Android Graphics Programming

Fritz Solms

August 14, 2015
Options:

- Basic 2D drawing on canvas
  - `android.graphics.drawable`
  - `android.view.animation`
- **JavaFX** port under development
  - Support for 3D graphics with
    - 2D & 3D shapes
    - General transformations
    - Effects like shadows & reflections
    - Perspective views (cameras)
    - Point & ambient light sources
    - Fixed colours & colour gradients
    - Texturing
- Low-level API into graphics hardware
  - **OpenGL-ES** (Embedded systems)
What is OpenGL?

**Definition**

*OpenGL* is a software interface to graphics hardware.

- Developed by Silicon Graphics (1991/2)
- Standard managed by non-profit Khronos group
- Interface with GPU’s
  - i.e. API *implemented in hardware*
  - accelerated hardware rendering
  - widely supported
- 2D & 3D vector graphics
- 200+ functions
  - language-independent spec using *C*-style
  - binding to large no of languages
    - *C, Java, JavaScript, …*
OpenGL Features

- 2D & 3D rendering
- Transformations
  - translation, scaling, rotation, ...
- Depth-cueing & fogging
  - making objects further away dimmer
- Antialiasing
  - smoothing ragged edges
- Flat & smooth shading
- Lighting
  - point, diffuse, directional
    - ambient (bouncing around scene)
- Orthographic projection
  - including perspective projection, clipping, ...
- Texturing
  - applying colour, vector & bitmaps images to geometric shapes
- Materials
  - specifying reflection & emission
    - ambient, diffuse, specular reflection
Open GL Rendering Pipeline

Map parametric curves & surfaces onto vertices.

Intermediate storage

Vertex Data

Display List

Pixel Data

Transformation (scaling, rotation, clipping, ...) of pixel data, then stored in texture memory or sent to rasterization.

Evaluation

Pixel operations

Texture assembly

Per vertex operations & primitive assembly

Rasterization

Fragment operations

Frame Buffer

Apply textures onto geometric objects.

Convert geometric & pixel data to fragments representing pixels in frame buffer.

Perform final operations on frame-buffer pixels including applying texture memory to fragment, fog, blending, dithering, ...

Convert vertices to primitives, performing projection from 3D world onto 2D rendering plane (screen).
OpenGL Supporting Libraries

- freeglut & deprecated GLUT
  - Access to OS’s windowing
  - Monitoring input devices (kbd, mouse, …)
- FLWK, Qt & wxWidgets
  - cross-platform widget libraries
- Allegro5, SDL & SFML
  - Media libraries for gaming, C and C++
- Mesa 3D
  - open-source software implementation of API (Linux)
    - without graphical hardware devices (GPUs)
OpenGL-ES

- OpenGL for *Embedded Systems*
  - Mobile devices
  - Automobile devices
  - Avionics & defense industry
  - Gaming stations
- Parallel standard to OpenGL
  - subset of OpenGL
  - extra features for embedded systems
    - fixed point numerical data types
- Different versions provide support at various levels
  - OpenGL ES 1
    - fixed rendering pipeline, limited shapes, no materials, . . .
  - OpenGL ES 2.0+
    - programmable rendering pipeline, vertex buffers, multi-texture, materials
  - OpenGL ES 3.0+
    - multiple rendering targets,
    - texture compression, 3D textures, depth textures, vertex textures, . . .
    - shading language
OpenGL-ES on Android

- Well supported:
  - OpenGL-ES 1.1 → Android 1.0+
  - OpenGL-ES 2.0 → Android 2.2+
    - supported by most devices
  - OpenGL-ES 3.0 → Android 4.3+
  - OpenGL-ES 3.1 → Android 5.0+
- Support also restricted by hardware (GPU)
  - Need to check support
  - Particularly OpenGL ES 3.0+
    - requires hardware implementation of graphics pipeline
Checking Device Support for OpenGL-ES (1/2)

- Specify required version as int
  - upper 16 bits → major version
  - lower 16 bits → minor version

```java
private static final int requiredGlEsVersion = 0x20000;
```

- Can convert between decimal & internal representations

```java
public static int convertDoubleToTwo16bitInt(float x)
{
    System.out.println(x%1);
    int minor = Math.round(x%1 * 10);
    int major = Math.round(x);
    return major*65536 + minor;
}

public static double convertTwo16BitIntToDouble(int internalRepresentation)
{
    int major = internalRepresentation / 65536; // 2^16
    int minor = internalRepresentation - major * 65536;
    return Double.parseDouble(Integer.toString(major) + "." + Integer.toString(minor));
}
```
Checking Device Support for OpenGL-ES (1/2)

