MORE OPENGL

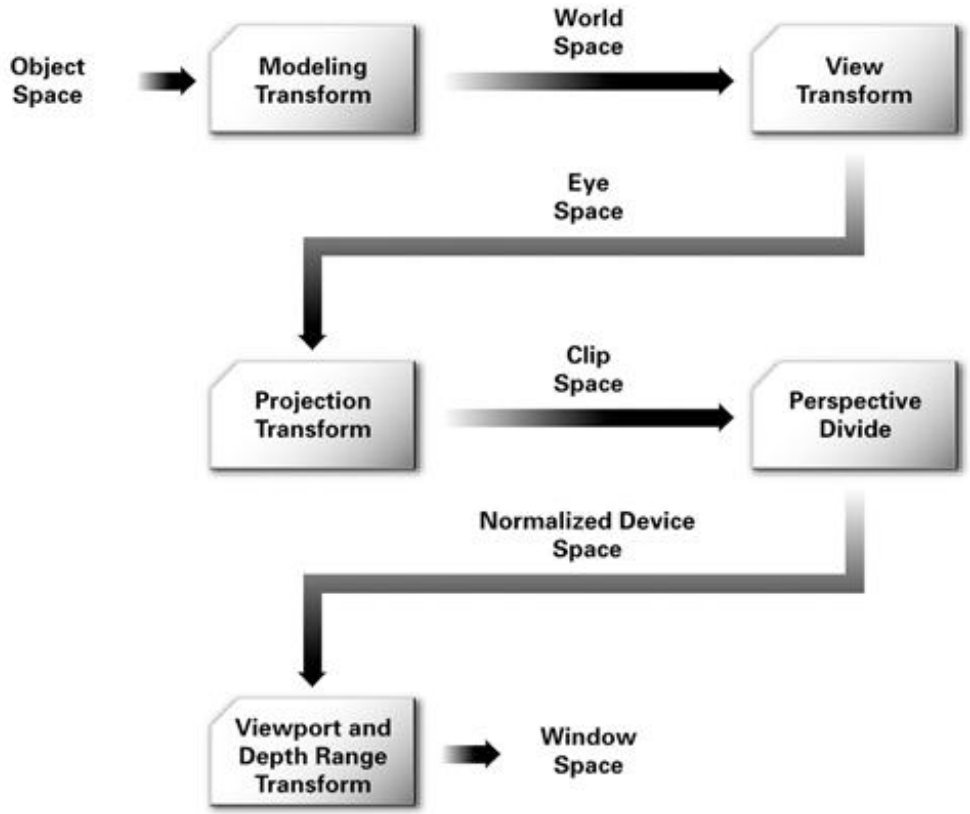
Pramook Khungurn

CS 4621, Fall 2011

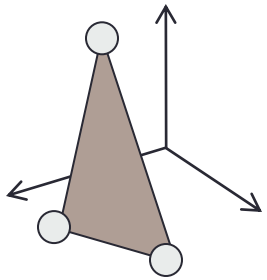
SETTING UP THE CAMERA

Recall: OpenGL Vertex Transformations

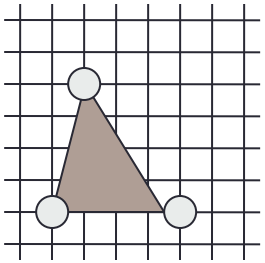
- Coordinates specified by glVertex are transformed.
- End result: window coordinates (in pixels)



