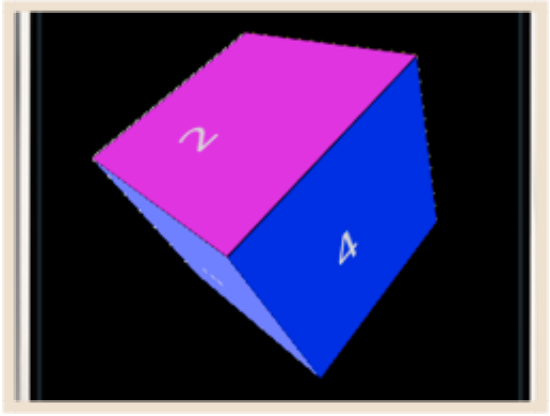
SIMON MAURICE - IPHONE - OPENGL ES



Monday, 22 June 2009

OpenGL ES 16 - Blender Models Part 3: Textures and UV Mapped Objects

Quick Intro: *I'm basically uploading this tutorial now because I've really moved on from this point and I'm finding it really hard to come back and do some expansion on this. So rather than just sit here and stare at it, I'm just going to pop this one up and keep going with the next stage: exporting a map file, loading that up, walking around (without walking through walls) etc.*

There is code for this tutorial, I hope it still works because, like I said, I've moved on. All that really needs to be added now from this is multiple textures & multiple meshes per scene which is all I was going to add. However, that can also be covered in the map tutorial coming next.

Not sure when I'll get it up as over the next three weeks, I'm very busy with being involved in a community theatre production (no, I'm not in cast, I'm on the technical team of course!) but I should get the next one up in a couple of days.

Anyway, here we go...

Doh!

Made a bit of an opps in the last tutorial. Remember how I said that script would not work for cubes, planes etc because they were quads and the script was designed to fail at the assert if the face was a quad? Well, that's just my mistake. It does work.

You see, I did that whole "Mesh -> Faces -> Convert Quads to Triangles" menu function (or **CTRL-T**) but every time I went back in to the console and then looked at the data, I still had

quads, not triangles. I just assumed it was something that Blender did for editing and never thought that much of it. Turning a quad into two triangles isn't really hard. So off I went, wrote a script for it and was using it quite happily until today when I was reading some more of the Blender documentation.

I was reading the API for Object.getData() as I was trying to work out the difference between an "N" object vs normal (eg NMesh vs a Mesh), and something struck me immediately in the notes (which I had clearly missed before):

Make sure the object you are getting the data from isn't in EditMode before calling this function; otherwise you'll get the data before entering EditMode.

Yeah, mega "oooops".

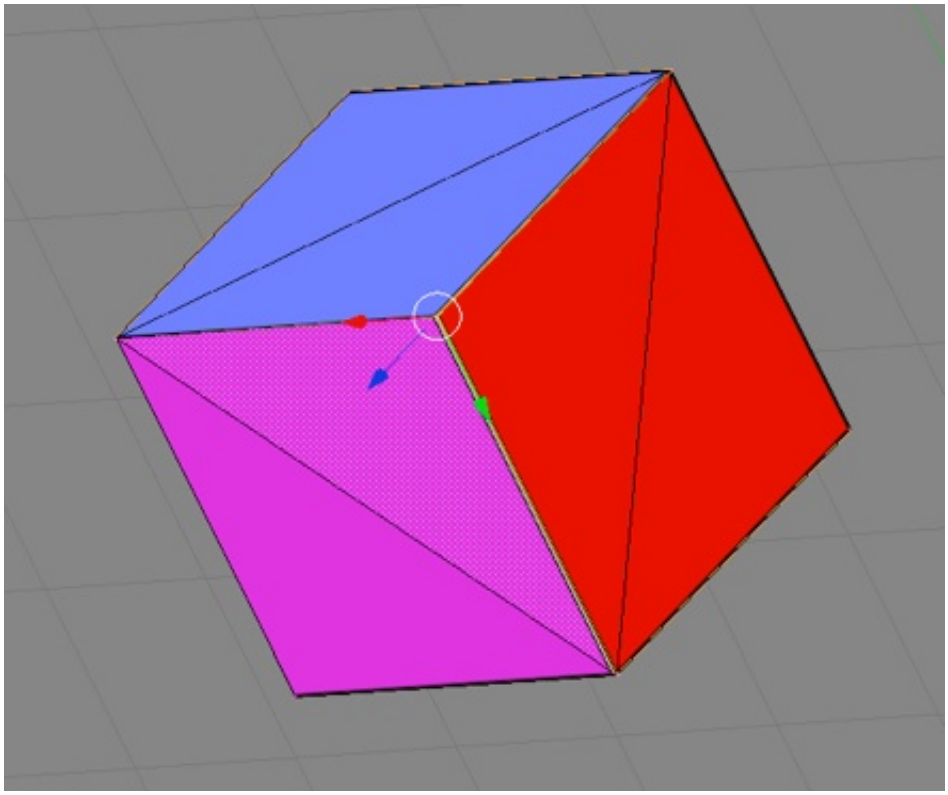
You see, I was not exiting edit mode in the 3D editor *before* making another call to getData() to get the object's data. Hence I was getting back quads and not triangles.

I didn't think anything of it because I had read so many export scripts which had a triangulate quad functions so it just seemed normal for me to roll my own as well. I did know about the Blender Python call Mesh.quadToTriangle() but I read that was "experimental" so I avoided it (I don't like using experimental or in development APIs, they're far too random).

As Denis Waitley put it: *"The only person who never makes mistakes is the person who never does anything"*.

To me, that quote says it all.

