Tuesday, 31 March 2009

OpenGL ES 03 - Transformations

Today, we're going to build on what's been covered so far and put both the triangle and square on the screen at the same time. To do this, we are going to move them. Moving an object is one thing of what's called **Transformation**.

With OpenGL ES, there are three different types of transformations which you can use on your model (object). They are:

- Translate - Move the object within 3D space.
- Rotate - Rotation around either the X, Y, or Z axis.
- Scale - Alter the size of the object. Primarily this is used in 2D orthographic projection systems as in 3D the farther an object is away (ie the more negative the Z co-ordinate) the smaller the object is rendered. But can of course be used for "special effects".

To demonstrate these different functions, what we're going to do is to bring both the square and the triangle onto the screen at the same time using the translate function and then proceed to the other two.

Translate
In order to effect a translation, OpenGL ES gives us a single function

which we can use, aptly called `glTranslatef()`. Note the "f" at the end of the word translate? That just means that we are going to feed OpenGL floating point data. OpenGL ES also provides the opportunity to use fixed point data and thus call the function `glTranslatex()`. Fixed point data would be used on hardware without a dedicated floating point maths co-processor but as the iPhone has one built in, we don't need to use fixed-point maths and we can just stick to floating point maths.

I just wanted to mention that in case you're Xcode code completion prompts you to using `glTranslatex()` and you weren't sure what the difference is.

OK, time to start cutting some code. Fire up Xcode and open your project. I hope you took my advice and commented out the triangle data and rendering calls and didn't delete them otherwise you're going to have to re-type them again.

First of all, let's look at the two vertex arrays. We are going to make a change to the vertex data but only for the Z co-ordinates. Change all the Z co-ordinates to 0.0 like the following:

```
const GLfloat triangleVertices[] = {
        0.0, 1.0, 0.0,                    // Triangle top centre
        -1.0, -1.0, 0.0,                  // bottom left
        1.0, -1.0, 0.0                    // bottom right
    };

const GLfloat squareVertices[] = {
        -1.0, 1.0, 0.0,                   // Top left
        -1.0, -1.0, 0.0,                  // Bottom left
        1.0, -1.0, 0.0,                   // Bottom right
        1.0, 1.0, 0.0                     // Top right
    };
```

Do you remember why we had the Z co-ordinates as -6.0? It was because we needed to set the objects back into the screen because our "camera" is at (0.0, 0.0, 0.0). What we're going to do is to use the `glTranslatef()` function to set them back 6 points instead of specifying it in our vertex arrays.

First, we need to tell OpenGL what we're going to be translating: either the Projection (view of the world) or the Objects (models within that world). In this case, it's the square and the triangle so we need to tell OpenGL this. Below the call to `glClear()` in your `drawView` method, call the following OpenGL function:

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
```

The call to `glMatrixMode` tells OpenGL to work on our vertex data rather than the projection. In our `setupView` method, we call this same function but use the `GL_PROJECTION` enum as the parameter. OpenGL is an engine which remains in the same "state" until you tell it otherwise. So, the Matrix Mode will stay in `GL_PROJECTION` until we say go to the model view by calling `glMatrixMode(GL_MODELVIEW)`. Now that we have set the OpenGL state to `GL_MODELVIEW`, it will remain there until we tell it otherwise.

So, in fact, we could actually make this call at the end of the `setupView` method we created in the first tutorial if we wanted maximum performance. However, we are in tutorial world and not real world at the moment so just leave it in the `drawView` method.

I know I haven't really covered using OpenGL ES in projection mode yet so don't panic if you don't fully understand the above. All we're doing is putting some objects onto a screen and playing around with them while you learn OpenGL ES.

Now, remove the comments around the draw triangle code which looks like this:

```
glVertexPointer(3, GL_FLOAT, 0, triangleVertices);
    glEnableClientState(GL_VERTEX_ARRAY);
    glDrawArrays(GL_TRIANGLES, 0, 3);
```

Before these three lines of code, add the following two lines:

```
glLoadIdentity();
glTranslatef(-1.5, 0.0, -6.0);
glVertexPointer(3, GL_FLOAT, 0, triangleVertices);
    glEnableClientState(GL_VERTEX_ARRAY);
    glDrawArrays(GL_TRIANGLES, 0, 3);
```

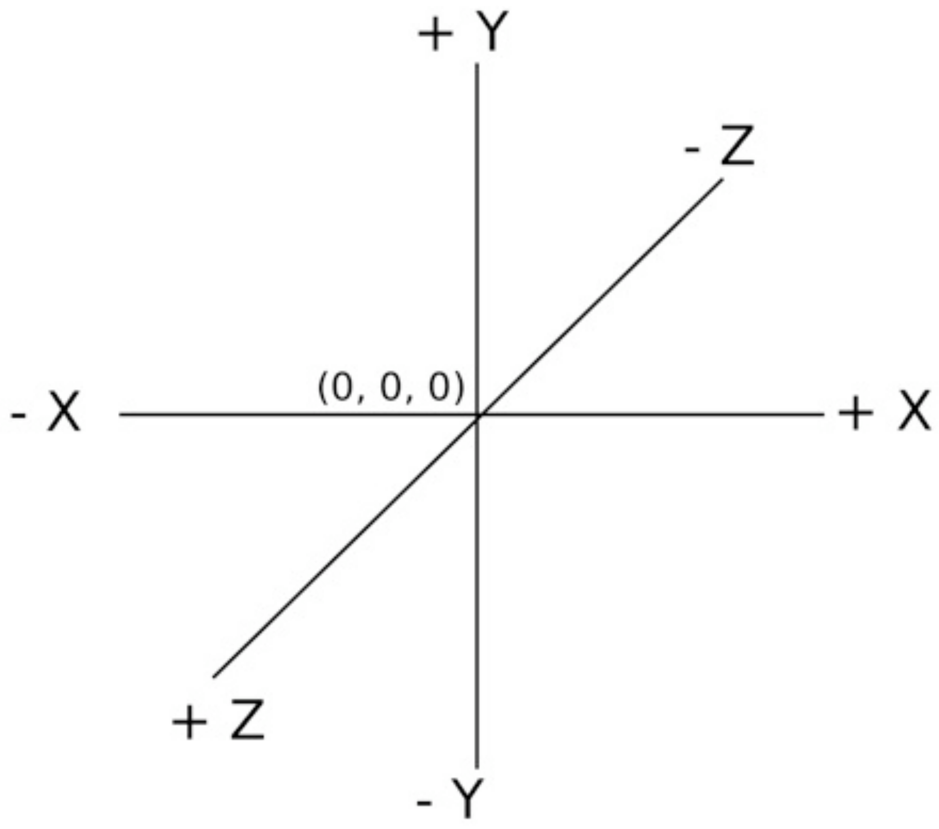
`glLoadIdentity()` is just a connivence function to basically reset everything back to the original conditions. If we didn't call this function, the `glTranslatef()` call would continue to move the object to the left and back into the screen until it disappeared. I'm actually going to cover a better way to do this later on (the tutorial after next in fact) so just accept for now that we're just resetting the object data.

The next function call is where the action happens.

`glTranslatef()` takes three parameters:

```
    glTranslatef(GLfloat xtrans, GLfloat ytrans,
GLfloat Ztrans);
```

Just review my drawing of the 3D world space before continuing.



Remember the "camera" is at (0.0, 0.0, 0.0). Our call to `glTranslatef()` above has the following values:

```
    xtrans = -1.5
    ytrans =  0.0
    ztrans = -6.0
```

The next thing you need to recall is that when we draw both the square and the triangle, the appeared in the centre of the screen. If we were to draw them as per the previous tutorials, they would be drawn on top of each other.

So, in order to get around this, I have moved the X co-ordinate of the centre of the triangle to the left 1.5 points. Referring to the image above of the co-ordinate space, you can see that left of screen centre is negative, hence the negative 1.5.

The -6.0 for the Z transformation replaces the -6.0 which we had originally in the object's vertex array.

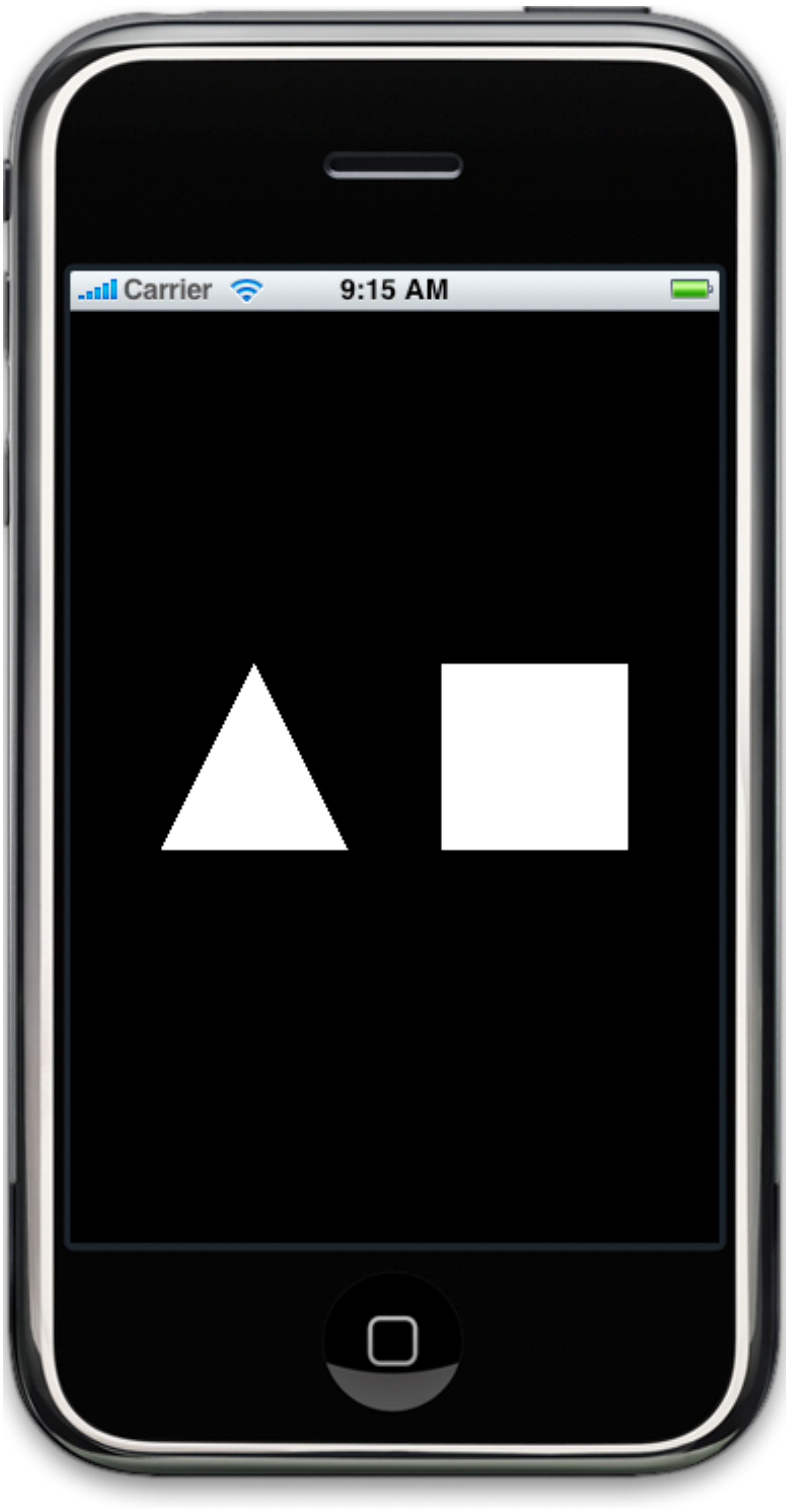
We have moved the triangle to the left 1.5 and back 6.0 points.

Moving Onto the Square
The code for the square is almost the same as the triangle. It is as follows:

```
    glLoadIdentity();
    glTranslatef(1.5, 0.0, -6.0);
    glVertexPointer(3, GL_FLOAT, 0, squareVertices);
    glEnableClientState(GL_VERTEX_ARRAY);
    glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

This time, we have moved the square to the *right* by passing a positive value to the `xtrans` value in `glTranslatef()`.

Hit “Build and Go” and have a look at what’s on the screen, you should get this image:



Note that the Y co-ordinate is still centred on the screen, the sizing is the same as before because we have set them back -6.0, and they are nicely placed side-by-side.

Before We Move On

Experiment by changing values of the `xtrans`, `ytrans`, and `ztrans` in `glTranslatef()` and see what happens. I spent many an hour just changing values and watching what happens. You can even try commenting out `glLoadIdentity()` and see what happens.

Rotation

Get your code back to the way it is above after your experimentation and let's have a quick look at rotation. We'll rotate in 2D as our objects are only 2D (but in a 3D world). Later on, when we create a full 3D object, we can rotate in full 3D (and yes, we can texture map it...).

Rotation is as simple as:

```
glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat
z);
```

This function is simple to use. First you supply how much of an angle to rotate which is the first argument, then we simply specify which axis or axes to rotate.

I will demonstrate rotation in two ways. First, we will do a static rotation, then animate it by keeping the rotation going; making our square and cube spin in effect.

