OpenGL|ES Tutorials (3): 3D & Keyboard Input

How to Draw 3D Objects

Drawing 3D objects is very simple. In the previous tutorial you learnt how to create a triangle (primitive). Using this, you can make a cube: A cube is just 6 squares joined together. A square is just two triangles.

Introduction

This tutorial uses the code from the previous tutorial. All we will be doing is adding an extra function which draws the cube; then, in our Render function we will omit the call to function “DrawTriangle()” and a call to our new function. The Entire source code will not be reproduced, only the sections that have changed.

Function: DrawCube()

Just under function DrawTriangle add the following function:

```
void DrawCube()
{
  GLfloat cube[] =
  {
  // FRONT
  -0.5f, -0.5f,  0.5f,
   0.5f, -0.5f,  0.5f,
  -0.5f,  0.5f,  0.5f,
   0.5f,  0.5f,  0.5f,
  // BACK
  -0.5f, -0.5f, -0.5f,
  -0.5f,  0.5f, -0.5f,
   0.5f, -0.5f, -0.5f,
   0.5f,  0.5f, -0.5f,
  // LEFT
  -0.5f, -0.5f,  0.5f,
  -0.5f,  0.5f,  0.5f,
  -0.5f, -0.5f, -0.5f,
  -0.5f,  0.5f, -0.5f,
  // RIGHT
   0.5f, -0.5f, -0.5f,
   0.5f,  0.5f, -0.5f,
   0.5f, -0.5f,  0.5f,
   0.5f,  0.5f,  0.5f,
  // TOP
  -0.5f,  0.5f,  0.5f,
   0.5f,  0.5f,  0.5f,
   -0.5f,  0.5f, -0.5f,
   0.5f,  0.5f, -0.5f,
  // BOTTOM
  -0.5f, -0.5f,  0.5f,
  -0.5f, -0.5f, -0.5f,
   0.5f, -0.5f,  0.5f,
   0.5f, -0.5f, -0.5f,
35:         };
36:
37:         glVertexPointer(3, GL_FLOAT, 0, cube);
38:         glEnableClientState(GL_VERTEX_ARRAY);
39:
40:         // FRONT AND BACK
41:          glColor4f(1.0f, 0.0f, 0.0f, 1.0f);    //Color: RED
42:          glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
43:          glDrawArrays(GL_TRIANGLE_STRIP, 4, 4);
44:
45:          // LEFT AND RIGHT
46:          glColor4f(0.0f, 1.0f, 0.0f, 1.0f);   //Color: GREEN
47:          glDrawArrays(GL_TRIANGLE_STRIP, 8, 4);
48:          glDrawArrays(GL_TRIANGLE_STRIP, 12, 4);
49:
50:          // TOP AND BOTTOM
51:          glColor4f(0.0f, 0.0f, 1.0f, 1.0f);   //Color: BLUE
52:          glDrawArrays(GL_TRIANGLE_STRIP, 16, 4);
53:          glDrawArrays(GL_TRIANGLE_STRIP, 20, 4);
54:     }
```

Analysis: This function is much more complicated than function DrawTriangle. The array list contains co-ordinates for each vertex, of each square of the cube. Note that we simply only supply 4 co-ordinates for each square. This is because later on, you will see that we are using GL_TRIANGLE_STRIP to draw the arrays, instead if GL_TRIANGLES. I recommend you read through the array list and picture in your how head how each vertex of each square is plotted onto the invisible axis. This will give you a better understanding of what is going on, rather than just copying and pasting it.

glColor4f sets the “pen colour”. Again, this follows the same format as the glClearColor aforementioned. Parameter 1 is the “RED” value, 2 is the “GREEN” value, 3 is the “BLUE” value and 4 is the “ALPHA” value. In this case the colour is set to R-1 G-0 B-0 A-1, which means we get a solid red. You can alter these values for different colours. Any colour can be made from specifying these values. Notice we have not used a colour array, meaning we will be creating a flat colour this time.

Next we set then pen colour to red on line 41, and immediately after we draw the FRONT square (Vertexes 0 - 4), and the BACK square (Vertexes 4 – 8). Notice that glDrawArrays now takes GL_TRIANGLE_STRIP as it’s second parameter, rather than GL_TRIANGLES. This has basically means we are now drawing a polygon, instead of a triangle. You can probably work out yourself that the glDrawArrays statement on line 42, starts at the beginning of the Array (0), then draws the next four (FRONT Square); and that the next glDrawArrays statement on line 43 start at the fourth vertex (4) and draws the next four (BACK Square). This process is repeated twice, once for the LEFT and RIGHT squares (Where the pen colour is set to GREEN), and once for the TOP and BOTTOM squares (Where the pen colour is set to BLUE).

