**Projection Matrices**

**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**

![Pipeline View Diagram](image)

**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

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

normalization \( \Rightarrow \) find transformation to convert specified clipping volume to default

\( \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \)

Orthogonal Matrix

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

\[ ST = S(s_x, s_y, s_z) \cdot T(d_x, d_y, d_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \end{bmatrix} \]

Final Projection

• Set \( z = 0 \)
  • Equivalent to the homogeneous coordinate transformation
    \[ M_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \]
  • Hence, general orthogonal projection in 4D is
    \[ P = M_{\text{orth}} \cdot ST \]
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

Shear Matrix

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

Projection matrix

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

General case:

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

- The projection matrix $P = STH$ transforms the original clipping volume to the default clipping volume.

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 $x = \pm z, y = \pm z$.

Perspective Matrices

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

$$x'' = -x/z, \quad y'' = -y/z, \quad z'' = -(\alpha + \beta/z)$$

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

If we pick

$$\alpha = \frac{\text{near} + \text{far}}{\text{far} - \text{near}},$$
$$\beta = \frac{2 \ast \text{near} \ast \text{far}}{\text{near} - \text{far}}$$

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

Hence the new clipping volume is the default clipping volume.

Normalization Transformation

$$z = -x$$
$$z = \text{far}$$
$$z = -\text{near}$$

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.

OpenGL Perspective

- `glFrustum` allows for an unsymmetric viewing frustum (although `gluPerspective` does not)
OpenGL Perspective Matrix

- The normalization in `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
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