Projection

Computer Graphics
Instructor: Sungkil Lee

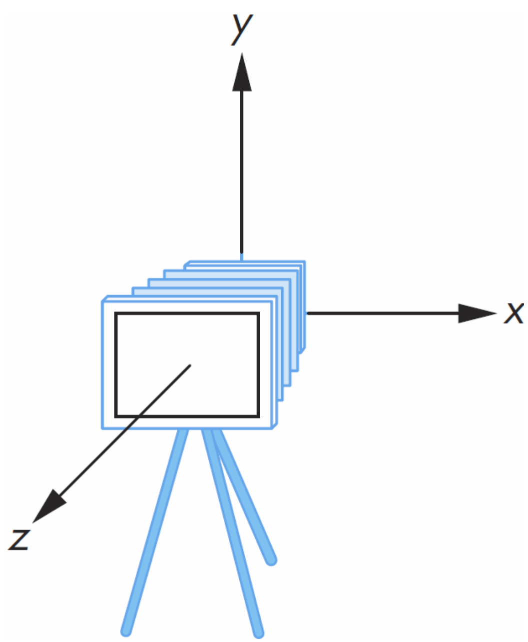
Today

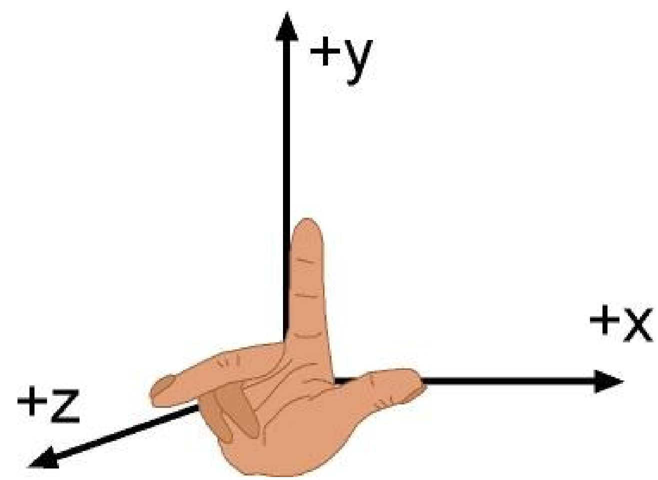
- **Orthographic projection**
- **Perspective projection**
  - Simple perspective projection
  - Symmetric Perspective Projection in OpenGL
  - General Non-Symmetric Perspective Projection in OpenGL

Coordinate Systems Revisited

Camera Convention in OpenGL

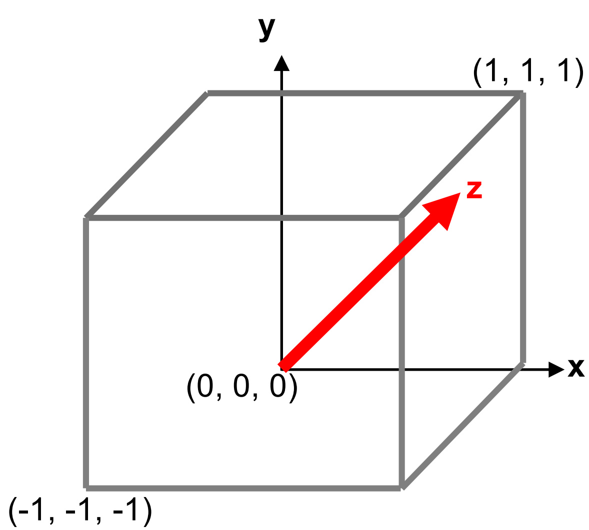
- **In camera space,**
  - a camera is located at the origin, directing in the negative z-direction.
  - camera coordinate systems (frames) use RHS convention.
  - Recall the following figures.





Normalized Device Coordinates in OpenGL

- **Normalized Device Coordinate (*NDC*) System**
  - Through projections for OpenGL, the camera-space points are transformed to NDC.
  - Recall that: NDC uses LHS convention: z-axis goes far from your eye
  - This is intended for **depth test**, which maintains objects with the smallest depths as visible.



