Android Graphics

1 Introduction

If you were not shut off from the world in the past year, you must have heard about Android. Development in communication has been shrinking the world and thanks to the advancement of Wi Fi technologies, some developing countries are able to leapfrog into the twenty-first century without the burden of dismantling the infra structure and equipments of wired communication. Mobile devices have become ubiquitous and even more sophisticated than PCs. Transitioning from working with PCs to mobile devices such as smart phones and tablet computers has become a trend. Android is developed in response to this trend. It is an open-source operating system based on the Linux kernel with applications developed using the java programming language . The Android operating system was first developed by Android, Inc. Google acquired Android, Inc. in July 2005, and became the leading developer of the Android OS. In November 2007, the Open Handset Alliance, which initially was a consortium of 34 companies formed to develop Android. The consortium was later expanded to absorb many more companies in a joint effort to further develop Android, which is the real innovation in the mobile technology and an improvement in the our life. Because of its openness, in less than two years, Android has come from nowhere to become the dominant smart phone operating system in the US, eclipsing all other major players in the field. News, details and relevant links of Android can be found from its official web site,

http://www.android.com/

Wikipedia has a good article about the history and miscellaneous information of Android at

http://en.wikipedia.org/wiki/Android_(operating_system)

Technical information can obtained from Android's official developer web site at:

http://developer.android.com/guide/index.html

This site provides a lot of information and how-to for developing Android applications and publishing them. The site defines Android as follows:

> *Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.*

The developer site also lists various features of Android, including

1. **Application framework** enabling reuse and replacement of components
2. **Dalvik virtual machine** optimized for mobile devices
3. **Integrated browser** based on the open source WebKit engine
4. **Optimized graphics** powered by a custom 2D graphics library

5. **3D graphics** based on the OpenGL ES 1.0 and ES 2.0 specifications
6. **SQLite** for structured data storage
7. **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
8. **GSM Telephony** (hardware dependent)
9. **Bluetooth, EDGE, 3G**, and **WiFi** (hardware dependent)
10. **Camera, GPS, compass**, and **accelerometer** (hardware dependent)
11. **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

The following figure shows the architecture of Android.