Recall: OpenGL Vertex Transformation



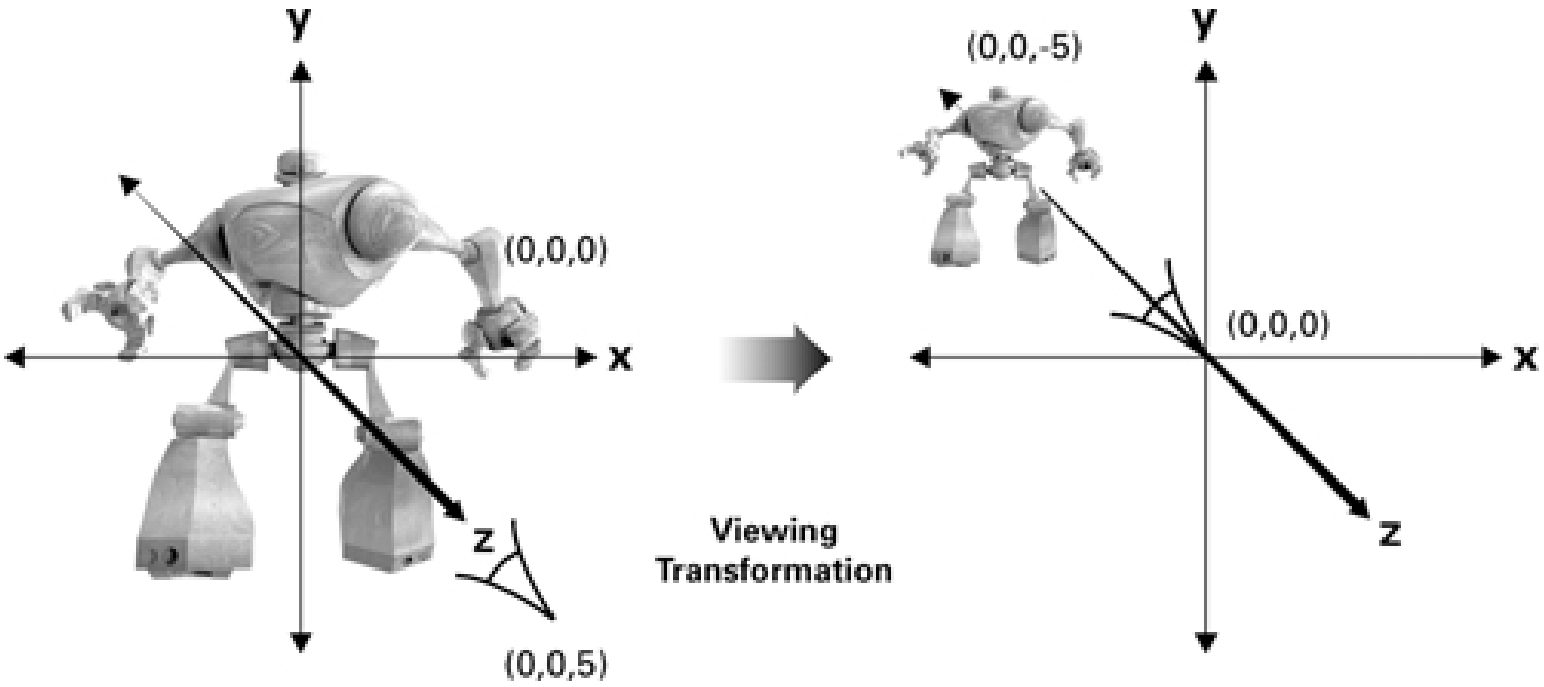
$$\begin{bmatrix} x_w \\ y_w \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \text{Viewport} \\ \text{Transform} \end{bmatrix} \begin{bmatrix} \text{Perspective} \\ \text{Divide} \end{bmatrix} \begin{bmatrix} \text{Projection} \\ \text{Transform} \end{bmatrix} \begin{bmatrix} \text{View} \\ \text{Transform} \end{bmatrix} \begin{bmatrix} \text{Model} \\ \text{Transform} \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$



Recall: View Transform

- Used to set the camera.
- Eye space is a coordinate system where:
  - Eye is at (0,0,0).
  - Look in negative z direction.
  - Y-axis is “up.”
- View transform
  - world space → eye space

