Chapter 2 - Graphics Programming with JOGL

- Graphics Software: Classification and History
- JOGL Hello World Program
- 2D Coordinate Systems in JOGL
- Dealing with Window Reshaping
- 3D Coordinate Systems in JOGL

Software Using Graphics

- Graphics is always finally rendered by hardware:
  - Monitor, projector, head-mounted display, (2D/3D) printer, plotter, vehicle …
- Special-purpose software packages:
  - Dedicated to a special application area
  - Based on general graphics software as lower layers
  - User may need high geometrical skills in some cases, but principles of graphics programming are hidden from user
  - Examples: CAD software, architectural software, medical software
- General-purpose graphics software:
  - Typically libraries or frameworks to be used for construction of other software
  - Defines a “computer graphics application programming interface” (CG API)
  - Exist on various abstraction levels
  - Can be bound to various programming languages (mostly used: C++)

Low-Level and High-Level CG APIs

- Low-Level APIs
  - – Provide functionality for constructing and rendering (3D) graphical views
  - – Abstracts away from concrete graphics hardware (existence and size of buffers, hardware support for certain functionality)
  - – Targets at hardware-supported execution
  - – Dominant examples: OpenGL (open standard), Direct3D (Microsoft)
- High-Level APIs
  - – Provide further abstraction for creation of scene, usually based on *scene graph*
  - – Targets portability across platforms
  - – Implementation based on low-level API
  - – Typical examples:
    - Java 3D (runs on OpenGL or Direct3D), not further developed since 2008
    - Open Inventor (originally IRIS Inventor)
    - VRML
    - RenderMan (Pixar)

History of Graphics Software Standards

- Graphical Kernel System (GKS), 1984
  - First graphics software standard adopted by ISO
  - Originally 2D, 3D extension was added later
- Programmer's Hierarchical Interactive Graphics System (PHIGS)
  - Successor of GKS, ISO standard by 1989
  - 3D oriented
  - Implemented for instance by DEC, IBM, Sun, and based on X Window system
  - Considered to be the graphics standard of the 90s
- Major player appears: Silicon Graphics Inc. (SGI)
  - Producer of graphics workstations (founded 1981)
  - "IRIS" workstations popular in research and development
  - Software based on proprietary dialect of Unix ("IRIX")
  - Bankruptcy in 2009, acquired by "Rackable Systems",
    renamed to "Silicon Graphics International",
    concentrating on High-Performance Computing

IRIS Graphics Library and OpenGL

- IRIS GL = Integrated Raster Imaging System Graphics Library
  - Developed by Silicon Graphics
  - Became popular on other hardware platforms
- 1990s:
  Hardware-independent version of IRIS GL
  = OpenGL
  - First OpenGL spec by SGI 1992
  - Maintained by OpenGL Architecture Review Board
  - Later transition to “Khronos Group”
    - Industry consortium
    - Selected members: AMD, Apple, Google, Intel, Motorola, Mozilla, Samsung, Oracle/Sun, Texas Instruments
  - Has been influential on development of 3D acceleration hardware

SGI IRIS 4D/35 (1991)
35 MHz CPU (RISC)
Up to 128 MB RAM

OpenGL Evolution

- Until OpenGL 1.5 (2003)
  - Fixed Function Pipeline (FFP): Triangles, textures and attributes passed to GPU
  - GPU simply renders based on given information
- Programmable Shaders
  - Appearing since 2000 in new graphics hardware
  - Custom code executed on GPU, not only fixed functions
  - OpenGL Shading Language (GLSL)
  - Programmable Shader Pipeline (PSP)
- OpenGL 2.0 (2004): Subsumes FFP, PSP, and GLSL
- 2005: OpenGL ES for Embedded Systems
- 2007: PSP only subset for Embedded Systems
- July 2010: OpenGL 4.1, fully compatible with OpenGL ES 2.0
- Examples based on Java OpenGL 2.0 (compatible with FFP and PSP)

