

Android Programming

3D Graphics with OpenGL ES

(Including Nehe's Port)

TABLE OF CONTENTS (HIDE)

1. Introduction
2. Getting Started with 3D Graphic
 - 2.1 OpenGL ES
 - 2.2 GLSurfaceView
 - 2.3 Example 1: Setting up OpenGL
 - 2.4 Example 2: Drawing 2D Shape
 - 2.5 Example 3: Color (Nehe Lesson 1)
 - 2.6 Example 4: Rotation (Nehe Lesson 2)
 - 2.7 Example 5: 3D Shapes - Rotations
 - 2.8 Example 6: Texture (Nehe Lesson 3)
 - 2.9 Example 6a: Photo-Cube
 - 2.10 Example 7a: User Inputs (Nehe Lesson 4)
 - 2.11 Example 7b: Texture Filters
 - 2.12 Example 7c: Lighting (Nehe Lesson 5)
 - 2.13 Example 8: Blending (Nehe Lesson 6)
 - 2.14 Example 8a: Bouncing Ball in 3D
3. Android Port for Nehe's Lessons

1. Introduction

Read

1. Android Training "Displaying Graphics with OpenGL ES" @ <http://developer.android.com/training/graphics/opengl/index.html>.
2. Android API Guides "OpenGL ES" @ <http://developer.android.com/guide/topics/graphics/opengl.html>.
3. Android Reference "Package android.opengl" @ <http://developer.android.com/reference/android/opengl/package-summary.html>.

2. Getting Started with 3D Graphics on Android

2.1 OpenGL ES

Android supports OpenGL ES in packages `android.opengl`, `javax.microedition.khronos.opengles` and `javax.microedition.khronos.egl`.

2.2 GLSurfaceView

For 3D graphics programming, you need to program your own custom view, instead using XML-layout. Fortunately, a 3D OpenGL ES view called `GLSurfaceView` is provided, which greatly simplifies our tasks.

I shall use the Nehe's Lessons (<http://nehe.gamedev.net>) to illustrate Android 3D programming

2.3 Example 1: Setting up OpenGL ES using GLSurfaceView (Nehe Lesson 1: Setting Up)

Create an android application called "Nehe 01", with project name "Nehe01", package name "com.test". Create a blank activity called "MyGLActivity".

The following program sets up the `GLSurfaceView`, and show a blank (dark green) screen.

MyGLActivity.java

```
package com.test;

import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
/**
 * Our OpenGL program's main activity
 */
public class MyGLActivity extends Activity {

    private GLSurfaceView glView; // Use GLSurfaceView
```

```

// Call back when the activity is started, to initialize the view
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    glView = new GLSurfaceView(this);           // Allocate a GLSurfaceView
    glView.setRenderer(new MyGLRendererer(this)); // Use a custom renderer
    this.setContentView(glView);               // This activity sets to GLSurfaceView
}

// Call back when the activity is going into the background
@Override
protected void onPause() {
    super.onPause();
    glView.onPause();
}

// Call back after onPause()
@Override
protected void onResume() {
    super.onResume();
    glView.onResume();
}
}

```

Dissecting MyActivity.java

We define MyActivity by extending Activity, so as to override onCreate(), onPause() and onResume(). We then override onCreate() to allocate a GLSurfaceView, set the view's renderer to a custom renderer (to be defined below), and set this activity to use the view.

MyGLRendererer.java

This is our custom OpenGL renderer class.

```

package com.test;

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 MyGLRendererer implements GLSurfaceView.Renderer {
    Context context;    // Application's context

    // Constructor with global application context
    public MyGLRendererer(Context context) {
        this.context = context;
    }

    // Call back when the surface is first created or re-created
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set color's clear-value to black
        gl.glClearDepthf(1.0f);                 // Set depth's clear-value to farthest
        gl.glEnable(GL10.GL_DEPTH_TEST);        // Enables depth-buffer for hidden surface removal
        gl.glDepthFunc(GL10.GL_LEQUAL);         // The type of depth testing to do
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST); // nice perspective view
        gl.glShadeModel(GL10.GL_SMOOTH);        // Enable smooth shading of color
        gl.glDisable(GL10.GL_DITHER);           // Disable dithering for better performance

        // You OpenGL|ES initialization code here
        // .....
    }

    // Call back after onSurfaceCreated() or whenever the window's size changes
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        if (height == 0) height = 1; // To prevent divide by zero
        float aspect = (float)width / height;
    }
}

```

```

// Set the viewport (display area) to cover the entire window
gl.glViewport(0, 0, width, height);

// Setup perspective projection, with aspect ratio matches viewport
gl.glMatrixMode(GL10.GL_PROJECTION); // Select projection matrix
gl.glLoadIdentity();                 // Reset projection matrix
// Use perspective projection
GLU.gluPerspective(gl, 45, aspect, 0.1f, 100.f);

gl.glMatrixMode(GL10.GL_MODELVIEW); // Select model-view matrix
gl.glLoadIdentity();                 // Reset

// You OpenGL|ES display re-sizing code here
// .....
}

// Call back to draw the current frame.
@Override
public void onDrawFrame(GL10 gl) {
    // Clear color and depth buffers using clear-value set earlier
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

    // You OpenGL|ES rendering code here
    // .....
}
}

```

Dissecting MyGLRenderer.java

Our custom rendering class implements interface `GLSurfaceView.Renderer`, which is responsible to make the OpenGL calls to render a frame. It declare 3 methods to be called back by the Android graphics sub-system upon specific GL events.

1. **`onSurfaceCreated(GL10 gl, EGLConfig config)`**: Called when the surface is first created or recreated. It can be used to perform one-time initialization tasks such as setting the clear-value for color and depth, enabling depth-test, etc.
2. **`onSurfaceChanged(GL10 gl, int width, int height)`**: Called when the surface is first displayed and after window's size changes. It is used to set the view port and projection mode.

In our OpenGL renderer, we set the Android's view port (display area) to cover the entire screen from (0,0) to (width-1, height-1):

```
gl.glViewport(0, 0, width, height);
```

We also choose the perspective projection and set the projection volume, with aspect ratio matches the view port, as follows:

```

// OpenGL uses two transformation matrices: projection matrix and model-view matrix
// We select the projection matrix to setup the projection
gl.glMatrixMode(GL10.GL_PROJECTION); // Select projection matrix
gl.glLoadIdentity();                 // Reset projection matrix
// Use perspective projection with the projection volume defined by
//   fovy, aspect-ratation, z-near and z-far
GLU.gluPerspective(gl, 45, aspect, 0.1f, 100.f);

// Select the model-view matrix to manipulate objects (Deselect the projection matrix)
gl.glMatrixMode(GL10.GL_MODELVIEW); // Select model-view matrix
gl.glLoadIdentity();                 // Reset

```

3. **`onDrawFrame(GL10 gl)`**: Called to draw the current frame. You OpenGL rendering codes here.
- In our OpenGL renderer, We clear the color and depth buffers (using the clear-values set via `glClear*` earlier).

```
gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
```

In the Activity class, we construct a custom renderer, and use `setRenderer()` to set it for the view:

```

glView = new GLSurfaceView(this); // Allocate a GLSurfaceView
glView.setRenderer(new MyGLRenderer()); // Set the renderer for the view

```

Run the program. You shall see a blank screen.

2.4 Example 2: Drawing 2D Shapes (Nehe Lesson 2: Your First Polygon)

Let us get started by drawing 2D polygons as illustrated (Push Ctrl-F11 to switch the emulator in landscape orientation):

Create an android application called "Nehe 02", with project name "Nehe02", package name



"com.test". Create a blank activity called "MyGLActivity".

Triangle.java

We first define a class called Triangle.

```
package com.test;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

/*
 * A triangle with 3 vertices.
 */
public class Triangle {
    private FloatBuffer vertexBuffer; // Buffer for vertex-array
    private ByteBuffer indexBuffer;   // Buffer for index-array

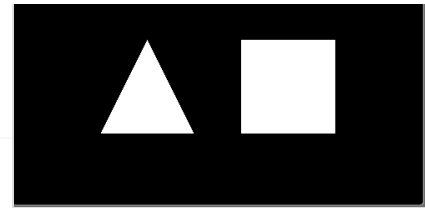
    private float[] vertices = { // Vertices of the triangle
        0.0f, 1.0f, 0.0f, // 0. top
        -1.0f, -1.0f, 0.0f, // 1. left-bottom
        1.0f, -1.0f, 0.0f // 2. right-bottom
    };
    private byte[] indices = { 0, 1, 2 }; // Indices to above vertices (in CCW)

    // Constructor - Setup the data-array buffers
    public Triangle() {
        // Setup vertex-array buffer. Vertices in float. A float has 4 bytes.
        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
        vbb.order(ByteOrder.nativeOrder()); // Use native byte order
        vertexBuffer = vbb.asFloatBuffer(); // Convert byte buffer to float
        vertexBuffer.put(vertices);         // Copy data into buffer
        vertexBuffer.position(0);           // Rewind

        // Setup index-array buffer. Indices in byte.
        indexBuffer = ByteBuffer.allocateDirect(indices.length);
        indexBuffer.put(indices);
        indexBuffer.position(0);
    }

    // Render this shape
    public void draw(GL10 gl) {
        // Enable vertex-array and define the buffers
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);

        // Draw the primitives via index-array
        gl.glDrawElements(GL10.GL_TRIANGLES, indices.length, GL10.GL_UNSIGNED_BYTE, indexBuffer);
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    }
}
```



Dissecting Triangle.java

In OpenGL ES, you cannot define individual vertex via `glVertex` command (this command is not supported in ES due to inefficiency). Instead, you have to use a vertex array to define a group of vertices. This is done in two steps:

1. We first define the (x, y, z) location of the vertices in a Java array:

```
private float[] vertices = { // Vertices of the triangle
    0.0f, 1.0f, 0.0f, // 0. top
    -1.0f, -1.0f, 0.0f, // 1. left-bottom
    1.0f, -1.0f, 0.0f // 2. right-bottom
};
```

2. We then allocate the vertex-array buffer, and transfer the data into the buffer. We use nio's buffer because they are placed on the native heap and are not garbage-collected.

```
private FloatBuffer vertexBuffer;
.....
// Allocate a raw byte buffer. A float has 4 bytes
```

```

ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
// Need to use native byte order
vbb.order(ByteOrder.nativeOrder());
// Convert the byte buffer into float buffer
vertexBuffer = vbb.asFloatBuffer();
// Transfer the data into the buffer
vertexBuffer.put(vertices);
// Rewind
vertexBuffer.position(0);

```

To render from the vertex-array, we need to enable client-state vertex-array:

```
gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
```

We can then use method `glDrawArrays()` to render from the vertex array directly, or `glDrawElements()` to render via an index array.