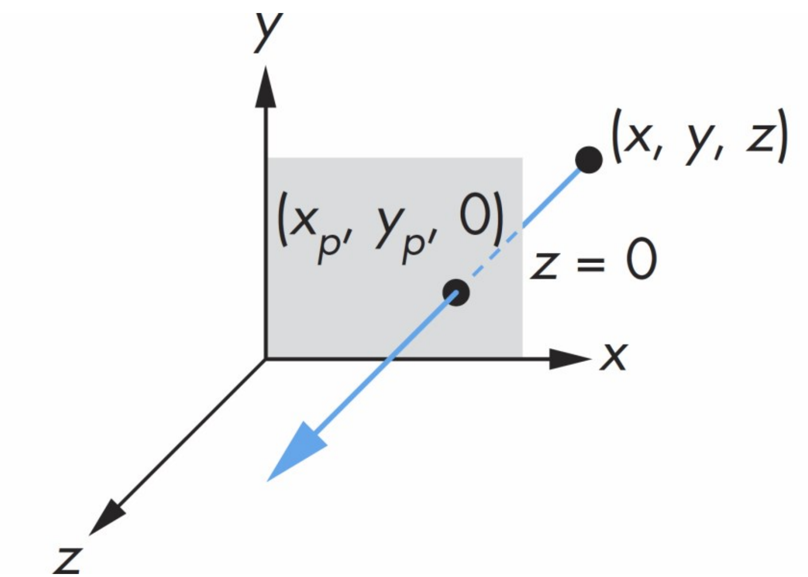
Orthographic Projection Matrix

Parallel Projection

- **A parallel projection is the limit of a perspective projection**
  - We send a COP to the infinity; that is, DOP.
- **However, we will derive the equations for parallel projections directly using the fact that the projectors are parallel.**
  - rather than deriving the equations for a perspective projection and computing their limiting behavior.

Orthographic Projection

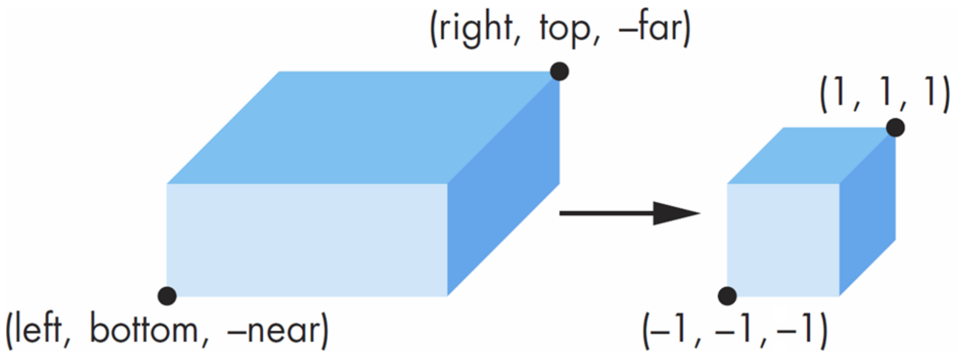
- **A special case of the parallel projection,**
  - in which the projectors are perpendicular to the projection plane
  - corresponds to a camera which has an infinite focal length.
  - Orthogonal projection with the projection plane z=0
    - x and y are retained, but only z_p is set to 0.



$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Orthographic Projection in OpenGL

Simple orthographic projection + View volume normalization



- **Orthographic projection in OpenGL**
  - View volume normalization (VVN): all the 3D points are normalized into canonical view volume $[-1,1]^3$ ().
  - **We never explicitly set $\mathbf{z = 0}$**, and retain depth information as long as possible in the OpenGL pipeline for **depth test/buffering**.
  - That means, even after projection, we still stay in 4D homogeneous coordinates.

View Volume Normalization?

- **Why we do not discard depth after projection?**
  - We still need depths to perform **depth test** (before/after fragment shading) in the pipeline.
  - For this, the depth is typically normalized in a fixed range (e.g., [0,1])

