News

• extra TA office hours in lab 005
  • Tue 2-5 (Kai)
  • Wed 2-5 (Garrett)
  • Thu 1-3 (Garrett), Thu 3-5 (Kai)
  • Fri 2-4 (Garrett)
• Tamara's usual office hours in lab
  • Fri 4-5
Reading for This and Next 2 Lectures

- FCG Chapter 7 Viewing
- FCG Section 6.3.1 Windowing Transforms
- RB rest of Chap Viewing
- RB rest of App Homogeneous Coords
Review: Display Lists

- precompile/cache block of OpenGL code for reuse
  - usually more efficient than **immediate mode**
    - exact optimizations depend on driver
  - good for multiple instances of same object
    - but cannot change contents, not parametrizable
  - good for static objects redrawn often
    - display lists persist across multiple frames
    - interactive graphics: objects redrawn every frame from new viewpoint from moving camera
  - can be nested hierarchically
- snowman example: 3x performance improvement, 36K polys
Review: Computing Normals

- normal
  - direction specifying orientation of polygon
    - $w=0$ means direction with homogeneous coords
    - vs. $w=1$ for points/vectors of object vertices
  - used for lighting
    - must be normalized to unit length
  - can compute if not supplied with object

$$N = (P_2 - P_1) \times (P_3 - P_1)$$
Review: Transforming Normals

- cannot transform normals using same matrix as points
  - nonuniform scaling would cause to be not perpendicular to desired plane!

\[
P \quad \quad \quad \quad P' = MP
\]
\[
N \quad \quad \quad \quad N' = QN
\]

given M, what should Q be?

\[
Q = (M^{-1})^T
\]

inverse transpose of the modelling transformation
Review: Rendering Pipeline

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping → Frame-buffer

Scan Conversion → Texturing → Depth Test → Blending
Review: Projective Rendering Pipeline

OCS - object/model coordinate system
WCS - world coordinate system
VCS - viewing/camera/eye coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device/display/screen coordinate system

O2W: modeling transformation
W2V: viewing transformation
V2C: projection transformation

C2N: perspective divide
N2D: viewport transformation

clipping
normalized
device
NDCS
device
DCS
Review: Viewing Transformation

- **OCS** (Object Coordinate System)
- **WCS** (World Coordinate System)
- **VCS** (Viewing Coordinate System)

Object transformations:
- **M_{mod}**: Modeling transformation

Viewing transformations:
- **M_{cam}**: Viewing transformation

OpenGL ModelView matrix
Review: Basic Viewing

• starting spot - OpenGL
  • camera at world origin
    • probably inside an object
  • y axis is up
  • looking down negative z axis
    • why? RHS with x horizontal, y vertical, z out of screen
• translate backward so scene is visible
  • move distance d = focal length

• where is camera in P1 template code?
  • 5 units back, looking down -z axis
Convenient Camera Motion

- rotate/translate.scale versus
  - eye point, gaze/lookat direction, up vector

- demo: Robins transformation, projection
OpenGL Viewing Transformation

\[
gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz)\]