In the above example, we set up an index array, which indexes into the vertex array (and its associated color array), as follows. Take note that the vertices are arranged in counter-clockwise (CCW) manner, with normal pointing out of the screen (or positive z-direction).

```

private ByteBuffer indexBuffer;
.....
byte[] indices = { 0, 1, 2 }; // CCW
.....
indexBuffer = ByteBuffer.allocateDirect(indices.length); // Allocate raw byte buffer
indexBuffer.put(indices); // Transfer data into buffer
indexBuffer.position(0); // rewind

```

We render the triangle in the `draw()` method, with the following steps:

1. We first enable vertex-array client states.

```
gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
```

2. We then specify the location of the buffers via:

```

gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
// gl*Pointer(int size, int type, int stride, Buffer pointer)
// size: number of coordinates per vertex (must be 2, 3, or 4).
// type: data type of vertex coordinate, GL_BYTE, GL_SHORT, GL_FIXED, or GL_FLOAT
// stride: the byte offset between consecutive vertices. 0 for tightly packed.

```

3. Finally, we render the primitives using `glDrawElements()`, which uses the index array to reference the vertex and color arrays.

```

gl.glDrawElements(GL10.GL_TRIANGLES, numIndices, GL10.GL_UNSIGNED_BYTE, indexBuffer);
// glDrawElements(int mode, int count, int type, Buffer indices)
// mode: GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, or GL_TRIANGLES
// count: the number of elements to be rendered.
// type: data-type of indices (must be GL_UNSIGNED_BYTE or GL_UNSIGNED_SHORT).
// indices: pointer to the index array.
gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);

```

Square.java

Similarly, let's define a quad. Take note that OpenGL ES does not support quad as a primitive. We need to draw two triangles instead. We shall use the same color for all the vertices of the quad. The color can be set via `glColor`.

```

package com.test;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

/*
 * A square drawn in 2 triangles (using TRIANGLE_STRIP).
 */
public class Square {
    private FloatBuffer vertexBuffer; // Buffer for vertex-array

    private float[] vertices = { // Vertices for the square
        -1.0f, -1.0f, 0.0f, // 0. left-bottom
        1.0f, -1.0f, 0.0f, // 1. right-bottom
        -1.0f, 1.0f, 0.0f, // 2. left-top
        1.0f, 1.0f, 0.0f // 3. right-top
    };
};

```

```

// Constructor - Setup the vertex buffer
public Square() {
    // Setup vertex array buffer. Vertices in float. A float has 4 bytes
    ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
    vbb.order(ByteOrder.nativeOrder()); // Use native byte order
    vertexBuffer = vbb.asFloatBuffer(); // Convert from byte to float
    vertexBuffer.put(vertices);         // Copy data into buffer
    vertexBuffer.position(0);           // Rewind
}

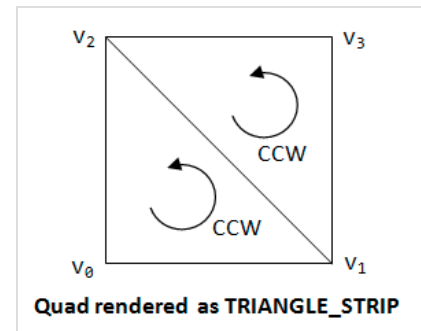
// Render the shape
public void draw(GL10 gl) {
    // Enable vertex-array and define its buffer
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
    // Draw the primitives from the vertex-array directly
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, vertices.length / 3);
    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
}
}

```

Dissecting Square.java

OpenGL ES 1.0 does not support quad as primitive. We shall draw a quad using TRIANGLE_STRIP, composing of 2 triangles v0v1v2 and v2v1v3, in counter-clockwise orientation.

For the triangle, we use `glDrawElements()` which uses an index array to reference the vertex and color array. For the quad, we shall use `glDrawArrays()` to directly render from the vertex-array, as follows:



```

gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);           // Enable vertex array
gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer); // Set the location of vertex array
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, numVertices);
// glDrawArrays(int mode, int first, int count)
// mode: GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, or GL_TRIANGLES
// first: the starting index in the enabled arrays.
// count: the number of indices to be rendered.
gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);

```

GL Renderer

Now, modify the *renderer* to draw the triangle and quad.

```

package com.test;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

public class MyGLRenderer implements GLSurfaceView.Renderer {

    Triangle triangle;    // ( NEW )
    Square quad;          // ( NEW )

    // Constructor
    public MyGLRenderer(Context context) {
        // Set up the data-array buffers for these shapes ( NEW )
        triangle = new Triangle(); // ( NEW )
        quad = new Square();       // ( NEW )
    }

    // Call back when the surface is first created or re-created.
    @Override

```

```

public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    // NO CHANGE - SKIP
    .....
}

// Call back after onSurfaceCreated() or whenever the window's size changes.
@Override
public void onSurfaceChanged(GL10 gl, int width, int height) {
    // NO CHANGE - SKIP
    .....
}

// Call back to draw the current frame.
@Override
public void onDrawFrame(GL10 gl) {
    // Clear color and depth buffers using clear-values set earlier
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

    gl.glLoadIdentity();           // Reset model-view matrix ( NEW )
    gl.glTranslatef(-1.5f, 0.0f, -6.0f); // Translate left and into the screen ( NEW )
    triangle.draw(gl);              // Draw triangle ( NEW )

    // Translate right, relative to the previous translation ( NEW )
    gl.glTranslatef(3.0f, 0.0f, 0.0f);
    quad.draw(gl);                  // Draw quad ( NEW )
}
}

```

We run the shapes' setup codes in the renderer's constructor, as they only have to run once. We invoke the shapes' draw() in renderer's onDrawFrame() which renders the shapes upon each frame refresh.

GL Activity

There is no change to the Activity codes in MyGLActivity.

2.5 Example 3: Color (Nehe Lesson 3: Color)

Triangle.java

Modify the triangle.java as follow:

```

package com.test;

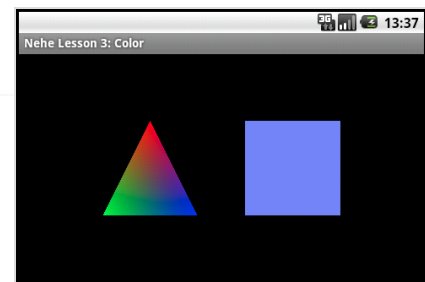
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

/*
 * A triangle with 3 vertices. Each vertex has its own color.
 */
public class Triangle {
    private FloatBuffer vertexBuffer; // Buffer for vertex-array
    private FloatBuffer colorBuffer;  // Buffer for color-array (NEW)
    private ByteBuffer indexBuffer;   // Buffer for index-array

    private float[] vertices = { // Vertices of the triangle
        0.0f, 1.0f, 0.0f, // 0. top
        -1.0f, -1.0f, 0.0f, // 1. left-bottom
        1.0f, -1.0f, 0.0f // 2. right-bottom
    };
    private byte[] indices = { 0, 1, 2 }; // Indices to above vertices (in CCW)
    private float[] colors = { // Colors for the vertices (NEW)
        1.0f, 0.0f, 0.0f, 1.0f, // Red (NEW)
        0.0f, 1.0f, 0.0f, 1.0f, // Green (NEW)
        0.0f, 0.0f, 1.0f, 1.0f // Blue (NEW)
    };
};

// Constructor - Setup the data-array buffers
public Triangle() {
    // Setup vertex-array buffer. Vertices in float. A float has 4 bytes
    ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);

```



```

vbb.order(ByteOrder.nativeOrder()); // Use native byte order
vertexBuffer = vbb.asFloatBuffer(); // Convert byte buffer to float
vertexBuffer.put(vertices);          // Copy data into buffer
vertexBuffer.position(0);             // Rewind

// Setup color-array buffer. Colors in float. A float has 4 bytes (NEW)
ByteBuffer cbb = ByteBuffer.allocateDirect(colors.length * 4);
cbb.order(ByteOrder.nativeOrder()); // Use native byte order (NEW)
colorBuffer = cbb.asFloatBuffer(); // Convert byte buffer to float (NEW)
colorBuffer.put(colors);            // Copy data into buffer (NEW)
colorBuffer.position(0);            // Rewind (NEW)

// Setup index-array buffer. Indices in byte.
indexBuffer = ByteBuffer.allocateDirect(indices.length);
indexBuffer.put(indices);
indexBuffer.position(0);
}

// Render this shape
public void draw(GL10 gl) {
    // Enable arrays and define the buffers
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
    gl.glEnableClientState(GL10.GL_COLOR_ARRAY); // Enable color-array (NEW)
    gl.glColorPointer(4, GL10.GL_FLOAT, 0, colorBuffer); // Define color-array buffer (NEW)

    // Draw the primitives via index-array
    gl.glDrawElements(GL10.GL_TRIANGLES, indices.length, GL10.GL_UNSIGNED_BYTE, indexBuffer);
    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glDisableClientState(GL10.GL_COLOR_ARRAY); // Disable color-array (NEW)
}
}

```

Dissecting Triangle.java

During rendering, the vertex-array will be rendered together with other attributes (such as color, texture and normal), if these attributes are enabled.

In the above example, we define the colors of the vertices and copy them into a color-array buffer. We enable color-array client-state. The colors will be rendered together with the vertices in `glDrawElements()`.

