

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ**

**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ  
РАДИОТЕХНИКИ, ЭЛЕКТРОНИКИ И АВТОМАТИКИ  
(ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)**

**Факультет: Вычислительных машин и систем**

**КОМПЬЮТЕРНАЯ ГРАФИКА**

**Методические указания по выполнению  
лабораторной работы №3**

**Тема: "Основы работы с текстурами в OpenGL"**

**Составитель: Чертков К.К.**

**Москва, 2008**

**ЦЕЛЬ РАБОТЫ:** познакомиться с основами применения текстур в OpenGL.

**ЗАДАЧА:** создать трехмерную модель заданного объекта в 3DS Max, используя соответствующие текстуры, и программу на OpenGL для управления им.

## 1. Работа с двумерными текстурами

OpenGL поддерживает работу с одномерными, двумерными, трехмерными и кубическими текстурами. Текстуры могут содержать цветное изображение, альфа-канал, значения глубины из z-буфера, закодированные координаты нормализованных векторов, а также любую другую информацию. Одно из наиболее частых применений текстур — нанесение изображений на поверхность объектов. Например, на рис. 1 показан шарик с нанесенной на его поверхность фотографией розы. Текстуры позволяют, не усложняя геометрическую модель, создавать объекты, очень близкие по внешнему виду к своим оригиналам. В данной лабораторной работе рассматриваются двумерные текстуры.

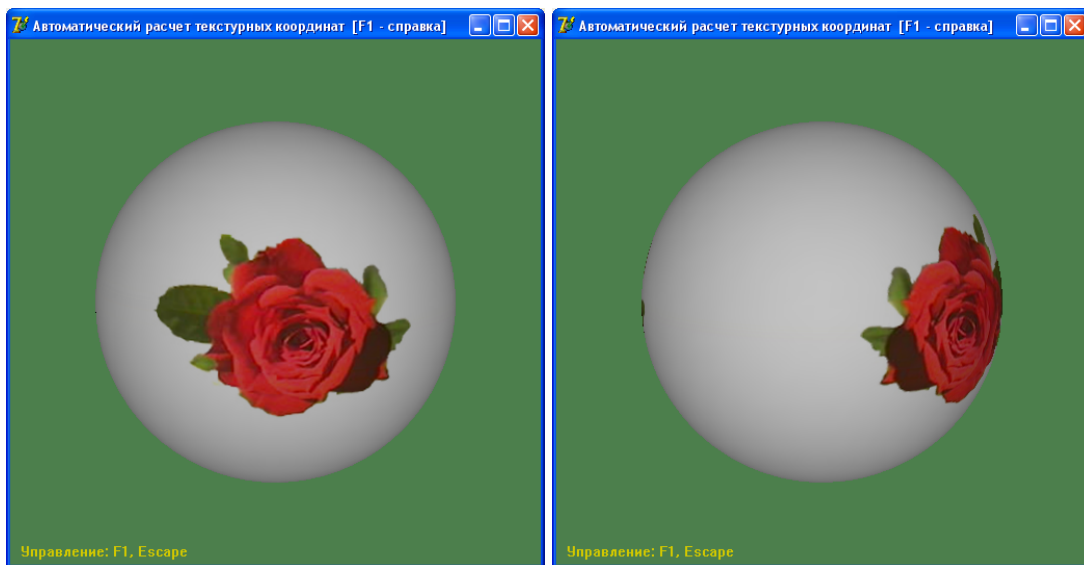


Рис. 1. Шар с текстурой

### Текстурная система координат

То, как текстура будет располагаться на поверхности объекта, определяется текстурными координатами его вершин. Каждая вершина имеет четыре однородные текстурные координаты

$(s, t, r, q)$ . Для двумерной текстуры имеют значение только первые две нормализованные координаты  $\left(\frac{s}{q}, \frac{t}{q}\right)$ .

