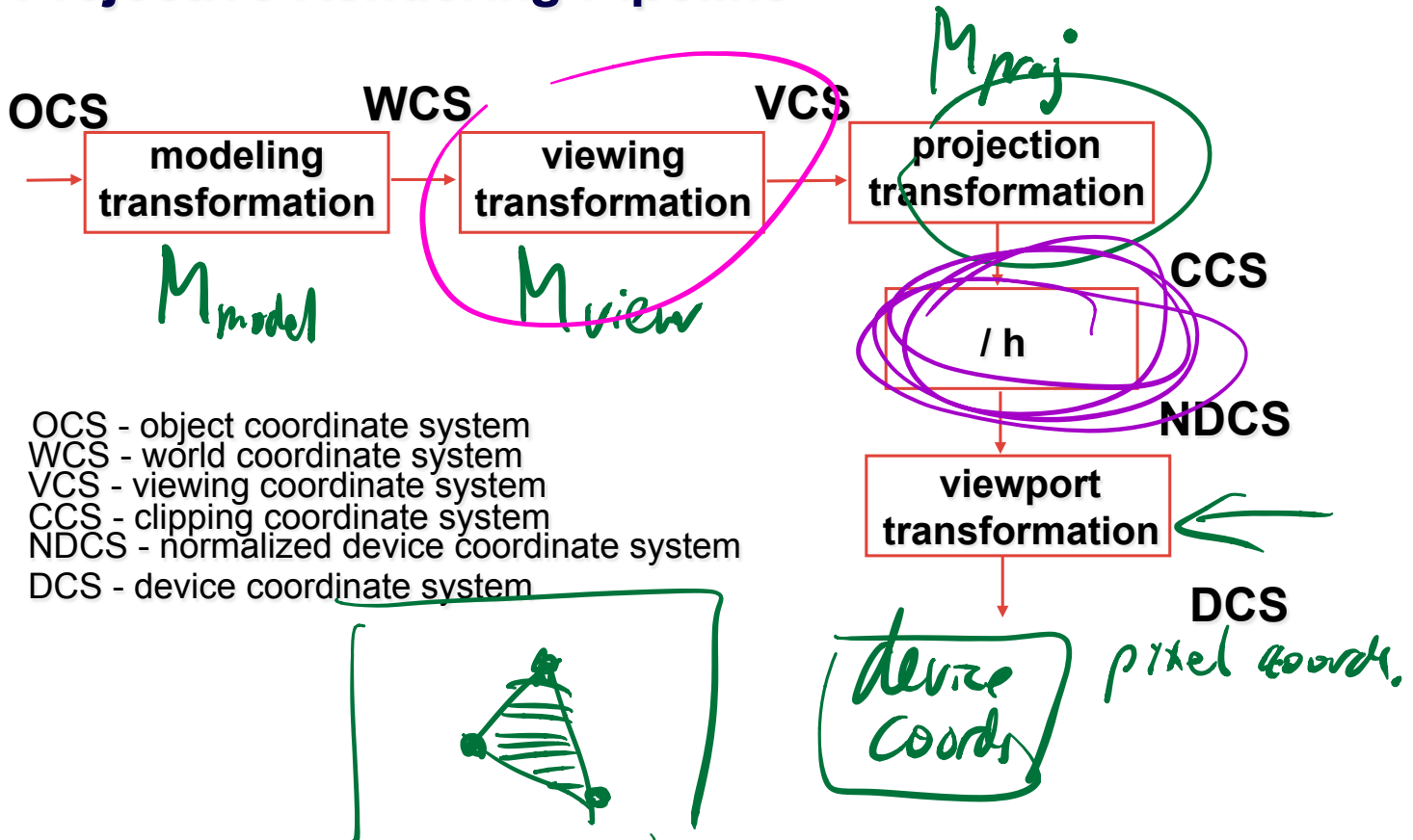
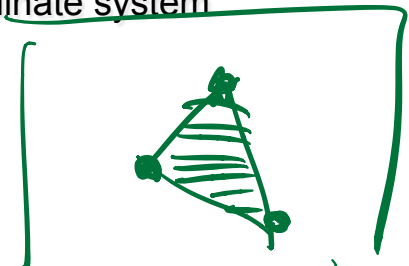


# Viewing and Projection Transformations

## Projective Rendering Pipeline



- OCS - object coordinate system
- WCS - world coordinate system
- VCS - viewing coordinate system
- CCS - clipping coordinate system
- NDCS - normalized device coordinate system
- DCS - device coordinate system



# Viewing Transformation

---

## *Defining the camera position and orientation*

- eye point
- target point
- up vector

**three.js:**

```
camera.position.set(0,12,20); // eye
camera.up.set(0,1,0); // up vector
camera.lookAt(0,0,0); // specify target; compute M_view
// internally: object.matrix.lookAt(eye,target,up)
```

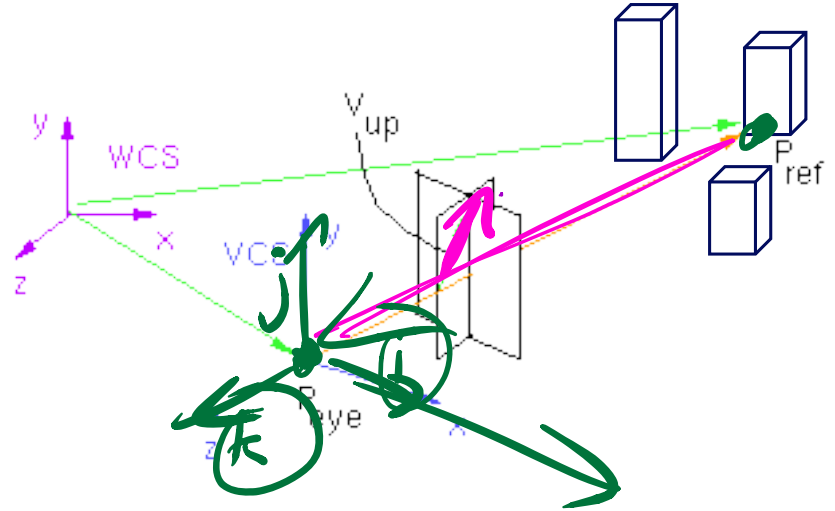
# Computing i,j,k

$$\vec{k}_{wcs} = \frac{P_{eye} - P_{ref}}{\|P_{eye} - P_{ref}\|}$$

$$\vec{i}_{wcs} = \frac{V_{up} \times \vec{k}}{\|V_{up} \times \vec{k}\|}$$

$$\vec{j}_{wcs} = \vec{k} \times \vec{i}$$

$$M_{view} = M_{cam}^{-1}$$



$$P_{wcs} = \begin{bmatrix} i & j & k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} P_{cam} P_{eye}$$

