

Android and OpenGL

Introduction to Android Smartphone Programming

University of Freiburg

Matthias Keil
Institute for Computer Science
Faculty of Engineering
University of Freiburg

10. Januar 2013



**UNI
FREIBURG**



1 OpenGL Introduction

2 Displaying Graphics

3 Interaction

4 Notes

5 Summary





- Short for: Open Graphics Library^[4].
- Enables creation of 2D and 3D graphics.
- Special API for embedded systems available on Android:
OpenGL ES API.
- Two important classes: *GLSurfaceView* and
GLSurfaceView.Renderer.





GLSurfaceView View to draw and manipulate objects using OpenGL.

GLSurfaceView.Renderer Interface defining methods to draw (render) graphics.

- Add renderer to GLSurfaceView using *GLSurfaceView.setRenderer()*.
- Extend GLSurfaceView to capture touch screen events.
- Extend Android manifest when using OpenGL ES 2.0:

```
1 <!-- Tell the system this app requires OpenGL
   ES 2.0. -->
2 <uses-feature android:glEsVersion="0x00020000"
   android:required="true" />
```





```
1 class MyGLSurfaceView extends GLSurfaceView {
2     public MyGLSurfaceView(Context context){
3         super(context);
4         setRenderer(new MyRenderer());
5         // Called when using OpenGL ES 2.0
6         setEGLContextClientVersion(2);
7     }
8 }
```



- Includes three methods to be implemented to draw graphics.

`onSurfaceCreated()` Called once when creating the
GLSurfaceView.

Should include all actions to do only once.

`onDrawFrame()` Called on each redraw of GLSurfaceView.

Do all drawing and redrawing of graphic objects
here.

`onSurfaceChanged()` Called when the geometry of
GLSurfaceView changes, for example size screen or
orientation.

Add code to respond to those changes.





- Two different OpenGL ES API versions available: 1.0 (together with version 1.1 extensions) and 2.0.
- Both usable to create high performance graphics for 3D games and visualizations.
- Graphic programming for one of the versions differs significantly to programming for the other version.
- Version 1.0/1.1 is easier to use as there are more convenience methods available.
- Version 2.0 provides higher degree of control, enabling creating of effects that are hard to realize in version 1.0/1.1.





- Shapes are graphic objects to be drawn in OpenGL.
- Shapes are defined using three-dimensional coordinates.
- Coordinates get written into *ByteBuffer* that is passed into the graphics pipeline for processing.
- Coordinate format: $[X, Y, Z]$
- Examples: Center of view: $[0,0,0]$, top right corner: $[1,1,0]$, bottom left corner: $[-1,-1,0]$.



Displaying Graphics

Example: Defining Triangle

University of Freiburg



UNI
FREIBURG

```
1 class Triangle {
2     private FloatBuffer vertexBuffer; ...
3     public Triangle() {
4         // initialize vertex byte buffer for shape
           coordinates (4 bytes per coordinate)
5         ByteBuffer bb = ByteBuffer.allocateDirect(
           triangleCoords.length * 4);
6         // use the device hardware's native byte
           order
7         bb.order(ByteOrder.nativeOrder());
8         // create a floating point buffer
           vertexBuffer = bb.asFloatBuffer();
9         // add the coordinates to the FloatBuffer
10        vertexBuffer.put(triangleCoords);
11        // set the buffer to read the first
           coordinate
12        vertexBuffer.position(0);
13    }
14 }
```





Vertex Shader Contains code for rendering the vertices of a shape.

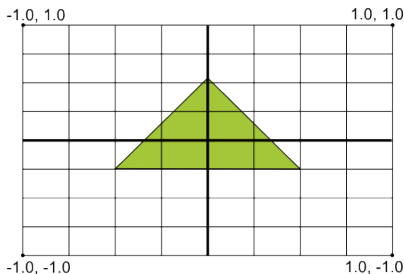
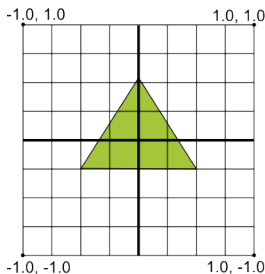
Fragment Shader Contains code for rendering the face (visible front) of shape with colors or textures.

Program OpenGL ES object containing shaders used.

- At least one vertex shader and one fragment shader needed to draw a shape.
- Both shaders must be compiled and then added to the program.



- Problem: Device screen is no square, but OpenGL assumes that_[1].
- The picture shows what happens. Left: How it should look. Right: How it looks in horizontal orientation.
- Solution: Use *projection modes* and *camera views* to transform coordinates.





- Create *projection matrix* and *camera view matrix*.
- Apply both to the OpenGL rendering pipeline.
- Projection matrix recalculates coordinates of the graphic objects to adjust the screen size.
- Camera view matrix creates transformation that shows object from specific eye position.