Recall: View Transform

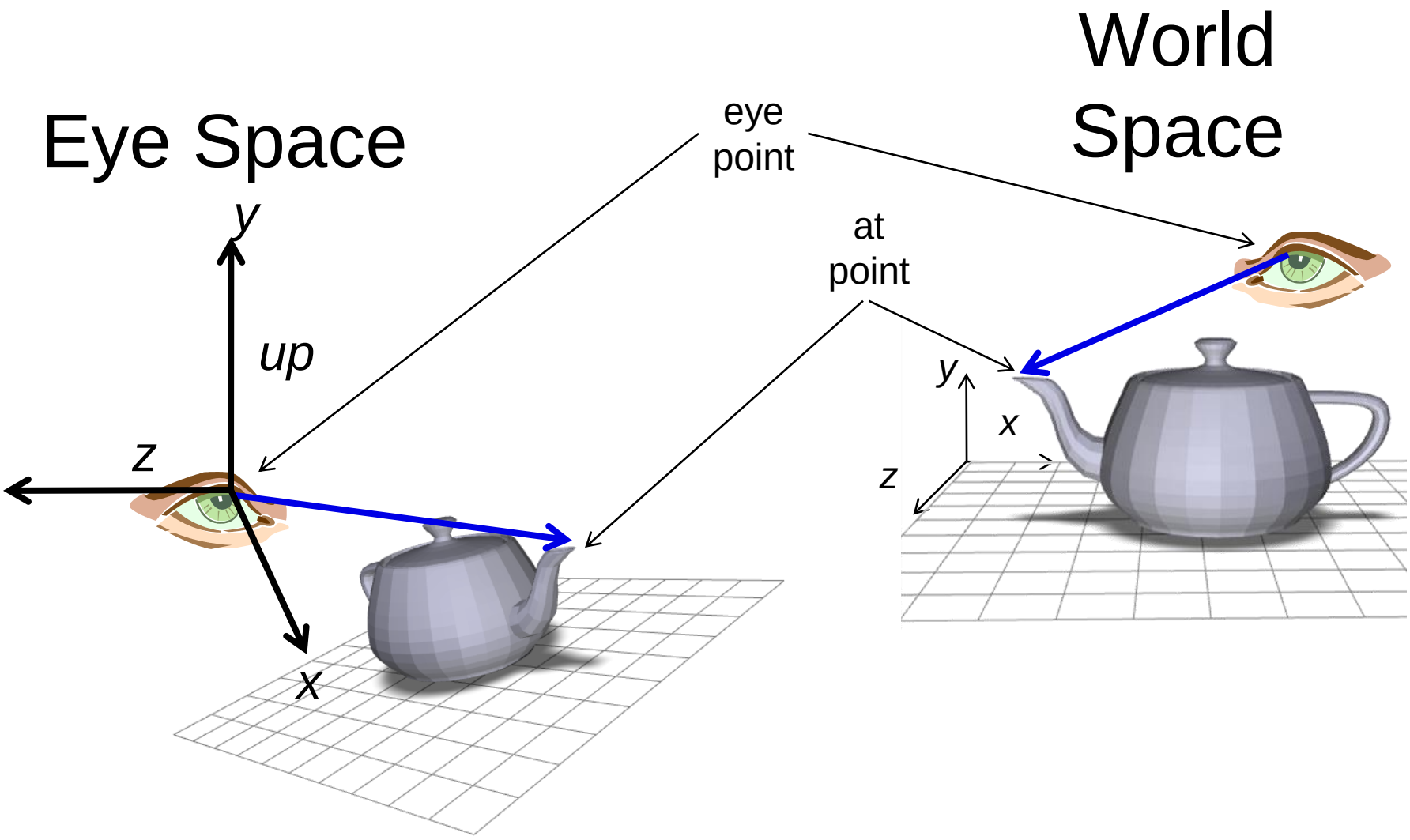


LookAt Transform

- A simple way to set up the camera.

- Parameters
  - eye = position of the camera in world space
  - at = position that the camera looks at in world space
  - up = vector in world space telling which direction is considered “up”

LookAt Transform in Action



gluLookAt

- gluLookAt(eyeX, eyeY, eyeZ, atX, atY, atZ, upX, upY, upZ)
- Set current matrix M = ML where L is the LookAt matrix.
- L transforms the coordinate so that:
  - The origin of the new coordinate system is **eye**.
  - The –z direction is the direction from **eye** to **at**.
    - z parallel to **eye** – **at**
    - y points in the general direction of **up**
- Much like camera setup in PA1.

Where to Use glLookAt?

- Before the any modeling transforms.
- Most of the time:

```
final GL2 gl = drawable.getGL().getGL2();
GLU glu = new GLU();

gl.glMatrixMode(GL2.GL_MODELVIEW);
gl.glLoadIdentity();

glu.gluLookAt(0, 0, 5, 0, 0, 0, 0, 1, 0);

gl.glTranslated(1, -1, 0);
gl.glRotate(30, 0, 0, 1);

/* draw stuffs */
```

Select and clear modelview matrix.

View transform

Modeling transform

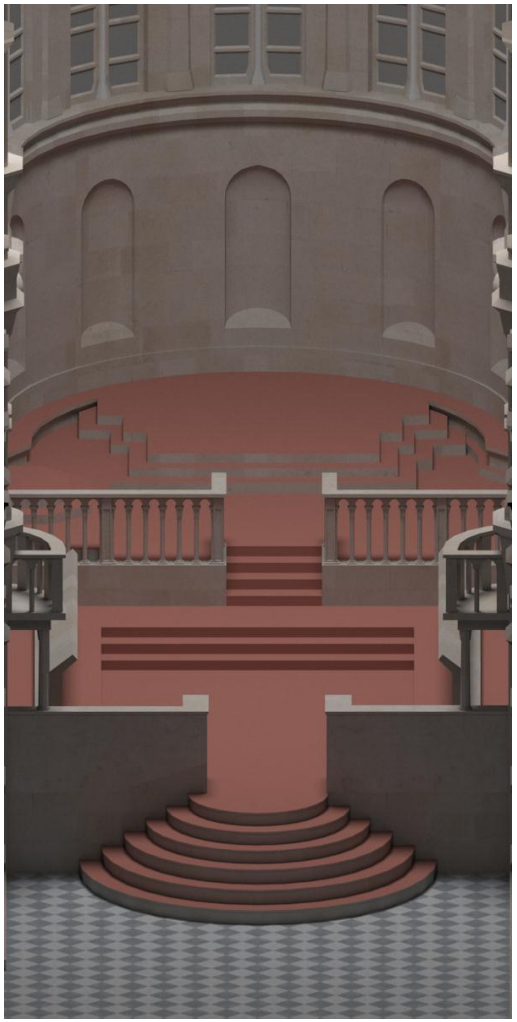
SETTING UP PROJECTION

Projection Transform

- eye space → clip space
- Coordinates in clip space tells us which vertex we see.
- Decision process: visible vertices must satisfy:
  - $-1 \leq x \leq 1$
  - $-1 \leq y \leq 1$
  - $-1 \leq z \leq 1$
  - This is called the *canonical view volume.*

- Projection transform also define 3D → 2D mapping.
  - Affects sense of depth.

