Texture Mapping

The Basic Idea

Texture mapping is a computer graphics operation in which a separate image, referred to as the texture, is stretched onto a piece of 3D geometry and follows it however it is transformed. This image is also known as a texture map. This can be any image. Some graphics hardware requires the image's pixel dimensions to be a power of two. (This restriction has been lifted on many graphics cards, but just to be safe...). The X and Y dimensions do not need to be the same power of two, just a power of two. So, a 128x512 image would be OK, a 129x511 image might not.

Also, to prevent confusion, the texture pixels are not called pixels. A pixel is a dot in the final screen image. A dot in the texture image is called a texture element, or texel.

Similarly, to avoid terminology confusion, a texture's width and height dimensions are not called X and Y. They are called S and T. A texture map is not generally indexed by its actual resolution coordinates. Instead, it is indexed by a coordinate system that is resolution-independent. The left side is always S=0, the right side is S=1, the bottom is T=0, and the top is T=1. Thus, you do not need to be aware of the texture's resolution when you are specifying coordinates that point into it. Think of S and T as a measure of what fraction of the way you are into the texture.

The mapping between the geometry of the 3D object and the S and T of the texture image works like this:

You specify an (s,t) pair at each vertex, along with the vertex coordinate. At the same time that OpenGL is interpolating the coordinates, colors, etc. inside the polygon, it is also interpolating the (s,t) coordinates. Then, when OpenGL goes to draw each pixel, it uses that pixel's interpolated (s,t) to lookup a color in the texture image.

Enable texture mapping:

```
glEnable( GL_TEXTURE_2D );
```

Draw your polygons, specifying s and t at each vertex:

```
glBegin( GL_POLYGON );
glTexCoord2f( s0, t0 );
glNormal3f( nx0, ny0, nz0 );
glVertex3f( x0, y0, z0 );
glTexCoord2f( s1, t1 );
glNormal3f( nx1, ny1, nz1 );
glVertex3f( x1, y1, z1 );
. . .
glEnd( );
```

If this geometry is static (i.e., will never change), it is a good idea to put this all into a display list.

Disable texture mapping:

```
glDisable( GL_TEXTURE_2D );
```

Using a Texture: Assign an (s,t) to each vertex

Enable texture mapping:

```
glEnable( GL_TEXTURE_2D );
```

Draw your polygon, specifying a s,t at each vertex:

```
gBegin( GL_POLYGON );
gTexCoordCoord( x0, y0 );
gTexCoordCoord( x1, y1 );
gTexCoordCoord( x2, y2 );
gTexCoordCoord( x3, y3 );
gTexCoordCoord( x4, y4 );
. . .
gEnd( );
```

If this geometry is static (i.e., will never change), it is a good idea to put this all into a display list.

Disable texture mapping:

```
gDisable( GL_TEXTURE_2D );
```

Using a Texture: How do you know what (s,t) to assign to each vertex?

The easiest way to figure out what s and t are at a particular vertex is to figure out what fraction across the object the vertex is living at. For a plane,

```
s = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}
t = \frac{Y - Y_{\text{min}}}{Y_{\text{max}} - Y_{\text{min}}}
```

Or, for a sphere,

```
s = \frac{\Theta - (-\pi)}{2\pi}
t = \frac{\Phi - (-\pi/2)}{\pi}
```

From the Sphere code:

```c
s = (\cos\theta + M_{\text{PI}}) / (2M_{\text{PI}});
t = (\sin\theta + M_{\text{PI}/2}) / M_{\text{PI}};
```
Using a Texture: How do you know what \((s, t)\) to assign to each vertex?

\(\text{glTexCoord2f}(s_0, t_0)\);

Uh-oh. Now what? Here’s where it gets tougher….

You really are at the mercy of whoever did the modeling...

Be careful where \(s\) abruptly transitions from 1. back to 0.

Reading in a Texture from a BMP File

unsigned char *BmpToTexture();
unsigned char *Texture;
int width, height;

Texture = BmpToTexture( "filename.bmp", &width, &height );

This function can be found in the bmp2text.cpp file. The file filename.bmp needs to be created by something that writes uncompressed 24-bit color BMP files, or converted to the uncompressed BMP format by a tool such as ImageMagick’s convert or Adobe Photoshop.

Define the texture wrapping parameters. This will control what happens when a texture coordinate is greater than 1.0 or less than 0.0:

\[
glTexParameteri(\ GL\_TEXTURE\_2D,\ GL\_TEXTURE\_WRAP\_S,\ \text{wrap})\ ;
glTexParameteri(\ GL\_TEXTURE\_2D,\ GL\_TEXTURE\_WRAP\_T,\ \text{wrap})\ ;
\]

where \text{wrap} is:

- \text{GL\_REPEAT} specifies that this pattern will repeat (i.e., wrap-around) if transformed texture coordinates less than 0.0 or greater than 1.0 are encountered.
- \text{GL\_CLAMP} specifies that the pattern will “stick” to the value at 0.0 or 1.0.

Define the texture filter parameters. This will control what happens when a texture is scaled up or down.

\[
glTexParameteri(\ GL\_TEXTURE\_2D,\ GL\_TEXTURE\_MAG\_FILTER,\ \text{filter})\ ;
glTexParameteri(\ GL\_TEXTURE\_2D,\ GL\_TEXTURE\_MIN\_FILTER,\ \text{filter})\ ;
\]

where \text{filter} is:

- \text{GL\_NEAREST} specifies that point-sampling is to be used when the texture map needs to be magnified or minified.
- \text{GL\_LINEAR} specifies that bilinear interpolation among the four nearest neighbors is to be used when the texture map needs to be magnified or minified.
Texture Environment

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.

Texture Objects

The OpenGL glTexImage2D function doesn't just use that texture. It downloads it from the CPU to the GPU, every time that call is made! After the download, this texture becomes the "current texture image".

glTexImage2D(GL_TEXTURE_2D, level, ncomps, width, height, border, GL_RGB, GL_UNSIGNED_BYTE, Texture);

If your scene has only one texture, this is easy to manage. Just do it once and forget about it.

But, if you have several textures, all to be used at different times on different objects, it will be important to maximize the efficiency of how you create, store, and manage those textures. In this case you should bind texture objects.
glBindTexture(GL_TEXTURE_2D, tex1); // designate the tex1 texture as the “current texture”
// and set its parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

texEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

TextureArray1 = TextureArray2(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, TextureArray1);

Then, later on in Display():

Then, later on in Display():

Before Binding

After Binding

Some Great Uses for Texture Mapping you have seen in the Movies

Mandelzoom:
In this case, the texture is a pure equation, so you never run out of resolution.

Extra Topic: Procedural Texture Mapping

You can also create a texture from data on-the-fly. In this case, the fragment shader takes a grid of heights and uses cross-products to produces surface normal vectors for lighting. While this is “procedural”, the amount of height data is finite, so you can still run out of resolution.

Although this looks like an incredible amount of polygonal scene detail, the geometry for this scene consists of just a single quadrilateral.