

# Introduction to Computer Graphics

AMES101, Lingqi Yan, UC Santa Barbara

## Lecture 4: Transformation Cont.



# Announcement

- Homework 0 will be released TODAY
- This lecture will be difficult :)

# Last Lecture

- Transformation
  - Why study transformation
  - 2D transformations: rotation, scale, shear
  - Homogeneous coordinates
  - Composite transform
  - 3D transformations

# Today

- 3D transformations
- Viewing (观测) transformation
  - View (视图) / Camera transformation

可以理解为外餐矩阵，如何通过基本变换将相机放到标准位姿
  - Projection (投影) transformation
    - Orthographic (正交) projection
    - Perspective (透视) projection

# 3D Transformations

Use homogeneous coordinates again:

- 3D point =  $(x, y, z, 1)^T$
- 3D vector =  $(x, y, z, 0)^T$

In general,  $(x, y, z, w)$  ( $w \neq 0$ ) is the 3D point:

$$(x/w, y/w, z/w)$$

# 3D Transformations

Use  $4 \times 4$  matrices for affine transformations

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

**What's the order?**

**Linear Transform first or Translation first?**

# 3D Transformations

**Scale**

$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Translation**

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

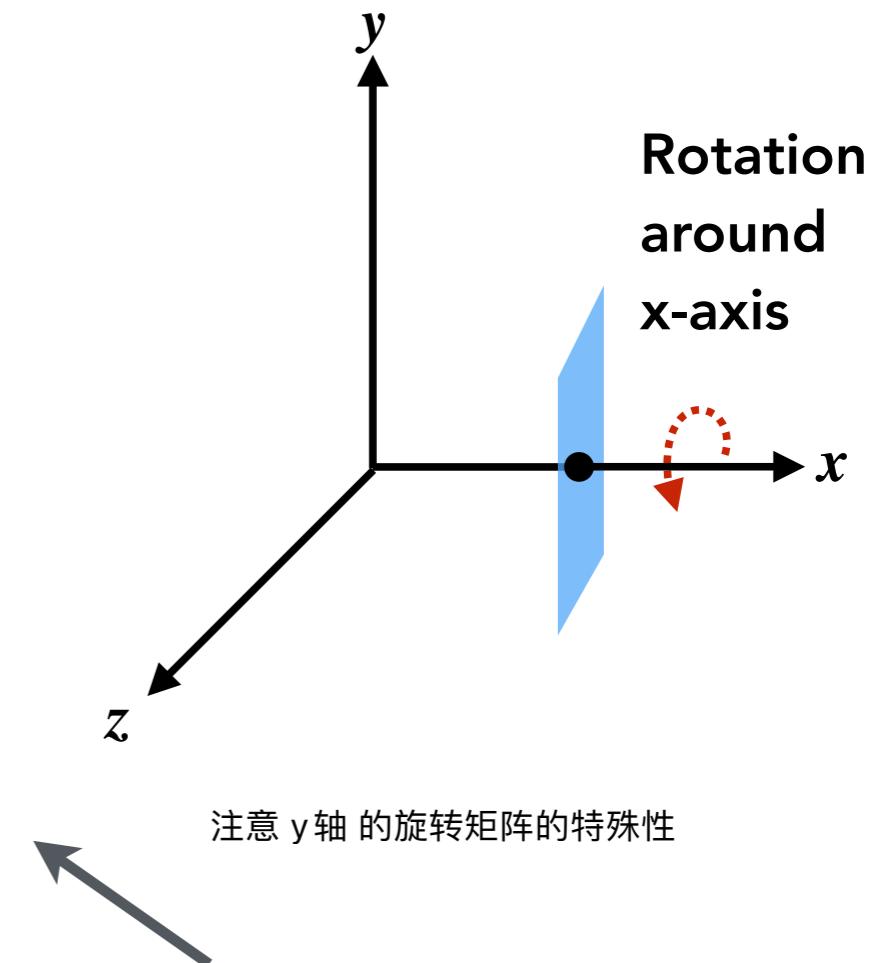
# 3D Transformations

## Rotation around x-, y-, or z-axis

$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R}_y(\alpha) = \begin{pmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R}_z(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



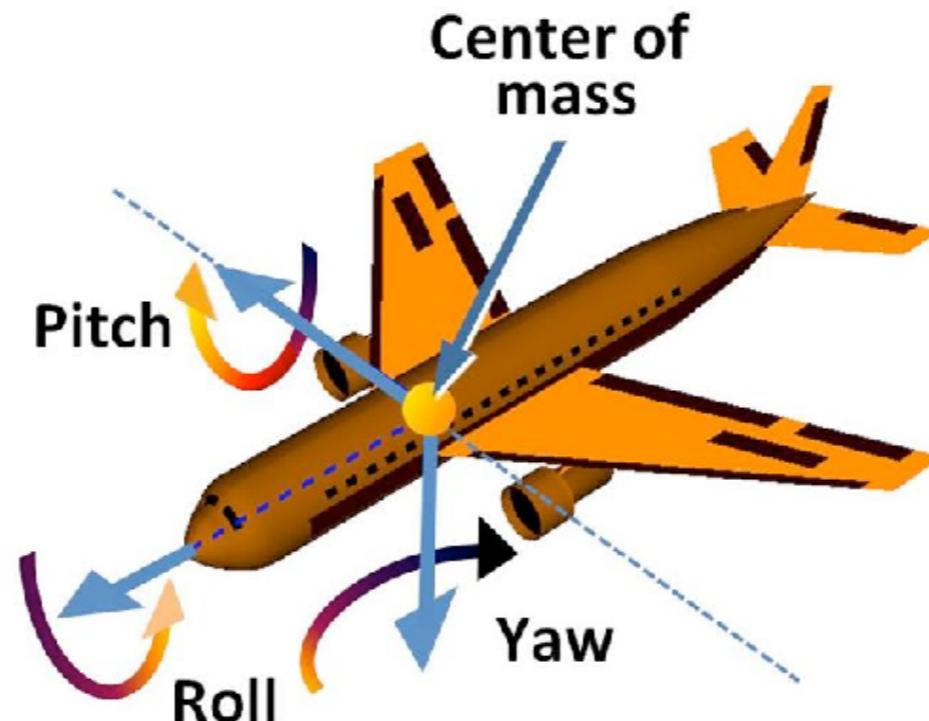
Anything strange about  $\mathbf{R}_y$ ?

# 3D Rotations

Compose any 3D rotation from  $\mathbf{R}_x, \mathbf{R}_y, \mathbf{R}_z$ ?

$$\mathbf{R}_{xyz}(\alpha, \beta, \gamma) = \mathbf{R}_x(\alpha) \mathbf{R}_y(\beta) \mathbf{R}_z(\gamma)$$

- So-called **Euler angles**
- Often used in flight simulators: roll, pitch, yaw



# Rodrigues' Rotation Formula

Rotation by angle  $\alpha$  around axis  $\mathbf{n}$

$$\mathbf{R}(\mathbf{n}, \alpha) = \cos(\alpha) \mathbf{I} + (1 - \cos(\alpha)) \mathbf{n}\mathbf{n}^T + \sin(\alpha) \underbrace{\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{pmatrix}}_{\mathbf{N}}$$

How to prove this magic formula?

Check out the supplementary material on the course website!

