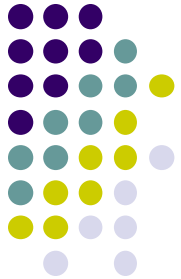
# Mobile 3D Graphics

## OpenGL ES

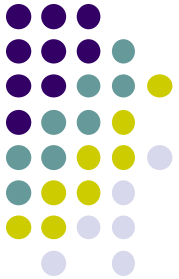
## Add motion & Respond to touch events

# Add motion



- Drawing objects on screen is a pretty basic feature of OpenGL, but you can do this with other Android graphics framework classes, including **Canvas** and **Drawable** objects.
- **OpenGL ES** provides additional capabilities for **moving** and **transforming** drawn objects in **three dimensions** or in other unique ways to create compelling user experiences.

# Rotate a shape

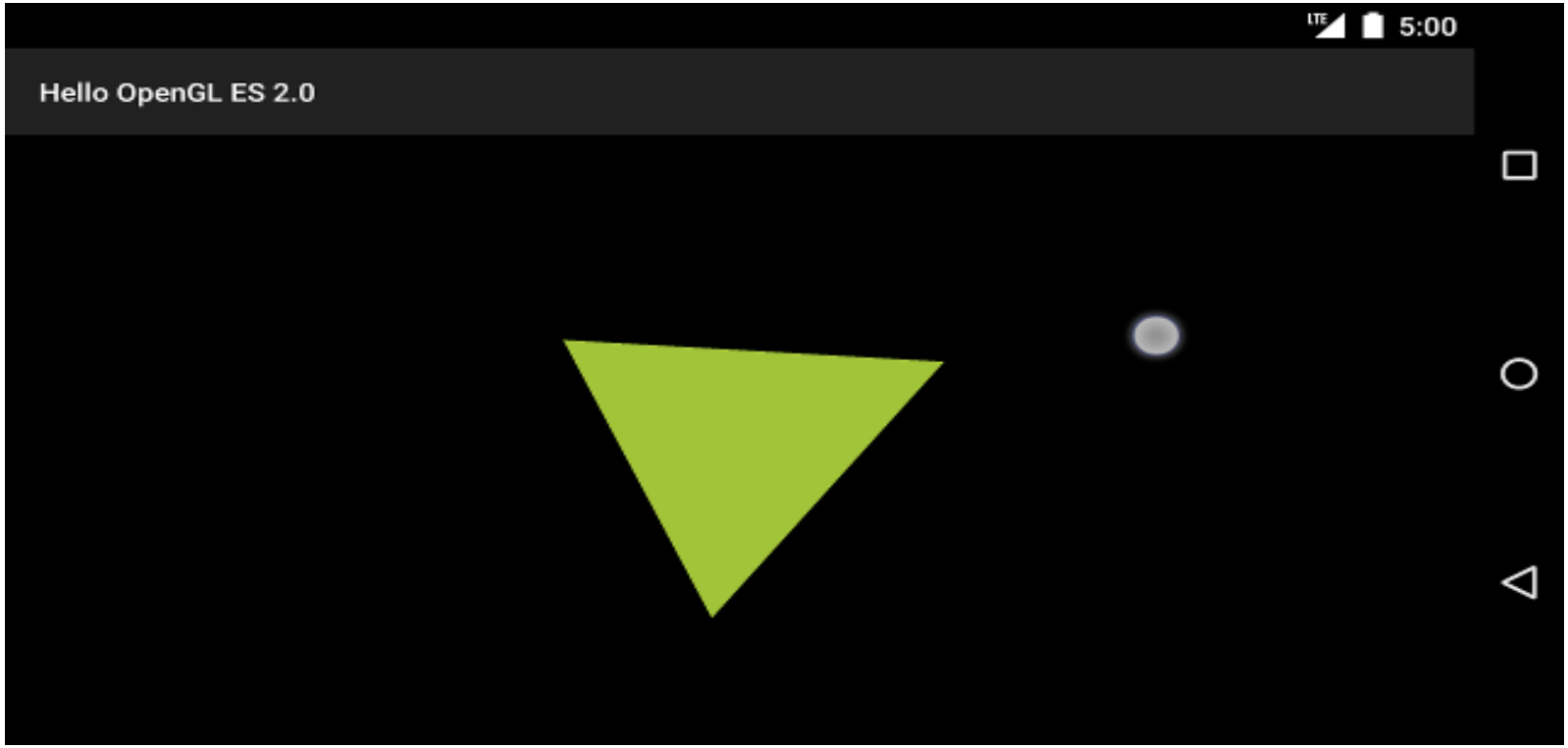


- **In your renderer**, create another **transformation matrix** (**a rotation matrix**) and then **combine** it with your **projection and camera view transformation matrices**

```
private float[] mRotationMatrix = new float[16];
Override
public void onDrawFrame(GL10 gl) {
    float[] scratch = new float[16];
    ...

    // Create a rotation transformation for the triangle
    long time = SystemClock.uptimeMillis() % 4000L;
    float angle = 0.090f * ((int) time);
    Matrix.setRotateM(mRotationMatrix, 0, angle, 0, 0, -1.0f);

    // Combine the rotation matrix with the projection and camera view
    // Note that the mMVPMatrix factor *must be first* in order
    // for the matrix multiplication product to be correct.
    Matrix.multiplyMM(scratch, 0, mMVPMatrix, 0, mRotationMatrix, 0);

    // Draw triangle
    mTriangle.draw(scratch);
}
```

# Respond to touch events

- The key to making your **OpenGL ES** application touch interactive is expanding your implementation of **GLSurfaceView** to **override** the **onTouchEvent()** to listen for touch events.



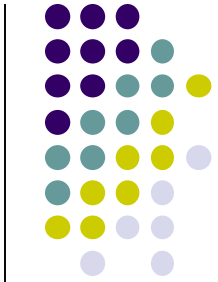
# Setup a touch listener

In order to make your **OpenGL ES** application respond to touch events:

- you must **implement** the **onTouchEvent()** method in your **GLSurfaceView class**.
- The example implementation shows how to listen for **MotionEvent.ACTION_MOVE** events and translate them to an angle of rotation for a shape.

```
private final float TOUCH_SCALE_FACTOR = 180.0f / 320;
private float mPreviousX;
private float mPreviousY;

@Override
public boolean onTouchEvent(MotionEvent e) {
    // MotionEvent reports input details from the touch screen
    // and other input controls. In this case, you are only
    // interested in events where the touch position changed.