Square.java

Modify Square.java as follows:

```

package com.test;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

/*
 * A square drawn in 2 triangles (using TRIANGLE_STRIP). This square has one color.
 */
public class Square {
    private FloatBuffer vertexBuffer; // Buffer for vertex-array
    private float[] vertices = { // Vertices for the square
        -1.0f, -1.0f, 0.0f, // 0. left-bottom
        1.0f, -1.0f, 0.0f, // 1. right-bottom
        -1.0f, 1.0f, 0.0f, // 2. left-top
        1.0f, 1.0f, 0.0f // 3. right-top
    };

    // Constructor - Setup the vertex buffer
    public Square() {
        // Setup vertex array buffer. Vertices in float. A float has 4 bytes
        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
        vbb.order(ByteOrder.nativeOrder()); // Use native byte order
        vertexBuffer = vbb.asFloatBuffer(); // Convert from byte to float
        vertexBuffer.put(vertices); // Copy data into buffer
        vertexBuffer.position(0); // Rewind
    }
}

```

```

// Render the shape
public void draw(GL10 gl) {
    // Enable vertex-array and define its buffer
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
    gl.glColor4f(0.5f, 0.5f, 1.0f, 1.0f);    // Set the current color (NEW)
    // Draw the primitives from the vertex array directly
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, vertices.length / 3);
    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
}
}

```

Dissecting Square.java

Our square has one color. That is, all vertices are rendered using the same color. Hence, there is no need to define a color-array. Instead, we added a `glColor*` command before rendering the square using `glDrawArrays()`.

GL Renderer and GL Activity

No change.

2.6 Example 4: Rotation (Nehe Lesson 4: Rotation)

To rotate the shapes created in the previous example, we need to make some minor modifications to our renderer.

```

package com.test;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

public class MyGLRenderer implements GLSurfaceView.Renderer {

    private Triangle triangle;
    private Square quad;

    // Rotational angle and speed (NEW)
    private float angleTriangle = 0.0f; // (NEW)
    private float angleQuad = 0.0f;    // (NEW)
    private float speedTriangle = 0.5f; // (NEW)
    private float speedQuad = -0.4f;   // (NEW)

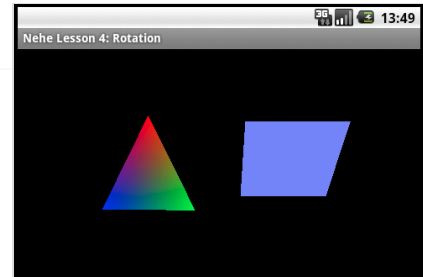
    // Constructor
    public MyGLRenderer(Context context) {
        // Set up the buffers for these shapes
        triangle = new Triangle();
        quad = new Square();
    }

    // Call back when the surface is first created or re-created.
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // NO CHANGE - SKIP
        .....
    }

    // Call back after onSurfaceCreated() or whenever the window's size changes.
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        // NO CHANGE - SKIP
        .....
    }

    // Call back to draw the current frame.
    @Override
    public void onDrawFrame(GL10 gl) {
        // Clear color and depth buffers using clear-values set earlier
    }
}

```



```

gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

gl.glLoadIdentity();           // Reset model-view matrix
gl.glTranslatef(-1.5f, 0.0f, -6.0f); // Translate left and into the screen
gl.glRotatef(angleTriangle, 0.0f, 1.0f, 0.0f); // Rotate the triangle about the y-axis (NEW)
triangle.draw(gl);             // Draw triangle

gl.glLoadIdentity();           // Reset the mode-view matrix (NEW)
gl.glTranslatef(1.5f, 0.0f, -6.0f); // Translate right and into the screen (NEW)
gl.glRotatef(angleQuad, 1.0f, 0.0f, 0.0f); // Rotate the square about the x-axis (NEW)
quad.draw(gl);                 // Draw quad

// Update the rotational angle after each refresh (NEW)
angleTriangle += speedTriangle; // (NEW)
angleQuad += speedQuad;        // (NEW)
}
}

```

A `glRotate*` command was added to rotate the shape, with angle of rotation updated after each refresh.

2.7 Example 5: 3D Shapes - Rotating Color Cube and Pyramid (Nehe Lesson 5: 3D Shapes)

Pyramid.java

Set up the color pyramid as follows:

```

package com.test;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

public class Pyramid {
    private FloatBuffer vertexBuffer; // Buffer for vertex-array
    private FloatBuffer colorBuffer; // Buffer for color-array
    private ByteBuffer indexBuffer; // Buffer for index-array

    private float[] vertices = { // 5 vertices of the pyramid in (x,y,z)
        -1.0f, -1.0f, -1.0f, // 0. left-bottom-back
        1.0f, -1.0f, -1.0f, // 1. right-bottom-back
        1.0f, -1.0f, 1.0f, // 2. right-bottom-front
        -1.0f, -1.0f, 1.0f, // 3. left-bottom-front
        0.0f, 1.0f, 0.0f // 4. top
    };

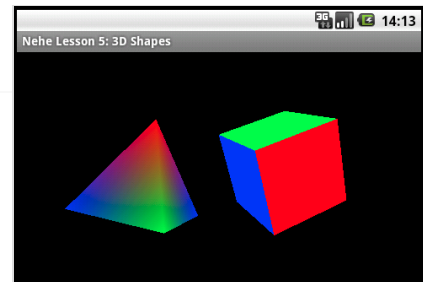
    private float[] colors = { // Colors of the 5 vertices in RGBA
        0.0f, 0.0f, 1.0f, 1.0f, // 0. blue
        0.0f, 1.0f, 0.0f, 1.0f, // 1. green
        0.0f, 0.0f, 1.0f, 1.0f, // 2. blue
        0.0f, 1.0f, 0.0f, 1.0f, // 3. green
        1.0f, 0.0f, 0.0f, 1.0f // 4. red
    };

    private byte[] indices = { // Vertex indices of the 4 Triangles
        2, 4, 3, // front face (CCW)
        1, 4, 2, // right face
        0, 4, 1, // back face
        4, 0, 3 // left face
    };

    // Constructor - Set up the buffers
    public Pyramid() {
        // Setup vertex-array buffer. Vertices in float. An float has 4 bytes
        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
        vbb.order(ByteOrder.nativeOrder()); // Use native byte order
        vertexBuffer = vbb.asFloatBuffer(); // Convert from byte to float
        vertexBuffer.put(vertices); // Copy data into buffer
        vertexBuffer.position(0); // Rewind

        // Setup color-array buffer. Colors in float. An float has 4 bytes
        ByteBuffer cbb = ByteBuffer.allocateDirect(colors.length * 4);
    }
}

```



```

cbb.order(ByteOrder.nativeOrder());
colorBuffer = cbb.asFloatBuffer();
colorBuffer.put(colors);
colorBuffer.position(0);

// Setup index-array buffer. Indices in byte.
indexBuffer = ByteBuffer.allocateDirect(indices.length);
indexBuffer.put(indices);
indexBuffer.position(0);
}

// Draw the shape
public void draw(GL10 gl) {
    gl.glFrontFace(GL10.GL_CCW); // Front face in counter-clockwise orientation

    // Enable arrays and define their buffers
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
    gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
    gl.glColorPointer(4, GL10.GL_FLOAT, 0, colorBuffer);

    gl.glDrawElements(GL10.GL_TRIANGLES, indices.length, GL10.GL_UNSIGNED_BYTE,
        indexBuffer);

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

```

Cube.java

Similarly, set up the color cube as follows:

```

package com.test;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

public class Cube {
    private FloatBuffer vertexBuffer; // Buffer for vertex-array
    private int numFaces = 6;

    private float[][] colors = { // Colors of the 6 faces
        {1.0f, 0.5f, 0.0f, 1.0f}, // 0. orange
        {1.0f, 0.0f, 1.0f, 1.0f}, // 1. violet
        {0.0f, 1.0f, 0.0f, 1.0f}, // 2. green
        {0.0f, 0.0f, 1.0f, 1.0f}, // 3. blue
        {1.0f, 0.0f, 0.0f, 1.0f}, // 4. red
        {1.0f, 1.0f, 0.0f, 1.0f} // 5. yellow
    };

    private float[] vertices = { // Vertices of the 6 faces
        // FRONT
        -1.0f, -1.0f, 1.0f, // 0. left-bottom-front
        1.0f, -1.0f, 1.0f, // 1. right-bottom-front
        -1.0f, 1.0f, 1.0f, // 2. left-top-front
        1.0f, 1.0f, 1.0f, // 3. right-top-front
        // BACK
        1.0f, -1.0f, -1.0f, // 6. right-bottom-back
        -1.0f, -1.0f, -1.0f, // 4. left-bottom-back
        1.0f, 1.0f, -1.0f, // 7. right-top-back
        -1.0f, 1.0f, -1.0f, // 5. left-top-back
        // LEFT
        -1.0f, -1.0f, -1.0f, // 4. left-bottom-back
        -1.0f, -1.0f, 1.0f, // 0. left-bottom-front
        -1.0f, 1.0f, -1.0f, // 5. left-top-back
        -1.0f, 1.0f, 1.0f, // 2. left-top-front
        // RIGHT
        1.0f, -1.0f, 1.0f, // 1. right-bottom-front
        1.0f, -1.0f, -1.0f, // 6. right-bottom-back
    };
}

```

```

    1.0f, 1.0f, 1.0f, // 3. right-top-front
    1.0f, 1.0f, -1.0f, // 7. right-top-back
// TOP
-1.0f, 1.0f, 1.0f, // 2. left-top-front
    1.0f, 1.0f, 1.0f, // 3. right-top-front
-1.0f, 1.0f, -1.0f, // 5. left-top-back
    1.0f, 1.0f, -1.0f, // 7. right-top-back
// BOTTOM
-1.0f, -1.0f, -1.0f, // 4. left-bottom-back
    1.0f, -1.0f, -1.0f, // 6. right-bottom-back
-1.0f, -1.0f, 1.0f, // 0. left-bottom-front
    1.0f, -1.0f, 1.0f // 1. right-bottom-front
};

// Constructor - Set up the buffers
public Cube() {
    // Setup vertex-array buffer. Vertices in float. An float has 4 bytes
    ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
    vbb.order(ByteOrder.nativeOrder()); // Use native byte order
    vertexBuffer = vbb.asFloatBuffer(); // Convert from byte to float
    vertexBuffer.put(vertices); // Copy data into buffer
    vertexBuffer.position(0); // Rewind
}

// Draw the shape
public void draw(GL10 gl) {
    gl.glFrontFace(GL10.GL_CCW); // Front face in counter-clockwise orientation
    gl.glEnable(GL10.GL_CULL_FACE); // Enable cull face
    gl.glCullFace(GL10.GL_BACK); // Cull the back face (don't display)

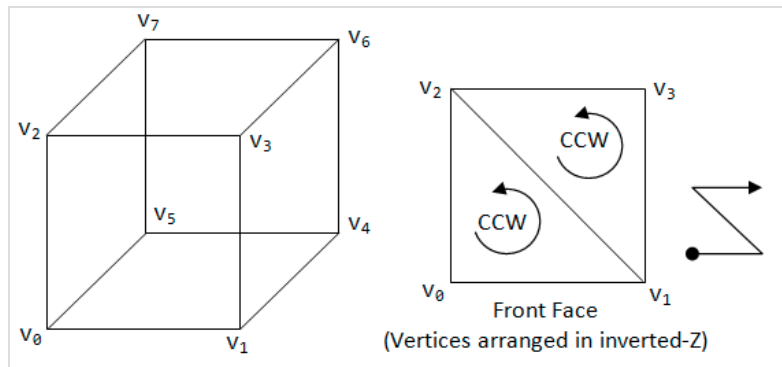
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);

    // Render all the faces
    for (int face = 0; face < numFaces; face++) {
        // Set the color for each of the faces
        gl.glColor4f(colors[face][0], colors[face][1], colors[face][2], colors[face][3]);
        // Draw the primitive from the vertex-array directly
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, face*4, 4);
    }
    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glDisable(GL10.GL_CULL_FACE);
}
}

```

The vertices of the color cube is labeled as follows.