- postmultiplies current matrix, so to be safe:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz)
// now ok to do model transformations
```

- demo: Nate Robins tutorial projection
Convenient Camera Motion

- rotate/translate/scale versus
  - eye point, gaze/lookat direction, up vector
From World to View Coordinates: W2V

- translate **eye** to origin
- rotate **view** vector (**lookat** – **eye**) to **w** axis
- rotate around **w** to bring **up** into **vw**-plane
Deriving W2V Transformation

- translate **eye** to origin

\[
T = \begin{bmatrix}
1 & 0 & 0 & e_x \\
0 & 1 & 0 & e_y \\
0 & 0 & 1 & e_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Deriving W2V Transformation

• rotate *view* vector (*lookat* – *eye*) to *w* axis
  
  • *w*: normalized opposite of *view/gaze* vector *g*

\[
w = -\hat{g} = -\frac{g}{\|g\|}
\]
Deriving W2V Transformation

• rotate around $w$ to bring up into $vw$-plane
  • $u$ should be perpendicular to $vw$-plane, thus perpendicular to $w$ and up vector $t$
  • $v$ should be perpendicular to $u$ and $w$

$$u = \frac{t \times w}{\|t \times w\|} \quad v = w \times u$$
Deriving W2V Transformation

- rotate from WCS \( xyz \) into \( uvw \) coordinate system with matrix that has columns \( u, v, w \)

\[
\begin{align*}
\mathbf{u} &= \frac{\mathbf{t} \times \mathbf{w}}{||\mathbf{t} \times \mathbf{w}||} \\
\mathbf{v} &= \mathbf{w} \times \mathbf{u} \\
\mathbf{w} &= -\mathbf{g} = -\frac{\mathbf{g}}{||\mathbf{g}||}
\end{align*}
\]

\[
\mathbf{R} = \begin{bmatrix}
\mathbf{u}_x & \mathbf{v}_x & \mathbf{w}_x & 0 \\
\mathbf{u}_y & \mathbf{v}_y & \mathbf{w}_y & 0 \\
\mathbf{u}_z & \mathbf{v}_z & \mathbf{w}_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
\mathbf{T} = \begin{bmatrix}
1 & 0 & 0 & e_x \\
0 & 1 & 0 & e_y \\
0 & 0 & 1 & e_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
\mathbf{M}_{W2V} = \mathbf{R} \mathbf{T}
\]

- reminder: rotate from \( uvw \) to \( xyz \) coord sys with matrix \( \mathbf{M} \) that has columns \( u,v,w \)
W2V vs. V2W

- $M_{W2V} = TR$

  $$T = \begin{bmatrix} 1 & 0 & 0 & e_x \\ 0 & 1 & 0 & e_y \\ 0 & 0 & 1 & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- we derived position of camera in world
  - invert for world with respect to camera
- $M_{V2W} = (M_{W2V})^{-1} = R^{-1} T^{-1}$

  $$R^{-1} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- inverse is transpose for orthonormal matrices
- inverse is negative for translations
W2V vs. V2W

- $M_{W2V} = TR$

- we derived position of camera in world
  - invert for world with respect to camera
- $M_{V2W} = (M_{W2V})^{-1} = R^{-1}T^{-1}$

$$
M_{view2world} =
\begin{bmatrix}
  u_x & u_y & u_z & 0 \\
  v_x & v_y & v_z & 0 \\
  w_x & w_y & w_z & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  1 & 0 & 0 & -e_x \\
  0 & 1 & 0 & -e_y \\
  0 & 0 & 1 & -e_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
  u_x & u_y & u_z & -e_x \\
  v_x & v_y & v_z & -e_y \\
  w_x & w_y & w_z & -e_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
$$
Moving the Camera or the World?

- two equivalent operations
  - move camera one way vs. move world other way
- example
  - initial OpenGL camera: at origin, looking along -z axis
  - create a unit square parallel to camera at z = -10
  - translate in z by 3 possible in two ways
    - camera moves to z = -3
      - Note OpenGL models viewing in left-hand coordinates
    - camera stays put, but world moves to -7
  - resulting image same either way
    - possible difference: are lights specified in world or view coordinates?
- third operation: scaling the world
  - smaller vs farther away
World vs. Camera Coordinates Example

\[ a = (1,1)_w \]
\[ b = (1,1)_{c_1} = (5,3)_w \]
\[ c = (1,1)_{c_2} = (1,3)_{c_1} = (5,5)_w \]
Projections I
Pinhole Camera

- ingredients
  - box, film, hole punch
- result
  - picture

www.kodak.com
www.pinhole.org
www.debevec.org/Pinhole
Pinhole Camera

- theoretical perfect pinhole
- light shining through tiny hole into dark space yields upside-down picture
Pinhole Camera

- non-zero sized hole
- blur: rays hit multiple points on film plane
Real Cameras

- pinhole camera has small aperture (lens opening)
  - minimize blur

- problem: hard to get enough light to expose the film

- solution: lens
  - permits larger apertures
  - permits changing distance to film plane without actually moving it
    - cost: limited depth of field where image is in focus

Graphics Cameras

