АЛЕКСЕЙ БОРЕСКОВ

bhv

РАСШИРЕНИЯ OpenGL

- КРОССПЛАТФОРМЕННЫЕ БИБЛИОТЕКИ
- ТРЕХМЕРНЫЕ ТЕКСТУРЫ
- СИСТЕМЫ ЧАСТИЦ
- КУБИЧЕСКИЕ ТЕКСТУРНЫЕ КАРТЫ
- ПОПИКСЕЛЬНОЕ, ДИФФУЗНОЕ И БЛИКОВОЕ ОСВЕЩЕНИЕ
- КАРТЫ ВЫСОТ И НОРМАЛЕЙ
- ШЕЙДЕРЫ, GLSL

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Алексей Боресков

РАСШИРЕНИЯ OpenGL

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.068+800.92
ББК 32.973.26-018.1
Б82

Боресков А. В.

Б82 Расширения OpenGL. — СПб.: БХВ-Петербург, 2005. — 688 с.: ил.

ISBN 5-94157-614-5

Описываются основные и наиболее популярные расширения библиотеки OpenGL, их использование на платформах Windows и Linux. Представлена реализация большого количества эффектов, созданных с помощью этих расширений. Показан механизм расширений и его использование для доступа к возможностям ускорителей с помощью языка шейдеров высокого уровня GLSL. Приведено много примеров реализации различных задач, решающихся с помощью расширений OpenGL. Изложенные в книге материалы помогут разработчикам при написании приложений, использующих трехмерную графику: игр, систем визуализации данных, систем проектирования.

На компакт-диске содержатся полные тексты примеров, приведенных в книге, исходные коды авторских библиотек, вспомогательные программы.

Для разработчиков графических приложений, студентов и аспирантов соответствующих специальностей

УДК 681.3.068+800.92
ББК 32.973.26-018.1

Группа подготовки издания:

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

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.04.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 55,47.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

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

ISBN 5-94157-614-5

Оглавление

Введение (о чем эта книга)

Одной из наиболее быстро развивающихся областей информатики (как у нас принято называть *computer science*) является трехмерная компьютерная графика. Прогресс в этой области легко можно увидеть, сравнивая компьютерные игры за последние годы. Чуть более десяти лет прошло с момента выхода первой версии *Doom'a* (10 декабря 1994 года) до появления *Doom III*, однако графическая наполненность игры и уровень реализма выросли на несколько порядков.

Для программистов, занимающихся трехмерной графикой, особенно важно появление и быстрое развитие различных графических библиотек при существенном росте возможностей видеоадаптеров. Современный видеоадаптер — это фактически мощный ускоритель трехмерной графики (далее мы будем применять термин графический ускоритель или GPU (*Graphics Processing Unit*), обладающий огромными возможностями и производительностью.

Эффективное использование всех ресурсов современных графических ускорителей возможно только с соответствующими интерфейсами API (*Application Program Interface*). Такой API выступает в качестве связующего звена между приложением и графическим ускорителем (рис. В1).

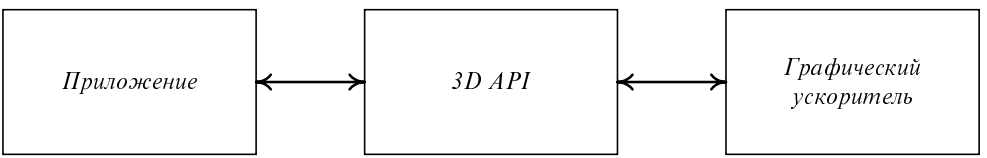


Рис. В1. Упрощенная схема взаимодействия приложения с графическим ускорителем

Одной из важных задач API является предоставление пользователю некоторого интерфейса, абстрагированного от конкретного графического ускорителя. Специфические особенности реального ускорителя обычно инкапсулируются в его драйвере (рис. В2).

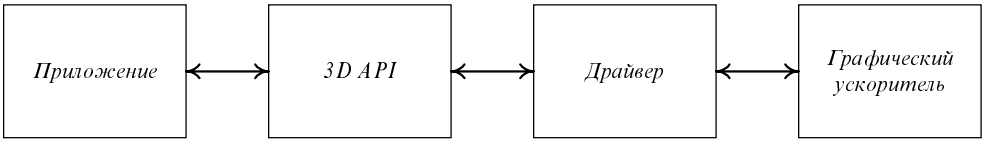


Рис. В2. Схема взаимодействия приложения с графическим ускорителем с учетом драйвера

На данный момент существуют всего два таких API, получившие достаточно широкое распространение для программирования трехмерной графики в реальном времени.

Это OpenGL, уже много лет являющийся стандартом как в области программирования трехмерной графики, так и обучения ей, и Direct3D, разработанный и усиленно проталкиваемый на рынок компанией Microsoft.

Библиотека OpenGL была изначально разработана компанией Silicon Graphics Inc., известной своими разработками в области трехмерной компьютерной графики, на основе собственной библиотеки IRIS GL.

Начиная с 1992 года, разработкой и поддержкой OpenGL занимается корпорация *OpenGL Architecture Review Board* (ARB). В состав OpenGL ARB входят такие компании, как Digital Equipment Corp., Evans & Sutherland, Hewlett-Packard Co., IBM Corp., Intel Corp., Silicon Graphics Inc., ATI, Compaq, nVidia, Intergraph, Sun Microsystems и Microsoft.

С самого начала OpenGL задумывался как открытый процедурный интерфейс к графическому ускорителю (в отличие от Direct3D, первые версии которого были ориентированы исключительно на программный (*software*) рендеринг, что сказалось на его общей архитектуре).

Важнейшими преимуществами (и характеристиками) OpenGL являются его простота, эффективность и независимость от конкретной аппаратной и программной платформы. Приложения на основе OpenGL хорошо работают как на персональных компьютерах под управлением Microsoft Windows, так и на компьютерах Silicon Graphics, Sun, Macintosh и др. Для переноса многих приложений (в частности, большинства примеров) достаточно просто перекомпилировать исходный текст на новой платформе.

