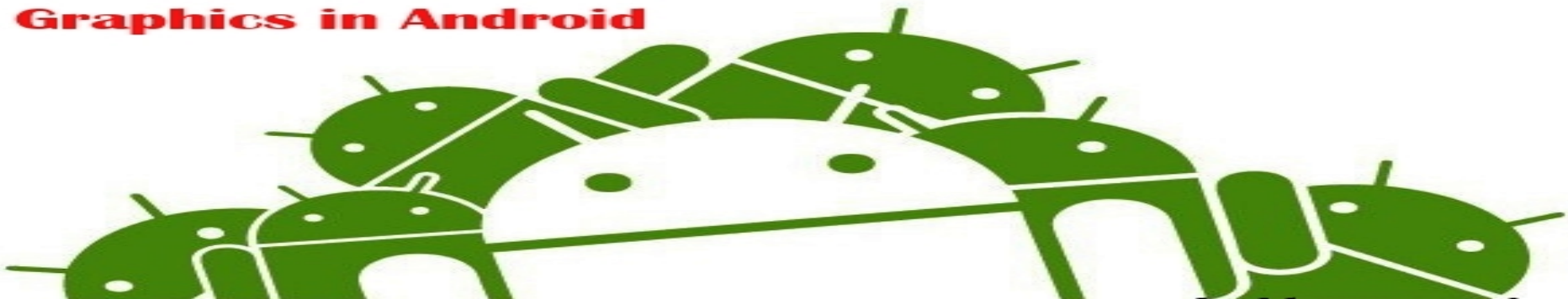
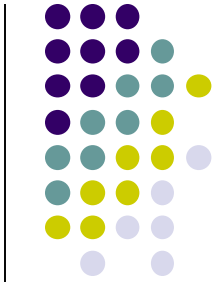
Mobile 3D Graphics

OpenGL ES

Apply projection and camera views

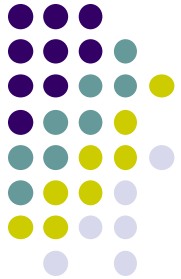


Apply projection and camera views



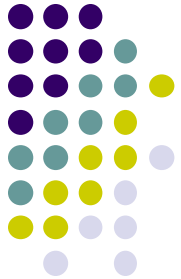
- **projection and camera views** allow you to **display drawn objects** in a way that more closely resembles how you see physical objects with your eyes.
- This simulation of physical viewing is done with **mathematical transformations** of drawn object coordinates

projection



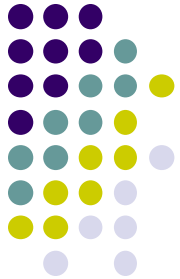
- This **transformation** adjusts the coordinates of drawn objects based on the **width** and **height** of the **GLSurfaceView** where they are displayed.
- **Without this calculation**, objects drawn by **OpenGL ES** are **skewed** by the unequal proportions of the view window.
- **A projection transformation** typically only has to be **calculated** when the proportions of the OpenGL view are **established** or **changed** in the **onSurfaceChanged()** method of your **renderer**.

Camera View



- This **transformation** adjusts the **coordinates** of drawn objects based on a **virtual camera position**.
- **A camera view transformation** might be **calculated** only once when you establish your **GLSurfaceView**, or might change dynamically based on user actions or your application's function.

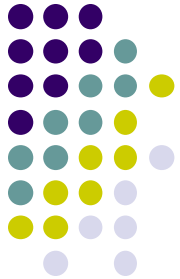
Define a projection



- The data for a **projection transformation** is calculated in the **onSurfaceChanged()** method of your **GLSurfaceView.Renderer** class.

```
// mMVPMatrix is an abbreviation for "Model View Projection Matrix"
private final float[] mMVPMatrix = new float[16];
private final float[] mProjectionMatrix = new float[16];
private final float[] mViewMatrix = new float[16];

@Override
public void onSurfaceChanged(GL10 unused, int width, int height) {
    GLES20.glViewport(0, 0, width, height);

    float ratio = (float) width / height;

    // this projection matrix is applied to object coordinates
    // in the onDrawFrame() method
    Matrix.frustumM(mProjectionMatrix, 0, -ratio, ratio, -1, 1, 3, 7);
}
```

