OpenGL VII: Scratching the Surface of Texture Mapping

Dale Rogerson
Microsoft Developer Network Technology Group

May 12, 1995

Click to open or copy the files in the EasyTex sample application for this technical article.

Click to open or copy the files in the PicCube sample application for this technical article.

Click to open or copy the files in the GLlib DLL for this technical article.

Abstract

This article explains how to apply bitmaps to OpenGL™ surfaces to give them a realistic appearance. The bitmaps are known as *textures* and can resemble wood, marble, or any other interesting material or pattern. The process of applying or mapping a texture to a surface is known as *texture mapping*. The EasyTex and PicCube sample applications demonstrate the concepts discussed in this article.

Stamps and Boxes

One Christmas, back when I was not only mentally but legally a kid, my parents gave me a Guns of Navarone play set, consisting of a 3-foot plastic mountain with large yellow artillery guns. A World War II play set is an odd gift for Christmas, but I was happy with the present. I was also pleased to see that the play set came in a large box. I loved playing in the large box—it became a tank, a plane, or a fox hole at the whim of my imagination. However, that's not all I did with the large box. I also texture-mapped it.

Back then, music companies had subscription music services (as they do now), which offered 10 records for a penny. (Of course, after your first order, you had to buy several records at the inflated club prices.) The club catalog included stamps representing each album cover. You would order the records you wanted by sticking the stamps on the order form. I didn't have a penny, so I couldn't get the 10 records. So, instead of putting the stamps on the order form, I put them on my large box. Eventually, the entire box became textured with these stamps, as illustrated in Figure 1.

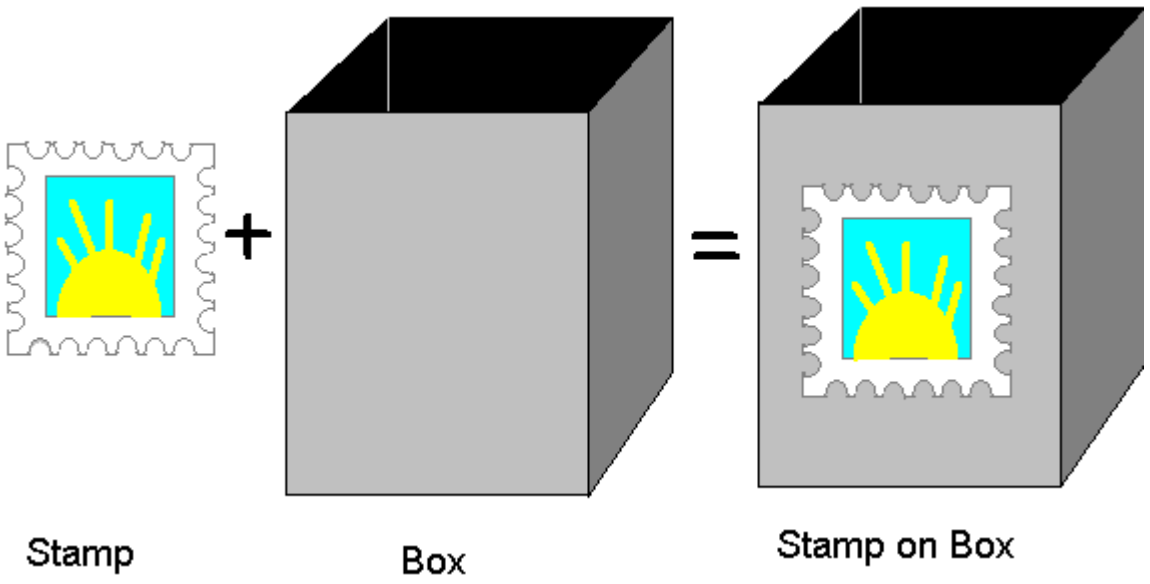

Figure 1. Texturing a box with stamps

OpenGL™ contains a feature called *texture mapping*, which is similar to sticking stamps on a box. Texture mapping is the application (or mapping) of an image (or texture) to the surface of a polygon or other graphics primitive, as illustrated in Figure 2.

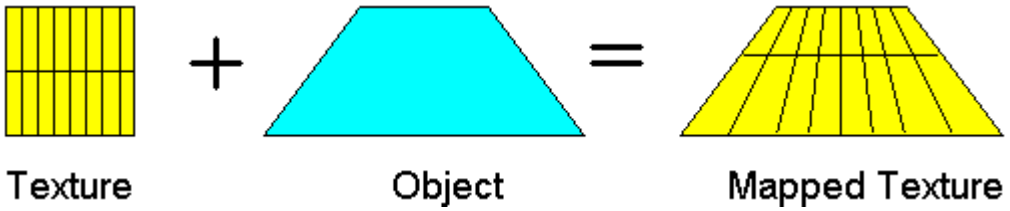

Figure 2. Mapping a texture to a polygon

Let's say you want an OpenGL representation of your living room. If your living room has a wood table, you could scan a piece of wood to use as a texture and map the wood texture to the polygons that make up your table. You could also apply textures to the wallpaper and carpet. For example, you could use a scanned image of your mother to texture-map a picture frame on the wall. The Doom video game almost exclusively uses texture mapping (but not through OpenGL) to create the feeling of being in scary places.