OpenGL Language Bindings

- Traditional language binding for OpenGL: C++
  - Very good performance on many platforms
  - Leads to additional complexity in bridging to window management systems
- For this lecture: Java Binding for Open GL (JOGL)
  - Originally developed by Kenneth Bradley Russell & Christopher John Kline
  - Further developed by Sun Microsystems Game Technology Group
  - Since 2010, Open Source project
    - Now hosted under “jogamp.org”
  - Requires download of JAR files and native libraries
    - Not “pure Java” but based on platform-specific native code
- Interesting trend: WebGL
  - JavaScript API for OpenGL
  - Based on OpenGL ES 2.0, uses HTML5 canvas and DOM interface
  - Supported by Firefox, Chrome (and somehow by Safari, Opera)

Chapter 2 - Graphics Programming with JOGL

- Graphics Software: Classification and History
- JOGL Hello World Program
- 2D Coordinate Systems in JOGL
- Dealing with Window Reshaping
- 3D Coordinate Systems in JOGL

JOGL Hello World (Based on Swing) – Basics

```
package hello;

import javax.swing.*;
import javax.media.opengl.*;
import javax.media.opengl.awt.GLCanvas;


public class HelloWorld extends JFrame {

   GLCanvas canvas;

   public HelloWorld() {

      GLProfile glp = GLProfile.getDefault();
      GLCapabilities caps = new GLCapabilities(glp);
      canvas = new GLCanvas(caps);

      add(canvas);

      setTitle("Jogl Hello World");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(300,200);
      setVisible(true);
   }

  public static void main(String args[]) {
     new HelloWorld();
  }
}
```

JOGL Hello World – Event Listener Approach

```
   public class HelloWorld extends JFrame {

   GLCanvas canvas;

   public HelloWorld() {
      ...
      canvas = new GLCanvas(caps);
      canvas.addGLEventListener(new SceneView());
      add(canvas);
      ...
   }

   class SceneView implements GLEventListener {

      public void init(GLAutoDrawable drawable) {...
      }

      public void display(GLAutoDrawable drawable) {...
      }

      public void reshape(GLAutoDrawable drawable, int arg1, int arg2, int arg3, int arg4) {...
      }

      public void dispose(GLAutoDrawable drawable) {
      }
  }

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

JOGL Hello World – Displaying Something

```
public class HelloWorld extends JFrame {
   public HelloWorld() {
      … canvas.addGLEventListener(new SceneView());…
   }

   class SceneView implements GLEventListener {

      public void init(GLAutoDrawable drawable) {
         GL2 gl = drawable.getGL().getGL2();
         gl.glClearColor(0, 0, 0, 0); // black background
         }

      public void display(GLAutoDrawable drawable) {
         GL2 gl = drawable.getGL().getGL2();
         gl.glClear(GL2.GL_COLOR_BUFFER_BIT); // clear background
         gl.glColor3d(1, 0, 0); //draw in red