# Today

- 3D transformations
- Viewing transformation
  - View / Camera transformation
  - Projection transformation
    - Orthographic projection
    - Perspective projection

# View / Camera Transformation

- What is view transformation?
- Think about how to take a photo
  - Find a good place and arrange people (**model** transformation)
  - Find a good “angle” to put the camera (**view** transformation)
  - Cheese! (**projection** transformation)

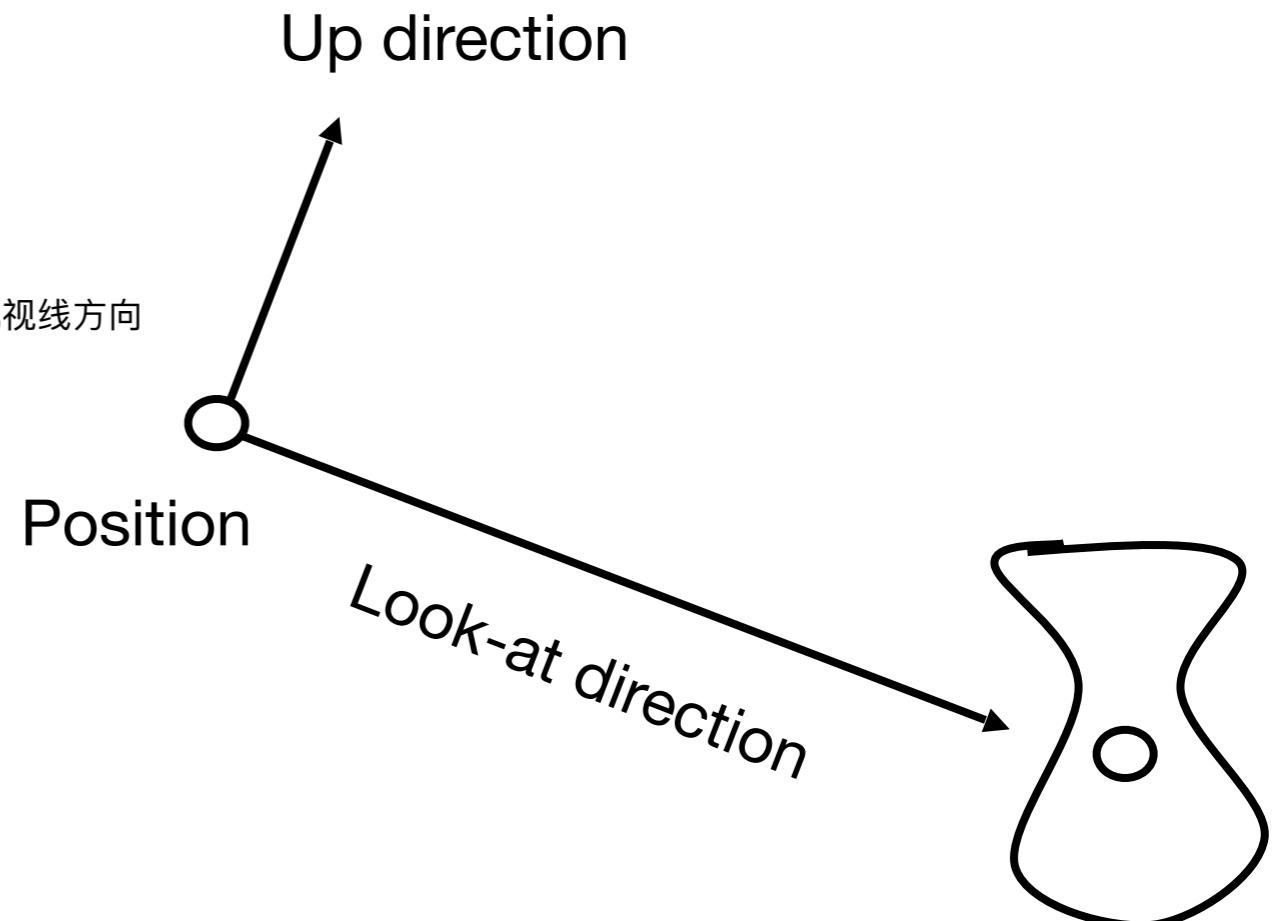
# View / Camera Transformation

将相机通过坐标变换到坐标系的原点坐标系上

- How to perform view transformation?

- Define the camera first

- Position  $\vec{e}$  相机原点位置
- Look-at / gaze direction  $\hat{g}$  相机视线方向
- Up direction  $\hat{t}$  相机正像向上  
(assuming perp. to look-at)

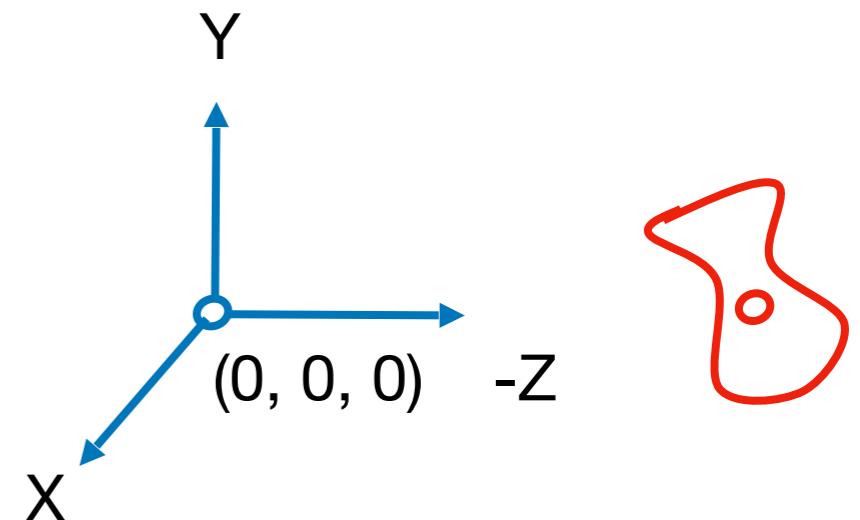
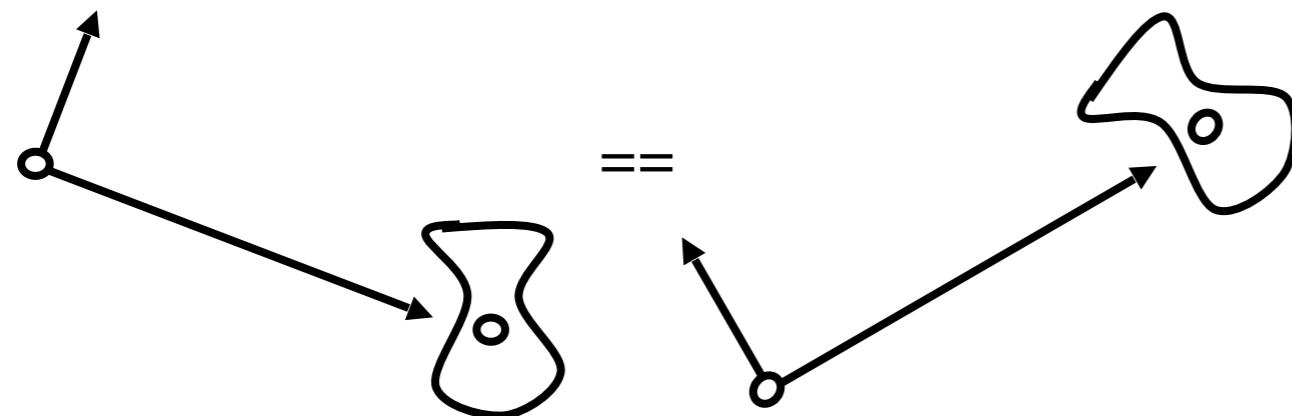


