Viewing

CMSC 435/634
Coordinate System / Space

- Origin + Axes
- Reference frame
- Convert with $4 \times 4$ matrix
- OpenGL convention, name matrix $XFromY$
  - $\vec{p}_{table} = TableFromPencil \vec{p}_{pencil}$
  - $\vec{p}_{room} = RoomFromTable \ TableFromPencil \vec{p}_{pencil}$
  - $\vec{p}_{room} = RoomFromPencil \vec{p}_{pencil}$
- D3D convention is transposed, so $YtoX$
  - $\vec{p}_{table} = \vec{p}_{pencil} \ PencilToTable$
  - $\vec{p}_{room} = \vec{p}_{pencil} \ PencilToTable \ TableToRoom$
  - $\vec{p}_{room} = \vec{p}_{pencil} \ PencilToRoom$
Common Spaces

• Object / Model
  • Logical coordinates for modeling
  • May have several more levels

• World
  • Common coordinates for everything

• View / Camera / Eye
  • eye/camera at (0,0,0), looking down Z (or -Z) axis
  • planes: left, right, top, bottom, near/hither, far/yon

• Normalized Device Coordinates (NDC) / Clip
  • Visible portion of scene in a box
  • Common choices: (-1,-1,-1)–(1,1,1) or (-1,-1,0)–(1,1,1)
  • Might occasionally see (0,0,0)–(1,1,1)

• Raster / Pixel / Viewport
  • 0,0 to x-resolution, y-resolution

• Device / Screen
  • May translate to fit actual screen
Nesting

Room

Desk

Student Book

Notebook

Desk

Student Notebook

Podium

Laptop

Board

Eraser
Matrix Stack

- Remember transformation, return to it later
- Push a copy, modify the copy, pop
- Keep both matrix and inverse

```
// Code
// Stack
// Identity

transform (WorldFromRoom);    // WfR
push;
// WfR
transform (RoomFromDesk);     // WfR WfD
push;
// WfR WfD
transform (DeskFromStudent);  // WfR WfD WfS
pop;
// WfR WfD
push;
// WfR WfD
transform (DeskFromBook);     // WfR WfD WfB
...
```
Model $\rightarrow$ World / Model $\rightarrow$ View

- **Model $\rightarrow$ World**
  - All shading and rendering in World space
  - Transform all objects and lights
- **Ray tracing implicitly does World $\rightarrow$ Raster**
- **Model $\rightarrow$ View**
  - Serves just as well for single view
  - Old-style OpenGL had a *MODELVIEW* matrix and did lighting in view space
World → View

- Also called Viewing or Camera transform
- LookAt
  - Use $\vec{\text{eye}}$, $\vec{\text{to}}$, $\vec{\text{up}}$ to build $\hat{u}$, $\hat{v}$, $\hat{w}$
  - $WorldFromView = [ \hat{u} \mid \hat{v} \mid \hat{w} \mid \vec{\text{eye}} ]$
- Roll / Pitch / Yaw
  - Translate to camera center, rotate around camera
  - $WorldFromView = T \ R_y \ R_x \ R_z$
  - $ViewFromWorld = R_z^{-1} \ R_x^{-1} \ R_y^{-1} \ T^{-1}$
  - Can have gimbal lock
- Orbit
  - Rotate around object center, translate out
  - $WorldFromView = R_y \ R_x \ R_z \ T$
  - $ViewFromWorld = T^{-1} \ R_z^{-1} \ R_x^{-1} \ R_y^{-1}$
  - Also can have gimbal lock
View→NDC

- Also called *Projection* transform
- Orthographic / Parallel
  - Translate & Scale to view volume
  - $[l, r], [b, t], [n, f] \rightarrow [-1, 1], [-1, 1], [-1, 1]$
  - Translate to $[0, r-l], [0, t-b], [0, f-n]$
  - Scale to $[0, 2], [0, 2], [0, 2]$
  - Translate to $[-1, 1], [-1, 1], [-1, 1]$
    
    \[
    \begin{bmatrix}
    2 \cdot \frac{r}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\
    0 & 2 \cdot \frac{t}{t-b} & 0 & -\frac{t+b}{t-b} \\
    0 & 0 & 2 \cdot \frac{n}{n-f} & -\frac{n+f}{n-f} \\
    0 & 0 & 0 & 1
    \end{bmatrix}
    \]

- Perspective
  - More complicated...
NDC→Raster

- Also called *Viewport* transform
- $[-1, 1], [-1, 1], [-1, 1] \rightarrow [-\frac{1}{2}, n_x - \frac{1}{2}], [-\frac{1}{2}, n_y - \frac{1}{2}], [-\frac{1}{2}, n_z - \frac{1}{2}]$

- Translate to $[0, 2], [0, 2], [0, 2]$
- Scale to $[0, n_x], [0, n_y], [0, n_z]$
- Translate to $[-\frac{1}{2}, n_x - \frac{1}{2}], [-\frac{1}{2}, n_y - \frac{1}{2}], [-\frac{1}{2}, n_z - \frac{1}{2}]$

$$\begin{bmatrix}
\frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\
0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\
0 & 0 & \frac{n_z}{2} & \frac{n_z - 1}{2} \\
0 & 0 & 0 & 1
\end{bmatrix}$$
Raster $\rightarrow$ Screen

- Usually just a translation
  - More complicated for tiled displays, domes, etc.
- Usually handled by windowing system
Perspective View Frustum

- Orthographic view volume is a rectangular volume
- Perspective is a truncated pyramid or *frustum*
**Perspective Transform**

- Ray tracing
  - Given screen \((s_x, s_y)\), parameterize all points \(\vec{p}\)

