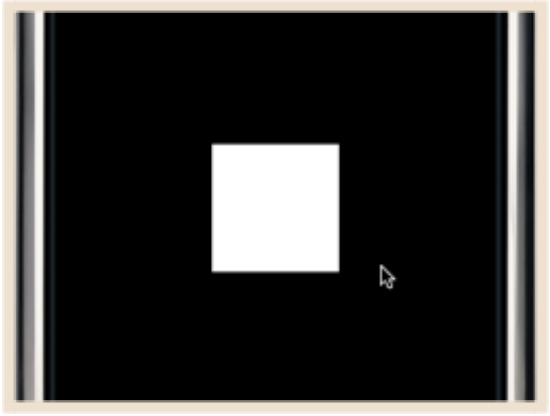
SIMON MAURICE - IPHONE - OPENGL ES



Saturday, 28 March 2009

OpenGL ES 02 - Drawing Primitives 2 - Squares

Strictly speaking, squares are not a primitive in OpenGL ES but, let's face it, they're pretty handy and just as easy as rendering a triangle. In this tutorial, we're going to take the code from the triangle primitive and turn it into a square. Again, it will be rendered statically but we're going to get into transformations (ie moving them around) quickly enough. Of course, once we've done a square, we can then make a cube, then even a texture mapped cube...

Quick Recap and Details of This Tutorial
Last tutorial, we took our "blank canvas" Xcode project and rendered a solid white triangle. In doing so, you created a vertex array, told OpenGL about the data and it's format using `glVertexPointer()`, put it in the state for rendering the vertex array, and then rendered it using `glDrawArrays()`.

Today, we're going to take that code, and make a square from the triangle. In order to do that, we will only need to change a couple of lines of code. The first one is probably obvious, we need to specify 4 points for the square instead of 3 for a triangle. Then we need to tell OpenGL to draw it differently by passing a different drawing methodology to `glDrawArrays()`.

Let's get going.

Defining the Square's Vertices
Open up the Xcode project from the last tutorial and go to the `drawView` method. Comment out the `triangleVertices[]` constant - don't discard it as we'll use it when we get into transformations - and add in the following code:

```
const GLfloat squareVertices[] = {
        -1.0, 1.0, -6.0,            // Top left
        -1.0, -1.0, -6.0,           // Bottom left
        1.0, -1.0, -6.0,            // Bottom right
        1.0, 1.0, -6.0              // Top right
    };
```

That describes our square. Note the anti-clockwise direction of the vertices (the four points) of the square?

Now go down into the body and comment out the drawing code for the triangle, again we'll bring this code back to life later on. So comment out the three function calls of `glVertexArray()`, `glEnableClientState()`, and `glDrawArrays()` and add the following code:

```
    glVertexPointer(3, GL_FLOAT, 0, squareVertices);
    glEnableClientState(GL_VERTEX_ARRAY);
    glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

It's the same three functions but just slightly different.

```
   glVertexPointer(3, GL_FLOAT, 0, squareVertices);
```

The only change here is that we are telling OpenGL to use a different set of vertices; the square now and not the triangle.

`glEnableClientState()` is the same as we are telling OpenGL to draw from a vertex array (not a colour array or something else).

```
   glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

This is the main change. In the last tutorial, we used `GL_TRIANGLES` as the first parameter and 3 as the 3rd parameter. Remember the second parameter is the offset from the start of the array to being from, again this is zero because our vertex array only contains the vertices for the square.

The first argument is the drawing mode and you've now seen two possible drawing modes for OpenGL. I do want to take the time now to discuss the different drawing modes. They are:

```
GL_POINTS
GL_LINES
GL_LINE_LOOP
GL_LINE_STRIP
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
```

We haven't discussed points or lines yet so I'll stick to the last three which refer to triangles. Before I commence, I want to remind you that a vertex array may contain more than one triangle so while you have only seen one object per vertex array, you're not limited to that.