         gl.glBegin(GL2.GL_LINES); // draw H
            gl.glVertex2d(-0.8, 0.8);
            gl.glVertex2d(-0.8, -0.8);
            gl.glVertex2d(-0.8, 0.0);
            gl.glVertex2d(-0.4, 0.0);
            gl.glVertex2d(-0.4, 0.8);
            gl.glVertex2d(-0.4, -0.8);
         gl.glEnd();
         ...
      } ...
   }
… }
```

OpenGL Name Conventions (JOGL)

- OpenGL functions
  - start with “gl”
  - are written in mixed case
- OpenGL constants
  - start with “GL_”
  - are written in upper case
- Number of parameters (for colors or points)
  - are given as number included in function name
- Versions of functions, different in argument number or types
  - are indicated by letter(s) at the end of function name, for instance:
  - “d” for “double”
  - “f” for “float”
  - “i” for “integer”
  - * at end of function name (in doc) indicates that several versions exist

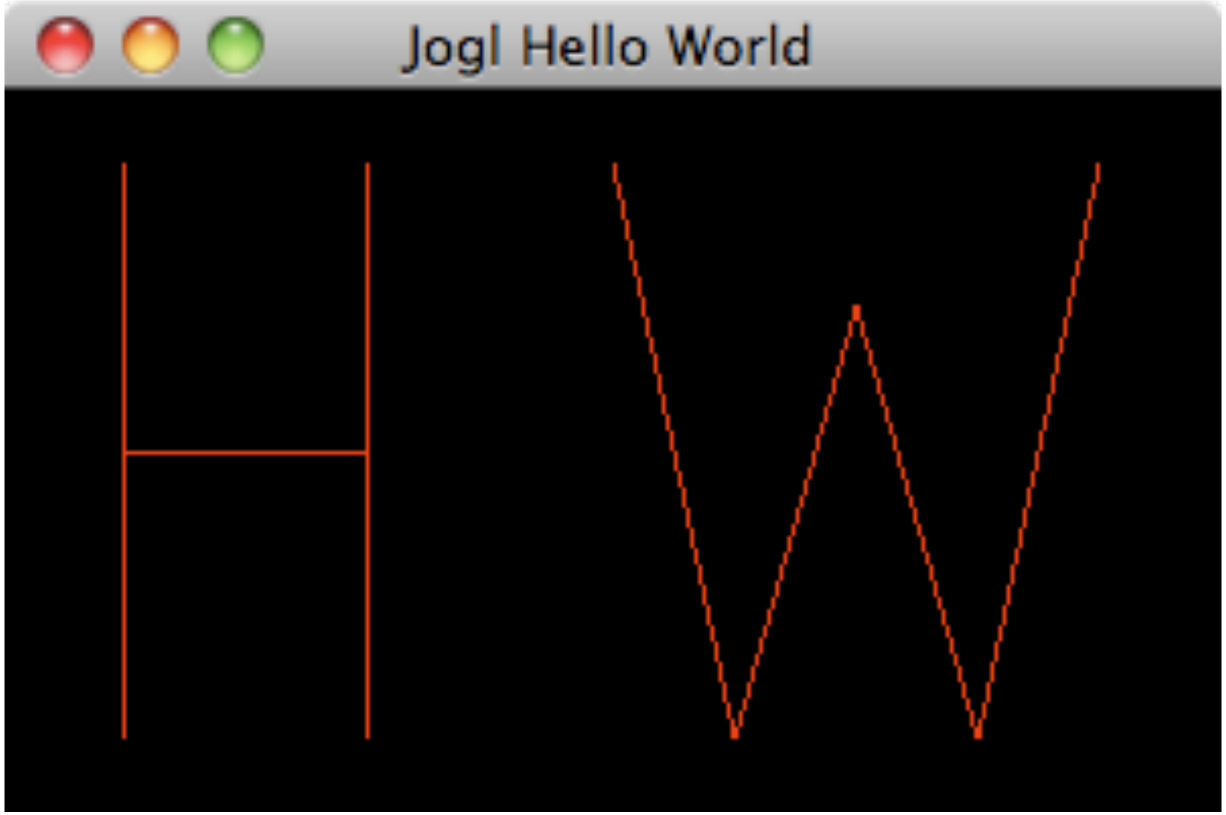
The OpenGL State Machine

- OpenGL stores internally a large amount of information
  - Current colors to be used for drawing something
  - Capability restrictions of the available hardware
  - Various matrices related to viewpoint and projection (see later)
  - …
- These “global variables” are not fully compatible with object-oriented thinking
  - In your code: Get access to relevant global information store (if necessary, same code at different places)
  - Adjust global information before triggering actions

```
GL2 gl = drawable.getGL().getGL2();
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT); // clear background
    gl.glColor3d(1, 0, 0); //draw in red
```

Questions

- This was code for drawing an “H”.
- How to draw an “W” besides it?
- Which coordinate system is used?

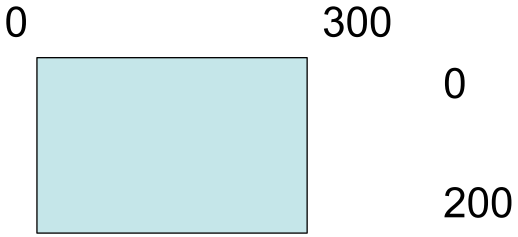


Chapter 2 - Graphics Programming with JOGL

- Graphics Software: Classification and History
- JOGL Hello World Program
- 2D Coordinate Systems in JOGL
- Dealing with Window Reshaping
- 3D Coordinate Systems in JOGL

How to Create a Classical 2D Coordinate System?

- There are many coordinate systems involved:
  - World coordinates: Where the object are placed in a (virtual) universe
  - View coordinates: Where the objects appear from a certain viewpoint
  - Device coordinates: Where an object’s pixel appears on a device
- Simple case 2D, defining classical world coordinates:
  - x values increasing towards right
  - y values increasing downwards
  - Integer coordinate values
  - Parameters: *left*, *right*, *bottom*, *top*



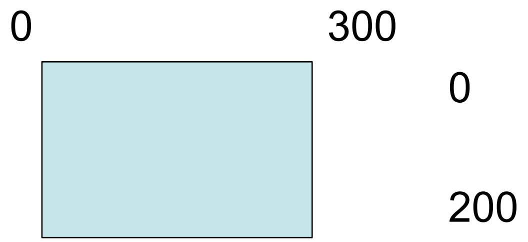
- Special case of a 3D “orthogonal projection”
  - Depth values irrelevant here: “2D vertices” (get z-coordinate value 0)
  - Look at the 3D scene without distortions, do not omit objects at depth 0 (so-called “near plane” at least 0, “far plane” greater than 0)
- OpenGL: glOrtho(*left*, *right*, *bottom*, *top*, *near*, *far*)

HelloWorld (2D) Using Self-Defined Coordinates