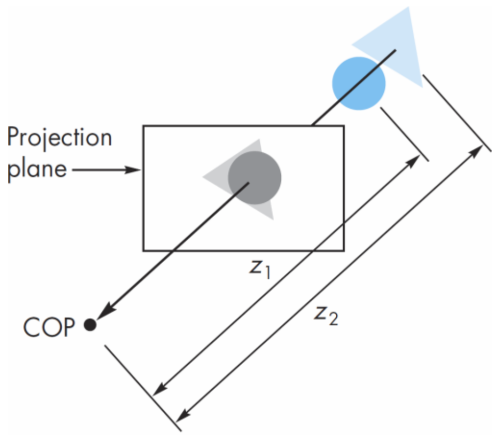


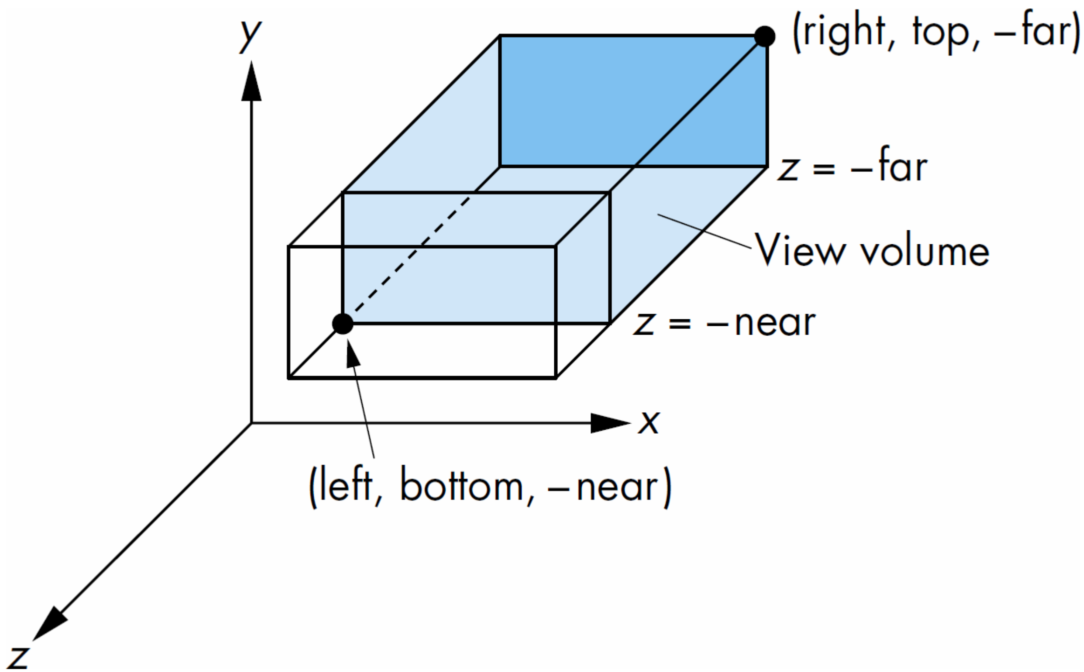
FIGURE 4.41 The z-buffer algorithm.

- **Another benefit of VVN: *efficient clipping***
  - VVN lets us **clip against simple cube** regardless of type of projection, including both perspective and orthogonal viewing.

Orthographic Projection in OpenGL

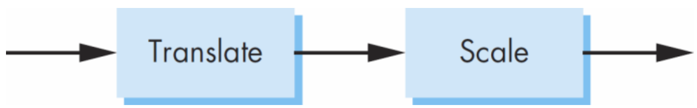
- **We need to implement a function:**
  - near and far are depths of clipping planes (i.e., depth bounds)
  - $\mathbf{0 < \boldsymbol{near} < \boldsymbol{far}}$: both are the ***positive*** distances from camera (COP).

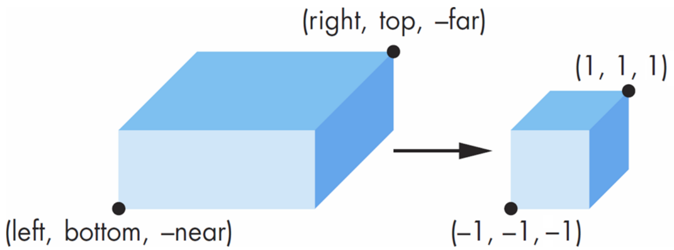
```
mat4 Ortho(left, right, bottom, top, near, far)
```



Orthographic Projection Matrix

- **Two-step view volume normalization**





  - (1) Translation: move center to the origin

$$\mathbf{T} = \mathbf{T}\left(-\frac{left + right}{2}, -\frac{bottom + top}{2}, +\frac{near + far}{2}\right)$$

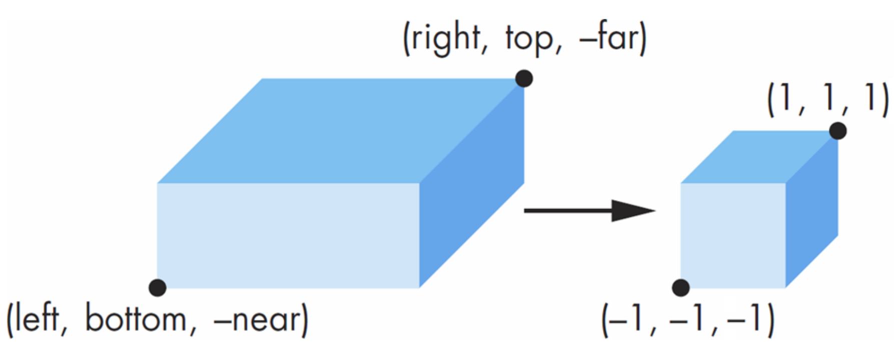
  - (2) Scale to have sides of length 2

$$\mathbf{S} = \mathbf{S}\left(\frac{2}{right - left}, \frac{2}{top - bottom}, \frac{2}{\boldsymbol{near - far}}\right)$$

Orthographic Projection Matrix

- **The combined form of view volume normalization matrix:**
  - (l, r, t, b, n, f) := (left, right, top, bottom, near, far)

$$
\boldsymbol{P} = \boldsymbol{ST} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{l+r}{l-r} \\ 0 & \frac{2}{t-b} & 0 & \frac{b+t}{b-t} \\ 0 & 0 & \frac{2}{n-f} & \frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