# View / Camera Transformation

- Key observation

只是保证成像平面是固定的，场景与相机同步转换

- If the camera and all objects move together,  
the “photo” will be the same

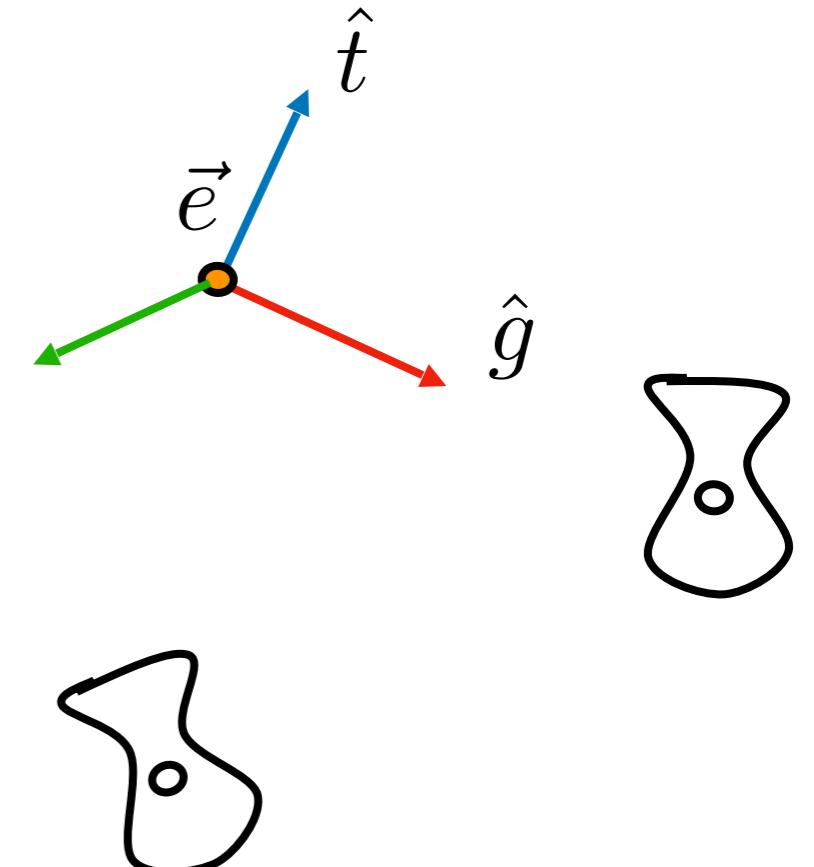
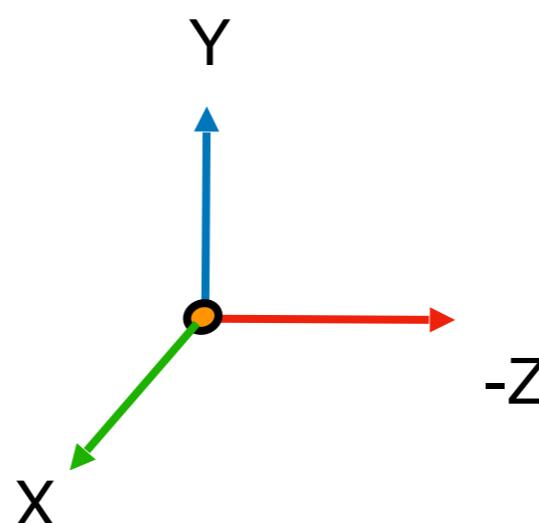


- How about that we always transform the camera to
  - The origin, up at Y, look at -Z
  - And transform the objects along with the camera

# View / Camera Transformation

- Transform the camera by  $M_{view}$ 
  - So it's located at the origin, up at Y, look at -Z

- $M_{view}$  in math?
  - Translates e to origin
  - Rotates g to -Z
  - Rotates t to Y
  - Rotates  $(g \times t)$  To X
  - Difficult to write!



# View / Camera Transformation

- $M_{view}$  in math?

- Let's write  $M_{view} = R_{view}T_{view}$

由于一个齐次矩阵是先旋转后平移，所以这里需要单独写平移矩阵

- Translate e to origin

$$T_{view} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotate g to -Z, t to Y,  $(g \times t)$  To X

- Consider its **inverse** rotation: X to  $(g \times t)$ , Y to t, Z to -g

$$R_{view}^{-1} = \begin{bmatrix} x_{\hat{g} \times \hat{t}} & x_t & x_{-g} & 0 \\ y_{\hat{g} \times \hat{t}} & y_t & y_{-g} & 0 \\ z_{\hat{g} \times \hat{t}} & z_t & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



WHY?

$$R_{view} = \begin{bmatrix} x_{\hat{g} \times \hat{t}} & y_{\hat{g} \times \hat{t}} & z_{\hat{g} \times \hat{t}} & 0 \\ x_t & y_t & z_t & 0 \\ x_{-g} & y_{-g} & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# View / Camera Transformation

- Summary
  - Transform objects together with the camera
  - Until camera's at the origin, up at Y, look at -Z
- Also known as ModelView Transformation
- But why do we need this?
  - For projection transformation!

# Today

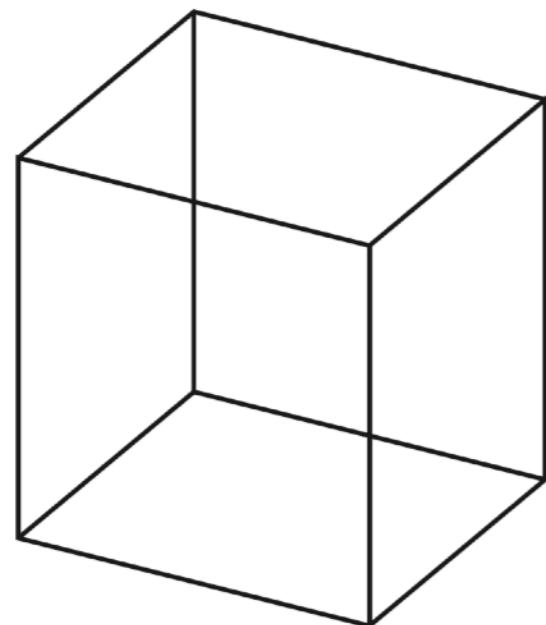
- 3D transformations
- Viewing transformation
  - View / Camera transformation
  - **Projection transformation**
    - Orthographic projection
    - Perspective projection

# Projection Transformation

- Projection in Computer Graphics
  - 3D to 2D
  - Orthographic projection
  - Perspective projection

