Perspective Projection (cont.)

Wolfgang Heidrich

Course News

Assignment 1
- Due February 2

Homework 2
- Exercise problems for perspective
- Discussed in labs next week
- Solutions online (as prep for quiz)

Quiz 1
- Next Wednesday (Jan 26)
Course News (cont.)

Reading list
- Previously published chapters numbers were from an old book version...

Reading for Quiz (new book version):
- Math prereq: Chapter 2.1-2.4, 4
- Intro: Chapter 1
- Affine transformations: Ch. 6 (was: Ch. 5, old book)
- Perspective: Ch 7 (was: Ch. 6, old book)
  - Also reading for this week...

The Rendering Pipeline

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

Geometry Processing

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

Rasterization → Fragment Processing
**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 of projective rendering pipeline](image)

**Perspective Transformation**

*In computer graphics:*

- Image plane is conceptually *in front* of the center of projection

- Perspective transformations belong to a class of operations that are called *projective transformations*

- Linear and affine transformations also belong to this class

- *All* projective transformations can be expressed as $4 \times 4$ matrix operations

![Diagram of perspective transformation](image)
**Perspective Projection**

**Synopsis:**
- Project all geometry through a common **center of projection** (eye point) onto an **image plane**

![Diagram](image1)

**Example:**
- Assume image plane at \( z = -1 \)
- A point \([x, y, z, l]^T\) projects to
  \([-x/z, -y/z, -z/l, l]^T = [x, y, z, -z]^T\)

![Diagram](image2)
Perspective Projection

Analysis:
- This is a special case of a general family of transformations called *projective transformations*.
- These can be expressed as 4x4 homogeneous matrices!
  - *E.g. in the example:*

\[
T \begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix} = \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & -1 & 0
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  z \\
  -z
\end{bmatrix} = \begin{bmatrix}
  -x/z \\
  -y/z \\
  -1 \\
  1
\end{bmatrix}
\]

Projective Transformation

Note:
- This version of the perspective transformation removes all information about the original object depth.
  - *The matrix is singular, so the information is irrecoverably lost.*
- Later it will be important to have information about the original object depth for visibility computations.
  - *We can achieve this by modifying the third row of the matrix, as we’ll see later.*
Projective Transformations

Transformation of space:
- Center of projection moves to infinity
- Viewing frustum is transformed into a parallelepiped

Demos

Tuebingen applets from Frank Hanisch
- (this is the English version)
Projective Transformations

**Convention:**
- Viewing frustum is mapped to a specific parallelepiped
  - Normalized Device Coordinates (NDC)
- Only objects inside the parallelepiped get rendered
- Which parallelepiped is used depends on the rendering system

**OpenGL:**
- Left and right image boundary are mapped to $x=-1$ and $x=+1$
- Top and bottom are mapped to $y=-1$ and $y=+1$
- Near and far plane are mapped to -1 and 1
Projective Transformations

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

Determining the matrix representation

- Need to observe 5 points in general position, e.g.
  - \([\text{left}, 0, 0, 1]^T \rightarrow [1, 0, 0, 1]^T\)
  - \([0, \text{top}, 0, 1]^T \rightarrow [0, 1, 0, 1]^T\)
  - \([0, 0, -f, 1]^T \rightarrow [0, 0, 1, 1]^T\)
  - \([0, 0, -n, 1]^T \rightarrow [0, 0, 0, 1]^T\)
  - \([\text{left} * f/n, \text{top} * f/n, -f, 1]^T \rightarrow [1, 1, 1, 1]^T\)
- Solve resulting equation system to obtain matrix
\[
\begin{bmatrix}
E & 0 & A & 0 \\
0 & F & B & 0 \\
0 & 0 & C & D \\
0 & 0 & -1 & 0
\end{bmatrix}
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix}
= \begin{bmatrix}
x = \text{left} \rightarrow x'/w' = 1 \\
x = \text{right} \rightarrow x'/w' = -1 \\
y = \text{top} \rightarrow y'/w' = 1 \\
y = \text{bottom} \rightarrow y'/w' = -1 \\
z = \text{near} \rightarrow z'/w' = 1 \\
z = \text{far} \rightarrow z'/w' = -1
\end{bmatrix}
\]

