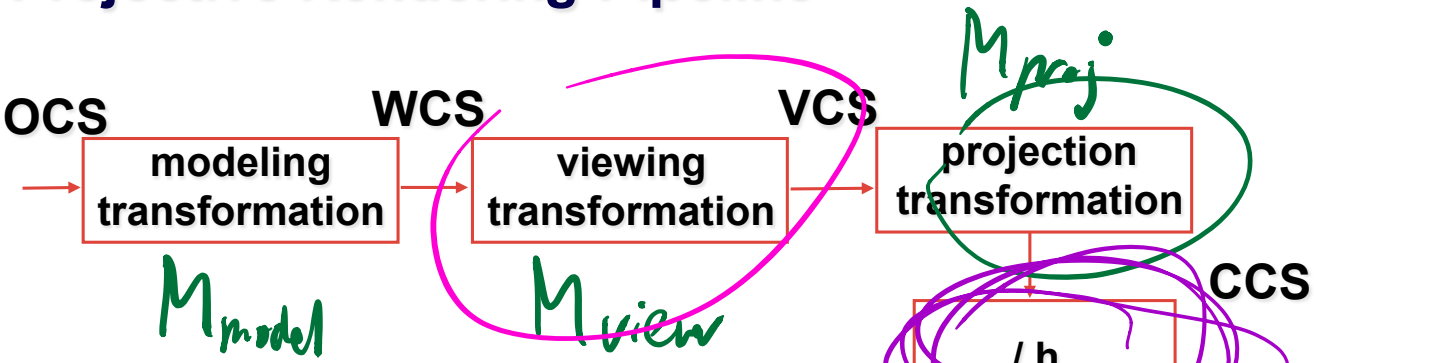
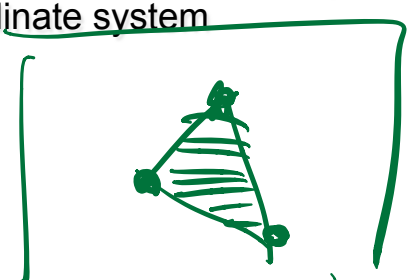
Viewing and Projection Transformations

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:

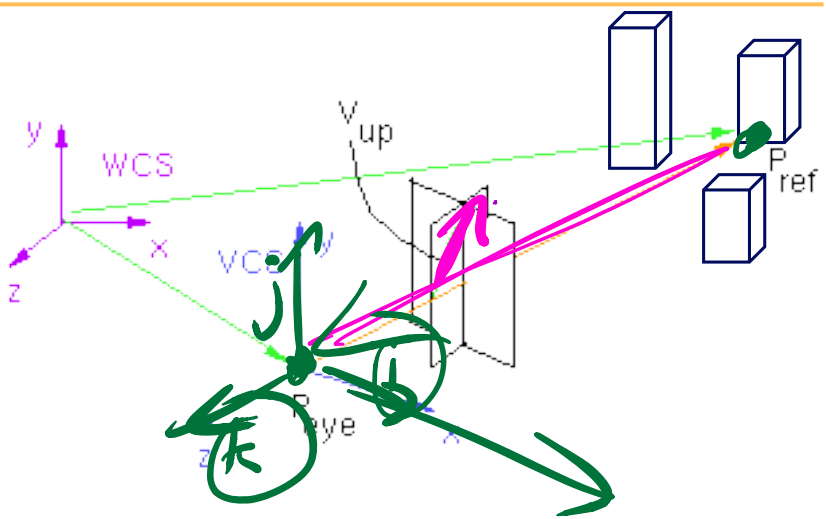
```
camera.position.set(0,12,20);   // eye
camera.up.set(0,1,0);   // up vector
camera.lookAt(0,0,0);   // specify target; compute M_view
   // internally:   object.matrix.lookAt(eye,target,up)
```

