Homework 1, due 1/27/2015 (Tue)

1. (40 points)
   Write an Android graphics program using OpenGL ES 1X that renders a colored 3D polyhedron for each of the following polyhedrons
   1. a cube
   2. a tetrahedron
   3. a dodecahedron
   4. an icosahedron
   as shown in the figure below. Each program allows the user to rotate it by dragging the mouse. Then modify the program so that the object can rotate along the x, y, and z axis automatically.

Part 1: a cube

Instructions on how to render a cube in Android were included in the first set of lecture notes, so I utilized these in order to achieve this goal. I also added my rotation code (seen at the bottom) in order to give it the required rotations.
class Cube {
    private FloatBuffer vertexBuffer;
    private FloatBuffer colorBuffer;
    private ByteBuffer indexBuffer;
    // Coordinates of 8 vertices of 6 cube faces
    private float vertices[] = {
        -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
        -1.0f, 1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f,
    };
    // Colors of vertices
    private float colors[] = {
        0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f,
        1.0f, 0.5f, 0.0f, 1.0f, 1.0f, 0.5f, 0.0f, 1.0f,
        0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
        0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,
    };
    // indices of 12 triangles (6 squares) in GL_CCW
    // referencing vertices[] array coordinates
    private byte indices[] = {
        5, 4, 0, 1, 5, 0, 6, 5, 1, 2, 6, 1, 7, 6, 2, 3,
        7, 2, 4, 7, 3, 0, 4, 3, 6, 7, 4, 5, 6, 4, 1, 0, 3, 2, 1, 3,
    };

    public Cube() {
        // initialize vertex Buffer for cube
        // argument=(# of coordinate values*4 bytes per float)
        ByteBuffer byteBuf = ByteBuffer.allocateDirect(vertices.length * 4);
        byteBuf.order(ByteOrder.nativeOrder());
        // create a floating point buffer from the ByteBuffer
        vertexBuffer = byteBuf.asFloatBuffer();
        // add the vertices coordinates to the FloatBuffer
        vertexBuffer.put(vertices);
        // set the buffer to read the first vertex coordinates
        vertexBuffer.position(0);
        // Do the same to colors array
        byteBuf = ByteBuffer.allocateDirect(colors.length * 4);
        byteBuf.order(ByteOrder.nativeOrder());
        colorBuffer = byteBuf.asFloatBuffer();
        colorBuffer.put(colors);
        colorBuffer.position(0);
        // indices are integers
        indexBuffer = ByteBuffer.allocateDirect(indices.length);
        indexBuffer.put(indices);
        indexBuffer.position(0);
    }
}
Part 2: a tetrahedron

I took the code from my attempt at the dodecahedron and altered the number of indices/vertexes to work for a tetrahedron. I added the rotation code (seen at the bottom of this report) in order to produce the same rotations as the other models.

Relevant Code

```java
private ByteBuffer[] faceIndexBuffer = new ByteBuffer[4]; // buffer holding faces
private FloatBuffer[] colorBuffer = new FloatBuffer[4]; // buffer holding colors
private final int nfaces = 4;

float vertices[] = new float[] {
    1.0f, 1.0f, 1.0f, // V0
    -1.0f, 1.0f, -1.0f, // V1
    1.0f, -1.0f, -1.0f, // V2
    -1.0f, -1.0f, 1.0f // V3
};

//Faces
byte faceIndices[][] = {
    {1, 2, 3}, // F0
    {0, 3, 2}, // F1
    {0, 1, 3}, // F2
    {0, 2, 1}  // F3
};
```
Part 3: Dodecahedron

The dodecahedron was where I started this homework assignment, and what I used as the base for all the other shapes (excluding cube). I created buffers to hold the vertices and the indices, declared the vertices and indices, and then placed them into the buffers.

