Projection Matrices

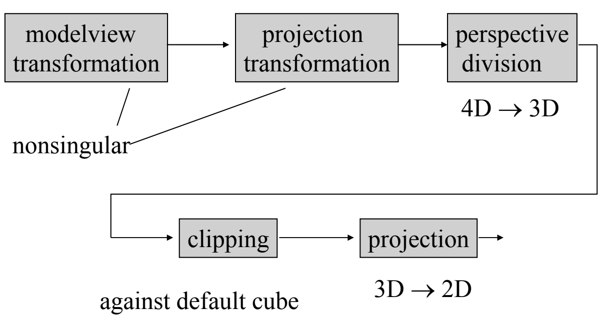
Objectives

- 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



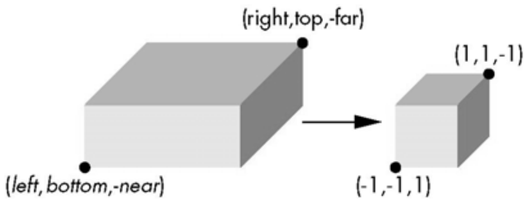
Notes

- 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



Orthogonal Matrix

- Two steps
  - Move center to origin

    T(-(left+right)/2, -(bottom+top)/2,(near+far)/2))
  - Scale to have sides of length 2

    S(2/(left-right),2/(top-bottom),2/(near-far))

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

Orthogonal Matrix

$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} s_x & 0 & 0 & s_x d_x \\ 0 & s_y & 0 & s_y d_y \\ 0 & 0 & s_z & s_z d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{left+right}{left-right} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{near-far} & \frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Final Projection

- Set $z=0$
- Equivalent to the homogeneous coordinate transformation

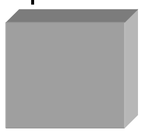
$$\mathbf{M}_{orth} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Hence, general orthogonal projection in 4D is

$\mathbf{P} = \mathbf{M}_{orth}\mathbf{ST}$

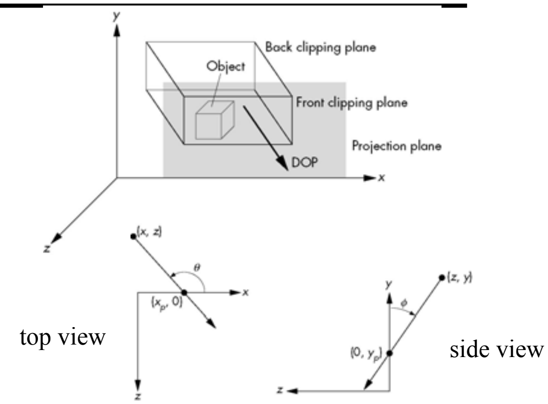
Oblique Projections

- 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



Shear Matrix

xy shear (*z* values unchanged)

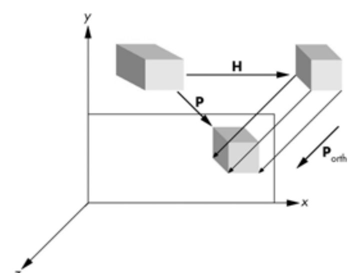
$$\mathbf{H}(\theta,\phi) = \begin{bmatrix} 1 & 0 & -\cot\theta & 0 \\ 0 & 1 & -\cot\varphi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection matrix

$$\mathbf{P} = \mathbf{M}_{\text{orth}}\, \mathbf{H}(\theta,\phi)$$

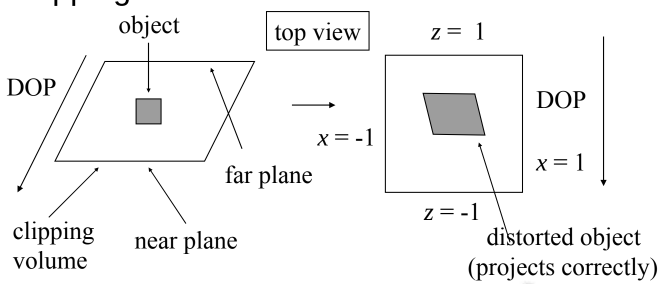
General case: $\mathbf{P} = \mathbf{M}_{\text{orth}}\, \mathbf{STH}(\theta,\phi)$

Equivalency



Effect on Clipping

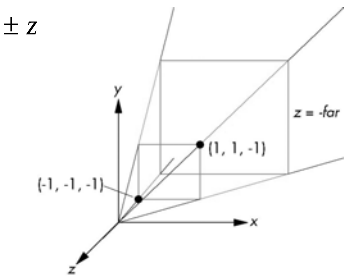
- The projection matrix $\mathbf{P} = \mathbf{STH}$ transforms the original clipping volume to the default clipping volume



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

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Note that this matrix is independent of the far clipping plane

Generalization

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

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 α and β

Picking α and β

If we pick

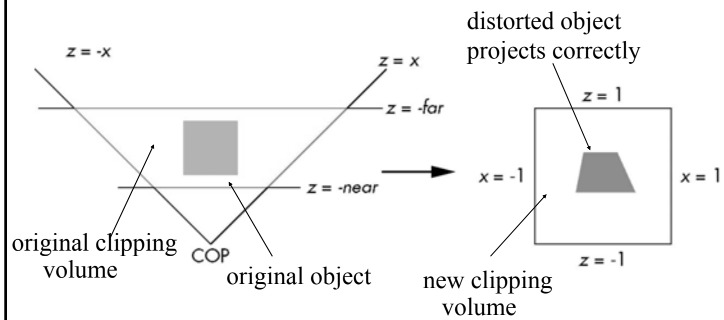
$$\alpha = \frac{\text{near} + \text{far}}{\text{far} - \text{near}}$$

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

the near plane z= near is mapped to $z = -1$
the far plane z = 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

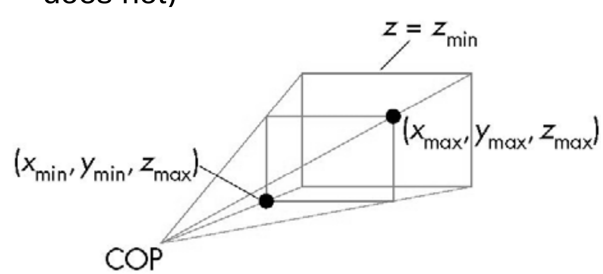


Normalization and Hidden-Surface Removal

- 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

$$\mathbf{P} = \mathbf{NSH}$$

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