OpenGL Graphics

Introduction

- OpenGL ES is a 2D and 3D graphics API specifically for embedded system.
- The android platform supports OpenGL ES.
- The OpenGL standard is now adopted on most operating systems.
- The OpenGL standard is now being managed by Khronos Group (www.khronos.org).

Introduction

- Using OpenGL ES we can draw in 3D space.
- The first step is specifying a series of points also known as vertices.
- Each one of the points has three coordinates: x, y and z.
- The points are joined together into a variety of shapes also known as primitive shapes. These shapes include points, lines and triangles.

The `glVertexPointer` Method

- We use this method to specify an array of points we want to draw. Each point has three values: x, y & z.
- The origin is at the centered of the visual display.
- The z axis becomes negative when moving into the display and positive when moving out of it. The x axis becomes positive as we move right and negative as we move left.
- These coordinates depend on the direction from which we are viewing the scene.

The `glVertexPointer` Method

- We need to convert our array of floats into an acceptable C like native buffer.

```
...
float[] vec  = {  -0.5f,-0.5f,0,
                   0.5f,-0.5f,0,
                   0.0f,0.5f,0};

java.nio.ByteBuffer byteBuffer =
    java.nio.ByteBuffer.allocateDirect(3 * 3 * 4);

byteBuffer.order(ByteOrder.nativeOrder());

java.nio.FloatBuffer floatBuffer = byteBuffer.asFloatBuffer();

floatBuffer.put(vec);

...
```

The `glVertexPointer` Method

- Now, that we have a native buffer filled in with the values of our points we can call the `glVertexPointer` method.

```
...
g.glVertexPointer( 3, // 3 coordinates
                   GL10.GL_FLOAT, // each value is float
                   0, // bytes offset between points
                   floatBuffer );
...
```

http://www.opengl.org/sdk/docs/man/xhtml/glVertexPointer.xml

The `glDrawElements` Method

- Now, that we have a native buffer filled in with the values of our points we can call the `glDrawElements` to draw them with one of the primitive shapes OpenGL ES supports.

```
...
g.glDrawElements( GL10.GL_TRIANGLES, // the shape
                  3, // number of elements to be drawn
                  GL10.GL_UNSIGENED_SHORT,
                  shortBuffer );
...
```

http://www.opengl.org/sdk/docs/man/xhtml/glDrawElements.xml

The `glDrawElements` Method

- The fourth argument is an array of indexes of the values the `glVertexPointer` function received.

Sample

```
package com.abelski.samples;

import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;

public class SimpleOpenGLActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        GLSurfaceView view = new GLSurfaceView(this);
        view.setRenderer(new OpenGLRenderer());
        setContentView(view);
    }
}
```

Sample

```
package com.abelski.samples;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import java.nio.ShortBuffer;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.opengl.GLSurfaceView.Renderer;

public class OpenGLRenderer implements Renderer
{
    public void onSurfaceCreated(GL10 gl, EGLConfig config)
    {
        //...
    }
    public void onSurfaceChanged(GL10 gl, int width, int height)
    {
        gl.glViewport(0, 0, width, height);
        gl.glClearColor(0.4f, 0.5f, 0.6f, 0.5f);
    }
```

Sample

```
public void onDrawFrame(GL10 gl)
{
    gl.glClearColor(0.4f, 0.5f, 0.6f, 0.5f);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

    ByteBuffer bufferF = ByteBuffer.allocateDirect(3 * 3 * 4);
    bufferF.order(ByteOrder.nativeOrder());
    FloatBuffer bufferFloat = bufferF.asFloatBuffer();
    float[] coords = {
            -0.5f, -0.5f, 0f, // (x1, y1, z1)
            0.5f, -0.5f, 0f, // (x2, y2, z2)
            0f, 0.5f, 0f // (x3, y3, z3)
        };
    bufferFloat.put(coords);
    bufferFloat.position(0);
```