\[
y' = Fy + Bz, \quad y' = \frac{Fy + Bz}{w}, \quad 1 = \frac{Fy + Bz}{w}, \quad 1 = \frac{Fy + Bz}{-z}, \\
1 = F \frac{y}{-z} + B \frac{z}{-z}, \quad 1 = F \frac{y}{-z} - B, \quad 1 = F \frac{\text{top}}{-(-\text{near})} - B, \\
1 = F \frac{\text{top}}{\text{near}} - B
\]

Similarly for other 5 planes.

6 planes, 6 unknowns

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

\[
\begin{bmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{t-b}{t-b} & 0 \\
0 & 0 & -\frac{(f+n)}{f-n} & -\frac{2fn}{f-n} \\
0 & 0 & \frac{f-n}{f-n} & -1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & -5/3 & -8/3 \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

view volume
left = -1, right = 1
bot = -1, top = 1
near = 1, far = 4

Projective Transformations

Asymmetric Viewing Frusta
Sheared Perspective

Architectural Photography
Aside: Shift/Tilt photography

http://www.tiltshiftphotography.net/examples.php

Projective Transformations

Alternative specification of symmetric frusta

- Field-of-view (fov) $\alpha$
- Fov/2
- Field-of-view in y-direction (fovy) + aspect ratio

Frustum
Perspective Matrices in OpenGL

**Perspective Matrices:**
- `glFrustum( left, right, bottom, top, near, far )`
  - Specifies perspective transform (near, far are always positive)

**Convenience Function:**
- `gluPerspective( fovy, aspect, near, far )`
  - Another way to do perspective

Projective Transformations

**Properties:**
- All transformations that can be expressed as homogeneous 4x4 matrices (in 3D)
- 16 matrix entries, but multiples of the same matrix all describe the same transformation
  - 15 degrees of freedom
  - The mapping of 5 points uniquely determines the transformation
Projective Transformations

Properties
- Lines are 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)*

Orthographic Camera Projection

- Camera’s back plane parallel to lens
- Infinite focal length
- No perspective convergence
- Just throw away z values

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

- Planar projections
- Perspective: 1, 2, 3-point parallel
- Oblique
- Cabinet
- Cavalier
- Orthographic

Axonometric:
- Isometric
- Dimetric
- Trimetric

Perspective Projections classified by vanishing points

- One-point perspective
- Two-point perspective
- Three-point perspective

Wolfgang Heidrich

http://ceprofs.tamu.edu/tkramer/ENGR%20111/5.1/20
Axonometric Projections

- projectors perpendicular to image plane

3 Equal axes  2 Equal axes  0 Equal axes
3 Equal angles  2 Equal angles  0 Equal angles

A. ISOMETRIC  B. DIMETRIC  C. TRIMETRIC

View Volumes

- specifies field-of-view, used for clipping
- restricts domain of $z$ stored for visibility test
View Volume

**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

Projective Rendering Pipeline

- Object (OCS) to World (WCS) transformation
- World (WCS) to Viewing (VCS) transformation
- Viewing (VCS) to Normalized Device (NDCS) transformation
- Normalized Device (NDCS) to Device (DCS) transformation

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
Window-To-Viewport Transformation

**Generate pixel coordinates**
- Map $x$, $y$ from range $-1...1$ (*normalized device coordinates*) to pixel coordinates on the screen
- Map $z$ from $-1...1$ to $0...1$ (used later for visibility)
- Involves 2D scaling and translation

![Diagram showing the transformation from window to viewport](image)

**Coming Up:**

**Monday:**
- Transformations of planes and normals

**Wednesday**
- Quiz...

**Friday**
- Lighting/shading