This article is only going to scratch the surface of using OpenGL texture mapping. We will cover the following topics:

- What the EasyTex sample application demonstrates
- What the PicCube sample application demonstrates
- How to translate a Windows ® device-independent bitmap (DIB) into an OpenGL texture
- How to enable texture mapping
- How to specify texture coordinates when mapping a texture to a surface

Texture mapping is more of an art than a science. Texture-mapping complicated shapes requires lots of experimentation, experience, and patience. See Chapter 9 in the *OpenGL Programming Guide* (also called the Red Book; see the bibliography at the end of this article) for additional information on texture mapping.

The EasyTex Sample Application

The EasyTex sample application demonstrates the concepts of texture mapping presented in this article. EasyTex is based on the EasyGL sample application, which has been the basis for all my articles in this OpenGL article series.

EasyTex adds support for texture-mapping the box displayed in the window's client area. Figure 3 is a simulated screen shot of EasyTex. (As in previous articles, I didn't use a real screen shot because the Microsoft ® Development Library software doesn't display 256-color images.)

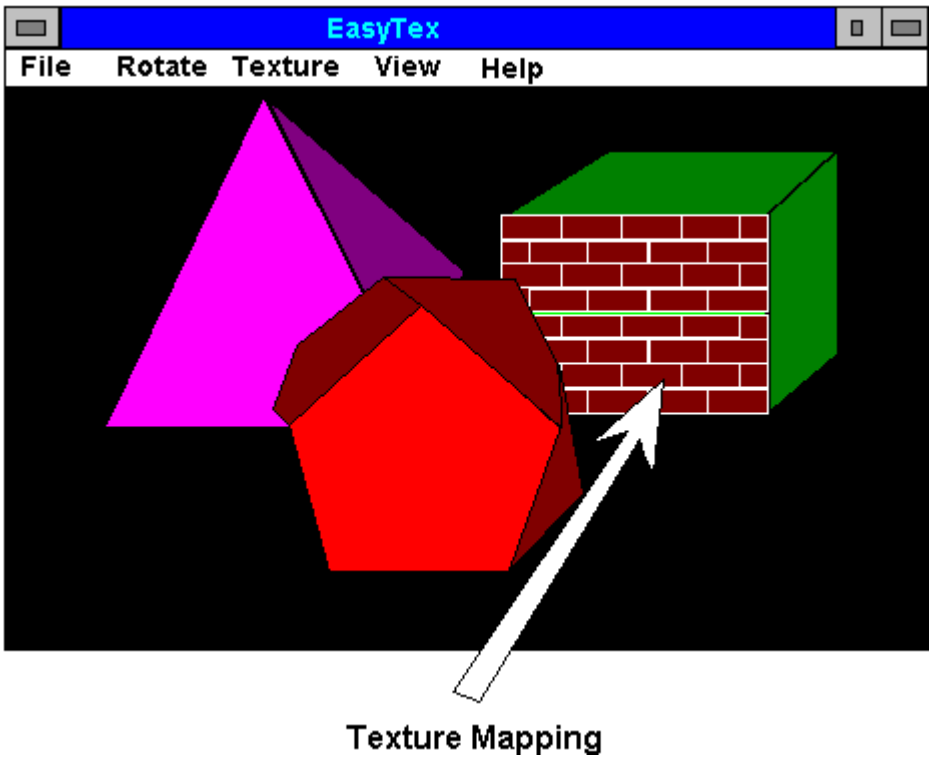

Figure 3. Simulated screen shot of EasyTex

Initially, the box is textured with three wallpaper bitmaps (HONEY.BMP, BRICK.BMP, and EGYPT.BMP) and the Microsoft Developer Network logo. Bitmaps that are smaller than 128 pixels in width or height are repeated on the surface of the box. Bitmaps that are larger than 128 pixels in width or height are not repeated. The Microsoft Developer Network logo is not repeated.

The EasyTex Texture menu (Figure 4) provides control over how the box is texture-mapped.