$M_{cam}$

# Viewing Transformation

 $M_{view}$ 

$$\begin{aligned} M_{cam} &= \text{Translate}(E_x, E_y, E_z) \text{Rotate}(\dots) \\ &= \begin{bmatrix} 1 & 0 & 0 & E_x \\ 0 & 1 & 0 & E_y \\ 0 & 0 & 1 & E_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_x & j_x & k_x & 0 \\ i_y & j_y & k_y & 0 \\ i_z & j_z & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} M_{view} &= M_{cam}^{-1} = \text{Rotate}(\dots)^{-1} \text{Translate}(E_x, E_y, E_z)^{-1} \\ &= \begin{bmatrix} i_x & i_y & i_z & 0 \\ j_x & j_y & j_z & 0 \\ k_x & k_y & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -E_x \\ 0 & 1 & 0 & -E_y \\ 0 & 0 & 1 & -E_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

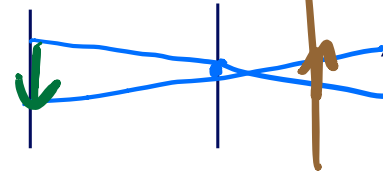
# Projection Transformation

$$M_{proj}$$

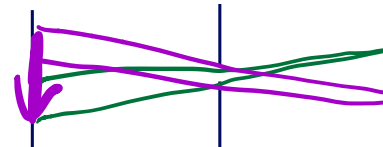
**3D scene  $\rightarrow$  2D image**

pinhole camera

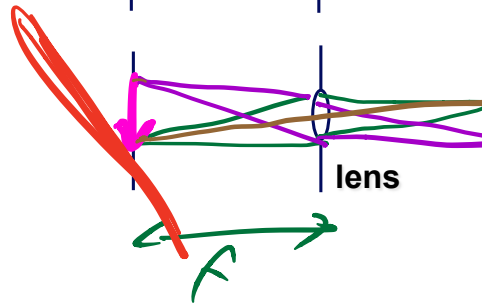
image plane



real pinhole camera



camera

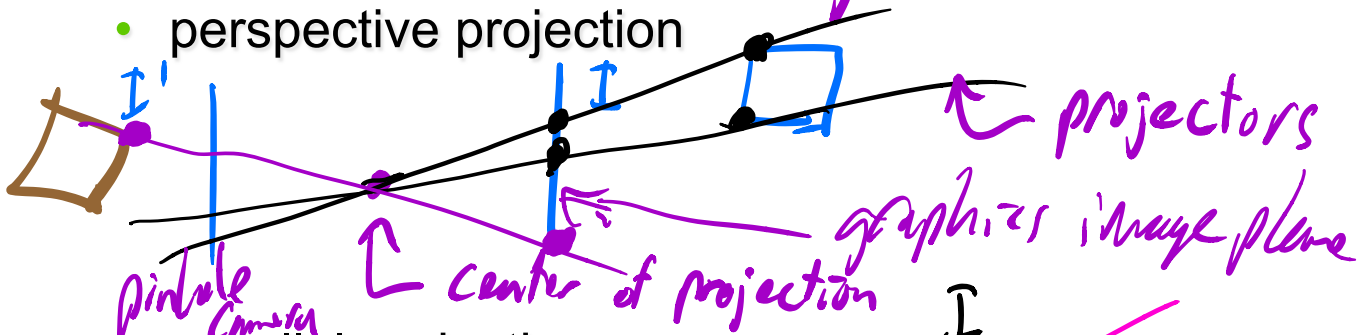


# Projection

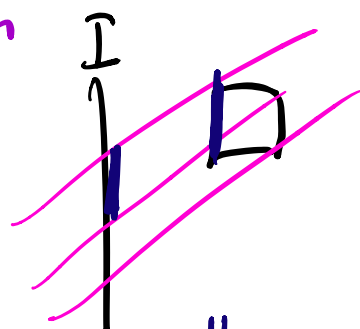
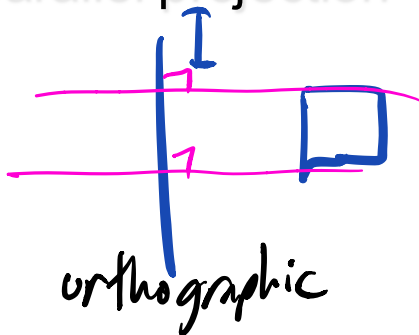
$p \quad p' \quad p' = f(p)$   
 $\mathbb{R}^m \rightarrow \mathbb{R}^n \quad n < m$

