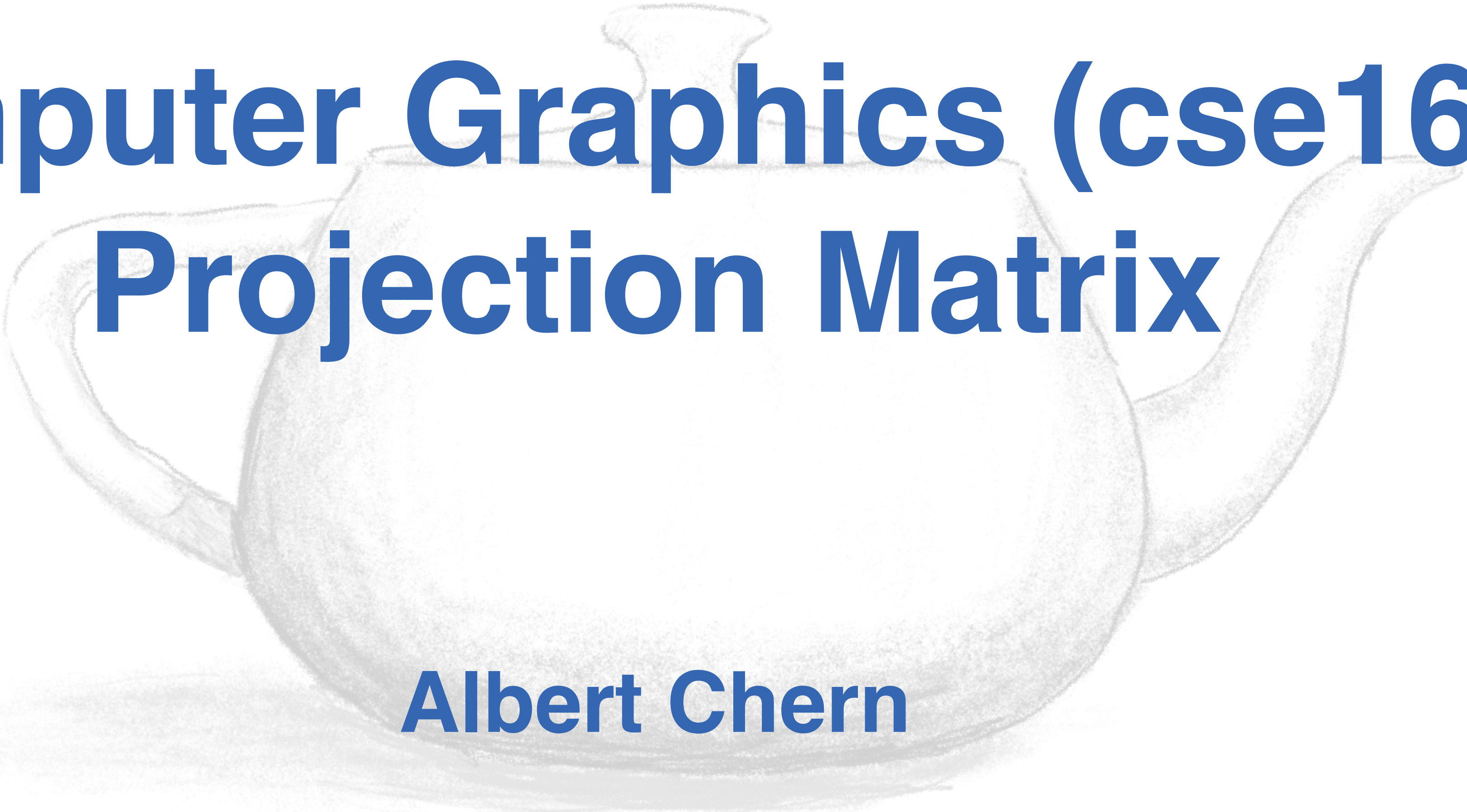




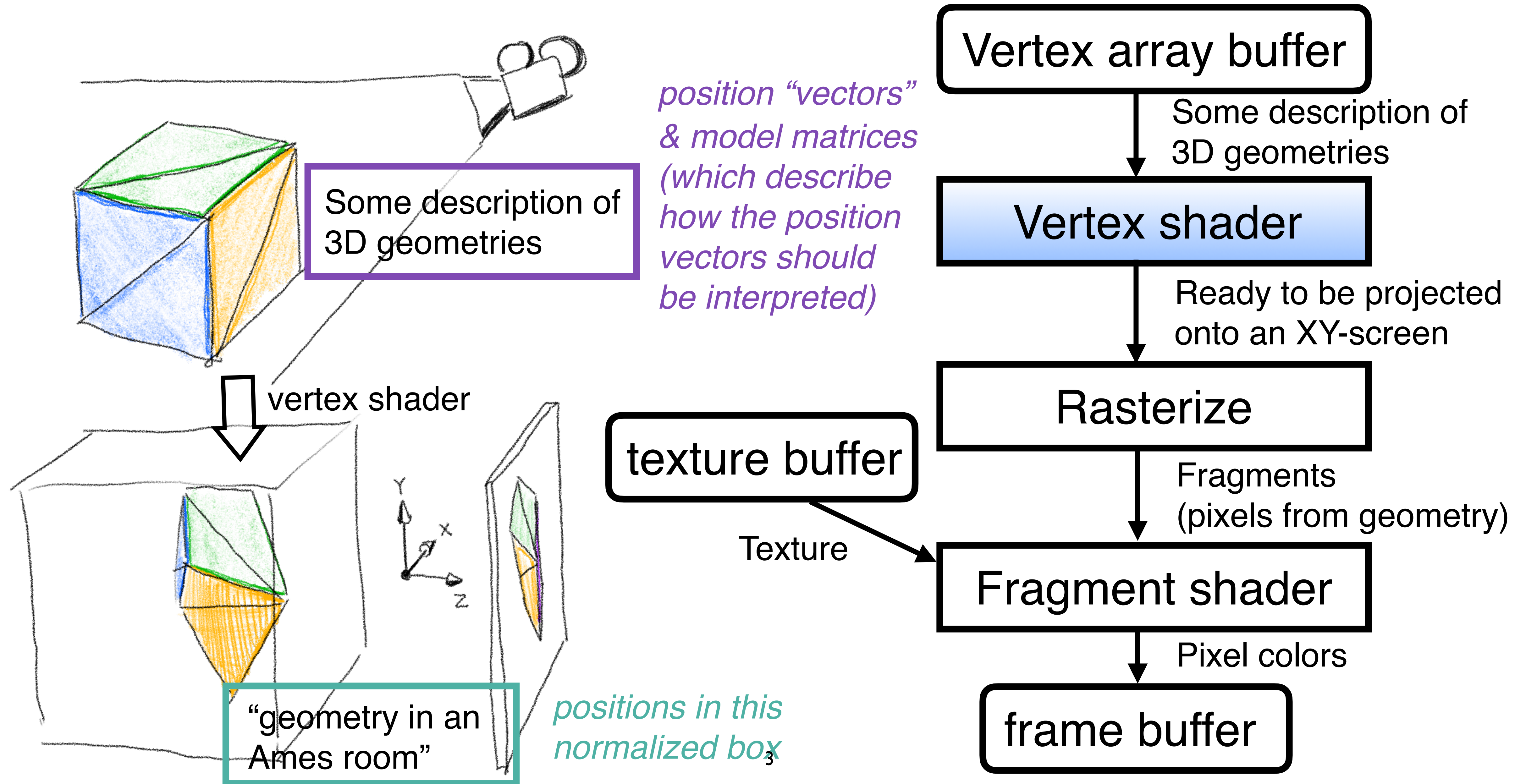
# Computer Graphics (cse167)

