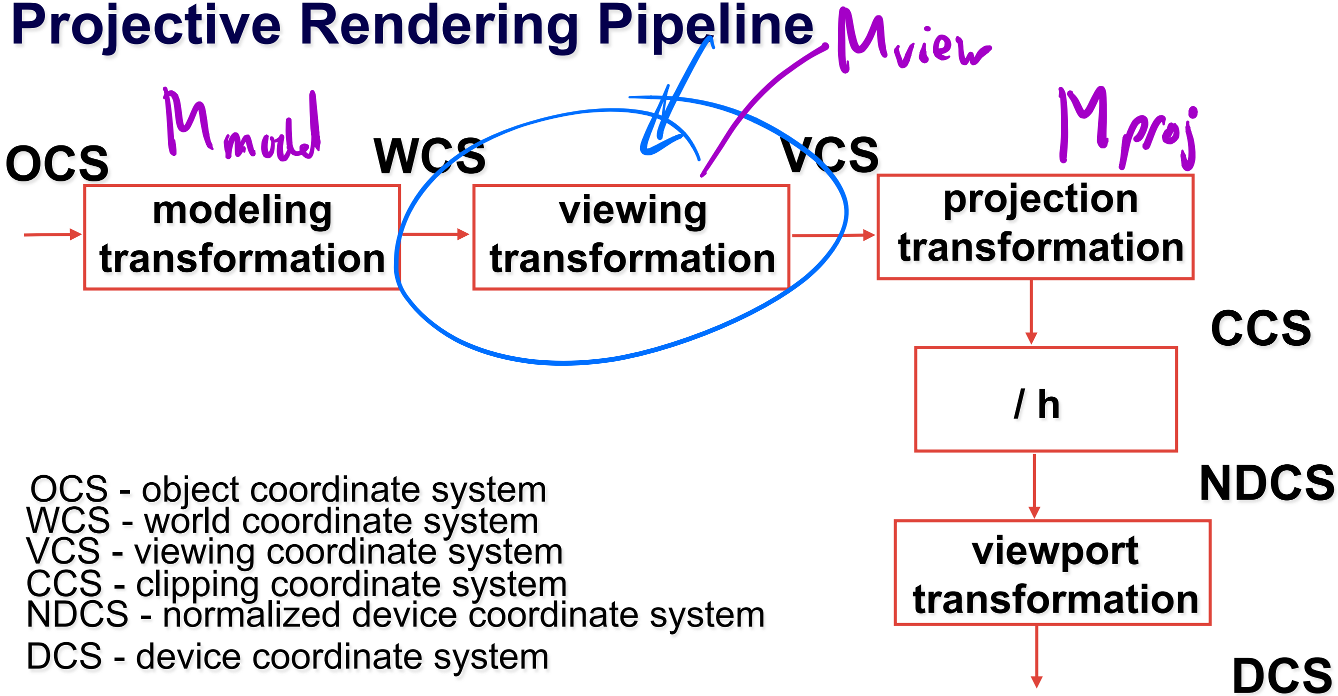
Viewing and Projection Transformations

Projective Rendering Pipeline

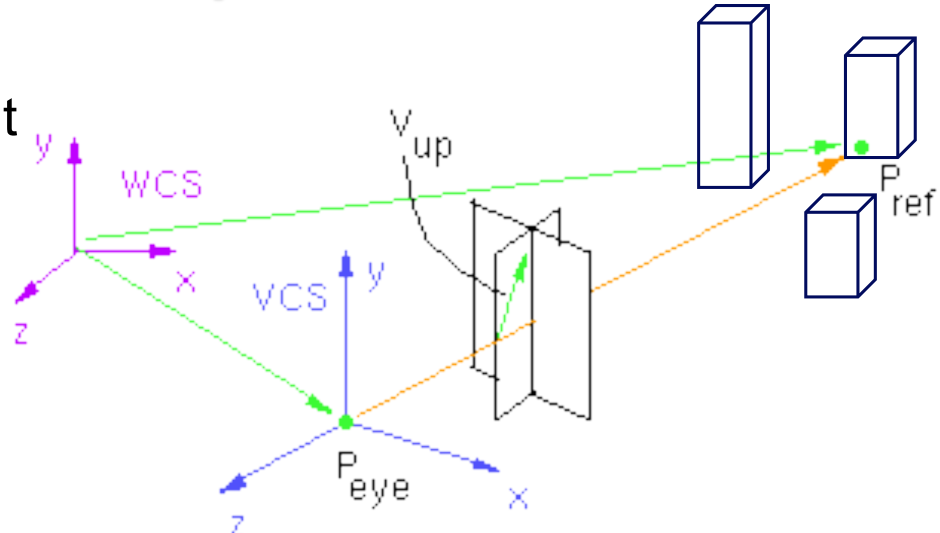


OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

Viewing Transformation

Defining the camera position and orientation

- eye point
- reference point
- up vector



three.js:

```
camera.position.set(30,0,0);
camera.up = new THREE.Vector3(0,0,1);
camera.lookAt(0,0,0);
// also:   object.matrix.lookAt(eye,center,up)
```

