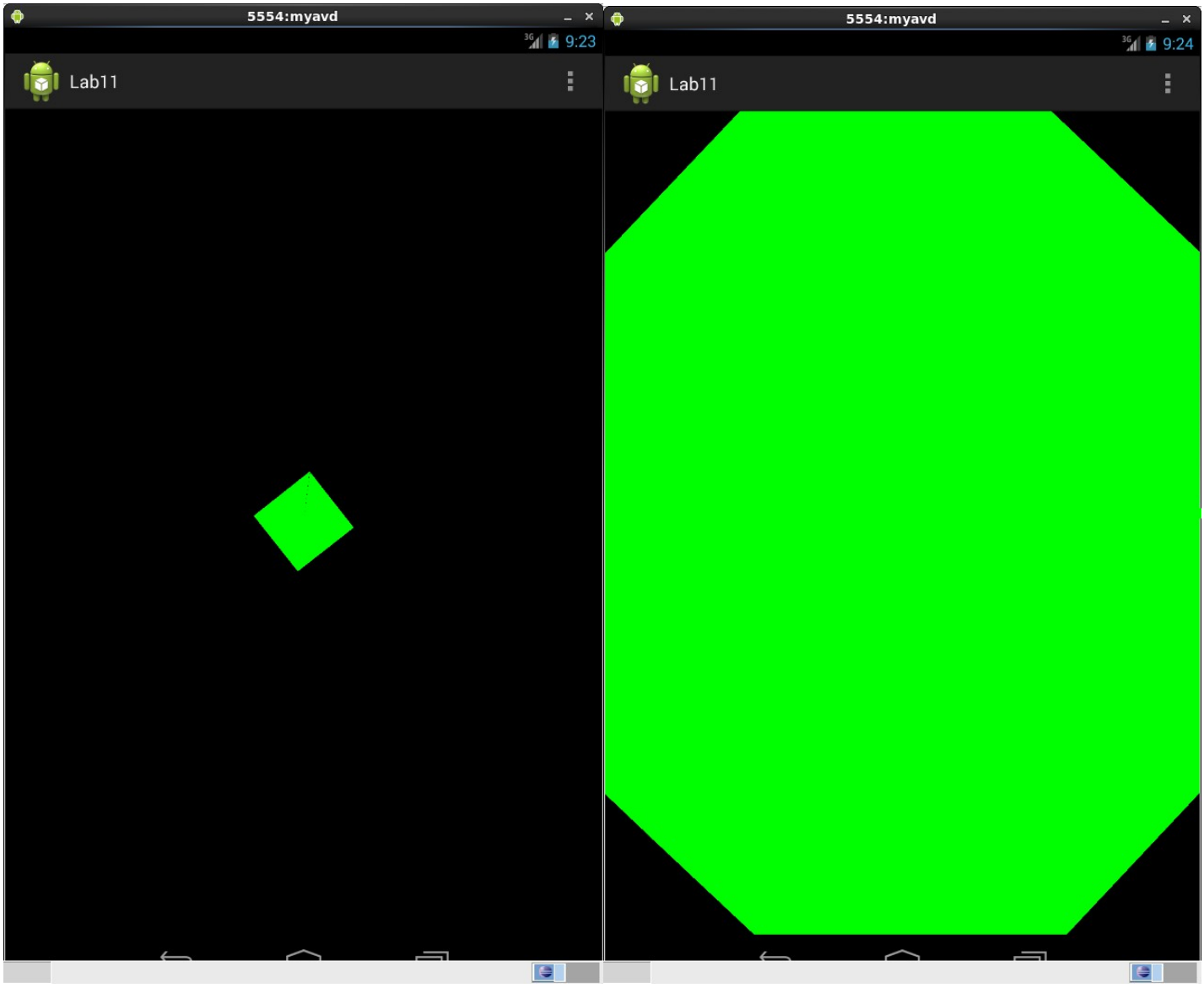
CSE520
Samuel Marrujo
Professor Yu
Lab 12