Sample

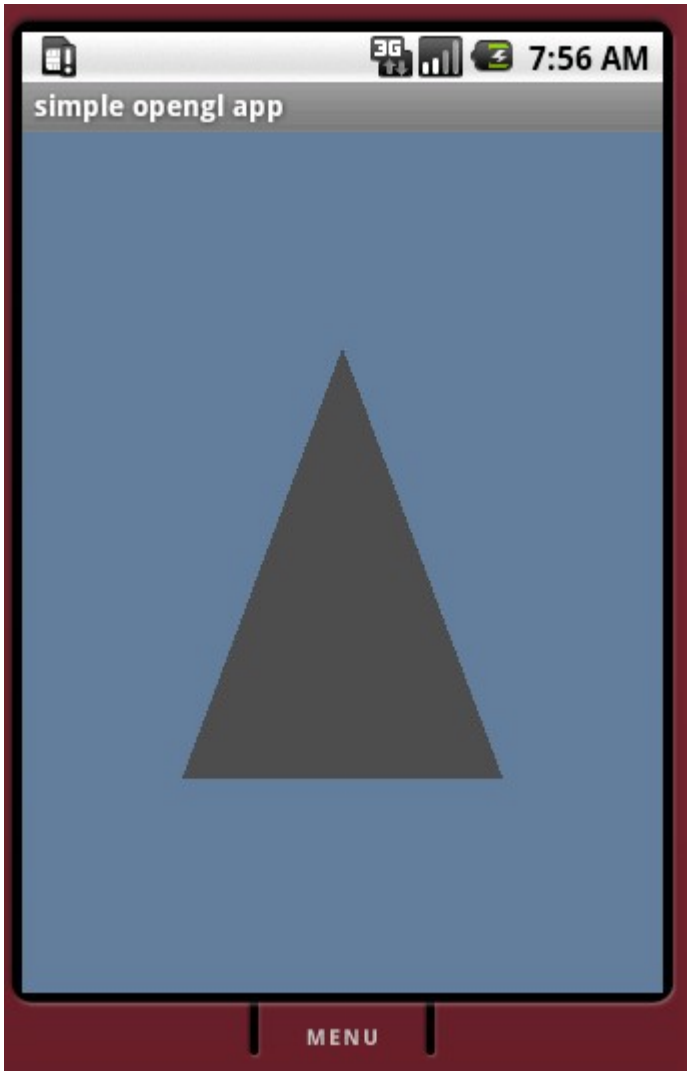
```
        ByteBuffer bufferS = ByteBuffer.allocateDirect(3*2);
        bufferS.order(ByteOrder.nativeOrder());
        ShortBuffer bufferShort = bufferS.asShortBuffer();
        bufferShort.put(new short[]{0,1,2});
        bufferShrt.position(0);

        gl.glColor4f(0.3f, 0.3f, 0.3f, 0.2f);

        gl.glVertexPointer(
            3,
            GL10.GL_FLOAT,
            0,
            bufferFloat);

        gl.glDrawElements(
            GL10.GL_TRIANGLES,
            3,
            GL10.GL_UNSIGNED_SHORT,
            bufferShort);
    }
}
```

Sample



OpenGL Graphics

Introduction

- OpenGL ES is a 2D and 3D graphics API specifically for embedded system.
- The android platform supports OpenGL ES.
- The OpenGL standard is now adopted on most operating systems.
- The OpenGL standard is now being managed by Khronos Group (www.khronos.org).

Introduction

- Using OpenGL ES we can draw in 3D space.
- The first step is specifying a series of points also known as vertices.
- Each one of the points has three coordinates: x, y and z.
- The points are joined together into a variety of shapes also known as primitive shapes. These shapes include points, lines and triangles.

The `glVertexPointer` Method

- We use this method to specify an array of points we want to draw. Each point has three values: x, y & z.
- The origin is at the centered of the visual display.
- The z axis becomes negative when moving into the display and positive when moving out of it. The x axis becomes positive as we move right and negative as we move left.
- These coordinates depend on the direction from which we are viewing the scene.

The `glVertexPointer` Method

- We need to convert our array of floats into an acceptable C like native buffer.

```
...
float[] vec  = {  -0.5f,-0.5f,0,
                   0.5f,-0.5f,0,
                   0.0f,0.5f,0};

java.nio.ByteBuffer byteBuffer =
    java.nio.ByteBuffer.allocateDirect(3 * 3 * 4);

byteBuffer.order(ByteOrder.nativeOrder());

java.nio.FloatBuffer floatBuffer = byteBuffer.asFloatBuffer();

floatBuffer.put(vec);

...
```

