Principle of computer graphics

Getting started with OpenGL 2.1

Objectives for this Lesson

- What Is OpenGL? What is and how it works.
- OpenGL/JOGL Applications Open Source Examples.
- Basic Definitions Model, Framebuffer, etc. What mean these words?
- OpenGL Command Syntax Conventions and notations used by OpenGL commands.
- OpenGL Rendering Pipeline Description of the typical sequence of operations for processing geometric and image data.

Objectives for this Lesson

- Setting programming environment,
- First application with JOGL (Callback Framework)
- Second example using Active Rendering
- Animation with JOGL, explains in high level terms how to create pictures on the screen that move.
- Bibliography

What Is OpenGL ?

- The OpenGL graphics system is a software interface to graphics hardware (The GL stands for Graphics Library).
- The interface consists of about 200 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications.
- For interactive programs that produce color images of moving three-dimensional objects. Sometimes they are so realistic that you can't distinguish them from real photos.
- First introduced 1992.

What Is OpenGL ?

❖ OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

❖ No commands for performing windowing tasks or obtaining user input are included in OpenGL;

❖ Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. With OpenGL, you must build up your desired model from a small set of geometric primitives - points, lines, and polygons.

OpenGL

- OpenGL has become the industry standard for graphics applications and games.
- OpenGL Specification:
  - www.opengl.org/documentation/specs/
- OpenGL State Diagram
  - www.opengl.org/documentation/specs/version1.1/state.pdf

Racer - Application Example

What is JOGL ?

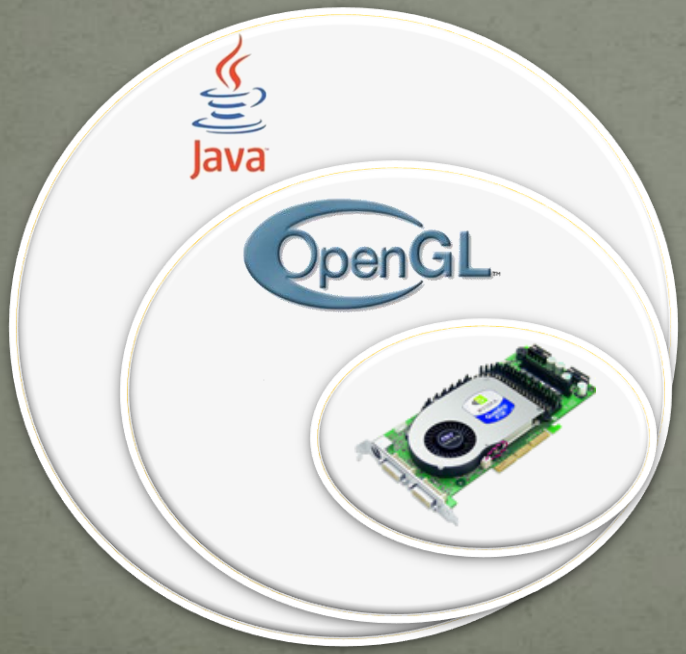
- JOGL is one of the open-source technologies initiated by the Game Technology Group at Sun Microsystems back in 2003 (the others are JInput and JOAL).
- Therefore, it doesn't include support for gaming elements such as sound or input devices, which are nicely dealt with by JOAL and Jinput.
- Since OpenGL is originally written in C, JOGL provides full access to the APIs (via Java Native Interface (JNI)) in the OpenGL 2.0 specification, as well as vendor extensions, and can be combined with AWT and Swing components.

What is JOGL ?

- It supports both of the main shader languages, GLSL and Nvidia's Cg.
- JOGL's utility classes include frame-based animation, texture loading, file IO, and screenshot capabilities.
- Two programming frameworks are possible with JOGL:
  - Callbacks
  - Active rendering
- Official website: http://kenai.com/projects/jogl

Why Java ?

- Our goal is to understand OpenGL. Java is simply a means to an end.
- Write once, run anywhere (WORA)



JOGL –Game/Graphical Engines

- Avengina
  - http://www.avengina.org
- Jake2 (It is a port of the GPL'd Quake2 game engine)
  - http://bytonic.de/html/jake2.html
- Elflight Engine
  - http://www.codededge.com
- jME (jMonkey Engine)
  - http://www.jmonkeyengine.com
- World Wind
  - http://worldwind.arc.nasa.gov/java/demos/

JOGL - Application Examples

More examples: http://download.java.net/media/jogl/www/

Basic Definitions

Basic Definitions

- Models, or objects, are constructed from geometric primitives - points, lines, and polygons - that are specified by their vertices.
- Pixel, is the smallest visible element the display hardware can put on the screen.
- Rendering, process by which a computer creates images from models and consists of pixels drawn on the screen.

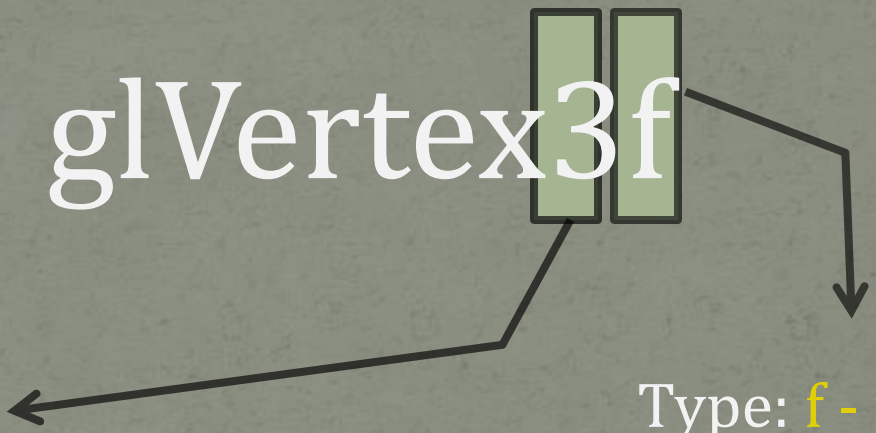
Basic Definitions

- Bitplane, area of memory that holds one bit of information for every pixel on the screen. E.g. bit might indicate how red a particular pixel is supposed to be.
- Framebuffer, stores all bitplanes, and holds all the information that the graphics display needs to control the color and intensity of all the pixels on the screen