Draw a rotating and shrinking square using GLSL for Android

In this lab, we are to create a square that rotates and shrinks. I have successfully completed this task, and here are pictures of the following:



Code:

Used two same classes, with different vertices. The other class just has a different set of vertices to compile a square.

```
package opengles.lab11;

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

import android.opengl.GLES20;

public class Triangle {
  // Source code of vertex shader
  private final String vsCode =
    "attribute vec4 vPosition;" +
    "uniform mat4 uMVPMatrix;" +
        "uniform float deltaT;\n" +
    "void main(){               \n" +
    "float s = 2.0f;" +
    "s = s + 2.0 * sin ( 0.02 * deltaT );" +
    "vec4 temp = vec4 ( s, s, s, 1 );" +
             // The matrix must be included as part of gl_Position
             // Note that the uMVPMatrix factor *must be first* in order
             // for the matrix multiplication product to be correct.
    " gl_Position = uMVPMatrix * (temp * vPosition ); \n" +
    "}  \n";


   // Source code of fragment shader
   private final String fsCode =
     "precision mediump float;" +
     "uniform vec4 vColor;" +
     "void main() {" +
     "  gl_FragColor = vColor;" +
       "}";

  private int program;
  private int vertexShader;
  private int fragmentShader;
  private FloatBuffer vertexBuffer;
  private int vertexCount = 3;

   // number of coordinates per vertex in this array
   static final int COORDS_PER_VERTEX = 3;
   static float triangleCoords[] = {   // in counterclockwise order:
      0.5f,  0.5f, 0.0f, // top right vertex
     -0.5f,  -0.5f, 0.0f, // bottom left
      0.5f,  -0.5f, 0.0f  // bottom right
    };
   private float deltaT = 0.0f;
    // Set color of displaying object
    // with red, green, blue and alpha (opacity) values
    float color[] = { 0.0f, 1.0f, 0.0f, 1.0f };
```

```
    // Create a Triangle object
    Triangle(){
      // create empty OpenGL ES Program, load, attach, and link shaders
      program = GLES20.glCreateProgram();
      vertexShader = loadShader(GLES20.GL_VERTEX_SHADER, vsCode);
      fragmentShader = loadShader(GLES20.GL_FRAGMENT_SHADER, fsCode);
      GLES20.glAttachShader ( program, vertexShader );// add the vertex shader to
program
      GLES20.glAttachShader(program, fragmentShader); // add the fragment shader
to program
      GLES20.glLinkProgram(program);                  // creates OpenGL ES program
executables
      GLES20.glUseProgram( program);                  // use shader program

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

    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);

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

      return shader;
    }
    public void draw(float[] mvpMatrix) {
      int mMVPMatrixHandle = GLES20.glGetUniformLocation(program, "uMVPMatrix");
      GLES20.glUniformMatrix4fv(mMVPMatrixHandle, 1, false, mvpMatrix, 0);
      //GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);
      draw();
    }
    public void draw() {
      // Add program to OpenGL ES environment
      GLES20.glUseProgram(program);

      // get handle to vertex shader's attribute variable vPosition
      int positionHandle = GLES20.glGetAttribLocation(program, "vPosition");
      int deltaTHandle = GLES20.glGetUniformLocation(program, "deltaT");
        // Enable a handle to the triangle vertices
      GLES20.glEnableVertexAttribArray(positionHandle);

        // Prepare the triangle coordinate data
      int vertexStride = 0;
```

```
      GLES20.glVertexAttribPointer(positionHandle, COORDS_PER_VERTEX,
                                   GLES20.GL_FLOAT, false,
                                   vertexStride, vertexBuffer);

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

        // Set color for drawing the triangle
      GLES20.glUniform4fv(mColorHandle, 1, color, 0);
      GLES20.glUniform1f( deltaTHandle, deltaT);
      deltaT += 0.2;

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

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