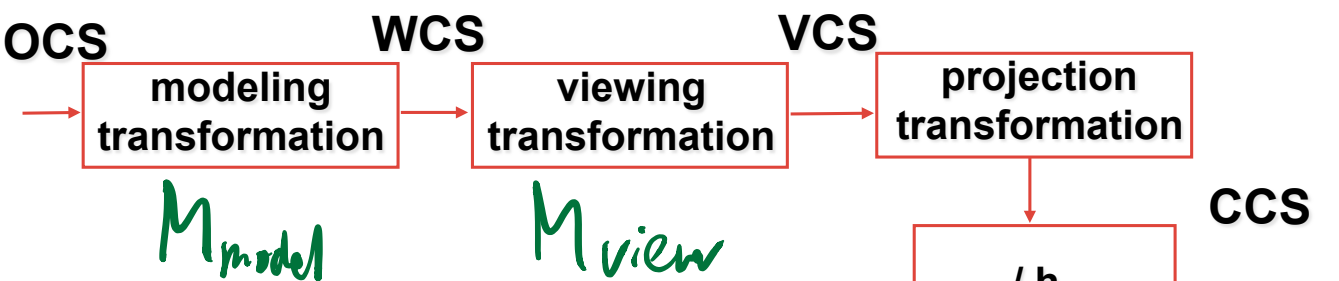
Viewing and Projection Transformations

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:

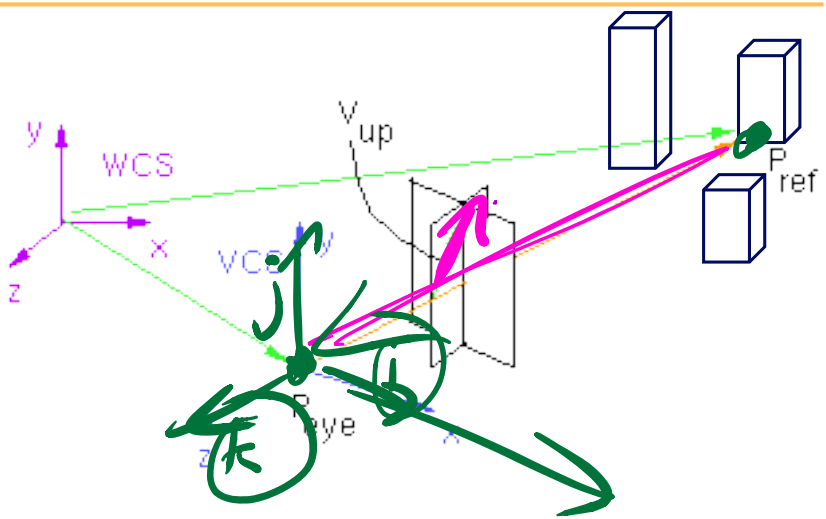
```
camera.position.set(0,12,20);   // eye
camera.up.set(0,1,0);   // up vector
camera.lookAt(0,0,0);   // specify target; compute M_view
   // internally:   object.matrix.lookAt(eye,target,up)
```