P_{eye} V_{up} P_{ref}

Computing i,j,k

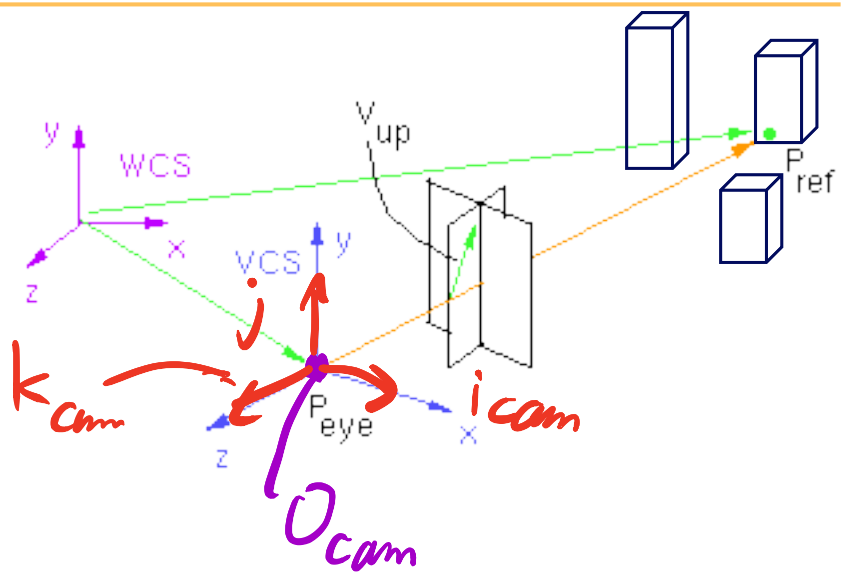
$$O_{cam} = P_{eye}$$

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

$$\vec{I}_{cam} = \vec{V}_{up} \times \vec{k}$$

$$\vec{i}_{cam} = \vec{I} / \| \vec{I} \|$$

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



Viewing Transformation M_{view}

$\vec{j}_{cam}$ O_{cam}

$\vec{i}_{cam}$ $\vec{E}_{cam}$

O_{cam}

$$M_{cam} = \text{Translate}(E_x, E_y, E_z)\text{Rotate}(\ldots)$$

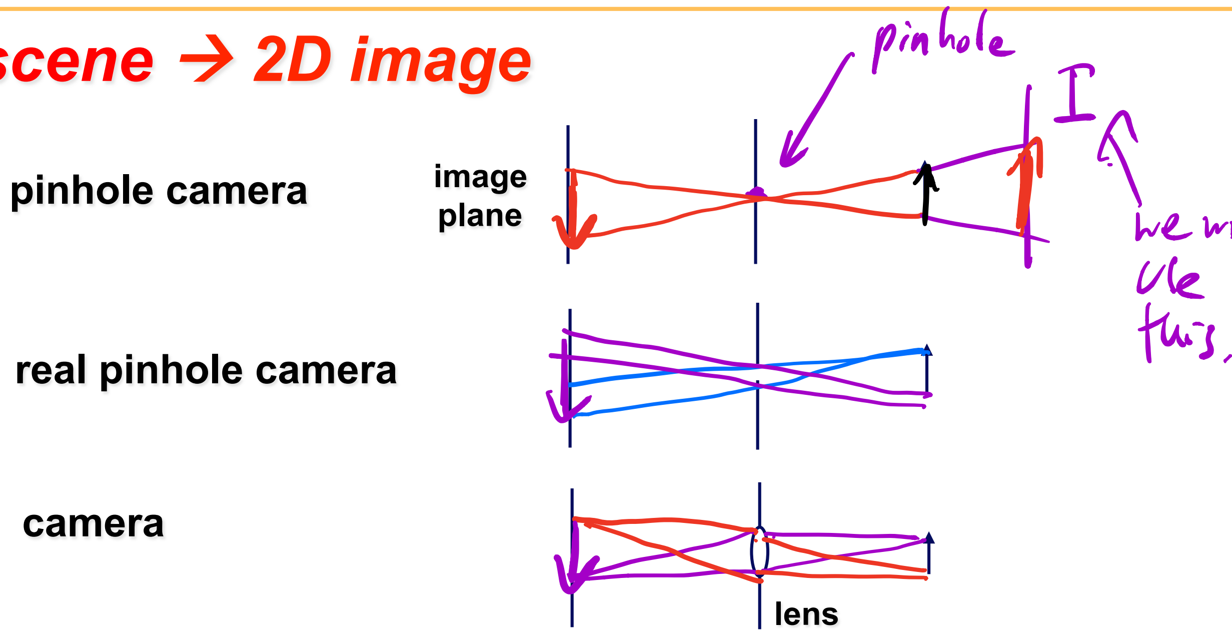
$$P_{wcs} = M_{cam} P_{cam} \qquad = \begin{bmatrix} 1 & 0 & 0 & E_x \\ 0 & 1 & 0 & E_y \\ 0 & 0 & 1 & E_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_x & j_x & k_x & 0 \\ i_y & j_y & k_y & 0 \\ i_z & j_z & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \vec{i}_{cam} & \vec{j}_{cam} & \vec{E}_{cam} & O_{cam} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{view} = M_{cam}^{-1} = \text{Rotate}(\ldots)^{-1}\text{Translate}(E_x, E_y, E_z)^{-1}$$

$$P_{cam} = M_{view} P_{wcs} \qquad = \begin{bmatrix} i_x & i_y & i_z & 0 \\ j_x & j_y & j_z & 0 \\ k_x & k_y & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -E_x \\ 0 & 1 & 0 & -E_y \\ 0 & 0 & 1 & -E_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection Transformation M_{proj}

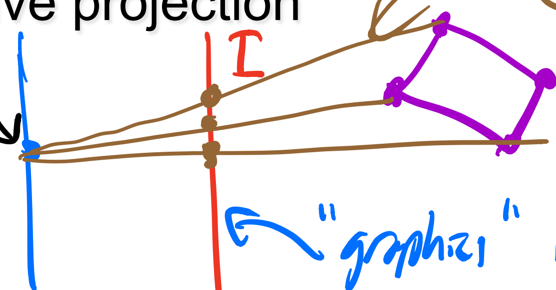
3D scene → 2D image



Projection

- definition $\mathbb{R}^m \to \mathbb{R}^n \quad n < m \quad 2 < 3$
- perspective projection

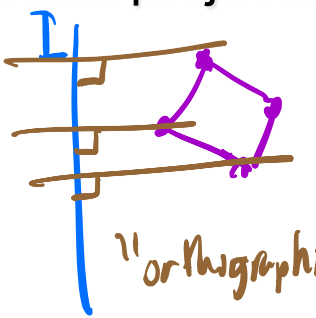
pinhole "center of projection"



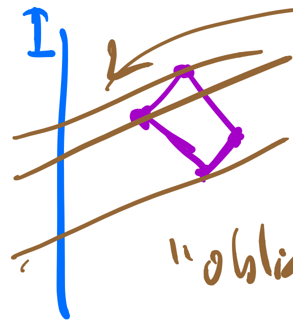
"projector"

"graphics" image plane.