Computing i,j,k

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

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

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



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

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

M_{cam}

Viewing Transformation M_{view}

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

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

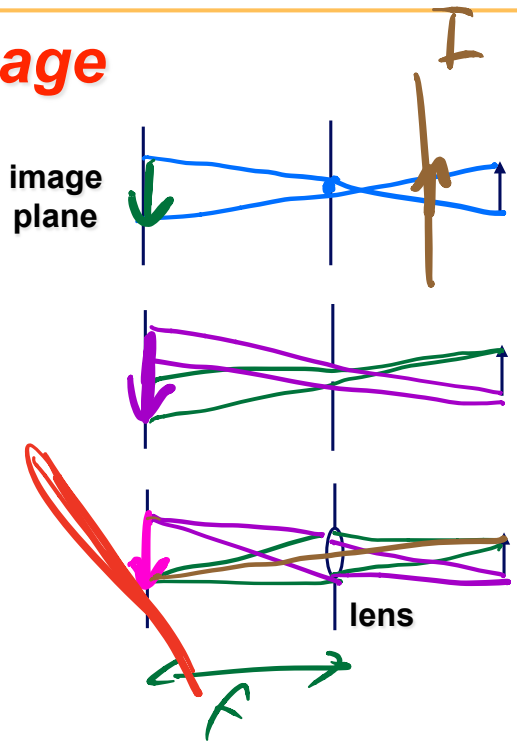
Projection Transformation M_{proj}

3D scene → 2D image

pinhole camera

real pinhole camera

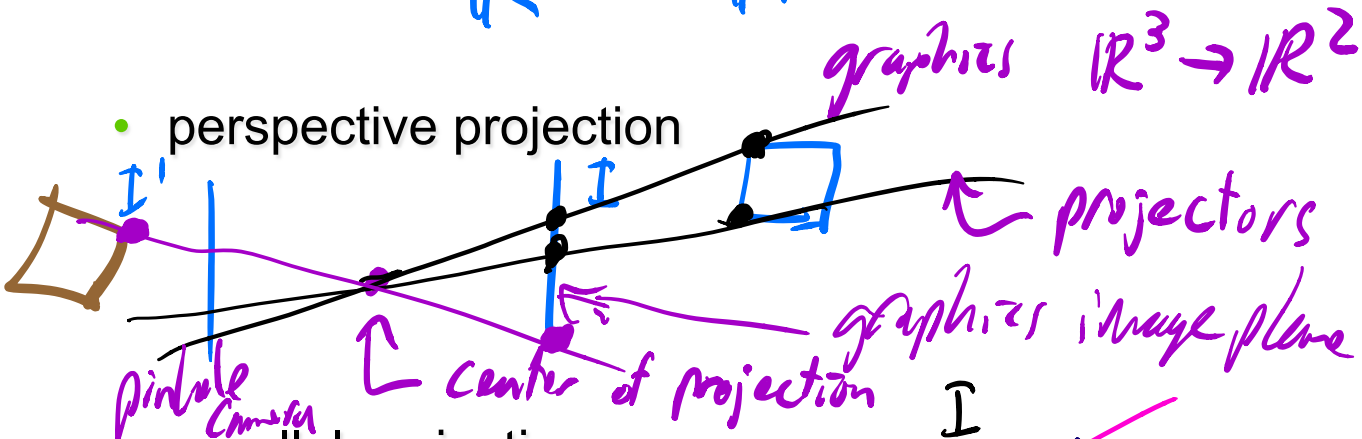
camera



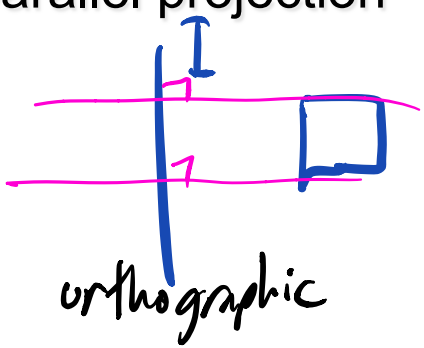
Projection

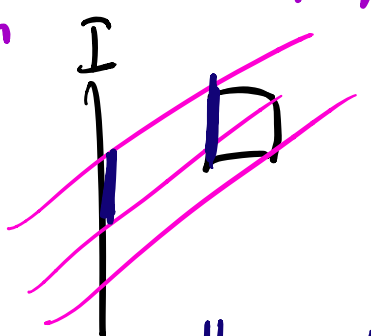
- definition

- perspective projection



- parallel projection

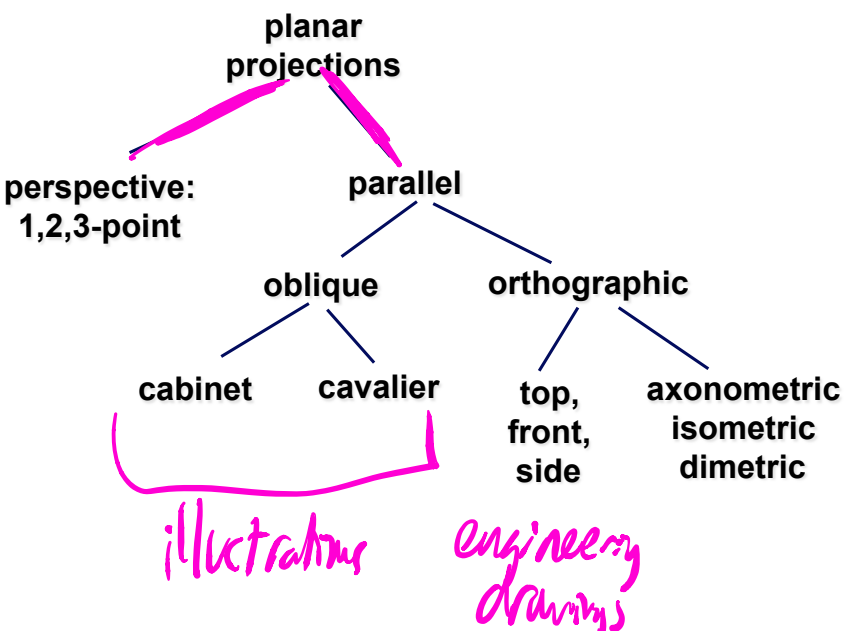




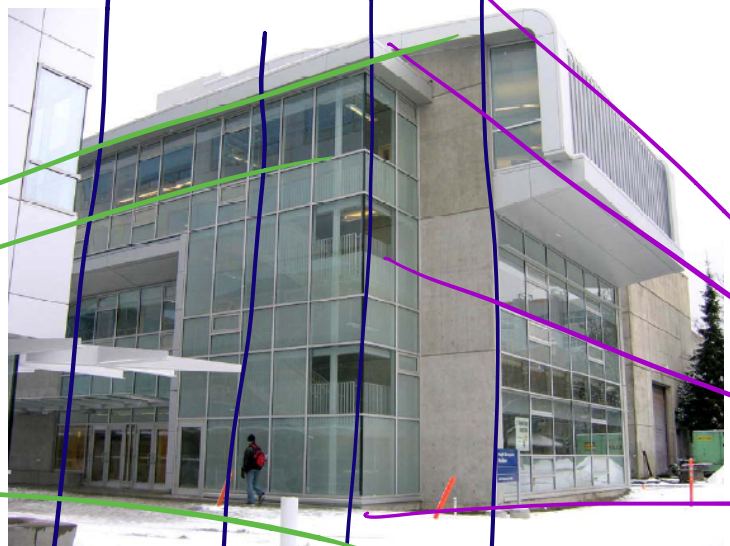
projectors are all parallel to each other

Projections

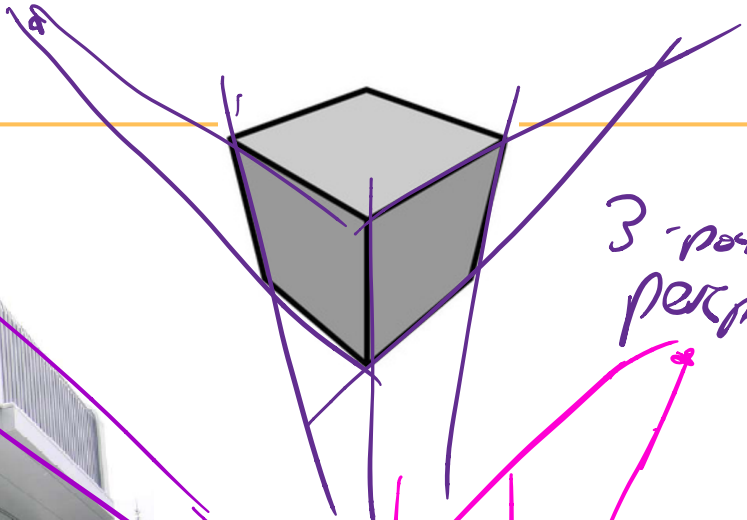
Taxonomy

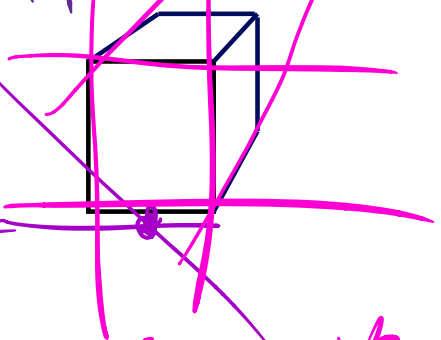


2-point perspective



3-point persp.