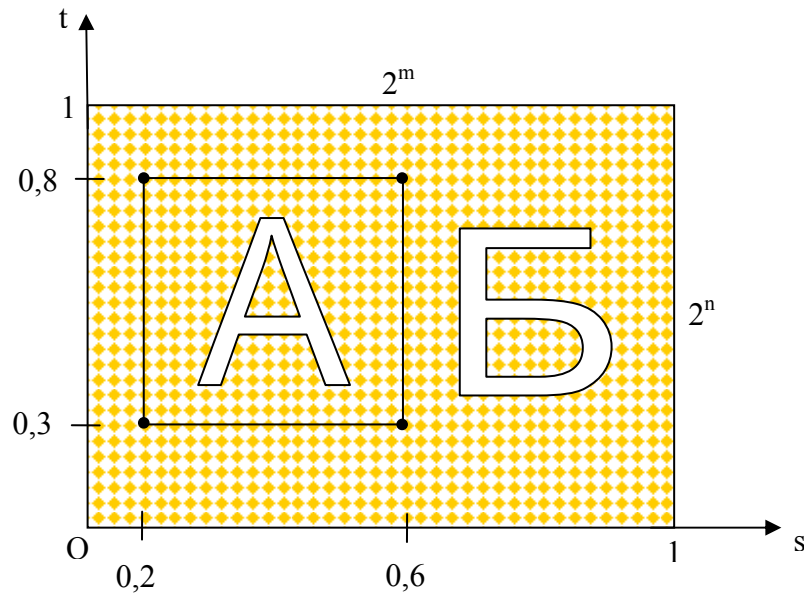


Рис. 2. Двумерная текстурная система координат

В OpenGL до версии 2.0 при отсутствии расширения *GL\_ARB\_texture\_non\_power\_of\_two* можно использовать только текстуры с размерами  $2^m \times 2^n$ . Дополнительно можно задать рамку толщиной в один тексел. Тогда размеры текстуры будут равны  $(2^m + 2) \times (2^n + 2)$ . Независимо от размеров текстуры наибольшее значение координат  $s$  и  $t$  равно 1, наименьшее – 0 (рис. 2). Если у какой-либо вершины одна или обе координаты выходят из этого диапазона, то результат будет зависеть от того, включен ли или выключен режим повторения текстуры. Если этот режим включен, то используется только дробная часть текстурных координат. Если выбран один из режимов ограничения текстурных координат, то те части поверхности, которые выходят за границы текстуры, будут всегда закрашиваться или цветом граничных точек или цветом рамки. Подробнее об этом см. в разделе «Ограничение текстурных координат».

Текстурные координаты вершины задаются одной из команд *glTexCoord2d(s, t : double)*, *glTexCoord2f(s, t : single)*, *glTexCoord2dv(v : ^double)*, *glTexCoord2d(v : ^single)* и т.п., которая должна быть вызвана до команды *glVertex\*(...)*. Ниже приведен пример рисования квадрата с такими же текстурными координатами, как у квадрата на рис. 2.

```

var
  //пространственные координаты вершин квадрата
  quad_c : array[0..3, 0..2] of double = (
    (-9, -9, 0),
    ( 9, -9, 0),
    ( 9,  9, 0),
    (-9,  9, 0)
  );

  //текстурные координаты вершин квадрата
  quad_tc : array[0..3, 0..1] of double = (
    (0.2, 0.3),
    (0.6, 0.3),
    (0.6, 0.8),
    (0.2, 0.8)
  );

  //нарисуем квадрат
  glBegin(GL_QUADS);
  for i := 0 to 3 do begin
    glTexCoord2dv(@quad_tc[i]);
    glVertex3dv(@quad_c[i]);
  end;
  glEnd;

```

Если объект рисуется командами *glDrawElements(...)*, *glDrawArrays(...)* и т.п., то дополнительно к массиву вершин и нормалей необходимо разрешить использование массива текстурных координат и определить указатель на него:

```

//разрешим использование массива пространственных и текстурных координат
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);

//определим указатели на массивы
glVertexPointer(3, GL_DOUBLE, 0, @quad_c);
glTexCoordPointer(2, GL_DOUBLE, 0, @quad_tc);

//нарисуем квадрат
glDrawArrays(GL_QUADS, 0, 4);

