**Taxonomy**

- **Planar Projections**
  - Parallel: Orthographic, Axonometric, Oblique
  - Perspective: One Point, Two Point, Three Point

- **Viewing Projections**
  - Front, Top, Side

- **Planar Projections**
  - One Point, Two Point, Three Point

**The Modeling-Viewing Pipeline**

1. **Object Coordinates**
2. **World Coordinates**
3. **View Coordinates**
4. **Clip Coordinates**
5. **Normalized Device Coordinates**
6. **Device Coordinates**

**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, \( \mathbf{v}_{up} \)

**Defining the VCS**

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.

\[
\mathbf{v}_{up} = \mathbf{v}_{up} \times \mathbf{n} \parallel \mathbf{v}_{up} \times \mathbf{n} \parallel
\]

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

\[
\mathbf{u} = \frac{\mathbf{v}_{up} \times \mathbf{n}}{\|\mathbf{v}_{up} \times \mathbf{n}\|}
\]
From WCS to VCS

The viewing transformation should:

• Map the origin of the WCS, \( \mathbf{O} \), to the Eye, \( \mathbf{E} \).

If the point \( P \) has coordinates \((x, y, z)\) in WCS
and \((u, v, w)\) in VCS
then -

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
\end{bmatrix}
= \begin{bmatrix} u & v & w \end{bmatrix}
\]

\[
\begin{bmatrix}
  e_x & e_y & e_z \\
  u & v & w \\
\end{bmatrix}
\]
From WCS to VCS

The viewing transformation should:

- \( x \rightarrow u \)
- \( y \rightarrow v \)
- \( z \rightarrow n \)

Map the origin of the WCS, \( O \), to the Eye, \( E \).

From WCS to VCS - OpenGL

```cpp
glm::lookAt(glm::vec3 eye, glm::vec3 lookat_pt, glm::vec3 upvec);
```

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:

\[
\begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
-N & 0 & 0 & 1
\end{bmatrix}
\]
If the extents of the near plane are given by $L, R, T, B$ then:

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

$$\frac{(R-L)}{2} \frac{(T-B)}{2} - N$$

Now we scale along $u$ and $v$ so that we get $u = u', v = v'$ as the top side faces of the frustum.

Now we transform the frustum to a canonical frustum.

Now we scale along $u$ and $v$ so that we get $u = u', v = v'$ as the top side faces of the 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.
So the complete transformation is:

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

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.

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.

\[
\begin{bmatrix}
\frac{(x+1)(R-L)}{2} & \frac{(x+1)(T-B)}{2} & \frac{(x+1)}{2} & 0 \\
\frac{(y+1)(R-L)}{2} & \frac{(y+1)(T-B)}{2} & \frac{(y+1)}{2} & 0 \\
\frac{(z+1)(R-L)}{2} & \frac{(z+1)(T-B)}{2} & \frac{(z+1)}{2} & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
\frac{Z}{W} \\
\frac{Y}{W} \\
\frac{X}{W} \\
1 \\
\end{bmatrix}
\]

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

\[
N \text{ and } F \text{ give the distance of Near and Far clipping planes from the Eye and must be positive numbers and must not be equal.}
\]

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

Intuition: Visibility computation or Hidden Surface Removal

Algorithm: The Z-Buffer Algorithm

OpenGL:

\[
glClear(GL_DEPTH_BUFFER_BIT); \\
glEnable(GL_DEPTH_TEST);
\]

Why carry the depth through?