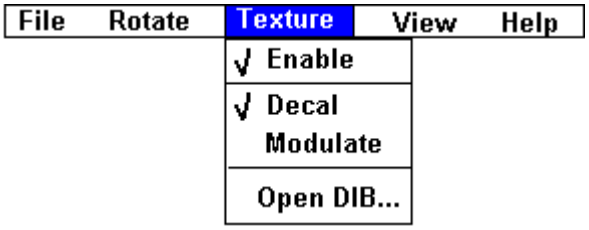

Figure 4. EasyTex Texture menu

The Texture menu contains the following options:

- The Enable command turns texture mapping on and off.
- The Decal command determines whether the texture will be applied like a decal, covering the underlying surface. When this option is enabled, the colors of the texture cover the colors of the object.
- The Modulate command determines whether the texture will modify the underlying surface. When this option is enabled, the colors of the texture modify the colors of the object.

  The "Choosing the Texturing Mode" section later in this article explains the differences between the Decal and Modulate options in more detail.
- The Open DIB command allows the user to load a DIB or bitmap (through a File Open dialog box) and use it as a texture map. The DIB replaces the face of the cube, as illustrated earlier in Figure 3. You can load any Windows DIB for use as a texture. Try images with at least 256 colors for fun results.

PicCube Sample Screen Saver

I wrote another sample application, PicCube, to demonstrate texture mapping. PicCube is a Windows screen saver that displays a rotating, floating cube with six texture-mapped sides. You can pick a different bitmap to use as a texture for each side of the cube.

The screen saver part of PicCube comes from Nigel Thompson's article, "Creating 32-Bit Screen Savers with Visual C++ and MFC" in the MSDN Library.

To use PicCube as a screen saver, copy the piccube.scr file to your Windows NT™ \SYSTEM32 directory. Choose Desktop from the Control Panel to select the file as the screen saver, choose Setup, and use the Setup dialog box to pick the bitmaps for texturing the sides of the cube.

PicCube uses a statically linked version of GLlib, which contains a class library for OpenGL. Make sure that when you build PicCube, GLlib, and Nigel Thompson's animation libraries, you link statically to the Microsoft Foundation Class Library (MFC) instead of linking to the shared library. I decided that PicCube should use statically linked versions of GLlib and MFC to facilitate installation.

Translating DIBs into Textures

The first step for texture-mapping an object is getting the texture. You can generate textures at run time, but in most cases you will use DIBs stored as files or resources. We covered loading and translating Windows DIBs into OpenGL images in the technical article "OpenGL V: Translating Windows DIBs" in the MSDN Library.

Once the Windows DIB has been loaded and translated into an OpenGL image, we have to modify the image once more before we use it as a texture. The OpenGL **glTexImage2D** command defines the image to be used for texture mapping. This command requires both the width and the height of the image to be a power of two, so we must resize the image if its dimensions are different.

I added a new member function to **CGLImage** to handle resizing images:
CGLImage::TexMapScalePow2 resizes the image using **gluScaleImage**. The **StretchDIBits** function in the Windows graphics device interface (GDI) is faster than the **gluScaleImage** utility function in OpenGL. However, **gluScaleImage** generates higher-quality images.

The code for **CGLImage::TexMapScalePow2** is shown below. I based this code on the code used in the OpenGL screen savers, which are included with Windows NT version 3.5.

```
//
// TexMapScalePow2 - Scale image to power of 2 in height and width.
//
BOOL CGLImage::TexMapScalePow2(CGL* pGL)
{
   GLint glMaxTexDim ;
   double xPow2, yPow2;
   int ixPow2, iyPow2;
   int xSize2, ySize2;

   glGetIntegerv(GL_MAX_TEXTURE_SIZE, &glMaxTexDim);
   glMaxTexDim = min(128, glMaxTexDim);

   if (m_iWidth <= glMaxTexDim)
      xPow2 = log((double)m_iWidth) / log(2.0);
   else
      xPow2 = log((double)glMaxTexDim) / log(2.0);

   if (m_iHeight <= glMaxTexDim)
      yPow2 = log((double)m_iHeight) / log(2.0);
   else
      yPow2 = log((double)glMaxTexDim) / log(2.0);

   ixPow2 = (int)xPow2;
   iyPow2 = (int)yPow2;

   if (xPow2 != (double)ixPow2)
      ixPow2++;
   if (yPow2 != (double)iyPow2)
      iyPow2++;

   xSize2 = 1 << ixPow2;
```

```
    ySize2 = 1 << iyPow2;

    BYTE *pData = (BYTE*)malloc(xSize2 * ySize2 * 3 * sizeof(BYTE));
    if (!pData) return FALSE;

    pGL->MakeCurrent() ;
    gluScaleImage(GL_RGB, m_iWidth, m_iHeight,
                  GL_UNSIGNED_BYTE, m_pBits,
                  xSize2, ySize2, GL_UNSIGNED_BYTE,
                  pData);

    free(m_pBits);
    m_pBits = pData;
    m_iWidth = xSize2 ;
    m_iHeight = ySize2 ;

    return TRUE ;
}
```

I won't go into much detail on how **TexMapScalePow2** works, but I do want to explain the following two lines in the code:

```
glGetIntegerv(GL_MAX_TEXTURE_SIZE, &glMaxTexDim);
glMaxTexDim = min(128, glMaxTexDim);
```

The first line gets the maximum allowable size for an image to be used as an OpenGL texture map. On my system, this value happens to be 1024. This size may vary, depending on your system, especially if you have an OpenGL-accelerated video card.