## Projection Matrix

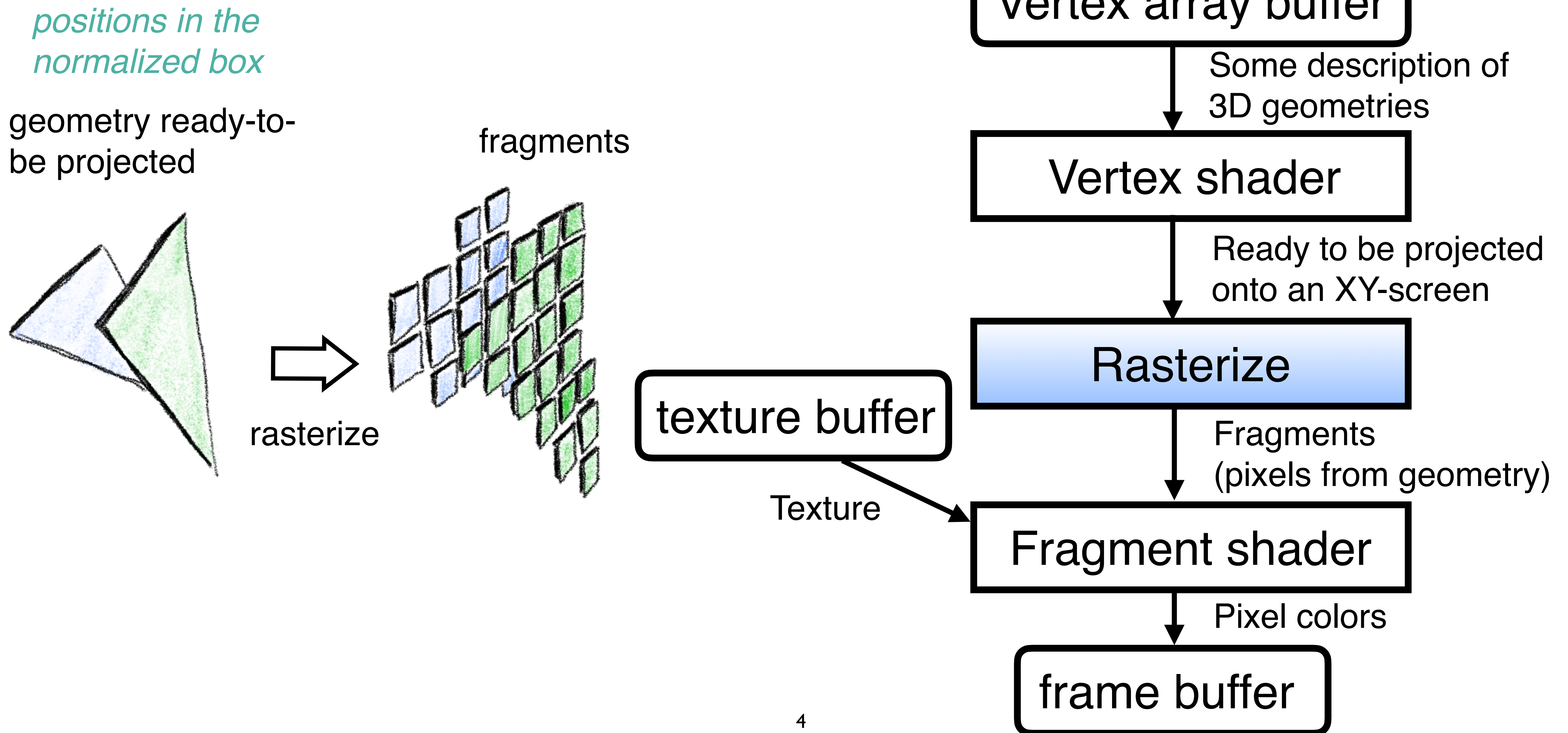
**Albert Chern**

# Camera Coordinate & Normalized Camera Space

# Transformation stage

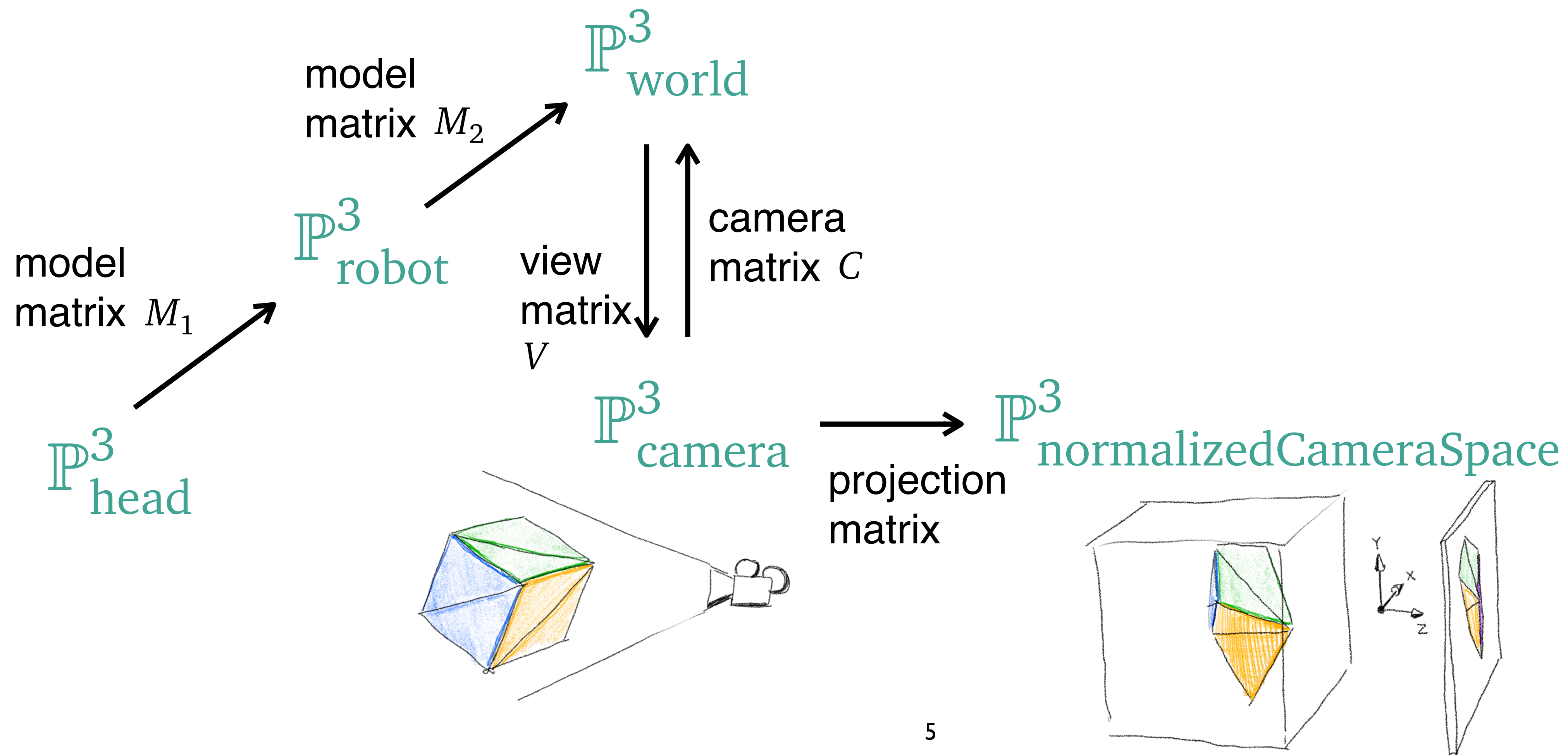


# Rasterization

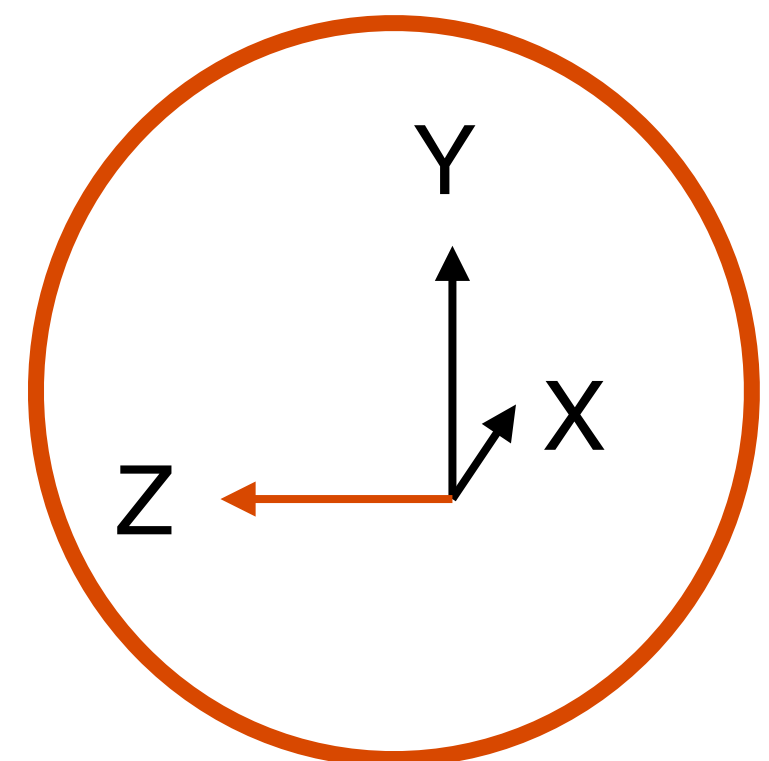


# Transformations and spaces

- The model matrices and the view matrix are affine.
- The projection matrix is projective.

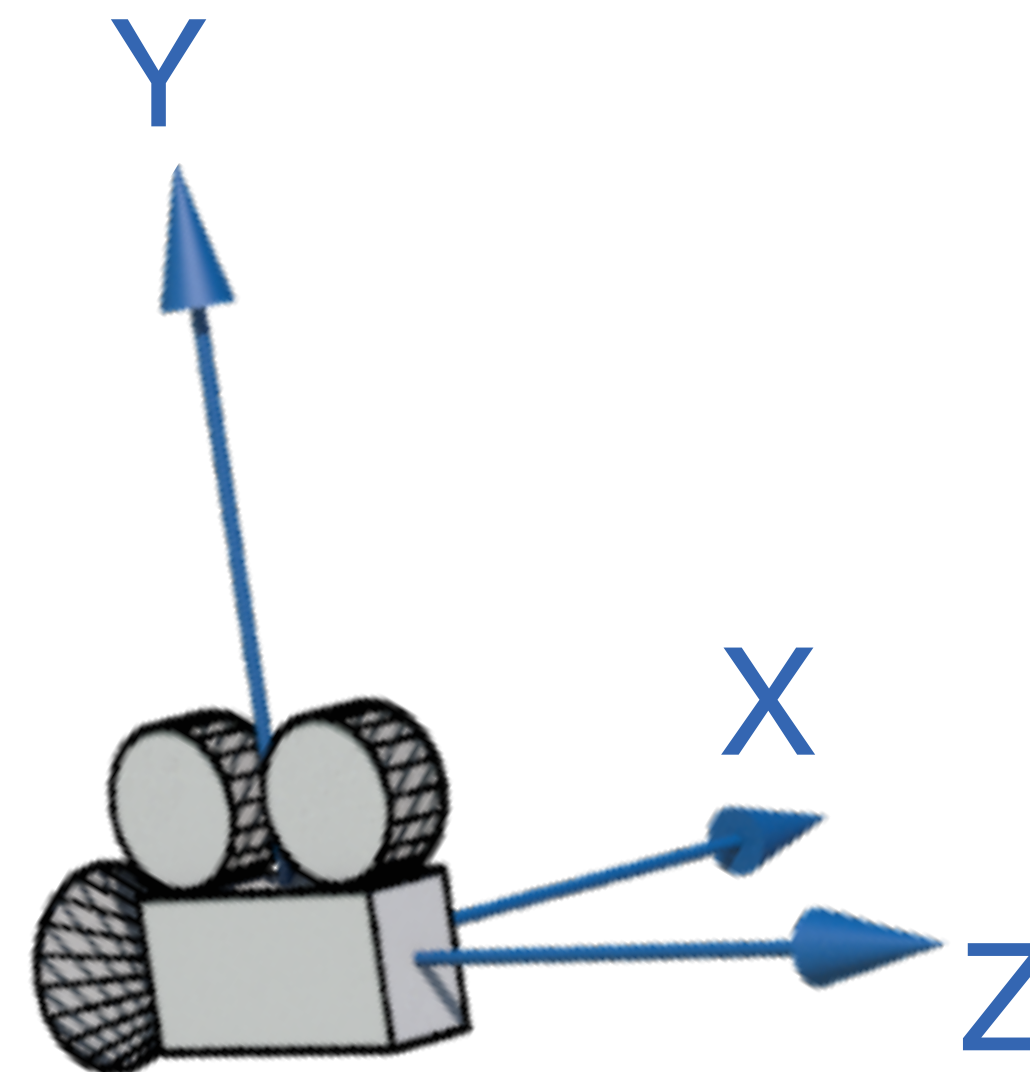


OpenGL  
normalized  
device  
coordinates  
is actually:



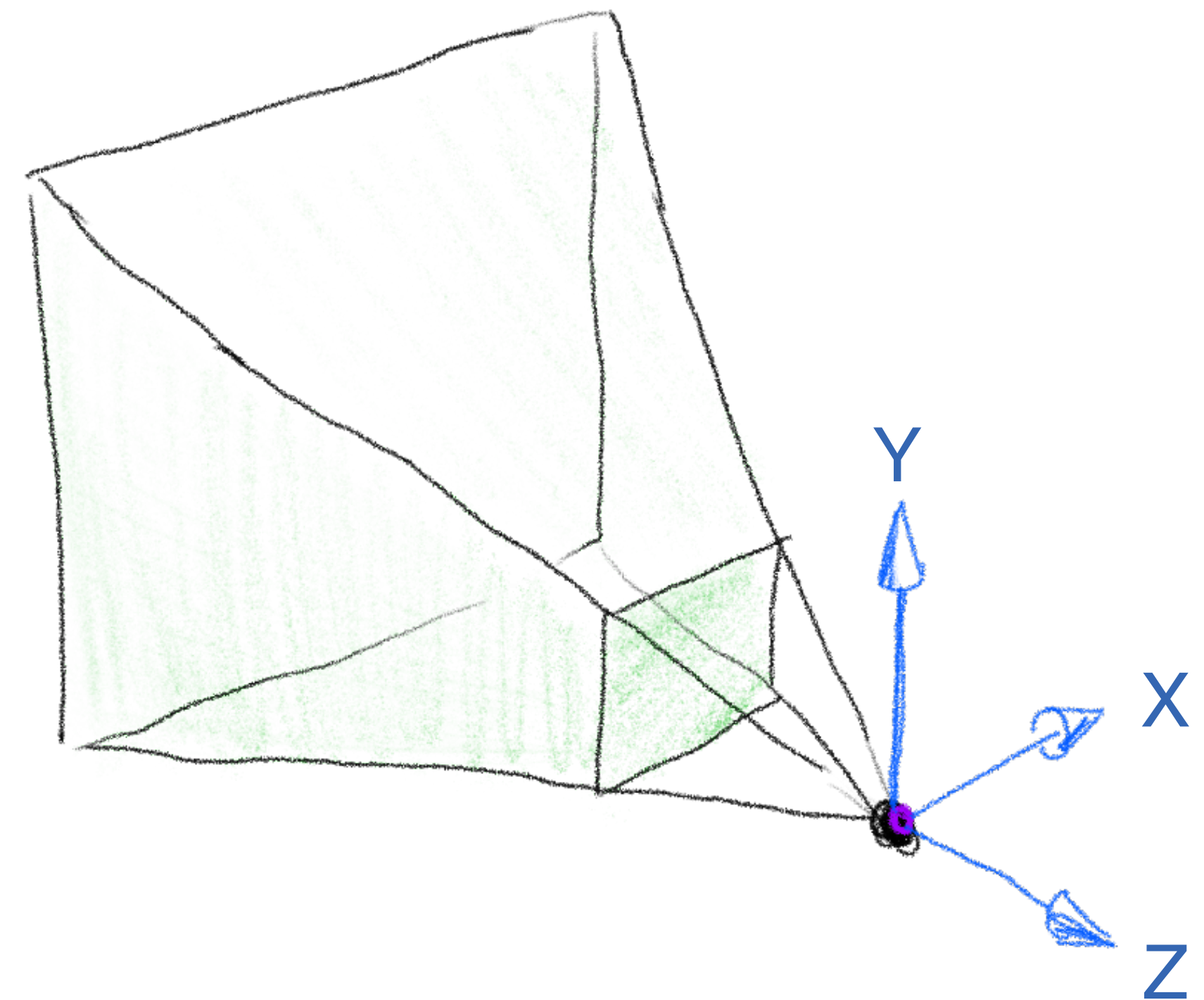
# Camera coordinate

- $\mathbb{P}_{\text{camera}}^3$  uses the camera coordinate system
  - ▶ The camera is at the origin of this coordinate system.
  - ▶ The camera looks towards the -Z axis.
  - ▶ The up vector is the plus Y axis.
  - ▶ Multiplying the [view\*model] matrix is to express positions under the camera coordinate.



# Camera coordinate

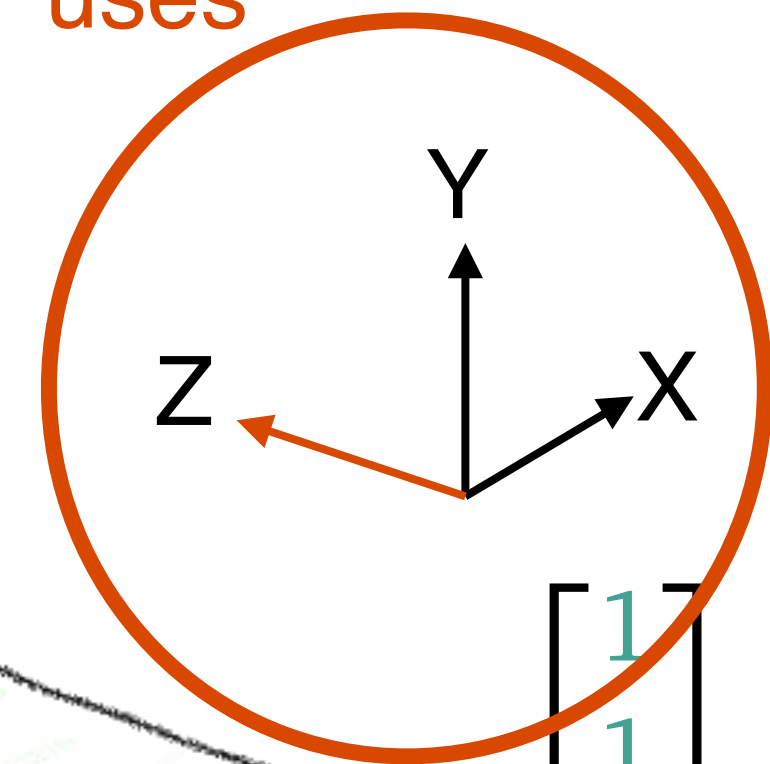
- With some camera parameters, we draw the **viewing frustum** in  $\mathbb{P}_{\text{camera}}^3$ 
  - ▶ The viewing frustum is the visible region of the camera.
  - ▶ The viewing frustum is a rectangular cone (apex at origin) clipped by a far-plane and a near plane.
  - ▶ Light ray connecting to the camera are lines passing through the origin (and intersecting the frustum).



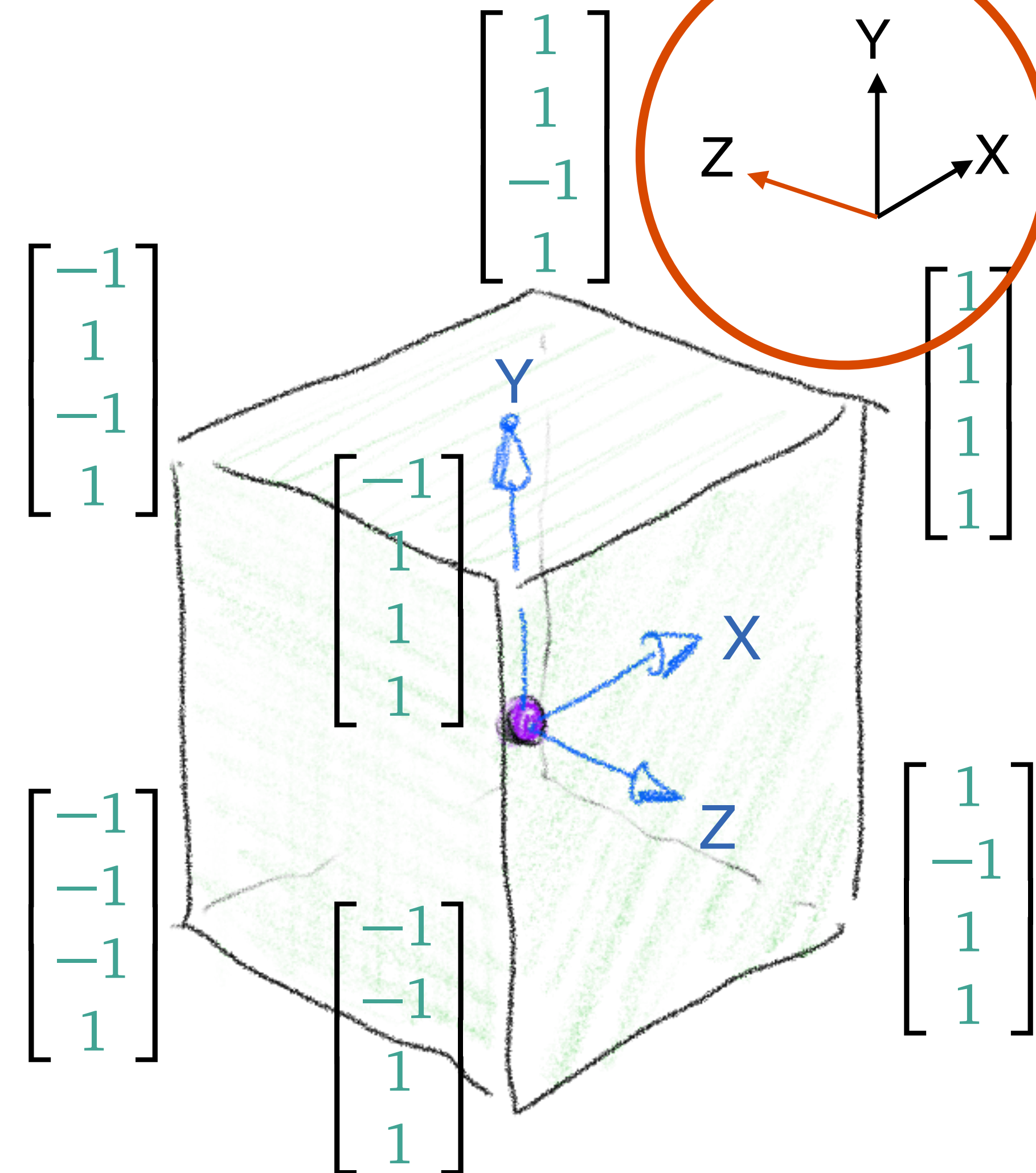
# Normalized camera space

- $\mathbb{P}^3$  `normalizedCameraSpace` is the coordinate system used by the rasterizer.

