture);
```

```
FD3DDevice.SetTextureStageState(
    1, D3DTSS_COLOROP, D3DTOP_BLENDTEXTUREALPHA);
FD3DDevice.SetTextureStageState(1, D3DTSS_TEXCOORDINDEX, 0);
FD3DDevice.SetTexture(1, FD3Dtexture2);
```

Параметры первой текстуры мы не трогаем, а для второй текстуры включаем режим использования альфа-канала для смешивания и указываем на то, что будем использовать текстурные координаты, принадлежащие первой текстуре (вообще, мы можем использовать 8 индексов текстурных координат — для каждого этапа текстурирования свои).

Motion Blur

И последнее, что мы изучим в этой главе, — это эффект размытия при движении (Motion Blur). Подобный эффект мы можем наблюдать при быстром перемещении объекта. В приложениях Direct3D подобного эффекта можно добиться путем многочисленной прорисовки одного и того же объекта в одном кадре.

Давайте рассмотрим пример из каталога Example_34 (рис. 5.41). На сцене будет вращаться чайник, а за ним потянется постепенно затухающий след еще из 9 чайников. Прорисовка чайника и его следа будет производиться в отдельном методе `DrawBlurMeshObject` (листинг 5.36).

Листинг 5.36. Метод `DrawBlurMeshObject`

```
{*********************************************************************}
{** Вывод mesh-объекта                                              **}
{*********************************************************************}
function TMainForm.DrawBlurMeshObject(Mesh: ID3DXMesh; Angle: Single;
  Scale: Single; Material: TD3DColorValue): HResult;
var
  TempMatrix:   TD3DMatrix;
  WorldMatrix:  TD3DMatrix;
  WorldMatrixX: TD3DMatrix;
  WorldMatrixY: TD3DMatrix;
  WorldMatrixZ: TD3DMatrix;

  d3dMaterial: TD3DMaterial9;
```

```
  I: integer;
begin
  // Включаем прозрачность
  FD3DDevice.SetRenderState(D3DRS_ALPHABLENDENABLE, 1);
  FD3DDevice.SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
  FD3DDevice.SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA);

  // Устанавливаем текстуру
  FD3DDevice.SetTexture(0, FD3DXTexture);

  // Отключаем z-буфер
  FD3DDevice.SetRenderState(D3DRS_ZENABLE, D3DZB_FALSE);

  // Рисуем цепочку mesh-объектов
  for I := 0 to 9 do
  begin

    // Задаем свойства материала
    ZeroMemory(@d3dMaterial, SizeOf(d3dMaterial));
    d3dMaterial.Diffuse.r := 1 / (10 - i);
    d3dMaterial.Diffuse.g := 1 / (10 - i);
    d3dMaterial.Diffuse.b := 1 / (10 - i);
    d3dMaterial.Diffuse.a := 1;
    d3dMaterial.Ambient := d3dMaterial.Diffuse;
    // d3dMaterial.Emissive := d3dMaterial.Diffuse;

    // Устанавливаем материал
    FD3DDevice.SetMaterial(d3dMaterial);

    // Получение единичной матрицы
    D3DXMatrixIdentity(WorldMatrix);

    // Установка положения
    D3DXMatrixTranslation(WorldMatrix,
     - Sin(GradToRad(FRotAngle + Angle + i * 5)) * 2,
       Cos(GradToRad(FRotAngle + Angle + i * 5)) * 2, 0);
```

```
    // Поворот по всем трем осям x, y и z
    D3DXMatrixRotationX(WorldMatrixX, GradToRad(FRotAngle + i * 5));
    D3DXMatrixRotationY(WorldMatrixY, GradToRad(FRotAngle + i * 5));
    D3DXMatrixRotationZ(WorldMatrixZ, GradToRad(FRotAngle + i * 5));

    // Накладываем все модификации на мировую матрицу
    D3DXMatrixMultiply(WorldMatrix, WorldMatrix, WorldMatrixX);
    D3DXMatrixMultiply(WorldMatrix, WorldMatrix, WorldMatrixY);
    D3DXMatrixMultiply(WorldMatrix, WorldMatrix, WorldMatrixZ);

    // Уменьшаем mesh-объект
    D3DXMatrixScaling(TempMatrix, Scale, Scale, Scale);
    D3DXMatrixMultiply(WorldMatrix, WorldMatrix, TempMatrix);

    // Устанавливаем мировую матрицу для куба
    FD3DDevice.SetTransform(D3DTS_WORLD, WorldMatrix);

    // Если рисуем последний mesh-объект в цепочке, то...
    if I = 9 then
    begin
      // ... отключаем прозрачность
      FD3DDevice.SetRenderState(D3DRS_ALPHABLENDENABLE, 0);

      // ... включаем z-буфер
      FD3DDevice.SetRenderState(D3DRS_ZENABLE, D3DZB_TRUE);
    end;

    // Выводим mesh-объект
    Result := Mesh.DrawSubset(0);
  end;
end;
```

Рассмотрим подробнее принцип работы данной процедуры. Первым делом мы включаем прозрачность (на чайник будет наложена текстура с альфа-каналом) и отключаем *z*-буфер (дабы наши объекты рисовались не в пространстве, а на плоскости друг за другом). Далее в цикле меняем освещенность материала чайника и рисуем его с учетом всех мировых преобразова-

ний. Как только в цикле доходим до рисования последнего чайника, мы отключаем прозрачность и включаем z-буфер.

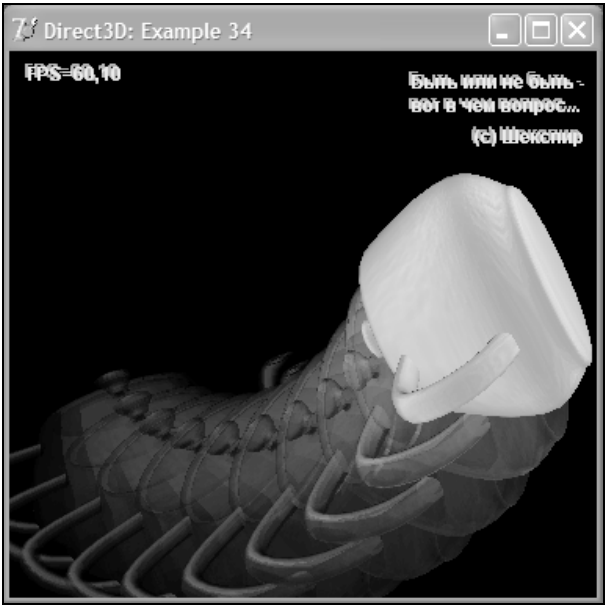


Рис. 5.41. Motion Blur

Сама же процедура прорисовки сцены будет очень простой (листинг 5.37).

Листинг 5.37. Прорисовка сцены с чайником

```
{*********************************************************************}
{** Прорисовка сцены                                                 **}
{*********************************************************************}
function TMainForm.RenderScene: HResult;
begin
  // Чистим устройство
  Result := ClearDevice;
  // Если произошла ошибка, то завершаем работу
  if FAILED(Result) then EXIT;

  // Начало сцены
  FD3DDevice.BeginScene;
```

```
  try
    // Увеличиваем угол поворота
    inc(FRotAngle, 1);

    // Задаем режим фильтрации текстуры
    if FLinerFilter then
    begin
      // Линейная фильтрация
      FD3DDevice.SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
      FD3DDevice.SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
      FD3DDevice.SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);
    end
    else begin
      // Точечная фильтрация
      FD3DDevice.SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_POINT);
      FD3DDevice.SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_POINT);
      FD3DDevice.SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_POINT);
    end;

    // Рисуем чайник со следом
    DrawBlurMeshObject(FD3DTeaport, 300, 0.7,
      D3DXColor(0.8, 0.7, 0.9, 1));

    // Выводим текст
    DrawTextMessage;
  finally

    // Завершаем сцену
    FD3DDevice.EndScene;
  end;

  // Переключение буферов
  FD3DDevice.Present(NIL, NIL, 0, NIL);

  // Увеличиваем счетчик кадров
  inc(FFrameCount);
end;
```

Шейдеры

Основные сведения

Мы начнем изучение данной темы с понятия графического конвейера. Давайте вспомним, как мы строили наши примеры. Первым делом мы инициализировали подсистему Direct3D, создавали и настраивали объект устройства, задавали формат вершин и строили трехмерные объекты из примитивов (на примере куба). В процедуре прорисовки сцены между методами `IDirect3DDevice9.BeginScene` и `IDirect3DDevice9.EndScene` мы писали код вывода объекта на экран. Так вот, для того чтобы объект был отображен на экране, он должен пройти определенные стадии обработки сквозь так называемый *графический конвейер* (Graphics Pipeline). На рис. 5.42 вы можете наглядно увидеть все стадии работы конвейера.

Вплоть до версии DirectX 8.0 этот конвейер был непрограммируемым, но, начиная с 8-й версии DirectX, появилась возможность встраивать в поток обработки данных свои микропрограммы. Эти микропрограммы и называются *шейдерами* (Shader).

На вход конвейера поступают данные о вершинах и примитивах. Затем они попадают в блок *тесселяции*, где разбиваются на более мелкие составляющие (примитивы более высокого порядка разбиваются на треугольники). Следом идет этап трансформаций и освещения (T&L, Transform and Lighting). На данном этапе происходит преобразование позиций вершин и выполняется расчет освещенности. Как видно из схемы, данный этап может быть реализован и в программируемом конвейере в виде вершинного шейдера (Vertex Shader). На следующем этапе происходит отсечение невидимых блоков изображения, что увеличивает производительность. А в конце этапа будет произведена растеризация, в ходе которой будут обработаны пикселы. На этапе обработки пикселов мы можем использовать так называемые пиксельные шейдеры (Pixel Shader) или мультитекстурирование, чтобы задать значение цвета. И последняя стадия — это набор тестов: тест прозрачности, тест глубины и т. д. После прохождения всех тестов (на пиксельном уровне) данные готовы к отображению на экране.

Следует отметить так же и то, что состояние конвейера в процессе работы можно менять. Мы можем менять параметры вывода (`IDirect3DDevice9.SetRenderState`), параметры трансформации (`IDirect3DDevice9.SetTransform`), параметры наложения текстуры (`IDirect3DDevice9.SetTextureStageState`) и сэмплера (`IDirect3DDevice9.SetSamplerState`).

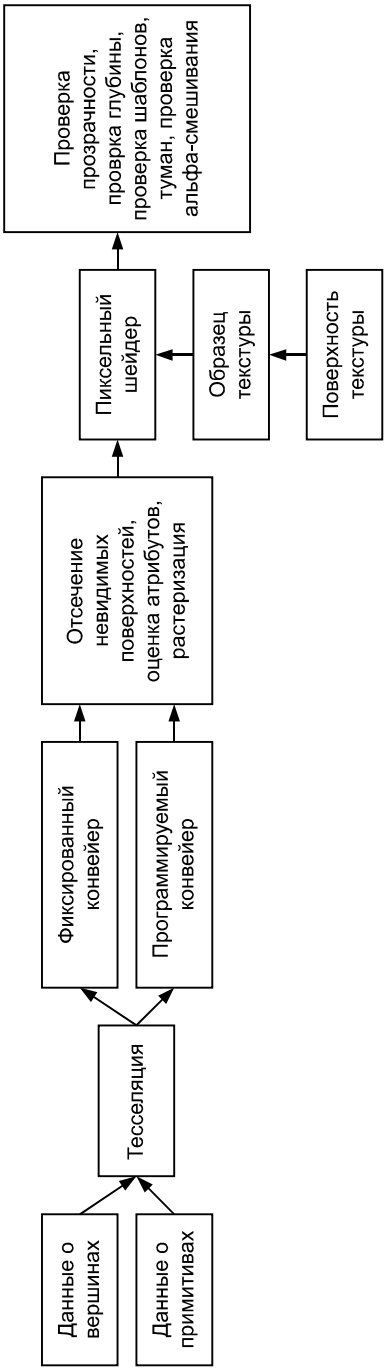


Рис. 5.42. Схема графического конвейера Direct3D

Введение в HLSL

Одним из наиболее интересных нововведений в DirectX 9.0, на мой взгляд, является язык *HLSL* (High-Level Shader Language, высокоуровневый язык шейдеров). Его синтаксис во многом схож с языком C++. Это означает, что разработка шейдеров не будет представлять особых проблем для программиста, т. к. код шейдера будет читабельным. Именно на этом языке мы с вами и будем писать наши шейдеры.

Так что же такое шейдер? Что он собой представляет? Каков принцип его работы? Какие типы шейдеров бывают? Я постараюсь дать ответы на все эти вопросы.

В нашем случае шейдер представляет собой микропрограмму на языке HLSL. Это означает, что шейдер получает некоторый набор данных на входе, обрабатывает их и передает обработанные данные на выход. Несмотря на всю абстрактность утверждения, именно по такому принципу работают как вершинные, так и пиксельные шейдеры.

Вершинный шейдер предназначен, как следует из названия, для работы с набором вершин. В нем будут производиться трансформация вершин и расчет освещения.

Пиксельный шейдер предназначен для финального расчета цвета каждой точки сцены. Он заменяет процесс мультитекстурирования в пиксельном конвейере Direct3D.

Ни вершинный шейдер, ни пиксельный ничего не знают о поступающих в них данных. На входе вершинного шейдера — вершина, на входе пиксельного — пиксел. Микропрограмма должна предоставить набор данных на выход, только исходя из данных на входе.

Итак, приступим к изучению языка HLSL. Первое, с чего мы начнем изучение, — это типы данных. Язык HLSL поддерживает целый ряд типов данных — от простых булевых типов, целочисленных типов и типов с плавающей точкой до сложных типов, таких как вектор, матрица и структура.

К поддерживаемым скалярным типам можно отнести следующие типы:

- `bool` — `true` или `false`;
- `int` — 32-битное знаковое целое;
- `half` — 16-битное число с плавающей точкой;
- `float` — 32-битное число с плавающей точкой;
- `double` — 64-битное число с плавающей точкой.

Следует заметить, что не все графические процессоры поддерживают целочисленные значения, а это означает, что они будут эмулироваться через типы с плавающей запятой.

Объявление переменных будет выглядеть следующим образом:

```
float Angle;
float Bias = 0.23;
int Index;
int IndexArray[3] = {2, 1, 3};
```

Переменная может быть объявлена с модификатором `static` или `extern`:

```
static float Bias;
extern float Angle;
```

В первом случае мы указываем на то, что переменная является статической. Это означает, что мы не сможем изменить ее значение из своего приложения ни при помощи метода `IDirect3DDevice9.SetVertexShaderConstant` или `IDirect3DDevice9.SetPixelShaderConstant`, ни с помощью методов интерфейса `ID3DXConstantTable`. Модификатор `extern` прямо противоположен, по сути, первому модификатору.

Векторные типы данных могут задаваться либо с указанием ключевого слова `vector`, либо используя тип элементов вектора: `vector <type, size>` или `typeN`, например:

```
vector <float, 3> Direction;
float3 Direction;
float  Direction[3];
```

Матричные типы объявляются подобно векторным:

```
matrix <float, 2, 2> Transform;
float2x2 Transform;
```

Структуры могут состоять из величин различных типов:

```
struct VS_OUTPUT
{
  float4 Pos: POSITION;
  float4 Diffuse: COLOR;
};
```

Для различных операций имеются свои операторы:

- операторы назначения: `=`, `+=`, `-=`, `*=`, `/=`;
- унарные операторы: `!`, `-`, `+`;
- арифметические операторы: `+`, `-`, `*`, `/`, `%`;
- логические операторы: `&&`, `||`, `?`;

- операторы сравнения: `<`, `>`, `==`, `!=`, `<=`, `>=`;
- инкремент и декремент: `++`, `--`;
- приведение типов: `(`*тип*`)`;
- запятая: `,`;
- член структуры: `.`;
- член массива: `[индекс]`.

Начало и окончание блока операторов обозначается фигурными скобками (`{}`), которые в языке Pascal обозначают блок комментариев.

Оператор ветвления имеет следующий вид:

```
if (выражение) then оператор [else оператор]
```

Циклы можно задавать любым из трех способов:

```
do оператор while (выражение);

while (выражение) оператор;
for (выражение1;выражение2;выражение3) оператор;
```

Функции языка HLSL подобны функциям языка Pascal, за исключением того, что не поддерживается рекурсия:

- `abs(x)` — вычисление абсолютной величины каждого компонента;
- `acos(x)` — вычисление арккосинуса каждого компонента;
- `all(x)` — проверка всех компонентов на ненулевое значение;
- `any(x)` — проверка вхождения нулевого значения;
- `asin(x)` — вычисление арксинуса;
- `atan(x)` — вычисление арктангенса;
- `atan2(y, x)` — вычисление арктангенса `x/y`;
- `ceil(x)` — наименьшее целое, которое больше или равно `x`;
- `clamp(x, min, max)` — фиксация `x` в диапазоне `[min, max]`;
- `clip(x)` — отказ от текущего пиксела, если любой компонент `x` меньше нуля;
- `cos(x)` — вычисление косинуса;
- `cosh(x)` — вычисление гиперболического косинуса;
- `cross(a, b)` — вычисление перекрестного произведения двух трехмерных векторов;
- `D3DCOLORtoUBYTE4(x)` — масштабирование компонентов четырехмерного вектора `x` для компенсации поддержки `UBYTE4` в некоторых аппаратных средствах;

- `ddx(x)` — возвращает частную производную переменной `x` относительно экранной координаты `x`;
- `ddy(x)` — возвращает частную производную переменной `y` относительно экранной координаты `y`;
- `degrees(x)` — перевод из радиан в градусы;
- `determinant(m)` — вычисление определителя матрицы;
- `distance(a, b)` — вычисление расстояния между точками `a` и `b`;
- `dot(x, y)` — вычисление значения "dot product" между двумя векторами;
- `exp(x)` — вычисление экспоненты;
- `exp2(x)` — вычисление экспоненты по основанию 2;
- `faceforward(n, i, ng)` — проверка видимости полигона;
- `floor(x)` — наибольшее целое, которое меньше или равно `x`;
- `fmod(x, y)` — возвращает такое значение `f`, что `x = i * y + f`, где `i` — целое, `f` имеет тот же знак, что и `x`, а абсолютное значение `f` меньше абсолютного значения `y`;
- `frac(x)` — возвращает дробную часть `x`;
- `frexp(x, out exp)` — возвращает мантиссу `x`;
- `fwidth(x)` — возвращает абсолютную величину частных производных или `abs(ddx(x)) + abs(ddy(x))`;
- `isfinite(x)` — проверка величины `x` на конечность;
- `isinf(x)` — проверка величины `x` на бесконечность;
- `isnan(x)` — проверка `x` на `NAN` или `QNAN`;
- `ldexp(x, exp)` — возвращает `x * 2exp`;
- `length(v)` — возвращает длину вектора;
- `lerp(x, y, s)` — возвращает `x + s(y - x)`, что является линейной интерполяцией между `x` и `y`;
- `lit(n dot l, n dot h, m)` — возвращает вектор освещения (объемное освещение, диффузное освещение, отражающее освещение, 1);
- `log(x)` — логарифм `x`;
- `log10(x)` — десятичный логарифм `x`;
- `log2(x)` — двоичный логарифм `x`;
- `max(a, b)` — максимум;
- `min(a, b)` — минимум;

- `modf(x, out ip)` — разделяет `x` на дробную и целую части, имеющие тот же знак, что и `x`. Знаковая дробная часть `x` будет возвращена, а целая часть будет помещена в параметр `ip`;
- `mul(x, y)` — матричное умножение `x` на `y`;
- `noise(x)` — генерация шума;
- `normalize(v)` — нормализация вектора;
- `pow(x, y)` — возвращает x^y;
- `radians(x)` — перевод из градусов в радианы;
- `reflect(i, n)` — возвращает отраженный вектор;
- `refract(i, n, ri)` — возвращает вектор преломления;
- `round(x)` — округление до ближайшего целого;
- `rsqrt(x)` — возвращает `1/sqrt(x)`;
- `saturate(x)` — фиксирует `x` в диапазоне `[0, 1]`;
- `sign(x)` — возвращает знак `x` (`-1` при отрицательном значении, `0` при нулевом и `1` при положительном значении);
- `sin(x)` — возвращает синус `x`;
- `sincos(x, out s, out c)` — возвращает синус и косинус `x`;
- `sinh(x)` — возвращает гиперболический синус `x`;
- `smoothstep(min, max, x)` — возвращает `0`, если `x < min`, и `1`, если `x > max`. Если `x` находится в диапазоне `[min, max]`, то будет возвращена плавная интерполяция между `0` и `1`;
- `sqrt(x)` — возвращает квадратный корень `x`;
- `step(a, x)` — возвращает `(x >= y) ? 1 : 0`;
- `tan(x)` — тангенс `x`;
- `tanh(x)` — гиперболический тангенс `x`;
- `transpose(m)` — транспонирование матрицы.

Для работы с текстурой мы должны будем определить в шейдере переменную типа `sampler`. Данная переменная будет определять текстуру. Давайте рассмотрим функции работы с текстурами, определенными в HLSL:

- `tex1D(s, t)` — чтение из одномерной текстуры;
- `tex1D(s, t, ddx, ddy)` — чтение из одномерной текстуры, с производными;
- `tex1Dproj(s, t)` — чтение из одномерной текстурной проекции;
- `tex1Dbias(s, t)` — чтение из одномерной текстуры со смещением;

- `tex2D(s, t)` — чтение из двумерной текстуры;
- `tex2D(s, t, ddx, ddy)` — чтение из двумерной текстуры с производными;
- `tex2Dproj(s, t)` — чтение из двумерной текстурной проекции;
- `tex2Dbias(s, t)` — чтение из двумерной текстуры со смещением;
- `tex3D(s, t)` — чтение из трехмерной текстуры;
- `tex3D(s, t, ddx, ddy)` — чтение из трехмерной текстуры с производными;
- `tex3Dproj(s, t)` — чтение из трехмерной текстурной проекции;
- `tex3Dbias(s, t)` — чтение из трехмерной текстуры со смещением;
- `texCUBE(s, t)` — чтение из кубической текстуры;
- `texCUBE(s, t, ddx, ddy)` — чтение из кубической текстуры с производными;
- `texCUBEproj(s, t)` — чтение из кубической текстурной проекции;
- `texCUBEbias(s, t)` — чтение из кубической текстуры со смещением.

Входные и выходные параметры шейдеров имеют определенную семантику. Семантика входных параметров вершинного шейдера такова:

- `POSITION[n]` — позиция;
- `BLENDWEIGHT[n]` — весовые коэффициенты смешивания;
- `BLENDINDICES[n]` — индексы смешивания;
- `NORMAL[n]` — вектор нормали;
- `PSIZE[n]` — размер точки;
- `COLOR[n]` — диффузный и отражающий цвета;
- `TEXCOORD[n]` — текстурные координаты;
- `TANGENT[n]` — касательная;
- `BINORMAL[n]` — бинормаль;
- `TESSFACTOR[n]` — фактор тесселяции.

Здесь `n` — число поддерживаемых ресурсов. Например: `COLOR0`, `TEXCOORD1` и т. п.

Выходные параметры вершинного шейдера имеют следующую семантику:

- `POSITION` — позиция;
- `PSIZE` — размер точки;
- `FOG` — коэффициент тумана вершины;
- `COLOR[n]` — цвет;
- `TEXCOORD[n]` — текстурные координаты.

Семантика входных параметров пиксельного шейдера имеет гораздо меньше параметров, чем семантика входных параметров вершинного шейдера:

- `COLOR[n]` — диффузный и отражающий цвета;
- `TEXCOORD[n]` — текстурные координаты.

Здесь `n` находится в диапазоне от `0` до числа поддерживаемых регистров.

Семантика выходных параметров пиксельного шейдера следующая:

- `COLOR[n]` — цвет;
- `TEXCOORD[n]` — текстурные координаты;
- `DEPTH[n]` — глубина.

Вершинные шейдеры

Давайте рассмотрим пример из каталога Example_35 (рис. 5.43). В этом примере мы будем использовать вершинный шейдер для расчета положения вершин объекта в пространстве.

Рис. 5.43. Обработка вершин при помощи шейдера

Микропрограмма шейдера представлена в листинге 5.38.

Листинг 5.38. Текст модуля vs.fx примера Direct3D\Example_35

```
// Произведение матриц
float4x4 matWorldViewProject;

// Входные данные
struct VS_INPUT
{
  float4 Pos: POSITION;
};

// Выходные данные
struct VS_OUTPUT
{
  float4 Pos: POSITION;
};

// Основная процедура вершинного шейдера
VS_OUTPUT main(VS_INPUT In)
{
  // Результат
  VS_OUTPUT Out = (VS_OUTPUT)0;

  // Вычисляем позицию вершины
  Out.Pos = mul(In.Pos, matWorldViewProject);

  // Возвращаем результат
  return Out;
}
```

Первым делом мы определяем в шейдере переменную `matWorldViewProject`, являющуюся произведением мировой, видовой и проекционной матриц. Структуры, описывающие данные на входе и на выходе, содержат всего одну переменную — это позиция вершины в пространстве. Именно поэтому мы будем видеть на экране белый силуэт чайника — описание вершины не содержит цветовой составляющей.

Основная процедура шейдера будет получать на входе описание позиции вершины. Новая позиция вершины будет рассчитываться как произведение начальной позиции и матрицы, являющейся произведением трех матриц.

Вершинный шейдер использует интерфейс `IDirect3DVertexShader9`. Доступ к переменным шейдера мы будем осуществлять при помощи интерфейса `ID3DXConstantTable`. Создание шейдера и получение дескриптора переменной (по имени) будет производиться в процедуре `CreateShader` (листинг 5.39).

Листинг 5.39. Создание вершинного шейдера

```
{*************************************************************************}
{** Создание шейдера                                                    **}
{*************************************************************************}
function TMainForm.CreateShader: HResult;
var
  pCode: ID3DXBuffer;
begin
  // Компиляция вершинного шейдера
  Result := D3DXCompileShaderFromFile('vs.fx', NIL, NIL, 'main',
    'vs_2_0', 0, @pCode, NIL, @FD3DVertexConstant);
  if FAILED(Result) then EXIT;

  // Создание объекта вершинного шейдера
  Result := FD3DDevice.CreateVertexShader(
    pCode.GetBufferPointer,
    FD3DVertexShader);
  pCode := NIL;
  if FAILED(Result) then EXIT;

  // Получение дескриптора параметра шейдера
  matWorldViewProjectHandle := FD3DVertexConstant.GetConstantByName(NIL,
    'matWorldViewProject');
end;
```

Как видно из данной процедуры, первое, что нам необходимо проделать, — это скомпилировать файл с шейдером. Для этого предназначена процедура `D3DXCompileShaderFromFile`. Отдельно стоит заметить, что данная процеду-

ра будет использоваться для загрузки как вершинных, так и пиксельных шейдеров:

```
function D3DXCompileShaderFromFile(
  pSrcFile: PChar;
  pDefines: PD3DXMacro;
  pInclude: ID3DXInclude;
  pFunctionName: PAnsiChar;
  pProfile: PAnsiChar;
  Flags: DWORD;
  ppShader: PID3DXBuffer;
  ppErrorMsgs: PID3DXBuffer;
  ppConstantTable: PID3DXConstantTable):
HResult; stdcall;
external d3dx9shaderDLL name 'D3DXCompileShaderFromFileA';
```

Здесь:

- `pSrcFile` — путь к файлу шейдера;
- `pDefines` — необязательно может использоваться нулевое значение;
- `pInclude` — необязательный указатель на интерфейс `ID3DXInclude`, используемый для работы с директивами `#include` в процессе компиляции шейдера. В нашем случае должно задаваться нулевое значение;
- `pFunctionName` — название процедуры шейдера;
- `pProfile` — профиль (версия) шейдера;
- `Flags` — флаги управления процессом компиляции шейдера;
- `ppShader` — возвращает буфер, содержащий созданный шейдер. Этот буфер содержит скомпилированный код шейдера, а также разнообразную отладочную информацию;
- `ppErrorMsgs` — возвращает буфер, в который будут занесены сообщения об ошибках и предупреждения, возникшие в процессе компиляции;
- `ppConstantTable` — возвращает интерфейс `ID3DXConstantTable`, который будет использоваться для доступа к переменным шейдера.

Следующий после компиляции этап — создание вершинного шейдера при помощи метода `IDirect3DDevice9.CreateVertexShader`:

```
function CreateVertexShader(
  pFunction: PDWord;
  out ppShader: IDirect3DVertexShader9):
HResult; stdcall;
```

Здесь:

- `pFunction` — указатель на массив данных шейдера;
- `ppShader` — указатель на переменную, в которую будет занесен интерфейс вершинного шейдера.

Претерпит изменения и процедура настройки вида сцены `SetupCamera` — теперь в ней мы будем производить установку вершинного шейдера методом `IDirect3DDevice9.SetVertexShader`, который содержит всего один параметр — указатель на интерфейс вершинного шейдера, и устанавливать значение параметра шейдера. Никаких установок трансформаций (`IDirect3DDevice9.SetTransform`) в явном виде в программе присутствовать не будет. Все трансформации вершин будут рассчитываться в вершинном шейдере (листинг 5.40).

Листинг 5.40. Настройка вида сцены

```
{*********************************************************************}
{** Настройка вида сцены                                            **}
{*********************************************************************}
function TMainForm.SetupCamera: HResult;
var
  Eye: TD3DVector;
  At: TD3DVector;
  Up: TD3DVector;

  ViewMatrix: TD3DMatrix;
  ProjectionMatrix: TD3DMatrix;

  WorldMatrix:  TD3DMatrix;
  WorldMatrixX: TD3DMatrix;
  WorldMatrixY: TD3DMatrix;
  WorldMatrixZ: TD3DMatrix;

  Matrix,
  matViewWorld: TD3DMatrix;
begin
  // Результат по умолчанию
  Result := E_FAIL;
```

```
if (FD3DDevice = NIL) or (FD3DVertexShader = NIL) or
   (FD3DVertexConstant = NIL) then EXIT;

// Вектор, определяющий положение глаз наблюдателя
Eye.x := 0; Eye.y := 0; Eye.z := -5;
// Направление камеры
At.x := 0;  At.y := 0;  At.z := 0;
// Вектор, определяющий текущие мировые координаты. Обычно
// имеет значение (0, 1, 0).
Up.x := 0;  Up.y := 1;  Up.z := 0;

// Строим левостороннюю матрицу вида
D3DXMatrixLookAtLH(ViewMatrix, Eye, At, Up);

// Строим левостороннюю матрицу проекции
D3DXMatrixPerspectiveFovLH(ProjectionMatrix, D3DX_PI / 4, 1, 1, 100);

// Увеличиваем угол поворота
inc(FRotAngle, 1);

// Вращаем по всем осям x, y и z
D3DXMatrixRotationX(WorldMatrixX, GradToRad(FRotAngle));
D3DXMatrixRotationY(WorldMatrixY, GradToRad(FRotAngle));
D3DXMatrixRotationZ(WorldMatrixZ, GradToRad(FRotAngle));

// Накладываем все модификации на мировую матрицу
D3DXMatrixMultiply(WorldMatrix, WorldMatrixX, WorldMatrixY);
D3DXMatrixMultiply(WorldMatrix, WorldMatrix,  WorldMatrixZ);

// Строим произведение всех матриц
D3DXMatrixMultiply(matViewWorld, WorldMatrix, ViewMatrix);
D3DXMatrixMultiply(Matrix, matViewWorld, ProjectionMatrix);

// Установка шейдера
FD3DDevice.SetVertexShader(FD3DVertexShader);
```

```
  // Задаем параметр шейдера - произведение матриц
  Result := FD3DVertexConstant.SetMatrix(FD3DDevice,
    matWorldViewProjectHandle, Matrix);
end;
```

А процедура обработки сцены будет такой, как в листинге 5.41.

Листинг 5.41. Прорисовка сцены

```
{*********************************************************************}
{** Прорисовка сцены                                                **}
{*********************************************************************}
function TMainForm.RenderScene: HResult;
begin
  // Чистим устройство
  Result := ClearDevice;
  // Если произошла ошибка, то завершаем работу
  if FAILED(Result) then EXIT;

  // Начало сцены
  FD3DDevice.BeginScene;
  try
    // Настройка вида
    SetupCamera;

    // Рисуем чайник
    FD3DTeaport.DrawSubset(0);
  finally

    // Завершаем сцену
    FD3DDevice.EndScene;
  end;

  // Переключение буферов
  FD3DDevice.Present(NIL, NIL, 0, NIL);
end;
```

Теперь давайте рассмотрим пример из каталога Example_36 (рис. 5.44). В этом примере будет производиться расчет диффузного освещения вершины исходя из направления источника света. Вершинный шейдер будет иметь вид, представленный в листинге 5.42.

Листинг 5.42. Текст модуля vs.fx примера Direct3D\Example_36

```
// Произведение матриц
float4x4 matWorldViewProject;
// Матрица вида
float4x4 matView;
// Источник света
float4   vecLightDirection;

// Параметры диффузного освещения
static float4 Diffuse = {0.0f, 1.0f, 0.0f, 1.0f};

// Входные данные
struct VS_INPUT
{
  float4 Pos: POSITION;
  float4 Normal: NORMAL;
};

// Выходные данные
struct VS_OUTPUT
{
  float4 Pos: POSITION;
  float4 Diffuse: COLOR;
};

// Основная процедура вершинного шейдера
VS_OUTPUT main(VS_INPUT In)
{
  // Результат
  VS_OUTPUT Out = (VS_OUTPUT)0;
```

```
  // Вычисляем позицию вершины
  Out.Pos = mul(In.Pos, matWorldViewProject);

  vecLightDirection.w = 0.0;
  In.Normal.w = 0.0;

  // Вычисляем направление относительно вида
  vecLightDirection = mul(vecLightDirection, matView);
  // Нормаль
  In.Normal = mul(In.Normal, matWorldViewProject);

  // Вычисляем коэффициент
  float T = dot(vecLightDirection, In.Normal);
  if (T < 0.0) T = 0.0;

  // Расчет освещения вершины
  Out.Diffuse = T * Diffuse;

  // Возвращаем результат
  return Out;
}
```

Теперь наш шейдер содержит три параметра — две матрицы и один вектор. Для работы с ними мы должны сначала получить их дескрипторы:

```
// Получение дескрипторов параметров шейдера
matWorldViewProjectHandle := FD3DVertexConstant.GetConstantByName(NIL,
  'matWorldViewProject');
matViewHandle := FD3DVertexConstant.GetConstantByName(NIL, 'matView');
vecLightDirectionHandle := FD3DVertexConstant.GetConstantByName(NIL,
  'vecLightDirection');
```

А в процедуре настройки вида сцены передавать значения в шейдер следующим образом:

```
// Задаем параметр шейдера - произведение матриц
FD3DVertexConstant.SetMatrix(FD3DDevice, matWorldViewProjectHandle,
  Matrix);
```

```
// Задаем параметр шейдера - матрица вида
FD3DVertexConstant.SetMatrix(FD3DDevice, matViewHandle, ViewMatrix);

// Задаем параметр шейдера - направление источника света
LightDirection := D3DXVector4(0, 0, -1, 0);
Result := FD3DVertexConstant.SetVector(FD3DDevice,
  vecLightDirectionHandle, LightDirection);
```

Рис. 5.44. Обработка вершин при помощи шейдера

Пиксельные шейдеры

Ранее мы уже давали определение пиксельного шейдера. Как и следует из названия, он предназначен для обработки не вершин, а пикселов. Результатом его работы будет финальный цвет точки. Пиксельный шейдер использует интерфейс `IDirect3DPixelShader9`. Пример из каталога Example_37 (рис. 5.45) учитывает возможности вершинного и пиксельного шейдеров одновременно. Вершинный шейдер остался без изменения от предыдущего примера, а пиксельный шейдер имеет вид, представленный в листинге 5.43.

Листинг 5.43. Текст модуля ps.fx примера Direct3D\Example_37

```
// Интенсивность
static const float DiffuseIntensity = 1.2;

// Входные данные
struct VS_INPUT
{
  float4 Pos: POSITION;
  float4 Diffuse: COLOR;
};

// Основная процедура пиксельного шейдера
float4 main(VS_INPUT In): COLOR
{
  // Результат
  float4 Out = (float4)0;

  // Расчет цвета вершины
  Out.r = 1 - DiffuseIntensity * In.Diffuse.b * 0.2;
  Out.g = 1 - DiffuseIntensity * In.Diffuse.g * 1.7;
  Out.b = 1 - DiffuseIntensity * In.Diffuse.r * 1.9;

  // Возвращаем результат
  return Out;
}
```

В пиксельном шейдере цвет точки будет инвертирован и рассчитан особым образом. На входе шейдера — структура, определяющая интерполированную позицию и цветовую составляющую точки, а на выходе, собственно, рассчитанный нами определенным образом цвет. Это означает, что при помощи пиксельных шейдеров можно создавать различные реалистичные атмосферные эффекты, огонь, дым и т. д.

Код создания пиксельного шейдера в приложении будет следующим:

```
// Компиляция пиксельного шейдера
Result := D3DXCompileShaderFromFileW('ps.fx', NIL, NIL, 'main',
  'ps_1_1', 0, @pCode, NIL, NIL);
if FAILED(Result) then EXIT;
```

```
// Создание объекта шейдера
Result := FD3DDevice.CreatePixelShader(
  pCode.GetBufferPointer,
  FD3DPixelShader);
pCode := NIL;
```

Установку пиксельного шейдера производим там же, где и установку вершинного:

```
// Установка вершинного шейдера
FD3DDevice.SetVertexShader(FD3DVertexShader);

// Установка пиксельного шейдера
FD3DDevice.SetPixelShader(FD3DPixelShader);
```

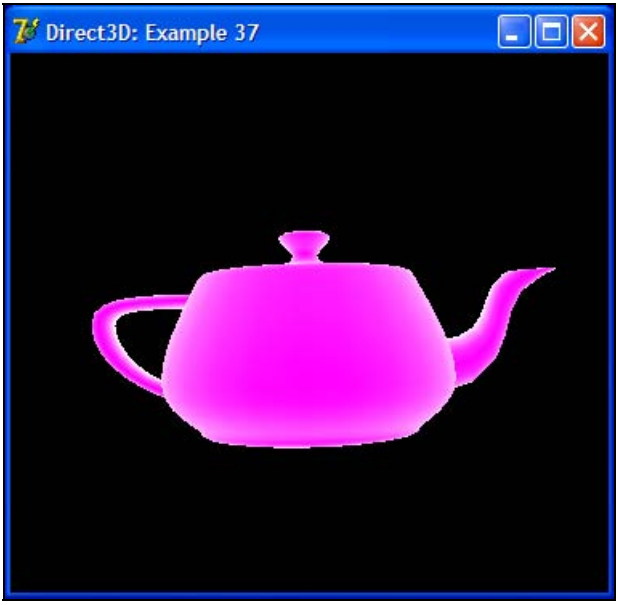


Рис. 5.45. Обработка данных вершинным и пиксельным шейдерами

Работа с текстурой

Последний пример из каталога Example_38 (рис. 5.46) наглядно показывает нам способ работы с текстурой в вершинном шейдере. В этом примере вы

увидите некое подобие зеркального чайника — от вращающегося чайника будет отражаться картинка.

Вершинный шейдер представлен в листинге 5.44.

Листинг 5.44. Текст модуля vs.fx примера Direct3D\Example_38

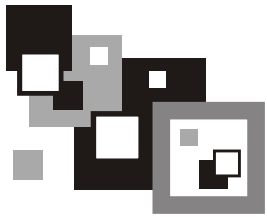
```
// Произведение матриц
float4x4 matWorldViewProject;

// Входные данные
struct VS_INPUT
{
  float4 Pos: POSITION;
  float2 Tex: TEXCOORD0;
};

// Выходные данные
struct VS_OUTPUT
{
  float4 Pos: POSITION;
  float2 Tex: TEXCOORD0;
};

// Основная процедура вершинного шейдера
VS_OUTPUT main(VS_INPUT In)
{
  // Результат
  VS_OUTPUT Out = (VS_OUTPUT)0;

  // Вычисляем позицию вершины
  Out.Pos = mul(In.Pos, matWorldViewProject);

  // Рассчитываем текстурные координаты
  Out.Tex = - In.Tex * Out.Pos / 3 + 0.5;

  // Возвращаем результат
  return Out;
}
```

Обратите внимание — теперь входные и выходные параметры шейдера содержат текстурные координаты.

Рис. 5.46. Работа с текстурой

На этом мы заканчиваем изучение подсистемы Direct3D и переходим к рассмотрению подсистемы DirectDraw.

Глава 6



DirectDraw

В этой главе мы вкратце изучим работу с графической подсистемой DirectDraw, являющейся частью DirectX Graphics, начиная с выхода в свет 8-ой версии DirectX. Несмотря на то, что данная подсистема несколько устарела, и ее интерфейсы не будут развиваться, поддержка данной подсистемы в DirectX останется и в дальнейшем, и часть графических задач все же целесообразнее решать именно при помощи этой подсистемы.

Изучение данной подсистемы, равно как и изучение подсистемы Direct3D, будет построено по принципу рассмотрения примеров, располагающихся в каталоге Examples\DirectX Graphics\DirectDraw\Example_*xx*, где *xx* — порядковый номер примера. Начнем мы с обзора возможностей данной подсистемы. Изучим типы поверхностей, разберем понятие блиттинга, научимся работать с DirectDraw в полноэкранных и оконных приложениях. Научимся работать с цветовыми ключами, рассмотрим понятие палитры и завершим изучение DirectDraw использованием оверлеев DirectX.

Обзор библиотеки

Подсистема DirectDraw представляет собой специальный программный интерфейс, предназначенный для работы с различными видеоадаптерами напрямую. Это и есть основное отличие от интерфейса GDI (Graphics Device Interface, специальный интерфейс Windows, который используется для рисования). Последний интерфейс должен был быть переносимым, вследствие чего получился достаточно медленным и непроизводительным, несмотря на все многолетние старания. И именно поэтому большинство графических приложений пишется с использованием прямого доступа к видеопамяти.

Инициализация

Основным интерфейсом, отвечающим за работу подсистемы DirectDraw, является интерфейс `IDirectDraw7`. Для начала работы с подсистемой DirectDraw ее необходимо проинициализировать, вызвав метод `DirectDrawCreateEx`:

```
function DirectDrawCreateEx(
  lpGUID: PGUID;
  out lplpDD: IDirectDraw7;
  const iid: TGUID;
  pUnkOuter: IUnknown):
HResult; stdcall; external DirectDrawDll;
```

Здесь:

- `lpGUID` — уникальный указатель, определяющий драйвер устройства. Указав нулевое значение, мы будем использовать активный драйвер дисплея;
- `lplpDD` — переменная, в которую будет записан интерфейс `IDirectDraw7` созданного объекта;
- `iid` — уникальный идентификатор создаваемого интерфейса;
- `pUnkOuter` — используется при агрегации. Должно задаваться нулевое значение.

Первый пример

Давайте рассмотрим пример из каталога Example_01. Это самый первый простейший пример, в котором только выполняется инициализация подсистемы DirectDraw на старте и производится освобождение ресурсов при завершении работы приложения. Пример построен по тому же принципу, что и примеры из предыдущей главы. Текст первого примера приведен в листинге 6.1.

Листинг 6.1. Текст модуля FormMain.pas примера DirectDraw\Example_01

```
UNIT FormMain;
{*************************************************************************}
{** DirectDraw: Example_01                                              **}
{** Автор: Есенин Сергей Анатольевич                                    **}
{*************************************************************************}
```

```
{**} INTERFACE {*************************************************************}

{**} USES {******************************************************************}
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, DirectDraw, ExtCtrls, AppEvnts;

{**} TYPE {******************************************************************}
  TMainForm = class(TForm)
    applicationEventsMain: TApplicationEvents;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure applicationEventsMainIdle(Sender: TObject;
      var Done: Boolean);
    procedure applicationEventsMainMinimize(Sender: TObject);
    procedure applicationEventsMainRestore(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  PRIVATE
    FDD7: IDirectDraw7;

    FIsActive: boolean;

    function InitDirectDraw: HResult;
    procedure FreeDirectDraw;

    function RenderScene: HResult;
  PUBLIC
  END;

{**} VAR {*******************************************************************}
  MainForm: TMainForm;

{**} IMPLEMENTATION {********************************************************}

{$R *.dfm}

{**} { TMainForm } {*********************************************************}
```

```
{*************************************************************************}
{** Инициализация подсистемы DirectDraw                                 **}
{*************************************************************************}
function TMainForm.InitDirectDraw: HResult;
begin
  // Создание объекта DirectDraw
  Result := DirectDrawCreateEx (NIL, FDD7, IDirectDraw7, NIL);
  if Result <> DD_OK then Exit;

  // Установка уровня взаимодействия
  Result := FDD7.SetCooperativeLevel(Handle, DDSCL_NORMAL);
  if Result <> DD_OK then Exit;
end;

{*************************************************************************}
{** Освобождаем ресурсы                                                 **}
{*************************************************************************}
procedure TMainForm.FreeDirectDraw;
begin
  FDD7 := NIL;
end;

{*************************************************************************}
{** Инициализаця подсистемы при создании формы                          **}
{*************************************************************************}
procedure TMainForm.FormCreate(Sender: TObject);
begin
  if FAILED(InitDirectDraw) then
  begin
    ShowMessage('Error initializing DirectDraw...');
    Halt;
  end;
end;

{*************************************************************************}
{** Освобождаем ресурсы при завершении работы программы                 **}
{*************************************************************************}
```

```
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  FreeDirectDraw;
end;

{*************************************************************************}
{** Прорисовка сцены                                                    **}
{*************************************************************************}
function TMainForm.RenderScene: HResult;
begin
  Result := S_OK;
end;

{*************************************************************************}
{** Различные действия                                                  **}
{*************************************************************************}
procedure TMainForm.applicationEventsMainIdle(Sender: TObject;
  var Done: Boolean);
begin
  if FIsActive then
  begin
    RenderScene;
  end;

  Done := FALSE;
end;

{*************************************************************************}
{** Сворачиваем приложение                                              **}
{*************************************************************************}
procedure TMainForm.applicationEventsMainMinimize(Sender: TObject);
begin
  FIsActive := FALSE;
end;
```

```
{*************************************************************************}
{** Восстанавливаем приложение                                          **}
{*************************************************************************}
procedure TMainForm.applicationEventsMainRestore(Sender: TObject);
begin
  FIsActive := TRUE;
end;

{*************************************************************************}
{** Приложение активно                                                  **}
{*************************************************************************}
procedure TMainForm.FormActivate(Sender: TObject);
begin
  FIsActive := TRUE;
end;

END.
```

Уровни взаимодействия

Уровни взаимодействия определяют способ взаимодействия подсистемы DirectDraw с дисплеем. Для установки уровня взаимодействия предназначен метод `IDirectDraw7.SetCooperativeLevel`:

```
function SetCooperativeLevel(
  hWnd: HWND;
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `hWnd` — дескриптор основного окна приложения. Может принимать нулевое значение при использовании флага `DDSCL_NORMAL` в параметре `dwFlags`;
- `dwFlags` — флаги, определяющие режим работы. Параметр может принимать одно из следующих значений либо их комбинацию:
  - `DDSCL_ALLOWMODEX` — использование так называемых режимов работы дисплея Mode X. Данный флаг может применяться только в комбинации с флагами `DDSCL_EXCLUSIVE` и `DDSCL_FULLSCREEN`;

- DDSCL_ALLOWREBOOT — включает реакцию на нажатие клавиш <Ctrl>+ +<Alt>+<Del> в эксклюзивном (полноэкранном) режиме работы;
- DDSCL_CREATEDEVICEWINDOW — флаг предназначен только для операционных систем Windows 98/2000. Указывает на то, что DirectDraw создаст и будет управлять окном по умолчанию;
- DDSCL_EXCLUSIVE — эксклюзивный уровень доступа. Флаг используется в комбинации с флагом DDSCL_FULLSCREEN;
- DDSCL_FPUPRESERVE — статус FPU постоянно обновляется подсистемой Direct3D;
- DDSCL_FPUSETUP — поддержка оптимальной производительности FPU для Direct3D;
- DDSCL_FULLSCREEN — полноэкранный эксклюзивный режим работы приложения. Использование GDI прекращается. Флаг должен указываться совместно с флагом DDSCL_EXCLUSIVE;
- DDSCL_MULTITHREADED — безопасный многопоточный режим работы DirectDraw;
- DDSCL_NORMAL — приложение является обычным Win32-приложением. Этот флаг не может использоваться совместно с флагами DDSCL_ALLOWMODEX, DDSCL_EXCLUSIVE и DDSCL_FULLSCREEN;
- DDSCL_NOWINDOWCHANGES — минимизация и восстановление окна при активации запрещены DirectDraw;
- DDSCL_SETDEVICEWINDOW — флаг поддерживается только в Windows 98/2000. Он указывает, что дескриптор hWnd является дескриптором окна устройства. Флаг не может быть использован совместно с флагом DDSCL_SETFOCUSWINDOW;
- DDSCL_SETFOCUSWINDOW — флаг поддерживается только в Windows 98/2000. Он указывает, что дескриптор hWnd является дескриптором окна активного приложения. Флаг не может быть использован совместно с флагом DDSCL_SETDEVICEWINDOW.

В примере из каталога Example_02 мы будем использовать эксклюзивный режим работы (листинг 6.2).

Листинг 6.2. Инициализация подсистемы DirectDraw

```
{*************************************************************************}
{** Инициализация подсистемы DirectDraw                                 **}
{*************************************************************************}
function TMainForm.InitDirectDraw: HResult;
```

```
begin
  // Создание объекта DirectDraw
  Result := DirectDrawCreateEx (NIL, FDD7, IDirectDraw7, NIL);
  if Result <> DD_OK then Exit;

  // Установка уровня взаимодействия
  Result := FDD7.SetCooperativeLevel(Handle,
    DDSCL_FULLSCREEN or DDSCL_EXCLUSIVE);
  if Result <> DD_OK then Exit;
end;
```

Полноэкранный режим работы

Иногда бывает полезно сделать так, чтобы на экране оставалось только ваше приложение. При этом остальные запущенные приложения будут продолжать свою работу и даже выводить данные на то, что они считают экраном. Но нас это уже не будет беспокоить, т. к. механизмы GDI будут продолжать записывать данные в видеопамять, но мы не увидим ничего лишнего. Также следует отметить, что в полноэкранном режиме возрастает производительность программы, но в таком режиме одновременно может работать только одно приложение.

Переключение в полноэкранный режим производится вызовом метода `IDirectDraw7.SetDisplayMode`:

```
function SetDisplayMode(
  dwWidth: DWORD;
  dwHeight: DWORD;
  dwBPP: DWORD;
  dwRefreshRate: DWORD;
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `dwWidth`, `dwHeight` — ширина и высота экрана в полноэкранном режиме;
- `dwBPP` — число битов на пиксел (BPP, Bits Per Pixel) или так называемая *глубина цвета*;
- `dwRefreshRate` — частота обновления экрана. Можно использовать нулевое значение для установки частоты обновления по умолчанию для данного режима;

- `dwFlags` — флаг, определяющий режим работы. Может принимать значение `DDSDM_STANDARDVGAMODE`, которое указывает на то, что будет использоваться так называемый режим Mode 13 вместо режима Mode X (320×240×8). Для использования другого разрешения, глубины цвета или режима Mode X мы должны задать нулевое значение данного параметра.

Поверхности

Поверхностью (Surface) можно считать некую линейную область видеопамяти (иногда системной памяти). Подсистема DirectDraw при создании поверхности сама определяет, в какой памяти (в видеопамяти или в системной) создавать заданную поверхность. Мы можем вывести на поверхность любое изображение, будь то обычная линия или трехмерная сцена, однако в любом случае изображение будет двумерным, т. е. буфер глубины не используется.

Работу с поверхностями нам обеспечивает интерфейс `IDirectDrawSurface7`. Нужно отметить следующие типы поверхностей: первичная (главная) поверхность, вторичная поверхность и внеэкранные поверхности. Все эти типы поверхностей мы и будем использовать в своих примерах.

Первичная поверхность создается вызовом метода `IDirectDraw7.CreateSurface`:

```
function CreateSurface(
  const lpDDSurfaceDesc: TDDSurfaceDesc2;
  out lplpDDSurface: IDirectDrawSurface7;
  pUnkOuter: IUnknown):
HResult; stdcall;
```

Здесь:

- `lpDDSurfaceDesc` — структура `TDDSurfaceDesc2`, описывающая параметры поверхности;
- `lplpDDSurface` — переменная, в которую будет записан указатель на интерфейс созданной поверхности;
- `pUnkOuter` — используется для агрегации. Должно быть указано нулевое значение.

Создание первичной поверхности будет выглядеть следующим образом:

```
// Параметры основной поверхности
ZeroMemory(@ddsd, SizeOf(ddsd));
ddsd.dwSize := SizeOf(ddsd);
ddsd.dwFlags := DDSD_CAPS or DDSD_BACKBUFFERCOUNT;
```

```
ddsd.ddsCaps.dwCaps :=
  DDSCAPS_PRIMARYSURFACE or DDSCAPS_COMPLEX or DDSCAPS_FLIP;
ddsd.dwBackBufferCount := 1;


// Создаем первичную поверхность
Result := FDD7.CreateSurface(ddsd, FDDSPrimary, NIL);
if Result <> DD_OK then Exit;
```

При указании параметров поверхности мы установили флаги `DDSD_CAPS` и `DDSD_BACKBUFFERCOUNT`. Первый флаг говорит о том, что мы хотим использовать структуру `ddsCaps`, а второй флаг указывает на использование вторичной поверхности. Параметр `dwCaps` структуры `ddsCaps` задает параметры поверхности, такие как первичная поверхность (`DDSCAPS_PRIMARYSURFACE`), сложная (составная) поверхность (`DDSCAPS_COMPLEX`) с возможностью смены (переключения) страниц (`DDSCAPS_FLIP`). Число вторичных поверхностей указывается в параметре `dwBackBufferCount`. В нашем случае будет использована всего одна вторичная поверхность.

Вторичную поверхность мы будем создавать иначе. Точнее, мы не станем ее создавать в прямом смысле слова, а просто получим указатель на нее у первичной поверхности методом `IDirectDrawSurface7.GetAttachedSurface`:

```
function GetAttachedSurface(
  const lpDDSCaps: TDDSCaps2;
  out lplpDDAttachedSurface: IDirectDrawSurface7):
HResult; stdcall;
```

Здесь:

- `lpDDSCaps` — структура `TDDSCaps2`, описывающая аппаратные характеристики поверхности;
- `lplpDDAttachedSurface` — переменная, в которую будет записан указатель на интерфейс вторичной поверхности.

Соответственно, код получения вторичной поверхности будет следующим:

```
// Вторичная поверхность
ZeroMemory(@ddscaps, SizeOf(ddscaps));
ddscaps.dwCaps := DDSCAPS_BACKBUFFER;
Result := FDDSPrimary.GetAttachedSurface(ddscaps, FDDSSecondary);
```

Внеэкранная поверхность будет создаваться по аналогии с первичной поверхностью, за исключением некоторых параметров:

```
// Параметры внеэкранной поверхности
ZeroMemory (@ddsd, SizeOf(ddsd));
```

```
ddsd.dwSize   := SizeOf(ddsd);
ddsd.dwFlags  := DDSD_CAPS or DDSD_WIDTH or DDSD_HEIGHT;
ddsd.dwWidth  := 100;
ddsd.dwHeight := 100;
ddsd.ddsCaps.dwCaps := DDSCAPS_OFFSCREENPLAIN;

// Создаем внеэкранную поверхность
if FAILED(FDD7.CreateSurface(ddsd, FImageBuffer, NIL)) then EXIT;
```

В параметрах поверхности мы указали использование поверхности определенного размера (флаги `DDSD_WIDTH` и `DDSD_HEIGHT`), а характеристики поверхности содержат всего один флаг, указывающий на использование внеэкранной поверхности (`DDSCAPS_OFFSCREENPLAIN`).

Рисуем на поверхности

Давайте вспомним, каким образом мы можем нарисовать что-то на окне. Первым делом нам необходимо получить контекст устройства, на котором мы будем рисовать вызовом функции `GetDC`. Зная контекст устройства, мы можем рисовать на нем все, что только заблагорассудится. В конце работы мы должны освободить захваченные ресурсы методом `ReleaseDC`. Практически то же самое можно проделать и с поверхностями DirectDraw, только подсистема DirectDraw позволяет нам записывать данные непосредственно в видеопамять, за счет чего во много раз возрастает производительность приложения.

Для отображения изображения в формате BMP на поверхности мы будем использовать метод `DDCopyBitmap` (листинг 6.3).

Листинг 6.3. Метод `DDCopyBitmap`

```
{*************************************************************************}
{** Копирование изображения на поверхность                              **}
{*************************************************************************}
function DDCopyBitmap(DDSurface: IDirectDrawSurface7; Bitmap: TBitmap;
  dx, dy: integer): HRESULT;
var
  hdcImage: HDC;
  dc: HDC;
  bm: HBitmap;
```

```
begin
  // Результат по умолчанию
  result := E_FAIL;
  if (Bitmap = NIL) or (DDSurface = NIL) then EXIT;

  // Создаем контксст устройства, совместимого с указанным в памяти
  hdcImage := CreateCompatibleDC(0);
  // Выбираем объект
  bm := SelectObject(hdcImage, Bitmap.Handle);

  // Получаем контекст устройства
  result := DDSurface.GetDC(dc);

  if (result = DD_OK) then begin
    // Копируем изображение
    BitBlt(dc, 0, 0, dx, dy, hdcImage, 0, 0, SRCCOPY);

    // Освобождаем контекст устройства
    DDSurface.ReleaseDC(dc);
  end;

  // Освобождаем ресурсы
  SelectObject(hdcImage, bm);
  DeleteDC(hdcImage);
end;
```

Изображение будет загружаться из картинки во внеэкранную поверхность. Таким образом, код создания внеэкранной поверхности будет иметь следующий вид:

```
// Внеэкранная поверхность
bitmap := TBitmap.Create;
try
  bitmap.LoadFromFile('Texture.bmp');

  // Параметры внеэкранной поверхности
  ZeroMemory (@ddsd, SizeOf(ddsd));
  ddsd.dwSize   := SizeOf(ddsd);
```

```
  ddsd.dwFlags  := DDSD_CAPS or DDSD_WIDTH or DDSD_HEIGHT;
  ddsd.dwWidth  := bitmap.Width;
  ddsd.dwHeight := bitmap.Height;
  ddsd.ddsCaps.dwCaps := DDSCAPS_OFFSCREENPLAIN;

  // Создаем внеэкранную поверхность
  if FAILED(FDD7.CreateSurface(ddsd, FImageBuffer, NIL)) then EXIT;

  DDCopyBitmap(FImageBuffer, bitmap, bitmap.Width, bitmap.Height);
finally
  FreeAndNIL(bitmap);
end;
```

Блиттинг

Блиттингом называется копирование блока графических данных из одного места видеопамяти (или системной памяти) в другое. Иными словами, всю подсистему DirectDraw можно представить в виде аппаратно-независимого механизма блиттинга.

Для копирования мы будем вызывать методы `IDirectDrawSurface7.Blt` или `IDirectDrawSurface7.BltFast`. Мы не станем досконально разбирать все параметры этих функций, однако стоит рассмотреть отличие между двумя этими методами. Метод `IDirectDrawSurface7.BltFast` осуществляет, по сути, простое быстрое копирование, а метод `IDirectDrawSurface7.Blt` поддерживает отсечение при работе в оконном режиме с установленным объектом `IDirectDrawClipper` (мы рассмотрим это немного позднее при изучении работы с DirectDraw в оконном режиме) и масштабирование изображения. Так же метод `IDirectDrawSurface7.Blt` позволяет вращать изображение, строить зеркальное отображение и т. д. В документации говорится, что метод `IDirectDrawSurface7.BltFast` выполняется примерно на 10% быстрее, чем метод `IDirectDrawSurface7.Blt`.

Производить блиттинг мы будем на вторичной поверхности — на нее мы станем отображать внеэкранный буфер:

```
// Получаем размер внеэкранного буфера
ZeroMemory(@ddsd, SizeOf(ddsd));
ddsd.dwSize := SizeOf(ddsd);
FImageBuffer.GetSurfaceDesc(ddsd);
imgRect := Rect(0, 0, ddsd.dwWidth - 1, ddsd.dwHeight - 1);
```

```
// Осуществляем блиттинг на вторичную поверхность
FDDSSecondary.Blt(@imgRect, FImageBuffer, NIL,
  DDBLTFAST_NOCOLORKEY or DDBLTFAST_WAIT, NIL);
```

Переключение страниц

Заполнив нужным образом вторичную поверхность, мы должны отобразить ее на экране. Это можно проделать при помощи так называемого *переключения страниц*, при котором содержимое вторичной поверхности мгновенно будет отображено на экране.

Мы не просто так задавали при создании первичной поверхности флаг `DDSCAPS_FLIP`. Именно он указывает на то, что для отображения будет использоваться переключение страниц. Переключение страниц осуществляется очень быстро и гладко, т. е. не будет заметно никаких мерцаний на экране. Давайте вспомним, что при создании первичной поверхности нами был также указан флаг `DDSCAPS_COMPLEX`, а это означает, что первичная поверхность будет состоять более чем из одной поверхности. В действительности вторичная поверхность является частью первичной, и при переключении страниц первичная и вторичная поверхности будут просто меняться местами.

Переключение страниц осуществляется методом `IDirectDrawSurface7.Flip`:

```
function Flip(
  lpDDSurfaceTargetOverride: IDirectDrawSurface7;
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `lpDDSurfaceTargetOverride` — поверхность, на которую будет переброшен указатель по окончании операции переключения. Если будет использовано нулевое значение, то подсистема DirectDraw произведет переключение автоматически и активной станет следующая поверхность в цепочке;
- `dwFlags` — флаг, определяющий операцию переключения. Может принимать нулевое значение. Мы будем использовать значение `DDFLIP_WAIT`, которое означает ожидание поверхности до тех пор, пока она не станет доступна.