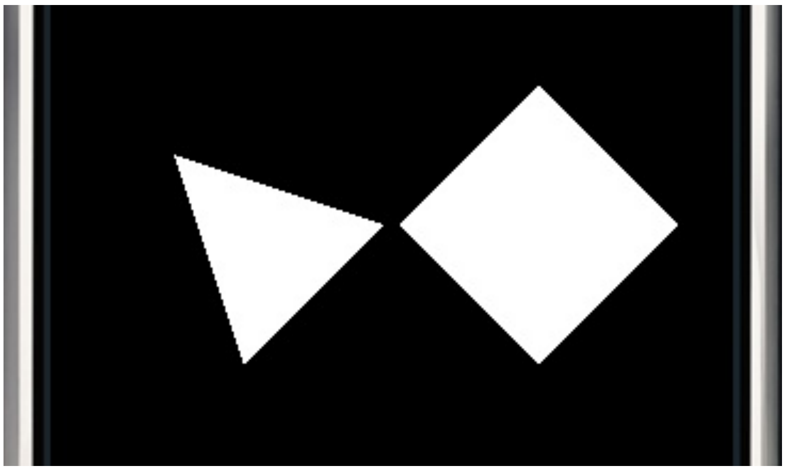
First, let's just do a simple rotation. Go to the `drawView` method and add the change the drawing code for the triangle and the square as follows:

```
    glLoadIdentity();
    glTranslatef(-1.5, 0.0, -6.0);
    glRotatef(45.0, 0.0, 0.0, 1.0);          // Add this line
    glVertexPointer(3, GL_FLOAT, 0, triangleVertices);
    glEnableClientState(GL_VERTEX_ARRAY);
    glDrawArrays(GL_TRIANGLES, 0, 3);

    glLoadIdentity();
    glTranslatef(1.5, 0.0, -6.0);
    glRotatef(45.0, 0.0, 0.0, 1.0);          // Add this line
    glVertexPointer(3, GL_FLOAT, 0, squareVertices);
    glEnableClientState(GL_VERTEX_ARRAY);
    glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

All we've done is rotated the triangle and square by 45^{o} along the Z axis. See the value 1.0 for the Z parameter? That tells OpenGL to rotate our object 45^{o} along the Z axis.

Make the changes to the code and hit "Build and Run" and you should get the following on screen:



Rotation along the Z axis "spins" the object on the screen - much like looking at a car tyre spinning when you're side onto the car. Remember, the Z axis goes into the screen so it is rotating around this axis.

Rotating the X axis, would be like looking at a spinning car tyre with the car heading towards you (ie you facing the car's front grill). Rotating the Y axis is what the tyre would look like as the driver turns the steering wheel to avoid you (hopefully!). Don't panic if you're a bit confused by that, this next example will allow you to play with rotation and really see what's going on.

Spinning Our Objects

In order to make our square and triangle "spin", we need to increase the angle each time we draw the frame. Switch to `EAGLView.h` and add the following variable:

```
    GLfloat rota;
```

Then switch back to `EAGLView.m` and, in the `initWithCoder` method, add the following line below the `animationInterval` assignment:

```
  rota = 0.0;
```

All we've done is to create a variable to hold the current rotation angle.

Now, head back to `drawView`, and add the following line of code right before the first `glLoadIdentity()` function call:

```
  rota += 0.5;
```

All we're doing is increasing the rotation angle by 0.5º each time we draw our two objects. Finally, change both the `glRotatef()` function calls to the following:

```
        glRotatef(rota, 0.0, 0.0, 1.0);
```

So, what we're doing is increasing the rotation angle every time we draw the objects, resulting in a spinning object. The first time we draw the objects, they will be rotated through 0.5º; the second time they are drawn, they will be rotated 1.0º etc.

Hit build and run and the two objects should spin like a car tyre when viewed side on.

For Your Experimentation
Before leaving this tutorial, I want you to do a couple of things and note what happens:

1. Change the axis which you rotate around. Make the Z axis 0.0 and turn on each the X axis and Y axis one axis at a time and note how the objects rotate so you can get an understanding of rotation along each individual axis.
2. Change the 1.0 on the current rotation axis to -1.0. Note that they now rotate in the opposite direction.
3. Change the `rota` parameter in `glRotatef()` to `-rota`. What happens?

I hope you got something out of this. Here's the completed code for the tutorial:

AppleCoder-OpenGLES-03.zip

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

Now, I don't know about you but I'm a bit bored with white objects. Next thing we're going to do is to add some colour to these objects.

Until next time, hooroo!
Simon Maurice

.

previous next