Two Popular Projections



orthographic

perspective

Projection Transform in OpenGL

- OpenGL stores projection transform matrix.
- To choose the matrix:
  - gl.glMatrixMode(GL2.GL_PROJECTION)
- Any matrix manipulation commands can be used.
  - gl.glLoadIdentity
  - gl.glLoadMatrix
  - gl.glTranslated, gl.glRotated, gl.glScaled
- There are specific commands to specify projection matrix.

Specifying Orthographic Projection

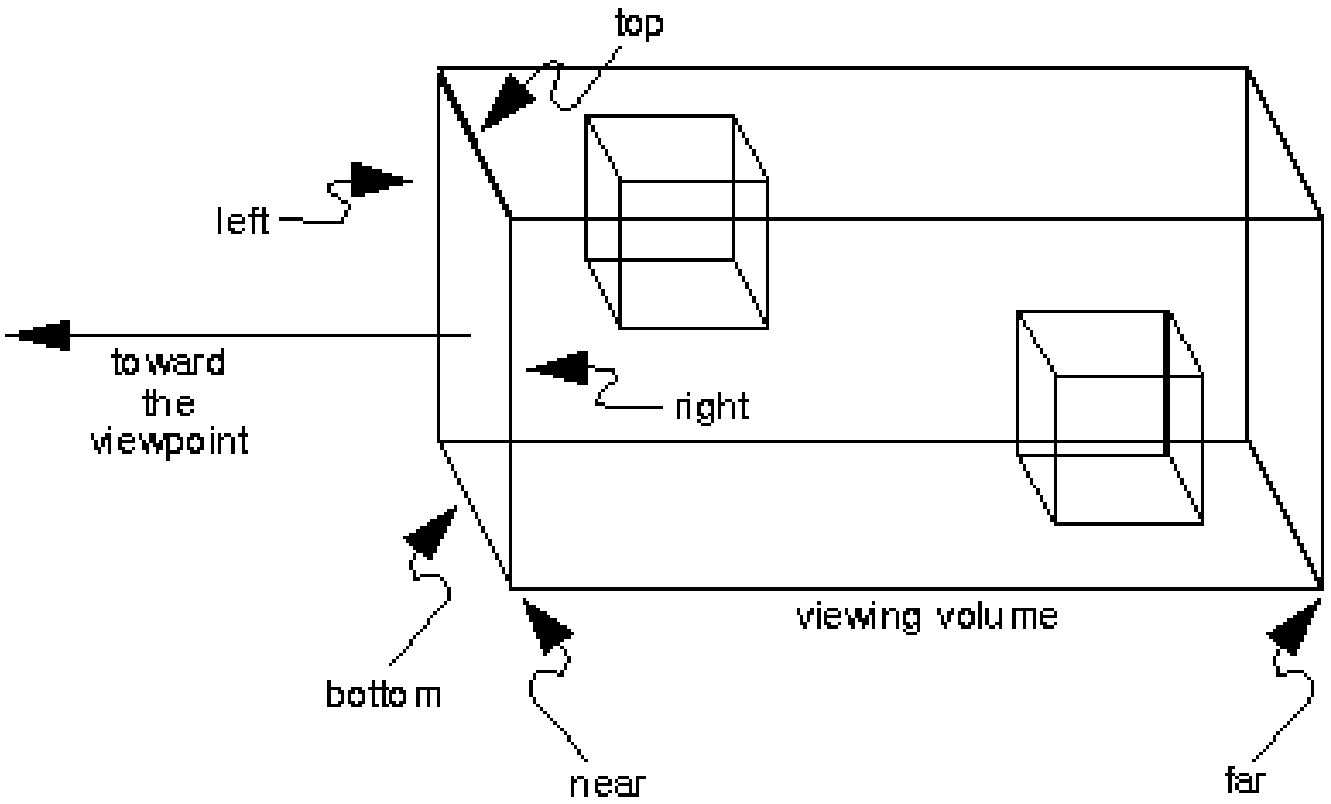
- Specifying the prism of view volume in eye space.