Пример переключения поверхностей:

```
// Переключаем поверхности
FDDSPrimary.Flip(NIL, DDFLIP_WAIT);
```

А теперь давайте рассмотрим пример из каталога Example_03 (рис. 6.1). Этот пример содержит в себе все, что было описано ранее.

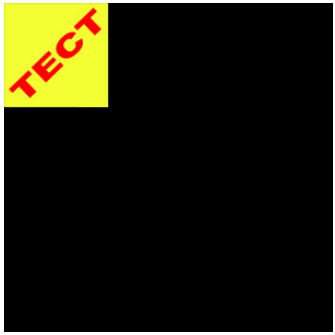


Рис. 6.1. Вывод изображения в полноэкранном режиме

Пример представляет собой полноэкранное приложение, в котором на черном фоне в левом верхнем углу будет выведено изображение из файла.

Потеря доступа к поверхности

При работе в полноэкранном режиме мы можем получить такую ошибку при выводе изображения, как `DDERR_NOEXCLUSIVEMODE`. Иными словами, приложение теряет эксклюзивный доступ к устройству. Такое может произойти, например, при переключении на другую задачу нажатием комбинации клавиш <Alt>+<Tab>. Для проверки такого рода ситуации имеется метод `IDirectDraw7.TestCooperativeLevel`:

```
function TestCooperativeLevel: HResult; stdcall;
```

Метод не имеет параметров. Его необходимо вызывать в цикле до тех пор, пока результатом его вызова не станет `DD_OK`.

Все существующие на данный момент поверхности должны быть восстановлены. Для этого предназначен метод `IDirectDraw7.RestoreAllSurfaces`:

```
function RestoreAllSurfaces: HResult; stdcall;
```

Рассмотрим пример из каталога Example_04 (рис. 6.2). После запуска приложения по экрану буду перемещаться несколько изображений. Попробуйте переключиться во время работы на другое приложение и обратно — не возникнет никаких ошибок, и все изображения будут корректно рисоваться и дальше.

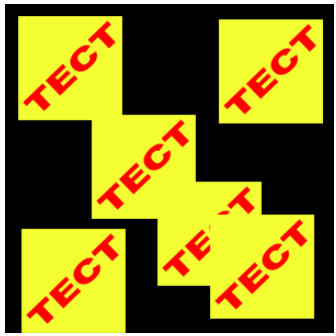


Рис. 6.2. Пример анимации в полноэкранном режиме

Давайте более детально разберем работу данного примера. У нас появилась процедура очистки поверхности `ClearSurface` (листинг 6.4).

Листинг 6.4. Очистка поверхности

```
{*********************************************************************}
{** Чистка поверхности                                              **}
{*********************************************************************}
function ClearSurface(Color: Cardinal; Surface: IDirectDrawSurface7) :
HRESULT;
var
  ddbltfx : TDDBLTFX;
begin
  // Результат по умолчанию
  Result := E_FAIL;
  if Surface = NIL then EXIT;

  // Задаем цвет фона
  ZeroMemory(@ddbltfx, SizeOf(ddbltfx));
  ddbltfx.dwSize := SizeOf(ddbltfx);
  ddbltfx.dwFillColor := Color;

  // Заполняем поверхность
  Result := Surface.Blt(NIL, NIL, NIL,
    DDBLT_COLORFILL or DDBLT_WAIT, @DDBLTFX);
end;
```

Фон мы будем заполнять черным цветом. Анимация будет производиться следующим образом: одна и та же внеэкранная поверхность будет прорисована на вторичной поверхности в разных местах, а затем будет происходить смена страниц. Положение и приращение координат каждого изображения будут описываться структурой:

```
TSurfacePos = record
   X,  Y: integer;
  dX, dY: integer;
end;
```

Процедура прорисовки будет иметь вид, представленный в листинге 6.5.

Листинг 6.5. Прорисовка сцены

```
{*********************************************************************}
{** Прорисовка сцены                                                **}
{*********************************************************************}
function TMainForm.RenderScene: HResult;
var
  imgRect: TRect;
  I: integer;
begin
  // результат по умолчанию
  Result := E_FAIL;
  if (FImageBuffer = NIL) or (FDDSSecondary = NIL) or
     (FDDSPrimary = NIL) then EXIT;

  // Очищаем вторичную поверхность
  ClearSurface(clBlack, FDDSSecondary);

  // Рисуем всеэкранную поверхность на вторичной MAX_SURFACES раз
  for I := 0 to MAX_SURFACES - 1 do
  begin
    with Surfaces[I] do
    begin
      inc(X, dX);
      inc(Y, dY);
```

```
      if (X + FimgWidth > MAX_WIDTH) or (X < 0) then
      begin
        dX := -dX;
        inc(X, dX);
      end;

      if (Y + FimgHeight > MAX_HEIGHT) or (Y < 0) then
      begin
        dY := -dY;
        inc(Y, dY);
      end;

      imgRect := Rect(X, Y, FimgWidth + X - 1, FimgHeight + Y - 1);

      FDDSSecondary.BltFast(X, Y, FImageBuffer, NIL, DDBLTFAST_WAIT);
    end;
  end;

  // Проверяем режим работы
  CheckCooperativeLevel;

  // Переключаем поверхности
  Result := FDDSPrimary.Flip(NIL, DDFLIP_WAIT);

  // Если есть потерянные поверхности, то...
  if Result = DDERR_SURFACELOST then
  begin
    // ... восстанавливаем их
    Result := RestoreSurfaces;
  end;
end;
```

Режим работы мы проверяем следующим образом (листинг 6.6).

Листинг 6.6. Проверка режима взаимодействия

```
{************************************************************************}
{** Проверка режима взаимодействия                                     **}
{************************************************************************}
function TMainForm.CheckCooperativeLevel: HResult;
begin
  // Проверка текущего режима
  Result := FDD7.TestCooperativeLevel;

  // Если что-то не так, то...
  while Result <> DD_OK do
  begin
    // Продолжаем обрабатывать сообщения
    Application.ProcessMessages;

    // И снова проверяем
    Result := FDD7.TestCooperativeLevel;
  end;
end;
```

А процедура восстановления поверхностей будет иметь вид, представленный в листинге 6.7.

Листинг 6.7. Восстанавление поверхности

```
{************************************************************************}
{** Восстанавливаем поверхности                                        **}
{************************************************************************}
function TMainForm.RestoreSurfaces: HResult;
var
  bitmap: TBitmap;
begin
  // Восстанавливаем все поверхности
  Result := FDD7.RestoreAllSurfaces;
  if Result <> DD_OK then Halt;
```

```
  // Обновляем содержимое внеэкранного буфера
  bitmap := TBitmap.Create;
  try
    bitmap.LoadFromFile('Texture.bmp');
    DDCopyBitmap(FImageBuffer, bitmap, bitmap.Width, bitmap.Height);
  finally
    FreeAndNil(bitmap);
  end;
end;
```

Цветовые ключи

Подсистема DirectDraw поддерживает операции с цветовыми ключами для различного рода поверхностей. *Цветовой ключ* представляет собой механизм отображения прозрачных изображений, т. е. под цветовым ключом понимается цвет, который будет считаться прозрачным.

В DirectDraw имеется специальная структура `TDDColorKey`, описывающая цветовой ключ. Она состоит из двух полей:

- `dwColorSpaceLowValue` — младшее значение цветового диапазона, которое будет использовано в качестве цветового ключа;
- `dwColorSpaceHighValue` — старшее значение цветового диапазона, которое будет использовано в качестве цветового ключа.

Установка цветового ключа для поверхности производится методом `IDirectDrawSurface7.SetColorKey`:

```
function SetColorKey(
  dwFlags: DWORD;
  lpDDColorKey: PDDColorKey):
HResult; stdcall;
```

Здесь:

- `dwFlags` — флаги, определяющие работу:
  - `DDCKEY_COLORSPACE` — структура содержит описание цветового пространства. Значение не будет установлено, если будет использован единственный цветовой ключ;
  - `DDCKEY_DESTBLT` — цветовой ключ будет использоваться как цветовой ключ назначения при операции блиттинга;

- DDCKEY_DESTOVERLAY — цветовой ключ будет использоваться как цветовой ключ назначения при оверлейных операциях;
- DDCKEY_SRCBLT — цветовой ключ будет использоваться как цветовой ключ источника при операции блиттинга;
- DDCKEY_SRCOVERLAY — цветовой ключ будет использоваться как цветовой ключ источника при оверлейных операциях;

❒ lpDDColorKey — указатель на структуру TDDColorKey, описывающую цветовой ключ.

В примере из каталога Example_05 (рис. 6.3) по экрану на фоне картинки будут перемещаться шарики. Белый цвет будет являться для шарика цветовым ключом.

Цветовой ключ для поверхности устанавливаем следующим образом:

```
// Устанавливаем цветовой ключ
ddck.dwColorSpaceLowValue := $00FFFFFF;
ddck.dwColorSpaceHighValue := ddck.dwColorSpaceLowValue;

Result := FImageBuffer.SetColorKey(DDCKEY_SRCBLT, @ddck);
```

А в методе прорисовки нам будет нужно указать флаг использования цветового ключа:

```
FDDSSecondary.BltFast(X, Y, FImageBuffer, NIL,
  DDBLTFAST_WAIT or DDBLTFAST_SRCCOLORKEY);
```

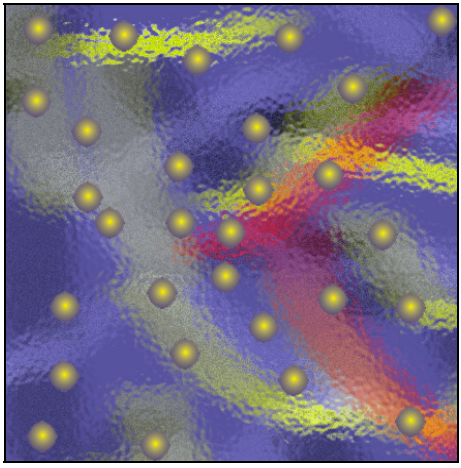


Рис. 6.3. Использование цветовых ключей

Палитра

Палитру можно условно представить как проиндексированный набор цветов. Если поверхность имеет палитру, то это означает, что каждая точка поверхности хранит не цвет, а индекс в массиве цветов.

Для работы с палитрой предназначен интерфейс `IDirectDrawPalette`. Первым делом необходимо подготовить набор цветов палитры (массив элементов типа `TPaletteEntry`), затем палитру следует создать методом `IDirectDraw7.CreatePalette` и назначить палитру поверхности методом `IDirectDrawSurface7.SetPalette`:

```
// Создаем палитру
Result := FDD7.CreatePalette(DDPCAPS_8BIT or DDPCAPS_ALLOW256,
  @FPalette, FDDPalette, NIL);
if Result <> DD_OK then Exit;


// Устанавливаем палиту для первичной поверхности
Result := FDDSPrimary.SetPalette(FDDPalette);
```

Следует заметить, что палитру мы можем использовать только на 8-битных поверхностях (и поверхностях с меньшей глубиной цвета).

Прямой доступ к поверхности

Прямой доступ к поверхности мы можем получить при помощи методов `IDirectDrawSurface7.Lock` и `IDirectDrawSurface7.Unlock`. Первый метод блокирует доступ к поверхности и возвращает ее описатель:

```
function Lock(
  lpDestRect: PRect;
  out lpDDSurfaceDesc: TDDSurfaceDesc2;
  dwFlags: DWORD;
  hEvent: THandle):
HResult; stdcall;
```

Здесь:

- `lpDestRect` — указатель на структуру, описывающую прямоугольную область поверхности, которая будет заблокирована;
- `lpDDSurfaceDesc` — структура, в которую будут занесены параметры поверхности;

- `dwFlags` — флаги, определяющие режим блокировки;
- `hEvent` — не используется. Должно стоять нулевое значение.

Второй метод разблокирует поверхность:

```
function Unlock(
  lpRect: PRect):
HResult; stdcall;
```

Здесь `lpRect` — указатель на структуру, описывающую прямоугольную область поверхности, которая использовалась при блокировке. Параметр должен принимать нулевое значение, если при блокировке параметр `lpDestRect` был нулевым.

Огонь

Давайте на примере рассмотрим работу с палитрой и прямым доступом к поверхности. Наверняка многие из вас уже неоднократно могли наблюдать такой интересный эффект, как эффект огня — в различных играх, экранных заставках и т. д. Пример из каталога Example_06 (рис. 6.4) представляет собой реализацию одного из алгоритмов огня.

Итак, теперь по порядку. Первым делом мы инициализируем подсистему DirectDraw, создаем 8-битную комплексную поверхность и получаем указатель на вторичную поверхность. Для описания элементов палитры мы будем использовать следующий массив:

```
FPalette: array[0..255] of TPaletteEntry;
```

Задав элементы палитры, мы создаем соответствующий объект и устанавливаем палитру на первичную поверхность. Собственно процедура инициализации теперь будет выглядеть так, как представлено в листинге 6.8.

Листинг 6.8. Инициализация подсистемы DirectDraw для создания эффекта огня

```
{*********************************************************************}
{** Инициализация подсистемы DirectDraw                             **}
{*********************************************************************}
function TMainForm.InitDirectDraw: HResult;
var
  ddsd: TDDSurfaceDesc2;
  ddscaps: TDDSCaps2;
  I: integer;
```

```
begin
  // Создание объекта DirectDraw
  Result := DirectDrawCreateEx(NIL, FDD7, IDirectDraw7, NIL);
  if Result <> DD_OK then Exit;

  // Установка уровня взаимодействия
  Result := FDD7.SetCooperativeLevel(Handle,
    DDSCL_FULLSCREEN or DDSCL_EXCLUSIVE);
  if Result <> DD_OK then Exit;

  // Установка полноэкранного режима
  Result := FDD7.SetDisplayMode(MAX_WIDTH, MAX_HEIGHT, MAX_BPP,
    MAX_RATE, 0);
  if Result <> DD_OK then Exit;

  // Параметры основной поверхности
  ZeroMemory(@ddsd, SizeOf(ddsd));
  ddsd.dwSize := SizeOf(ddsd);
  ddsd.dwFlags := DDSD_CAPS or DDSD_BACKBUFFERCOUNT;
  ddsd.ddsCaps.dwCaps :=
    DDSCAPS_PRIMARYSURFACE or DDSCAPS_COMPLEX or DDSCAPS_FLIP;
  ddsd.dwBackBufferCount := 1;

  // Создаем первичную поверхность
  Result := FDD7.CreateSurface(ddsd, FDDSPrimary, NIL);
  if Result <> DD_OK then Exit;

  // Вторичная поверхность
  ZeroMemory(@ddscaps, SizeOf(ddscaps));
  ddscaps.dwCaps := DDSCAPS_BACKBUFFER;
  Result := FDDSPrimary.GetAttachedSurface(ddscaps, FDDSSecondary);
  if Result <> DD_OK then Exit;

  // Заполняем значения цветов палитры
  for I := 0 to 63 do
```

```
  begin
    FPalette[I].peRed   := I * 4;
    FPalette[I].peGreen := 0;
    FPalette[I].peBlue  := 0;
  end;

  for I := 64 to 127 do
  begin
    FPalette[I].peRed   := 255;
    FPalette[I].peGreen := (I - 64) * 4;
    FPalette[I].peBlue  := 0;
  end;

  for I := 128 to 191 do
  begin
    FPalette[I].peRed   := 255;
    FPalette[I].peGreen := 255;
    FPalette[I].peBlue  := (I - 128) * 4;
  end;

  for I := 192 to 255 do
  begin
    FPalette[I].peRed   := 255;
    FPalette[I].peGreen := 255;
    FPalette[I].peBlue  := 255;
  end;

  // Создаем палитру
  Result := FDD7.CreatePalette(DDPCAPS_8BIT or DDPCAPS_ALLOW256,
    @FPalette, FDDPalette, NIL);
  if Result <> DD_OK then Exit;

  // Устанавливаем палиту для первичной поверхности
  Result := FDDSPrimary.SetPalette(FDDPalette);
end;
```

Определим следующий тип данных:

```
{**} TYPE {************************************************************}
  PFireDesc = ^TFireDesc;
  TFireDesc = array[0..MAX_WIDTH * MAX_HEIGHT - 1] of BYTE;
```

Алгоритм построения картинки огня будет следующим: мы возьмем два массива элементов типа `TFireDesc`, имеющих размер, равный размеру экрана при установленном нами разрешении:

```
FFirePrimary,
FFireSecondary: TFireDesc;
```

Первый массив будет содержать последнее описание сцены с огнем (т. е. каждый элемент массива будет индексом в палитре цветов), а элементы второго массива будут высчитываться так: каждая точка этого массива является средним значением соответствующих точек, расположенных вокруг нее из первого массива.

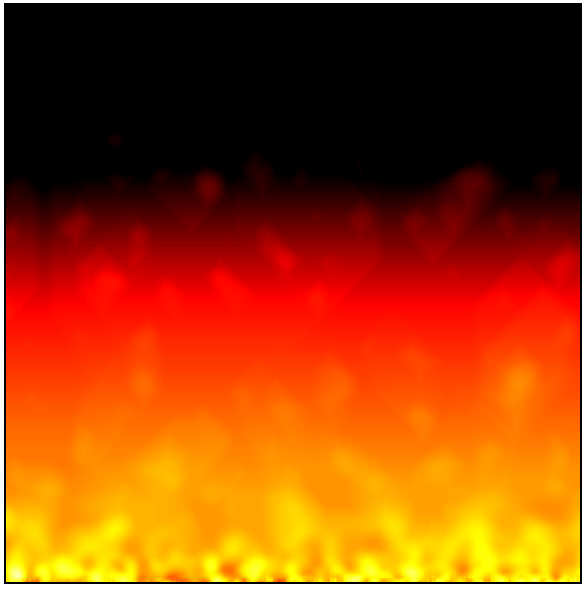


Рис. 6.4. Огонь

Процедура прорисовки сцены будет иметь вид, представленный в листинге 6.9.

Листинг 6.9. Прорисовка сцены для создания эффекта огня

```
{*********************************************************************}
{** Прорисовка сцены                                                **}
{*********************************************************************}
function TMainForm.RenderScene: HResult;
var
  I, J, K:  Integer;
  Temp:     Pointer;
  FireDesc: PFireDesc;
  SurfaceDescription: TDDSurfaceDesc2;
  hr: HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;
  if (FDD7 = NIL) or (FDDSPrimary = NIL) or
    (FDDSSecondary = NIL) or (FDDPalette = NIL) then EXIT;

  // Проверяем режим работы
  CheckCooperativeLevel;

  // Начальные значения
  I := 0;
  while (I < MAX_WIDTH) do
  begin
    FFirePrimary[I] := Random(5) * 255;
    FFirePrimary[I + MAX_WIDTH] := Random(5) * 255;
    inc(I, Random(3) + 1);
  end;

  // Вычисляем цвет каждой точки
  for I := MAX_WIDTH * 2 + 1 to MAX_WIDTH * MAX_HEIGHT - 2 do
    FFireSecondary[I] := (
      (FFirePrimary[I - 1] +
       FFirePrimary[I + 1] +
       FFirePrimary[I] +
```

```
      FFirePrimary[I - (MAX_WIDTH - 1)] +
      FFirePrimary[I - (MAX_WIDTH + 1)] +
      FFirePrimary[I - (MAX_WIDTH * 2 - 1)] +
      FFirePrimary[I - (MAX_WIDTH * 2)] +
      FFirePrimary[I - (MAX_WIDTH * 2 + 1)]) shr 3);

// Блокируем доступ к поверхности
ZeroMemory(@SurfaceDescription, SizeOf(TDDSurfaceDesc2));
SurfaceDescription.dwSize := SizeOf(TDDSurfaceDesc2);

hr := FDDSSecondary.Lock(NIL, SurfaceDescription,
  DDLOCK_SURFACEMEMORYPTR or DDLOCK_WRITEONLY or DDLOCK_WAIT, 0);

// Если потерян доступ к поверхности, то...
if hr = DDERR_SURFACELOST then
begin
  // ... восстанавливаем и...
  RestoreSurfaces;

  // ... снова блокируем
  FDDSSecondary.Lock(NIL, SurfaceDescription,
    DDLOCK_SURFACEMEMORYPTR or DDLOCK_WRITEONLY or DDLOCK_WAIT, 0);
end;

try
  // Получаем указатель на начало блока описания точек поверхности
  FireDesc := SurfaceDescription.lpSurface;

  J := 0;
  K := MAX_WIDTH * MAX_HEIGHT - MAX_WIDTH;

  // Копируем на поверхность массив просчитанных цветов
  for I := 0 to MAX_HEIGHT — 1 do
  begin
    CopyMemory(@FireDesc[J], @FFireSecondary[K], MAX_WIDTH);
    inc(J, SurfaceDescription.lPitch);
```

```
      dec(K, MAX_WIDTH);
    end;

  finally

    // Разблокировка вторичной поверхности
    FDDSSecondary.Unlock(NIL);
  end;

  // Меняем местами массивы
  Temp := @FFirePrimary;
  FFirePrimary := FFireSecondary;
  FFireSecondary := TFireDesc(Temp^);

  // Переключаем поверхности
  Result := FDDSPrimary.Flip(NIL, DDFLIP_WAIT);

  // Если есть потерянные поверхности, то...
  if Result = DDERR_SURFACELOST then
  begin
    // ... восстанавливаем их
    Result := RestoreSurfaces;
  end;
end;
```

Следует также отметить и тот факт, что при восстановлении поверхностей мы восстанавливаем и палитру (листинг 6.10).

Листинг 6.10. Восстанавление поверхности для создания эффекта огня

```
{*********************************************************************}
{** Восстанавливаем поверхности                                     **}
{*********************************************************************}
function TMainForm.RestoreSurfaces: HResult;
begin
  // Восстанавливаем все поверхности
```

```
  Result := FDD7.RestoreAllSurfaces;
  if Result <> DD_OK then Halt;

  // Восстанавливаем палитру
  Result := FDDSPrimary.SetPalette(FDDPalette);
end;
```

Оконный режим работы

Работа в оконном режиме с подсистемой DirectDraw сходна с работой в полноэкранном режиме, за исключением нескольких оговорок. Не секрет, что полноэкранные приложения более производительные, чем оконные. Мы не сможем использовать смену страниц в полноэкранном режиме, и все рисование будет производиться при помощи блиттинга на основную поверхность. Это означает, что мы должны установить нормальный уровень взаимодействия, а в качестве вторичной поверхности использовать внеэкранный буфер. Так же нам понадобится такой объект DirectDraw, как *клипер*. Это объект с интерфейсом `IDirectDrawClipper`, предназначенный для определения области отсечения.

Рассмотрим пример из каталога Example_07. Этот пример повторяет по своей сути пример Example_05. Основное отличие — возможность работы в двух режимах: оконном и полноэкранном. Посмотрите на процесс инициализации DirectDraw (листинг 6.11).

Листинг 6.11. Инициализация подсистемы DirectDraw в зависимости от режима

```
{*********************************************************************}
{** Инициализация подсистемы DirectDraw                             **}
{*********************************************************************}
function TMainForm.InitDirectDraw: HResult;
var
  ddsd: TDDSurfaceDesc2;
  ddscaps: TDDSCaps2;
  bitmap: TBitmap;
  ddck: TDDColorKey;
begin
  // Создание объекта DirectDraw
  Result := DirectDrawCreateEx (NIL, FDD7, IDirectDraw7, NIL);
  if Result <> DD_OK then Exit;
```

```
if FWindowed then
begin
  // Установка уровня взаимодействия
  Result := FDD7.SetCooperativeLevel(Handle, DDSCL_NORMAL);
  if Result <> DD_OK then Exit;

  // Параметры основной поверхности
  ZeroMemory(@ddsd, SizeOf(ddsd));
  ddsd.dwSize := SizeOf(ddsd);
  ddsd.dwFlags := DDSD_CAPS;
  ddsd.ddsCaps.dwCaps := DDSCAPS_PRIMARYSURFACE;
  ddsd.dwBackBufferCount := 1;

  // Создаем первичную поверхность
  Result := FDD7.CreateSurface(ddsd, FDDSPrimary, NIL);
  if Result <> DD_OK then Exit;

  // Создаем клипер
  Result := FDD7.CreateClipper(0, FDDClipper, NIL);
  if Result <> DD_OK then Exit;

  // Привязываем клипер к окну приложения
  Result := FDDClipper.SetHWnd(0, Handle);
  if Result <> DD_OK then Exit;

  // Устанавливаем клипер на первичную поверхность
  Result := FDDSPrimary.SetClipper(FDDClipper);
  if Result <> DD_OK then Exit;

  // Вторичная внеэкранная поверхность
  ZeroMemory (@ddsd, SizeOf(ddsd));
  ddsd.dwSize   := SizeOf(ddsd);
  ddsd.dwFlags  := DDSD_CAPS or DDSD_WIDTH or DDSD_HEIGHT;
  ddsd.dwWidth  := ClientWidth;
  ddsd.dwHeight := ClientHeight;
  ddsd.ddsCaps.dwCaps := DDSCAPS_OFFSCREENPLAIN;
```

```
  // Создаем внеэкранную поверхность
  if FAILED(FDD7.CreateSurface(ddsd, FDDSSecondary, NIL)) then EXIT;
end else
begin
  // Установка уровня взаимодействия
  Result := FDD7.SetCooperativeLevel(Handle,
    DDSCL_FULLSCREEN or DDSCL_EXCLUSIVE);
  if Result <> DD_OK then Exit;

  // Установка полноэкранного режима
  Result := FDD7.SetDisplayMode(MAX_WIDTH, MAX_HEIGHT, MAX_BPP,
    MAX_RATE, 0);
  if Result <> DD_OK then Exit;

  // Параметры основной поверхности
  ZeroMemory(@ddsd, SizeOf(ddsd));
  ddsd.dwSize := SizeOf(ddsd);
  ddsd.dwFlags := DDSD_CAPS or DDSD_BACKBUFFERCOUNT;
  ddsd.ddsCaps.dwCaps :=
    DDSCAPS_PRIMARYSURFACE or DDSCAPS_COMPLEX or DDSCAPS_FLIP;
  ddsd.dwBackBufferCount := 1;

  // Создаем первичную поверхность
  Result := FDD7.CreateSurface(ddsd, FDDSPrimary, NIL);
  if Result <> DD_OK then Exit;

  // Вторичная поверхность
  ZeroMemory(@ddscaps, SizeOf(ddscaps));
  ddscaps.dwCaps := DDSCAPS_BACKBUFFER;
  Result := FDDSPrimary.GetAttachedSurface(ddscaps, FDDSSecondary);
end;

// Внеэкранная поверхность
bitmap := TBitmap.Create;
try
  bitmap.LoadFromFile('Texture.bmp');
```

```
  // Параметры внеэкранной поверхности
  ZeroMemory (@ddsd, SizeOf(ddsd));
  ddsd.dwSize   := SizeOf(ddsd);
  ddsd.dwFlags  := DDSD_CAPS or DDSD_WIDTH or DDSD_HEIGHT;
  FimgWidth     := bitmap.Width;
  FimgHeight    := bitmap.Height;
  ddsd.dwWidth  := FimgWidth;
  ddsd.dwHeight := FimgHeight;
  ddsd.ddsCaps.dwCaps := DDSCAPS_OFFSCREENPLAIN;

  // Создаем внеэкранную поверхность
  if FAILED(FDD7.CreateSurface(ddsd, FImageBuffer, NIL)) then EXIT;

  DDCopyBitmap(FImageBuffer, bitmap, bitmap.Width, bitmap.Height);

  // Устанавливаем цветовой ключ
  ddck.dwColorSpaceLowValue := $00FFFFFF;
  ddck.dwColorSpaceHighValue := ddck.dwColorSpaceLowValue;

  Result := FImageBuffer.SetColorKey(DDCKEY_SRCBLT, @ddck);
finally
  FreeAndNil(bitmap);
end;

// Вторая внеэкранная поверхность
bitmap := TBitmap.Create;
try
  bitmap.LoadFromFile('Back.bmp');

  // Параметры внеэкранной поверхности
  ZeroMemory (@ddsd, SizeOf(ddsd));
  ddsd.dwSize   := SizeOf(ddsd);
  ddsd.dwFlags  := DDSD_CAPS or DDSD_WIDTH or DDSD_HEIGHT;
  ddsd.dwWidth  := bitmap.Width;
  ddsd.dwHeight := bitmap.Height;
  ddsd.ddsCaps.dwCaps := DDSCAPS_OFFSCREENPLAIN;
```

```
    // Создаем внеэкранную поверхность
    if FAILED(FDD7.CreateSurface(ddsd, FBackImageBuffer, NIL)) then EXIT;

    DDCopyBitmap(FBackImageBuffer, bitmap, bitmap.Width, bitmap.Height);
  finally
    FreeAndNil(bitmap);
  end;
end;
```

Инициализация будет производиться в зависимости от флага, определяющего режим работы. Как уже было сказано выше, в оконном режиме отсутствует возможность переключения страниц, и из-за этого нам придется рисовать на экране в каждом режиме по-разному:

```
// Выводим на экран в зависимости от режима
if FWindowed then
begin
  imgRect := BoundsRect;
  FDDSPrimary.Blt(@imgRect, FDDSSecondary, NIL,
    DDBLTFAST_WAIT or DDBLTFAST_SRCCOLORKEY, NIL);
end else
begin
  FDDSPrimary.Flip(NIL, DDBLT_WAIT);
end;
```

Переключение между режимами будет производиться точно так же, как и в случае с подсистемой Direct3D (листинг 6.12).

Листинг 6.12. Переключение между режимами

```
{*********************************************************************}
{** Обработка нажатия клавиш                                        **}
{*********************************************************************}
procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  // Нажатие <ALT>+<Enter> приводит к смене режима
  if (ssAlt in Shift) and (Key = VK_RETURN) then
```

```
  begin
    FWindowed := not FWindowed;
    FreeDirectDraw;
    InitDirectDraw;
  end;
end;
```

Оверлеи DirectX

Последнее, что мы изучим в этой главе, — работа с оверлеями DirectX. Единственное, что понадобится нам сейчас для работы (помимо установленной системы DirectX на компьютере), — это видеокарта с поддержкой оверлеев.

Итак, под *оверлеями* следует понимать особый род поверхностей. Эти поверхности предназначены для отображения видео и вывода изображений над первичной поверхностью, не производя при этом блиттинг на нее и не изменяя ее содержимое любым другим способом. Поддержка оверлеев полностью осуществляется аппаратными средствами.

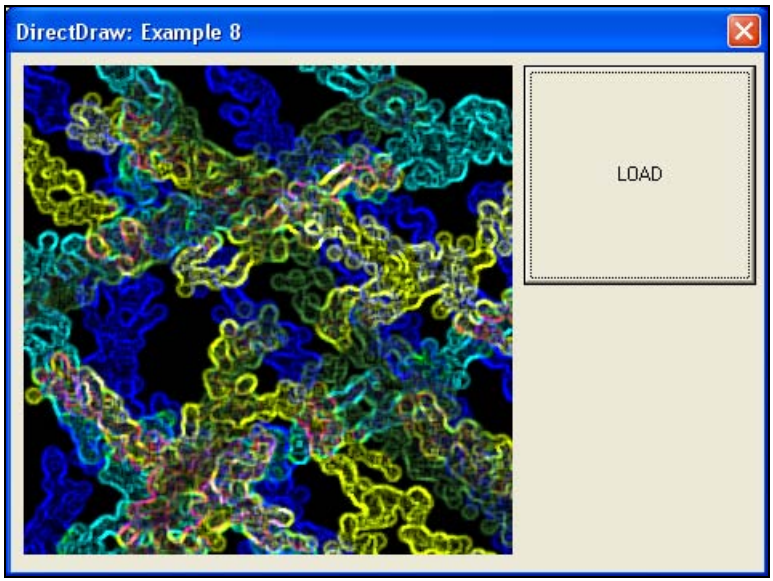


Рис. 6.5. Работа с оверлеями DirectX

Рассмотрим пример из каталога Example_08 (рис. 6.5). В этом примере имеется возможность загрузки bmp-файла в качестве фонового рисунка на рабочий стол. Следует заметить, что ширина и высота картинок должны быть степенью двойки, например, 64×64, 256×512 и т. п. Принцип работы будет следующим: создаем первичную поверхность, затем создаем оверлей. При его создании мы обязательно должны указать флаги `DDSCAPS_OVERLAY` и `DDSCAPS_VIDEOMEMORY`. Флаг `DDSCAPS_VIDEOMEMORY` обязателен — оверлеи могут быть созданы только в видеопамяти. Формат точек поверхности будет подобран в процессе ее создания. Оверлейная поверхность будет комплексной и с возможностью переключения страниц.

Для вывода на экран изображения мы должны убрать заставку с рабочего стола и установить свой цвет фона, а в конце работы приложения обязаны все это восстановить. Для этого имеется специальная функция, которую мы будем вызывать при старте приложения и в конце его работы. Это наша функция, и она имеет название `UpdateDesktop`.

Общий алгоритм вывода изображения на оверлей можно представить следующим образом:

1. Удаляем оверлей.
2. Загружаем изображение.
3. Создаем оверлей с нужными параметрами.
4. Копируем изображение на оверлейную поверхность.
5. Выводим изображение на экран.

Полный текст примера работы с оверлеями проводится в листинге 6.13.

Листинг 6.13. Текст модуля FormMain.pas примера DirectDraw\Example_08

```
UNIT FormMain;
{*************************************************************************}
{** DirectDraw: Example_08                                              **}
{** Автор: Есенин Сергей Анатольевич                                    **}
{*************************************************************************}

{**} INTERFACE {*********************************************************}

{**} USES {**************************************************************}
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, DirectDraw, ExtCtrls, AppEvnts, Registry, ExtDlgs,
  StdCtrls;
```

```
{**} TYPE {************************************************************}
  TMainForm = class(TForm)
    imageMain: TImage;
    buttonLoad: TButton;
    openPictureDialogMain: TOpenPictureDialog;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure buttonLoadClick(Sender: TObject);
  PRIVATE
    FDD7: IDirectDraw7;
    FDDSPrimary: IDirectDrawSurface7;
    FDDSSecondary: IDirectDrawSurface7;
    FDDSOverlay: IDirectDrawSurface7;

    FColor:     DWORD;
    FWallpaper: string;

    FImage:     array of DWORD;
    FWidth:     integer;
    FHeight:    integer;

    function InitDirectDraw: HResult;
    procedure FreeDirectDraw;

    function CreateOverlay: HResult;
    procedure DeleteOverlay;

    procedure LoadImage(ImagePath: string);
    procedure FreeImage;

    function GetPixelValue: DWORD;

    function ImageToSurface: HResult;
    function RefreshOverlay: HResult;
  PUBLIC
  END;
```

```
{**} VAR {************************************************************}
  MainForm: TMainForm;
  oldColor: DWORD = CLR_INVALID;
  desktopIndex: DWORD = COLOR_DESKTOP;
  backColor: DWORD = $00010101;

{**} IMPLEMENTATION {*************************************************}

{$R *.dfm}

{**} { TMainForm } {**************************************************}

{************************************************************************}
{** Установка параметров рабочего стола                                **}
{************************************************************************}
function UpdateDesktop(var color: DWORD; var wallpaper: string): HResult;
var
  WallpaperNew: string;
  ColorNew: DWORD;
  Registry: TRegistry;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Запоминаем значения
  WallpaperNew := wallpaper;
  ColorNew := color;

  Registry := TRegistry.Create;
  try
    Registry.RootKey := HKEY_CURRENT_USER;

    // Открываем ключ реестра с параметрами рабочего стола
    if Registry.OpenKey('Control Panel\Desktop', FALSE) then
    begin
      // Меняем обои
```

```
      Wallpaper := Registry.ReadString('Wallpaper');
      Registry.WriteString('Wallpaper', WallpaperNew);
      Registry.CloseKey;

      // Устанавливаем цвет фона рабочего стола
      Color := GetSysColor(desktopIndex);
      SetSysColors(1, desktopIndex, ColorNew);

      // обновляем параметры
      SystemParametersInfo(SPI_SETDESKWALLPAPER, 0,
        NIL, SPIF_SENDWININICHANGE);

      Result := S_OK;
    end;

  finally
    FreeAndNil(Registry);
  end;
end;

{*************************************************************************}
{** Инициализация подсистемы DirectDraw                                 **}
{*************************************************************************}
function TMainForm.InitDirectDraw: HResult;
var
  ddsd: TDDSurfaceDesc2;
begin
  // Создание объекта DirectDraw
  Result := DirectDrawCreateEx (NIL, FDD7, IDirectDraw7, NIL);
  if Result <> DD_OK then EXIT;

  // Установка уровня взаимодействия
  Result := FDD7.SetCooperativeLevel(Handle, DDSCL_NORMAL);
  if Result <> DD_OK then EXIT;

  // Параметры основной поверхности
  ZeroMemory(@ddsd, SizeOf(ddsd));
```

```
  ddsd.dwSize := SizeOf(ddsd);
  ddsd.dwFlags := DDSD_CAPS;
  ddsd.ddsCaps.dwCaps := DDSCAPS_PRIMARYSURFACE;
  ddsd.dwBackBufferCount := 1;

  // Создаем первичную поверхность
  Result := FDD7.CreateSurface(ddsd, FDDSPrimary, NIL);
end;

{************************************************************************}
{** Освобождаем ресурсы                                                **}
{************************************************************************}
procedure TMainForm.FreeDirectDraw;
begin
  FDDSPrimary := NIL;
  FDD7 := NIL;
end;

{************************************************************************}
{** Инициализаця подсистемы при создании формы                         **}
{************************************************************************}
procedure TMainForm.FormCreate(Sender: TObject);
begin
  FColor := backColor;
  FWallpaper := '';
  UpdateDesktop(FColor, FWallpaper);

  if FAILED(InitDirectDraw) then
  begin
    ShowMessage('Error initializing DirectDraw...');
    Halt;
  end;
end;

{************************************************************************}
{** Освобождаем ресурсы при завершении работы программы               **}
{************************************************************************}
```

```
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  DeleteOverlay;
  UpdateDesktop(FColor, FWallpaper);

  FreeImage;
  FreeDirectDraw;
end;

{**************************************************************************}
{** Повторное создание оверлея с новым изображением                      **}
{**************************************************************************}
procedure TMainForm.buttonLoadClick(Sender: TObject);
begin
  if openPictureDialogMain.Execute then
  begin
    // Удаляем оверлей
    DeleteOverlay;

    // Загружаем изображение
    LoadImage(openPictureDialogMain.FileName);
    imageMain.Picture.LoadFromFile(openPictureDialogMain.FileName);

    // Создаем оверлей с нужными параметрами
    CreateOverlay;

    // Копируем изображение на оверлейную поверхность
    ImageToSurface;

    // Выводим изображение на экран
    RefreshOverlay;
  end;
end;

{**************************************************************************}
{** Загрузка изображения                                                 **}
{**************************************************************************}
```

```
procedure TMainForm.LoadImage(ImagePath: string);
var
  x, y, I: integer;
  bitmap: TBitmap;
begin
  bitmap := TBitmap.Create;
  try
    bitmap.LoadFromFile(ImagePath);

    FWidth  := bitmap.Width;
    FHeight := bitmap.Height;

    FreeImage;
    SetLength(FImage, FWidth * FHeight);

    for x := 0 to FWidth - 1 do
      for y := 0 to FHeight - 1 do
      begin
        I := x + y * FWidth;
        FImage[I] := bitmap.Canvas.Pixels[x, y];
      end;
  finally
    FreeAndNIL(bitmap);
  end;
end;

{**************************************************************************}
{** Освобождение ресурсов изображения                                    **}
{**************************************************************************}
procedure TMainForm.FreeImage;
begin
  SetLength(FImage, 0);
end;

{**************************************************************************}
{** Создание оверлея                                                     **}
{**************************************************************************}
```

```
function TMainForm.CreateOverlay: HResult;
var
  ddsd: TDDSurfaceDesc2;
  ddscaps: TDDSCaps2;
  ddformats: array[0..3] of TDDPixelFormat;
  I: integer;
begin
  // Задаем форматы точек, т. к. заранее не известно,
  // какой из форматов будет работать на вашей видеокарте
  ZeroMemory(@ddformats, SizeOf(ddformats));
  ddformats[0].dwSize := sizeof(TDDPixelFormat);
  ddformats[0].dwFlags := DDPF_RGB;
  ddformats[0].dwFourCC := 0;
  ddformats[0].dwRGBBitCount := 32;
  ddformats[0].dwRBitMask := $F800;
  ddformats[0].dwGBitMask := $07e0;
  ddformats[0].dwBBitMask := $001F;
  ddformats[0].dwRGBAlphaBitMask := 0;

  ddformats[1].dwSize := sizeof(TDDPixelFormat);
  ddformats[1].dwFlags := DDPF_RGB;
  ddformats[1].dwFourCC := 0;
  ddformats[1].dwRGBBitCount := 32;
  ddformats[1].dwRBitMask := $7C00;
  ddformats[1].dwGBitMask := $03e0;
  ddformats[1].dwBBitMask := $001F;
  ddformats[1].dwRGBAlphaBitMask := 0;

  ddformats[2].dwSize := sizeof(TDDPixelFormat);
  ddformats[2].dwFlags := DDPF_FOURCC;
  ddformats[2].dwFourCC := MAKEFOURCC('U','Y','V','Y');
  ddformats[2].dwYUVBitCount := 32;

  ddformats[3].dwSize := sizeof(TDDPixelFormat);
  ddformats[3].dwFlags := DDPF_FOURCC;
  ddformats[3].dwFourCC := MAKEFOURCC('Y','U','Y','2');
  ddformats[3].dwYUVBitCount := 32;
```

```
  // Параметры оверлейной поверхности
  ZeroMemory(@ddsd, SizeOf(ddsd));
  ddsd.dwSize := sizeof(ddsd);
  ddsd.ddsCaps.dwCaps    :=
    DDSCAPS_OVERLAY or
    DDSCAPS_FLIP or
    DDSCAPS_COMPLEX or
    DDSCAPS_VIDEOMEMORY;
  ddsd.dwFlags           :=
    DDSD_CAPS or
    DDSD_HEIGHT or
    DDSD_WIDTH or
    DDSD_BACKBUFFERCOUNT or
    DDSD_PIXELFORMAT;
  ddsd.dwWidth           := FWidth;
  ddsd.dwHeight          := FHeight;
  ddsd.dwBackBufferCount := 1;

  // Пробуем создать поверхность
  for I := 0 to 3 do
  begin
    ZeroMemory(@ddsd.ddpfPixelFormat, SizeOf(ddsd.ddpfPixelFormat));
    ddsd.ddpfPixelFormat := ddformats[I];

    Result := FDD7.CreateSurface(ddsd, FDDSOverlay, NIL);
    if Result = DD_OK then BREAK;
  end;

  if Result <> DD_OK then EXIT;

  // Если оверлейная поверхность создана, то получаем ее
  // вторичную поверхность
  ZeroMemory(@ddscaps, SizeOf(ddscaps));
  ddscaps.dwCaps  := DDSCAPS_BACKBUFFER;
  Result := FDDSOverlay.GetAttachedSurface(ddscaps, FDDSSecondary);
end;
```

```
{************************************************************************}
{** Удаление оверлея                                                   **}
{************************************************************************}
procedure TMainForm.DeleteOverlay;
begin
  FDDSSecondary := NIL;
  FDDSOverlay := NIL;
end;

{************************************************************************}
{** Копирование изображения на оверлей                                 **}
{************************************************************************}
function TMainForm.ImageToSurface: HResult;
var
  ddsd:  DDSURFACEDESC2;
  w, h:  integer;
  pitch: integer;
  dest:  PBYTE;
  x, y:  integer;
  clr:   DWORD;
  Y0, U0, V0,
  Y1, U1, V1: BYTE;
  dest2: PDWORD;
  pixel: DWORD;
begin
  ZeroMemory(@ddsd, sizeof(ddsd));
  ddsd.dwSize := sizeof(ddsd);

  // Блокируем доступ к оверлею
  Result := FDDSOverlay.Lock(NIL, ddsd,
    DDLOCK_SURFACEMEMORYPTR or DDLOCK_WRITEONLY or DDLOCK_WAIT, 0);
  if Result <> DD_OK then EXIT;

  try
```

```
// Рисуем в зависимости от формата
if (ddsd.ddpfPixelFormat.dwFlags = DDPF_FOURCC) then
begin
  w       := ddsd.dwWidth;
  h       := ddsd.dwHeight;
  pitch   := ddsd.lPitch;
  dest    := ddsd.lpSurface;

  for y := 0 to h - 1 do
  begin
    x := 0;
    while (x < w - 1) do
    begin
      clr := FImage[x + y * FWidth];

      Y0 := trunc(
         0.29 * GetRValue(clr) +
         0.59 * GetGValue(clr) +
         0.14 * GetBValue(clr));
      U0 := trunc(
         128.0 —
         0.14 * GetRValue(clr) —
         0.29 * GetGValue(clr) +
         0.43 * GetBValue(clr));
      V0 := trunc(
         128.0 +
         0.36 * GetRValue(clr) —
         0.29 * GetGValue(clr) —
         0.07 * GetBValue(clr));

      clr := FImage[x + y * FWidth + 1];
      Y1 := trunc(
         0.29 * GetRValue(clr) +
         0.57 * GetGValue(clr) +
         0.14 * GetBValue(clr));
```

```
      U1 := trunc(
         128.0 -
         0.14 * GetRValue(clr) -
         0.29 * GetGValue(clr) +
         0.43 * GetBValue(clr));
      V1 := trunc(
         128.0 +
         0.36 * GetRValue(clr) -
         0.29 * GetGValue(clr) -
         0.07 * GetBValue(clr));

      if (ddsd.ddpfPixelFormat.dwFourCC =
        MAKEFOURCC('Y','U','Y','2')) then
      begin
        dest^ := Y0; inc(dest, sizeof(Byte));
        dest^ := (U0 + U1) div 2; inc(dest, sizeof(Byte));
        dest^ := Y1; inc(dest, sizeof(Byte));
        dest^ := (V0 + V1) div 2; inc(dest, sizeof(Byte));
      end else
      begin
        dest^ := (U0 + U1) div 2; inc(dest, sizeof(Byte));
        dest^ := Y0; inc(dest, sizeof(Byte));
        dest^ := (V0 + V1) div 2; inc(dest, sizeof(Byte));
        dest^ := Y1; inc(dest, sizeof(Byte));
      end;

      inc(x, 2);
    end;

    inc(dest, (pitch - w * 2));
  end;

endelse
begin
  w       := ddsd.dwWidth;
```

```
      h       := ddsd.dwHeight;
      pitch   := ddsd.lPitch div 2;
      dest2   := ddsd.lpSurface;

      for y := 0 to h - 1 do
      begin
        for x := 0 to w - 1 do
        begin
          clr := FImage[x + y * FWidth];
          Pixel := RGB(GetBValue(clr), GetGValue(clr), GetRValue(clr));
          dest2^ := pixel;
          inc(dest2);
        end;

        inc(dest2, (pitch - w * 2));
      end;
    end;

  finally
    // Разблокировка доступа к поверхности
       Result := FDDSOverlay.Unlock(NIL);
  end;
end;

{*************************************************************************}
{** Получение цвета                                                     **}
{*************************************************************************}
function TMainForm.GetPixelValue: DWORD;
var
  oldPixel: COLORREF;
  DC: HDC;
  pixel: DWORD;
  ddsd: DDSURFACEDESC2;
  hr: HRESULT;
```

```
begin
  // Значения по умолчанию
  pixel := CLR_INVALID;
  oldPixel := 0;

       hr := FDDSPrimary.GetDC(dc);
  if (SUCCEEDED(hr)) then
  begin
    // Получаем цвет в верхнем левом углу
    oldPixel := GetPixel(DC, 0, 0);

    // Устанавливаем цвет в верхнем левом углу
    SetPixel(DC, 0, 0, backColor);
    FDDSPrimary.ReleaseDC(dc);
  end;

  ddsd.dwSize := sizeof(ddsd);

  // Блокируем доступ к поверхности
  hr := FDDSPrimary.Lock(NIL, ddsd, DDLOCK_SURFACEMEMORYPTR or
    DDLOCK_WRITEONLY or DDLOCK_WAIT, 0);
  if (SUCCEEDED(hr)) then
  begin
    // Получаем первую точку поверхности
    pixel := DWORD(ddsd.lpSurface^);

    // Если формат менее, чем 32 бита, то...
    if (ddsd.ddpfPixelFormat.dwRGBBitCount < 32) then
      // ... приводим значение к нужному виду
      pixel := (pixel and (1 shl ddsd.ddpfPixelFormat.dwRGBBitCount)) -1;

    // разблокируем доступ к поверхности
    FDDSPrimary.Unlock(NIL);
  end;
```

```
  hr := FDDSPrimary.GetDC(DC);
  if (SUCCEEDED(hr)) then
  begin
    // Возвращаем цвет точки в первоначальное значение
    SetPixel(DC, 0, 0, oldPixel);
    FDDSPrimary.ReleaseDC(DC);
  end;

  // Возвращаем значение цвета
  Result := pixel;
end;

{**************************************************************************}
{** Обновление оверлея                                                   **}
{**************************************************************************}
function TMainForm.RefreshOverlay: HResult;
var
  rcSrc,
  rcDst: TRect;
  OverlayFX: TDDOverlayFX;
begin
  // Результат по умолчанию
  result := E_FAIL;
  if (FDD7 = NIL) or (FDDSPrimary = NIL) or
     (FDDSOverlay = NIL) or (FDDSSecondary = NIL) then EXIT;

  // Область для копирования
  rcDst.left   := 0;
  rcDst.top    := 0;
  rcDst.right  := GetSystemMetrics(SM_CXSCREEN) - 1;
  rcDst.bottom := GetSystemMetrics(SM_CYSCREEN) - 1;

  // Область копирования
  rcSrc.left   := 0;
  rcSrc.top    := 0;
```

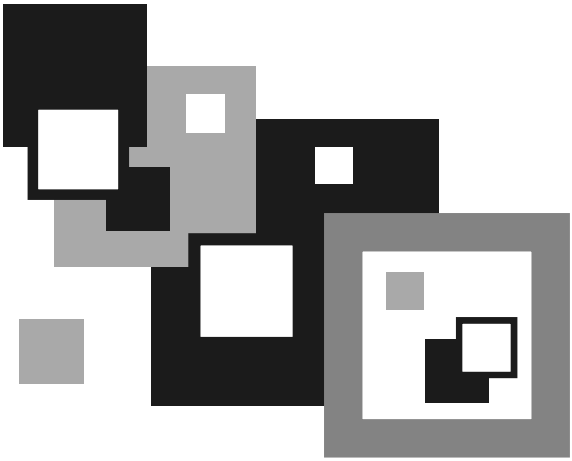
```
  rcSrc.right  := FWidth - 1;
  rcSrc.bottom := FHeight - 1;

  // Устанавливаем цветовой ключ
  ZeroMemory(@OverlayFX, sizeof(TDDOverlayFX));
  OverlayFX.dwSize := sizeof(TDDOverlayFX);
  OverlayFX.dckDestColorkey.dwColorSpaceLowValue  := GetPixelValue;
  OverlayFX.dckDestColorkey.dwColorSpaceHighValue := GetPixelValue;

  // Показываем оверлей
  Result := FDDSOverlay.UpdateOverlay(@rcSrc, FDDSPrimary, @rcDst,
    DDOVER_SHOW or DDOVER_DDFX or DDOVER_KEYDESTOVERRIDE, @OverlayFX);
end;

END.
```

Итоги

Данная часть книги была посвящена изучению графической подсистемы DirectX Graphics. Первая половина части содержала описание подсистемы Direct3D, а во второй части мы познакомились с подсистемой DirectDraw.

Мы разобрали, каким способом при помощи подсистемы Direct3D можно работать с двумерной и трехмерной графикой. Изучили работу с матрицами и выяснили, какие типы матриц присутствуют в Direct3D и для чего каждая из них предназначена. Научились строить сложные объекты из примитивов, изучили работу с материалом и освещением, научились накладывать на объекты текстуру и использовать такие эффекты, как туман. Разобрались, как выводить на сцену одновременно несколько объектов; разобрали работу с текстом на плоскости и в пространстве. Научились использовать такие возможности Direct3D, как мультитекстурирование и прозрачность. Изучили работу вершинных и пиксельных шейдеров и основы языка HLSL.

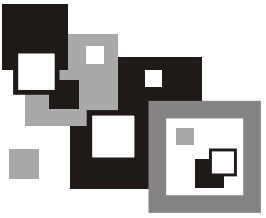
Так же мы кратко изучили возможности подсистемы DirectDraw. Разобрали, какие типы поверхностей существуют в DirectDraw. Научились работать в полноэкранном и оконном режимах, научились рисовать на поверхности, использовать палитру и цветовые ключи. На примере создания эффекта огня показали, как можно получить непосредственный доступ к точкам поверхности. И в конце изучили возможность работы с оверлеями DirectX.



ЧАСТЬ III

DIRECTSOUND

Глава 7



Общие сведения

Компонент DirectSound воспроизводит только файлы формата WAV (Waveform Audio File Format, аудиоформат оцифрованной записи звука компании Microsoft, являющийся стандартом звуковых файлов для персональных компьютеров).

Область применения

Основной областью применения являются игры. Модель позволяет с легкостью оперировать множеством источников звука и их характеристиками. Также одной из областей применения являются различные программы для синтеза музыки и звуков.

Достоинства

Благодаря практически прямому доступу к аппаратной части звуковой платы этот компонент позволяет нам размещать определенные данные непосредственно в ее памяти, оперировать звучанием (объемный звук, микширование и т. д.).

Недостатки

К недостаткам следует отнести тот факт, что DirectSound по большей части ориентирован на существующие модели звуковых ускорителей, и, в последствии, есть вероятность несовместимости с новыми моделями. Слабая под-

держка записи звука в ряде случаев заставляет разработчика прибегать к другим интерфейсам работы со звуком (например, интерфейс MME). Также недостатком можно считать то, что он поддерживает всего один формат аудиоданных.

Принцип работы

Вся работа с подсистемой строится на использовании буферов — некой зарезервированной области памяти, хранящей образец воспроизводимого звука. Существует один первичный буфер (используемый по большей части в служебных целях) и ряд вторичных буферов, которыми собственно и описываются источники звука. Буферы делятся на аппаратные (hardware) и программные (software). К аппаратным буферам можно отнести буферы, располагающиеся в памяти адаптера, а программные — в основной памяти, к которой адаптер не имеет прямого доступа. Данные из вторичных буферов обычно поступают в первичный буфер и уже оттуда передаются в адаптер.

Еще одно отличие первичного буфера от вторичного — это размер. Для первичного буфера он фиксирован и выбирается драйвером DirectSound, а вторичные буферы могут иметь произвольный размер.

Вторичный буфер может быть статическим (static) либо потоковым (streaming). В статических буферах размещаются постоянные звуки, которые обычно имеют небольшой размер, а потоковые буферы представляют себой некое окно, через которое приложение по частям прогоняет звуковой поток. Подсистема DirectSound по возможности старается размещать в памяти адаптера только статические буферы. Также приложение само может управлять размещением буферов.

Уровни взаимодействия

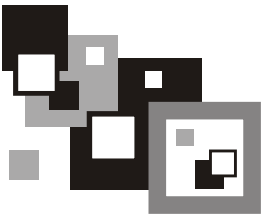
В DirectSound существует 4 уровня взаимодействия приложений между собой и звуковым адаптером. Уровень взаимодействия определяет приоритет при доступе к звуковому адаптеру.

Обычный (normal) уровень взаимодействия определяет универсальные характеристики первичного буфера: частота дискретизации 22 050 Гц и разрядность 8 битов. Это сказывается на качестве звучания, но, с другой стороны, позволяет приложениям работать с адаптером без конфликтов. Далее следуют приоритетный (priority) и эксклюзивный (exclusive) уровни взаимодействия. Приоритетный уровень позволяет задавать формат первичному

буферу и управлять аппаратными ресурсами адаптера. Но это происходит только в тот момент времени, когда окно приложения является активным. При переключении между окнами DirectSound устанавливает новые параметры первичного буфера. В отличие от приоритетного уровня, эксклюзивный уровень предоставляет исключительный доступ приложения к адаптеру, из-за чего заглушается звучание остальных приложений (для DirectX версии 8.0 и последующих аналогичен уровню взаимодействия priority). Последний уровень — уровень доступа к первичному буферу (write — primary) — позволяет приложению напрямую записывать данные в первичный буфер. На этом уровне запрещена работа с вторичными буферами. Уровень доступен только для устройств, имеющих специализированный драйвер DirectSound.