OpenGL Command Syntax

- A command declaration has the form:
  - “gl” Name { Dim{1..n} } { b s i f d ub us ui } { v }(args);
  - ‘v' indicates vector format
  - absence of ‘v' indicates scalar format
- Data Types:
  - f - float
  - d - double float
  - s - signed short integer
  - i - signed integer
  - b - character
  - ub - unsigned character
  - us - unsigned short integer
  - ui - unsigned integer

OpenGL Command Syntax



The 4th dimension is for homogeneous coordinates, by default it is 1.

OpenGL as a State Machine

- Set a value (state) and this will be valid until you set it to something else.
- For example:

```
gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
…
gl.glClear(GL.GL_COLOR_BUFFER_BIT);
…
gl.glClear(GL.GL_COLOR_BUFFER_BIT);
```

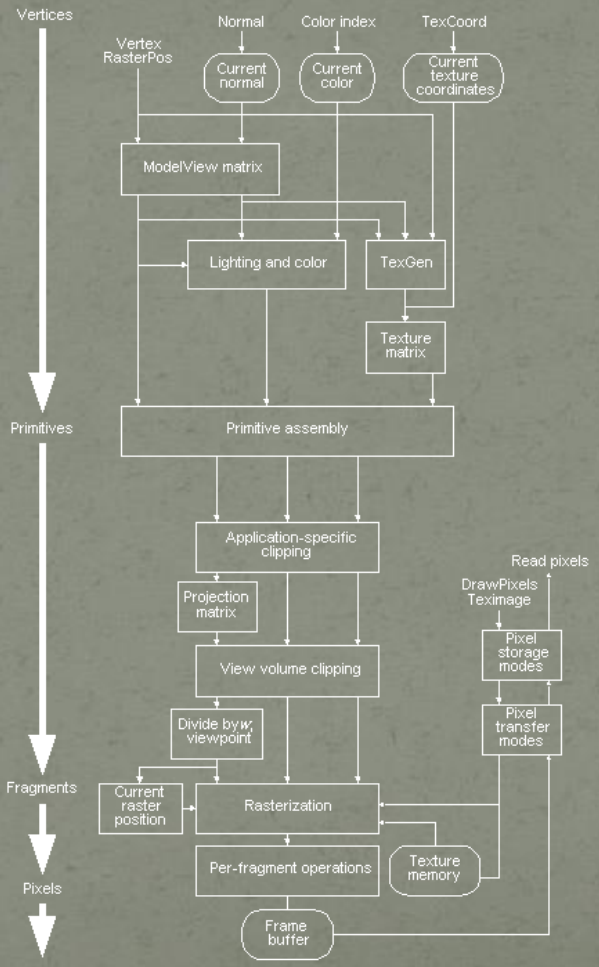
OpenGL Rendering Pipeline

- Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects. (Points, lines, polygons, images, and bitmaps are considered to be primitives)
- Arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene.

OpenGL Rendering Pipeline

- Calculate the color of all the objects. The color might be explicitly assigned by the application, determined from specified lighting conditions, obtained by pasting a texture onto the objects, or some combination of these three actions.
- Convert the mathematical description of objects and their associated color information to pixels on the screen. This process is called rasterization.

OpenGL Rendering Pipeline



Installing JOGL

Installing JOGL

- Download Eclipse or NetBeans (Optional)
- Download JOGL (use JSR-231)

  http://download.java.net/media/jogl/builds/nightly/
- Download Javadocs

  http://java.sun.com/javase/6/docs/api/

Installing JOGL

Its execution requires:

```
java -cp “c:\jogl\jogl.all.jar;."
        -Djava.library.path=“c:\jogl"
            -Dsun.java2d.noddraw=true ExerciseOne
```

The “sun.java2d.noddraw” property disables Java 2D’s use of DirectDraw on Windows [1]. This avoid any nasty interactions between DirectDraw and OpenGL, which can cause application crashes, poor performance, and flickering.

[1] The property is only needed if you’re working on a Windows platform.

Installing JOGL

If you don't like lengthy command line arguments, then modify:

Windows

The CLASSPATH environment variable and PATH

Solaris and Linux

LD_LIBRARY_PATH

Mac OS X

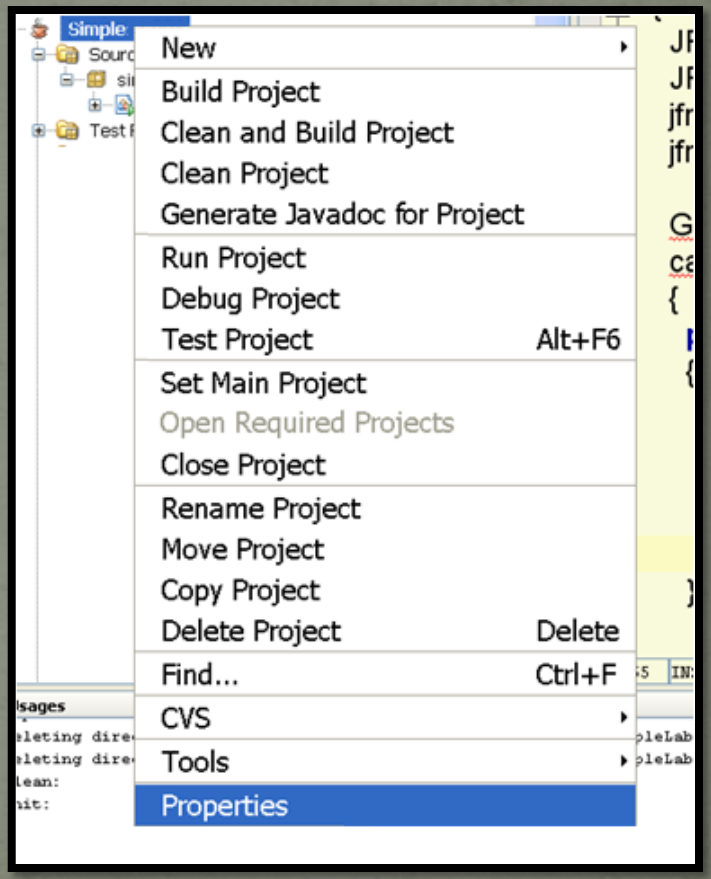
DYLD_LIBRARY_PATH

NetBeans Project With a JOGL library

These instructions are correct for NetBeans 5.0.

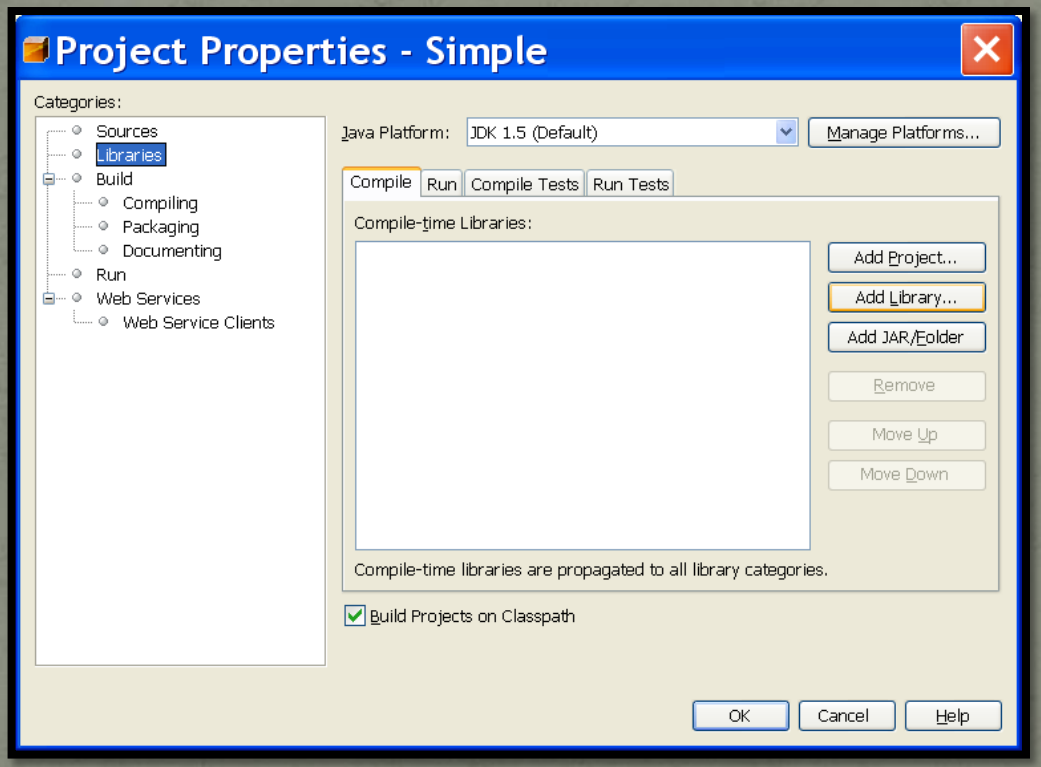
1. First create a new project, which is a Java application within NetBeans

2. Right click on the project name in the navigator view and select properties:



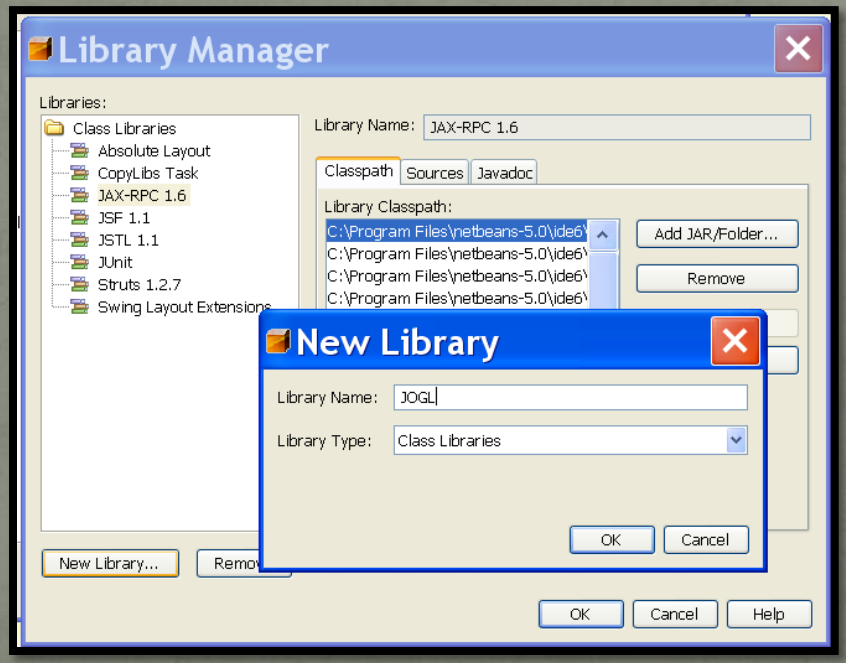
NetBeans Project With a JOGL library

Click on libraries and choose Add Library:



NetBeans Project With a JOGL library

Choose the manage libraries option in order to create a new library containing the JOGL jar file. Click on New Library and give the new library the JOGL.