- Defined by 3 pairs of numbers:
  - left, right --- extent in x axis
  - top, bottom --- extent in y axis
  - near, far --- extent in -z axis (because we look in -z direction)

- View volume:

  $\{(x,y,z) : left \le x \le right, top \le y \le bottom, near \le -z \le far\}$

Orthographic View Volume



Orthographic Projection Matrix

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Details in Chapter 7 of Shirley & Marschner

OpenGL Commands

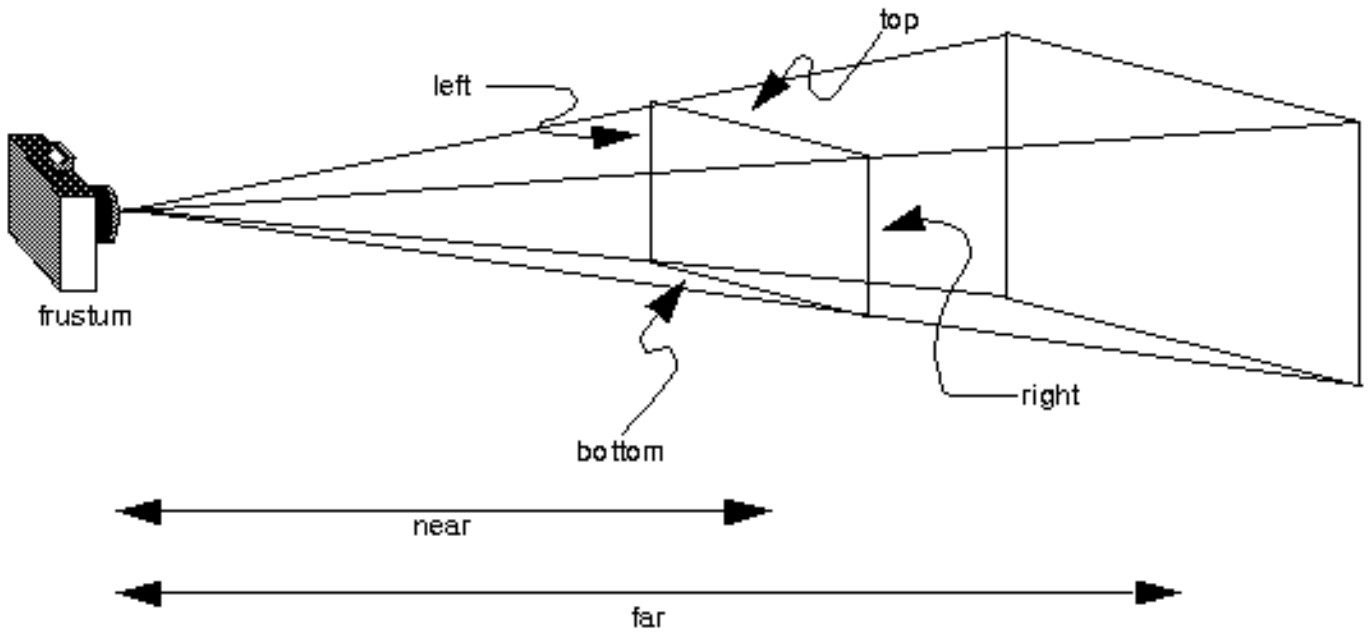
- GL2.glOrtho(left, right, bottom, top, near, far)
  - Set M = MO where O is the orthographic projection matrix.
- GLU.gluOrtho2D(left, right, bottom, top)
  - Just a glOrtho with near = 0 and far = 1
- Before using any of the above, use the following commands:

```
gl.glMatrixMode(GL2.GL_PROJECTION);
gl.glLoadIdentity();
```

to choose the right matrix mode, and clear the projection matrix.

Specifying Perspective Projection

- Specify six numbers like orthographic projection.
- But now, we're defining a frustum.
  - frustum = capped pyramid



Perspective Projection Matrix

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Details in Chapter 7 of Shirley & Marschner