Благодаря возможности смешивания сигналов в DirectSound понижается вероятность конфликтов за доступ к адаптеру между приложениями — фоновые приложения заглушаются вместо полного отказа от доступа к интерфейсу воспроизведения, когда адаптер уже занят.

Глава 8



Вывод звука

Интерфейсы

Для работы со звуком необходимо создать объект устройства с интерфейсом `IDirectSound8`. Данный интерфейс пришел на смену интерфейсу `IDirectSound` и содержит всего один метод:

- `VerifyCertification` — проверяет на сертификацию драйвер устройства.

Остальные методы, унаследованные от интерфейса `IDirectSound`, не считая методов, унаследованных от интерфейса `IUnknown`, следующие:

- `CreateSoundBuffer` — создание буфера для управления звуковым потоком;
- `GetCaps` — получение информации о возможностях устройства;
- `DuplicateSoundBuffer` — создает новый вторичный буфер, исходя из существующего, использующий ту же область памяти, что и оригинальный буфер;
- `SetCooperativeLevel` — установка уровня взаимодействия;
- `Compact` — уплотнение внутренней памяти звукового адаптера (не используется);
- `GetSpeakerConfig` — получение информации об аудиосистеме;
- `SetSpeakerConfig` — установка параметров аудиосистемы;
- `Initialize` — инициализация объекта устройства, созданного при помощи функции `CoCreateInstance`.

Рассмотрим некоторые из этих методов более детально.

Создание буфера:

```
function CreateSoundBuffer(
  const pcDSBufferDesc: TDSBufferDesc;
  out ppDSBuffer: IDirectSoundBuffer;
```

```
  pUnkOuter: IUnknown):
HResult; stdcall;
```

Здесь:

- `pcDSBufferDesc` — адрес структуры `TDSBufferDesc`, описывающей буфер;
- `ppDSBuffer` — адрес переменной, которая получает интерфейс `IDirectSoundBuffer` созданного объекта буфера;
- `pUnkOuter` — указатель на объект с интерфейсом `IUnknown` — должен быть нулевым.

Структура `TDSBufferDesc` имеет следующий вид:

- `dwSize` — размер структуры в байтах;
- `dwFlags` — флаги, определяющие режимы работы буфера:
  - `DSBCAPS_CTRL3D` — поддержка объемного звучания. Нельзя комбинировать с флагом `DSBCAPS_CTRLPAN` и нельзя использовать для стереоформата аудиоданных;
  - `DSBCAPS_CTRLFREQUENCY` — возможность управления частотой дискретизации;
  - `DSBCAPS_CTRLFX` — поддержка обработки звуковых эффектов;
  - `DSBCAPS_CTRLPAN` — возможность управления положением на панораме. Нельзя комбинировать с флагом `DSBCAPS_CTRL3D`;
  - `DSBCAPS_CTRLPOSITIONNOTIFY` — возможность получать уведомления по достижении определенных позиций в буфере;
  - `DSBCAPS_CTRLVOLUME` — возможность управления громкостью звучания;
  - `DSBCAPS_GETCURRENTPOSITION2` — используется для более точного определения позиции воспроизведения при вызове метода `IDirectSoundBuffer8.GetCurrentPosition`. Без использования данного флага позиция воспроизведения будет немного опережать действительную;
  - `DSBCAPS_GLOBALFOCUS` — буфер становится глобальным и его воспроизведение не прекращается, даже если приложение из активного переходит в фоновый режим за одним исключением — если фокус получает приложение, использующее подсистему DirectSound, и имеет уровень взаимодействия write — primary;
  - `DSBCAPS_LOCDEFER` — буфер будет размещен в аппаратной памяти или в основной только в момент активации;
  - `DSBCAPS_LOCHARDWARE` — размещение буфера в памяти адаптера;
  - `DSBCAPS_LOCSOFTWARE` — размещение буфера в основной памяти;

- DSBCAPS_MUTE3DATMAXDISTANCE — данный флаг относится только к буферам, поддерживающим объемное звучание, и позволяет выключать звучание источника при достижении максимального расстояния от слушателя, дабы не расходовать впустую время процессора. Как только расстояние сокращается, звучание будет автоматически восстановлено;
- DSBCAPS_PRIMARYBUFFER — создание первичного буфера;
- DSBCAPS_STATIC — статический буфер;
- DSBCAPS_STICKYFOCUS — звучание буфера не будет остановлено даже при переключении на приложения, не использующие DirectSound;

- dwBufferBytes — размер буфера в байтах. Если мы создаем первичный буфер (флаг DSBCAPS_PRIMARYBUFFER), то размер должен быть нулевым. Для вторичного буфера это значение ограничено константами DSBSIZE_MIN и DSBSIZE_MAX;
- dwReserved — зарезервированное поле, должно быть нулевым;
- lpwfxFormat — адрес структуры TWaveFormatEx, описывающей формат данных буфера. Для первичного буфера имеет нулевое значение. Формат первичного буфера устанавливается методом IDirectSoundBuffer8.SetFormat;
- guid3DAlgorithm — уникальный идентификатор алгоритма моделирования объемного звука системой из двух источников. Относится только к пространственным буферам, которые используются в DirectSound3D, в остальных случаях значение поля игнорируется.

Формат данных буфера описывается структурой TWaveFormatEx:

- wFormatTag — формат звуковых данных (в нашем случае WAVE_FORMAT_PCM);
- nChannels — число каналов (определяет режим моно или стерео);
- nSamplesPerSec — частота дискретизации;
- nAvgBytesPerSec — число байтов в секундном интервале (для стереосигнала);
- nBlockAlign — размер блока данных;
- wBitsPerSample — битовое разрешение выборки;
- cbSize — размер в байтах (не используется).

Получение информации о возможностях устройства:

```
function GetCaps(
  out pDSCaps: TDSCaps):
HResult; stdcall;
```

Здесь:

- pDSCaps — адрес структуры TDSCaps, описывающей возможности устройства.

Структура TDSCaps состоит из следующих полей:

- dwSize — размер структуры в байтах;
- dwFlags:
  - DSCAPS_CERTIFIED — драйвер был протестирован и сертифицирован компанией Microsoft;
  - DSCAPS_CONTINUOUSRATE — устройство поддерживает все частоты дискретизации, лежащие в диапазоне от dwMinSecondarySampleRate до dwMaxSecondarySampleRate с погрешностью 10 Гц;
  - DSCAPS_EMULDRIVER — для устройства не установлен драйвер DirectSound. Работа подсистемы будет эмулироваться, что скажется на производительности;
  - DSCAPS_PRIMARY16BIT — поддержка 16-разрядных форматов первичным буфером;
  - DSCAPS_PRIMARY8BIT — поддержка 8-разрядных форматов первичным буфером;
  - DSCAPS_PRIMARYMONO — поддержка монофонических форматов первичным буфером;
  - DSCAPS_PRIMARYSTEREO — поддержка стереофонических форматов первичным буфером;
  - DSCAPS_SECONDARY16BIT — поддержка 16-разрядных форматов вторичным буфером;
  - DSCAPS_SECONDARY8BIT — поддержка 8-разрядных форматов вторичным буфером;
  - DSCAPS_SECONDARYMONO — поддержка монофонических форматов вторичным буфером;
  - DSCAPS_SECONDARYSTEREO — поддержка стереофонических форматов вторичным буфером;
- dwMinSecondarySampleRate, dwMaxSecondarySampleRate — минимальная и максимальная частота дискретизации;
- dwPrimaryBuffers — число поддерживаемых первичных буферов (всегда равно единице);
- dwMaxHwMixingAllBuffers — число буферов, которые могут быть смешаны аппаратно;

- dwMaxHwMixingStaticBuffers — максимальное число статических буферов;
- dwMaxHwMixingStreamingBuffers — максимальное число потоковых буферов;
- dwFreeHwMixingAllBuffers — количество всех свободных буферов с аппаратным смешиванием;
- dwFreeHwMixingStaticBuffers — количество статических свободных буферов с аппаратным смешиванием;
- dwFreeHwMixingStreamingBuffers — количество потоковых свободных буферов с аппаратным смешиванием;
- dwMaxHw3DallBuffers — максимальное количество свободных буферов с аппаратным смешиванием для источников объемного звука;
- dwMaxHw3DStaticBuffers — максимальное количество свободных буферов с аппаратным смешиванием для статических источников объемного звука;
- dwMaxHw3DStreamingBuffers — максимальное количество свободных буферов с аппаратным смешиванием для потоковых источников объемного звука;
- dwFreeHw3DallBuffers — количество свободных буферов с аппаратным смешиванием для всех источников объемного звука;
- dwFreeHw3DstaticBuffers — количество свободных буферов с аппаратным смешиванием для статических источников объемного звука;
- dwFreeHw3DStreamingBuffers — количество свободных буферов с аппаратным смешиванием для потоковых источников объемного звука;
- dwTotalHwMemBytes — объем в байтах памяти звукового адаптера для размещения статических буферов;
- dwFreeHwMemBytes — свободный объем в байтах памяти звукового адаптера;
- dwMaxContigFreeHwMemBytes — размер в байтах самого большого свободного непрерывного блока памяти в адаптере;
- dwUnlockTransferRateHwBuffers — скорость (Кбайт/с) пересылки данных в память адаптера;
- dwPlayCpuOverheadSwBuffers — приблизительная оценка времени центрального процессора (в %), необходимого для смешивания буферов, располагающихся в основной памяти;
- dwReserved1, dwReserved2 — зарезервированные поля.

Создание копии вторичного буфера:

```
function DuplicateSoundBuffer(
  pDSBufferOriginal: IDirectSoundBuffer;
  out ppDSBufferDuplicate: IDirectSoundBuffer):
HResult; stdcall;
```

Здесь:

- `pDSBufferOriginal` — ссылка на копируемый буфер;
- `ppDSBufferDuplicate` — ссылка на переменную, куда будет занесен указатель нового объекта.

Установка уровня взаимодействия:

```
function SetCooperativeLevel(
  hwnd: HWND;
  dwLevel: DWORD):
HResult; stdcall;
```

Здесь:

- `hwnd` — указатель на форму приложения, для которой задается уровень взаимодействия;
- `dwLevel` — устанавливаемый уровень взаимодействия:
  - `DSSCL_NORMAL` — нормальный уровень;
  - `DSSCL_PRIORITY` — приоритетный уровень;
  - `DSSCL_EXCLUSIVE` — эксклюзивный уровень;
  - `DSSCL_WRITEPRIMARY` — уровень доступа к первичному буферу.

Создание объекта производится вызовом функции `DirectSoundCreate8`:

```
function DirectSoundCreate8(
  pcGuidDevice: PGUID;
  out ppDS8: IDirectSound8;
  pUnkOuter: IUnknown):
HResult; stdcall; external DirectSoundDLL;
```

Здесь:

- `pcGuidDevice` — адрес уникального идентификатора устройства или нулевое значение для выбора стандартного;
- `ppDS8` — адрес указателя, в который будет занесен интерфейс создаваемого объекта;
- `pUnkOuter` — указатель на объект с `IUnknown`, должен быть нулевым.

Данный объект позволяет манипулировать характеристиками звукового адаптера. Для создания данного объекта необходимо передать идентификатор нужного нам устройства, полученный посредством вызова функции `DirectSoundEnumerate`, либо будет создано устройство по умолчанию:

```
function DirectSoundEnumerate(
  lpDSEnumCallback: TDSEnumCallback;
  lpContext: Pointer):
```

```
HResult; stdcall;
external DirectSoundDLL name 'DirectSoundEnumerateA';
```

Здесь:

- `lpDSEnumCallback` — адрес функции перебора, которая будет вызываться для кажого обнаруженного устройства системы;
- `lpContext` — адрес произвольного значения, которое будет передаваться функции перебора.

Следующий шаг перед началом работы — установка уровня взаимодействия `IDirectSound8.SetCooperativeLevel`.

После установки уровня взаимодействия можно приступать к созданию буферов. Буфер — это объект с интерфейсом `IDirectSoundBuffer8`. Данный интерфейс содержит следующие методы (в том числе унаследованные от интерфейса `IDirectSoundBuffer`):

- `SetFX` — включает эффекты в буфере;
- `AcquireResources` — распределяет ресурсы для буфера, который был создан с флагом `DSBCAPS_LOCDEFER`;
- `GetObjectInPath` — получение интерфейса объекта, связанного с буфером;
- `GetCaps` — получение информации о возможностях буфера;
- `GetCurrentPosition` — получение текущих позиций указателей в буфере;
- `GetFormat` — получение формата звуковых данных буфера;
- `GetVolume` — получение громкости;
- `GetPan` — получение положения на панораме;
- `GetFrequency` — получение частоты дискретизации;
- `GetStatus` — получение состояния буфера;
- `Initialize` — инициализация буфера;
- `Lock` — подготовка буфера (или его части) для записи данных и получение указателей и размеров областей буфера, в которые данные могут быть записаны;
- `Play` — воспроизведение буфера с текущей позиции;
- `SetCurrentPosition` — установка текущей позиции указателей в буфере;
- `SetFormat` — установка формата звуковых данных;
- `SetVolume` — установка громкости;
- `SetPan` — установка положения на панораме;
- `SetFrequency` — установка частоты дискретизации;
- `Stop` — остановка воспроизведения буфера;

- `Unlock` — завершение операции обновления буфера;
- `Restore` — восстановление буфера после потери.

Рассмотрим необходимые для работы методы по отдельности.

Получение текущих позиций указателей в буфере:

```
function GetCurrentPosition(
  pdwCurrentPlayCursor,
  pdwCurrentWriteCursor: PDWORD):
HResult; stdcall;
```

Здесь:

- `pdwCurrentPlayCursor` — адрес переменной, в которую будет записано смещение в байтах для позиции воспроизведения;
- `pdwCurrentWriteCursor` — адрес переменной, в которую будет записано смещение в байтах для позиции записи.

Получение формата звуковых данных буфера:

```
function GetFormat(
  pwfxFormat: PWaveFormatEx;
  dwSizeAllocated: DWORD;
  pdwSizeWritten: PDWORD):
HResult; stdcall;
```

Здесь:

- `pwfxFormat` — адрес структуры типа `TWaveFormatEx`, в которую будет записан формат данных буфера;
- `dwSizeAllocated` — размер описателя в байтах;
- `pdwSizeWritten` — адрес переменной, в которую будет записан реальный размер заполненного описателя.

Получение громкости:

```
function GetVolume(
  out plVolume: Longint):
HResult; stdcall;
```

Здесь `plVolume` — адрес переменной, в которую будет записан текущий уровень громкости буфера.

Получение положения на панораме:

```
function GetPan(
  out plPan: Longint):
HResult; stdcall;
```

Здесь `plPan` — адрес переменной, в которую будет записано текущее положение на панораме.

Получение частоты дискретизации:

```
function GetFrequency(
  out pdwFrequency: DWORD):
HResult; stdcall;
```

Здесь `pdwFrequency` — адрес переменной, в которую будет записана текущая частота дискретизации.

Получение состояния буфера:

```
function GetStatus(
  out pdwStatus: DWORD):
HResult; stdcall;
```

Здесь `pdwStatus` — адрес переменной, в которую будет записано текущее состояние буфера.

Подготовка буфера для записи данных:

```
function Lock(
  dwOffset, dwBytes: DWORD;
  ppvAudioPtr1: PPointer;
  pdwAudioBytes1: PDWORD;
  ppvAudioPtr2: PPointer;
  pdwAudioBytes2: PDWORD;
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `dwOffset` — смещение относительно начала буфера;
- `dwBytes` — размер участка буфера для записи в байтах;
- `ppvAudioPtr1`, `ppvAudioPtr2` — адреса указателей на область памяти, в которую будут записаны указатели частей полученного участка буфера. Из-за кольцевой структуры буфера полученный участок может пересекать границу буфера и попадать на его начало. Если `ppvAudioPtr2` возвращает нулевой результат, то это значит, что мы получили непрерывный участок буфера;
- `pdwAudioBytes1`, `pdwAudioBytes2` — адреса переменных, в которые будет записан размер первого и второго участков буфера в байтах;

- `dwFlags` — флаги, влияющие на модификацию буфера:
  - `DSBLOCK_FROMWRITECURSOR` — блокировка начинается с позиции записи. Параметр `dwOffset` в этом случае игнорируется;
  - `DSBLOCK_ENTIREBUFFER` — запрашивается доступ ко всему доступному для записи участку. Параметр `dwBytes` игнорируется.

Воспроизведение буфера с текущей позиции:

```
function Play(
  dwReserved1,
  dwPriority,
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `dwReserved1` — параметр зарезервирован, должен быть нулевым;
- `dwPriority` — приоритет данного источника звука. Используется, если при создании буфера был указан флаг `DSBCAPS_LOCDEFER`. Значение варьируется в диапазоне от `$00000000` до `$FFFFFFFF`;
- `dwFlags` — флаги, определяющие параметры воспроизведения:
  - `DSBPLAY_LOOPING` — циклическое проигрывание буфера.

Установка текущей позиции указателей в буфере:

```
function SetCurrentPosition(
  dwNewPosition: DWORD):
HResult; stdcall;
```

Здесь `dwNewPosition` — смещение в байтах от начала буфера.

Установка формата звуковых данных:

```
function SetFormat(
  pcfxFormat: PWaveFormatEx):
HResult; stdcall;
```

Здесь `pcfxFormat` — формат звуковых данных, имеющий тип `TWaveFormatEx`.

Установка уровня громкости:

```
function SetVolume(
  lVolume: Longint):
HResult; stdcall;
```

Здесь `lVolume` — новое значение уровня громкости в сотых долях децибела. Значение находится в диапазоне от `DSBVOLUME_MAX` до `DSBVOLUME_MIN`.

Установка положения на панораме:

```
function SetPan(
  lPan: Longint):
HResult; stdcall;
```

Здесь `lPan` — новое положение на панораме — относительная величина, показывающая соотношение звучания между левым и правым каналами. Значение лежит в диапазоне от `DSBPAN_LEFT` до `DSBPAN_RIGHT`.

Установка частоты дискретизации:

```
function SetFrequency(
  dwFrequency: DWORD):
HResult; stdcall;
```

Здесь `dwFrequency` — новое значение частоты (в герцах). Величина оказывает влияние на скорость воспроизведения и высоту звука. Данный метод применяется только для вторичных буферов. В случае указания величины `DSBFREQUENCY_ORIGINAL` будет установлена частота дискретизации по умолчанию для данного буфера.

Остановка воспроизведения буфера:

```
function Stop: HResult; stdcall;
```

Завершение операции обновления буфера:

```
function Unlock(
  pvAudioPtr1: Pointer;
  dwAudioBytes1: DWORD;
  pvAudioPtr2: Pointer;
  dwAudioBytes2: DWORD):
HResult; stdcall;
```

Здесь:

- `pvAudioPtr1`, `pvAudioPtr2` — указатели на области памяти, полученные при вызове метода `IDirectSoundBuffer8.Lock`;
- `dwAudioBytes1`, `dwAudioBytes2` — количество байт, которые были записаны в участки памяти на самом деле.

Восстановление буфера после потери:

```
function Restore: HResult; stdcall;
```

Если при создании буфера был указан флаг `DSBCAPS_LOCDEFER`, то существует возможность указания места расположения буфера.

Создание буферов

И первичный, и вторичный буферы создаются посредством вызова `IDirectSound8.CreateSoundBuffer`. В результате вызова данного метода мы получаем объект с интерфейсом `IDirectSoundBuffer` посредством вызова метода `IDirectSoundBuffer.QueryInterface`, для которого мы можем запросить интерфейс `IDirectSoundBuffer8`. Для приложения, работающего на обычном уровне взаимодействия, первичный буфер не нужен.

Создав буфер, мы должны заполнить его данными. Это делается последствием вызова функции `IDirectSoundBuffer8.Lock`, возвращающей указатели на доступные заблокированные участки буфера. Заполнив буфер нужными данными, мы вызываем метод `IDirectSoundBuffer8.Unlock`.

Для воспроизведения буфера имеется метод `IDirectSoundBuffer8.Play`. По умолчанию буфер воспроизводится всего один раз при вызове данного метода, но если в качестве последнего параметра указать флаг `DSBPLAY_LOOPING`, то воспроизведение буфера будет циклическим. Остановить воспроизведение можно посредством вызова метода `IDirectSoundBuffer8.Stop`.

Существует возможность изменения параметров звучания, таких как частота дискретизации (`IDirectSoundBuffer8.SetFrequency`), положение на панораме (`IDirectSoundBuffer8.SetPan`) и громкость (`IDirectSoundBuffer8.SetVolume`).

Потеря буферов

Сущетствует потенциальная опасность потери данных буферов. Такая ситуация может возникнуть, например, в следующих случаях: если буферы размещены в памяти звукового адаптера, а мы удаляем его из системы, или когда программа с уровнем доступа к первичному буферу (write — primary) получает управление. Во втором случае подсистема DirectSound сделает так, что буферы приложений, имеющих более низкий уровень взаимодействия, прекратят свое звучание и станут помеченными как потерянные. Это происходит из-за того, что DirectSound не может правильно отслеживать текущие положения во вторичных буферах, не имея доступа к первичному буферу. Если происходит обратное — приложение с более высоким уровнем взаимодействия становится неактивным, то и его первичный буфер будет помечен как потерянный. Также буфер может быть потерян и при нехватке системных ресурсов.

Узнать о потере буфера можно при попытке обращения к буферу. В результате мы получим код ошибки `DSERR_BUFFERLOST`. Все потерянные буферы долж-

ны быть восстановлены, когда приложение снова становится активным. Для восстановления буфера используется метод `IDirectSoundBuffer8.Restore`.

А теперь рассмотрим работу с DirectSound подробнее на конкретном примере.

Звуковые эффекты

Подсистема DirectX обеспечивает поддержку различных звуковых эффектов. Причем стандартные звуковые эффекты доступны для всех DirectX-приложений, а остальные эффекты должны быть зарегистрированы в системе как DMO (DirectX Media Object).

Для использования звуковых эффектов при создании буфера нужно обязательно указать флаг `DSBCAPS_CTRLFX`. Установка звуковых эффектов производится вызовом метода `IDirectSoundBuffer8.SetFX`:

```
function SetFX(
  dwEffectsCount: DWORD;
  pDSFXDesc: PDSEffectDesc;
  pdwResultCodes: PDWORD):
HResult; stdcall;
```

Здесь:

- `dwEffectsCount` — число эффектов;
- `pDSFXDesc` — адрес массива (численностью `dwEffectsCount`) элементов `TDSEffectDesc`, описывающих эффекты;
- `pdwResultCodes` — адрес массива (численностью `dwEffectsCount`) элементов типа `DWORD`, в которые будет занесен результат выполнения функции.

Структура `TDSEffectDesc`, описывающая звуковой эффект, имеет следующий вид:

- `dwSize` — размер структуры в байтах;
- `dwFlags` — флаги, определяющие режимы размещения эффекта в буфере:
  - `0` — производится попытка расположить эффект в аппаратной части, если это возможно. Если эффект не поддерживается аппаратными средствами (а начиная с DirectX 9.0 так оно и есть), то используется основная память;
  - `DSFX_LOCHARDWARE` — эффект должен располагаться в аппаратной части. Если эффект отсутствует в аппаратной части, то возвращается

ошибка. Поскольку DirectX 9.0 не поддерживает аппаратное ускорение для звуковых эффектов, этот флаг не используется;

- DSFX_LOCSOFTWARE — эффект должен располагаться в программной части, даже если аппаратная часть поддерживает класс эффекта guidDSFXClass. Если эффект отсутствует в программной части, то возвращается ошибка. В DirectX 9.0 все эффекты располагаются в основной памяти независимо от того, установлен этот флаг или нет;

- guidDSFXClass — уникальный идентификатор класса звукового эффекта:
  - GUID_DSFX_STANDARD_CHORUS — хор;
  - GUID_DSFX_STANDARD_COMPRESSOR — компрессор;
  - GUID_DSFX_STANDARD_DISTORTION — искажение;
  - GUID_DSFX_STANDARD_ECHO — эхо;
  - GUID_DSFX_STANDARD_FLANGER — фланец;
  - GUID_DSFX_STANDARD_GARGLE — эффект полоскания горла;
  - GUID_DSFX_STANDARD_I3DL2REVERB — интерактивная реверберация;
  - GUID_DSFX_STANDARD_PARAMEQ — параметрический эквалайзер;
  - GUID_DSFX_WAVES_REVERB — волновая реверберация;
- dwReserved1, dwReserved2 — зарезервированные параметры — должны быть нулевыми.

После установки звуковых эффектов мы можем получить интерфейсы объектов каждого из соответствующих звуковых эффектов для более детального контроля эффекта:

```
function GetObjectInPath(
  const rguidObject: TGUID;
  dwIndex: DWORD;
  const rguidInterface: TGUID;
  out ppObject{IUnknown}):
HResult; stdcall;
```

Здесь:

- rguidObject — уникальный идентификатор класса объекта для поиска (пример — GUID_DSFX_STANDARD_ECHO). Для поиска объекта среди всех классов необходимо использовать идентификатор GUID_All_Objects;
- dwIndex — индекс объекта;
- rguidInterface — уникальный идентификатор нужного интерфейса;
- ppObject — адрес переменной, в которую будет записан найденный указатель на интерфейс.

В результате вызова данного метода могут быть получены указатели на следующие интерфейсы:

- `IDirectSoundFXChorus8`;
- `IDirectSoundFXCompressor8`;
- `IDirectSoundFXDistortion8`;
- `IDirectSoundFXEcho8`;
- `IDirectSoundFXFlanger8`;
- `IDirectSoundFXGargle8`;
- `IDirectSoundFXI3DL2Reverb8`;
- `IDirectSoundFXParamEq8`;
- `IDirectSoundFXWavesReverb8`.

Любой из данных интерфейсов содержит следующие методы:

- `SetAllParameters` — установить параметры звукового эффекта для буфера;
- `GetAllParameters` — получить параметры звукового эффекта для буфера.

Для каждого звукового эффекта определена соответствующая структура:

- `TDSFXChorus`;
- `TDSFXCompressor`;
- `TDSFXDistortion`;
- `TDSFXEcho`;
- `TDSFXFlanger`;
- `TDSFXGargle`;
- `TDSFXI3DL2Reverb`;
- `TDSFXParamEq`;
- `TDSFXWavesReverb`.

Для сброса всех звуковых эффектов необходимо вызвать метод `IDirectSoundBuffer8.SetFX` с нулевыми параметрами.

Классы *TdxSound* и *TdxSoundManager*

Для более удобной работы с подсистемой DirectSound автором разработаны классы `TdxSound` и `TdxSoundManager` и написан пример их использования. Уровень взаимодействия (Cooperative Level) — нормальный, соответственно, отсутствует первичный буфер. Класс `TdxSoundManager` позволяет опериро-

вать неограниченным множеством вторичных буферов (точнее максимально допустимым количеством, ограниченным подстстемой DirectSound), представленных классом `TdxSound`, который, в свою очередь, позволяет воспроизводить вторичный буфер (в том числе циклически), останавливать воспроизведение, устанавливать частоту дискретизации, положение на панораме, громкость звучания и устанавливать различные звуковые эффекты. Классы расположены в модуле UdxSoundManager.pas каталога Classes на прилагаемом к книге компакт-диске.

Сначала рассмотрим общую структуру классов:

```
TdxSound = class
PRIVATE
  FdsBuffer8:   IDirectSoundBuffer8;
  FData:        PByte;
  FWAVFileName: string;
  FDataSize:    integer;

  FdsChorus8:     IDirectSoundFXChorus8;
  FdsCompressor8: IDirectSoundFXCompressor8;
  FdsDistortion8: IDirectSoundFXDistortion8;
  FdsEcho8:       IDirectSoundFXEcho8;
  FdsFlanger8:    IDirectSoundFXFlanger8;
  FdsGargle8:     IDirectSoundFXGargle8;
  FdsParamEq8:    IDirectSoundFXParamEq8;
  FdsReverb8:     IDirectSoundFXWavesReverb8;

  FParamsChorus:     TDSFXChorus;
  FParamsCompressor: TDSFXCompressor;
  FParamsDistortion: TDSFXDistortion;
  FParamsEcho:       TDSFXEcho;
  FParamsFlanger:    TDSFXFlanger;
  FParamsGargle:     TDSFXGargle;
  FParamsParamEq:    TDSFXParamEq;
  FParamsReverb:     TDSFXWavesReverb;

  dwResults:    array[0..SFX_COUNT - 1] of DWORD;
  dsEffect:     array[0..SFX_COUNT - 1] of TDSEffectDesc;
```

```
  function LoadWAVFile(wfx: PWaveFormatEx; dwSize: PDWORD): PByte;
  function FillBuffer: HResult;

  function RestoreBuffer(WasRestored: PBOOL): HResult;

  function GetEffectGuid(Effect: WORD): TGUID;
PUBLIC
  constructor Create(DirectSound8: IDirectSound8; WAVFileName:
    string);
  destructor Destroy; override;

  function PlaySound(Looping: boolean = false): HResult;
  function StopSound(): HResult;

  function SetFrequency(Frequency: Cardinal): HResult;
  function GetFrequency(var Frequency: Cardinal): HResult;

  function SetPan(Pan: integer): HResult;
  function GetPan(var Pan: integer): HResult;

  function SetVolume(Volume: integer): HResult;
  function GetVolume(var Volume: integer): HResult;

  function SetEffects(EffectsMask: DWORD): HResult;
  function GetEffects(var EffectsMask: DWORD): HResult;

  function SetEffectParams(Effect: DWORD; const Params: pointer):
    HResult;
  function GetEffectParams(Effect: DWORD; Params: pointer):
    HResult;
END;

TdxSoundManager = class
PRIVATE
  FDirectSound: IDirectSound8;
```

```
  FHandle: THandle;
  FSounds: TList;

PUBLIC
  constructor Create(AHandle: THandle);
  destructor Destroy; override;

  function Initialize: HResult;

  function SoundCount: integer;

  function CreateSound(WAVFileName: string): HResult;
  function DeleteSound(Index: integer): HResult;

  function GetSound(Index: integer): TdxSound;
END;
```

Каждый класс несет свою смысловую нагрузку: класс `TdxSound` инкапсулирует в себе механизмы работы с вторичным буфером, а класс `TdxSoundManager` является коллекцией вторичных буферов.

Класс `TdxSound` предоставляет следующий набор методов:

- конструктор `Create` и деструктор `Destroy`;
- `PlaySound`, `StopSound` — воспроизведение и остановка воспроизведения буфера;
- `GetFrequency`, `SetFrequency` — получение и установка частоты дескритезации;
- `GetPan`, `SetPan` — получение и установка положения на панораме;
- `GetVolume`, `SetVolume` — получение и установка громкости звучания;
- `GetEffects`, `SetEffects` — получение и установка звуковых эффектов;
- `GetEffectParams`, `SetEffectParams` — получение и установка параметров звукового эффекта.

Работа с WAV-файлами производится при помощи модуля MMSystem и приведенного ниже набора функций вида `mmio`*XXX*:

- `mmioOpen` — открыть файл;
- `mmioClose` — закрыть файл;
- `mmioDescend` — открыть вложенный блок;

- `mmioAscend` — закрыть вложенный блок;
- `mmioRead` — прочитать данные из открытого блока;
- `mmioStringToFOURCC` — преобразование строки в 4-буквенный идентификатор.

Класс `TdxSoundManager` включает в себя следующие методы:

- конструктор `Create` и деструктор `Destroy`;
- `Initialize` — инициализация подсистемы DirectSound;
- `SoundCount` — число источников звука;
- `CreateSound` — создание источника звука;
- `DeleteSound` — удаление источника звука;
- `GetSound` — получение источника звука по его индексу.

Полный исходный код класса представлен в листинге 8.1.

Листинг 8.1. Текст модуля UdxSoundManager.pas

```
UNIT UdxSoundManager;
{*********************************************************************
{**  Работа с файлами формата WAV - загрузка и воспроизведение   **}
{**  Автор: Есенин Сергей Анатольевич                            **}
{*********************************************************************}

{**} INTERFACE {*************************************************}

{**} USES {******************************************************}
  Windows, Classes, SysUtils, MMSystem, DirectSound, Dialogs,
  ComObj, Math, ActiveX;

{**} CONST {*****************************************************}
  SFX_COUNT = 8;

  SFX_STANDARD_CHORUS           = $000000001;
  SFX_STANDARD_COMPRESSOR       = $000000002;
  SFX_STANDARD_DISTORTION       = $000000004;
  SFX_STANDARD_ECHO             = $000000008;
  SFX_STANDARD_FLANGER          = $000000010;
```

```
  SFX_STANDARD_GARGLE           = $000000020;
  SFX_STANDARD_PARAMEQ          = $000000040;
  SFX_WAVES_REVERB              = $000000080;

{**} TYPE {*****************************************************}
  TdxSound = class
  PRIVATE
    FdsBuffer8:   IDirectSoundBuffer8;
    FData:        PByte;
    FWAVFileName: string;
    FDataSize:    integer;

    FdsChorus8:     IDirectSoundFXChorus8;
    FdsCompressor8: IDirectSoundFXCompressor8;
    FdsDistortion8: IDirectSoundFXDistortion8;
    FdsEcho8:       IDirectSoundFXEcho8;
    FdsFlanger8:    IDirectSoundFXFlanger8;
    FdsGargle8:     IDirectSoundFXGargle8;
    FdsParamEq8:    IDirectSoundFXParamEq8;
    FdsReverb8:     IDirectSoundFXWavesReverb8;

    FParamsChorus:     TDSFXChorus;
    FParamsCompressor: TDSFXCompressor;
    FParamsDistortion: TDSFXDistortion;
    FParamsEcho:       TDSFXEcho;
    FParamsFlanger:    TDSFXFlanger;
    FParamsGargle:     TDSFXGargle;
    FParamsParamEq:    TDSFXParamEq;
    FParamsReverb:     TDSFXWavesReverb;

    dwResults:    array[0..SFX_COUNT - 1] of DWORD;
    dsEffect:     array[0..SFX_COUNT - 1] of TDSEffectDesc;

    function LoadWAVFile(wfx: PWaveFormatEx; dwSize: PDWORD): PByte;
    function FillBuffer: HResult;
```

```
    function RestoreBuffer(WasRestored: PBOOL): HResult;

    function GetEffectGuid(Effect: WORD): TGUID;

  PUBLIC

    constructor Create(DirectSound8: IDirectSound8; WAVFileName:
      string);
    destructor Destroy; override;

    function PlaySound(Looping: boolean = false): HResult;
    function StopSound(): HResult;
    function SetFrequency(Frequency: Cardinal): HResult;
    function SetPan(Pan: integer): HResult;
    function SetVolume(Volume: integer): HResult;

    function GetFrequency(var Frequency: Cardinal): HResult;
    function GetPan(var Pan: integer): HResult;
    function GetVolume(var Volume: integer): HResult;

    function SetEffects(EffectsMask: DWORD): HResult;
    function GetEffects(var EffectsMask: DWORD): HResult;

    function SetEffectParams(Effect: DWORD; const Params: pointer):
      HResult;
    function GetEffectParams(Effect: DWORD; Params: pointer):
      HResult;
  END;

  TdxSoundManager = class
  PRIVATE
    FDirectSound: IDirectSound8;

    FHandle: THandle;
    FSounds: TList;
```

```
  PUBLIC
    constructor Create(AHandle: THandle);
    destructor Destroy; override;

    function Initialize: HResult;

    function SoundCount: integer;

    function CreateSound(WAVFileName: string): HResult;
    function DeleteSound(Index: integer): HResult;

    function GetSound(Index: integer): TdxSound;
  END;

{**} IMPLEMENTATION

{**} {TdxSoundManager} {****************************************}

{*****************************************************************}
{**  Конструктор класса                                         **}
{*****************************************************************}
constructor TdxSoundManager.Create(AHandle: THandle);
begin
  // Инициализация подсистемы COM
  CoInitializeEx(NIL, COINIT_MULTITHREADED);


  // Запоминаем указатель на главную форму
  FHandle := AHandle;

  // Создаем контейнер вторичных буферов
  FSounds := TList.Create;

  // Обнуляем ссылку на объект DirectSound
  FDirectSound := NIL;
end;
```

```
{*****************************************************************}
{**  Деструктор класса                                          **}
{*****************************************************************}
destructor TdxSoundManager.Destroy;
var
  I: integer;
begin
  if FSounds <> NIL then
  begin
    // Обнуляем все ссылки на вторичные буферы
    for I := 0 to FSounds.Count - 1 do
    begin
      if FSounds[I] <> NIL then
      begin
        TdxSound(FSounds[I]).Free;
        FSounds[I] := NIL;
      end;
    end;

    // Чистим и уничтожаем контейнер вторичных буферов
    FSounds.Clear;
    FreeAndNil(FSounds);
  end;

  // Обнуляем ссылку на объект DirectSound
  FDirectSound := NIL;

  // Завершаем работу с COM
  CoUninitialize;
end;

{*****************************************************************}
{**  Инициализация подсистемы                                   **}
{*****************************************************************}
function TdxSoundManager.Initialize: HResult;
```

```
begin
  // Инициализируем подсистему DirectSound
  result := DirectSoundCreate8(NIL, FDirectSound, NIL);
  if FAILED(result) then EXIT;

  // Устанавливаем уровень взаимодействия
  result := FDirectSound.SetCooperativeLevel(FHandle, DSSCL_NORMAL);
  if FAILED(result) then EXIT;

  result := S_OK;
end;

{*********************************************************************}
{**  Создаем источник звука                                         **}
{*********************************************************************}
function TdxSoundManager.CreateSound(WAVFileName: string): HResult;
var
  Sound: TdxSound;
begin
  result := E_FAIL;

  Sound := TdxSound.Create(FDirectSound, WAVFileName);
  if Sound <> NIL then
  begin
    FSounds.Add(Sound);

    result := S_OK;
  end;
end;

{*********************************************************************}
{**  Удаляем источник звука                                         **}
{*********************************************************************}
function TdxSoundManager.DeleteSound(Index: integer): HResult;
var
  Sound: TdxSound;
```

```
begin
  result := E_FAIL;
  if (Index < 0) or (Index >= FSounds.Count) then EXIT;

  Sound := FSounds.Items[Index];
  if Sound = NIL then EXIT;

  FSounds.Remove(Sound);
  FreeAndNil(Sound);

  result := S_OK;
end;

{*****************************************************************}
{**  Получаем источник звука по индексу                         **}
{*****************************************************************}
function TdxSoundManager.GetSound(Index: integer): TdxSound;
begin
  result := NIL;
  if (Index < 0) or (Index >= FSounds.Count) then EXIT;

  result := FSounds.Items[Index];
end;

{*****************************************************************}
{**  Получаем число источников звука                            **}
{*****************************************************************}
function TdxSoundManager.SoundCount: integer;
begin
  result := FSounds.Count;
end;

{**} {TdxSound} {************************************************}

{*****************************************************************}
{**  Конструктор класса                                          **}
{*****************************************************************}
```

```
constructor TdxSound.Create(DirectSound8: IDirectSound8;
  WAVFileName: string);
var
  FDSTmpBuf:  IDirectSoundBuffer;
  bdsc:       TDSBufferDesc;
  wfx:        TWaveFormatEx;
begin
  FdsChorus8     := NIL;
  FdsCompressor8 := NIL;
  FdsDistortion8 := NIL;
  FdsEcho8       := NIL;
  FdsFlanger8    := NIL;
  FdsGargle8     := NIL;
  FdsParamEq8    := NIL;
  FdsReverb8     := NIL;

  if DirectSound8 = NIL then EXIT;

  // Проверяем, существует ли файл
  if not FileExists(WAVFileName) then EXIT;

  FWAVFileName := WAVFileName;
  FdsBuffer8 := NIL;

  // Загружаем аудиоданные
  FData := LoadWAVFile(@wfx, @FDataSize);
  if FData = NIL then EXIT;

  // Заполняем структуру, описывающую вторичный буфер
  ZeroMemory(@bdsc, sizeof(bdsc));
  bdsc.dwSize         := sizeof(bdsc);
  bdsc.dwFlags        := DSBCAPS_CTRLFX or
                         DSBCAPS_CTRLPAN or
                         DSBCAPS_CTRLVOLUME or
                         DSBCAPS_CTRLFREQUENCY or
                         DSBCAPS_GLOBALFOCUS;
```

```
  bdsc.dwBufferBytes  := Max(FDataSize,
                             wfx.nSamplesPerSec *
                             DSBSIZE_FX_MIN div 1000 * 2);
  bdsc.lpwfxFormat    := @wfx;
  // Создаем временный буфер
  if FAILED(DirectSound8.CreateSoundBuffer(bdsc, FDSTmpBuf, NIL))
     then EXIT;

  // Получаем интерфейс IDirectSoundBuffer8 для вторичного буфера
  if FAILED(FDSTmpBuf.QueryInterface(IID_IDirectSoundBuffer8,
     FdsBuffer8)) then EXIT;

  // Обнуляем временный буфер
  FDSTmpBuf := NIL;

  // Заполняем буфер данными
  FillBuffer;
end;

{*****************************************************************}
{**  Деструктор класса                                          **}
{*****************************************************************}
destructor TdxSound.Destroy;
begin
  FdsChorus8     := NIL;
  FdsCompressor8 := NIL;
  FdsDistortion8 := NIL;
  FdsEcho8       := NIL;
  FdsFlanger8    := NIL;
  FdsGargle8     := NIL;
  FdsParamEq8    := NIL;
  FdsReverb8     := NIL;

  // Освобождаем память, выделенную под аудиоданные
  if Fdata <> NIL then FreeMemory(FData);
```

```
  FdsBuffer8 := NIL;
end;

{*****************************************************************}
{** Получаем частоту дискретизации                              **}
{*****************************************************************}
function TdxSound.GetFrequency(var Frequency: Cardinal): HResult;
begin
  result := FdsBuffer8.GetFrequency(Frequency);
end;

{*****************************************************************}
{** Получаем положение на панораме                              **}
{*****************************************************************}
function TdxSound.GetPan(var Pan: integer): HResult;
begin
  result := FdsBuffer8.GetPan(Pan);
end;

{*****************************************************************}
{** Получаем громкость звучания                                 **}
{*****************************************************************}
function TdxSound.GetVolume(var Volume: integer): HResult;
begin
  result := FdsBuffer8.GetVolume(Volume);
end;

{*****************************************************************}
{** Загрузка аудиоданных из WAV-файла                           **}
{*****************************************************************}
function TdxSound.LoadWAVFile(wfx: PWaveFormatEx; dwSize: PDWORD): PByte;
var
  hwav: THandle;
  Child, Parent: TMMCKInfo;
begin
  result := NIL;
```

```
// Открываем WAV-файл
hwav := mmioOpen(PChar(FWAVFileName), NIL, MMIO_READ or
          MMIO_ALLOCBUF);
try
  try
    if hwav = 0 then ABORT;

    ZeroMemory(@parent, sizeof(parent));
    parent.fccType :=  mmioStringToFOURCC('wave', MMIO_TOUPPER);
    Child := Parent;

    // Ищем блок 'RIFF'
    if (mmioDescend(hwav, @parent, NIL, MMIO_FINDRIFF) <> 0) then
        ABORT;



    // Рассматриваем блок 'fmt '
    child.ckid := mmioStringToFOURCC('fmt ', 0);
    if (mmioDescend(hwav, @child, @parent, 0) <> 0) then ABORT;

    // Получаем формат аудиоданных
    if (mmioRead(hwav, @wfx^, sizeof(wfx^)) <> sizeof(wfx^)) then
        ABORT;

    // Проверяем формат
    if (wfx.wFormatTag <> WAVE_FORMAT_PCM) then ABORT;

    // Поднимаемся на уровень вверх, чтобы получить доступ
    // к аудиоданным
    if (mmioAscend(hwav, @child, 0) <> 0) then ABORT;

    // Ищем блок 'data'
    child.ckid := mmioStringToFOURCC('data', 0);
    if (mmioDescend(hwav, @child, @parent, MMIO_FINDCHUNK) <> 0)
        then ABORT;
```

```
      // Выделяем память под аудиоданные
      result := GetMemory(child.cksize);

      // Читаем аудиоданные
      mmioRead(hwav, PChar(result), child.cksize);

      dwSize^ := child.cksize;
    except

      if result <> NIL then
      begin
        FreeMemory(result);
        result := NIL;
      end;
    end;

  finally

    // Закрываем файл
    mmioClose(hwav, 0);
  end;
end;

{*********************************************************************}
{** Воспроизводим буфер                                             **}
{*********************************************************************}
function TdxSound.PlaySound(Looping: boolean): HResult;
begin
  FdsBuffer8.SetCurrentPosition(0);
  result := FdsBuffer8.Play(0, 0, integer(Looping));
end;

{*********************************************************************}
{** Устанавливаем эффекты воспроизведения для буфера                **}
{*********************************************************************}
function TdxSound.SetEffects(EffectsMask: DWORD): HResult;
```

```
const
  Effects: array[0..SFX_COUNT - 1] of WORD =
  ( SFX_STANDARD_CHORUS,
    SFX_STANDARD_COMPRESSOR,
    SFX_STANDARD_DISTORTION,
    SFX_STANDARD_ECHO,
    SFX_STANDARD_FLANGER,
    SFX_STANDARD_GARGLE,
    SFX_STANDARD_PARAMEQ,
    SFX_WAVES_REVERB );
var
  I: integer;
  Count: integer;
  dwStatus: DWORD;
begin
  ZeroMemory(@dsEffect, sizeof(dsEffect));
  Count := 0;
  // Заполняем структуру, описывающую эффекты
  for I := 0 to SFX_COUNT - 1 do
  begin
    if (EffectsMask and Effects[I] =  Effects[I]) then
    begin
      dsEffect[Count].dwSize := sizeof(dsEffect[Count]);
      dsEffect[Count].dwFlags := 0;
      dsEffect[Count].guidDSFXClass := GetEffectGuid(Effects[I]);

      inc(Count);
    end;
  end;

  // Получаем текущий статус буфера
  result := FdsBuffer8.GetStatus(dwStatus);
  if FAILED(result) then EXIT;

  // Проверяем, не воспроизводится ли буфер
  if (dwStatus and DSBSTATUS_PLAYING <> 0) then
```

```
begin
  result := StopSound;
  if FAILED(result) then EXIT;
end;

// Очищаем список эффектов
result := FdsBuffer8.SetFX(0, NIL, NIL);
if Count = 0 then EXIT;

// Устанавливаем новый набор эффектов
ZeroMemory(@dwResults, sizeof(dwResults));
result := FdsBuffer8.SetFX(Count, @dsEffect, @dwResults);

// В случае успеха получаем интерфейсы эффектов
if (SUCCEEDED(result)) then
begin
  FdsBuffer8.GetObjectInPath(GUID_DSFX_STANDARD_CHORUS, 0,
    IID_IDirectSoundFXChorus8, FdsChorus8);
  FdsBuffer8.GetObjectInPath(GUID_DSFX_STANDARD_COMPRESSOR, 0,
    IID_IDirectSoundFXCompressor8, FdsCompressor8);
  FdsBuffer8.GetObjectInPath(GUID_DSFX_STANDARD_DISTORTION, 0,
    IID_IDirectSoundFXDistortion8, FdsDistortion8);
  FdsBuffer8.GetObjectInPath(GUID_DSFX_STANDARD_ECHO, 0,
    IID_IDirectSoundFXEcho8, FdsEcho8);
  FdsBuffer8.GetObjectInPath(GUID_DSFX_STANDARD_FLANGER, 0,
    IID_IDirectSoundFXFlanger8, FdsFlanger8);
  FdsBuffer8.GetObjectInPath(GUID_DSFX_STANDARD_GARGLE, 0,
    IID_IDirectSoundFXGargle8, FdsGargle8);
  FdsBuffer8.GetObjectInPath(GUID_DSFX_STANDARD_PARAMEQ, 0,
    IID_IDirectSoundFXParameq8, FdsParameq8);
  FdsBuffer8.GetObjectInPath(GUID_DSFX_WAVES_REVERB, 0,
    IID_IDirectSoundFXWavesReverb8, FdsReverb8);
  EXIT;
end;
```

```
  // Анализ ошибок
  case result of
    CO_E_NOTINITIALIZED   :   ShowMessage('CO_E_NOTINITIALIZED');
    DSERR_CONTROLUNAVAIL  :   ShowMessage('CDSERR_CONTROLUNAVAIL');
    DSERR_GENERIC         :   ShowMessage('CDSERR_GENERIC');
    DSERR_INVALIDPARAM    :   ShowMessage('CDSERR_INVALIDPARAM');
    DSERR_INVALIDCALL     :   ShowMessage('CDSERR_INVALIDCALL');
    DSERR_NOINTERFACE     :   ShowMessage('CDSERR_NOINTERFACE');
    DSERR_PRIOLEVELNEEDED :   ShowMessage('CDSERR_PRIOLEVELNEEDED');
  end;
end;

{*********************************************************************}
{** Получаем эффекты воспроизведения для буфера                     **}
{*********************************************************************}
function TdxSound.GetEffects(var EffectsMask: DWORD): HResult;
begin
  EffectsMask := 0;

  if FdsChorus8 <> NIL then
     EffectsMask := EffectsMask or SFX_STANDARD_CHORUS;
  if FdsCompressor8 <> NIL then
     EffectsMask := EffectsMask or SFX_STANDARD_COMPRESSOR;
  if FdsDistortion8 <> NIL then
     EffectsMask := EffectsMask or SFX_STANDARD_DISTORTION;
  if FdsEcho8 <> NIL then
     EffectsMask := EffectsMask or SFX_STANDARD_ECHO;
  if FdsFlanger8 <> NIL then
     EffectsMask := EffectsMask or SFX_STANDARD_FLANGER;
  if FdsGargle8 <> NIL then
     EffectsMask := EffectsMask or SFX_STANDARD_GARGLE;
  if FdsParamEq8 <> NIL then
     EffectsMask := EffectsMask or SFX_STANDARD_PARAMEQ;
  if FdsReverb8 <> NIL then
     EffectsMask := EffectsMask or SFX_WAVES_REVERB;
```

```
  result := S_OK;
end;

{*****************************************************************}
{** Устанавливаем частоту дискретизации                         **}
{*****************************************************************}
function TdxSound.SetFrequency(Frequency: Cardinal): HResult;
begin
  result := FdsBuffer8.SetFrequency(Frequency);
end;

{*****************************************************************}
{** Устанавливаем положение на панораме                         **}
{*****************************************************************}
function TdxSound.SetPan(Pan: integer): HResult;
begin
  result := FdsBuffer8.SetPan(Pan);
end;

{*****************************************************************}
{** Устанавливаем громкость звучания                            **}
{*****************************************************************}
function TdxSound.SetVolume(Volume: integer): HResult;
begin
  result := FdsBuffer8.SetVolume(Volume);
end;

{*****************************************************************}
{** Останавливаем воспроизведение                               **}
{*****************************************************************}
function TdxSound.StopSound: HResult;
begin
  result := FdsBuffer8.Stop;
end;
```

```
{*********************************************************************}
{** Заполняем буфер аудиоданными                                    **}
{*********************************************************************}
function TdxSound.FillBuffer: HResult;
var
  AudioPtr1, AudioPtr2: Pointer;
  AudioBytes1, AudioBytes2: DWORD;
begin
  // Обрабатываем ситуацию потери буфера
  result := RestoreBuffer(NIL);
  if FAILED(Result) then EXIT;
  // Блокируем буфер и получаем указатель на заблокированный блок
  // данных
  result := FdsBuffer8.Lock(0, FDataSize, @AudioPtr1, @AudioBytes1,
    @AudioPtr2, @AudioBytes2, 0);
  if FAILED(result) then EXIT;

  // Копируем данные в буфер
  CopyMemory(AudioPtr1, FData, AudioBytes1);
  if (AudioPtr2 <> NIL) then
    CopyMemory(AudioPtr2, Pointer(DWORD(FData) + AudioBytes1),
               AudioBytes2);

  // Разблокировка буфера
  FdsBuffer8.Unlock(AudioPtr1, AudioBytes1, AudioPtr2, AudioBytes2);
end;

{*********************************************************************}
{** Восстанавливаем буфер после потери                              **}
{*********************************************************************}
function TdxSound.RestoreBuffer(WasRestored: PBOOL): HResult;
var
  dwStatus: DWORD;
begin
  if WasRestored <> NIL then WasRestored^ := FALSE;
```

```
  // Получаем текущий статус буфера
  result := FdsBuffer8.GetStatus(dwStatus);
  if FAILED(result) then EXIT;

  // Проверяем на потерю
  if (dwStatus and DSBSTATUS_BUFFERLOST <> 0) then
  begin
    // Пытаемся восстановить
    Result := FdsBuffer8.Restore;
    while (Result = DSERR_BUFFERLOST) do
    begin
      Sleep(10);
      Result := FdsBuffer8.Restore;
    end;

    if WasRestored <> NIL then WasRestored^ := TRUE;

    Result:= S_OK;
  end else
  begin
    Result:= S_FALSE;
  end;
end;

{*****************************************************************}
{** Получаем идентификатор эффекта по внутреннему индексу       **}
{*****************************************************************}
function TdxSound.GetEffectGuid(Effect: WORD): TGUID;
begin
  result := GUID_NULL;

  case Effect of
    SFX_STANDARD_CHORUS:
      result := GUID_DSFX_STANDARD_CHORUS;
    SFX_STANDARD_COMPRESSOR:
      result := GUID_DSFX_STANDARD_COMPRESSOR;
```

```
    SFX_STANDARD_DISTORTION:
      result := GUID_DSFX_STANDARD_DISTORTION;
    SFX_STANDARD_ECHO:
      result := GUID_DSFX_STANDARD_ECHO;
    SFX_STANDARD_FLANGER:
      result := GUID_DSFX_STANDARD_FLANGER;
    SFX_STANDARD_GARGLE:
      result := GUID_DSFX_STANDARD_GARGLE;
    SFX_STANDARD_PARAMEQ:
      result := GUID_DSFX_STANDARD_PARAMEQ;
    SFX_WAVES_REVERB:
      result := GUID_DSFX_WAVES_REVERB;
  end;
end;

{*********************************************************************}
{** Получаем параметры эффекта                                      **}
{*********************************************************************}
function TdxSound.GetEffectParams(Effect: DWORD; Params: pointer):
  HResult;
begin
  result := E_FAIL;

  case Effect of
    SFX_STANDARD_CHORUS:

      if FdsChorus8 <> NIL then
      begin
        result := FdsChorus8.GetAllParameters(FParamsChorus);

        CopyMemory(Params, @FParamsChorus, sizeof(FParamsChorus));
      end;

    SFX_STANDARD_COMPRESSOR:
      if FdsCompressor8 <> NIL then
```

```
    begin
      result :=
        FdsCompressor8.GetAllParameters(FParamsCompressor);

      CopyMemory(Params, @FParamsCompressor,
        sizeof(FParamsCompressor));
    end;

  SFX_STANDARD_DISTORTION:
    if FdsDistortion8 <> NIL then
    begin
      result :=
        FdsDistortion8.GetAllParameters(FParamsDistortion);

      CopyMemory(Params, @FParamsDistortion,
        sizeof(FParamsDistortion));
    end;

  SFX_STANDARD_ECHO:
    if FdsEcho8 <> NIL then
    begin
      result := FdsEcho8.SetAllParameters(FParamsEcho);

      CopyMemory(Params, @FParamsEcho, sizeof(FParamsEcho));
    end;

  SFX_STANDARD_FLANGER:
    if FdsFlanger8 <> NIL then
    begin
      result := FdsFlanger8.GetAllParameters(FParamsFlanger);

      CopyMemory(Params, @FParamsFlanger, sizeof(FParamsFlanger));
    end;

  SFX_STANDARD_GARGLE:
    if FdsGargle8 <> NIL then
```

```
      begin
        result := FdsGargle8.GetAllParameters(FParamsGargle);

        CopyMemory(Params, @FParamsGargle, sizeof(FParamsGargle));
      end;

    SFX_STANDARD_PARAMEQ:
      if FdsParamEq8 <> NIL then
      begin
        result := FdsParamEq8.GetAllParameters(FParamsParamEq);

        CopyMemory(Params, @FParamsParamEq, sizeof(FParamsParamEq));
      end;

    SFX_WAVES_REVERB:
      if FdsReverb8 <> NIL then
      begin
        result := FdsReverb8.GetAllParameters(FParamsReverb);

        CopyMemory(Params, @FParamsReverb, sizeof(FParamsReverb));
      end;
  end;
end;

{*****************************************************************}
{** Устанавливаем параметры эффекта                             **}
{*****************************************************************}
function TdxSound.SetEffectParams(Effect: DWORD;
  const Params: pointer): HResult;
begin
  result := E_FAIL;

  case Effect of
    SFX_STANDARD_CHORUS:
      if FdsChorus8 <> NIL then
```

```
    begin
      FParamsChorus := TDSFXChorus(Params^);
      result :=
        FdsChorus8.SetAllParameters(FParamsChorus);
    end;

  SFX_STANDARD_COMPRESSOR:
    if FdsCompressor8 <> NIL then
    begin
      FParamsCompressor := TDSFXCompressor(Params^);
      result :=
        FdsCompressor8.SetAllParameters(FParamsCompressor);
    end;

  SFX_STANDARD_DISTORTION:
    if FdsDistortion8 <> NIL then
    begin
      FParamsDistortion := TDSFXDistortion(Params^);
      result :=
        FdsDistortion8.SetAllParameters(FParamsDistortion);
    end;

  SFX_STANDARD_ECHO:
    if FdsEcho8 <> NIL then
    begin
      FParamsEcho := TDSFXEcho(Params^);
      result :=
        FdsEcho8.SetAllParameters(FParamsEcho);
    end;

  SFX_STANDARD_FLANGER:
    if FdsFlanger8 <> NIL then
    begin
      FParamsFlanger := TDSFXFlanger(Params^);
      result :=
        FdsFlanger8.SetAllParameters(FParamsFlanger);
    end;
```

```
    SFX_STANDARD_GARGLE:
      if FdsGargle8 <> NIL then
      begin
        FParamsGargle := TDSFXGargle(Params^);
        result :=
          FdsGargle8.SetAllParameters(FParamsGargle);
      end;

    SFX_STANDARD_PARAMEQ:
      if FdsParamEq8 <> NIL then
      begin
        FParamsParamEq := TDSFXParamEq(Params^);
        result :=
          FdsParamEq8.SetAllParameters(FParamsParamEq);
      end;

    SFX_WAVES_REVERB:
      if FdsReverb8 <> NIL then
      begin
        FParamsReverb:= TDSFXWavesReverb(Params^);

        result :=
          FdsReverb8.SetAllParameters(FParamsReverb);
      end;
  end;
end;

END.
```

Пример использования классов *TdxSound* и *TdxSoundManager*

На прилагаемом к книге компакт-диске в папке Examples\DirectSound\DS_Test находится пример использования классов `TdxSound` и `TdxSoundManager`. Пример состоит из одной главной формы, на которой расположены различные элементы управления: список загруженных WAV-файлов, кнопки до-

бавления файла в список удаления, воспроизведения и остановки, регулятор громкости, регулятор частоты дискретизации, регулятор положения на панораме и флаг циклического проигрывания звуковых файлов, список доступных звуковых эффектов и их параметры. Общий вид приложения представлен на рис. 8.1.

