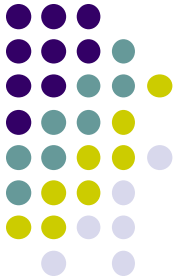
Mobile 3D Graphics

OpenGL ES

Define & Draw shapes

Define a Triangle & Square



- **In OpenGL**, the typical way to do this is to define **a vertex array** of floating point numbers for the **coordinates**.
- **Write** these **coordinates** into a **ByteBuffer**, that is passed into the **OpenGL ES graphics pipeline** for processing.

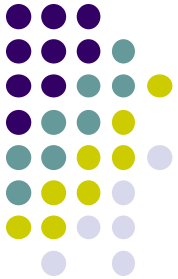
Define a triangle

```
public class Triangle {

    private FloatBuffer vertexBuffer;

    // number of coordinates per vertex in this array
    static final int COORDS_PER_VERTEX = 3;
    static float triangleCoords[] = {   // in counterclockwise order:
             0.0f,  0.622008459f, 0.0f, // top
            -0.5f, -0.311004243f, 0.0f, // bottom left
             0.5f, -0.311004243f, 0.0f  // bottom right
    };

    // Set color with red, green, blue and alpha (opacity) values
    float color[] = { 0.63671875f, 0.76953125f, 0.22265625f, 1.0f };

    public Triangle() {
        // initialize vertex byte buffer for shape coordinates
        ByteBuffer bb = ByteBuffer.allocateDirect(
                // (number of coordinate values * 4 bytes per float)
                triangleCoords.length * 4);
        // use the device hardware's native byte order
        bb.order(ByteOrder.nativeOrder());

        // create a floating point buffer from the ByteBuffer
        vertexBuffer = bb.asFloatBuffer();
        // add the coordinates to the FloatBuffer
        vertexBuffer.put(triangleCoords);
        // set the buffer to read the first coordinate
        vertexBuffer.position(0);
    }
}
```

Define a square

```
public class Square {

    private FloatBuffer vertexBuffer;
    private ShortBuffer drawListBuffer;

    // number of coordinates per vertex in this array
    static final int COORDS_PER_VERTEX = 3;
    static float squareCoords[] = {
            -0.5f,  0.5f, 0.0f,   // top left
            -0.5f, -0.5f, 0.0f,   // bottom left
             0.5f, -0.5f, 0.0f,   // bottom right
             0.5f,  0.5f, 0.0f }; // top right

    private short drawOrder[] = { 0, 1, 2, 0, 2, 3 }; // order to draw vertices

    public Square() {
        // initialize vertex byte buffer for shape coordinates
        ByteBuffer bb = ByteBuffer.allocateDirect(
        // (# of coordinate values * 4 bytes per float)
                squareCoords.length * 4);
        bb.order(ByteOrder.nativeOrder());
        vertexBuffer = bb.asFloatBuffer();
        vertexBuffer.put(squareCoords);
        vertexBuffer.position(0);

        // initialize byte buffer for the draw list
        ByteBuffer dlb = ByteBuffer.allocateDirect(
        // (# of coordinate values * 2 bytes per short)
                drawOrder.length * 2);
        dlb.order(ByteOrder.nativeOrder());
        drawListBuffer = dlb.asShortBuffer();
        drawListBuffer.put(drawOrder);
        drawListBuffer.position(0);
    }
}
```

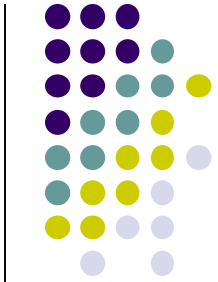
Initialize shapes



- **Before you do any drawing**, you must **initialize** and **load** the shapes you plan to draw.
- you should initialize them in the **onSurfaceCreated()** method of your **renderer** for memory and processing efficiency.

```
public class MyGLRenderer implements GLSurfaceView.Renderer {

    ...
    private Triangle mTriangle;
    private Square   mSquare;

    public void onSurfaceCreated(GL10 unused, EGLConfig config) {
        ...
        // initialize a triangle
        mTriangle = new Triangle();
        // initialize a square
        mSquare = new Square();
    }
    ...
}
```

Draw a shape



- **you must define the following**:

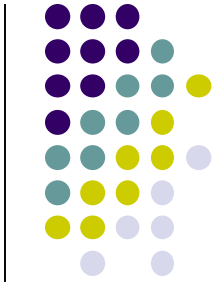
1. ***Vertex Shader*** - **OpenGL ES** graphics code for **rendering** the **vertices** of a shape.

2. ***Fragment Shader*** - **OpenGL ES** code for **rendering** the **face** of a shape with **colors** or **textures**.

3. ***Program*** - An **OpenGL ES** object that contains the **shaders** you want to use for **drawing** one or more shapes.

- These **shaders** must be **compiled** and then **added** to an OpenGL ES **program**, which is then used to draw the shape.

