Textures in OpenGL

There are 6 steps to using textures in OpenGL:

1. Enable texturing
2. Make a 2D array of colors for the texture
3. Set up parameters for how the texture wraps.
4. Say how the texture combines with other material properties
5. Create the texture
6. Give texture coordinates to go with your geometry.

The code in the following explanations is taken from the TexSimpleFloat.java program.

1 To enable texturing:

```
gl.glEnable(GL2.GL_TEXTURE_2D);
```

Note that OpenGL also supports 3D textures. 3D textures are interesting and useful, but the textures are harder to create and we aren't going to use them this term.

2. Make a 2D array of colors

As with splines, OpenGL wants to communicate with the graphics card so it wants a buffer of colors. Although conceptually a texture is a 2D array, we need to write that into a flat 1D array. I find it fairly easy to do something like the following. For accuracy the number of ROWS and COLS of the texture map should be powers of 2.

```
public void makeTexture(float buffer[]) {
        // ROWS and COLS are the size of the texture
        for (int i = 0; i < ROWS; i++)
                for (int j = 0; j < COLS; j++) {
                        if (isEven(i/8+j/8))
                                assignColor(buffer, i, j, 0f, 0f, 0f)
                        else
                                assignColor(buffer, i, j, 1f, 0f, 0f)
                }
}
```

where assignColor( ) copies the 3 color coordinates into the buffer at index i*COLS*3+j*3

```
public void assignColor( float dest[], int i, int j, float colorR, float colorG, float colorB){
		dest[i*COLS*3+j*3]=colorR;
		dest[i*COLS*3+j*3+1]=colorG;
		dest[i*COLS*3+j*3+2]=colorB;
}
```

3. Setup parameters for how the texture wraps

If WRAP is set, texture coordinate 3.5 is the same as 0.5.

For MAG (nification) the question is: If (s, t) aren't integers, what point in the texture will (s, t) map to?

```
gl.glTexParameterf(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_WRAP_S, GL2.GL_REPEAT);
gl.glTexParameterf(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_WRAP_T, GL2.GL_REPEAT);
gl.glTexParameterf(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_MAG_FILTER,
                                                                 GL2.GL_NEAREST);
gl.glTexParameterf(GL2.GL_TEXTURE_2D, GL2.GL_TEXTURE_MIN_FILTER,
                                                                 GL2.GL_NEAREST);
```

Instead of GL2.GL_REPEAT you could say GL2.GL_CLAMP, which means "don't wrap".

4. Say how the texture combines with other materials:

```
gl.glTexEnvi(GL2.GL_TEXTURE_ENV, GL2.GL_TEXTURE_ENV_MODE, GL2.GL_REPLACE);
```

In place of GL2.GL_REPLACE you could say GL2.GL_MODULATE.

"Replace' means to use the texture color in place of any other color computation or material property. "Modulate" means to blend the texture color with color computed from the material properties.

5. Create the texture map

gl.glTexImage2D(GL2.*GL_TEXTURE_2D, 0,*
GL2.GL_RGB, COLS, ROWS, 0, GL2.GL_RGB,
GL2.GL_FLOAT, FloatBuffer.wrap(buffer));

The args are

GL2.GL_TEXTURE_2D
level (for recursive textures; we'll leave this 0)
GL2.GL_RGB (format for texture)
numbers of columns and rows; powers of 2
border (should be 0)
GL2.GL_RBG (format for output)
GL_FLOAT (format for color coordinates)
the buffer

Finally, when creating polygons, assign a texture coordinate to each vertex. Like colors, the texture coordinates are properties that stay fixed until you change them. When a vertex is created the current texture coordinates are used for it. We set texture coordinates with

```
        gl2.GLTexCoord2(s, t)
```

t refers to columns of the texture map, s to rows.

Multiple Textures

For both performance and convenience you probably want to create and install all of your textures at the start of the program, and then invoke them when you need them.

See the program MultipleTextures.java for an example of the following:

We do this in two steps.

1. gl2.glGenTextures(num, intList, offset)
   num is the number of textures you want to create, *intList* is a flat array of num integers, *offset* is usually 0.
2. gl2.glBindTexture(GL2.GL_TEXURE_2D, id)
   id should be one of the texture ids you generaeted with glGenTextures. If *id* is currently unused this creates a new texture, and stores it in *id.* Subsequent calls like glTexImage2D affect this texture. If *id* is used this makes that texture the current one, so texture coordinates refer to this texture.

Automatic Texture Generation

If you have complex polygons, calculating all of the texture coordinates can be painful. OpenGL can calculate the texture map as a direct mapping from (x, y, z) world coordinates to (s, t) texture coordinates. It defines s as

$$s = ax+by+cz+d.$$

and defines t similarly.