NetBeans Project With a JOGL library

Next select JOGL from the possible libraries and click on Add Jar/Folder. Select gluegen-rt.jar, ativewindow.all.jar, jogl.all.jar and click Add Jar/Folder.

Now select add library to incorporate the JOGL library classes into the project.

NetBeans OpenGL Pack :

http://kenai.com/projects/netbeans-opengl-pack

Ecplipse Project With a JOGL library

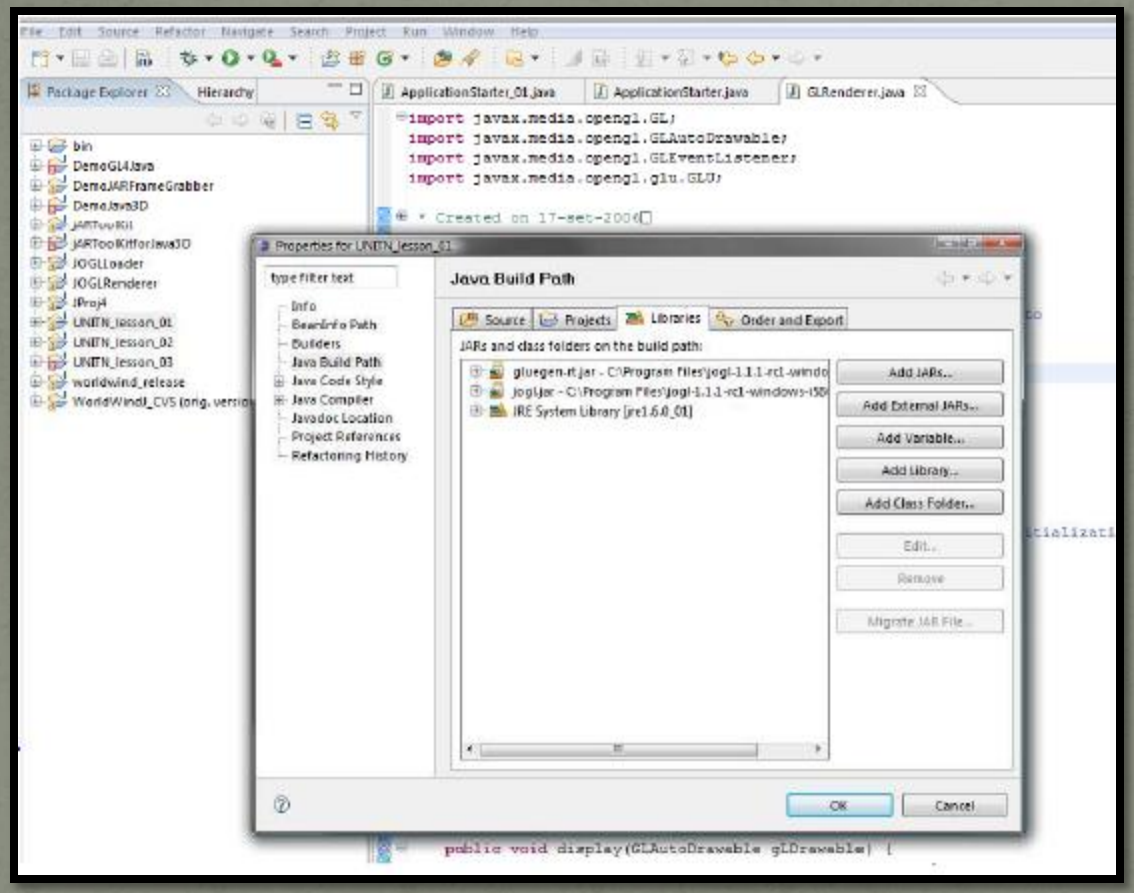
1. Create an empty ECLIPSE project
2. Right-click on the project

   Properties -> Java Buld Path -> libraries
3. Add External Jar:

   gluegen-rt.jar, nativewindow.all.jar, jogl.all.jar
4. Modify PATH environment variable to include:

   gluegen-rt.dll, jogl_gl2.dll, nativewindow*.dll

Ecplipse Project With a JOGL library



Basic Concepts about JOGL

Programming Frameworks

Two programming frameworks for JOGL will be described in this lesson:

- Callbacks -which I refer to as event-based rendering.
- Active rendering - which is “more appropriate” for game development.

The Callback Framework

The two main JOGL GUI classes are GLCanvas and GLJPanel, which implement the GLAutoDrawable interface, allowing them to be utilized as ‘drawing surfaces’ for OpenGL commands.

GLCanvas is employed in a similar way to AWT's Canvas class. It's a heavyweight component, so care must be taken when combining in with Swing. However, it executes OpenGL operations very quickly due to hardware acceleration.

The Callback Framework

GLJPanel is a lightweight widget which works seamlessly with Swing. In the past, it's gained a reputation for being slow since it copies the OpenGL frame buffer into a BufferedImage before displaying it. However, it's speed has improved significantly in Java SE 6.

A key advantage of GLJPanel over GLCanvas is that it allows 3D graphics (courtesy of OpenGL) and 2D elements in Swing to be combined in new, exciting ways.

Using GLCanvas

A GLCanvas object is paired with a GLEventListener listener, which responds to changes in the canvas, and to drawing requests.