```

## Создание текстурных объектов

В OpenGL используется концепция текстурных объектов. Каждый такой объект имеет набор параметров (повторение / ограничение текстуры, используемый метод фильтрации и т.п.) и хранит изображение текстуры. Перед выводом изображения текстурный объект должен быть подключен к текстурному модулю. Если в системе имеется несколько текстурных модулей, то возможно наложение одновременно нескольких текстур. Текстуры модули также имеют свой набор параметров (способ наложения текстуры на поверхность и пр.).

Для создания текстуры необходимо прежде всего получить свободный идентификатор при помощи команды *glGenTextures(n : integer; textures : ^array of integer)*. Первый ее параметр указывает требуемое количество идентификаторов, второй — массив, в который они должны быть помещены. Эта команда только возвращает идентификаторы, пометив их как занятые. Создание текстурного объекта происходит при первом вызове *glBindTexture(target, texture : integer)*. Последующие вызовы с тем же идентификатором заново связывают текстурный объект с текстурным модулем. Для двумерных текстур первый параметр этой команды всегда должен быть равен константе *GL\_TEXTURE\_2D*. Как второй параметр передается идентификатор, полученный от предыдущей команды. Для удаления текстурных объектов предназначена команда *glDeleteTextures(n : integer; textures : ^array of integer)*. Первый параметр устанавливается равным числу удаляемых объектов, через второй передается указатель на массив их идентификаторов.

```
var
  texId : integer = 0;

  .....

  //получим свободный идентификатор
  glGenTextures(1, @texId);

  //создадим двумерную текстуру
  glBindTexture(GL_TEXTURE_2D, texId);

  .....

  //удалим текстуру
  glDeleteTextures(1, @texId);
```

Минимальный набор параметров, который нужно определить перед использованием текстуры — это применяемый метод фильтрации, ограничение / повторение текстурных

координат и способ комбинирования цвета текстуры с исходным цветом поверхности (функцию текстурирования).

### Выбор метода фильтрации текстуры

Способ фильтрации задается как свойство текстурного объекта при помощи команды *glTexParameter*(*target*, *pname*, *param* : *integer*). Поэтому его нужно определять для каждой текстуры отдельно. Первый параметр этой команды указывает вид текстурного объекта, связанного в данный момент с текстурным модулем, и для двумерных текстур должен быть равен *GL\_TEXTURE\_2D*. Как второй передается либо константа *GL\_TEXTURE\_MIN\_FILTER*, либо – *GL\_TEXTURE\_MAG\_FILTER*. Первая из них указывает, что определяется вид фильтрации, применяемый при уменьшении текстуры, то есть, когда она оказывается больше, чем изображаемый объект. Вторая указывает на вид фильтрации, применяемый, соответственно, при увеличении текстуры. Последний параметр может принимать либо значение *GL\_NEAREST*, либо *GL\_LINEAR*. В первом случае для окраски каждого фрагмента поверхности из текстуры будет выбираться ближайший к его центру тексел. Во втором — взвешенное среднее четырех ближайших текселов. Второй режим работает несколько медленнее, но дает более гладкое изображение.

```
//определим вид фильтрации текстуры при ее уменьшении и увеличении
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

### Ограничение текстурных координат

Способ ограничения текстурных координат *s* и *t* также определяется командой *glTexParameter*. Здесь второй параметр должен быть соответственно равен *GL\_TEXTURE\_WRAP\_S* или *GL\_TEXTURE\_WRAP\_T*. Третий параметр может принимать значения *GL\_REPEAT*, *GL\_CLAMP*, *GL\_CLAMP\_TO\_EDGE*, *GL\_MIRRORED\_REPEAT* и *GL\_CLAMP\_TO\_BORDER*.

В режиме *GL\_REPEAT* используется только дробная часть текстурных координат. Если *f* – текущее значение координаты, то будет использовано значение  $f - \lfloor f \rfloor$  (функция  $\lfloor \rfloor$  производит округление в сторону  $-\infty$ , независимо от знака аргумента).

В режиме *GL\_CLAMP* значения координат приводятся к диапазону  $[0,1]$ .

В режиме *GL\_CLAMP\_TO\_EDGE* значения координат приводятся к диапазону  $\left[\frac{1}{2N}, 1 - \frac{1}{2N}\right]$ , где  $N$  – размер изображения текстуры в направлении оси  $s$  или  $t$ .

В режиме *GL\_MIRRORED\_REPEAT* текущее значение координаты  $f$  преобразуются с помощью следующей функции

$$\text{mirror}(f) = \begin{cases} f - \lfloor f \rfloor, & \lfloor f \rfloor - \text{четное} \\ 1 - (f - \lfloor f \rfloor), & \lfloor f \rfloor - \text{нечетное} \end{cases}$$

Результат обрабатывается как в режиме *GL\_CLAMP\_TO\_EDGE*.

В режиме *GL\_CLAMP\_TO\_BORDER* для окраски точек поверхности с координатами вне диапазона  $[0,1]$  применяются тексели, принадлежащие рамке текстуры. Если рамка в изображении текстуры отсутствует, то берется постоянный цвет, заданный как цвет рамки.

Значения координат при этом приводятся к диапазону  $\left[-\frac{1}{2N}, 1 + \frac{1}{2N}\right]$ .