```
public void init(GLAutoDrawable drawable) {
     GL2 gl = drawable.getGL().getGL2();
     gl.glClearColor(0, 0, 0, 0); // black background

     gl.glMatrixMode(GL2.GL_PROJECTION);
     gl.glLoadIdentity();
     gl.glOrthof(0, 300, 200, 0, 0, 1);
       // left, right, bottom, top, near, far
}
```

- Specifying objects (vertices):

```
     gl.glBegin(GL2.GL_LINES); // draw H
       gl.glVertex2i(25, 25);
       gl.glVertex2i(25, 175);
       gl.glVertex2i(25, 100);
       gl.glVertex2i(100, 100);
       gl.glVertex2i(100, 25);
       gl.glVertex2i(100, 175);
     gl.glEnd();
```



HelloWorld Using Different Self-Defined Coordinates

```
public void init(GLAutoDrawable drawable) {
     GL2 gl = drawable.getGL().getGL2();
     gl.glClearColor(0, 0, 0, 0); // black background

     gl.glMatrixMode(GL2.GL_PROJECTION);
     gl.glLoadIdentity();
     gl.glOrthof(0, 6.5f, 4, 0, 0, 1); // left, right, bottom, top, …
}
```

- Specifying objects (vertices):

```
     gl.glBegin(GL2.GL_LINES); // draw H
        gl.glVertex2d(0.5, 0.5);
        gl.glVertex2d(0.5, 3.5);
        gl.glVertex2d(0.5, 2);
        gl.glVertex2d(2, 2);
        gl.glVertex2d(2, 0.5);
        gl.glVertex2d(2, 3.5);
     gl.glEnd();
```

Chapter 2 - Graphics Programming with JOGL

- Graphics Software: Classification and History
- JOGL Hello World Program
- 2D Coordinate Systems in JOGL
- Dealing with Window Reshaping
- 3D Coordinate Systems in JOGL

Window Reshaping

- Windows (JFrame objects) can be moved and resized using operating system functions
- After every reshape/repositioning, “display” is called
  - Everything is redrawn according to current projection
- What happens, e.g., with a square when the window is reshaped to a different aspect ratio?
- Solution:
  - (a) Notification on reshape: Event handler
  - (b) Drawing on specific part of canvas: View port

Reshape() Callback Function

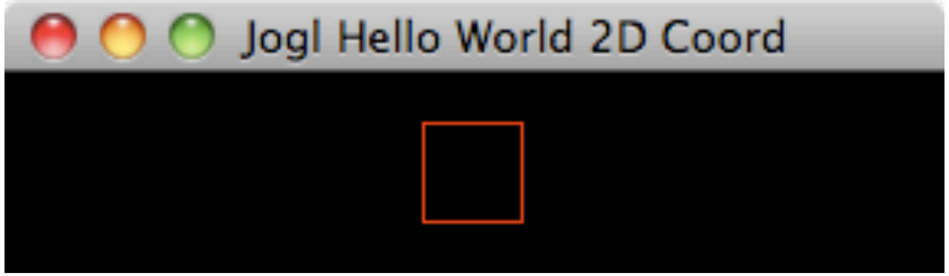
- Event handler (callback) function

