

CSE520
Samuel Marujo
Professor Yu
Lab 04

Draw a rotating tetrahedron with Android

In this lab, we are to take the tetrahedron and rotate it using the Android libraries and using the Eclipse IDE. I have successfully completed this task, and here is a picture of the following:



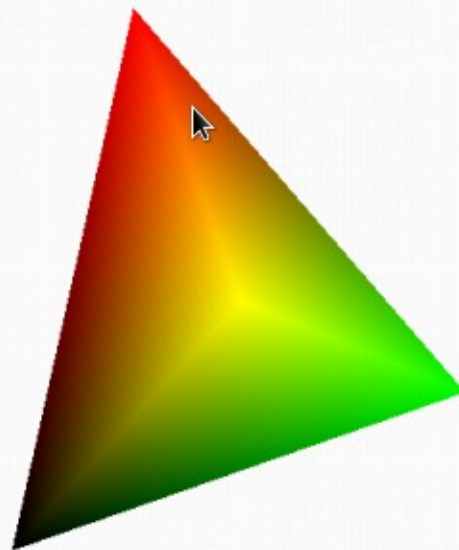
5554:newavd



3G 9:27



Lab3c



Facebook - Mozilla Fire...



003914325@jb359-18:...



Java - Lab3c/src/cs520/...

Main Activity Code:

```
package cs520.lab3c;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

import android.app.Activity;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLSurfaceView.Renderer;
import android.os.Bundle;
import android.view.MotionEvent;

/**
 * Wrapper activity demonstrating the use of {@link GLSurfaceView}, a view
 * that uses OpenGL drawing into a dedicated surface.
 *
 * Shows:
 * + How to redraw in response to user input.
 */
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Create our Preview view and set it as the content of our
        // Activity
        mGLSurfaceView = new TouchSurfaceView(this);
        setContentView(mGLSurfaceView);
        mGLSurfaceView.requestFocus();
        mGLSurfaceView.setFocusableInTouchMode(true);

        /*
        mGLSurfaceView = new GLSurfaceView(this);
        CubeRenderer1 renderer = new CubeRenderer1();
        mGLSurfaceView.setRenderer(renderer);
        setContentView(mGLSurfaceView);
        */
    }

    @Override
    protected void onResume() {
        // Ideally a game should implement onResume() and onPause()
        // to take appropriate action when the activity looses focus
        super.onResume();
        mGLSurfaceView.onResume();
    }

    @Override
    protected void onPause() {
        // Ideally a game should implement onResume() and onPause()
        // to take appropriate action when the activity looses focus
        super.onPause();
        mGLSurfaceView.onPause();
    }
}
```

```

    private GLSurfaceView mGLSurfaceView;
}

/**
 * Implement a simple rotation control.
 */
class TouchSurfaceView extends GLSurfaceView {

    public TouchSurfaceView(Context context) {
        super(context);
        mRenderer = new CubeRenderer();
        setRenderer(mRenderer);
        //RENDERMODE_WHEN_DIRTY means "do not call onDrawFrame() unless
        // something explicitly requests rendering with requestRender()
        setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
    }

    @Override public boolean onTrackballEvent(MotionEvent e) {
        mRenderer.mAngleX += e.getX() * TRACKBALL_SCALE_FACTOR;
        mRenderer.mAngleY += e.getY() * TRACKBALL_SCALE_FACTOR;
        requestRender();
        return true;
    }

    @Override public boolean onTouchEvent(MotionEvent e) {
        float x = e.getX();
        float y = e.getY();
        switch (e.getAction()) {
            case MotionEvent.ACTION_MOVE:
                float dx = x - mPreviousX;
                float dy = y - mPreviousY;
                mRenderer.mAngleX += dx * TOUCH_SCALE_FACTOR;
                mRenderer.mAngleY += dy * TOUCH_SCALE_FACTOR;
                requestRender();
            }
        mPreviousX = x;
        mPreviousY = y;
        return true;
    }

    /**
     * Render a cube.
     */
    private class CubeRenderer implements GLSurfaceView.Renderer {
        public CubeRenderer() {
            mCube = new Cube();
            mTetrahedron = new Tetrahedron();
        }

        public void onDrawFrame(GL10 gl) {
            /*
             * Usually, the first thing one might want to do is to clear
             * the screen. The most efficient way of doing this is to use
             * glClear().
             */
            gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

