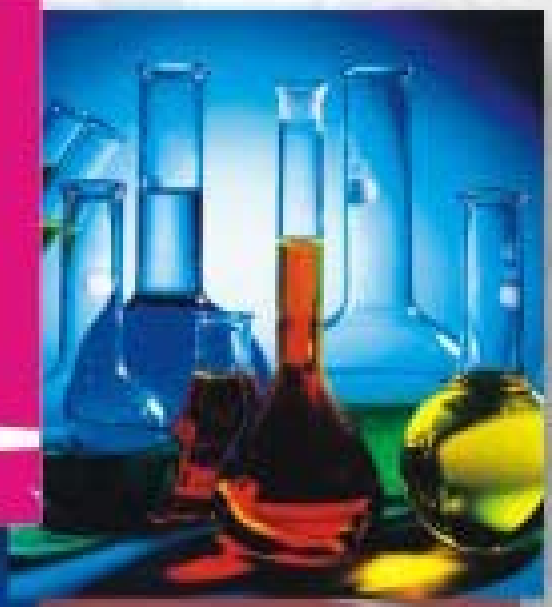
АЛЕКСЕЙ БОРЕСКОВ

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

УДК 681.3.068+800.92
ББК 32.973.26-018.1
Б82

Бобросов 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.

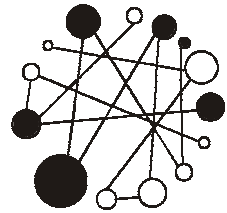
Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

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

ISBN 5-94157-614-5

Оглавление

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).