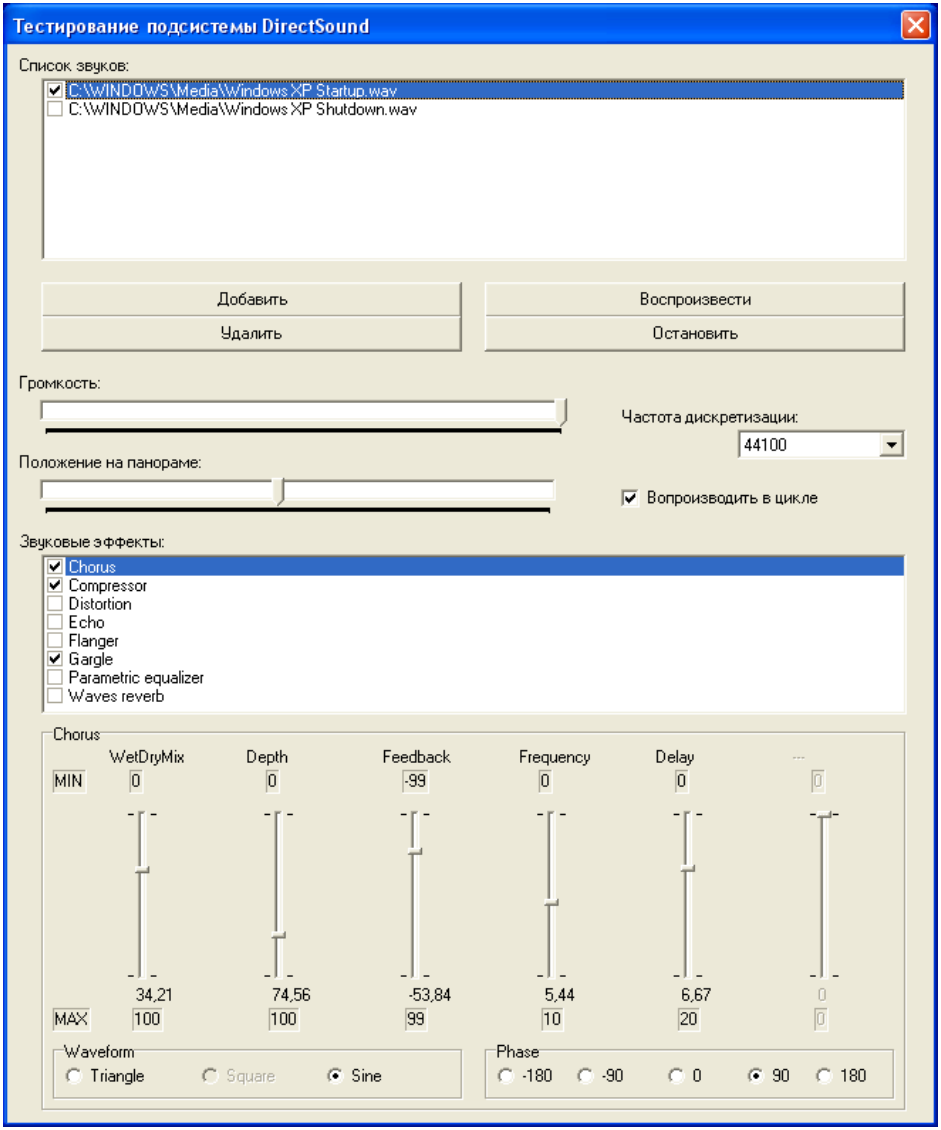


Рис. 8.1. Общий вид тестирующего приложения для класса `TdxSoundManager`

В листинге 8.2 приводится исходный код основного модуля тестирующего приложения.

Листинг 8.2. Текст модуля FormMain.pas проекта DS_Sound

```
UNIT FormMain;
{*****************************************************************}
{**  Тестирование классов TdxSound и TdxSoundManager             **}
{**  Автор: Есенин Сергей Анатольевич                            **}
{*****************************************************************}

{**} INTERFACE {**************************************************}

{**} USES {*******************************************************}
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, CheckLst, ComCtrls,
  DirectSound, UdxSoundManager, ExtCtrls;

{**} TYPE {*******************************************************}
  TMainForm = class(TForm)
    checkListBoxWAV: TCheckListBox;
    labelSoundList: TLabel;
    buttonAdd: TButton;
    buttonRemove: TButton;
    buttonPlay: TButton;
    trackBarVolume: TTrackBar;
    labelVolume: TLabel;
    labelFrequency: TLabel;
    comboBoxFrequency: TComboBox;
    labelPan: TLabel;
    trackBarPan: TTrackBar;
    openDialogWAVFiles: TOpenDialog;
    checkBoxLooping: TCheckBox;
    buttonStop: TButton;
    checkListBoxEffects: TCheckListBox;
    labelEffects: TLabel;
    groupBoxParams: TGroupBox;
```

```
    tbParam1: TTrackBar;
    tbParam2: TTrackBar;
    tbParam3: TTrackBar;
    tbParam4: TTrackBar;
    tbParam5: TTrackBar;
    tbParam6: TTrackBar;
    staticTextMax: TStaticText;
    staticTextMin: TStaticText;
    lbParamName1: TLabel;
    lbParamName2: TLabel;
    lbParamName3: TLabel;
    lbParamName4: TLabel;
    lbParamName5: TLabel;
    lbParamName6: TLabel;
    lbValue1: TLabel;
    lbValue2: TLabel;
    lbValue3: TLabel;
    lbValue4: TLabel;
    lbValue5: TLabel;
    lbValue6: TLabel;
    stMaxValue1: TStaticText;
    stMaxValue2: TStaticText;
    stMaxValue3: TStaticText;
    stMaxValue4: TStaticText;
    stMaxValue5: TStaticText;
    stMaxValue6: TStaticText;
    stMinValue1: TStaticText;
    stMinValue2: TStaticText;
    stMinValue4: TStaticText;
    stMinValue3: TStaticText;
    stMinValue6: TStaticText;
    stMinValue5: TStaticText;
    groupBoxWaveform: TGroupBox;
    groupBoxPhase: TGroupBox;
    rbTriangle: TRadioButton;
    rbSquare: TRadioButton;
```

```
    rbm180: TRadioButton;
    rbm90: TRadioButton;
    rbZero: TRadioButton;
    rb90: TRadioButton;
    rb180: TRadioButton;
    rbSine: TRadioButton;
    procedure buttonAddClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure buttonRemoveClick(Sender: TObject);
    procedure checkListBoxWAVClick(Sender: TObject);
    procedure buttonPlayClick(Sender: TObject);
    procedure trackBarVolumeChange(Sender: TObject);
    procedure trackBarPanChange(Sender: TObject);
    procedure comboBoxFrequencyChange(Sender: TObject);
    procedure buttonStopClick(Sender: TObject);
    procedure checkListBoxEffectsClickCheck(Sender: TObject);
    procedure SetEffectParams(Sender: TObject);
    procedure checkListBoxEffectsClick(Sender: TObject);
  PRIVATE
    FParamsChorus:     TDSFXChorus;
    FParamsCompressor: TDSFXCompressor;
    FParamsDistortion: TDSFXDistortion;
    FParamsEcho:       TDSFXEcho;
    FParamsFlanger:    TDSFXFlanger;
    FParamsGargle:     TDSFXGargle;
    FParamsParamEq:    TDSFXParamEq;
    FParamsReverb:     TDSFXWavesReverb;

    FCurrentEffect:    integer;
    FInRefreshing:     boolean;

    procedure SetParamValues(Index: integer; Value: Single;
      minValue, maxValue: Single; ParamName: string;
      IsEnabled: boolean);
```

```
    procedure SetWaveformParams(Index: integer; IsParam1Enabled,
      IsParam2Enabled, IsParam3Enabled: boolean; IsEnabled:
      boolean);
    procedure SetPhaseParams(Index: integer; IsEnabled: boolean);

    procedure RefreshEffectParams;
  PUBLIC
    property CurrentEffect: integer read FCurrentEffect write
      FCurrentEffect;
  END;

{**} VAR {***************************************************}
  MainForm: TMainForm;
  SoundManager: TdxSoundManager;

{**} IMPLEMENTATION {*****************************************}

{$R *.dfm}
{*****************************************************************}
{**  Добавление нового звукового буфера                         **}
{*****************************************************************}
procedure TMainForm.buttonAddClick(Sender: TObject);
begin
  if openDialogWAVFiles.Execute then
  begin
    if FAILED(SoundManager.CreateSound(openDialogWAVFiles.FileName))
    then ShowMessage('Неверный формат данных!')
    else
    begin
       checkListBoxWAV.Items.Add(openDialogWAVFiles.FileName);
       checkListBoxWAV.Checked[checkListBoxWAV.Count - 1] := TRUE;
       checkListBoxWAV.ItemIndex := checkListBoxWAV.Count - 1;
       checkListBoxWAVClick(NIL);
    end;
  end;
end;
```

```
{*********************************************************************}
{**  Создание и инициализация объекта SoundManager                  **}
{*********************************************************************}
procedure TMainForm.FormCreate(Sender: TObject);
var
  WinDir: array[0..MAX_PATH + 1] of char;
begin
  GetWindowsDirectory(WinDir, 256);
  openDialogWAVFiles.InitialDir := WinDir + '\Media';

  SoundManager := TdxSoundManager.Create(handle);

  checkListBoxWAVClick(NIL);
  if FAILED(SoundManager.Initialize) then
  begin
    FreeAndNil(SoundManager);
  end;
end;

{*********************************************************************}
{**  Удаление объекта SoundManager                                  **}
{*********************************************************************}
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  if SoundManager <> NIL then
     FreeAndNil(SoundManager);
end;

{*********************************************************************}
{**  Удаление буфера из списка                                      **}
{*********************************************************************}
procedure TMainForm.buttonRemoveClick(Sender: TObject);
begin
  if checkListBoxWAV.ItemIndex < 0 then EXIT;
```

```
  if not FAILED(SoundManager.DeleteSound(checkListBoxWAV.ItemIndex))
  then begin
    checkListBoxWAV.Items.Delete(checkListBoxWAV.ItemIndex);
    checkListBoxWAVClick(NIL);
  end;
end;

{*****************************************************************}
{**  Получение информации о буфере                              **}
{*****************************************************************}
procedure TMainForm.checkListBoxWAVClick(Sender: TObject);
var
  Frequency: Cardinal;
  Pan: integer;
  Volume: integer;
  Sound: TdxSound;
  EffectsMask:  DWORD;
  I: integer;
begin
  FInRefreshing := TRUE;
  try
    for I := 0 to checkListBoxEffects.Count - 1 do
      checkListBoxEffects.Checked[I] := FALSE;

    SetParamValues(1, 0, 0, 0, '---', FALSE);
    SetParamValues(2, 0, 0, 0, '---', FALSE);
    SetParamValues(3, 0, 0, 0, '---', FALSE);
    SetParamValues(4, 0, 0, 0, '---', FALSE);
    SetParamValues(5, 0, 0, 0, '---', FALSE);
    SetParamValues(6, 0, 0, 0, '---', FALSE);

    SetWaveformParams(-1, FALSE, FALSE, FALSE, FALSE);
    SetPhaseParams(-1, FALSE);

    if checkListBoxWAV.ItemIndex < 0 then EXIT;
```

```
Sound := SoundManager.GetSound(checkListBoxWAV.ItemIndex);
if Sound = NIL then EXIT;

Sound.GetFrequency(Frequency);
Sound.GetPan(Pan);
Sound.GetVolume(Volume);

trackBarVolume.Position := Volume;
trackBarPan.Position := Pan;
case Frequency of
  8000:  comboBoxFrequency.ItemIndex := 0;
  11025: comboBoxFrequency.ItemIndex := 1;
  22050: comboBoxFrequency.ItemIndex := 2;
  44100: comboBoxFrequency.ItemIndex := 3;
  else   comboBoxFrequency.ItemIndex := -1;
end;

if FAILED(Sound.GetEffects(EffectsMask)) then EXIT;

checkListBoxEffects.Checked[0] :=
  EffectsMask and SFX_STANDARD_CHORUS = SFX_STANDARD_CHORUS;
checkListBoxEffects.Checked[1] :=
  EffectsMask and SFX_STANDARD_COMPRESSOR =
                  SFX_STANDARD_COMPRESSOR;
checkListBoxEffects.Checked[2] :=
  EffectsMask and SFX_STANDARD_DISTORTION =
                  SFX_STANDARD_DISTORTION;
checkListBoxEffects.Checked[3] :=
  EffectsMask and SFX_STANDARD_ECHO = SFX_STANDARD_ECHO;
checkListBoxEffects.Checked[4] :=
  EffectsMask and SFX_STANDARD_FLANGER = SFX_STANDARD_FLANGER;
checkListBoxEffects.Checked[5] :=
  EffectsMask and SFX_STANDARD_GARGLE = SFX_STANDARD_GARGLE;

checkListBoxEffects.Checked[6] :=
  EffectsMask and SFX_STANDARD_PARAMEQ = SFX_STANDARD_PARAMEQ;
```

```
    checkListBoxEffects.Checked[7] :=
      EffectsMask and SFX_WAVES_REVERB = SFX_WAVES_REVERB;

    for I := 0 to checkListBoxEffects.Count - 1 do
      if checkListBoxEffects.Checked[I] then
      begin
        checkListBoxEffects.ItemIndex := I;
        checkListBoxEffectsClick(NIL);
        EXIT;
      end;

  finally
    FInRefreshing := FALSE;
  end;
end;

{*****************************************************************}
{**  Воспроизведение отмеченных буферов                         **}
{*****************************************************************}
procedure TMainForm.buttonPlayClick(Sender: TObject);
var
  I: integer;
  Sound: TdxSound;
begin
  for I := 0 to checkListBoxWAV.Count - 1 do
    if checkListBoxWAV.Checked[I] then
    begin
      Sound := SoundManager.GetSound(I);

      if Sound <> NIL then
         Sound.PlaySound(checkBoxLooping.Checked);
    end;
end;

{*****************************************************************}
{** Изменение громкости звучания текущего буфера               **}
{*****************************************************************}
```

```
procedure TMainForm.trackBarVolumeChange(Sender: TObject);
var
  Sound: TdxSound;
begin
  if checkListBoxWAV.ItemIndex < 0 then EXIT;
  Sound := SoundManager.GetSound(checkListBoxWAV.ItemIndex);
  if Sound = NIL then EXIT;

  Sound.SetVolume(trackBarVolume.Position);
end;

{*****************************************************************}
{** Изменение положения на панораме текущего буфера             **}
{*****************************************************************}
procedure TMainForm.trackBarPanChange(Sender: TObject);
var
  Sound: TdxSound;
begin
  if checkListBoxWAV.ItemIndex < 0 then EXIT;
  Sound := SoundManager.GetSound(checkListBoxWAV.ItemIndex);
  if Sound = NIL then EXIT;
  Sound.SetPan(trackBarPan.Position);
end;

{*****************************************************************}
{** Изменение частоты дискретизации текущего буфера             **}
{*****************************************************************}
procedure TMainForm.comboBoxFrequencyChange(Sender: TObject);
var
  Sound: TdxSound;
begin
  if checkListBoxWAV.ItemIndex < 0 then EXIT;
  Sound := SoundManager.GetSound(checkListBoxWAV.ItemIndex);
  if Sound = NIL then EXIT;

  Sound.SetFrequency(StrToInt(comboBoxFrequency.Text));
end;
```

```
{*****************************************************************}
{** Остановка воспроизведения отмеченных буферов                **}
{*****************************************************************}
procedure TMainForm.buttonStopClick(Sender: TObject);
var
  I: integer;
  Sound: TdxSound;
begin
  for I := 0 to checkListBoxWAV.Count - 1 do
    if checkListBoxWAV.Checked[I] then
    begin
      Sound := SoundManager.GetSound(I);
      if Sound <> NIL then
         Sound.StopSound;
    end;
end;

{*****************************************************************}
{** Установка звуковых эффектов для буфера                     **}
{*****************************************************************}
procedure TMainForm.checkListBoxEffectsClickCheck(Sender: TObject);
var
  EffectsMask:  DWORD;
  Sound: TdxSound;
begin
  SetParamValues(1, 0, 0, 0, '---', FALSE);
  SetParamValues(2, 0, 0, 0, '---', FALSE);
  SetParamValues(3, 0, 0, 0, '---', FALSE);
  SetParamValues(4, 0, 0, 0, '---', FALSE);
  SetParamValues(5, 0, 0, 0, '---', FALSE);
  SetParamValues(6, 0, 0, 0, '---', FALSE);

  SetWaveformParams(-1, FALSE, FALSE, FALSE, FALSE);
  SetPhaseParams(-1, FALSE);

  Sound := SoundManager.GetSound(checkListBoxWAV.ItemIndex);
  if Sound = NIL then
```

```
  begin
    checkListBoxEffects.Checked[checkListBoxEffects.ItemIndex] :=
      FALSE;
    EXIT;
  end;

  EffectsMask := 0;
  if checkListBoxEffects.Checked[0] then
    EffectsMask := EffectsMask or SFX_STANDARD_CHORUS;
  if checkListBoxEffects.Checked[1] then
    EffectsMask := EffectsMask or SFX_STANDARD_COMPRESSOR;
  if checkListBoxEffects.Checked[2] then
    EffectsMask := EffectsMask or SFX_STANDARD_DISTORTION;
  if checkListBoxEffects.Checked[3] then
    EffectsMask := EffectsMask or SFX_STANDARD_ECHO;
  if checkListBoxEffects.Checked[4] then
    EffectsMask := EffectsMask or SFX_STANDARD_FLANGER;
  if checkListBoxEffects.Checked[5] then
    EffectsMask := EffectsMask or SFX_STANDARD_GARGLE;
  if checkListBoxEffects.Checked[6] then
    EffectsMask := EffectsMask or SFX_STANDARD_PARAMEQ;
  if checkListBoxEffects.Checked[7] then
    EffectsMask := EffectsMask or SFX_WAVES_REVERB;

  if FAILED(Sound.SetEffects(EffectsMask)) then
    checkListBoxEffects.Checked[checkListBoxEffects.ItemIndex] :=
    not checkListBoxEffects.Checked[checkListBoxEffects.ItemIndex];
end;

{*****************************************************************}
{** Обновление значений параметров текущего звукового эффекта    **}
{*****************************************************************}
procedure TMainForm.RefreshEffectParams;
var
  Sound: TdxSound;
  Index: integer;
```

```
begin
  Sound := SoundManager.GetSound(checkListBoxWAV.ItemIndex);
  if Sound = NIL then EXIT;

  Index := -1;
  FInRefreshing := TRUE;
  try
    case CurrentEffect of
      SFX_STANDARD_CHORUS:
        begin
          groupBoxParams.Caption := 'Chorus';
          Sound.GetEffectParams(SFX_STANDARD_CHORUS,
            @FParamsChorus);

          SetParamValues(1, FParamsChorus.fWetDryMix,
            DSFXCHORUS_WETDRYMIX_MIN, DSFXCHORUS_WETDRYMIX_MAX,
            'WetDryMix', TRUE);
          SetParamValues(2, FParamsChorus.fDepth,
            DSFXCHORUS_DEPTH_MIN, DSFXCHORUS_DEPTH_MAX, 'Depth',
            TRUE);
          SetParamValues(3, FParamsChorus.fFeedback,
            DSFXCHORUS_FEEDBACK_MIN, DSFXCHORUS_FEEDBACK_MAX,
            'Feedback', TRUE);
          SetParamValues(4, FParamsChorus.fFrequency,
            DSFXCHORUS_FREQUENCY_MIN, DSFXCHORUS_FREQUENCY_MAX,
            'Frequency', TRUE);
          SetParamValues(5, FParamsChorus.fDelay,
            DSFXCHORUS_DELAY_MIN, DSFXCHORUS_DELAY_MAX, 'Delay',
            TRUE);
          SetParamValues(6, 0, 0, 0, '---', FALSE);

          case FParamsChorus.lWaveform of
            DSFXCHORUS_WAVE_TRIANGLE : Index := 0;
            DSFXCHORUS_WAVE_SIN      : Index := 2;
          end;
          SetWaveformParams(Index, TRUE, FALSE, TRUE, TRUE);
```

```
      case FParamsChorus.lPhase of
        DSFXCHORUS_PHASE_NEG_180 : Index := 0;
        DSFXCHORUS_PHASE_NEG_90  : Index := 1;
        DSFXCHORUS_PHASE_ZERO    : Index := 2;
        DSFXCHORUS_PHASE_90      : Index := 3;
        DSFXCHORUS_PHASE_180     : Index := 4;
      end;
      SetPhaseParams(Index, TRUE);
    end;

  SFX_STANDARD_COMPRESSOR:
    begin
      groupBoxParams.Caption := 'Compressor';
      Sound.GetEffectParams(SFX_STANDARD_COMPRESSOR,
        @FParamsCompressor);

      SetParamValues(1, FParamsCompressor.fAttack,
        DSFXCOMPRESSOR_ATTACK_MIN, DSFXCOMPRESSOR_ATTACK_MAX,
        'Attack', TRUE);
      SetParamValues(2, FParamsCompressor.fGain,
        DSFXCOMPRESSOR_GAIN_MIN, DSFXCOMPRESSOR_GAIN_MAX,
        'Gain', TRUE);
      SetParamValues(3, FParamsCompressor.fPredelay,
        DSFXCOMPRESSOR_PREDELAY_MIN,
        DSFXCOMPRESSOR_PREDELAY_MAX, 'Predelay', TRUE);
      SetParamValues(4, FParamsCompressor.fRatio,
        DSFXCOMPRESSOR_RATIO_MIN, DSFXCOMPRESSOR_RATIO_MAX,
        'Ratio', TRUE);
      SetParamValues(5, FParamsCompressor.fRelease,
        DSFXCOMPRESSOR_RELEASE_MIN, DSFXCOMPRESSOR_RELEASE_MAX,
        'Release', TRUE);
      SetParamValues(6, FParamsCompressor.fThreshold,
        DSFXCOMPRESSOR_THRESHOLD_MIN,
        DSFXCOMPRESSOR_THRESHOLD_MAX, 'Threshold', TRUE);
```

```
      SetWaveformParams(Index, FALSE, FALSE, FALSE, FALSE);
      SetPhaseParams(Index, FALSE);
    end;

  SFX_STANDARD_DISTORTION:
    begin
      groupBoxParams.Caption := 'Distortion';
      Sound.GetEffectParams(SFX_STANDARD_DISTORTION,
        @FParamsDistortion);

      SetParamValues(1, FParamsDistortion.fEdge,
        DSFXDISTORTION_EDGE_MIN, DSFXDISTORTION_EDGE_MAX,
        'Edge', TRUE);
      SetParamValues(2, FParamsDistortion.fGain,
        DSFXDISTORTION_GAIN_MIN, DSFXDISTORTION_GAIN_MAX,
        'Gain', TRUE);
      SetParamValues(3, FParamsDistortion.fPostEQBandwidth,
        DSFXDISTORTION_POSTEQBANDWIDTH_MIN,
        DSFXDISTORTION_POSTEQBANDWIDTH_MAX, 'PostEQBandWidth',
        TRUE);
      SetParamValues(4,
        FParamsDistortion.fPostEQCenterFrequency,
        DSFXDISTORTION_POSTEQCENTERFREQUENCY_MIN,
        DSFXDISTORTION_POSTEQCENTERFREQUENCY_MAX,
        'PostEQCenterFreq', TRUE);

      SetParamValues(5, FParamsDistortion.fPreLowpassCutoff,
        DSFXDISTORTION_PRELOWPASSCUTOFF_MIN,
        DSFXDISTORTION_PRELOWPASSCUTOFF_MAX, 'PreLowpassCutoff',
        TRUE);
      SetParamValues(6, 0, 0, 0, '---', FALSE);

      SetWaveformParams(Index, FALSE, FALSE, FALSE, FALSE);
      SetPhaseParams(Index, FALSE);
    end;
```

```
SFX_STANDARD_ECHO:
  begin
    groupBoxParams.Caption := 'Echo';
    Sound.GetEffectParams(SFX_STANDARD_ECHO, @FParamsEcho);

    SetParamValues(1, FParamsEcho.fFeedback,
      DSFXECHO_FEEDBACK_MIN, DSFXECHO_FEEDBACK_MAX,
      'Feedback', TRUE);
    SetParamValues(2, FParamsEcho.fLeftDelay,
      DSFXECHO_LEFTDELAY_MIN, DSFXECHO_LEFTDELAY_MAX,
      'LeftDelay', TRUE);
    SetParamValues(3, FParamsEcho.fRightDelay,
      DSFXECHO_RIGHTDELAY_MIN, DSFXECHO_RIGHTDELAY_MAX,
      'RightDelay', TRUE);
    SetParamValues(4, FParamsEcho.fWetDryMix,
      DSFXECHO_WETDRYMIX_MIN, DSFXECHO_WETDRYMIX_MAX,
      'WetDryMix', TRUE);
    SetParamValues(5, FParamsEcho.lPanDelay,
      DSFXECHO_PANDELAY_MIN, DSFXECHO_PANDELAY_MAX,
      'PanDelay', TRUE);
    SetParamValues(6, 0, 0, 0, '---', FALSE);
    SetWaveformParams(Index, FALSE, FALSE, FALSE, FALSE);
    SetPhaseParams(Index, FALSE);
  end;

SFX_STANDARD_FLANGER:
  begin
    groupBoxParams.Caption := 'Flanger';
    Sound.GetEffectParams(SFX_STANDARD_FLANGER,
      @FParamsFlanger);

    SetParamValues(1, FParamsFlanger.fDelay,
      DSFXFLANGER_DELAY_MIN, DSFXFLANGER_DELAY_MAX, 'Delay',
      TRUE);
    SetParamValues(2, FParamsFlanger.fDepth,
```

```
        DSFXFLANGER_DEPTH_MIN, DSFXFLANGER_DEPTH_MAX, 'Depth',
        TRUE);
      SetParamValues(3, FParamsFlanger.fFeedback,
        DSFXFLANGER_FEEDBACK_MIN, DSFXFLANGER_FEEDBACK_MAX,
        'Feedback', TRUE);
      SetParamValues(4, FParamsFlanger.fFrequency,
        DSFXFLANGER_FREQUENCY_MIN, DSFXFLANGER_FREQUENCY_MAX,
        'Frequency', TRUE);
      SetParamValues(5, FParamsFlanger.fWetDryMix,
        DSFXFLANGER_WETDRYMIX_MIN, DSFXFLANGER_WETDRYMIX_MAX,
        'WetDryMix', TRUE);
      SetParamValues(6, FParamsFlanger.lPhase,
        DSFXFLANGER_PHASE_MIN, DSFXFLANGER_PHASE_MAX, 'Phase',
        TRUE);

      case FParamsFlanger.lWaveform of
        DSFXFLANGER_WAVE_TRIANGLE : Index := 0;
        DSFXFLANGER_WAVE_SIN      : Index := 2;
      end;
      SetWaveformParams(Index, TRUE, FALSE, TRUE, TRUE);

      case FParamsFlanger.lPhase of
        DSFXFLANGER_PHASE_NEG_180: Index := 0;
        DSFXFLANGER_PHASE_NEG_90 : Index := 1;
        DSFXFLANGER_PHASE_ZERO   : Index := 2;
        DSFXFLANGER_PHASE_90     : Index := 3;
        DSFXFLANGER_PHASE_180    : Index := 4;
      end;
      SetPhaseParams(Index, TRUE);
    end;

  SFX_STANDARD_GARGLE:
    begin
      groupBoxParams.Caption := 'Gargle';
      Sound.GetEffectParams(SFX_STANDARD_GARGLE,
        @FParamsGargle);
```

```
      SetParamValues(1, FParamsGargle.dwRateHz,
        DSFXGARGLE_RATEHZ_MIN, DSFXGARGLE_RATEHZ_MAX, 'RateHz',
        TRUE);
      SetParamValues(2, 0, 0, 0, '---', FALSE);
      SetParamValues(3, 0, 0, 0, '---', FALSE);
      SetParamValues(4, 0, 0, 0, '---', FALSE);
      SetParamValues(5, 0, 0, 0, '---', FALSE);
      SetParamValues(6, 0, 0, 0, '---', FALSE);

      case FParamsGargle.dwWaveShape of
        DSFXGARGLE_WAVE_TRIANGLE : Index := 0;
        DSFXGARGLE_WAVE_SQUARE   : Index := 1;
      end;
      SetWaveformParams(Index, TRUE, FALSE, TRUE, TRUE);

      SetPhaseParams(Index, FALSE);
    end;

  SFX_STANDARD_PARAMEQ:
    begin
      groupBoxParams.Caption := 'Parametric equalizer';
      Sound.GetEffectParams(SFX_STANDARD_PARAMEQ,
        @FParamsParamEq);

      SetParamValues(1, FParamsParamEq.fBandwidth,
        DSFXPARAMEQ_BANDWIDTH_MIN, DSFXPARAMEQ_BANDWIDTH_MAX,
        'Bandwidth', TRUE);
      SetParamValues(2, FParamsParamEq.fCenter,
        DSFXPARAMEQ_CENTER_MIN, DSFXPARAMEQ_CENTER_MAX,
        'Center', TRUE);
      SetParamValues(3, FParamsParamEq.fGain,
        DSFXPARAMEQ_GAIN_MIN, DSFXPARAMEQ_GAIN_MAX, 'Gain',
        TRUE);
      SetParamValues(4, 0, 0, 0, '---', FALSE);
      SetParamValues(5, 0, 0, 0, '---', FALSE);
      SetParamValues(6, 0, 0, 0, '---', FALSE);
```

```
        SetWaveformParams(Index, FALSE, FALSE, FALSE, FALSE);
        SetPhaseParams(Index, FALSE);
      end;

    SFX_WAVES_REVERB:
      begin
        groupBoxParams.Caption := 'Waves reverb';
        Sound.GetEffectParams(SFX_WAVES_REVERB, @FParamsReverb);

        SetParamValues(1, FParamsReverb.fHighFreqRTRatio,
          DSFX_WAVESREVERB_HIGHFREQRTRATIO_MIN,
          DSFX_WAVESREVERB_HIGHFREQRTRATIO_MAX, 'HighFreqRTRatio',
          TRUE);
        SetParamValues(2, FParamsReverb.fInGain,
          DSFX_WAVESREVERB_INGAIN_MIN,
          DSFX_WAVESREVERB_INGAIN_MAX, 'InGain', TRUE);
        SetParamValues(3, FParamsReverb.fReverbMix,
          DSFX_WAVESREVERB_REVERBMIX_MIN,
          DSFX_WAVESREVERB_REVERBMIX_MAX, 'ReverbMix', TRUE);
        SetParamValues(4, FParamsReverb.fReverbTime,
          DSFX_WAVESREVERB_REVERBTIME_MIN,
          DSFX_WAVESREVERB_REVERBTIME_MAX, 'ReverbTime', TRUE);
        SetParamValues(5, 0, 0, 0, '---', FALSE);
        SetParamValues(6, 0, 0, 0, '---', FALSE);

        SetWaveformParams(Index, FALSE, FALSE, FALSE, FALSE);
        SetPhaseParams(Index, FALSE);
      end;
    end;
  finally
    FInRefreshing := FALSE;
  end;
end;

{*****************************************************************}
{** Установка новых значений параметров текущего эффекта       **}
{*****************************************************************}
```

```
procedure TMainForm.SetEffectParams(Sender: TObject);
var
  Sound: TdxSound;
begin
  if FInRefreshing then EXIT;

  Sound := SoundManager.GetSound(checkListBoxWAV.ItemIndex);
  if Sound = NIL then EXIT;

  case CurrentEffect of
    SFX_STANDARD_CHORUS:
      begin
        FParamsChorus.fWetDryMix := tbParam1.Position / 100;
        FParamsChorus.fDepth     := tbParam2.Position / 100;
        FParamsChorus.fFeedback  := tbParam3.Position / 100;
        FParamsChorus.fFrequency := tbParam4.Position / 100;
        FParamsChorus.fDelay     := tbParam5.Position / 100;

        if rbTriangle.Checked then
           FParamsChorus.lWaveform := DSFXCHORUS_WAVE_TRIANGLE
        else if rbSine.Checked then
           FParamsChorus.lWaveform := DSFXCHORUS_WAVE_SIN;

        if rbm180.Checked then
           FParamsChorus.lPhase := DSFXCHORUS_PHASE_NEG_180
        else if rbm90.Checked then
           FParamsChorus.lPhase := DSFXCHORUS_PHASE_NEG_90
        else if rbZero.Checked then
           FParamsChorus.lPhase := DSFXCHORUS_PHASE_ZERO
        else if rb90.Checked then
           FParamsChorus.lPhase := DSFXCHORUS_PHASE_90
        else if rb180.Checked then
           FParamsChorus.lPhase := DSFXCHORUS_PHASE_180;

        Sound.SetEffectParams(SFX_STANDARD_CHORUS, @FParamsChorus)
      end;
```

```
SFX_STANDARD_COMPRESSOR:
  begin
    FParamsCompressor.fAttack     := tbParam1.Position / 100;
    FParamsCompressor.fGain       := tbParam2.Position / 100;
    FParamsCompressor.fPredelay   := tbParam3.Position / 100;
    FParamsCompressor.fRatio      := tbParam4.Position / 100;
    FParamsCompressor.fRelease    := tbParam5.Position / 100;
    FParamsCompressor.fThreshold  := tbParam6.Position / 100;

    Sound.SetEffectParams(SFX_STANDARD_COMPRESSOR,
      @FParamsCompressor)
  end;

SFX_STANDARD_DISTORTION:
  begin
    FParamsDistortion.fEdge := tbParam1.Position / 100;
    FParamsDistortion.fGain := tbParam2.Position / 100;
    FParamsDistortion.fPostEQBandwidth :=
      bParam3.Position / 100;
    FParamsDistortion.fPostEQCenterFrequency :=
      tbParam4.Position / 100;
    FParamsDistortion.fPreLowpassCutoff :=
      tbParam5.Position / 100;

    Sound.SetEffectParams(SFX_STANDARD_DISTORTION,
      @FParamsDistortion);
  end;

SFX_STANDARD_ECHO:
  begin
    FParamsEcho.fFeedback   := tbParam1.Position / 100;
    FParamsEcho.fLeftDelay  := tbParam2.Position / 100;
    FParamsEcho.fRightDelay := tbParam3.Position / 100;
    FParamsEcho.fWetDryMix  := tbParam4.Position / 100;
    FParamsEcho.lPanDelay   := trunc(tbParam5.Position / 100);
```

```
    Sound.SetEffectParams(SFX_STANDARD_ECHO, @FParamsEcho);
  end;

SFX_STANDARD_FLANGER:
  begin
    FParamsFlanger.fDelay     := tbParam1.Position / 100;
    FParamsFlanger.fDepth     := tbParam2.Position / 100;
    FParamsFlanger.fFeedback  := tbParam3.Position / 100;
    FParamsFlanger.fFrequency := tbParam4.Position / 100;
    FParamsFlanger.fWetDryMix := tbParam5.Position / 100;
    FParamsFlanger.lPhase     := trunc(tbParam6.Position / 100);

    if rbTriangle.Checked then
       FParamsFlanger.lWaveform := DSFXFLANGER_WAVE_TRIANGLE
    else if rbSine.Checked then
       FParamsFlanger.lWaveform := DSFXFLANGER_WAVE_SIN;

    if rbm180.Checked then
       FParamsFlanger.lPhase := DSFXFLANGER_PHASE_NEG_180

    else if rbm90.Checked then
       FParamsFlanger.lPhase := DSFXFLANGER_PHASE_NEG_90
    else if rbZero.Checked then
       FParamsFlanger.lPhase := DSFXFLANGER_PHASE_ZERO
    else if rb90.Checked then
       FParamsFlanger.lPhase := DSFXFLANGER_PHASE_90
    else if rb180.Checked then
       FParamsFlanger.lPhase := DSFXFLANGER_PHASE_180;

    Sound.SetEffectParams(SFX_STANDARD_FLANGER,
      @FParamsFlanger);
  end;

SFX_STANDARD_GARGLE:
  begin
    FParamsGargle.dwRateHz    := trunc(tbParam1.Position / 100);
```

```
      if rbTriangle.Checked then
        FParamsGargle.dwWaveShape := DSFXGARGLE_WAVE_TRIANGLE
      else if rbSquare.Checked then
        FParamsGargle.dwWaveShape := DSFXGARGLE_WAVE_SQUARE;

      Sound.SetEffectParams(SFX_STANDARD_GARGLE, @FParamsGargle);
    end;

  SFX_STANDARD_PARAMEQ:
    begin
      FParamsParamEq.fBandwidth := tbParam1.Position / 100;
      FParamsParamEq.fCenter    := tbParam2.Position / 100;
      FParamsParamEq.fGain      := tbParam3.Position / 100;
      Sound.SetEffectParams(SFX_STANDARD_PARAMEQ,
        @FParamsParamEq);
    end;

  SFX_WAVES_REVERB:
    begin
      FParamsReverb.fHighFreqRTRatio := tbParam1.Position / 100;
      FParamsReverb.fInGain          := tbParam2.Position / 100;
      FParamsReverb.fReverbMix       := tbParam3.Position / 100;
      FParamsReverb.fReverbTime      := tbParam4.Position / 100;
      Sound.SetEffectParams(SFX_WAVES_REVERB, @FParamsReverb);
    end;
end;

lbValue1.Caption := FloatToStrF(tbParam1.Position / 100,
  ffGeneral, 7, 2);
lbValue2.Caption := FloatToStrF(tbParam2.Position / 100,
  ffGeneral, 7, 2);
lbValue3.Caption := FloatToStrF(tbParam3.Position / 100,
  ffGeneral, 7, 2);
lbValue4.Caption := FloatToStrF(tbParam4.Position / 100,
  ffGeneral, 7, 2);
```

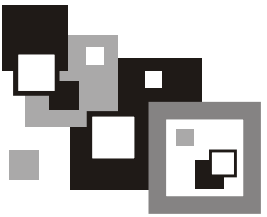
```
  lbValue5.Caption := FloatToStrF(tbParam5.Position / 100,
    ffGeneral, 7, 2);
  lbValue6.Caption := FloatToStrF(tbParam6.Position / 100,
    ffGeneral, 7, 2);
end;

{*****************************************************************}
{** Запоминаем текущий звуковой эффект                          **}
{*****************************************************************}
procedure TMainForm.checkListBoxEffectsClick(Sender: TObject);
begin
  CurrentEffect := 0;
  if not checkListBoxEffects.Checked[checkListBoxEffects.ItemIndex]
  then EXIT;

  case checkListBoxEffects.ItemIndex of
    0: CurrentEffect := SFX_STANDARD_CHORUS;
    1: CurrentEffect := SFX_STANDARD_COMPRESSOR;
    2: CurrentEffect := SFX_STANDARD_DISTORTION;
    3: CurrentEffect := SFX_STANDARD_ECHO;
    4: CurrentEffect := SFX_STANDARD_FLANGER;
    5: CurrentEffect := SFX_STANDARD_GARGLE;
    6: CurrentEffect := SFX_STANDARD_PARAMEQ;
    7: CurrentEffect := SFX_WAVES_REVERB;
  end;

  RefreshEffectParams;
end;

{*****************************************************************}
{** Установка параметров звукового эффекта (по номеру эффекта)  **}
{*****************************************************************}
procedure TMainForm.SetParamValues(Index: integer; Value: Single;
  minValue, maxValue: Single; ParamName: string; IsEnabled: boolean);
var
  tbParam: TTrackBar;
```

```
  lbParamName,
  lbValue: TLabel;
  stMinValue,
  stMaxValue: TStaticText;
begin
  tbParam := FindComponent(Format('tbParam%d', [Index])) as
    TTrackBar;

  lbValue := FindComponent(Format('lbValue%d', [Index])) as TLabel;
  lbParamName := FindComponent(Format('lbParamName%d', [Index])) as
    TLabel;

  stMinValue := FindComponent(Format('stMinValue%d', [Index])) as
    TStaticText;
  stMaxValue := FindComponent(Format('stMaxValue%d', [Index])) as
    TStaticText;

  if (tbParam = NIL) or (lbParamName = NIL) or (lbValue = NIL) or
     (stMinValue = NIL) or (stMaxValue = NIL) then EXIT;

  tbParam.Min := trunc(minValue * 100);
  tbParam.Max := trunc(maxValue * 100);
  tbParam.Position := trunc(Value * 100);

  lbParamName.Caption := ParamName;
  stMinValue.Caption  := FloatToStrF(minValue, ffGeneral, 7, 2);
  stMaxValue.Caption  := FloatToStrF(maxValue, ffGeneral, 7, 2);
  lbValue.Caption     := FloatToStrF(Value, ffGeneral, 7, 2);

  tbParam.Enabled     := IsEnabled;
  lbParamName.Enabled := IsEnabled;
  lbValue.Enabled     := IsEnabled;
  stMinValue.Enabled  := IsEnabled;
  stMaxValue.Enabled  := IsEnabled;
end;
```

```
{*****************************************************************}
{** Установка параметра фазы звукового эффекта                  **}
{*****************************************************************}
procedure TMainForm.SetPhaseParams(Index: integer; IsEnabled: boolean);
begin
  groupBoxPhase.Enabled := IsEnabled;
  if Index >= 0 then
  begin
    case Index of
      0: rbm180.Checked := TRUE;
      1: rbm90.Checked := TRUE;
      2: rbZero.Checked := TRUE;
      3: rb90.Checked := TRUE;
      4: rb180.Checked := TRUE;
    end;
  end;
  rbm180.Enabled := IsEnabled;
  rbm90.Enabled  := IsEnabled;
  rbZero.Enabled := IsEnabled;
  rb90.Enabled   := IsEnabled;
  rb180.Enabled  := IsEnabled;
end;

{*****************************************************************}
{** Установка параметра формы сигнала звукового эффекта         **}
{*****************************************************************}
procedure TMainForm.SetWaveformParams(Index: integer; IsParam1Enabled,
  IsParam2Enabled, IsParam3Enabled, IsEnabled: boolean);
begin
  groupBoxWaveform.Enabled := IsEnabled;
  if Index >= 0 then
  begin
    case Index of
      0: rbTriangle.Checked := TRUE;
```

```
      1: rbSquare.Checked := TRUE;
      2: rbSine.Checked := TRUE;
    end;
  end;
  rbTriangle.Enabled := IsParam1Enabled;
  rbSquare.Enabled := IsParam2Enabled;
  rbSine.Enabled := IsParam3Enabled;
end;

END.
```

Глава 9



Захват звука

Интерфейсы

Помимо воспроизведения звука, в подсистеме DirectSound существует возможность его записи (захвата). Для этого предусмотрен интерфейс `IDirectSoundCapture8`. Он содержит три метода:

- `CreateCaptureBuffer` — создание буфера захвата;
- `GetCaps` — получение возможностей системы захвата звука;
- `Initialize` — инициализация.

Создание объекта с данным интерфейсом производится вызовом метода `DirectSoundCaptureCreate8`:

```
function DirectSoundCaptureCreate8(
  pcGuidDevice: PGUID;
  out ppDSC8: IDirectSoundCapture8;
  pUnkOuter: IUnknown):
HResult; stdcall; external DirectSoundDLL;
```

Здесь:

- `pcGuidDevice` — адрес уникального идентификатора устройства или нулевое значение для выбора стандартного;
- `ppDSC8` — адрес указателя, в который будет занесен интерфейс создаваемого объекта;
- `pUnkOuter` — указатель на объект с интерфейсом `IUnknown`, должен быть нулевым.

Получить полный список всех доступных устройств захвата можно при помощи метода `DirectSoundCaptureEnumerate`:

```
function DirectSoundCaptureEnumerate(
  lpDSEnumCallback: TDSEnumCallback;
```

```
  lpContext: Pointer):
HResult; stdcall;
external DirectSoundDLL name 'DirectSoundCaptureEnumerateA';
```

Здесь:

- `lpDSEnumCallback` — адрес функции перебора, которая будет вызываться для кажого обнаруженного устройства захвата;
- `lpContext` — адрес произвольного значения, которое будет передаваться функции перебора.

Буфер захвата

Следующий шаг — создание буфера захвата. *Буфер захвата* — объект с интерфейсом `IDirectSoundCaptureBuffer8`. Методы данного интерфейса представлены далее (в том числе и методы, унаследованные от интерфейса `IDirectSoundCaptureBuffer`).

- `GetObjectInPath` — получение ссылки на объект, связанный с буфером;
- `GetFXStatus` — получение статуса эффектов при захвате звука;
- `GetCaps` — получение параметров буфера захвата;
- `GetCurrentPosition` — получение текущих позиций в буфере;
- `GetFormat` — получение формата аудиоданных буфера;
- `GetStatus` — получение статуса буфера захвата;
- `Initialize` — инициализация;
- `Lock` — блокирование участка буфера для операций чтения или записи;
- `Start` — начало захвата данных в буфер;
- `Stop` — останов процесса захвата;
- `Unlock` — разблокировка буфера.

Создание буфера захвата производится вызовом метода `IDirectSoundCapture8.CreateCaptureBuffer`:

```
function CreateCaptureBuffer(
  const pcDSCBufferDesc: TDSCBufferDesc;
  out ppDSCBuffer: IDirectSoundCaptureBuffer;
  pUnkOuter: IUnknown):
HResult; stdcall;
```

Здесь:

- `pcDSCBufferDesc` — адрес структуры `TDSCBufferDesc`, описывающей буфер;
- `ppDSCBuffer` — адрес переменной, которая получает интерфейс `IDirectSoundCaptureBuffer` созданного объекта буфера;
- `pUnkOuter` — указатель на объект с интерфейсом `IUnknown`, должен быть нулевым.

Рассмотрим структуру `TDSCBufferDesc`. Она практически полностью идентична структуре `TDSBufferDesc`:

- `dwSize` — размер структуры в байтах;
- `dwFlags` — флаги, определяющие режимы работы буфера;
- `dwBufferBytes` — размер буфера захвата в байтах;
- `dwReserved` — зарезервированное поле;
- `lpwfxFormat` — адрес структуры типа `TWaveFormatEx`, описывающей формат данных буфера. Для первичного буфера имеет нулевое значение;
- `dwFXCount` — число элементов массива `lpDSCFXDesc`. Поле должно иметь нулевое значение, если флаг `DSCBCAPS_CTRLFX` не установлен;
- `lpDSCFXDesc` — ссылка на массив элементов типа `TDSCEffectDesc`, описывающих поддерживаемые аппаратные звуковые эффекты.

Звуковые эффекты описываются структурой `TDSCEffectDesc`:

- `dwSize` — размер структуры в байтах;
- `dwFlags` — флаги, определяющие параметры эффекта:
  - `DSCFX_LOCHARDWARE` — эффект, определенный параметром `guidDSCFXInstance`, должен быть аппаратным;
  - `DSCFX_LOCSOFTWARE` — эффект, определенный параметром `guidDSCFXInstance`, должен быть программным.

  В результате вызова функции данный флаг может принимать одно из следующих значений:
  - `DSCFXR_LOCHARDWARE` — эффект создан аппаратно;
  - `DSCFXR_LOCSOFTWARE` — эффект создан программно;
- `guidDSCFXClass` — идентификатор класса эффекта. Может принимать одно из следующих значений:
  - `GUID_DSCFX_CLASS_AEC` — подавление акустического эха;
  - `GUID_DSCFX_CLASS_NS` — подавление шума;

- guidDSCFXInstance — уникальный идентификатор эффекта. Принимает одно из следующих значений:
  - GUID_DSCFX_MS_AEC — подавление эха (эффект от компании Microsoft). Доступен только программно;
  - GUID_DSCFX_MS_NS — подавление шума (эффект от компании Microsoft). Доступен только программно;
  - GUID_DSCFX_SYSTEM_AEC — системное подавление эха;
  - GUID_DSCFX_SYSTEM_NS — системное подавление шума;
- dwReserved1, dwReserved2 — зарезервированные поля. Должны принимать нулевое значение.

После создания буфера захвата мы получаем объект, который поддерживает интерфейс IDirectSoundCaptureBuffer. Для получения интерфейса IDirectSoundCaptureBuffer8 следует вызвать метод IDirectSoundCaptureBuffer.QueryInterface с параметром IID_IDirectSoundCaptureBuffer8. Затем необходимо получить интерфейс уведомлений IDirectSoundNotify8. Это делается при помощи метода IDirectSoundCaptureBuffer.QueryInterface с параметром IID_IDirectSoundNotify8. Следом мы создаем объект "событие" (CreateEvent) и устанавливаем позицию уведомления IDirectSoundNotify8.SetNotificationPositions.

Захват аудио

Теперь, произведя все настройки, мы можем приступать непосредственно к захвату аудиоданных. Захват начинается после вызова метода IDirectSoundCapture8.Start:

```
function Start(
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь dwFlags — флаг, определяющий поведение буфера захвата: DSCBSTART_LOOPING — по достижении конца буфера запись возобновляется сначала и так до тех пор, пока процесс захвата не будет остановлен.

Ждем окончания записи при помощи API-функции WaitForSingleObject, параметром которой является тот самый созданный нами ранее объект "событие".

Получить записанные в буфер данные можно, вызвав метод `IDirectSoundCapture8.Lock`, возвращающий указатели на доступные заблокированные участки буфера, содержащие данные:

```
function Lock(
  dwOffset, dwBytes: DWORD;
  ppvAudioPtr1: PPointer;
  pdwAudioBytes1: PDWORD;
  ppvAudioPtr2: PPointer;
  pdwAudioBytes2: PDWORD;
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `dwOffset` — смещение относительно начала буфера;
- `dwBytes` — размер участка буфера для записи в байтах;
- `ppvAudioPtr1`, `ppvAudioPtr2` — адреса указателей на область памяти, в которые будут записаны указатели частей полученного участка буфера. Если `ppvAudioPtr2` возвращает нулевой результат, то это значит, что получен непрерывный участок буфера;
- `pdwAudioBytes1`, `pdwAudioBytes2` — адреса переменных, в которые будет записан размер первого и второго участка буфера в байтах;
- `dwFlags` — флаги, влияющие на модификацию буфера:
  - `DSBLOCK_ENTIREBUFFER` — запрашивается доступ ко всему доступному для записи участку. Параметр `dwBytes` игнорируется.

Получив данные из буфера, необходимо вызвать метод `IDirectSoundCapture8.Unlock`:

```
function Unlock(
  pvAudioPtr1: Pointer;
  dwAudioBytes1: DWORD;
  pvAudioPtr2: Pointer;
  dwAudioBytes2: DWORD):
HResult; stdcall;
```

Здесь:

- `pvAudioPtr1`, `pvAudioPtr2` — указатели на области памяти, полученные при вызове метода `IDirectSoundCapture8.Lock`;
- `dwAudioBytes1`, `dwAudioBytes2` — количество байтов, которые были прочитаны из данных участков на самом деле.

Для удобной работы с захватом аудиоданных мной разработан класс `TdxSoundCapture`, который расположен в каталоге Classes на прилагаемом к книге компакт-диске.

Класс *TdxSoundCapture*

Данный класс предназначен для упрощения действий по захвату звука. Помимо конструктора и деструктора, класс содержит всего два метода. Это метод инициализации:

```
function Initialize: HResult;
```

и метод захвата звука:

```
function StartCapture(
  WAVFile: string;
  CaptureTime: DWORD;
  Channels: WORD = 2;
  SamplesPerSec: DWORD = 11025;
  BitsPerSample: WORD = 16):
HResult;
```

Здесь:

- `WAVFile` — путь к WAV-файлу;
- `CaptureTime` — время захвата в секундах;
- `Channels` — число каналов;
- `SamplesPerSec` — частота дискретизации;
- `BitsPerSample` — битовое разрешение выборки.

Одной из особенностей данного класса является способ работы с файлами формата WAV — не используются функции `mmio`*`XXX`* модуля MMSystem.pas — запись осуществляется напрямую.

Полный текст модуля представлен в листинге 9.1.

Листинг 9.1. Текст модуля UdxSoundCapture.pas

```
UNIT UdxSoundCapture;
{*****************************************************************}
{** Захват звука                                                **}
{** Автор: Есенин Сергей Анатольевич                            **}
{*****************************************************************}
```

```
{**} INTERFACE {*****************************************************}

{**} USES {**********************************************************}
  Windows, Classes, SysUtils, MMSystem, DirectSound, Dialogs;

{**} TYPE {**********************************************************}
  TdxSoundCapture = class
  PRIVATE
    FDirectSoundCapture: IDirectSoundCapture8;
    FHandle: THandle;

  PUBLIC
    constructor Create(AHandle: THandle);
    destructor Destroy; override;

    function Initialize: HResult;

    function StartCapture(WAVFile: string; CaptureTime: DWORD;
      Channels: WORD = 2; SamplesPerSec: DWORD = 11025;
      BitsPerSample: WORD = 16): HResult;
  END;

{**} IMPLEMENTATION {************************************************}

{**} {TdxSoundCapture} {*********************************************}

{*****************************************************************}
{** Конструктор класса                                          **}
{*****************************************************************}
constructor TdxSoundCapture.Create(AHandle: THandle);
begin

  // Запоминаем указатель на главную форму
  FHandle := AHandle;
```

```
  // Обнуляем ссылку на объект DirectSoundCapture
  FDirectSoundCapture := NIL;
end;

{*********************************************************************}
{** Деструктор класса                                               **}
{*********************************************************************}
destructor TdxSoundCapture.Destroy;
begin
  // Обнуляем ссылку на объект DirectSoundCapture
  FDirectSoundCapture := NIL;
end;

{*********************************************************************}
{** Инициализация                                                   **}
{*********************************************************************}
function TdxSoundCapture.Initialize: HResult;
begin
  // Инициализируем подсистему захвата звука
  result := DirectSoundCaptureCreate8(
    NIL,
    FDirectSoundCapture,
    NIL);
end;

{*********************************************************************}
{** Захват аудио                                                    **}
{*********************************************************************}
function TdxSoundCapture.StartCapture(WAVFile: string;
  CaptureTime: DWORD; Channels: WORD; SamplesPerSec: DWORD;
  BitsPerSample: WORD): HResult;
type
  // Структура, описывающая заголовок WAV-файла
  TWAVHeader = packed record
    wav_riff_id:   array[0..3] of char;
    wav_riff_len:  DWORD;
```

```
    wav_chuck_id:  array[0..3] of char;
    wav_fmt:       array[0..3] of char;
    wav_chuck_len: DWORD;
    wav_type:      WORD;
    wav_channels:  WORD;
    wav_freq:      DWORD;
    wav_bytes:     DWORD;
    wav_align:     WORD;
    wav_bits:      WORD;

    wav_data_id:   array[0..3] of char;
    wav_data_len:  DWORD;
  end;

var
  FDSTmpBuffer:   IDirectSoundCaptureBuffer;
  FCaptureBuffer: IDirectSoundCaptureBuffer8;
  FCaptureNotify: IDirectSoundNotify8;
  dsbd:           TDSCBufferDesc;
  wfx:            TWaveFormatEx;
  wh:             TWAVHeader;
  pn:             TDSBPositionNotify;
  AudioPtr:       Pointer;
  AudioBytes:     DWORD;
  Data:           PByte;
  WAVReader:      TFileStream;

begin
  // Заполняем структуру, описывающую WAV-формат
  ZeroMemory(@wfx, sizeof(wfx));
  wfx.wFormatTag      := WAVE_FORMAT_PCM;
  wfx.nChannels       := Channels;
  wfx.nSamplesPerSec  := SamplesPerSec;
  wfx.wBitsPerSample  := BitsPerSample;
  wfx.nBlockAlign     := wfx.wBitsPerSample div 8 *
```

```
  wfx.nChannels;
wfx.nAvgBytesPerSec := wfx.nSamplesPerSec * wfx.nBlockAlign;

// Заполняем структуру, описывающую буфер захвата
ZeroMemory(@dsbd, sizeof(dsbd));
dsbd.dwSize        := sizeof(dsbd);
dsbd.lpwfxFormat   := @wfx;
dsbd.dwBufferBytes := dsbd.lpwfxFormat.nAvgBytesPerSec *
  CaptureTime;

// Создаем временный буфер
result := FDirectSoundCapture.CreateCaptureBuffer(
  dsbd,
  FDSTmpBuffer,
  NIL);
if FAILED(result) then EXIT;

// Получаем интерфейс IDirectSoundCaptureBuffer8 для буфера
// захвата
result := FDSTmpBuffer.QueryInterface(
  IID_IDirectSoundCaptureBuffer8,
  FCaptureBuffer);
if FAILED(result) then EXIT;

// Обнуляем временный буфер
FDSTmpBuffer := NIL;

// Получаем интерфейс уведомлений для буфера захвата
result := FCaptureBuffer.QueryInterface(
  IID_IDirectSoundNotify8,
  FCaptureNotify);
if FAILED(result) then EXIT;

// Создаем событие для ожидания окончания записи
pn.dwOffset := DSBPN_OFFSETSTOP;
pn.hEventNotify := CreateEvent(NIL, FALSE, FALSE, NIL);
```

```
// Устанавливаем позицию уведомления (как только будет
// достигнута данная позиция, сразу сработает связанное с
// ним событие)
FCaptureNotify.SetNotificationPositions(1, @pn);
FCaptureNotify := NIL;
// Начинаем захват аудио
FCaptureBuffer.Start(0);

// Дожидаемся окончания захвата
WaitForSingleObject(pn.hEventNotify, INFINITE);

// Удаляем событие
CloseHandle(pn.hEventNotify);

// Блокируем буфер и получаем указатель на заблокированный
// блок данных
FCaptureBuffer.Lock(0, 0, @AudioPtr, @AudioBytes, NIL, NIL,
  DSCBLOCK_ENTIREBUFFER);

// Заполняем заголовок WAV-файла
wh.wav_align     := wfx.nBlockAlign;
wh.wav_bits      := wfx.wBitsPerSample;
wh.wav_bytes     := wfx.nAvgBytesPerSec;
wh.wav_channels  := wfx.nChannels;
wh.wav_chuck_id  := 'WAVE';
wh.wav_chuck_len := 16;
wh.wav_data_id   := 'data';
wh.wav_data_len  := AudioBytes;
wh.wav_fmt       := 'fmt ';
wh.wav_freq      := wfx.nSamplesPerSec;
wh.wav_riff_id   := 'RIFF';
wh.wav_riff_len  := wh.wav_data_len + sizeof(wh);
wh.wav_type      := wfx.wFormatTag;

// Выделяем память под аудиоданные
Data := GetMemory(AudioBytes);
```

```
  try
    // Копируем данные из буфера захвата
    CopyMemory(Data, AudioPtr, AudioBytes);

    // Разблокировка буфера
    FCaptureBuffer.Unlock(AudioPtr, AudioBytes, NIL, 0);
    // Создаем WAV-файл
    WAVReader := TFileStream.Create(WAVFile, fmCreate);
    try
      // Записываем в файл заголовок
      WAVReader.WriteBuffer(wh, sizeof(wh));

      // Записываем в файл аудиоданные
      WAVReader.WriteBuffer(Data^, AudioBytes);
    finally
      // Завершаем работу с WAV-файлом
      FreeAndNil(WAVReader);
    end;
  finally
    // Обнуляем буфер захвата
    FCaptureBuffer := NIL;
    // Освобождаем память, выделенную под аудиоданные
    FreeMemory(Data)
  end;

  result := S_OK;
end;

END.
```

Пример использования класса *TdxSoundCapture*

Как и для предыдущего класса, на компакт-диске в папке Examples\DirectSound\DS_CAP_TEST находится пример использования класса `TdxSoundCapture`. Пример состоит из одной главной формы, на которой

располагаются следующие элементы управления: окно для ввода имени сохраняемого файла, поле ввода времени записи, список количества каналов, список частот дискретизации, список количества битов и кнопка начала записи. Общий вид приложения можно увидеть на рис. 9.1.