- definition

- perspective projection



- parallel projection

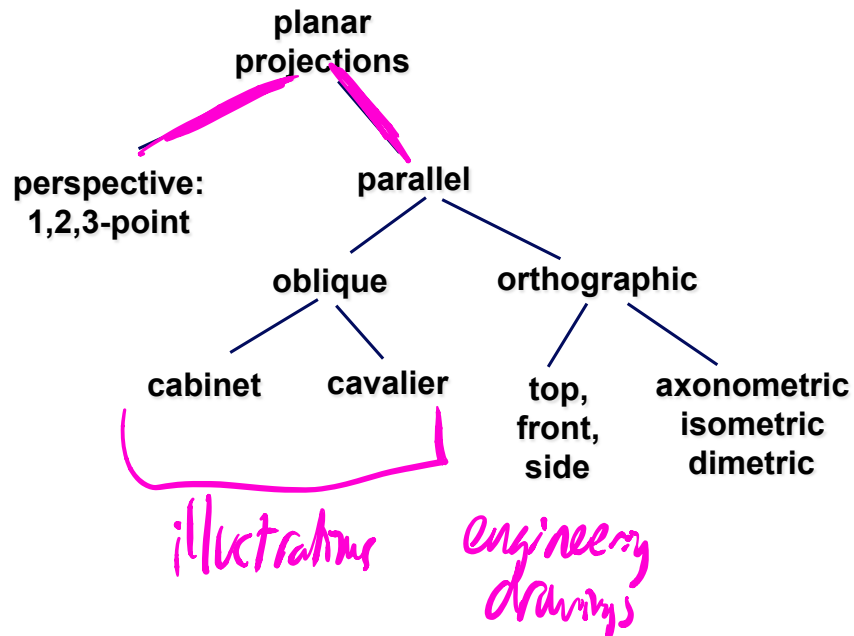


projectors are all parallel to each other

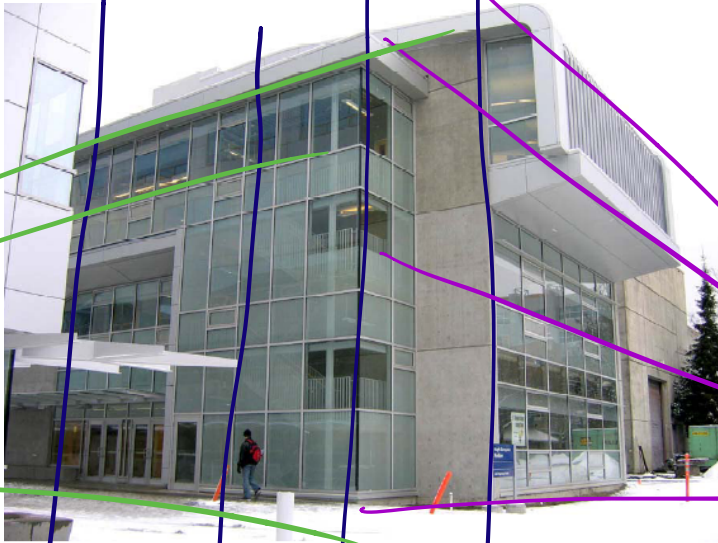
# Projections

---

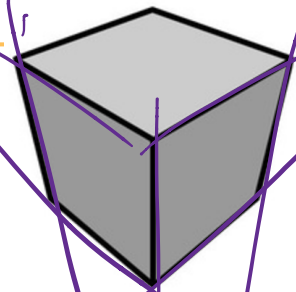
## *Taxonomy*



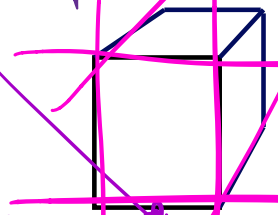
2-point perspective



parallel



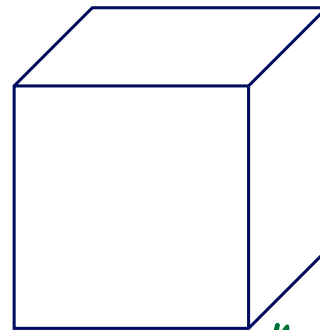
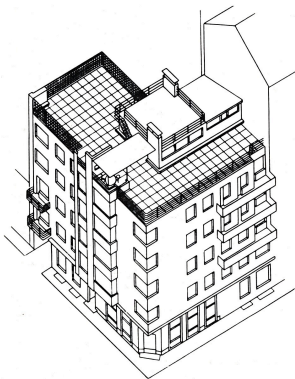
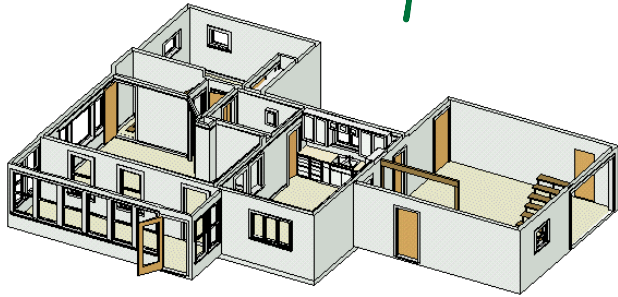
3-point persp.



One-point perspective

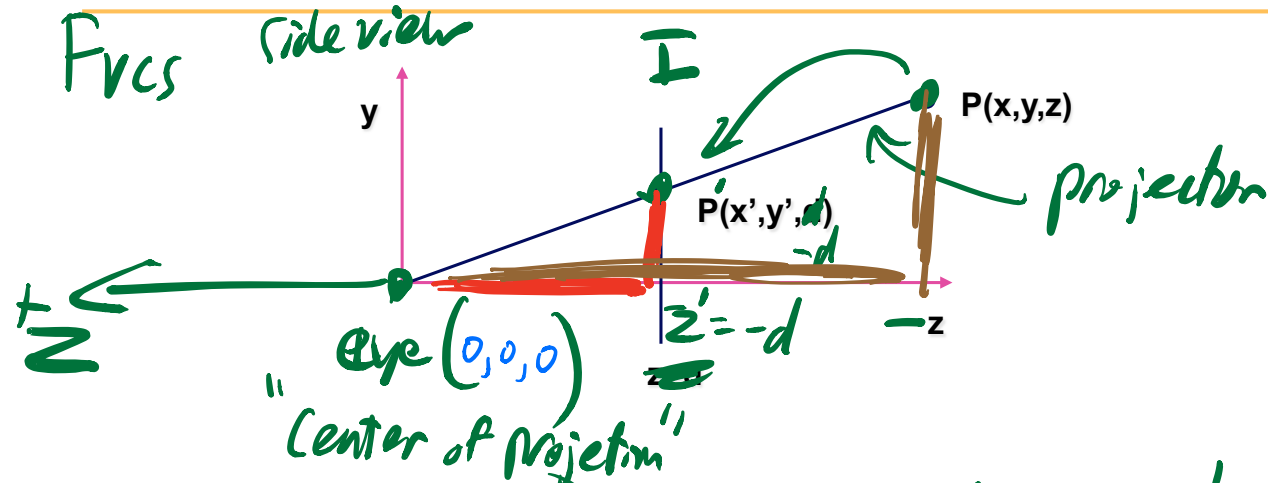


# parallel projections



not orthographic  
"oblique"