- parallel projection



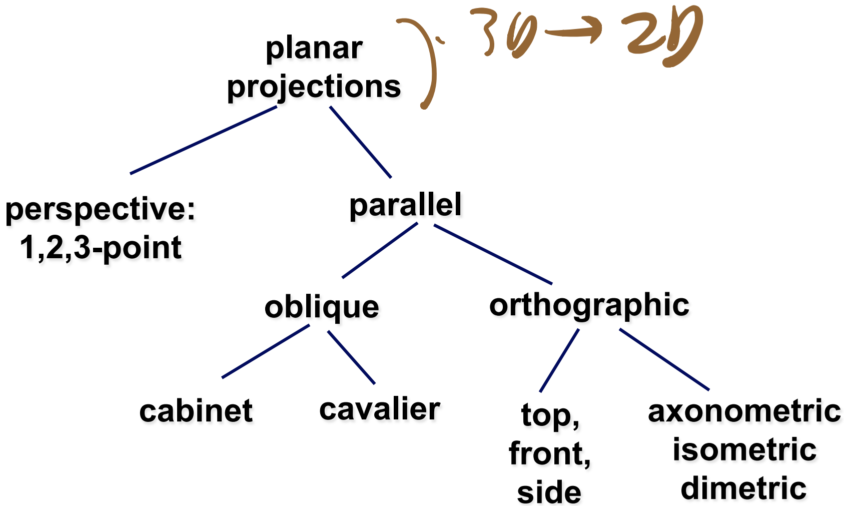
"orthographic"

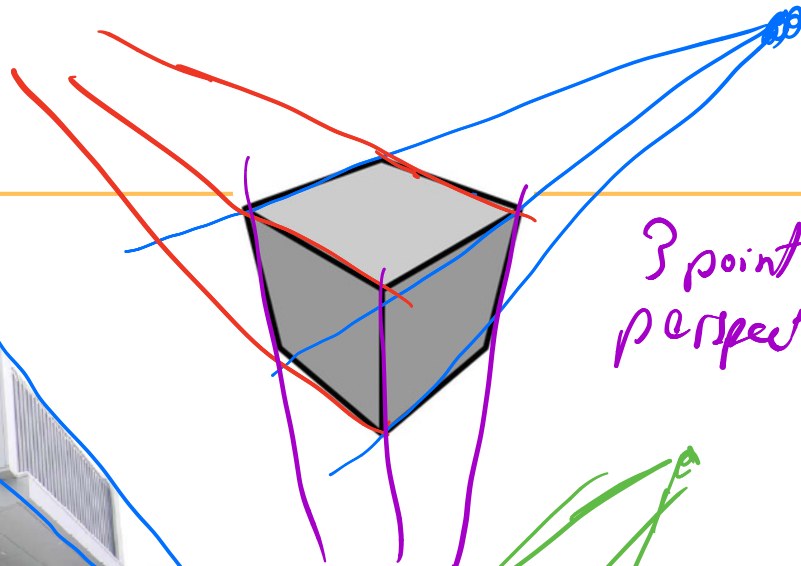


"oblique"

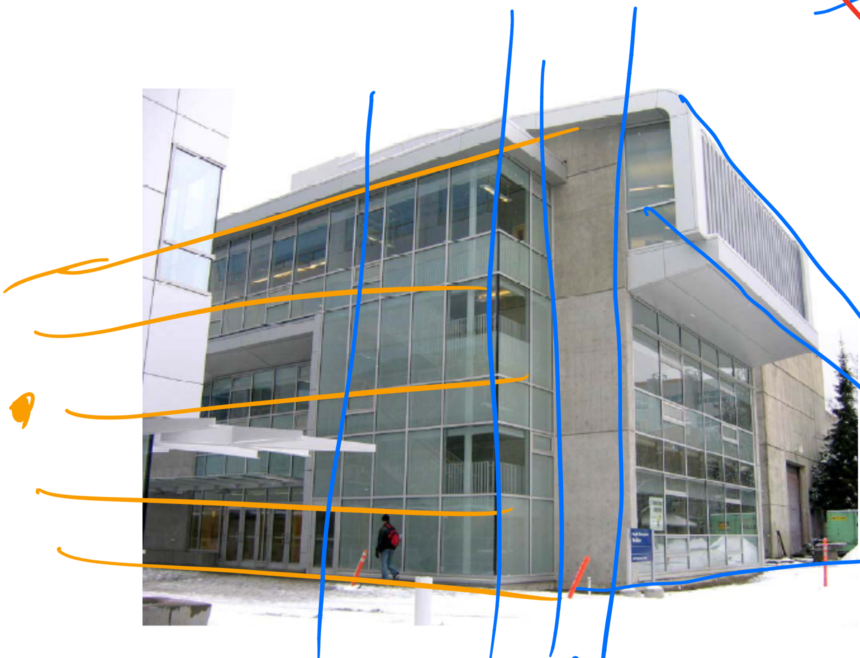
Projections

Taxonomy



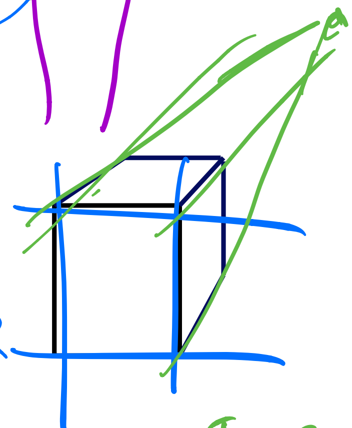


3 point perspective.



parallel

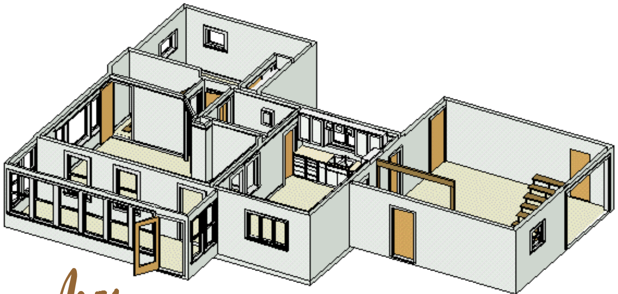
2 point perspective.