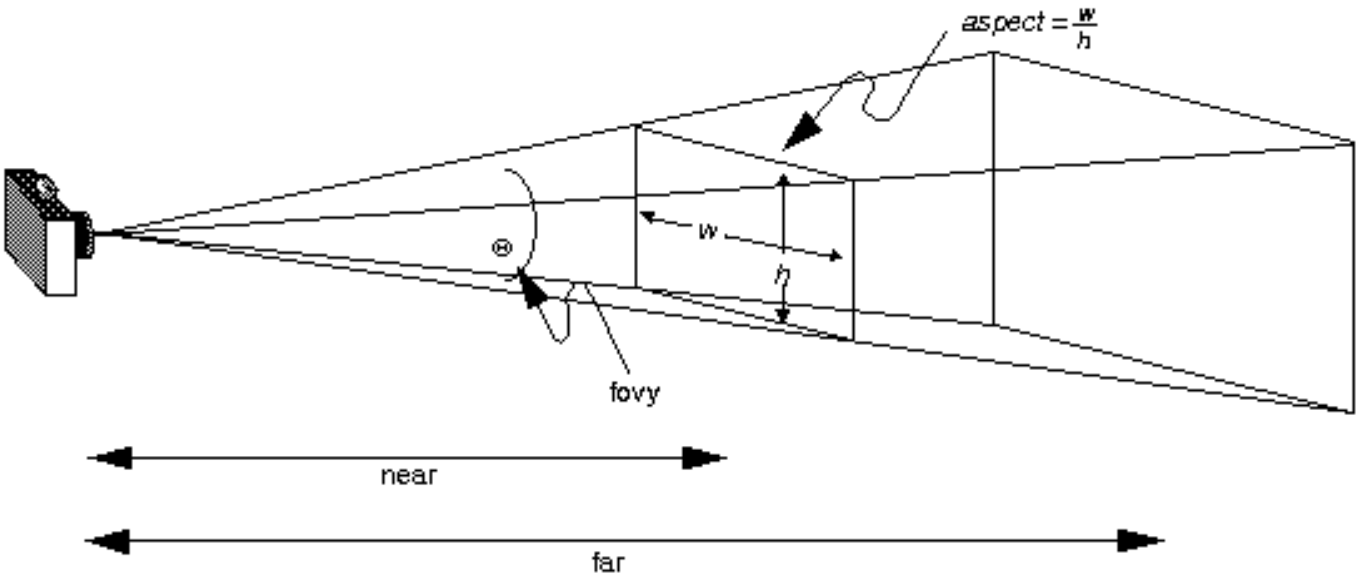
OpenGL Commands

- GL2.glFrustum(left, right, bottom, top, near, far)
  - Set M = MP where P is the perspective projection matrix.

- GLU.gluPerspective(fovy, aspect, near, far)
  - Call glFrustum with:
    - top = near * tan(fovy / 2)
    - bottom = -top
    - right = aspect * top
    - left = -right

gluPerspective

- fovy = “field of view Y”
  - Angle of corresponding to the height of the near cap.
- aspect = aspect ratio of the near cap.
- View volume:

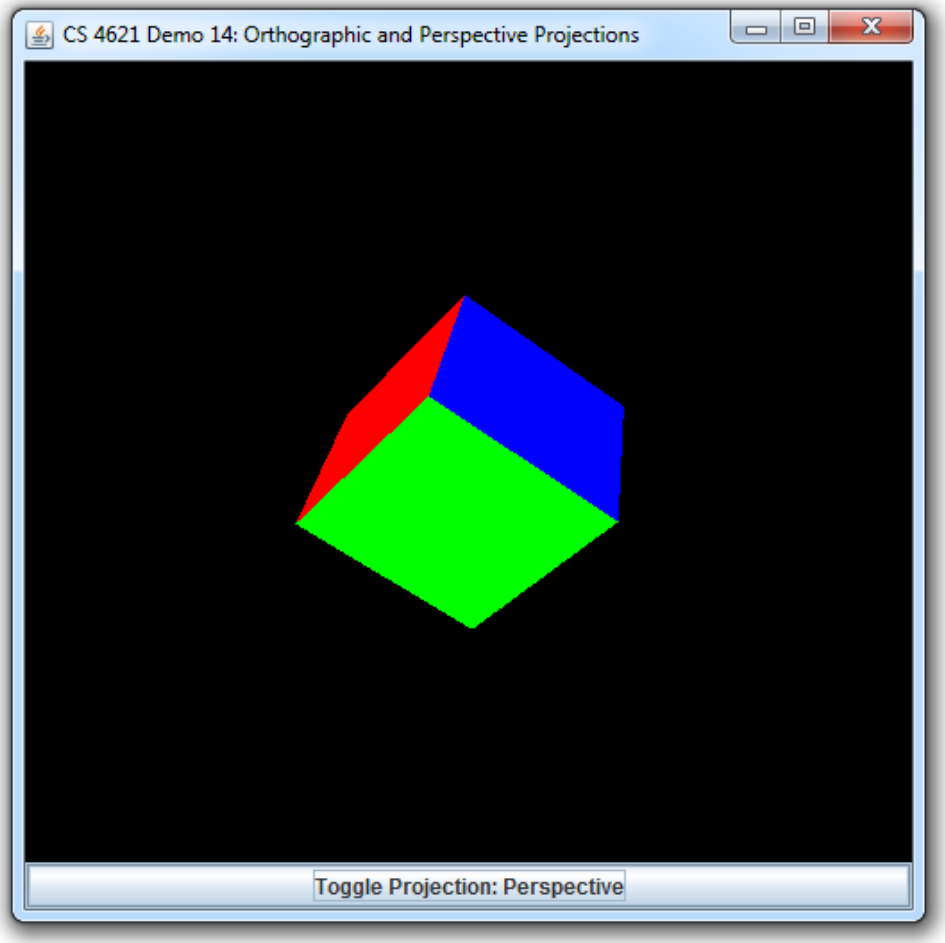


glFrustum VS gluPerspective

- glFrustum
  - Can create frustra not symmetric around x and y axes.
  - Harder to set up.
  - Useful in rendering rooms with windows.

- gluPerspective
  - Frustra are symmetric around x and y axes.
  - Easy to setup due to intuitive parameters.
  - Less flexible.
  - Good for viewing from the eye.