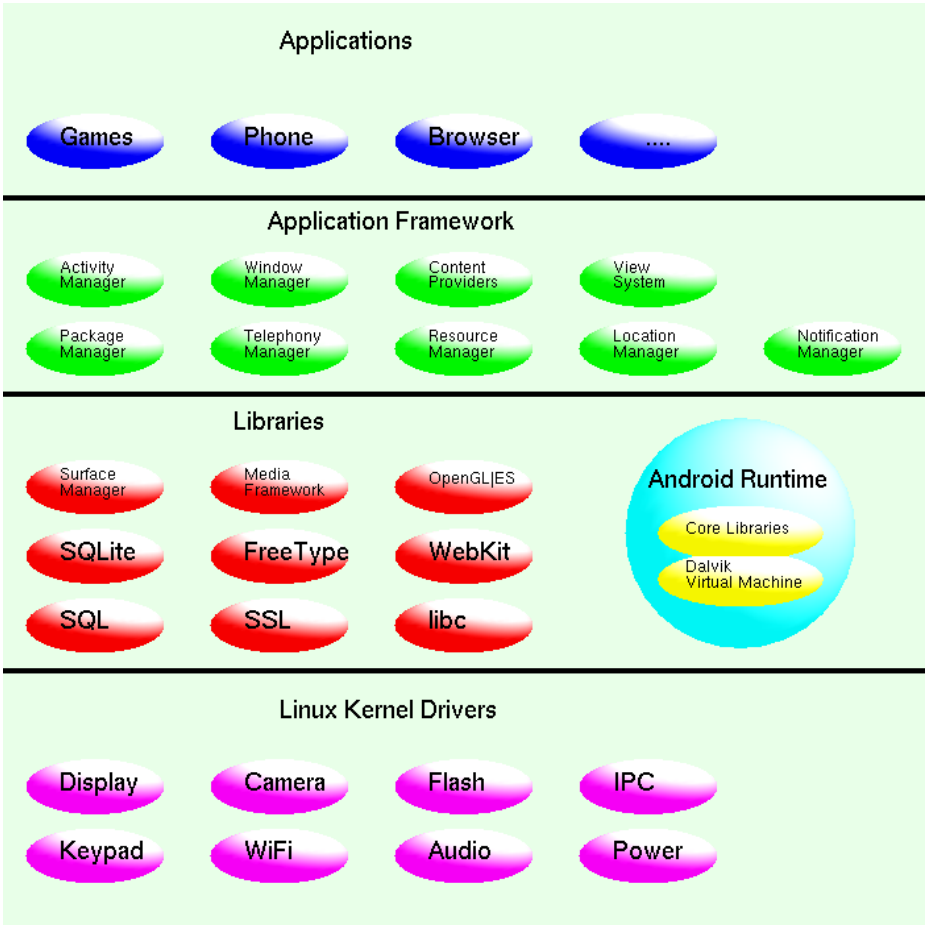

Figure C-1 Android Architecture

In summary, Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to develop applications on the Android platform using the java programming language. Moreover, Android includes a set of C/C++ libraries that can be used by various components of the Android system. These capabilities are exposed to developers through the Android application framework as shown in Figure C-1.

Figure C-1 also shows that an Android application runs in its own process, with its own instance of the Dalvik virtual machine (VM). The Dalvik Executable (.dex) format is used to execute files in the Dalvik VM; the format is optimized for minimal memory footprint. After a java program has been compiled, the classes will be transformed into the *.dex* format by the **dx** tool so that it can be run in the Dalvik VM. The Linux kernel provides underlying functionality such as threading and low-level memory management for the Dalvik VM.

Like the rest of Android, Dalvik is open-source software and is published under the terms of the Apache License 2.0. Dalvik is known to be a clean-room implementation rather than a development on top of a standard java runtime. This could mean that it does not inherit copyright-based license restrictions from either the standard-edition or open-source-edition java runtimes.

Android has undergone several versions of revision. Each new version is named after a desert:

1. Android 1.6 (Donut)
2. Android 2.02.1 (Eclair)
3. Android 2.2 (Froyo)
4. Android 2.3 (Gingerbread)
5. Android 3.0 (Honeycomb)

As of this writing, the latest version is 4.0 (Android Icecream Sandwich), which was released in November, 2011. This version merges Android 2.3 (Gingerbread) and Android 3.0 (Honeycomb) into one operating system for use on all Android devices. This will allow us to incorporate Honeycombs features such as the holographic user interface, new launcher and more (previously available only on tablets) into our smart phone apps, and easily scale our apps to work on different devices. Ice Cream Sandwich will also add new functionality.

We encountered some minor problems in running Android 3.0 and 4.0 emulators in some platforms, especially in 64-bit machines. It seems that versions 2.2 and 2.3 are the most stable versions with richest features at this moment. The Android programs presented in this book are developed and tested with Android version 2.3.3, API level 10.

Though Android apps are written in java, 3D graphics programs are written with OpenGL ES. Actually, our main concern here is to explain the principles of 3D graphics. The graphics functions, OpenGL commands, that we are going to use are open standards in the industry. They have the same form and syntax, whether they are presented in C/C++ or Android Java.

2 Development Tools

The Android official site provides the information and tools to develop Android applications. One can refer to the site

http://developer.android.com/index.html

to learn the details and download the development tools. The following link shows how to install Android and set up the development environment for the first time:

http://developer.android.com/sdk/installing.html

Since Android applications are written in java, in most situations developing an Android application is simply writing some java programs utilizing the Android libraries. The programs can be compiled and built with use of *Apache Ant*, a software tool for automating software build processes. Ant is similar to *Make* but it is implemented in java, and is best suited to building java projects. However, it is more convenient to do the development using **Eclipse**, a multi-language open *software development environment* consisting of an *integrated development environment* (IDE) along with an extensible plug-in system. Eclipse is mostly written in java, and is an ideal IDE for developing java applications; it can be also used to develop applications of other programming languages such as C/C++ and PHP by means of various plug-ins.