```
//белый цвет для рамки текстуры
var border_color : array[0..3] of single = (1.0, 1.0, 1.0, 1.0);

.....

//определим режим ограничения текстурных координат s и t
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);

//определим цвет рамки текстуры
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, @border_color);
```

Режим *GL\_CLAMP\_TO\_EDGE* доступен, начиная с версии 1.2 OpenGL, *GL\_CLAMP\_TO\_BORDER* — с версии 1.3, *GL\_MIRRORED\_REPEAT* — с версии 1.4.

### Функции текстурирования

То, как текстура будет накладываться на поверхность объекта, зависит от ее формата (*GL\_RGB*, *GL\_RGBA*, *GL\_ALPHA* и пр.) и выбранной функции комбинирования цвета текстуры

с цветом поверхности ( $GL\_REPLACE$ ,  $GL\_MODULATE$ ,  $GL\_DECAL$  и пр.). Цвет  $C_s$  и альфа-канал  $A_s$ , полученный в результате фильтрации текстуры, для разных форматов текстуры определяется по-разному. В таблице ниже приведено соответствие между исходными компонентами  $R_t, G_t, B_t, A_t$  текстуры в различных форматах и вышеуказанными значениями.

Формат текстуры	$C_s$	$A_s$
<b>GL_ALPHA</b>	$(0, 0, 0)$	$A_t$
<b>GL_RGB</b>	$(R_t, G_t, B_t)$	1
<b>GL_RGBA</b>	$(R_t, G_t, B_t)$	$A_t$

Текстуры в формате  $GL\_ALPHA$  содержат только альфа-канал  $A_t$ , задающий прозрачность. Их можно использовать при рисовании сетки, жалюзей и других предметов с множеством отверстий (см. раздел «Использование альфа-канала текстуры»). Текстуры в формате  $GL\_RGB$  хранят три цветовых компонента  $R_t, G_t, B_t$ . Значение  $A_t$  для всех текселов равно 1. Текстуры в формате  $GL\_RGBA$  содержат все четыре компонента, и прозрачность может задаваться для каждого тексела индивидуально.

В таблице ниже показано, как для трех различных функций текстурирования  $GL\_REPLACE$ ,  $GL\_MODULATE$  и  $GL\_DECAL$  происходит смешивание цвета текстуры с цветом поверхности (через  $C_f$  и  $A_f$  обозначены соответственно цвет и альфа-канал поверхности, через  $C_v$  и  $A_v$  — получаемые значения цвета и альфа-канала).

Формат текстуры	Функция текстурирования		
	<b>GL_REPLACE</b>	<b>GL_MODULATE</b>	<b>GL_DECAL</b>
<b>GL_ALPHA</b>	$C_v = C_f$ $A_v = A_s$	$C_v = C_f$ $A_v = A_f \cdot A_s$	не определено
<b>GL_RGB</b>	$C_v = C_s$ $A_v = A_f$	$C_v = C_f \cdot C_s$ $A_v = A_f$	$C_v = C_f$ $A_v = A_s$
<b>GL_RGBA</b>	$C_v = C_s$ $A_v = A_s$	$C_v = C_f \cdot C_s$ $A_v = A_f \cdot A_s$	$C_v = C_f \cdot (1 - A_s) + C_s \cdot A_s$ $A_v = A_f$



Функция текстурирования задается как свойство текстурного модуля командой *glTexEnvf(target, pname, param : integer)*. Как первый параметр должна быть указана константа *GL\_TEXTURE\_ENV*, как второй — *GL\_TEXTURE\_ENV\_MODE*, как третий — *GL\_REPLACE*, *GL\_MODULATE* или *GL\_DECAL*.

