1 Overall Process

• Viewing is the process of mapping points in the world coordinate system to points in the device coordinate system. This consists of three general steps:

  1. Select the viewing volume.
  2. Project the points in the viewing volume.
  3. Map the projected points to the target device.

2 Viewing Volumes

• Figures 1 and 2 show how orthographic and frustum viewing volumes are defined in OpenGL, respectively. Figure 3 shows the same frustum viewing volume, but expressed with different arguments (e.g. `gluPerspective(fovy, aspect, near, far)`).

• Note that these aren’t the only ways to define these volumes, nor are they the only volumes that you can do (remember fish-eye view?). But, since these are the volumes that OpenGL uses, we will study their implementation from that standpoint.

• So that’s step 1. For step 2, we need to take whatever is in that viewing volume, and see how they project onto the plane defined by \( N \) in relation to the camera or viewpoint.

• Step 3 takes whatever is on the near plane and “transfers” it to the display device, such as a window (or sub-window) on the screen.

3 Projection to the Near Plane

• The approach we take is to normalize the camera coordinates into an intermediate space, and then convert that space into the device coordinates. This intermediate space is called the normalized device coordinate system (NDC).
Figure 1: Orthographic viewing volume, as defined in OpenGL.
Figure 2: Frustum viewing volume, as defined in OpenGL.
Figure 3: Frustum viewing volume, as defined in OpenGL via `gluPerspective(fovy, aspect, near, far)`. 
• This normalized space converts any \((x, y, z)\) within the viewing volume so that:

\[
\begin{align*}
-1 &\leq x \leq 1 \\
-1 &\leq y \leq 1 \\
-1 &\leq z \leq 1
\end{align*}
\]

• Thus, the conversion is to map \(L \rightarrow -1\) and \(R \rightarrow 1\) along the \(x\) axis and \(T \rightarrow -1\) and \(B \rightarrow 1\) along the \(y\) axis.

• Note however that along the \(z\) axis, the mapping is \(-N \rightarrow -1\) and \(-F \rightarrow 1\), since \(N\) and \(F\) are expressed as distances from the camera but our coordinate system is right-handed.

3.1 Orthogonal Projection

• Observe how, in an orthogonal viewing volume, \((x, y)\) projects to exactly \((x, y)\) on the near plane; no calculations are necessary.

• Figure 4 illustrates this visually. For our derivation, we will convert \((x, y, z)\) in terms of camera coordinates into \((x', y', z')\) in normalized device coordinates.

• Pause for a moment to think about it — how would you derive this?
Figure 4: Orthogonal conversion from camera to normalized device coordinates.
OK, time’s up. Note how we can do everything with ratios:

\[
\frac{x - L}{R - L} = \frac{x' - (-1)}{1 - (-1)} = \frac{x' + 1}{2}
\]

Then we just solve for \(x'\)...

\[
x' = \frac{2(x - L)}{R - L} - 1 = \frac{2x - 2L - (R - L)}{R - L}
\]

\[
x' = \left(\frac{2}{R - L}\right)x - \left(\frac{R + L}{R - L}\right)
\]  

(1)

Deriving \(y'\) is analogous, leading to:

\[
y' = \left(\frac{2}{T - B}\right)y - \left(\frac{T + B}{T - B}\right)
\]

(2)

Ditto for \(z'\), taking note of how we’re using \(-N\) and \(-F\):

\[
z' = \left(\frac{-2}{F - N}\right)z - \left(\frac{F + N}{F - N}\right)
\]

(3)

This is a transform — and so it can be expressed as a matrix!

\[
\begin{bmatrix}
\frac{2}{R-L} & 0 & 0 & -\frac{R+L}{R-L} \\
0 & \frac{2}{T-B} & 0 & -\frac{T+B}{T-B} \\
0 & 0 & \frac{F-2}{F-N} & -\frac{F+B}{F-N} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

(4)

3.2 Perspective Projection

For perspective projection, we have to make an extra calculation to see where any arbitrary \((x, y)\) lands at \(z = -N\). Figure 5 shows this relationship, looking down on the \(xz\) plane.