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}

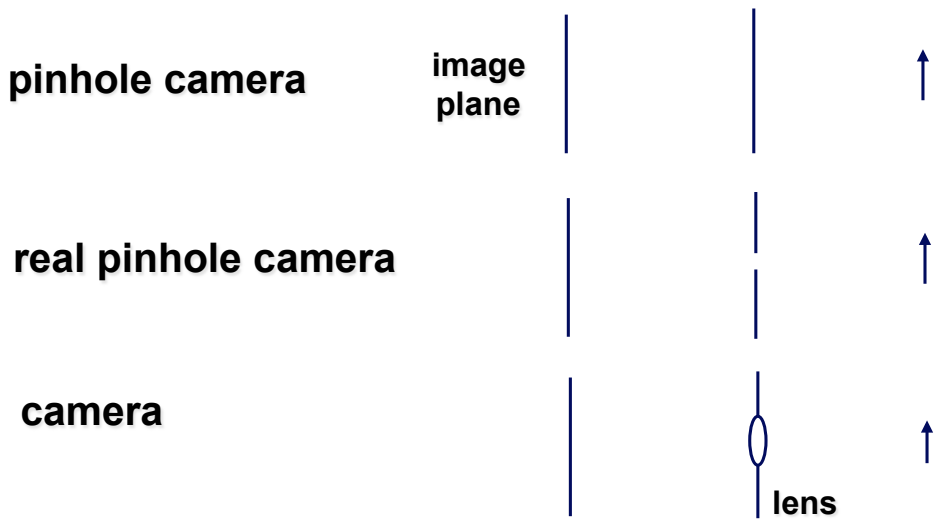
Viewing Transformation M_{view}

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

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

Projection Transformation M_{proj}

3D scene → 2D image

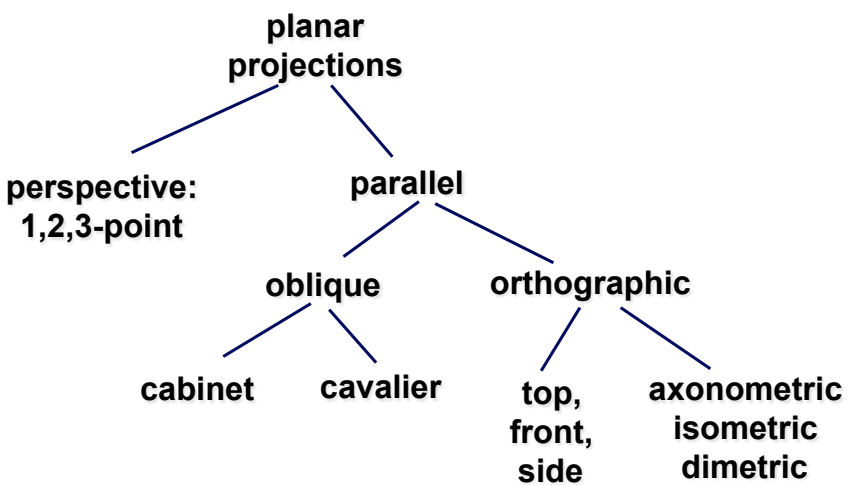


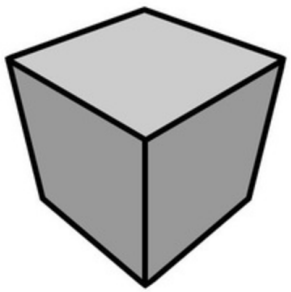
Projection

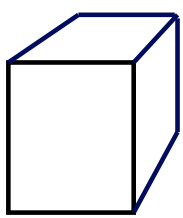
- definition
- perspective projection
- parallel projection