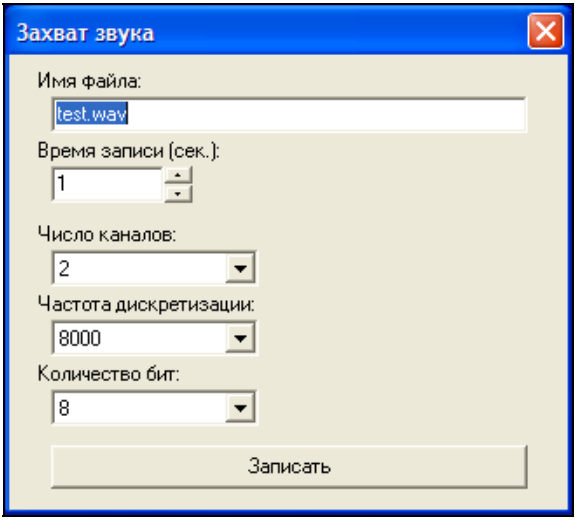


Рис. 9.1. Общий вид тестирующего приложения для класса `TdxSoundCapture`

Код главного модуля тестирующего приложения для класса приведен в листинге 9.2.

Листинг 9.2. Текст модуля FormMain.pas проекта DS_Capture

```
UNIT FormMain;
{*************************************************************************}
{** Тестирование класса TdxSoundCapture                                 **}
{** Автор: Есенин Сергей Анатольевич                                    **}
{*************************************************************************}

{**} INTERFACE {*********************************************************}

{**} USES {**************************************************************}
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
```

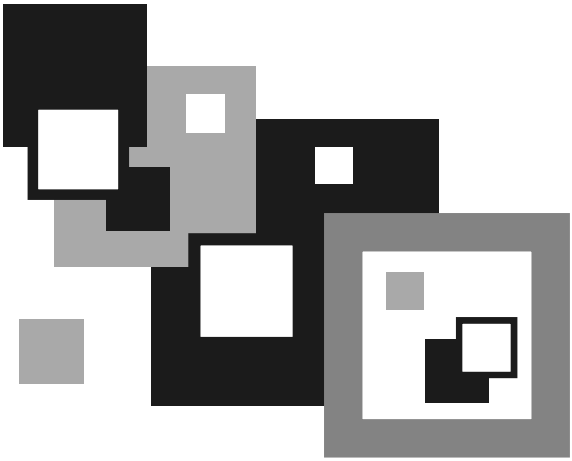
```
  Controls, Forms, Dialogs, StdCtrls, ComCtrls,
  dxSoundCapture;

{**} TYPE {*************************************************************}
  TMainForm = class(TForm)
    editFileName: TEdit;
    labelFileName: TLabel;
    labelCaptureTime: TLabel;
    editCaptureTime: TEdit;
    upDownCaptureTime: TUpDown;
    buttonCapture: TButton;
    labelChannels: TLabel;
    labelFrequency: TLabel;
    comboBoxFrequency: TComboBox;
    labelBits: TLabel;
    comboBoxBits: TComboBox;
    comboBoxChannels: TComboBox;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure buttonCaptureClick(Sender: TObject);
  PRIVATE
    { Private declarations }
  PUBLIC
    { Public declarations }
  END;

{**} VAR {**************************************************************}
  MainForm: TMainForm;
  SoundCapture: TdxSoundCapture;

{**} IMPLEMENTATION {***************************************************}

{$R *.dfm}

{*************************************************************************}
{** Создание и инициализация объекта SoundCapture                      **}
{*************************************************************************}
```

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  SoundCapture := TdxSoundCapture.Create(Handle);
  if FAILED(SoundCapture.Initialize) then
  begin
    buttonCapture.Enabled := FALSE;
    FreeAndNil(SoundCapture);
  end else
  begin
    comboBoxFrequency.ItemIndex := 0;
    comboBoxBits.ItemIndex := 0;
    comboBoxChannels.ItemIndex := 1;
  end;
end;

{************************************************************************}
{** Удаление объекта SoundCapture                                      **}
{************************************************************************}
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  if SoundCapture <> NIL then
  begin
     FreeAndNil(SoundCapture);
  end;
end;

{************************************************************************}
{** Захват звука                                                       **}
{************************************************************************}
procedure TMainForm.buttonCaptureClick(Sender: TObject);
var
  WAVFile: string;
  CaptureTime: DWORD;
  Channels: WORD;
  SamplesPerSec: DWORD;
  BitsPerSample: WORD;
```

```
begin
  buttonCapture.Enabled := FALSE;

  try
    WAVFile       := ChangeFileExt(editFileName.Text, '.wav');
    CaptureTime   := upDownCaptureTime.Position;
    Channels      := StrToInt(comboBoxChannels.Text);
    SamplesPerSec := StrToInt(comboBoxFrequency.Text);
    BitsPerSample := StrToInt(comboBoxBits.Text);

    if FAILED(SoundCapture.StartCapture(WAVFile, CaptureTime,
         Channels, SamplesPerSec, BitsPerSample)) then
       ShowMessage('Запись невозможна!');
  finally
    buttonCapture.Enabled := TRUE;
  end;
end;

END.
```

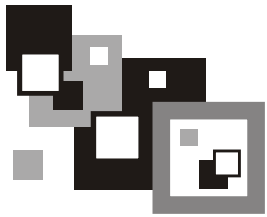
Итоги

На этом мы с вами заканчиваем изучение подсистемы DirectSound. Нами было изучено устройство данной подсистемы, ее возможности, достоинства и недостатки. Были рассмотрены три класса: `TdxSound` — класс, являющийся по сути представлением вторичного буфера или, иными словами, источником звука, `TdxSoundManager` — менеджер вторичных буферов и `TdxSoundCapture` — для захвата звука. Данные классы содержат в себе минимальную функциональность — только базовые действия. Это связано с тем, что для того чтобы реализовать все возможности DirectSound в данных классах, пришлось бы фактически использовать все API DirectSound. Поэтому доработка до необходимой функциональности предоставляется самим читателям.



ЧАСТЬ IV

DirectMusic



Глава 10

Работа с MIDI и WAV-файлами

Интерфейсы

В состав подсистемы DirectMusic входит достаточно большое количество интерфейсов. Все они имеют префикс `IDirectMusic` (как пример интерфейсы `IDirectMusicPort8`, `IDirectMusicScript8` и т. п.). Нам же для работы будет достаточно рассмотреть следующие интерфейсы:

- `IDirectMusic8` — самый главный интерфейс подсистемы DirectMusic. Он предназначен для управления такими составляющими DirectMusic, как буферы, порты и объекты часов. Может не использоваться в приложении в явном виде (т. е. работа с DirectMusic будет осуществляться посредством других интерфейсов более высокого уровня);
- `IDirectMusicPerformance8` — интерфейс по своей сути является менеджером управления сегментами;
- `IDirectMusicLoader8` — предназначен для поиска, перечисления, кэширования и загрузки объектов. В нашем случае он будет использован для загрузки звуковых данных;
- `IDirectMusicSegment8` — представляет собой сегмент аудиоданных.

В отличие от подсистемы DirectSound, в подсистеме DirectMusic нет функции создания объекта DirectMusic посредством одного вызова (`DirectSoundCreate` для DirectSound). Поэтому создавать объекты интерфейсов `IDirectMusicLoader8` и `IDirectMusicPerformance8` необходимо вызовом функции `CoCreateInstance`, предварительно не забыв проинициализировать библиотеку COM вызовом метода `CoInitializeEx` (и не забыв вызвать метод `CoUninitialize` в конце работы).

Рассмотрим некоторые методы интерфейса менеджера воспроизведения `IDirectMusicPerformance8` подробнее.

Инициализация:

```
function InitAudio(
  ppDirectMusic: PIDirectMusic;
  ppDirectSound: PIDirectSound;
  hWnd: hWnd;
  dwDefaultPathType,
  dwPChannelCount,
  dwFlags: DWORD;
  pParams: PDMUSAudioParams):
HResult; stdcall;
```

Здесь:

- `ppDirectMusic` — адрес переменной, в которую в результате работы метода будет записан указатель на объект с интерфейсом `IDirectMusic` (мы не будем использовать интерфейс `IDirectMusic` и укажем `NIL` в качестве этого параметра);
- `ppDirectSound` — адрес переменной, в которую в результате работы метода будет записан указатель на объект с интерфейсом `IDirectSound` (указав `NIL` в качестве параметра, мы просто не будем иметь доступа с созданному объекту, но, тем не менее, он будет создан);
- `hWnd` — указатель на форму приложения, используемую для создания объекта DirectSound. Может быть нулевым;
- `dwDefaultPathType` — значение задает аудиопуть по умолчанию;
- `dwPChannelCount` — задает число каналов воспроизведения (если значение `dwDefaultPathType` отлично от нулевого);
- `dwFlags` — флаги, определяющие режим работы менеджера воспроизведения:
  - `DMUS_AUDIOF_3D` — использовать 3D-буфер (этот флаг не реализован в DirectMusic);
  - `DMUS_AUDIOF_ALL` — использовать все возможности;
  - `DMUS_AUDIOF_BUFFERS` — использовать несколько буферов;
  - `DMUS_AUDIOF_DMOS` — использование дополнительных возможностей (флаг не реализован);
  - `DMUS_AUDIOF_ENVIRON` — моделирование окружающей среды (флаг не реализован);

- DMUS_AUDIOF_EAX — эффект EAX (флаг не реализован);
- DMUS_AUDIOF_STREAMING — поддержка потокового аудио;

- pParams — адрес структуры типа TDMUSAudioParams, которая определяет параметры синтезатора и в которую будут записаны текущие установленные параметры. Значение может быть нулевым в случае использования параметров по умолчанию.

Воспроизведение сегмента:

```
function PlaySegmentEx(
  pSource: IUnknown;
  pwzSegmentName: PWideChar;
  pTransition: IUnknown;
  dwFlags: DWORD;
  i64StartTime: Int64;
  ppSegmentState: PIDirectMusicSegmentState;
  pFrom,
  pAudioPath: IUnknown):
HResult; stdcall;
```

Здесь:

- pSource — сегмент воспроизведения;
- pwzSegmentName — не используется. Должно быть нулевое значение;
- pTransition — указатель на интерфейс шаблона сегмента, используемого для настройки перехода на этот сегмент. Может принимать нулевое значение;
- dwFlags — флаги, определяющие поведение данного метода, параметр может принимать и нулевое значение;
- i64StartTime — время начала воспроизведения сегмента. При использовании нулевого значения воспроизведение начнется сразу, как только это станет возможным;
- ppSegmentState — адрес переменной, в которую будет занесен объект с интерфейсом IDirectMusicSegmentState, предназначенным для управления состоянием текущего проигрываемого сегмента. Значение может быть нулевым;
- pFrom — указатель на интерфейс состояния сегмента или аудиопути для остановки воспроизведения, когда будет начато воспроизведение нового сегмента. Может принимать нулевое значение;
- pAudioPath — аудиопуть. Может использоваться нулевое значение.

Остановка воспроизведения:

```
function Stop(
  pSegment: IDirectMusicSegment;
  pSegmentState: IDirectMusicSegmentState;
  mtTime: TMusicTime;
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `pSegment` — сегмент, воспроизведение которого необходимо остановить;
- `pSegmentState` — объект состояния сегмента;
- `mtTime` — время, в течение которого воспроизведение должно быть остановлено;
- `dwFlags` — флаги, определяющие момент остановки.

Если параметры `pSegment` и `pSegmentState` имеют значение `NIL` одновременно, то будет остановлено звучание всех сегментов, находящихся в менеджере воспроизведения.

Метод `IDirectMusicPerformance8.Stop` заменен в настоящее время более новым методом `IDirectMusicPerformance8.StopEx`, который может останавливать воспроизведение сегмента, объекта состояния сегмента и аудиопути:

```
function StopEx(
  pObjectToStop: IUnknown;
  i64StopTime: int64;
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `pObjectToStop` — указатель на объект сегмента, состояния сегмента или аудиопути;
- `i64StopTime` — время, в течение которого воспроизведение должно быть остановлено. При нулевом значении воспроизведение будет остановлено немедленно;
- `dwFlags` — флаги, определяющие момент остановки:
  - `DMUS_SEGF_AUTOTRANSITION` — не реализован;
  - `DMUS_SEGF_BEAT` — остановка на следующей границе такта или после истечения времени, заданного параметром `i64StopTime`;

- `DMUS_SEGF_DEFAULT` — остановка на границе по умолчанию, заданной вызовом метода `IDirectMusicSegment8.SetDefaultResolution`;
- `DMUS_SEGF_GRID` — остановка на следующей границе сетки или после истечения времени, заданного параметром `i64StopTime`;
- `DMUS_SEGF_MEASURE` — остановка на следующей границе измерений или после истечения времени, заданного параметром `i64StopTime`;
- `DMUS_SEGF_REFTIME` — значение `i64StopTime` является ссылочным временем;
- `DMUS_SEGF_SEGMENTEND` — остановка в конце основного сегмента;
- `DMUS_SEGF_MARKER` — остановка на следующем маркере.

Значение параметра может быть нулевым. Флаг `DMUS_SEGF_REFTIME` может быть объединен с любым другим флагом.

Проверить, воспроизводится ли сегмент в настоящее время или нет, можно с помощью следующего метода:

```
function IsPlaying(
  pSegment: IDirectMusicSegment;
  pSegState: IDirectMusicSegmentState):
HResult; stdcall;
```

Здесь:

- ❐ `pSegment` — проверяемый сегмент. При нулевом значении проверяется `pSegState`;
- ❐ `pSegState` — проверяемый объект состояния сегмента. При нулевом значении проверяется `pSegment`.

В интерфейсе `IDirectMusicLoader8` нам интересен всего один метод:

```
function LoadObjectFromFile(
  const rguidClassID: TGUID;
  const iidInterfaceID: TGUID;
  pwzFilePath: PWideChar;
  out ppObject):
HResult; stdcall;
```

Здесь:

- ❐ `rguidClassID` — уникальный идентификатор класса объекта (в нашем случае будет использоваться `CLSID_DirectMusicSegment`);
- ❐ `iidInterfaceID` — уникальный идентификатор интерфейса (в нашем случае это `IID_IDirectMusicSegment8`);

- `pwzFilePath` — путь к файлу;
- `ppObject` — адрес переменной, которая получит указатель на нужный нам интерфейс.

Из методов интерфейса `IDirectMusicSegment8` мы рассмотрим три.

Загрузка данных в синтезатор или аудиопуть:

```
function Download(
  pAudioPath: IUnknown):
HResult; stdcall;
```

Здесь `pAudioPath` — аудиопуть для получения данных.

Выгрузка данных:

```
function Unload(
  pAudioPath: IUnknown):
HResult; stdcall;
```

Здесь `pAudioPath` — аудиопуть, из которого будут выгружены данные.

Установка числа циклов воспроизведения сегмента:

```
function SetRepeats(
  dwRepeats: DWORD):
HResult; stdcall;
```

Здесь `dwRepeats` — число циклов повторения воспроизведения. При нулевом значении сегмент будет воспроизведен всего один раз. При значении `DMUS_SEG_REPEAT_INFINITE` воспроизведение будет продолжаться бесконечно, пока не будет остановлено.

Порядок работы

Итак, мы разобрали основные методы необходимых нам интерфейсов. Теперь же для более наглядного представления о порядке действий выпишем их все в хронологическом порядке.

1. Инициализация библиотеки COM:

```
// Инициализация библиотеки COM
CoInitializeEx(NIL, COINIT_MULTITHREADED);
```

2. Создание объектов загрузчика и менеджера воспроизведения:

```
// Создаем загрузчик
Result := CoCreateInstance(CLSID_DirectMusicLoader, NIL,
                           CLSCTX_INPROC,
```

```
                                IID_IDirectMusicLoader8, FLoader);
if FAILED(Result) then EXIT;

// Создаем менеджер воспроизведения
Result := CoCreateInstance(CLSID_DirectMusicPerformance, NIL, \
    CLSCTX_INPROC, IID_IDirectMusicPerformance8, FPerformance);
if FAILED(Result) then EXIT;
```

3. Инициализация менеджера воспроизведения:

```
// Инициализируем менеджер воспроизведения
Result := FPerformance.InitAudio(NIL, NIL, FHandle,
  DMUS_APATH_DYNAMIC_STEREO, 128, DMUS_AUDIOF_ALL, NIL);
```

4. Загрузка данных из файла и получение интерфейса управления сегментом:

```
// Загружаем аудиоданные и получаем указатель на сегмент
if FAILED(FLoader.LoadObjectFromFile(CLSID_DirectMusicSegment,
  IID_IDirectMusicSegment8, PWideChar(FileName),
  FSegment)) then EXIT;
```

5. Передача данных из сегмента менеджеру воспроизведения:

```
// Загружаем данные в менеджер
if FAILED(FSegment.Download(FPerformance)) then EXIT;
```

6. Установка числа циклов воспроизведения сегмента:

```
// Устанавливаем число циклов воспроизведения
Result := FSegment.SetRepeats(dwRepeats);
```

7. Воспроизведение сегмента:

```
if IsPrimary then
  // Воспроизводим основной сегмент
  Result := FPerformance.PlaySegmentEx(FSegment, NIL, NIL, 0,
    0, NIL, NIL, NIL)
else
  // Воспроизводим вторичный сегмент
  Result := FPerformance.PlaySegmentEx(FSegment, NIL, NIL,
    DMUS_SEGF_SECONDARY, 0, NIL, NIL, NIL);
```

8. Завершение воспроизведения:

```
// Останавливаем воспроизведение сегмента
Result := FPerformance.StopEx(FSegment, 0, 0);
```

9. Освобождение памяти, занимаемой сегментом:

```
// Проверяем наличие сегмента
if FSegment <> NIL then
begin
  // Выгружаем данные из сегмента
  FSegment.Unload(FPerformance);

  // Обнуляем сегмент воспроизведения
  FSegment := NIL;
end;
```

10. Освобождение памяти, занимаемой менеджером воспроизведения и загрузчиком:

```
// Обнуление ссылок на загрузчик и менеджер воспроизведения
FLoader := NIL;
FPerformance := NIL;
```

11. Завершение работы с COM:

```
// Завершение работы с COM
CoUninitialize;
```

Классы *TdxMusicSegment* и *TdxMusicManager*

Для упрощения работы с данной подсистемой мною разработаны два класса: классы `TdxMusicSegment` и `TdxMusicManager`, которые описывают музыкальный сегмент и менеджер воспроизведения соответственно. Классы расположены в модуле UdxMusicManager.pas каталога Classes на прилагаемом к книге компакт-диске (листинг 10.1).

Листинг 10.1. Текст модуля UdxMusicManager.pas

```
UNIT UdxMusicManager;
{*************************************************************************}
{** Работа с файлами формата MIDI и WAV посредством DirectMusic        **}
{** Автор: Есенин Сергей Анатольевич                                    **}
{*************************************************************************}
```

```
{**} INTERFACE {*************************************************************}

{**} USES {******************************************************************}
  Windows, Classes, SysUtils, MMSystem, DirectMusic, ComObj, ActiveX;

{**} TYPE {******************************************************************}
  TdxMusicSegment = class;

  TdxMusicManager = class
  PRIVATE
    FPerformance: IDirectMusicPerformance8;
    FLoader: IDirectMusicLoader8;

    FSegmentList: TList;

    FHandle: THandle;

  PUBLIC
    constructor Create(Handle: THandle);
    destructor Destroy; override;

    function Initialize: HResult;

    function CreateSegment(FileName: WideString): TdxMusicSegment;
    function DeleteSegment(Index: integer): HResult; overload;
    function DeleteSegment(Segment: TdxMusicSegment): HResult; overload;
    function GetSegment(Index: integer): TdxMusicSegment;
    function SegmentCount: integer;
  END;

  TdxMusicSegment = class
  PRIVATE
    FPerformance: IDirectMusicPerformance8;
    FLoader: IDirectMusicLoader8;
    FSegment: IDirectMusicSegment8;
```

```
  PUBLIC
    constructor Create(APerformance: IDirectMusicPerformance8;
      ALoader: IDirectMusicLoader8; ASegment: IDirectMusicSegment8);
    destructor Destroy; override;

    function SetRepeats(dwRepeats: DWORD = DMUS_SEG_REPEAT_INFINITE):
      HResult;

    function Play(IsPrimary: boolean = FALSE): HResult;
    function Stop: HResult;

    function IsPlaying: boolean;
  END;

{**} IMPLEMENTATION {************************************************}

{**} { TdxMusicManager } {*******************************************}

{*********************************************************************}
{** Конструктор класса                                              **}
{*********************************************************************}
constructor TdxMusicManager.Create(Handle: THandle);
begin
  // Инициализация библиотеки COM
  CoInitializeEx(NIL, COINIT_MULTITHREADED);

  // Обнуление ссылок на менеджер воспроизведения и загрузчик
  FPerformance := NIL;
  FLoader := NIL;

  // Запоминаем указатель на главную форму
  FHandle := Handle;

  // Создаем контейнер сегментов воспроизведения
  FSegmentList := TList.Create;
end;
```

```
{*************************************************************************}
{**  Создаем сегмент воспроизведения                                    **}
{*************************************************************************}
function TdxMusicManager.CreateSegment(FileName: WideString):
  TdxMusicSegment;
var
  FSegment: IDirectMusicSegment8;
begin
  // Обнуляем ссылку на сегмент
  FSegment := NIL;

  // Результат по умолчанию
  Result := NIL;

  // Проверка наличия менеджера воспроизведения и загрузчика
  if (FPerformance = NIL) or (FLoader = NIL) then EXIT;

  // Загружаем аудиоданные и получаем указатель на сегмент
  if FAILED(FLoader.LoadObjectFromFile(CLSID_DirectMusicSegment,
    IID_IDirectMusicSegment8, PWideChar(FileName),
    FSegment)) then EXIT;

  // Загружаем данные в менеджер
  if FAILED(FSegment.Download(FPerformance)) then EXIT;

  // Создаем объект, инкапсулирующий сегмент воспроизведения
  Result := TdxMusicSegment.Create(FPerformance, FLoader, FSegment);

  // Добавляем объект в контейнер
  if Result <> NIL then
     FSegmentList.Add(Result);
end;

{*************************************************************************}
{**  Удаляем сегмент воспроизведения по индексу                         **}
{*************************************************************************}
```

```
function TdxMusicManager.DeleteSegment(Index: integer): HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если индекс неверный, то завершаем работу
  if (Index < 0) or (Index >= FSegmentList.Count) then EXIT;
  // Останавливаем воспроизведение сегмета
  TdxMusicSegment(FSegmentList.Items[Index]).Stop;

  // Освобождаем память, выделенную под сегмент
  TdxMusicSegment(FSegmentList.Items[Index]).Free;
  FSegmentList.Items[Index] := NIL;

  // Удаляем ссылку на сегмент из контейнера
  FSegmentList.Delete(Index);

  Result := S_OK;
end;

{*************************************************************************}
{**  Удаляем сегмент воспроизведения по ссылке                          **}
{*************************************************************************}
function TdxMusicManager.DeleteSegment(Segment: TdxMusicSegment):
HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Проверка сегмента
  if (Segment = NIL) or (FSegmentList.IndexOf(Segment) < 0) then EXIT;

  // Останавливаем воспроизведение сегмента
  Segment.Stop;

  // Удаляем ссылку на сегмент из контейнера
  FSegmentList.Remove(Segment);
```

```
  // Освобождаем память, выделенную под сегмент
  FreeAndNil(Segment);
end;

{**************************************************************************}
{**  Деструктор класса                                                   **}
{**************************************************************************}
destructor TdxMusicManager.Destroy;
var
  I: integer;
begin

  // Останавливаем работу менеджера воспроизведения
  if FPerformance <> NIL then
     FPerformance.Stop(NIL, NIL, 0, 0);

  // Очищаем контейнер сегментов
  if FSegmentList <> NIL then
  begin
    for I := 0 to FSegmentList.Count - 1 do
    begin
      TdxMusicSegment(FSegmentList.Items[I]).Stop;
      TdxMusicSegment(FSegmentList.Items[I]).Free;
      FSegmentList.Items[I] := NIL;
    end;

    FreeAndNil(FSegmentList);
  end;

  // Обнуление ссылок на загрузчик и менеджер воспроизведения
  FLoader := NIL;
  FPerformance := NIL;

  // Завершение работы с COM
  CoUninitialize;
end;
```

```
{*************************************************************************}
{**  Получение сегмента по индексу                                      **}
{*************************************************************************}
function TdxMusicManager.GetSegment(Index: integer): TdxMusicSegment;
begin
  // Результат по умолчанию
  Result := NIL;

  // Если индекс неверный, то завершаем работу
  if (Index < 0) or (Index >= FSegmentList.Count) then EXIT;

  // Получаем сегент по индексу из контейнера
  Result := FSegmentList.Items[Index];
end;

{*************************************************************************}
{**  Инициализация музыкального менеджера                               **}
{*************************************************************************}
function TdxMusicManager.Initialize: HResult;
begin
  // Создаем загрузчик
  Result := CoCreateInstance(CLSID_DirectMusicLoader, NIL, CLSCTX_INPROC,
    IID_IDirectMusicLoader8, FLoader);
  if FAILED(Result) then EXIT;

  // Создаем менеджер воспроизведения
  Result := CoCreateInstance(CLSID_DirectMusicPerformance, NIL,
    CLSCTX_INPROC, IID_IDirectMusicPerformance8, FPerformance);
  if FAILED(Result) then EXIT;

  // Инициализируем менеджер воспроизведения
  Result := FPerformance.InitAudio(NIL, NIL, FHandle,
    DMUS_APATH_DYNAMIC_STEREO,
    128, DMUS_AUDIOF_ALL, NIL);
end;
```

```
{*************************************************************************}
{**  Получение числа сегментов                                          **}
{*************************************************************************}
function TdxMusicManager.SegmentCount: integer;
begin
  Result := FSegmentList.Count;
end;

{**} { TdxMusicSegment } {*****************************************}

{*************************************************************************}
{**  Конструктор класса                                                 **}
{*************************************************************************}
constructor TdxMusicSegment.Create(APerformance: IDirectMusicPerfor-
mance8;
  ALoader: IDirectMusicLoader8; ASegment: IDirectMusicSegment8);
begin
  // Запоминаем менеджер воспроизведения
  FPerformance := APerformance;

  // Запоминаем загрузчик
  FLoader := ALoader;
  // Запоминаем сегмент воспроизведения
  FSegment := ASegment;
end;

{*************************************************************************}
{**  Деструктор класса                                                  **}
{*************************************************************************}
destructor TdxMusicSegment.Destroy;
begin
  // Проверяем наличие сегмента
  if FSegment <> NIL then
  begin
    // Выгружаем данные из сегмента
    FSegment.Unload(FPerformance);
```

```
    // Обнуляем сегмент воспроизведения
    FSegment := NIL;
  end;
end;

{*************************************************************************}
{**  Состояние воспроизведения сегмента                                 **}
{*************************************************************************}
function TdxMusicSegment.IsPlaying: boolean;
begin
  Result := (FPerformance.IsPlaying(FSegment, NIL) = S_OK);
end;

{*************************************************************************}
{**  Воспроизвести сегмент                                              **}
{*************************************************************************}
function TdxMusicSegment.Play(IsPrimary: boolean): HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Проверка наличия менеджера воспроизведения
  if FPerformance = NIL then EXIT;

  if IsPrimary then
    // Воспроизводим основной сегмент
    Result := FPerformance.PlaySegmentEx(FSegment, NIL, NIL, 0,
      0, NIL, NIL, NIL)
  else
    // Воспроизводим вторичный сегмент
    Result := FPerformance.PlaySegmentEx(FSegment, NIL, NIL,
      DMUS_SEGF_SECONDARY,
      0, NIL, NIL, NIL);
end;
```

```
{*************************************************************************}
{**  Установка числа циклов воспроизведения                             **}
{*************************************************************************}
function TdxMusicSegment.SetRepeats(dwRepeats: DWORD): HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Проверка наличия сегмента
  if FSegment = NIL then EXIT;

  // Устанавливаем число циклов воспроизведения
  Result := FSegment.SetRepeats(dwRepeats);
end;

{*************************************************************************}
{**  Остановка воспроизведения сегмента                                 **}
{*************************************************************************}
function TdxMusicSegment.Stop: HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Проверка наличия менеджера воспроизведения
  if FPerformance = NIL then EXIT;

  // Останавливаем воспроизведение сегмента
  Result := FPerformance.StopEx(FSegment, 0, 0);
end;

END.
```

Пример использования классов

Пример расположен в папке Examples\DirectMusic\DM_Test на компакт-диске. Окно приложения состоит из таких элементов, как список загруженных аудиофайлов, кнопки загрузки, воспроизведения и остановки воспроизведения звукового сегмента, а также различных элементов настройки типа сегмента и числа циклов воспроизведения (рис. 10.1).

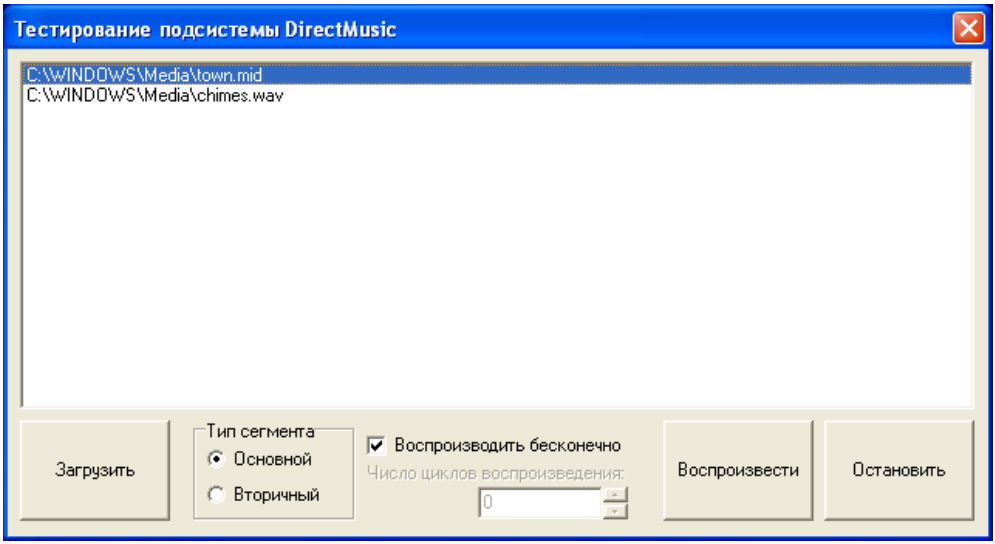


Рис. 10.1. Общий вид тестирующего приложения для классов `TdxMusicSegment` и `TdxMusicManager`

Код основного модуля, тестирующего классы приложения, приведен в листинге 10.2.

Листинг 10.2. Текст модуля FormMain.pas проекта DM_Test

```
UNIT FormMain;
{*************************************************************************}
{** Тестирование класса TdxMusicManager                                 **}
{** Автор: Есенин Сергей Анатольевич                                    **}
{*************************************************************************}
```

```
{**} INTERFACE {*************************************************************}

{**} USES {******************************************************************}
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, UdxMusicManager, StdCtrls, ExtCtrls, ComCtrls;

{**} TYPE {******************************************************************}
  TMainForm = class(TForm)
    listBoxMusic: TListBox;
    buttonLoad: TButton;
    buttonPlay: TButton;
    buttonStop: TButton;
    openDialogMusic: TOpenDialog;
    radioGroupSegmentType: TRadioGroup;
    checkBoxPlayInfinite: TCheckBox;
    labelPlayCount: TLabel;
    editPlayCount: TEdit;
    upDownPlayCount: TUpDown;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure buttonLoadClick(Sender: TObject);
    procedure buttonPlayClick(Sender: TObject);
    procedure buttonStopClick(Sender: TObject);
    procedure checkBoxPlayInfiniteClick(Sender: TObject);
  PRIVATE
  PUBLIC
  END;

{**} VAR {*******************************************************************}
  MainForm: TMainForm;
  MusicManager: TdxMusicManager;

{**} IMPLEMENTATION {********************************************************}

{$R *.dfm}
```

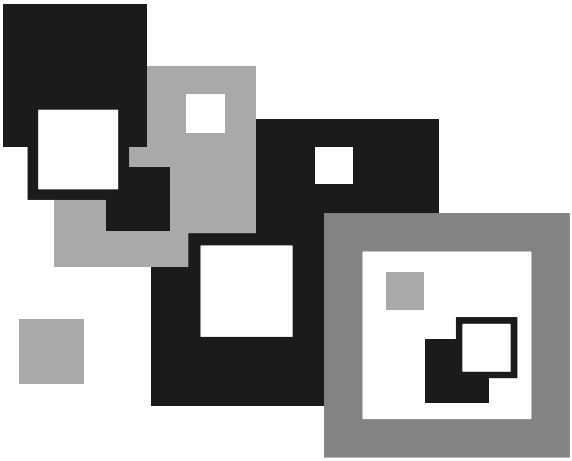
```
{*************************************************************************}
{**  Создание и инициализация объекта MusicManager                      **}
{*************************************************************************}
procedure TMainForm.FormCreate(Sender: TObject);
begin
  MusicManager := TdxMusicManager.Create(Handle);
  if FAILED(MusicManager.Initialize) then
     FreeAndNil(MusicManager);
end;

{*************************************************************************}
{**  Удаление объекта MusicManager                                      **}
{*************************************************************************}
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  if MusicManager <> NIL then
     FreeAndNil(MusicManager);
end;

{*************************************************************************}
{**  Добавление нового сегмента                                         **}
{*************************************************************************}
procedure TMainForm.buttonLoadClick(Sender: TObject);
var
  MusicSegment: TdxMusicSegment;
begin
  if openDialogMusic.Execute then
  begin
    MusicSegment := MusicManager.CreateSegment(openDialogMusic.FileName);
    if MusicSegment <> NIL then
    begin
      listBoxMusic.AddItem(openDialogMusic.FileName, MusicSegment);
      if listBoxMusic.ItemIndex < 0 then
         listBoxMusic.ItemIndex := 0;
    end;
  end;
end;
```

```
{************************************************************************}
{**  Воспроизведение звукового сегмента                                **}
{************************************************************************}
procedure TMainForm.buttonPlayClick(Sender: TObject);
var
  MusicSegment: TdxMusicSegment;
begin
  MusicSegment := TdxMusicSegment(
    listBoxMusic.Items.Objects[listBoxMusic.ItemIndex]);
  if MusicSegment = NIL then EXIT;

  if checkBoxPlayInfinite.Checked then
     MusicSegment.SetRepeats
  else
     MusicSegment.SetRepeats(upDownPlayCount.Position);

  if radioGroupSegmentType.ItemIndex = 0 then
     MusicSegment.Play(TRUE)
  else
     MusicSegment.Play;
end;

{************************************************************************}
{**  Остановка воспроизведения звукового сегмента                      **}
{************************************************************************}
procedure TMainForm.buttonStopClick(Sender: TObject);
var
  MusicSegment: TdxMusicSegment;
begin
  MusicSegment := TdxMusicSegment(
    listBoxMusic.Items.Objects[listBoxMusic.ItemIndex]);
  if MusicSegment = NIL then EXIT;

  MusicSegment.Stop;
end;
```

```
{*************************************************************************}
{**  Выбор режима воспроизведения сегмента                              **}
{*************************************************************************}
procedure TMainForm.checkBoxPlayInfiniteClick(Sender: TObject);
begin
  labelPlayCount.Enabled := not checkBoxPlayInfinite.Checked;
  editPlayCount.Enabled := not checkBoxPlayInfinite.Checked;
  upDownPlayCount.Enabled := not checkBoxPlayInfinite.Checked;
end;

END.
```

Итоги

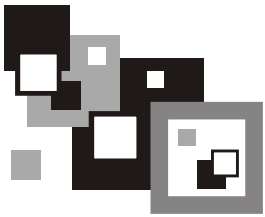
Итак, вот мы и закончили изучение подсистемы DirectMusic. В данной части книги мы разобрали работу с файлами формата MIDI и WAV при помощи достаточно мощной подсистемы. Были рассмотрены классы `TdxMusicSegment` и `TdxMusicManager`, облегчающие нашу работу. Можно смело утверждать, что знаний, полученных в этой части, будет достаточно для самостоятельной работы с DirectMusic, а также для последующего изучения таких ее составляющих, не вошедших в книгу, как сообщения, объекты часов, буферы, порты и т. д.



ЧАСТЬ V

DIRECTINPUT

Глава 11



Общие сведения

DirectInput — это компонент, входящий в состав Microsoft DirectX и обеспечивающий работу с устройствами ввода, такими как клавиатура, мышь, джойстик, другими игровыми устройствами (например, шлем виртуальной реальности) и устройствами с обратной связью. Благодаря аппаратной независимости повышается скорость работы со всеми устройствами ввода.

Работа с устройствами ввода идет непосредственно через драйвер устройства, благодаря чему подсистема DirectInput рассматривает именно текущее физическое состояние устройства (игнорируя, к примеру, установку автоповтора символа для клавиатуры).

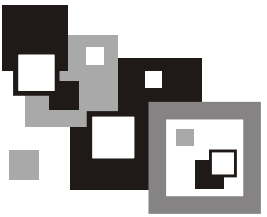
Режимы работы

Существуют два режима получения данных от устройств ввода — *буферизованный* (buffered) и *непосредственный* (immediate). В первом случае весь поток данных от устройства ввода помещается в специальный буфер и ваше приложение в любой момент времени может просмотреть содержимое этого буфера. Во втором случае мы считываем текущее состояние устройства. Режим работы определяется конкретными задачами. К примеру, в приложении, имеющем меню и различные элементы визуального интерфейса, разумнее использовать буферизованный режим, а в игре, в которой вы управляете каким-либо персонажем, удобнее использовать непосредственный режим.

Уровни взаимодействия

Уровни взаимодействия (кооперации) определяют, каким образом будет организован доступ к устройствам ввода между различными приложениями. Их можно разделить на две части: *активный* (foreground) и *фоновый* (background) режимы, а также *эксклюзивный* (exclusive) и *совместимый* или *обычный* (non-exclusive) режимы. В активном режиме доступ к устройствам ввода будет получен, только когда приложение активно. В фоновом режиме мы имеем доступ к устройствам ввода даже при работе приложения в фоновом режиме. Эксклюзивный уровень доступа дает приложению монопольный доступ к устройству, а в обычном режиме все приложения могут использовать устройство ввода.

Глава 12



Работа с устройствами ввода

Интерфейсы

Для эффективной работы с подсистемой DirectInput достаточно трех интерфейсов:

- `IDirectInput8`;
- `IDirectInputDevice8`;
- `IDirectInputEffect`.

`IDirectInput8` — основной интерфейс для работы с подсистемой DirectInput, заменяющий более ранние версии интерфейсов `IDirectInput`, `IDirectInput2` и `IDirectInput7`.

Объект `IDirectInput8` создается вызовом функции `DirectInput8Create`:

```
function DirectInput8Create(
  hinst: THandle;
  dwVersion: DWORD;
  const riidltf: TGUID;
  out ppvOut{: Pointer};
  punkOuter: IUnknown):
HResult; stdcall; external DirectInput8Dll;
```

Здесь:

- `hinst` — дескриптор приложения;
- `dwVersion` — версия DirectInput, используемого приложением (по умолчанию принято использовать `DIRECTINPUT_VERSION`);
- `riidltf` — уникальный идентификатор создаваемого интерфейса (в нашем случае это `IID_IDirectInput8`);

- ppvOut — адрес указателя, в который будет занесен интерфейс создаваемого объекта;
- punkOuter — указатель на объект с IUnknown, должен быть нулевым.

Рассмотрим все методы данного интерфейса:

- CreateDevice — создание и инициализация объекта устройства ввода с интерфейсом IDirectInputDevice8;
- EnumDevices — получение списка доступных устройств ввода;
- GetDeviceStatus — получение статуса устройства;
- RunControlPanel — вызов Панели управления для установки нового устройства ввода или изменения конфигурации;
- Initialize — инициализация объекта DirectInput. Не используется, т. к. вызов DirectInput8Create автоматически инициализирует объект DirectInput после создания;
- FindDevice — получение уникального идентификатора устройства ввода, подключенного к системе;
- EnumDevicesBySemantics — получение списка доступных устройств ввода наиболее подходящих по семантике;
- ConfigureDevices — отображение страницы свойств устройства ввода.

Устройство ввода создается вызовом IDirectInput8.CreateDevice:

```
function CreateDevice(
  const rguid: TGUID;
  out lplpDirectInputDevice: IDirectInputDevice8A;
  pUnkOuter: IUnknown):
HResult; stdcall;
```

Здесь:

- rguid — глобальный уникальный идентификатор создаваемого устройства:
  - GUID_SysKeyboard — клавиатура;
  - GUID_SysMouse — мышь;
- lplpDirectInputDevice — адрес указателя, в который будет занесен интерфейс IDirectInputDevice8 создаваемого объекта;
- pUnkOuter — указатель на объект с IUnknown, должен быть нулевым.

Как видно из описания данной функции, стандартными устройствами ввода являются только клавиатура и мышь. Для использования джойстика необходимо вызывать функцию IDirectInput8.EnumDevices.

Интерфейс `IDirectInputDevice8` предназначен для управления различными устройствами ввода. Перечислим все методы:

- `Acquire` — захват доступа к устройству ввода;
- `BuildActionMap` — формирование карты действий для устройства и получение информации по ней;
- `CreateEffect` — создание и инициализация глобального эффекта;
- `EnumCreatedEffectObjects` — перечисление созданных для устройства эффектов;
- `EnumEffects` — перечисление всех эффектов, поддерживаемых устройством с обратной связью;
- `EnumEffectsInFile` — перечисление всех эффектов, хранимых в файле, созданном утилитой Force Editor или любым другим приложением, использующим тот же формат файла;
- `EnumObjects` — перечисление объектов ввода/вывода, доступных данному устройству;
- `Escape` — посылка аппаратно-специфической команды устройству с обратной связью;
- `GetCapabilities` — опрос возможностей устройства;
- `GetDeviceData` — извлечение данных из буфера устройства;
- `GetDeviceInfo` — получение информации об устройстве;
- `GetDeviceState` — получение непосредственных данных о состоянии устройства ввода;
- `GetEffectInfo` — получение информации об эффекте;
- `GetForceFeedbackState` — получение состояния устройства с обратной связью;
- `GetImageInfo` — получение информации об образе устройства для отображения на странице свойств;
- `GetObjectInfo` — получение информации об объекте устройства, таком как кнопка или ось;
- `GetProperty` — получение значения параметра устройства;
- `Initialize` — инициализация объекта DirectInputDevice. Не используется, т. к. вызов `IDirectInput8.CreateDevice` автоматически инициализирует объект устройства после его создания;
- `Poll` — извлечение данных из опрашиваемых объектов устройства ввода;
- `RunControlPanel` — вызов Панели управления DirectInput для изменения конфигурации текущего устройства;
- `SendDeviceData` — посылка данных на устройство вывода (не используется);

- `SendForceFeedbackCommand` — посылка команды устройству с обратной связью;
- `SetActionMap` — установка формата данных для устройства и назначение действий для объектов устройства ввода;
- `SetCooperativeLevel` — установка уровня взаимодействия;
- `SetDataFormat` — установка формата получаемых от устройства данных;
- `SetEventNotification` — установка события, которое сработает, когда изменится состояние устройства;
- `SetProperty` — установка значения параметра устройства;
- `Unacquire` — освобождение устройства;
- `WriteEffectToFile` — запись информации об одном или нескольких эффектах устройств с обратной связью в файл.

Некоторые методы требуют более детального рассмотрения.

Захват и освобождение устройства ввода:

```
function Acquire: HResult; stdcall;
function Unacquire: HResult; stdcall;
```

Установка формата данных:

```
function SetDataFormat(
  const lpdf: TDIDataFormat):
HResult; stdcall;
```

Здесь:

- `lpdf` — адрес структуры, описывающей формат данных. Доступны следующие предустановленные значения:
- `c_dfDIKeyboard` — клавиатура;
- `c_dfDIMouse`, `c_dfDIMouse2` — мышь;
- `c_dfDIJoystick`, `c_dfDIJoystick2` — джойстик.

Установка уровня взаимодействия:

```
function SetCooperativeLevel(
  hwnd: HWND;
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `hwnd` — дескриптор окна приложения, которое должно быть ассоциировано с устройством ввода;

- dwFlags — флаги, описывающие уровни взаимодействия:
  - DISCL_BACKGROUND — фоновый режим, доступ к устройству ввода есть даже при работе приложения в фоновом режиме;
  - DISCL_FOREROUND — доступ к устройству возможен только в активном состоянии приложения;
  - DISCL_NONEXCLUSIVE — все приложения могут использовать устройство ввода;
  - DISCL_EXCLUSIVE — устройство ввода доступно только данному приложению;
  - DISCL_NOWINKEY — отключение кнопки **Пуск** в системе Microsoft Windows.

  Установками по умолчнию считается комбинация следующих флагов: DISCL_NONEXCLUSIVE и DISCL_BACKGROUND.

Получение данных от устройства:

```
function GetDeviceState(
  cbData: DWORD;
  lpvData: Pointer):
HResult; stdcall;
```

Здесь:

- cbData — размер буфера для получения данных (в байтах);
- lpvData — буфер для получения данных.

Интерфейс IDirectInputEffect используется для управления устройствами с обратной связью и нами не рассматривается.

Общий алгоритм работы

Итак, рассмотрев достаточно подробно интерфейсы работы с устройствами ввода, мы можем составить примерный алгоритм действий:

1. Создание объекта DirectInput.
2. Создание устройства ввода.
3. Установка формата данных для устройства.
4. Установка уровня взаимодействия.
5. Захват доступа к устройству.
6. Работа с устройством ввода.
7. Освобождение устройства ввода.

Фактически это универсальный алгоритм для различных устройств ввода, таких как клавиатура, мышь и джойстик.

Клавиатура

Работу с устройствами ввода мы рассмотрим на конкретных примерах (листинг будет приведен далее). Алгоритм работы с клавиатурой будет следующим:

1. Создание объекта DirectInput:
```
Result := DirectInput8Create(HInstance, DIRECTINPUT_VERSION,
  IID_IDirectInput8, FDirectInput, NIL);
```
2. Создание устройства ввода:
```
Result := FDirectInput.CreateDevice(GUID_SysKeyboard, FKeyboard,
  NIL);
```
3. Установка формата данных для устройства:
```
Result := FKeyboard.SetDataFormat(c_dfDIKeyboard);
```
4. Установка уровня взаимодействия:
```
Result := FKeyboard.SetCooperativeLevel(FHandle,
  DISCL_BACKGROUND or DISCL_NONEXCLUSIVE);
```
5. Захват доступа к устройству:
```
FKeyboard.Acquire;
```
6. Работа с устройством ввода:
```
Result := FKeyboard.GetDeviceState(SizeOf(KeyboardBuffer^),
  KeyboardBuffer);
```
7. Освобождение устройства ввода:
```
FKeyboard.Unacquire;
```

Опрос клавиатуры производится при помощи вызова метода `IDirectInputDevice8.GetDeviceState`, вторым параметром которого является буфер клавиатуры `KeyboardBuffer` — фактически массив из 256 байт:

```
{**} TYPE {*****************************************************}
  PdxKeyboardState = ^TdxKeyboardState;
  TdxKeyboardState = array [0..255] of Byte;
```

Проверка нажатия клавиши выполняется следующим образом:

```
if KeyBuffer[DIK_UP] = $080 then ...
```

где `DIK_UP` — обозначает клавишу <↑>.

Далее приводится список основных констант, определяющих клавиши:

- `DIK_ESCAPE` — клавиша <Esc>;
- `DIK_0 – DIK_9` — клавиши <0>—<9>;
- `DIK_MINUS` — клавиша <->;
- `DIK_EQUALS` — клавиша <=>;
- `DIK_BACK` — клавиша <Backspace>;
- `DIK_TAB` — клавиша <Tab>;
- `DIK_A – DIK_Z` — клавиши <A>—<Z>;
- `DIK_LBRACKET` — клавиша <[>;
- `DIK_RBRACKET` — клавиша <]>;
- `DIK_RETURN` — клавиша <Enter>;
- `DIK_LCONTROL` — левая клавиша <Ctrl>;
- `DIK_SEMICOLON` — клавиша <;>;
- `DIK_APOSTROPHE` — клавиша <'>;
- `DIK_GRAVE` — клавиша <~>;
- `DIK_LSHIFT` — левая клавиша <Shift>;
- `DIK_BACKSLASH` — клавиша <\>;
- `DIK_COMMA` — клавиша <,>;
- `DIK_PERIOD` — клавиша <.>;
- `DIK_SLASH` — клавиша </>;
- `DIK_RSHIFT` — правая клавиша <Shift>;
- `DIK_MULTIPLY` — клавиша <*> на цифровой клавиатуре;
- `DIK_LMENU` — левая клавиша <Alt>;
- `DIK_SPACE` — клавиша <Пробел>;
- `DIK_CAPITAL` — клавиша <Caps Lock>;
- `DIK_F1—DIK_F12` — клавиши <F1>—<F12>;
- `DIK_NUMLOCK` — клавиша <Num Lock>;
- `DIK_SCROLL` — клавиша <Scroll Lock>;
- `DIK_NUMPAD0—DIK_NUMPAD9` — цифровые клавиши на дополнительной (цифровой) клавиатуре;
- `DIK_SUBTRACT` — клавиша <-> на цифровой клавиатуре;
- `DIK_ADD` — клавиша <+> на клавиатуре;

- DIK_DECIMAL — клавиша <.> на цифровой клавиатуре;
- DIK_NUMPADENTER — клавиша <Enter> на цифровой клавиатуре;
- DIK_RCONTROL — правая клавиша <Ctrl>;
- DIK_DIVIDE — клавиша </> на цифровой клавиатуре;
- DIK_RMENU — правая клавиша <Alt>;
- DIK_PAUSE — клавиша <Pause>;
- DIK_HOME — клавиша <Home>;
- DIK_UP — клавиша <↑>;
- DIK_PRIOR — клавиша <PgUp>;
- DIK_LEFT — клавиша <←>;
- DIK_RIGHT — клавиша <→>;
- DIK_END — клавиша <End>;
- DIK_DOWN — клавиша <↓>;
- DIK_NEXT — клавиша <PgDn>;
- DIK_INSERT — клавиша <Insert>;
- DIK_DELETE — клавиша <Delete>;
- DIK_LWIN — левая клавиша <⊞>;
- DIK_RWIN — правая клавиша <⊞>;
- DIK_APPS — клавиша <AppMenu>;
- DIK_POWER — клавиша <System Power>;
- DIK_SLEEP — клавиша <System Sleep>;
- DIK_WAKE — клавиша <System Wake>.

Мышь

Основное отличие в работе с мышью от работы с клавиатурой заключается в наличии осей перемещения *x* и *y*, появлении колеса прокрутки (mouse wheel) и уменьшении количества клавиш. Координаты *x* и *y* возвращаются по умолчанию относительными, но существует возможность настройки мыши на возвращение и абсолютных координат. Алгоритм работы с мышью:

1. Создание объекта DirectInput:

```
Result := DirectInput8Create(HInstance, DIRECTINPUT_VERSION,
  IID_IDirectInput8, FDirectInput, NIL);
```

2. Создание устройства ввода:

```
Result := FDirectInput.CreateDevice(GUID_SysMouse, FMouse, NIL);
if FAILED(Result) then EXIT;
```

3. Установка формата данных для устройства:

```
Result := FMouse.SetDataFormat(c_dfDIMouse);
```

4. Установка уровня взаимодействия:

```
Result := FMouse.SetCooperativeLevel(FHandle,
  DISCL_BACKGROUND or DISCL_NONEXCLUSIVE);
```

5. Захват доступа к устройству:

```
FMouse.Acquire;
```

6. Работа с устройством ввода:

```
Result := FMouse.GetDeviceState(SizeOf(MouseBuffer^), MouseBuffer);
```

7. Освобождение устройства ввода:

```
FMouse.Unacquire;
```

Отдельно стоит сказать о том, что повторной инициализации объекта DirectInput в приложении не требуется, и, если он уже создан, то первый шаг нужно пропустить.

Данные от мыши записываются в специальный буфер `MouseBuffer`:

```
{**} TYPE {*****************************************************}
  PdxMouseState = ^TdxMouseState;
  TdxMouseState = TDIMouseState;
```

Тип `TDIMouseState` — структура, описывающая атрибуты мыши:

```
_DIMOUSESTATE = packed record
  lX: Longint;
  lY: Longint;
  lZ: Longint;
  rgbButtons: array[0..3] of Byte;
end;
TDIMouseState = _DIMOUSESTATE;
```

Здесь:

- `lX` — значение по оси *x*;
- `lY` — значение по оси *y*;

- `lZ` — значение прокрутки (при отсутствии колеса прокртутки всегда нулевое значение);
- `rgbButtons` — массив кнопок мыши. Старший бит каждого элемента установлен, если соответствующая кнопка нажата.

Джойстик

Работа с джойстиком сходна с работой с мышью — присутствуют и оси, и кнопки. По умолчанию джойстик возвращает абсолютные координаты. Как уже было сказано ранее, стандартными устройствами ввода являются клавиатура и мышь, и для использования джойстика необходимо вызвать функцию `IDirectInput8.EnumDevices`. Соответственно, немного изменяется и алгоритм работы с данным устройством:

1. Создание объекта DirectInput (если еще не создан):

```
Result := DirectInput8Create(HInstance, DIRECTINPUT_VERSION,
  IID_IDirectInput8, FDirectInput, NIL);
```

2. Поиск и создание устройства ввода:

```
// Получаем список идентификаторов игровых устройств
Result := FDirectInput.EnumDevices(
  DI8DEVCLASS_GAMECTRL,
  @EnumJoysticks,
  NIL,
  DIEDFL_ATTACHEDONLY);

// Создаем игровое устройство
for I := 0 to FJoyCount - 1 do
begin
  Result := FDirectInput.CreateDevice(FJoyList[I], FJoystick, NIL);
  if SUCCEEDED(Result) then Break;
end;
```

3. Установка формата данных для устройства:

```
Result := FJoystick.SetDataFormat(c_dfDIJoystick);
```

4. Установка уровня взаимодействия:

```
Result := FJoystick.SetCooperativeLevel(FHandle,
  DISCL_BACKGROUND or DISCL_EXCLUSIVE);
```

5. Захват доступа к устройству:
```
FJoystick.Acquire;
```
6. Работа с устройством ввода:
```
Result := FJoystick.GetDeviceState(SizeOf(JoyBuffer^), JoyBuffer);
```
7. Освобождение устройства ввода:
```
FJoystick.Unacquire;
```

Данные джойстика записываются в буфер `JoyBuffer`:

```
{**} TYPE {*****************************************************}
  PdxJoyState = ^TdxJoyState;
  TdxJoyState = TDIJoyState;
```

Тип `TDIJoyState` представляет собой следущую запись:

```
DIJOYSTATE = packed record
  lX: Longint;                          (* x-axis position        *)
  lY: Longint;                          (* y-axis position        *)
  lZ: Longint;                          (* z-axis position        *)
  lRx: Longint;                         (* x-axis rotation        *)
  lRy: Longint;                         (* y-axis rotation        *)
  lRz: Longint;                         (* z-axis rotation        *)
  rglSlider: array[0..1] of Longint;    (* extra axes positions   *)
  rgdwPOV: array[0..3] of DWORD;        (* POV directions         *)
  rgbButtons: array[0..31] of Byte;     (* 32 buttons             *)
end;
TDIJoyState = DIJOYSTATE;
```

Здесь:

- `lX` — значение по оси *x*;
- `lY` — значение по оси *y*;
- `lZ` — значение по оси *z*;
- `lRx` — угол поворота по оси *x*;
- `lRy` — угол поворота по оси *y*;
- `lRz` — угол поворота по оси *z*;
- `rglSlider` — дополнительные оси;
- `rgdwPOV` — массив диспетчеров управления, таких как точки непосредственного просмотра (point-of-view);
- `rgbButtons` — массив кнопок джойстика. Старший бит каждого элемента установлен, если соответствующая кнопка нажата.

Класс *TdxInputManager*

Вспомогательный класс `TdxInputManager` упрощает работу с устройствами ввода — клавиатурой, мышью и джойстиком. Режим работы подсистемы DirectInput — непосредственный (immediate). Класс содержит всего пять основных методов:

- инициализация:

```
function Initialize: HRESULT;
```

- установка маски опроса устройств:

```
procedure SetDeviceMask(
  DeviceMask: DWORD);
```

- опрос клавиатуры:

```
function GetKeyboardState(
  KeyboardBuffer: PdxKeyboardState):
HRESULT;
```

- опрос мыши:

```
function GetMouseState(
  MouseBuffer: PdxMouseState):
HRESULT;
```

- опрос джойстика:

```
function GetJoystickState(
  JoyBuffer: PdxJoyState):
HRESULT;
```

Возможность получения данных от устройства ввода зависит от соответствующего флага, разрешающего работу с устройством или запрещающего ее. Флаги устанавливаются в функции установки маски опроса устройств. Маска опроса устройств ввода представляет собой битовую маску, в которой 3 бита отвечают за включение в опрос соответствующего биту устройства. Константы, определяющие устройства ввода, таковы:

- `idKeyboard  = $00000001;`
- `idMouse     = $00000002;`
- `idJoystick  = $00000004.`

Итак, преведем код модуля UdxInputManager.pas (листинг 12.1).