OpenGL  
actually  
uses

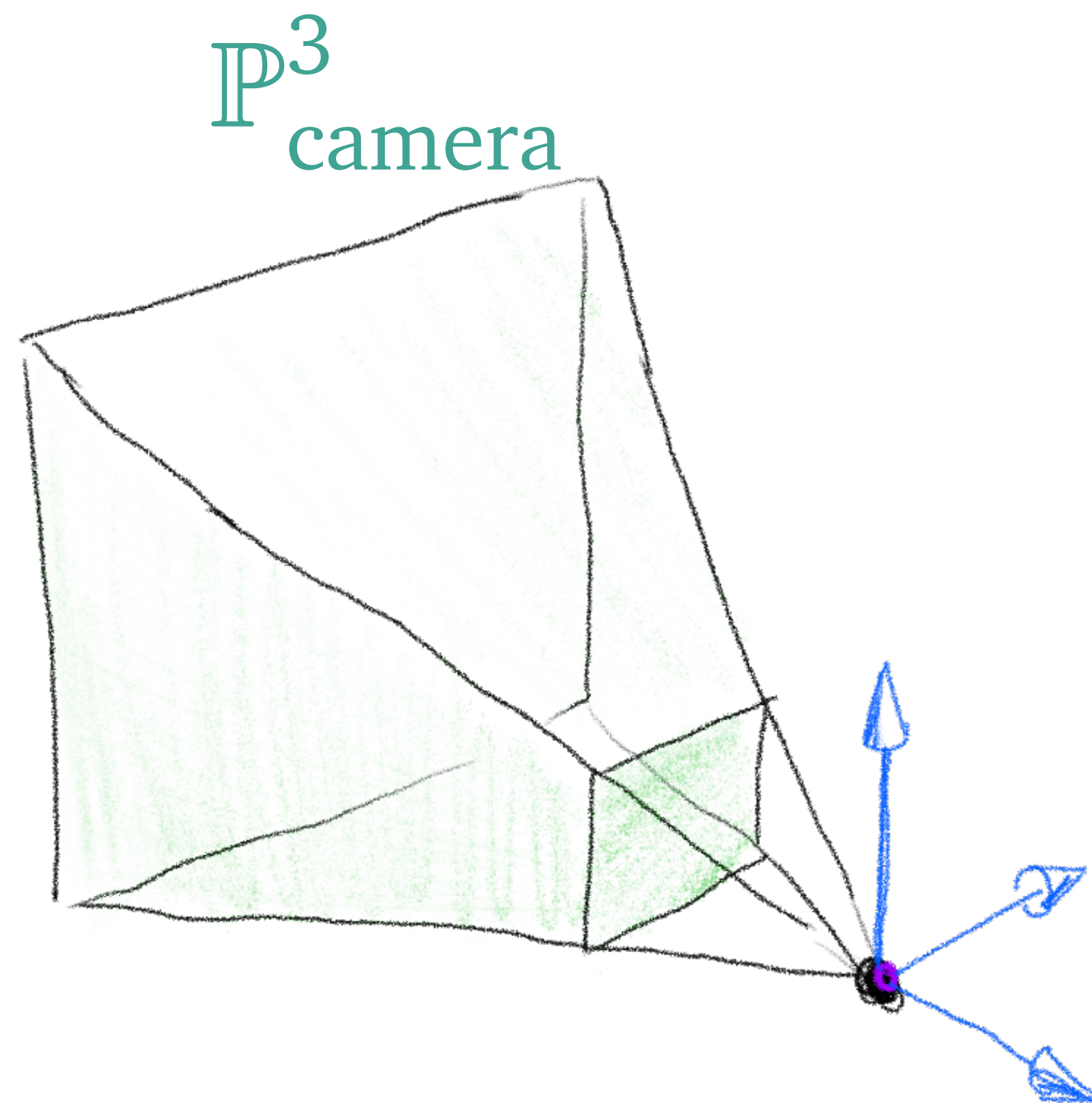


- ▶ The rasterizer will rasterize along the Z direction
- ▶ The rasterizer only rasterizes geometry in the cube centered at the origin of sidelength 2. This cube is also known as the **clipping box**.
- ▶ The light rays connecting to the camera are lines parallel to the Z axis. That is, camera is at the infinity point  $(0 : 0 : 1 : 0)$
- ▶ The Z component of each geometry is an information of the relative closeness to the camera (the larger the closer)



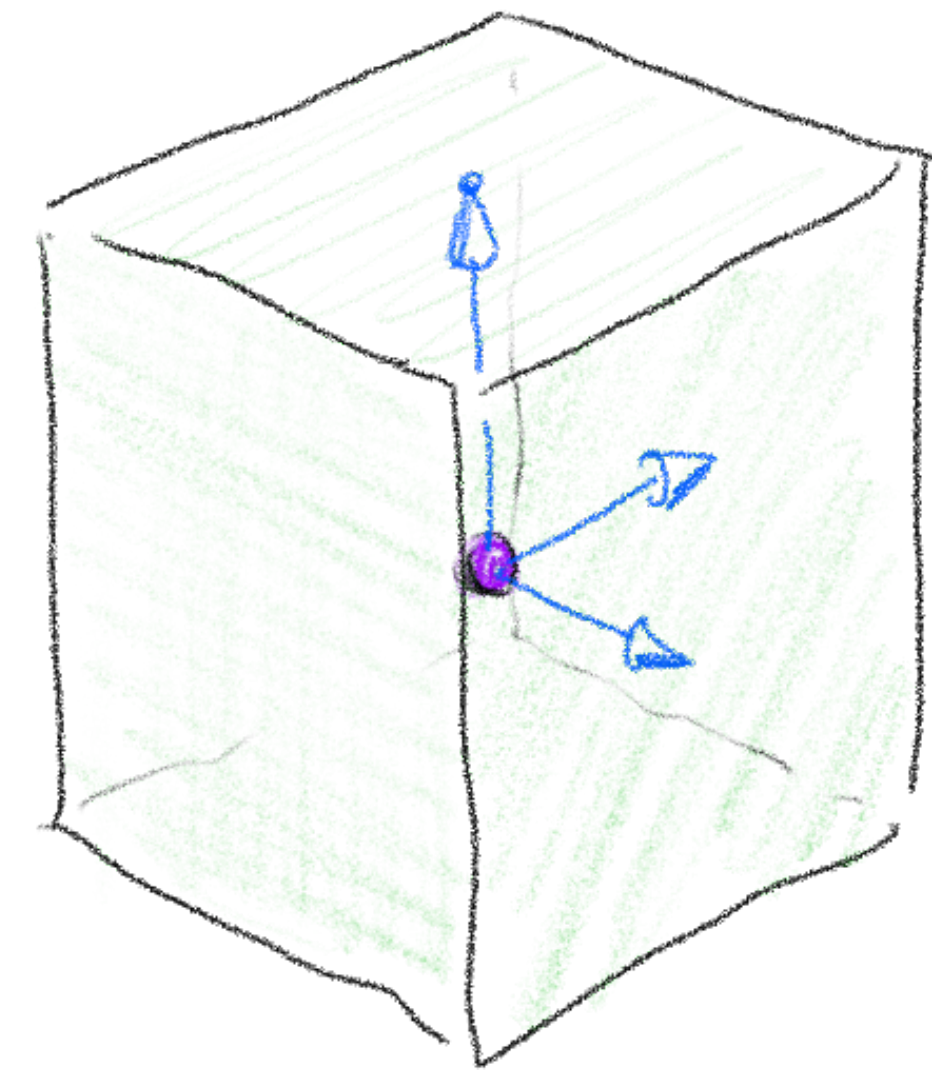
# Projective Transformation

- Given a viewing frustum in the camera coordinate, there is a unique projective transformation that maps the frustum to the clipping box, and maps lines through the origin to lines parallel to Z.