`GL_TRIANGLES` - passing this parameter means that OpenGL treats the vertex array in sets of 3 vertices. So, for the first three, it makes a triangle bound by vertex 1, vertex 2, and vertex 3. Then it will process the next three vertices and so on to the end of the array.

`GL_TRIANGLE_STRIP` - OpenGL will start off using the first two vertices, then for each successive vertex, it will use the previous 2 vertices to make a triangle. That is, for squareVertices[6~8], a triangle is formed with squareVerticies[0~2] and squareVerticies[3~5]. For squareVertices[9~11], a triangle is formed with squareVertices[3~5] and squareVertices[6~8]. And so on through the array.

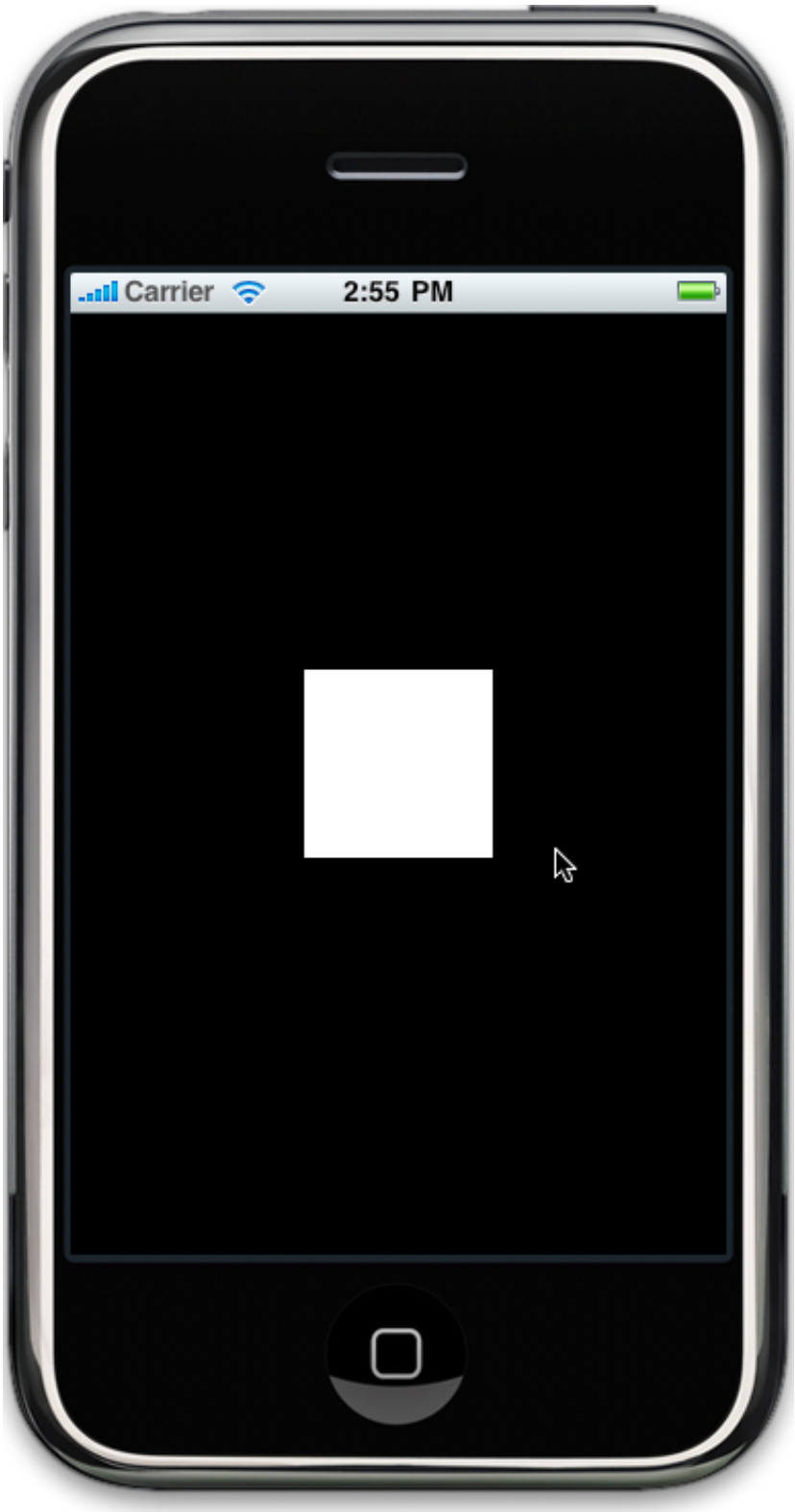
Note: the reference to squareVertices[0~2] refers to:

squareVertices[0] - X co-ordinate
squareVertices[1] - Y co-ordinate
squareVertices[2] - Z co-ordinate

I'll cover this in an example further on in this tutorial if that doesn't make sense.

`GL_TRIANGLE_FAN` - After the first 2 vertices, for each successive vertex, OpenGL will form a triangle with with the previous vertex and the first vertex. So, for squareVertices[6~8], a triangle is formed with squareVertices[3~5] (previous vertex) and squareVertices[0~2] (first vertex).

Since we used `GL_TRIANGLES_FAN`, we will get a square on the display. Hit "Build & Go", and you should be rewarded with a white square on the screen like this:



Go back and have a look at your vertex array. Try to imagine the triangles being drawn to make the square. OpenGL renders it like this.

Triangle Point 1: squareVertices[0~2] -- Top left of square
Triangle Point 2: squareVertices[3~5] -- Bottom left of square
Triangle Point 3: squareVertices[6~8] -- Bottom right of square

Taking the above 3 points, OpenGL draws a triangle which will make up the bottom left half of the square. Imagine the square is divided in

two with a diagonal line starting at the top left corner of the square to the bottom right corner. Notice how two triangles form? OpenGL just drew the bottom left half.

Note: the reference to squareVertices[0~2] refers to:
squareVertices[0] - X co-ordinate
squareVertices[1] - Y co-ordinate
squareVertices[2] - Z co-ordinate

Triangle Point 1: squareVertices[9~11] -- Top right of square
Triangle Point 2: squareVertices[6~8] -- **Previous vertex**, bottom right
Triangle Point 3: squareVertices[0~2] -- **First vertex**, top left

Taking only 1 new point, OpenGL is able render a triangle to complete the square.

GL_TRIANGLE_STRIP
Go back to the code and change the first parameter into `glDrawArrays()` from `GL_TRIANGLE_FAN` to `GL_TRIANGLE_STRIP`:

```
glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
```

Hit "Build & Go" and you'll get the following image:

Let's have a look and see why we didn't get a square due to the single change in the drawing method. OpenGL processes our vertex array in the following manner:

Triangle Point 1: squareVertices[0~2] -- top left
Triangle Point 2: squareVertices[3~5] -- bottom left
Triangle Point 3: squareVertices[6~8] -- bottom right

OpenGL now renders a triangle using the first three points, hence, the bottom left half of the previous square rendered as per the previous example.

Triangle Point 1: squareVertices[9~11] -- top right
Triangle Point 2: squareVertices[6~8] -- **Previous Vertex,** bottom right
Triangle Point 3: squareVertices[3~5] -- **2nd Previous**, bottom left