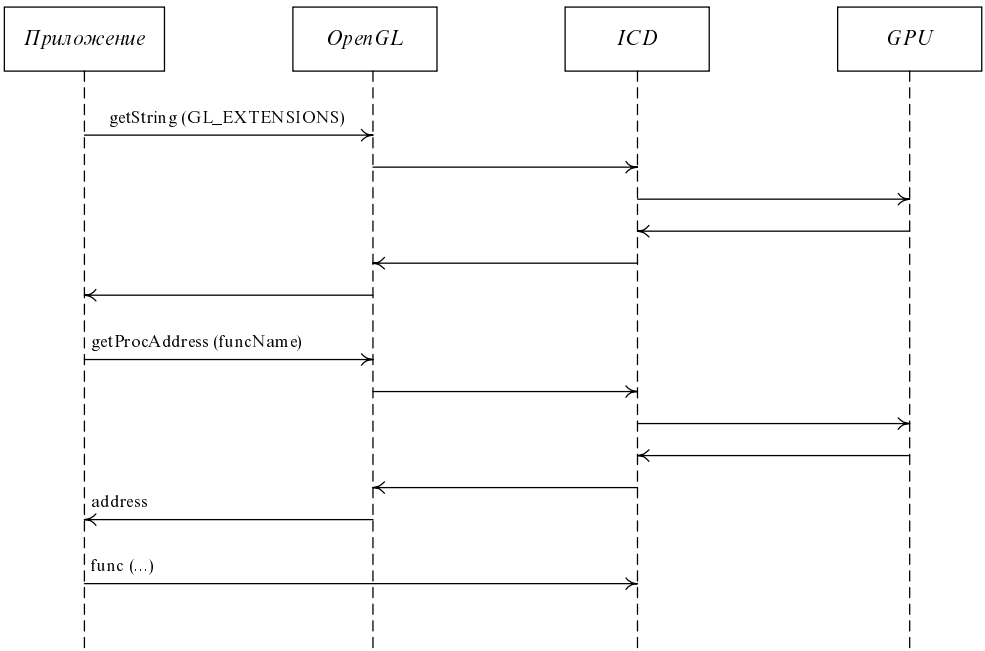


Рис. 1.1. Структура доступа к расширениям

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';
```

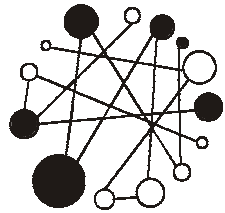
```
          printf ( "\t%s\n", name );
     }
}
int main ( int argc, char * argv [] )
{
     glutInit            ( &argc, argv );
     glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB |
                           GLUT_DEPTH );
     glutInitWindowSize  ( 400, 400 );
     int     win = glutCreateWindow ( "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 );
```

Глава 2



Простейшие расширения

Мультитекстурирование, расширение ARB_multitexture

Для получения ряда визуальных эффектов (карты освещенности, туман, микрофактурные текстуры и т. п.) часто возникает необходимость наложения на грань не одной, а сразу нескольких текстур, причем зачастую с разными законами наложения (рис. 2.1).

К сожалению, стандартный OpenGL позволяет накладывать только одну текстуру за раз (проход), что приводит к необходимости реализовывать для вывода граней несколько проходов. При этом временные затраты на рендеринг объектов возрастают в соответствующее число раз.

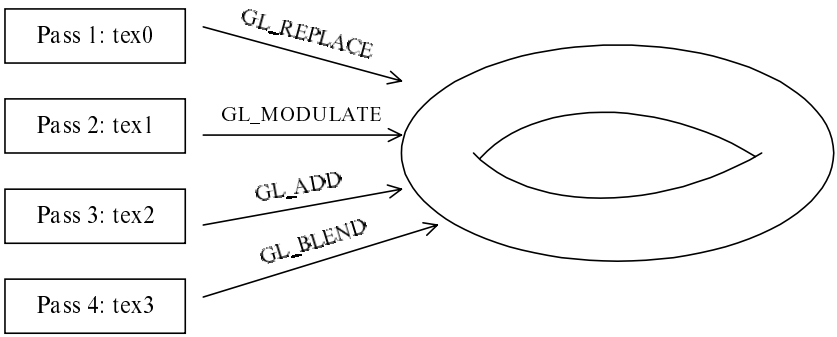


Рис. 2.1. Наложение нескольких текстур

Поэтому естественным является желание добавить возможность за один проход накладывать сразу несколько текстур. И не случайно, что первым официальным ARB-расширением, принятым в сентябре 1998 года, является именно ARB_multitexture.

Данное расширение позволяет за один проход накладывать сразу несколько текстур, при этом для каждой выводимой текстуры можно задать свои пара-

метры наложения, набор текстурных координат, матрицу преобразования текстурных координат и т. п. Общая схема мультитекстурирования представлена на рис. 2.2.

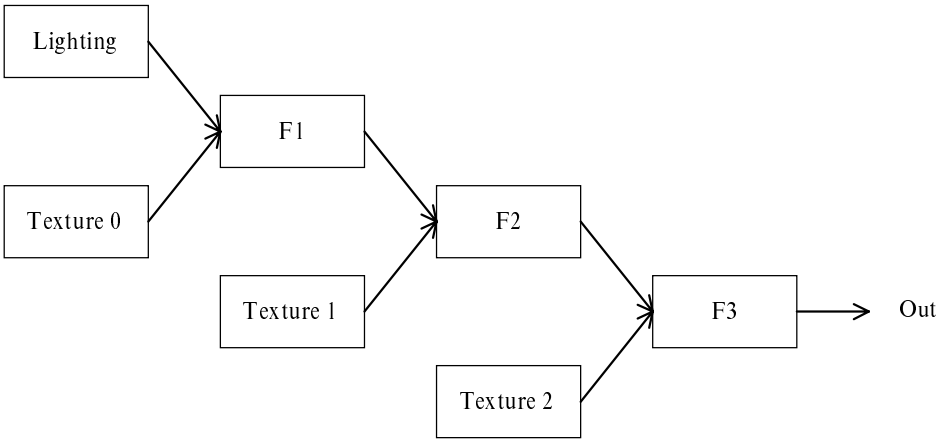


Рис. 2.2. Общая схема мультитекстурирования в OpenGL

Сейчас поддержка мультитекстурирования входит в стандарт OpenGL (начиная с версии 1.2.1), и только под Windows (так как Microsoft по-прежнему поддерживает лишь OpenGL версии 1.1) приходится работать с мультитекстурированием через соответствующее расширение.

Для проверки поддержки данного расширения можно использовать функцию `isExtensionSupported`, рассмотренную в *главе 1*.

Если

```
isExtensionSupported ( "GL_ARB_multitexture" )
```

возвращает значение `true`, то данное расширение поддерживается. Различные графические ускорители поддерживают разное число текстурных блоков (*texture units*, различных текстур, которые можно наложить за один проход). Их число для данного ускорителя (и драйвера) можно определить при помощи функции `glGetIntegerv`:

```
int     maxTextureUnits;
glGetIntegerv ( GL_MAX_TEXTURE_UNITS_ARB, &maxTextureUnits );
```

Максимальное число текстурных блоков, поддерживаемых расширением GL_ARB_multitexture, равно 32. Данное расширение вводит ряд функций и констант. Прототипы основных функций приводятся далее.