При этом, поскольку сама библиотека OpenGL работает только с трехмерной графикой, то для использования ее на конкретной платформе обычно требуются дополнительные библиотеки, занимающиеся организацией взаимодействия с оконной системой, работой с текстурами и т. п.

В отличие от нее Direct3D, входящий в состав DirectX, предоставляет пользователю многие возможности (входящие зачастую не в Direct3D непосредственно, а в другие компоненты DirectX), не имеющие прямого отношения к программированию трехмерной графики, но довольно удобные для про-

граммистов (такие как взаимодействие с оконной системой, работа с текстурами и устройствами ввода).

Для OpenGL подобная функциональность предоставляется обычно целым рядом дополнительных библиотек, заметно облегчающих работу с ней. В этой книге мы будем использовать библиотеки GLU (*OpenGL Utilities*), GLUT (*OpenGL Utilities Toolkit*), а также некоторые библиотеки, разработанные автором. Все они содержатся на прилагаемом к книге компакт-диске. В конце книги вы найдете много полезных ссылок.

Большой ряд полезных возможностей (взаимодействие с оконной системой и устройствами ввода, работа с текстурами и звуком и т. п.) предоставляется библиотекой SDL (*Simple DirectMedia Layer*), также находящейся на компакт-диске, прилагаемом к книге.

Одной из самых удобных черт OpenGL является его спокойное эволюционное развитие, многие статьи и примеры, посвященные программированию с использованием OpenGL, по-прежнему сохраняют свою актуальность и полезность (чего никак нельзя сказать о Direct3D, который претерпевал сильные изменения фактически с каждой новой версией, причем большинство этих изменений значительно обесценивали опыт работы с предыдущей версией. Фактически если вы хотите найти пример реализации какой-то возможности на Direct3D, то вам нужно искать его именно на той версии Direct3D, с которой вы хотите работать в дальнейшем).

Еще одной неудобной чертой библиотеки Direct3D является то, что (как и весь интерфейс DirectX) она построена с использованием пресловутой модели COM. Для лиц, пишущих свои "программы" на Visual Basic, это, конечно, очень удобно. Однако для разработчиков на языке C++ модель COM представляет собой большое неудобство по сравнению с простым процедурным интерфейсом OpenGL.

Однако при таком эволюционном развитии OpenGL позволяет программисту легко и эффективно использовать в своих программах все возможности современных графических ускорителей (например, в игре *Doom III* для работы с графикой служит именно библиотека OpenGL; более того, в ее ресурсных файлах вы можете легко найти используемые вершинные и фрагментные программы).

Книга, которую вы сейчас держите в своих руках, посвящена именно тому, как при работе с OpenGL получить доступ ко всем возможностям современных графических ускорителей.

Основой этого являются так называемые расширения (*extensions*) OpenGL, позволяющие разработчикам графических ускорителей быстро добавлять новые функции, сразу делая их доступными разработчикам программного обеспечения.

Предполагается, что читатель уже знаком с библиотекой OpenGL (если нет, то [4—8] являются достаточно хорошими книгами, которые помогут вам изучить ее) и основой языка C++, поскольку все примеры к книге написаны именно на этом языке.

Дополнительную информацию, новые примеры и статьи вы можете найти на сайте автора по адресу: **www.steps3d.narod.ru**.

Практически все примеры к книге (которые вы можете найти на прилагаемом компакт-диске) компилируются и работают как под управлением операционной системы Microsoft Windows, так и под управлением операционной системы Linux. Для сборки примеров служат так называемые *make*-файлы (файл Makefile отвечает за сборку примеров для Linux, Makefile.nmake — для Windows). В некоторых случаях вам может понадобиться дополнительная настройка, которая выполняется путем изменения путей в *make*-файлах.

Обратите внимание, что в некоторых примерах задействованы новейшие возможности современных графических ускорителей, и это обусловливает довольно высокие требования, предъявляемые к графическому ускорителю и версии драйвера к нему. Поэтому некоторые примеры не будут выполняться на отдельных графических ускорителях.

Для компиляции примеров под Microsoft Windows использовался компилятор Visual C++ 6.0, под Linux — компилятор gcc.

Вся глава 1 книги посвящена расширениям OpenGL и основам работы с ними. В ней подробно разбирается, что такое расширения OpenGL, где можно получить полную документацию по всем существующим на данный момент времени расширениям, какие бывают расширения и каковы общие принципы работы с ними. Здесь также рассматриваются особенности работы с расширениями OpenGL под Windows и Linux, объяснения сопровождаются примерами, показывающими как получить полный список расширений, поддерживаемых вашим графическим ускорителем и драйвером к нему, как проверить поддержку того или иного расширения и начать его использовать. В главе 1 также описана разработанная автором библиотека libExt, предназначенная для кроссплатформенной работы с расширениями OpenGL.

Глава 2 книги посвящена простейшим расширениям OpenGL — мультитекстурированию, которое уже фактически стало частью стандарта OpenGL (и только под Windows, где все вынуждены использовать OpenGL версии 1.1, работа с мультитекстурированием идет через механизм расширений), texture_env_add (задающему еще один закон наложения текстуры), EXT_fog_coord (позволяющему создавать свои законы действия тумана, в частности, слои тумана с заданными характеристиками), EXT_secondary_color (позволяющему задавать для каждой вершины еще один цвет, что будет полезно при работе с другими расширениями) и ARB_texture_border_clamp. На

этих примерах иллюстрируются общие принципы работы с расширениями: проверка поддержки, инициализация, непосредственное использование. Показывается, как использование мультитекстурирования позволяет повысить быстродействие программы за счет комбинации сразу нескольких текстур с различными законами наложения на один проход. Приводится пример создания слоя тумана с помощью расширения EXT_fog_coord.