Function: Render()

```
void Render()
{
  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  DrawCube();

  glFinish();
  eglSwapBuffers(FrameworkGL_GetDisplay(), FrameworkGL_GetSurface());
}
```

Analysis: The call to function DrawTriangle() has been omittied. A call to function DrawCube() has instead been added.

Output

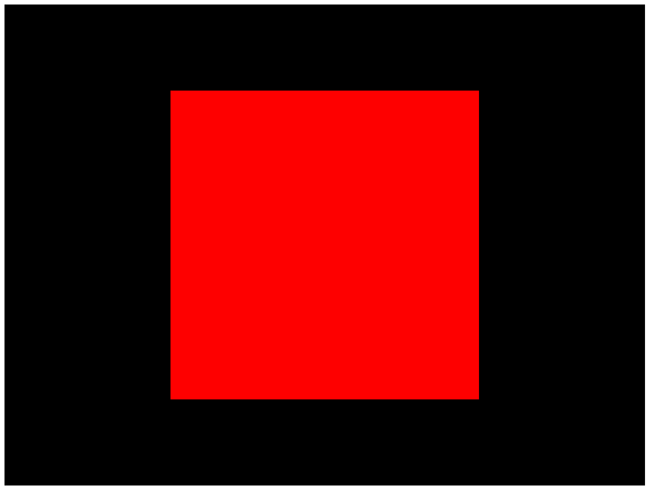

Figure 3.1 Tutorial 3 Output (Part 1)

A Square..? I can probably guess you are not very impressed by this output. You created a cube, but you can only see a square. This is simply because the front section of the cube is directly above the other parts, effectively there are hidden underneath it. However, imagine that you were looking at the square from an angle. Then you would be able to see more. This is discussed in the next section: Moving the Cube.

Keyboard Input

In this section you will learn how to rotate, and move the square in response to user input.

Introduction

Using the code from the previous listing, we will be adding only a few variables and statements to move the cube.

Declarations

At the start of your code, make the following amendments.

```
#include <windows.h>
#include <GLES\gl.h>
#include <GLES\egl.h>
#include <KGSDK\framework.h>
#include <KGSDK\frameworkGL.h>

static float rot[3];
static float loc[3];

float xrot = 0.0f;    //NEW - Measures the rotation on the x-axis
float yrot = 0.0f;    //NEW - Measures the rotation on the y-axis
```

Analysis: Two new variables have been added. “xrot” and “yrot” are both declared as type float, and they are both initialized to 0.0f. xrot will keep count of the amount the cube has been rotated on the x-axis, and y-rot will keep count of the amount the cube has been rotated on the x-axis.

Function: Update

We will add a few lines of code in this function to look out for key presses, and process them accordingly.

```
void Update()
{
  GLfixed mat[4][4];

  memset(mat, 0, sizeof(mat));
  mat[0][0] = (int) (65536.0f *  cos(rot[0]));
  mat[0][2] = (int) (65536.0f *  sin(rot[0]));
  mat[1][1] = (int) (65536.0f *  cos(rot[1]));
  mat[1][2] = (int) (65536.0f *  sin(rot[1]));
  mat[2][0] = (int) (65536.0f * -sin(rot[0]) * cos(rot[1]));
  mat[2][2] = (int) (65536.0f *  cos(rot[0]) * cos(rot[1]));
  mat[3][2] = (int) (65536.0f * loc[2]);
  mat[3][3] = 65536;

  glMatrixMode(GL_MODELVIEW);
  glLoadMatrixx(&mat[0][0]);

  if (Framework_IsButtonPressed(FRAMEWORK_BUTTON_LEFT_SHOULDER ))
     loc[2] += 0.3f;
  // If the Left Shoulder Button has been pressed: Zoom out
  if (Framework_IsButtonPressed(FRAMEWORK_BUTTON_RIGHT_SHOULDER))
     loc[2] -= 0.3f;
  // If the Right Shoulder Button has been pressed: Zoom In
24:     if (Framework_IsButtonPressed(FRAMEWORK_BUTTON_DPAD_DOWN     ))
25:        xrot -= 1.0f;
26:     // If the Down button (DPAD) has been pressed: Rotate the cube downards
27:     if (Framework_IsButtonPressed(FRAMEWORK_BUTTON_DPAD_UP       ))
28:        xrot += 1.0f;
29:     // If the Up button (DPAD) has been pressed: Rotate the cube upwards
30:     if (Framework_IsButtonPressed(FRAMEWORK_BUTTON_DPAD_LEFT     ))
31:        yrot -= 1.0f;
32:     // If the Left button (DPAD) has been pressed: Rotate the cube towards the left
33:     if (Framework_IsButtonPressed(FRAMEWORK_BUTTON_DPAD_RIGHT     ))

34:        yrot += 1.0f;
35:     // If the Right button (DPAD) has been pressed: Rotate the cube towards the Right
36:   }
```