The vertices of all the faces are arranged in counter-clockwise orientation with normal pointing outwards in a consistent manner. This enables us to cull the back face with the following codes:



```

gl.glFrontFace(GL10.GL_CCW); // Set the front face
gl.glEnable(GL10.GL_CULL_FACE); // Enable cull face
gl.glCullFace(GL10.GL_BACK); // Cull the back face

```

GL Renderer

The renderer is as follows:

```
package com.test;
```

```

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

public class MyGLRenderer implements GLSurfaceView.Renderer {

    private Pyramid pyramid;    // (NEW)
    private Cube cube;          // (NEW)

    private static float anglePyramid = 0; // Rotational angle in degree for pyramid (NEW)
    private static float angleCube = 0;    // Rotational angle in degree for cube (NEW)
    private static float speedPyramid = 2.0f; // Rotational speed for pyramid (NEW)
    private static float speedCube = -1.5f;  // Rotational speed for cube (NEW)

    // Constructor
    public MyGLRenderer(Context context) {
        // Set up the buffers for these shapes
        pyramid = new Pyramid();    // (NEW)
        cube = new Cube();          // (NEW)
    }

    // Call back when the surface is first created or re-created.
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // NO CHANGE - SKIP
        .....
    }

    // Call back after onSurfaceCreated() or whenever the window's size changes.
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        // NO CHANGE - SKIP
        .....
    }

    // Call back to draw the current frame.
    @Override
    public void onDrawFrame(GL10 gl) {
        // Clear color and depth buffers
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        // ----- Render the Pyramid -----
        gl.glLoadIdentity();           // Reset the model-view matrix
        gl.glTranslatef(-1.5f, 0.0f, -6.0f); // Translate left and into the screen
        gl.glRotatef(anglePyramid, 0.1f, 1.0f, -0.1f); // Rotate (NEW)
        pyramid.draw(gl);              // Draw the pyramid (NEW)

        // ----- Render the Color Cube -----
        gl.glLoadIdentity();           // Reset the model-view matrix
        gl.glTranslatef(1.5f, 0.0f, -6.0f); // Translate right and into the screen
        gl.glScalef(0.8f, 0.8f, 0.8f);    // Scale down (NEW)
        gl.glRotatef(angleCube, 1.0f, 1.0f, 1.0f); // rotate about the axis (1,1,1) (NEW)
        cube.draw(gl);                  // Draw the cube (NEW)

        // Update the rotational angle after each refresh (NEW)
        anglePyramid += speedPyramid;    // (NEW)
        angleCube += speedCube;          // (NEW)
    }
}

```

Cube

There are many ways to render a cube. You could define all the vertices of the 6 faces as in the above example. You could also define one representative face, and render the face 6 times with proper translation and rotation.

Example 1: Cube1.java

```

package com.test;

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

```

```

import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;
/*
 * Define the vertices for only one face (the front face).
 * Render the cube by translating and rotating the face.
 */
public class Cube1 {
    private FloatBuffer vertexBuffer; // Buffer for vertex-array

    private float[][] colors = { // Colors of the 6 faces
        {1.0f, 0.5f, 0.0f, 1.0f}, // 0. orange
        {1.0f, 0.0f, 1.0f, 1.0f}, // 1. violet
        {0.0f, 1.0f, 0.0f, 1.0f}, // 2. green
        {0.0f, 0.0f, 1.0f, 1.0f}, // 3. blue
        {1.0f, 0.0f, 0.0f, 1.0f}, // 4. red
        {1.0f, 1.0f, 0.0f, 1.0f} // 5. yellow
    };

    private float[] vertices = { // Vertices for the front face
        -1.0f, -1.0f, 1.0f, // 0. left-bottom-front
        1.0f, -1.0f, 1.0f, // 1. right-bottom-front
        -1.0f, 1.0f, 1.0f, // 2. left-top-front
        1.0f, 1.0f, 1.0f // 3. right-top-front
    };

    // Constructor - Set up the buffers
    public Cube1() {
        // Setup vertex-array buffer. Vertices in float. An float has 4 bytes
        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
        vbb.order(ByteOrder.nativeOrder()); // Use native byte order
        vertexBuffer = vbb.asFloatBuffer(); // Convert from byte to float
        vertexBuffer.put(vertices); // Copy data into buffer
        vertexBuffer.position(0); // Rewind
    }

    // Draw the color cube
    public void draw(GL10 gl) {
        gl.glFrontFace(GL10.GL_CCW); // Front face in counter-clockwise orientation
        gl.glEnable(GL10.GL_CULL_FACE); // Enable cull face
        gl.glCullFace(GL10.GL_BACK); // Cull the back face (don't display)

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);

        // Front
        gl.glColor4f(colors[0][0], colors[0][1], colors[0][2], colors[0][3]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);

        // Right - Rotate 90 degree about y-axis
        gl.glRotatef(90.0f, 0.0f, 1.0f, 0.0f);
        gl.glColor4f(colors[1][0], colors[1][1], colors[1][2], colors[1][3]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);

        // Back - Rotate another 90 degree about y-axis
        gl.glRotatef(90.0f, 0.0f, 1.0f, 0.0f);
        gl.glColor4f(colors[2][0], colors[2][1], colors[2][2], colors[2][3]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);

        // Left - Rotate another 90 degree about y-axis
        gl.glRotatef(90.0f, 0.0f, 1.0f, 0.0f);
        gl.glColor4f(colors[3][0], colors[3][1], colors[3][2], colors[3][3]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);

        // Bottom - Rotate 90 degree about x-axis
        gl.glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
        gl.glColor4f(colors[4][0], colors[4][1], colors[4][2], colors[4][3]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);

        // Top - Rotate another 180 degree about x-axis
        gl.glRotatef(180.0f, 1.0f, 0.0f, 0.0f);
        gl.glColor4f(colors[5][0], colors[5][1], colors[5][2], colors[5][3]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    }
}

```

```

        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisable(GL10.GL_CULL_FACE);
    }
}

```

Example 2: Cube2.java

```

package com.test;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;

/*
 * Define the vertices for a representative face.
 * Render the cube by translating and rotating the face.
 */
public class Cube2 {
    private FloatBuffer vertexBuffer; // Buffer for vertex-array

    private float[] vertices = { // Vertices for a face at z=0
        -1.0f, -1.0f, 0.0f, // 0. left-bottom-front
        1.0f, -1.0f, 0.0f, // 1. right-bottom-front
        -1.0f, 1.0f, 0.0f, // 2. left-top-front
        1.0f, 1.0f, 0.0f // 3. right-top-front
    };

    private float[][] colors = { // Colors of the 6 faces
        {1.0f, 0.5f, 0.0f, 1.0f}, // 0. orange
        {1.0f, 0.0f, 1.0f, 1.0f}, // 1. violet
        {0.0f, 1.0f, 0.0f, 1.0f}, // 2. green
        {0.0f, 0.0f, 1.0f, 1.0f}, // 3. blue
        {1.0f, 0.0f, 0.0f, 1.0f}, // 4. red
        {1.0f, 1.0f, 0.0f, 1.0f} // 5. yellow
    };

    // Constructor - Set up the buffers
    public Cube2() {
        // Setup vertex-array buffer. Vertices in float. An float has 4 bytes
        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
        vbb.order(ByteOrder.nativeOrder()); // Use native byte order
        vertexBuffer = vbb.asFloatBuffer(); // Convert from byte to float
        vertexBuffer.put(vertices); // Copy data into buffer
        vertexBuffer.position(0); // Rewind
    }

    // Draw the shape
    public void draw(GL10 gl) {
        gl.glFrontFace(GL10.GL_CCW); // Front face in counter-clockwise orientation
        gl.glEnable(GL10.GL_CULL_FACE); // Enable cull face
        gl.glCullFace(GL10.GL_BACK); // Cull the back face (don't display)

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);

        // front
        gl.glPushMatrix();
        gl.glTranslatef(0.0f, 0.0f, 1.0f);
        gl.glColor4f(colors[0][0], colors[0][1], colors[0][2], colors[0][3]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
        gl.glPopMatrix();

        // left
        gl.glPushMatrix();
        gl.glRotatef(270.0f, 0.0f, 1.0f, 0.0f);
        gl.glTranslatef(0.0f, 0.0f, 1.0f);
        gl.glColor4f(colors[1][0], colors[1][1], colors[1][2], colors[1][3]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
        gl.glPopMatrix();

        // back
        gl.glPushMatrix();
        gl.glRotatef(180.0f, 0.0f, 1.0f, 0.0f);

```

```

gl.glTranslatef(0.0f, 0.0f, 1.0f);
gl.glColor4f(colors[2][0], colors[2][1], colors[2][2], colors[2][3]);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glPopMatrix();

// right
gl.glPushMatrix();
gl.glRotatef(90.0f, 0.0f, 1.0f, 0.0f);
gl.glTranslatef(0.0f, 0.0f, 1.0f);
gl.glColor4f(colors[3][0], colors[3][1], colors[3][2], colors[3][3]);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glPopMatrix();

// top
gl.glPushMatrix();
gl.glRotatef(270.0f, 1.0f, 0.0f, 0.0f);
gl.glTranslatef(0.0f, 0.0f, 1.0f);
gl.glColor4f(colors[4][0], colors[4][1], colors[4][2], colors[4][3]);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glPopMatrix();

// bottom
gl.glPushMatrix();
gl.glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
gl.glTranslatef(0.0f, 0.0f, 1.0f);
gl.glColor4f(colors[5][0], colors[5][1], colors[5][2], colors[5][3]);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glPopMatrix();

gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
gl.glDisable(GL10.GL_CULL_FACE);
}
}

```

2.8 Example 6: Texture (Nehe Lesson 6: Texture)

Let's convert our color-cube into a texture-cube.