В главе 3 рассматривается расширение SGIS_generate_mipmap, позволяющее автоматически создавать все необходимые для пирамидального фильтрования (*mipmapping*) уровни текстуры непосредственно в GPU, а также расширения EXT_bgra и EXT_abgr, задающие новые внутренние форматы текстур, EXT_texture_filter_anisotropic, позволяющее в ряде случаев заметно улучшить качество изображения, и ARB_texture_non_power_of_two. Последнее позволяет свободно использовать текстуры, размеры которых не являются степенью двойки.

Глава 4 посвящена расширениям EXT_texture_env_combine и ARB_texture_env_combine, позволяющим задавать сложные законы наложения текстуры на объект. Ряд примеров показывают различные возможности, открываемые с помощью этих расширений.

Глава 5 посвящена новому типу текстур — кубическим текстурным картам (*cubic texture maps*). В ней подробно рассматривается отличие кубических карт от обычных (одно- и двумерных текстур), способы создания, загрузки и использования кубических карт. Рассматривается применение кубических карт для задания произвольной функции направления в трехмерном пространстве и для моделирования отражения окружающей среды поверхностью объекта (*environment mapping*).

Глава 6 вводит еще один тип текстур — трехмерные (3D) текстуры (сейчас они также вошли в стандарт OpenGL, но под Microsoft Windows работа с ними ведется через расширение EXT_texture3D). В этой главе рассматривается расширение EXT_texture3D, вводящее данный тип текстур, способы создания и использования трехмерных текстур.

В главе 7 описываются расширения ARB_point_parameters и ARB_point_sprite. На ряде примеров показывается, как можно с помощью этих расширений эффективно реализовывать различные типы так называемых систем частиц (*particle systems*), служащих для моделирования различных естественных явлений, таких как огонь, взрывы, дымовые следы и т. п.

В главе 8 вводится понятие попиксельного освещения (*per-pixel lighting*). Рассматривается простейшая модель диффузного освещения и приводятся примеры ее реализации через расширение ARB_texture_env_dot3. Здесь также вводится и подробно рассматривается понятие карт нормалей (*bump maps*), способы их задания и построения по картам высот (*height maps*). В этой главе вводится крайне важное для понимания всего попиксельного освещения

понятие касательного пространства (*tangent space*) и показывается его роль при реализации попиксельного диффузного освещения сложных объектов.

Глава 9 вводит понятие *register combiner* и рассматривает расширение NV_register_combiners, появление которого впервые позволило реализовать в реальном времени довольно сложные законы попиксельного освещения. В этой главе подробно рассматриваются организация конвейера рендеринга при использовании *register combiner*, все параметры и законы преобразования данных как в *general combiner*, так и в *final combiner*. Также показывается, как можно реализовать попиксельное диффузное и бликовое (*specular*) освещение, и приводятся многочисленные практические примеры.

Глава 10 рассматривает расширение ARB_vertex_buffer_object и на ряде примеров показывает использование вершинных и индексных буферов для повышения быстродействия программы при работе со сложными сценами.

Глава 11 посвящена *p*-буферам и рендерингу в текстуру. Использование рендеринга в текстуру позволяет легко реализовать ряд сложных визуальных эффектов, включая неровные зеркала, дрожание горячего воздуха, преломление (многочисленные примеры подобных эффектов можно увидеть в игре *Doom III*), постобработку результатов рендеринга. Здесь рассматриваются все необходимые для этого расширения и приводится программная реализация *p*-буферов в виде классов на языке C++. Рассматриваются особенности работы с *p*-буферами для платформ Windows и Linux.

В главе 12 описывается впервые появившееся для графических ускорителей GeForce 3 расширение NV_texture_shader. Вводится понятие шагов текстурирования и элементарных шейдеров. Рассматриваются все вводимые этим расширением шейдеры и примеры их практического использования. Также в этой главе рассматривается такой эффект, как EMBM (*Environment Mapped Bump Mapping*), и приводится его программная реализация с помощью расширения NV_texture_shader. Кроме этого, в главе 12 приводятся и другие примеры работы с расширением NV_texture_shader.

Вся глава 13 посвящена определению видимости в сложных сценах. В ней рассматриваются расширения HP_occlusion_test, NV_occlusion_query и ARB_occlusion_query, позволяющие динамически определять видимость объектов в сложных трехмерных сценах. В этой главе вводятся основные понятия, связанные с видимостью и отсечением объектов по видимости. Приведенные примеры показывают, как правильное использование отсечения (по пирамиде видимости и через расширение ARB_occlusion_query) позволяет заметно ускорить рендеринг сложной сцены.

Глава 14 посвящена сжатию текстур и работе со сжатыми текстурами в приложениях. В ней рассматриваются расширения ARB_texture_compression и EXT_texture_compression_s3tc и их использование. Подробно описывается алгоритм сжатия текстур S3TC и загрузка сжатых текстур из DDS-файлов.

Вся глава 15 посвящена вершинным программам (иногда называемым вершинными шейдерами) и работе с ними через расширение ARB_vertex_program. В этой главе вводится понятие вершинной программы, показывается ее место в конвейере рендеринга, рассматриваются типы регистров, с которыми работает вершинная программа, доступные ей параметры состояния OpenGL. Подробно рассматриваются способ загрузки вершинных программ и используемая система команд. Приводится инкапсуляция вершинной программы в класс языка C++, облегчающая работу с ней. Также даются многочисленные примеры вершинных программ, включая выполнение необходимых преобразований и подготовку вершинных данных для попиксельного освещения.

Глава 16 посвящена еще одному типу программ, поддерживаемых современными графическими ускорителями, — фрагментным программам. Рассматривается расширение ARB_fragment_program, место фрагментной программы в конвейере рендеринга, используемые фрагментной программой регистры, система команд. В главе приводятся многочисленные примеры фрагментных программ для реализации сложных моделей освещения и постобработки результатов рендеринга.