```

```

/*
 * Now we're ready to draw some 3D objects
 */

gl.glMatrixMode(GL10.GL_MODELVIEW);
gl.glLoadIdentity();
gl.glTranslatef(0, 0, -3.0f);
gl.glRotatef(mAngleX, 0, 1, 0);
gl.glRotatef(mAngleY, 1, 0, 0);

gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
//mCube.draw(gl);
mTetrahedron.draw(gl);
/*
mAngleX += 1.0f; //Added by Tong for testing
mAngleY += 2.0f;

gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
*/
}

public void onSurfaceChanged(GL10 gl, int width, int height) {
    gl.glViewport(0, 0, width, height);

    /*
     * Set our projection matrix. This doesn't have to be done
     * each time we draw, but usually a new projection needs to
     * be set when the viewport is resized.
     */

    float ratio = (float) width / height;
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity();
    gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
}

public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    /*
     * By default, OpenGL enables features that improve quality
     * but reduce performance. One might want to tweak that
     * especially on software renderer.
     */
    gl.glDisable(GL10.GL_DITHER);

    /*
     * Some one-time OpenGL initialization can be made here
     * probably based on features of this particular context
     */
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
              GL10.GL_FASTEST);

    gl.glClearColor(1,1,1,1);
    gl.glEnable(GL10.GL_CULL_FACE);
    gl.glShadeModel(GL10.GL_SMOOTH);
}

```

```

        gl.glEnable(GL10.GL_DEPTH_TEST);
    }
    private Cube mCube;
    private Tetrahedron mTetrahedron;
    public float mAngleX;
    public float mAngleY;
}

private final float TOUCH_SCALE_FACTOR = 180.0f / 320;
private final float TRACKBALL_SCALE_FACTOR = 36.0f;
private CubeRenderer mRenderer;
private float mPreviousX;
private float mPreviousY;
}

class CubeRenderer1 implements GLSurfaceView.Renderer
{
    GL10 gl;
    Cube cube = new Cube();
    Tetrahedron tetrahedron = new Tetrahedron();
    private float angleX;
    private float angleZ;
    private final int nfaces = 12;
    // @Override
    // Refresh automatically
    public void onDrawFrame(GL10 gl)
    {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(0.0f, 0.0f, -4.0f);
        // gl.glTranslatef(0.0f, 0.0f, -3.0f);
        gl.glRotatef( angleX, 1.0f, 0.0f, 0.0f ); // Rotate about x-axis
        gl.glRotatef( angleZ, 0.0f, 0.0f, 1.0f ); // Rotate about z-axis
        // cube.draw(gl);
        tetrahedron.draw(gl);
        angleX += 1.0f;
        angleZ += 2.0f;
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
    }

    public void onSurfaceChanged(GL10 gl, int width, int height)
    {
        gl.glViewport(0, 0, width, height);
        float ratio = (float) width / height;
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
    }

    public void onSurfaceCreated(GL10 gl, EGLConfig config)
    {
        gl.glDisable(GL10.GL_DITHER);
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_FASTEST);
    }
}

```

```

        gl.glClearColor(1, 1, 1, 0);
        gl.glEnable(GL10.GL_CULL_FACE);
        gl.glShadeModel(GL10.GL_SMOOTH);
        gl.glEnable(GL10.GL_DEPTH_TEST);
    }
}