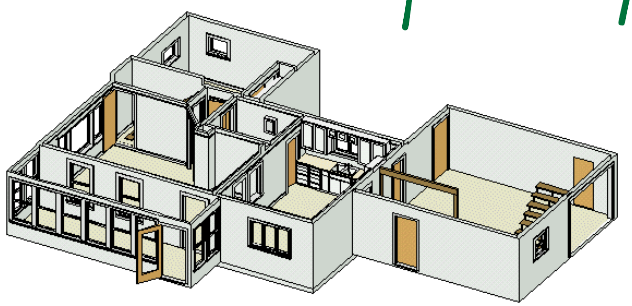


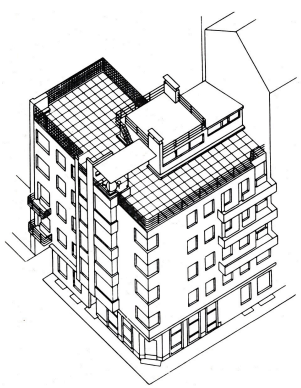


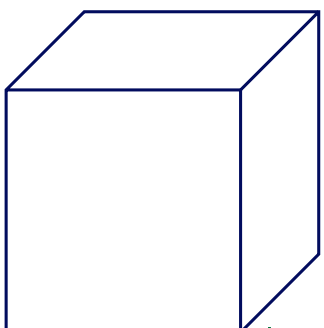
parallel

one-point perspective

parallel projections



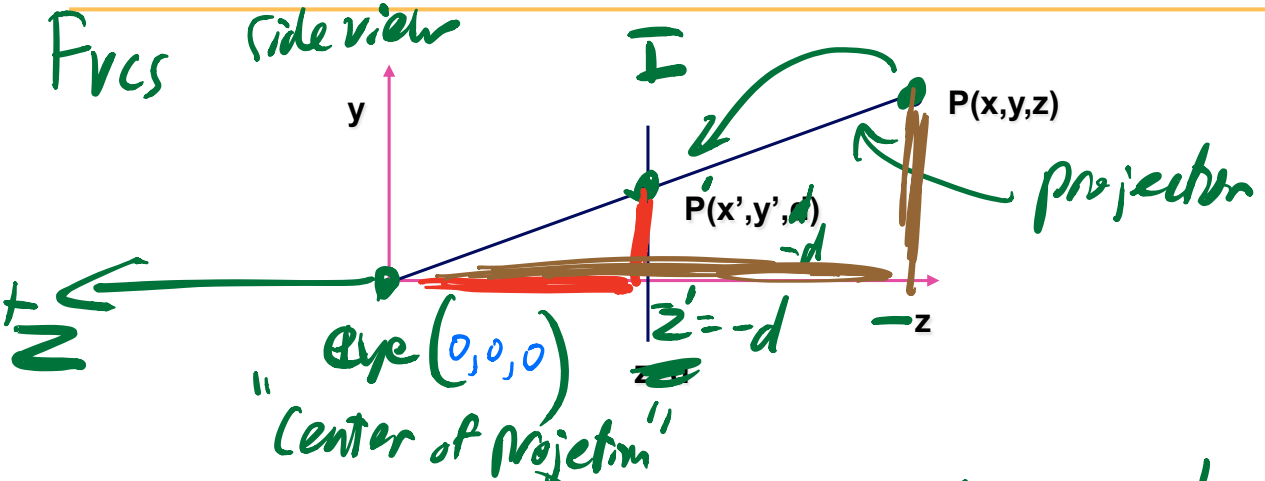




not orthographic

"oblique"

Perspective Projection



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

$$= y \frac{1}{(z/-d)}$$

Homogeneous Coordinates

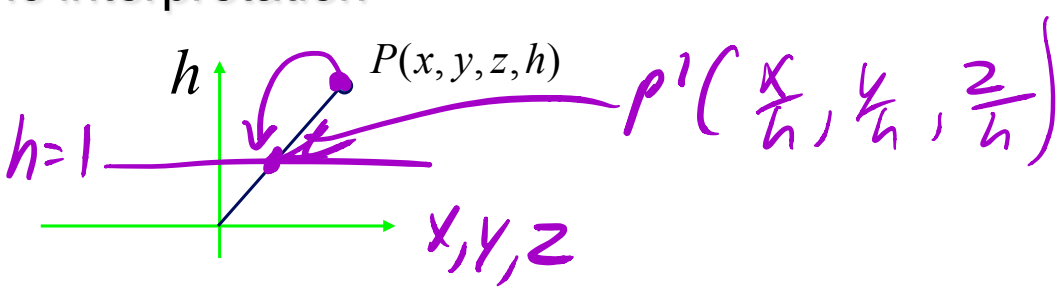
homogeneous **cartesian**

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

(1, 1, 1, 2) → (0.5, 0.5, 0.5)

(5, 5, 5, 10) → (0.5, 0.5, 0.5)

