Viewing Transformations
Viewing transformations

- Move objects from their 3D locations to their positions in a 2D view

The viewing transformation also project any pixels viewing ray back to the pixel’s position in image space
Decomposition of viewing transforms

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

there are several names for these spaces: “camera space” = “eye space”, “canonical view volume” = “clip space”= “normalized device coordinates”, “screen space=pixel coordinates”
and for the transforms: “camera transformation” = “viewing transformation”
Viewport transform

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

\[(x, y, z) \in [-1, 1]^3 \quad x' \in [-.5, n_x - .5] \quad y' \in [-.5, n_y - .5]\]
Viewport transform

- Camera transform
- Projection transform
- Viewport transform

$M_{vp}$

<whiteboard>
Orthographic Projection Transform

\[
\begin{align*}
M_{\text{orth}} &= (r, t, h) \\
      &= (l, b, f) \\
      &= (-1, -1, -1) \\
      &= (1, 1, 1)
\end{align*}
\]
Camera Transform

- Camera transform
- Projection transform
- Viewport transform
Camera Transform

How do we specify the camera configuration?
Camera Transform

How do we specify the camera configuration?

\[ \text{eye position} \]
Camera Transform

How do we specify the camera configuration?
Camera Transform

How do we specify the camera configuration?

**up vector**
Camera Transform

How do we specify the camera configuration?
Camera Transform

\[
\begin{align*}
  w &= -\frac{g}{\|g\|} \\
  u &= \frac{t \times w}{\|t \times w\|} \\
  v &= w \times u
\end{align*}
\]

\(M_{\text{cam}}\) <whiteboard>
Perspective Viewing
rigid – translation and rotation only – parallel lines and angles are preserved
affine – scaling, shear, translation, rotation – parallel lines preserved, angles **not** preserved
projective – parallel lines and angles **not** preserved
note that the height, \( y' \), in camera space is proportional to \( y \) and inversely proportion to \( z \). We want to be able to specify such a transformation with our 4x4 matrix machinery.
How can we represent this with our 4x4 matrices?

Note that the height, $y'$, in camera space is proportional to $y$ and inversely proportional to $z$. We want to be able to specify such a transformation with our 4x4 matrix machinery.
Note: this makes our homogeneous representation for points unique only up to a constant
Projective Transformations

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

Example:

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

We can now implement perspective projection!
note that both x and y will be transformed
This simple projection matrix won’t suffice. We need to preserve z information for later hidden surface removal.

whiteboard: derive \( P \)
The perspective transformation does not preserve $z$ completely, but it preserves $z = n, f$ and is monotone (preserves ordering) with respect to $z$.
So far we’ve mapped the view frustum to a rectangular box. This rectangular box has the same near face as the view frustum. The far face has been mapped down to the far face of the box. This mapping is given by $P$. The bottom figure shows how lines in the view frustum get mapped to the rect. box.
We’re not quite done yet thought, because the projection transform should map the view frustum to the canonical view volume.
We need a second mapping to get our points into the canonical view volume. This second mapping is a mapping from one box to another. So it’s given by an orthographic mapping, \( M_{\text{orth}} \). The final perspective transformation is the composition of \( P \) and \( M_{\text{orth}} \).
Here’s how you set up a perspective view in OpenGL. Note that near and far are both negative, but you pass their absolute values to OpenGL.
Sometimes it's more convenient to just give an angle, the field-of-view, and an aspect ratio, instead of l, r, t, b. The glu library provides such a function. It will figure out l, r, t, b, and call glFrustum for you.
Clipping after the perspective transformation can cause problems
OpenGL clips **after** projection and **before** perspective division

\[-w \leq x \leq w\]
\[-w \leq y \leq w\]
\[-w \leq z \leq w\]