- Retrieve device configuration info from ActivityManager

```java
ConfigurationInfo deviceConfigurationInfo = ((ActivityManager) getSystemService(Context.ACTIVITY_SERVICE)).getDeviceConfigurationInfo();
```

- Check whether required version available

```java
if (deviceConfigurationInfo.reqGlEsVersion >= requiredGlEsVersion) ...
```

- Specifying required support in manifest
  - so that only installed on supporting devices from Google Play

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="za.ac.up.cs.openglesdemo">
  
  <application ...>
    <activity ...>
      </activity>
    </application>

    <uses-feature android:glEsVersion="0x20000" android:required="true" />

</manifest>
```
OpenGL-ES Activity

- Android UI component: GLSurfaceView
  - For games: typically full-screen

```java
public class GraphicsActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        getWindow().clearFlags(WindowManager.LayoutParams.FLAG_FORCE_NOT_FULLSCREEN);

        ConfigurationInfo deviceConfigurationInfo = ((ActivityManager)
            getSystemService(Context.ACTIVITY_SERVICE)).getDeviceConfigurationInfo();

        if (deviceConfigurationInfo.reqGlEsVersion >= requiredGlEsVersion) {
            surfaceView = new SomeGLSurfaceView(this);
        } else {
            new AlertDialog.Builder(this).setMessage("Insufficient GL-ES support").create().show();
        }
        setContentView(surfaceView);
    }
}
```
GLSurfaceView

- Purely a view
  - Rendering by a GLSurfaceView_Renderer

```java
public class SomeSurfaceView extends GLSurfaceView {
    public SomeGLSurfaceView(Context context) {
        super(context);

        // Specify GL-EL version used for view
        setEGLContextClientVersion(2);

        renderer = new MyRenderer();

        // Set renderer used to draw onto surface
        setRenderer(renderer);
    }

    private final MyRenderer renderer;
}
```
Renderer

- Responsible for drawing on surface
- API polluted by 10.version
  - GL10 parameter not used

```java
public class MyRenderer implements GLSurfaceView.Renderer {

    public void onSurfaceCreated(GL10 unused, EGLConfig config) {
        // Set the background frame color
        GLES20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        // 3 colour & 1 alpha channel
    }

    public void onDrawFrame(GL10 unused) {
        // Redraw background color
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);
    }

    public void onSurfaceChanged(GL10 unused, int width, int height) {
        GLES20.glViewport(0, 0, width, height);
    }
}
```
OpenGL (ES) Coordinate System

Right-Hand System (RHS) or Counter-Clockwise (CCW) System

(0,0,0)
OpenGL Vertex

- Position in 3D space
  - join points for edges
  - locations of cameras, light sources, ...
- Specified as float[]
  - Need to ass f – default: double
  - streamed into ByteBuffer for performance
  - FloatBuffer view onto ByteBuffer

```java
private float vertices[] = {
  0.0f, 0.5f, 0.0f, // top vertex of triangle
  -0.5f, -0.5f, 0.0f, // bottom left
  0.5f, -0.5f, 0.0f, // bottom right
};

ByteBuffer vertexByteBuffer = ByteBuffer.allocateDirect(vertices.length*4);
vertexByteBuffer.order(ByteOrder.nativeOrder());
FloatBuffer vertexBuffer = vertexByteBuffer.asFloatBuffer();
vertexBuffer.put(vertices);
vertexBuffer.position(0);
```
Edges & Faces

- **Edge**
  - line between 2 vertices
    - often shared between faces
  - can be transformed
    - affecting vertices & faces

- **Face**
  - *flat* surface
  - usually a triangle
Decomposing Shapes into Triangles

- Continue decomposing until either
  - flat enough
  - minimum size reached
Shapes

- Ultimately draw either
  - GL_POINTS
  - GL_LINES – split vertex array in pairs & draw lines
  - GL_LINE_STRIP – sequence of vertex points
  - GL_LINE_LOOP – closing loop
  - GL_TRIANGLES – split vertex array into 3’s
  - GL_TRIANGLE_STRIP – v0, v1, v2 - v2, v1, v3 - v3, v2, v4
  - GL_TRIANGLE_FAN – v1, v2, v3 - v1, v3, v4 - v1, v4, v5