Based on Figure 5, we can invoke the proportions between similar triangles to say:

\[
\frac{x}{z} = \frac{x_e}{-N}
\]

\[
x_e = \frac{Nx}{-z}
\]
Figure 5: Perspective conversion from camera to normalized device coordinates.
• The same goes for $y$ and $y_e$:

$$\frac{y}{z} = \frac{y_e}{-N}$$

$$y_e = \frac{N y}{-z}$$

• Now we can plug $x_e$ and $y_e$ into (1) and (2), then substitute in terms of $x$ and $y$:

$$x' = \left( \frac{2}{R - L} \right) x_e - \left( \frac{R + L}{R - L} \right)$$

$$= \left( \frac{2}{R - L} \right) \left( \frac{N x}{-z} \right) - \left( \frac{R + L}{R - L} \right)$$

$$= \left( \frac{2N}{R - L} \right) x + z \left( \frac{R + L}{R - L} \right) / -z$$

$$y' = \left( \frac{2}{T - B} \right) y_e - \left( \frac{T + B}{T - B} \right)$$

$$= \left( \frac{2N}{T - B} \right) y + z \left( \frac{T + B}{T - B} \right) / -z$$

• Based on these, we can partially build our projection matrix — note how the division by $-z$ is handled by the last row, based on our use of homogeneous coordinates:

$$\begin{bmatrix}
\frac{2N}{R - L} & 0 & \frac{R + L}{R - L} & 0 \\
0 & \frac{2N}{T - B} & \frac{R + B}{T - B} & 0 \\
0 & 0 & \alpha & \beta \\
0 & 0 & -1 & 0
\end{bmatrix}$$

(5)

• But what are $\alpha$ and $\beta$? Based on matrix multiplication, we can say that:

$$z' = \frac{\alpha z + \beta}{-z}$$

• Since the mapping to NDC will have $z = -N$ when $z' = -1$ and $z = -F$ when $z' = 1$, then we can replace $z$ and $z'$ with these identities to get two equations in two variables, namely $\alpha$ and $\beta$:

$$-1 = \frac{\alpha(-N) + \beta}{N}$$

$$1 = \frac{\alpha(-F) + \beta}{F}$$

• Multiplying out the denominators yields:

$$-N = -\alpha N + \beta$$

$$-F = \alpha F - \beta$$

(6)

(7)
• Adding (6) and (7) eliminates $\beta$:

$$-N - F = -\alpha N + \alpha F$$

$$-(F + N) = \alpha(F - N)$$

$$\alpha = \frac{F + N}{F - N}$$

(8)

• Substituting (8) for $\alpha$ in (7) leads to:

$$-N = \left(\frac{F + N}{F - N}\right)N + \beta$$

$$\beta = -N - \left(\frac{F + N}{F - N}\right)N$$

$$= -N[(F - N) + (F + N)]$$

$$= -2NF$$

$$F - N$$

• ...and so we have our perspective projection matrix!

$$\begin{bmatrix}
\frac{2N}{R-L} & 0 & \frac{R+L}{R-L} & 0 \\
0 & \frac{2N}{T-B} & \frac{T+L}{T-B} & 0 \\
0 & 0 & -\frac{F+N}{F-N} & -2NF \\
0 & 0 & -1 & 0
\end{bmatrix}$$

(9)

4 Projection to Viewport

• Projected points on the near plane now need to be converted into the display device’s coordinates — the viewport.

• Another way to think about this is to convert from NDCs to device coordinates (DCs).

• For a typical window or screen with dimensions $(width, height)$, this would be a coordinate system with some origin $(x_o, y_o)$ on the upper left corner and $(x_o + width - 1, y_o + height - 1)$ on the lower right — which is precisely what you set when you call `glViewport()`.

• Yet again, similar ratios come to our aid, based on Figure 6. Thus:

$$\frac{x' - (-1)}{1 - (-1)} = \frac{x' + 1}{2} = \frac{x_s - x_o}{width}$$

$$x_s = \frac{width(x' + 1)}{2} + x_o = \left(\frac{width}{2}\right) x' + \frac{width + 2x_o}{2}$$
Figure 6: Conversion to viewport (device) coordinates.
• $y_s$ is similar, except that you need to “invert” along the $y$ axis in the end since device coordinates typically increase from top to bottom:

$$y_s = \text{height} - \left( \frac{\text{height}}{2} y' + \frac{\text{height} + 2y_o}{2} \right)$$

$$= -\frac{\text{height}}{2} y' + \left( \text{height} - \frac{\text{height} + 2y_o}{2} \right)$$

• Of course, this is all best processed as a matrix, $3 \times 3$ this time because it’s two-dimensional.

• But wait — what happened to $z'$? We didn’t throw that away — we’ll be using that later on.

• But wait again — through all of this, hasn’t the camera been fixed at $(0, 0, 0)$, looking down the $z$-axis? The answer is yes. We haven’t covered how we handle moving and orienting the camera. Stay tuned...