Eclipse

One can obtain information of Eclipse and download it from its official web site at

http://www.eclipse.org/

Eclipse can be easily installed and run in any supported platform. Using Linux as an example, the following steps show how to install Eclipse along with the Android Development Tools (ADT).

1. Go to *http://www.eclipse.org/*; click **Download Eclipse**; choose **Eclipse for RCP and RAP Developers: Linux 32 Bit**, and download the package into a local directory, say, */apps/downloads*.
2. Unpack the downloaded package into the directory */apps* by:

   ```
   $ cd /apps
   $ gunzip -c /apps/download/eclipse-rcp-helios-SR2-linux-gtk.tar.gz | tar xvf -
   ```

3. Then start Eclipse by:

   ```
   $ cd eclipse
   $ ./eclipse
   ```

4. From the eclipse IDE, install the **Android Development Tools** (ADT):
   - Click**Help** $>$ **Install New Software**
   - In the "Work with" box, type *http://dl-ssl.google.com/android/eclipse/*; hit "Enter"; select all the "Development Tools"; click **Next**; click **Next**; accept the license "agreement to", and click **Finish** to install ADT.
5. After the ADT installation, restart Eclipse.
6. Click **Window** and you should see the entry **Android SDK and AVD Manager**.
7. Add the Android SDK directory by clicking **Preference**; select **Android** and enter the location of your Android SDK. Now click on **Android SDK and AVD Manager** to proceed.
8. If you are new to eclipse, click on **Tutorials** and follow the instructions to create a **Hello World** application.

As an example of writing Android applications in the Eclipse IDE, we present the steps of writing a **Hello World** application. In this example, we will run the application in the Android Emulator. You can also find this example at the Android tutorial web site at

http://developer.android.com/resources/tutorials/hello-world.html

In the description, we use the specified Android version 2.3.3.

1. Start Eclipse.
2. In the Eclipse IDE, choose **Preferences** > **Android**.
3. Install a platform in Eclipse:
   (a) Choose **Window** > **Android SDK Manager**, which displays a panel showing the Android platform packages in your system.
   (b) As an example, choose **Android 2.3.3(API 10)** and its subcomponents "SDK Platform" and "SDK Samples"; then click **Install 2 packages**; check **Accept All**; click **Install**. Eclipse will download the package from the Internet and install it.
   (c) When it is finished you can press the key 'ESC' to clear the panel.
4. Create an Android Virtual Device (AVD), which defines the system image and device settings of the emulator:
   (a) In Eclipse, choose **Window** > **AVD Manager**, which displays the *Android Virtual Device Manager* panel.
   (b) Click **New..**, which displays the *Create new Android Virtual Device (AVD)* dialog.
   (c) Enter a name for the AVD, say, "avd233".
   (d) Select the target to be *Android 2.3.3 – API Level 10*.
   (e) Click **Create AVD**.
   (f) Press 'ESC' to exit the AVD panel.
5. Create a new Android project:
   (a) In Eclipse, select **File** > **New** > **Project**, which displays the *New Project* dialog as shown in Figure C-2 below.