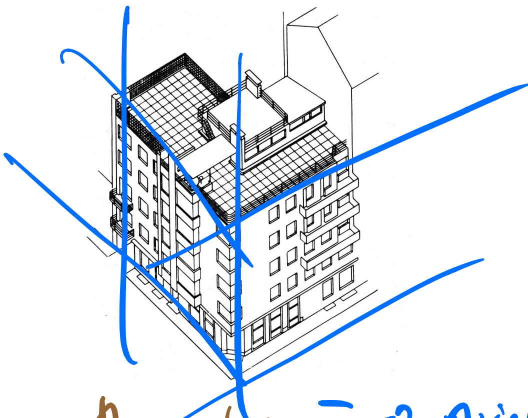
1-point perspective

rock

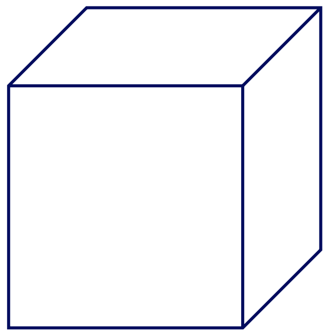
meaningless to talk about 1,2,3 pt. perspective.



orthographic

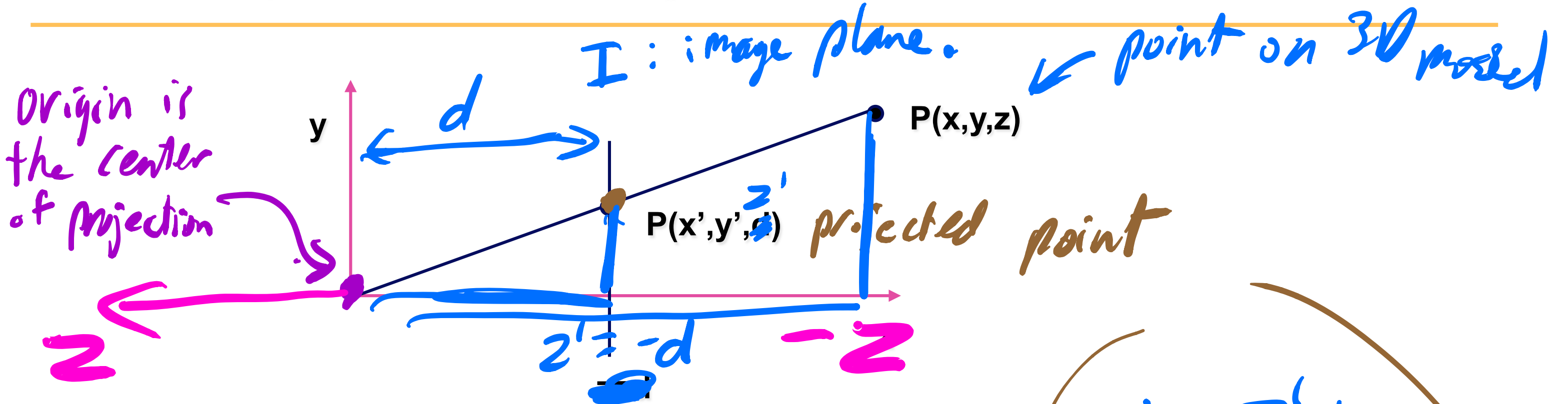


orthographic = 0 point perspective.



oblique.

Perspective Projection



similar triangles: $\frac{y'}{z'} = \frac{y}{z}$

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

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

similarly for x: $\boxed{x' = (-d)\frac{x}{z}}$

$\boxed{z' = -d}$

Homogeneous Coordinates

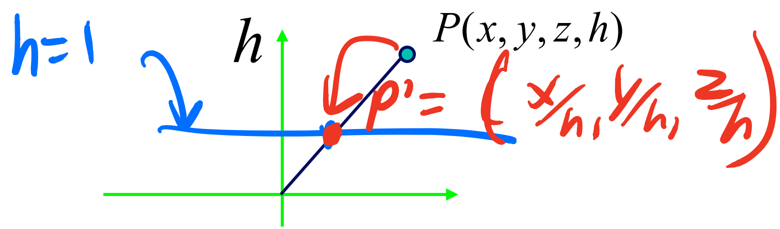
4D **homogeneous** 3D **cartesian**

Note: sometimes w is used $h \equiv w$

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

$(5, 5, 5, 10)$ $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}$

- redundant representation
- h=0: point at infinity (direction) $\Rightarrow$ Use this to model vectors.
- geometric interpretation



Perspective Projection

A simple version of M_{proj}

This will implement the previous projection

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

$h = -z/d$

$$\xrightarrow{/h} \begin{bmatrix} -d \cdot \frac{x}{z} \\ -d \cdot \frac{y}{z} \\ \cancel{-d \cdot \frac{z}{z}} \end{bmatrix} \quad p'$$