TextureCube.java

Let's modify our earlier Cube2.java to set up the texture array.

```

package com.test;

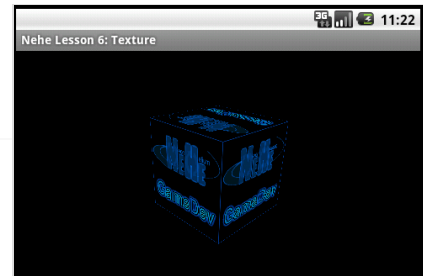
import java.io.IOException;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLUtils;

/*
 * A cube with texture.
 * Define the vertices for only one representative face.
 * Render the cube by translating and rotating the face.
 */
public class TextureCube {
    private FloatBuffer vertexBuffer; // Buffer for vertex-array
    private FloatBuffer texBuffer;    // Buffer for texture-coords-array (NEW)

    private float[] vertices = { // Vertices for a face
        -1.0f, -1.0f, 0.0f, // 0. left-bottom-front
        1.0f, -1.0f, 0.0f, // 1. right-bottom-front
        -1.0f, 1.0f, 0.0f, // 2. left-top-front
        1.0f, 1.0f, 0.0f // 3. right-top-front
    };

    float[] texCoords = { // Texture coords for the above face (NEW)
        0.0f, 1.0f, // A. left-bottom (NEW)

```



```

    1.0f, 1.0f, // B. right-bottom (NEW)
    0.0f, 0.0f, // C. left-top (NEW)
    1.0f, 0.0f // D. right-top (NEW)
};
int[] textureIDs = new int[1]; // Array for 1 texture-ID (NEW)

// Constructor - Set up the buffers
public TextureCube() {
    // Setup vertex-array buffer. Vertices in float. An float has 4 bytes
    ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
    vbb.order(ByteOrder.nativeOrder()); // Use native byte order
    vertexBuffer = vbb.asFloatBuffer(); // Convert from byte to float
    vertexBuffer.put(vertices); // Copy data into buffer
    vertexBuffer.position(0); // Rewind

    // Setup texture-coords-array buffer, in float. An float has 4 bytes (NEW)
    ByteBuffer tbb = ByteBuffer.allocateDirect(texCoords.length * 4);
    tbb.order(ByteOrder.nativeOrder());
    texBuffer = tbb.asFloatBuffer();
    texBuffer.put(texCoords);
    texBuffer.position(0);
}

// Draw the shape
public void draw(GL10 gl) {
    gl.glFrontFace(GL10.GL_CCW); // Front face in counter-clockwise orientation
    gl.glEnable(GL10.GL_CULL_FACE); // Enable cull face
    gl.glCullFace(GL10.GL_BACK); // Cull the back face (don't display)

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
    gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY); // Enable texture-coords-array (NEW)
    gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, texBuffer); // Define texture-coords buffer (NEW)

    // front
    gl.glPushMatrix();
    gl.glTranslatef(0.0f, 0.0f, 1.0f);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    gl.glPopMatrix();

    // left
    gl.glPushMatrix();
    gl.glRotatef(270.0f, 0.0f, 1.0f, 0.0f);
    gl.glTranslatef(0.0f, 0.0f, 1.0f);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    gl.glPopMatrix();

    // back
    gl.glPushMatrix();
    gl.glRotatef(180.0f, 0.0f, 1.0f, 0.0f);
    gl.glTranslatef(0.0f, 0.0f, 1.0f);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    gl.glPopMatrix();

    // right
    gl.glPushMatrix();
    gl.glRotatef(90.0f, 0.0f, 1.0f, 0.0f);
    gl.glTranslatef(0.0f, 0.0f, 1.0f);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    gl.glPopMatrix();

    // top
    gl.glPushMatrix();
    gl.glRotatef(270.0f, 1.0f, 0.0f, 0.0f);
    gl.glTranslatef(0.0f, 0.0f, 1.0f);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    gl.glPopMatrix();

    // bottom
    gl.glPushMatrix();
    gl.glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
    gl.glTranslatef(0.0f, 0.0f, 1.0f);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);

```

```

gl.glPopMatrix();

gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY); // Disable texture-coords-array (NEW)
gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
gl.glDisable(GL10.GL_CULL_FACE);
}

// Load an image into GL texture
public void loadTexture(GL10 gl, Context context) {
    gl.glGenTextures(1, textureIDs, 0); // Generate texture-ID array

    gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[0]); // Bind to texture ID
    // Set up texture filters
    gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
    gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);

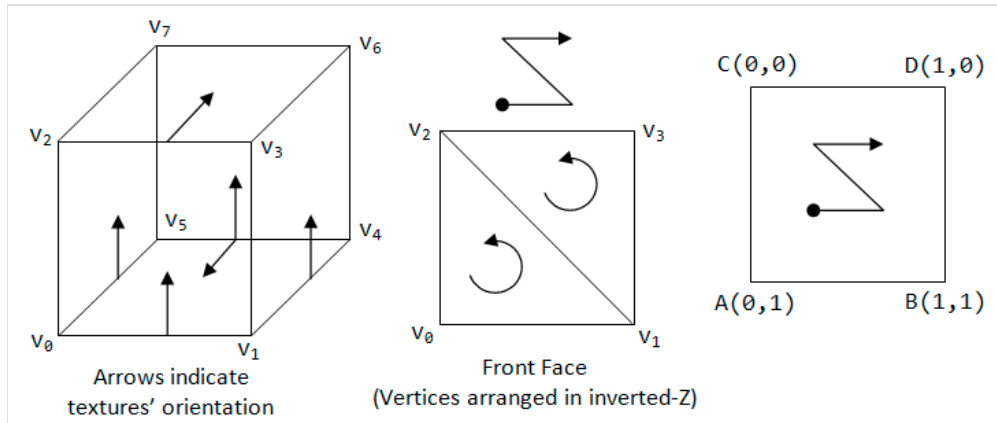
    // Construct an input stream to texture image "res\drawable\nehe.png"
    InputStream istream = context.getResources().openRawResource(R.drawable.nehe);
    Bitmap bitmap;
    try {
        // Read and decode input as bitmap
        bitmap = BitmapFactory.decodeStream(istream);
    } finally {
        try {
            istream.close();
        } catch (IOException e) { }
    }

    // Build Texture from loaded bitmap for the currently-bind texture ID
    GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
    bitmap.recycle();
}
}

```

Dissecting TextureCube.java

The vertices of all the 6 faces are arranged in a consistent manner (inverted-Z). Hence, we can use the same texture coordinates for all 6 faces. We define the texture coords once, and put into the texture buffer 6 times. Take note that texture coordinates' origin is at the top-left corner. The coordinates are normalized to [0, 1].



```

private FloatBuffer texBuffer; // Texture Coords Buffer
.....
float[] texCoords = { // Define the texture coord, applicable to all 6 faces
    // FRONT
    0.0f, 1.0f, // A. left-bottom
    1.0f, 1.0f, // B. right-bottom
    0.0f, 0.0f, // C. left-top
    1.0f, 0.0f // D. right-top
};

// Allocate texture buffer
ByteBuffer tbb = ByteBuffer.allocateDirect(texCoords.length * 4 * 6);
tbb.order(ByteOrder.nativeOrder());
texBuffer = tbb.asFloatBuffer();
// All the 6 faces have the same texture coords, repeat 6 times
for (int face = 0; face < 6; face++) {
    texBuffer.put(texCoords);
}

```

```
texBuffer.position(0);    // Rewind
```

To render the texture, we simply enable client-state texture-coords-array (together with vertex-array). The vertices and texture-coords will be rendered together.

```
gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, texBuffer);

for (int face = 0; face < 6; face++) {
    // Render each face in TRIANGLE_STRIP using 4 vertices
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, face*4, 4);
}
```

We store the texture image "nehe.png" into folder "res/drawable".

The following steps are needed to setup texture and load an image:

1. [TODO]

Take note that we have removed all the color information.

GL Renderer

We need to modify our renderer to setup texture as follows:

```
package com.test;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

public class MyGLRenderer implements GLSurfaceView.Renderer {

    private Context context;    // Application context needed to read image (NEW)
    private TextureCube cube;
    private static float angleCube = 0;    // rotational angle in degree for cube
    private static float speedCube = -1.5f; // rotational speed for cube

    // Constructor
    public MyGLRenderer(Context context) {
        this.context = context;    // Get the application context (NEW)
        cube = new TextureCube();
    }

    // Call back when the surface is first created or re-created.
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set color's clear-value to black
        gl.glClearDepthf(1.0f);    // Set depth's clear-value to farthest
        gl.glEnable(GL10.GL_DEPTH_TEST); // Enables depth-buffer for hidden surface removal
        gl.glDepthFunc(GL10.GL_LEQUAL); // The type of depth testing to do
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST); // nice perspective view
        gl.glShadeModel(GL10.GL_SMOOTH); // Enable smooth shading of color
        gl.glDisable(GL10.GL_DITHER); // Disable dithering for better performance

        // Setup Texture, each time the surface is created (NEW)
        cube.loadTexture(gl, context); // Load image into Texture (NEW)
        gl.glEnable(GL10.GL_TEXTURE_2D); // Enable texture (NEW)
    }

    // Call back after onSurfaceCreated() or whenever the window's size changes.
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        // NO CHANGE - SKIP
        .....
    }

    // Call back to draw the current frame.
    @Override
    public void onDrawFrame(GL10 gl) {
```

```

// Clear color and depth buffers
gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

// ----- Render the Cube -----
gl.glLoadIdentity();           // Reset the current model-view matrix
gl.glTranslatef(0.0f, 0.0f, -6.0f); // Translate into the screen
gl.glRotatef(angleCube, 0.1f, 1.0f, 0.2f); // Rotate
cube.draw(gl);

// Update the rotational angle after each refresh.
angleCube += speedCube;
}
}

```

2.9 Example 6a: Photo-Cube

Let's convert the texture cube into photo cube with different images for each of the 6 faces.

