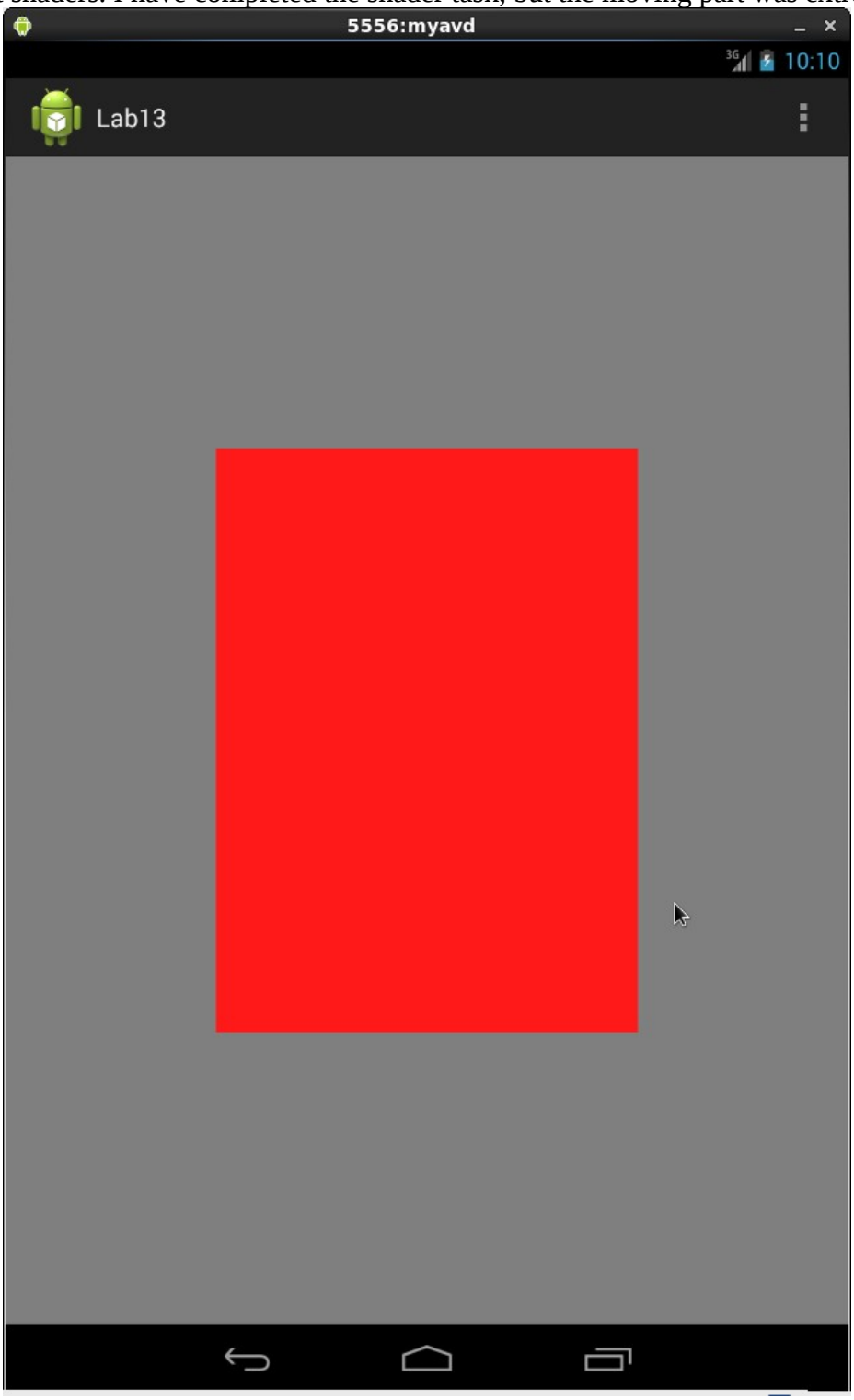
CSE520
Samuel Marrujo
Professor Yu
Lab 13

Draw a moving particle using GLSL for Android

In this lab, we are to create a square that moves across the screen, and use separate vertex/fragment shaders. I have completed the shader task, but the moving part was extremely difficult:



Code:

```
package opengl.lab13;

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

import java.io.IOException;
import java.io.InputStream;

import android.app.Activity;
import android.app.ActionBar;
import android.app.Fragment;
import android.content.Context;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.os.Build;

import android.util.Log;
import android.widget.EditText;

import android.opengl.GLES20;

public class Square {
  // Source code of vertex shader

  private static String LOG_APP_TAG = "tag";
  private String vertexshaderCode = null;
  private String fragmentshaderCode = null;

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

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

    };
    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
```

```
    Square( Context context0 ){
      // create empty OpenGL ES Program, load, attach, and link shaders
      context = context0;
      vertexshaderCode = getVertexShaderCode();
      fragmentshaderCode = getFragmentShaderCode();
      program = GLES20.glCreateProgram();
      vertexShader = loadShader(GLES20.GL_VERTEX_SHADER, vertexshaderCode);
      fragmentShader = loadShader(GLES20.GL_FRAGMENT_SHADER, fragmentshaderCode);
      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( squareCoords.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(squareCoords);
      // set the buffer to read the first coordinate
      vertexBuffer.position(0);
    } //Triangle Constructor

    protected String getVertexShaderCode()
    {
     InputStream inputStream = null;
     String str = null;
     try {
            inputStream = context.getResources().openRawResource(R.raw.vshader);
            byte[] reader = new byte[inputStream.available()];;
            while (inputStream.read(reader) != -1) {}
            str = new String ( reader );
     }  catch(IOException e) {
             Log.e(LOG_APP_TAG, e.getMessage());
         }

       return  str;
    }

    protected String getFragmentShaderCode()
    {
      InputStream inputStream = null;
     String str = null;
     try {
            inputStream = context.getResources().openRawResource(R.raw.fshader);
            byte[] reader = new byte[inputStream.available()];;
            while (inputStream.read(reader) != -1) {}
            str = new String ( reader );
     }  catch(IOException e) {
            Log.e(LOG_APP_TAG, e.getMessage());
        }
```

```
      return  str;
    }

    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
        GLES20.glVertexAttribPointer(positionHandle, COORDS_PER_VERTEX,
                                     GLES20.GL_FLOAT, false, 0, vertexBuffer);

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

        // Set color for drawing the triangle
        GLES20.glUniform4fv(colorHandle, 1, color, 0);
        GLES20.glUniform1f( deltaTHandle, deltaT);
        deltaT += 0.2;
        // Draw the triangle
        GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);
        //GLES20.glVertexAttrib3f(positionHandle,10,10,0);
        // Disable vertex array
        GLES20.glDisableVertexAttribArray(positionHandle);
    }
}
```