Если главы 15 и 16 были посвящены программам на языках низкого уровня (фактически на языке ассемблера), то в главе 17 рассматривается шейдерный язык высокого уровня — GLSL (*OpenGL Shading Language*). В этой главе излагаются основы языка GLSL, использование шейдеров, написанных на языке GLSL, в OpenGL-программах.

В главе 18 приводится инкапсуляция шейдера на языке GLSL в виде класса языка C++ и рассматривается применение шейдеров на языке GLSL для реализации сложных законов рендеринга и анимации.

В главах 17 и 18 приведено много различных шейдеров, написанных на GLSL, которые могут вам пригодиться.

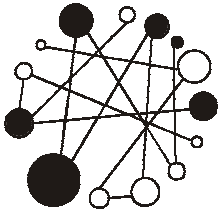
Приложение 1 посвящено основам линейной алгебры. В нем рассматриваются векторы, матрицы и их основные преобразования. Подробно описываются различные системы координат и переход из одной системы в другую. Это приложение поможет вам освежить свои знания в данной области, поскольку эти понятия очень важны при изложении ряда глав.

Приложение 2 включает основные модели освещения: диффузную и бликовую. В нем также рассматривается освещение Гуро и Фонга.

В приложении 3 приведено описание содержимого компакт-диска, прилагаемого к книге.

В списке литературы вы найдете ссылки на книги, изучение которых может оказаться вам весьма полезным, и много ссылок на различные ресурсы в Интернете.

Глава 1



Понятие расширений OpenGL, их основные типы и особенности работы с ними под Windows и Linux

Библиотека OpenGL является фактическим стандартом в мире профессиональной трехмерной графики. Она позволяет легко работать в реальном времени со сложными трехмерными объектами как на различных платформах (Windows, UNIX, Mac OS), так и на различных графических ускорителях (GeForce, Radeon, Matrox и др). Однако, как мы это сейчас наблюдаем, возможности современных графических ускорителей растут очень быстро, причем этот рост носит не только количественный (быстродействие), но и качественный (новые возможности) характер.

К тому же эти новые функции зачастую реализуются совершенно по-разному для различных моделей и производителей графических ускорителей, что создает серьезные проблемы разработчикам программного обеспечения, желающим быстро получить доступ ко всем этим возможностям.

Таким образом возникает ситуация, когда, с одной стороны, хочется, чтобы новые аппаратные возможности как можно быстрее стали доступны разработчикам ПО, а с другой — вносить серьезные изменения в сам OpenGL каждый раз, когда появляется что-то новое, было бы крайне нежелательным.

Компания Microsoft пошла именно путем внесения изменений в свою графическую библиотеку Direct3D. Периодически выпускается новая версия, измененная для поддержки новых возможностей. При этом эти изменения зачастую оказываются довольно радикальными и фактически обнуляют усилия, потраченные на изучение предыдущей версии этой библиотеки.

Иногда бывает, что некоторые возможности так и не вошли в очередную версию Direct3D в связи с какими-то соображениями компании Microsoft.

Разработчики OpenGL пошли другим путем — вместо постоянных серьезных изменений интерфейса библиотеки был введен механизм, позволяющий разработчикам графических ускорителей самим давать разработчикам про-

граммного обеспечения доступ к новым аппаратным возможностям. Этот механизм получил название расширений OpenGL (*OpenGL extensions*).

Фактически каждое расширение — это документированный набор новых функций и констант (близких к стилю, принятому в OpenGL) имеющий свое уникальное имя. За реализацию возможностей, предоставляемых тем или иным расширением, отвечает драйвер ICD (*Installable Client Driver*), который обычно пишется фирмой-разработчиком соответствующего графического ускорителя.

Это позволяет приложению получить полный список имен поддерживаемых расширений и адреса функций, вводимых тем или иным расширением (рис. 1.1).

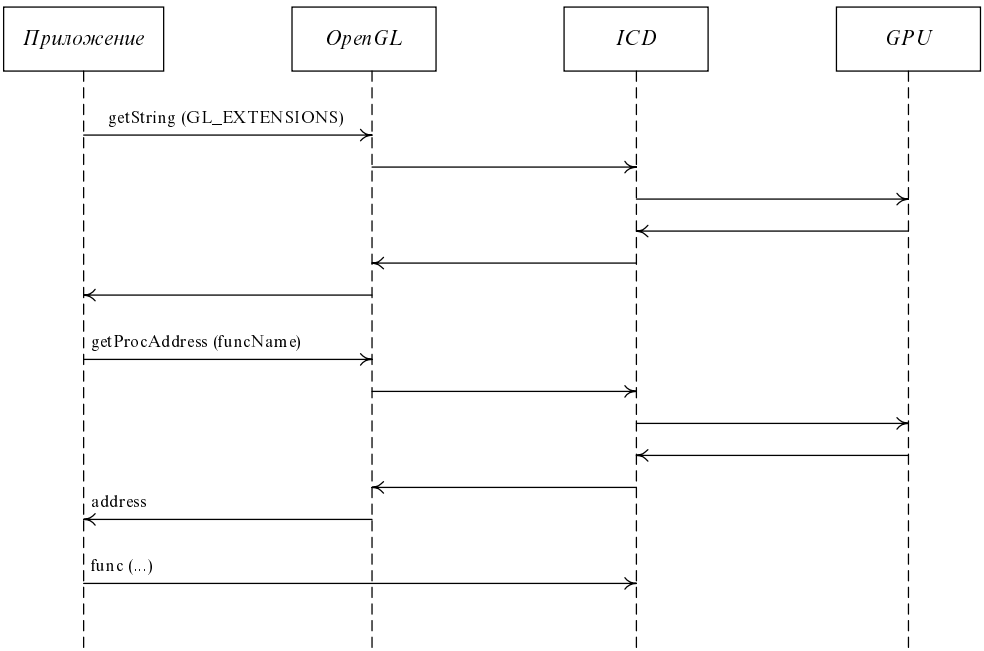


Рис. 1.1. Структура доступа к расширениям

При таком подходе с появлением какой-либо новой возможности достаточно просто документировать ее как новое расширение и добавить поддержку в драйвер. Разработчики ПО могут непосредственно во время выполнения программы проверить, поддерживается ли данное расширение, и в случае поддержки сразу же начать его использовать.