PhotoCube.java

```

package com.test;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLUtils;

/*
 * A photo cube with 6 pictures (textures) on its 6 faces.
 */
public class PhotoCube {
    private FloatBuffer vertexBuffer; // Vertex Buffer
    private FloatBuffer texBuffer;    // Texture Coords Buffer

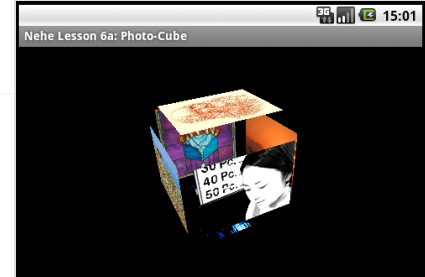
    private int numFaces = 6;
    private int[] imageFileIDs = { // Image file IDs
        R.drawable.caldera,
        R.drawable.candice,
        R.drawable.mule,
        R.drawable.glass,
        R.drawable.leonardo,
        R.drawable.tmsk
    };

    private int[] textureIDs = new int[numFaces];
    private Bitmap[] bitmap = new Bitmap[numFaces];
    private float cubeHalfSize = 1.2f;

    // Constructor - Set up the vertex buffer
    public PhotoCube(Context context) {
        // Allocate vertex buffer. An float has 4 bytes
        ByteBuffer vbb = ByteBuffer.allocateDirect(12 * 4 * numFaces);
        vbb.order(ByteOrder.nativeOrder());
        vertexBuffer = vbb.asFloatBuffer();

        // Read images. Find the aspect ratio and adjust the vertices accordingly.
        for (int face = 0; face < numFaces; face++) {
            bitmap[face] = BitmapFactory.decodeStream(
                context.getResources().openRawResource(imageFileIDs[face]));
            int imgWidth = bitmap[face].getWidth();
            int imgHeight = bitmap[face].getHeight();
            float faceWidth = 2.0f;
            float faceHeight = 2.0f;
            // Adjust for aspect ratio
            if (imgWidth > imgHeight) {
                faceHeight = faceHeight * imgHeight / imgWidth;
            } else {
                faceWidth = faceWidth * imgWidth / imgHeight;
            }
        }
    }
}

```



```

        float faceLeft = -faceWidth / 2;
        float faceRight = -faceLeft;
        float faceTop = faceHeight / 2;
        float faceBottom = -faceTop;

        // Define the vertices for this face
        float[] vertices = {
            faceLeft, faceBottom, 0.0f, // 0. left-bottom-front
            faceRight, faceBottom, 0.0f, // 1. right-bottom-front
            faceLeft, faceTop, 0.0f, // 2. left-top-front
            faceRight, faceTop, 0.0f, // 3. right-top-front
        };
        vertexBuffer.put(vertices); // Populate
    }
    vertexBuffer.position(0); // Rewind

    // Allocate texture buffer. An float has 4 bytes. Repeat for 6 faces.
    float[] texCoords = {
        0.0f, 1.0f, // A. left-bottom
        1.0f, 1.0f, // B. right-bottom
        0.0f, 0.0f, // C. left-top
        1.0f, 0.0f // D. right-top
    };
    ByteBuffer tbb = ByteBuffer.allocateDirect(texCoords.length * 4 * numFaces);
    tbb.order(ByteOrder.nativeOrder());
    texBuffer = tbb.asFloatBuffer();
    for (int face = 0; face < numFaces; face++) {
        texBuffer.put(texCoords);
    }
    texBuffer.position(0); // Rewind
}

// Render the shape
public void draw(GL10 gl) {
    gl.glFrontFace(GL10.GL_CCW);

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
    gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, texBuffer);

    // front
    gl.glPushMatrix();
    gl.glTranslatef(0f, 0f, cubeHalfSize);
    gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[0]);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    gl.glPopMatrix();

    // left
    gl.glPushMatrix();
    gl.glRotatef(270.0f, 0f, 1f, 0f);
    gl.glTranslatef(0f, 0f, cubeHalfSize);
    gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[1]);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 4, 4);
    gl.glPopMatrix();

    // back
    gl.glPushMatrix();
    gl.glRotatef(180.0f, 0f, 1f, 0f);
    gl.glTranslatef(0f, 0f, cubeHalfSize);
    gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[2]);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 8, 4);
    gl.glPopMatrix();

    // right
    gl.glPushMatrix();
    gl.glRotatef(90.0f, 0f, 1f, 0f);
    gl.glTranslatef(0f, 0f, cubeHalfSize);
    gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[3]);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 12, 4);
    gl.glPopMatrix();

    // top

```

```

        gl.glPushMatrix();
        gl.glRotatef(270.0f, 1f, 0f, 0f);
        gl.glTranslatef(0f, 0f, cubeHalfSize);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[4]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 16, 4);
        gl.glPopMatrix();

        // bottom
        gl.glPushMatrix();
        gl.glRotatef(90.0f, 1f, 0f, 0f);
        gl.glTranslatef(0f, 0f, cubeHalfSize);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[5]);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 20, 4);
        gl.glPopMatrix();

        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
    }

    // Load images into 6 GL textures
    public void loadTexture(GL10 gl) {
        gl.glGenTextures(6, textureIDs, 0); // Generate texture-ID array for 6 IDs

        // Generate OpenGL texture images
        for (int face = 0; face < numFaces; face++) {
            gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[face]);
            // Build Texture from loaded bitmap for the currently-bind texture ID
            GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap[face], 0);
            bitmap[face].recycle();
        }
    }
}

```

MyGLRenderer.java

```

package com.test;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

public class MyGLRenderer implements GLSurfaceView.Renderer {
    private PhotoCube cube; // (NEW)
    private static float angleCube = 0; // rotational angle in degree for cube
    private static float speedCube = -1.5f; // rotational speed for cube

    // Constructor
    public MyGLRenderer(Context context) {
        cube = new PhotoCube(context); // (NEW)
    }

    // Call back when the surface is first created or re-created.
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set color's clear-value to black
        gl.glClearDepthf(1.0f); // Set depth's clear-value to farthest
        gl.glEnable(GL10.GL_DEPTH_TEST); // Enables depth-buffer for hidden surface removal
        gl.glDepthFunc(GL10.GL_LEQUAL); // The type of depth testing to do
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST); // nice perspective view
        gl.glShadeModel(GL10.GL_SMOOTH); // Enable smooth shading of color
        gl.glDisable(GL10.GL_DITHER); // Disable dithering for better performance

        // Setup Texture, each time the surface is created (NEW)
        cube.loadTexture(gl); // Load images into textures (NEW)
        gl.glEnable(GL10.GL_TEXTURE_2D); // Enable texture (NEW)
    }

    // Call back after onSurfaceCreated() or whenever the window's size changes.
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {

```

```

        // NO CHANGE - SKIP
        .....
    }

    // Call back to draw the current frame.
    @Override
    public void onDrawFrame(GL10 gl) {
        // Clear color and depth buffers
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        // ----- Render the Cube -----
        gl.glLoadIdentity();           // Reset the model-view matrix
        gl.glTranslatef(0.0f, 0.0f, -6.0f); // Translate into the screen
        gl.glRotatef(angleCube, 0.15f, 1.0f, 0.3f); // Rotate
        cube.draw(gl);

        // Update the rotational angle after each refresh.
        angleCube += speedCube;
    }
}

```

2.10 Example 7a: User Inputs (Nehe Lesson 7 Part 1: Key-Controlled)

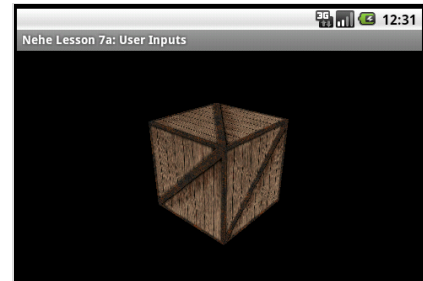
Nehe lesson 7 is far too complex, I shall break it into 3 parts: Key-controlled, Texture Filters, and Lighting.

TextureCube.java

No change, except using image "crate.png".

MyGLRenderer.java

We modify our renderer by adding variables and transformation methods to control the cube z-position, x and y rotational angles and speeds.



```

package com.test;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

public class MyGLRenderer implements GLSurfaceView.Renderer {

    private Context context;
    private TextureCube cube;
    // For controlling cube's z-position, x and y angles and speeds (NEW)
    float angleX = 0; // (NEW)
    float angleY = 0; // (NEW)
    float speedX = 0; // (NEW)
    float speedY = 0; // (NEW)
    float z = -6.0f; // (NEW)

    // Constructor
    public MyGLRenderer(Context context) {
        this.context = context;
        cube = new TextureCube();
    }

    // Call back when the surface is first created or re-created.
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set color's clear-value to black
        gl.glClearDepthf(1.0f); // Set depth's clear-value to farthest
        gl.glEnable(GL10.GL_DEPTH_TEST); // Enables depth-buffer for hidden surface removal
        gl.glDepthFunc(GL10.GL_LEQUAL); // The type of depth testing to do
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST); // nice perspective view
        gl.glShadeModel(GL10.GL_SMOOTH); // Enable smooth shading of color
        gl.glDisable(GL10.GL_DITHER); // Disable dithering for better performance
    }
}