```
void glMultiTexCoord{1234}{sifd}      ( GLenum texture,T coords )
void glMultiTexCoord{1234}{sifd}v ( GLenum texture,T coords )
```

```
void glClientActiveTexture( GLenum texture );
void glActiveTexture( GLenum texture );
```

Для работы с этим расширением мы будем использовать библиотеку libExt, введенную в предыдущей главе. Для инициализации указателей на вводимые функции следует перед началом работы с этим (как и с любым другим) расширением вызвать функцию `initExtensions ()`.

После того как указатели на функции получены, можно (естественно после загрузки текстур) задать используемые текстуры.

Сначала следует при помощи функции `glActiveTextureARB` задать активный текстурный модуль:

```
glActiveTextureARB ( texture );
```

Параметр этой функции задает номер текстурного модуля и может принимать одно из следующих значений: `GL_TEXTURE0_ARB`, `GL_TEXTURE1_ARB`, `GL_TEXTURE2_ARB`, ..., `GL_TEXTURE31_ARB`.

По умолчанию (т. е. до первого вызова `glActiveTextureARB`) активным является нулевой текстурный блок (`GL_TEXTURE0_ARB`).

После задания текстурного модуля следует разрешить использование текстуры (например, при помощи команды `glEnable ( GL_TEXTURE_2D )`), задать конкретную текстуру и ее параметры (при помощи функций `glBindTexture` и `glTexParameter`), а также указать способ применения этой текстуры. В листинге 2.1 приводится простой пример задания двух текстурных модулей.

Листинг 2.1. Задание двух текстур с использованием мультитекстурирования

```
glBindTexture   ( GL_TEXTURE_2D, texture1 );
glTexParameterf ( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
glTexParameterf ( GL_TEXTURE_2D, GL_TEXTURE_MAX_FILTER, GL_LINEAR  );
glTexImage2D    ( GL_TEXTURE_2D, GL_RGBA, width1, height1, 0, format1,
                  GL_UNSIGNED_BYTE, pixels1 );
glBindTexture   ( GL_TEXTURE_2D, texture2 );
glTexParameterf ( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
glTexParameterf ( GL_TEXTURE_2D, GL_TEXTURE_MAX_FILTER, GL_LINEAR  );
glTexImage2D    ( GL_TEXTURE_2D, GL_RGBA, width2, height2, 0, format2,
                  GL_UNSIGNED_BYTE, pixels2 );
glActiveTextureARB ( GL_TEXTURE0_ARB );
glEnable           ( GL_TEXTURE_2D );
glBindTexture      ( GL_TEXTURE_2D, texture1 );
glTexEnvi          ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE );
glActiveTextureARB ( GL_TEXTURE1_ARB );
```

```
glEnable          ( GL_TEXTURE_2D );
glBindTexture     ( GL_TEXTURE_2D, texture2 );
glMatrixMode      ( GL_TEXTURE );
glScalef          ( 3, 3, 3 );
glTexEnvi         ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );
```

В приведенном листинге первая группа команд определяет и загружает две текстуры, после чего они выбираются в соответствующих текстурных блоках. Обратите внимание, что для разных текстурных блоков задаются разные законы наложения текстуры (в данном примере `GL_REPLACE` и `GL_MODULATE`) и разные текстурные матрицы (в первом случае по умолчанию — единичная матрица, а во втором — единичная матрица, умноженная на три).

При задании вершин выводимого объекта для каждой вершины каждого текстурного блока следует определить свой набор текстурных координат при помощи функции `glMultiTexCoord`:

```
glMultiTexCoord2f  ( texture, u, v );
glMultiTexCoord2fv ( texture, ptr  );
```

Первый параметр задает один из текстурных блоков (т. е. принимает одно из значений `GL_TEXTUREn_ARB`, `n = 0,...,31`). Дальше задаются текстурные координаты либо как набор координат, либо как указатель на массив координат.

Заметьте, что по аналогии с функцией `glTexCoord` существуют варианты функции `glMultiTexCoord` для разных типов входных данных и разного числа текстурных координат (от 1 до 4).

Обратите внимание, что вызов `glTexCoord` задает текстурные координаты для нулевого текстурного блока, т. е. обращение к `glTexCoord` эквивалентно вызову `glMultiTexCoord` с параметром `texture`, равным `GL_TEXTURE0_ARB`.

В следующем примере (листинг 2.2) показывается задание текстурных координат с помощью двух текстурных блоков.

Листинг 2.2. Одновременное задание текстурных координат сразу для двух текстурных блоков

```
glBegin ( GL_TRIANGLES );
  glMultiTexCoord2fv ( GL_TEXTURE0_ARB, &t0 [0] );
  glMultiTexCoord2fv ( GL_TEXTURE1_ARB, &t1 [0] );
  glVertex3fv        ( &v [0]                   );
  glMultiTexCoord2fv ( GL_TEXTURE0_ARB, &t0 [1] );
  glMultiTexCoord2fv ( GL_TEXTURE1_ARB, &t1 [1] );
  glVertex3fv        ( &v [1]                   );
```