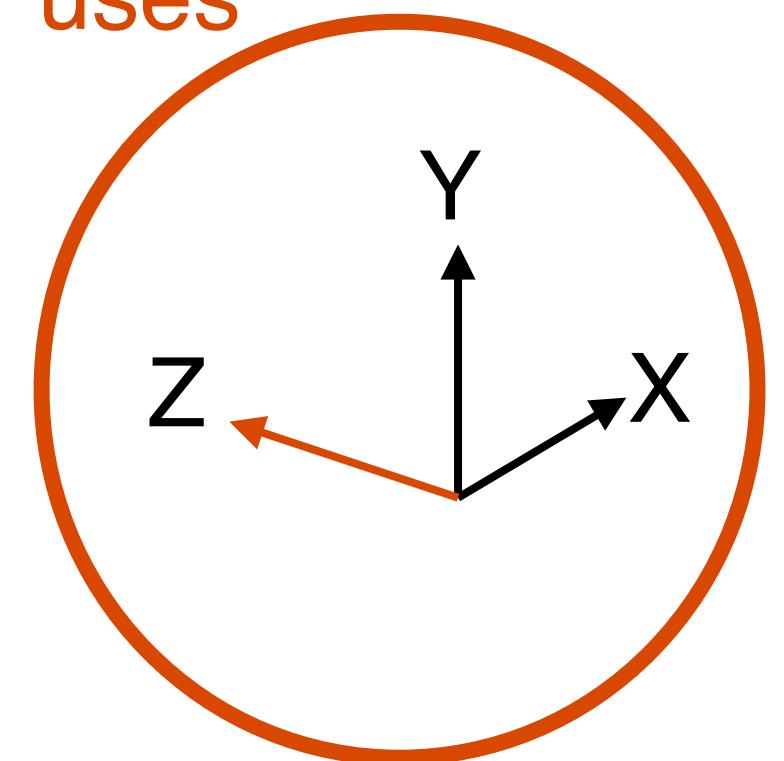


→  
projection  
matrix

$\mathbb{P}^3_{\text{normalizedCameraSpace}}$



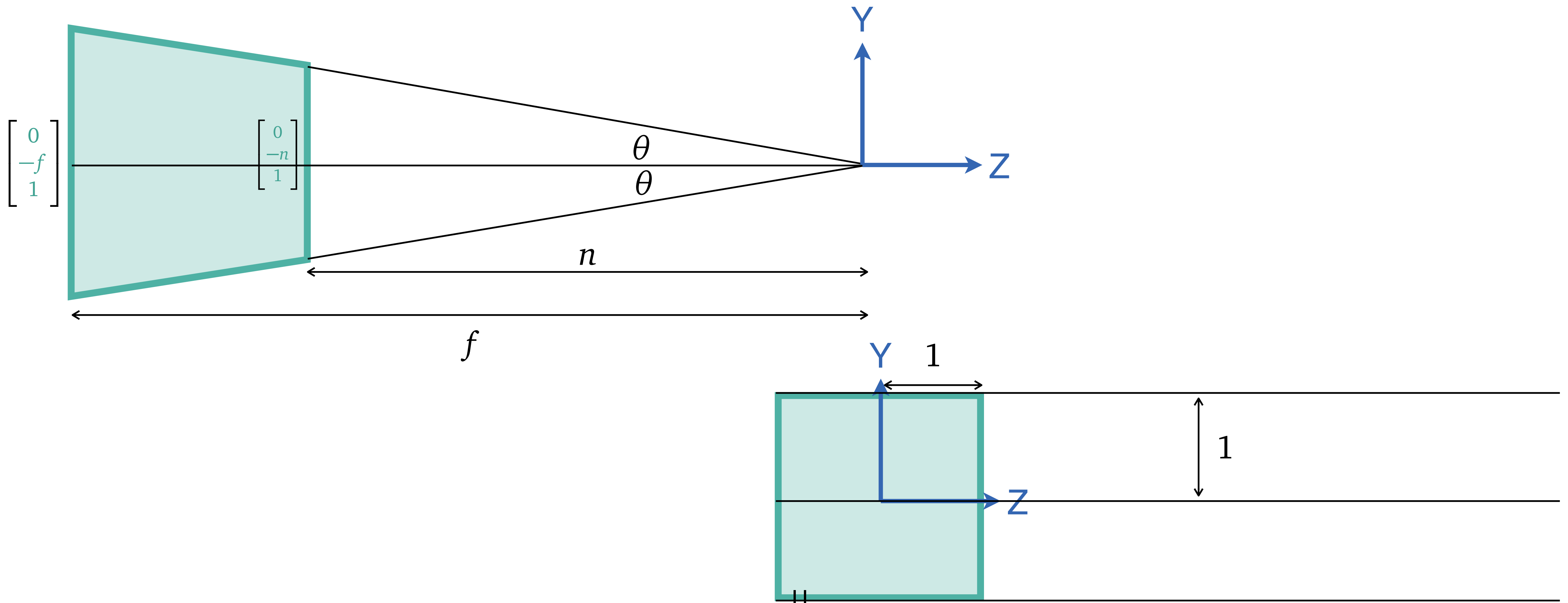
OpenGL  
actually  
uses



# Trapezoid to Square

# Trapezoid to square

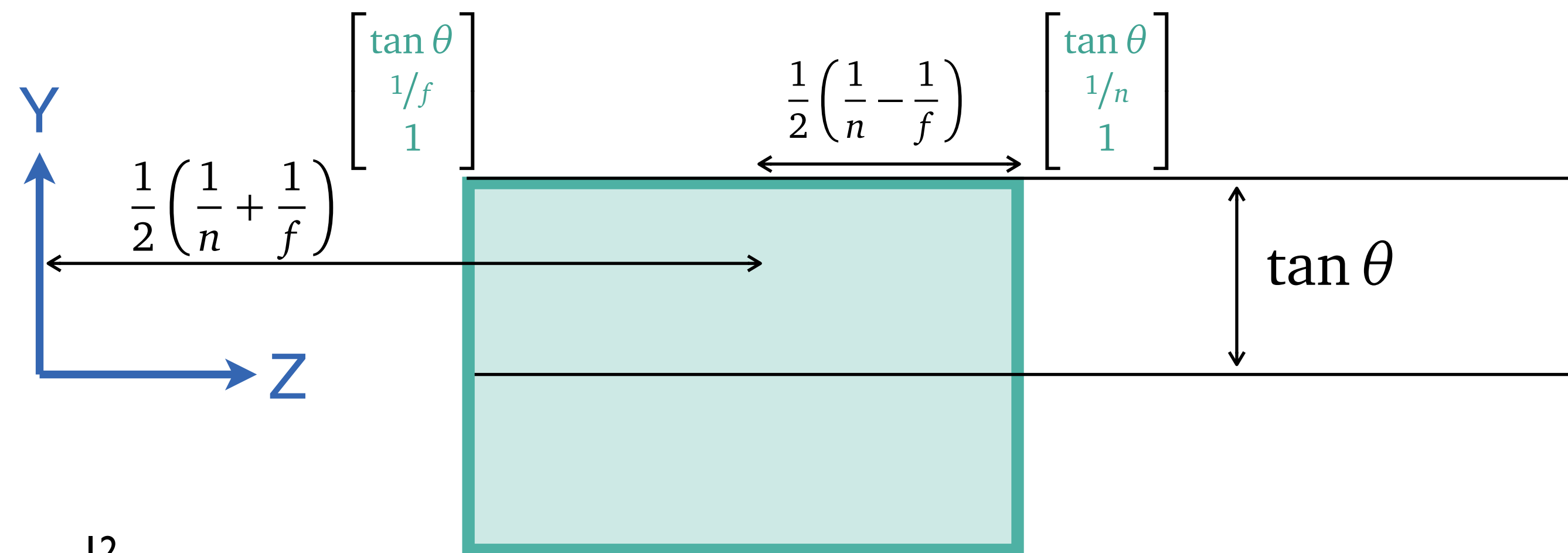
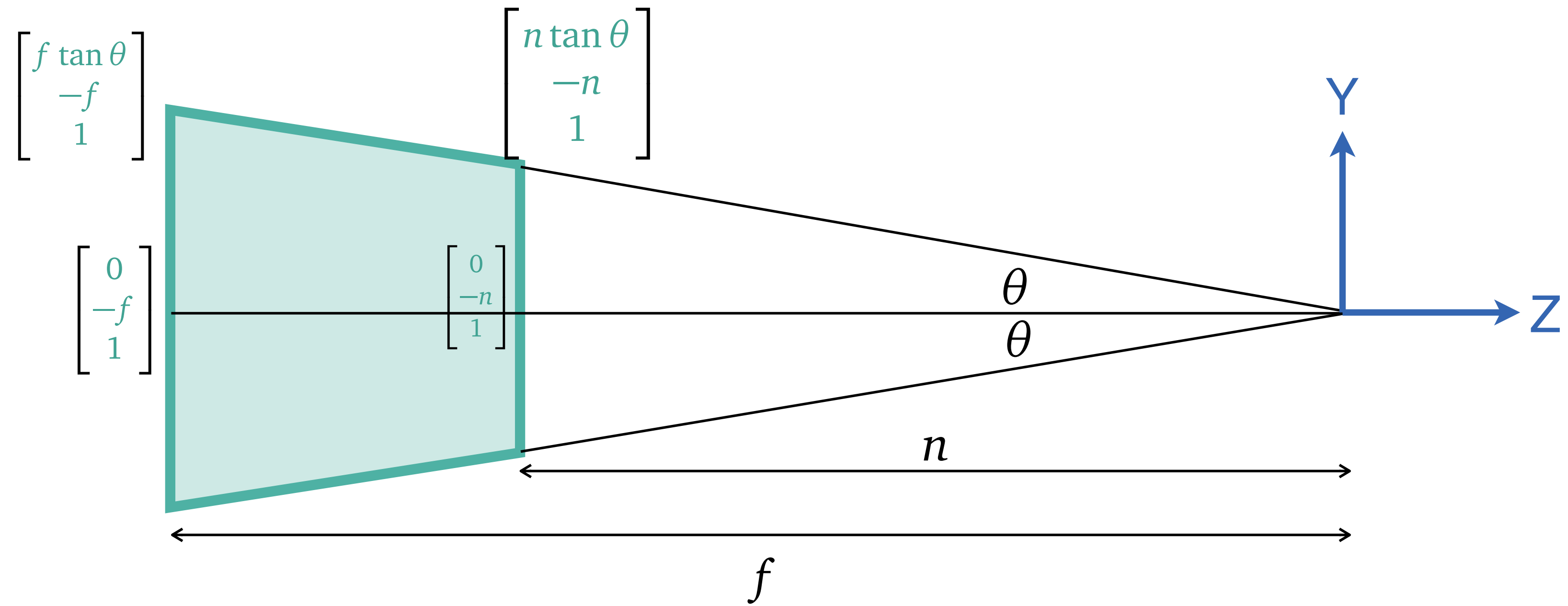
- Design a 3x3 matrix (applied to homogeneous coordinates) that deforms the trapezoid to the square.