Листинг 12.1. Текст модуля UdxInputManager.pas

```
UNIT UdxInputManager;
{*********************************************************************}
{** DirectInput: работа с различными устройствами ввода              **}
{** Автор: Есенин Сергей Анатольевич                                 **}
{*********************************************************************}

{**} INTERFACE {*****************************************************}

{**} USES {**********************************************************}
  Windows, ComObj, ActiveX, DirectInput;

{**} CONST {*********************************************************}
  idKeyboard  = $00000001;
  idMouse     = $00000002;
  idJoystick  = $00000004;

{**} TYPE {**********************************************************}
  PdxKeyboardState = ^TdxKeyboardState;
  TdxKeyboardState = array [0..255] of Byte;

  PdxMouseState = ^TdxMouseState;
  TdxMouseState = TDIMouseState;

  PdxJoyState = ^TdxJoyState;
  TdxJoyState = TDIJoyState;

  TdxInputManager = class
  PRIVATE
    FDirectInput:   IDirectInput8;

    FKeyboard: IDirectInputDevice8;
    FMouse:    IDirectInputDevice8;
    FJoystick: IDirectInputDevice8;
```

```
    FAcquireKeyboard: boolean;
    FAcquireMouse:    boolean;
    FAcquireJoystick: boolean;

    FHandle: THandle;
  PUBLIC
    constructor Create(AHandle: THandle);
    destructor Destroy; override;

    function Initialize: HRESULT;

    procedure SetDeviceMask(DeviceMask: DWORD);

    function GetKeyboardState(KeyboardBuffer: PdxKeyboardState): HRESULT;
    function GetMouseState(MouseBuffer: PdxMouseState): HRESULT;
    function GetJoystickState(JoyBuffer: PdxJoyState): HRESULT;
  END;

{**} IMPLEMENTATION {************************************************}

{**} { TdxInputManager } {*******************************************}

{*********************************************************************}
{** Конструктор класса                                              **}
{*********************************************************************}
constructor TdxInputManager.Create(AHandle: THandle);
begin
  // Обнуляем ссылки на объекты
  FDirectInput   := NIL;
  FKeyboard := NIL;
  FMouse    := NIL;
  FJoystick := NIL;

  // Запоминаем указатель на главную форму
  FHandle := AHandle;
end;
```

```
{*************************************************************************}
{** Деструктор класса                                                   **}
{*************************************************************************}
destructor TdxInputManager.Destroy;
begin
  if FJoystick <> NIL then
  begin
    FJoystick.Unacquire;
    FJoystick := NIL;
  end;

  if FMouse <> NIL then
  begin
    FMouse.Unacquire;
    FMouse := NIL;
  end;

  if FKeyboard <> NIL then
  begin
    FKeyboard.Unacquire;
    FKeyboard := NIL;
  end;

  FDirectInput := NIL;
end;

{*************************************************************************}
{** Перечисление игровых устройств                                     **}
{*************************************************************************}
var
  FJoyCount: integer;
  FJoyList: TGUIDList;

function EnumJoysticks(const pdinst: PDIDeviceInstance;
  pvRef: pointer): boolean; stdcall;
begin
  CopyMemory(@FJoyList[FJoyCount], @pdinst^.guidInstance, SizeOf(TGUID));
```

```
  inc(FJoyCount);
  Result := DIENUM_CONTINUE;
end;

{*************************************************************************}
{** Инициализация подсистемы                                            **}
{*************************************************************************}
function TdxInputManager.Initialize: HRESULT;
var
  I: integer;
begin
  // Инициализируем подсистему DirectInput
  Result := DirectInput8Create(HInstance, DIRECTINPUT_VERSION,
    IID_IDirectInput8, FDirectInput, NIL);
  if FAILED(Result) then EXIT;

  // Создаем объекты для работы с клавиатурой и мышью
  Result := FDirectInput.CreateDevice(GUID_SysKeyboard, FKeyboard, NIL);
  if FAILED(Result) then EXIT;
  Result := FDirectInput.CreateDevice(GUID_SysMouse, FMouse, NIL);
  if FAILED(Result) then EXIT;

  // Обнуляем число игровых устройств и список их идентификаторов
  FJoyCount := 0;
  ZeroMemory(@FJoyList, SizeOf(FJoyList));

  // Получаем список идентификаторов игровых устройств
  Result := FDirectInput.EnumDevices(
    DI8DEVCLASS_GAMECTRL,
    @EnumJoysticks,
    NIL,
    DIEDFL_ATTACHEDONLY);
  if FAILED(Result) then EXIT;

  // Создаем игровое устройство
  for I := 0 to FJoyCount - 1 do
```

```
begin
  Result := FDirectInput.CreateDevice(FJoyList[I], FJoystick, NIL);
  if SUCCEEDED(Result) then Break;
end;
if FAILED(Result) then EXIT;

// Устанавливаем форматы каждого из устройств
Result := FKeyboard.SetDataFormat(c_dfDIKeyboard);
if FAILED(Result) then EXIT;
Result := FMouse.SetDataFormat(c_dfDIMouse);
if FAILED(Result) then EXIT;

if FJoystick <> NIL then
begin
  Result := FJoystick.SetDataFormat(c_dfDIJoystick);
  if FAILED(Result) then EXIT;
end;

// Устанавливаем уровни взаимодействия
if FKeyboard <> NIL then
begin
  Result := FKeyboard.SetCooperativeLevel(FHandle, DISCL_BACKGROUND or
    DISCL_NONEXCLUSIVE);
  if FAILED(Result) then EXIT;
end;

if FMouse <> NIL then
begin
  Result := FMouse.SetCooperativeLevel(FHandle, DISCL_BACKGROUND or
    DISCL_NONEXCLUSIVE);
  if FAILED(Result) then EXIT;
end;

if FJoystick <> NIL then
begin
  Result := FJoystick.SetCooperativeLevel(FHandle, DISCL_BACKGROUND or
```

```
      DISCL_EXCLUSIVE);
  end;
end;

{*************************************************************************}
{** Опрос состояния клавиатуры                                          **}
{*************************************************************************}
function TdxInputManager.GetKeyboardState(KeyboardBuffer:
PdxKeyboardState): HRESULT;
begin
  result := E_FAIL;
  if not (FAcquireKeyboard) or (FKeyboard = NIL) then EXIT;

  // Опрос клавиатуры
  Result := FKeyboard.GetDeviceState(SizeOf(KeyboardBuffer^),
    KeyboardBuffer);

  // Если устройство потеряно
  if Result = DIERR_INPUTLOST then
  begin
    // Захватываем снова
    FKeyboard.Acquire();

    // Производим повторный опрос
    Result := FKeyboard.GetDeviceState(SizeOf(KeyboardBuffer^),
      KeyboardBuffer);
  end;
end;

{*************************************************************************}
{** Опрос состояния мыши                                                **}
{*************************************************************************}
function TdxInputManager.GetMouseState(MouseBuffer: PdxMouseState):
HRESULT;
begin
  result := E_FAIL;
  if not (FAcquireMouse) or (FMouse = NIL) then EXIT;
```

```
  // Производим опрос мыши, данные записываются в буфер-массив
  Result := FMouse.GetDeviceState(SizeOf(MouseBuffer^), MouseBuffer);

  // Если устройство потеряно
  if Result = DIERR_INPUTLOST then
  begin
    // Захватываем снова
    FMouse.Acquire();

    // Производим повторный опрос
    Result := FMouse.GetDeviceState(SizeOf(MouseBuffer^), MouseBuffer);
  end;
end;

{*************************************************************************}
{** Опрос состояния джойстика                                           **}
{*************************************************************************}
function TdxInputManager.GetJoystickState(JoyBuffer: PdxJoyState):
  HRESULT;
begin
  result := E_FAIL;
  if not (FAcquireJoystick) or (FJoystick = NIL) then EXIT;

  // Опрашиваем состояние джойстика
  Result := FJoystick.GetDeviceState(SizeOf(JoyBuffer^), JoyBuffer);

  // Если устройство потеряно
  if Result = DIERR_INPUTLOST then
  begin
    // Захватываем снова
    FJoystick.Acquire();

    // Производим повторный опрос
    Result := FJoystick.GetDeviceState(SizeOf(JoyBuffer^), JoyBuffer);
  end;
end;
```

```
{*************************************************************************}
{** Установка параметров опроса                                         **}
{*************************************************************************}
procedure TdxInputManager.SetDeviceMask(DeviceMask: DWORD);
begin
  // Установка флагов опроса устройств
  FAcquireKeyboard := ((DeviceMask and idKeyboard) = idKeyboard);
  FAcquireMouse := ((DeviceMask and idMouse) = idMouse);
  FAcquireJoystick := ((DeviceMask and idJoystick) = idJoystick);

  // Получение доступа или запрещение доступа к устройству
  // в зависимости от флага
  if FKeyboard <> NIL then
  begin
    if (DeviceMask and idKeyboard) = idKeyboard then
      FKeyboard.Acquire
    else
      FKeyboard.Unacquire;
  end;

  if FMouse <> NIL then
  begin
    if (DeviceMask and idMouse) = idMouse then
      FMouse.Acquire
    else
      FMouse.Unacquire;
  end;

  if FJoystick <> NIL then
  begin
    if (DeviceMask and idJoystick) = idJoystick then
      FJoystick.Acquire
    else
      FJoystick.Unacquire;
  end;
end;

END.
```

Пример использования класса *TdxInputManager*

Пример находится в папке Examples\DirectInput\DI_Test на прилагаемом к книге компакт-диске. Окно приложения разбито на три области — **Джойстик**, **Клавиатура** и **Мышь** — с целью наглядной демонстрации работы с устройствами ввода (рис. 12.1).

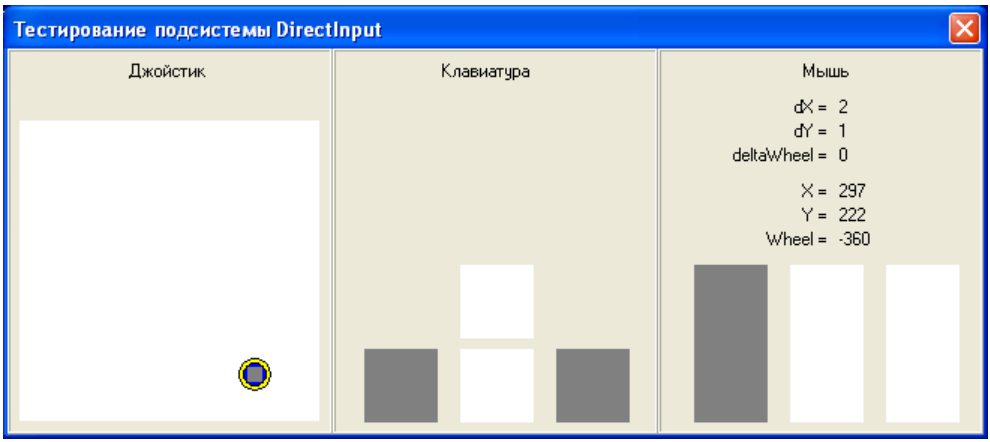


Рис. 12.1. Общий вид тестирующего приложения для класса `TdxInputManager`

Код основного модуля, тестирующего класс приложения, приведен в листинге 12.2.

Листинг 12.2. Текст модуля FormMain.pas проекта DI_Test

```
UNIT FormMain;
{*****************************************************************}
{** Тестирование класса TdxInputManager                         **}
{** Автор: Есенин Сергей Анатольевич                            **}
{*****************************************************************}

{**} INTERFACE {*************************************************}

{**} USES {******************************************************}
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
```

```
  Controls, Forms, Dialogs, DirectInput, AppEvnts, StdCtrls,
  ExtCtrls, UdxInputManager;

{**} TYPE {*****************************************************}
  TMainForm = class(TForm)
    panelJoystick: TPanel;
    panelKeyboard: TPanel;
    panelMouse: TPanel;
    paintBoxJoystick: TPaintBox;
    paintBoxUp: TPaintBox;
    paintBoxDown: TPaintBox;
    paintBoxLeft: TPaintBox;
    paintBoxRight: TPaintBox;
    paintBoxMouseLeft: TPaintBox;
    paintBoxMouseMiddle: TPaintBox;
    paintBoxMouseRight: TPaintBox;
    labelJoystick: TLabel;
    labelKeyboard: TLabel;
    labelMouse: TLabel;
    labelXInfo: TLabel;
    labelYInfo: TLabel;
    labelZInfo: TLabel;
    labelX: TLabel;
    labelY: TLabel;
    labelZ: TLabel;
    labelDXInfo: TLabel;
    labelDYInfo: TLabel;
    labelDY: TLabel;
    labelDX: TLabel;
    labelDZInfo: TLabel;
    labelDZ: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure QueryDevices;
    procedure FormDestroy(Sender: TObject);
  PRIVATE
    procedure ApplicationIdle(Sender: TObject; var Done: Boolean);
```

```
  PUBLIC
  END;

{**} VAR {***************************************************}
  MainForm: TMainForm;
  InputManager: TdxInputManager;
  LastX,
  LastY,
  LastZ: integer;
  ButtonFireFirst,
  ButtonFireSecond: boolean;
  MouseX,
  MouseY,
  MouseZ: integer;

{**} IMPLEMENTATION {*****************************************}

{$R *.dfm}

{************************************************************}
{** Опрашиваем устройства ввода                            **}
{************************************************************}
procedure TMainForm.QueryDevices;
var
  KeyBuffer:   TdxKeyboardState;
  MouseBuffer: TdxMouseState;
  JoyBuffer:   TdxJoyState;
  dx, dy, dz:  integer;
begin
  if InputManager = NIL then EXIT;

  // Опрос клавиатуры
  if SUCCEEDED(InputManager.GetKeyboardState(@KeyBuffer)) then
  begin
    if KeyBuffer[DIK_UP] = $080 then
    begin
```

```
    paintBoxUp.Canvas.Brush.Color := clGray;
  end else begin
    paintBoxUp.Canvas.Brush.Color := clWhite;
  end;

  if KeyBuffer[DIK_DOWN] = $080 then
  begin
    paintBoxDown.Canvas.Brush.Color := clGray;
  end else begin
    paintBoxDown.Canvas.Brush.Color := clWhite;
  end;

  if KeyBuffer[DIK_LEFT] = $080 then
  begin
    paintBoxLeft.Canvas.Brush.Color := clGray;
  end else begin
    paintBoxLeft.Canvas.Brush.Color := clWhite;
  end;

  if KeyBuffer[DIK_RIGHT] = $080 then
  begin
    paintBoxRight.Canvas.Brush.Color := clGray;
  end else begin
    paintBoxRight.Canvas.Brush.Color := clWhite;
  end;

  paintBoxUp.Canvas.FillRect(paintBoxUp.Canvas.ClipRect);
  paintBoxDown.Canvas.FillRect(paintBoxDown.Canvas.ClipRect);
  paintBoxLeft.Canvas.FillRect(paintBoxLeft.Canvas.ClipRect);
  paintBoxRight.Canvas.FillRect(paintBoxRight.Canvas.ClipRect);
end;

// Опрос мыши
if SUCCEEDED(InputManager.GetMouseState(@MouseBuffer)) then
begin
  if MouseBuffer.rgbButtons[0] = $080 then
```

```
begin
  paintBoxMouseLeft.Canvas.Brush.Color := clGray;
end else begin
  paintBoxMouseLeft.Canvas.Brush.Color := clWhite;
end;

if MouseBuffer.rgbButtons[2] = $080 then
begin
  paintBoxMouseMiddle.Canvas.Brush.Color := clGray;
end else begin
  paintBoxMouseMiddle.Canvas.Brush.Color := clWhite;
end;

if MouseBuffer.rgbButtons[1] = $080 then
begin
  paintBoxMouseRight.Canvas.Brush.Color := clGray;
end else begin
  paintBoxMouseRight.Canvas.Brush.Color := clWhite;
end;

// Выводим параметры только при изменении любого из них
if ((MouseBuffer.lX <> 0) or
    (MouseBuffer.lY <> 0) or
    (MouseBuffer.lZ <> 0)) then
begin
  labelDX.Caption := IntToStr(MouseBuffer.lX);
  labelDY.Caption := IntToStr(MouseBuffer.lY);
  labelDZ.Caption := IntToStr(MouseBuffer.lZ);
end;

labelX.Caption := IntToStr(MouseX);
labelY.Caption := IntToStr(MouseY);
labelZ.Caption := IntToStr(MouseZ);
paintBoxMouseLeft.Canvas.FillRect(
  paintBoxMouseLeft.Canvas.ClipRect);
paintBoxMouseMiddle.Canvas.FillRect(
```

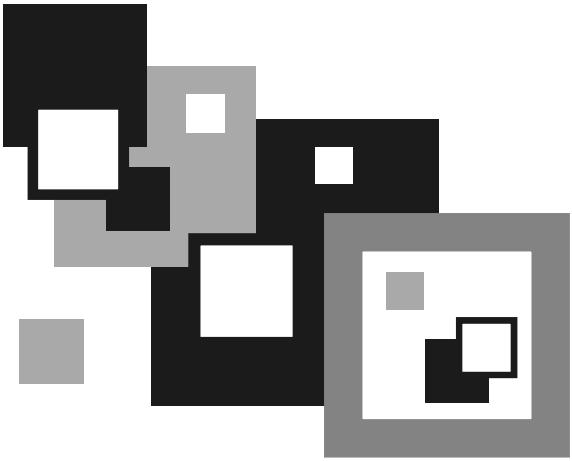
```
    paintBoxMouseMiddle.Canvas.ClipRect);
  paintBoxMouseRight.Canvas.FillRect(
    paintBoxMouseRight.Canvas.ClipRect);

  MouseX := MouseX + MouseBuffer.lX;
  MouseY := MouseY + MouseBuffer.lY;
  MouseZ := MouseZ + MouseBuffer.lZ;
end;

// Опрос джойстика
if SUCCEEDED(InputManager.GetJoystickState(@JoyBuffer)) then
begin
  dx := trunc(paintBoxJoystick.Width * (JoyBuffer.lX / 65535));
  dy := trunc(paintBoxJoystick.Height * (JoyBuffer.lY / 65535));
  dz := trunc(10 * (JoyBuffer.lZ / 65535));

  // При изменении любого параметра перерисовываем изображение
  if (dx <> LastX) or (dy <> LastY) or (dz <> LastZ) or
     (ButtonFireFirst <> (JoyBuffer.rgbButtons[1] > 0)) or
     (ButtonFireSecond <> (JoyBuffer.rgbButtons[0] > 0)) then
  begin
    paintBoxJoystick.Canvas.Brush.Color := clWhite;
    paintBoxJoystick.Canvas.FillRect(
      paintBoxJoystick.Canvas.ClipRect);

    if JoyBuffer.rgbButtons[1] > 0 then
    begin
      paintBoxJoystick.Canvas.Brush.Color := clYellow;
      paintBoxJoystick.Canvas.Ellipse(Rect(dx - dz - 7,
        dy - dz - 7, dx + dz + 7, dy + dz + 7));
    end;

    if JoyBuffer.rgbButtons[0] > 0 then
    begin
      paintBoxJoystick.Canvas.Brush.Color := clBlue;
      paintBoxJoystick.Canvas.Ellipse(Rect(dx - dz - 4,
```

```
          dy - dz - 4, dx + dz + 4, dy + dz + 4));
      end;

      paintBoxJoystick.Canvas.Brush.Color := clGray;
      paintBoxJoystick.Canvas.FillRect(Rect(dx - dz - 1,
        dy - dz - 1, dx + dz + 1, dy + dz + 1));
    end;

    LastX := dX;
    LastY := dY;
    LastZ := dZ;

    ButtonFireFirst  := JoyBuffer.rgbButtons[1] > 0;
    ButtonFireSecond := JoyBuffer.rgbButtons[0] > 0;
  end;
end;

{*****************************************************************}
{** Создание и инициализация объекта InputManager              **}
{*****************************************************************}
procedure TMainForm.FormCreate(Sender: TObject);
begin
  DoubleBuffered := TRUE;

  InputManager := TdxInputManager.Create(Handle);
  if FAILED(InputManager.Initialize) then
  begin
    FreeAndNil(InputManager);
    EXIT;
  end;

  InputManager.SetDeviceMask(idJoystick or idMouse);
  InputManager.SetDeviceMask(idJoystick or idMouse or idKeyboard);

  LastX := -1;
  LastY := -1;
  LastZ := -1;
```

```
  MouseX := 0;
  MouseY := 0;
  MouseZ := 0;

  ButtonFireFirst  := FALSE;
  ButtonFireSecond := FALSE;

  Application.OnIdle := ApplicationIdle;
end;

{*****************************************************************}
{** Удаление объекта InputManager                               **}
{*****************************************************************}
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  FreeAndNil(InputManager);
end;

{*****************************************************************}
{** Опрос устройств                                             **}
{*****************************************************************}
procedure TMainForm.ApplicationIdle(Sender: TObject; var Done: Boolean);
begin
  QueryDevices;

  Done := FALSE;
end;

END.
```

Итоги

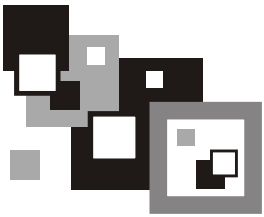
В этой части мы изучили работу с различными устройствами ввода, такими как клавиатура, мышь и джойстик. Были достаточно подробно рассмотрены интерфейсы `IDirectInput8` и `IDirectInputDevice8`, обеспечивающие работу с устройствами ввода. Был рассмотрен режим непосредственного получения данных от устройства и рассмотрен класс `TdxInputManager`, упрощающий работу с устройствами. Материала данной части читателю вполне достаточно для разработки собственных приложений с использованием подсистемы DirectInput.



ЧАСТЬ VI

DirectShow

Глава 13



Основные сведения

Область применения

Подсистема DirectShow представляет собой архитектуру, позволяющую работать с мультимедиапотоками на платформе Microsoft Windows. Мы можем смотреть фильмы, слушать музыку, записывать изображение с платы видеозахвата и звук с микрофона, даже не догадываясь о том, как и при помощи чего это происходит, и какие механизмы задействованы. А в большинстве случаев это происходит именно благодаря подсистеме DirectShow. Мы можем работать с мультимедиапотоком, встраивать собственные обработчики данных, именуемые фильтрами, и даже разрабатывать собственные форматы мультимедиаданных.

Одной из основных областей применения данной технологии является, несомненно, разработка систем безопасности. Запись аудио и видео с целью последующего анализа на сегодняшний день является практически самым эффективным способом борьбы с различными правонарушениями. Детекторы движения, детекторы оставленных предметов, распознавание образов — все это и многое другое используется практически на всех крупных промышленных объектах и даже частными лицами с целью охраны от посягательства со стороны третьих лиц.

Поддерживаемые форматы

Из документации Microsoft DirectX 9c SDK следует, что подсистема DirectShow имеет открытую архитектуру и будет поддерживать любой мультимедиаформат так долго, пока будут существовать фильтры для грамматического разбора и декодирования данного формата.

Поддерживаемые форматы файлов:

- Windows Media® Audio (WMA);

- Windows Media® Video (WMV);
- Advanced Systems Format (ASF);
- Motion Picture Experts Group (MPEG);
- Audio-Video Interleaved (AVI);
- QuickTime (version 2 and lower);
- WAV;
- AIFF;
- AU;
- SND;
- MIDI.

Поддерживаемые форматы сжатия:

- Windows Media Video;
- ISO MPEG-4 video version 1.0;
- Microsoft MPEG-4 version 3;
- Sipro Labs ACELP;
- Windows Media Audio;
- MPEG Audio Layer-3 (MP3) (decompression only);
- Digital Video (DV);
- MPEG-1 (decompression only);
- MJPEG;
- Cinepak.

Фильтры и граф фильтров

Основной единицей в разработке приложений с использованием DirectShow является фильтр. *Фильтр* (filter) — это программный компонент, который встраивается в поток мультимедиаданных и может выполнять определенные действия:

- читать данные из файла;
- получать видео непосредственно с источника;
- декодировать различные мультимедиаформаты, как, например, формат MPEG-1;
- передавать данные на графическую или звуковую плату.

Фильтр получает данные на входе, выполняет с ними определенные действия и передает их дальше. Вот пример: если фильтр декодирует поток видео в формате MPEG-1, то на входе идет закодированный поток в формате MPEG, а на выходе будет набор несжатых кадров.

Наше приложение может содержать целую серию соединенных определенным образом фильтров (будь то стандартные фильтры DirectShow или написанные самостоятельно), когда выход одного фильтра является входом для другого. Такой набор объединенных воедино фильтров называется *графом фильтров* (Filter Graph).

На рис. 13.1 приведен пример графа фильтров для просмотра файла в формате AVI.

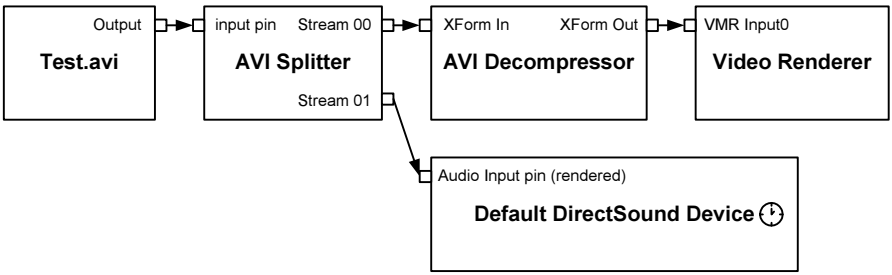


Рис. 13.1. Пример графа фильтров

Первый фильтр читает данные с диска (файл Test.avi) и передает их дальше. Второй фильтр (AVI Splitter) разбивает данные на два потока — поток сжатого видео и поток аудиоданных. Сжатое видео попадает в фильтр декомпрессии (AVI Decompressor), который распаковывает кадры и передает их фильтру, выводящему кадры на экран посредством DirectDraw или GDI (Video Renderer). Аудиоданные передаются фильтру, воспроизводящему звук при помощи подсистемы DirectSound (Default DirectSound Device).

Приложение не должно управлять всем этим потоком данных напрямую. Для этого существует специальный высокоуровневый компонент, именуемый *менеджером графа фильтров* (Filter Graph Manager). Приложение делает лишь высокоуровневые вызовы, такие как Run или Stop, которые запускают и останавливают передачу данных в графе. Если потребуется более детальный контроль действий с потоком мультимедиаданных, то можно получить непосредственный доступ к фильтрам через COM-интерфейсы. Также менеджер графа фильтров может отсылать различные уведомления вашему приложению (например, уведомление при достижении конца видеофайла при просмотре, чтобы ваше приложение установило его позицию на начало).

Точка соединения фильтров является также COM-объектом, именуемым *контактом* (Pin). Перевод с английского "pin" как "контакт" достаточно вольный и может не совпадать с чьей-то точкой зрения. Вариантами перевода могут также служить слова "штырек", "вывод", "штекер", "пин" и т. п.

Контакты используются фильтрами для передачи данных от одного фильтра к последующему. На рис. 13.1 стрелками показаны направления передачи данных между фильтрами.

Фильтр может находиться в одном из трех состояний: запущен, остановлен и в состоянии паузы. В запущенном состоянии фильтр обрабатывает поступающие данные. Когда он останавливается, то перестает обрабатывать данные. Смысл состояния паузы в том, чтобы послать сигнал графу так, чтобы команда запуска графа была обработана немедленно.

Типы фильтров

Все фильтры можно разбить по определенным параметрам на следующие категории:

- *фильтры-источники* (Source Filter) — передают данные в граф из какого-либо источника данных. Источником данных может быть файл на диске, сеть, устройство захвата изображения и т. п. Разные фильтры-источники оперируют различными типами источников данных;
- *преобразующие фильтры* (Transform Filter) — получают данные от источника, обрабатывают их и создают выходной поток данных. Простым примером трансформационных фильтров могут служить кодировщик и декодер;
- *фильтры вывода* (Renderer Filter) — располагаются в конце цепи графа. Они получают данные и представляют их пользователю. Пример — фильтр вывода изображения на экран (Video Renderer), который рисует кадры на экране; фильтр воспроизведения звука (Default DirectSound Device), который работает со звуковой платой, и, наконец, фильтр, записывающий данные в файл;
- *разделяющие фильтры* (Splitter Filter) — разделяют входящий поток данных на два и более потоков. Простой пример — разделение мультимедиапотока фильтром на потоки видео- и аудиоданных;
- *мультиплексор* (Mux Filter) — прямая противоположность разделяющему фильтру. Данный фильтр получает на входе два потока и более и объединяет их в один.

Менеджер графа фильтров

Мы с вами уже ввели определение данного компонента DirectShow. *Менеджер графа фильтров* представляет собой COM-объект, позволяющий контролировать фильтры в графе фильтров, и предоставляет множество функций, таких как:

- координация смены состояний фильтров;
- установка ссылочных часов;
- передача событий приложению;
- обеспечение методов построения графа фильтров приложением.

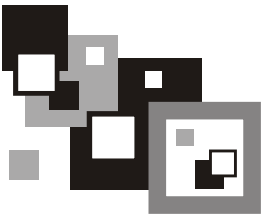
А теперь рассмотрим каждый пункт подробнее.

Смена состояний фильтров должна производиться в конкретном порядке. Следовательно, приложения сами по себе не могут управлять изменением состояния фильтров напрямую. Поэтому, получив одну-единственную команду, менеджер графа фильтров выполнит ее применительно ко всем фильтрам в нужном порядке.

Все фильтры в графе используют одни и те же часы, именуемые ссылочными часами. Ссылочные часы отвечают за синхронизацию потоков данных.

Менеджер графа фильтров использует очередь событий для информирования приложения о произошедших в графе событиях. Данный механизм очень похож на механизм оконных сообщений Windows.

Методы, предоставляемые приложению менеджером графа фильтров, позволяют добавлять фильтры в граф, соединять фильтры между собой и разъединять их.



Глава 14

Работа с MP3, AVI, MPEG и другими мультимедиаформатами

Интерфейсы

Подсистема DirectShow состоит из огромного количества интерфейсов, из которых нам для вывода видеопотока и аудио потребуется только небольшое их количество. В предыдущей главе нами были рассмотрены такие понятия, как фильтр, контакт, граф фильтров и менеджер графа фильтров. Все эти понятия описываются определенными интерфейсами:

- `IBaseFilter` — интерфейс управления фильтром;
- `IPin` — интерфейс управления контактом;
- `IGraphBuilder` — интерфейс для построения графа фильтров;
- `IMediaControl` — обеспечивает возможность управления графом фильтров.

Помимо трех этих интерфейсов нам в дальнейшем потребуется еще несколько:

- интерфейс управления позиционированием в потоке (`IMediaSeeking`);
- интерфейс управления выводом звука (`IBasicAudio`);
- интерфейс управления механизмом событий (`IMediaEventEx`);
- интерфейс управления выводом видеоданных (`IVideoWindow`);
- интерфейс перехвата кадра из потока видео (`ISampleGrabber`).

Далее будут рассмотрены все перечисленные интерфейсы более детально.

Интерфейс управления фильтром

Интерфейс `IBaseFilter` содержит следующие методы:

- `EnumPins` — перечисление всех контактов (входов и выходов) фильтра;
- `FindPin` — поиск контакта по идентификатору;
- `QueryFilterInfo` — получение информации о фильтре;
- `JoinFilterGraph` — уведомление фильтра о присоединении или отсоединении от графа;
- `QueryVendorInfo` — получение информации о поставщике фильтра.

Из всех перечисленных методов мы будем в дальнейшем использовать только два.

Это перечисление всех контактов:

```
function EnumPins(
  out ppEnum: IEnumPins):
HResult; stdcall;
```

Здесь `ppEnum` — адрес переменной, в которую будет занесен объект с интерфейсом `IEnumPins`, предоставляющим удобный способ навигации по контактам.

И поиск контакта по идентификатору:

```
function FindPin(
  Id: PWideChar;
  out ppPin: IPin):
HResult; stdcall;
```

Здесь:

- `Id` — идентификатор контакта;
- `ppPin` — адрес переменной, в которую будет занесен объект с интерфейсом `IPin`.

Интерфейс управления контактом

Перечислим все методы интерфейса `IPin`, но в явном виде не будем их использовать в своей работе:

- `Connect` — подключение одного контакта к другому;
- `ReceiveConnection` — принимает соединение от другого контакта;

- `Disconnect` — разрывает текущее соединение;
- `ConnectedTo` — получение контакта, соединенного с текущим;
- `ConnectionMediaType` — получение типа данных текущего контакта;
- `QueryPinInfo` — получение информации о контакте (название, родительский фильтр, направление);
- `QueryDirection` — получение направления контакта (вход или выход);
- `QueryId` — получение идентификатора контакта;
- `QueryAccept` — проверка допустимости указанного типа данных текущим контактом;
- `EnumMediaTypes` — перечисление допустимых типов данных контакта;
- `QueryInternalConnections` — получение контактов, присоединенных к текущему;
- `EndOfStream` — уведомление контакта о конце потока данных;
- `BeginFlush` — начало операции сброса данных;
- `EndFlush` — окончание операции сброса данных;
- `NewSegment` — оповещение контакта о том, что полученный сегмент данных после данного вызова будет сгруппирован как сегмент.

Интерфейс построения графа фильтров

Интерфейс `IGraphBuilder` служит для построения графа фильтров и содержит следующие методы:

- `Connect` — соединяет между собой два контакта;
- `Render` — добавляет цепь фильтров к специфическому выходному контакту для воспроизведения;
- `RenderFile` — строит граф фильтров для воспроизведения указанного файла;
- `AddSourceFilter` — добавляет фильтр-источник к графу для указанного файла;
- `SetLogFile` — установка файла протоколирования различных операций;
- `Abort` — указываем объекту построения графа на завершение текущей задачи сразу, как только это станет возможным;
- `ShouldOperationContinue` — запрос возможности продолжения текущей операции.

Интерфейс управления графом фильтров

Интерфейс `IMediaControl` предоставляет ряд методов для управления графом фильтров:

- `Run` — запуск всех фильтров графа;
- `Pause` — приостановка всех фильтров в графе;
- `Stop` — остановка всех фильтров в графе;
- `GetState` — получение состояния графа фильтров;
- `RenderFile` — не документировано. Предназначено для Visual Basic;
- `AddSourceFilter` — не документировано. Предназначено для Visual Basic;
- `get_FilterCollection` — не документировано. Предназначено для Visual Basic;
- `get_RegFilterCollection` — не документировано. Предназначено для Visual Basic;
- `StopWhenReady` — приостановка работы графа с разрешением фильтрам образовать очередь и после этого остановка работы графа.

Основными для нас, на данном этапе, являются всего три метода:

- запуск фильтров графа:
```
function Run: HResult; stdcall;
```
- приостановка всех фильтров в графе:
```
function Pause: HResult; stdcall;
```
- остановка всех фильтров в графе:
```
function Stop: HResult; stdcall;
```

Интерфейс управления позиционированием в потоке

Управлять позиционированием и скоростью воспроизведения потока данных нам помогает интерфейс `IMediaSeeking`. Рассмотрим более детально его методы:

- `GetCapabilities` — получение всех возможностей мультимедиапотока;
- `CheckCapabilities` — проверка поддержки потоком какой-либо возможности;
- `IsFormatSupported` — проверка поддержки временного формата применительно к операциям поиска;

- `QueryPreferredFormat` — получение предпочтительного формата для операций поиска;
- `GetTimeFormat` — получение временного формата, используемого в настоящий момент времени;
- `IsUsingTimeFormat` — проверка, используют ли операции поиска указанный временной формат;
- `SetTimeFormat` — установка указанного временного формата для последующих операций поиска;
- `GetDuration` — получение длительности воспроизведения потока;
- `GetStopPosition` — получение времени относительно длины потока, в течение которого воспроизведение будет остановлено;
- `GetCurrentPosition` — получение текущей позиции относительно общей длины потока;
- `ConvertTimeFormat` — преобразование между временными форматами;
- `SetPositions` — установка текущей позиции и позиции остановки;
- `GetPositions` — получение текущей позиции и позиции остановки относительно общей длины потока;
- `GetAvailable` — получение временного интервала, в котором поиск будет эффективным;
- `SetRate` — установка скорости воспроизведения потока;
- `GetRate` — получение скорости воспроизведения потока;
- `GetPreroll` — получение количества данных, которые будут стоять перед позицией старта.

Нами в процессе работы будут использоваться далеко не все методы. Рассмотрим эти методы более детально.

Проверка поддержки временного формата:

```
function IsFormatSupported(
  const pFormat: TGUID):
HResult; stdcall;
```

Здесь `Format` — указатель на уникальный идентификатор формата. Может принимать одно из следующих значений:

- `TIME_FORMAT_NONE` — формат отсутствует;
- `TIME_FORMAT_FRAME` — формат кадров изображения;
- `TIME_FORMAT_SAMPLE` — формат образцов потока;

- `TIME_FORMAT_FIELD` — чередующиеся области видео;
- `TIME_FORMAT_BYTE` — смещение в байтах относительно потока;
- `TIME_FORMAT_MEDIA_TIME` — ссылочный временной формат (100 нс).

Установка указанного временного формата:

```
function SetTimeFormat(
  const pFormat: TGUID):
HResult; stdcall;
```

Здесь `pFormat` — указатель на уникальный идентификатор формата. Принимаемые значения описаны выше.

Получение длительности воспроизведения потока:

```
function GetDuration(
  out pDuration: int64):
HResult; stdcall;
```

Здесь `pDuration` — переменная, в которую будет занесена длительность потока.

Получение текущей позиции и позиции остановки:

```
function GetPositions(
  out pCurrent,
  pStop: int64):
HResult; stdcall;
```

Здесь:

- `pCurrent` — переменная, в которую будет занесена текущая позиция воспроизведения;
- `pStop` — переменная, в которую будет занесена позиция остановки.

Установка текущей позиции и позиции остановки:

```
function SetPositions(
  var pCurrent: int64;
  dwCurrentFlags: DWORD;
  var pStop: int64;
  dwStopFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `pCurrent` — текущая позиция воспроизведения;

- ❒ `dwCurrentFlags` — битовая комбинация флагов:
  - флаги позиционирования:
    - ◊ `AM_SEEKING_NoPositioning` — не менять позицию;
    - ◊ `AM_SEEKING_AbsolutePositioning` — абсолютное позиционирование;
    - ◊ `AM_SEEKING_RelativePositioning` — относительное позиционирование;
    - ◊ `AM_SEEKING_IncrementalPositioning` — позиция остановки относительно текущей позиции;
  - флаги модификации:
    - ◊ `AM_SEEKING_SeekToKeyFrame` — поиск ближайшего ключевого кадра;
    - ◊ `AM_SEEKING_ReturnTime` — вернуть эквивалентное ссылочное время;
    - ◊ `AM_SEEKING_Segment` — использовать сегментарный поиск;
    - ◊ `AM_SEEKING_NoFlush` — не использовать сброс данных;
- ❒ `pStop` — позиция остановки;
- ❒ `dwStopFlags` — аналогичен параметру `dwCurrentFlags`.

Получение скорости воспроизведения потока:

```
function GetRate(
  out pdRate: double):
HResult; stdcall;
```

Здесь `pdRate` — переменная, в которую будет занесена текущая скорость воспроизведения потока.

Установка скорости воспроизведения потока:

```
function SetRate(
  dRate: double):
HResult; stdcall;
```

Здесь `dRate` — скорость воспроизведения потока.

Получение текущей позиции:

```
function GetCurrentPosition(
  out pCurrent: int64):
HResult; stdcall;
```

Здесь `pCurrent` — текущая позиция воспроизведения.

Интерфейс управления выводом звука

В процессе воспроизведения мультимедиапотока нам может потребоваться произвести ряд настроек звукового потока, таких как громкость и баланс.

Для этого имеется специальный интерфейс `IBasicAudio`. Мы будем использовать все методы данного интерфейса — всего их четыре:

- `put_Volume` — установка громкости;
- `get_Volume` — получение значения громкости;
- `put_Balance` — установка баланса;
- `get_Balance` — получение значения баланса.

Установка уровня громкости:

```
function put_Volume(
  lVolume: Longint):
HResult; stdcall;
```

Здесь `lVolume` — уровень громкости.

Получение значения уровня громкости:

```
function get_Volume(
  out plVolume: Longint):
HResult; stdcall;
```

Здесь `plVolume` — переменная, в которую будет занесено текущее значение уровня громкости потока.

Установка уровня баланса:

```
function put_Balance(
  lBalance: Longint):
HResult; stdcall;
```

Здесь `lBalance` — уровень баланса.

Получение значения уровня баланса:

```
function get_Balance(
  out plBalance: Longint):
HResult; stdcall;
```

Здесь `plBalance` — переменная, в которую будет занесено текущее значение уровня баланса.

Интерфейс управления механизмом событий

Интерфейс управления потоком событий `IMediaEventEx` происходит от интерфейса `IMediaEvent`, который и содержит методы получения уведомлений и предоставляет возможность переопределения обработчиков событий гра-

фа. Сам же интерфейс `IMediaEventEx` добавляет возможность установки окна для получения событий.

В отличие от интерфейса `IMediaEvent`, данный интерфейс не поддерживает автоматизацию и, следовательно, не может быть использован из среды Microsoft Visual Basic (к среде разработки Borland Delphi это не относится).

Интерфейс содержит всего три метода:

- `SetNotifyWindow` — регистрация окна обработки сообщений;
- `SetNotifyFlags` — включение или отключение обработки событий;
- `GetNotifyFlags` — получение статуса обработки событий.

Остальные методы унаследованы от интерфейса `IMediaEvent`:

- `GetEventHandle` — получение указателя объекта события с ручным управлением, которое подает сигнал при наличии элементов в очереди обработки;
- `GetEvent` — получение следующего события из очереди;
- `WaitForCompletion` — ожидание графа фильтров для обработки всех данных;
- `CancelDefaultHandling` — отказ от собственной обработки менеджером графа фильтров указанного события;
- `RestoreDefaultHandling` — возобновление собственной обработки менеджером графа фильтров указанного события;
- `FreeEventParams` — освобождение ресурсов, связанных с параметрами события.

Несколько методов требуют более детального рассмотрения, т. к. будут непосредственно использованы нами в дальнейшем.

Регистрация окна обработки сообщений:

```
function SetNotifyWindow(
  hwnd: OAHWND;
  lMsg: Longint;
  lInstanceData: Longint):
HResult; stdcall;
```

Здесь:

- `hwnd` — дескриптор окна обработки сообщений, либо нулевое значение для остановки обработки события;
- `lMsg` — обрабатываемое сообщение;
- `lInstanceData` — значение `lParam`, которое будет передано обработчику сообщения.

Получение следующего события из очереди:

```
function GetEvent(
  out lEventCode: Longint;
  out lParam1,
      lParam2: Longint;
  msTimeout: DWORD):
HResult; stdcall;
```

Здесь:

- `lEventCode` — переменная, которая получает код события;
- `lParam1`, `lParam2` — переменные, в которые будут переданы параметры события;
- `msTimeout` — интервал времени ожидания (мс) при получении сообщения.

Освобождение ресурсов, связанных с параметрами события:

```
function FreeEventParams(
  lEvCode: Longint;
  lParam1,
  lParam2: Longint):
HResult; stdcall;
```

Здесь:

- `lEvCode` — код события;
- `lParam1`, `lParam2` — параметры события.

Интерфейс управления выводом видеоданных

Если воспроизводимый мультимедиапоток содержит видеоданные, то они будут отображены в отдельном окне, как показано на рис. 14.1.

Конечно, это не совсем удобно, и наверняка вам захочется встроить окно просмотра видео в какое-либо место своего приложения. Для этого и предусмотрен интерфейс `IVideoWindow`.

Интерфейс содержит достаточно большое количество методов, из всего многообразия которых нам потребуются четыре:

- `put_Owner` — установка родительского окна;
- `put_WindowStyle` — установка стиля окна;
- `put_Visible` — отобразить или спрятать окно;
- `SetWindowPosition` — установить положение окна.

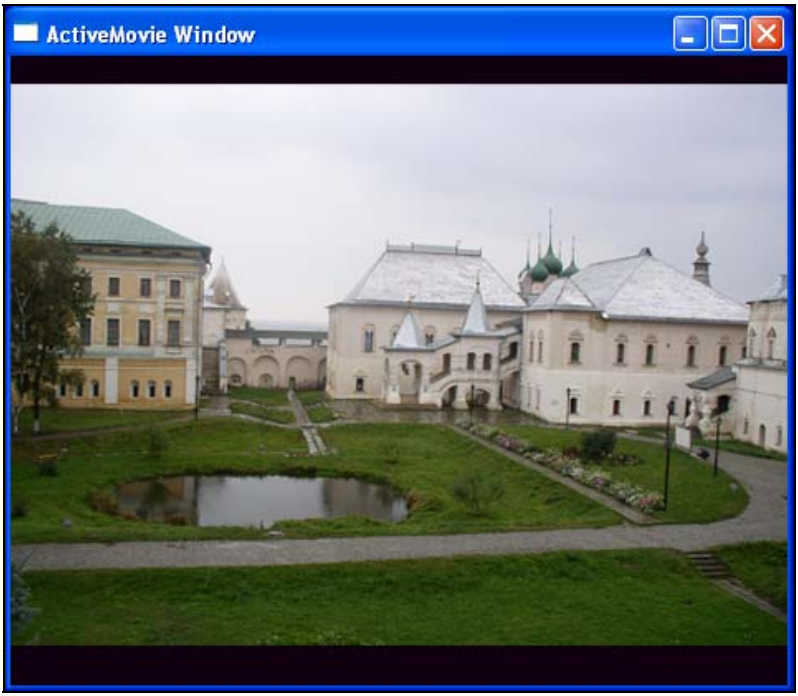


Рис. 14.1. Окно просмотра видео

Установка родительского окна:

```
function put_Owner(
  Owner: OAHWND):
HResult; stdcall;
```

Здесь `Owner` — дескриптор родительского окна.

Установка стиля окна:

```
function put_WindowStyle(
  WindowStyle: Longint):
HResult; stdcall;
```

Здесь `WindowStyle` — стиль окна.

Отобразить или спрятать окно:

```
function put_Visible(
  Visible: LongBool):
HResult; stdcall;
```

Здесь `Visible` — флаг, в зависимости от значения которого окно будет либо отображено, либо скрыто.

Установить положение окна:

```
function SetWindowPosition(
  Left,
  Top,
  Width,
  Height: Longint):
HResult; stdcall;
```

Здесь:

- `Left` — координата *x* окна;
- `Top` — координата *y* окна;
- `Width` — ширина окна;
- `Height` — высота окна.

Интерфейс перехвата кадра из потока видео

Зачастую, просматривая на компьютере какой-то фильм или музыкальный клип, мы задаемся мыслью получить текущий кадр из видеопотока. Первой мыслью становится нажатие стандартной для системы Microsoft Windows клавиши <Print Screen> (для копирования в буфер обмена изображения снимка рабочего стола со всеми запущенными приложениями), либо комбинации клавиш <Alt>+<Print Screen> (для копирования в буфер обмена изображения активного окна приложения) с целью последующей обработки в любом графическом редакторе. К сожалению, в зависимости от режима воспроизведения видео это нам удается далеко не всегда.

Но в подсистеме DirectShow существует специальный интерфейс `ISampleGrabber`, который нам существенно все упрощает. Данный интерфейс представляет собой стандартный фильтр DirectShow. Суть его работы заключается в том, что, будучи встроен в цепочку фильтров графа следом за фильтром декомпрессии, он получает распакованные кадры изображения и позволяет нам копировать их в свой буфер с целью последующей обработки.

На рис. 14.2 приведен пример использования данного фильтра в графе. Фильтр перехвата изображения в примере называется GRABBER.

Если предположить, что мы попытаемся применить подобный фильтр не к потоку видео, а к потоку аудиоданных, то не произойдет никакой ошибки — наш фильтр просто не будет встроен в граф и окажется не у дел, как показано на рис. 14.3.

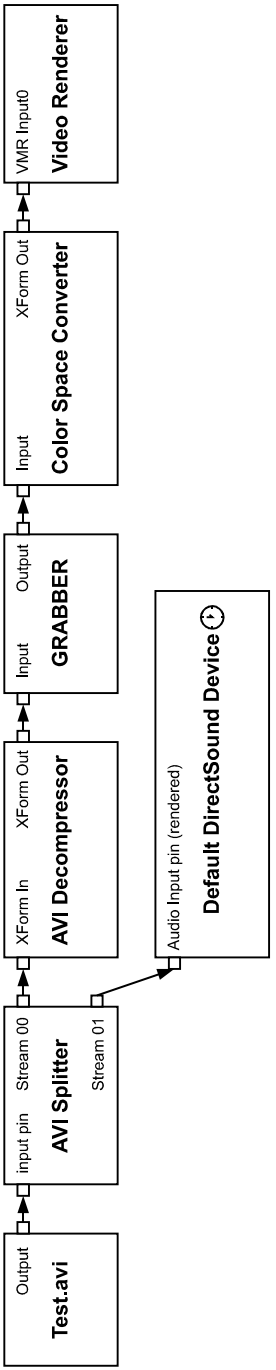


Рис. 14.2. Пример графа со встроенным фильтром перехвата

Рис. 14.3. Пример графа для воспроизведения MP3 с "выпавшим" фильтром перехвата

Перечислим методы интерфейса `ISampleGrabber` и разберем более детально некоторые из них:

- `SetOneShot` — указываем останавливать ли работу графа после получения кадра фильтром перехвата;
- `SetMediaType` — устанавливаем формат данных контакта на входе фильтра;
- `GetConnectedMediaType` — получаем формат данных контакта на входе фильтра;
- `SetBufferSamples` — указываем, копировать ли данные в буфер в том виде, в котором они проходили через фильтр перехвата;
- `GetCurrentBuffer` — получение текущего кадра изображения в буфер, а также его размера;
- `GetCurrentSample` — к настоящему времени метод не реализован;
- `SetCallback` — установка механизма обратного вызова при поступлении новых кадров.

Метод установки режима работы графа после срабатывания фильтра перехвата:

```
function SetOneShot(
  OneShot: BOOL):
HResult; stdcall;
```

Здесь `OneShot` — булево значение, указывающее, останавливать работу графа или нет.

Установка формата данных контакта на входе фильтра:

```
function SetMediaType(
  var pType: TAMMediaType):
HResult; stdcall;
```

Здесь `pType` — структура `TAMMediaType`, описывающая нужный формат.

Получение формата данных контакта на входе фильтра:

```
function GetConnectedMediaType(
  out pType: TAMMediaType):
HResult; stdcall;
```

Здесь `pType` — структура `TAMMediaType`, в которую будет записан текущий формат.

Указываем, копировать ли данные в буфер в том виде, в котором они проходили через фильтр перехвата:

```
function SetBufferSamples(
  BufferThem: BOOL):
HResult; stdcall;
```

Здесь `BufferThem` — булево значение, определяющее способ записи данных в буфер.

Получаем текущий кадр изображения в буфер, а также его размер:

```
function GetCurrentBuffer(
  var pBufferSize: longint;
  pBuffer: Pointer):
HResult; stdcall;
```

Здесь:

- `pBufferSize` — переменная, указывающая размер буфера. Если `pBuffer` имеет нулевое значение, то в данную переменную будет занесен текущий размер кадра. Если значение `pBuffer` отлично от нуля, то в данной переменной должен быть указан реальный размер буфера в байтах. На выходе из функции значением переменной станет реальное число байтов, которые были записаны в буфер, и это значение может быть меньше размера буфера.
- `pBuffer` — массив байтов размером `pBufferSize`, в который будет записано изображение, либо нулевое значение (для получения размера буфера).

Алгоритм работы

Мы рассмотрели все необходимые нам для работы интерфейсы, и пришло время наконец-то разобраться в порядке работы с ними. Мы рассмотрим все действия по порядку.

1. Создаем объект для построения графа фильтров:
   ```
   Result := CoCreateInstance(CLSID_FilterGraph, NIL,
     CLSCTX_INPROC_SERVER, IID_IGraphBuilder, FGraphBuilder);
   ```
2. Создаем фильтр перехвата:
   ```
   Result := CoCreateInstance(CLSID_SampleGrabber, NIL,
     CLSCTX_INPROC_SERVER, IID_IBaseFilter, FBaseFilter);
   ```
3. Получаем интерфейс фильтра перехвата:
   ```
   FBaseFilter.QueryInterface(IID_ISampleGrabber, FSampleGrabber);
   ```
4. Добавляем фильтр в наш граф:
   ```
   Result := FGraphBuilder.AddFilter(FBaseFilter, 'GRABBER');
   ```
5. Настраиваем фильтр перехвата:
   ```
   // Устанавливаем формат данных для фильтра перехвата
   ZeroMemory(@MediaType, sizeof(TAMMediaType));
   ```

```
with MediaType do
begin
  majortype  := MEDIATYPE_Video;
  subtype    := MEDIASUBTYPE_RGB24;
  formattype := FORMAT_VideoInfo;
end;

Result := FSampleGrabber.SetMediaType(MediaType);
if FAILED(Result) then EXIT;

// Данные будут записаны в буфер в том виде, в котором они
// проходят через фильтр
FSampleGrabber.SetBufferSamples(TRUE);

// Граф не будет остановлен для получения кадра
FSampleGrabber.SetOneShot(FALSE);
```

6. Запрашиваем интерфейс управления графом фильтров:
```
FGraphBuilder.QueryInterface(IID_IMediaControl, FMediaControl);
```
7. Запрашиваем интерфейс управления окном вывода видео:
```
FGraphBuilder.QueryInterface(IID_IVideoWindow,  FVideoWindow);
```
8. Запрашиваем интерфейс управления позиционированием мультимедиа-потока:
```
FGraphBuilder.QueryInterface(IID_IMediaSeeking, FMediaSeeking);
```
9. Запрашиваем интерфейс управления звуковым потоком:
```
FGraphBuilder.QueryInterface(IID_IBasicAudio,   FBasicAudio);
```
10. Строим граф фильтров для нашего файла:
```
Result := FGraphBuilder.RenderFile(PWideChar(FFileName), NIL);
```
11. В зависимости от типа поддерживаемого формата настраиваем различные параметры воспроизведения:
```
// Проверка поддерживаемых форматов
if (FMediaSeeking.IsFormatSupported(TIME_FORMAT_FRAME) = S_OK) then
begin
  // Устанавливаем покадровый формат
  FMediaSeeking.SetTimeFormat(TIME_FORMAT_FRAME);
  // Получаем число кадров
  FMediaSeeking.GetDuration(FFrameCount);
```

```
    // Выводим сообщение на окне с видео
    SetWindowText(FHandle, PAnsiChar(strVideoLoaded));
    // Обновляем окно видео
    InvalidateRect(FHandle, NIL, FALSE);

    // Устанавливаем родительское окно для вывода изображения
    FVideoWindow.put_Owner(FHandle);
    // Устанавливаем стиль видео окна
    FVideoWindow.put_WindowStyle(WS_CHILD or WS_CLIPSIBLINGS);

    FIsTimeFormat := FALSE;
  end else if (FMediaSeeking.IsFormatSupported(TIME_FORMAT_MEDIA_TIME) =
               S_OK) then
  begin
    // Устанавливаем ссылочный временной формат (100 нс)
    FMediaSeeking.SetTimeFormat(TIME_FORMAT_MEDIA_TIME);

    // Получаем число интервалов
    FMediaSeeking.GetDuration(FFrameCount);
    // Сокращаем их количество для более удобной работы
    FFrameCount := FFrameCount div frameDuration;

    FIsTimeFormat := TRUE;
  end;
```

12. Управляем воспроизведением графа (интерфейс `IMediaControl`), позицией воспроизведения (интерфейс `IMediaSeeking`), фильтром перехвата изображения (`ISampleGrabber`) и т. д.
13. Управляем аудиопотоком (интерфейс `IBasicAudio`).
14. Управляем механизмом сообщений (интерфейс `IMediaEventEx`).
15. В конце работы освобождаем память.

Класс *TdxMediaPlayer*

Все описанные выше шаги реализованы в специальном классе `TdxMediaPlayer`. Данный класс представляет собой минимально необходимую функциональность для упрощения разработки приложений, в которых

требуется воспроизведение различных мультимедиапотоков. Потоки, которые могут воспроизводиться подсистемой DirectShow, были описаны выше. К ним можно отнести такие форматы данных, как WAV, MIDI и MP3, форматы видео AVI, WMA, MPEG и др.

Класс расположен в модуле UdxMediaPlayer.pas (листинг 14.1) каталога Classes на прилагаемом к книге компакт-диске.

К возможностям данного класса можно отнести следующее:

- воспроизведение большого количества форматов данных (аудио и видео);
- не требуется никаких дополнительных настроек — после метода инициализации (в котором задается путь к файлу с данными) можно сразу начинать воспроизведение мультимедиапотока;
- помимо метода воспроизведения потока, имеются методы приостановки воспроизведения, полной остановки воспроизведения, методы позиционирования в потоке данных, перемотки по кадрам вперед и назад, управления скоростью воспроизведения потока, имеется метод получения текущего кадра (только для потока видеоданных), методы управления потоком аудиоданных и окном вывода изображения и методы управления потоком событий.

Листинг 14.1. Текст модуля UdxMediaPlayer.pas

```
UNIT UdxMediaPlayer;
{*************************************************************************}
{** Воспроизведение мультимедиапотоков посредством DirectShow          **}
{** Автор: Есенин Сергей Анатольевич                                   **}
{*************************************************************************}

{**} INTERFACE {*********************************************************}

{**} USES {**************************************************************}
  Windows, SysUtils, Graphics, ComObj, ActiveX, DirectShow9, Dialogs;

{**} TYPE {**************************************************************}
  TdxMediaPlayer = class
  PRIVATE
    FGraphBuilder:  IGraphBuilder;
    FMediaControl:  IMediaControl;
    FVideoWindow:   IVideoWindow;
```

```
    FBaseFilter:    IBaseFilter;
    FSampleGrabber: ISampleGrabber;
    FMediaSeeking:  IMediaSeeking;
    FBasicAudio:    IBasicAudio;
    FMediaEvent:    IMediaEventEx;

    FFileName:      WideString;
    FPlaying:       boolean;
    FHandle:        THandle;
    FFrameCount:    int64;

    FIsTimeFormat:  boolean;

    function CreateGraph: HResult;
    procedure ResetGraph;

  PUBLIC
    property FrameCount: int64 read FFrameCount;

    constructor Create(AHandle: THandle);
    destructor Destroy; override;

    function Initialize(AFileName: WideString): HResult;

    function  Play: HResult;
    procedure Stop;
    procedure Pause;

    function  StepPrev: HResult;
    function  StepNext: HResult;

    function  Faster: HResult;
    function  Slower: HResult;

    function GetVolume(out Value: Longint): HResult;
    function SetVolume(Value: Longint): HResult;
```

```
    function GetBalance(out Value: Longint): HResult;
    function SetBalance(Value: Longint): HResult;

    function SetWindowPosition(const R: TRect): HResult;

    function GetPlayingPosition(out P: int64): HResult;
    function SetPlayingPosition(P: int64): HResult;

    function CaptureBitmap(FileName: string): HResult;

    function RegisterEventMessage(AHandle: THandle; Msg: Cardinal):
      HResult;
    function GetEvent(out lEventCode: Longint; out lParam1, lParam2:
      Longint; msTimeout: DWORD): HResult;
  END;

{**} IMPLEMENTATION {************************************************}

{**} CONST {*********************************************************}
  strVideoLoaded    = '!!! Видео загружено !!!';
  strVideoNotLoaded = '!!! Видео не загружено !!!';

  frameDuration     = 500000;

{**} { TdxMediaPlayer } {********************************************}

{*********************************************************************}
{** Конструктор класса                                              **}
{*********************************************************************}
constructor TdxMediaPlayer.Create(AHandle: THandle);
begin
  // Запоминаем указатель на окно вывода видео
  FHandle := AHandle;

  // Очищаем граф
  ResetGraph;
end;
```

```
{*************************************************************************}
{** Деструктор класса                                                   **}
{*************************************************************************}
destructor TdxMediaPlayer.Destroy;
begin
  // Очищаем граф
  ResetGraph;
end;

{*************************************************************************}
{** Инициализация                                                       **}
{*************************************************************************}
function TdxMediaPlayer.Initialize(AFileName: WideString): HResult;
begin
  // Очищаем граф
  ResetGraph;

  // Запоминаем путь к файлу
  FFileName := AFileName;

  // Строим граф
  CreateGraph;

  Result := S_OK;
end;

{*************************************************************************}
{** Построение графа                                                    **}
{*************************************************************************}
function TdxMediaPlayer.CreateGraph: HResult;
var
  MediaType: TAMMediaType;
begin
  // Результат по умолчанию
  Result := E_FAIL;
```

