SIMON MAURICE - IPHONE - OPENGL ES

Wednesday, 22 April 2009

OpenGL ES 10 - Multiple Textures, Repeating Textures, and The End of the Book Era

My friend stopped by last night and we had a couple of beers and chatted about various things. Then I told him I was writing these tutorials. All the time I was talking, he had this "what the f%#k?" look on his face, gradually getting worse as I said I was putting them for all to read on the internet. Putting information on the internet for free just seemed too weird for him. He just didn't get it. He, being a consultant for a large IT company, declared he would never do anything like that because it would reduce the amount of "billable hours" he could charge clients. Why don't I "just write a book"? After all, I could earn some money too.

You see, I'm happy to share my knowledge with you for free. I'm not going to do a book and charge money for my knowledge, I don't think that's appropriate in modern times. It was fine in the 80's and 90's before the majority of the population was on the internet and before AltaVista made full text search of web pages possible (yes, they did it before Google).

Whilst on my bookshelf contains such classics as *Computer Graphics:*

Principles and Practice, various editions of *Graphics Gems*, and *Programming in 3 Dimensions,* I haven't purchased a book in 10 years or more. Not because I'm not learning; I've learned two new platforms in the past decade, written thousands of lines of code for photorealistic rendering tools that I only started using since 2000, and even learnt assembly language for a series of micro-controllers.

No books required anymore. People who came before me shared their information in blogs or websites as badly designed and edited as mine. No charge. I learnt Mac OS X and the iPhone OS from information from Apple and websites like iphonedevsdk.com or icodeblog.com. Didn't need a book. Last I looked there were no decent books on iPhone programming. Two of them that I know of include Apple undocumented code which is 100% guaranteed to get your app rejected from the app store. It's not right that people can charge money for such bad practice.

Teaching something to a fellow programming doesn't detract from my value; in fact it probably enhances things for me as there can be robust discussions as a result of this series and even new friendships forged. It does make me feel good when I get your emails thanking me for series which helps my karma. Besides, I love to code.

It's my time to put back into the community. I know I'm not a teacher, but at least you don't pay for it.

Now, getting on with today's tutorial and, as promised, I'm answering a request to use multiple textures. We're going to texture map our cube again but I'm going to get rid of the checkerplate image. Instead we are going to load 6 new textures, one for each side of the cube. Then we're going to look at texture wrapping and texture clamping.

The new textures for this tutorial are located here:
Tutorial10Textures.zip

loadTexture[] Changes
The first thing we need to do is to make the `loadTexture[]` method more flexible. Basically, we're going to change the call to

```
- (void)loadTexture:(NSString *)name
intoLocation:(GLuint)location;
```

Pretty straight forward. We're going to change the `loadTexture[]` method so we can tell the method which texture to load and where to put it. Next, switch to the implementation and change the `loadTexture[]` method as follows:

```
- (void)loadTexture:(NSString *)name
intoLocation:(GLuint)location {
```

```
    CGImageRef textureImage = [UIImage imageNamed:name].CGImage;
```

Don't forget to remove the @"checkerplate.png" and replace it with our variable name. Go down and delete the glGenTextures() method and change the glBindTexture() method to the following:

```
glBindTexture(GL_TEXTURE_2D, location);
```

That's it. Now this method is far more useful. In order to use it, we need to make a call to `glGenTextures()` in our `initWithCoder[]` method, then we just make the call to `loadTexture[]` for each texture we're going to load. In this tutorial, we're going to load 6 textures, so, in `initWithCoder[]` after the call to `setupView`, add the following lines:

```
        [self setupView];
        glGenTextures(6, &textures[0]);
        [self loadTexture:@"bamboo.png"
intoLocation:textures[0]];
        [self loadTexture:@"flowers.png"
intoLocation:textures[1]];
        [self loadTexture:@"grass.png" intoLocation:textures[2]];
        [self loadTexture:@"lino.png" intoLocation:textures[3]];
        [self loadTexture:@"metal.png" intoLocation:textures[4]];
        [self loadTexture:@"schematic.png"
intoLocation:textures[5]];
```

Also, switch over to the header file and change the definition of out textures variable from 1 to 6 like this:

```
GLuint textures[6];
```

Getting on With the Rendering

OK, now down to `drawView[]`. Delete eveything relating to the pyramid, we're not using it any more. We do need to edit the cube drawing code. All we need to do is to delete the colours which we used to tint our previous texture. So make it look like this:

```
    // Our new drawing code goes here
    rota += 0.2;

    glPushMatrix();
    {
        glTranslatef(0.0, 0.0, -4.0);        // Change this line
        glRotatef(rota, 1.0, 1.0, 1.0);
        glVertexPointer(3, GL_FLOAT, 0, cubeVertices);
        glEnableClientState(GL_VERTEX_ARRAY);

        // Draw the front face
        glDrawArrays(GL_TRIANGLE_FAN, 0, 4);

        // Draw the top face
        glDrawArrays(GL_TRIANGLE_FAN, 4, 4);

        // Draw the rear face
        glDrawArrays(GL_TRIANGLE_FAN, 8, 4);
```

```
        // Draw the bottom face
        glDrawArrays(GL_TRIANGLE_FAN, 12, 4);

        // Draw the left face
        glDrawArrays(GL_TRIANGLE_FAN, 16, 4);

        // Draw the right face
        glDrawArrays(GL_TRIANGLE_FAN, 20, 4);
    }
    glPopMatrix();
```

You can leave the `glPushMatrix() and glPopMatrix()` there. It doesn't matter but if you can't sleep at night knowing that you're code isn't 100% optimised, remove them.