# Trapezoid to square

- Inversion

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} y \\ 1 \\ -z \end{bmatrix} \sim \begin{bmatrix} -y/z \\ -1/z \\ 1 \end{bmatrix}$$

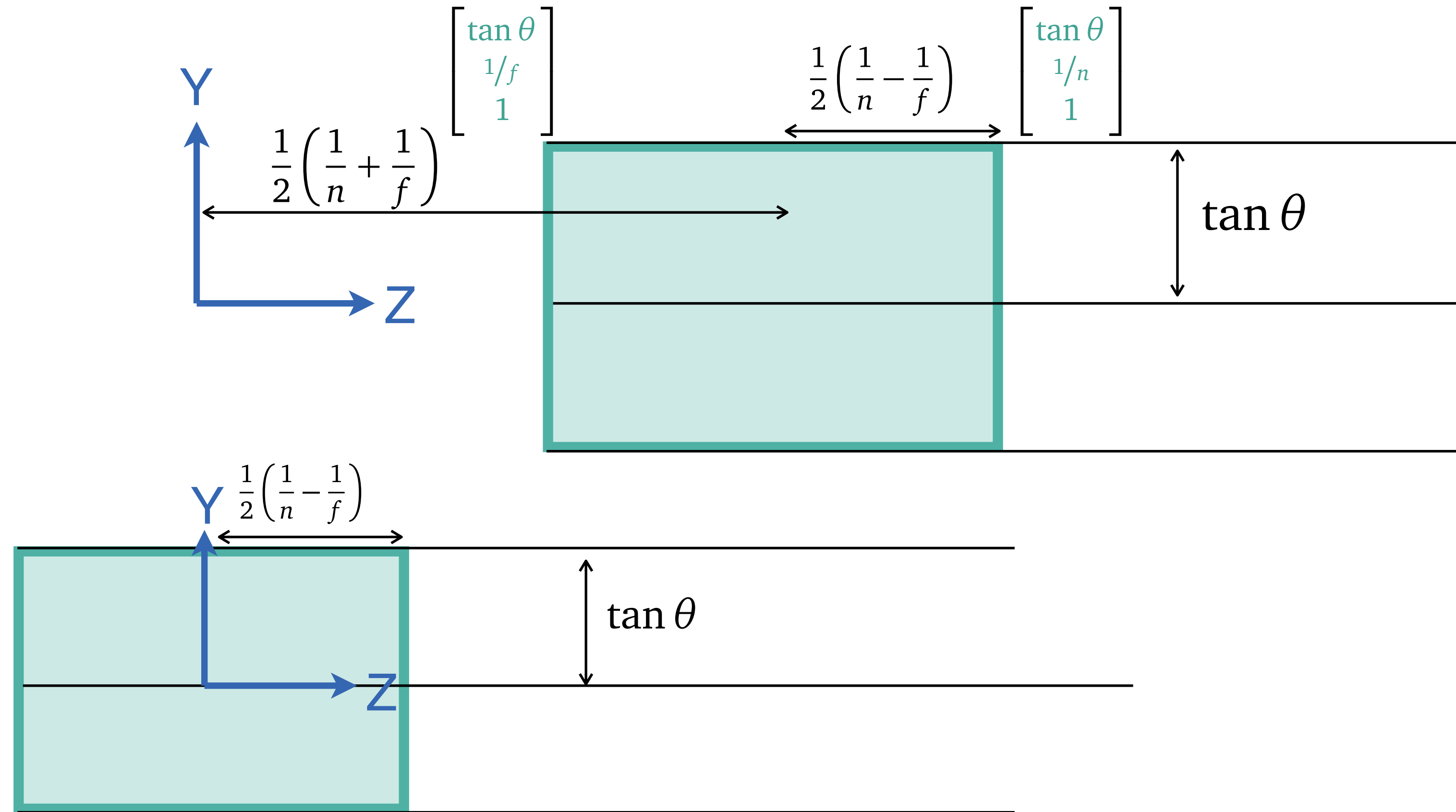


# Trapezoid to square

- Inversion

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} y \\ 1 \\ -z \end{bmatrix} \sim \begin{bmatrix} -y/z \\ -1/z \\ 1 \end{bmatrix}$$

- Translation  $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\frac{f+n}{2fn} \\ 0 & 0 & 1 \end{bmatrix}$



# Trapezoid to square

- Inversion

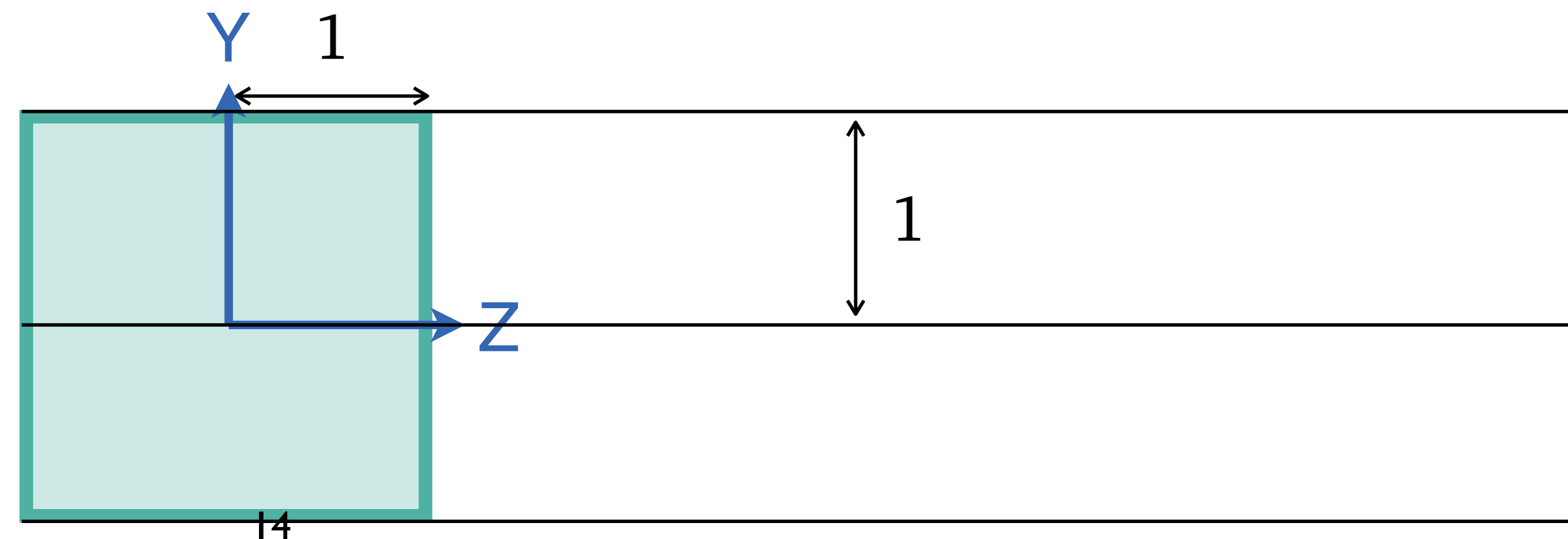
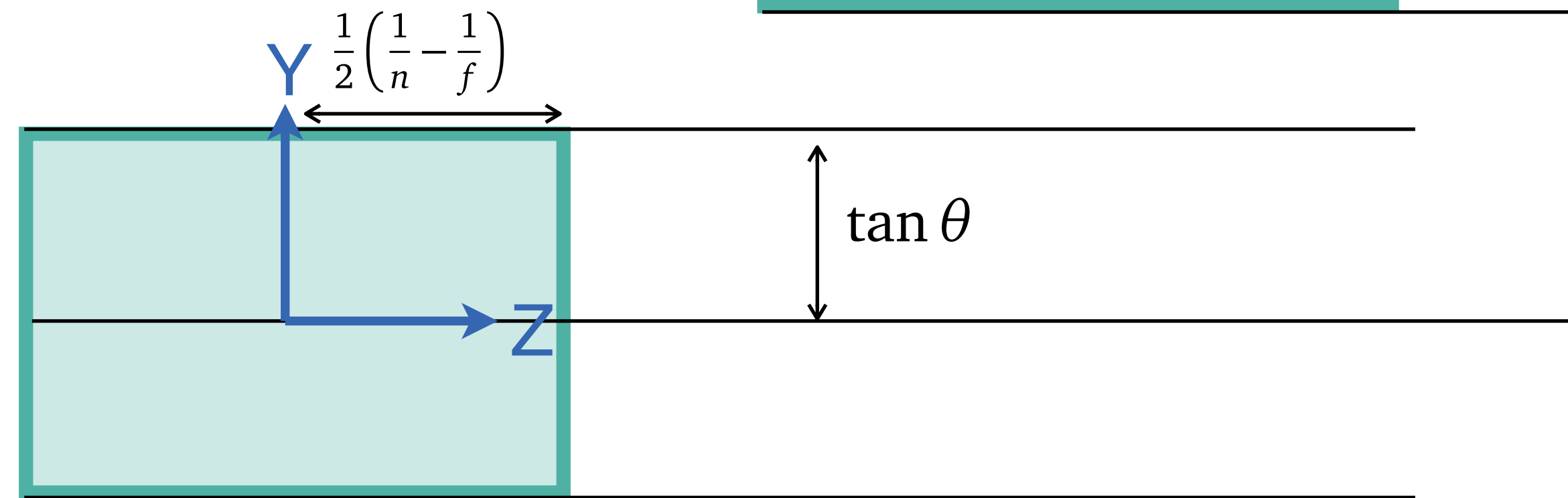
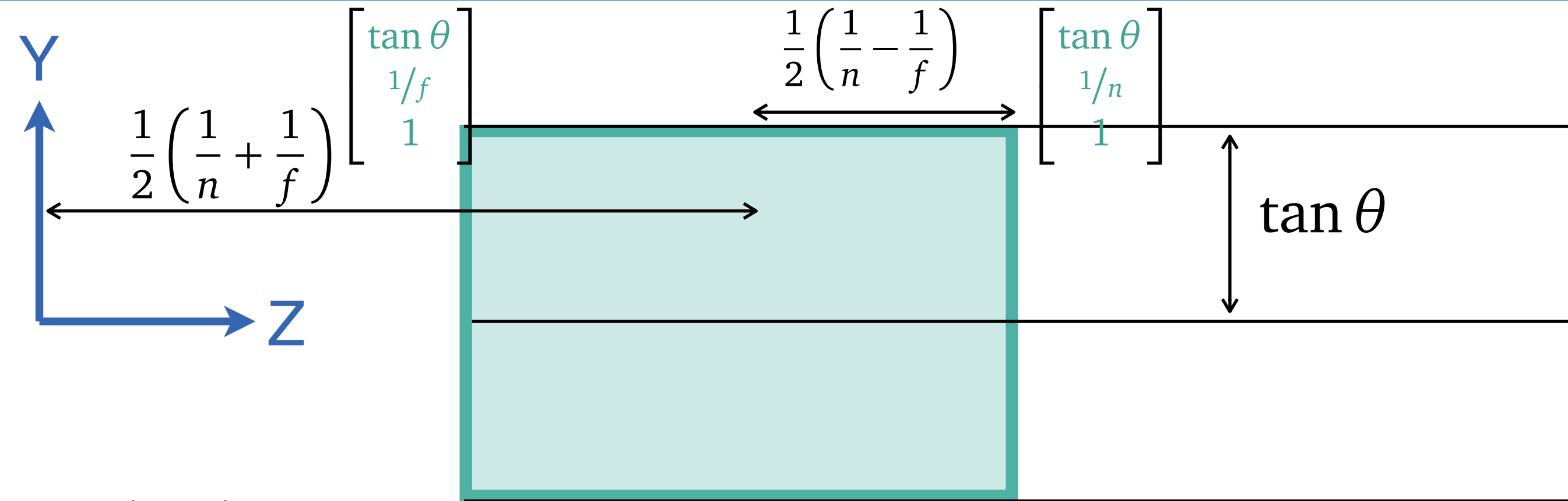
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} y \\ 1 \\ -z \end{bmatrix} \sim \begin{bmatrix} -y/z \\ -1/z \\ 1 \end{bmatrix}$$

