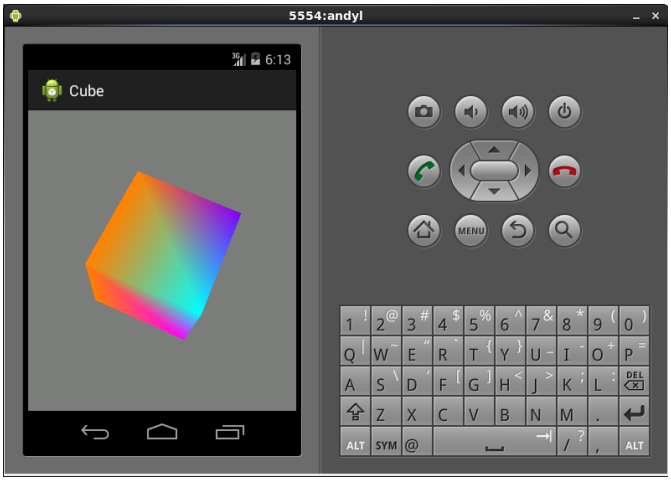
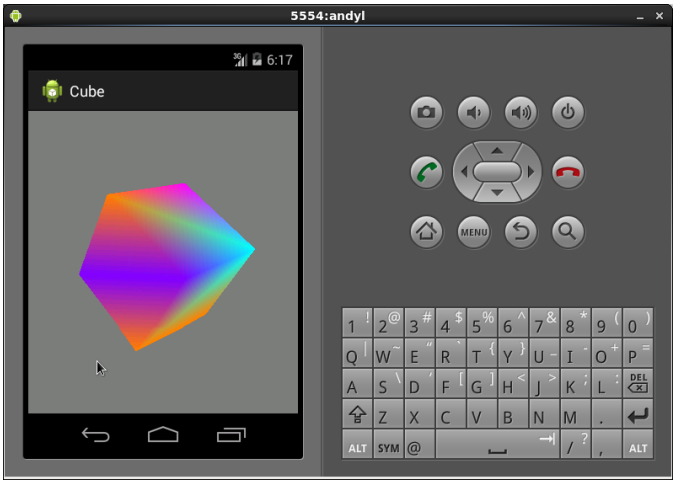
Yazhuo Liu
Homework 1

Cube:





Code:

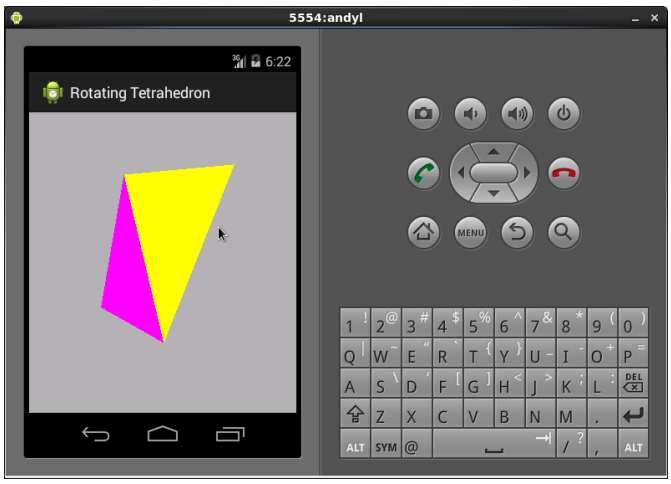
```
//CubeRenderer.java
public class CubeRenderer implements GLSurfaceView.Renderer {
        public float angleX = 0.0f; //rotation angle
        public float angleZ = 0.0f;
        private Cube cube = new Cube();
        public void onSurfaceCreated(GL10 gl, EGLConfig config) {
                // Set the background frame color to grey, opaque
                gl.glClearColor(0.7f, 0.7f, 0.7f, 1.0f);
                gl.glEnable( GL10.GL_CULL_FACE ); //Enable culling faces
                gl.glCullFace ( GL10.GL_BACK ); //don't render back faces
        }

        public void onDrawFrame(GL10 gl) {
                // Redraw background color
                gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
                // Set GL_MODELVIEW transformation mode
                gl.glMatrixMode(GL10.GL_MODELVIEW);
                gl.glLoadIdentity(); // Reset the matrix to identity matrix
                // Move objects away from view point to observe
                gl.glTranslatef(0.0f, 0.0f, -7.0f);
```