The second line results in **TexMapScalePow2** generating an image that is no larger than 128 x 128 pixels. For the generic OpenGL implementation on Windows NT, the smaller the texture, the faster OpenGL works. Graphics cards accelerated for OpenGL may implement texture mapping in hardware, resulting in little performance degradation when using large bitmaps. For EasyTex, I picked 128 pixels because it provided decent speed with reasonable detail. A more generic version of **TexMapScalePow2** would take this value as a parameter. The OpenGL screen savers also limit the size of textures for the same reason.

Initializing Texture Mapping

Before we can see our textures mapped to the screen, we have to initialize texture mapping. In EasyTex, these initializations are called from the **CScene::OnInit** member function.

Enabling Texture Mapping

In EasyTex, I used the **glEnable** function to enable texture mapping:

```
glEnable(GL_TEXTURE_2D);
```

Setting Pixel Alignment

The pixel alignment must be set. The image pixel data stored by **CGLImage** is aligned on a byte boundary. The following command tells OpenGL what the pixel alignment is:

```
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
```

See the article "OpenGL V: Translating Windows DIBs" in the MSDN Library for more information on pixel alignment.

Specifying Repeating Textures

If you want to repeat the texture over the surface, use the GL_REPEAT option:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

If you set the correct texture coordinates, OpenGL will repeat the texture over the surface instead of displaying a single copy of the texture.

Another option is GL_CLAMP, which displays one copy of the texture on the surface. EasyTex and PicCube use GL_REPEAT, and we'll continue our discussion of repeating textures in the next section on texture coordinates. For information on GL_CLAMP, see the "Repeating and Clamping Textures" section in Chapter 9 of the Red Book.

Picking the Filtering Method

When OpenGL maps the texture to the surface, rarely does one pixel in the texture (a *texel*) map to one pixel on the screen. A single texel may map to several screen pixels (magnification) or to only a part of one screen pixel (minification). You can control how OpenGL magnifies or minifies a texel. The following two lines use the easiest and fastest method for specifying the filtering method:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

Notice that you can specify the magnification and minification methods separately. The following table shows the allowable values for the above OpenGL parameters:

OpenGL Parameter	Values
GL_TEXTURE_MAG_FILTER	GL_NEAREST
	GL_LINEAR
GL_TEXTURE_MIN_FILTER	GL_NEAREST
	GL_LINEAR
	GL_NEAREST_MIPMAP_NEAREST
	GL_NEAREST_MIPMAP_LINEAR
	GL_LINEAR_MIPMAP_NEAREST
	GL_LINEAR_MIPMAP_LINEAR

The OpenGL screen savers define the output quality of the magnification and minification, as shown below.

Quality	OpenGL Parameter	Value
High	GL_TEXTURE_MAG_FILTER	GL_LINEAR
	GL_TEXTURE_MIN_FILTER	GL_LINEAR_MIPMAP_NEAREST
Medium	GL_TEXTURE_MAG_FILTER	GL_LINEAR
	GL_TEXTURE_MIN_FILTER	GL_LINEAR
Low	GL_TEXTURE_MAG_FILTER	GL_NEAREST
	GL_TEXTURE_MIN_FILTER	GL_NEAREST

The screen savers do not actually use the high-quality setting, because it is much slower than the medium- and low-quality settings. The Red Book discusses these parameters in detail.

Choosing the Texturing Mode

Next, we need to decide which texturing mode we want to use. OpenGL has two modes—decal and modulate:

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
```

and

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

When the texture mode is GL_DECAL, the texture covers the object below it, as shown in Figure 5. Notice that the mapped texture is the same color as the original texture.

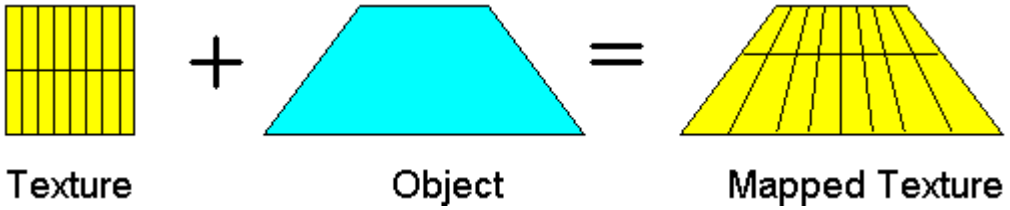

Figure 5. Texture mapping using the GL_DECAL mapping mode

When the texture mode is GL_MODULATE, the color values in the texture modify the intensity of the colors in the object instead of replacing the colors. Notice in Figure 6 below that the mapped texture is the same color as the original object. The lines in the texture turn dark blue when they are mapped to the object.

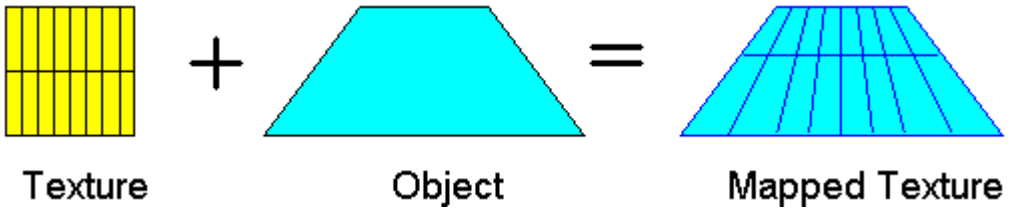

Figure 6. Texture mapping using the GL_MODULATE mapping mode

If you use GL_MODULATE with dark textures, the mapped surface may end up black. For example, the REDBRICK.BMP bitmap I use for the face of the cube becomes black when it modulates the surface. To retain the color of a texture, I recommend that you use a white surface.

The Decal and Modulate commands in the EasyTex Texture menu allow you to change the texture mapping mode.

Specifying the Texture

Before OpenGL can texture-map a surface, it needs a texture to use. The OpenGL command for specifying the two-dimensional texture is **glTexImage2D**. The **glTexImage2D** parameters are very similar to **glDrawPixels** parameters, which are discussed in "OpenGL V: Translating Windows DIBs."

I added a new member function, **TexImage2D**, to **CGLImage** to encapsulate the call to **glTexImage2D**. The code for **CGLImage::TexImage2D** is listed below.

```
void CGLImage::TexImage2D(CGL* pGL)
{
   ASSERT((m_iWidth != 0) && (m_iHeight !=0)) ;
   ASSERT(m_pBits) ;

   // m_iWidth and m_iHeight must be power of 2.
   pGL->MakeCurrent() ;
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

   glTexImage2D( GL_TEXTURE_2D,         // Reserved
                 0,                     // Detail level
                 3,                     // Components used for modulating
                                        // and blending
                 m_iWidth, m_iHeight,   // Width and height
                 0,                     // Border size
                 m_PixelFormat,         // GL_RGB or GL_RGBA
                 GL_UNSIGNED_BYTE,      // Data type of pixel values
                 m_pBits                // Pixel values
               );
```

```
   pGL->OutputGlError("CGLImage::TexImage2D") ;
}
```

The first parameter to **glTexImage2D** is reserved for future use and must be GL_TEXTURE_2D. The final six parameters (width, height, border size, pixel format, data type, and pixel values) are the same as the **glDrawPixels** parameters we discussed in "OpenGL V: Translating Windows DIBs." This leaves two interesting parameters: the second (detail level) and third (components for modulating and blending).

Multiple Levels of Detail

The second parameter to **glTexImage2D** sets the level of detail for the image. You can specify multiple images in different resolutions for the same texture—OpenGL will then use the smaller images to texture-map the smaller areas, and the bigger images to texture-map the larger areas. Using multiple levels of detail can prevent strange artifacts in moving shapes.

Using multiple levels of detail makes sense. When you get closer to an object, the object gets bigger and you can see more detail. When you are farther away, the object is smaller and you see fewer details. If you select intelligently, using different resolution images for the texture can result in more realistic scenes and animation.

For example, when you are far away from a stop sign, you see only a red circle. When you get closer, you see a red shape with some letters. Finally, you see the entire stop sign (Figure 7).

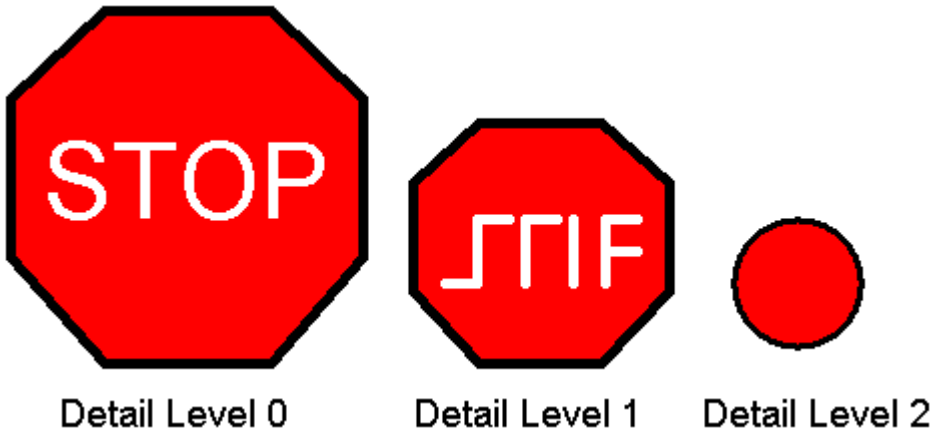

Figure 7. Multiple levels of detail for a stop sign

Instead of using different detail levels, what if OpenGL took the large bitmap and tried to squish it into the small bitmap? In the squishing process, the white letters and the black border would be combined with the red color, resulting in a small approximation of the original—a pink blob instead of a red, round stop sign.

The OpenGL utility library includes the **gluBuild2DMipmaps** function, which builds all the detail levels when given the largest image. For more information on this function, see the Red Book.

By the way, I didn't have time to add multiple levels of detail to EasyTex. The OpenGL screen savers don't use multiple-resolution textures either, because it takes too long to build them.

Components for Modulating and Blending

The third parameter to **glTexImage2D** specifies how many of the R, G, B, and A components of each pixel in the texture make up a texel. If you specify a value of 1, a single component is used. PicCube and EasyTex specify a value of 3 for this parameter, so they use the entire RGB triple to make a texel. For more information on this parameter, see the "Modulating and Blending" section (p. 274) of the Red Book.

Specifying Texture Coordinates

Not only must OpenGL know what texture to use, it must also know where to map the texture and how to orient the texture before mapping it. The OpenGL **glTexCoord** command specifies the mapping between

texture coordinates and polygon vertexes. The coordinate passed to **glTexCoord** is associated with the vertex specified in the next **glVertex** call. An example is given below.

