Viewing/Projections

Week 5, Wed Oct 6
News

• assignment 1 posted
Viewing
(Review?)
Using Transformations

• three ways
  • modelling transforms
    • place objects within scene (shared world)
    • affine transformations
  • viewing transforms
    • place camera
    • rigid body transformations: rotate, translate
  • projection transforms
    • change type of camera
    • projective transformation
Rendering Pipeline

- Scene graph
  - Object geometry
- Modelling
  - Transforms
- Viewing
  - Transform
- Projection
  - Transform
Rendering Pipeline

- result
  - all vertices of scene in shared 3D world coordinate system
Rendering Pipeline

- result
  - scene vertices in 3D view (camera) coordinate system
Rendering Pipeline

- result
  - 2D screen coordinates of clipped vertices
Viewing and Projection

- need to get from 3D world to 2D image
- projection: geometric abstraction
  - what eyes or cameras do
- two pieces
  - viewing transform:
    - where is the camera, what is it pointing at?
  - perspective transform: 3D to 2D
    - flatten to image
Rendering Pipeline

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping

Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer
Rendering Pipeline

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping

→ Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer
OpenGL Transformation Storage

• modeling and viewing stored together
  • possible because no intervening operations
• perspective stored in separate matrix

• specify which matrix is target of operations
  • common practice: return to default modelview mode after doing projection operations
    
glMatrixMode(GL_MODELVIEW);
    glMatrixMode(GL_PROJECTION);
  

Coordinate Systems

• result of a transformation
• names
  • convenience
    • mouse: leg, head, tail
  • standard conventions in graphics pipeline
    • object/modelling
    • world
    • camera/viewing/eye
    • screen/window
    • raster/device
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

Diagram:
- OCS → O2W (modeling transformation)
- WCS → W2V (viewing transformation)
- VCS → V2C (projection transformation)
- CCS → C2N (perspective divide)
- NDCS → N2D (viewport transformation)
- DCS

Flow:
1. Object/Model Coordinate System (OCS) transforms to World Coordinate System (WCS) via O2W transformation.
2. WCS transforms to Viewing/Camera/Eye Coordinate System (VCS) via W2V transformation.
3. VCS transforms to Clipping Coordinate System (CCS) via V2C projection transformation.
4. CCS transforms to Normalized Device Coordinate System (NDCS) via perspective divide (C2N).
5. NDCS transforms to Device Coordinate System (DCS) via viewport transformation (N2D).
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

![Diagram of pinhole camera with light passing through tiny hole to create an upside-down image]

- perfect pinhole
- one ray of projection
- film plane
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
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

projection plane

center of projection (eye point)

how tall should this bunny be?
Basic Perspective Projection

similar triangles

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

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

but \[ z' = d \]

- nonuniform foreshortening
- not affine
Perspective Projection

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

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

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

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

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

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

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} \\
z/d \\

d
\end{bmatrix}
\]

is homogenized version of

where \( w = z/d \)

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

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

perspective view volume

orthographic view volume
Demo: Perspective and Ortho Volumes

- Nate Robins tutorial (projection)
Canonical View Volumes

- standardized viewing volume representation

perspective

orthographic
orthogonal
parallel

x or y = +/- z
back plane
front plane

1
-1
-z

front plane
back plane

x or y
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 $z = +/- 1$
Understanding Z

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

VCS
- x=left
- y=top
- z=-near

NDCS
- (1,1,1)
- (-1,-1,-1)
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
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

VCS
- x=left
- y=top
- z=-near

y=bottom

x=right

z=-far

NDCS
- (1,1,1)
- (-1,-1,-1)
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

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

\[ y = \text{top} \rightarrow y' = 1 \]
\[ 1 = a \cdot \text{top} + b \]

\[ y = \text{bot} \rightarrow y' = -1 \]
\[ -1 = a \cdot \text{bot} + b \]

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

\[ 1 - a \cdot \text{top} = -1 - a \cdot \text{bot} \]

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

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

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

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

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

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

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

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

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

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

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

• Brown applets: viewing techniques
  • parallel/orthographic camera transformations

• http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/viewing_techniques.html
Projections II
Asymmetric Frusta

- our formulation allows asymmetry
- why bother?
Asymmetric Frusta

- our formulation allows asymmetry
- why bother? binocular stereo
  - view vector not perpendicular to view plane

![Asymmetric Frusta Diagram](image)

- Right Eye
- Left Eye
- Right Eye
- Left Eye
Simpler Formulation

- left, right, bottom, top, near, far
  - nonintuitive
  - often overkill
- look through window center
  - symmetric frustum
- constraints
  - left = -right, bottom = -top
Field-of-View Formulation

- FOV in one direction + aspect ratio (w/h)
  - determines FOV in other direction
  - also set near, far (reasonably intuitive)
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

glFrustum(left,right,bot,top,near,far);
or

glPerspective(fovy,aspect,near,far);
Demo: Frustum vs. FOV

- Nate Robins tutorial (take 2):
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

**Transformation Stages:**
1. Modeling Transformation (O2W)
2. Viewing Transformation (W2V)
3. Projection Transformation (V2C)
4. Perspective Divide (C2N)
5. Viewport Transformation (N2D)

**Coordinate Systems:**
- OCS
- WCS
- VCS
- CCS
- NDCS
- DCS
Projection Normalization

- warp perspective view volume to orthogonal view volume
  - render all scenes with orthographic projection!
  - aka perspective warp
Perspective Normalization

- perspective viewing frustum transformed to cube
- orthographic rendering of cube produces same image as perspective rendering of original
Predistortion
Demos

- Tuebingen applets from Frank Hanisch
  - http://www.gris.uni-tuebingen.de/projects/grdev/doc/html/etc/AppletIndex.html#Transformationen
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
Separate Warp From Homogenization

- warp requires only standard matrix multiply
  - distort such that orthographic projection of distorted objects is desired persp projection
    - \( w \) is changed
  - clip after warp, before divide
  - division by \( w \): homogenization