Projective Rendering Pipeline

M_{view} M_{proj}

LookAt() alter h

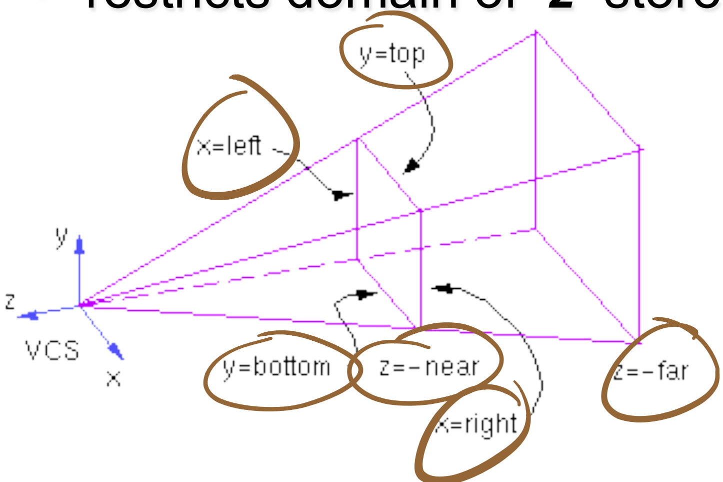
OCS → **modeling transformation** → WCS → **viewing transformation** → VCS → **projection transformation** → CCS → **/ h** → NDCS → **viewport transformation** → DCS

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

M_{view}: position & orientation of camera
M_{proj}: ortho or perspective projection, also specifies field of view.

View Volumes: more about M_{proj}

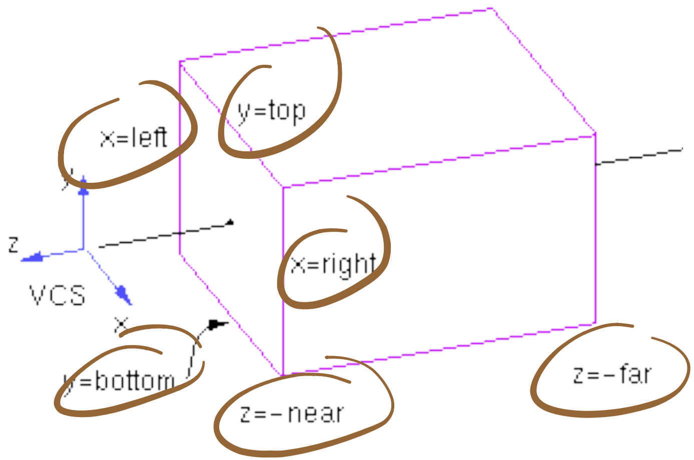
- specifies field-of-view, used for clipping
- restricts domain of z stored for visibility test