- Translation

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\frac{f+n}{2fn} \\ 0 & 0 & 1 \end{bmatrix}$$

- Scaling

$$\begin{bmatrix} \frac{1}{\tan \theta} & 0 & 0 \\ 0 & \frac{2fn}{f-n} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

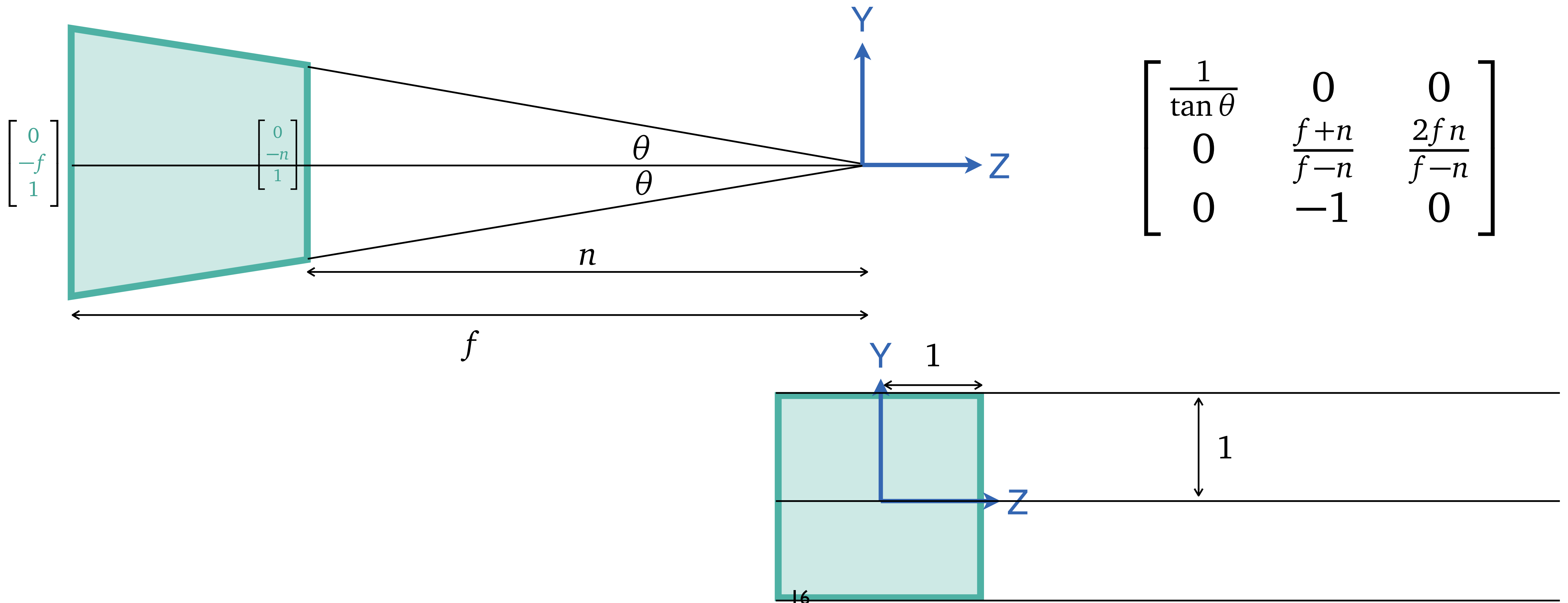


# Trapezoid to square

$$\begin{bmatrix} \frac{1}{\tan \theta} & 0 & 0 \\ 0 & \frac{2fn}{f-n} & 0 \\ 0 & 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\frac{f+n}{2fn} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}}_{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{f+n}{2fn} & 1 \\ 0 & -1 & 0 \end{bmatrix}} = \begin{bmatrix} \frac{1}{\tan \theta} & 0 & 0 \\ 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & -1 & 0 \end{bmatrix}$$

# Trapezoid to square

- Design a 3x3 matrix (applied to homogeneous coordinates) that deforms the trapezoid to the square.



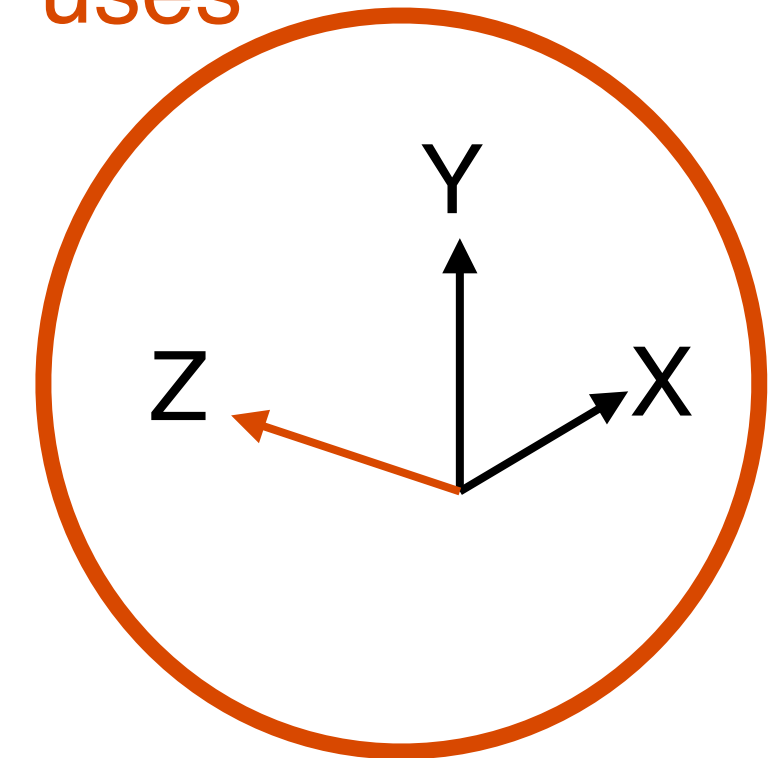
# Add x coordinate

$$\begin{bmatrix} \frac{1}{\tan \theta} & 0 & 0 \\ 0 & \frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & -1 & 0 \end{bmatrix}$$

# Add x coordinate

$$\begin{bmatrix} \frac{1}{a \tan \theta} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan \theta} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