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



Perspective Projection

A simple version of M_{proj}

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

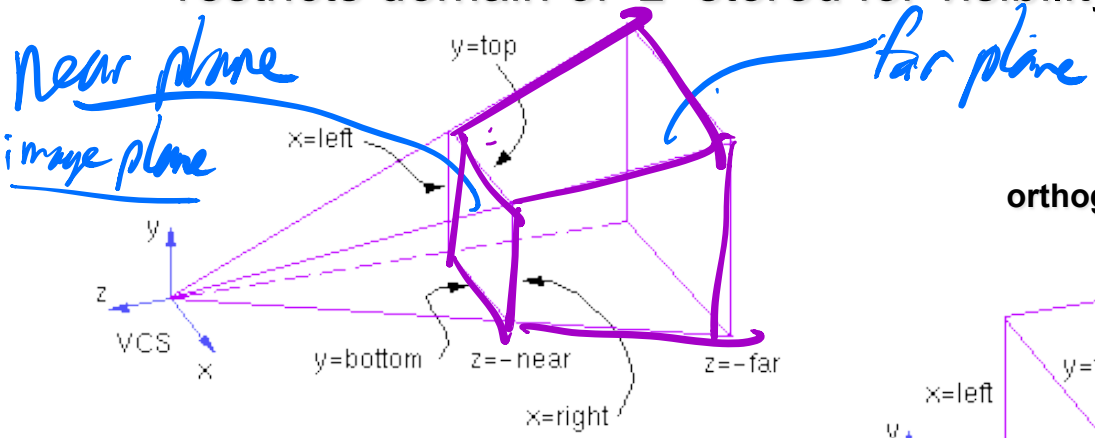
$h = -z/d$ $\qquad h = 1$

$/\, h$

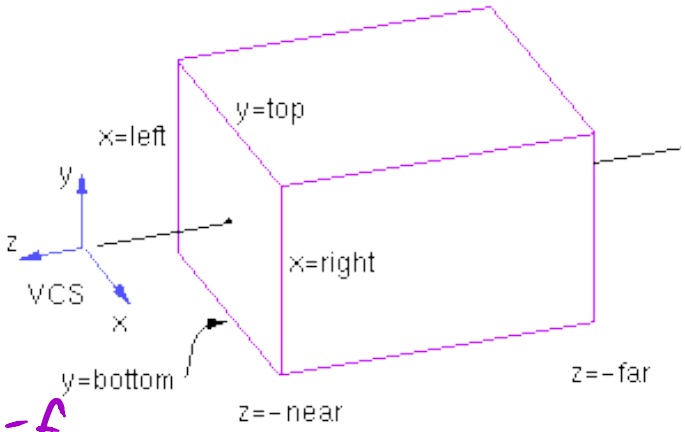
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} -d\,x/z \\ -d\,y/z \\ -d \end{bmatrix}$$

View Volumes: more about M_{proj}

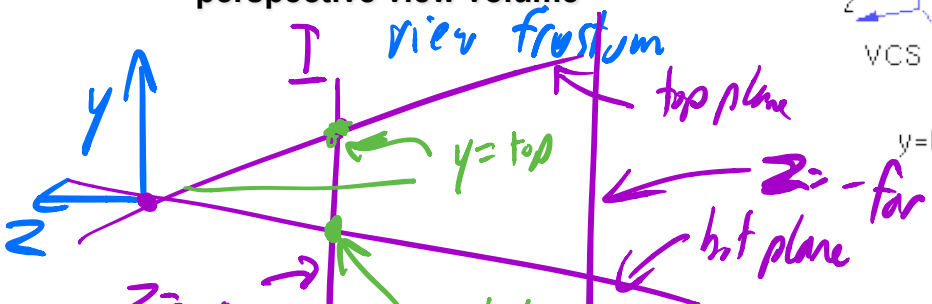
- specifies field-of-view, used for clipping
- restricts domain of ***z*** stored for visibility test