```
public void reshape
    (GLAutoDrawable drawable, int x, int y, int w, int h)
```

- Called when window is reshaped (and before first display)
  - Definition of projection can be done within *reshape()*
- Afterwards contents are rendered using *display()*
- Parameters (pixels):
  - *x*, *y*: Position on screen (of bottom left corner of window)
  - *w*, *h*: New width and height of window
- Defining a *view port* on which to draw within the window:
  - *glViewport()* function
  - Parameters in analogy to *reshape()*

Keeping a Square Squared

```
public void display(GLAutoDrawable drawable) {
   GL2 gl = drawable.getGL().getGL2();
   ...
   gl.glBegin(GL2.GL_LINE_LOOP); // draw square
      gl.glVertex2d(1, 1);
      gl.glVertex2d(1, 3);
      gl.glVertex2d(3, 3);
      gl.glVertex2d(3, 1);
   gl.glEnd();
}
```



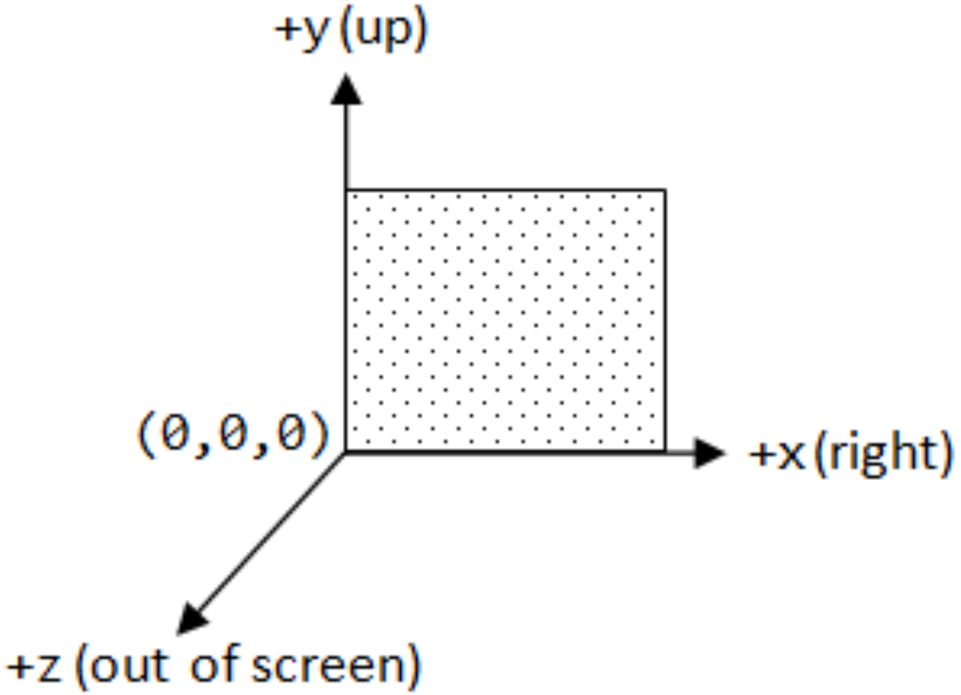