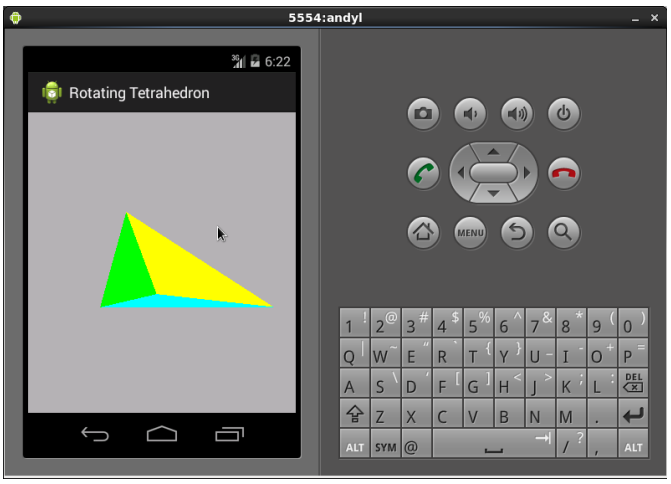
```
            // Rotate about a diagonal of cube
            SystemClock.sleep(10);
            angleX -= 2;
            angleZ -= 1;
            gl.glRotatef(angleX, 1, 0, 0);
            gl.glRotatef(angleZ, 0, 0, 1);
            cube.draw(gl); // Draw the cube
            gl.glLoadIdentity(); // Reset transformation matrix
        }

        @Override
        public void onSurfaceChanged(GL10 gl, int width, int height) {
            gl.glViewport(0, 0, width, height);
            gl.glMatrixMode(GL10.GL_PROJECTION);
            gl.glLoadIdentity(); // Reset projection matrix
            // Setup viewing volume
            GLU.gluPerspective(gl,45.0f,(float)width/(float)height,0.1f,100.0f);
            gl.glViewport(0, 0, width, height);
            gl.glMatrixMode(GL10.GL_MODELVIEW);
            gl.glLoadIdentity(); // Reset transformation matrix
        }
}

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, -1.0f, 1.0f,
                        1.0f, 1.0f, 1.0f,                -1.0f, 1.0f, 1.0f
```

```
};

// Colors of vertices
private float colors[] = {
                0.5f, 0.0f, 1.0f, 1.0f,          1.0f, 0.5f, 0.0f, 1.0f,
                1.0f, 0.5f, 0.0f, 1.0f,          1.0f, 0.5f, 0.0f, 1.0f,
                1.0f, 0.5f, 0.0f, 1.0f,  0.5f, 0.0f, 1.0f, 1.0f,
                0.0f, 1.0f, 1.0f, 1.0f,  1.0f, 0.0f, 1.0f, 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);
        }

        //Typical drawing routine using vertex array
        public void draw(GL10 gl) {
                //Counterclockwise order for front face vertices
                gl.glFrontFace(GL10.GL_CCW);
                //Points to the vertex buffers
                gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
                gl.glColorPointer(4, GL10.GL_FLOAT, 0, colorBuffer);
                //Enable client states
                gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
                gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
                //Draw vertices as triangles
                gl.glDrawElements(GL10.GL_TRIANGLES, 36, GL10.GL_UNSIGNED_BYTE,
indexBuffer);
                //Disable client state
                gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
                gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
        }
}
```

Tetrahedron:





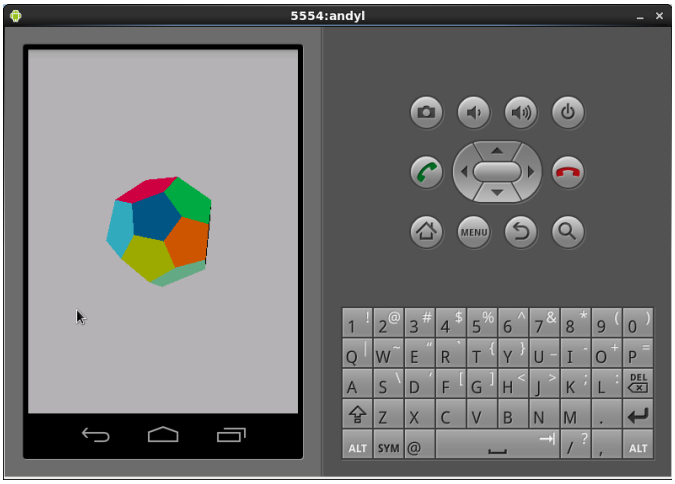
Code:

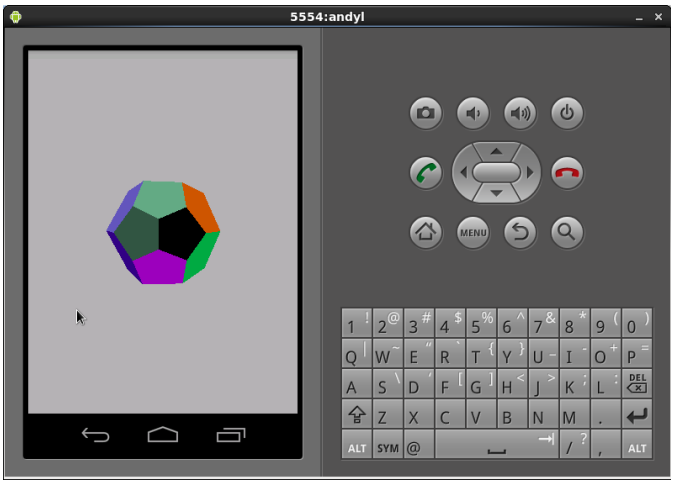
//TetraRenderer.java

```
public class TetraRenderer implements GLSurfaceView.Renderer {

      private FloatBuffer triangle;
      private FloatBuffer triangle2;
      private FloatBuffer triangle3;
      private FloatBuffer triangle4;

      public float angle = 0.0f;

    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // Set the background frame color to blue
        gl.glClearColor(0.7f, 0.7f, 0.7f, 1.0f);
        gl.glEnable( GL10.GL_CULL_FACE ); //Enable culling faces
        gl.glCullFace ( GL10.GL_BACK ); //don't render back faces
        // initialize the triangle vertex array
        initShapes();
        // Enable use of vertex arrays
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    }

    public void onDrawFrame(GL10 gl) {
        // Redraw background color
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        //gl.glRotatef(-50, 1, 0, 0);

        GLU.gluLookAt(gl, -4, -4, 5, 0.2f, 0.2f, 0f, 0f, 1.0f, 0.0f);
        SystemClock.sleep ( 20 );
        angle += 2;
        //rotate triangle
        gl.glRotatef(angle, 0, 0, 1);
        //magnify triangle
        gl.glScalef ( 1, 0.8f, 0.8f);

        // Draw the triangle
        gl.glColor4f(1.0f, 1.0f, 0.0f, 0.0f);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangle);
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
        gl.glColor4f(1.0f, 0.0f, 1.0f, 0.0f);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangle2);
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
        gl.glColor4f(0.0f, 1.0f, 0.0f, 0.0f);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangle3);
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
        gl.glColor4f(0.0f, 1.0f, 1.0f, 0.0f);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangle4);
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);

    }

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

```
    private void initShapes(){

      float vertices_1[] = {
                  -0.6f, -0.6f, 0.85f,
                  0.6f, -0.6f, 0.85f,
                  0.0f, 0.6f, 0
      };

      float vertices_2[] = {
                  0.0f, 0.6f, 0,
                  0.6f, -0.6f, 0.85f,
                  0, 0, -0.6f,
      };

      float vertices_3[] = {
                  0, 0.6f, 0,
                  0, 0, -0.6f,
                  -0.6f, -0.6f, 0.85f
      };

      float vertices_4[] = {
                  0, 0, -0.6f,
                  0.6f, -0.6f, 0.85f,
                  -0.6f, -0.6f, 0.85f,
      };

        // initialize vertex Buffer for triangle
        ByteBuffer v1 = ByteBuffer.allocateDirect(vertices_1.length * 4);
        v1.order(ByteOrder.nativeOrder());
        triangle = v1.asFloatBuffer();
        triangle.put(vertices_1);
        triangle.position(0);

        ByteBuffer v2 = ByteBuffer.allocateDirect(vertices_2.length * 4);
        v2.order(ByteOrder.nativeOrder());
        triangle2 = v2.asFloatBuffer();
        triangle2.put(vertices_2);
        triangle2.position(0);

        ByteBuffer v3 = ByteBuffer.allocateDirect(vertices_3.length * 4);
        v3.order(ByteOrder.nativeOrder());
        triangle3 = v3.asFloatBuffer();
        triangle3.put(vertices_3);
        triangle3.position(0);

        ByteBuffer v4 = ByteBuffer.allocateDirect(vertices_4.length * 4);
        v4.order(ByteOrder.nativeOrder());
        triangle4 = v4.asFloatBuffer();
        triangle4.put(vertices_4);
        triangle4.position(0);
    }
}
```

Dodecahedron:





//DodecahedronRenderer.java