The `glVertexPointer` Method

- Now, that we have a native buffer filled in with the values of our points we can call the `glVertexPointer` method.

```
...
g.glVertexPointer( 3, // 3 coordinates
                   GL10.GL_FLOAT, // each value is float
                   0, // bytes offset between points
                   floatBuffer );
...
```

http://www.opengl.org/sdk/docs/man/xhtml/glVertexPointer.xml

The `glDrawElements` Method

- Now, that we have a native buffer filled in with the values of our points we can call the `glDrawElements` to draw them with one of the primitive shapes OpenGL ES supports.

```
...
g.glDrawElements( GL10.GL_TRIANGLES, // the shape
                  3, // number of elements to be drawn
                  GL10.GL_UNSIGENED_SHORT,
                  shortBuffer );
...
```

http://www.opengl.org/sdk/docs/man/xhtml/glDrawElements.xml

The `glDrawElements` Method

- The fourth argument is an array of indexes of the values the `glVertexPointer` function received.

Sample

```
package com.abelski.samples;

import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;

public class SimpleOpenGLActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        GLSurfaceView view = new GLSurfaceView(this);
        view.setRenderer(new OpenGLRenderer());
        setContentView(view);
    }
}
```

Sample

```
package com.abelski.samples;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import java.nio.ShortBuffer;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.opengl.GLSurfaceView.Renderer;

public class OpenGLRenderer implements Renderer
{
    public void onSurfaceCreated(GL10 gl, EGLConfig config)
    {
        //...
    }
    public void onSurfaceChanged(GL10 gl, int width, int height)
    {
        gl.glViewport(0, 0, width, height);
        gl.glClearColor(0.4f, 0.5f, 0.6f, 0.5f);
    }
```


Calling `glClearColor` sets the background color. Calling `glClear` cleans the color and the depth buffers in order to show the clear color we have just set.

Sample

```
public void onDrawFrame(GL10 gl)
{
    gl.glClearColor(0.4f, 0.5f, 0.6f, 0.5f);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

    ByteBuffer bufferF = ByteBuffer.allocateDirect(3 * 3 * 4);
    bufferF.order(ByteOrder.nativeOrder());
    FloatBuffer bufferFloat = bufferF.asFloatBuffer();
    float[] coords = {
            -0.5f, -0.5f, 0f, // (x1, y1, z1)
            0.5f, -0.5f, 0f, // (x2, y2, z2)
            0f, 0.5f, 0f // (x3, y3, z3)
        };
    bufferFloat.put(coords);
    bufferFloat.position(0);
```


Calling `glEnableClientState` informs the Open GL engine that we are going to use vertex arrays we want to draw.

Sample

```
            ByteBuffer bufferS = ByteBuffer.allocateDirect(3*2);
            bufferS.order(ByteOrder.nativeOrder());
            ShortBuffer bufferShort = bufferS.asShortBuffer();
            bufferShort.put(new short[]{0,1,2});
            bufferShrt.position(0);

            gl.glColor4f(0.3f, 0.3f, 0.3f, 0.2f);

            gl.glVertexPointer(
                3,
                GL10.GL_FLOAT,
                0,
                bufferFloat);

            gl.glDrawElements(
                GL10.GL_TRIANGLES,
                3,
                GL10.GL_UNSIGNED_SHORT,
                bufferShort);
        }
    }
```


Calling `glEnableClientState` informs the Open GL engine that we are going to use vertex arrays we want to draw.

Calling `gl.glColor4f(0.3f, 0.3f, 0.3f, 0.2f);` sets the color to be used in drawing the triangle.

Calling `gl.glVertexPointer` the first argument is the number of coordinates. The default value is 4. The second argument specifies the type of each argument in the array. The third argument specifies the byte offset between consecutive vertices. The default value is 0. If stride is 0, the vertices are understood to be tightly packed in the array. The fourth argument is the coordinates themselves. More info about glVertexPointer method can be found at http://www.opengl.org/sdk/docs/man/xhtml/glVertexPointer.xml.

Calling `gl.glDrawElements` the first argument is of type int. It represents the shape we want to draw. The second argument is the number of vertices. Vertices, pl of vertix, are the points at which the sides of angles intersect. The third argument specifies the type of the values the fourth argument holds. More info about the `glDrawElements` method can be found at http://www.opengl.org/sdk/docs/man/xhtml/glDrawElements.xml.

Sample