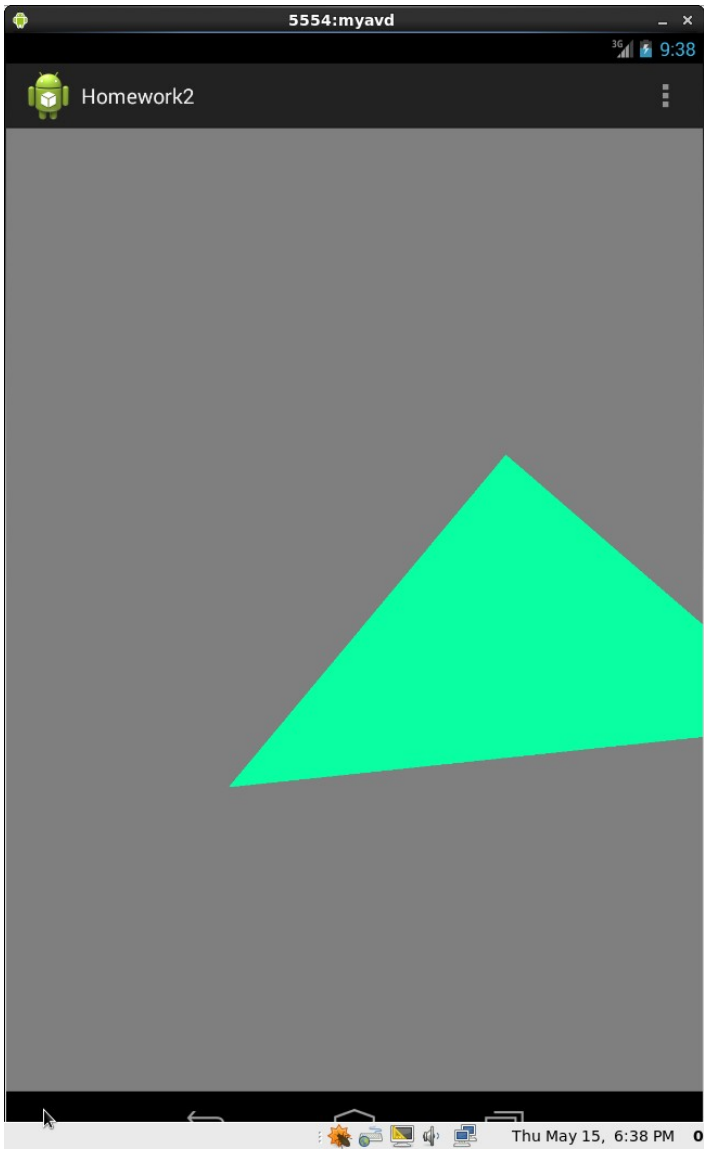
CSE520
Samuel Marrujo
Professor Yu
Homework 3