The code is

```
gl.glEnable(GL2.GL_TEXTURE_GEN_S);
gl.glEnable(GL2.GL_TEXTURE_GEN_T);

gl.glTexGeni(GL2.GL_S,  GL2.GL_TEXTURE_GEN_MODE,  GL2.GL_OBJECT_LINEAR);
gl.glTexGeni(GL2.GL_T,  GL2.GL_TEXTURE_GEN_MODE,  GL2.GL_OBJECT_LINEAR);

gl.glTexGenfv(GL2.GL_S,  GL2.GL_OBJECT_PLANE,  planes, 0);
gl.glTexGenfv(GL2.GL_T,  GL2.GL_OBJECT_PLANE,  planet, 0);
```

This says "turn on texture synthesis for both s and t and take the [a, b, c, d] parameters from arrays planes and planet."

For example, suppose we have a triangle with vertices (10, 0, 0), (0, 10, 0) and (0, 0, 10) and we want s to be 0, 1, and 0,5 at these three points. Then s = ax+by+cz+d gives us 3 equations in a, b, c, and d::

$$0 = 10a+d$$
$$1 = 10b+d$$
$$0.5 = 10c+d$$

3 linear equations under-determine 4 variables, so there are multiiple solutions. One easy solution is to take d=0; from this it is easy to calculate a=0, b = 0.1, and c = 0.05, so our *planes* array becomes {0, 0.1, 0.05, 0}

See program AutoGenTexture.java

To Make Textures From Image Files

1. Convert the image file to the PPM (portable pixmap) format. Photoshop can do this, or there are lots of format-translation programs available on the web.

2. Read the texture array from the file into an int[ ][ ] array. The format for most ppm files and the one used by Photoshop is

```
P6
# Any number of comment lines
#
...
cols  rows  max  // as text integers
if max < 256 the rest of the file consists of
          cols*rows*3 unsigned bytes, each byte
          representing a primary color in the
          range 0 to 255
otherwise the rest of the file is cols*rows*3
2-byte pairs, each pair representing a primary
in the range 0 to 2^16-1
```

If the rows and columns aren't powers of 2, scale the texture with

```
glu.gluScaleImage(GL2.GL_RGB, cols_in, rows_in,
        GL2.GL_FLOAT, original_array,
        cols_out, rows_out, GL2.GL_FLOAT, new_array)
```

Then put into a 1-dimensional buffer and proceed as before.

See program FileTexture.java, which puts a texture fro file spots.ppm onto a rectangle.

To Wrap Textures Around Quadric Objects

There are 2 ways to do this: you can use quadric textures or automatic texture generation. Quadric textures are easy but don't give much control. Automatic textures give some control but don't tend to wrap as well onto rounded surfaces.

For quadric textures, create the texture as usual with calls to glTexParameter( ) and glTexImage2( ). Then call

```
glu.gluQuadricTexture(quad_object, GL2.GL_TRUE)
```

to apply the texture.

See the program QuadricTextures.java. The upper image is done with quadric textures.

To use automatic textures with quadrics, just define the automatic textures as usual. In OpenGL's terminology, on the texture array s parameterizes the columns, t the rows. If that is mapped onto a cylinder, t goes from 0 at the base to 1 at the top; s goes around the cylinder from 0 to 1.

O a sphere s wraps around the latitude circles (like the equator). t moves along the longitude lines, from 0 at the south pole to 1 at the north pole.

In the program QuadricTextures.java, the lower quadric uses automatic textures.

Textures For Spline Surfaces

Again, there are 2 ways to texture spline surfaces: spline textures and automatic textures. Automatic textures work the same as elsewhere. For spline textures create the texture as usual. You need to enable mapping texture coordinates:

```
gl.glEnable(GL2.GL_MAP2_TEXTURE_COORD_2);
```

You need to make a 2x2 array of texture coordinates, organized as a float buffer:

```
float texel[] = {0f, 0f, 1f, 0f, 0f, 1f, 1f, 1f}
```

This has (0, 0) in the upper left corner, (1, 0) in the upper right, (0, 1) in the lower left, and (1, 1) in the lower right.

Then, just as you normally call

glMap2f(...) to build a spline evaluator, you also call

```
gl.glMap2f(GL2.GL_MAP2_TEXTURE_COORD_2, 0f, 1f, 2, 2, 0f, 1f,
                                4, 2, FloatBuffer.wrap(texel));
```

to build an evaluator for the texture coordinates. The arguments are the same every time.   When you then call glMapGrid2f( ) to create a mesh for the splines and glEvalMesh2 to display it, the texture coordinates are evaluated along with the geometry and the texture is applied to the mesh.

See the programs

- SplineTexture.java, which puts a checkerboard texture on a bottle,
- SplineFileTextureSpots.java, which puts a texture from the file spots.ppm on the bottle, and
- SplineFileTexture.java, which puts a texture from the file Martini.ppm on one patch of the bottle.