Introduction to OpenGL and JOGL
Contents

- Part 1 – A brief introduction to OpenGL
  - What is OpenGL? When is it useful?
  - Some features of OpenGL

- Part 2 - OpenGL in Java (JOGL)
  - Set up
  - A simple example
  - Overheads and deployment
  - Interaction with the garbage collector

- Conclusions
Part 1 - OpenGL
What is OpenGL?

- A standard specification defining a cross-platform API for writing applications that produce 2D and 3D computer graphics. (Wikipedia)
- Hardware independent
  - Complexity of dealing with different graphics hardware is hidden from the user
  - The capabilities of a specific piece of graphics hardware are hidden. All OpenGL implementations must support the full feature set
- Hardware accelerated
  - Implementations provided by graphics card vendors (NVIDIA, ATI)
  - Computation is offloaded to the GPU where possible
  - Multiple pipelines in the GPU allow multiple vertices and pixels to be processed in parallel
- C interface. Native bindings allow access from other languages
Example: A 3D square

glTranslatef(0, 7, 0);
glRotatef(45, 0, 0, 2);

glTranslatef(8, 6, 0);
glScalef(2, 2, 2);

glTranslatef(8, 0, 0);
The Geometry Color

- Triangle color defined per vertex
- Interpolate colors over the triangle
Texturing

- Painting images onto geometrical objects
Geometry Projection

- Orthogonal

- Perspective
When is OpenGL Useful?

- When is OpenGL useful
  - You want to render something in 3D
    - Data visualisation
  - Virtual/Augmented reality
  - You want to render something quickly in 2D
  - You want to render something realistically (e.g. using textures and lighting)

- Why is OpenGL a good choice?
  - Cross platform (Windows / Linux / Mac)
  - Hardware acceleration
  - Features (e.g. texture mapping, alpha blending, lighting, render-to-texture)
A Simple Example

drawBox() {

glClearColor(0.0, 0.0, 0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glOrtho(0,1,0,1,-1,1);
glBegin(GL_POLYGON);
  glVertex2f(-0.5, -0.5);
glVertex2f(-0.5, 0.5);
  glVertex2f(0.5, 0.5);
  glVertex2f(0.5, -0.5);
  glVertex2f(-0.5, -0.5);
glEnd();
glFlush();
}

Sets the background colour
Clears the window
Sets the drawing colour
Sets the view
Start drawing a primitive
Define the primitive
Finish drawing the primitive
Ensure the commands are executed
OpenGL - Primitive types

GL_POINTS
GL_LINES
GL_LINE_STRIP
GL_LINE_LOOP
GL_POLYGON
GL_QUADS
GL_QUAD_STRIP
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
Defining OpenGL primitives

```
glBegin( GL_PRIMITIVE_TYPE)
glVertex(...) 
glVertex(...) 
...

```

glEnd() Block.
Transformation Matrices

OpenGL provide 3 transformation matrix stacks:

- **Perspective Matrix** – Used for viewing transformations – equivalent to positioning and aiming a camera.

- **Modeling Matrix** – Used for modeling transformations – equivalent to positioning and orienting the model to be drawn.

- **Texture Matrix** – Used for texture transformations – equivalent to positioning and orienting the texture to be drawn over a polygon.
Transformation functions

- `glLoadIdentity()`
- `glTranslatef(TYPE x, TYPE y, TYPE z)`
- `glRotate(TYPE angle, TYPE x, TYPE y, TYPE z)`
- `glScale(TYPE x, TYPE y, TYPE z)`
- `glPushMatrix()`
- `glPopMatrix()`
glLoadIdentity

- `glLoadIdentity()`
- Loads the identity matrix into the current transformation matrix.
- Used to reset the current transformation matrix before performing a transformation.
Translation

- `glTranslate(TYPE x, TYPE y, TYPE z)`

- Multiplies the current transformation matrix by a matrix that moves an object (the local coordinate system) by the given x, y, and z values.
Rotation

- `glRotate(TYPE angle, TYPE x, TYPE y, TYPE z)`

Multiplies the current transformation matrix by a matrix that rotates an object (or the local coordinate system) in a counter clockwise direction about the ray from the origin through the point `(x, y, z)`. The angle parameter specifies the angle of rotation in degrees.
Scaling

- **glScale**(TYPE x, TYPE y, TYPE z)

- Multiplies the current transformation matrix by a matrix that stretches, shrinks, or reflects an object (or the local coordinate system) along the axes. Each x, y, and z coordinate of every point in the object is multiplied by the corresponding argument x, y, or z.
Controlling the transformation matrix stacks

- `glPushMatrix()`
  - Pushed the current transformation matrix into the stack.

- `glPopMatrix()`
  - Loads the matrix at the top of the stack into the current transformation matrix.
Some Features of OpenGL

- Texture mapping
  - Allows you to “glue” an image (or texture) to a polygon

```c
//Generate a new texture id
glGenTextures(1, &texName);
//Bind to the texture
glBindTexture(GL_TEXTURE_2D, texName);
//Load in some image data
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, imageWidth, imageHeight, 0, GL_BGRA, GL_UNSIGNED_BYTE, imageData);

//Enable texturing
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texName);
//Draw the primitive, providing texture co-ordinates at each vertex
glBegin(GL_QUADS);
glTexCoord2f(0,0); glVertex3f(0,0);
glTexCoord2f(0,1); glVertex3f(0,1);
//...
//...
glEnd();

glDeleteTextures(1, texName);
```
Some Features of OpenGL