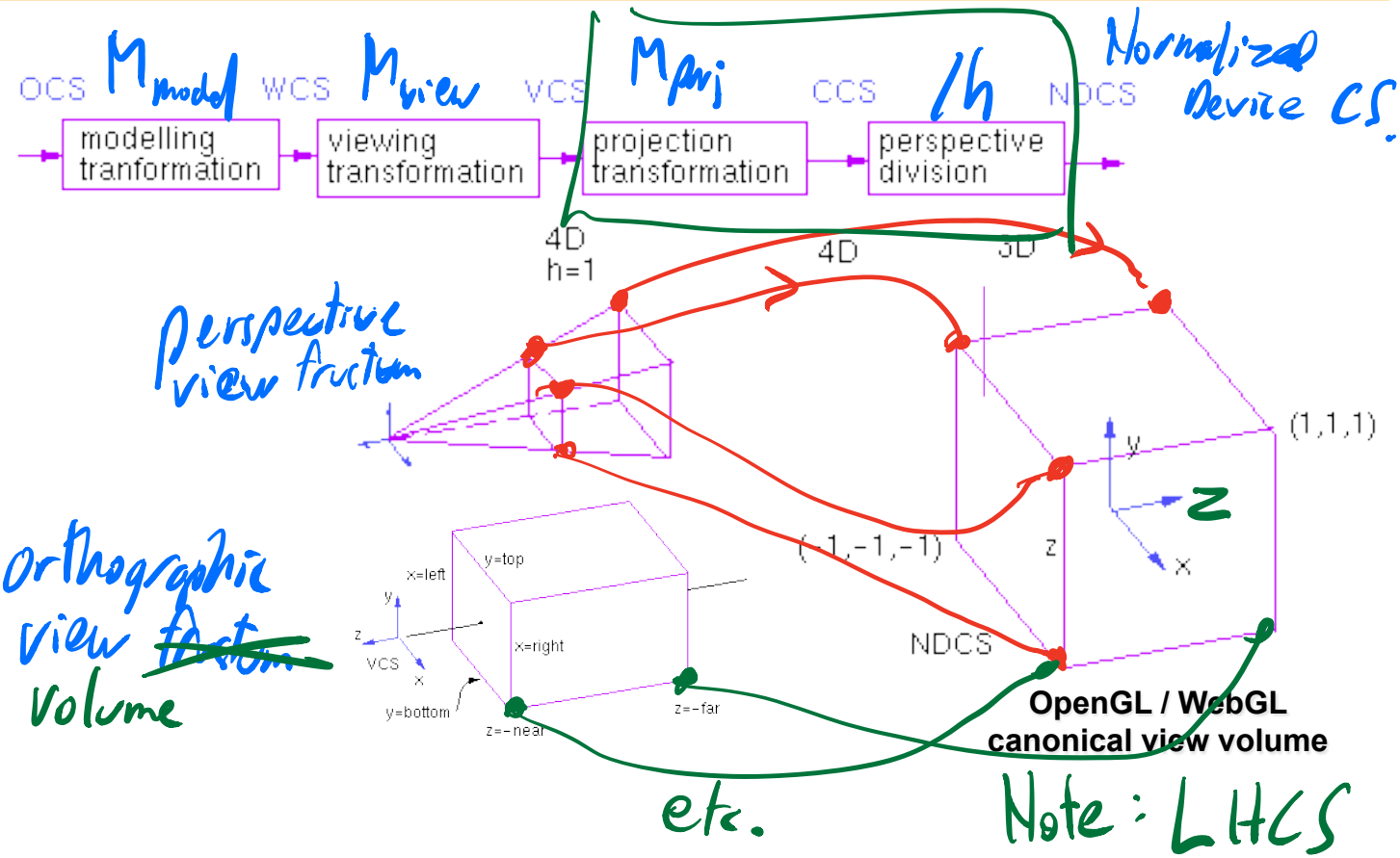
orthographic view volume



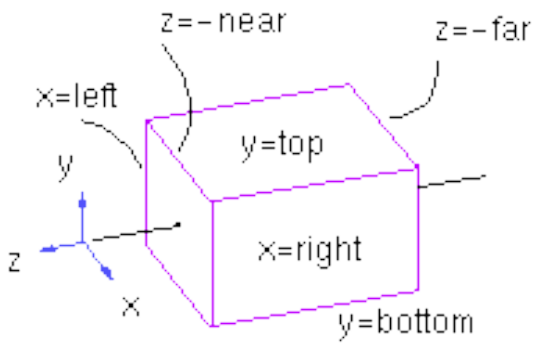
perspective view volume

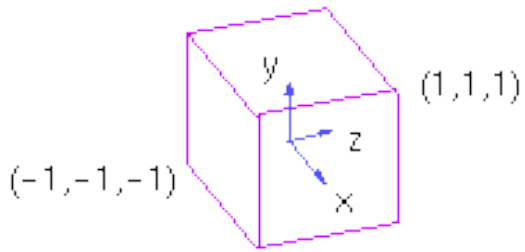


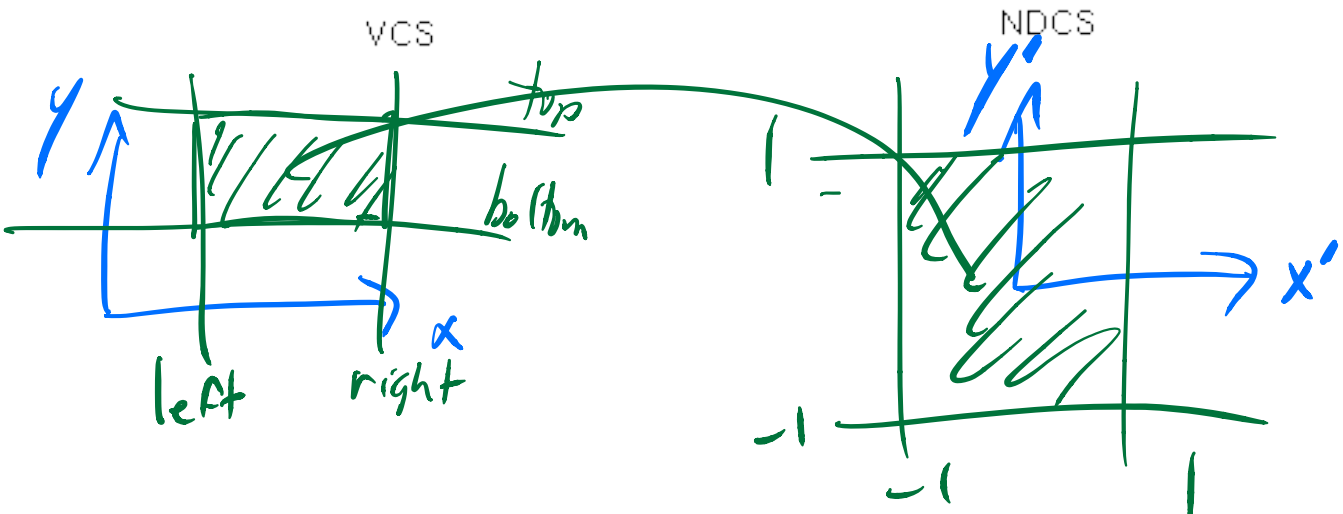
View Volumes



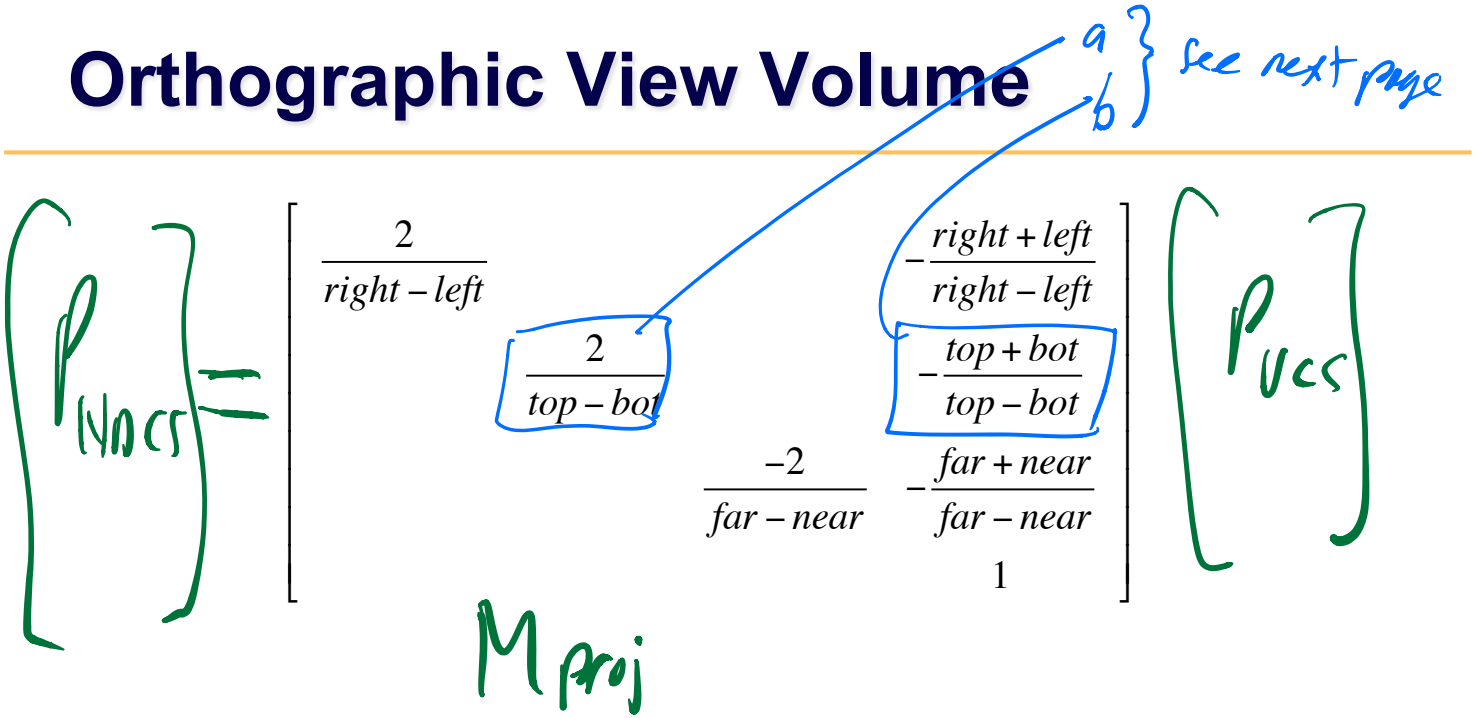
Orthographic View Volume







Orthographic View Volume



three.js

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

Orthographic View Volume

Derivation

linear transformation

$$y' = ay + b$$

$$1 = a(top) + b$$

$$-1 = a(bot) + b$$

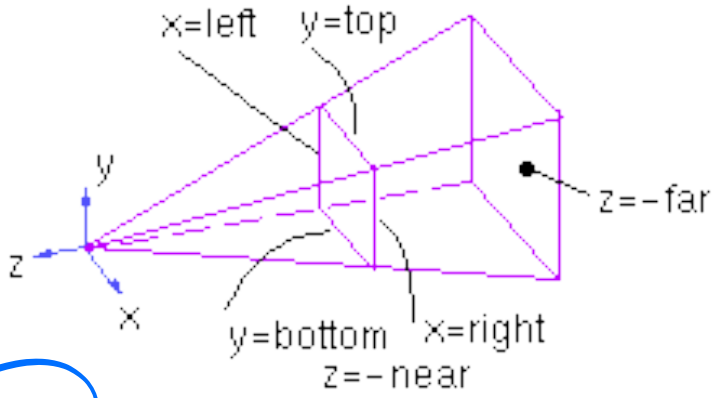
2 equations
2 unknowns

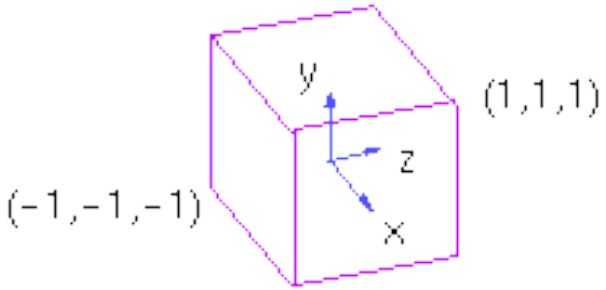
solving for a and b gives:

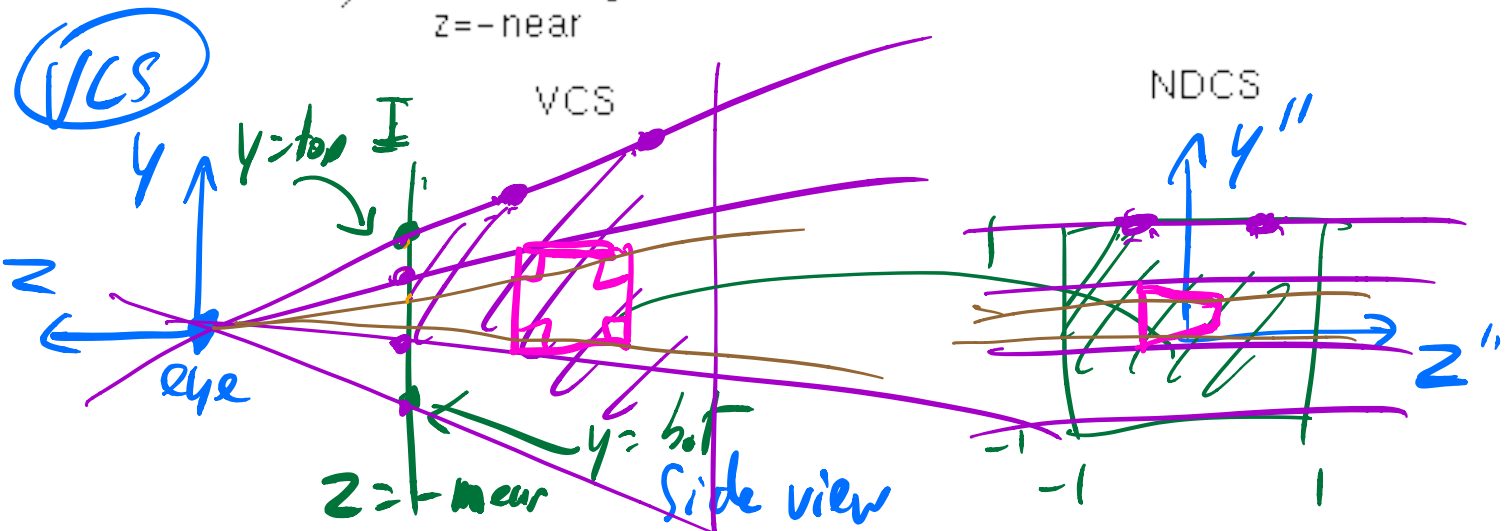
$$a = \frac{2}{top - bot}$$

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

Perspective View Volume







Perspective View Volume

Derivation

earlier:

$$\begin{bmatrix} -\frac{dx}{z} \\ -\frac{dy}{z} \\ -d \end{bmatrix}_{NDCS} \xleftarrow{/h} \begin{bmatrix} x \\ y \\ z \\ -z/d \end{bmatrix}_{CCS} = \underbrace{\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -1/d & \end{bmatrix}}_{M_{proj}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{VCS}$$

with additional ability to scale, etc.:

$$\begin{bmatrix} -\frac{Ex}{z} - A \\ -\frac{Fy}{z} - B \\ -C - \frac{D}{z} \end{bmatrix}_{NDCS} \xleftarrow{/h} \begin{bmatrix} Ex + Az \\ Fy + Bz \\ Cz + D \\ -z \end{bmatrix}_{h} = \begin{bmatrix} x' \\ y' \\ z' \\ h' \end{bmatrix}_{CCS} = \underbrace{\begin{bmatrix} E & & A & \\ F & B & & \\ & & C & D \\ & & -1 & \end{bmatrix}}_{M_{proj}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{VCS}$$

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

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

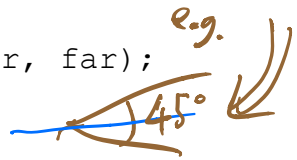
F B

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

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.