Analysis: In this section, the only modifications are the additions of lines 18 through to line 35. Line 18 Checks to see whether the Left Shoulder button has been pressed, and then on line 19 the value of loc[2] is incremented. Line 21 and 22 do the same, but decrements the value of loc[2] if the Right Shoulder button has been pressed. Lines 24 – 28 increment/decrement the value of xrot, in response to whether the DPAD Up button, or DPAD down button has been pressed, respectively. Lines 30 – 34 also do the same but with the value of yrot, and in response to whether the DPAD LEFT/RIGHT has been pressed. You can experiment, once you have completely this tutorial, with the amount each value is incremented/decremented. For more information on Gizmondo Key Buttons, see the relevant documentation supplied with the KGSDK.

Function: Render

Finally section: All we have to do now is rotate the cube!

```
1:      void DrawCube()
2:      {
3:        GLfloat cube[] =
4:        {
5:        // FRONT
6:        -0.5f, -0.5f,  0.5f,
7:         0.5f, -0.5f,  0.5f,
8:        -0.5f,  0.5f,  0.5f,
9:         0.5f,  0.5f,  0.5f,
10:       // BACK
11:       -0.5f, -0.5f, -0.5f,
12:       -0.5f,  0.5f, -0.5f,
13:        0.5f, -0.5f, -0.5f,
14:        0.5f,  0.5f, -0.5f,
15:       // LEFT
16:       -0.5f, -0.5f,  0.5f,
17:       -0.5f,  0.5f,  0.5f,
18:       -0.5f, -0.5f, -0.5f,
19:       -0.5f,  0.5f, -0.5f,
20:       // RIGHT
21:        0.5f, -0.5f, -0.5f,
22:        0.5f,  0.5f, -0.5f,
23:        0.5f, -0.5f,  0.5f,
24:        0.5f,  0.5f,  0.5f,
25:       // TOP
26:        -0.5f,  0.5f,  0.5f,
27:         0.5f,  0.5f,  0.5f,
28:         -0.5f,  0.5f, -0.5f,
29:         0.5f,  0.5f, -0.5f,
30:        // BOTTOM
31:        -0.5f, -0.5f,  0.5f,
32:        -0.5f, -0.5f, -0.5f,
33:         0.5f, -0.5f,  0.5f,
34:         0.5f, -0.5f, -0.5f,
35:        };
36:
37:          glRotatef(xrot, 1.0f, 0.0f, 0.0f); //Rotate, xrot, on x-axis
38:          glRotatef(yrot, 0.0f, 1.0f, 0.0f); //Rotate, yrot , on y-axis
39:
40:          glVertexPointer(3, GL_FLOAT, 0, cube);
41:          glEnableClientState(GL_VERTEX_ARRAY);
42:
43:          // FRONT AND BACK
44:          glColor4f(1.0f, 0.0f, 0.0f, 1.0f);  //Color: RED
45:          glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
46:          glDrawArrays(GL_TRIANGLE_STRIP, 4, 4);
47:
48:          // LEFT AND RIGHT
49:          glColor4f(0.0f, 1.0f, 0.0f, 1.0f); //Color: GREEN
50:          glDrawArrays(GL_TRIANGLE_STRIP, 8, 4);
51:          glDrawArrays(GL_TRIANGLE_STRIP, 12, 4);
52:
53:          // TOP AND BOTTOM
54:          glColor4f(0.0f, 0.0f, 1.0f, 1.0f); //Color: BLUE
55:          glDrawArrays(GL_TRIANGLE_STRIP, 16, 4);
56:          glDrawArrays(GL_TRIANGLE_STRIP, 20, 4);
57:
58:      }
```

Analysis: Two extra lines have been added: Lines 37 and 38. This is a new function: glRotatef. The first function tells OpenGL how much to rotate the (amount to rotate). The second parameter states whether the rotation should be done on the x-axis, the third parameter states whether the rotation should be done on the y-axis, and the fourth parameter states whether the rotation should be on the z-axis. Line 37 rotates the cube, value xrot, on the x-axis; and line 38 rotates the cube, value yrot, on the y-axis.