# Perspective Projection



$$\frac{y'}{z'} = \frac{y}{z}$$

$$y' = y \cdot \frac{z'}{z} = y \cdot \frac{(-d)}{z}$$

$$= y \cdot \left( \frac{-d}{z} \right) \quad h$$

# Homogeneous Coordinates

homogeneous

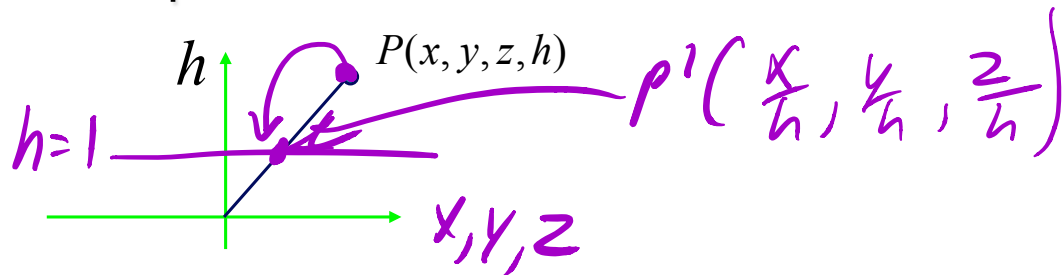
cartesian

$$(x, y, z, h) \longrightarrow \left( \frac{x}{h}, \frac{y}{h}, \frac{z}{h} \right)$$

Handwritten examples:

$$\begin{aligned} (1, 1, 1, 2) &\longrightarrow (0.5, 0.5, 0.5) \\ (5, 5, 5, 10) &\longrightarrow (0.5, 0.5, 0.5) \end{aligned}$$

- redundant representation
- $h=0$ : point at infinity (direction)
- geometric interpretation



# Perspective Projection

A simple version of  $M_{proj}$

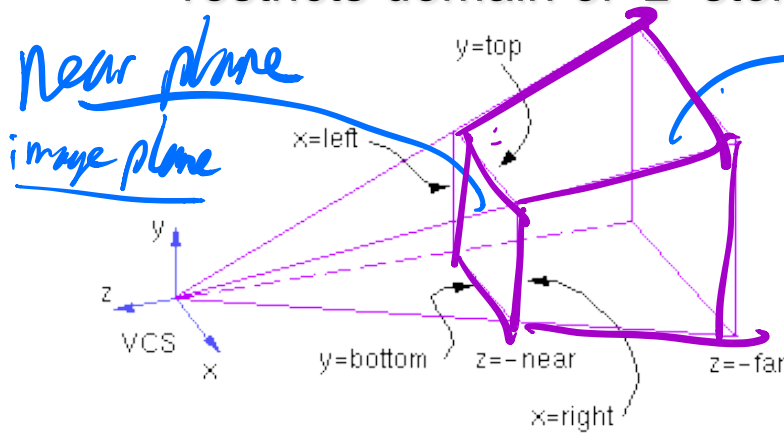
$$h \begin{bmatrix} x \\ y \\ z \\ -z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1/d & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad h=1$$

$/h$

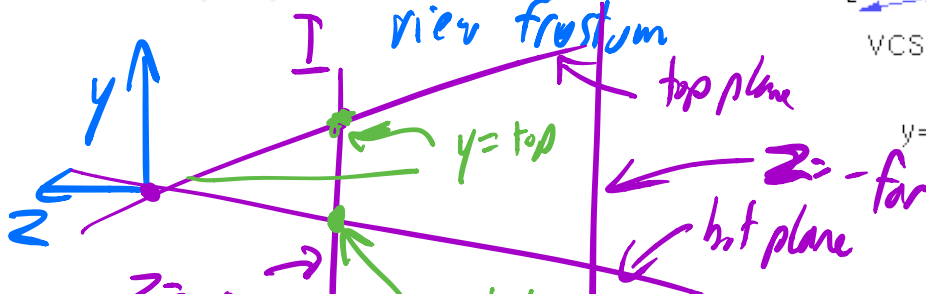
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} -d x/z \\ -d y/z \\ -d z/z \end{bmatrix}$$

# View Volumes: more about $M_{proj}$

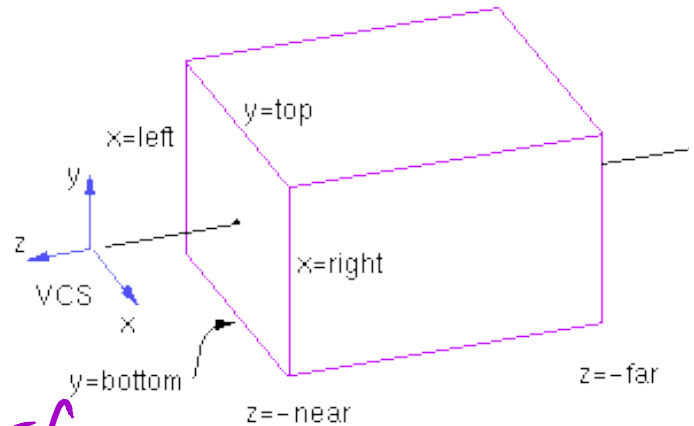
- specifies field-of-view, used for clipping
- restricts domain of  $z$  stored for visibility test