When the canvas is first created, GLEventListener's init(GLAutoDrawable drawable) method is called; this method can be used to initialize the OpenGL state (for instance to setup lights and display lists).

Using GLCanvas

Whenever the canvas is resized, including when it's first drawn, GLEventListener‘s reshape(GLAutoDrawable drawable, int x, int y, int width, int height) is executed. It can be overridden to initialize the OpenGL viewport and projection matrix (i.e. how the 3D scene is viewed). reshape() is also invoked if the canvas is moved relative to its parent component.

Whenever the canvas' display() method is called, the display() method in GLEventListener is executed. Code for rendering the 3D scene should be placed in that method.

Using GLCanvas

The GLCanvas can be placed directly inside the JFrame, but by wrapping it in a JPanel, the JFrame can contain other (lightweight) GUI components as well.

Using GLEventListener

The GLEventListener also includes dispose(), called by the drawable before the OpenGL context is destroyed by an external event, like a reconfiguration of the GLAutoDrawable closing an attached window, but also manually by calling destroy.

Using GLJPanel

Since the GLJPanel is a lightweight Swing component, it can also be added directly to the enclosing JFrame

The Active Rendering Framework

- The active rendering framework utilizes the new features in JSR-231 for directly accessing the drawing surface and context (OpenGL's internal state). This means that there's no longer any need to utilize GUI components that implement the GLAutoDrawable interface, such as GLCanvas application can employ a subclass of AWT's Canvas, with its own rendering thread
- The principal advantage of the active rendering approach is that it allows the programmer to more directly control the application's execution flow (suspend, etc.)

The Active Rendering Framework

The rendering thread can be summarized using the following pseudocode:

```
make the context current for this thread;
initialize rendering;
while game isRunning {
        update application state;
        render scene;
        put the scene onto the canvas;
        sleep a while;
        do optional updates without rendering them;
        gather statistics;
}
discard the rendering context;
exit;
```

The Active Rendering Framework

Make the context current for the thread with:

```
private void makeContentCurrent() {
        try {
                while (context.makeCurrent() == GLContext.CONTEXT_NOT_CURRENT) {
                        System.out.println("Context not yet current...");
                        Thread.sleep(100);
                }
        }
        catch (InterruptedException e) {
                        e.printStackTrace();
        }
}
```

And put scene into canvas with:

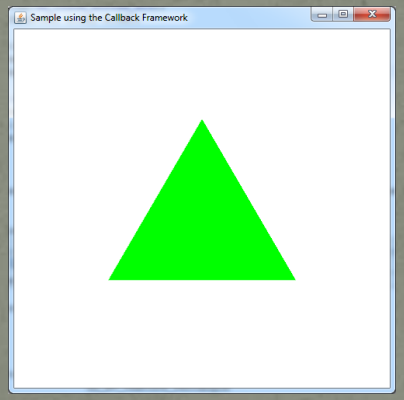
```
drawable.swapBuffers();
```

See ExerciseTwo for a complete example

JOGL - First Example

OpenGL - First Example

- Very simple example shows many of JOGL elements that are common to all graphic programs.
- Only displays green triangle on white background, shown in Java Swing JFrame object. Full Code on course web site.



GLCapabilities

- When creating GLCanvas and GLJPanel instances, the user may configure a certain set of OpenGL parameters in the form of a GLCapabilities object.
- These customise how OpenGL will perform rendering of drawable objects on the screen.
- In this case we will enable anti aliasing.

```
public class ExerciseOne implements GLEventListener {
        public static void main(String[] args) {
            JFrame.setDefaultLookAndFeelDecorated(false);
            JFrame jframe = new JFrame("Sample using the Callback Framework");
            jframe.setSize(500, 500);
            jframe.setLocationRelativeTo(null);

            /** Create a profile, in this case OpenGL <= 3.0 */
            GLProfile profile = GLProfile.get(GLProfile.GL2);

            /** Configure context */
            GLCapabilities capabilities = new GLCapabilities(profile);

            /** e.g. Enable anti aliasing */
            capabilities.setNumSamples(2);
            capabilities.setSampleBuffers(true);

            GLJPanel canvas = new GLJPanel(capabilities);
            canvas.addGLEventListener(new ExerciseOne());

            /** Put the canvas into a JFrame window */
            jframe.getContentPane().add(canvas);
            jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            /** Show window */
            jframe.setVisible(true);
        }
        ....
}
```

```
public void display(GLAutoDrawable drawable) {
        GL2 gl = drawable.getGL().getGL2();

        gl.glClear(GL.GL_COLOR_BUFFER_BIT);
        gl.glColor3f(0.0f, 1.0f, 0.0f);

        gl.glBegin(GL2.GL_POLYGON);
        gl.glVertex2f(0f, 0.5f);
        gl.glVertex2f(0.5f, -0.4f);

        /** This is just to illustrate using FloatBuffer when v is defined */
        gl.glVertex2fv( FloatBuffer.wrap(new float[] { -0.5f, -0.4f }) );
        gl.glEnd();

        /**
          * Does not return until the effects of all previously
          * called GL commands are complete.
          */
        gl.glFinish();
}
```

Display method

- Code between glBegin() and glEnd(), define the object to be drawn (Polygon)
- Polygon's "corners" are defined by the glVertex3f() commands, or in 2D diagram glVertex2f().
- glFlush() ensures that the drawing commands are actually executed rather than stored in a buffer awaiting additional OpenGL commands

What is a GLAutoDrawable?

“A higher-level abstraction than GLDrawable which supplies an event based mechanism (GLEventListener) for performing OpenGL rendering. A GLAutoDrawable automatically creates a primary rendering context which is associated with the GLAutoDrawable for the lifetime of the object“

A GLAutoDrawable object can access the OpenGL current context with the getGL() method.