正交投影，不会改变物体形状

Orthographic  
projection



投影变换会改变物体的平行性质

Perspective  
projection

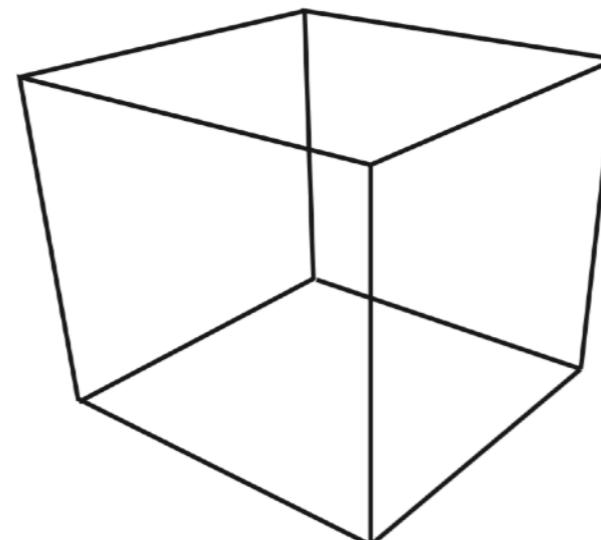
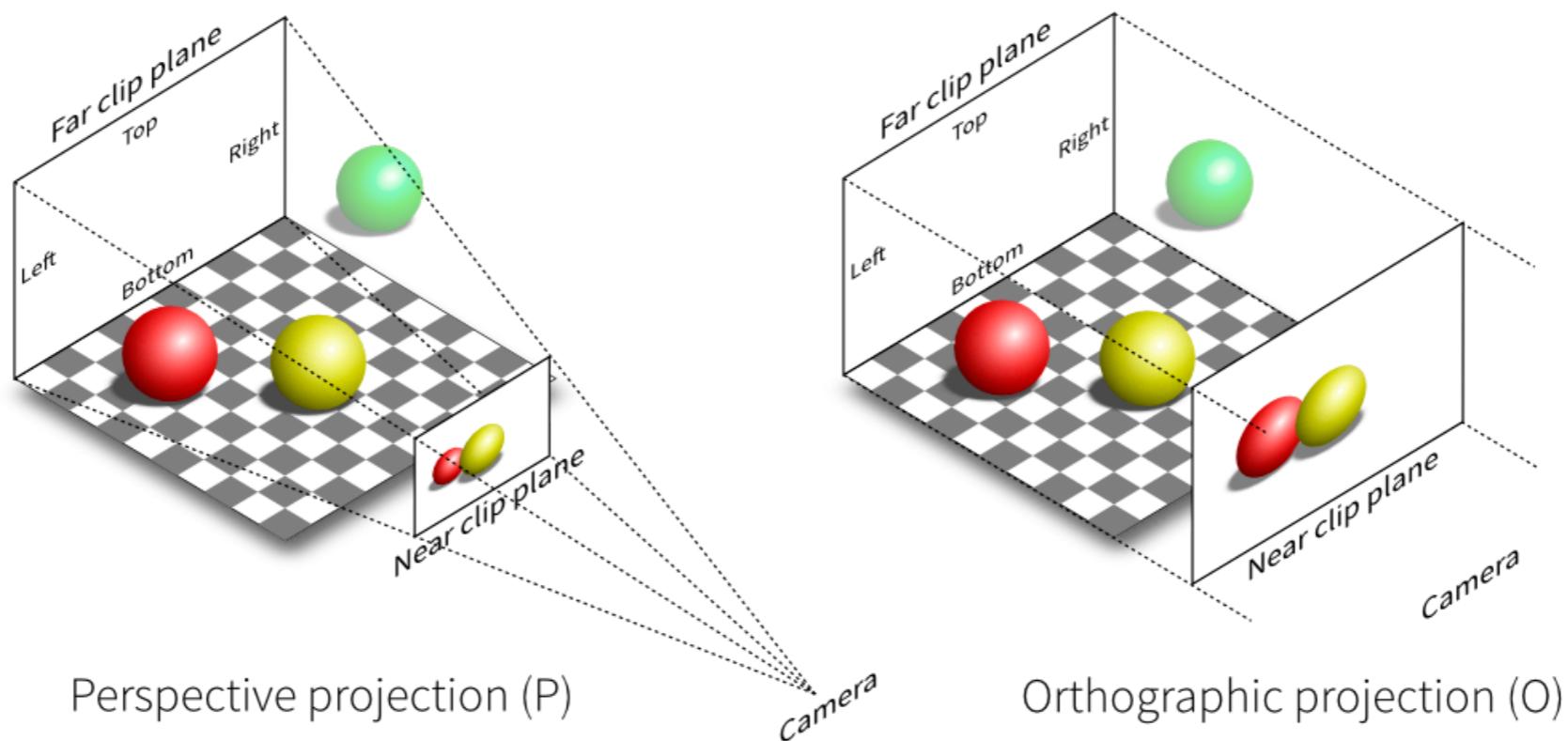


Fig. 7.1 from *Fundamentals of Computer Graphics, 4th Edition*

# Projection Transformation

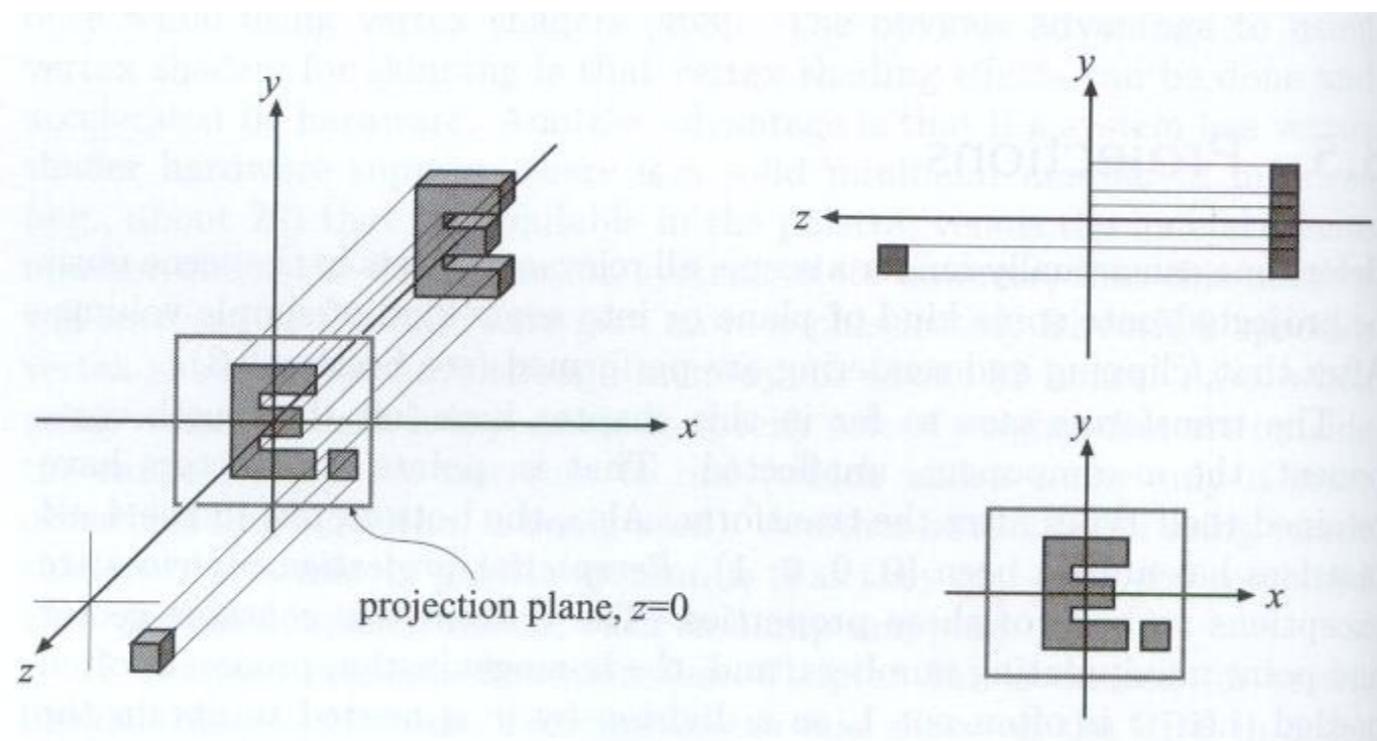
- Perspective projection vs. orthographic projection