- Create and use projection matrix in `onSurfaceChanged()` of the `GLSurfaceView.Renderer` implementation.
- Use geometry of device seen to recalculate coordinates.

```
1 public void onSurfaceChanged(GL10 gl, int width
    , int height) {
2     gl.glViewport(0, 0, width, height);
3     float ratio = (float) width / height;
4     // set matrix to projection mode
5     gl.glMatrixMode(GL10.GL_PROJECTION);
6     // reset the matrix to its default state
7     gl.glLoadIdentity();
8     // Define and apply the projection matrix
9     gl.glFrustumf(-ratio, ratio, -1, 1, 3, 7);
10 }
```





- Define a projection matrix in terms of six planes.

```
1 public static void frustumM (float [] m, int  
    offset, float left, float right, float  
    bottom, float top, float near, float far)
```



Displaying Graphics

Example in OpenGL ES 1.0: Camera Transformation Matrix

University of Freiburg



- Apply camera view in `onDrawFrame()` of the `GLSurfaceView.Renderer` implementation.
- Use `GLU.gluLookAt()` to create a transformation simulating the camera position.

```
1 public void onDrawFrame(GL10 gl) {
2     ...
3     // Set GL_MODELVIEW transformation mode
4     gl.glMatrixMode(GL10.GL_MODELVIEW);
5     // reset the matrix to its default state
6     gl.glLoadIdentity();
7     // When using GL_MODELVIEW, you must set the
8     camera view
9     GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 0f,
10    1.0f, 0.0f);
11     ...
12 }
```





- Define a transformation in terms of an eye point, a center of view, and an up vector.

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





- 1 Define a Projection^[5].
 - 2 Define a Camera View.
 - 3 Apply Projection and Camera Transformations on all objects to draw.
- Step 1 and 2 very similar to OpenGL ES 1.0.





- Apply Projection and Camera Transformations on all objects to draw.
- Edit *draw* method of a shape:

```
1 public void draw(float [].mvpMatrix) {...
2 // get shape's transformation matrix
3 matrix = GLES20.glGetUniformLocation(mProgram
4     , "uMVPMatrix");
5 // Apply projection and view transformation
6 GLES20.glUniformMatrix4fv(matrix, 1, false,
7    .mvpMatrix, 0);
8 // Draw the shape
9 GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0,
10    .vertexCount);
11 ...
12 }
```





- Rotation can be simply added using OpenGL ES 2.0
- Create rotation matrix and combine it with projection and camera view transformation matrices.
- Extend *onDrawFrame* method.



Displaying Graphics

Adding Motion Example

University of Freiburg



UNI
FREIBURG

```
1 float [] mRotationMatrix = new float [16];
2 // Create a rotation transformation for the
   triangle
3 long time = SystemClock.uptimeMillis() % 4000
   L;
4 float angle = 0.090f * ((int) time);
5 Matrix.setRotateM(mRotationMatrix, 0, mAngle,
   0, 0, -1.0f);
6 // Combine the rotation matrix with the
   projection and camera view
7 Matrix.multiplyMM(mMVPMatrix, 0,
   mRotationMatrix, 0, mMVPMatrix, 0);
8 // Draw shape
9 mTriangle.draw(mMVPMatrix);
```





- Can be implemented by overriding the method *onTouchEvent(MotionEvent)* of the class *View*.
- *MotionEvent* gives you various information about where the event happened and how.
- Example: *long MotionEvent.getDownTime()* returns the time in ms when user started to press down.
- Also possible to recover *historical/old* coordinates of the event_[3].
- Easy simulation in the emulator possible: Click, hold and move the mouse.



- Class *Random* can produce a random number^[6].
- Class *Sensor* is used to access sensors of the cellphone, e.g. the gyroscope^[8].
- Class *MediaPlayer* enables playing of sounds^[2].
- Usage: Put a sound file into folder *res/raw/*.
- Supported file formats include ogg vorbis, wav, mp3 and more.

```
1 MediaPlayer mediaPlayer = MediaPlayer.create(  
    context, R.raw.soundfile);  
2 mediaPlayer.start();
```



- Drawing with OpenGL takes place on *GLSurfaceView*.
- *GLSurfaceView.Renderer* is responsible to draw the shapes.
- Important to decide which OpenGL ES version to take.
- Shapes are defined using three-dimensional coordinates.
- Different shaders needed to draw a shape.
- *Projection matrix* is used to adjust graphics to the device screen.
- *Camera transformation matrix* is used to simulate a camera position.
- Rotation motion can be added using an additional matrix.
- Touch screen interaction can be implemented overriding method *onTouchEvent*.





ANDROID DEVELOPERS.

Mapping Coordinates for Drawn Objects.

<http://developer.android.com/guide/topics/graphics/opengl.html#coordinate-mapping>.



ANDROID DEVELOPERS.

Media Playback.

<http://developer.android.com/guide/topics/media/mediaplayer.html>.



ANDROID DEVELOPERS.

MotionEvent.

<http://developer.android.com/reference/android/view/MotionEvent.html>.



ANDROID DEVELOPERS.

OpenGL.

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



ANDROID DEVELOPERS.

OpenGL ES 2.0: Applying Projection and Camera Views.

<http://developer.android.com/training/graphics/opengl/projection.html#projection>.



ANDROID DEVELOPERS.

Random.

<http://developer.android.com/reference/java/util/Random.html>.



ANDROID DEVELOPERS.

Tutorial: Displaying Graphics with OpenGL ES.

<http://developer.android.com/training/graphics/opengl/index.html>.



ANDROID DEVELOPERS.

Using the Gyroscope.