- **Blending**
  - Combines the colour of the fragment being processed with the pixel already stored in the framebuffer
  - Can achieve some effects very cheaply. e.g. Negative Lightening

```c
//Enable blending
glEnable(GL_BLEND);
//Set the blending function
glBlendFunc(GL_ONE_MINUS_DST_COLOR, GL_ZERO);
```

- **Lighting** – ambient / diffuse / specular / emissive

- **Render to texture**
  - Very powerful. Useful for composition (e.g. picture in picture) and effects (e.g. fast motion blur)

- **P-Buffers**

- **Framebuffer objects** (OpenGL 2.0)
The Redbook

- OpenGL Programming Guide
  - Current version 2.1
  - Version 1.1 is online
  - HIGHLY recommended!!
Part 2 - JOGL
Bindings for OpenGL

- OpenGL applications are usually written in C/C++
- Bindings allow OpenGL to be used from other languages
  - Java - JOGL, LWJGL
  - C# / Mono – Tao framework, OpenTK
  - Python - PyOpenGL
- Example – JOGL
  - Uses Java Native Interface (JNI) to access OpenGL
  - Exposes the entire OpenGL API through a few classes
  - JOGL consists of:
    - Java jar files (cross platform): jogl.jar gluegen-rt.jar
    - Platform specific libraries: jogl.dll gluegen-rt.dll jogl_cg.dll jogl_awt.dll
Setup JOGL

- http://www.timelessname.com/jogl/lesson01/
- JOGL Official Web site
  - https://jogl.dev.java.net
Examples


A typical OpenGL program involves THREE steps
- Set up lighting/materials/textures
- Set up the viewing/transformation matrices
- Set the geometric primitives
OpenGL do the actual drawing!
3D Viewing in OpenGL
Viewing

- `gluLookAt` – Move the camera around
- `gluPerspective` – define the camera parameters
  - Field of View
  - Aspect Ratio
  - Clipping planes
OpenGL - Primitive types

- **GL_POINTS**
- **GL_LINES**
- **GL_LINE_STRIP**
- **GL_LINE_LOOP**
- **GL_POLYGON**
- **GL_QUADS**
- **GL_QUAD_STRIP**
- **GL_TRIANGLES**
- **GL_TRIANGLE_STRIP**
- **GL_TRIANGLE_FAN**
Defining OpenGL primitives

```
glBegin( GL_PRIMITIVE_TYPE)
glVertex(...)  
glVertex(...)  

...

glEnd() Block.
```
The Java garbage collector will not clear up OpenGL resources.

- Textures
- Display lists
- Off-screen buffers

...so it is necessary to do so manually.

**Java programmer’s first attempt:**

*Use finalize*

- May never get run
- Could be executed by any thread
- OpenGL context may no longer be available when the finalizer is called

```java
public class TexturedObject {

private int[] mTex = new int[1];
private GL mGl;

public TexturedObject(GLAutoDrawable iDrawable) {
    mGl = iDrawable.getGL();
    mGl.glGenTextures(1, mTex);
}

public void glDraw() {
    mGl.glBindTexture(GL.GL_TEXTURE_2D, mTex[0]);
    //draw object
}

public void finalize() {
    mGl.glDeleteTextures(1, mTex);
}
}
```

*not a good idea 😞*
Java programmer’s second attempt: The glDestroy method

- Programmer must remember to call glDestroy when the object is no longer required
- Programmer must call glDestroy from the rendering thread

Not very Java-like. We want:

- To be able to destroy the object from any thread
- Automatic cleanup in case our lazy programmer forgets
JOGL and Garbage Collection

- **OpenGL Cleanup Pattern**
  - A destroy method that can be called from any thread
  - Schedules OpenGL cleanup to be performed by the rendering thread
  - Cleanup performed automatically if the programmer forgets

```java
public class TexturedObject implements GLDispose {

    private void mDoneCleanup = false;
    ...

    public void destroy() {
        scheduleGlCleanup(false);
    }

    public void finalize() {
        scheduleGlCleanup(true);
    }

    private synchronized void scheduleGlCleanup(
        boolean iFromFinalizer
    ) {
        if(!mDoneCleanup) {
            mDoneCleanup = true;
            if(iFromFinalizer) {
                //Warn the programmer
            }
            //Schedule cleanup in rendering thread
        }
    }

    public void glDestroy(GLAutoDrawable iDrawable) {
        GL lGl = iDrawable.getGL();
        lGl.glDeleteTextures(1, mTex);
    }
}
```

```java
public interface GLDispose {

    void glDestroy(GLAutoDrawable iDrawable);
}
```
Conclusion

- OpenGL is a low-level cross-platform API for computer graphics
- Useful for:
  - Rendering realistic scenes in 3D
  - Rendering 3D or 2D scenes quickly
  - Effects (e.g. Texture mapping, alpha blending, lighting, render-to-texture etc etc)
- Applications usually written in C/C++, but bindings allow access from other languages
  - But it is hard to hide resource de-allocation in managed languages
Resources

• OpenGL
  • OpenGL Programming Guide (D Shreiner et al)
  • OpenGL Shading Language (R Rost)
• Bindings
  • Java – JOGL ( 
1. Functions describing lines or surfaces are evaluated to obtain vertices

2. Converts vertices into primitives
   - Performs per-vertex operations (e.g. normal computation)

3. Converts primitives into fragments
   - A fragment corresponds to a pixel and contains colour, depth and sometimes texture co-ordinates

4. Per-fragment Operations
   - Texturing
   - Depth testing
   - Blending