cisc3650, spring 2012, lab V.1 / prof sklar
Android Graphics, part 2

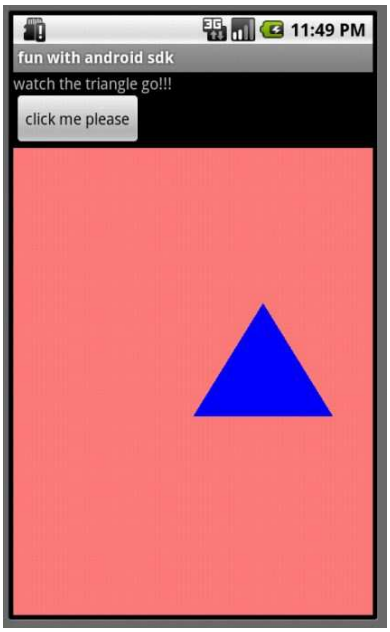
## overview

If you have completed all the Android labs, then you will have gone through all the basic Tutorials on the Android Developer site. If you have not completed these, then you should go back and finish them.

This lab is about combining aspects of these lab exercises, to help you better understand components of Android on your own.

## exercise

The goal of this exercise is to make an Android app that looks like this:




This combines a `LinearLayout` with a `TextView`, a `Button` and a custom `GLSurfaceView`.

Refer to the earlier Android labs where you went through the Tutorials that created the following projects: `HelloLinearLayout`, `HelloDatePicker`, and `HelloOpenGLES10`.

There are main two steps to this exercise. The first step is to get the layout looking the way you want it to. The second step is to get the interface to behave the way you want it to.

# (1) LOOK

Something you need to know about in order to get the application to look like the example, above, has to do with the relationship between layout elements that are created in the `main.xml` resource file and the Java source code file.

Look at `HelloOpenGLES10.java`. Here, you create an `HelloOpenGLES10SurfaceView` object, which is defined by a class that extends the (imported) `android.opengl.GLSurfaceView` class. The line:

```
setContentView( myGLView );
```

sets the entire layout to contain this single view (i.e., extended surface view).

Whereas, if you look at `HelloDatePicker.java`, you'll see the line:

```
setContentView( R.layout.main );
```

which sets the layout according to what is defined in `main.xml`.

In contrast, the example above mixes widgets defined in `main.xml` (a `TextView` and a `Button` object) with the custom surface view widget that is created in the Java code.

So, you will need to know how to mix widgets that are created in different places. The trick is to leave the standard widgets in the `main.xml` resource file and to create the custom widgets in the Java code. Set the layout in the Java code to `main.xml`, and then instantiate an object containing a reference to the layout (a `LinearLayout` in my example, with identifier property set to `android:id="@id/linearlayout1"+`) that covers the device:

```
setContentView( R.layout.main );
LinearLayout myLL = (LinearLayout)findViewById( R.id.linearlayout1 );
```

Then you can create your custom widget and add it to that layout in the Java code:

```
myGLView = new MixOpenGLES10SurfaceView( this );
myLL.addView( myGLView );
```

## (2) BEHAVE

In my example, the triangle is animated and moves back and forth horizontally across the display. Clicking on the button toggles the animation. In other words: when the app starts up, the triangle is moving. When you click on the button, the triangle stops. When you click on the button again, the triangle starts moving again.

This means creating a `View.OnClickListener()` and an `onClick()` event handler for the `Button` that you have defined in `main.xml` (refer to the `HelloDatePicker` example). Then you have to connect the actions in the `onClick()` event handler to the custom surface view where your triangle is animating.

## (3) EXPLORE AND HAVE FUN!!!

After you get this working, explore! Go back to the examples you have created and try combining more widgets.

Go to the Android Developer Resources page and go through some of the more advanced examples and articles.