CS230 : Computer Graphics
Lecture 6: Viewing Transformations

Tamar Shinar
Computer Science & Engineering
UC Riverside
Rendering approaches

1. image-oriented
   foreach pixel ...

2. object-oriented
   foreach object ...

*object-oriented rendering* – e.g., OpenGL graphics pipeline, Renderman (REYES)
task: figure out where a point in the geometry will land on the final image pixels
Projection: map 3D scene to 2D image
Orthographic projection

- parallel lines appear parallel (unlike perspective proj.)
- equal length lines appear equal length (unlike perspective proj.)
Perspective projection
two-point perspective  three-point perspective
Viewing Transformations
Viewing transformations

- Map points from their 3D locations to their positions in a 2D view

The viewing transformation also projects any point along pixel’s viewing ray back to the pixel’s position in screen (or image) space.
Decomposition of viewing transforms

- **Camera transform**
  - rigid body transformation
  - place camera at origin

- **Projection transform**
  - $x, y, z$ in $[-1, 1]$
  - depends on type of projection

- **Viewport transform**
  - map to pixel coordinates

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”
(x, y, z) → (x', y', z')

(x, y, z) ∈ [-1, 1]^3

x' ∈ [-.5, nx - .5]
y' ∈ [-.5, ny - .5]
z' ∈ [-1, 1]
Viewport transform

$M_{vp}$

$\langle$whiteboard$>$

$\{n_x, n_y\}$
Orthographic Projection Transform

Camera transform

Projection transform

Viewport transform

$M_{ortho}$

<whiteboard>
Camera Transform

1. Camera transform
2. Projection transform
3. Viewport transform

Diagram:
- Camera space
- Canonical view volume
- Screen space
Camera Transform

*How do we specify the camera configuration?*
*(orthogonal case)*
Camera Transform

How do we specify the camera configuration?

eye position
Camera Transform

How do we specify the camera configuration?

gaze direction
Camera Transform

*How do we specify the camera configuration?*

*up vector*
Camera Transform

How do we specify the camera configuration?
Camera Transform

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

\[ M_{\text{cam}} \ <\text{whiteboard}> \]
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.
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.
Projective Transformations

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

Use 4th coordinate as the denominator

Example:

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

Note: this makes our homogeneous representation for points unique only up to a constant
Projective Transformations

We can now implement perspective projection!

\[
\begin{pmatrix}
\tilde{x} \\
\tilde{y} \\
\tilde{z} \\
w
\end{pmatrix} \rightarrow \begin{pmatrix}
x \\
y \\
z \\
w
\end{pmatrix} = \begin{pmatrix}
\tilde{x} \\
\tilde{y} \\
\tilde{z} \\
w
\end{pmatrix} \begin{pmatrix}
1 \\
0 \\
0 \\
1
\end{pmatrix}
\]

Example:

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

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

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

note that both x and y will be transformed
Simple perspective projection

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

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

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

This simple projection matrix won’t suffice. We need to preserve \( z \) information for later hidden surface removal.

whiteboard: derive P
Perspective Projection

\[ P = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n + f & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix} \]

\[ z' = (n + f) - \frac{nf}{z} \]

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 $M_{\text{orth}}$. 

\[
M_{\text{per}} = M_{\text{orth}}P
\]