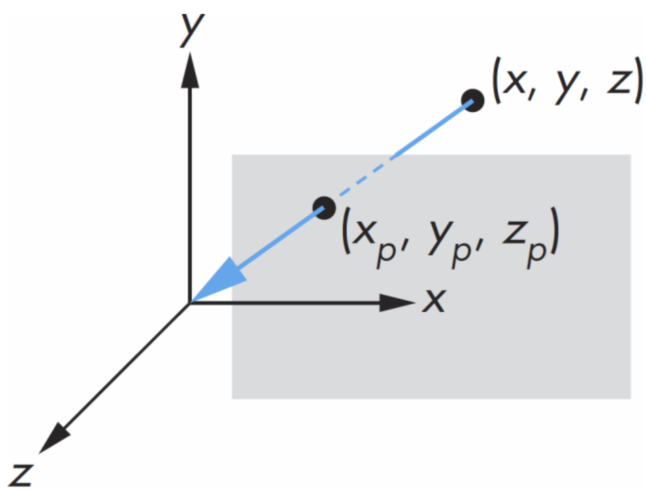


Simple Perspective Projection Matrix

Simple Perspective Projection

- **Our approach:**
  - Again, we first consider the mathematics for a simple projection and extend it to the projection for OpenGL, including view volume normalization (VVN).

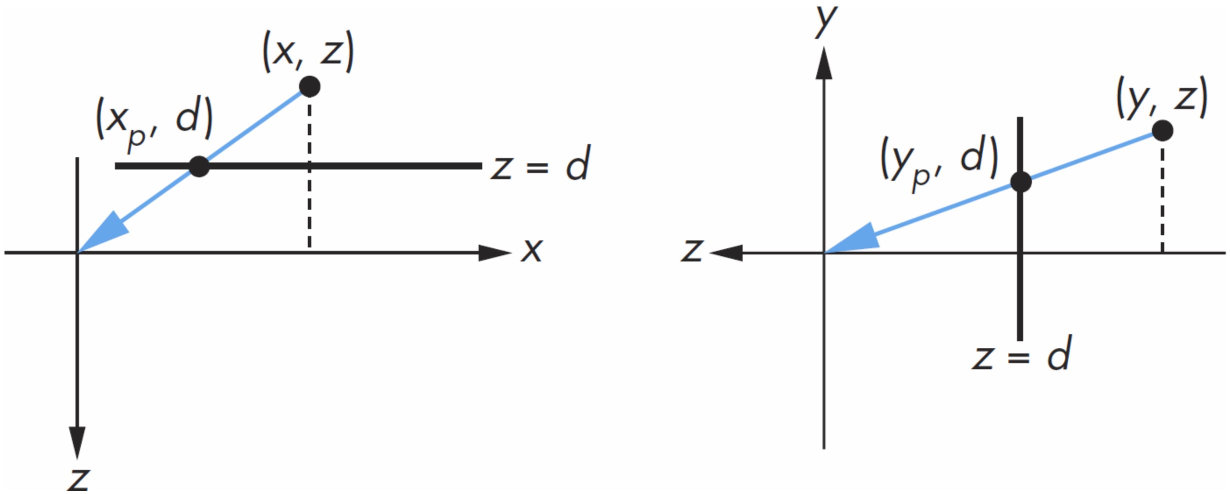
- **Here, first catch the idea of simple perspective projection.**
  - Set the projection plane in front of the COP.
  - The point (x, y, z) is projected into the point (x_p, y_p, z_p).
  - All the projectors pass through the origin (COP).



Simple Perspective Projection

- **Set projection plane as orthogonal to z-axis at:**

$$z_p = d, \text{ where } d < 0$$



  - Then, we have a relationship which describes nonlinear foreshortening:

$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d}$$

Simple Perspective Projection

- **The homogeneous coordinate form:**

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

  - which makes $\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ to $\mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$

Simple Perspective Projection

- **Perspective division:**

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

  - Since $q_w \neq 1$, we must divide by w to return to the Cartesian coordinates. This *perspective division* yields the desired result:

$$x_p = \frac{x}{z/d},$$

$$y_p = \frac{y}{z/d},$$

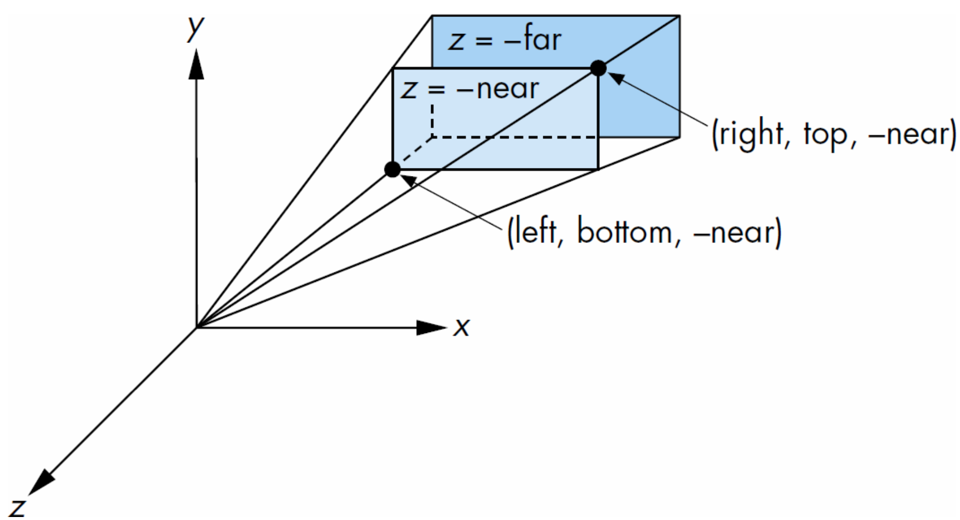
$$z_p = \frac{z}{z/d} = d$$

Symmetric Perspective Projection in OpenGL

Perspective Projection in OpenGL

Simple perspective projection + View volume normalization

- Perspective projection in OpenGL again includes view volume normalization.
- For simplicity, we only consider a symmetric view volume ($\boldsymbol{r} = -\boldsymbol{l}, \boldsymbol{b} = -\boldsymbol{t}$) for the same near and far clipping planes.

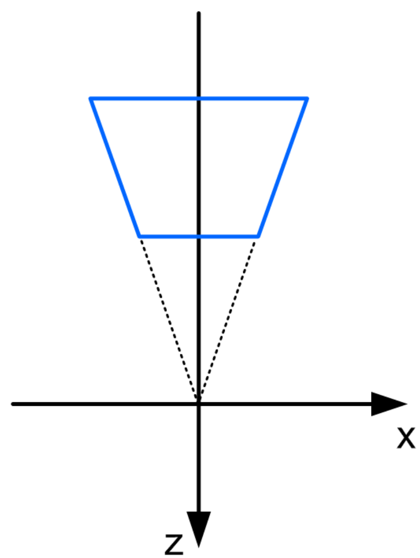


Perspective Projection Matrix

(1) the first step is scaling the frustum to $x = \mp z, y = \mp z,$

- The symmetric input frustum is described as:

$$\boxed{x = \mp \frac{r}{n} z, \qquad y = \mp \frac{t}{n} z, \qquad -f \le z \le -n}$$



Perspective Projection Matrix

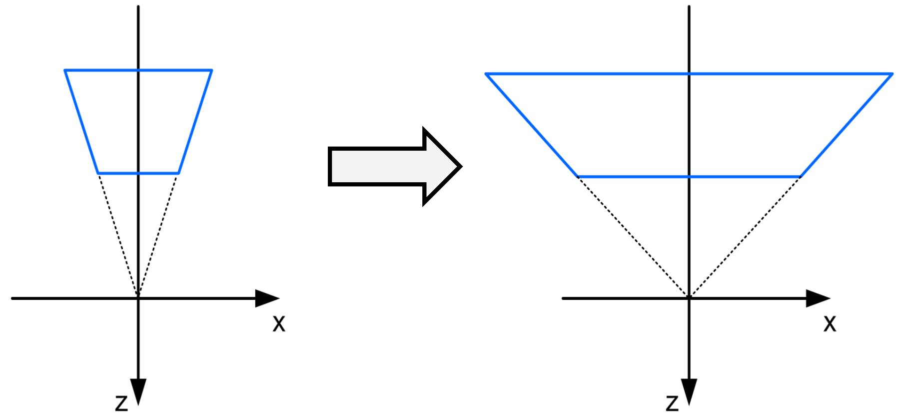
(1) the first step is scaling the frustum to $x = \mp z, y = \mp z,$

- given the symmetric input frustum

$$x = \mp \frac{r}{n} z, \qquad y = \mp \frac{t}{n} z, \qquad -f \le z \le -n$$

- the scaling matrix can be:

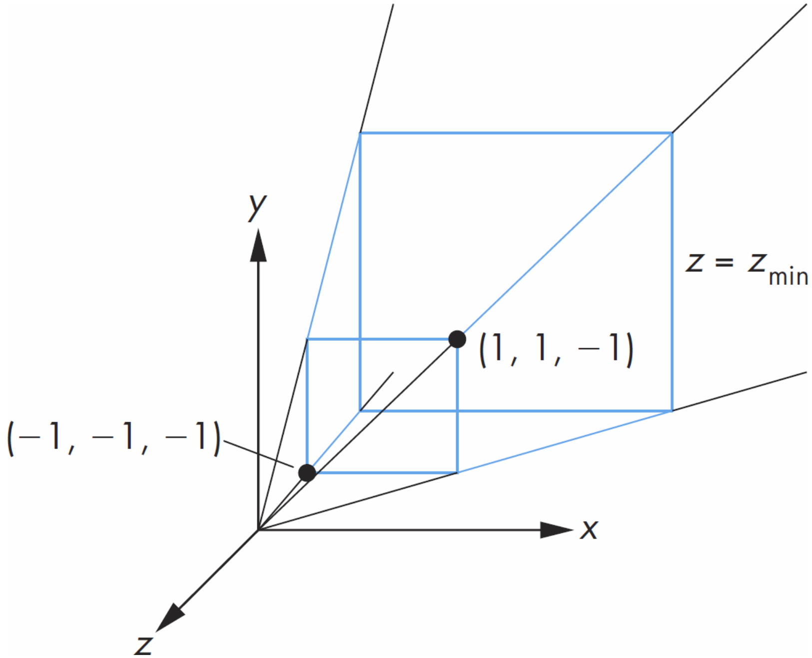
$$\mathbf{S} = \begin{bmatrix} n/r & 0 & 0 & 0 \\ 0 & n/t & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Perspective Projection Matrix

(2) secondly, perform the simple perspective projection with depth normalization for the view volume:

- the near clipping plane at $z = -1$,
- determined by the planes $x = \mp z,\ y = \mp z$ (scaled in the 1st step).