```
public void reshape(GLAutoDrawable drawable, int x, int y, int w, int h) {
   GL2 gl = drawable.getGL().getGL2();
   gl.glViewport
      (Math.max(0,(w-h)/2),Math.max(0,(h-w)/2), Math.min(w,h), Math.min(w,h));
   gl.glMatrixMode(GL2.GL_PROJECTION);
   gl.glLoadIdentity();
   gl.glOrthof(0, 4, 4, 0, 0, 1);
}
```

Chapter 2 - Graphics Programming with JOGL

- Graphics Software: Classification and History
- JOGL Hello World Program
- 2D Coordinate Systems in JOGL
- Dealing with Window Reshaping
- 3D Coordinate Systems in JOGL

OpenGL 3D Reference Coordinates

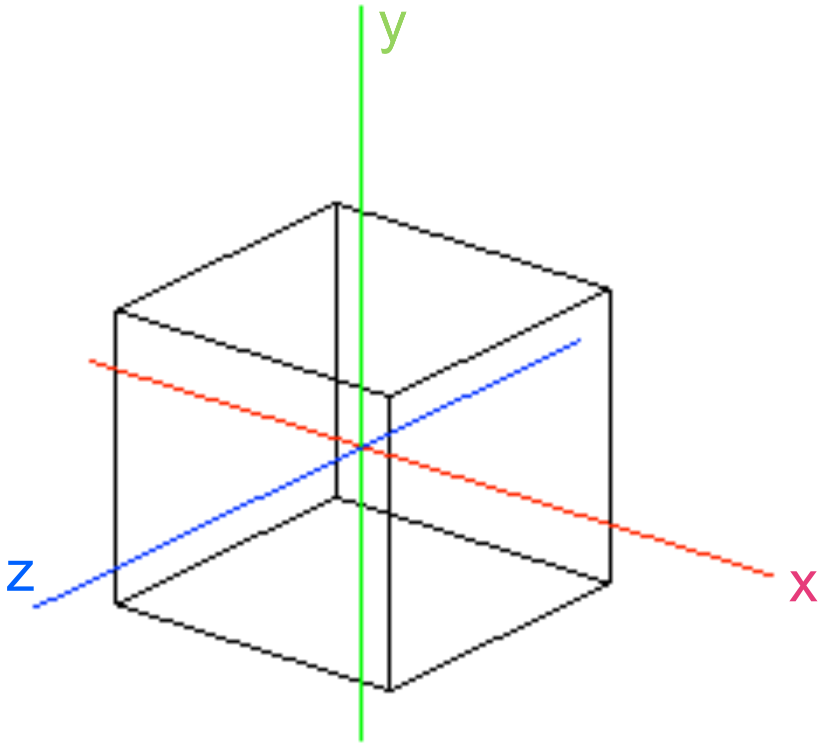
- Right-handed coordinate system
- Typically, y axis is not inverted
    - y axis points “up”
    - inversion is typical in 2D graphics



Pictures: http://www3.ntu.edu.sg/home/ehchua/

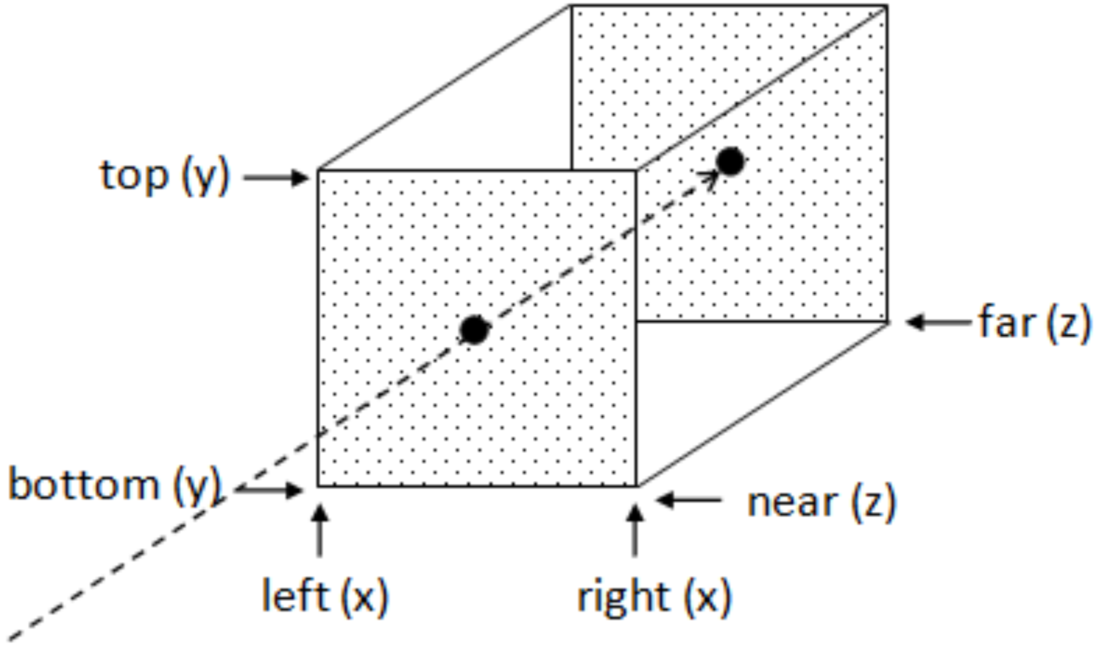
Simple 3D Object: Wireframe of a Cube