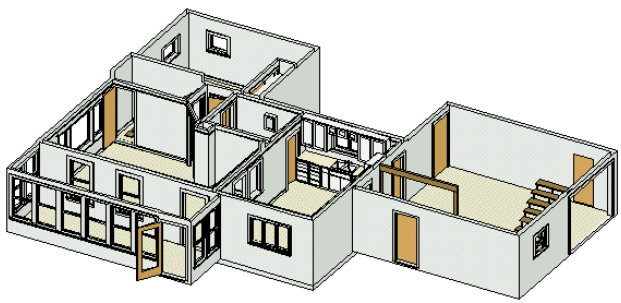
Projections

Taxonomy

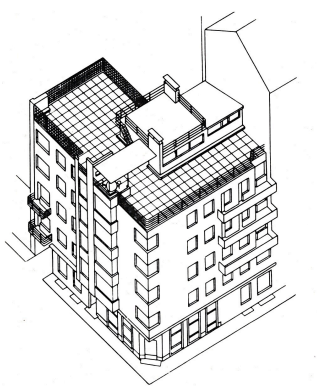


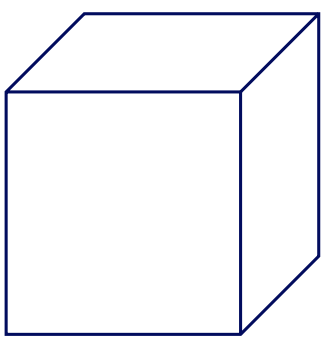




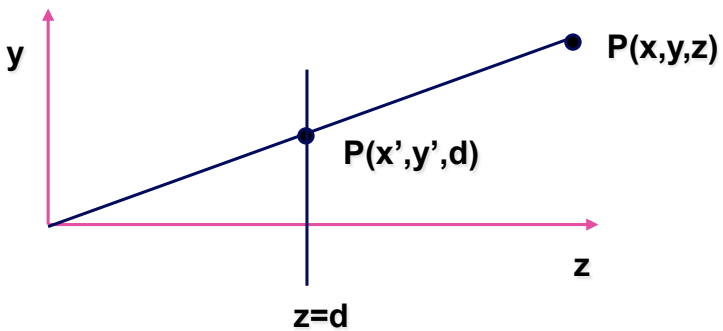



Overall Damage
0%
2M 10s
Replay





Perspective Projection

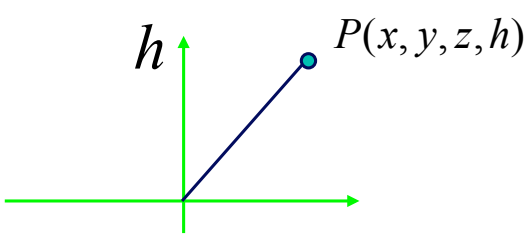


Homogeneous Coordinates

homogeneous **cartesian**

$$(x, y, z, h) \longrightarrow$$

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



Perspective Projection

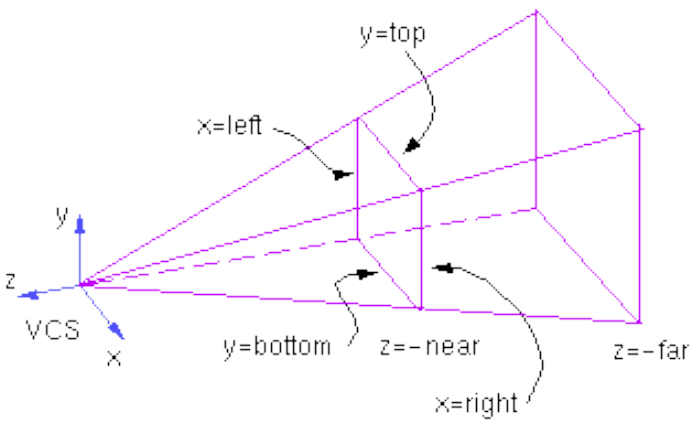
A simple version of M_{proj}

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

$$/\, h \searrow \quad \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

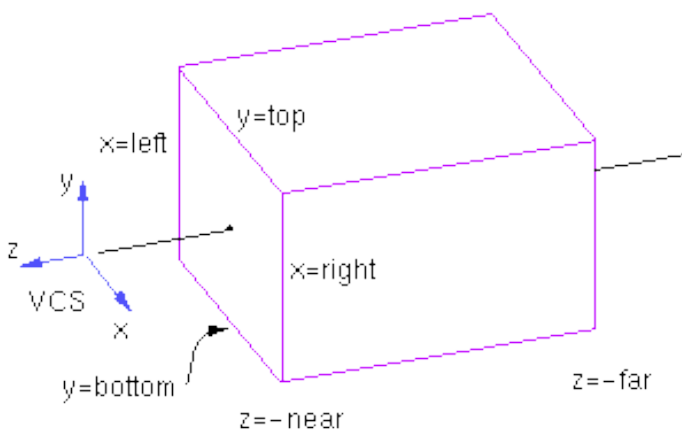
View Volumes: more about M_{proj}