What is a GLDrawable?

“An abstraction for an OpenGL rendering target. A GLDrawable's primary functionality is to create OpenGL contexts which can be used to perform rendering. A GLDrawable does not automatically create an OpenGL context, but all implementations of GLAutoDrawable do so upon creation.“

Used Methods

- glClearColor() establishes what color the window will be cleared to.
- glClear() clears the window. Once the clearing color is set, the window is cleared to that color whenever glClear() is called.
- glColor3f() command establishes what color to use for drawing objects – white for this example.
- glFinish() The command does not return until the effects of all previously called GL commands are complete.

```
public void init(GLAutoDrawable drawable) {

        GL gl = drawable.getGL();

        /** This sets the background color */
        gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

}

public void dispose(GLAutoDrawable drawable) {
      System.out.println("Now its time to perform
                                  the release of all OpenGL
                                  resources per GLContext,
                                  such as memory buffers
                                 and GLSL programs.");
}
```

```
public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
        final GL2 gl = drawable.getGL().getGL2();
        if (height <= 0) // Avoid a divide by zero error!
            height = 1;
        final float windowRatio = (float) width / (float) height;
        gl.glViewport(horOffset, vertOffset, width, height);

        // This tells the now we are working on the projection matrix
        gl.glMatrixMode(GL2.GL_PROJECTION);
        gl.glLoadIdentity();
        glu.gluPerspective(FOV, windowRatio, closeClippingDist, farClippingDist);

        // This tells the now we are working on the modelview matrix
        // Where our object info are stored
        gl.glMatrixMode(GL2.GL_MODELVIEW);
        gl.glLoadIdentity();
}
```

Possible Example

What is gl.glMatrixMode ?

- It sets the current matrix model. More specifically it specifies which matrix stack is the target for subsequent matrix operations
- This can be:
  - GL_MODELVIEW
  - GL_PROJECTION
  - GL_TEXTURE

Concepts explained next week

Orthographic Projection

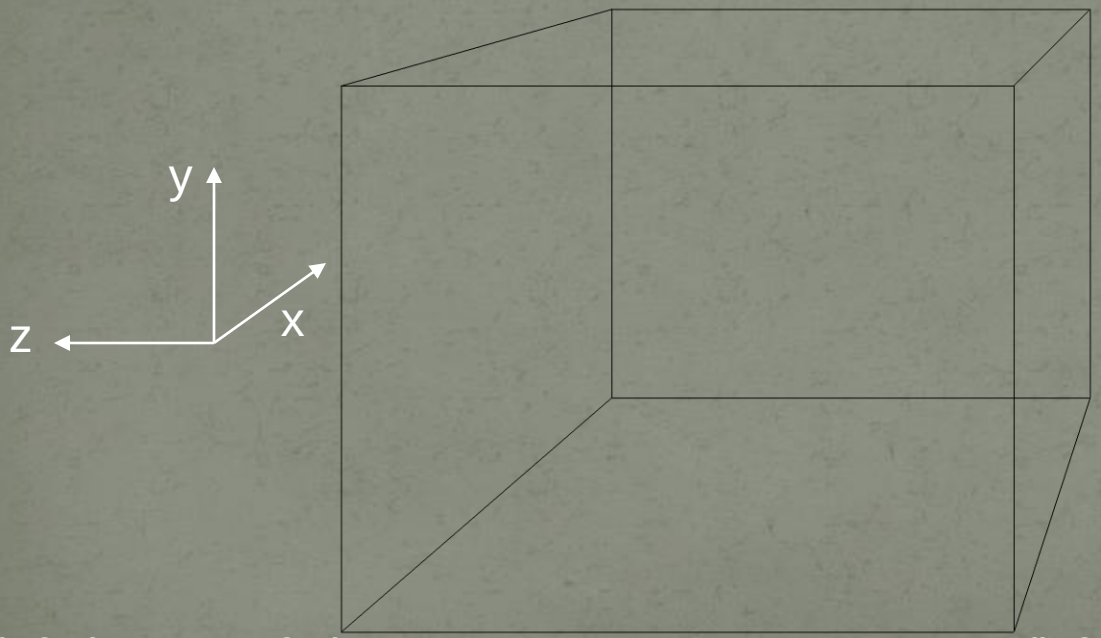
Specifies coordinate system OpenGL assumes as it draws the final image and how the image gets mapped to the screen for orthographic parallel viewing volume.

Orthographic Projection

gl.glOrtho(*left, right, bottom, top, near, far);*

(right, top, -far)

(right, top, -near)



initial point of view

(left, bottom, -far)

(left, bottom, -near)

Perpective Projection

Note glOrtho does not define perspective in 3D, it only defines the space that will be *observable*.

For perspective different OpenGL operators are necessary:

```
gl.glViewport(0, 0, w, h);
gl.glMatrixMode(GL.GL_PROJECTION);
glu.gluPerspective(FOV, windowRatio,
                   closeClippingDist, farClippingDist);
```

These will be discussed in detail in later sections.