```
//цвет текстуры заменяет цвет поверхности
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
```

### Загрузка изображения текстуры

Изображение в текстурный объект можно загружать из памяти приложения или копировать из буфера кадра. Для загрузки двумерной текстуры из памяти используется функция *glTexImage2D(target, level, internalFormat, width, height, border, format, type : integer; data : pointer)*. Параметр *target* должен быть равен *GL\_TEXTURE\_2D*. Параметр *level* задает уровень детализации. Если не используется MIP-фильтрация, то он должен быть равен 0 (базовый уровень). Параметр *internalFormat* определяет формат текстуры и может принимать значения *GL\_ALPHA*, *GL\_RGB*, *GL\_RGBA* и т.д. Параметры *width* и *height* определяют ширину и высоту текстуры и должны задаваться в соответствии со сказанным в разделе «Текстурная система координат». Параметр *border* должен быть равен 1, если текстурное изображение содержит рамку, и 0 в противном случае. Параметр *format* указывает, в каком формате представлены данные загружаемого изображения. Здесь должны передаваться константы *GL\_ALPHA*, *GL\_BGR*, *GL\_BGRA* и т.п. Параметр *type* задает тип компонентов загружаемого изображения. Обычное значение — *GL\_UNSIGNED\_BYTE*. Через параметр *data* передается указатель на данные изображения. Ниже показан пример создания и загрузки изображения, состоящего из желтых и черных клеток. На рис. 3 показано, как оно выглядит.

```
const
  iw = 128; //ширина текстуры (должна быть равна 2^n)
  ih = 128; //высота текстуры (должна быть равна 2^m)

var
  //изображение текстуры
  b : array [0 .. iw - 1, 0 .. ih - 1, 1 .. 3] of byte;

  .....
```

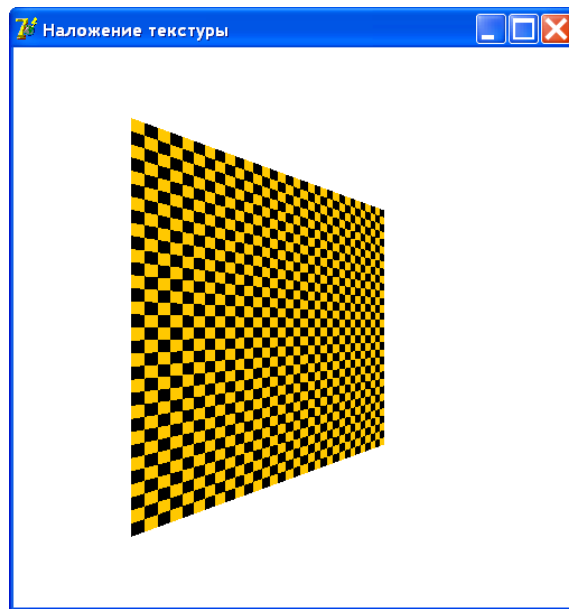
```

//рассчитаем цвета точек текстуры
for i := 0 to iw-1 do
for j := 0 to ih-1 do
  if (((i div 16) + (j div 16)) and 1) = 0 then begin
    b[i,j,1] := 0; //B
    b[i,j,2] := 0; //G
    b[i,j,3] := 0; //R
  end else begin
    b[i,j,1] := 0; //B
    b[i,j,2] := 200; //G
    b[i,j,3] := 255; //R
  end;

  .....