Для этого, если соответствующее расширение поддерживается, то через стандартный механизм получаются адреса новых функций. Способа получе-

ния новых констант непосредственно средствами OpenGL не предусмотрено, вместо этого они должны быть документированы в описании расширения. Таким образом, через описание соответствующего расширения разработчики получают доступ к нему (т. е. к вводимым этим расширением функциям и константам).

Во избежание возможного конфликта между расширениями от различных производителей принято довольно простое правило наименования расширений. Название каждого расширения начинается с префикса (GL, WGL, GLX), определяющего, является ли данное расширение общим (GL) или же оно соответствует какой-то определенной платформе (Windows — WGL, X Window System — GLX). Далее, после символа подчеркивания идет тип расширения, за ним — собственно название, например, GL_EXT_fog_coord. Основные типы расширений OpenGL приводятся в табл. 1.1.

Таблица 1.1. *Основные типы расширений OpenGL*

Тип	**Значение**
ARB	Расширения, введенные OpenGL Architecture Review Board
EXT	Расширения, введенные совместно различными производителями
3DFX	Расширения, введенные компанией 3DFX
APPLE	Расширения, введенные компанией Apple Inc.
ATI	Расширения, введенные компанией ATI Technoligies Inc.
HP	Расширения, введенные компанией Hewlett-Packard Co.
IBM	Расширения, введенные компанией Internation Buisness Machines Inc.
INTEL	Расширения, введенные компанией Intel Corp.
KTX	Расширения, введенные компанией Kinetix
NV	Расширения, введенные компанией NVIDIA
MESA	Расширения, введенные в реализации Mesa
SGI	Расширения, введенные компанией Silicon Graphics Inc.
SGIX	Расширения, введенные компанией Silicon Graphics Inc.
SGIS	Расширения, введенные компанией Silicon Graphics Inc.
SUN	Расширения, введенные компанией Sun Microsystems
WIN	Расширения, введенные компанией Microsoft Corp.

Подобная схема наименования расширений позволяет избежать возможных конфликтов имен между различными производителями.

Поскольку часто оказывалось, что одна и та же функциональность у разных производителей содержалась в различных расширениях, велась определенная работа по унификации расширений. Так, все расширения EXT являются результатом совместной работы различных производителей. Ряд расширений стандартизировались OpenGL Architecture Review Board и имеют тип ARB. Существует постоянно пополняемый список всех расширений и документации к ним, доступный в Интернете по адресу **http://oss.sgi.com/projects/ogl-sample/registry**.

Кроме списка расширений и их описаний поддерживаются также списки заголовочных файлов, содержащих описания расширений. Список общих (GL) расширений содержится в файле glext.h, полный список всех расширений для платформы Windows — в файле wglext.h, список расширений для X Window System — glxext.h. Все эти файлы вы можете найти на прилагаемом к книге компакт-диске, а самые последние версии — в Интернете по адресу: **http://oss.sgi.com/projects/ogl-sample/registry/**.

Для того чтобы во время выполнения программы получить список всех доступных расширений, используется функция `glGetString` с параметром `GL_EXTENSIONS`. Она возвращает список всех доступных программе расширений в виде строки имен, разделенных пробелами. Далее приводится пример подобной строки для графического ускорителя GeForce 2 MX.

```
GL_ARB_imaging GL_ARB_multitexture GL_ARB_point_parameters
GL_ARB_point_sprite GL_ARB_shader_objects GL_ARB_shading_language_100
GL_ARB_texture_compression GL_ARB_texture_cube_map GL_ARB_texture_env_add
GL_ARB_texture_env_combine GL_ARB_texture_env_dot3
GL_ARB_texture_mirrored_repeat GL_ARB_transpose_matrix
GL_ARB_vertex_buffer_object GL_ARB_vertex_program GL_ARB_vertex_shader
GL_ARB_window_pos GL_S3_s3tc GL_EXT_texture_env_add GL_EXT_abgr
GL_EXT_bgra GL_EXT_blend_color GL_EXT_blend_minmax GL_EXT_blend_subtract
GL_EXT_clip_volume_hint GL_EXT_compiled_vertex_array GL_EXT_Cg_shader
GL_EXT_draw_range_elements GL_EXT_fog_coord GL_EXT_multi_draw_arrays
GL_EXT_packed_pixels GL_EXT_paletted_texture GL_EXT_pixel_buffer_object
GL_EXT_point_parameters GL_EXT_rescale_normal GL_EXT_secondary_color
GL_EXT_separate_specular_color GL_EXT_shared_texture_palette
GL_EXT_stencil_wrap GL_EXT_texture_compression_s3tc
GL_EXT_texture_cube_map GL_EXT_texture_edge_clamp
GL_EXT_texture_env_combine GL_EXT_texture_env_dot3
GL_EXT_texture_filter_anisotropic GL_EXT_texture_lod
GL_EXT_texture_lod_bias GL_EXT_texture_object GL_EXT_vertex_array
GL_IBM_rasterpos_clip GL_IBM_texture_mirrored_repeat GL_KTX_buffer_region
GL_NV_blend_square GL_NV_fence GL_NV_fog_distance
GL_NV_light_max_exponent GL_NV_packed_depth_stencil
GL_NV_pixel_data_range GL_NV_point_sprite GL_NV_register_combiners
GL_NV_texgen_reflection GL_NV_texture_env_combine4
GL_NV_texture_rectangle GL_NV_vertex_array_range
GL_NV_vertex_array_range2 GL_NV_vertex_program GL_NV_vertex_program1_1
GL_SGIS_generate_mipmap GL_SGIS_multitexture GL_SGIS_texture_lod
GL_SUN_slice_accum GL_WIN_swap_hint WGL_EXT_swap_control
```

Обратите внимание на внушительный размер списка расширений для этого довольно старого графического ускорителя. Аналогичный список для ускорителя серии GeForceFX занял бы несколько страниц.