```
glBegin(GL_QUADS);
   glNormal3d(-s,  0.0,  0.0);

   glTexCoord2d(0.0, 0.0);
   glVertex3d(-s, -s, -s);

   glTexCoord2d(1.0, 0.0);
   glVertex3d(-s, -s, s);

   glTexCoord2d(1.0, 1.0);
   glVertex3d(-s, s, s);

   glTexCoord2d(0.0, 1.0);
   glVertex3d(-s, s, -s);
glEnd();
```

Therefore, the texture coordinate (0, 0) is mapped to the object vertex (–s, –s, –s), as illustrated in Figure 8.

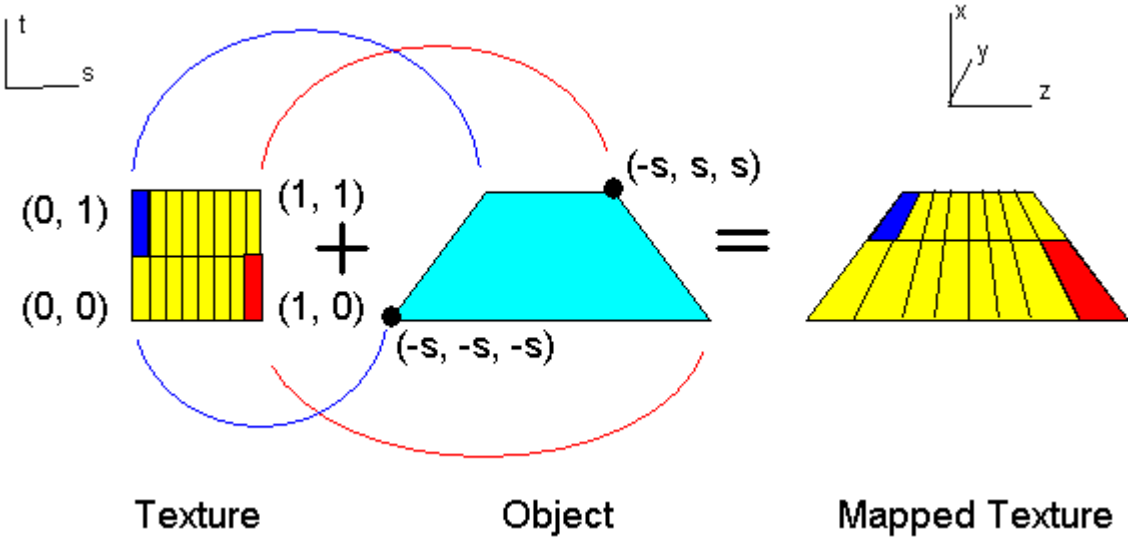

Figure 8. Specifying texture coordinates

If you pick your texture coordinates carefully, you can flip the texture, as shown in Figure 9. You can also flip the texture accidentally if you are not careful. My first version of PicCube flipped the image on three cube faces. I caught the error when I used a bitmap with text.

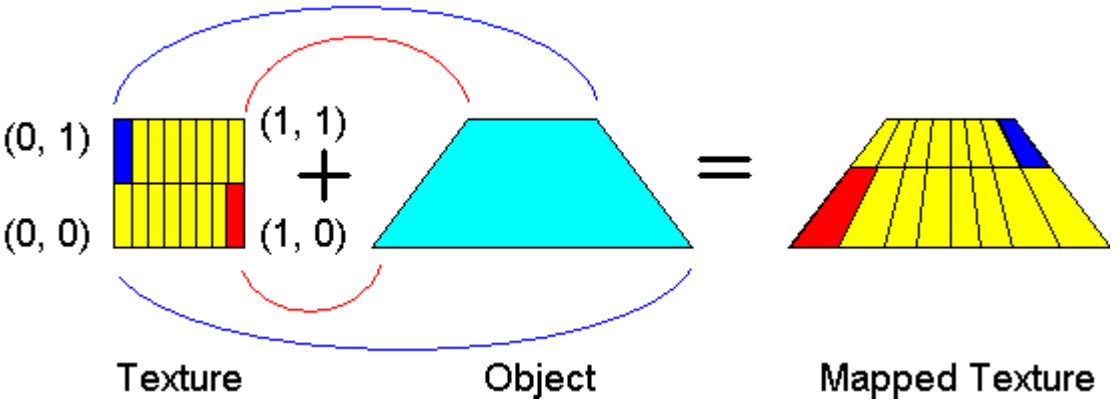

Figure 9. Flipping a texture

Repeating Textures

To make a square look like a brick wall, you can repeat the brick texture over the surface of the square. To repeat a texture, use texture coordinates greater than 1.0 (EasyTex and PicCube use 5.0).

