Perspective Transformations
Viewing Transformations
Viewing transformations

- Move objects from their 3D locations to their positions in a 2D view
Decomposition of viewing transforms

Viewing transforms depend on: camera position and orientation, type of projection, field of view, image resolution.
Viewport transform

\[(x, y, z) \rightarrow (x', y', z')\]

\[(x, y, z) \in [-1, 1]^3\]

\[x' \in [-0.5, n_x - 0.5]\]

\[y' \in [-0.5, n_y - 0.5]\]
Viewport transform

- Camera transform
- Projection transform
- Viewport transform

\[ M_{vp} \]

<whiteboard>
Orthographic Projection Transform

- Camera transform
- Projection transform
- Viewport transform

\[ M_{orth} \]

\[(r, t, h) \]

\[(l, b, f) \] → \[(-1, -1, -1) \] → \[(1, 1, 1) \]
Camera Transform

Camera transform

Projection transform

Viewport transform

world space

camera space

canonical view volume

screen space
Camera Transform

How do we specify the camera configuration?
Camera Transform

How do we specify the camera configuration?
Camera Transform

How do we specify the camera configuration?

gaze direction
Camera Transform

*How do we specify the camera configuration?*
Camera Transform

How do we specify the camera configuration?
Camera Transform

\[ w = -\frac{g}{\|g\|} \]
\[ u = \frac{t \times w}{\|t \times w\|} \]
\[ v = w \times u \]

\[ M_{\text{cam}} \]
Perspective Viewing
rigid

affine

projective
Projective Transformations

\[ y' = \frac{d}{z} y \]
How can we represent this with our 4x4 matrices?
Projective Transformations

\[
\begin{pmatrix}
\tilde{x} \\
\tilde{y} \\
\tilde{z} \\
\tilde{w}
\end{pmatrix}
\rightarrow
\begin{align*}
x &= \frac{\tilde{x}}{\tilde{w}} \\
y &= \frac{\tilde{y}}{\tilde{w}} \\
z &= \frac{\tilde{z}}{\tilde{w}}
\end{align*}
\]

Example:

\[
M = \begin{pmatrix}
2 & 0 & -1 \\
0 & 3 & 0 \\
0 & \frac{2}{3} & \frac{1}{3}
\end{pmatrix}
\]

[Shirley, Marschner]
Projective Transformations

\[
\begin{pmatrix}
\tilde{x} \\
\tilde{y} \\
\tilde{z} \\
w
\end{pmatrix}
\rightarrow
\begin{align*}
x &= \frac{\tilde{x}}{w} \\
y &= \frac{\tilde{y}}{w} \\
z &= \frac{\tilde{z}}{w}
\end{align*}
\]

Example:

\[
M = \begin{pmatrix}
2 & 0 & -1 \\
0 & 3 & 0 \\
0 & \frac{2}{3} & \frac{1}{3}
\end{pmatrix}
\]

We can now implement perspective projection!
Perspective Projection

\[ y' = \frac{d}{z} y \]

both x and y get multiplied by \( d/z \)
Simple perspective projection

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
= 
\begin{pmatrix}
x \\
y \\
z \\
1/d
\end{pmatrix}
\Rightarrow
\begin{align*}
x' &= \frac{d}{z}x \\
y' &= \frac{d}{z}y \\
z' &= \frac{d}{z}z = d
\end{align*}

This achieves a simple perspective projection onto the view plane \( z = d \)

but we’ve lost all information about \( z \)!
Perspective Projection

\[
P = \begin{pmatrix}
  n & 0 & 0 & 0 & 0 \\
  0 & n & 0 & 0 & 0 \\
  0 & 0 & n+f & -fn & 0 \\
  0 & 0 & 1 & 0 & 0 \\
\end{pmatrix} \quad z' = (n + f) - \frac{n f}{z}
\]

Example:

\[
\begin{align*}
  n &= -1 \\
  f &= -2
\end{align*}
\]
\[ M_{\text{per}} = M_{\text{orth}} P \]
OpenGL Perspective Viewing

\[ \text{glFrustum}(x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}, \text{near}, \text{far}) \]

Clipping volume (frustrum) for a perspective projection
Using Field of View

With `glFrustum` it is often difficult to get the desired view with `gluPerspective(fovy, aspect, near, far)` often provides a better interface.

![Diagram showing field of view](image)

- front plane
- aspect $= \frac{w}{h}$
Clipping after the perspective transformation can cause problems.
OpenGL clips **after** projection and **before** perspective division

\[-w \leq x \leq w\]

\[-w \leq y \leq w\]

\[-w \leq z \leq w\]