- real pinhole camera: image inverted

- computer graphics camera: convenient equivalent
General Projection

- image plane need not be perpendicular to view plane
Perspective Projection

- our camera must model perspective
Perspective Projection

• our camera must model perspective
Projective Transformations

- planar geometric projections
  - planar: onto a plane
  - geometric: using straight lines
  - projections: 3D -> 2D
- aka projective mappings
- counterexamples?
Projective Transformations

- properties
  - lines mapped to lines and triangles to triangles
  - parallel lines do **NOT** remain parallel
    - e.g. rails vanishing at infinity
  - affine combinations are **NOT** preserved
    - e.g. center of a line does not map to center of projected line (perspective foreshortening)
Perspective Projection

- project all geometry
  - through common center of projection (eye point)
  - onto an image plane
Perspective Projection

How tall should this bunny be?

Center of projection (eye point)

Projection plane

How tall should this bunny be?
Basic Perspective Projection

similar triangles

\[ \frac{x'}{d} = \frac{x}{z} \rightarrow x' = \frac{x \cdot d}{z} \]

\[ \frac{y'}{d} = \frac{y}{z} \rightarrow y' = \frac{y \cdot d}{z} \]

but \[ z' = d \]

• nonuniform foreshortening
• not affine
Perspective Projection

• desired result for a point \([x, y, z, 1]^\top\) projected onto the view plane:

\[
x' = \frac{x}{d} = \frac{x}{z}, \quad y' = \frac{y}{d} = \frac{y}{z}
\]

\[
x' = \frac{x \cdot d}{z} = \frac{x}{z/d}, \quad y' = \frac{y \cdot d}{z} = \frac{y}{z/d}, \quad z' = d
\]

• what could a matrix look like to do this?
Simple Perspective Projection Matrix

\[
\begin{bmatrix}
x \\
\frac{y}{z/d} \\
\frac{y}{z/d} \\
\frac{y}{z/d} \\
d
\end{bmatrix}
\]
Simple Perspective Projection Matrix

\[
\begin{bmatrix}
  x/z/d \\
  y/z/d \\
  z/d \\
  d
\end{bmatrix}
\]

is homogenized version of

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  z/d
\end{bmatrix}
\]

where \( w = z/d \)
Simple Perspective Projection Matrix

\[
\begin{bmatrix}
\frac{x}{z/d} \\
\frac{y}{z/d} \\
\frac{z}{d} \\
\end{bmatrix}
\]

is homogenized version of

where w = z/d

\[
\begin{bmatrix}
x \\
y \\
z \\
z/d \\
\end{bmatrix}
= \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1 \\
\end{bmatrix}
\]
Perspective Projection

- expressible with 4x4 homogeneous matrix
  - use previously untouched bottom row
- perspective projection is irreversible
  - many 3D points can be mapped to same (x, y, d) on the projection plane
  - no way to retrieve the unique z values
Moving COP to Infinity

- as COP moves away, lines approach parallel
- when COP at infinity, **orthographic** view
Orthographic Camera Projection

- camera’s back plane parallel to lens
- infinite focal length
- no perspective convergence
- just throw away z values

\[
\begin{bmatrix}
    x_p \\
    y_p \\
    z_p \\
    1
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 & 0 \\
    0 & 0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]
Perspective to Orthographic

• transformation of space
  • center of projection moves to infinity
  • view volume transformed
    • from frustum (truncated pyramid) to parallelepiped (box)
View Volumes

- specifies field-of-view, used for clipping
- restricts domain of \( z \) stored for visibility test
Canonical View Volumes

- standardized viewing volume representation

perspective

orthographic
orthogonal
parallel

$$x \text{ or } y = \pm z$$
Why Canonical View Volumes?

• permits standardization
  • clipping
    • easier to determine if an arbitrary point is enclosed in volume with canonical view volume vs. clipping to six arbitrary planes
  • rendering
    • projection and rasterization algorithms can be reused
Normalized Device Coordinates

- convention
  - viewing frustum mapped to specific parallelepiped
    - Normalized Device Coordinates (NDC)
    - same as clipping coords
  - only objects inside the parallelepiped get rendered
- which parallelepiped?
  - depends on rendering system
Normalized Device Coordinates

left/right $x = +/- 1$, top/bottom $y = +/- 1$, near/far $z = +/- 1$

Camera coordinates

NDC

Frustum

$z = -n$

$z = -f$

$x = -1$

$z = 1$

$x = 1$

$z = -1$
Understanding Z

- z axis flip changes coord system handedness
- RHS before projection (eye/view coords)
- LHS after projection (clip, norm device coords)
Understanding Z

near, far always positive in OpenGL calls

