Assignment 6 - Viewing I & II

Alyssa Perry

Tuesday, March 18, 2025

1. Parallel Projection vs. Perspective Projection

   When we project a 3D point onto a view plane, we generally have two main methods to choose from: **parallel projection** and **perspective projection**. With parallel projection, everything stays in proportion because all the projection lines are parallel. This means that the size of an object doesn't change based on how far it is from us, which is super helpful in technical fields like engineering where accurate measurements matter. A special type of parallel projection is **orthogonal projection**, where the projection lines hit the view plane at right angles, keeping angles and distances intact. There's also **oblique projection**, which allows the projection lines to hit the view plane at an angle, giving a unique look to the objects.

   On the flip side, **perspective projection** is more like how we actually see the world. The projection lines converge at a vanishing point, making objects look smaller as they get further away, which creates depth. While this method doesn't keep the actual size or proportions of the objects, it gives a more lifelike representation of 3D scenes. You'll often see perspective projection in computer graphics, video games, and simulations where depth perception is key for making things look real. However, it can be a bit more complex to calculate compared to parallel projection, which is usually quicker and simpler.

2. Normalized View Volume and Its Importance

   A **normalized view volume** is basically a standard way to represent space in computer graphics, making it easier to render 3D scenes. Picture it as a **cube-shaped** volume that stretches from $[-1, 1]$ on the x, y, and z axes, with the **center right at the origin**. The coordinate system in this normalized view volume follows a **left-handed convention**, meaning the z-axis goes into the scene as it increases.

   Transforming a camera's view volume into this normalized view volume is a vital step in the graphics pipeline. It helps standardize the viewing space across all camera setups. Without this transformation, each scene would need its own set of calculations for things like clipping, depth testing, and rendering. By mapping everything to the same $[-1, 1]$ range, we make rendering more efficient and consistent, which is a big win for graphics processing. Plus, this transformation allows us to take advantage of hardware acceleration and simplifies viewport transformations, ensuring that objects are positioned correctly on the screen, no matter how the original camera was set up.

3. Determining the Viewing Volume in Orthogonal Projection

   In **orthogonal projection**, we define the viewing volume with six planes that create a rectangular box in 3D space. You can easily determine this box using **two opposite corner points**: the **minimum corner** (x_l, y_b, z_n) and the **maximum corner** (x_r, y_t, z_f).

   Each of these coordinates plays a specific role in outlining the viewable area. The values (x_l, x_r) set the **horizontal range**, with x_l marking the left side and x_r marking the right side. For the vertical range, (y_b, y_t) come into play, where y_b is the bottom and y_t is the top of the view volume. Lastly, (z_n, z_f) define the **depth range**, with z_n as the near clipping plane and z_f as the far clipping plane. Anything outside this box gets clipped and won't show up on the screen.

   By using these two corner points, we establish the entire 3D space that should be visible in orthogonal projection. This method ensures that all objects within the defined boundaries are accurately projected onto the view plane without distortion, which is especially handy for applications that need precise spatial representation, like CAD software and architectural visualization.

4. Camera Basis and Transformation Matrix

   Given:

   - Camera position: $e = (0, 1, 0)$
   - Gaze vector: $g = (0, -1, 0)$
   - View-up vector: $V = (1, 1, 0)$

   Step 1: Compute Camera Basis

$$\begin{aligned} w &= -\frac{g}{|g|} = -\frac{(0,-1,0)}{1} = (0,1,0) \\ u &= \frac{V \times w}{|V \times w|} = \frac{(1,1,0) \times (0,1,0)}{|(1,1,0) \times (0,1,0)|} = \frac{(0,0,1)}{1} = (0,0,1) \\ v &= w \times u = (0,1,0) \times (0,0,1) = (1,0,0) \end{aligned}$$

   Step 2: Compute 4×4 Camera Transformation Matrix

$$\begin{aligned} M_{world \to view} &= \begin{bmatrix} u_x & u_y & u_z & -u \cdot e \\ v_x & v_y & v_z & -v \cdot e \\ w_x & w_y & w_z & -w \cdot e \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

5. View Coordinates of a Given Point

   Given: World coordinates $P = (1, 2, 3)$.

$$\begin{aligned} P' &= M_{world\to view} \times P \\ &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0\cdot 1 + 0\cdot 2 + 1\cdot 3 + 0\cdot 1 \\ 1\cdot 1 + 0\cdot 2 + 0\cdot 3 - 1\cdot 1 \\ 0\cdot 1 + 1\cdot 2 + 0\cdot 3 - 1\cdot 1 \\ 0\cdot 1 + 0\cdot 2 + 0\cdot 3 + 1\cdot 1 \end{bmatrix} \\ &= \begin{bmatrix} 3 \\ 0 \\ 1 \\ 1 \end{bmatrix} \end{aligned}$$

   View coordinates: $(3, 0, 1)$.

6. Orthogonal Projection Matrix

   Given:

   - $x_l = 1, x_r = 4$
   - $y_b = 1, y_t = 5$
   - $z_n = -2, z_f = -5$

$$
\begin{aligned}
M_{ortho\rightarrow norm} &= \begin{bmatrix} \frac{2}{x_r - x_l} & 0 & 0 & -\frac{x_r + x_l}{x_r - x_l} \\ 0 & \frac{2}{y_t - y_b} & 0 & -\frac{y_t + y_b}{y_t - y_b} \\ 0 & 0 & \frac{-2}{z_f - z_n} & -\frac{z_f + z_n}{z_f - z_n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \frac{2}{4-1} & 0 & 0 & -\frac{4+1}{4-1} \\ 0 & \frac{2}{5-1} & 0 & -\frac{5+1}{5-1} \\ 0 & 0 & \frac{-2}{-5+2} & -\frac{-5-2}{-5+2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \frac{2}{3} & 0 & 0 & -\frac{5}{3} \\ 0 & \frac{1}{2} & 0 & -\frac{6}{4} \\ 0 & 0 & \frac{2}{3} & \frac{7}{3} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 0.67 & 0 & 0 & -1.67 \\ 0 & 0.5 & 0 & -1.5 \\ 0 & 0 & -0.67 & -2.33 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$