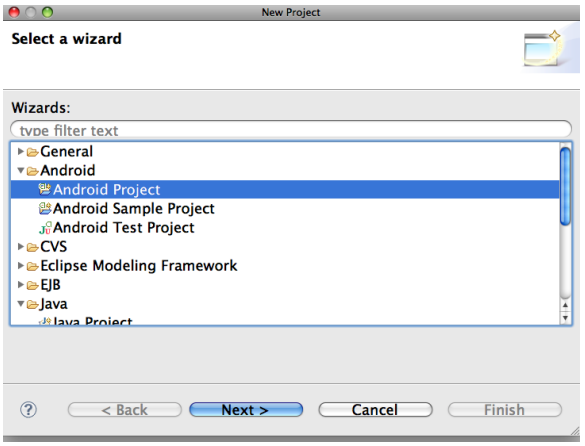

Figure C-2 Eclipse New Project Dialog

   (b) Select **Android Project** and click **Next**, which displays the *New Android Project* dialog.
   (c) Enter "HelloWorldProject" *Project Name* and click **Next**.
   (d) Choose *Build Target* to be *Android 2.3.3* and click **Next**.
   (e) Enter "HelloWorld" for *Application Name*, "android.hello" for *Package Name*, "HelloWorld" for *Create Activity*, **10(Android 2.3.3)** for Minimum SDK and click **Finish**.

The *HelloWorldProject* Android project is now ready. It should be visible in the *Package Explorer* on the left of the Eclipse IDE. (You may need to click **Plug-in Device..** on the left to display **Package Explorer**.

6. From the **Package Explorer**, choose **HelloWorldProject** > **src** > **android.hello** > **HelloWorld.java**. Double-click on **HelloWorld.java** to open it, which should look like the following:

```
package android.hello;

import android.app.Activity;
import android.os.Bundle;

public class HelloWorld extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }
}
```

7. Revise "HelloWorld.java" to the following that constructs a user interface (UI):

```
package android.hello;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class HelloWorld extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
       TextView tv = new TextView(this);
       tv.setText("Hello, Android!");
       setContentView(tv);
  }
}
```

8. Run the application by choosing from Eclipse, **Run** > **Run**. Then select **Android Application** and click **OK**. The Android emulator will start and run the application. You should see on your screen something like Figure C-3 below.

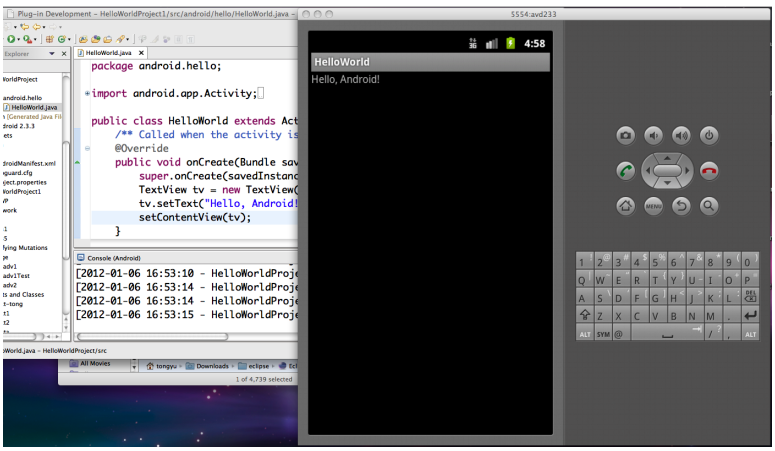

Figure C-3 HelloWorld Android Application

If you do not see the message "Hello, Android!", click the **menu** button of the Android emulator which will run the application.

3 OpenGL ES

The Android framework supports both the OpenGL ES 1.0/1.1 and OpenGL ES 2.0 APIs. OpenGL ES, where ES is short for embedded system, is a flavor of the OpenGL specification tailored for embedded devices. OpenGL ES is royalty-free and cross-platform. Its APIs have full-function support for 2D and 3D graphics on embedded systems such as mobile phones and appliances. OpenGL ES 1.X uses the traditional fixed-pipeline architecture and emphasizes hardware acceleration of the API. It offers enhanced functionality, good image quality and high performance. OpenGL ES 2.X is for programmable hardware. It emphasizes a programmable 3D graphics pipeline and allows the user to create shader and program objects. With ES 2.X, one can also write vertex and fragment shaders in the OpenGL ES Shading Language. On the other hand, OpenGL ES 2.0 does not support the fixed function transformation and fragment pipeline that OpenGL ES 1.x supports.