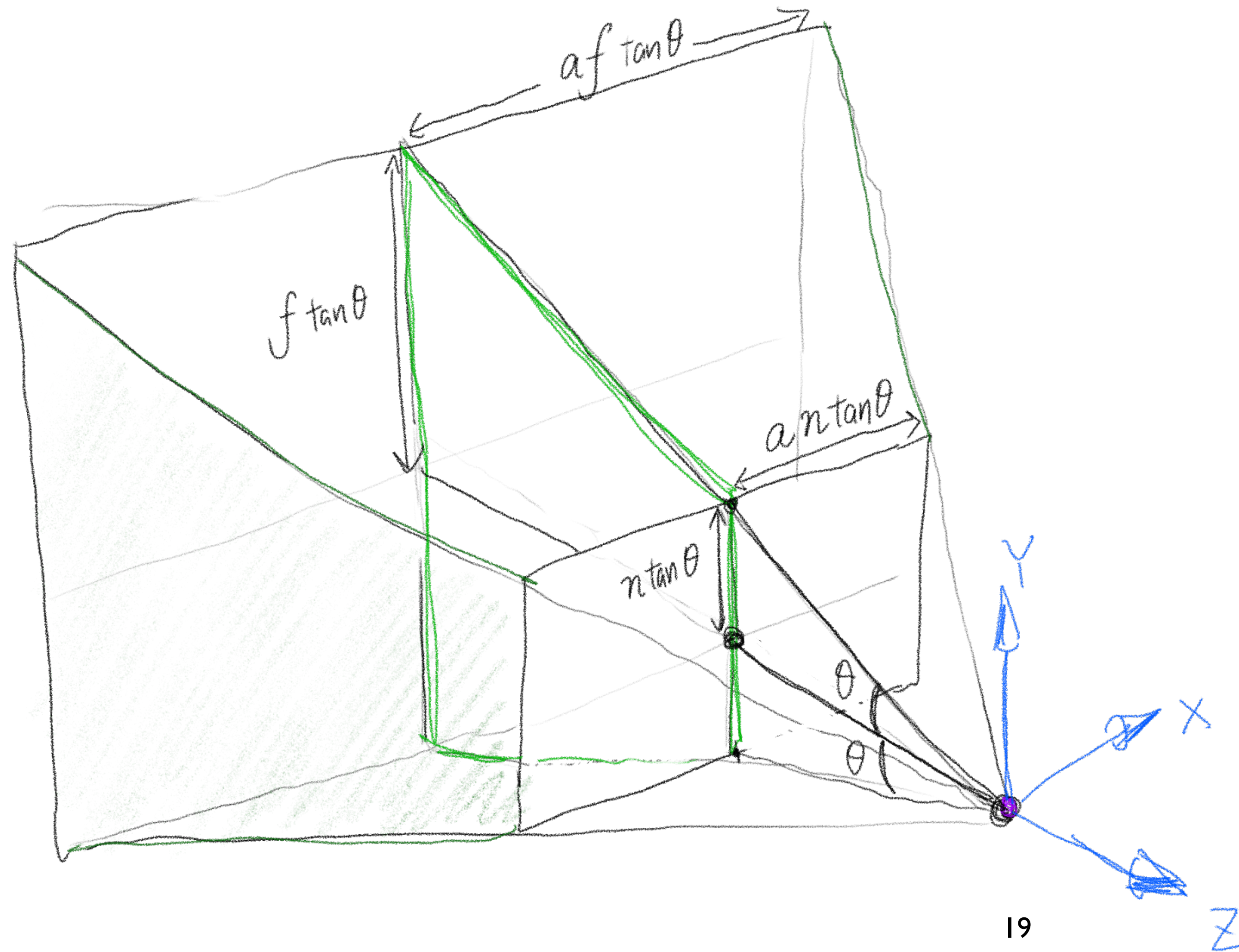
OpenGL  
actually  
uses



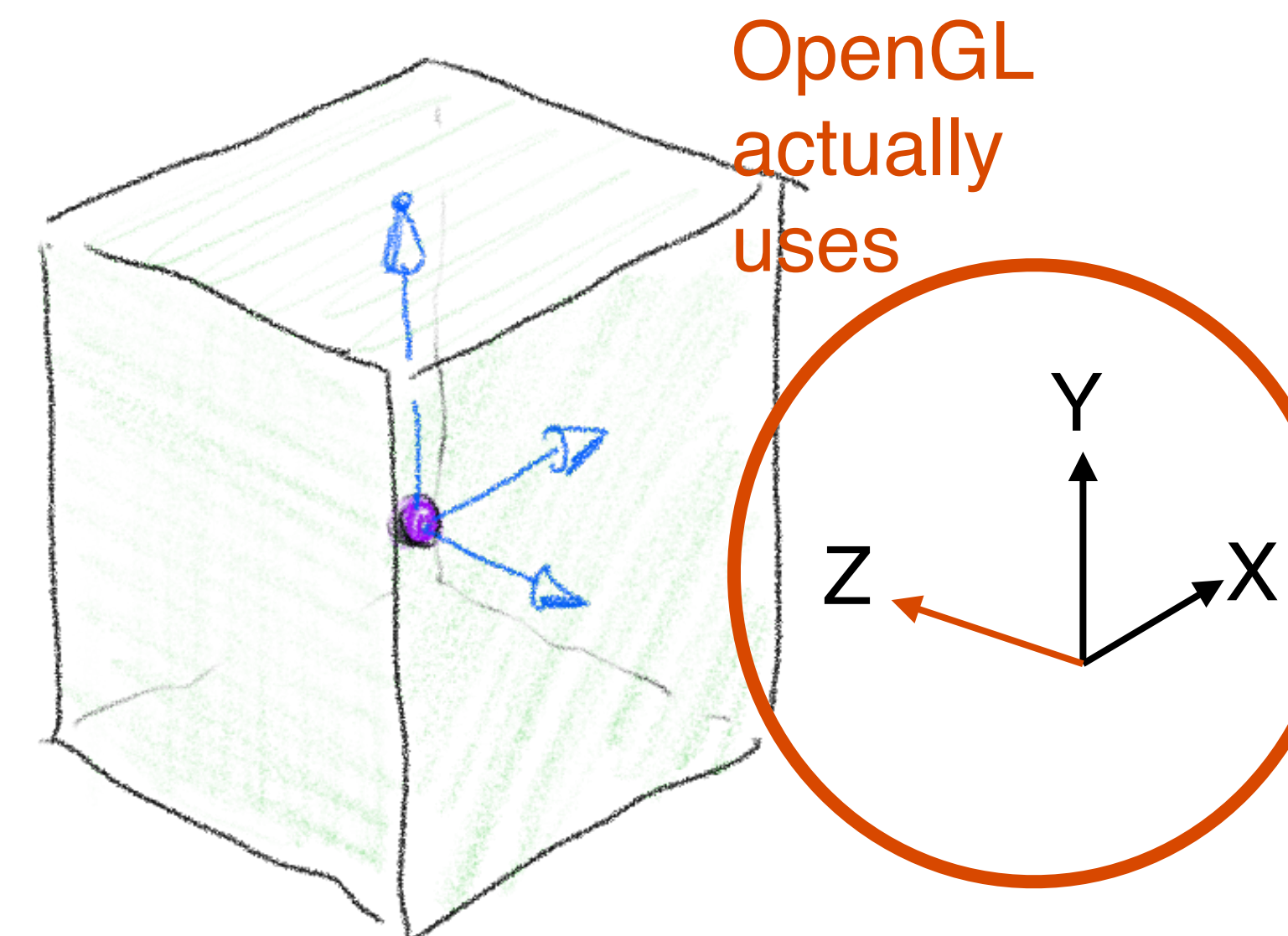
so just add a minus  
sign to the z-row

# Trapezoid to square

- The 4x4 matrix mapping the frustum to the box



$$\begin{bmatrix} \frac{1}{a \tan \theta} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan \theta} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



**mat4 Perspective( frustum\_description )**

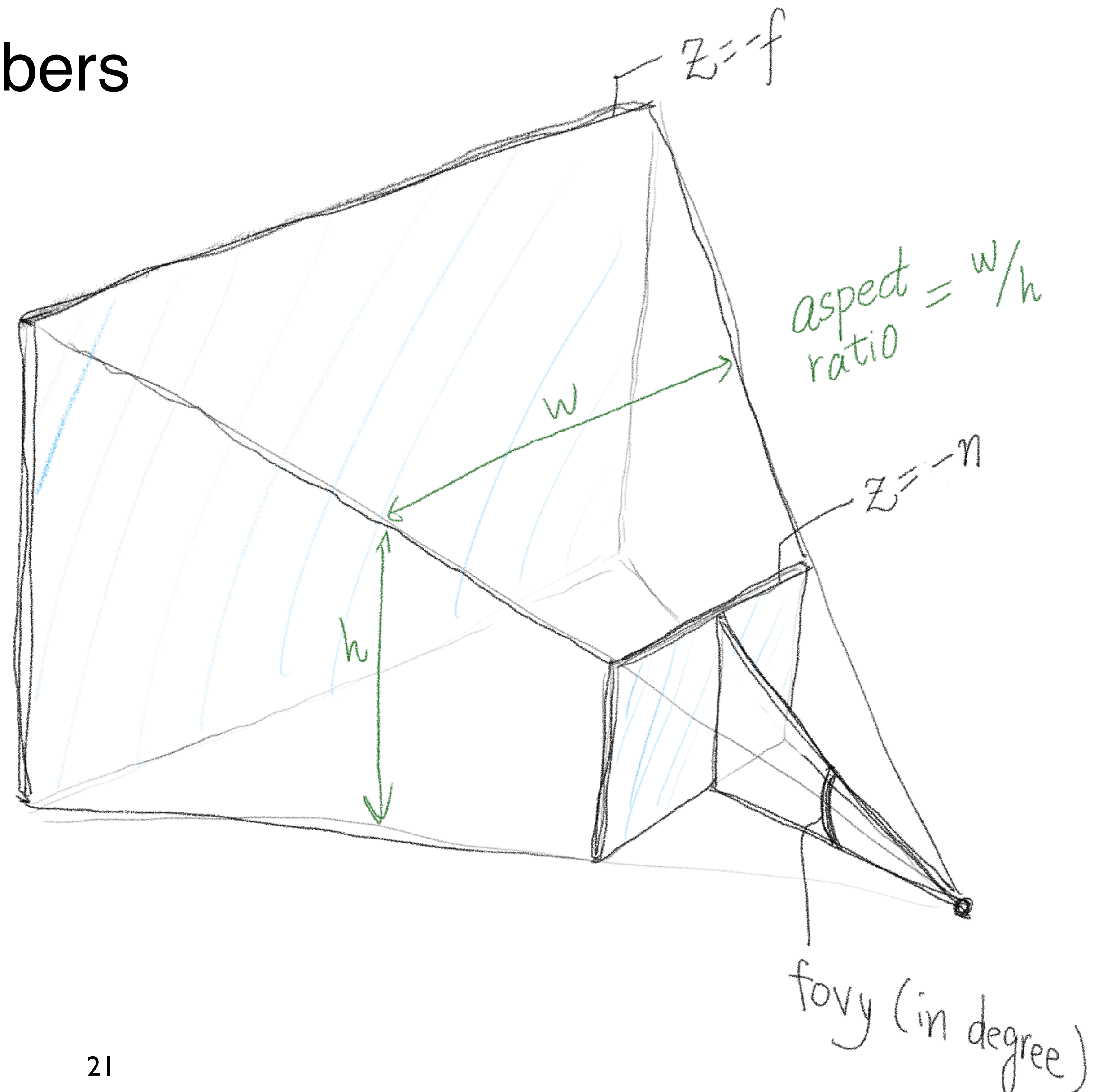
# Perspective

- gluPerspective takes 4 numbers

- ▶ fovy (in degree)  $\left(\theta = \frac{1}{2}\text{rad(fovy)}\right)$
- ▶ aspect ratio  $a$
- ▶ zNear  $n$
- ▶ zFar  $f$

and produces the 4x4 matrix

$$\begin{bmatrix} \frac{1}{a \tan \theta} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan \theta} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



# Next time

- Basic OpenGL