```
gl.glBegin(GL2.GL_LINE_LOOP); // draw front side
   gl.glVertex3d(-1, -1, 1);
   gl.glVertex3d(1, -1, 1);
   gl.glVertex3d(1, 1, 1);
   gl.glVertex3d(-1, 1, 1);
gl.glEnd();

gl.glBegin(GL2.GL_LINE_LOOP); // draw back side
   gl.glVertex3d(-1, -1, -1);
   gl.glVertex3d(1, -1, -1);
   gl.glVertex3d(1, 1, -1);
   gl.glVertex3d(-1, 1, -1);
gl.glEnd();

gl.glBegin(GL2.GL_LINES); // draw connections
   gl.glVertex3d(-1, -1, -1); gl.glVertex3d(-1, -1, 1);
   gl.glVertex3d(1, -1, -1); gl.glVertex3d(1, -1, 1);
   gl.glVertex3d(1, 1, -1); gl.glVertex3d(1, 1, 1);
   gl.glVertex3d(-1, 1, -1); gl.glVertex3d(-1, 1, 1);
gl.glEnd();
```



Projection from 3D to 2D Coordinates

- Two main types of projection (details later):
  - parallel (orthographic) – comparable to a telescopic view from distance
  - perspective
- Here: Parallel (orthographic) projection
  - Defines a “clipping volume” (parallelepiped = “box”)
  - Specification by six values
    - left (x axis)
    - right (x axis)
    - bottom (y axis)
    - top (y axis)
    - near (z axis)
    - far (z axis)



View direction

Pictures: http://www3.ntu.edu.sg/home/ehchua/

Specifying an Orthographic Projection in JOGL

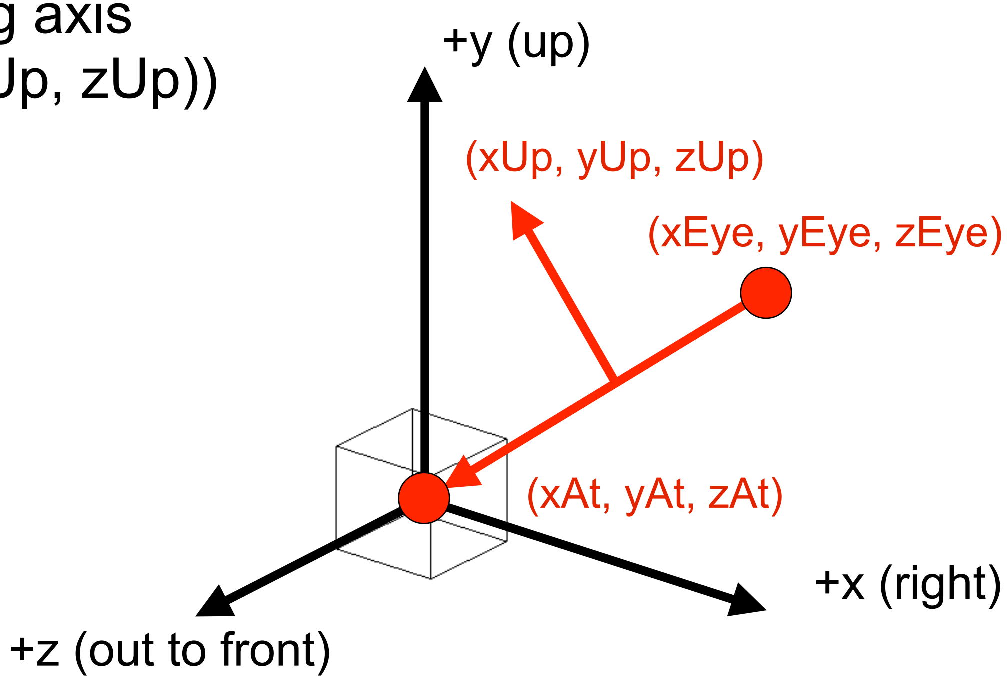
- Typically done within “reshape” callback function
- Two “matrix modes”, switchable
  - Projection (relevant here)
  - Modelview (model transformations and camera positioning)
- Commands essentially combine matrices
  - Reset with identity matrix for a clear starting point