<https://stackoverflow.com/questions/36573283/from-perspective-picture-to-orthographic-picture>

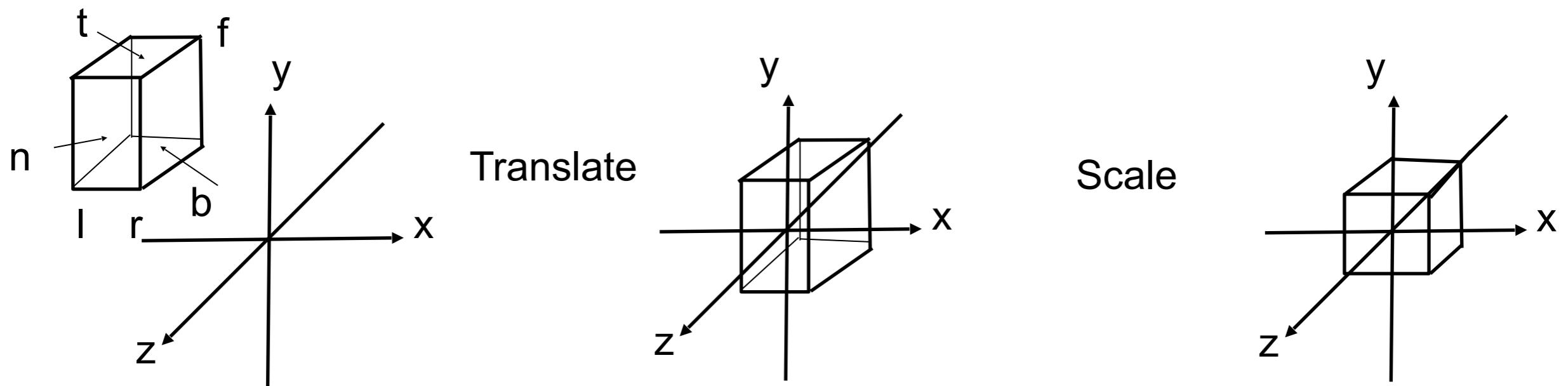
# Orthographic Projection

- A simple way of understanding
  - Camera located at origin, looking at -Z, up at Y (looks familiar?)
  - **Drop Z coordinate**
  - Translate and scale the resulting rectangle to  $[-1, 1]^2$



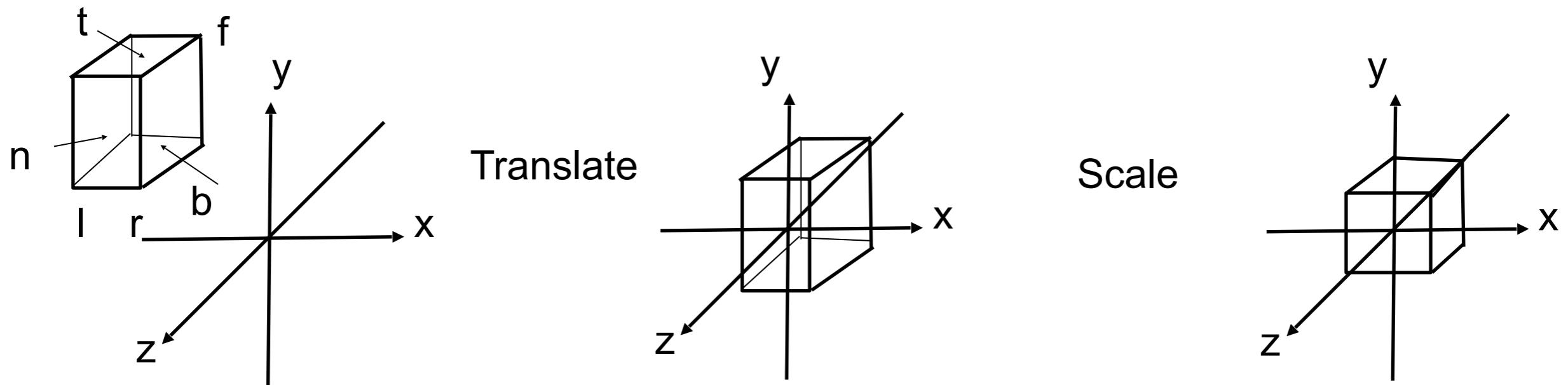
# Orthographic Projection

- In general
  - We want to map a cuboid  $[l, r] \times [b, t] \times [f, n]$  to the “canonical (正则、规范、标准)” cube  $[-1, 1]^3$



# Orthographic Projection

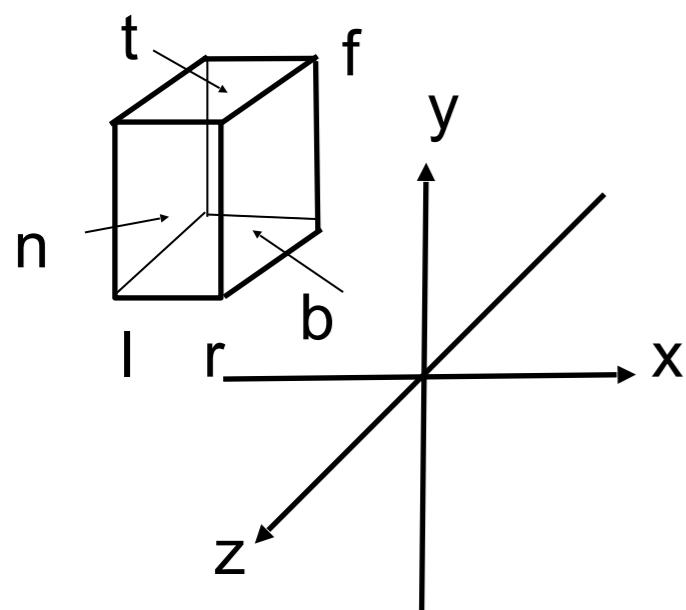
- Slightly different orders (to the “simple way”)
  - Center cuboid by translating
  - Scale into “canonical” cube



