Lab 8 - Texture Mapping

You may work in pairs on this assignment. To receive credit, push your code to your git repository by 11:59PM on April 10th. Be sure to send me an email with your partner's name and which repository the code is pushed to.

In this lab, you'll be exploring the basics of texture mapping. Texture mapping provides added realism when rendering objects. Rather than computing all details of a material, they can be looked up in a texture. A texture can simply be an image that represents the color of an object, but can also contain other information such as normal, or specular data.

Part 1 - Texture coordinates

Texture coordinates are used to look up the information contained within a texture. Each vertex in your geometry requires a texture coordinate. A texture coordinate is a 2 element vector (use glm's vec2), which has components that range from 0 to 1. Choose coordinates for each vertex so that our texture image is mapped to each face. The x component of your texture coordinate maps from the left side (0) of the image to right (1), and the y components maps from the bottom (0) to the top (1). Create an array of uv coordinates that correspond to each of the cube's vertices, upload them to a buffer and bind them to a uv variable in your vertex and fragment shaders. This is very similar to what you did for normals in lab 6. Before moving on, make sure your cube has uv coordinates by binding the x and y components of your uv coordinate to the red and green components of your output color in your fragment shader (i.e something like color_out = vec4(uv.x, uv.y, 0, 1)). Each face of your cube should have a black corner (0,0), a red corner (1,0), a green corner (0,1) and a yellow corner (1,1).

Part 2 - Load a texture

This texture data was taken from the OpenGL Programming Guide, Eighth Edition. It represents an 8x8 black and white checkered pattern. Copy it into your initializeCube method.

```
static const GLubyte tex_checkerboard_data[] =
{
   0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00,
   0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF,
   0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00,
   0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF,
   0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00,
   0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF,
   0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00,
   0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF
};
```

Now you need to create a texture object. Created a private member variable GLuint textureObject in glwidget.h. Use the glGenTextures function to create a texture object in OpenGL:

```
glGenTextures(1, &textureObject);
```

Then you need bind the texture using glBindTexture:

```
glBindTexture(GL_TEXTURE_2D, textureObject);
```

Now load the texture data into the texture object using glTexImage2D:

```
glTexImage2D(GL_TEXTURE_2D,
0,GL_RED,8,8,0,GL_RED,GL_UNSIGNED_BYTE,tex_checkerboard_data);
```

The above function call tells OpenGL that our texture is 8 pixel x 8 pixels and we're only specifying data for a red component as an unsigned byte. Now we need to tell OpenGL how to sample from our texture:

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

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

The last step is to add a uniform sampler2D variable to our fragment shader:

```
uniform sampler2D tex;
```

Now to sample a color from the texture we can use the GLSL function, texture (make sure you use the uv variable you created in Part 1):

```
color_out = texture(tex, uv);
```

Things to notice

Try scaling multiplying your uv coordinates by 2. Notice how the faces of the cube change. The glTexParameter calls determine how your texture is sampled. In this case, we've set the wrap behavior of our texture to GL_CLAMP_TO_EDGE. Because we scaled our texture up by

2, a portion of the face is clamped to the color at the edge of our texture. Other settings can tell OpenGL to repeat the texture pattern or to mirror it.

We've also set how OpenGL should select a color when the number of pixels in the texture don't match up exactly with the pixels on the face of the cube. We've told OpenGL to just pick the nearest color, but you can also tell OpenGL to interpolate the nearest pixels to get a more antialiased look. Experiment with the different settings you can pass to glTexParameter.

Recommended Reading

OpenGL Programming Guide, Eighth Edition, Chapter 6
Texture Mapping - Wikipedia