```
glBegin(GL_QUADS);
      glNormal3d(-s,  0.0,  0.0);

      glTexCoord2d(0.0, 0.0);
      glVertex3d(-s, -s, -s);

      glTexCoord2d(5.0, 0.0);
      glVertex3d(-s, -s, s);

      glTexCoord2d(5.0, 5.0);
      glVertex3d(-s, s, s);

      glTexCoord2d(0.0, 5.0);
      glVertex3d(-s, s, -s);
   glEnd();
```

Figures 10 and 11 illustrate the effects of the two values (5.0 and 1.0).

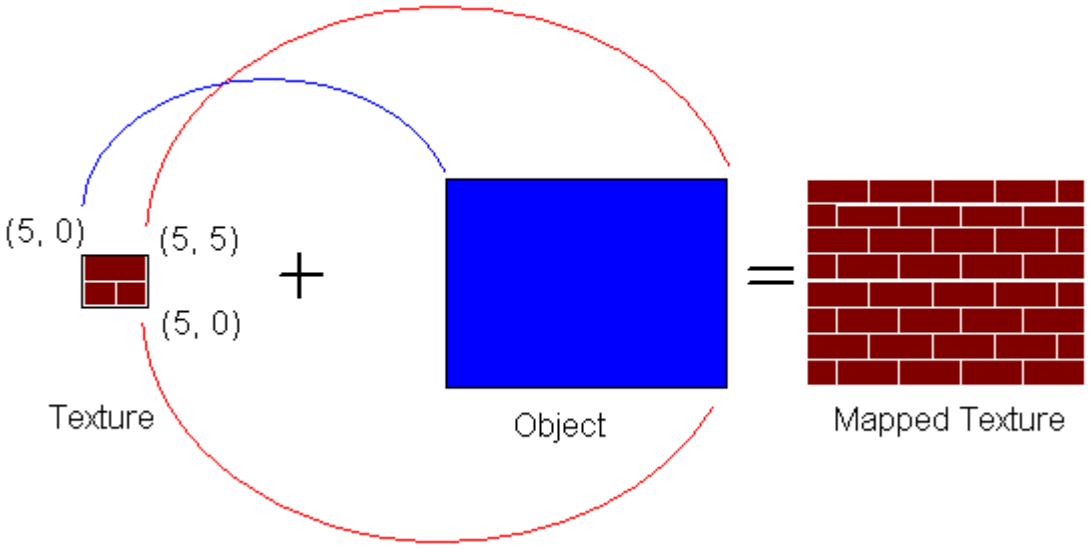

Figure 10. Specifying texture coordinates > 1.0

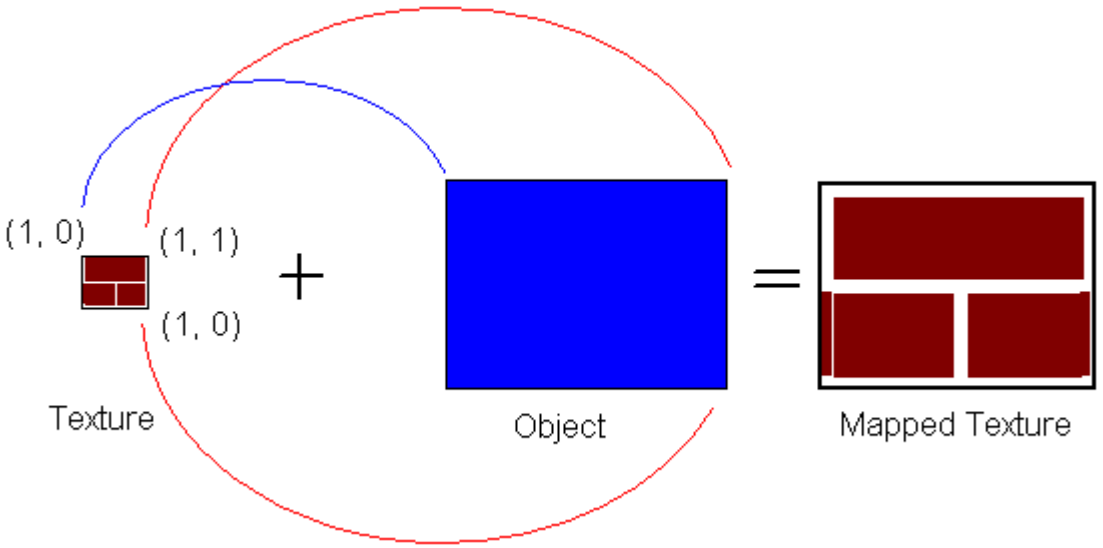

Figure 11. Specifying texture coordinates < 1.0 (or = 1.0)

To repeat textures, you must set the following two texture parameters, as we discussed previously:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

Conclusion

Texture mapping is one of the best ways to make a 3-D scene appear more realistic. Adding textures for grass, trees, wood, marble, and other surfaces can make a scene come alive. OpenGL provides a powerful set of primitives that facilitates the process of texture-mapping simple objects. Texture-mapping complicated shapes can still be very difficult—at least more so than licking stamps and sticking them on a box.

Bibliography

Sources of Information on OpenGL

Crain, Dennis. "Windows NT OpenGL: Getting Started." April 1994. (MSDN Library, Technical Articles)

Neider, Jackie, Tom Davis, and Mason Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Release 1*. Reading, MA: Addison-Wesley, 1993. ISBN 0-201-63274-8. (This book is also known as the "Red Book".)

OpenGL Architecture Review Board. *OpenGL Reference Manual: The Official Reference Document for OpenGL, Release 1*. Reading, MA: Addison-Wesley, 1992. ISBN 0-201-63276-4. (This book is also known as the "Blue Book".)

Prosise, Jeff. "Advanced 3-D Graphics for Windows NT 3.5: Introducing the OpenGL Interface, Part I." *Microsoft Systems Journal* 9 (October 1994). (MSDN Library Archive Edition, Library, Books and Periodicals)