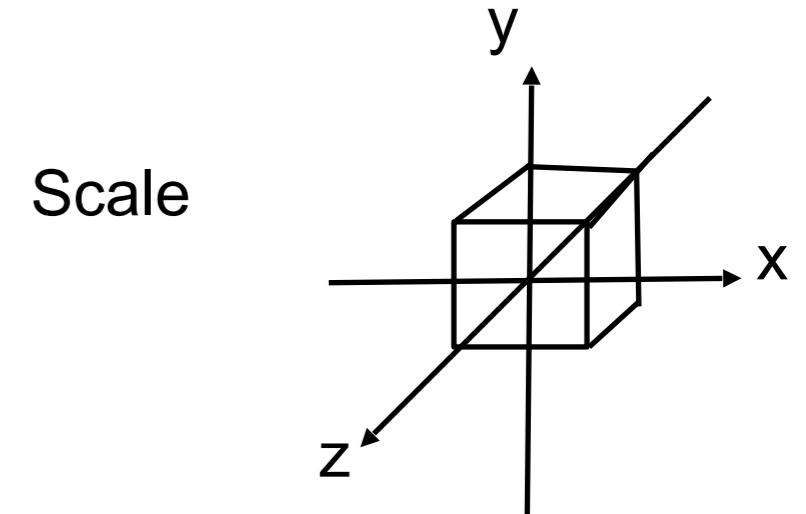
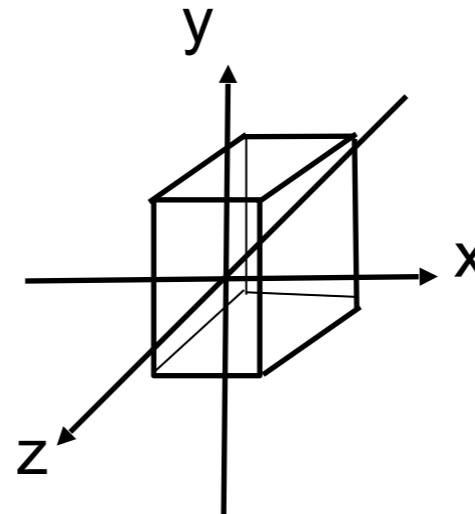
# Orthographic Projection

- Transformation matrix?
  - Translate (**center** to origin) **first**, then scale (length/width/height to **2**)

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

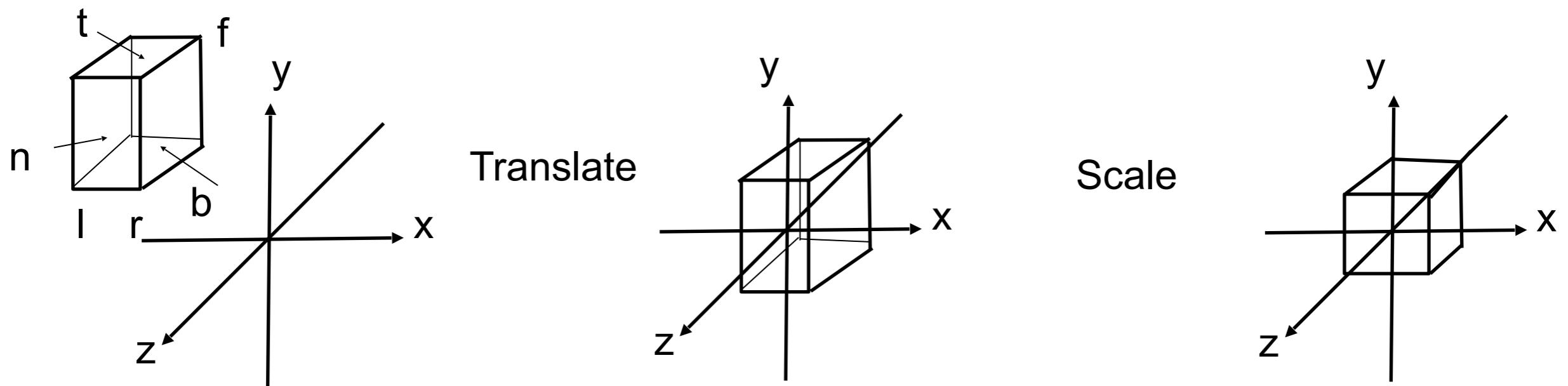


Translate



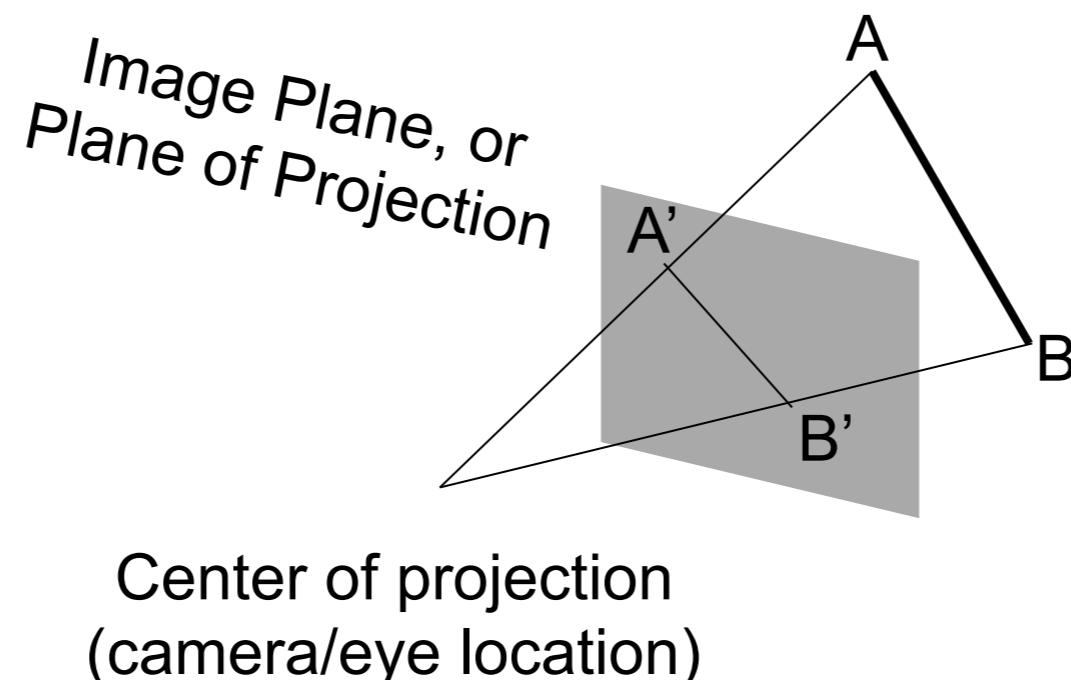
# Orthographic Projection

- Caveat
  - Looking at / along -Z is making near and far not intuitive ( $n > f$ )
  - FYI: that's why OpenGL (a Graphics API) uses left hand coords.