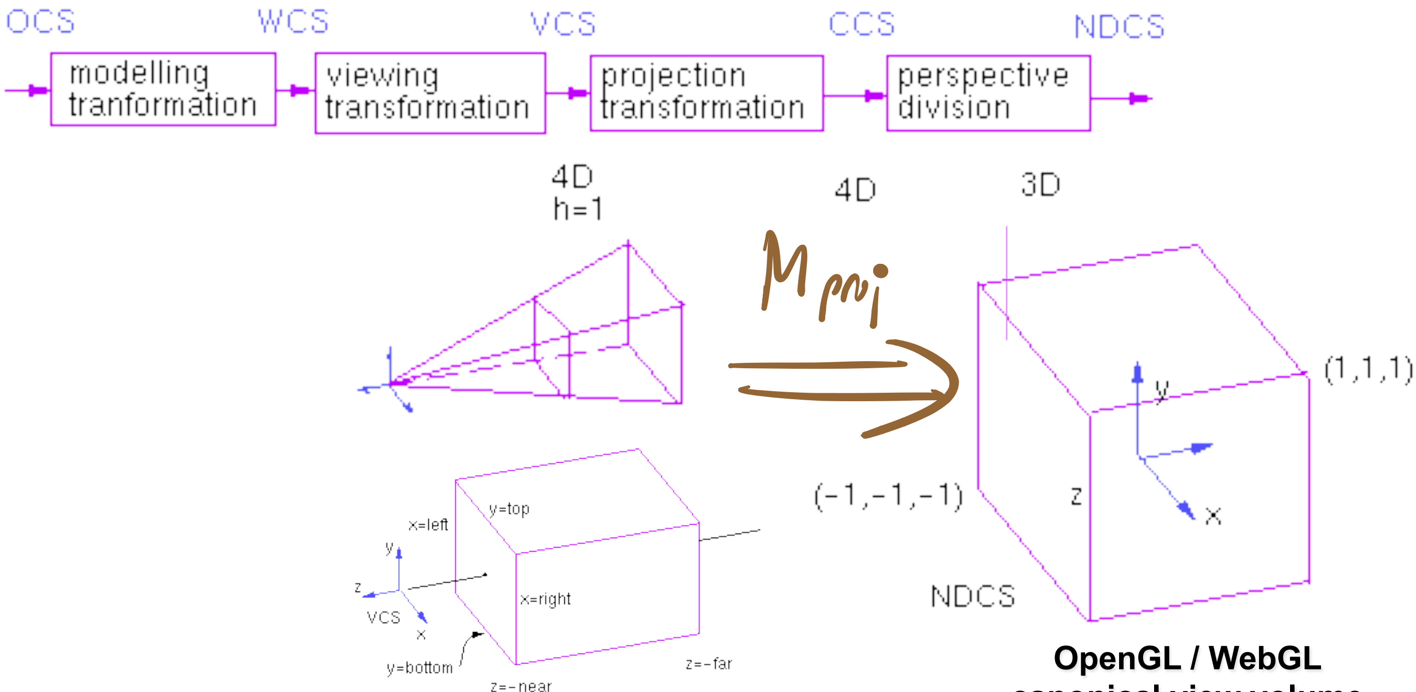
perspective view volume

"view frustum"
(pyramid without the tip)
→ 6 numbers to define the 6 planes

orthographic view volume



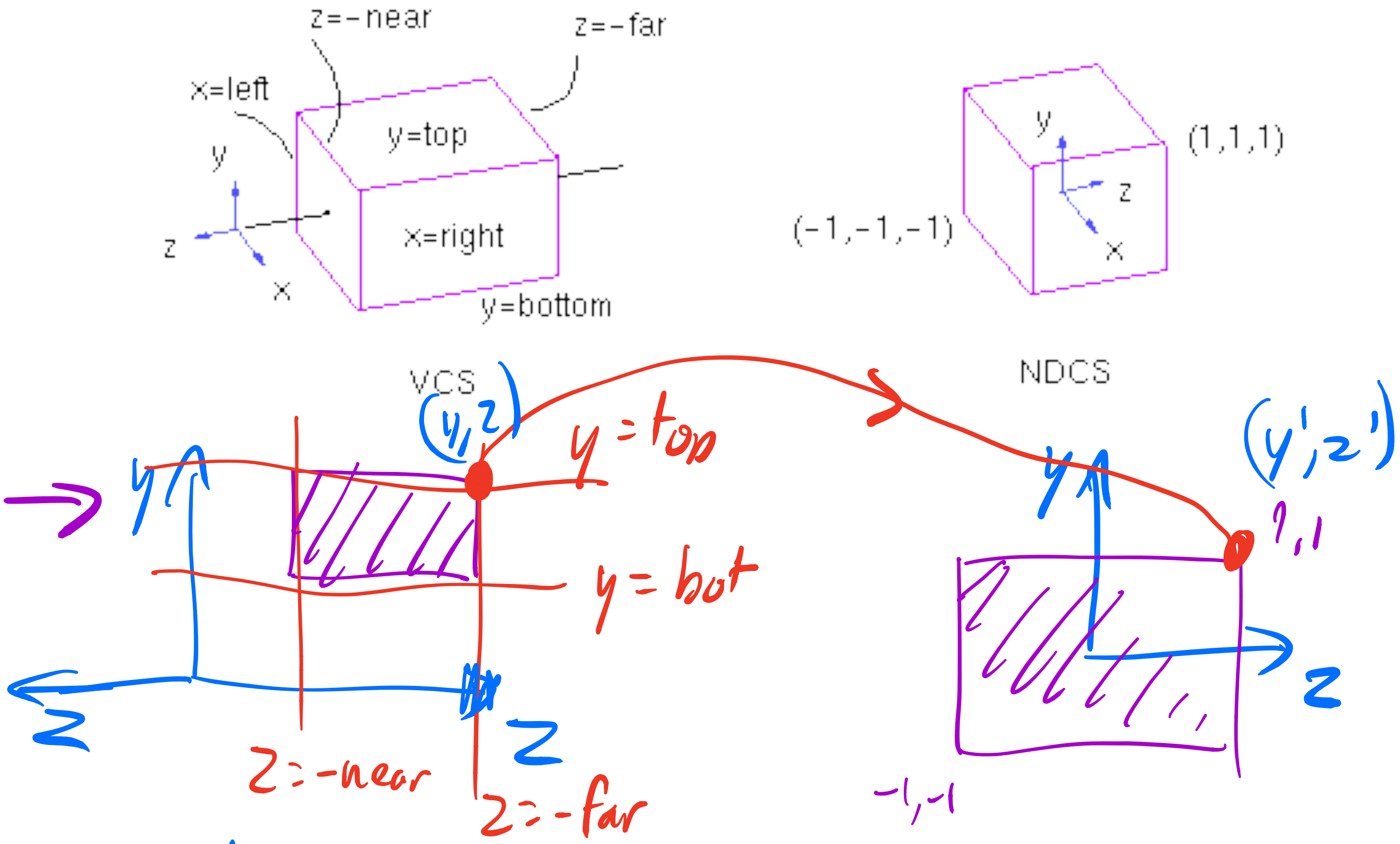
View Volumes



OpenGL / WebGL canonical view volume

Note: NDCS is a left-handed CS

Orthographic View Volume



$$\left[\begin{array}{l} y' = ay + b \\ 1 = a(top) + b \\ -1 = a(bot) + b \end{array}\right] \rightarrow \begin{array}{l} a = \dfrac{2}{(top-bot)} \\ b = -\dfrac{(top+bot)}{(top-bot)} \end{array}$$