Заметьте также, что для получения этой строки необходимо сперва проинициализировать OpenGL, иначе функция `glGetString` возвращает значение `NULL`.

Листинг 1.1 содержит простейшую программу, печатающую полный список всех доступных расширений. При этом для простоты инициализации OpenGL используется библиотека GLUT, хотя никакого рисования здесь не производится.

Листинг 1.1. Печать списка всех поддерживаемых GL-расширений

```
#ifdef _WIN32
     #include     <windows.h>
#endif
#include     <GL/gl.h>
#include     "glut.h"
#include     "../glext.h"
#include     <stdio.h>
#include     <ctype.h>
void     printExtList ( const char * extension )
{
     char     name [1024];
     int      i, j;
     printf ( "Supported extensions:\n" );
     for ( i = 0, j = 0; extension [i] != '\0'; i++ )
          if ( !isspace ( extension [i] ) )
               name [j++] = extension [i];
          else
          {
               name [j] = '\0';e
               printf ( "\t%s\n", name );
               j = 0;
          }
     if ( j > 0 )
     {
          name [j] = '\0';
```

```
          printf ( "\t%s\n", name );
     }
}
int main ( int argc, char * argv [] )
{
     glutInit            ( &argc, argv );
     glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB |
                           GLUT_DEPTH );
     glutInitWindowSize  ( 400, 400 );
     int     win = glutCreateWindow ( "OpenGL example 1" );
     const char * vendor    = (const char *)
              glGetString ( GL_VENDOR     );
     const char * renderer  = (const char *)
              glGetString ( GL_RENDERER   );
     const char * version   = (const char *)
              glGetString ( GL_VERSION    );
     const char * extension = (const char *)
              glGetString ( GL_EXTENSIONS );
     printf ( "Vendor:   %s\nRenderer: %s\nVersion:  %s\n",
                    vendor, renderer, version );
     printExtList ( extension );
     return 0;
}
```

Теперь для проверки того, поддерживается ли данное расширение или нет, достаточно просто получить строку со списком расширений и посмотреть, содержится ли в ней имя интересующего расширения.

Обратите внимание, что простой проверки только с помощью функции `strstr` недостаточно, поскольку надо убедиться, что было найдено имя именно данного расширения, а не начало названия какого-либо другого расширения. Для этого достаточно проверить возвращенное функцией `strstr` значение на наличие пробела или `\0` в том месте, где должно заканчиваться название интересующего нас расширения. Далее приводится исходный текст функции, выполняющей такую проверку.

```
bool     isExtensionSupported ( const char * ext )
{
    const char * extensions = (const char *)
                     glGetString ( GL_EXTENSIONS );
```

```
    const char * start      = extensions;
    const char * ptr;
    while ( ( ptr = strstr ( start, ext ) ) != NULL )
    {
                                        // we've found, ensure
                                      // name is exactly ext
        const char * end = ptr + strlen ( ext );
        if ( isspace ( *end ) || *end == '\0' )
            return true;
        start = end;
    }
    return false;
}
```

Вместо `isExtensionSupported` при работе с библиотеками GLUT можно воспользоваться функцией

```
int glutExtensionSupported (char * extension);
```

Она возвращает ненулевое значение в случае, когда расширение с заданным именем поддерживается.

Обратите, однако, внимание, что таким образом проверяется поддержка только расширений типа GL, для проверки расширений WGL и GLX следует использовать другой способ, который будет рассмотрен несколько позже.

Следующим шагом, после проверки того, поддерживается ли данное расширение вашим графическим ускорителем и драйвером к нему (на самом деле поддержка расширений зависит не только от самого графического ускорителя, но и от установленного драйвера, поэтому всегда пользуйтесь последней версией драйвера для своего графического ускорителя), является получение нужных для работы адресов вводимых данным расширением функций.

Для каждой вводимой расширением функции в соответствующем заголовочном файле (glext.h, wglext.h, glxext.h) содержится определение типа указателя на данную функцию.

Так, для функции `glFogCoordEXT`, вводимой расширением GL_EXT_fog_coord, определяется следующий тип:

```
typedef void (APIENTRY * PFNGLFOGCOORDFEXTPROC)
       (GLfloat coord);
```

Таким образом тип `PFNGLFOGCOORDFEXTPROC` является адресом указателя на функцию `glFogCoordEXT`.

Поэтому если ввести переменную `glFogCoordEXT`, определив ее следующим образом

```
PFNGLFOGCOORDFEXTPROC            glFogCoordfEXT ;
```

и потом присвоить ей значение адреса этой функции, то обращение

```
glFogCoordfEXT ( 0.5f );
```

будет корректным обращением к функции `glFogCoordfEXT`, вводимой данным расширением.

Таким образом, для получения из программы доступа к функциям, вводимым тем или иным расширением, достаточно ввести набор переменных — указателей на эти функции и проинициализировать их соответствующими адресами. После этого можно обращаться к ним как к обычным функциям.

Рассмотрим теперь, как по имени функции получить ее адрес.

К сожалению, это сильно зависит от используемой платформы. Поэтому мы сначала рассмотрим, как это делается для платформы Microsoft Windows, а потом перейдем к платформе Linux.

Для того чтобы под Windows получить адрес определенной, вводимой каким-либо расширением, функции служит функция `wglGetProcAddress`.

```
PROC wglGetProcAddress(LPCSTR lpszProc);
```

Она по имени требуемой функции возвращает указатель на нее.

Обратите внимание, что для корректного присвоения ее соответствующему указателю следует выполнить явное приведение типа возвращаемого этой функцией адреса.