```
public class DodecahedronRenderer implements Renderer
{
  GL10 gl;
  Dodecahedron dodecahedron = new Dodecahedron();
  public float anglex = 0;
  public float anglez = 0;

  //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, -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
      dodecahedron.draw(gl);
      SystemClock.sleep(10);
      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);
```

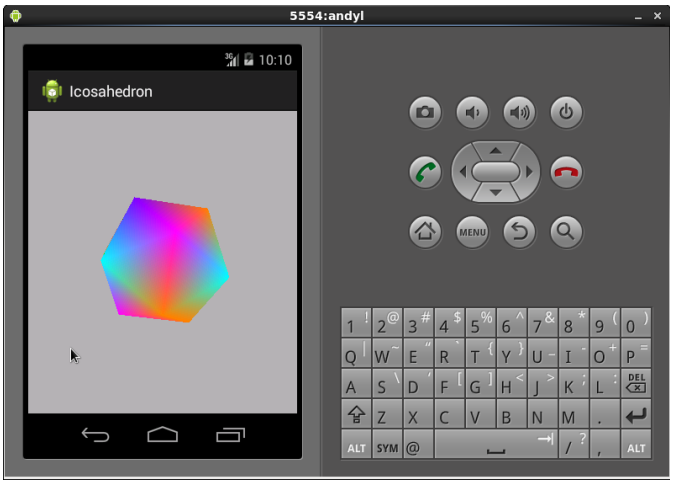
```
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_FASTEST);
        gl.glClearColor(0.7f, 0.7f, 0.7f, 1.0f);
        gl.glEnable(GL10.GL_CULL_FACE);
        gl.glShadeModel(GL10.GL_SMOOTH);
        gl.glEnable(GL10.GL_DEPTH_TEST);
    }
}
```

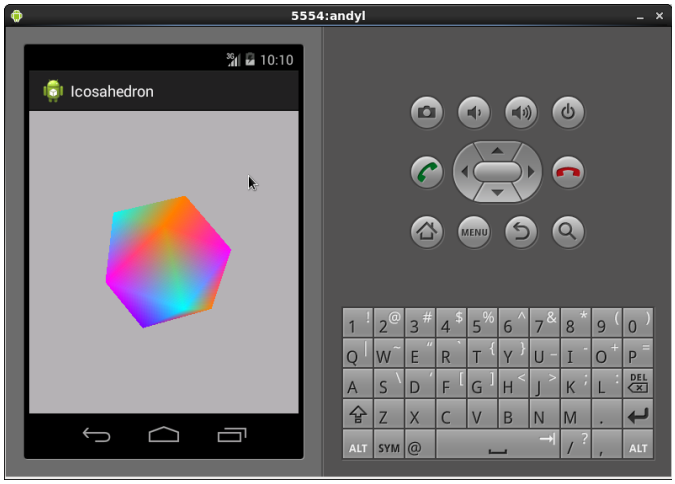
//Dodecahedron.java

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

  private final int nfaces = 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,
              0.323607f,-0.0f,-0.847214f,
              -0.323607f,-0.0f,-0.847214f,
              -0.323607f,0.0f,0.847214f,
              0.323607f,0.0f,0.847214f,
              -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,
              0.523607f,-0.523607f,-0.523607f,
              0.0f,-0.847214f,-0.323607f,
              0.0f,-0.847214f,0.323607f,
  };

  //Faces
  byte faceIndices[][] = {
              {0,1,2, 1,3,2, 3,4,2},
              {1,0,5, 0,6,5, 6,7,5},
              {1,5,3, 5,8,3, 8,9,3},
              {0,2,6, 2,10,6, 10,11,6},
              {3,9,4, 9,12,4, 12,13,4},
              {2,4,10, 4,13,10, 13,14,10},
              {6,11,7, 11,15,7, 15,16,7},
              {5,7,8, 7,16,8, 16,17,8},
              {9,8,12, 8,17,12, 17,18,12},
              {11,10,15, 10,14,15, 14,19,15},
              {13,12,14, 12,18,14, 18,19,14},
              {16,15,17, 15,19,17, 19,18,17}
  };

  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);
      }
  }

  private void setColor ( GL10 gl,  int i )
  {
     float R = (float) (i % 5) / 5;
     float G = (float) (i % 3) / 3;
     float B = (float) (i % 4) / 4;
     gl.glColor4f( R, G, B, 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]);
    }
  }
}
```

Icosahedron:





//IcoRenderer.java

```
public class IcoRenderer implements GLSurfaceView.Renderer {
      public float angleX = 0.0f; //rotation angle
      public float angleZ = 0.0f;
      private Ico ico = new Ico();
```

```
	public void onSurfaceCreated(GL10 gl, EGLConfig config) {
		// Set the background frame color to grey, opaque
		gl.glClearColor(0.7f, 0.7f, 0.7f, 1.0f);
		gl.glEnable( GL10.GL_CULL_FACE ); //Enable culling faces
		gl.glCullFace ( GL10.GL_BACK ); //don't render back faces
	}

	public void onDrawFrame(GL10 gl) {
		// Redraw background color
		gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
		// Set GL_MODELVIEW transformation mode
		gl.glMatrixMode(GL10.GL_MODELVIEW);
		gl.glLoadIdentity(); // Reset the matrix to identity matrix
		// Move objects away from view point to observe
		gl.glTranslatef(0.0f, 0.0f, -5.0f);
		// Rotate about a diagonal of cube
		SystemClock.sleep(10);
		angleX -= 1;
		angleZ -= 2;
		gl.glRotatef(angleX, 1, 0, 0);
		gl.glRotatef(angleZ, 0, 0, 1);
		ico.draw(gl); // Draw the cube
		gl.glLoadIdentity(); // Reset transformation matrix
	}

	@Override
	public void onSurfaceChanged(GL10 gl, int width, int height) {
		gl.glViewport(0, 0, width, height);
		gl.glMatrixMode(GL10.GL_PROJECTION);
		gl.glLoadIdentity(); // Reset projection matrix
		// Setup viewing volume
		GLU.gluPerspective(gl,45.0f,(float)width/(float)height,0.1f,100.0f);
		gl.glViewport(0, 0, width, height);
		gl.glMatrixMode(GL10.GL_MODELVIEW);
		gl.glLoadIdentity(); // Reset transformation matrix
	}
}

class Ico {
	private FloatBuffer vertexBuffer;
	private FloatBuffer colorBuffer;
	private ByteBuffer indexBuffer;

	// Coordinates of 8 vertices of 6 cube faces
	private float vertices[] = {
			0.0f, 0.0f, 0.951057f,
			0.0f, 0.850651f, 0.425325f,
			0.809017f, 0.262866f, 0.425325f,
			0.5f, -0.688191f, 0.425325f,
			-0.5f, -0.688191f, 0.425325f,
			-0.809017f, 0.262866f, 0.425325f,
			0.0f, -0.850651f, -0.425325f,
			-0.809017f, -0.262866f, -0.425325f,
			-0.5f, 0.688191f, -0.425325f,
			0.5f, 0.688191f, -0.425325f,
			0.809017f, -0.262866f, -0.425325f,
			0.0f, 0.0f, -0.951057f
	};
```

```
// Colors of vertices
private float colors[] = {
            0.5f, 0.0f, 1.0f, 1.0f,       1.0f, 0.5f, 0.0f, 1.0f,
            1.0f, 0.5f, 0.0f, 1.0f,       1.0f, 0.5f, 0.0f, 1.0f,
            1.0f, 0.5f, 0.0f, 1.0f,       0.5f, 0.0f, 1.0f, 1.0f,
            0.0f, 1.0f, 1.0f, 1.0f,       1.0f, 0.0f, 1.0f, 1.0f,
            0.0f, 1.0f, 1.0f, 1.0f,       1.0f, 0.0f, 1.0f, 1.0f,
            1.0f, 0.5f, 0.0f, 1.0f,       1.0f, 0.5f, 0.0f, 1.0f,
};

//referencing vertices[] array coordinates
private byte indices[] = {
              2,1,0,          3,2,0,           0,4,3,           4,6,3,
              6,10,3,         3,10,2,          10,9,2,          1,2,9,
              10,6,11,        11,9,10,    5,0,1,           4,0,5,
              7,6,4,          7,4,5,           5,8,7,           1,8,5,
              9,8,1,          8,9,11,          7,8,11,          6,7,11

};

public Ico() {
      //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);
}

//Typical drawing routine using vertex array
public void draw(GL10 gl) {
      //Counterclockwise order for front face vertices
      gl.glFrontFace(GL10.GL_CCW);
      //Points to the vertex buffers
      gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
      gl.glColorPointer(4, GL10.GL_FLOAT, 0, colorBuffer);
      //Enable client states
      gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
      gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
      //Draw vertices as triangles
      gl.glDrawElements(GL10.GL_TRIANGLES, 60, GL10.GL_UNSIGNED_BYTE,
```

```
indexBuffer);
            //Disable client state
            gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
            gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
      }
}
```