**Viewing in 3D: Camera Analogy**

- Pin-hole camera model (all objects in perfect focus)
- *Camera Transform*: Makes the camera (user) the center of a new coordinate system
- **Specification**: View reference point, Up vector, Look-at direction

---

**Vertex Transformation: Order of Operations**

1. **Modelview Matrix**
2. **Projection Matrix**
3. **Perspective Division**
4. **Viewport Transformation**

---

**Camera Model**

- **View Reference Point**
- **Up Vector**
- **Look-at Direction**

---

**Camera Transformation**

- Essentially a change of basis.
- Two operations, (1) Translate camera origin to world, (2) rotate camera axes \((U, V, N)\) to coincide with \((X, Y, Z)\)
- An example of transformation of coordinate systems.

\[
\begin{bmatrix}
\mathbf{M}_{\text{wv}}
\end{bmatrix} =
\begin{bmatrix}
\mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z & 0 \\
\mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z & 0 \\
\mathbf{n}_x & \mathbf{n}_y & \mathbf{n}_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & -r_x \\
0 & 1 & 0 & -r_y \\
0 & 0 & 1 & -r_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
How do you determine $U, V, N$?

- Must specify View Reference Point ($V_{RP}$), Up Vector, ($U_{P}$) and Look at Direction ($N_{i}$)
- Normalize $\vec{N}', \vec{U}'$

\[
\vec{N} = \frac{\vec{N}_i}{|\vec{N}_i|}
\]
\[
\vec{V}' = \frac{\vec{U}_P}{|\vec{U}_P|}
\]

- Calculate $\vec{U}$ and $\vec{V}$

\[
\vec{U} = \vec{N} \times \vec{V}'
\]
\[
\vec{V} = \vec{U} \times \vec{N}
\]

- $(\vec{U}, \vec{V}, \vec{N})$ form the axes of the camera coordinate system.

Projections

- Prior to rendering, 3D objects must be projected to a plane.
- Orthogonal vs. Perspective Projections

Orthographic Projection

- Simple, just skip the Z(depth) coordinate and render
- Applications: Architectural drawings, accurate extraction of dimensions

\[
M_{ortho} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \rightarrow
\begin{bmatrix}
P_z \\
P_y \\
P_x \\
1
\end{bmatrix} =
\begin{bmatrix}
P_z \\
P_y \\
P_x \\
1
\end{bmatrix}
\]

- Insufficient to project to a plane (positive and negative z values are projected.)
- Instead, project to a certain interval (near plane, far plane),
- Typically restricted to the rectilinear shape ($l, r, b, t, n, f$),
- Transformed to the the canonical view volume, axis-aligned cube with corners $(-1, -1, -1)$ and $(1, 1, 1)$
- OpenGL: `glOrtho()`
Orthographic Projection Transform

\[ \vec{M}_o = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ \frac{2}{t-b} & 0 & 0 & 0 \\ \frac{2}{f-n} & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \]

Perspective Projection

- Projectors from object converge to a point, the center of projection
- Projected views more realistic, distant objects are smaller
- Closer to our perception of the world.
- Non-linear transform, parallel lines do not remain parallel after projection.

Perspective Projection

- Projectors from object converge to a point, the center of projection
- Projected views more realistic, distant objects are smaller
- Closer to our perception of the world.
- Non-linear transform, parallel lines do not remain parallel after projection.

Perspective Projection

Projection Matrix

\[ \vec{M}_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix} \]

Dividing by the \( w \) term (homogenizing),

\[ q_x = -\frac{dP_x}{P_z}, \quad q_y = -\frac{dP_y}{P_z}, \quad q_z = -d \frac{P_z}{P_z} \]
Perspective Projection: Transform to Canonical Volume

- View Volume: Truncated pyramid (frustum)
- Perspective Transformation to a cube
- Projectors become parallel, viewer moves to infinity
- Untransformed cube in view space becomes a warped cube after perspective transformation.

OpenGL Perspective Transform Matrix

- Multiplication by \( S(1, 1, -1) \)
- Near and Far values \((n', f')\) are positive values, with \(0 < n' < f'\).

\[
\tilde{M}_{\text{OpenGL}} = \begin{bmatrix}
\frac{2n'}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n'}{t-b} & \frac{t+b}{t-b} & 0 \\
0 & 0 & \frac{f+n'}{f-n'} & -\frac{2fn'}{f-n'} \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

DirectX Perspective Transform Matrix

- Maps the near plane to \( z = 0 \), and far plane to \( z = 1 \).
- Uses a left-handed coordinate system
- DirectX looks along the positive z-axis and has positive near and far values.

\[
\tilde{M}_{\text{DirectX}} = \begin{bmatrix}
\frac{2n'}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n'}{t-b} & \frac{t+b}{t-b} & 0 \\
0 & 0 & \frac{f+n'}{f-n'} & -\frac{2fn'}{f-n'} \\
0 & 0 & -1 & 0
\end{bmatrix}
\]

⇒ Slightly different forms for OpenGL and DirectX implementations.