Onto Texture Mapping
OpenGL ES only supports 2D texture maps so using the same technique as we did last time won't work. What we did was to export the unique vertices only which is fine for just exporting something in the one colour, but as soon as you apply materials in Blender (textures, colours etc), you have one co-ordinate with three possible combinations of materials. Have a look at this cube:



That selected vertex has one co-ordinate, one normal, but three possible materials as it is shared amongst three different triangles. So what we need to do in the export script is ensure that we actually generate three co-ordinates, one for each triangle in order to get textures working with UV mapping.

In the first instance, I exported everything and fed it into glDrawElements() and glDrawArrays() just to benchmark to see which was the quicker. glDrawArrays() won out slightly so I'm going to revert to using that for drawing in this tutorial.

I made two changes to the export script. The first was to change the way I wrote out the file to include the UV co-ordinates from Blender as follows:

```
# Write an interleaved array containing vertex co-ordinate, normal, and UV
co-ord
for face in mesh.faces:
    for i in range(0, 3):
        file.write(struct.pack("fff", *face.v[i].co))
        file.write(struct.pack("fff", *face.v[i].no))
        file.write(struct.pack("ff", *face.uv[i]))
```

The UV co-ordinates are stored in the Face object but we still need to get the vertex data from the face's actual vertex array (face.v).

Then I just did something to fix the whole edit mode thingy mentioned above.

```
inEditMode = Blender.Window.EditMode()
if inEditMode:
    Blender.Window.EditMode(0)
Blender.Window.FileSelector(saveAllMeshes, "iPhone Export",
newFileName("gldata"))
if inEditMode:
    Blender.Window.EditMode(1)
```

That just means that we exit edit mode before running the script's body and so changes are taken into account. You still need to triangulate the model prior to running the script (**CTRL-T**) or from the Mesh menu in the 3D view.

Changes to the Blender Model Class
Now, when loading the model, I've created a data structure just to make life a bit easier for both loading and rendering the model. It's pretty straightforward:

```
typedef struct __GLVertexElement {
    GLfloat coordiante[3];
    GLfloat normal[3];
    GLfloat texCoord[2];
} GLVertexElement;
```

And then in the class definition, the instance variable:

```
GLVertexElement *data;
```

gets populated.

Then, loading the class becomes a little easier as we just need to load one single chunk of data once we know how many triangles there are:

```
    data = malloc(sizeof(GLVertexElement) * triangleCount * 3);
```

```
    [[handle readDataOfLength:sizeof(GLVertexElement) * triangleCount * 3]
getBytes:data];
```

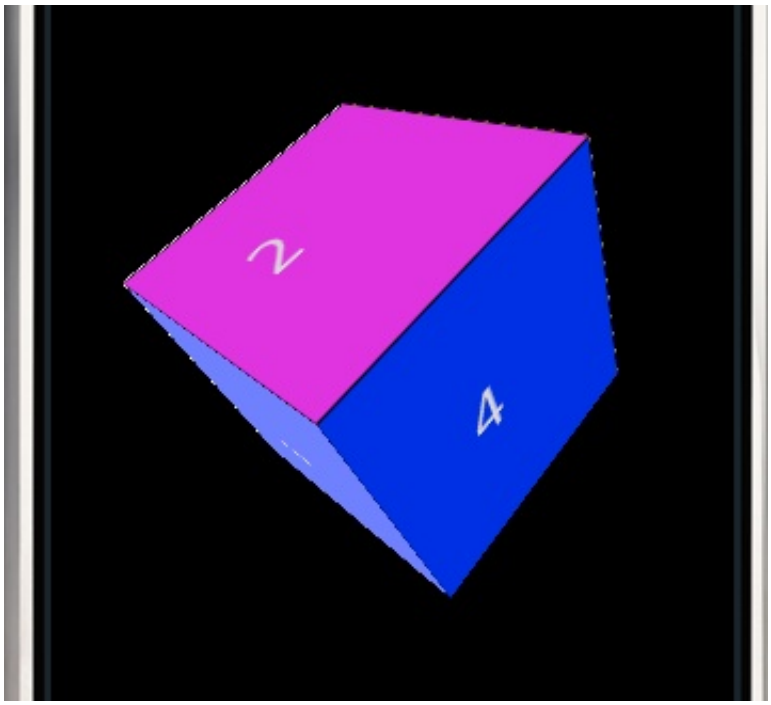
Rendering the model just becomes:

```
    glVertexPointer(3, GL_FLOAT, sizeof(GLVertexElement), data);
    glNormalPointer(GL_FLOAT, sizeof(GLVertexElement), data->normal);
    glTexCoordPointer(2, GL_FLOAT, sizeof(GLVertexElement), data->texCoord);
    glDrawArrays(GL_TRIANGLES, 0, triangleCount*3);
```

Of course, this gets wrapped with the glEnable/glEnableClientState/glDisable/glDisableClientState functions. The only change here is to use glDrawArrays rather than glDrawElements().

Finally, just added a method to load the texture file which is just the same as before.

The end result is a cube like this:



A Quick Ending
Like I said at the start, I'm just going to wind this one up as I want (or, more to the point) already have moved on from this.

Next time, we'll load something with a map and get back into new OpenGL techniques as well as other information needed for handling a map efficiently.

As usual, here's the code: OpenGLES16.zip

There's no .blend file for the cube as I don't think I saved it. If you can't work out UV mapping in Blender (it changed a couple of versions ago and a lot of the tutorials on the internet haven't caught up yet), I'll put something up elsewhere on this website with a real quick tutorial in the next couple of hours.

Until next time, hooroo!
Simon Maurice

Linking to these pages o

previous next

Made on a Mac


Email Me