Output

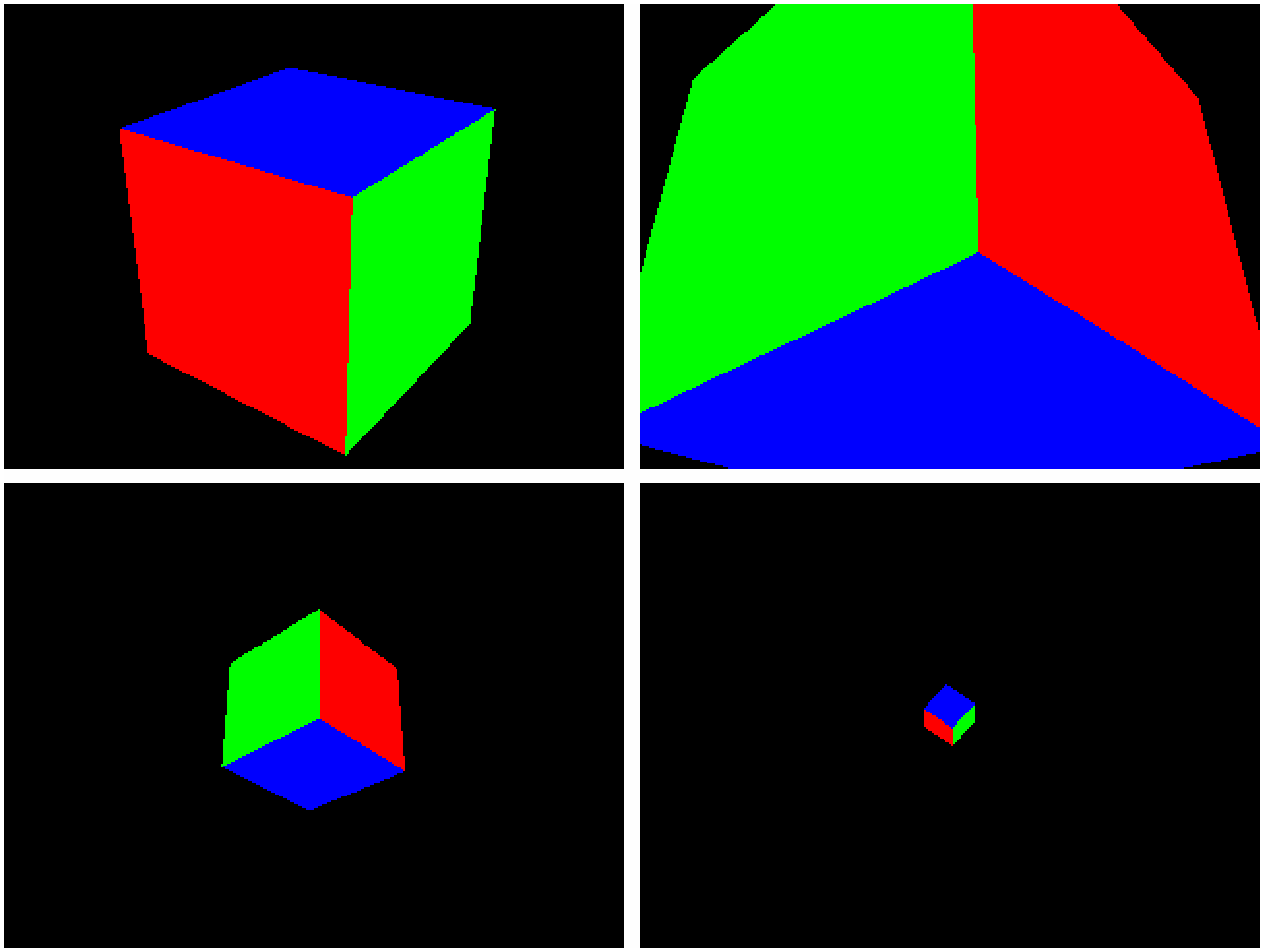

Figure 3.2 Tutorial 3 Output (Part 2)

Now you can also the cube's other faces, which are different colours as we drew them. Using the shoulder buttons you can also position the camera (Zoom in/Zoom Out).

- End of OpenGL|ES Tutorials: 3D & Keyboard Input -