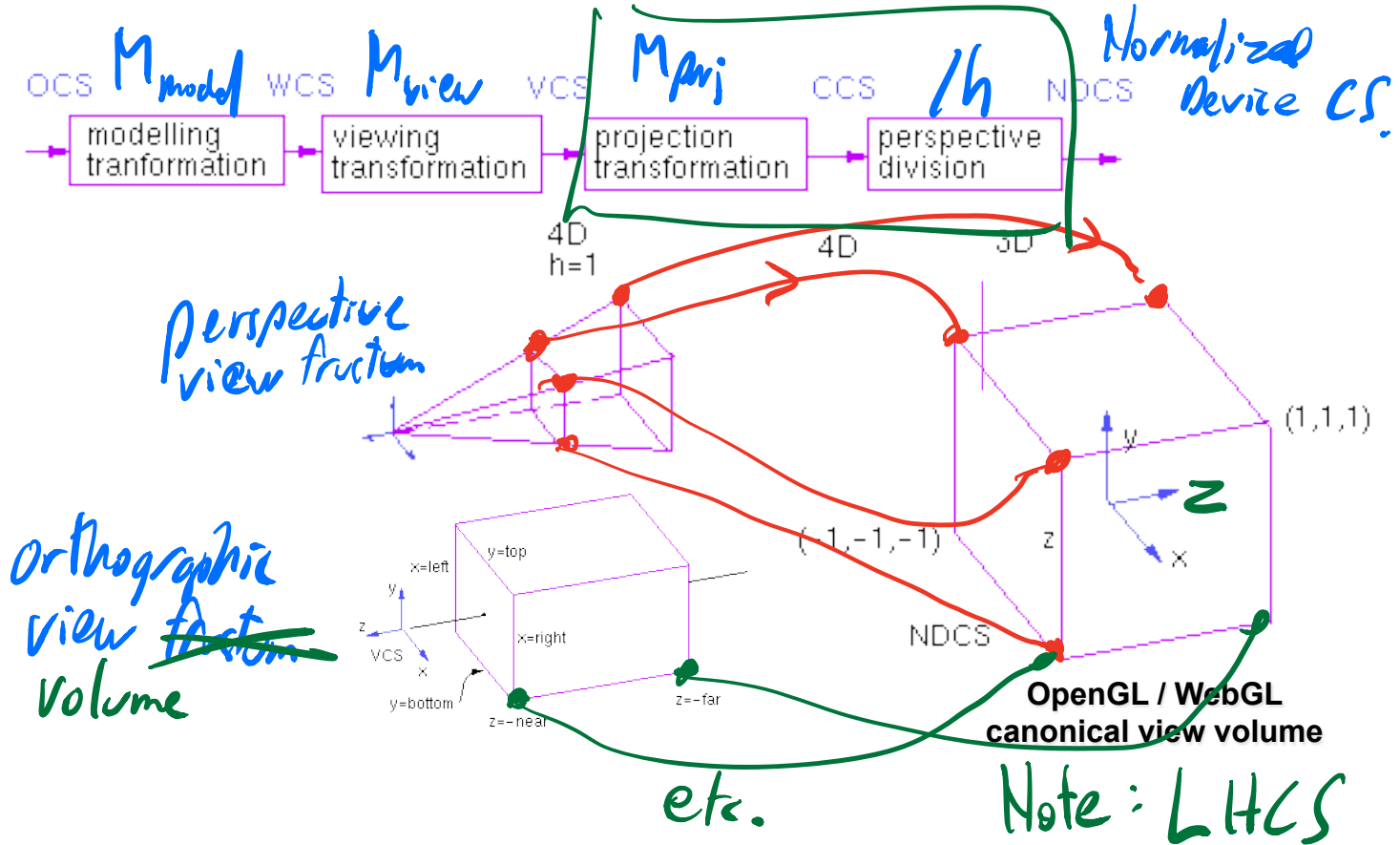
perspective view volume



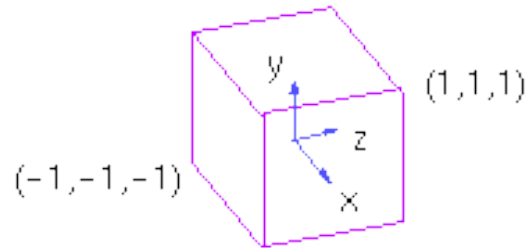
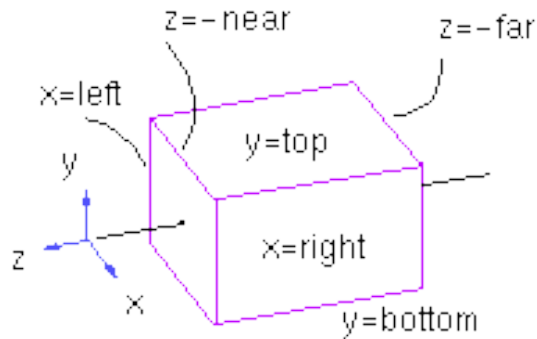
orthographic view volume



# View Volumes

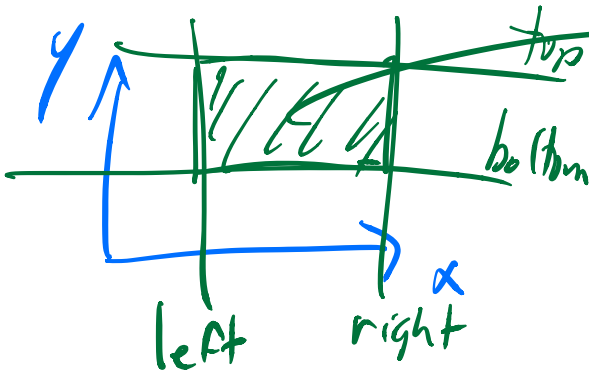


# Orthographic View Volume



VCS

NDCS



# Orthographic View Volume

$\left. \begin{matrix} a \\ b \end{matrix} \right\}$  see next page

$$P_{VCS} = \begin{bmatrix} \frac{2}{right - left} & \frac{2}{top - bot} & \frac{right + left}{right - left} & \frac{top + bot}{top - bot} \\ \frac{-2}{far - near} & \frac{far + near}{far - near} & & 1 \end{bmatrix} P_{VCS}$$

$M_{proj}$

↳ really just a scale and translate.

three.js

```
var cam = new THREE.OrthographicCamera(left, right, top, bot, near, far)
```



# Orthographic View Volume

---

## Derivation

linear transformation

$$y' = ay + b$$

$$\begin{aligned} 1 &= a(\text{top}) + b \\ -1 &= a(\text{bot}) + b \end{aligned}$$

