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

overview

This lab consists of several exercises that will comprise part of the homework assignment for Unit V.

Refer to the "OpenGL ES 1.0" Tutorial, below:

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

exercises

Each of the following exercises is based on portions of the "OpenGL ES 1.0" Tutorial (URL above), as indicated.

1. **Create an Activity with GLSurfaceView**

   Work through the "Create an Activity with GLSurfaceView" portion of the Tutorial.

   When you run the example, you'll just see a grey background in the Android emulator.

   Modify the example as follows:
   - Change the background color.
   - Add a message to display in the LogCat window of Eclipse, using the `Log.d()` function, like this:

     ```
     public void onCreate(Bundle savedInstanceState) {
        Log.d( "HelloOpenGLES10", "in onCreate()" );
       .
       .
       .
     }
     ```

     which is in the *HelloOpenGLES10.java* file. The `Log.d()` function takes two arguments. The first argument is a string that appears in the "Tag" column of the LogCat window. The second argument is a string that appears in the "Text" column of the LogCat window. This is a useful way to display debugging messages while your code runs.

     Note that you have to import the Log package in order to use this function:

     ```
     import android.util.Log;
     ```

2. **Draw a Shape on GLSurfaceView**

   Continue with the Tutorial, working through the "Draw a Shape on GLSurfaceView, Draw the Triangle" section.

   Modify the example as follows:
   - Change the color of the triangle.
   - Change the size of the triangle.
   - Change the position of the triangle.

3. **Apply Projection and Camera View**

   Continue with the Tutorial, working through the "Apply Projection and Camera View" section.

   This function call:

   ```
   GLU.gluLookAt( gl, 0, 0, -5, 0f, 0f, 0f, 0f, 1.0f, 0.0f );
   ```

sets the position of the camera and its viewpoint.

The function is defined as follows:

```
public static void gluLookAt ( GL10 gl,
                               float eyeX, float eyeY, float eyeZ,
                               float centerX, float centerY, float centerZ,
                               float upX, float upY, float upZ )
```

where:

- `gl` : a GL10 interface
- `eyeX, eyeY, eyeZ` : the (x, y, z) position of the camera
- `centerX, centerY, centerZ` : the (x, y, z) position that the camera is looking at (i.e., where the camera is pointing)
- `upX, upY, upZ` : (x, y, z) vector that defines "up" in the viewpoint

To better understand how this works, try modifying the *eye* vector to $(0, 0, -7)$ instead of $(0, 0, -5)$. This is like moving the camera away from the point it is looking at. The triangle should get smaller.

Then try modifying the eye vector to *increase* the z value, such as $(0, 0, -3)$, which will be like moving closer to the point the camera is looking at. The triangle should get bigger.

Try changing other values in the eye, center and up vectors, and see what happens to the triangle.

4. **Add Motion**

Continue with the Tutorial, working through the "Add Motion" section.

The example adjusts the rotation angle for drawing by multiplying the current drawing matrix by the rotation matrix specified in the function parameters, as follows:

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

After you try this example, try changing the parameters in this function call, for example using the rotation matrix $(angle, 1, 0, 0)$ and see how this changes the axis of rotation.

You can also change the position of the triangle by using the translation matrix. The function call is:

```
gl.glTranslatef( x, y, z );
```

Try changing one value, such as the x value of the translation matrix each time the `onDrawFrame()` function is called. For example, create a class variable called xt (of type `float`) and initialize it to $0.0f$ in the $+$`onSurfaceCreated()` function. Then, in the `onDrawFrame()` function, call `glTranslatef()` using the translation matrix $(xt, 0, 0)$. After calling `glTranslatef()`, increment the value of xt by $0.1f$, so that next time the `onDrawFrame()` function is called, the value of the translation matrix will be slightly different. The triangle should gradually move to the left (or right, up, down—depending on how you set your Camera View parameters).

Note that the minimum and maximum values of the x coordinate (and y and z coordinates) are set in the function call `glFrustumf()`, which sets the clipping planes:

```
gl.glFrustumf( -ratio, ratio, -1, 1, 3, 7 );
```

The function arguments are the minimum and maximum values for the vertical, horizontal and depth clipping planes, as in:

```
gl.glFrustumf( left, right, bottom, top, near, far );
```

So if you try the translation exercise above, make sure that your x value ranges between the $left$ and $right$ parameters, and y ranges between $bottom$ and top.

5. **Respond to Touch Events**

   Continue with the Tutorial, working through the "Respond to Touch Events" section.

   This will let you rotate the triangle by dragging inside the drawing canvas on the Android emulator.

   There are two aspects to this functionality. One is to create an event handler that responds to "touch" events: `onTouchEvent()`. The other is to set the "rendering mode" so that the renderer (i.e., the `HelloOpenGLES10Renderer` class), particularly the `onDrawFrame()` function is only called when rendering needs to be done:

   ```
   setRenderMode( GLSurfaceView.RENDERMODE_WHEN_DIRTY );
   ```

   In this case, we only want to re-render the graphics when the user has touched and dragged on the view surface.

   Once you have this example working, try modifying the example so that the rotating triangle (or however you left the moving triangle from the previous section) stops moving when you click (i.e., touch) inside the view surface. This will involve two changes. First, you'll need to re-render the graphics continuously, so that the animation happens—i.e., you won't need the above call to `setRenderMode()`. Second, inside the `onTouchEvent()` function, you'll want to handle "click" events instead of "move" events. A "move" event is handled inside this switch statement:

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

   Instead, you'll want to handle `MotionEvent.ACTION_DOWN` events, which occur when the user "clicks" inside the view surface (or presses down with their finger on the touch screen).

   Have a look at the Android reference documentation for the `GLSurfaceView` class to learn about different types of touchscreen events that can be handled.