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

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

            float dx = x - mPreviousX;
            float dy = y - mPreviousY;

            // 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 ;
            }

            mRenderer.setAngle( mRenderer.getAngle() +
                    ((dx + dy) * TOUCH_SCALE_FACTOR));
            requestRender();
    }

    mPreviousX = x;
    mPreviousY = y;
    return true;
}
```

# Expose the rotation angle



- Since the **renderer** code is running on a **separate thread** from the main user interface thread of your application,
- you must **declare** this **public variable** as **volatile**.

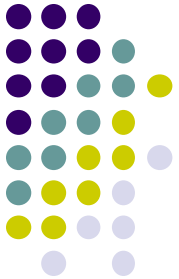
```
public class MyGLRenderer implements
GLSurfaceView.Renderer {
    ...

    public volatile float mAngle;

    public float getAngle() {
        return mAngle;
    }

    public void setAngle(float angle) {
        mAngle = angle;
    }
}
```

# Apply rotation

- To apply the rotation generated by touch input
- comment out the code that **generates an angle** and add a **variable** that contains the touch input generated angle

```
public void onDrawFrame(GL10 gl) {
    ...
    float[] scratch = new float[16];

    // Create a rotation for the triangle
    // long time = SystemClock.uptimeMillis() % 4000L;
    // float angle = 0.090f * ((int) time);
    Matrix.setRotateM(mRotationMatrix, 0, mAngle, 0, 0, -1.0f);

    // Combine the rotation matrix with the projection and camera
view
    // Note that the mMVPMatrix factor *must be first* in order
    // for the matrix multiplication product to be correct.
    Matrix.multiplyMM(scratch, 0, mMVPMatrix, 0,
mRotationMatrix, 0);

    // Draw triangle
    mTriangle.draw(scratch);
}
```

# References



- Build an OpenGL ES environment
  https://developer.android.com/training/graphics/opengl/environment
- Define shapes
  https://developer.android.com/training/graphics/opengl/shapes
- Draw shapes
  https://developer.android.com/training/graphics/opengl/draw#top_of_page
- Apply projection and camera views
  https://developer.android.com/training/graphics/opengl/projection#top_of_page
- Add motion
  https://developer.android.com/training/graphics/opengl/motion#java
- Respond to touch events
  https://developer.android.com/training/graphics/opengl/touch#java