Since Android 1.0, the Android framework has supported the OpenGL ES 1.0 and 1.1 API specifications. Starting from Android 2.2 (API Level 8), the framework supports the OpenGL ES 2.0 API specification. One can find the API specifications at the site,

http://developer.android.com/guide/topics/graphics/opengl.html

However, the Android emulator does not support OpenGL ES 2.0. Therefore, our discussion here is based on OpenGL ES 1.X, which has certain limitations. In particular, it does not support direct vertex handling. For example, there are no **glBegin/glEnd** and **glVertex*** functions. Some constants such as GL_POLYGONS and GL_QUADS are missing.

We will discuss an example of using OpenGL ES 1.0 in Android. This example can be found in the tutorial section of the Android developer site at

http://developer.android.com/resources/tutorials/opengl/opengl-es10.html

Create an Activity with GLSurfaceView

Before we start using OpenGL to create graphics in Android, we have to implement the class *GLSurfaceView*, which extends *SurfaceView* and the class *GLSurfaceView.Renderer*, which is responsible for making OpenGL calls to render a frame. Using Android 2.3.3 (Level 10) as an example, we first show how to create an activity with *GLSurfaceView*:

1. In Eclipse, choose **File** > **New** > **Project** > **Android Project** and click **Next**.

2. Enter "HelloES" for *Project Name*, and click **Next**; choose *Android 2.3.3* for *Build Target* and click **Next**.

3. Enter the following information for the *New Android Project*:

Application Name:	HelloES
Package Name:	opengl.es10
Create Activity:	HelloESActivity
Minimum SDK:	10 (Android 2.3.3)

Then click **Finish** to create the new project.

4. Project **HelloES** should appear in *Package Explorer* of Eclipse. Choose **HelloES** > **src** > **opengl.es10** > **HelloESActivity.java** to open the file "HelloESActivity.java". Modify this file as follows:

```
package opengl.es10;

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

public class HelloESActivity extends Activity {
  private GLSurfaceView mGLView;

  @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Create a GLSurfaceView instance and set it
      // as the ContentView for this Activity.
      mGLView = new HelloESSurfaceView(this);
      setContentView(mGLView);
  }

  @Override
  protected void onPause() {
      super.onPause();
      // The following call pauses the rendering thread.
      mGLView.onPause();
  }

  @Override
  protected void onResume() {
      super.onResume();
      // The following call resumes a paused rendering thread.
      mGLView.onResume();
  }

  class HelloESSurfaceView extends GLSurfaceView {

      public HelloESSurfaceView(Context context){
          super(context);

          // Set the Renderer for drawing on the GLSurfaceView
          setRenderer(new HelloESRenderer());
      }
  }
}
```

Note that you should see an error indicator at *setRenderer(new HelloESRenderer());* . This is because up to this point, we have not defined the class *HelloESRenderer*.

In the *HelloESActivity* class shown above, we use a single *GLSurfaceView* for its view; the class also implements callbacks for pausing and resuming activities. The *HelloESSurfaceView* class is responsible for setting the renderer to draw on the *GLSurfaceView*.