Prosise, Jeff. "Advanced 3-D Graphics for Windows NT 3.5: The OpenGL Interface, Part II." *Microsoft Systems Journal* 9 (November 1994). (MSDN Library Archive Edition, Books and Periodicals)

Prosise, Jeff. "Understanding Modelview Transformations in OpenGL for Windows NT." *Microsoft Systems Journal* 10 (February 1995).

Rogerson, Dale. "OpenGL I: Quick Start." December 1994. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL II: Windows Palettes in RGBA Mode." December 1994. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL III: Building an OpenGL C++ Class." January 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL IV: Color Index Mode." January 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL V: Translating Windows DIBs." January 1995. (MSDN Library, Technical Articles)

Rogerson, Dale. "OpenGL VI: Rendering on DIBs with PFD_DRAW_TO_BITMAP." April 1995. (MSDN Library, Technical Articles)

Microsoft Win32 Software Development Kit (SDK) for Windows NT 3.5 *OpenGL Programmer's Reference*.

Sources of Information on DIBs

Gery, Ron. "DIBs and Their Use." March 1992. (MSDN Library, Technical Articles)

Gery, Ron. "Using DIBs with Palettes." March 1992. (MSDN Library, Technical Articles)

Microsoft Win32 Software Development Kit (SDK) for Windows NT 3.5 *Video for Windows*

Rodent, Herman. "16- and 32-Bit-Per-Pixel DIB Formats for Windows: The Color of Things to Come." January 1993. (MSDN Library Archive, Technical Articles)

Thompson, Nigel. *Animation Techniques for Win32*. Redmond, WA: Microsoft Press, 1995.

Thompson, Nigel. "Creating Programs Without a Standard Windows User Interface Using Visual C++ and MFC." September 1994. (MSDN Library, Technical Articles)

Thompson, Nigel. "Simple Custom Controls for 32-Bit Visual C++ Applications." November 1994. (MSDN Library, Technical Articles)