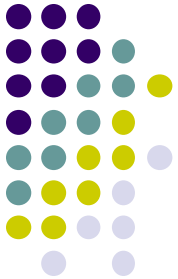
Define a camera view



- Complete the process of transforming your drawn objects by adding a **camera view transformation** as part of the drawing process in your **renderer**

```
@Override
public void onDrawFrame(GL10 unused) {
    ...
    // Set the camera position (View matrix)
    Matrix.setLookAtM(mViewMatrix, 0, 0, 0, -3, 0f, 0f, 0f, 0f, 1.0f, 0.0f);

    // Calculate the projection and view transformation
    Matrix.multiplyMM(mMVPMatrix, 0, mProjectionMatrix, 0, mViewMatrix, 0);

    // Draw shape
    mTriangle.draw(mMVPMatrix);
}
```

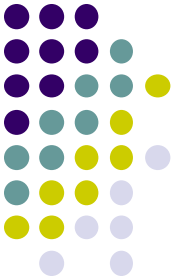
Apply projection and camera transformations



- add a **matrix variable** to the **vertex shader** previously defined in the **Triangle class**:

```
public class Triangle {

    private final String vertexShaderCode =
        // This matrix member variable provides a hook to manipulate
        // the coordinates of the objects that use this vertex shader
        "uniform mat4 uMVPMatrix;" +
        "attribute vec4 vPosition;" +
        "void main() {" +
        // the matrix must be included as a modifier of gl_Position
        // Note that the uMVPMatrix factor *must be first* in order
        // for the matrix multiplication product to be correct.
        "  gl_Position = uMVPMatrix * vPosition;" +
        "}";

    // Use to access and set the view transformation
    private int mMVPMatrixHandle;

    ...
}
```

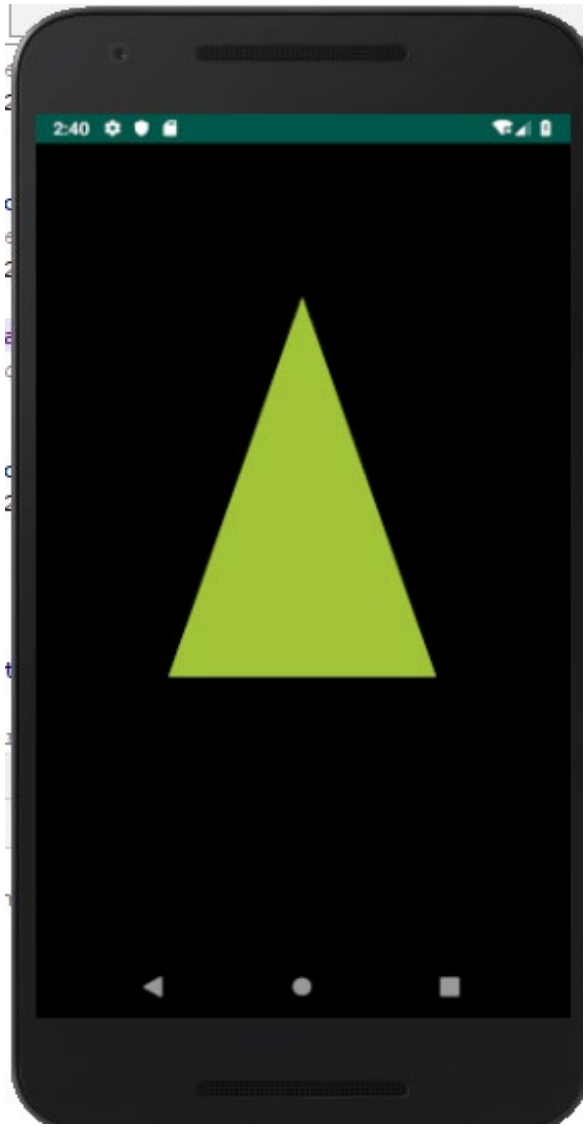
Apply projection and camera transformations