5. In Eclipse, choose **File** > **New** > **File** and enter "HelloESRenderer.java" for *File*

name and click **Finish** to create a new file for the following class *HelloESRenderer*, which implements the *GLSurfaceView.Renderer* interface:

```
package opengl.es10;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

import android.opengl.GLSurfaceView;

public class HelloESRenderer implements GLSurfaceView.Renderer {

  public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    // Set the background frame color to blue
    gl.glClearColor(0.0f, 0.0f, 0.9f, 1.0f);
    // 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);
  }

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

6. Now you can run the application by choosing **Run** > **Run** > **Android Application** and click **OK**. The Android emulator will start and will show a blue background screen. The code above is pretty much self-explained. The functions **glClear**(), **glClearColor**(), and **glViewport**() are the standard OpneGL commands we have discussed. The only functions new to us are the few used by Android to do the initialization, which include the following:

   - **onSurfaceCreated**() is called once for setting up the *GLSurfaceView* environment.
   - **onDrawFrame**() is called whenever we redraw the *GLSurfaceView*. This is similar to the *display*() function we have used in our OpenGL C programs.
   - **onSurfaceChanged**() is called when the geometry of the **GLSurfaceView** changes. This is similar to the **glutPostRedisplay**() used in C programs.

Draw a Triangle on GLSurfaceView

With the template code provided above, we should be able to make 2D or 3D graphics with our knowledge of OpenGL. We discuss here how draw to a triangle.

By default, OpenGL ES assumes a world coordinate system where the center of the *GLSurfaceView* frame is at $(0, 0, 0)$; the coordinates of the lower left corner and upper right corner are at $(-1, -1, 0)$ and $(1, 1, 0)$ respectively. Therefore, as an example, we specify a triangle with the following vertex coordinates:

$$(-0.6, -0.5, 0), (0.6, -0.5, 0), (0.0, 0.5, 0)$$

We will display this triangle with green color on the blue background. To accomplish this, we modify the *HelloESRenderer* class as follows:

```
package opengl.es10;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.opengl.GLSurfaceView;

public class HelloESRenderer implements GLSurfaceView.Renderer {

  private FloatBuffer triangle;

  public void onSurfaceCreated(GL10 gl, EGLConfig config) {
      // Set the background frame color to blue
      gl.glClearColor(0.0f, 0.0f, 0.9f, 1.0f);
      // 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);
      // Draw the triangle using green color
      gl.glColor4f(0.0f, 1.0f, 0.0f, 0.0f);
      gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangle);
      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[] = {
          // (x, y, z) of triangle
          -0.6f, -0.5f, 0,
           0.6f, -0.5f, 0,
           0.0f,  0.5f, 0
      };

      // initialize vertex Buffer for triangle
      // argument=(# of coordinate values * 4 bytes per float)
      ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
      // use the device hardware's native byte order
      vbb.order(ByteOrder.nativeOrder());
      // create a floating point buffer from the ByteBuffer
      triangle = vbb.asFloatBuffer();
      // add the coordinates to the FloatBuffer
      triangle.put(vertices);
      // set the buffer to read the first vertex coordinates
      triangle.position(0);
  }
}
```

We can recompile our graphics application by first clicking on the file **HelloESActivity.java** and then choosing **Run** in Eclipse. The Android emulator will run the application and we should see a green triangle displayed on a blue background as shown in Figure C-4

below.

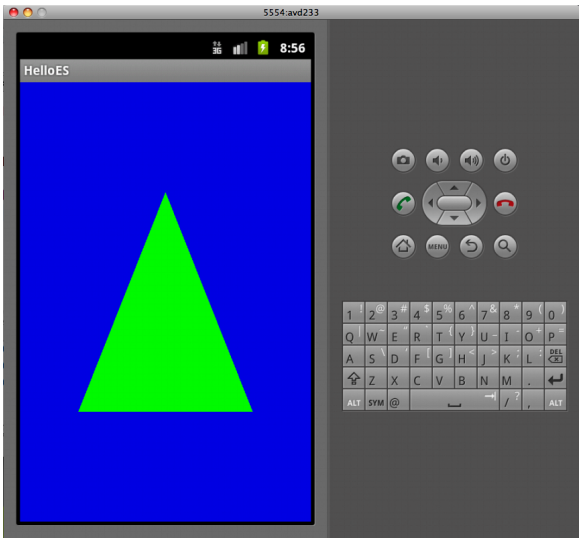

Figure C-4 HelloES Graphics Output

Setting Camera View and Transformations

Just as we do in a usual C program, we use **gluLookAt**() to set the camera view and position. The modelview transformation and projection transformation functions discussed in Chapter 3 and Chapter 4 also apply here. As an example, we modify the functions **onDrawFrame**() and **onSurfaceChange**() of the class *HelloESRenderer* as follows to include projection and modelview transformations; our camera is at $(0, 0, 5)$ viewing along the negative z direction. We also scale the triangle by a factor of 3 along the y-direction and rotate it about the z-axis by 30^o. (Note again that we first scale, then rotate.)