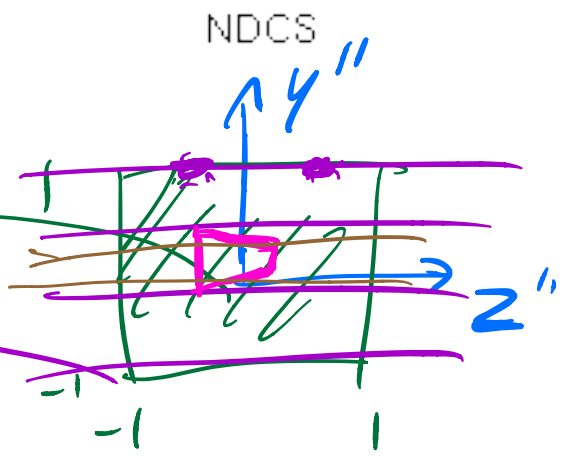
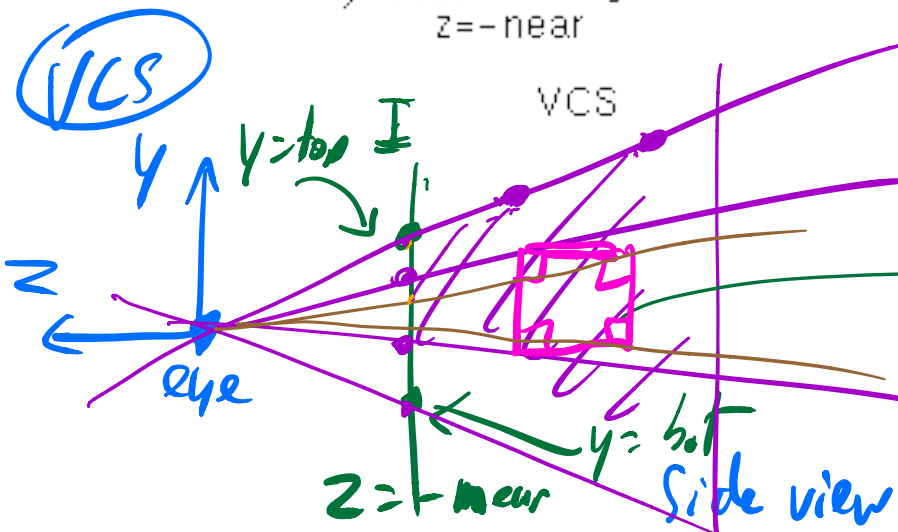
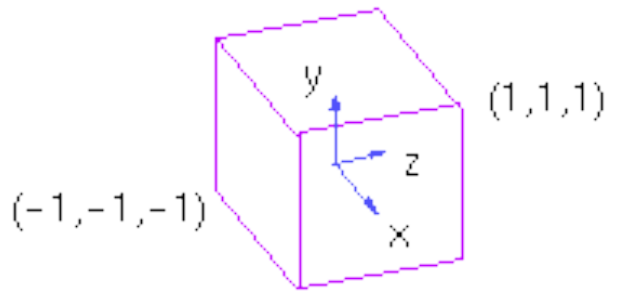
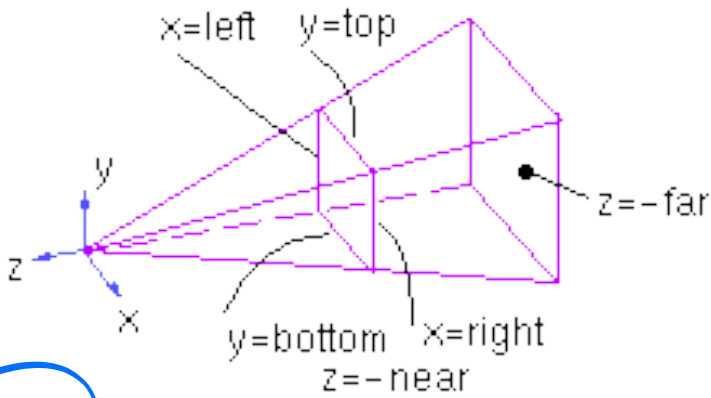
2 equations  
2 unknowns

solving for a and b gives:

$$a = \frac{2}{\text{top} - \text{bot}}$$

$$b = \frac{-(\text{top} + \text{bot})}{\text{top} - \text{bot}}$$

# Perspective View Volume



# Perspective View Volume

## Derivation

earlier:

$$\begin{array}{c}
 \left[ \begin{array}{c} -\frac{dx}{z} \\ -\frac{dy}{z} \\ -d \end{array} \right] \xrightarrow{1/h} \left[ \begin{array}{c} x \\ y \\ z \\ -z/d \end{array} \right] = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -1/d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \\
 \text{NDCS} \qquad \text{CCS} \qquad \qquad \qquad M_{proj} \qquad \qquad \qquad \text{VCS}
 \end{array}$$

with additional ability to scale, etc.:

$$\begin{array}{c}
 \left[ \begin{array}{c} -\frac{Ex}{z} - A \\ -\frac{Fy}{z} - B \\ -C - \frac{D}{z} \end{array} \right] \xrightarrow{1/h} \left[ \begin{array}{c} Ex + Az \\ Fy + Bz \\ Cz + D \\ -z \end{array} \right] = \begin{bmatrix} E & & & \\ & F & B & \\ & & C & D \\ & & & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \\
 \text{NDCS} \qquad \qquad \qquad \text{CCS} \qquad \qquad \qquad M_{proj} \qquad \qquad \qquad \text{VCS}
 \end{array}$$

# Perspective View Volume

*n = near    f = far  
t = top  
b = bot*

view volume  
left = -1, right = 1  
bot = -1, top = 1  
near = 1, far = 4

$$\begin{bmatrix} \frac{2n}{r-l} & \frac{r+l}{r-l} & 0 & 0 \\ 0 & \frac{t+b}{t-b} & 0 & 0 \\ \frac{2fn}{f-n} & \frac{-(f+n)}{f-n} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

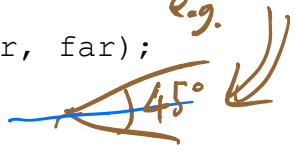
*Handwritten annotations: The first two columns are boxed. The first box contains  $\frac{2n}{t-b}$  and is labeled 'F'. The second box contains  $\frac{t+b}{t-b}$  and is labeled 'B'. Lines connect these boxes to the corresponding terms in the matrix above.*

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -5/3 & -8/3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**three.js**

```
var camera = new THREE.PerspectiveCamera(fov, aspect, near, far)
// which eventually calls:
//     matrix.makePerspective(left, right, top, bottom, near, far);
```