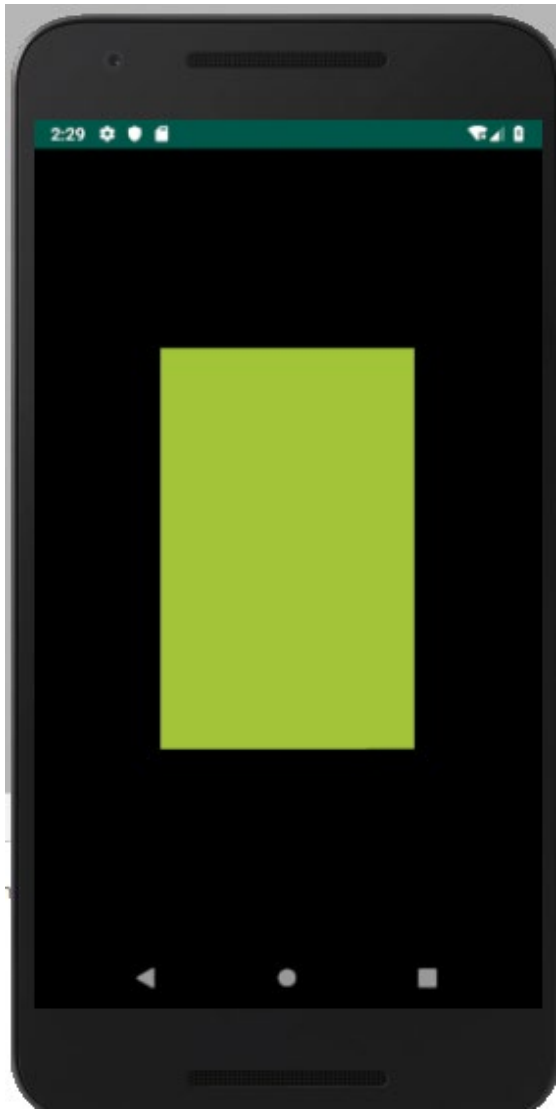
- modify the **draw()** method of your graphic objects to accept the combined transformation matrix and apply it to the shape:

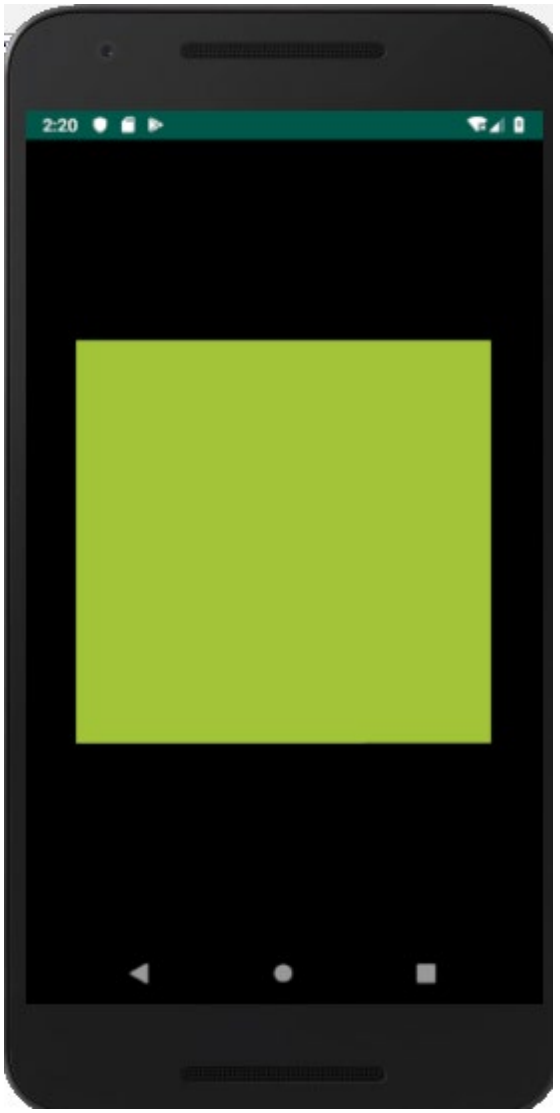
```
public void draw(float[] mvpMatrix) { // pass in the calculated transformation matrix
    ...

    // get handle to shape's transformation matrix
    mMVPMatrixHandle = GLES20.glGetUniformLocation(mProgram, "uMVPMatrix");

    // Pass the projection and view transformation to the shader
    GLES20.glUniformMatrix4fv(mMVPMatrixHandle, 1, false, mvpMatrix, 0);

    // Draw the triangle
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, vertexCount);

    // Disable vertex array
    GLES20.glDisableVertexAttribArray(mPositionHandle);
}
```

projection and camera transformations

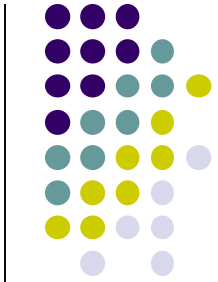


projection and camera transformations





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