- Perspective Transform
  - Given \(\vec{p}\), find \((s_x, s_y)\)
  - Use similar triangles
    - \(s_y/d = p_y/p_z\)
    - So \(s_y = d \cdot p_y/p_z\)
Homogeneous Equations

- Same total degree for every term
- Introduce a new redundant variable
  - $a X + b Y + c = 0$
    - $X = x/w$, $Y = y/w$
    - $a x/w + b y/w + c = 0$
    - $→ a x + b y + c w = 0$
  - $a X^2 + b X Y + c Y^2 + d X + e Y + f = 0$
    - $X = x/w$, $Y = y/w$
    - $a x^2/w^2 + b x y/w^2 + c y^2/w^2 + d x/w + e y/w + f = 0$
    - $→ a x^2 + b x y + c y^2 + d x w + e y w + f w^2 = 0$
Homogeneous Coordinates

- Rather than \((X, Y, Z, 1)\), use \((x, y, z, w)\)
- Real 3D point is \((X, Y, Z, 1) = (x/w, y/w, z/w, w/w)\)
- Can represent Perspective Transform as 4x4 matrix

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0
\end{bmatrix}
\begin{bmatrix}
p_x \\
p_y \\
p_z \\
1
\end{bmatrix}
= \begin{bmatrix}
p_x \\
p_y \\
p_z \\
p_z/d
\end{bmatrix}
\rightarrow \begin{bmatrix}
d & p_x/p_z \\
d & p_y/p_z \\
d & p_z/d \\
d
\end{bmatrix}
\]
Homogeneous Depth

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0
\end{bmatrix}
\begin{bmatrix}
p_x \\
p_y \\
p_z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
p_x \\
p_y \\
p_z \\
p_z/d
\end{bmatrix}
\rightarrow 
\begin{bmatrix}
d & p_x/p_z \\
d & p_y/p_z \\
d
\end{bmatrix}
\]

- Lose depth information
- Can't get \( d \frac{p'}{z} = p_z \)
  - Plus \( x/z, y/z, z \) isn't linear
- Use *Projective Geometry*
Projective Geometry

- If \( x, y, z \) lie on a plane, \( x/z, y/z, 1/z \) also lie on a plane
- \( 1/z \) is strictly ordered: if \( z_1 < z_2 \), then \( 1/z_1 > 1/z_2 \)
- New matrix:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
p_x \\
p_y \\
p_z \\
1
\end{bmatrix}
= \begin{bmatrix}
p_x \\
p_y \\
1 \\
p_z
\end{bmatrix}
\rightarrow \begin{bmatrix}
p_x/p_z \\
p_y/p_z \\
1/p_z
\end{bmatrix}
\]
Getting Fancy

• Add scale & translate
  • Field of View
  • Near/far range

\[
\begin{bmatrix}
a & 0 & 0 & 0 \\
0 & a & 0 & 0 \\
0 & 0 & b & c \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
p_x \\
p_y \\
p_z \\
1 \\
\end{bmatrix}
= 
\begin{bmatrix}
ap_x \\
ap_y \\
b p_z + c \\
-p_z \\
\end{bmatrix}
\rightarrow 
\begin{bmatrix}
-a p_x/p_z \\
-a p_y/p_z \\
-b - c/p_z \\
\end{bmatrix}
\]

• \(a = \cotan(FOV/2)\)

• Solve for \(n \rightarrow -1\) and \(f \rightarrow 1\)
  • \(b = (n + f)/(n - f)\)
  • \(c = (2nf)/(f - n)\)
Advanced Field of View

- Given field of view angle in square window
- Non-square window
  - Given both horizontal and vertical fields of view
  - Given one field of view and *aspect ratio*
  - Given diagonal field of view (rare in graphics)
- Given image dimensions, set distance
  - Camera image sensor size and focal length
  - Often given for 35mm film size (24mm useable)

![Diagram showing 35mm and 24mm image sizes](image.png)

- \( vfov = 2 \tan(\frac{12}{d}) \)
- 35mm lens = 37.8°; 200mm lens = 6.9°
Off-Center Projection

- Tiled displays, Head tracking
- Distance $d$ is **perpendicular** distance to image plane
- Left/right and top/bottom are not centered on 0,0
- Field of view $= \theta_t - \theta_b$
Fun with Homogeneous Coordinates

• Any finite scale factor of \( x,y,z,w \) represents the same point
  • \( (kX, kY, kZ, k) = (wX, wY, wZ, w) = (X, Y, Z, 1) \)
• \( w=0 \) for vectors, or points \( \infty \) far away in \( X,Y,Z \) direction
  • \( \lim_{w \to 0} (x/w, y/w, z/w) \)
• Can skip scale factor when normalizing
  • normalize\((x/w, y/w, z/w) = \) normalize\((x, y, z) \)
• Projection transform puts eye at \( z = -\infty \)
  • Projection\(^{-1} \ast (0, 0, -1, 0) \) is eye position
  • \( = -3rd \) column
Unify point and directional lights

• Point light location: \((L_x, L_y, L_z, 1)\)
• Directional light = light \(\infty\) far away in \((L_x, L_y, L_z)\) direction
  • \((L_x, L_y, L_z, 0)\)
• Light vector:

\[
\hat{l} = \text{normalize} \ (L - P)
\]

\[
= \text{normalize} \left( \frac{L_{xyz}}{L_w} - \frac{P_{xyz}}{P_w} \right)
\]

\[
= \text{normalize} \left( \frac{P_wL_{xyz} - L_wP_{xyz}}{L_wP_w} \right)
\]

\[
= \text{normalize} \ (P_wL_{xyz} - L_wP_{xyz})
\]

• If you know \(P_w = 1\) in world space:

\[
\hat{l} = \text{normalize} \ (L_{xyz} - L_wP_{xyz})
\]