Now that we've made some changes, let's stop for a second and think about about OpenGL's state. I keep coming back to the "state" of OpenGL because that's the nature of this beast. Turn something on and it's stays on until you tell it otherwise.

So, remember way back when loading textures, we would use `glBindTexture()` to tell OpenGL which texture we want to work on? So, after loading the six textures, OpenGL will be "bound" to the last texture we loaded. If you were to hit "Build and Go", you'd get the cube with all faces rendered in the one texture.

Since we have a texture for each side of our cube, we need to tell OpenGL which texture we want it to use each time we make a call to `glDrawArrays()`. Thus, our texturing code looks like this:

```
    glPushMatrix();
    {
        glTranslatef(0.0, 0.0, -4.0);
        glRotatef(rota, 1.0, 1.0, 1.0);
        glVertexPointer(3, GL_FLOAT, 0, cubeVertices);
        glEnableClientState(GL_VERTEX_ARRAY);

        // Draw the front face
        glBindTexture(GL_TEXTURE_2D, textures[0]);
        glDrawArrays(GL_TRIANGLE_FAN, 0, 4);

        // Draw the top face
        glBindTexture(GL_TEXTURE_2D, textures[1]);
        glDrawArrays(GL_TRIANGLE_FAN, 4, 4);

        // Draw the rear face
        glBindTexture(GL_TEXTURE_2D, textures[2]);
        glDrawArrays(GL_TRIANGLE_FAN, 8, 4);

        // Draw the bottom face
        glBindTexture(GL_TEXTURE_2D, textures[3]);
        glDrawArrays(GL_TRIANGLE_FAN, 12, 4);

        // Draw the left face
        glBindTexture(GL_TEXTURE_2D, textures[4]);
        glDrawArrays(GL_TRIANGLE_FAN, 16, 4);

        // Draw the right face
```

```
        glBindTexture(GL_TEXTURE_2D, textures[5]);
        glDrawArrays(GL_TRIANGLE_FAN, 20, 4);
    }
    glPopMatrix();
```

Hit "Build and Go" and you'll be rewarded with six different textures on our cube.

Hope that answers the questions for those of you who wasn't sure how this was done.

Texture Wrapping

The next request which I've been asked for is repeating or wrapping textures. Repeating a texture is simply just having the same texture repeat multiple times. It's quite useful for using smaller elements to texture a wall (provided they all fit together neatly that is).

What we'll do is to just take one of the sides of our cube and repeat the texture four times. I'm going to change the bamboo texture which is the texture rendered onto the front face. You can do any face you like.

It's very easy. Do you recall how OpenGL decides which pixel from our image to render on our objects? It uses the texture co-ordinate array which I told you before had to be between 0.0 and 1.0. If we want an image to repeat twice on an object, we just change the 1.0 to a 2.0 and voila, it will be repeated.

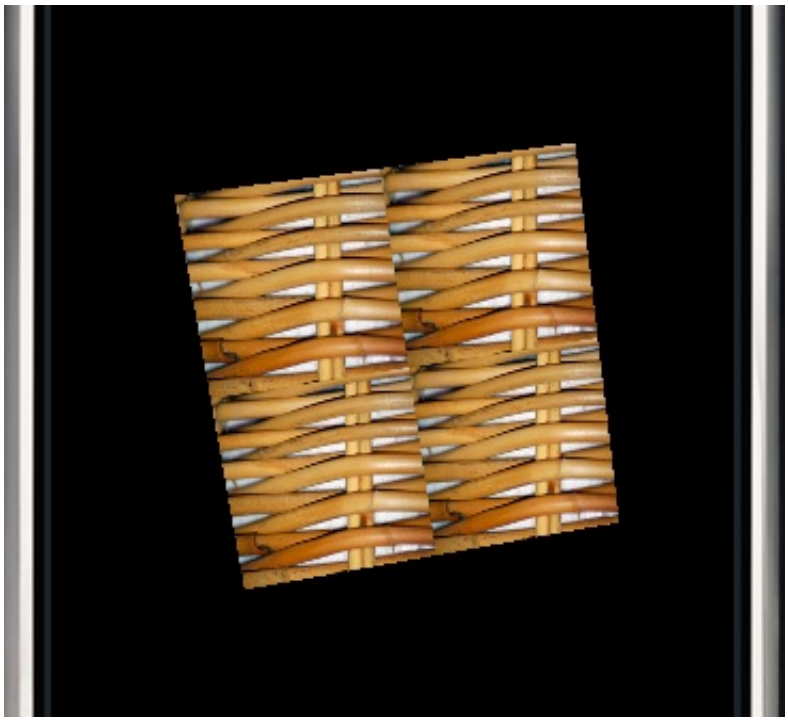
So, in our texture co-ordinate array, change the front face to the following:

```
const GLshort squareTextureCoords[] = {
    // Front face
    0, 2,       // top left
    0, 0,       // bottom left
    2, 0,       // bottom right
```

```
        2, 2,       // top right

        // Top face
        0, 1,       // top left
        0, 0,       // bottom left
        1, 0,       // bottom right
        1, 1,       // top right
```