//загрузим изображение в текстурный объект
glTexImage2D(
  GL_TEXTURE_2D,      //двумерная текстура
  0,                  //уровень детализации
  GL_RGB,              //внутренний формат текстуры
  iw, ih,             //ширина и высота текстуры
  0,                  //ширина рамки
  GL_BGR,              //формат точки загружаемого изображения
  GL_UNSIGNED_BYTE,    //тип цветовых компонентов загружаемого
                      //изображения(1 байт без знака)
  @b);                //указатель на данные изображения

```



**Рис. 3. Сформированная текстура**  
(повторяется 4 раза по вертикали и по горизонтали)

Для загрузки текстуры из файла можно использовать следующую функцию:

```
//загружает текстуру и возвращает ее идентификатор
//в случае ошибки возвращается значение (-1)
//fn - имя файла с изображением
function LoadTexture(fn : WideString) : integer;
var
  pixels : PBytes;
  tex_id : integer;
  w, h : LongWord;
begin
  Result := -1;
  pixels := nil;

  //загрузим изображение текстуры в формате 32 бита на точку
  LoadTexImage(fn, PixelFormat32bppARGB, pixels, w, h);

  if pixels = nil then exit;

  //получим свободный идентификатор для текстурного объекта
  glGenTextures(1, @tex_id);

  //создадим текстурный объект
  glBindTexture(GL_TEXTURE_2D, tex_id);

  //определим параметры фильтрации текстуры
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

  //разрешим повторение текстуры
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

  //определим функцию текстурирования
  glTexEnvf(GL_TEXTURE_2D, GL_TEXTURE_ENV_MODE, GL_MODULATE);

  //загрузим изображение в текстурный объект
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_BGRA,
    GL_UNSIGNED_BYTE, pixels);

  //освободим память
  FreeMem(pixels);

  //вернем идентификатор текстурного объекта
  Result := tex_id;
end;
```

Используемая выше процедура *LoadTexImage* находится в файле *utils.pas*. Ниже приводится ее код.

```

//Загружает изображение из файла file_name. Преобразует его в формат
//format и возвращает указатель на его данные через параметр image_data.
//Через параметры width и height возвращаются ширина и высота изображения
//Поддерживаются форматы BMP, GIF, JPEG, PNG, TIFF.
procedure LoadTexImage(file_name : WideString; format : TPixelFormat;
                        var image_data : PBytes; var width : UINT;
                        var height : UINT);
var
    tex : TGPPixmap;
    bmd : TBitmapData;
    gdiplusStartupInput : TGdiplusStartupInput;
    gdiplusToken : UINT;
    status : TStatus;
    r : TGPPRect;

    gdi_loaded : boolean;
    i : integer;
begin
    gdi_loaded := false;
    tex := nil;
    bmd.Scan0 := nil;
    gdiplusToken := 0;
    gdiplusStartupInput.GdiplusVersion := 1;
    gdiplusStartupInput.SuppressBackgroundThread := false;
    gdiplusStartupInput.SuppressExternalCodecs := false;
    gdiplusStartupInput.DebugEventCallback := nil;

    try
        //инициализация GDI+
        status := GdiplusStartup(gdiplusToken, @gdiplusStartupInput, nil);
        if status <> Ok then raise Exception.Create('');
        gdi_loaded := true;

        //загрузим изображение текстуры
        tex := TGPPixmap.Create(file_name);
        if (tex.GetWidth() = 0) or (tex.GetHeight() = 0) then
            raise Exception.Create('');