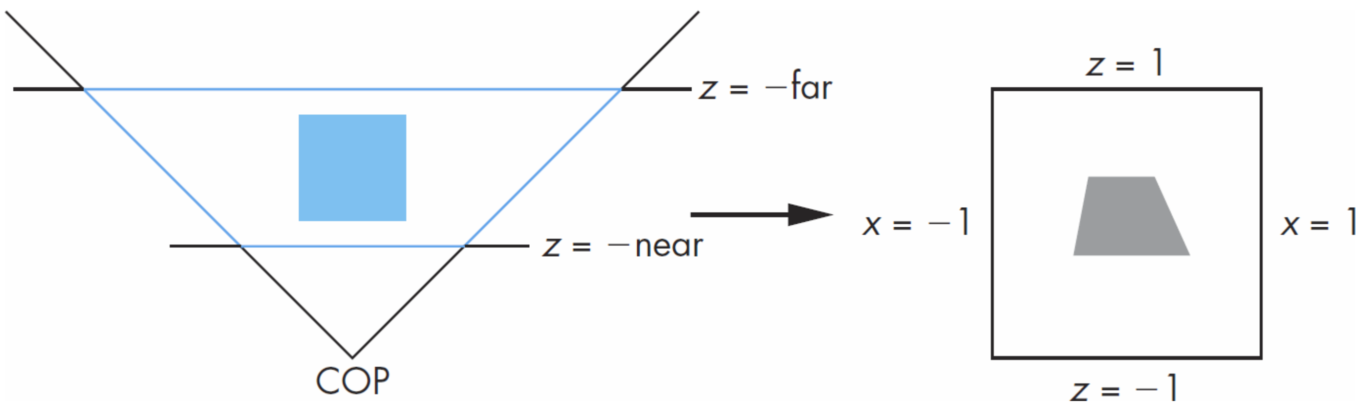


Perspective Depth Normalization

- **To normalize depth, we need a transformation like:**

$$\boxed{z = -near \rightarrow z = -1, \qquad z = -far \rightarrow z = 1}$$

  - Note that the signs of depths are inverted during this conversion, which is intended to maintain the ordering of depths from the viewer.



Perspective Depth Normalization

- **Consider a matrix form,**
  - modified from a simple perspective projection (was $\alpha = 1, \beta = 0$),

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

  - After perspective division, the point $(x, y, z, 1)$ goes to

$$(x', y', z') = \left(-\frac{x}{z}, -\frac{y}{z}, -\left(\alpha + \frac{\beta}{z}\right)\right)$$

- **For xy clipping planes ($x = \mp z, y = \mp z$),**
  - $(x', y') \in [-1,1]^2$ is now *normalized in the canonical view volume* (CVV).
  - What remains is normalizing depth clipping planes to CVV.

Perspective Depth Normalization

- **For near and far clipping planes:**
  - $z = -near$, and $z = -far$ are transformed to:

$$z' = -\left(\alpha - \frac{\beta}{near}\right) \quad \text{and} \quad z' = -\left(\alpha - \frac{\beta}{far}\right)$$

  - Picking α and β such that $z' = -1$, and $z' = 1$, and solving the equations

$$-1 = -\left(\alpha - \frac{\beta}{near}\right) \quad \text{and} \quad 1 = -\left(\alpha - \frac{\beta}{far}\right)$$

  - gives us:

$$\boldsymbol{\alpha = \frac{near+far}{near-far}}, \quad \boldsymbol{\beta = \frac{2 \cdot near \cdot far}{near-far}}$$