Relevant Code

```java
public class Dodecahedron {
    private FloatBuffer vertexBuffer; // buffer holding vertices
    private ByteBuffer[] faceIndexBuffer = new ByteBuffer[12]; // buffer holding faces
    private FloatBuffer[] colorBuffer = new FloatBuffer[12]; // buffer holding colors

    private final int nfacs = 12; // number of faces in object

    // Vertices
    float vertices[] = new float[] {
        0.0f, 0.847214f, 0.323607f,
        0.0f, 0.847214f, -0.323607f,
        -0.523607f, 0.523607f, 0.523607f,
        -0.523607f, 0.523607f, -0.523607f,
        -0.847214f, 0.323607f, -0.0f,
        0.523607f, 0.523607f, -0.523607f,
        0.523607f, 0.523607f, 0.523607f,
        0.847214f, 0.323607f, -0.0f,
    };
```
public Dodecahedron() {
    // a float has 4 bytes so we allocate for each coordinate 4 bytes
    ByteBuffer byteBuffer = ByteBuffer.allocateDirect(this.vertices.length * 4);
    byteBuffer.order(ByteOrder.nativeOrder());
    vertexBuffer = byteBuffer.asFloatBuffer();
    vertexBuffer.put(vertices);
    vertexBuffer.position(0);
    for (int i = 0; i < nfaces; i++) {
        faceIndexBuffer[i] = ByteBuffer.allocateDirect(this.faceIndices[i].length);
        faceIndexBuffer[i].put(faceIndices[i]);
        faceIndexBuffer[i].position(0);
    }
}

public void draw(GL10 gl) {
    gl.glFrontFace(GL10.GL_CW);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
    for (int i = 0; i < nfaces; i++) {
        setColor ( gl, i );
        gl.glDrawElements(GL10.GL_TRIANGLES, faceIndexBuffer[i].limit(),
                          GL10.GL_UNSIGNED_BYTE, faceIndexBuffer[i]);
    }
}
Part 4: icosahedron

For the icosahedron, I used all of the code that I had for the dodecahedron, I simply changed the indices and vertices. This required a lot more trial and error than I would like to admit, but I very much like the end results.

Relevant Code

```java
private ByteBuffer[] faceIndexBuffer = new ByteBuffer[20]; // buffer holding faces
private FloatBuffer[] colorBuffer = new FloatBuffer[20]; // buffer holding colors
private final int nfaces = 20; //number of faces in object

private static float X = 0.525731112119133606f;
private static float Z = 0.850650808352039932f;

//Vertices
float vertices[] = new float[] {
-`X`,0.0f,`Z`, `X`,0.0f,`Z`, `-X`,0.0f,-`Z`, `X`,0.0f,-`Z`,
0.0f,`Z`,`X`, 0.0f,`Z`,-`X`, 0.0f,-`Z`,`X`, 0.0f,-`Z`,-`X`,
`X`,0.0f,`Z`, -`X`,0.0f,`Z`, `Z`,0.0f,-`Z`, -`Z`,0.0f,-`Z`, -`Z`,0.0f
};

//Faces
byte faceIndices[][] = {
{1,4,0}, {4,9,0}, {4,5,9}, {8,5,4}, {1,8,4},
{1,10,8}, {10,3,8}, {8,3,5}, {3,2,5}, {3,7,2},
{3,10,7}, {10,6,7}, {6,11,7}, {6,0,11}, {6,1,0},
};
```
Rotation:

This lab required two types of rotation; on mouse drag and on its own. The following code was used to provide those effects:

On-mouse drag

```java
public boolean onTouchEvent(MotionEvent e) {
    // MotionEvent reports input details from the touch screen
    // and other input controls. Here, we are only interested
    // in events where the touch position has changed.
    float x = e.getX();
    float y = e.getY();

    switch (e.getAction()) {
        case MotionEvent.ACTION_MOVE:
            float dx = x - previousX;
            float dy = y - previousY;
            // reverse direction of rotation above the mid-line
            if (y > getHeight() / 2)
                dx = dx * -1;
            // reverse direction of rotation to left of the mid-line
            if (x < getWidth() / 2)
                dy = dy * -1;
            renderer.angle += (dx + dy) * TOUCH_SCALE_FACTOR;
            requestRender();
            break;
    }

    previousX = x;
    previousY = y;
    return true;
}
```

Self rotation

```java
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, -3.0f);
    gl.glRotatef(anglex, 1.0f, 0.0f, 0.0f); // Rotate about x-axis
    gl.glRotatef(anelz, 0.0f, 0.0f, 1.0f); // Rotate about z-axis
    dodecahedron.draw(gl);
    anglex += 1.0f;
}```
angley += 1.5f;
anglez += 2.0f;
    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
}

Evaluation:

I believe that I completed every part of this homework assignment. Being able to use the same code for three out of the four shapes helped tremendously, and I learned a lot about how to manually map shapes out through their vertices. I tried to create a cube using the same base code but couldn't get it to render properly. This is definitely something that I will be messing with in the future just to see what I can create. I really enjoyed this project, and feel much more knowledgable in using OpenGL on android upon completion.

I am giving myself the full 40 points for this assignment. I did not attempt the extra credit, but I think I will try it in the future just to see if I can do it.