```
// Если не задан путь к файлу, то завершаем работу
if Trim(FFileName) = '' then EXIT;

// Создаем объект для построения графа фильтров
Result := CoCreateInstance(CLSID_FilterGraph, NIL,
  CLSCTX_INPROC_SERVER, IID_IGraphBuilder, FGraphBuilder);
if FAILED(Result) then EXIT;

// Создаем фильтр
Result := CoCreateInstance(CLSID_SampleGrabber, NIL,
  CLSCTX_INPROC_SERVER, IID_IBaseFilter, FBaseFilter);
if FAILED(Result) then EXIT;

// Получаем интерфейс фильтра перехвата
FBaseFilter.QueryInterface(IID_ISampleGrabber, FSampleGrabber);

// Добавляем фильтр в граф
Result := FGraphBuilder.AddFilter(FBaseFilter, 'GRABBER');
if FAILED(Result) then EXIT;

if FSampleGrabber <> NIL then
begin
  // Устанавливаем формат данных для фильтра перехвата
  ZeroMemory(@MediaType, sizeof(TAMMediaType));

  with MediaType do
  begin
    majortype  := MEDIATYPE_Video;
    subtype    := MEDIASUBTYPE_RGB24;
    formattype := FORMAT_VideoInfo;
  end;

  Result := FSampleGrabber.SetMediaType(MediaType);
  if FAILED(Result) then EXIT;
```

```
  // Данные будут записаны в буфер в том виде, в котором они
  // проходят через фильтр
  FSampleGrabber.SetBufferSamples(TRUE);

  // Граф не будет остановлен для получения кадра
  FSampleGrabber.SetOneShot(FALSE);
end;

// Запрашиваем интерфейс управления графом фильтров
FGraphBuilder.QueryInterface(IID_IMediaControl, FMediaControl);

// Запрашиваем интерфейс управления окном вывода видео
FGraphBuilder.QueryInterface(IID_IVideoWindow,  FVideoWindow);

// Запрашиваем интерфейс управления позиционированием медиапотока
FGraphBuilder.QueryInterface(IID_IMediaSeeking, FMediaSeeking);

// Запрашиваем интерфейс управления звуковым потоком
FGraphBuilder.QueryInterface(IID_IBasicAudio,   FBasicAudio);

// Строим граф фильтров для нашего файла
Result := FGraphBuilder.RenderFile(PWideChar(FFileName), NIL);
if FAILED(Result) then EXIT;

// Проверка поддерживаемых форматов
if (FMediaSeeking.IsFormatSupported(TIME_FORMAT_FRAME) = S_OK) then
begin
  // Устанавливаем покадровый формат
  FMediaSeeking.SetTimeFormat(TIME_FORMAT_FRAME);

  // Получаем число кадров
  FMediaSeeking.GetDuration(FFrameCount);
  // Выводим сообщение на окне с видео
  SetWindowText(FHandle, PAnsiChar(strVideoLoaded));
  // Обновляем окно видео
  InvalidateRect(FHandle, NIL, FALSE);
```

```
    // Устанавливаем родительское окно для вывода изображения
    FVideoWindow.put_Owner(FHandle);
    // Устанавливаем стиль видео окна
    FVideoWindow.put_WindowStyle(WS_CHILD or WS_CLIPSIBLINGS);

    FIsTimeFormat := FALSE;
  end else if (FMediaSeeking.IsFormatSupported(TIME_FORMAT_MEDIA_TIME) =
               S_OK) then
  begin
    // Устанавливаем ссылочный временной формат (100 нс)
    FMediaSeeking.SetTimeFormat(TIME_FORMAT_MEDIA_TIME);

    // Получаем число интервалов
    FMediaSeeking.GetDuration(FFrameCount);
    // Сокращаем их количество для более удобной работы
    FFrameCount := FFrameCount div frameDuration;

    FIsTimeFormat := TRUE;
  end;
end;

{**********************************************************************}
{** Чистка графа - освобождаем память и обнуляем свойства            **}
{**********************************************************************}
procedure TdxMediaPlayer.ResetGraph;
begin
  FBasicAudio    := NIL;
  FMediaEvent    := NIL;

  FPlaying       := FALSE;
  FFileName      := '';

  FMediaSeeking  := NIL;
  FSampleGrabber := NIL;
  FBaseFilter    := NIL;
```

```
  if FVideoWindow <> NIL then
  begin
    FVideoWindow.put_Visible(FALSE);
    FVideoWindow.put_Owner(0);
    FVideoWindow := NIL;
  end;

  FMediaControl  := NIL;
  FGraphBuilder  := NIL;

  SetWindowText(FHandle, PAnsiChar(strVideoNotLoaded));
  InvalidateRect(FHandle, NIL, FALSE);
end;

{*************************************************************************}
{** Запуск графа на воспроизведение                                     **}
{*************************************************************************}
function TdxMediaPlayer.Play: HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс управления, то завершаем работу
  if FMediaControl = NIL then EXIT;

  // Запускаем граф
  Result := FMediaControl.Run;

  FPlaying := SUCCEEDED(Result);
end;

{*************************************************************************}
{** Остановка воспроизведения и установка позиции на начало            **}
{*************************************************************************}
procedure TdxMediaPlayer.Stop;
begin
```

```
  // Проверка наличия интерфейса управления и состояния графа
  if (FMediaControl = NIL) or not (FPlaying) then EXIT;

  // Останавливаем граф
  FMediaControl.Stop;

  // Устанавливаем позицию на начало
  SetPlayingPosition(0);

  FPlaying := FALSE;
end;

{*************************************************************************}
{** Приостанавливаем воспроизведение потока                             **}
{*************************************************************************}
procedure TdxMediaPlayer.Pause;
begin
  // Если отсутствует интерфейс управления, то завершаем работу
  if FMediaControl = NIL then EXIT;

  // Переводим граф в состояние "пауза"
  FMediaControl.Pause;

  FPlaying := FALSE;
end;

{*************************************************************************}
{** Шаг назад                                                           **}
{*************************************************************************}
function TdxMediaPlayer.StepPrev: HResult;
var
  P, S: int64;
begin

  // Результат по умолчанию
  Result := E_FAIL;
```

```
  // Если отсутствует интерфейс позиционирования, то завершаем работу
  if FMediaSeeking = NIL then EXIT;

  // Устанавливаем позицию на шаг назад
  FMediaSeeking.GetPositions(P, S);
  dec(P);
  FMediaSeeking.SetPositions(P, AM_SEEKING_AbsolutePositioning,
    S, AM_SEEKING_NoPositioning);
end;

{*************************************************************************}
{** Шаг вперед                                                          **}
{*************************************************************************}
function TdxMediaPlayer.StepNext: HResult;
var
  P, S: int64;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс позиционирования, то завершаем работу
  if FMediaSeeking = NIL then EXIT;

  // Устанавливаем позицию на шаг вперед
  FMediaSeeking.GetPositions(P, S);
  inc(P);
  FMediaSeeking.SetPositions(P, AM_SEEKING_AbsolutePositioning,
    S, AM_SEEKING_NoPositioning);
end;

{*************************************************************************}
{** Увеличиваем скорость воспроизведения                                **}
{*************************************************************************}
function TdxMediaPlayer.Faster: HResult;
var
  Rate: double;
```

```
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс позиционирования, то завершаем работу
  if FMediaSeeking = NIL then EXIT;

  // Получаем текущую скорость воспроизведения...
  if SUCCEEDED(FMediaSeeking.GetRate(Rate)) then
  begin
    // ... и увеличиваем ее в два раза
    Result := FMediaSeeking.SetRate(Rate * 2);
  end;
end;

{*********************************************************************}
{** Уменьшаем скорость воспроизведения                              **}
{*********************************************************************}
function TdxMediaPlayer.Slower: HResult;
var
  Rate: double;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс позиционирования, то завершаем работу
  if FMediaSeeking = NIL then EXIT;

  // Получаем текущую скорость воспроизведения...
  if SUCCEEDED(FMediaSeeking.GetRate(Rate)) then
  begin
    // ... и уменьшаем ее в два раза
    Result := FMediaSeeking.SetRate(Rate / 2);
  end;
end;
{*********************************************************************}
```

```
{** Получаем громкость                                                **}
{**********************************************************************}
function TdxMediaPlayer.GetVolume(out Value: Integer): HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс управления звуком, то завершаем работу
  if FBasicAudio = NIL then EXIT;

  // Получаем громкость
  Result := FBasicAudio.get_Volume(Value);
end;

{**********************************************************************}
{** Устанавливаем громкость                                           **}
{**********************************************************************}
function TdxMediaPlayer.SetVolume(Value: Integer): HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс управления звуком, то завершаем работу
  if FBasicAudio = NIL then EXIT;

  // Устанавливаем громкость
  Result := FBasicAudio.put_Volume(Value);
end;

{**********************************************************************}
{** Получаем уровень баланса аудио                                    **}
{**********************************************************************}
function TdxMediaPlayer.GetBalance(out Value: Integer): HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;
```

```
  // Если отсутствует интерфейс управления звуком, то завершаем работу
  if FBasicAudio = NIL then EXIT;

  // Получаем уровень баланса аудио
  Result := FBasicAudio.get_Balance(Value);
end;

{*********************************************************************}
{** Устанавливаем уровень баланса аудио                             **}
{*********************************************************************}
function TdxMediaPlayer.SetBalance(Value: Integer): HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс управления звуком, то завершаем работу
  if FBasicAudio = NIL then EXIT;

  // Устанавливаем уровень баланса аудио
  Result := FBasicAudio.put_Balance(Value);
end;

{*********************************************************************}
{** Задаем позицию окна воспроизведения видео                       **}
{*********************************************************************}
function TdxMediaPlayer.SetWindowPosition(const R: TRect): HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс управления окном видео,
  // то завершаем работу
  if (FVideoWindow = NIL) or (FIsTimeFormat) then EXIT;

  // Задаем позицию окна воспроизведения видео
  Result := FVideoWindow.SetWindowPosition(R.Left, R.Top,
```

```
    R.Right - R.Left, R.Bottom - R.Top);
end;

{*************************************************************************}
{** Получаем текущую позицию воспроизведения                            **}
{*************************************************************************}
function TdxMediaPlayer.GetPlayingPosition(out P: int64): HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс позиционирования, то завершаем работу
  if FMediaSeeking = NIL then EXIT;

  // Получаем текущую позицию воспроизведения
  Result := FMediaSeeking.GetCurrentPosition(P);

  // Используем временной формат
  if FIsTimeFormat then P := P div frameDuration;
end;

{*************************************************************************}
{** Устанавливаем позицию воспроизведения                               **}
{*************************************************************************}
function TdxMediaPlayer.SetPlayingPosition(P: int64): HResult;
var
  PS, S: int64;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс позиционирования, то завершаем работу
  if FMediaSeeking = NIL then EXIT;

  // Получаем текущую позицию воспроизведения
  FMediaSeeking.GetPositions(PS, S);
```

```
  // Используем временной формат
  if FIsTimeFormat then P := P * frameDuration;

  // Устанавливаем позицию воспроизведения
  Result := FMediaSeeking.SetPositions(P, AM_SEEKING_AbsolutePositioning,
    S, AM_SEEKING_NoPositioning);
end;

{*************************************************************************}
{** Получаем кадр из видеопотока и сохраняем на диске                   **}
{*************************************************************************}
function TdxMediaPlayer.CaptureBitmap(FileName: string): HResult;
var
  bSize: integer;
  pVideoHeader: TVideoInfoHeader;
  MediaType: TAMMediaType;
  BitmapInfo: TBitmapInfo;
  Bitmap: TBitmap;
  Buffer: Pointer;
  tmp: array of byte;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс фильтра перехвата изображения,
  // то завершаем работу
  if FSampleGrabber = NIL then EXIT;

  // Получаем размер кадра
  Result := FSampleGrabber.GetCurrentBuffer(bSize, NIL);
  if (bSize <= 0) or FAILED(Result) then EXIT;

  // Создаем изображение
  bitmap := TBitmap.Create;
  try
    // Получаем тип медиапотока на входе у фильтра перехвата
```

```
    ZeroMemory(@MediaType, sizeof(TAMMediaType));
    Result := FSampleGrabber.GetConnectedMediaType(MediaType);
    if FAILED(Result) then EXIT;

    // Копируем заголовок изображения
    pVideoHeader := TVideoInfoHeader(MediaType.pbFormat^);
    ZeroMemory(@BitmapInfo, sizeof(TBitmapInfo));
    CopyMemory(@BitmapInfo.bmiHeader, @pVideoHeader.bmiHeader,
      sizeof(TBITMAPINFOHEADER));

    Buffer := NIL;

    // Создаем побитовое изображение
    bitmap.Handle := CreateDIBSection(0, BitmapInfo, DIB_RGB_COLORS,
      Buffer, 0, 0);

    // Выделяем память во временном массиве
    SetLength(tmp, bSize);

    try
      // Читаем изображение из медиапотока во временный буфер
      FSampleGrabber.GetCurrentBuffer(bSize, @tmp[0]);

      // Копируем данные из временного буфера в наше изображение
      CopyMemory(Buffer, @tmp[0], MediaType.lSampleSize);
      // Сохраняем изображение в файл
      Bitmap.SaveToFile(FileName);
    except
      // В случае сбоя возвращаем ошибочный результат
      Result := E_FAIL;
    end;
  finally
    // Освобождаем память
    SetLength(tmp, 0);
    FreeAndNil(Bitmap);
  end;
end;
```

```
{*************************************************************************}
{** Регистрация окна для обработки сообщения                            **}
{*************************************************************************}
function TdxMediaPlayer.RegisterEventMessage(AHandle: THandle;
  Msg: Cardinal): HResult;
begin
  // Получение интерфейса управления сообщениями
  Result := FGraphBuilder.QueryInterface(IID_IMediaEventEx, FMediaEvent);

  // Назначаем окно обработки сообщения
  if SUCCEEDED(Result) then
    Result := FMediaEvent.SetNotifyWindow(AHandle, Msg, 0);
end;

{*************************************************************************}
{** Получение сообщения из очереди                                      **}
{*************************************************************************}
function TdxMediaPlayer.GetEvent(out lEventCode, lParam1,
  lParam2: Integer; msTimeout: DWORD): HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;
  // Если отсутствует интерфейс управления сообщениями,
  // то завершаем работу
  if FMediaEvent = NIL then EXIT;

  // Считываем сообщение из очереди
  if SUCCEEDED(FMediaEvent.GetEvent(lEventCode, lParam1, lParam2,
               msTimeout)) then
  begin
    // Освобождаем ресурсы, связанные с событием
    Result := FMediaEvent.FreeEventParams(lEventCode, lParam1, lParam2);
  end;
end;

END.
```

Пример работы с классом *TdxMediaPlayer*

Каким бы простым в использовании не казался класс, самый лучший вариант научить, как с ним работать, — это привести пример. В данном случае в качестве примера, я хотел бы привести небольшой мультимедиапроигрыватель, который так и называется — Small Media Player.

Пример находится в каталоге Examples\DirectShow\DSH_Test на компакт-диске (листинг 14.2).

Окно примера условно разбито на две части: сверху находится область воспроизведения видео, а под ним различные элементы управления (рис. 14.4).

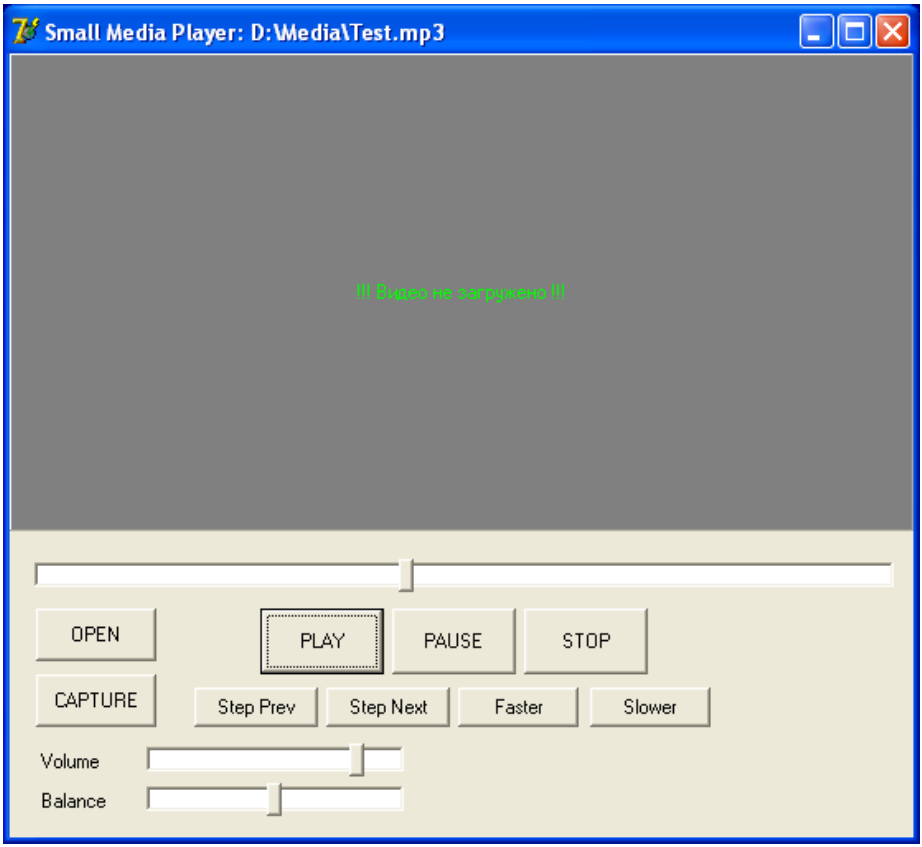


Рис. 14.4. Общий вид тестирующего класс `TdxMediaPlayer` приложения

Листинг 14.2. Текст модуля FormMain.pas проекта DSH_Test

```
UNIT FormMain;
{*************************************************************************}
{** Тестирование класса TdxMediaPlayer                                  **}
{** Автор: Есенин Сергей Анатольевич                                    **}
{*************************************************************************}

{**} INTERFACE {*********************************************************}

{**} USES {**************************************************************}
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, DirectShow9, ComObj, ActiveX, ExtCtrls,
  ComCtrls, UdxMediaPlayer;

{**} CONST {*************************************************************}
  WM_MEDIA_NOTIFY = WM_USER + 1000;

{**} TYPE {**************************************************************}
  TMainForm = class(TForm)
    panelVideo: TPanel;
    buttonCapture: TButton;
    trackBarProgress: TTrackBar;
    buttonPlay: TButton;
    buttonPause: TButton;
    buttonPrev: TButton;
    buttonNext: TButton;
    timerRefresh: TTimer;
    buttonOpen: TButton;
    openDialogVideo: TOpenDialog;
    buttonFast: TButton;
    buttonSlow: TButton;
    trackBarVolume: TTrackBar;
    buttonStop: TButton;
    labelVolume: TLabel;
    labelBalance: TLabel;
    trackBarBalance: TTrackBar;
```

```
    procedure FormResize(Sender: TObject);
    procedure buttonCaptureClick(Sender: TObject);
    procedure buttonPlayClick(Sender: TObject);
    procedure buttonPauseClick(Sender: TObject);
    procedure buttonNextClick(Sender: TObject);
    procedure buttonPrevClick(Sender: TObject);
    procedure trackBarProgressChange(Sender: TObject);
    procedure timerRefreshTimer(Sender: TObject);
    procedure buttonOpenClick(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure buttonFastClick(Sender: TObject);
    procedure buttonSlowClick(Sender: TObject);
    procedure trackBarVolumeChange(Sender: TObject);
    procedure buttonStopClick(Sender: TObject);
    procedure trackBarBalanceChange(Sender: TObject);
  PRIVATE
    FPositioning: boolean;
    procedure MediaNotify(var Msg: Tmessage); message WM_MEDIA_NOTIFY;
  PUBLIC
  END;

{**} VAR {************************************************************}
  MainForm: TMainForm;
  MediaPlayer: TdxMediaPlayer;
  Counter: integer = 1;

{**} IMPLEMENTATION {*************************************************}

{$R *.dfm}

{*********************************************************************}
{** Задаем позицию окна воспроизведения видео в соответствии с      **}
{** изменениями                                                     **}
{*********************************************************************}
```

```
procedure TMainForm.FormResize(Sender: TObject);
begin
  MediaPlayer.SetWindowPosition(Rect(0, 0, panelVideo.ClientRect.Right,
    panelVideo.ClientRect.Bottom));
end;

{**************************************************************************}
{** Получаем кадр из видеопотока и сохраняем на диске                    **}
{**************************************************************************}
procedure TMainForm.buttonCaptureClick(Sender: TObject);
begin
  if SUCCEEDED(
    MediaPlayer.CaptureBitmap(ExtractFileDir(Application.ExeName) +
    Format('\cap%d.bmp', [Counter]))) then
  begin
    // Увеличиваем счетчик сохраненных кадров
    inc(Counter);
  end;
end;

{**************************************************************************}
{** Воспроизведение мультимедиапотока                                    **}
{**************************************************************************}
procedure TMainForm.buttonPlayClick(Sender: TObject);
begin
  MediaPlayer.Play;
end;

{**************************************************************************}
{** Остановка воспроизведения                                            **}
{**************************************************************************}
procedure TMainForm.buttonStopClick(Sender: TObject);
begin
  MediaPlayer.Stop;
end;
```

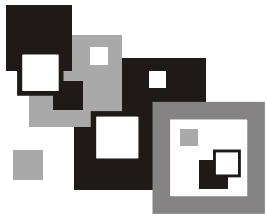
```
{************************************************************************}
{** Приостанавливаем воспроизведение потока                            **}
{************************************************************************}
procedure TMainForm.buttonPauseClick(Sender: TObject);
begin
  MediaPlayer.Pause
end;

{************************************************************************}
{** Перемотка вперед                                                   **}
{************************************************************************}
procedure TMainForm.buttonNextClick(Sender: TObject);
begin
  MediaPlayer.StepNext;
end;

{************************************************************************}
{** Перемотка назад                                                    **}
{************************************************************************}
procedure TMainForm.buttonPrevClick(Sender: TObject);
begin
  MediaPlayer.StepPrev;
end;

{************************************************************************}
{** Установка новой позиции воспроизведения                            **}
{************************************************************************}
procedure TMainForm.trackBarProgressChange(Sender: TObject);
begin
  if FPositioning then EXIT;

  FPositioning := TRUE;
  try
    MediaPlayer.SetPlayingPosition(trackBarProgress.Position * 100);
  finally
```

```
    FPositioning := FALSE;
  end;
end;

{*************************************************************************}
{** Обновление текущей позиции воспроизведения                          **}
{*************************************************************************}
procedure TMainForm.timerRefreshTimer(Sender: TObject);
var
  P: int64;
begin
  if FPositioning then EXIT;

  FPositioning := TRUE;
  try

    if SUCCEEDED(MediaPlayer.GetPlayingPosition(P)) then
       trackBarProgress.Position := P div 100;
  finally
    FPositioning := FALSE;
  end;
end;

{*************************************************************************}
{** Открытие нового мультимедиафайла                                    **}
{*************************************************************************}
procedure TMainForm.buttonOpenClick(Sender: TObject);
var
  Volume: longint;
begin
  if openDialogVideo.Execute then
  begin
    if SUCCEEDED(
       MediaPlayer.Initialize(WideString(openDialogVideo.FileName))) then
    begin
      Caption := 'Small Media Player: ' + openDialogVideo.FileName;
```

```
      trackBarProgress.Max := MediaPlayer.FrameCount div 100;

      MediaPlayer.RegisterEventMessage(Handle, WM_MEDIA_NOTIFY);

      MediaPlayer.SetWindowPosition(Rect(0, 0,
        panelVideo.ClientRect.Right,
        panelVideo.ClientRect.Bottom));

      if SUCCEEDED(MediaPlayer.GetVolume(Volume)) then
        trackBarVolume.Position := 10 - Volume div 100;
    end;
  end;
end;

{*************************************************************************}
{** Перетаскиваем форму мышью                                           **}
{*************************************************************************}
procedure TMainForm.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  ReleaseCapture;

  SendMessage(Handle, WM_NCLBUTTONDOWN, HTCAPTION, 0);
end;

{*************************************************************************}
{** Создание и инициализация объекта MediaPlayer                        **}
{*************************************************************************}
procedure TMainForm.FormCreate(Sender: TObject);
begin
  MediaPlayer := TdxMediaPlayer.Create(panelVideo.Handle);
end;

{*************************************************************************}
{** Удаление объекта MediaPlayer                                        **}
{*************************************************************************}
```

```
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  FreeAndNil(MediaPlayer);
end;

{**************************************************************************}
{** Ускоряем воспроизведение                                             **}
{**************************************************************************}
procedure TMainForm.buttonFastClick(Sender: TObject);
begin
  MediaPlayer.Faster;
end;

{**************************************************************************}
{** Замедляем воспроизведение                                            **}
{**************************************************************************}
procedure TMainForm.buttonSlowClick(Sender: TObject);
begin
  MediaPlayer.Slower;
end;

{**************************************************************************}
{** Изменяем громкость звучания                                          **}
{**************************************************************************}
procedure TMainForm.trackBarVolumeChange(Sender: TObject);
begin
  MediaPlayer.SetVolume(trackBarVolume.Position * 100);
end;

{**************************************************************************}
{** Обрабатываем сообщения от объекта MediaPlayer                        **}
{**************************************************************************}
procedure TMainForm.MediaNotify(var Msg: Tmessage);
var
  EventCode: Integer;
  Param1, Param2: Integer;
```

```
begin
  while MediaPlayer.GetEvent(EventCode, Param1, Param2, 0) = S_OK do
  begin
    if EventCode = EC_Complete then
    begin
      MediaPlayer.Stop;
      MediaPlayer.SetPlayingPosition(0);
    end;
  end;
end;

{*********************************************************************}
{** Изменяем баланс                                                 **}
{*********************************************************************}
procedure TMainForm.trackBarBalanceChange(Sender: TObject);
begin
  MediaPlayer.SetBalance(trackBarBalance.Position);
end;

END.
```



Глава 15

Захват аудио и видео

Захват видео

Термин *"захват видео"* обозначает возможность приложения записывать видеоданные, полученные от различных источников, используя API подсистемы DirectShow. Под устройствами захвата изображения понимаются не только камеры (Web-камеры, IP-камеры и т. п.), но и TV-тюнеры, различные видеорегистраторы и т. п. Захватываемое видеоизображение может быть записано на диск или просмотрено в режиме реального времени.

Большое количество новых устройств захвата изображения используют *модель драйверов Windows* (Windows Driver Model, WDM). Данная архитектура включает набор аппаратно-независимых драйверов, называемых драйверами класса, и набор аппаратно-зависимых мини-драйверов, которые поставляет производитель оборудования. Мини-драйверы реализуют всю специфичную для устройства функциональность.

Граф фильтров DirectShow представляет любое устройство захвата WDM как фильтр захвата WDM (WDM Capture Filter). Данный фильтр настраивается в зависимости от характеристик драйвера.

Некоторые старые устройства захвата видео все еще используют драйверы видео для Windows (Video for Windows, VFW). Несмотря на то, что данные драйверы уже устарели, в подсистеме DirectShow имеется специальный фильтр (VFW Capture), обеспечивающий работу с данным драйвером.

Захват звука

Используя подсистему DirectShow, мы можем разрабатывать приложения, в которых реализуется возможность записи звука. Такая возможность уже была рассмотрена нами при изучении подсистемы DirectSound. Разница здесь заключается в механизме записи данных и поддерживаемых форматах.

Запись звука может осуществляться с микрофона, проигрывателя либо иного устройства, подключенного к звуковой плате компьютера. К основным причинам, для чего все это нужно, можно смело отнести следующие:

- запись звука для дублирования видеопотока;
- перевод аналогового аудиопотока в цифровой формат;
- передача звука по сети.

Подсистема DirectShow позволяет записывать звук с различных аналоговых устройств, подключенных к звуковой плате посредством фильтра захвата аудио (Audio Capture Filter). Этот фильтр использует API Microsoft® Platform SDK `waveIn`*`XXX`* для управления всеми устройствами, чьи драйверы поддерживают это API. Каждая звуковая плата представлена отдельными экземплярами данного фильтра.

Фильтр захвата звука представляет каждый вход на звуковой плате, будь то микрофон или MIDI-вход, как входной контакт (Input Pin). Приложение может использовать данный контакт для разрешения или запрещения ввода, настройки частот, положения на панораме и т. д. Возможности управления зависят от драйвера. Для полного использования всех возможностей звуковой платы может даже понадобиться документация изготовителя карты.

Интерфейсы

В начале данной части книги мы с вами изучили ряд интерфейсов и понятий, которые повсеместно используются в подсистеме DirectShow.

Граф фильтров, который выполняет захват аудио или видео, называется *графом захвата* (Capture Graph). Данный граф имеет более сложную структуру, чем обычный граф воспроизведения. Для упрощения построения графа захвата приложением подсистема DirectShow предоставляет специальный объект, именуемый *построителем графа захвата* (Capture Graph Builder). Объект представлен интерфейсом `ICaptureGraphBuilder2`, содержащим методы построения и управления графом захвата.

Общую схему взаимодействия графов захвата и воспроизведения в приложении можно представить так, как показано на рис. 15.1.

Интерфейс `ICaptureGraphBuilder2` содержит следующие методы:

- `SetFiltergraph` — установка графа фильтров для использования;
- `GetFiltergraph` — получение используемого в настоящий момент графа;
- `SetOutputFileName` — создание файла для записи данных из графа;
- `FindInterface` — поиск интерфейса в графе, начиная с указанного фильтра;

- `RenderStream` — построение графа захвата;
- `ControlStream` — установка времени старта и остановки для различных потоков захватываемых данных;
- `AllocCapFile` — предварительная установка нужного размера файла для записи данных;
- `CopyCaptureFile` — копирование проверенных мультимедиаданных из файла захвата;
- `FindPin` — поиск определенного контакта в фильтре или проверка соответствия контакта указанному критерию.

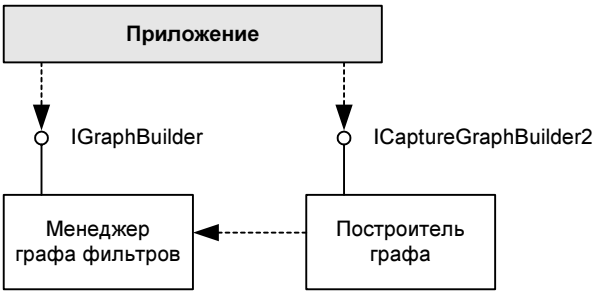


Рис. 15.1. Схема взаимодействия графов захвата и воспроизведения в приложении

Для работы нам будет достаточно всего четырех методов данного интерфейса.

Установка графа фильтров для использования:

```
function SetFiltergraph(
  pfg: IGraphBuilder):
HResult; stdcall;
```

Здесь `pfg` — указатель на объект графа с интерфейсом `IGraphBuilder`.

Построение графа захвата:

```
function RenderStream(
  pCategory,
  pType: PGUID;
  pSource: IUnknown;
  pfCompressor,
  pfRenderer: IBaseFilter):
HResult; stdcall;
```

Здесь:

- `pCategory` — указатель на уникальный идентификатор, определяющий категорию контакта. Может принимать нулевое значение для использования контакта независимо от категории. Мы будем использовать следующие значения:
  - `PIN_CATEGORY_CAPTURE` — захват данных;
  - `PIN_CATEGORY_PREVIEW` — предварительный просмотр;
- `pType` — указатель на уникальный идентификатор, задающий тип выходных данных, либо нулевое значение для использования любого контакта, независимо от категории. Идентификатор формата данных может принимать одно из следующих значений:
  - `MEDIATYPE_AnalogAudio` — аналоговое аудио;
  - `MEDIATYPE_AnalogVideo` — аналоговое видео;
  - `MEDIATYPE_Audio` — аудио;
  - `MEDIATYPE_AUXLine21Data` — используется для закрытых заголовков;
  - `MEDIATYPE_File` — файл (устаревший);
  - `MEDIATYPE_Interleaved` — чередуемое аудио и видео. Используется для цифрового видео (Digital Video, DV);
  - `MEDIATYPE_LMRT` — устаревший формат (не используется);
  - `MEDIATYPE_Midi` — формат MIDI;
  - `MEDIATYPE_MPEG2_PES` — пакеты MPEG-2 PES;
  - `MEDIATYPE_MPEG2_SECTION` — данные секции MPEG-2;
  - `MEDIATYPE_ScriptCommand` — данные в виде сценария. Используется в закрытых заголовках;
  - `MEDIATYPE_Stream` — поток данных без временных меток;
  - `MEDIATYPE_Text` — текст;
  - `MEDIATYPE_Timecode` — данные разбиты по кадрам. В подсистеме DirectShow нет фильтров, поддерживающих данный формат;
  - `MEDIATYPE_URL_STREAM` — устаревший формат (не используется);
  - `MEDIATYPE_Video` — видео;
- `pSource` — указатель на стартовый фильтр в цепи графа или на выходной контакт;
- `pfCompressor` — фильтр сжатия. Может принимать нулевое значение;
- `pfRenderer` — фильтр-приемник, такой как фильтр воспроизведения или мультиплексор. Может принимать нулевое значение.

Создание файла для записи данных из графа:

```
function SetOutputFileName(
  const pType: TGUID;
  lpstrFile: PWCHAR;
  out ppf: IBaseFilter;
  out ppSink: IFileSinkFilter):
HResult; stdcall;
```

Здесь:

- `pType` — указатель на уникальный идентификатор, определяющий подтип выходных мультимедиаданных или идентификатор класса мультиплексора или фильтра записи данных в файл. Если мы хотим задать подтип мультимедиаданных, то должны использовать одно из следующих значений:
  - `MEDIASUBTYPE_Avi` — данные в формате AVI;
  - `MEDIASUBTYPE_Asf` — данные в формате ASF;
- `lpstrFile` — имя файла;
- `ppf` — адрес указателя, куда будет передан интерфейс мультиплексора;
- `ppSink` — адрес указателя, куда будет передан интерфейс записи данных в файл `IFileSinkFilter`. Может принимать нулевое значение.

Поиск интерфейса в графе, начиная с указанного фильтра:

```
function FindInterface(
  pCategory,
  pType: PGUID;
  pf: IBaseFilter;
  const riid: TGUID;
  out ppint):
HResult; stdcall;
```

Здесь:

- `pCategory` — указатель на уникальный идентификатор, определяющий критерий поиска;
- `pType` — указатель на уникальный идентификатор, задающий тип выходных данных;
- `pf` — фильтр, с которого начинается поиск;
- `riid` — идентификатор искомого интерфейса;

- ppint — адрес переменной, в которую и будет записан искомый интерфейс.

Еще один интерфейс, который нами будет использован — это интерфейс управления мультиплексором IConfigAviMux. Его методы:

- SetMasterStream — установка основного потока для синхронизации с другими потоками в файле;
- GetMasterStream — получение основного потока;
- SetOutputCompatibilityIndex — установка индекса формата AVI-файла;
- GetOutputCompatibilityIndex — получение индекса формата AVI-файла.

Из методов данного интерфейса нам потребуется всего один — SetMasterStream. При одновременном захвате видеоданных и аудиопотока мы будем выставлять в качестве основного потока звуковой:

```
function SetMasterStream(
  iStream: Longint):
HResult; stdcall;
```

Здесь iStream — индекс потока или -1 при отсутствии основного потока. Потоки нумеруются с нуля.

Для управления свойствами фильтров и контактов нами будут использоваться следующие интерфейсы:

- IAMStreamConfig — интерфейс управления форматом данных выходного потока;
- ISpecifyPropertyPages — интерфейс управления страницами свойств.

Интерфейс IAMStreamConfig выступает в качестве вспомогательного для получения объекта с интерфейсом ISpecifyPropertyPages. Интерфейс управления страницами свойств ISpecifyPropertyPages содержит всего один метод — метод получения страниц свойств:

```
function GetPages(
  out pages: TCAGUID):
HResult; stdcall;
```

Здесь pages — указатель на элемент структуры TCAGUID, которую заполняет вызывающий оператор. Для заполнения поля pElems данной структуры будет вызван метод CoTaskMemAlloc и в конце работы память должна быть освобождена вызовом CoTaskMemFree.

Перечисление устройств определенного класса

Мы разобрали несколько необходимых для работы интерфейсов, но до сих пор не обсуждали способы получения собственно самого устройства захвата. Для работы со списком устройств нам потребуются следующие интерфейсы:

- `IMoniker` — позволяет получить указатель на объект, идентифицируемый моникером, или получить доступ к хранилищу свойств объекта;
- `IEnumMoniker` — используется для перечисления моникеров;
- `ICreateDevEnum` — нужен для создания объекта перечисления указанной категории устройств;
- `IPropertyBag` — предоставляет доступ к коллекции свойств объекта.

Рассмотрим подробнее перечисленные интерфейсы.

Первым делом мы должны создать объект с интерфейсом `ICreateDevEnum` для перечисления указанной категории устройств, содержащий всего один метод:

```
function CreateClassEnumerator(
  const clsidDeviceClass: TGUID;
  out ppEnumMoniker: IEnumMoniker;
  dwFlags: DWORD):
HResult; stdcall;
```

Здесь:

- `clsidDeviceClass` — идентификатор класса категории устройств;
- `ppEnumMoniker` — адрес переменной, которая получит интерфейс перечисления моникеров `IEnumMoniker`;
- `dwFlags` — битовая комбинация флагов, задающих режим работы данного метода:
  - `CDEF_DEVMON_CMGR_DEVICE` — перечисление аудио- и видеокодеков, использующих менеджер сжатия аудио (Audio Compression Manager, ACM) или менеджер сжатия видео (Video Compression Manager, VCM);
  - `CDEF_DEVMON_DMO` — перечисление объектов DMO (DirectX Media Objects);
  - `CDEF_DEVMON_FILTER` — перечисление собственных фильтров DirectShow;
  - `CDEF_DEVMON_PNP_DEVICE` — перечисление устройств Plug-and-Play.

  При нулевом значении будут перечислены все фильтры, которые попадают в указанную категорию.

Затем, при помощи интерфейса `IEnumMoniker` мы организуем цикл перебора всех моникеров. Нужен нам для этого метод `IEnumMoniker.Next`:

```
function Next(
  celt: Longint;
  out elt;
  pceltFetched: PLongint):
HResult; stdcall;
```

Здесь:

- `celt` — число возвращаемых моникеров;
- `elt` — переменная, в которую будет занесена ссылка на моникер;
- `pceltFetched` — по завершении работы метода данная переменная будет содержать реальное количество моникеров.

В интерфейсе `IMoniker`, позволяющем получить доступ к объекту либо хранилищу свойств, нам интересны всего два метода. Это метод получения интерфейса хранилища:

```
function BindToStorage(
  const bc: IBindCtx;
  const mkToLeft: IMoniker;
  const iid: TIID;
  out vObj):
HResult; stdcall;
```

Здесь:

- `bc` — указатель на интерфейс `IBindCtx`. В нашем случае `NIL`;
- `mkToLeft` — если используется составной моникер, то значением будет моникер, стоящий слева от текущего моникера. В нашем случае `NIL`;
- `iid` — уникальный идентификатор запрашиваемого интерфейса, указатель на который будет занесен в `vObj`;
- `vObj` — адрес указателя, в который будет занесен искомый интерфейс.

И метод получения нужного нам интерфейса, идентифицируемого моникером объекта:

```
function BindToObject(
  const bc: IBindCtx;
  const mkToLeft: IMoniker;
  const iidResult: TIID;
  out vResult):
HResult; stdcall;
```

Здесь:

- `bc` — указатель на интерфейс `IBindCtx`. В нашем случае `NIL`;
- `mkToLeft` — если используется составной моникер, то значением будет моникер, стоящий слева от текущего моникера. В нашем случае `NIL`;
- `iidResult` — ссылка на идентификатор интерфейса, который мы желаем получить для связи с объектом, идентифицируемым моникером;
- `vResult` — адрес указателя, в который будет занесен искомый интерфейс.

Режимы захвата и предварительного просмотра

Для предварительного просмотра видео мы должны построить соответствующий граф захвата:

```
Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_PREVIEW,
  @MEDIATYPE_Video, FVideoCaptureFilter, NIL, NIL);
```

Фильтры в графе могут размещаться так, как показано на рис. 15.2.

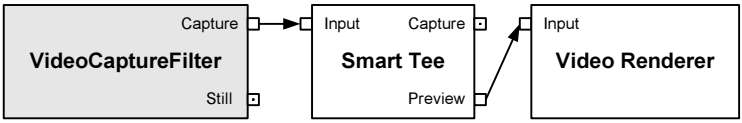


Рис. 15.2. Граф захвата видеоданных в режиме предварительного просмотра

В некоторых случаях нам может потребоваться такой режим работы программы, в котором во время записи поток видео отображался бы непосредственно и на экране. Это достигается за счет двух вызовов метода `ICaptureGraphBuilder2.RenderStream`:

```
Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_PREVIEW,
  @MEDIATYPE_Video, FVideoCaptureFilter, NIL, NIL);


Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_CAPTURE,
  @MEDIATYPE_Video, FVideoCaptureFilter, FVideoCompressFilter, FMux);
```

Соответственно изменится и диаграмма фильтров графа (рис. 15.3).

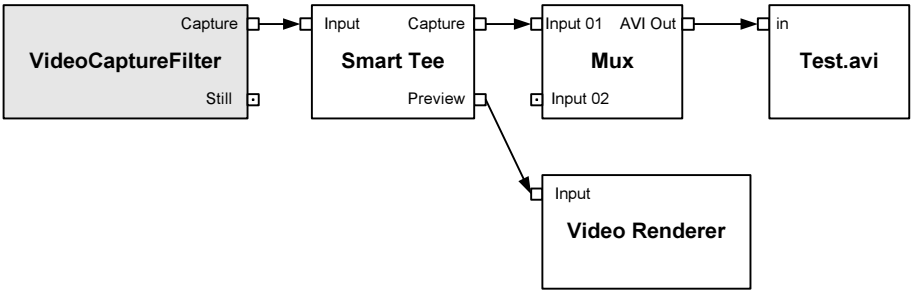


Рис. 15.3. Граф захвата видеоданных
в режиме одновременного захвата и предварительного просмотра

Из диаграмм хорошо видно, что следом за фильтром захвата встроен фильтр, именуемый Smart Tee (фильтр умного разделения). Данный фильтр автоматически встраивается в граф в том случае, когда фильтр захвата имеет только один контакт захвата на выходе и используется для разделения захватываемого потока на два — поток предварительного просмотра и поток захвата.

Несмотря на разделение потока на две составляющие, фильтр Smart Tee не дублирует данные физически.

В связи с тем, что в нашем примере выбран формат AVI в качестве выходного, и мы будем использовать режим предварительного прослушивания, то в граф автоматически будет встроен фильтр умного разделения для захвата и одновременного проигрывания данных. Это произойдет, даже если мы не будем использовать захват видео и записывать только аудиоданные. В итоге мы получим следующую картину графа захвата аудиоданных (рис. 15.4).

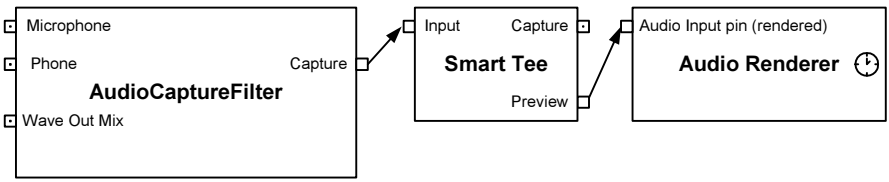


Рис. 15.4. Граф захвата аудиоданных
в режиме предварительного прослушивания

На рис. 15.4 хорошо видна последовательность фильтров: фильтр захвата аудио, следом идет фильтр умного разделения, от которого исходит поток данных на фильтр воспроизведения потока аудио.

После того как мы строим и запускаем граф захвата аудиоданных в режиме одновременного прослушивания и записи данных, у нас в граф добавляются

мультиплексор и фильтр, на выходе которого мы получаем нужный нам файл на диске (рис. 15.5).

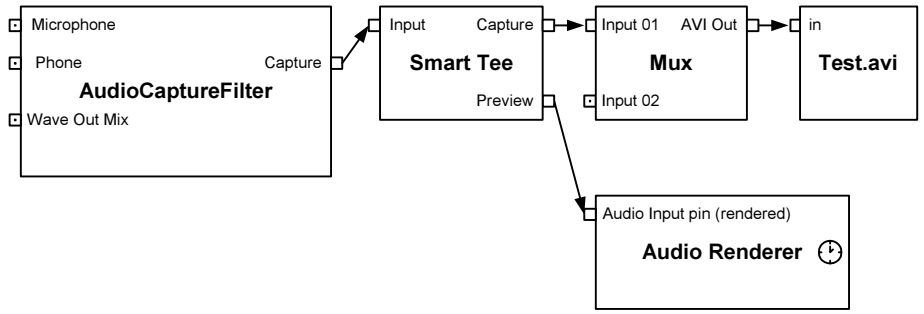


Рис. 15.5. Граф захвата аудиоданных
в режиме одновременного захвата и предварительного прослушивания

Запись видео со звуком

Разобравшись с записью аудиоданных и видео по отдельности, давайте разберемся и с возможностью одновременной их записи.

Для одновременного просмотра (и прослушивания) потоков данных нам необходимо построить соответствующим образом граф:

```
Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_PREVIEW,
  @MEDIATYPE_Video, FVideoCaptureFilter, NIL, NIL);

Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_PREVIEW,
  @MEDIATYPE_Audio, FAudioCaptureFilter, NIL, NIL);
```

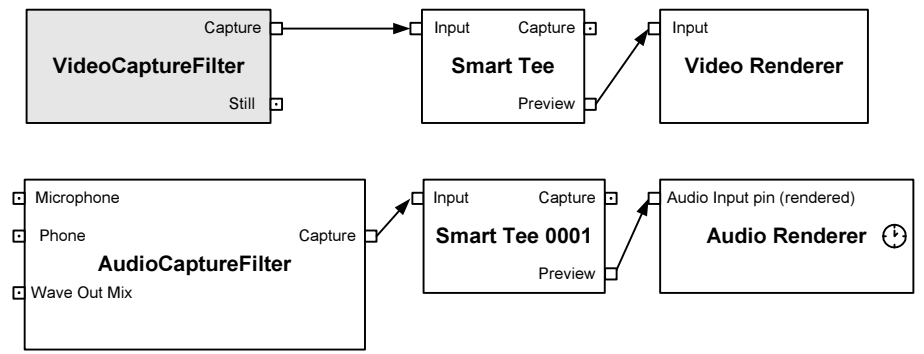


Рис. 15.6. Граф захвата данных видео и аудио в режиме предварительного просмотра

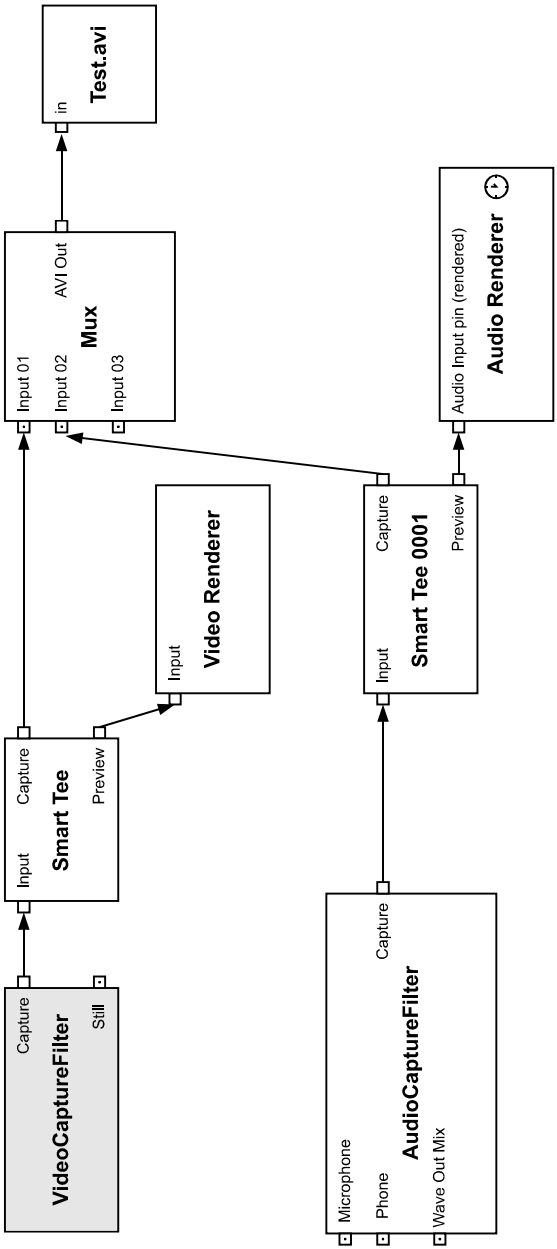


Рис. 15.7. Граф захвата данных видео и аудио в режиме одновременного захвата и предварительного просмотра

В режиме предварительного просмотра (и прослушивания) наш граф будет выглядеть так, как показано на рис. 15.6.

Хорошо видно, что он состоит из двух разных цепочек фильтров — одна цепь выводит изображение, а другая — звук. В каждую из цепей встроен свой фильтр умного разделения Smart Tee.

Следующим шагом мы включаем режим захвата аудиоданных и видео:

```
Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_CAPTURE,
  @MEDIATYPE_Video, FVideoCaptureFilter, FVideoCompressFilter, FMux);
Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_CAPTURE,
  @MEDIATYPE_Audio, FAudioCaptureFilter, FAudioCompressFilter, FMux);
```

Результат представлен на рис. 16.7.

Здесь мы видим, как от разных устройств захвата потоки проходят каждый через свой фильтр разделения Smart Tee, а затем разбиваются на две составляющие. Одна составляющая идет на предварительное воспроизведение, а вторые составляющие попадают в один и тот же мультиплексор, в котором соединяются в один поток и выводятся в файл.

Ранее уже было сказано, что, записывая поток видео и поток аудио, мы будем выставлять в качестве основного потока поток аудиоданных:

```
pConfigMux.SetMasterStream(1);
```

Это необходимо проделать для синхронизации потоков данных в файле.

Сжатие потоков аудио и видео

Запись видеопотока вместе со звуком, да еще и без использования механизмов сжатия — дело довольно дорогостоящее. Даже записанный минутный ролик в разрешении 640×480 точек может занимать больше сотни мегабайт.

Вот тут-то нам на помощь и приходят специальные трансформационные фильтры, именуемые *фильтрами сжатия* (или компрессии). Они существуют как для потока видео, так и для потока аудиоданных. Получить их список можно точно так же, как и список устройств захвата аудио и видео, указав специальные категории.

После встраивания в граф захвата фильтров сжатия, он может иметь вид, представленный на рис. 15.8.

Из рисунка видно, как потоки аудио и видео на пути к мультиплексору проходят через фильтры сжатия. В зависимости от настроек сжатия и самих фильтров сжатия размер файла может измениться в десятки и даже сотни раз. Не стоит только забывать о потере качества изображения и звука при сжатии.

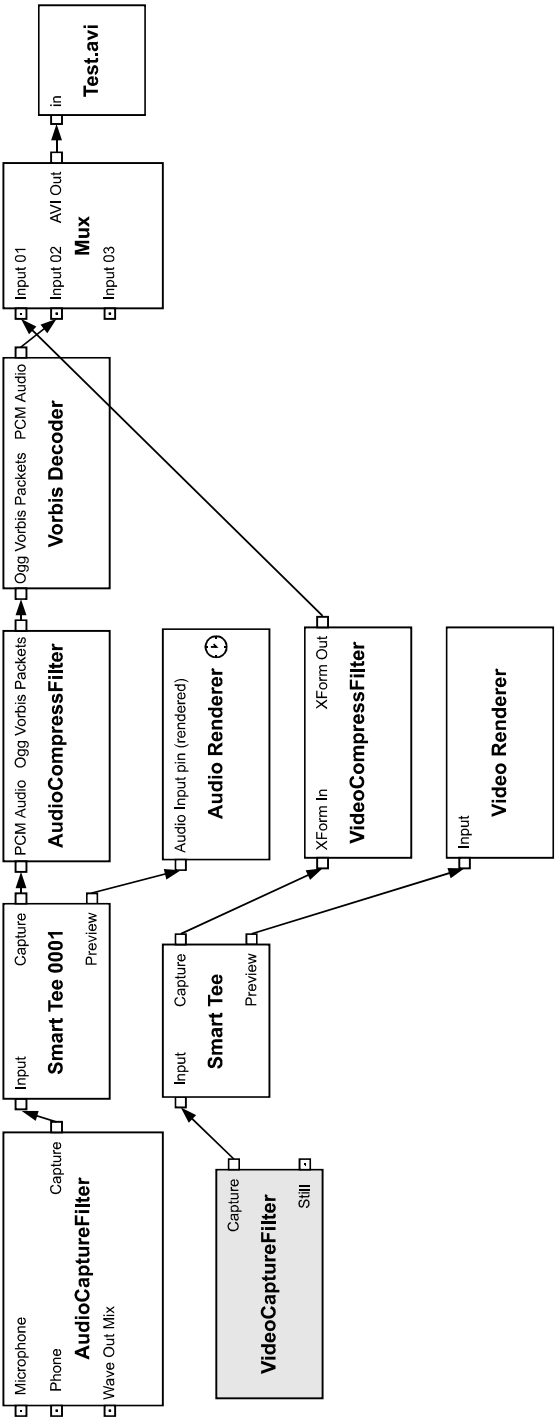


Рис. 15.8. Граф захвата со встроенными фильтрами сжатия потока аудио и видео

Страницы свойств

Прежде чем начать запись звука или изображения, было бы неплохо настроить ряд параметров для улучшения их качества либо, наоборот, уменьшить качество с целью экономии места на диске.

Доступ к страницам свойств фильтров можно получить, используя специальный интерфейс `ISpecifyPropertyPages`, который был описан ранее. А отображается страница свойств в виде модального диалога методом `OleCreatePropertyFrame` из библиотеки COM:

```
function OleCreatePropertyFrame(
  hwndOwner: HWnd;
  x,
  y: Integer;
  lpszCaption: POleStr;
  cObjects: Integer;
  pObjects: Pointer;
  cPages: Integer;
  pPageCLSIDs: Pointer;
  lcid: TLCID;
  dwReserved: Longint;
  pvReserved: Pointer):
HResult; stdcall;
```

Здесь:

- `hwndOwner` — дескриптор родительского окна для страницы свойств;
- `x` — зарезервировано;
- `y` — зарезервировано;
- `lpszCaption` — строка, которая будет использована в заголовке страницы свойств;
- `cObjects` — количество указателей на объекты, которые будут переданы в параметре `pObjects`;
- `pObjects` — массив указателей `IUnknown` на объекты, для которых должна быть вызвана страница свойств;
- `cPages` — количество страниц свойств, определенных в `pPageCLSIDs`;
- `pPageCLSIDs` — массив размера `cPages`, содержащий идентификаторы класса (CLSID) для каждой страницы свойств;
- `lcid` — идентификатор текущих локальных установок;

- `dwReserved` — зарезервировано, должно быть нулевым;
- `pvReserved` — зарезервировано, должно быть нулевым.

Не каждый фильтр имеет страницу свойств. Проверить это достаточно просто — нужно лишь попытаться получить интерфейс `ISpecifyPropertyPages` для фильтра:

```
Result := Filter.QueryInterface(ISpecifyPropertyPages, PropertyPages);
```

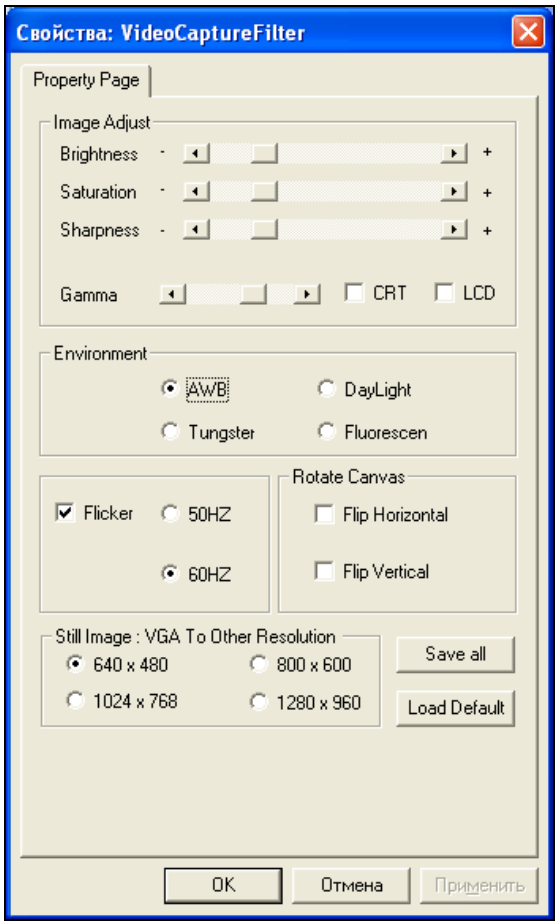


Рис. 15.9. Страница свойств фильтра получения видеоизображения

Также для некоторых фильтров мы можем настроить форматы выходных потоков данных, получив интерфейс `IAMStreamConfig`, про который так же было сказано ранее.

Я попробую привести в качестве примера страницы свойств установленного на моем компьютере оборудования. В качестве компьютера в настоящий момент выступает ноутбук BLISS 507S. Устройством захвата изображения является Web-камера Genius VideoCAM Slim USB2, а работа со звуком производится через встроенную звуковую плату.

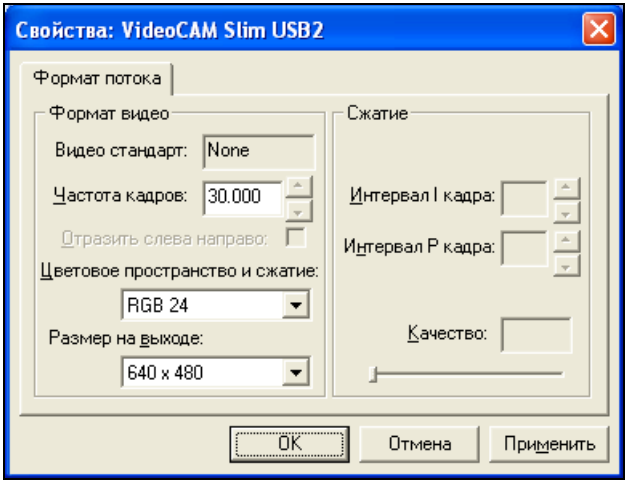


Рис. 15.10. Страница свойств выходного контакта фильтра получения видеоизображения

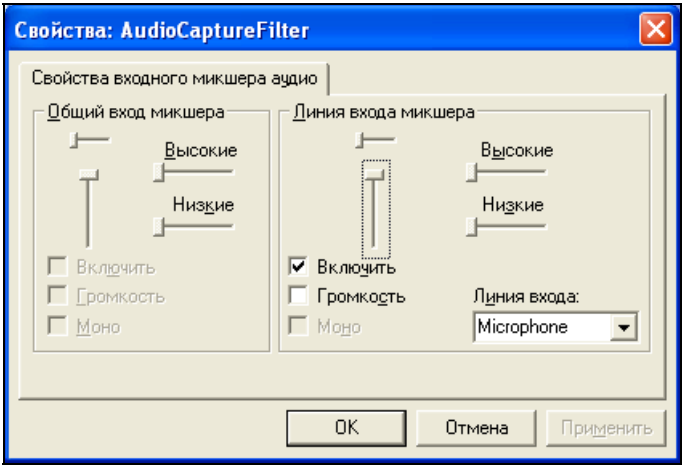


Рис. 15.11. Страница свойств фильтра захвата аудиопотока

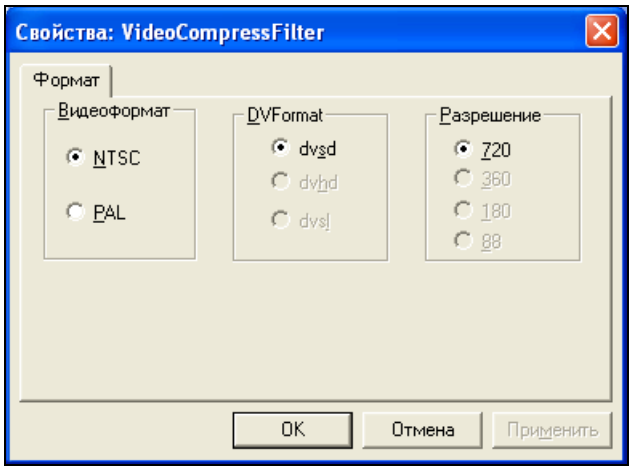


Рис. 15.12. Страница свойств фильтра сжатия видеопотока

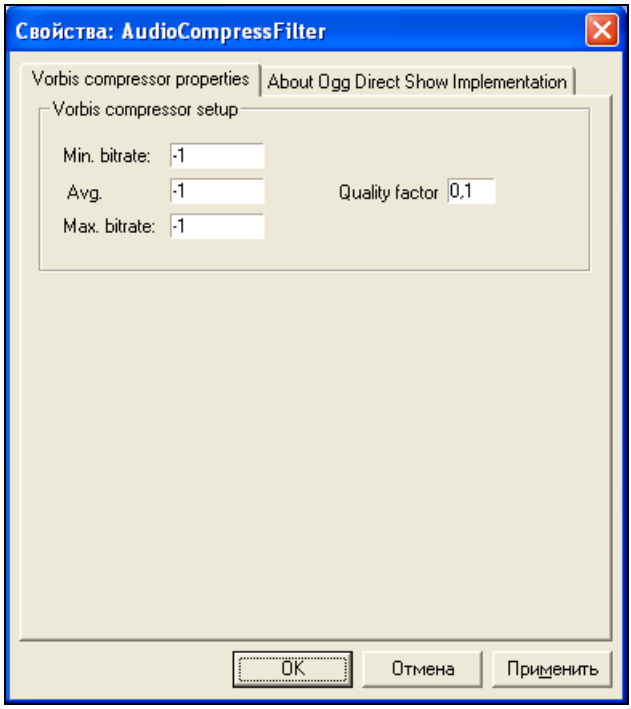


Рис. 15.13. Страница свойств фильтра сжатия потока аудио

Страница свойств устройства захвата изображения позволяет настраивать такие параметры, как яркость, контрастность, резкость, гамма и ряд других параметров (рис. 15.9).

Страница свойств выходного контакта фильтра дает нам возможность установки размера изображения на выходе, установки цветового пространства и т. п. (рис. 15.10).

Фильтр захвата аудио позволят настраивать такие параметры, как громкость, выбор линии входа и т. п. (рис. 15.11).

Примеры страниц свойств фильтров сжатия видео- и аудиопотоков, установленных на моем компьютере, имеют вид, представленный на рис. 15.12 и 15.13.

Алгоритм работы

Теперь пришло время рассказать о последовательности действий при работе с устройствами захвата изображения и звука. Было рассказано об интерфейсах, о сжатии потоков данных и настройках фильтров по отдельности. Теперь же составим примерный план наших действий.

1. Создаем объект для построения графа фильтров:
```
Result := CoCreateInstance(CLSID_FilterGraph, NIL,
  CLSCTX_INPROC_SERVER, IID_IGraphBuilder, FGraphBuilder);
```
2. Создаем объект для построения графа захвата:
```
Result := CoCreateInstance(CLSID_CaptureGraphBuilder2, NIL,
  CLSCTX_INPROC_SERVER, IID_ICaptureGraphBuilder2,
  FCaptureGraphBuilder);
```
3. Задаем граф фильтров для использования в построении графа захвата:
```
Result := FCaptureGraphBuilder.SetFiltergraph(FGraphBuilder);
```
4. Получаем фильтры захвата и сжатия, исходя из их имен:
```
FVideoCaptureFilter :=
    EnumerateDevices(CLSID_VideoInputDeviceCategory,
                     VideoCaptureDeviceName, NIL, TRUE);

FAudioCaptureFilter :=
    EnumerateDevices(CLSID_AudioInputDeviceCategory,
                     AudioCaptureDeviceName, NIL, TRUE);

FVideoCompressFilter :=
    EnumerateDevices(CLSID_VideoCompressorCategory,
                     VideoCompressDeviceName, NIL, TRUE);
```

```
FAudioCompressFilter :=
    EnumerateDevices(CLSID_AudioCompressorCategory,
                     AudioCompressDeviceName, NIL, TRUE);
```

5. Добавляем полученные фильтры в граф:

```
// Добавляем фильтр захвата видео в граф
if FVideoCaptureFilter <> NIL then
begin
  FGraphBuilder.AddFilter(FVideoCaptureFilter, 'VideoCaptureFilter');
end;

// Добавляем фильтр захвата звука в граф
if FAudioCaptureFilter <> NIL then
begin
  FGraphBuilder.AddFilter(FAudioCaptureFilter, 'AudioCaptureFilter');
end;

// Добавляем фильтр сжатия видео в граф
if FVideoCompressFilter <> NIL then
begin
  FGraphBuilder.AddFilter(FVideoCompressFilter,
                          'VideoCompressFilter');
end;

// Добавляем фильтр сжатия звука в граф
if FAudioCompressFilter <> NIL then
begin
  FGraphBuilder.AddFilter(FAudioCompressFilter,
                          'AudioCompressFilter');
end;
```

6. Строим граф захвата в зависимости от флагов захвата и предварительного просмотра, задаем формат выходных данных:

```
Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_PREVIEW,
  @MEDIATYPE_Video, FVideoCaptureFilter, NIL, NIL);

Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_PREVIEW,
  @MEDIATYPE_Audio, FAudioCaptureFilter, NIL, NIL);
```

```
Result := FCaptureGraphBuilder.SetOutputFileName(MEDIASUBTYPE_Avi,
  PWideChar(FCaptureFileName), FMux, FSink);

Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_CAPTURE,
  @MEDIATYPE_Video, FVideoCaptureFilter, FVideoCompressFilter, FMux);

Result := FCaptureGraphBuilder.RenderStream(@PIN_CATEGORY_CAPTURE,
  @MEDIATYPE_Audio, FAudioCaptureFilter, FAudioCompressFilter, FMux);
```

7. Запускаем граф захвата:
```
Result := FMediaControl.Run();
```
8. Освобождаем выделенную память.

Класс *TdxCaptureManager*

Класс `TdxCaptureManager` обеспечивает возможность захвата и предварительного просмотра потоков видео и аудио. По своей сути класс представляет удобный менеджер для работы с устройствами захвата. Располагается он в модуле UdxCaptureManager.pas каталога Classes на компакт-диске (листинг 15.1).

Листинг 15.1. Текст модуля UdxCaptureManager.pas

```
UNIT UdxCaptureManager;

{*************************************************************************}
{** Захват аудио и видео посредством DirectShow                         **}
{** Автор: Есенин Сергей Анатольевич                                    **}
{*************************************************************************}

{**} INTERFACE {*********************************************************}

{**} USES {**************************************************************}
  Windows, Classes, Graphics, SysUtils, ActiveX, DirectShow9, Dialogs;

{**} TYPE {**************************************************************}
  TdxCaptureManager = class
  PRIVATE
```

```
    FGraphBuilder:         IGraphBuilder;
    FCaptureGraphBuilder: ICaptureGraphBuilder2;
    FMux:                  IBaseFilter;
    FSink:                 IFileSinkFilter;
    FMediaControl:         IMediaControl;
    FVideoWindow:          IVideoWindow;

    FVideoCaptureFilter:  IBaseFilter;
    FAudioCaptureFilter:  IBaseFilter;
    FVideoCompressFilter: IBaseFilter;
    FAudioCompressFilter: IBaseFilter;

    FCaptureFileName:     WideString;
    FCapturing:           boolean;
    FVideoHandle:         THandle;
    FPreview:             boolean;
    FVideoRect:           TRect;

    FVideoCaptureDeviceName:  WideString;
    FAudioCaptureDeviceName:  WideString;
    FVideoCompressDeviceName: WideString;
    FAudioCompressDeviceName: WideString;

    procedure SetPreview(Value: boolean);

    function EnumerateDevices(const clsidDeviceClass: TGUID;
      DevName: WideString; DevList: TStrings;
      GetFirst: boolean = FALSE): IBaseFilter;

    procedure SetVideoCaptureDeviceName(Value: WideString);
    procedure SetAudioCaptureDeviceName(Value: WideString);
    procedure SetVideoCompressDeviceName(Value: WideString);
    procedure SetAudioCompressDeviceName(Value: WideString);

    function DisplayPropertyFrame(Filter: IBaseFilter;
      Handle: THandle): HResult;
```

```
PUBLIC
  property CaptureFileName: WideString read FCaptureFileName
                                        write FCaptureFileName;

  property Preview: boolean read FPreview write SetPreview;

  property VideoCaptureDeviceName: WideString
    read FVideoCaptureDeviceName
    write SetVideoCaptureDeviceName;
  property AudioCaptureDeviceName: WideString
    read FAudioCaptureDeviceName
    write SetAudioCaptureDeviceName;
  property VideoCompressDeviceName: WideString
    read FVideoCompressDeviceName
    write SetVideoCompressDeviceName;
  property AudioCompressDeviceName: WideString
    read FAudioCompressDeviceName
    write SetAudioCompressDeviceName;

  constructor Create(AHandle: THandle; ARect: TRect;
    APreview: boolean = TRUE);
  destructor Destroy; override;

  procedure ResetGraph;
  function ConstructGraph: HResult;

  procedure EnumVideoCaptureDevices(List: TStrings);
  procedure EnumAudioCaptureDevices(List: TStrings);
  procedure EnumVideoCompressDevices(List: TStrings);
  procedure EnumAudioCompressDevices(List: TStrings);

  function StartCapture: HResult;
  procedure StopCapture;

  function DisplayVideoCapturePinPropertyPage(
    Handle: THandle): HResult;
```

```
    function DisplayVideoCaptureDeviceProperty(Handle: THandle): HResult;
    function DisplayAudioCaptureDeviceProperty(Handle: THandle): HResult;
    function DisplayVideoCompressDeviceProperty(
      Handle: THandle): HResult;
    function DisplayAudioCompressDeviceProperty(
      Handle: THandle): HResult;
  END;

{**} IMPLEMENTATION {************************************************}

{**} { TdxCaptureManager } {*****************************************}

{*********************************************************************}
{** Конструктор класса                                              **}
{*********************************************************************}
constructor TdxCaptureManager.Create(AHandle: THandle; ARect: TRect;
  APreview: boolean);
begin
  // Запоминаем дескриптор окна предварительного просмотра
  FVideoHandle := AHandle;

  // Задаем состояние предварительного просмотра
  FPreview := APreview;

  // Позиция окна вывода на экране
  FVideoRect := ARect;

  // Обнуляем имя AVI-файла
  FCaptureFileName := '';

  // Обнуляем флаг захвата
  FCapturing := FALSE;
end;

{*********************************************************************}
{** Деструктор класса                                               **}
{*********************************************************************}
```

```
destructor TdxCaptureManager.Destroy;
begin
  // Освобождаем выделенную память
  ResetGraph;
end;

{*************************************************************************}
{** Построение графа фильтров                                           **}
{*************************************************************************}
function TdxCaptureManager.ConstructGraph: HResult;
var
  pConfigMux: IConfigAviMux;
  R: integer;
begin
  // Чистим граф
  ResetGraph;

  // Создаем объект для построения графа фильтров
  Result := CoCreateInstance(CLSID_FilterGraph, NIL,
    CLSCTX_INPROC_SERVER, IID_IGraphBuilder, FGraphBuilder);
  if FAILED(Result) then EXIT;

  // Создаем объект для построения графа захвата
  Result := CoCreateInstance(CLSID_CaptureGraphBuilder2, NIL,
    CLSCTX_INPROC_SERVER, IID_ICaptureGraphBuilder2,
     FCaptureGraphBuilder);
  if FAILED(Result) then EXIT;

  // Задаем граф фильтров для использования в построении графа захвата
  Result := FCaptureGraphBuilder.SetFiltergraph(FGraphBuilder);
  if FAILED(Result) then EXIT;

  // Получение устройства захвата видео
  FVideoCaptureFilter := EnumerateDevices(CLSID_VideoInputDeviceCategory,
    VideoCaptureDeviceName, NIL, TRUE);
```

```
// Получение устройства захвата звука
FAudioCaptureFilter := EnumerateDevices(CLSID_AudioInputDeviceCategory,
  AudioCaptureDeviceName, NIL, TRUE);

// Получение устройства сжатия видео
if VideoCompressDeviceName <> '' then
begin
  FVideoCompressFilter := EnumerateDevices(
    CLSID_VideoCompressorCategory,
    VideoCompressDeviceName, NIL, TRUE);
end;

// Получение устройства сжатия звука
if AudioCompressDeviceName <> '' then
begin
  FAudioCompressFilter := EnumerateDevices(
    CLSID_AudioCompressorCategory,
    AudioCompressDeviceName, NIL, TRUE);
end;

// Добавляем фильтр захвата видео в граф
if FVideoCaptureFilter <> NIL then
begin

  FGraphBuilder.AddFilter(FVideoCaptureFilter, 'VideoCaptureFilter');
end;

// Добавляем фильтр захвата звука в граф
if FAudioCaptureFilter <> NIL then
begin
  FGraphBuilder.AddFilter(FAudioCaptureFilter, 'AudioCaptureFilter');
end;

// Добавляем фильтр сжатия видео в граф
if FVideoCompressFilter <> NIL then
begin
```

```
  FGraphBuilder.AddFilter(FVideoCompressFilter, 'VideoCompressFilter');
end;

// Добавляем фильтр сжатия звука в граф
if FAudioCompressFilter <> NIL then
begin
  FGraphBuilder.AddFilter(FAudioCompressFilter, 'AudioCompressFilter');
end;

// Если задан режим предварительного просмотра, то...
if FPreview then
begin
  // ... выводим изображение
  if FVideoCaptureFilter <> NIL then
  begin

    Result := FCaptureGraphBuilder.RenderStream(
      @PIN_CATEGORY_PREVIEW,
      @MEDIATYPE_Video,
      FVideoCaptureFilter,
      NIL,
      NIL);
    if FAILED(Result) then EXIT;

    if FVideoHandle > 0 then
    begin
      // Запрашиваем интерфейс управления окном вывода изображения
      FGraphBuilder.QueryInterface(IID_IVideoWindow, FVideoWindow);

      if FVideoWindow <> NIL then
      begin
        // Устанавливаем стиль видео окна
        FVideoWindow.put_WindowStyle(WS_CHILD or WS_CLIPSIBLINGS);

        // Устанавливаем родительское окно для вывода изображения
        FVideoWindow.put_Owner(FVideoHandle);
```

```
        // Устанавливаем положение окна
        FVideoWindow.SetWindowPosition(
          FVideoRect.Left,
          FVideoRect.Top,
          FVideoRect.Right - FVideoRect.Left,
          FVideoRect.Bottom - FVideoRect.Top);

        // Показываем окно вывода изображения
        FVideoWindow.put_Visible(TRUE);
      end;
    end;

  end;

  // ... выводим звук
  if FAudioCaptureFilter <> NIL then
  begin
    Result := FCaptureGraphBuilder.RenderStream(
      @PIN_CATEGORY_PREVIEW,
      @MEDIATYPE_Audio,
      FAudioCaptureFilter,
      NIL,
      NIL);
    if FAILED(Result) then EXIT;
  end;
end;

// Если задан режим захвата, то...
if FCapturing then
begin
  // Создаем файл для записи данных из графа
  Result := FCaptureGraphBuilder.SetOutputFileName(MEDIASUBTYPE_Avi,
    PWideChar(FCaptureFileName), FMux, FSink);
  if FAILED(Result) then EXIT;

  // Устанавливаем режим захвата изображения
  if FVideoCaptureFilter <> NIL then
```

```
  begin
    Result := FCaptureGraphBuilder.RenderStream(
      @PIN_CATEGORY_CAPTURE,
      @MEDIATYPE_Video,
      FVideoCaptureFilter,
      FVideoCompressFilter,
      FMux);

    if FAILED(Result) then EXIT;
  end;

  // Устанавливаем режим захвата звука
  if FAudioCaptureFilter <> NIL then
  begin
    Result := FCaptureGraphBuilder.RenderStream(
      @PIN_CATEGORY_CAPTURE,
      @MEDIATYPE_Audio,
      FAudioCaptureFilter,
      FAudioCompressFilter,
      FMux);
    if FAILED(Result) then EXIT;

    // При захвате видео со звуком устанавливаем звуковой поток в
    // качестве основного для синхронизации с другими потоками в файле
    if FVideoCaptureFilter <> NIL then
    begin
      pConfigMux := NIL;
      Result := FMux.QueryInterface(IID_IConfigAviMux, pConfigMux);
      if (SUCCEEDED(Result)) then
      begin
        pConfigMux.SetMasterStream(1);
        pConfigMux := NIL;
      end;
    end;
  end;
end;
```

```
  // Запрашиваем интерфейс управления графом
  Result := FGraphBuilder.QueryInterface(IID_IMediaControl,
    FMediaControl);
  if FAILED(Result) then EXIT;

  // Запускаем граф
  Result := FMediaControl.Run();
end;

{*********************************************************************}
{** Перечисление устройств определенного класса                     **}
{*********************************************************************}
function TdxCaptureManager.EnumerateDevices(const clsidDeviceClass:
  TGUID; DevName: WideString; DevList: TStrings; GetFirst: boolean):
IBaseFilter;
var
  DeviceName:   OleVariant;
  PropertyName: IPropertyBag;
  pDevEnum:     ICreateDevEnum;
  pEnum:        IEnumMoniker;
  pFilter:      IBaseFilter;
  pMoniker:     IMoniker;
  hr:           HResult;
begin
  // Обнуляем ссылки
  pMoniker     := NIL;
  pFilter      := NIL;
  PropertyName := NIL;
  pDevEnum     := NIL;
  pEnum        := NIL;

  // Результат по умолчанию
  Result       := NIL;

  // Создаем объект для перечисления устройств
  hr := CoCreateInstance(CLSID_SystemDeviceEnum, NIL,
```

```
  CLSCTX_INPROC_SERVER, IID_ICreateDevEnum, pDevEnum);
if FAILED(hr) then EXIT;

// Создаем перечислитель для указанной категории устройств
hr := pDevEnum.CreateClassEnumerator(clsidDeviceClass, pEnum, 0);
if (hr <> S_OK) then EXIT;

// Цикл по устройствам
while (S_OK = pEnum.Next(1, pMoniker, NIL)) do
begin
  // Если нам нужен список устройств, то...
  if not GetFirst then
  begin
    // ... получаем интерфейс хранилища, которое содержит объект,
    // идентифицируемый моникером
    hr := pMoniker.BindToStorage(NIL, NIL, IPropertyBag, PropertyName);
    if FAILED(hr) then Continue;

    // Читаем значение свойства
    hr := PropertyName.Read('FriendlyName', DeviceName, NIL);
    if FAILED(hr) then Continue;

    // Добавляем название устройства в наш список
    if DevList <> NIL then DevList.Add(DeviceName);
  end

  // Если нам нужно получить первое устройство, то...
  else begin

    // Если указано имя устройства, то...
    if DevName <> '' then
    begin
      // ... получаем интерфейс хранилища, которое содержит объект,
      // идентифицируемый моникером
      hr := pMoniker.BindToStorage(NIL, NIL, IPropertyBag,
        PropertyName);
      if FAILED(hr) then Continue;
```

```
        // Читаем значение свойства
        hr := PropertyName.Read('FriendlyName', DeviceName, NIL);
        if FAILED(hr) then Continue;

        // Продолжаем поиск, если не совпадают имена устройств
        if (DeviceName <> DevName) then Continue;
      end;

      // Используя моникер, связываемся с объектом, который он
      // идентифицирует, и получаем нужный нам интерфейс
      hr := pMoniker.BindToObject(NIL, NIL, IID_IBaseFilter, pFilter);
      if SUCCEEDED(hr) then
      begin
        // Результат - полученный интерфейс
        Result := pFilter;
        // Освобождаем память
        pEnum := NIL;
        pDevEnum := NIL;
        pMoniker := NIL;
        PropertyName := NIL;

        // Выходим из процедуры досрочно
        EXIT;
      end;
    end;
  end;

  // Освобождаем память
  pEnum        := NIL;
  pDevEnum     := NIL;
  PropertyName := NIL;
  pFilter      := NIL;
  pMoniker     := NIL;
end;
```

```
{*************************************************************************}
{** Получение списка устройств захвата видео                            **}
{*************************************************************************}
procedure TdxCaptureManager.EnumVideoCaptureDevices(List: TStrings);
begin
  EnumerateDevices(CLSID_VideoInputDeviceCategory, '', List);
end;

{*************************************************************************}
{** Получение списка устройств сжатия видео                             **}
{*************************************************************************}
procedure TdxCaptureManager.EnumVideoCompressDevices(List: TStrings);
begin
  EnumerateDevices(CLSID_VideoCompressorCategory, '', List);
end;

{*************************************************************************}
{** Получение списка устройств захвата аудио                            **}
{*************************************************************************}
procedure TdxCaptureManager.EnumAudioCaptureDevices(List: TStrings);
begin
  EnumerateDevices(CLSID_AudioInputDeviceCategory, '', List);
end;

{*************************************************************************}
{** Получение списка устройств сжатия аудио                             **}
{*************************************************************************}
procedure TdxCaptureManager.EnumAudioCompressDevices(List: TStrings);
begin
  EnumerateDevices(CLSID_AudioCompressorCategory, '', List);
end;

{*************************************************************************}
{** Чистка графа - освобождаем память                                   **}
{*************************************************************************}
procedure TdxCaptureManager.ResetGraph;
```

```
begin
  FAudioCompressFilter := NIL;
  FVideoCompressFilter := NIL;
  FAudioCaptureFilter  := NIL;
  FVideoCaptureFilter  := NIL;
  FVideoWindow         := NIL;
  FMediaControl        := NIL;
  FSink                := NIL;
  FMux                 := NIL;
  FCaptureGraphBuilder := NIL;
  FGraphBuilder        := NIL;
end;

{*************************************************************************}
{** Установка имени устройства работы с видео                           **}
{*************************************************************************}
procedure TdxCaptureManager.SetVideoCaptureDeviceName(Value: WideString);
begin
  FVideoCaptureDeviceName := Value;
end;

{*************************************************************************}
{** Установка имени устройства сжатия видео                             **}
{*************************************************************************}
procedure TdxCaptureManager.SetVideoCompressDeviceName(Value:
  WideString);
begin
  FVideoCompressDeviceName := Value;
end;

{*************************************************************************}
{** Установка имени устройства работы со звуком                         **}
{*************************************************************************}
procedure TdxCaptureManager.SetAudioCaptureDeviceName(Value: WideString);
begin
  FAudioCaptureDeviceName := Value;
end;
```

```
{*************************************************************************}
{** Установка имени устройства сжатия звука                             **}
{*************************************************************************}
procedure TdxCaptureManager.SetAudioCompressDeviceName(Value:
  WideString);
begin
  FAudioCompressDeviceName := Value;
end;

{*************************************************************************}
{** Установка режима предварительного просмотра                         **}
{*************************************************************************}
procedure TdxCaptureManager.SetPreview(Value: boolean);
begin
  // Установка значения свойства
  FPreview := Value;

  // Перестраимваем граф
  ConstructGraph;
end;

{*************************************************************************}
{** Начинаем запись                                                     **}
{*************************************************************************}
function TdxCaptureManager.StartCapture: HResult;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если уже находимся в состоянии записи, то завершаем работу
  if FCapturing then EXIT;

  // Выставляем флаг записи
  FCapturing := TRUE;
```

```
  // Перестраиваем граф
  ConstructGraph;
end;

{*************************************************************************}
{** Останавливаем процесс записи                                        **}
{*************************************************************************}
procedure TdxCaptureManager.StopCapture;
begin
  // Если запись не производится, то завершаем работу
  if not FCapturing then EXIT;

  // Выставляем флаг записи
  FCapturing := FALSE;

  // Перестраиваем граф
  ConstructGraph;
end;

{*************************************************************************}
{** Вызов страницы свойств контакта потока видео                        **}
{*************************************************************************}
function TdxCaptureManager.DisplayVideoCapturePinPropertyPage(
  Handle: THandle): HResult;
var
  StreamConfig: IAMStreamConfig;
  PropertyPages: ISpecifyPropertyPages;
  Pages: CAUUID;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если отсутствует интерфейс работы с видео, то завершаем работу
  if FVideoCaptureFilter = NIL then EXIT;

  // Останавливаем работу графа
  FMediaControl.Stop;
```

```
try

  // Ищем интерфейс управления форматом данных выходного потока
  Result := FCaptureGraphBuilder.FindInterface(
    @PIN_CATEGORY_CAPTURE,
    @MEDIATYPE_Video,
    FVideoCaptureFilter,
    IID_IAMStreamConfig,
    StreamConfig);

  // Если интерфейс найден, то...
  if SUCCEEDED(Result) then
  begin
    // ... пытаемся найти интерфейс управления страницами свойств...
    Result := StreamConfig.QueryInterface(ISpecifyPropertyPages,
      PropertyPages);

    // ... и, если он найден, то...
    if SUCCEEDED(Result) then
    begin
      // ... получаем массив страниц свойств
      PropertyPages.GetPages(Pages);
      PropertyPages := NIL;

      // Отображаем страницу свойств в виде модального диалога
      OleCreatePropertyFrame(
         Handle,
         0,
         0,
         PWideChar(VideoCaptureDeviceName),
         1,
         @StreamConfig,
         Pages.cElems,
         Pages.pElems,
         0,
         0,
```

```
          NIL
        );

        // Освобождаем память
        StreamConfig := NIL;
        CoTaskMemFree(Pages.pElems);
      end;
    end;

  finally
    // Восстанавливаем работу графа
    FMediaControl.Run;
  end;
end;

{*********************************************************************}
{** Вызов страницы свойств заданного фильтра                        **}
{*********************************************************************}
function TdxCaptureManager.DisplayPropertyFrame(Filter: IBaseFilter;
  Handle: THandle): HResult;
var
  PropertyPages: ISpecifyPropertyPages;
  Pages: CAUUID;
  FilterInfo: TFilterInfo;
  pfilterUnk: IUnknown;
begin
  // Результат по умолчанию
  Result := E_FAIL;

  // Если фильтр не определен, то завершаем работу
  if Filter = NIL then EXIT;

  // Пытаемся найти интерфейс управления страницами свойств фильтра
  Result := Filter.QueryInterface(ISpecifyPropertyPages, PropertyPages);

  if (SUCCEEDED(Result)) then
  begin
```

```
    // Получение имени фильтра и указателя на интерфейс IUnknown
    Filter.QueryFilterInfo(FilterInfo);
    Filter.QueryInterface(IUnknown, pfilterUnk);

    // Получаем массив страниц свойств
    PropertyPages.GetPages(Pages);
    PropertyPages := NIL;

    // Отображаем страницу свойств в виде модального диалога
    OleCreatePropertyFrame(
       Handle,
       0,
       0,
       FilterInfo.achName,
       1,
       @pfilterUnk,
       Pages.cElems,
       Pages.pElems,
       0,
       0,
       NIL
    );

    // Освобождаем память
    pfilterUnk := NIL;
    FilterInfo.pGraph := NIL;

    CoTaskMemFree(Pages.pElems);
  end;
end;

{**************************************************************************}
{** Вызов страницы свойств устройства работы с видео                     **}
{**************************************************************************}
```

```
function TdxCaptureManager.DisplayVideoCaptureDeviceProperty(
  Handle: THandle): HResult;
begin
  Result := DisplayPropertyFrame(FVideoCaptureFilter, Handle);
end;

{*********************************************************************}
{** Вызов страницы свойств устройства сжатия видео                  **}
{*********************************************************************}
function TdxCaptureManager.DisplayVideoCompressDeviceProperty(
  Handle: THandle): HResult;
begin
  Result := DisplayPropertyFrame(FVideoCompressFilter, Handle);
end;

{*********************************************************************}
{** Вызов страницы свойств устройства работы со звуком              **}
{*********************************************************************}
function TdxCaptureManager.DisplayAudioCaptureDeviceProperty(
  Handle: THandle): HResult;
begin
  Result := DisplayPropertyFrame(FAudioCaptureFilter, Handle);
end;

{*********************************************************************}
{** Вызов страницы свойств устройства сжатия звука                  **}
{*********************************************************************}
function TdxCaptureManager.DisplayAudioCompressDeviceProperty(
  Handle: THandle): HResult;
begin
  Result := DisplayPropertyFrame(FAudioCompressFilter, Handle);
end;

END.
```

Пример использования класса *TdxCaptureManager*

Для демонстрации возможностей класса `TdxCaptureManager` мною написан специальный пример. Он позволяет захватывать изображение от различных устройств (в примере используется Web-камера) и звуковой поток. Имеются режимы предварительного просмотра (прослушивания) и режим записи на диск. Устройства захвата изображения, звука и устройства сжатия данных можно выбрать из соответствующих списков (рис. 15.14). Пример находится в каталоге Examples\DirectShow\DSH_Cap_Test на компакт-диске (листинг 15.2).

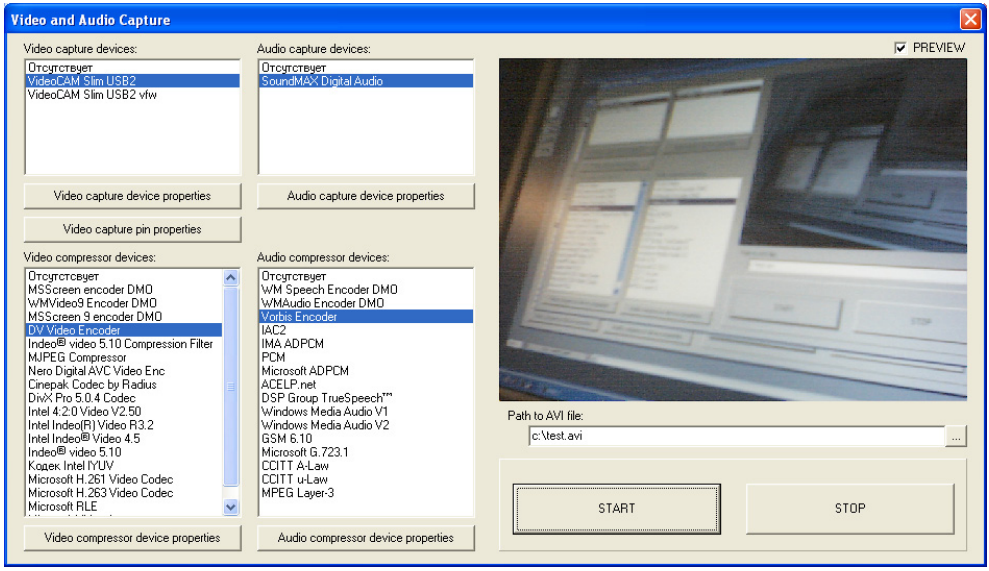


Рис. 15.14. Общий вид тестирующего приложения для класса `TdxCaptureManager`

Листинг 15.2. Текст модуля FormMain.pas проекта DSH_Cap_Test

```
UNIT FormMain;
{*************************************************************************}
{** Тестирование класса TdxCaptureManager                               **}
{** Автор: Есенин Сергей Анатольевич                                    **}
{*************************************************************************}
```

```
{**} INTERFACE {*************************************************************}

{**} USES {******************************************************************}
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, DirectShow9, ActiveX, ComObj, ExtCtrls, StdCtrls,
  UdxCaptureManager;

{**} TYPE {******************************************************************}
  TMainForm = class(TForm)
    panelVideo: TPanel;
    labelVideo: TLabel;
    labelAudio: TLabel;
    labelVideoCompressor: TLabel;
    labelAudioCompressor: TLabel;
    listBoxVideo: TListBox;
    listBoxVideoCompressor: TListBox;
    listBoxAudio: TListBox;
    listBoxAudioCompressor: TListBox;
    panelRecord: TPanel;
    editFileName: TEdit;
    checkBoxPreview: TCheckBox;
    buttonVideoProperties: TButton;
    buttonVideoCompressorProperties: TButton;
    buttonAudioProperties: TButton;
    buttonAudioCompressorProperties: TButton;
    buttonVideoPinProperties: TButton;
    labelFileName: TLabel;
    buttonStartCapture: TButton;
    buttonStopCapture: TButton;
    saveDialogMain: TSaveDialog;
    buttonFileName: TButton;

    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure buttonStartCaptureClick(Sender: TObject);
    procedure buttonStopCaptureClick(Sender: TObject);
```

```
    procedure listBoxVideoClick(Sender: TObject);
    procedure listBoxVideoCompressorClick(Sender: TObject);
    procedure listBoxAudioClick(Sender: TObject);
    procedure listBoxAudioCompressorClick(Sender: TObject);
    procedure checkBoxPreviewClick(Sender: TObject);
    procedure buttonVideoPropertiesClick(Sender: TObject);
    procedure buttonVideoCompressorPropertiesClick(Sender: TObject);
    procedure buttonAudioPropertiesClick(Sender: TObject);
    procedure buttonAudioCompressorPropertiesClick(Sender: TObject);
    procedure buttonVideoPinPropertiesClick(Sender: TObject);
    procedure buttonFileNameClick(Sender: TObject);
  PRIVATE
    FRefreshing: boolean;
  PUBLIC
  END;

{**} VAR {************************************************************}
  MainForm: TMainForm;
  CaptureManager: TdxCaptureManager;

{**} IMPLEMENTATION {*************************************************}

{$R *.dfm}

{*********************************************************************}
{** Создание и инициализация объекта CaptureManager                 **}
{*********************************************************************}
procedure TMainForm.FormCreate(Sender: TObject);
begin
  // Выставляем флаг обновления
  FRefreshing := TRUE;

  try
    // Создаем объект CaptureManager
    CaptureManager := TdxCaptureManager.Create(
      panelVideo.Handle,
      panelVideo.ClientRect);
```

```
listBoxVideo.Items.Add('Отсутствует');
listBoxVideoCompressor.Items.Add('Отсутстсвует');
listBoxAudio.Items.Add('Отсутствует');
listBoxAudioCompressor.Items.Add('Отсутствует');

// Заполняем списки аудио- и видеоустройств
with CaptureManager do
begin
  EnumVideoCaptureDevices(listBoxVideo.Items);
  EnumVideoCompressDevices(listBoxVideoCompressor.Items);
  EnumAudioCaptureDevices(listBoxAudio.Items);
  EnumAudioCompressDevices(listBoxAudioCompressor.Items);
end;

// Если в системе присутствуют устройства захвата изображения,
// то останавливаем свой выбор на первом из них
if listBoxVideo.Count > 1 then
  listBoxVideo.ItemIndex := 1
else
  listBoxVideo.ItemIndex := 0;
listBoxVideoClick(NIL);

// Устройство сжатия видео по умолчанию отсутствует
listBoxVideoCompressor.ItemIndex := 0;
listBoxVideoCompressorClick(NIL);

// Если в системе присутствуют устройства захвата аудио,
// то останавливаем свой выбор на первом из них
if listBoxAudio.Count > 1 then
  listBoxAudio.ItemIndex := 1
else
  listBoxAudio.ItemIndex := 0;
listBoxAudioClick(NIL);

// Устройство сжатия аудио по умолчанию отсутствует
listBoxAudioCompressor.ItemIndex := 0;
listBoxAudioCompressorClick(NIL);
```

```
    // Строим граф фильтров
    CaptureManager.ConstructGraph;

  finally

    FRefreshing := FALSE;
  end;
end;

{*************************************************************************}
{** Удаление объекта CaptureManager                                     **}
{*************************************************************************}
procedure TMainForm.FormDestroy(Sender: TObject);
begin
  if CaptureManager <> NIL then
     FreeAndNil(CaptureManager);
end;

{*************************************************************************}
{** Начинаем запись                                                     **}
{*************************************************************************}
procedure TMainForm.buttonStartCaptureClick(Sender: TObject);
begin
  with CaptureManager do
  begin
    // Задаем имя AVI-файла
    CaptureFileName := editFileName.Text;

    // Начинаем запись
    StartCapture;
  end;

  // Устанавливаем красный цвет панели-индикатора
  panelRecord.Color := clRed;
end;
```

```
{*************************************************************************}
{** Останавливаем запись                                                **}
{*************************************************************************}
procedure TMainForm.buttonStopCaptureClick(Sender: TObject);
begin
  // Восстанавливаем цвет панели-индикатора
  panelRecord.Color := clBtnFace;

  // Останавливаем запись
  CaptureManager.StopCapture;
end;

{*************************************************************************}
{** Выбор устройства работы с видео                                     **}
{*************************************************************************}
procedure TMainForm.listBoxVideoClick(Sender: TObject);
begin
  // Задаем имя устройства
  CaptureManager.VideoCaptureDeviceName :=
    listBoxVideo.Items.Strings[listBoxVideo.ItemIndex];

  // Перестраиваем граф фильтров
  if not FRefreshing then
     CaptureManager.ConstructGraph;
end;

{*************************************************************************}
{** Выбор устройства сжатия видео                                       **}
{*************************************************************************}
procedure TMainForm.listBoxVideoCompressorClick(Sender: TObject);
begin
  // Задаем имя устройства
  CaptureManager.VideoCompressDeviceName :=
    listBoxVideoCompressor.Items.Strings[
        listBoxVideoCompressor.ItemIndex];
```

```
  // Перестраиваем граф фильтров
  if not FRefreshing then
     CaptureManager.ConstructGraph;
end;

{*********************************************************************}
{** Выбор устройства работы со звуком                               **}
{*********************************************************************}
procedure TMainForm.listBoxAudioClick(Sender: TObject);
begin
  // Задаем имя устройства
  CaptureManager.AudioCaptureDeviceName :=
    listBoxAudio.Items.Strings[listBoxAudio.ItemIndex];

  // Перестраиваем граф фильтров
  if not FRefreshing then
     CaptureManager.ConstructGraph;
end;

{*********************************************************************}
{** Выбор устройства сжатия звука                                   **}
{*********************************************************************}
procedure TMainForm.listBoxAudioCompressorClick(Sender: TObject);
begin
  // Задаем имя устройства
  CaptureManager.AudioCompressDeviceName :=
    listBoxAudioCompressor.Items.Strings[
        listBoxAudioCompressor.ItemIndex];

  // Перестраиваем граф фильтров
  if not FRefreshing then
     CaptureManager.ConstructGraph;
end;

{*********************************************************************}
{** Управление режимом предварительного просмотра                   **}
{*********************************************************************}
```

```
procedure TMainForm.checkBoxPreviewClick(Sender: TObject);
begin
  CaptureManager.Preview := checkBoxPreview.Checked;
end;

{*********************************************************************}
{** Вызов страницы свойств устройства работы с видео                **}
{*********************************************************************}
procedure TMainForm.buttonVideoPropertiesClick(Sender: TObject);
begin
  if FAILED(CaptureManager.DisplayVideoCaptureDeviceProperty(Handle))
  then begin
    ShowMessage('Error in call: DisplayVideoCaptureDeviceProperty');
  end;
end;

{*********************************************************************}
{** Вызов страницы свойств устройства сжатия видео                  **}
{*********************************************************************}
procedure TMainForm.buttonVideoCompressorPropertiesClick(Sender: TOb-
ject);
begin
  if FAILED(CaptureManager.DisplayVideoCompressDeviceProperty(Handle))
  then begin
    ShowMessage('Error in call: DisplayVideoCompressDevicePropertys');
  end;
end;

{*********************************************************************}
{** Вызов страницы свойств устройства работы со звуком              **}
{*********************************************************************}
procedure TMainForm.buttonAudioPropertiesClick(Sender: TObject);
begin
  if FAILED(CaptureManager.DisplayAudioCaptureDeviceProperty(Handle))
  then begin
    ShowMessage('Error in call: DisplayAudioCaptureDeviceProperty');
  end;
end;
```

```
{*************************************************************************}
{** Вызов страницы свойств устройства сжатия звука                      **}
{*************************************************************************}
procedure TMainForm.buttonAudioCompressorPropertiesClick(Sender:
  TObject);
begin
  if FAILED(CaptureManager.DisplayAudioCompressDeviceProperty(Handle))
  then begin
    ShowMessage('Error in call: DisplayAudioCompressDeviceProperty');
  end;
end;

{*************************************************************************}
{** Вызов страницы свойств контакта потока видео                        **}
{*************************************************************************}
procedure TMainForm.buttonVideoPinPropertiesClick(Sender: TObject);
begin
  if FAILED(CaptureManager.DisplayVideoCapturePinPropertyPage(Handle))
  then begin
    ShowMessage('Error in call: DisplayVideoPropertyPage');
  end;
end;

{*************************************************************************}
{** Выбор имени сохраняемого файла                                      **}
{*************************************************************************}
procedure TMainForm.buttonFileNameClick(Sender: TObject);
begin
  if saveDialogMain.Execute then
  begin
    editFileName.Text := saveDialogMain.FileName;
  end;
end;

END.
```

Итоги

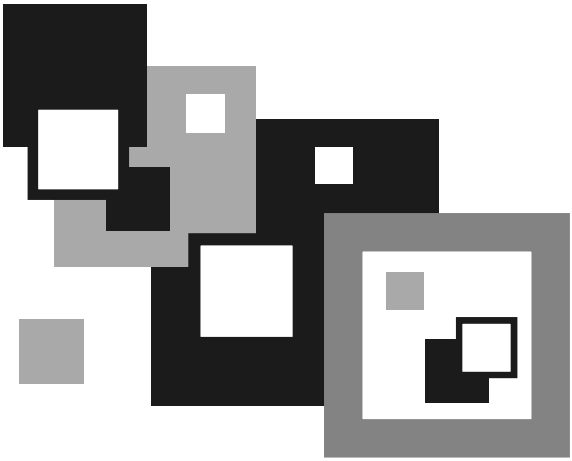
Данная часть книги является заключительной. Изученная подсистема, именуемая DirectShow, предоставляет нам обширные возможности по управлению потоками мультимедиаданных. Форматы поддерживаемых данных были перечислены в самом начале. Мы научились воспроизводить потоки мультимедиа и управлять процессом захвата изображения и звука и записи их в файл. Классы `TdxMediaPlayer` и `TdxCaptureManager` должны помочь вам в освоении данной темы.

Заключение

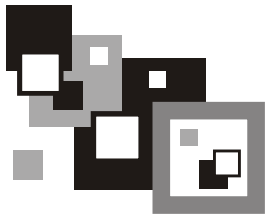
Хотелось бы надеяться, что моя книга была вам не только интересна, но и полезна в изучении такой мощной системы, как DirectX. Специально для тех, у кого найдутся какие-либо замечания или предложения, почтовый ящик издательства **mail@bhv.ru**. Пишите, задавайте вопросы.

Как я уже упоминал в самом начале, книга не является полным руководством по системе DirectX. В книге обсуждены такие важнейшие компоненты DirectX, как DirectX Graphics, DirectSound, DirectMusic, DirectInput и даже DirectShow. Рассмотрена работа с двумерной и трехмерной графикой, вершинные и пиксельные шейдеры и язык HLSL, работа с устройствами ввода, работа с различными форматами аудиофайлов. Мы научились воспроизводить различные мультимедиаданные, будь то звуковой формат MP3 или формат видео AVI. Научились получать изображение с устройства захвата (Web-камера, IP-камера, TV-тюнер и т. д.) и сохранять его на диске вместе со звуком, полученным от устройства захвата звука; научились сжимать полученные от устройств захвата данные. Работа с подсистемой DirectX Graphics была построена по принципу примеров, а для работы с остальными подсистемами были разработаны собственные классы и представлены примеры их использования.

Информации, изложенной в книге, должно быть вполне достаточно для понимания общих принципов функционирования различных составляющих DirectX, а также для дальнейшего самостоятельного изучения всех тонкостей. Предложенные классы для работы с системой DirectX могут наращиваться вами дополнительной функциональностью самостоятельно.



Приложения



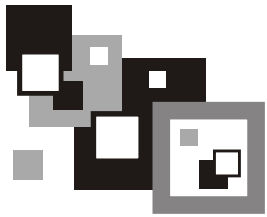
Приложение 1

Интернет-ресурсы

В настоящее время в Интернете имеется достаточно большое количество сайтов, посвященных разработке программ в среде Borland Delphi (и других средах, таких как Microsoft Visual C++, Borland C++ Builder и т. п.). Далеко не у всех этих сайтов основной тематикой является использование возможностей DirectX, но, тем не менее, они содержат ряд интересных статей по данной теме. Вот далеко не полный перечень этих сайтов с кратким описанием.

- **www.delphikingdom.ru** (королевство Delphi) — сайт, посвященный разработке программ в среде Borland Delphi. По своей сути является виртуальным клубом программистов. Содержит множество интереснейших статей по различным аспектам разработки программного обеспечения и не только.
- **www.rsdn.ru** (проект RSDN) — создан программистами для программистов. Главная цель создателей сайта заключается в том, чтобы хоть как-то компенсировать нехватку материалов по программированию на русском языке. Сайт содержит большую подборку статей разной тематики, различные форумы, новостную ленту и еще много чего интересного, что невозможно описать парой-тройкой слов.
- **www.xdev.ru** — сайт будет интересен в первую очередь разработчикам игр, как профессиональных, так и непрофессиональных, просто энтузиастам и любителям. На сайте имеется большое количество материала по DirectX, OpenGL, множество различных статей с общим содержанием; рассматриваются последние технологии — DirectX 8, DirectX 9, материалы о создании игр, скриптов, различных технологиях, трехмерной и двумерной графике и т. д. Имеются новостные ленты, в которые вы можете сами добавлять новости, колонка новостей сайта, где публикуется информация о последних обновлениях на сайте, новости индустрии разработки компьютерных игр. Действует система форумов. Имеются системы голосований и опросов.

- **www.gamedev.ru** (разработка игр) — основная цель проекта GameDev.ru — быть полезным разработчикам игр. В мировой индустрии производства компьютерных игр не так много очень хороших специалистов. Понятно, что у нас специалистов высокого плана вообще катастрофически мало. А кто занимается этой областью, знает, насколько тяжело, в основном методом проб и ошибок, достаются знания. Может быть, этот проект и станет для кого-то некоторой помощью, кто-то найдет новых знакомых и друзей с совпадающими интересами. Здесь будут собираться статьи самой разной сложности, как основы для новичков, так и более углубленные темы, например, оптимизация или описание некоторых дополнительных возможностей отдельно взятого железа. Любой может поделиться своими мыслями и соображениями. Здесь будет собираться материал, помогающий как начинающим, так и продвинутым. Участвовать может каждый. Например, кто-то рассказывает какую-то идею, которая его озарила, или же решение задачи, над которой он долго бился и не хочет, чтоб другие шли таким же тяжелым путем. Другие же могут оценить эту идею, дополнить, привести свое, может быть, даже более эффективное, решение.
- **www.citforum.ru** (море аналитической информации) — девиз сайта как нельзя лучше отражает его содержимое — поистине огромное количество материала на различную тематику (программирование, безопасность, сети, СУБД, операционные системы и т. д.).
- **www.delphimaster.ru** (мастера Delphi) — сайт, посвященный разработке программ в среде Borland Delphi. Создавая его, разработчики ставили себе цель сделать не "еще один сайт по Delphi", а нечто универсальное, что действительно поможет программисту именно тогда, когда это необходимо. Здесь не только много полезной и интересной информации (советы, статьи, подсказки, вопросы-ответы, конференция), но и еще обширный русскоязычный каталог по Delphi-ресурсам. Есть универсальная поисковая система по всему сайту.
- **www.msdn.microsoft.com/directx** — раздел сайта компании Microsoft, который посвящен DirectX.



Приложение 2

Описание содержимого компакт-диска

К книге прилагается компакт-диск, содержащий весь набор примеров работы с двумерной и трехмерной графикой, набор классов, упрощающих работу с такими компонентами DirectX, как DirectSound, DirectMusic, DirectInput и DirectShow, а также примеры к классам.

Диск состоит из нескольких каталогов — Classes, DirectX, Examples и Lib. Каталог Classes содержит набор разработанных классов для работы с DirectX. В папке DirectX хранятся заголовочные файлы DirectX. Каталог Examples содержит всевозможные примеры и, в свою очередь, разбит на подкаталоги (DirectSound, DirectInput и т. д.), которые содержат соответствующие тематике примеры. В папке Lib находятся библиотеки, необходимые для работы примеров с подсистемой Direct3D. Вам необходимо переписать библиотеки в такой каталог у себя на компьютере, к которому прописан путь в настройках Windows, например, Windows\System32.

Автор книги выражает благодарность владельцу сайта **http://www.clootie.ru** Алексею Барковому за любезно предоставленные заголовочные файлы DirectX и библиотеки для работы с D3DX.

Список литературы

1. Горнаков С. Г. DirectX 9: Уроки программирования на C++. — СПб.: БХВ-Петербург, 2004. — 400 с.: ил.
2. Документация компании Microsoft: DirectX 9c SDK.
3. Краснов М. В. DirectX. Графика в проектах Delphi. — СПб.: БХВ-Петербург, 2005. — 416 с.: ил.
4. Поляков А. Ю., Брусенцев В. А. Программирование графики GDI+ и DirectX. — СПб.: БХВ-Петербург, 2005. — 368 с.: ил.
5. Тихомиров Ю. В. Программирование трехмерной графики — СПб.: БХВ — Санкт-Петербург, 1999. — 256 с.: ил.
6. Томпсон Н. Секреты программирования трехмерной графики для Windows 95 / Перев. с англ. — СПб.: Питер, 1997. — 352 с.: ил.
7. Шикин А. В., Боресков А. В. Компьютерная графика. Динамика, реалистические изображения. — М.: ДИАЛОГ-МИФИ, 1996. — 288 с.

Предметный указатель

A

ActiveX 20
AVI 411, 446

B

BeginScene 59
Borland Delphi 11

C

Capture Graph 438
Capture Graph Builder 438
Class Identifier (CLID) 16
CoCreateInstance 29, 329
CoInitializeEx 19, 329
COM 391
Component Object Model (COM) 14
  завершение работы 19
  инициализация 19
CoUninitialize 19, 329
CreateComObject 29
CreateDevice 38
CreateSoundBuffer 242

D

D3DX 81, 121, 126
D3DXCompileShaderFromFile 172
DDCopyBitmap 194
Direct3DCreate9 37
DirectDrawCreateEx 185
DirectInput8Create 357
Directional 103
DirectSoundCaptureCreate8 310
DirectSoundCaptureEnumerate 310
DirectSoundCreate8 247
DirectSoundEnumerate 247
DirectX 9
  Direct3D 9, 37, 157, 235
  DirectDraw 9, 184, 235, 391
  DirectInput 9, 355, 385
  DirectMusic 10
  DirectPlay 10
  DirectSetup 10
  DirectShow 9, 486
  DirectSound 9, 391
  DirectX Graphics 9, 184, 235
  DirectX Media Objects 10, 254
DllCanUnloadNow 21
DllGetClassObject 21
DllRegisterServer 21
DllUnregisterServer 21
DrawPrimitive 60, 66
Dynamic Link Library (DLL) 14

E

EndScene 59

F

Failed 18
Filter 390
Filter Graph 391
Filter Graph Manager 391
Frames Per Second 142

G

GDI 184, 191, 391
GetDC 194
Graphics Pipeline 162

H

High-Level Shader Language (HLSL) 164
HLSL 164, 235
HResult 17

I

Index Buffer 98
Interface Identifier (IID) 17

L

Line Lists 54
Line Strips 54

M

Mesh-объект 126
  куб 126
  полигон 127
  сфера 126, 127
  тор 126, 127
  цилиндр 127
  чайник 127
MIDI 351, 390, 411
Motion Blur 157
MP3 411
MPEG 411
Mux Filter 392

O

Ole Automation 23
OleCreatePropertyFrame 451
OpenGL 10

P

Pascal 11
Pin 392
Point Light 103
Point Lists 54
Projection Matrix 79

Q

QueryInterface 16

R

ReleaseDC 194
Renderer Filter 392

S

SDK 389
Shader 162
Source Filter 392
Spherical Mapping 147
Splitter Filter 392
Spotlight 103
Succeeded 18
Surface 192

T

TD3DLight9 105
TD3DPresentParameters 39
TD3DViewport9 58
TDSBufferDesc 243, 312
Transform Filter 392
Triangle Fan 54
Triangle List 54
Triangle Strip 54
TWaveFormatEx 244
Type Library 22

V

Vertex Buffer 54
VFW 437
View Matrix 79

W

WAV 239, 351, 390, 411
Windows Driver Model (WDM) 437
WMA 411
World Matrix 79

Z

Z-буфер 96

А

Альфа-канал 155
Анизотропия 123

Б

Блиттинг 196
Буфер:
  вершин 54
  вторичный 240, 253
  глубины 96
  индексов 98
  первичный 240, 253
Буфер захвата 311

В

Восстановление доступа
 к поверхностям 202

Г

Глубина цвета 191
Граф:
  захвата 438
  фильтров 391
Графический конвейер 162
Громкость звучания 257

Д

Декартова система координат:
  левосторонняя 79
  правосторонняя 79

З

Запись звука 310
Захват:
  видео 437
  звука 437
Звуковые эффекты 254

И

Идентификатор:
  глобальный 16
  интерфейса 17
  класса 16
  уникальный 16
Интерфейс 15
  IAMStreamConfig 442
  IBaseFilter 394, 395
  IBasicAudio 394, 401
  ICaptureGraphBuilder2 438
  ICreateDevEnum 443
  ID3DXConstantTable 165, 172
  ID3DXMesh 126
  IDirect3D9 37
  IDirect3DDevice9 37
  IDirect3DIndexBuffer9 98
  IDirect3DPixelShader9 179
  IDirect3DVertexBuffer9 54
  IDirect3DVertexShader9 172
  IDirectDraw7 185
  IDirectDrawClipper 213
  IDirectDrawSurface7 192
  IDirectInput8 357, 385
  IDirectInputDevice8 359, 385
  IDirectInputEffect 361
  IDirectMusic8 329
  IDirectMusicLoader8 329
  IDirectMusicPerformance8 329
  IDirectMusicSegment8 329
  IDirectSound8 242
  IDirectSoundBuffer8 248
  IDirectSoundCapture8 310
  IDirectSoundFXChorus8 256
  IDirectSoundFXCompressor8 256
  IDirectSoundFXDistortion8 256
  IDirectSoundFXEcho8 256
  IDirectSoundFXFlanger8 256
  IDirectSoundFXGargle8 256
  IDirectSoundFXI3DL2Reverb8 256
  IDirectSoundFXParamEq8 256
  IDirectSoundFXWavesReverb8 256
  IEnumMoniker 443
  IGraphBuilder 394, 396

IInterface 15
IMediaControl 394, 397
IMediaEventEx 394, 401
IMediaSeeking 394, 397
IMoniker 443
IPin 394, 395
IPropertyBag 443
ISampleGrabber 394, 405
ISpecifyPropertyPages 442, 451
IUnknown 15
IVideoWindow 394, 403
Источник света 103
направленный 103
прожекторный 103
точечный 103

K

Класс:
TdxCaptureManager 457, 477, 486
TdxInputManager 368, 385
TdxMediaPlayer 410, 486
TdxMusicManager 336, 351
TdxMusicSegment 336, 351
TdxSound 256, 282, 326
TdxSoundCapture 315, 326
TdxSoundManager 256, 282, 326
абстрактный 15
Клипер 213
Контакт 392
Куб 89

M

Материал 103, 107
Матрица 75
вида 79
вращения 77
единичная 76
идентичности 76
масштабирования 78
мировая 79
отражения 78
переноса 76
проекции 79
сложение с матрицей 75
умножение на матрицу 76
умножение на число 75
Менеджер графа фильтров 391, 393
Метод:
абстрактный 15
Гуро 85
Микропрограмма 170
Модель драйверов Windows 437
Мультимедиапоток 390
Мультиплексор 447
Мультитекстурирование 155

H

Нормаль 84

O

Объект сеточный 126
Оверлеи DirectX 218
Огонь 206
Оконный режим 213
Освещение 61, 103
диффузное 61

П

Палитра 205
Переключение страниц 197
Поверхность 192
внеэкранная 192
вторичная 192
первичная 192
Подсчет числа кадров 142
Полноэкранный режим 191
Положение на панораме 257
Построения:
в пространстве 74
на плоскости 74
Построитель графа захвата 438
Потеря:
буферов 253
доступа к поверхности 198
устройства 52

Примитив 54
линия 54, 65
последовательность линий 54
последовательность треугольников 54, 69
последовательность треугольников с общей вершиной 54, 71
последовательности линий 65
точка 54, 56
треугольник 54, 68
Прозрачность 151
Прорисовка сцены 42
Прямой доступ к поверхности 205
Прямоугольник 74

Р

Регистрация COM-сервера 28
Реестр Windows 28
Режим:
захвата 445
оконный 49
полноэкранный 49
предварительного просмотра 445
Рисование 54

С

Сжатие:
потока аудио 449
потока видео 449
Страница свойств 451
Сферические текстурные координаты 147

Т

Текст:
на плоскости 135
трехмерный 140
Текстура 118
Типы фильтров 392
Туман 113
вершинный 114
линейный 114
пиксельный 114
экспоненциальный 114
Туннель 150

У

Устройства ввода:
джойстик 355, 366
клавиатура 355, 362
мышь 355, 364
Устройства захвата:
звука 438
изображения 437

Ф

Фильтр 390
Audio Capture Filter 438
AVI Splitter 391
Default DirectSound Device 391
Smart Tee 446
WDM Capture Filter 437
Фильтрация текстур 122
анизотропная 123
билинейная 123
ближайшая точка 123
линейная 123
многоуровневая 123
Формат:
AIFF 390
ASF 390
AU 390
AVI 390
MPEG 390
QuickTime 390
SND 390
WMA 389
WMV 390
Формат сжатия:
Cinepak 390
Digital Video (DV) 390
ISO MPEG-4 video version 1.0 390
Microsoft MPEG-4 version 3 390

MJPEG 390
MPEG Audio Layer-3 (MP3) (decompression only) 390
MPEG-1 (decompression only) 390
Sipro Labs ACELP 390
Windows Media Audio 390
Windows Media Video 390

Ц

Цвет 61
диффузный 106
зеркальный 106
излучаемый 107
окружающей среды 106
Цветовой ключ 203

Ч

Частота дискретизации 257

Ш

Шейдер 162
вершинный 164, 170, 235
пиксельный 164, 179, 235

Э

Эффект размытия при движении 157

Я

Язык HLSL 164