Perspective Depth Normalization

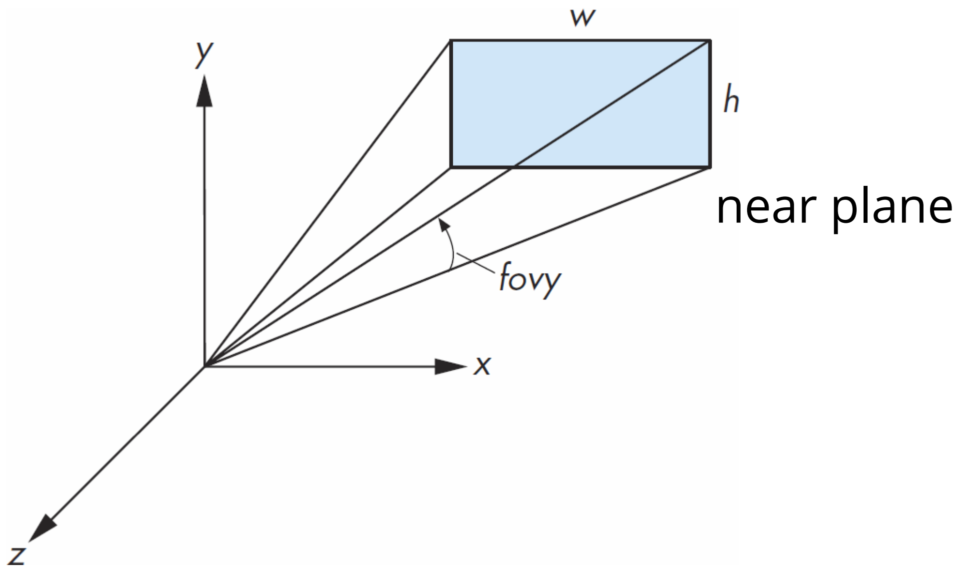
- **Final form:**
  - Then, the final form of symmetric perspective projection matrix (including both scaling and normalization) is:

$$\mathbf{P} = \mathbf{NS} = \begin{bmatrix} n/r & 0 & 0 & 0 \\ 0 & n/t & 0 & 0 \\ 0 & 0 & \dfrac{n+f}{n-f} & \dfrac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Alternative: Perspective()

- **Having l, r, t, b is often difficult due to their measurement.**
  - perspective() method provides more intuitive/controllable projection.
  - $fovy$ (field of view) and $aspect\_ratio$ (width/height of sensor):
    - 'y' in fovy: vertical

```
mat4 perspective(fovy, aspect_ratio, near, far)
```



$$t = n * \tan(\frac{fovy}{2}) \qquad r = t * aspect\_ratio \qquad \cot\left(\frac{fovx}{2}\right) = \cot\left(\frac{fovy}{2}\right) / aspect\_ratio$$

Alternative: Perspective()

```
mat4 perspective(fovy, aspect_ratio, near, far)
```

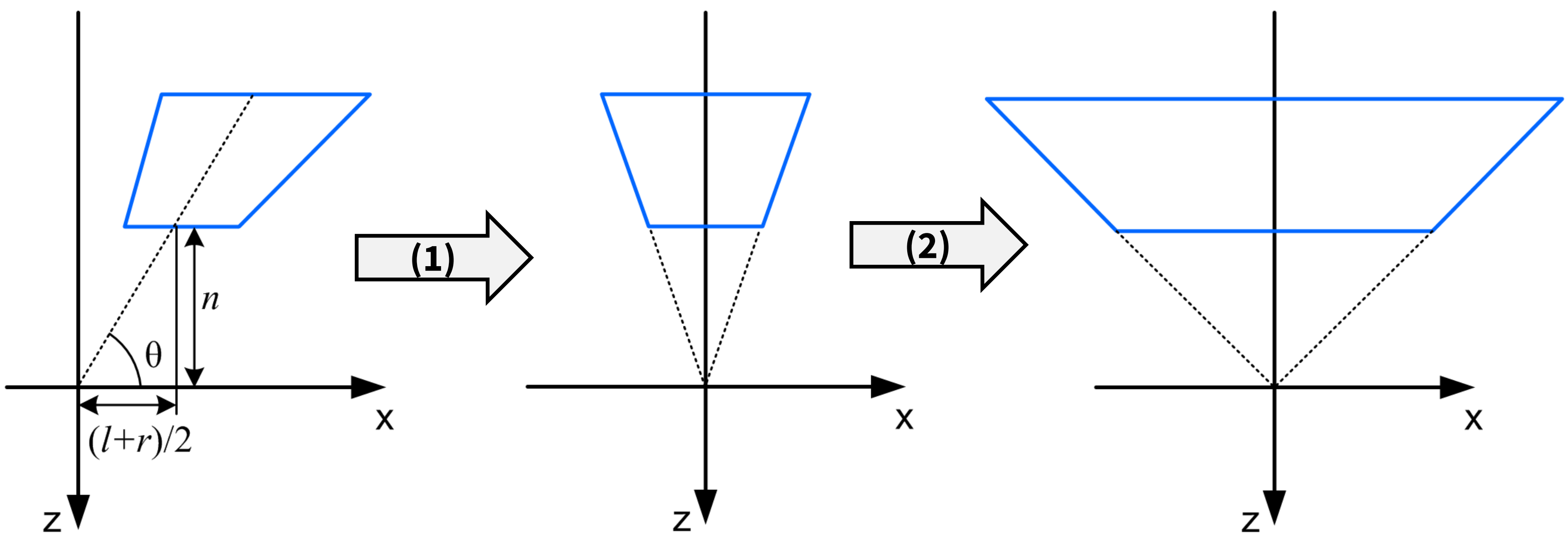
- Then, we have an alternative representation.
- This shape is more common than the frustum (i.e., LRBTNF).