```

```

        // Setup Texture, each time the surface is created
        cube.loadTexture(gl, context);    // Load image into Texture
        gl.glEnable(GL10.GL_TEXTURE_2D); // Enable texture
    }

    // Call back after onSurfaceCreated() or whenever the window's size changes.
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        // NO CHANGE - SKIP
        .....
    }

    // Call back to draw the current frame.
    @Override
    public void onDrawFrame(GL10 gl) {
        // Clear color and depth buffers
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        // ----- Render the Cube -----
        gl.glLoadIdentity();           // Reset the model-view matrix
        gl.glTranslatef(0.0f, 0.0f, z); // Translate into the screen (NEW)
        gl.glRotatef(angleX, 1.0f, 0.0f, 0.0f); // Rotate (NEW)
        gl.glRotatef(angleY, 0.0f, 1.0f, 0.0f); // Rotate (NEW)
        cube.draw(gl);

        // Update the rotational angle after each refresh (NEW)
        angleX += speedX; // (NEW)
        angleY += speedY; // (NEW)
    }
}

```

MyGLSurfaceView.java

In order to capture the user inputs, we need to customize the GLSurfaceView by extending a subclass, so as to override the event handlers (such as onKeyUp(), onTouchEvent()).

```

package com.test;

import android.content.Context;
import android.opengl.GLSurfaceView;
import android.view.KeyEvent;
import android.view.MotionEvent;
/*
 * Custom GL view by extending GLSurfaceView so as
 * to override event handlers such as onKeyUp(), onTouchEvent()
 */
public class MyGLSurfaceView extends GLSurfaceView {
    MyGLRenderer renderer;    // Custom GL Renderer

    // For touch event
    private final float TOUCH_SCALE_FACTOR = 180.0f / 320.0f;
    private float previousX;
    private float previousY;

    // Constructor - Allocate and set the renderer
    public MyGLSurfaceView(Context context) {
        super(context);
        renderer = new MyGLRenderer(context);
        this.setRenderer(renderer);
        // Request focus, otherwise key/button won't react
        this.requestFocus();
        this.setFocusableInTouchMode(true);
    }

    // Handler for key event
    @Override
    public boolean onKeyUp(int keyCode, KeyEvent evt) {
        switch(keyCode) {
            case KeyEvent.KEYCODE_DPAD_LEFT:    // Decrease Y-rotational speed
                renderer.speedY -= 0.1f;
                break;
            case KeyEvent.KEYCODE_DPAD_RIGHT:   // Increase Y-rotational speed

```

```

        rendererer.speedY += 0.1f;
        break;
    case KeyEvent.KEYCODE_DPAD_UP:    // Decrease X-rotational speed
        rendererer.speedX -= 0.1f;
        break;
    case KeyEvent.KEYCODE_DPAD_DOWN:  // Increase X-rotational speed
        rendererer.speedX += 0.1f;
        break;
    case KeyEvent.KEYCODE_A:          // Zoom out (decrease z)
        rendererer.z -= 0.2f;
        break;
    case KeyEvent.KEYCODE_Z:          // Zoom in (increase z)
        rendererer.z += 0.2f;
        break;
    }
    return true; // Event handled
}

// Handler for touch event
@Override
public boolean onTouchEvent(final MotionEvent evt) {
    float currentX = evt.getX();
    float currentY = evt.getY();
    float deltaX, deltaY;
    switch (evt.getAction()) {
        case MotionEvent.ACTION_MOVE:
            // Modify rotational angles according to movement
            deltaX = currentX - previousX;
            deltaY = currentY - previousY;
            rendererer.angleX += deltaY * TOUCH_SCALE_FACTOR;
            rendererer.angleY += deltaX * TOUCH_SCALE_FACTOR;
        }
        // Save current x, y
        previousX = currentX;
        previousY = currentY;
        return true; // Event handled
    }
}

```

Clearly, we use key 'A' and 'Z' for zoom out and zoom in. Touch events for modifying x and y rotational angles. Left, right, up and down buttons to control the x and y rotational speeds.

MyGLActivity.java

We need to modify our GL Activity to use the custom GL View.

```

package com.test;

import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
/*
 * OpenGL Main Activity.
 */
public class MyGLActivity extends Activity {
    private GLSurfaceView glView; // Use subclass of GLSurfaceView (NEW)

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Allocate a custom subclass of GLSurfaceView (NEW)
        glView = new MyGLSurfaceView(this);
        setContentView(glView); // Set View (NEW)
    }

    @Override
    protected void onPause() {
        super.onPause();
        glView.onPause();
    }

    @Override
    protected void onResume() {

```

```

        super.onResume();
        glView.onResume();
    }
}

```

You can, similarly, capture and process other events. [MORE]

2.11 Example 7b: Texture Filters (Nehe Lesson 7 Part 2: Texture Filter)

TextureCube.java

```

package com.test;

import java.io.IOException;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.opengles.GL10;
import javax.microedition.khronos.opengles.GL11;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLUtils;

/*
 * A cube with texture.
 * Three texture filters are to be set up.
 */
public class TextureCube {
    private FloatBuffer vertexBuffer; // Buffer for vertex-array
    private FloatBuffer texBuffer;    // Buffer for texture-coords-array

    private float[] vertices = { // Vertices for a face
        -1.0f, -1.0f, 0.0f, // 0. left-bottom-front
        1.0f, -1.0f, 0.0f, // 1. right-bottom-front
        -1.0f, 1.0f, 0.0f, // 2. left-top-front
        1.0f, 1.0f, 0.0f // 3. right-top-front
    };

    float[] texCoords = { // Texture coords for the above face
        0.0f, 1.0f, // A. left-bottom
        1.0f, 1.0f, // B. right-bottom
        0.0f, 0.0f, // C. left-top
        1.0f, 0.0f // D. right-top
    };

    int[] textureIDs = new int[3]; // Array for 3 texture-IDs (NEW)

    // Constructor - Set up the buffers
    public TextureCube() {
        // Setup vertex-array buffer. Vertices in float. An float has 4 bytes
        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
        vbb.order(ByteOrder.nativeOrder()); // Use native byte order
        vertexBuffer = vbb.asFloatBuffer(); // Convert from byte to float
        vertexBuffer.put(vertices);         // Copy data into buffer
        vertexBuffer.position(0);           // Rewind

        // Setup texture-coords-array buffer, in float. An float has 4 bytes
        ByteBuffer tbb = ByteBuffer.allocateDirect(texCoords.length * 4);
        tbb.order(ByteOrder.nativeOrder());
        texBuffer = tbb.asFloatBuffer();
        texBuffer.put(texCoords);
        texBuffer.position(0);
    }

    // Draw the shape
    public void draw(GL10 gl, int textureFilter) { // Select the filter (NEW)
        gl.glFrontFace(GL10.GL_CCW); // Front face in counter-clockwise orientation
        gl.glEnable(GL10.GL_CULL_FACE); // Enable cull face
        gl.glCullFace(GL10.GL_BACK); // Cull the back face (don't display)

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
    }
}

```

```

gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY); // Enable texture-coords-array
gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, texBuffer); // Define texture-coords buffer

// Select the texture filter to use via texture ID (NEW)
gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[textureFilter]);

// front
gl.glPushMatrix();
gl.glTranslatef(0.0f, 0.0f, 1.0f);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glPopMatrix();

// left
gl.glPushMatrix();
gl.glRotatef(270.0f, 0.0f, 1.0f, 0.0f);
gl.glTranslatef(0.0f, 0.0f, 1.0f);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glPopMatrix();

// back
gl.glPushMatrix();
gl.glRotatef(180.0f, 0.0f, 1.0f, 0.0f);
gl.glTranslatef(0.0f, 0.0f, 1.0f);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glPopMatrix();

// right
gl.glPushMatrix();
gl.glRotatef(90.0f, 0.0f, 1.0f, 0.0f);
gl.glTranslatef(0.0f, 0.0f, 1.0f);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glPopMatrix();

// top
gl.glPushMatrix();
gl.glRotatef(270.0f, 1.0f, 0.0f, 0.0f);
gl.glTranslatef(0.0f, 0.0f, 1.0f);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glPopMatrix();

// bottom
gl.glPushMatrix();
gl.glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
gl.glTranslatef(0.0f, 0.0f, 1.0f);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glPopMatrix();

gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
gl.glDisable(GL10.GL_CULL_FACE);
}

// Load an image and create 3 textures with different filters (NEW)
public void loadTexture(GL10 gl, Context context) {
    // Construct an input stream to texture image "res\drawable\crate.png"
    InputStream istream = context.getResources().openRawResource(R.drawable.crate);
    Bitmap bitmap;
    try {
        // Read and decode input as bitmap
        bitmap = BitmapFactory.decodeStream(istream);
    } finally {
        try {
            istream.close();
        } catch (IOException e) { }
    }
}

gl.glGenTextures(3, textureIDs, 0); // Generate texture-ID array for 3 textures (NEW)

// Create Nearest Filtered Texture and bind it to texture 0 (NEW)
gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[0]);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_NEAREST);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);

```

```

// Create Linear Filtered Texture and bind it to texture 1 (NEW)
gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[1]);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);

// Create mipmapped textures and bind it to texture 2 (NEW)
gl.glBindTexture(GL10.GL_TEXTURE_2D, textureIDs[2]);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_LINEAR_MIPMAP_NEAREST);
if(gl instanceof GL11) {
    gl.glTexParameterf(GL11.GL_TEXTURE_2D, GL11.GL_GENERATE_MIPMAP, GL11.GL_TRUE);
}
GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);

bitmap.recycle();
}
}

```

Dissecting TextureCube.java

[TODO]

MyGLRenderer.java

```

.....
public class MyGLRenderer implements GLSurfaceView.Renderer {
    .....
    int currentTextureFilter = 0; // Texture filter (NEW)
    .....

    @Override
    public void onDrawFrame(GL10 gl) {
        // Clear color and depth buffers
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        // ----- Render the Cube -----
        gl.glLoadIdentity(); // Reset the current model-view matrix
        gl.glTranslatef(0.0f, 0.0f, z); // Translate into the screen
        gl.glRotatef(angleX, 1.0f, 0.0f, 0.0f); // Rotate
        gl.glRotatef(angleY, 0.0f, 1.0f, 0.0f); // Rotate

        cube.draw(gl, currentTextureFilter); // (NEW)

        // Update the rotational angle after each refresh
        angleX += speedX;
        angleY += speedY;
    }
}

```

MyGLSurfaceView.java

```

.....
public class MyGLSurfaceView extends GLSurfaceView {
    .....
    // Handler for key event
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent evt) {
        switch(keyCode) {
            .....
            case KeyEvent.KEYCODE_DPAD_CENTER: // Select texture filter (NEW)
                renderer.currentTextureFilter = (renderer.currentTextureFilter + 1) % 3;
                break;
        }
    }
}