```

Renderer Code:

```

package cs520.lab3c;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.IntBuffer;

import javax.microedition.khronos.opengles.GL10;

/**
 * A vertex shaded cube.
 */
class Cube
{
    public Cube()
    {
        int one = 0x10000;
        int vertices[] = {
            -one, -one, -one,
            one, -one, -one,
            one, one, -one,
            -one, one, -one,
            -one, -one, one,
            one, -one, one,
            one, one, one,
            -one, one, one,
        };

        int colors[] = {
            0, 0, 0, one,
            one, 0, 0, one,
            one, one, 0, one,
            0, one, 0, one,
            0, 0, one, one,
            one, 0, one, one,
            one, one, one, one,
            0, one, one, one,
        };

        byte indices[] = {
            0, 4, 5, 0, 5, 1,
            1, 5, 6, 1, 6, 2,
            2, 6, 7, 2, 7, 3,
            3, 7, 4, 3, 4, 0,
            4, 7, 6, 4, 6, 5,
            3, 0, 1, 3, 1, 2
        };
    }
}

```

```

};

// Buffers to be passed to gl*Pointer() functions
// must be direct, i.e., they must be placed on the
// native heap where the garbage collector cannot
// move them.
//
// Buffers with multi-byte datatypes (e.g., short, int, float)
// must have their byte order set to native order

ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
vbb.order(ByteOrder.nativeOrder());
mVertexBuffer = vbb.asIntBuffer();
mVertexBuffer.put(vertices);
mVertexBuffer.position(0);

ByteBuffer cbb = ByteBuffer.allocateDirect(colors.length*4);
cbb.order(ByteOrder.nativeOrder());
mColorBuffer = cbb.asIntBuffer();
mColorBuffer.put(colors);
mColorBuffer.position(0);

mIndexBuffer = ByteBuffer.allocateDirect(indices.length);
mIndexBuffer.put(indices);
mIndexBuffer.position(0);
}

public void draw(GL10 gl)
{
    gl.glFrontFace(GL10.GL_CW);
    gl.glVertexPointer(3, GL10.GL_FIXED, 0, mVertexBuffer);
    gl.glColorPointer(4, GL10.GL_FIXED, 0, mColorBuffer);
    gl.glDrawElements(GL10.GL_TRIANGLES, 36, GL10.GL_UNSIGNED_BYTE,
mIndexBuffer);
}

private IntBuffer    mVertexBuffer;
private IntBuffer    mColorBuffer;
private ByteBuffer   mIndexBuffer;
}

class Tetrahedron
{
    public Tetrahedron()
    {
        int one = 0x10000;
        int vertices[] = {
            0, one, 0,
            0, -one, 0,
            one, 0, 0,
            one, 0, -one
        };

        int colors[] = {
            0,    0,    0,    one,
            one,    0,    0,    one,
            one,    one,    0,    one,
            0,    one,    0,    one,
        };
    }
}

```



```

};

byte indices[] = {
    1, 3, 2,
    0, 1, 2,
    0, 2, 3,
    0, 3, 1
};

// Buffers to be passed to gl*Pointer() functions
// must be direct, i.e., they must be placed on the
// native heap where the garbage collector cannot
// move them.
//
// Buffers with multi-byte datatypes (e.g., short, int, float)
// must have their byte order set to native order

ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
vbb.order(ByteOrder.nativeOrder());
mVertexBuffer = vbb.asIntBuffer();
mVertexBuffer.put(vertices);
mVertexBuffer.position(0);

ByteBuffer cbb = ByteBuffer.allocateDirect(colors.length*4);
cbb.order(ByteOrder.nativeOrder());
mColorBuffer = cbb.asIntBuffer();
mColorBuffer.put(colors);
mColorBuffer.position(0);

mIndexBuffer = ByteBuffer.allocateDirect(indices.length);
mIndexBuffer.put(indices);
mIndexBuffer.position(0);
}

public void draw(GL10 gl)
{
    gl.glFrontFace(GL10.GL_CW);
    gl.glVertexPointer(3, GL10.GL_FIXED, 0, mVertexBuffer);
    gl.glColorPointer(4, GL10.GL_FIXED, 0, mColorBuffer);
    gl.glDrawElements(GL10.GL_TRIANGLES, 12, GL10.GL_UNSIGNED_BYTE,
mIndexBuffer);
}

private IntBuffer    mVertexBuffer;
private IntBuffer    mColorBuffer;
private ByteBuffer   mIndexBuffer;
}

```