        //получим указатель на данные изображения
        r.X := 0; r.Y := 0; r.Width := tex.GetWidth();
        r.Height := tex.GetHeight();
        status := tex.LockBits(r, ImageLockModeRead, format, bmd);
        if status <> Ok then raise Exception.Create('');

        //ширина и высота изображения
        width := r.Width; height := r.Height;

        //выделим память для копии изображения
        GetMem(image_data, abs(bmd.Stride) * r.Height);

        //если изображение хранится в порядке "сверху-вниз",
        //то его нужно поменять на обратный
        if bmd.Stride > 0 then
            for i:= 0 to r.Height - 1 do
                move(PBytes(bmd.Scan0)[bmd.Stride * (r.Height - 1 - i)],

```

```

        image_data[i * bmd.Stride], bmd.Stride)
    else
        move(PBytes(bmd.Scan0)[0], image_data[0],
            abs(bmd.Stride) * r.Height);

    //освободим указатель на данные изображения
    tex.UnlockBits(bmd); bmd.Scan0 := nil;

    //удалим изображение текстуры
    tex.Free; tex := nil;

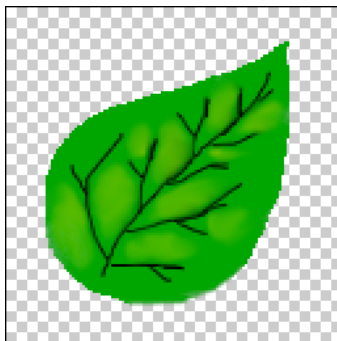
    //завершение работы с GDI+
    GdiplusShutdown(gdiplusToken); gdi_loaded := false;

except
    if image_data <> nil then begin
        FreeMem(image_data); image_data := nil;
    end;
    if bmd.Scan0 <> nil then tex.UnlockBits(bmd);
    if tex <> nil then tex.Free;
    if gdi_loaded then GdiplusShutdown(gdiplusToken);
end;
end;

```

### Использование альфа-канала текстур

Альфа-канал текстуры обычно используется для того, чтобы, не создавая сложной геометрической модели, изображать предметы с неровными краями или дырявой поверхностью. На рис. 4 показана текстура с изображением листа. Точки, принадлежащие листу, имеют значение  $A = 1$  и полностью непрозрачны. У окружающих лист точек  $A = 0$  и они не рисуются.



**Рис. 4. Текстура с прозрачным полем  
вокруг изображения листа**

Наложив эту текстуру на поверхность и запретив вывод точек с  $A = 0$ , мы получим изображение, показанное на рис. 5.

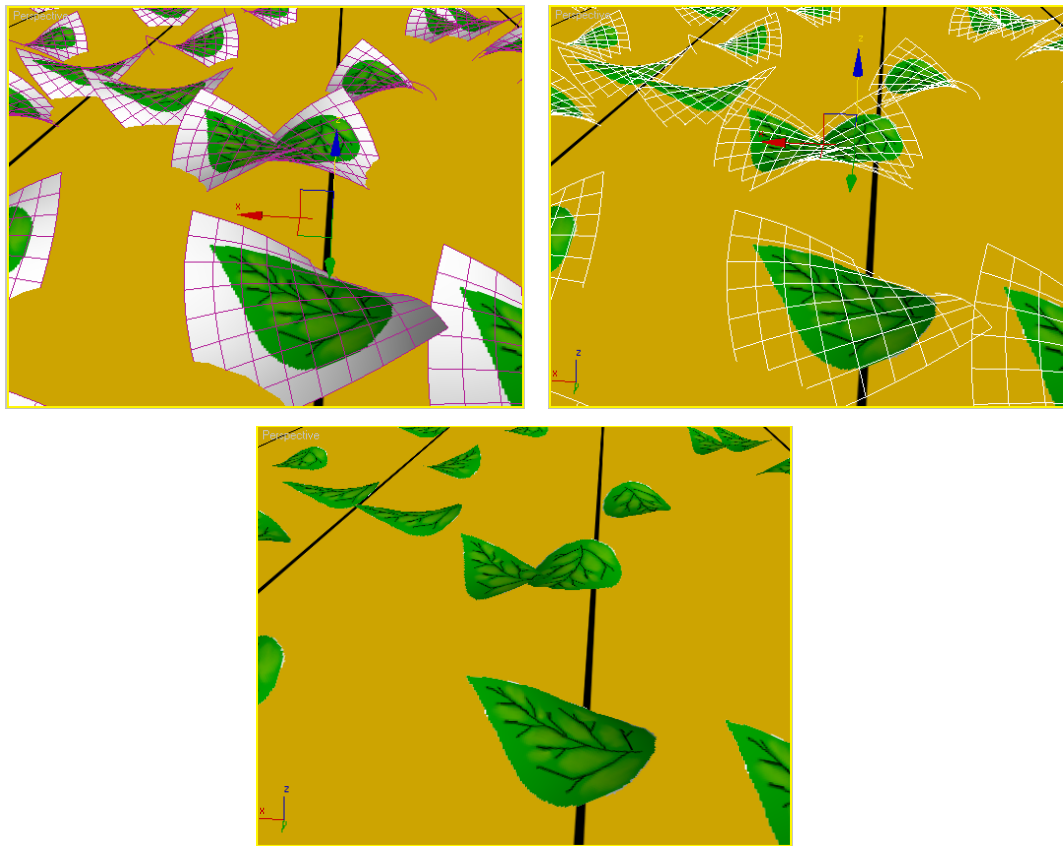


Рис. 5. Наложение текстуры с прозрачными участками

Запрет вывода точек с  $A = 0$  выполняется двумя следующими командами:

```
//включим альфа-тест для рисования листьев
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER, 0.0);
```

Первая из них включает проверку значений компоненты  $A$  выводимых точек, вторая указывает, что будут выводиться только те точки, у которых  $A > 0$ . Процедуре *glAlphaFunc(func : integer; ref : single)* в качестве первого параметра передается функция сравнения (*GL\_LESS*, *GL\_EQUAL*, *GL\_NOTEQUAL* и пр.), в качестве второго — значение, с которым будет сравниваться  $A$ .

## 2. Задание

1. Создать с помощью 3DMax модель в соответствии с выбранным вариантом задания. Модель должна включать в себя один или несколько объектов с наложенной текстурой.
2. Создать описание модели в структурах данных на одном из языков программирования.
3. Создать программу для управления данной моделью.

### *Варианты моделей*

1. Работаящий настольный вентилятор, медленно поворачивающийся вокруг вертикальной оси.
2. Катящееся по доске туда и обратно колесо. Доска должна качаться, не давая колесу скатиться.
3. Изображение компьютера на вращающемся столе.
4. Холодильник с открывающейся и закрывающейся дверцей.
5. «Рука» робота с двумя суставами.
6. Стол с выдвигающимися ящиками.
7. Избушка с трубой и открывающейся дверью и окошками.
8. Открывающийся и закрывающийся сундук.
9. Движущийся по плоскости трактор.
10. Молоток, бьющий по гвоздю.
11. Башня замка с подъемным мостом надо рвом.
12. Работаящая водяная мельница.
13. Работаящий разводной мост над каналом.
14. Гардероб с открывающимися дверцами и выдвигающимися ящиками.
15. Движущийся БТР с вращающейся башней.
16. Газовая плита с открывающейся дверцей.
17. Управляемый колесный погрузчик.
18. Движущийся самосвал с поднимающимся и опускающимся кузовом.
19. Работаящий экскаватор.
20. Поднимающийся и опускающийся лифт с открывающейся дверью.
21. Работаящий колодец с ведром.
22. Летящий парашют с пропеллером.

- 23. Летящий дирижабль с пропеллером.
- 24. Движущаяся по круглой трассе машина.
- 25. Работающая радарная установка.