```
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 its default state

   // When using GL_MODELVIEW, you must set the view point.
   //   camera at (0, 0, 5) look at (0,0,0), up = (0, 1, 0)
   GLU.gluLookAt(gl, 0, 0, 5, 0f, 0f, 0f, 0f, 1.0f, 0.0f);
   //rotate about z-axis for 30 degrees
   gl.glRotatef(30, 0, 0, 1);
   //magnify triangle by x3 in y-direction
   gl.glScalef ( 1, 3, 1);
   // Draw the triangle
   gl.glColor4f(0.0f, 1.0f, 0.0f, 0.0f);
   gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangle);
   gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
}

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

   float aratio = (float) width / height;  //aspect ratio
```

```
    float l, r, b, t, n, f;          //left,right,bottom,top,near,far
    b = -1.5f; t = 1.5f; n = 3.0f; f = 7.0f;
    l =  b * aratio; r = t * aratio;

    gl.glMatrixMode(GL10.GL_PROJECTION); //set matrix to projection mode
    gl.glLoadIdentity();                 // reset the matrix
    gl.glFrustumf( l, r, b, t, n, f);    // apply the projection matrix
  }
```

We also need to include the header statement "import android.opengl.GLU;" at the beginning of the file as we need to use the function **GLU.gluLookAt**(). After the modification, we can run the program **HelloESActivity.java** again. The output of the program is shown in Figure C-5.

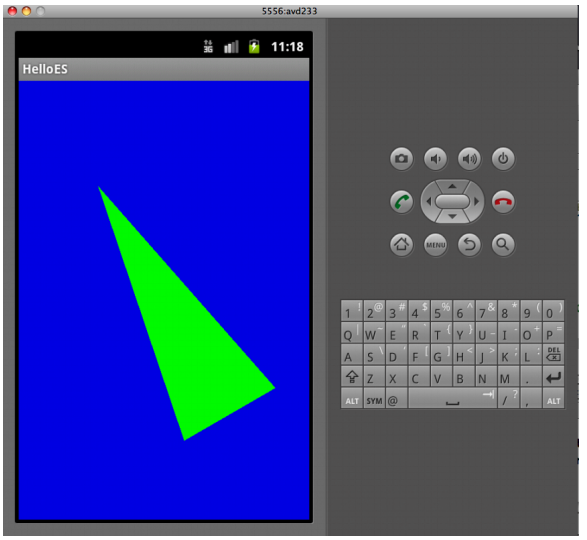

Figure C-5 HelloES Output with Projection and Modelview Transformations

4 Animation and Event Handling

We can make use of the methods (functions) provided by the Android class *SystemClock* to create animated graphics. The class consists of core timekeeping facilities. To use the methods. we have to import the class by adding the header statement,

import android.os.SystemClock;

There are three clocks we can use to keep time:

1. **currentTimeMillis**() gives the current time and date expressed in milliseconds since the epoch. This clock can be set by the user or the phone network.

2. **uptimeMillis**() gives the active time lapse in milliseconds since the system was booted. This clock stops when the process is in a blocked or a sleep state such as waiting for an I/O event or executing **Thread.sleep**().

3. **elapsedRealtime**() gives the counts in milliseconds since the system was booted, including the time when the process is blocked or in a sleep state.

There are several ways to control the timing events in an animation process:

1. **Thread.sleep**(*millis*) and **Object.wait**(*millis*) are standard blocking functions that can be used to generate desired time delays. When these functions are executed, the **uptimeMillis**() clock stops. The thread can be woken up by the function **Thread.interrupt**().
2. **SystemClock.sleep**(*millis*) is a utility function very similar to **Thread.sleep**(*millis*), except that it ignores **InterruptedException**.
3. We can use the *Handler* class to schedule asynchronous callbacks at an absolute or relative time. A handler object uses the **uptimeMillis**() clock to keep time. It requires an event loop to wait for an event to happen.
4. We can use the *AlarmManager* class to access the system alarm services such as triggering one-time or recurring events when the thread is in a blocked state.

As an example, let us rotate the triangle of Figure C-5 discussed above for 6^o every second. To accomplish this, all we need to do is to add to the class *HelloESRenderer* a data member *angle* of type float:

public float angle = 0.0f;

Then we make the following modifications to the code of its member function **onDrawFrame**():