С помощью функции `wglGetProcAddress` можно получать адреса не только GL-расширений, но и расширений платформы Windows — WGL-расширений.

Приводимый далее фрагмент кода, иллюстрирует (на примере функции `glFogCoordfEXT`) правильную работу с расширением (проверка поддержки, инициализация и использование).

```
#include          <glext.h>
PFNGLFOGCOORDFEXTPROC     glFogCoordfEXT = NULL;

if ( isExtensionSupported ( "GL_EXT_fog_coord" ) )
     glFogCoordfEXT = (PFNGLFOGCOORDFEXTPROC)
               wglGetProcAddress ( "glFogCoordfEXT" );
...
glFogCoordfEXT ( 0.5f );
...
```

Важной особенностью платформы Windows является то, что полученный адрес в общем случае пригоден только для конкретного контекста рендеринга, в котором он был получен, и в другом контексте может не работать.

Рассмотрим теперь, каким образом осуществляется работа с расширением для платформы Linux.

Работа с библиотекой OpenGL под управлением операционной системы Linux определяется OpenGL ABI (*OpenGL Application Binary Interface for Linux* (OpenGL ABI)). Этот документ можно найти в Интернете по адресу **http://oss.sgi.com/projects/ogl-sample/ABI/index.html**.

Поскольку в русскоязычной литературе информация по работе с OpenGL под Linux практически отсутствует, далее рассматриваются несколько важных моментов использования OpenGL под Linux.

Для работы с OpenGL под Linux вам понадобятся библиотеки libGL и libGLU, расположенные обычно в каталоге /usr/lib.

Если вы хотите использовать библиотеку GLUT, то вам придется собрать ее из исходных файлов (или скачать ее вариант в виде RPM-файла).

Необходимы следующие заголовочные файлы — GL/gl.h, GL/glx.h, GL/glu.h и GL/glext.h.

На данный момент уже существует версия 1.2 OpenGL, поставляемая с Linux (в отличие от Windows, где все до сих пор работают с версией 1.1).

Проверка поддержки расширения осуществляется под Linux точно так же, как и под Windows, для этого пригодна уже известная вам функция `isExtensionSupported`.

Функция для получения адреса вводимых расширением функций под Linux выглядит несколько иначе:

```
void * glXGetProcAddressARB ( const GLubyte * name );
```

Обратите внимание, что аргументом этой функции является не строка (`const char *`), а цепочка байтов, хотя и содержащая имя интересующей нас функции в виде ASCIIZ-строки. Поэтому необходимо приведение типа из строки в цепочку байтов.

В отличие от Microsoft Windows адреса, возвращаемые данной функцией, доступны в любом контексте рендеринга OpenGL. Эта функция может использоваться как для получения GL, так и GLX-расширений.

Вот вариант рассмотренной нами функции `glForCoordfEXT` для платформы Linux:

```
#include          <glext.h>
PFNGLFOGCOORDFEXTPROC     glFogCoordfEXT = NULL;
if ( isExtensionSupported ( "GL_EXT_fog_coord" ) )
```

```
    glFogCoordfEXT = (PFNGLFOGCOORDFEXTPROC)
      glXGetProcAddress ((const Glubyte *)"glFogCoordfEXT");
...
glFogCoordfEXT ( 0.5f );
...
```

Работа с WGL-расширениями

Поскольку эти расширения специфичны только для Windows, то проверка их поддержки непосредственно введенной ранее функцией `isExtensionSupported` невозможна, поскольку используемая там строка не содержит расширений платформы Windows.

Поэтому нужно получить строку со всеми поддерживаемыми расширениями, специфичными для Windows. Для этого сначала следует стандартным образом проверить поддержку расширения WGL_ARB_extensions, и в случае его поддержки получить указатель на функцию `wglGetExtensionsStringARB`.

```
if ( isExtensionSupported ( "WGL_ARB_extensions" ) )
    wglGetExtensionsStringARB = (PFNWGLGETEXTENSIONSSTRINGARBPROC)
                          getProcAddress ( "wglGetExtensionsStringARB" );
```

В результате мы получаем указатель на функцию, возвращающую список всех WGL-расширений для заданного контекста устройства (*device context*) в том же формате, в котором возвращается список всех платформонезависимых расширений:

```
const char * exts = wglGetExtensionsStringARB ( wglGetCurrentDC () );
```

Тогда для проверки какого-либо из WGL-расширений достаточно установить, встречается ли название расширения в полученной строке. Если да, то можно с помощью функции `wglGetProcAddress` получить адреса всех вводимых этим расширением функций.

Работа с GLX-расширениями

Под Linux также существует функция, возвращающая список всех поддерживаемых GLX-расширений. Однако она не является расширением и непосредственно доступна пользователям. Вот пример ее описания:

```
const char * glXQueryExtensionsString ( Display * dpy, int screen );
```

После того как при помощи этой функции был получен список всех GLX-расширений, достаточно рассмотренным ранее способом проверить наличие в этой строке названия расширения. В случае если соответствующее расши-

рение было найдено, то при помощи функции `glXGetProcAddress` можно получить адреса вводимых этим расширением функций.

Слегка модифицированный код функции `isExtensionSupported`, обеспечивающей проверку поддержки всех расширений, приводится в листинге 1.2.

Листинг 1.2. Модифицированный вариант проверки поддержки расширений

```
static     bool     isExtensionSupported ( const char * ext,
                                     const char * extList )
{
     const char * start = extList;
     const char * ptr;

     while ( ( ptr = strstr ( start, ext ) ) != NULL )
     {
                         // we've found, ensure name
                                  // is exactly ext
          const char * end = ptr + strlen ( ext );
          if ( isspace ( *end ) || *end == '\0' )
               return true;
          start = end;
     }
     return false;
}
bool     isExtensionSupported ( const char * ext )
{
    const char * extensions = (const char *) glGetString (
                                                  GL_EXTENSIONS );
    if ( isExtensionSupported ( ext, extensions ) )
     return true;
#ifdef     _WIN32                    // check Windoze extensions
    if ( wglGetExtensionsStringARB == NULL )
    return false;
    return isExtensionSupported ( ext,
             wglGetExtensionsStringARB ( wglGetCurrentDC () ) );
#else                          // check GLX extensions
    Display * display = glXGetCurrentDisplay ();
    int           screen  = DefaultScreen        ( display );
```

```
    return isExtensionSupported ( ext,
                        glXQueryExtensionsString ( display, screen ) );
#endif
}
```

Для удобства дальнейшей работы с расширениями все необходимые операции можно записать в небольшую библиотеку, причем желательно инкапсулировать всю платформозависимую часть.