```

2.12 Example 7c: Lighting (Nehe Lesson 7 Part 3: Lighting)

MyGLRenderer.java

```

package com.test;

```

```

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

public class MyGLRenderer implements GLSurfaceView.Renderer {

    private Context context;
    private TextureCube cube;
    // For controlling cube's z-position, x and y angles and speeds
    float angleX = 0;
    float angleY = 0;
    float speedX = 0;
    float speedY = 0;
    float z = -6.0f;

    int currentTextureFilter = 0; // Texture filter

    // Lighting (NEW)
    boolean lightingEnabled = false; // Is lighting on? (NEW)
    private float[] lightAmbient = {0.5f, 0.5f, 0.5f, 1.0f};
    private float[] lightDiffuse = {1.0f, 1.0f, 1.0f, 1.0f};
    private float[] lightPosition = {0.0f, 0.0f, 2.0f, 1.0f};

    // Constructor
    public MyGLRenderer(Context context) {
        this.context = context;
        cube = new TextureCube();
    }

    // Call back when the surface is first created or re-created.
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set color's clear-value to black
        gl.glClearDepthf(1.0f); // Set depth's clear-value to farthest
        gl.glEnable(GL10.GL_DEPTH_TEST); // Enables depth-buffer for hidden surface removal
        gl.glDepthFunc(GL10.GL_LEQUAL); // The type of depth testing to do
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST); // nice perspective view
        gl.glShadeModel(GL10.GL_SMOOTH); // Enable smooth shading of color
        gl.glDisable(GL10.GL_DITHER); // Disable dithering for better performance

        // Setup Texture, each time the surface is created
        cube.loadTexture(gl, context); // Load image into Texture
        gl.glEnable(GL10.GL_TEXTURE_2D); // Enable texture

        // Setup lighting GL_LIGHT1 with ambient and diffuse lights (NEW)
        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_AMBIENT, lightAmbient, 0);
        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_DIFFUSE, lightDiffuse, 0);
        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_POSITION, lightPosition, 0);
        gl.glEnable(GL10.GL_LIGHT1); // Enable Light 1 (NEW)
        gl.glEnable(GL10.GL_LIGHT0); // Enable the default Light 0 (NEW)
    }

    // Call back after onSurfaceCreated() or whenever the window's size changes.
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        // NO CHANGE - SKIP
        .....
    }

    // Call back to draw the current frame.
    @Override
    public void onDrawFrame(GL10 gl) {
        // Clear color and depth buffers
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        // Enable Lighting? (NEW)
        if (lightingEnabled) {
            gl.glEnable(GL10.GL_LIGHTING);
        } else {
            gl.glDisable(GL10.GL_LIGHTING);
        }
    }
}

```

```

    }

    // ----- Render the Cube -----
    gl.glLoadIdentity();           // Reset the model-view matrix
    gl.glTranslatef(0.0f, 0.0f, z); // Translate into the screen
    gl.glRotatef(angleX, 1.0f, 0.0f, 0.0f); // Rotate
    gl.glRotatef(angleY, 0.0f, 1.0f, 0.0f); // Rotate
    cube.draw(gl, currentTextureFilter);

    // Update the rotational angle after each refresh
    angleX += speedX;
    angleY += speedY;
}
}
}

```

MyGLSurfaceView.java

```

.....
public class MyGLSurfaceView extends GLSurfaceView {
    .....

    // Handler for key event
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent evt) {
        switch(keyCode) {
            .....
            case KeyEvent.KEYCODE_L: // Toggle lighting on/off (NEW)
                renderer.lightingEnabled = !renderer.lightingEnabled;
                break;
        }
        .....
    }
}
}

```

2.13 Example 8: Blending (Nehe Lesson 8: Blending)

TextureCube.java

Use texture image "glass.png". Remove the culling of back face.

MyGLRenderer.java

```

package com.test;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

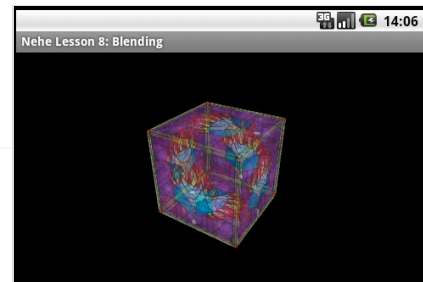
public class MyGLRenderer implements GLSurfaceView.Renderer {
    private Context context;
    private TextureCube cube;
    // For controlling cube's z-position, x and y angles and speeds
    float angleX = 0;
    float angleY = 0;
    float speedX = 0;
    float speedY = 0;
    float z = -6.0f;

    int currentTextureFilter = 0; // Texture filter

    // Lighting
    boolean lightingEnabled = false;
    private float[] lightAmbient = {0.5f, 0.5f, 0.5f, 1.0f};
    private float[] lightDiffuse = {1.0f, 1.0f, 1.0f, 1.0f};
    private float[] lightPosition = {0.0f, 0.0f, 2.0f, 1.0f};

    // Blending (NEW)
    boolean blendingEnabled = false; // Is blending on? (NEW)
}

```



```

// Constructor
public MyGLRenderer(Context context) {
    this.context = context;
    cube = new TextureCube();
}

// Call back when the surface is first created or re-created.
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set color's clear-value to black
    gl.glClearDepthf(1.0f); // Set depth's clear-value to farthest
    gl.glEnable(GL10.GL_DEPTH_TEST); // Enables depth-buffer for hidden surface removal
    gl.glDepthFunc(GL10.GL_LEQUAL); // The type of depth testing to do
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST); // nice perspective view
    gl.glShadeModel(GL10.GL_SMOOTH); // Enable smooth shading of color
    gl.glDisable(GL10.GL_DITHER); // Disable dithering for better performance

    // Setup Texture, each time the surface is created
    cube.loadTexture(gl, context); // Load image into Texture
    gl.glEnable(GL10.GL_TEXTURE_2D); // Enable texture

    // Setup lighting GL_LIGHT1 with ambient and diffuse lights
    gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_AMBIENT, lightAmbient, 0);
    gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_DIFFUSE, lightDiffuse, 0);
    gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_POSITION, lightPosition, 0);
    gl.glEnable(GL10.GL_LIGHT1); // Enable Light 1
    gl.glEnable(GL10.GL_LIGHT0); // Enable the default Light 0

    // Setup Blending (NEW)
    gl.glColor4f(1.0f, 1.0f, 1.0f, 0.5f); // Full brightness, 50% alpha (NEW)
    gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE); // Select blending function (NEW)
}

// Call back after onSurfaceCreated() or whenever the window's size changes.
@Override
public void onSurfaceChanged(GL10 gl, int width, int height) {
    // NO CHANGE - SKIP
    .....
}

// Call back to draw the current frame.
@Override
public void onDrawFrame(GL10 gl) {
    // Clear color and depth buffers
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

    // Enable lighting?
    if (lightingEnabled) {
        gl.glEnable(GL10.GL_LIGHTING);
    } else {
        gl.glDisable(GL10.GL_LIGHTING);
    }

    // Blending Enabled? (NEW)
    if (blendingEnabled) {
        gl.glEnable(GL10.GL_BLEND); // Turn blending on (NEW)
        gl.glDisable(GL10.GL_DEPTH_TEST); // Turn depth testing off (NEW)
    } else {
        gl.glDisable(GL10.GL_BLEND); // Turn blending off (NEW)
        gl.glEnable(GL10.GL_DEPTH_TEST); // Turn depth testing on (NEW)
    }

    // ----- Render the Cube -----
    gl.glLoadIdentity(); // Reset the model-view matrix
    gl.glTranslatef(0.0f, 0.0f, z); // Translate into the screen
    gl.glRotatef(angleX, 1.0f, 0.0f, 0.0f); // Rotate
    gl.glRotatef(angleY, 0.0f, 1.0f, 0.0f); // Rotate
    cube.draw(gl, currentTextureFilter);

    // Update the rotational angle after each refresh
    angleX += speedX;
    angleY += speedY;
}

```

```

    }
}

```

MyGLSurfaceView.java

```

.....
public class MyGLSurfaceView extends GLSurfaceView {
    .....

    // Handler for key event
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent evt) {
        switch(keyCode) {
            .....
            case KeyEvent.KEYCODE_B: // Toggle Blending on/off (NEW)
                renderer.blendingEnabled = !renderer.blendingEnabled;
                break;
        }
        .....
    }
}

```

2.14 Example 8a: Bouncing Ball in Cube

[TODO] No primitive to draw a sphere in OpenGL ES.

3. Android Port for Nehe's Lessons

I have ported some of the Nehe's lessons into Android. Refer to [Nehe](#) for the problem descriptions.

Setting Up:

- Nehe's Lesson #1: Setting up OpenGL's window. (Refer to above examples.)

OpenGL Basics: I consider Lessons 2-8 as OpenGL basic lessons, that are extremely important! (Refer to above examples.)

- Nehe's Lesson #2: Your first polygon
- Nehe's Lesson #3: Adding Color
- Nehe's Lesson #4: Rotation
- Nehe's Lesson #5: 3D Shape
- Nehe's Lesson #6: Texture
- Nehe's Lesson #7: Texture Filter, Lighting, and key-controlled
- Nehe's Lesson #8: Blending

Intermediate: [TODO]

REFERENCES & RESOURCES

- Introducing GLSurfaceView @ <http://developer.android.com/resources/articles/glsurfaceview.html>.
- GLSurfaceView reference @ <http://developer.android.com/reference/android/opengl/GLSurfaceView.html>.
- OpenGL ES 1.1 Reference Pages @ <http://www.khronos.org/opengles/sdk/1.1/docs/man>.
- android.opengl package API @ <http://developer.android.com/reference/android/opengl/package-summary.html>.
- Android's API Demos ⇒ Graphics ⇒ OpenGL ES.
- Nehe OpenGL Lessons @ <http://nehe.gamedev.net>.
- OpenGL "Red" book, "Blue" book.

Latest version tested: Android SDK 2.2

Last modified: October 18, 2010

Feedback, comments, corrections, and errata can be sent to Chua Hock-Chuan (ehchua@ntu.edu.sg) | [HOME](#)