Orthographic View Volume

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

a b

three.js

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

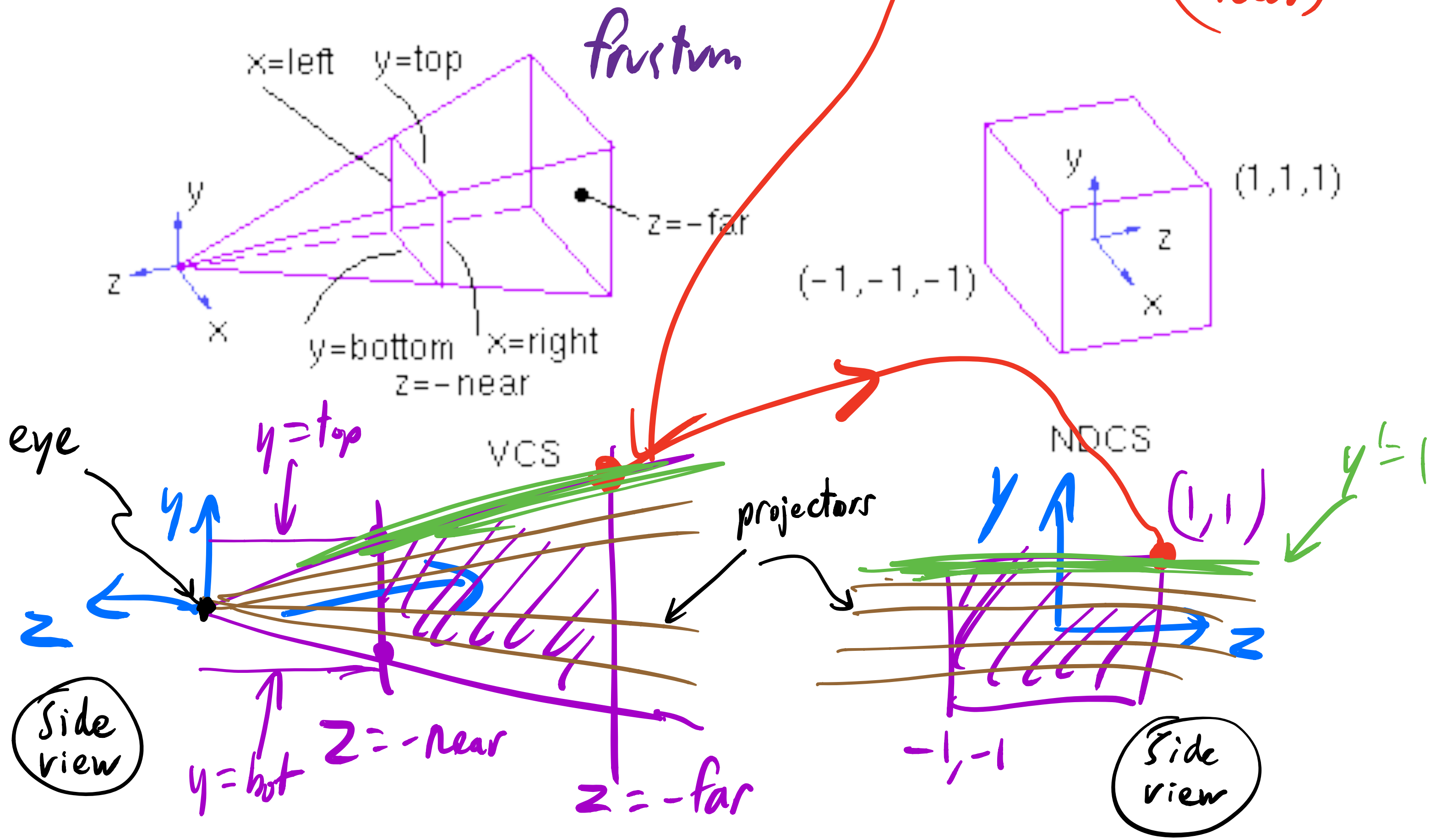
Orthographic View Volume

Derivation (see earlier slide)

solving for a and b gives:

Perspective View Volume

top plane: $y = \left(\frac{top}{-near}\right) z$



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.:

$$\begin{array}{c} \\ \\ \\ \end{array}\begin{bmatrix} -\frac{Ex}{z} - A \\ -\frac{Fy}{z} - B \\ -C - \frac{D}{z} \end{bmatrix}\begin{array}{c} x'' \\ y'' \\ z'' \end{array} \Leftarrow \begin{bmatrix} Ex + Az \\ Fy + Bz \\ Cz + D \\ -z \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ h' \end{bmatrix} = \begin{bmatrix} E & & A & \\ & F & B & \\ & & C & D \\ & & -1 & \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective View Volume

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

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

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

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: VCS NDCS

$$y = \left(\frac{top}{-near}\right) z \qquad y'' = 1$$

top plane

$$y'' = -\frac{Fy}{z} - B$$

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

bottom plane:

$$-1 = -F \cdot \frac{bot}{-near}\frac{\cancel{z}}{\cancel{z}} - B$$

$\Rightarrow$

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

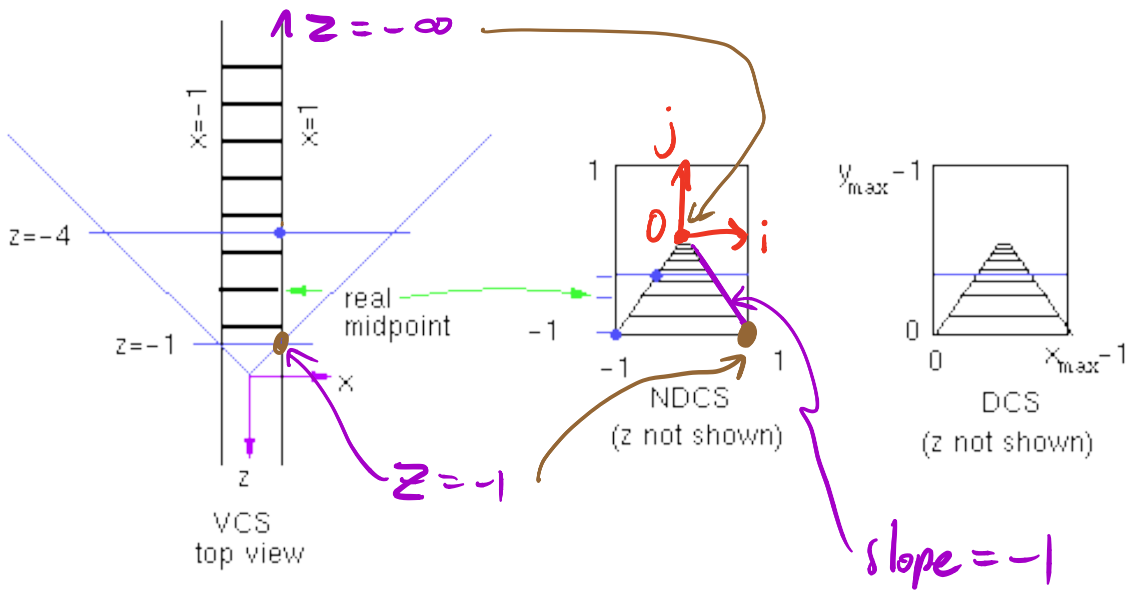
similar process for solving for the other unknowns, using the left/right and near/far planes