Modelview transformations

Modelview transfomations are used to position the view in the scene

glTranslate
glRotate
glScale

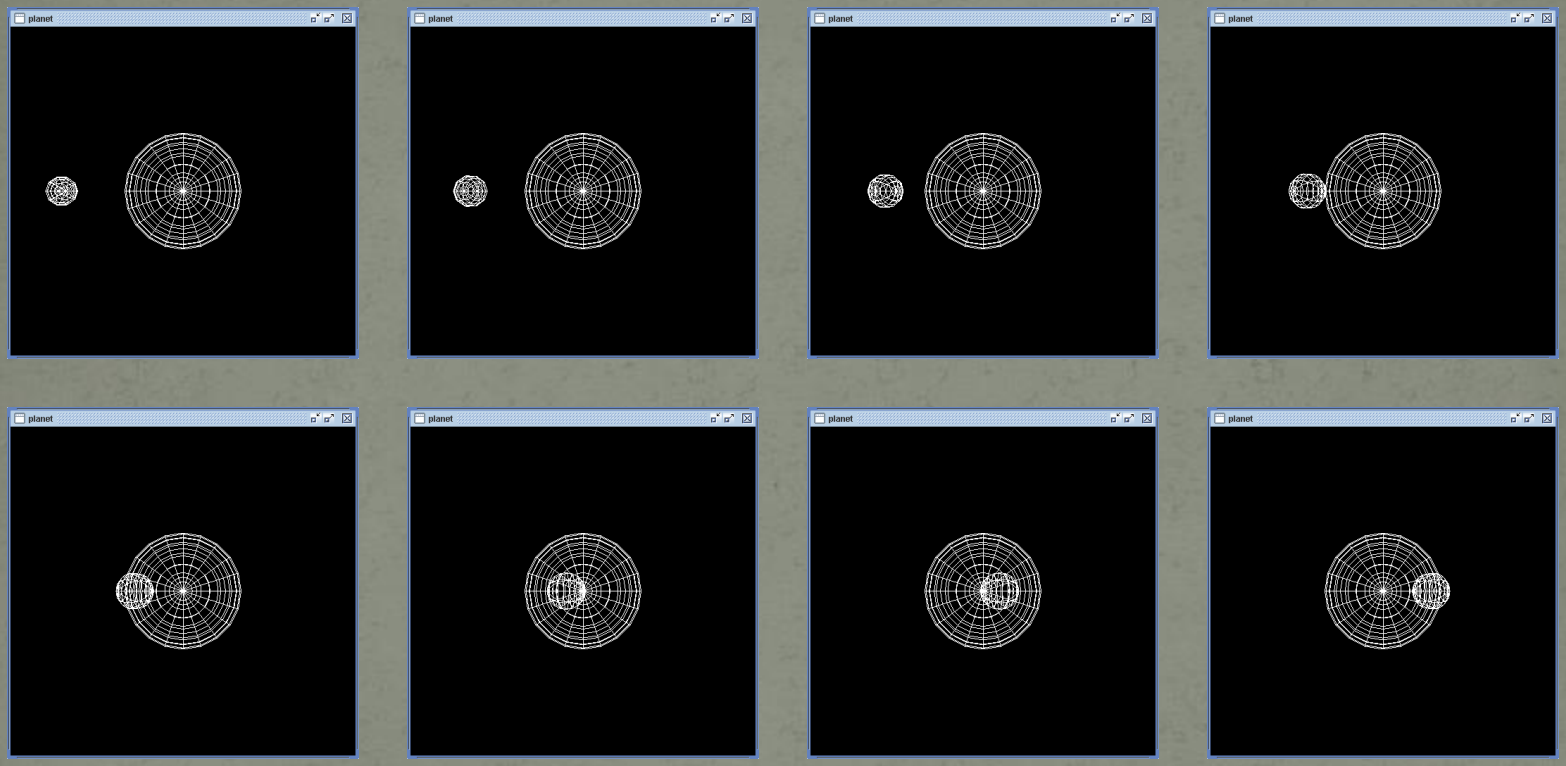
Concepts explained next week

Animation

Elementrary animation in OpenGL

- Computer-graphics screens typically refresh (redraw the picture) approximately 60 to 76 times per second
- Refresh rates faster than 120, however, are beyond the point of diminishing returns, since the human eye is only so good.
- The key reason that motion picture projection works is that each frame is complete when it is displayed.

Simple Animation Sequence



‘Moon’ orbits round ‘planet’. Each frame is redrawn with ‘moon’ rotated slightly each time. When played rapidly in sequence creates illusion of motion.

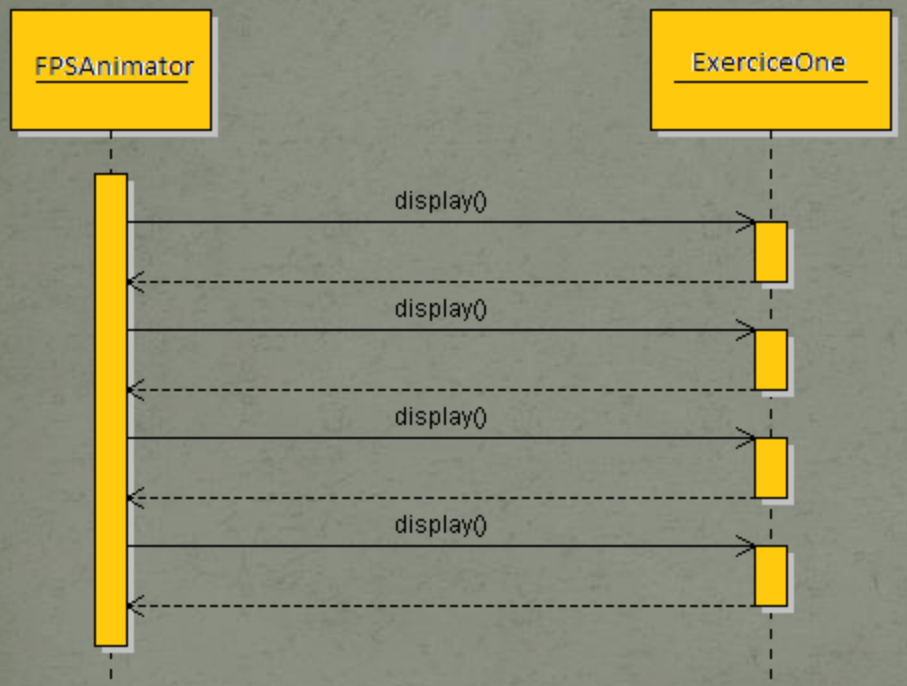
Using FPSAnimator

Aside from the canvas and listener, most games will need a mechanism for triggering regular updates to the canvas. This functionality is available through JOGL's FPSAnimator utility class, which can schedule a call to the canvas' display() method with a frequency set by the user.

