Project Taxonomy recap

- from 3D to 2D

- perspective
- parallel
- oblique
- orthographic
- cabinet
- cavalier
- top, front, side
- axonometric: isometric, dimetric, trimetric

Projection recap

Perspective Projection

- project all geometry through a common center of projection (eye point) onto an image plane
Projective Transformations
- 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

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

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)

Understanding Z
- near, far always positive in OpenGL calls
  - \( \text{glOrtho}(\text{left}, \text{right}, \text{bot}, \text{top}, \text{near}, \text{far}); \)
  - \( \text{glFrustum}(\text{left}, \text{right}, \text{bot}, \text{top}, \text{near}, \text{far}); \)
  - \( \text{glPerspective}(\text{fovy}, \text{aspect}, \text{near}, \text{far}); \)
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

Asymmetric Frusta

- our formulation allows asymmetry
  - why bother?

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)

Projection Normalization

- distort such that orthographic projection of distorted objects is desired persp projection
  - convenient coord sys: clipping, hidden surfaces
Basic Perspective Projection
• can express as homogeneous 4x4 matrix!
\[
\begin{bmatrix}
x \\
y \\
z \\
z/d \\
\end{bmatrix} = 
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & d \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
d \\
\end{bmatrix}
\]

• solving resulting equation system to obtain matrix
• need to observe 5 points in general position, e.g.

Projective Transformations
• can express as homogeneous 4x4 matrices!
• 16 matrix entries
  – multiples of same matrix all describe same transformation
  – 15 degrees of freedom
  – mapping of 5 points uniquely determines transformation

Perspective Projection
• specific example
  – assume image plane at z = -1
  – a point \([x,y,z,1]^T\) projects to \([-\sqrt{w},-y/z,-z/z,1]^T = [x,y,z,z]^T\)

Projective Transformations
• determining the matrix representation
  – need to observe 5 points in general position, e.g.
    • \([\text{left}0.0.1]^T \rightarrow [1.0.0.1]^T\)
    • \([0.0.\text{top}.1]^T \rightarrow [0.1.0.1]^T\)
    • \([0.0,-1.1^T] \rightarrow [0.0.1.1]^T\)
    • \([\text{left}0.\text{top}0.\text{left}-1.1]^T\)
  – solve resulting equation system to obtain matrix

Perspective Projection
• distorted such that orthographic projection of distorted objects is desired persp projection
  – separate division from standard matrix multiplies
  – division: normalization

Projection Normalization

Week 4, Mon 22 Oct 03 © Tamara Munzner 23
Orthographic Derivation

• scale, translate, reflect for new coord sys

\[
\begin{align*}
VCS & : x=left, y=top, z=-near \\
NDCS & : x=left, y=top, z=-far
\end{align*}
\]

Orthographic OpenGL

\[
\begin{align*}
glMatrixMode(GL_PROJECTION); \\
glLoadIdentity(); \\
glOrtho(left, right, bot, top, near, far);
\end{align*}
\]

Perspective Derivation

earlier:

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

complete: shear, scale, projection-normalization

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
b'
\end{bmatrix} = \begin{bmatrix}
E & 0 & A & 0 \\
0 & F & B & 0 \\
0 & 0 & C & D \\
0 & 0 & -1 & 0
\end{bmatrix} \begin{bmatrix}
x \\
y \\
z \\
b'
\end{bmatrix}
\]
Perspective Derivation

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
\end{bmatrix} =
\begin{bmatrix}
  E & 0 & A & 0 \\
  0 & F & B & 0 \\
  0 & 0 & C & D \\
  0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  w
\end{bmatrix}
\]

\[
x' = Ex + Az \\
y' = Fy + Bz \\
z' = Cz + D
\]

\[
w' = -z
\]

\[
s = \text{left} \rightarrow x'/w' = 1 \\
s = \text{right} \rightarrow x'/w' = -1 \\
s = \text{top} \rightarrow y'/w' = 1 \\
s = \text{bottom} \rightarrow y'/w' = -1 \\
s = \text{near} \rightarrow z'/w' = 1 \\
s = \text{far} \rightarrow z'/w' = -1
\]

\[
y' = Fs + Br \\
x' = Fs + Br
\]

\[
1 = F \frac{y + B - z}{-z} = F \frac{y - z}{-z} = F \frac{y}{-z} = F \frac{y}{-z}
\]

\[
1 = F \frac{top - near}{near}
\]

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

Perspective OpenGL

```c
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(left,right,bot,top,near,far);
or
glPerspective(fovy,aspect,near,far);
```

Perspective Example

- tracks in VCS:
  - left x=-1, y=-1
  - right x=1, y=-1

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

```
tracks in VCS:
left  x=-1, y=-1
right x=1, y=-1

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

```
1 -1 -5/z_{CS} / 1/3 -8/3 -5/3 -8/3 z_{CS}
-1 -1

x_{NDCS} = -1/z_{CS} \\
y_{NDCS} = 1/z_{CS} \\
z_{NDCS} = \frac{5}{3} - \frac{8}{3} z_{CS}
```
Viewport Transformation

- generate pixel coordinates
  - map \( x, y \) from range \(-1...1\) \((\text{normalized device coordinates})\) to pixel coordinates on the display
  - involves 2D scaling and translation

\[
\begin{align*}
    x_{\text{DCS}} &= \frac{x_{\text{NDCS}}}{2} + 1 \\
y_{\text{DCS}} &= \frac{y_{\text{NDCS}}}{2} + 1 \\
z_{\text{DCS}} &= \frac{z_{\text{NDCS}}}{2} + 1
\end{align*}
\]

OpenGL
\[
glViewport(x, y, a, b);
\]
default:
\[
glViewport(0, 0, w, h);
\]

Viewport Transformation

Projective Rendering Pipeline

- \( gl\text{Vertex3f}(x,y,z) \)
- \( gl\text{Translatef}(x,y,z) \)
- \( gl\text{Rotatef}(\theta, x, y, z) \)
- \( .....
\]

OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

- \( gl\text{Frustum}(..) \)
- \( glu\text{LookAt}(..) \)
- \( .....
\]