Для формирования кросс-платформенного кода изменяющуюся часть (получение адреса функции) лучше вынести в отдельную функцию:

```
static void  * getProcAddress ( const char * name )
{
#ifdef     _WIN32
    return wglGetProcAddress ( name );
#else
    return (void *)glXGetProcAddressARB (
               (const GLubyte *)name );
#endif
}
```

Обратите внимание на использование условной компиляции — символ `_WIN32` служит для определения того, происходит ли компиляция под платформой Windows или нет (здесь мы будем считать, что если этот символ не определен, то компиляция происходит под Linux).

На прилагаемом к книге компакт-диске содержится написанная автором библиотека для работы с расширениями libExt. Она не претендует на полноту и поэтому поддерживает лишь часть из имеющихся расширений. Однако все рассматриваемые в этой книге расширения ею поддерживаются, и она является кроссплатформенной (успешно компилируется и под Windows, и под Linux, и способ работы с ней не зависит от платформы).

В листинге 1.3 приводится фрагмент заголовочного файла libExt.h этой библиотеки, содержащий описание вводимых этим расширением функций (функции, вводимые различными расширениями, здесь намеренно не приводятся из-за их большого количества).

Листинг 1.3. Файл libExt.h

```
#ifdef     _WIN32
    #include     <windows.h>
```

```
#else
    #define     GLX_GLXEXT_LEGACY
#endif
#include     <GL/gl.h>
#include     <GL/glu.h>
#include     "../glext.h"
#ifdef     _WIN32
     #include     "../wglext.h"
#else
    #include     <GL/glx.h>
    #include     <GL/glxext.h>
 #endif
bool  isExtensionSupported  ( const char * ext );
void  initExtensions        ();
void  printfInfo            ();
void  assertExtensionsSupported  ( const char * extList );
```

Для удобства работы этот файл включает необходимые заголовочные файлы и для платформы Microsoft Windows файл windows.h, без которого многие примеры под Windows просто не компилируются. Данный файл определяет также следующие функции — `isExtensionSupported`, `assertExtensionsSupported`, `initExtensions` и `printfInfo`.

Функция `isExtensionSupported` уже рассматривалась нами, она просто проверяет, поддерживается ли данное расширение (при этом проверяется поддержка как общих расширений, так и расширений, специфичных для конкретной платформы).

Функция `assertExtensionsSupported` позволяет проверить поддержку сразу нескольких расширений. На вход ей передается строка с именами расширений (разделенных пробелами или запятыми). Если хотя бы одно расширение из переданного списка не поддерживается, то выдается диагностическое сообщение с именем неподдерживаемого расширения, и выполнение программы прерывается.

Функция `initExtensions` служит для инициализации указателей на вводимые расширениями функции. Библиотека libExt определяет большое число указателей на функции, вводимые всеми рассматриваемыми в этой книге расширениями, и с помощью этой функции в указатели заносятся адреса вводимых расширениями функций.

Функция `printfInfo` печатает на `stdout` информацию об используемом графическом ускорителе и его производителе, а также о поддерживаемых расширениях.

Обратите внимание, что библиотека GLAUX, которую многие используют для загрузки текстур из BMP-файлов, под Linux обычно недоступна.

Поэтому на прилагаемом к книге компакт-диске вы найдете написанную автором небольшую кроссплатформенную библиотеку libTexture, позволяющую загружать текстуры из BMP-, TGA-, DDS-, JPG- и PNG-файлов. Обе эти библиотеки будут активно использоваться на протяжении всей книги.

Для сборки примеров мы будем активно использовать *make*-файлы. Листинги 1.4 и 1.5 иллюстрируют простейшие примеры для Windows и Linux, обеспечивающие сборку приложений с поддержкой библиотек libExt и libTexture.

Листинг 1.4. Пример nmake-файла для сборки примеров под Windows

```
#
# Makefile to build examples for texture_env_combine
#
!include <win32.mak>
!include <..\nmake.inc>
EXES = env-combine-1.exe env-combine-2.exe env-combine-3.exe
env-combine-4.exe env-combine-5.exe
all: $(EXES)
env-combine-1.exe: env-combine-1.obj Torus.obj version.res $(OBJS)
     $(link) env-combine-1.obj Torus.obj $(OBJS) version.res $(LLDLIBS)
env-combine-2.exe: env-combine-2.obj Torus.obj version.res $(OBJS)
     $(link) env-combine-2.obj Torus.obj $(OBJS) version.res $(LLDLIBS)
env-combine-3.exe: env-combine-3.obj Torus.obj version.res $(OBJS)
     $(link) env-combine-3.obj Torus.obj $(OBJS) version.res $(LLDLIBS)
env-combine-4.exe: env-combine-4.obj Torus.obj version.res $(OBJS)
     $(link) env-combine-4.obj Torus.obj $(OBJS) version.res $(LLDLIBS)
env-combine-5.exe: env-combine-5.obj Torus.obj version.res $(OBJS)
     $(link) env-combine-5.obj Torus.obj $(OBJS) version.res $(LLDLIBS)
clean:
     @del env-combine-1.obj env-combine-2.obj env-combine-3.obj
env-combine-4.obj env-combine-5.obj Torus.obj version.res $(EXES) $(OBJS)
> nul
!include <..\rules.win32>
```