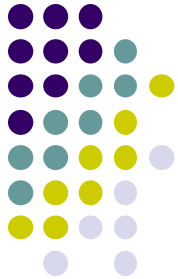
Draw a shape - define shaders



- **Example** of how to **define** basic **shaders** you can use to draw a shape in the **Triangle class**
- **Shaders** contain OpenGL Shading Language (GLSL) code

```
public class Triangle {

    private final String vertexShaderCode =
        "attribute vec4 vPosition;" +
        "void main() {" +
        "  gl_Position = vPosition;" +
        "}";

    private final String fragmentShaderCode =
        "precision mediump float;" +
        "uniform vec4 vColor;" +
        "void main() {" +
        "  gl_FragColor = vColor;" +
        "}";
    ...
}
```

Draw a shape - compiled shaders



- **Shaders** contain **OpenGL Shading Language (GLSL) code** that must be **compiled** prior to using it in the OpenGL ES environment.
- **To compile this code**,
  - create a **utility method** in your **renderer class**:

```
public static int loadShader(int type, String shaderCode){

    // create a vertex shader type (GLES20.GL_VERTEX_SHADER)
    // or a fragment shader type (GLES20.GL_FRAGMENT_SHADER)
    int shader = GLES20.glCreateShader(type);

    // add the source code to the shader and compile it
    GLES20.glShaderSource(shader, shaderCode);
    GLES20.glCompileShader(shader);

    return shader;
}
```

Draw a shape - add shaders to program object

- **In order to draw your shape**:
- you must **compile** the **shader code**,
- **add** them to a **OpenGL ES program object** and then **link the program**.
- **Do this in your drawn object's constructor**, so it is only done once.

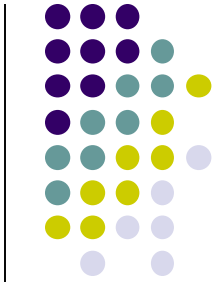
```
public class Triangle() {
    ...
    private final int mProgram;

    public Triangle() {
        ...
int vertexShader =
MyGLRenderer.loadShader(GLES20.GL_VERTEX_SHADER,
                                vertexShaderCode);
int fragmentShader =
MyGLRenderer.loadShader(GLES20.GL_FRAGMENT_SHADER,
                                fragmentShaderCode);

        // create empty OpenGL ES Program
        mProgram = GLES20.glCreateProgram();

        // add the vertex shader to program
        GLES20.glAttachShader(mProgram, vertexShader);

        // add the fragment shader to program
        GLES20.glAttachShader(mProgram, fragmentShader);

        // creates OpenGL ES program executables
        GLES20.glLinkProgram(mProgram);
    }
}
```

Draw a shape - draw()

- Create a **draw() method** for drawing the shape.

```
private int mPositionHandle;
private int mColorHandle;

private final int vertexCount = triangleCoords.length / COORDS_PER_VERTEX;
private final int vertexStride = COORDS_PER_VERTEX * 4; // 4 bytes per vertex

public void draw() {
    // Add program to OpenGL ES environment
    GLES20.glUseProgram(mProgram);

    // get handle to vertex shader's vPosition member
    mPositionHandle = GLES20.glGetAttribLocation(mProgram, "vPosition");

    // Enable a handle to the triangle vertices
    GLES20.glEnableVertexAttribArray(mPositionHandle);

    // Prepare the triangle coordinate data
    GLES20.glVertexAttribPointer(mPositionHandle, COORDS_PER_VERTEX,
                                 GLES20.GL_FLOAT, false,
                                 vertexStride, vertexBuffer);

    // get handle to fragment shader's vColor member
    mColorHandle = GLES20.glGetUniformLocation(mProgram, "vColor");

    // Set color for drawing the triangle
    GLES20.glUniform4fv(mColorHandle, 1, color, 0);

    // Draw the triangle
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);

    // Disable vertex array
    GLES20.glDisableVertexAttribArray(mPositionHandle);
}
```

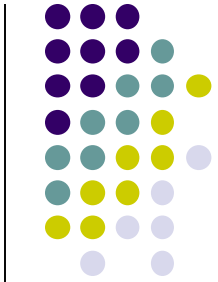
Draw a shape - onDrawFrame()



- Once you have all this code in place, drawing this object just requires a call to the **draw() method** from within your **renderer's onDrawFrame() method**:

```
public void onDrawFrame(GL10 unused) {
    ...

    mTriangle.draw();
}
```

References



- Build an OpenGL ES environment
  https://developer.android.com/training/graphics/opengl/environment
- Define shapes
  https://developer.android.com/training/graphics/opengl/shapes
- Draw shapes
  https://developer.android.com/training/graphics/opengl/draw#top_of_page
- Apply projection and camera views
  https://developer.android.com/training/graphics/opengl/projection#top_of_page