```
public void onDrawFrame(GL10 gl) {
  ....
  GLU.gluLookAt(gl, 0, 0, 5, 0f, 0f, 0f, 0f, 1.0f, 0.0f);
  SystemClock.sleep ( 1000 );  //delay for 1 second
  angle += 6;        //increment angle by 6 degrees
  //rotate triangle about z-axis
  gl.glRotatef(angle, 0.0f, 0.0f, 1.0f);
  //magnify triangle by x3 in y-direction
  gl.glScalef ( 1, 3, 1);
  // Draw the triangle
  .....
}
```

When we run the modified program, we will see the triangle of Figure C-5 rotating anti-clockwise 6^o every second.

If we want the triangle to interact with us rather than rotating automatically, we need to expand our implementation of *GLSurfaceView* to override the **onTouchEvent**() function to listen for touch events. Since we have defined above the data member *angle* of the *HelloESRenderer* class to be public, the member is exposed to other classes. We just need to modify the *HelloESSurfaceView* class to process touch events and pass the data to the renderer. To accomplish this, we have to include the import statement,

import android.view.MotionEvent;

In the **onDrawFrame**() function, we just comment out the time delay and increment statements as the *angle* value is determined by touch events:

```
public void onDrawFrame(GL10 gl) {
  ....
  // SystemClock.sleep ( 1000 );  //delay for 1 second
  // angle += 6;

  gl.glRotatef(angle, 0.0f, 0.0f, 1.0f);
}
```

Then we modify the *HelloESSurfaceView* class (in the file "HelloESActivity.java") as follows. We set the *renderer* member so that we have a handle to pass in rotation input and set the render mode to RENDERMODE_WHEN_DIRTY. We also override the **onTouchEvent**() method to listen for touch events and pass the parameters to our renderer:

```
class HelloESSurfaceView extends GLSurfaceView {
  private final float TOUCH_SCALE_FACTOR = 180.0f / 320;
  private HelloESRenderer renderer;
  private float previousX;
  private float previousY;

  public HelloESSurfaceView(Context context){
    super(context);
    // set the renderer member
    renderer = new HelloESRenderer();
    setRenderer(renderer);

    // Render the view only when there is a change
    setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
  }
  @Override
  public boolean onTouchEvent(MotionEvent e) {
    // MotionEvent reports input details from the touch screen
    // and other input controls. Here, we are only interested
    // in events where the touch position has changed.

    float x = e.getX();
    float y = e.getY();

    switch (e.getAction()) {
      case MotionEvent.ACTION_MOVE:

      float dx = x - previousX;
      float dy = y - previousY;

      // reverse direction of rotation above the mid-line
      if (y > getHeight() / 2)
        dx = dx * -1 ;

      // reverse direction of rotation to left of the mid-line
      if (x < getWidth() / 2)
        dy = dy * -1 ;

        renderer.angle += (dx + dy) * TOUCH_SCALE_FACTOR;
        requestRender();
    }

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

When we run the application, we should see the green triangle again. If we drag our mouse and move its cursor around the center in an anticlockwise direction, the triangle will rotate in a clockwise direction. Conversely, if we drag the mouse clockwise, the triangle rotates anticlockwise.

In summary, this chapter gives you an introduction to create 2D and 3D graphics in the Android platform using OpenGL. You can find a lot more examples and resources of creating graphics in Android at its official site and associated links at

http://developer.android.com/guide/topics/graphics/opengl.html

The Android open-source project has been creating huge opportunities for many people. It is joyful to make contributions to its growth and development.