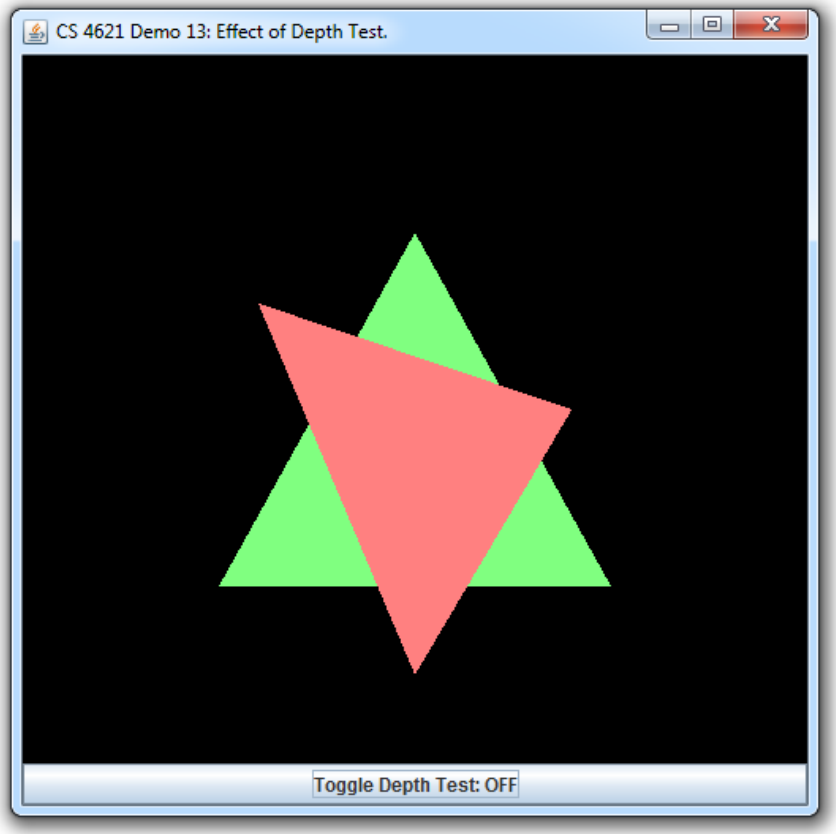
Demo 14


CS 4621 Demo 14: Orthographic and Perspective Projections
Toggle Projection: Perspective

DEPTH TEST

Visibility and Rendering Order

- So far:
  - What's drawn afterwards overwrites what's drawn before.