# Perspective Projection

- Most common in Computer Graphics, art, visual system
- Further objects are smaller
- Parallel lines not parallel; converge to single point

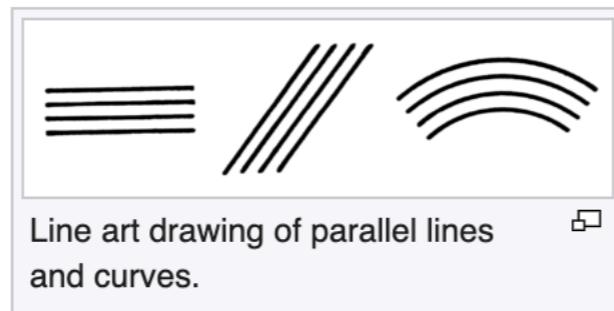


# Perspective Projection

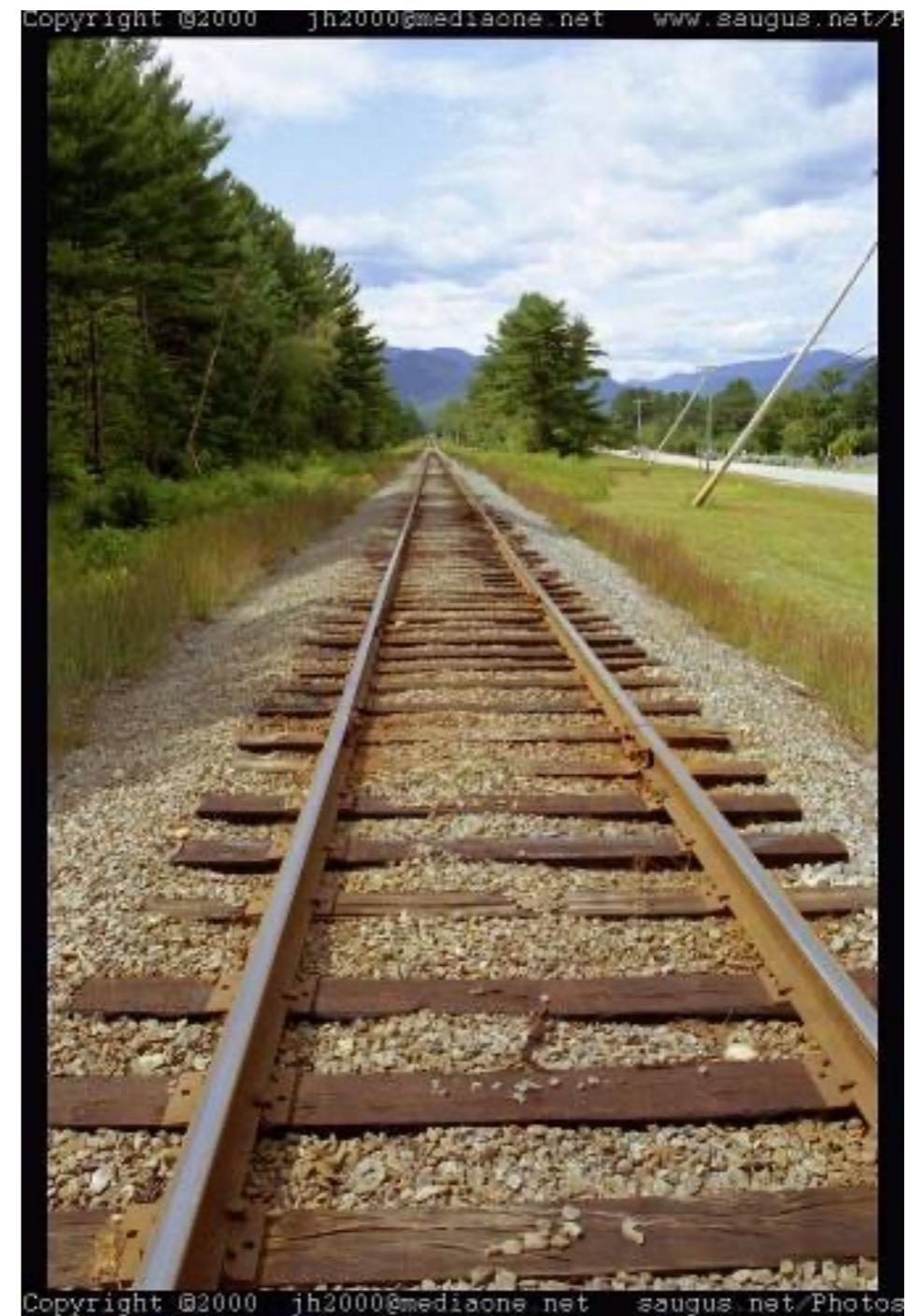
- Euclid was wrong??!!

In [geometry](#), **parallel** lines are [lines in a plane](#) which do not meet; that is, two lines in a plane that do not [intersect](#) or [touch](#) each other at any point are said to be parallel. By extension, a line and a plane, or two planes, in [three-dimensional Euclidean space](#) that do not share a point are said to be parallel. However, two lines in three-dimensional space which do not meet must be in a common plane to be considered parallel; otherwise they are called [skew lines](#). Parallel planes are planes in the same three-dimensional space that never meet.

Parallel lines are the subject of [Euclid's parallel postulate](#).<sup>[1]</sup> Parallelism is primarily a property of [affine geometries](#) and [Euclidean geometry](#) is a special instance of this type of geometry. In some other geometries, such as [hyperbolic geometry](#), lines can have analogous properties that are referred to as parallelism.



Line art drawing of parallel lines and curves.



[https://en.wikipedia.org/wiki/Parallel\\_\(geometry\)](https://en.wikipedia.org/wiki/Parallel_(geometry))

# Perspective Projection

- Before we move on
- Recall: property of homogeneous coordinates
  - $(x, y, z, 1)$ ,  $(kx, ky, kz, k \neq 0)$ ,  **$(xz, yz, z^2, z \neq 0)$**  all represent the same point  $(x, y, z)$  in 3D 坐标等比例缩放, 点的位置不变
  - e.g.  $(1, 0, 0, 1)$  and  $(2, 0, 0, 2)$  both represent  $(1, 0, 0)$
- Simple, but useful

# Perspective Projection

- How to do perspective projection
  - First “squish” the frustum into a cuboid ( $n \rightarrow n$ ,  $f \rightarrow f$ ) ( $M_{\text{persp} \rightarrow \text{ortho}}$ )
  - Do orthographic projection ( $M_{\text{ortho}}$ , already known!)