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

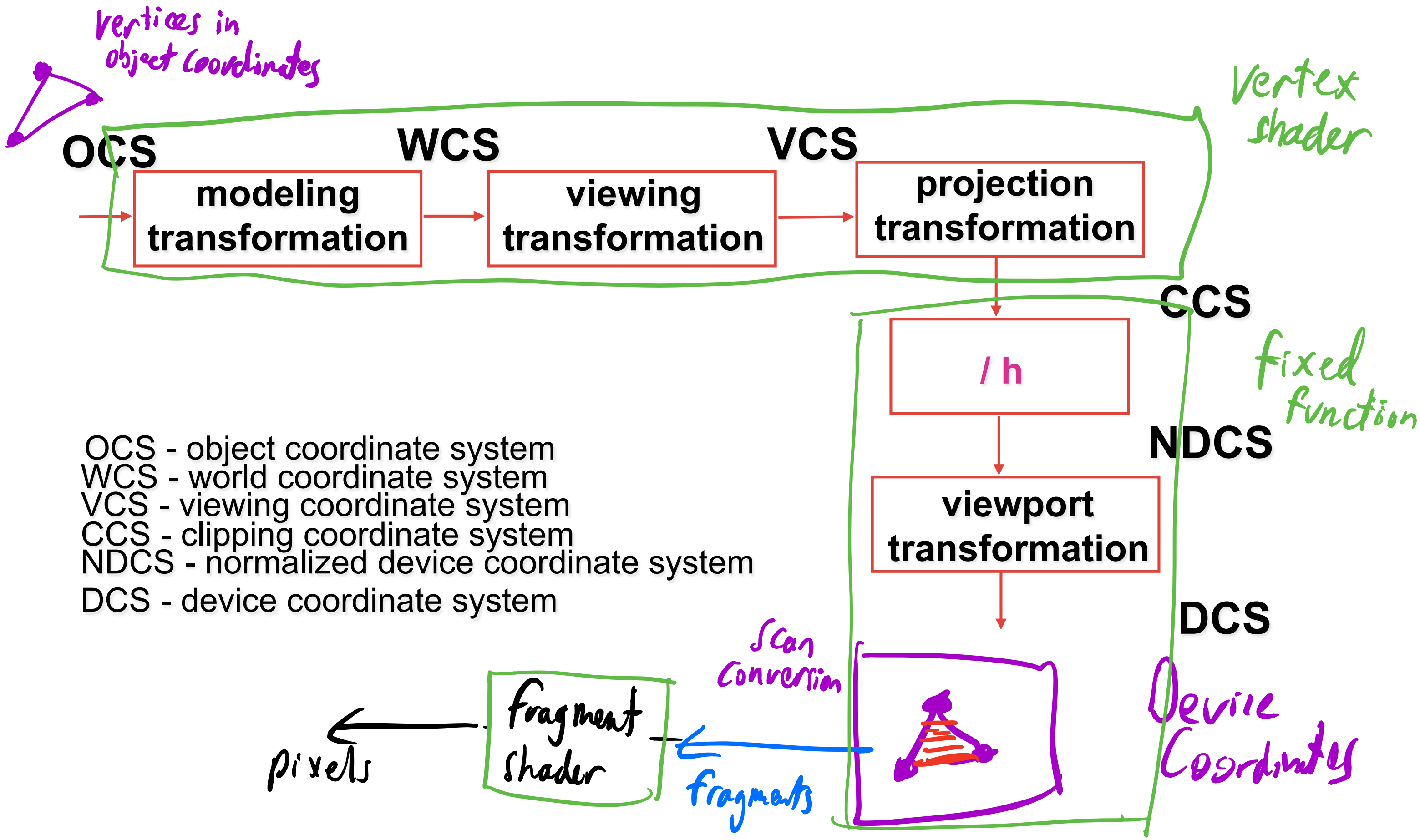
$$\begin{bmatrix} 1 \\ -1 \\ -\frac{5}{3}z - \frac{8}{3} \\ -z \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & -\frac{5}{3} & -\frac{8}{3} \\ & & -1 & \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ z \\ 1 \end{bmatrix} \quad \leftarrow \text{right railway track}$$

$$\xrightarrow{\;/h\;} \quad \left.\begin{bmatrix} -1/z \\ 1/z \\ \frac{5}{3} + \frac{8}{3 \cdot z} \end{bmatrix}\right\} \quad \text{slope} = \frac{y''}{x''} = -1$$

for $z = -\infty$

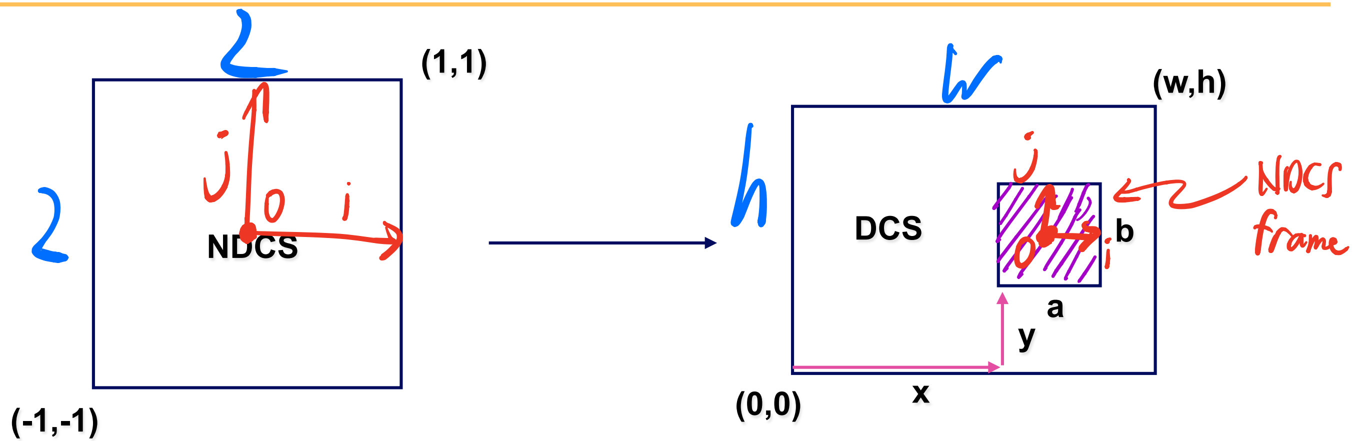
$(x'', y'') = 0$

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

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
```

$$P_{DCS} = \mathrm{Trans}\left(\frac{w}{2}, \frac{h}{2}, 0\right)\ \mathrm{Scale}\left(\frac{w}{2}, \frac{h}{2}, 1\right)$$

This is for $x=0$, $y=0$, $a=w$, $b=h$ (the typical case)