Projection Matrices

Ed Angel
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts
University of New Mexico
Objectives

- Derive the projection matrices used for standard OpenGL projections
- Introduce oblique projections
- Introduce projection normalization
Normalization

• Rather than derive a different projection matrix for each type of projection, we can convert all projections to orthogonal projections with the default view volume.

• This strategy allows us to use standard transformations in the pipeline and makes for efficient clipping.
Pipeline View

modelview transformation → projection transformation → perspective division

4D → 3D

nonsingular

clipping → Hidden surface removal → projection

3D → 2D

against default cube
Notes

• We stay in four-dimensional homogeneous coordinates through both the modelview and projection transformations
  - Both these transformations are nonsingular
  - Default to identity matrices (orthogonal view)

• Normalization lets us clip against simple cube regardless of type of projection

• Delay final projection until end
  - Important for hidden-surface removal to retain depth information as long as possible
Orthogonal Normalization

\[ glOrtho(left, right, bottom, top, near, far) \]

normalization ⇒ find transformation to convert specified clipping volume to default

\((right, top, -far)\)

\((left, bottom, -near)\)

\((-1, -1, 1)\)

\((1, 1, -1)\)
Orthogonal Matrix

• Two steps
  - Move center to origin
    \[ T(-\frac{\text{left+right}}{2}, -\frac{\text{bottom+top}}{2}, \frac{\text{near+far}}{2}) \]
  - Scale to have sides of length 2
    \[ S(\frac{2}{\text{left-right}}, \frac{2}{\text{top-bottom}}, \frac{2}{\text{near-far}}) \]

\[ P = ST = \begin{bmatrix}
  \frac{2}{\text{right-left}} & 0 & 0 & -\frac{\text{right-left}}{\text{right-left}} \\
  0 & \frac{2}{\text{top-bottom}} & 0 & -\frac{\text{top-bottom}}{\text{top-bottom}} \\
  0 & 0 & \frac{2}{\text{near-far}} & -\frac{\text{far+near}}{\text{far+near}} \\
  0 & 0 & 0 & \frac{1}{\text{far-near}}
\end{bmatrix} \]
Final Projection

• Set $z = 0$

• Equivalent to the homogeneous coordinate transformation

\[
M_{\text{orth}} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

• Hence, general orthogonal projection in 4D is

\[P = M_{\text{orth}}ST\]
Oblique Projections

• The OpenGL projection functions cannot produce general parallel projections such as

• However if we look at the example of the cube it appears that the cube has been sheared

• Oblique Projection = Shear + Orthogonal Projection
General Shear

- Top view
- Side view

Angel: Interactive Computer Graphics 4E © Addison-Wesley 2005
Shear Matrix

$xy$ shear ($z$ values unchanged)

$$H(\theta, \phi) = \begin{bmatrix}
1 & 0 & -\cot \theta & 0 \\
0 & 1 & -\cot \phi & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}$$

Projection matrix

$$P = M_{\text{orth}} H(\theta, \phi)$$

General case:

$$P = M_{\text{orth}} STH(\theta, \phi)$$
Equivalency
Effect on Clipping

- The projection matrix \( \mathbf{P} = \mathbf{STH} \) transforms the original clipping volume to the default clipping volume.

- **Top View**
  - Object
  - Near plane: \( z = -1 \)
  - Far plane: \( z = 1 \)
  - DOP: \( x = 1 \)
  - Distorted object (projects correctly)
Simple Perspective

Consider a simple perspective with the COP at the origin, the near clipping plane at $z = -1$, and a 90 degree field of view determined by the planes $x = \pm z$, $y = \pm z$. 

\[
\begin{align*}
(1, 1, -1) \\
(-1, -1, -1)
\end{align*}
\]
Simple projection matrix in homogeneous coordinates

\[
M = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

Note that this matrix is independent of the far clipping plane
Generalization

\[
N = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \alpha & \beta \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

after perspective division, the point \((x, y, z, 1)\) goes to

\[
x'' = x/z \\
y'' = y/z \\
Z'' = -(\alpha + \beta/z)
\]

which projects orthogonally to the desired point regardless of \(\alpha\) and \(\beta\)
Picking $\alpha$ and $\beta$

If we pick

$$\alpha = \frac{\text{near} + \text{far}}{\text{far} - \text{near}}$$

$$\beta = \frac{2\text{near} \times \text{far}}{\text{near} - \text{far}}$$

the near plane is mapped to $z = -1$
the far plane is mapped to $z = 1$
and the sides are mapped to $x = \pm 1, y = \pm 1$

Hence the new clipping volume is the default clipping volume
Normalization Transformation

original clipping volume

z = -x

COP

original object

z = -near

z = x

z = -far

distorted object projects correctly

x = -1

z = 1

x = 1

new clipping volume

z = -1

Angel: Interactive Computer Graphics 4E © Addison-Wesley 2005
Normalization and Hidden-Surface Removal

• Although our selection of the form of the perspective matrices may appear somewhat arbitrary, it was chosen so that if \( z_1 > z_2 \) in the original clipping volume then the for the transformed points \( z_1' > z_2' \)

• Thus hidden surface removal works if we first apply the normalization transformation

• However, the formula \( z'' = -(\alpha + \beta/z) \) implies that the distances are distorted by the normalization which can cause numerical problems especially if the near distance is small
• `glFrustum` allows for an unsymmetric viewing frustum (although `gluPerspective` does not)
The normalization in \texttt{glFrustum} requires an initial shear to form a right viewing pyramid, followed by a scaling to get the normalized perspective volume. Finally, the perspective matrix results in needing only a final orthogonal transformation:

\[ P = \text{NSH} \]

our previously defined perspective matrix \hspace{1cm} shear and scale
Why do we do it this way?

- Normalization allows for a single pipeline for both perspective and orthogonal viewing.
- We stay in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed for hidden-surface removal and shading.
- We simplify clipping.