Perspective View Volume

Derivation

top plane:

$y_{NDCS} = y'' = 1$

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

$1 = -\frac{F y}{z} - B$

$1 = -F\left(\frac{top}{-near}\right) - B$

$-1 = -F\left(\frac{bot}{-near}\right) - B$

Solve for F, B

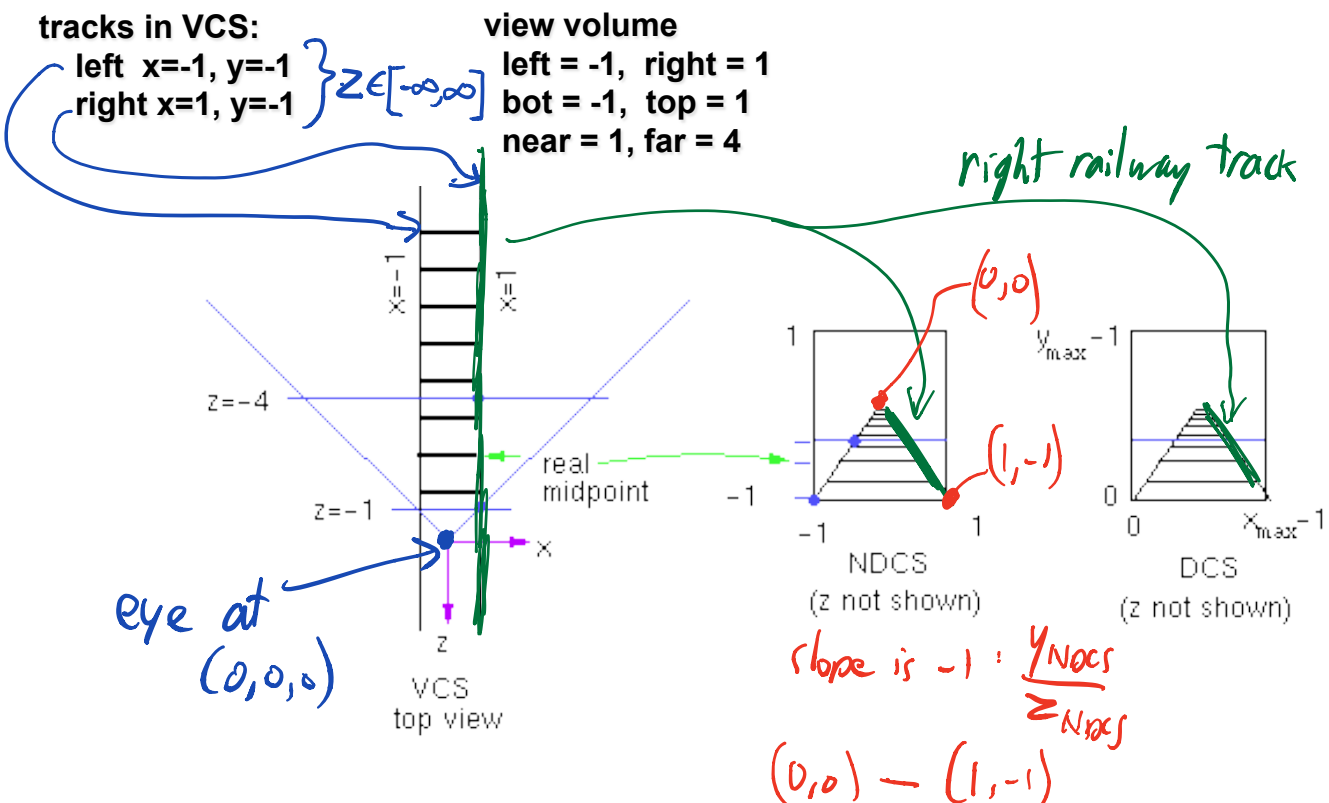
repeat for bot plane to get another eqn, then solve for F and B

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

(not covered in class, but still relevant)

