MORE OPENGL

Pramook Khungurn
CS 4620, Fall 2012
SETTING UP THE CAMERA
Recall: OpenGL Vertex Transformations

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

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

Eye Space

World Space

Eye point

at point

up
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:

```java
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);

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

gl.glTranslated(1, -1, 0);
gl.glRotate(30, 0, 0, 1);
/* draw stuffs */
```
SETTING UP PROJECTION
Projection Transform

- eye space $\rightarrow$ 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 $\rightarrow$ 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:
  \[ \{(x,y,z) : \text{left} \leq x \leq \text{right}, \text{top} \leq y \leq \text{bottom}, \text{near} \leq -z \leq \text{far}\} \]
Orthographic View Volume
Orthographic Projection Matrix

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

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.

- 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.
GLU

- OpenGL Utility Library
- C library usually ships with OpenGL
- Many useful functions:
  - Drawing of some curved surfaces.
  - Interpretation of OpenGL errors.
  - Simple camera setup.

- In JOGL,
  - Encapsulated by GLU class.
  - Create one when needed.
  - GLU glu = new GLU();
SETTING UP 2D COORDINATE SYSTEM
Default Viewing

- Eye at (0,0,1).
- Look in negative $z$ direction.
- Orthographic projection
  - No foreshortening.
  - No sense of depth.
  - 2D coordinate system if don’t care about $z$-axis.
Default Coordinate System

(-1,1)  (1,1)

(-1,-1)  (1,-1)
gluOrtho2D

• glu.gluOrtho2D(double left, double right, double bottom, double top)

• Just a glOrtho with near = 0 and far = 1

• For now, it sets the 2D coordinate system so that:
  • Bottom-left corner is (left, bottom)
  • Bottom-right corner (right, bottom)
  • Top-left corner is (left, top)
  • Top-right corner is (right, top)
**gluOrtho2D**

- Usage: Issue these three commands together.
  - `gl.glMatrixMode(GL2.GL_PROJECTION);`
  - `gl.glLoadIdentity();`
  - `glu.gluOrtho2D(left, right, bottom, top);`

- Will become clear afterwards why by next week.

- Only primitives drawn afterwards are affected.
  - Coordinate system is “defined” by projection matrix.
  - OpenGL remembers the matrix until it changes.
gluOrtho2D

- gluOrtho2D(-1,1,-1,1)
- gluOrtho2D(-2,2,-2,2)
- gluOrtho2D(-2,2,-1,1)
- gluOrtho2D(-1,1,-2,2)
When Resizing Window
But we probably want this…
What We Want

• When resizing window, aspect ratio of drawn pictures remain the same.
• Keep the same drawing code.
  • No change to display method.
• Can do so by changing the coordinate system.

• When to do this?
  • Each time the window size change.
• Implement reshape method.
### reshape

- public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height);

- \(x, y\)
  - Coordinate of top-left corner of GLCanvas in pixels.

- width, height
  - Size of GLCanvas in pixels.
Demo 4
Demo 4’s reshape

```java
@Override
default void reshape(GLAutoDrawable drawable, 
    int x, int y, int w, int h) {
    final GL2 gl = drawable.getGL().getGL2();
    final GLU glu = new GLU();

    if (w == 0) w = 1;
    if (h == 0) h = 1;

    double aspect = w * 1.0 / h;

    gl.glMatrixMode(GL2.GL_PROJECTION);
    gl.glLoadIdentity();
    if (w > h)
        glu.gluOrtho2D(-aspect, aspect, -1, 1);
    else
        glu.gluOrtho2D(-1, 1, -1/aspect, 1/aspect);
}
```
When Width > Height

(-aspect,1)   (-1,1)       (1,1)   (aspect,1)
(-aspect,-1)  (-1,-1)      (1,-1)  (aspect,-1)
When Width < Height

(-1,1/aspect) -> (1,1/aspect)

(-1,1) -> (1,1)

(-1,-1) -> (1,-1)

(-1,-1/aspect) -> (1,aspect)
Specifying Perspective Projection

- Specify six numbers like orthographic projection.
- But now, we’re defining a frustum.
  - frustum = capped pyramid
Perspective Projection Matrix

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

Details in Chapter 7 of Shirley & Marschner
DEPTH TEST
Visibility and Rendering Order

• So far:
  • What’s drawn afterwards overwrites what’s drawn before.
Actual Drawing Code

```cpp
setColor3d(0.5, 1, 0.5);
begin(GL_TRIANGLES);
vertex3d(0, 0.5, 0);
vertex3d(-0.5, -0.5, 0);
vertex3d(0.5, -0.5, 0);
end();

setColor3d(1, 0.5, 0.5);
begin(GL_TRIANGLES);
vertex3d(0, -0.75, 1);
vertex3d(0.40, 0, 1);
vertex3d(-0.40, 0.30, -1);
end();
```
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 framebuffer 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

<table>
<thead>
<tr>
<th>Swing’s Timer</th>
<th>JOGL’s Animator</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Use event dispatch thread</td>
<td>• Create it’s own thread</td>
</tr>
<tr>
<td>• No need to worry about concurrency.</td>
<td>• Possible concurrency issues.</td>
</tr>
<tr>
<td>• State update separated from display.</td>
<td>• Must perform state update in display().</td>
</tr>
<tr>
<td>• More code.</td>
<td>• Less code.</td>
</tr>
<tr>
<td>• More flexible.</td>
<td>• Less flexible.</td>
</tr>
</tbody>
</table>