*"field of view" in the vertical direction, e.g. 45°*



# Perspective View Volume

## Derivation

top plane:

$$y_{wvcs} = y^u = 1$$

$$y_{vcs} = \left( \frac{\text{top}}{-\text{near}} \right) z_{vcs}$$

$$\left. \begin{aligned} 1 &= -\frac{Fy}{z} - B \\ 1 &= -F \left( \frac{\text{top}}{-\text{near}} \right) - B \\ -1 &= -F \left( \frac{\text{bot}}{-\text{near}} \right) - B \end{aligned} \right\}$$

repeat for bot plane to get another eqn,  
then solve for F and B

similar process for solving for the other unknowns,  
using the left/right and near/far planes

Solve for F, B

(not covered in class, but still relevant)

# Perspective Projection -- Example

**Example** : looking at a railway track

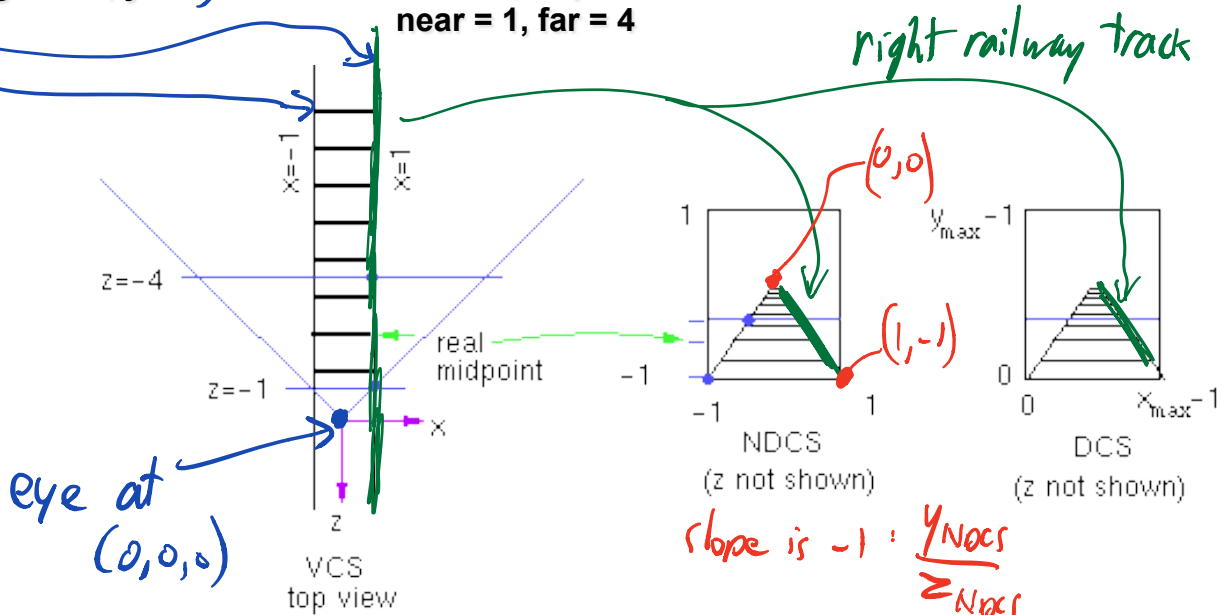
tracks in VCS:

left  $x=-1, y=-1$   
right  $x=1, y=-1$

$z \in [-\infty, \infty]$

view volume

left = -1, right = 1  
bot = -1, top = 1  
near = 1, far = 4



slope is  $-1 = \frac{y_{NDCS}}{z_{NDCS}}$

$(0,0) - (1,-1)$

# Perspective Projection -- Example

## Example

$$\begin{bmatrix} 1 \\ -1 \\ -\frac{5}{3}z - \frac{8}{3} \\ -z \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -\frac{5}{3} & -\frac{8}{3} \\ -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ z \\ 1 \end{bmatrix} \text{ vcs}$$

*right track*

*bottom right corner*

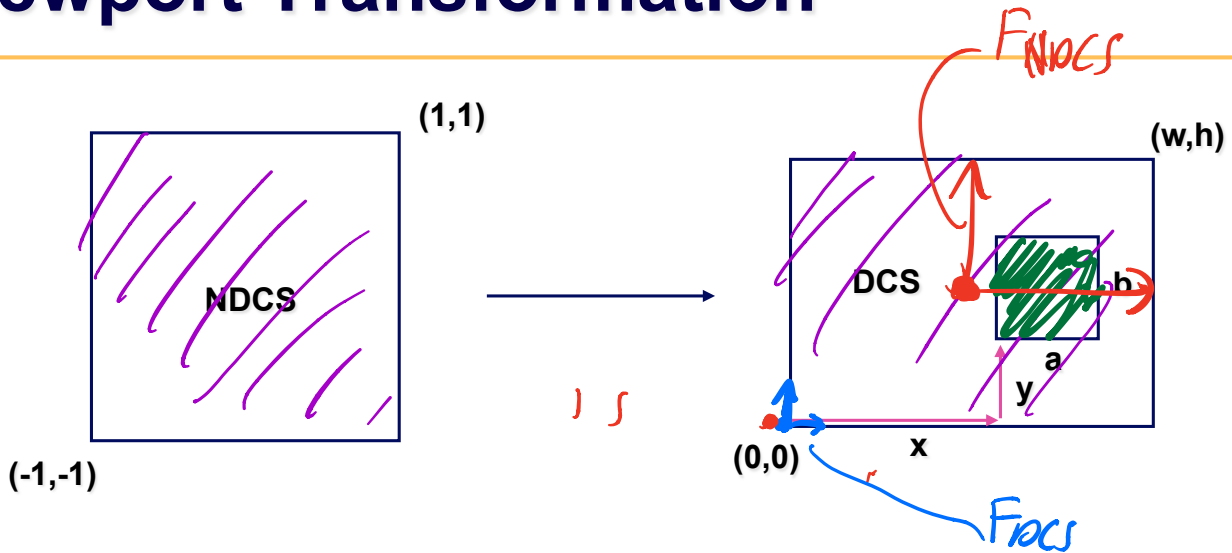
*center*

$$\xrightarrow{/h} \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ \frac{5}{3} + \frac{8}{3} \frac{1}{2} \end{bmatrix} \text{ NACS}$$

$$\begin{matrix} z = -1 \\ z = -\infty \end{matrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 0^+ \\ 0^- \\ \frac{5}{3} \end{bmatrix}$$

The diagram illustrates the process of converting a perspective projection matrix into Normalized Affine Camera Space (NACS). It starts with a 4x1 vector representing the projection of a point (1, -1, z, 1) through a camera matrix. The bottom element of this vector, -z, is circled in green and labeled 'right track'. An arrow labeled 'h' points to the NACS matrix, which is a 3x1 vector. The NACS matrix is then used to project points from the image plane (z = -1 and z = -∞) back into the original 3D space, resulting in two 3x1 vectors. The top vector is labeled 'bottom right corner' and the bottom vector is labeled 'center'.