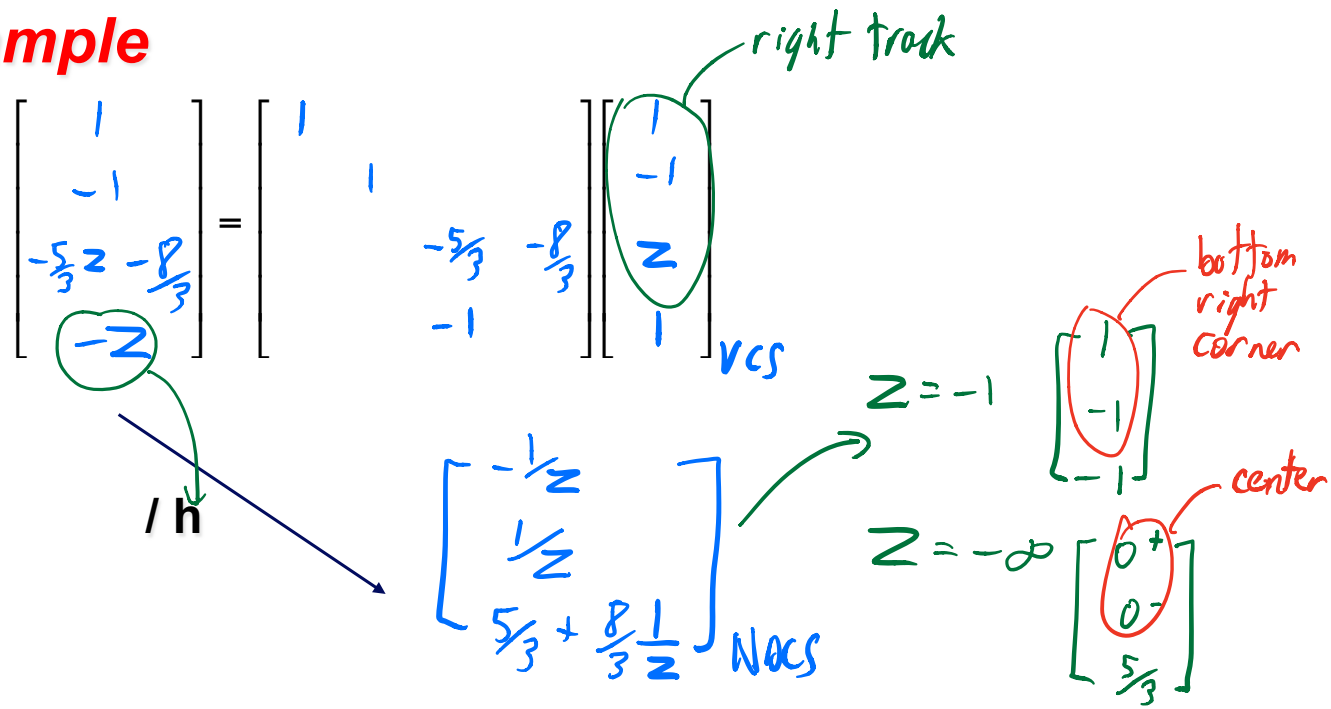
Perspective Projection -- Example

Example : looking at a railway track

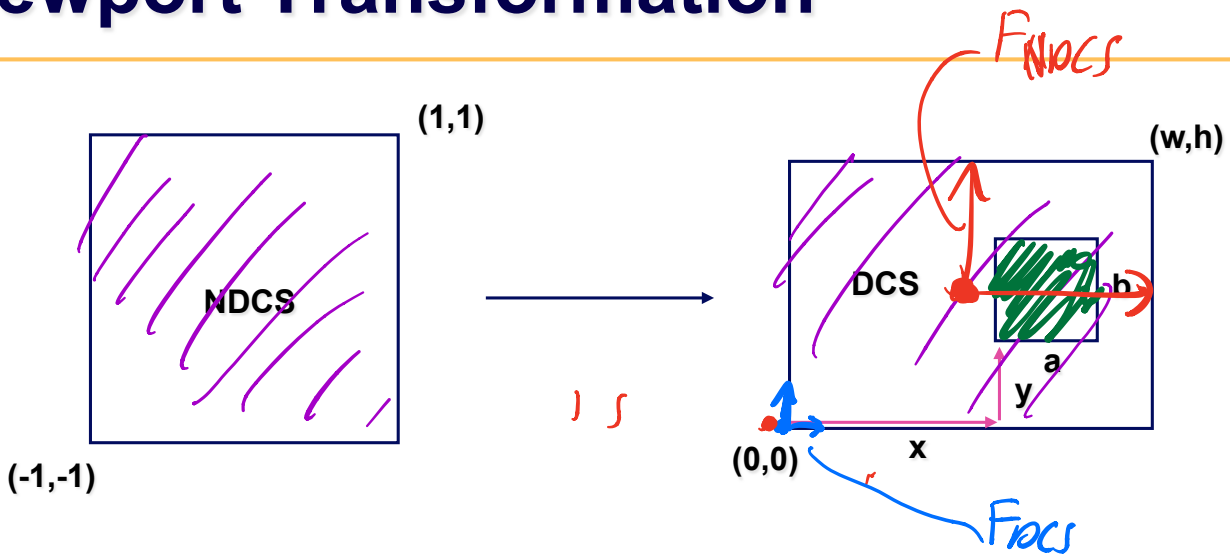


Perspective Projection -- Example

Example



Viewport Transformation

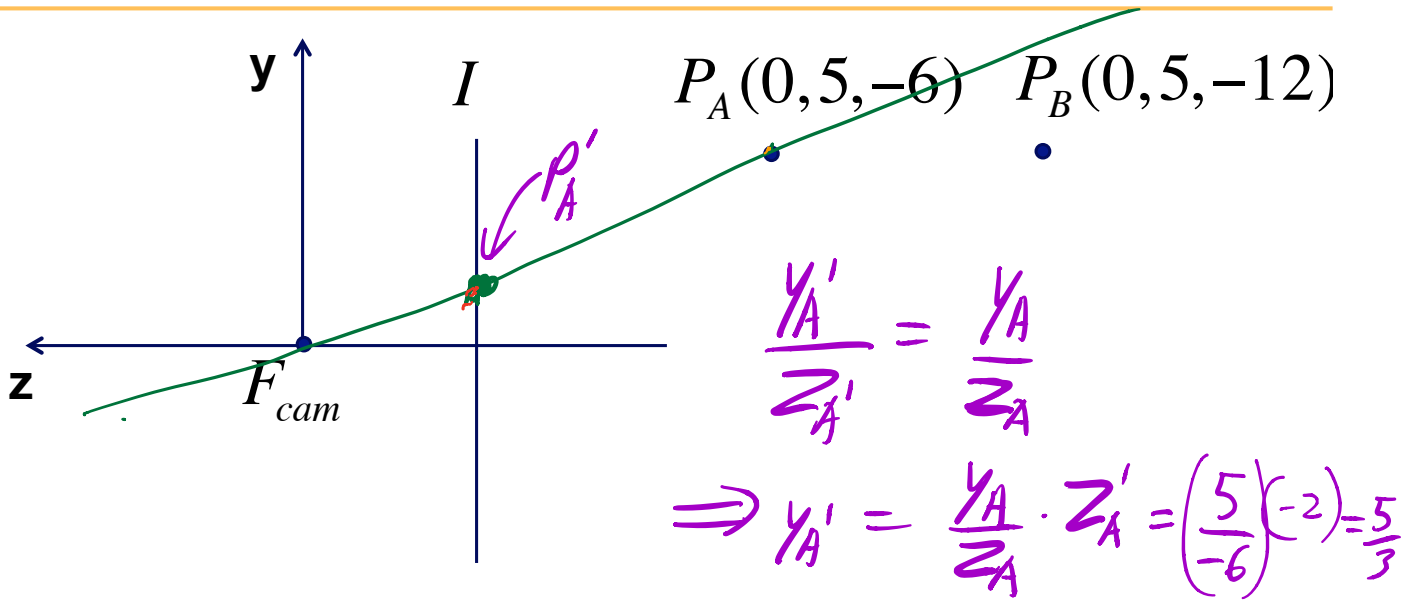



```
three.js:    renderer.setViewport(x,y,a,b);
WebGL:       gl.viewport(x,y,a,b);
             gl.viewport(0,0,w,h)   is default
```

$$P_{DCS} = Trans(w/2, h/2, 0)\, Scale(\tfrac{w}{2}, \tfrac{h}{2}, 1)\, P_{NDCS}$$

Name _____

Perspective Projection -- example

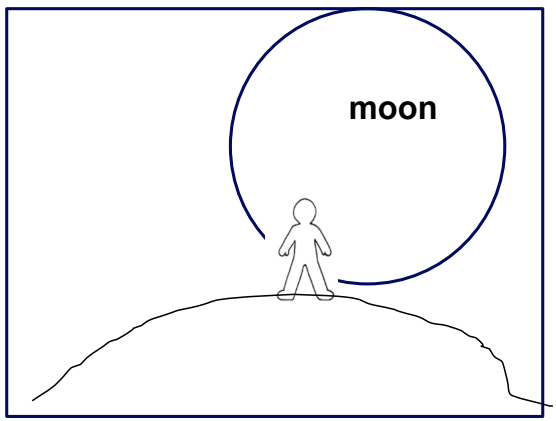


Compute the projected coordinates of the given points for a perspective projection.
The image plane is located at z = -2.

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

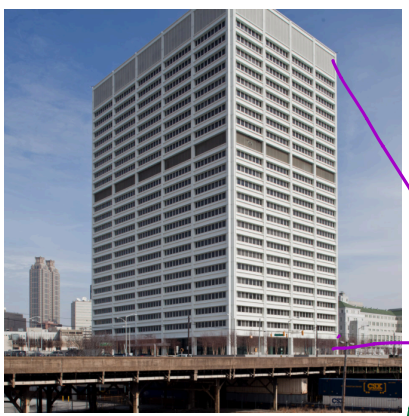
Move it further to the right away from the center of projection.
Equivalent to increasing the focal length of a lens; ie., telefoto lens

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 relative 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.