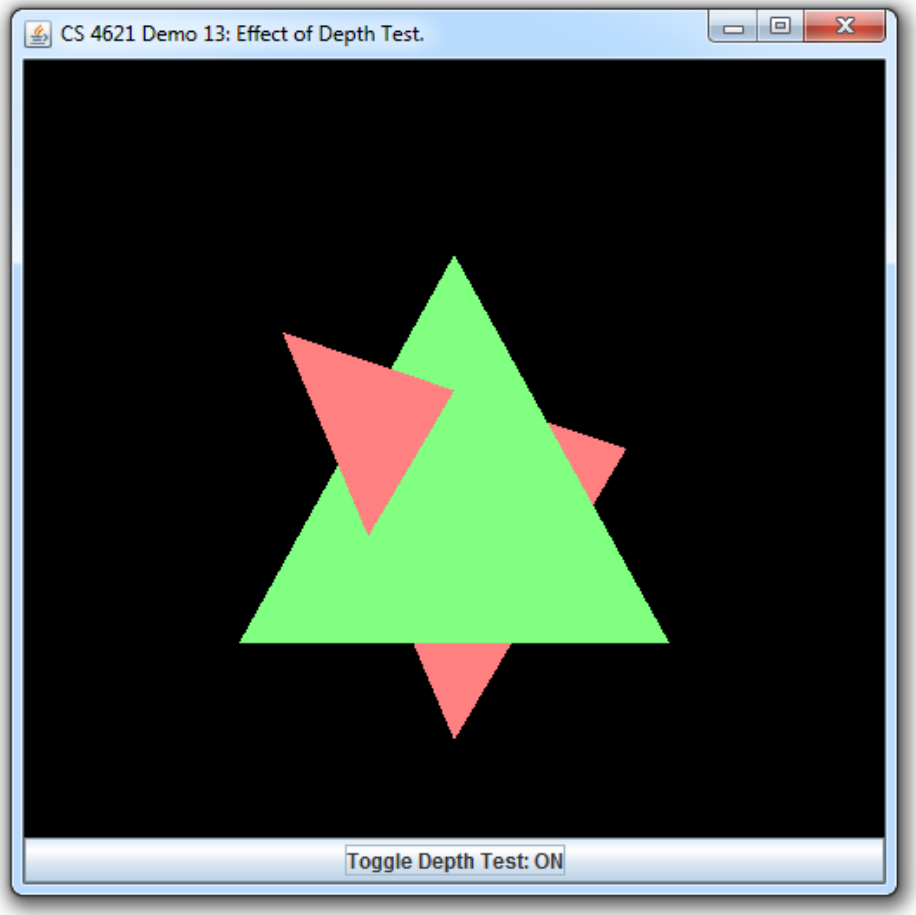


Actual Drawing Code

```
glColor3d(0.5, 1, 0.5);
glBegin(GL_TRIANGLES);
glVertex3d(0, 0.5, 0);
glVertex3d(-0.5, -0.5, 0);
glVertex3d(0.5, -0.5, 0);
glEnd();

glColor3d(1, 0.5, 0.5);
glBegin(GL_TRIANGLES);
glVertex3d(0, -0.75, 1);
glVertex3d(0.40, 0, 1);
glVertex3d(-0.40, 0.30, -1);
glEnd();
```

In 3D, should look like this…



Depth Test

- Functionality to simulate occlusion due to depth in 3D.
  - Nearer objects occlude farther objects.

- To turn on:
  - gl.glEnable(GL2.GL_DEPTH_TEST);
- To turn off:
  - gl.glDisable(GL2.GL_DEPTH_TEST);

- Algorithm: z-buffer (aka depth buffer)
  - Store depth value at each pixel.
  - Keep the fragment from object with the lowest z from viewer.

Depth Test and glClear

- Now, we have two buffers to worry about.
  - Color buffer
  - Depth buffer

- When calling glClear, must clear both buffers.

```
gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);
```

- Set the value to fill the depth buffer with glClearDepth.
  - Most of the time: gl.glClearDepth(1.0);
  - 1.0 is the maximum depth used by OpenGL.

ANIMATION

Repaint

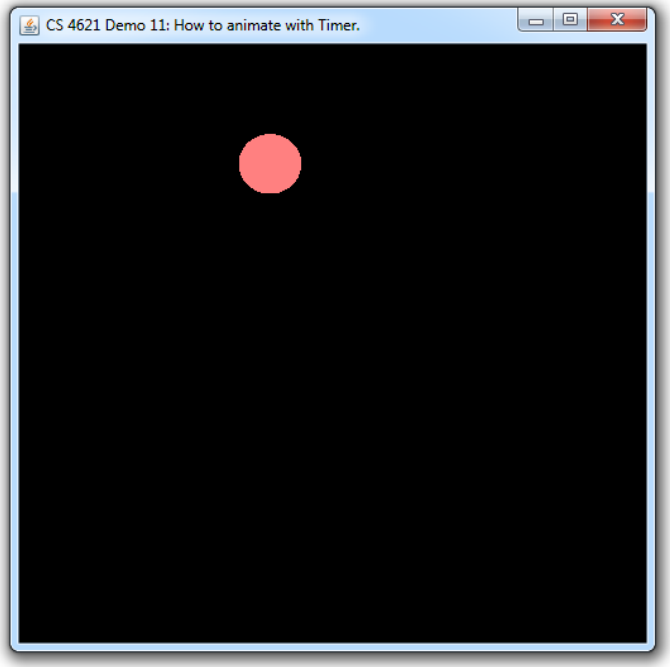
- Call GLCanvas.repaint() to have it draw stuffs again.

- Non-blocking
  - Just send a message to GLCanvas.
  - Method returns immediately
  - GLCanvas schedules a redraw as soon as possible.

- When?
  - Inside GLEventListener.resize(…).
  - Inside keyboard/mouse handler.

Creating Animation

- Two approaches.
  - Swing's Timer
  - JOGL's Animator



java.swing.Timer

- Have main class implement ActionListener.

- Create instance of Timer in class.
  - `timer = new Timer(33, this);`
  - 1st argument: delay between ticking in milliseconds.
  - 2nd argument: instance of ActionListener

- Call `timer.start();`
  - I call this in GLEventListener.init.
  - Don’t want timer to update the framebufer before it’s ready.

- Implement the actionPerformed method
  - If source of event = timer, then
    - Update states.
    - Call canvas.repaint();

JOGL's Animator

- Two variants:
  - Animator
    - Call GLCanvas.display repeatedly.
    - Short pause between calls.
  - FPSAnimator
    - Call GLCanvas.display periodically to achieve target frame rate.
    - Avoid using all CPU time.
    - We'll use this one.

JOGL's Animator

- Create
  - `animator = new FPSAnimator(30);`
  - Argument: target frames per second
- Call animator.add(canvas).
  - Register the canvas with the animator.
- Perform all state updates in GLEventListener.display().

Comparison

Swing's Timer

- Use event dispatch thread
  - No need to worry about concurrency.
- State update separated from display.
- More code.
- More flexible.

JOGL's Animator

- Create it's own thread
  - Possible concurrency issues.
- Must perform state update in display().
- Less code.
- Less flexible.