FPSAnimator

- FPSAnimator, Frames / Second animator Java thread.
- new FPSAnimator(canvas, 60);
this will try to force 60 frames per second to be rendered completely on screen.
- FPSAnimator does not guarantee that 60 frames will be shown, this is a maximum that can be shown.
- FPSAnimator will force all drawable objects to finish execution before frame is displayed.
- FPSAnimator forces the display method to complete execution up to the number specified in the argument.

FPSAnimator



FSPAnimator forces display() to execute up to N times each second. Guarantees that display will complete execution before next frame is rendered on screen.

Animator Thread

```
public static void main(String[ ] args) {

   /**
    *  Part 1: Set up OpenGL canvas and Java Swing JFrame and define
    *  drawing elements here.
    */

    FPSAnimator animator = new FPSAnimator(canvas, 60);

    /* Part 2: Initialise frame look and feel and attach drawing canvas */

    animator.start();

}
```

Homework

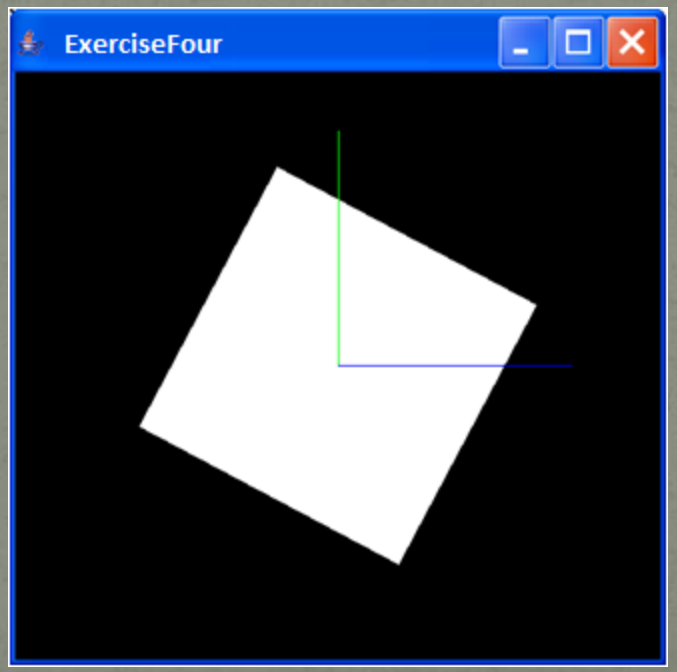
- Consider very simple animation that creates a 2D square and rotates square in real time in JFrame
- The square to rotate by small amount 60 times per second to create illusion of animation

Push / Pop Matrix

- glPushMatrix() means "remember where you are"
- glPopMatrix() means "go back to where you were."

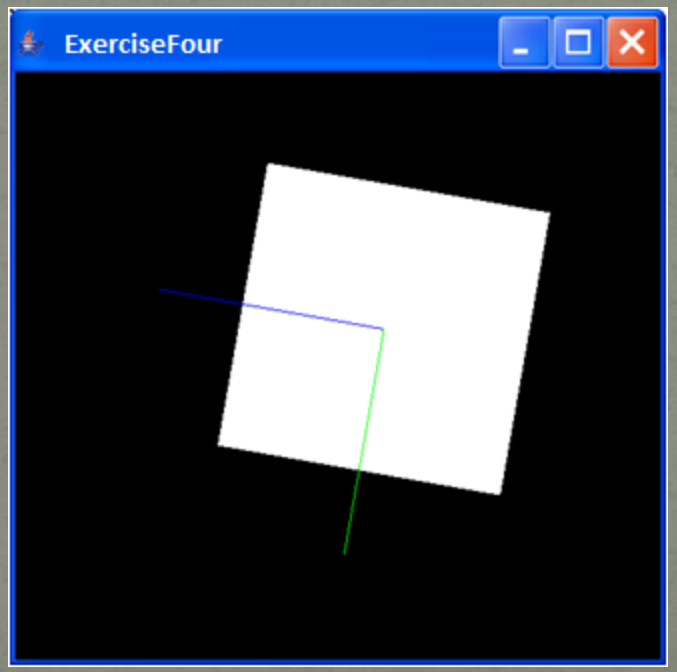
Push / Pop Matrix

- Using push and pop, the x and y axis remain stationary, whilst the square rotates

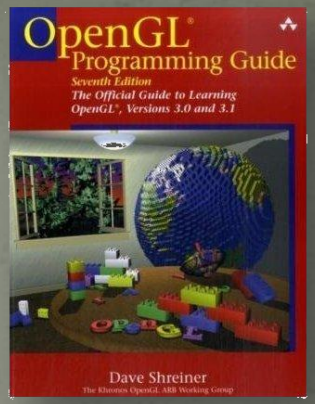


Push / Pop Matrix

- Without push and pop, the x and y axis rotate as well as the square



Books



- Free online copy of the OpenGL Redbook
  - www.glprogramming.com/red/index.html
- Goes in-depth into many aspects of computer graphics that are invaluable as aid to understanding subject.
- Note: written in C, not Java. Many examples from Redbook have been translated into Java and are available online:
  - http://pepijn.fab4.be/software/nehe-java-ports/