$$\mathbf{P} = \begin{bmatrix} \cot\left(\frac{fovx}{2}\right) & 0 & 0 & 0 \\ 0 & \cot(\frac{fovy}{2}) & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Appendix: General (Non-Symmetric) Perspective Projection in OpenGL

General Perspective Projection in OpenGL

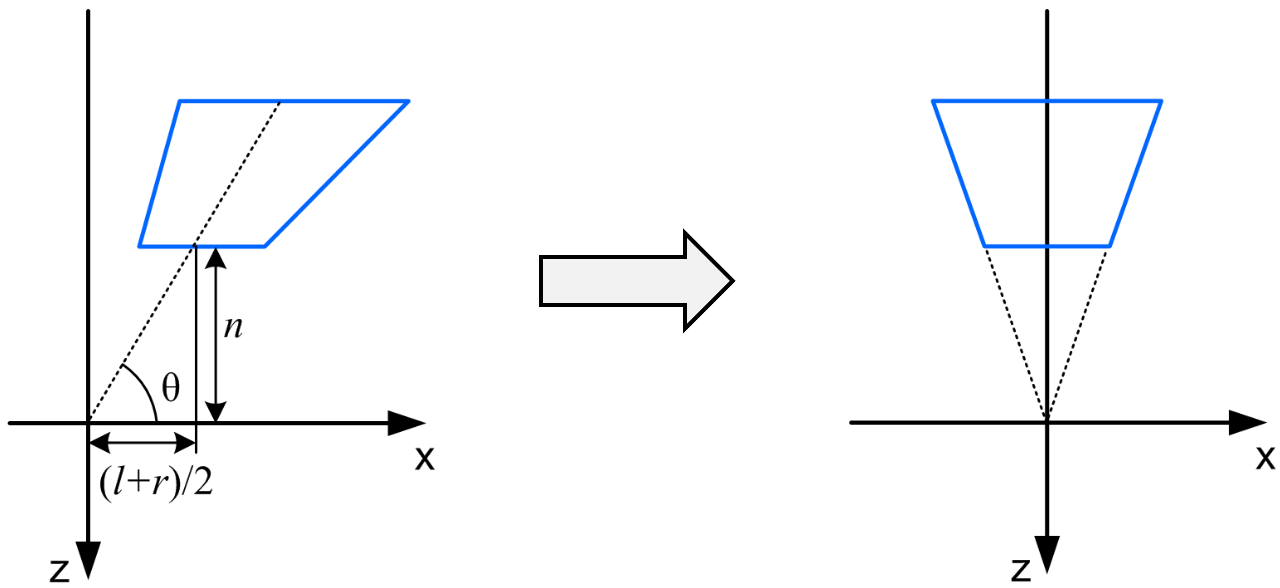
- **Three steps:**
  (1) shear transform to make the symmetric volume (new, here!)
  (2) scale the sides of the frustum to $x = \pm z$, $y = \pm z$
  (3) simple perspective projection with *perspective depth normalization*.



General Perspective Projection Matrix

(1) shear transform to make the symmetric volume

- See the top view for x shear (also, do the same for y shear)



General Perspective Projection Matrix

(1) shear transform to make the symmetric volume

- For the shear transform, we skew the points

$$(\frac{l+r}{2}, \frac{t+b}{2}, -n) \text{ to } (0,0,-n) \quad \text{and} \quad (\frac{(l+r)f}{2n}, \frac{(t+b)f}{2n}, -f) \text{ to } (0,0,-f)$$

- General cases (z < 0) can be written as :

$$(-\frac{(l+r)z}{2n}, -\frac{(t+b)z}{2n}, z) \text{ to } (0,0,z)$$

- Then, the shear transform matrix (note the inverted signs) :

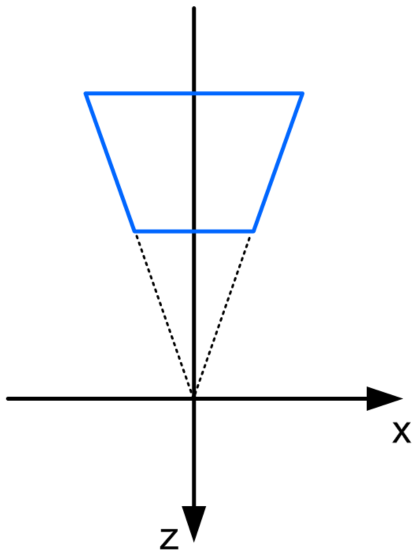
$$\mathbf{H} = \begin{bmatrix} 1 & 0 & (l+r)/2n & 0 \\ 0 & 1 & (t+b)/2n & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

General Perspective Projection Matrix

(1) shear transform to make the symmetric volume

- The resulting frustum is:

$$x = \mp\frac{(r-l)}{2n}z, y = \mp\frac{(t-b)}{2n}z, -f \leq z \leq -n$$



General Perspective Projection Matrix

- **Final form of general perspective projection matrix:**
  - multiplying a skew matrix H leads to the final form of:

$$\mathbf{P} = \mathbf{NSH} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$