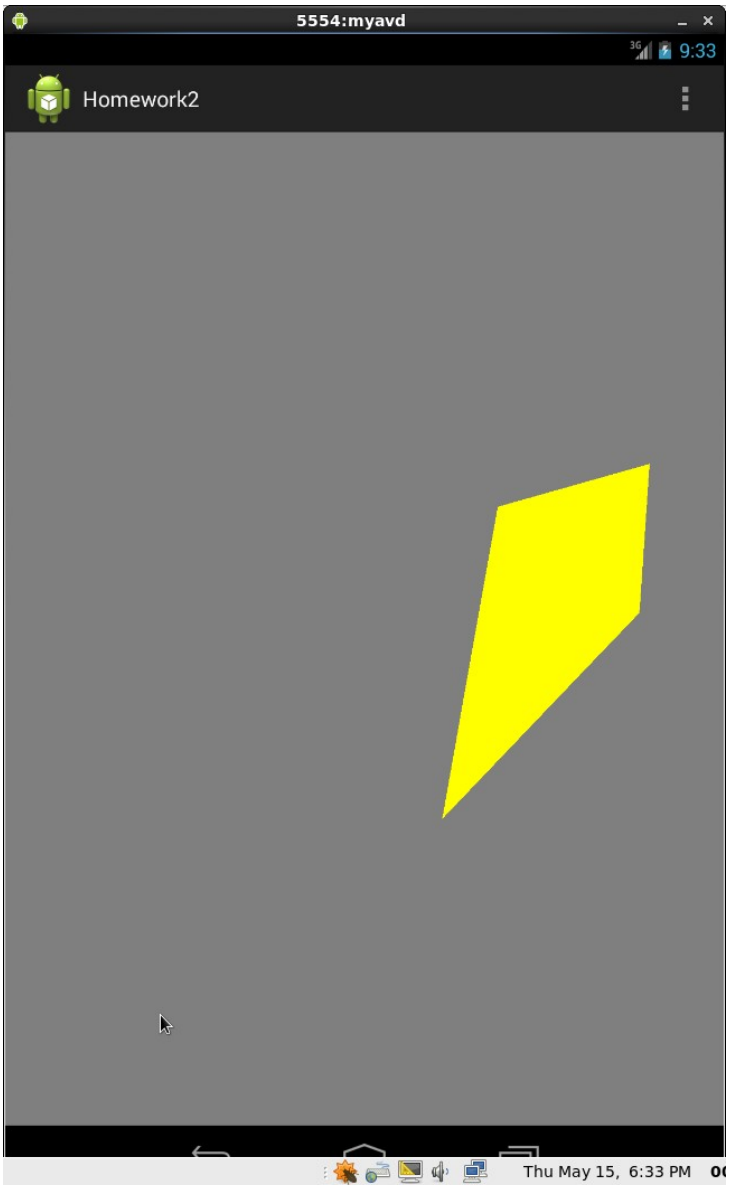
Homework 3

In this homework, we are to create a tetrahedron that rotates and changes color as time passes. Also, we are to create a cube that uses textures for each face. I have successfully completed this task to the best of my ability for the time presented, and here are pictures of the following:

Each picture will be on a separate page as shown below.
Also, I named the android project Homework 2, not realizing it is Homework 3. This is because this is the second homework we've done on Android.

#1:


5554:myavd
9:38
Homework2
Thu May 15, 6:38 PM


5554:myavd
9:33
Homework2
Thu May 15, 6:33 PM

#2:


5554:myavd
11:50
Homework3b
Thu May 15, 8:50 PM


5554:myavd
11:50
Homework3b
Thu May 15, 8:50 PM

Code:
Used 4 triangles to create a tetrahedron

#1:

```
package opengles.homework2;

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 Triangle {
  // 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 vertexBuffer1;
  private FloatBuffer vertexBuffer2;
  private FloatBuffer vertexBuffer3;
  private FloatBuffer vertexBuffer4;
  private int vertexCount = 3;
  private Context context;


   // number of coordinates per vertex in this array
   static final int COORDS_PER_VERTEX = 3;
   static float triangleCoords1[] = {   // in counterclockwise order:
      0.25f,  0.25f, 0.5f, // top right vertex
     -0.25f,  -0.25f, 0.5f, // bottom left
      0.25f,  0.0f, 1.0f  // bottom right
   };
   static float triangleCoords2[] = {   // in counterclockwise order:
      0.25f,  0.25f, 0.5f, // top right vertex
      0.75f,  -0.25f, 0.5f, // bottom left
      0.25f,  0.0f, 1.0f  // bottom right
```

```
    };
    static float triangleCoords3[] = {   // in counterclockwise order:
       0.25f,  0.0f, 1.0f, // top right vertex
      -0.25f,  -0.25f, 0.5f, // bottom left
       0.75f,   -0.25f, 0.5f  // bottom right
    };
    static float triangleCoords4[] = {   // in counterclockwise order:
       0.75f,  -0.25f, 0.5f, // top right vertex
      -0.25f,  -0.25f, 0.5f, // bottom left
       0.25f,  0.25f, 0.5f  // 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 };
     float color2[] = { 1.0f, 0.0f, 0.0f, 1.0f };

     // Create a Triangle object
     Triangle( 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 bb1 = ByteBuffer.allocateDirect( triangleCoords1.length * 4);
          bb1.order(ByteOrder.nativeOrder());
       ByteBuffer bb2 = ByteBuffer.allocateDirect( triangleCoords2.length * 4);
          bb2.order(ByteOrder.nativeOrder());
       ByteBuffer bb3 = ByteBuffer.allocateDirect( triangleCoords3.length * 4);
          bb3.order(ByteOrder.nativeOrder());
       ByteBuffer bb4 = ByteBuffer.allocateDirect( triangleCoords4.length * 4);
          bb4.order(ByteOrder.nativeOrder());
       // create a floating point buffer from the ByteBuffer
       vertexBuffer1 = bb1.asFloatBuffer();
       vertexBuffer2 = bb2.asFloatBuffer();
       vertexBuffer3 = bb3.asFloatBuffer();
       vertexBuffer4 = bb4.asFloatBuffer();
       // add the coordinates to the FloatBuffer
       vertexBuffer1.put(triangleCoords1);
       vertexBuffer2.put(triangleCoords2);
       vertexBuffer3.put(triangleCoords3);
       vertexBuffer4.put(triangleCoords4);
       // set the buffer to read the first coordinate
       vertexBuffer1.position(0);
       vertexBuffer2.position(0);
```

```
  vertexBuffer3.position(0);
  vertexBuffer4.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(vertexBuffer1);
  draw(vertexBuffer2);
  draw(vertexBuffer3);
  draw(vertexBuffer4);
}
```

```
    public void draw( final FloatBuffer coor) {
      // 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, coor);

        // get handle to fragment shader's vColor member
      int mColorHandle = GLES20.glGetUniformLocation(program, "color");
      int mColorHandle2 = GLES20.glGetUniformLocation(program, "color2");

        // Set color for drawing the triangle
      GLES20.glUniform4fv(mColorHandle, 1, color, 0);
      GLES20.glUniform4fv(mColorHandle2, 1, color2, 0);
      GLES20.glUniform1f( deltaTHandle, deltaT);
      deltaT += 0.1;


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

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

#2:

```
package opengles.homework3b;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;
/**
 *  OpenGL Custom renderer used with GLSurfaceView
 */
public class MyRenderer implements GLSurfaceView.Renderer {
   Context context;
   float scalar = 0.7f;
   float change = 0.01f;
   int bound = 0;
   private Cube cube;
   private static float angleCube = 0;
   private static float speedCube = -1.5f;
   public MyRenderer(Context context) {
      this.context = context;
      cube = new Cube();
   }
   @Override
   public void onSurfaceCreated(GL10 gl, EGLConfig config) {
      gl.glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
      gl.glClearDepthf(1.0f);
      gl.glEnable(GL10.GL_DEPTH_TEST);
      gl.glDepthFunc(GL10.GL_LEQUAL);
      gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);
      gl.glShadeModel(GL10.GL_SMOOTH);
      gl.glDisable(GL10.GL_DITHER);
      cube.loadTexture(gl, context);
      gl.glEnable(GL10.GL_TEXTURE_2D);
   }
   @Override
   public void onSurfaceChanged(GL10 gl, int width, int height) {
      if (height == 0) height = 1;
      float aspect = (float)width / height;
      gl.glViewport(0, 0, width, height);
      gl.glMatrixMode(GL10.GL_PROJECTION);
      gl.glLoadIdentity();
      GLU.gluPerspective(gl, 45, aspect, 0.1f, 100.f);

      gl.glMatrixMode(GL10.GL_MODELVIEW);
      gl.glLoadIdentity();
   }
   @Override
   public void onDrawFrame(GL10 gl) {
      gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
      gl.glLoadIdentity();
      gl.glTranslatef(0.0f, 0.0f, -6.0f);

      if ( bound == 0) {
        scalar += change;
        if (scalar > 1.2f)
              bound = 1;
      }
```

```
      else {
        scalar -= change ;
        if (scalar < 0.3f)
              bound = 0;
      }
      gl.glScalef(scalar, scalar, scalar);
      gl.glRotatef(angleCube, 1.0f, 1.0f, 1.0f);
      cube.draw(gl);
      angleCube += speedCube;
   }
}
```