# Viewport Transformation



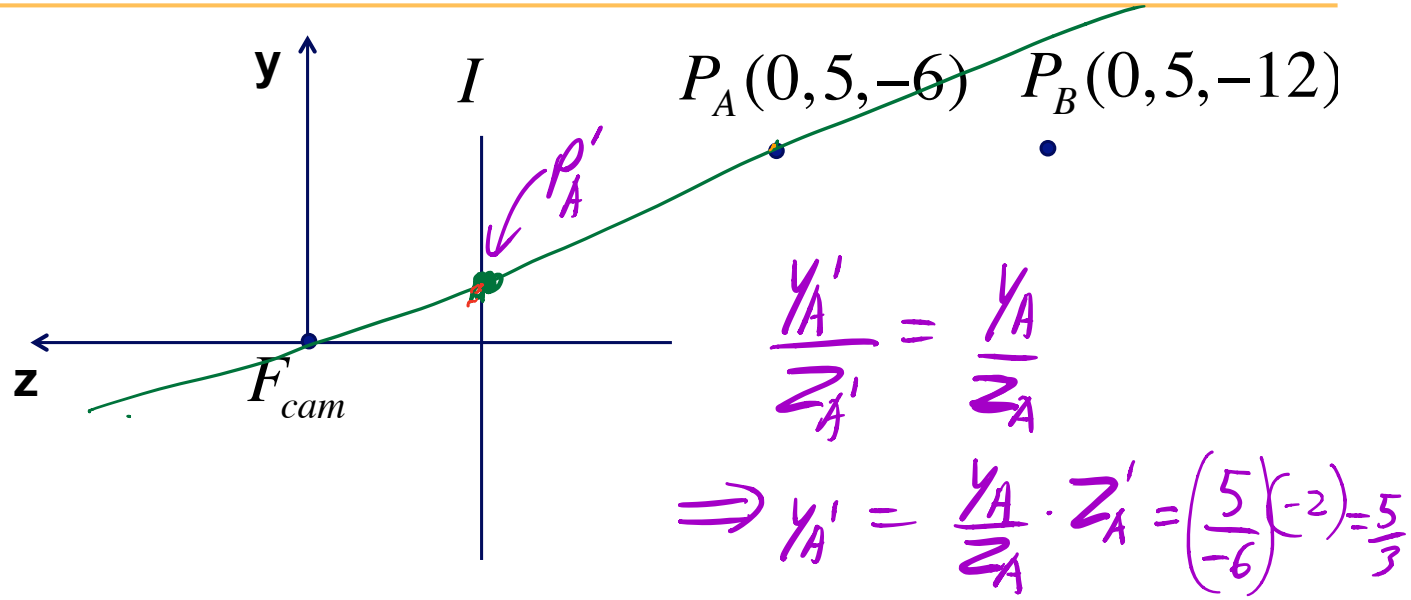
three.js: `renderer.setViewport(x,y,a,b);`  
 WebGL: `gl.viewport(x,y,a,b);`  
`gl.viewport(0,0,w,h)` is default

$$P_{DCS} = \text{Transl}\left(\frac{w}{2}, \frac{h}{2}, 0\right) \text{Scale}\left(\frac{w}{2}, \frac{h}{2}, 1\right) P_{NDCS}$$



Name \_\_\_\_\_

# Perspective Projection -- example

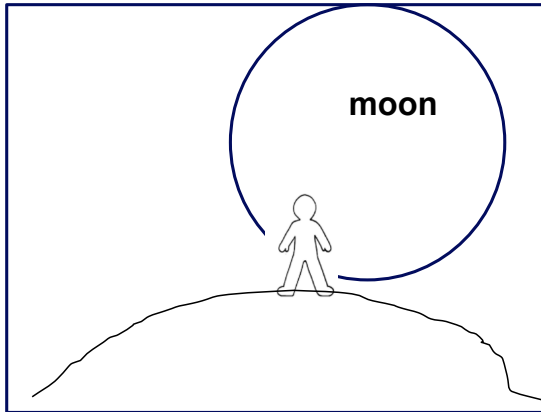


Compute the projected coordinates of the given points for a perspective projection. The image plane is located at  $z = -2$ .

In which direction should we move the image plane in order to obtain a larger image?

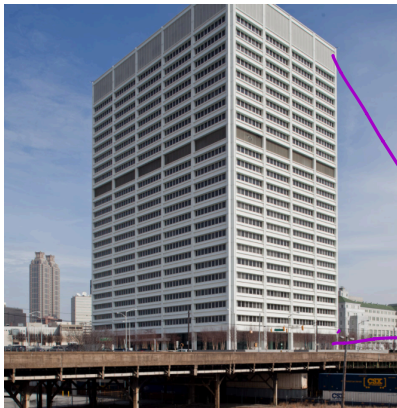
Move it further to the right away from the center of projection. Equivalent to increasing the focal length of a lens; i.e., telephoto lens

# Impossible Photography?



How could we take a photograph like the one on the left?

Stand very far away, which makes the rays more parallel, and therefore the objects have proportions closer to their <sup>relative</sup> actual relative proportions.



The edges of the building on the left are parallel, despite the viewer standing on the ground while taking the photograph. How is this possible?

Keep image plane vertical, i.e. point lens axis at the bottom of the building. Then either crop out the bottom half of the image, or use a tilt-shift camera.