```c
glOrtho(left,right,bot,top,near,far);
glFrustum(left,right,bot,top,near,far);
glPerspective(fovy,aspect,near,far);
```

perspective view volume

orthographic view volume

VCS
Understanding Z

• why near and far plane?
  • near plane:
    • avoid singularity (division by zero, or very small numbers)
  • far plane:
    • store depth in fixed-point representation (integer), thus have to have fixed range of values (0…1)
    • avoid/reduce numerical precision artifacts for distant objects
Orthographic Derivation

- scale, translate, reflect for new coord sys
Orthographic Derivation

- scale, translate, reflect for new coord sys

\[ y' = a \cdot y + b \]

\[ y = \text{top} \rightarrow y' = 1 \]
\[ y = \text{bot} \rightarrow y' = -1 \]
Orthographic Derivation

- scale, translate, reflect for new coord sys

\[ y' = a \cdot y + b \]

\[ y = top \rightarrow y' = 1 \quad 1 = a \cdot top + b \]

\[ y = bot \rightarrow y' = -1 \quad -1 = a \cdot bot + b \]

\[ b = 1 - a \cdot top, b = -1 - a \cdot bot \]

\[ 1 - a \cdot top = -1 - a \cdot bot \]

\[ 1 - (-1) = -a \cdot bot - (-a \cdot top) \]

\[ 2 = a(-bot + top) \]

\[ a = \frac{2}{top - bot} \]

\[ 1 = \frac{2}{top - bot} \cdot top + b \]

\[ b = 1 - \frac{2 \cdot top}{top - bot} \]

\[ b = \frac{(top - bot) - 2 \cdot top}{top - bot} \]

\[ b = \frac{-top - bot}{top - bot} \]
Orthographic Derivation

• scale, translate, reflect for new coord sys

\[ y' = a \cdot y + b \]

\[ y = \text{top} \rightarrow y' = 1 \]

\[ y = \text{bot} \rightarrow y' = -1 \]

\[ a = \frac{2}{\text{top} - \text{bot}} \]

\[ b = -\frac{\text{top} + \text{bot}}{\text{top} - \text{bot}} \]

same idea for right/left, far/near
Orthographic Derivation

• scale, translate, reflect for new coord sys

\[
P' = \begin{bmatrix}
\frac{2}{\text{right} - \text{left}} & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\
0 & \frac{2}{\text{top} - \text{bot}} & 0 & -\frac{\text{top} + \text{bot}}{\text{top} - \text{bot}} \\
0 & 0 & \frac{-2}{\text{far} - \text{near}} & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\
0 & 0 & 0 & 1
\end{bmatrix} P
\]
Orthographic Derivation

- **scale**, translate, reflect for new coord sys

\[
P' = \begin{bmatrix}
\frac{2}{\text{right} - \text{left}} & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\
0 & \frac{2}{\text{top} - \text{bot}} & 0 & -\frac{\text{top} + \text{bot}}{\text{top} - \text{bot}} \\
0 & 0 & \frac{-2}{\text{far} - \text{near}} & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\
0 & 0 & 0 & 1
\end{bmatrix} P
\]
Orthographic Derivation

- scale, translate, reflect for new coord sys

\[
P' = \begin{bmatrix}
\frac{2}{\text{right} - \text{left}} & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\
0 & \frac{2}{\text{top} - \text{bot}} & 0 & -\frac{\text{top} + \text{bot}}{\text{top} - \text{bot}} \\
0 & 0 & -\frac{2}{\text{far} - \text{near}} & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Orthographic Derivation

- scale, translate, reflect for new coord sys

\[
P' = \begin{bmatrix}
\frac{2}{\text{right} - \text{left}} & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\
0 & \frac{2}{\text{top} - \text{bot}} & 0 & -\frac{\text{top} + \text{bot}}{\text{top} - \text{bot}} \\
0 & 0 & \frac{-2}{\text{far} - \text{near}} & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\
0 & 0 & 0 & 1
\end{bmatrix} \cdot P
\]
Orthographic OpenGL

```c
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
glOrtho(left,right,bot,top,near,far);
```