The Modeling-Viewing Pipeline

Object Coordinates → World Coordinates → View Coordinates → Clip Coordinates → Normalized Device Coordinates → Device Coordinates

Modeling Transformations → Viewing Transformation → Projection Transformation

Viewing Transformation

Given:
1. In the World Coordinate System (WCS):
   a) Position of the Eye (E)
   b) The lookat point (A)
   c) The up vector, \( \mathbf{v}_{up} \)

Viewing Transformation

Given:
1. In the View Coordinate System (VCS):
   a) The distance of near and far clipping planes.
   b) Extents of the near plane, L, R, T, B.

Defining the VCS

\[ \mathbf{n} = \frac{\mathbf{A} - \mathbf{E}}{|| \mathbf{A} - \mathbf{E} ||} \]

\[ \mathbf{u} = \mathbf{v}_{up} \times \mathbf{n} / || \mathbf{v}_{up} \times \mathbf{n} || \]

\[ \mathbf{v} = \mathbf{A} - \mathbf{E} \]

\[ || \mathbf{v} || = \frac{|| \mathbf{A} - \mathbf{E} ||}{|| \mathbf{A} - \mathbf{E} ||} \]
The viewing transformation should:

- Map the origin of the WCS, \(O\), to the Eye, \(E\).
- Map the \(x\), \(y\), and \(z\) axes.

If the point \(P\) has coordinates \((x, y, z)\) in WCS, \((a, b, c)\) in VCS, then:

\[
\begin{bmatrix}
    a \\
    b \\
    c
\end{bmatrix}
= \begin{bmatrix}
    u & v & n \\
    x & y & z
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z
\end{bmatrix}
\]
From WCS to VCS

The viewing transformation should:
- Map the origin of the WCS, \(O\), to the Eye, \(E\).
- \(x \rightarrow u\)
- \(y \rightarrow v\)
- \(z \rightarrow n\)

\[
P \begin{bmatrix} a & b & c \end{bmatrix} = \begin{bmatrix} \vec{u} \cdot \vec{OP} \ \vec{v} \cdot \vec{OP} \ \vec{n} \cdot \vec{OP} \end{bmatrix} - \begin{bmatrix} \vec{u} \cdot \vec{OE} \ \vec{v} \cdot \vec{OE} \ \vec{n} \cdot \vec{OE} \end{bmatrix}
\]

\[
x = u \ \ y = v \ \ z = n
\]

Let the Near clipping plane be at \((0, 0, -N)\) and Far clipping plane be at \((0, 0, -F)\).

From VCS to CCS

We shear the frustum so that the direction of projection aligns with the \(n\) axis and frustum becomes symmetrically aligned about it.

If the extents of the near plane are given by \(L, R, T, B\) then:

\[
\begin{bmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
(L+R)/2 \\
(T+B)/2 \\
-N \\
1
\end{bmatrix}
\]
From VCS to CCS

If the extents of the near plane are given by \( L, R, T, B \) then:

\[
\begin{align*}
0 = (R + L)/2 - S_N, & \quad N = S_N, \\
0 = (T + B)/2 - S_N, & \quad S_N = \frac{T + B}{2} \\
\end{align*}
\]

We shear the frustum so that the direction of projection aligns with the \( n \) axis and frustum becomes symmetrically aligned about it.

After shearing the point \((L, B, -N)\) becomes \([(R - L)/2, (T - B)/2, -N]\)

After shearing the point \((R, T, -N)\) becomes \([(R - L)/2, (T - B)/2, -N]\)

So the scaling matrix should map \([-n, N, -N] \) to \([-n, 0, 0, 0]\)

Now we scale along \( u \) and \( v \) so that we get the \( n \) axis and projection aligns with faces of the frustum.

Now we transform the frustum to a canonical frustum.

This is equivalent to doing a perspective transform. It is called a projection normalization.

An Orthographic projection of a distorted object can be the same as a Perspective projection of the undistorted object.

Projection Normalization

So the complete transformation is:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

So the complete transformation is:
So the complete transformation is:

From VCS to NDCS

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

CCS coordinates retain the homogenous coordinate.
This is followed by a perspective divide stage where the coordinates are divided by the 'w' coordinate.
That puts all the coordinates within the normalized +/- 1 cube. This is called the Normalized Device Coordinate System.

From VCS to NDCS - OpenGL

\[
glm::frustum(L, R, B, T, N, F);
\]
or

\[
glm::perspective(fovy, aspect, N, F);
\]

N and F give the distance of Near and Far clipping planes from the Eye and must be positive numbers and must not be equal.

From NDCS to DCS

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

\[
y_w = \frac{y + 1}{2} \left( \frac{T - B}{T + B} \right)
\]

\[
z_w = \frac{z + 1}{2}
\]

glDepthRange(N_f, F_f); : Default range is 0 to 1

glViewport(x, y, width, height);

From NDCS to DCS - OpenGL

An Orthographic frustum is fully specified by L, R, T, B, N and F as well. Here there is no need to shear the frustum to make it symmetric – a translation can center it on the n-axis. This is followed by a scaling to transform the frustum to the canonical frustum. No projection normalization is required.

\[
A_v = \begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

glm::ortho(L, R, T, B, N, F)

or

glm::ortho(L, R, B, T)

Why carry the depth through?

Intuition: Visibility computation or Hidden Surface Removal

Algorithm: The Z-Buffer Algorithm

OpenGL:

glClear(GL_DEPTH_BUFFER_BIT);

glEnable(GL_DEPTH_TEST);