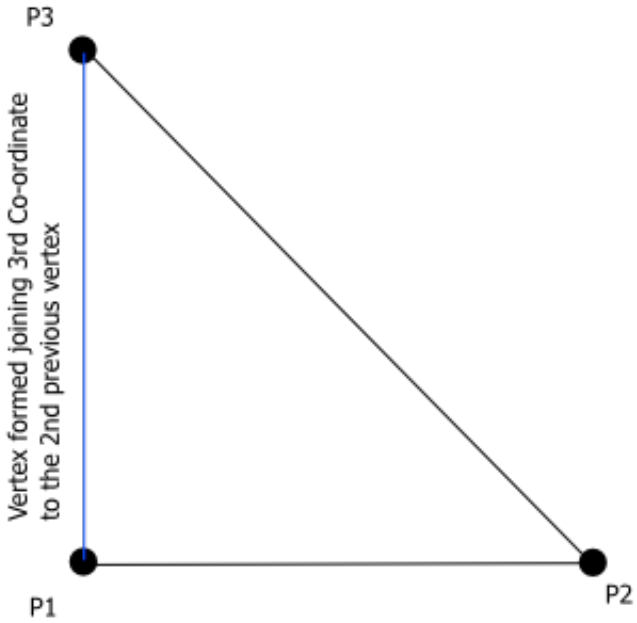
OpenGL now renders a triangle using these three points. In this example, it has rendered a triangle 90º away from what we want to make a square.

If we had of provided our vertex array differently, we could have produced a correct square using `GL_TRIANGLE_STRIP` just as well as we did we `GL_TRIANGLE_FAN`. Just remember the drawing method and your vertex array need to be considered together otherwise you'll get odd results like we did when we changed from `FAN` to `STRIP`.

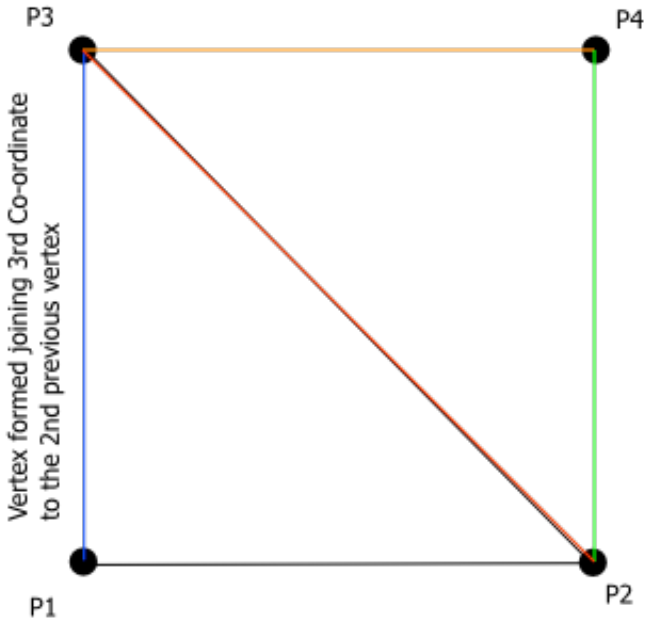
Note that you can still draw a proper square using `GL_TRIANGLE_STRIP`, you just need to specify your vertex array differently. For example:

```
const GLfloat stripSquare[] = {
    -1.0, -1.0, -6.0,               // bottom left
    1.0, -1.0, -6.0,                // bottom right
    -1.0, 1.0, -6.0,                // top left
    1.0, 1.0, -6.0                  // top right
};
```

So working with the above, we can see the first triangle will be formed through the first three vertices, producing a triangle as follows:



Now, by specifying the point of the top right vertex (P4), a new triangle will be formed with the top left (P3) and the 2nd previous vertex (P2) which is at the bottom right. The new vertices are shown in Orange, Green and Red below:



As a result, we have produced a square. The end result is still the same but it's just a reminder that you need to match your drawing methodology to the way you have specified your vertices.

Finally...
So you've now seen triangles and squares. We still need to cover points and lines yet. Both are quite simple and will be covered next. As we're building upon what we've already covered, we'll add some colour into the mix next time.

Once we can colour our objects, we'll move them around and then texture map them in 3D. Granted, it won't be Doom 3 but you'll start to be able to build 3D objects and then, I'll start to cover 3D worlds.

Here's the code for this tutorial:

AppleCoder-OpenGLES-02.zip

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

As always, feel free to email me. Home base for these tutorials is over at the iphonedevsdk.com forum under the tutorials section.

Until next time, hooroo!
Simon Maurice

previous next