```
GL2 gl = drawable.getGL().getGL2();
gl.glMatrixMode(GL2.GL_PROJECTION);
gl.glLoadIdentity();
gl.glOrthof(-3, 3, -3, 3, 0, 100);
```

left, right, bottom, top, near, far

Camera Positioning

- Camera is positioned within reference coordinates
- Necessary parameters:
  - Location of camera as *point* (xEye, yEye, zEye)
  - Viewing direction (in OpenGL given as *point* (xAt, yAt, zAt) looked at)
  - Orientation of the camera on viewing axis (in OpenGL given as *vector* (xUp, yUp, zUp))



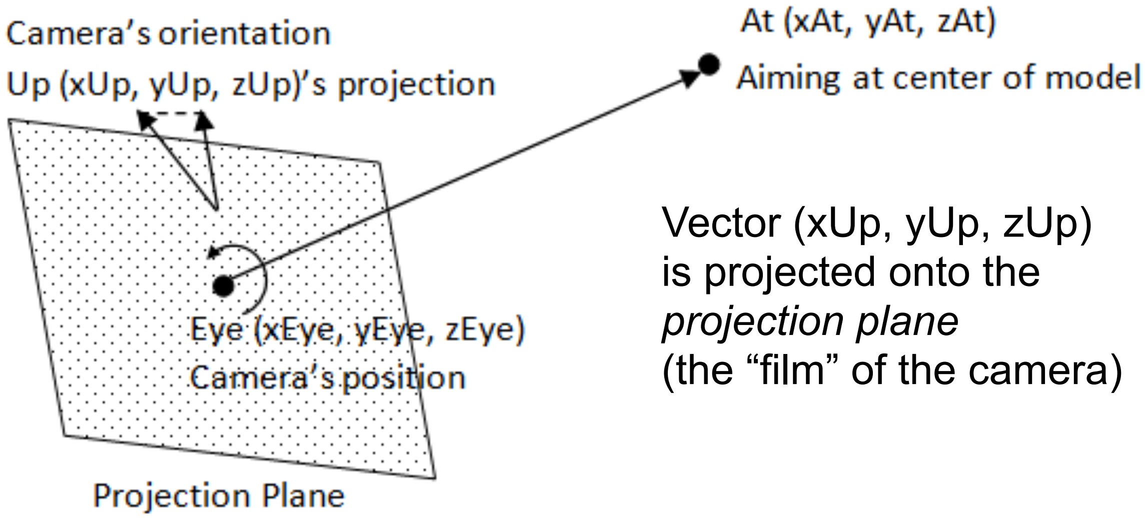
Camera Positioning in JOGL: LookAt

- Typically done within “display” callback function
- “Matrix mode” switched to “Modelview”

```
GLU glu = new GLU(); // utility library object
gl.glMatrixMode(GL2.GL_MODELVIEW);
gl.glLoadIdentity();
glu.gluLookAt(4, 3, 5, 0, 0, 0, 0, 1, 0);
```

xEye, yEye, zEye, xAt, yAt, zAt, xUp, yUp, zUp

GLU = GL Utilities



Vector (xUp, yUp, zUp)
is projected onto the
projection plane
(the “film” of the camera)

Pictures: http://www3.ntu.edu.sg/home/ehchua/

Many Questions?

- What happens if we apply a similar projection specification as in the 2D case?
  - 2D: `gl.glOrthof(0, 300, 200, 0, 0, 1);`
  - 3D: `gl.glOrthof(-3, 3, -3, 3, 0, 100);`
- Where is the coordinate system we are using actually defined?
- How can be better work with objects and views not concentrated at the coordinate origin?
- Why is there such a difference between the two “matrix modes” in OpenGL?
- … Maybe we have to understand coordinate systems and their transformations better ...

Literature Recommendations and links

- Hearn, Baker, Carithers: Computer Graphics with OpenGL,
  4th edition, Pearson 2011
- Lehrstuhl Prof. B. Möller, Uni Augsburg: Eine Einführung in JOGL
  http://www.informatik.uni-augsburg.de/lehrstuehle/dbis/pmi/lectures/ss10/graphikprogrammierung/script/