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

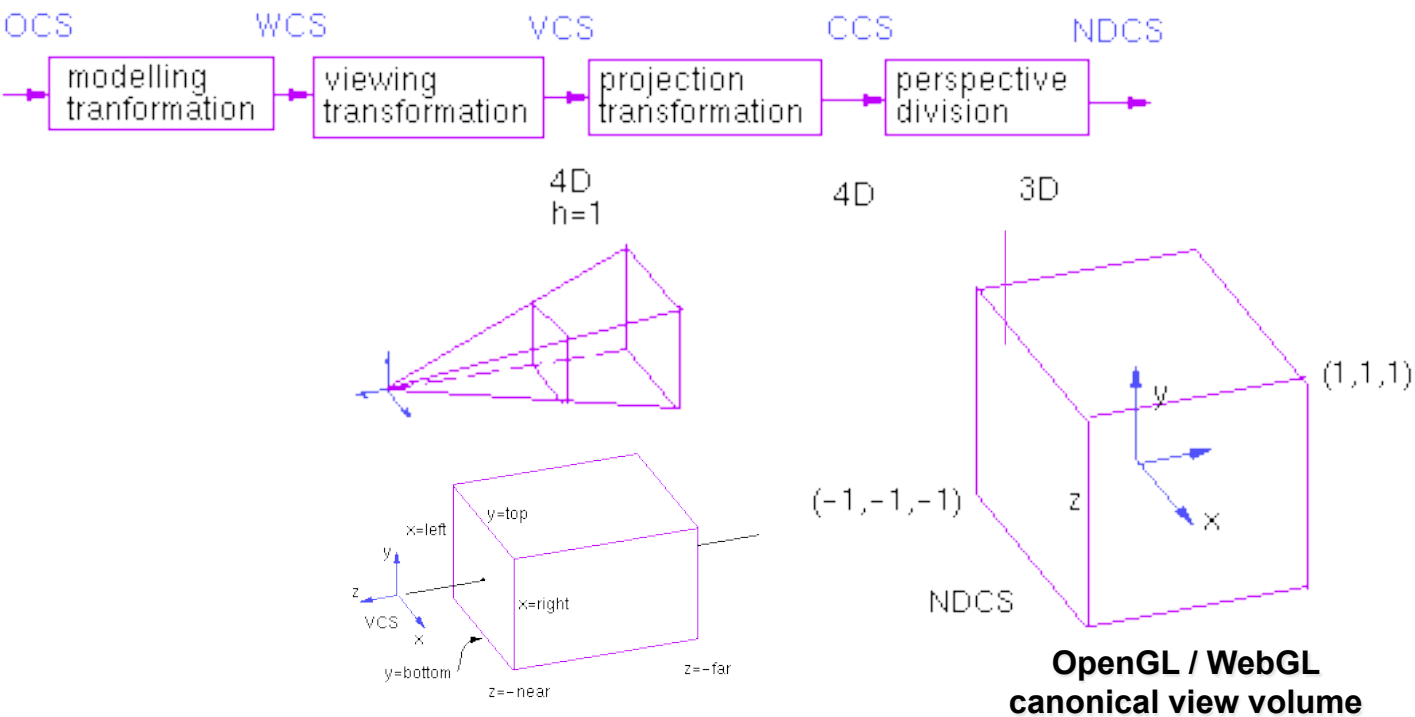


perspective view volume

orthographic view volume

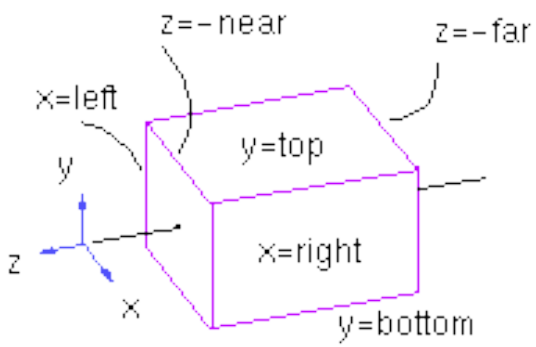


View Volumes

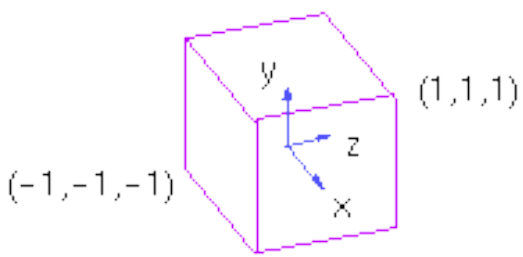


OpenGL / WebGL canonical view volume

Orthographic View Volume



VCS



NDCS

Orthographic View Volume

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

three.js

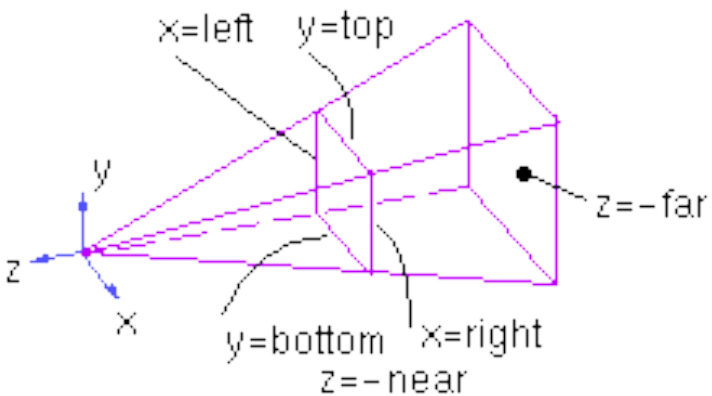
```
var cam = new THREE.OrthographicCamera(left,right,top,bot,near,far)
```

Orthographic View Volume

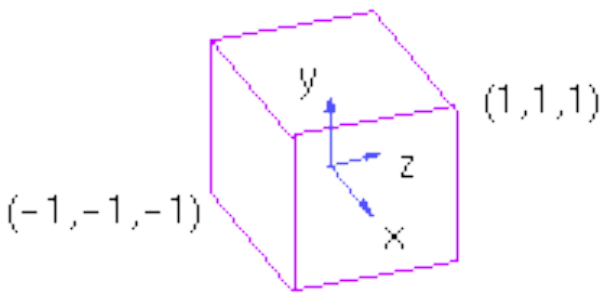
Derivation

solving for a and b gives:

Perspective View Volume



VCS



NDCS

Perspective View Volume

Derivation

earlier:

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

with additional ability to scale, etc.:

$$\begin{bmatrix} x' \\ y' \\ z' \\ h' \end{bmatrix} = \begin{bmatrix} E & & A & \\ & F & B & \\ & & C & D \\ & & -1 & \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective View Volume

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

$$\begin{bmatrix} \frac{2n}{r-l} & & \frac{r+l}{r-l} & \\ & \frac{2n}{t-b} & \frac{t+b}{t-b} & \\ & & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ & & -1 & \end{bmatrix} \qquad \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & -5/3 & -8/3 \\ & & -1 & \end{bmatrix}$$

three.js

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

Perspective View Volume

Derivation

top plane:

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

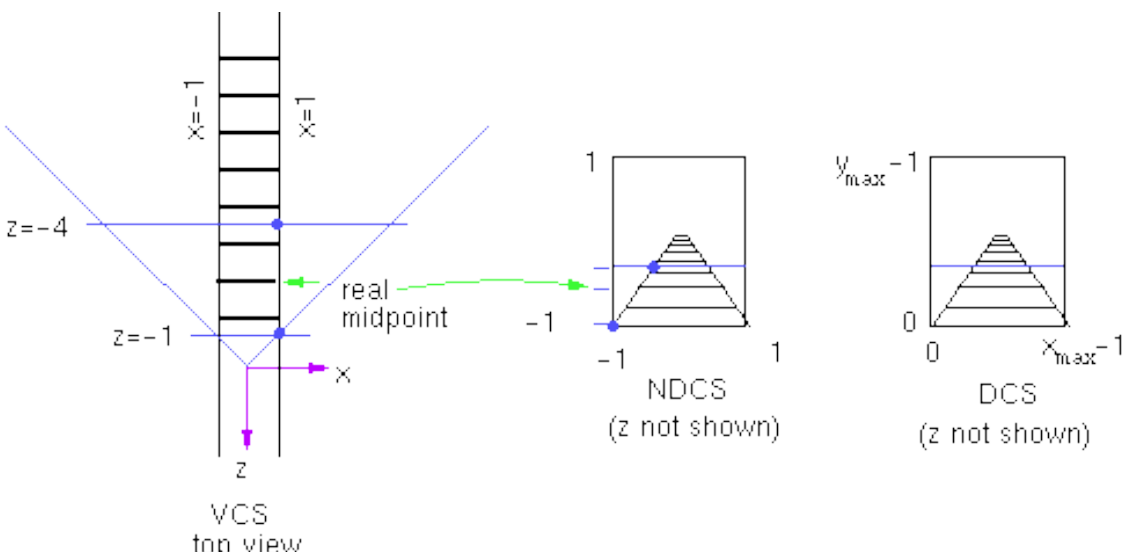
similar process for solving for the other unknowns,
using the left/right and near/far planes

Perspective Projection -- Example

Example

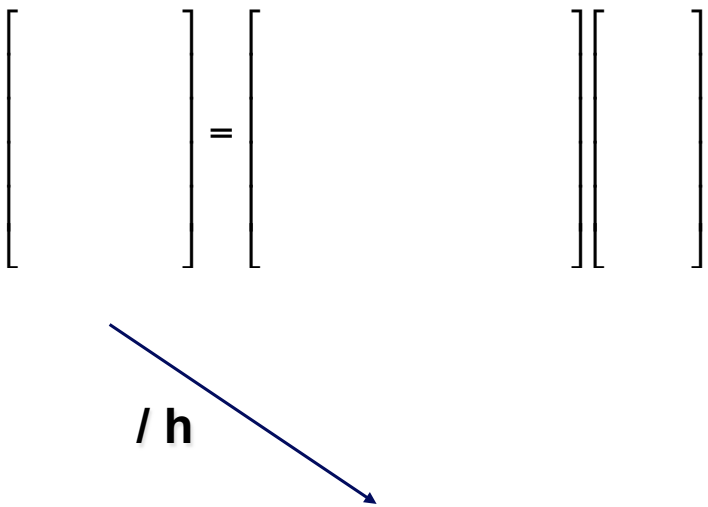
tracks in VCS:
left x=-1, y=-1
right x=1, y=-1