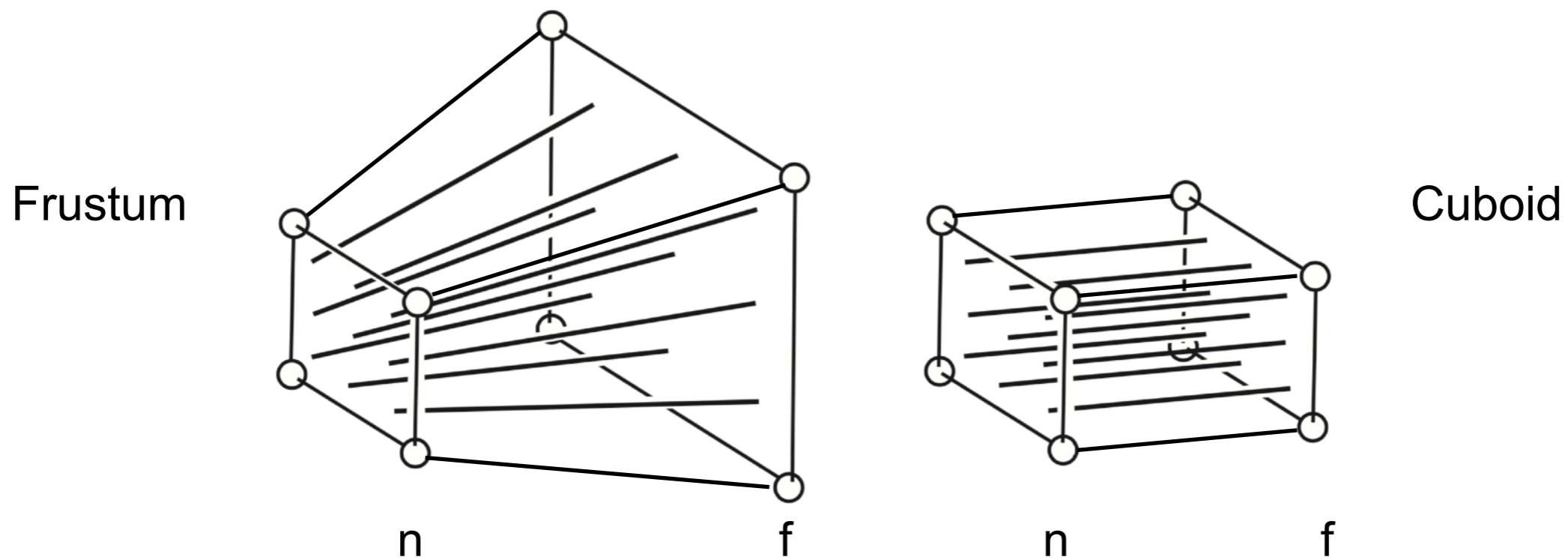
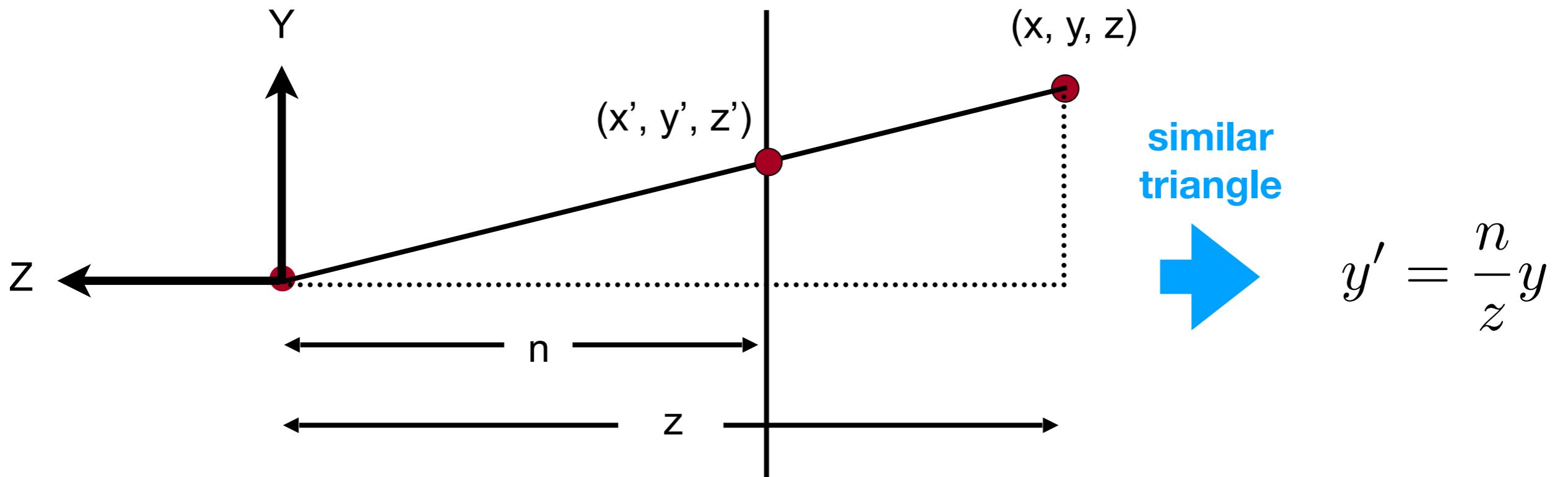


Fig. 7.13 from *Fundamentals of Computer Graphics, 4th Edition*

# Perspective Projection

- In order to find a transformation
  - Recall the key idea: Find the relationship between transformed points  $(x', y', z')$  and the original points  $(x, y, z)$



# Perspective Projection

- In order to find a transformation
  - Find the relationship between transformed points ( $x'$ ,  $y'$ ,  $z'$ ) and the original points ( $x$ ,  $y$ ,  $z$ )

$$y' = \frac{n}{z}y \quad x' = \frac{n}{z}x \text{ (similar to } y')$$

- In homogeneous coordinates,

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} nx/z \\ ny/z \\ \text{unknown} \\ 1 \end{pmatrix} \stackrel{\text{mult. by } z}{=} \begin{pmatrix} nx \\ ny \\ \text{still unknown} \\ z \end{pmatrix}$$

# Perspective Projection

- So the “squish” (persp to ortho) projection does this

$$M_{\text{persp} \rightarrow \text{ortho}}^{(4 \times 4)} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix}$$

- Already good enough to figure out part of  $M_{\text{persp} \rightarrow \text{ortho}}$

$$M_{\text{persp} \rightarrow \text{ortho}} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

WHY?

# Perspective Projection

- How to figure out the third row of  $M_{persp \rightarrow ortho}$

- Any information that we can use?

$$M_{persp \rightarrow ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- Observation: the third row is responsible for  $z'$

- Any point on the near plane will not change
  - Any point's  $z$  on the far plane will not change

(1) 任何近平面上的点不会改变 (也就是对于任意的  $(x, y, n, 1)$ , 经过这个矩阵变换后, 点的位置仍然不变)

(2) 任何远平面上的点,  $z$  值不会改变

# Perspective Projection

- Any point on the near plane will not change

$$M_{persp \rightarrow ortho}^{(4 \times 4)} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ \text{unknown} \\ z \end{pmatrix}$$

通用表达式

近平面的  $z = n$

replace  
z with n

$$\begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} == \begin{pmatrix} nx \\ ny \\ n^2 \\ n \end{pmatrix}$$

近平面点保持不变，仅仅对远平面点进行压缩

- So the third row must be of the form  $(0 \ 0 \ A \ B)$

$$(0 \ 0 \ A \ B) \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = n^2$$

n<sup>2</sup> has nothing  
to do with x and y

# Perspective Projection

- What do we have now?

$$(0 \ 0 \ A \ B) \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = n^2 \quad \rightarrow \quad An + B = n^2$$

- Any point's z on the far plane will not change

$$\begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} == \begin{pmatrix} 0 \\ 0 \\ f^2 \\ f \end{pmatrix} \quad \rightarrow \quad Af + B = f^2$$

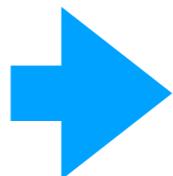
[0,0, f, 1] 与 x,y 无关的点, z 坐标不变

# Perspective Projection

- Solve for A and B

$$An + B = n^2$$

$$Af + B = f^2$$



$$A = n + f$$

$$B = -nf$$

- Finally, every entry in  $M_{\text{persp} \rightarrow \text{ortho}}$  is known!

- What's next?

- Do orthographic projection ( $M_{\text{ortho}}$ ) to finish

- $M_{\text{persp}} = M_{\text{ortho}} M_{\text{persp} \rightarrow \text{ortho}}$

**Thank you!**