Just hit "Build and Go" and you'll be rewarded with:

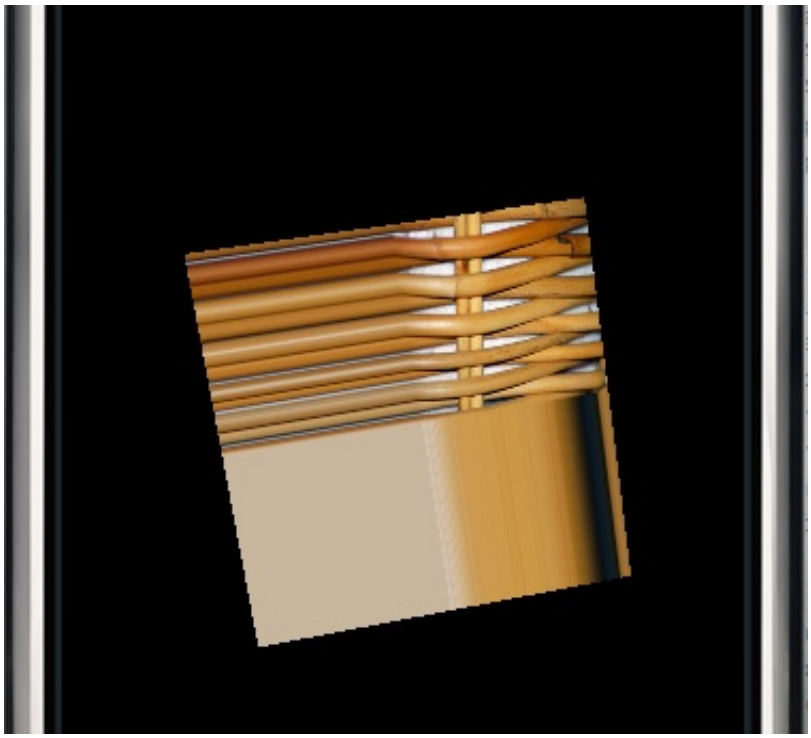


I just did the front face but you can do any (or all) faces if you like.

This is the default texture mapping state for OpenGL, when it hits values greater than 1.0 in the co-ordinate array, it texture wraps. The Opposite to this is texture **clamping**. You can enable texture clamping with a call to `glTexParameterf()`. For example, if we were to change the drawing code for the front face to:

```
        // Draw the front face
    glBindTexture(GL_TEXTURE_2D, textures[0]);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
    glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

The above code, coupled with our texture co-ordinates of > 1.0 will cause OpenGL to render the following:



The edge texture pixel has been extended outwards from the only correctly drawn texture in the top left. Hence, GL_CLAMP_TO_EDGE.

Let me just explain these two lines quickly:

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

We used a call to `glTexParameterf()` when we loaded the textures to set the texture filtering. Like many OpenGL functions, it does "double duty". There are three parameters but OpenGL ES only two come into play, the first parameter is always `GL_TEXTURE_2D` as this is the only texture format OpenGL ES supports (the three parameter function is kept for compatibility with OpenGL.

The second parameter tells OpenGL what "variable" we are going to change. In the above case it is `GL_TEXTURE_WRAP_x`.

The third parameter is just the value we want to set.

By setting the OpenGL "variable" or "setting" `GL_TEXTURE_WRAP_S` to `GL_CLAMP_TO_EDGE`, we are telling OpenGL not to repeat the textures along the *s* co-ordinates (yes "*s*", will explain below). We have then also done the same for the *t* co-ordinates. Try commenting one or both out and running it again in the simulator.

The default value for both of these settings is GL_REPEAT which is what was happing before we changed these to the clamped setting.

s t Co-ordinates?
No, not joking, OpenGL uses an s t co-ordinate system for it's textures. It's really there so as not to confuse you because you have to remember our textures do not live in our 3D world, they are something we ***apply*** to objects in our 3D world. So you can think of s as horizontal and t as vertical but as soon as you rotate the object that you mapped it onto, will it still apply? :)

Also, note that the full OpenGL supports 3D textures so you may also come across r & q co-ordinates.

That's it!
I'm going to leave it here for today as I'm going to be putting something else up tomorrow. I've now realised I've been making some personal mistakes in the past tutorials and I'm going to take everything in a new direction. We're going to have some fun!

As usual, here's the code for this tutorial:

AppleCoder-OpenGLES-10.zip

The home of the tutorials is in the "Tutorials" section of the iphonedevsdk.com forums. Check out the thread there.

Until next time, hooroo!
Simon Maurice

previous next

Made on a Mac

Email Me