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

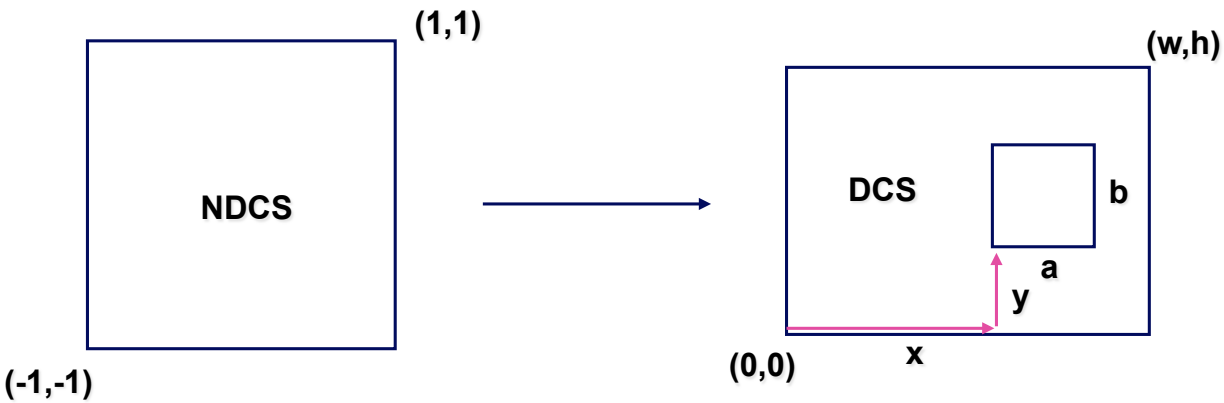


Perspective Projection -- Example

Example



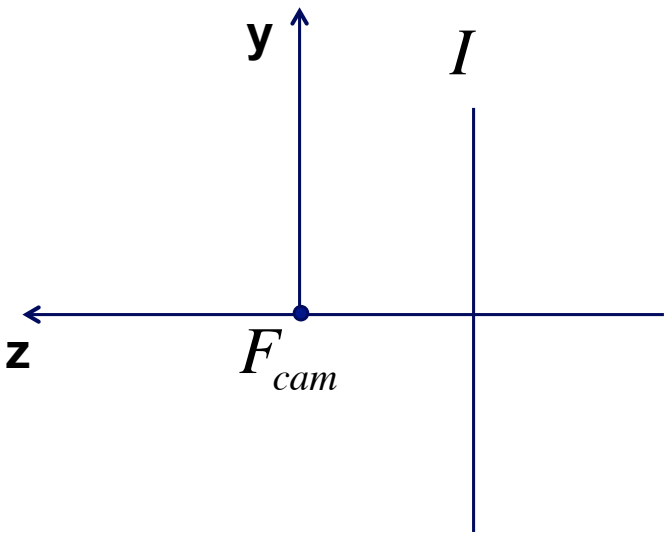
Viewport Transformation




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

Name _____

Perspective Projection -- example

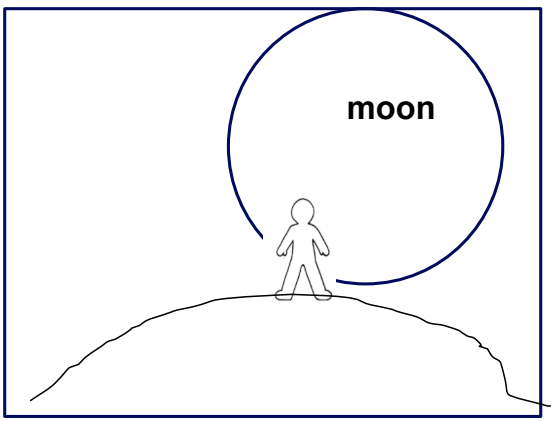


$P_A(0,5,-6)$ $P_B(0,5,-12)$

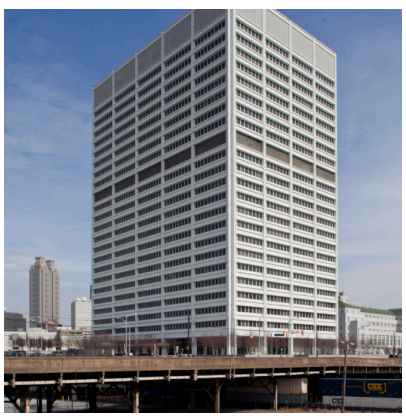
Compute the projected coordinates of the given points for a perspective projection. The image plane is located at z = -2.

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

Impossible Photography?



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



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?