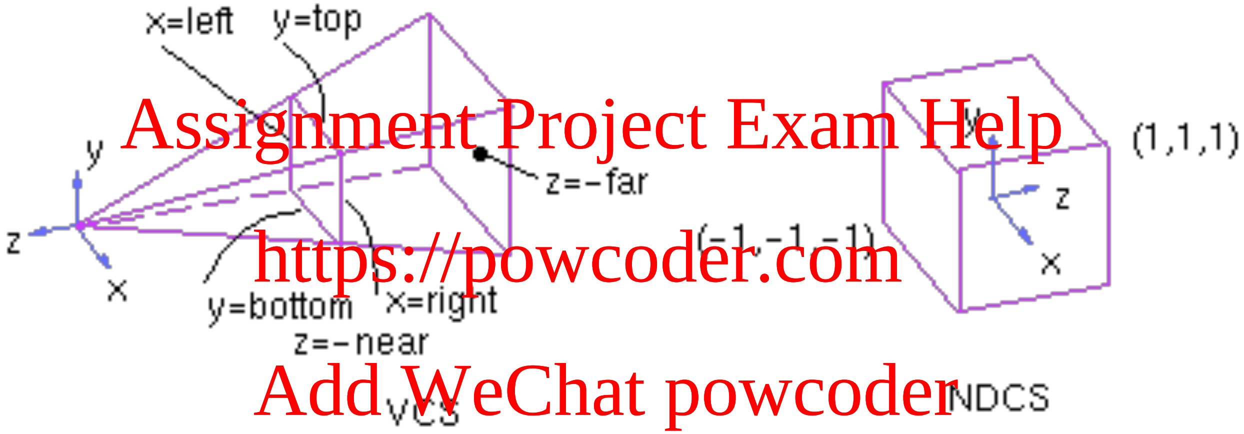
Derivation of the perspective transformation

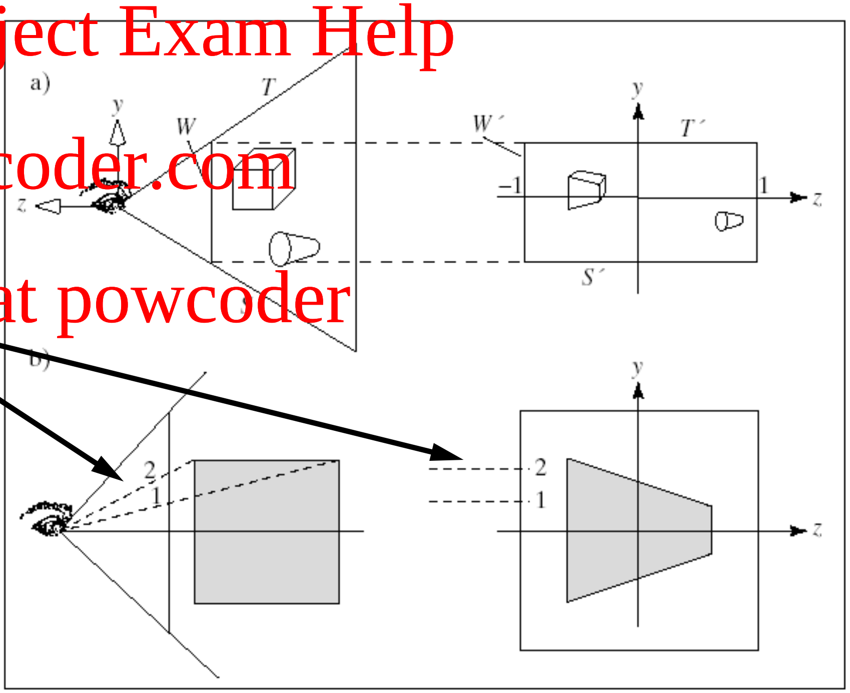


- It is basically a mapping of planes
- Normalized view volume is a left handed system
- However, there is an easier derivation

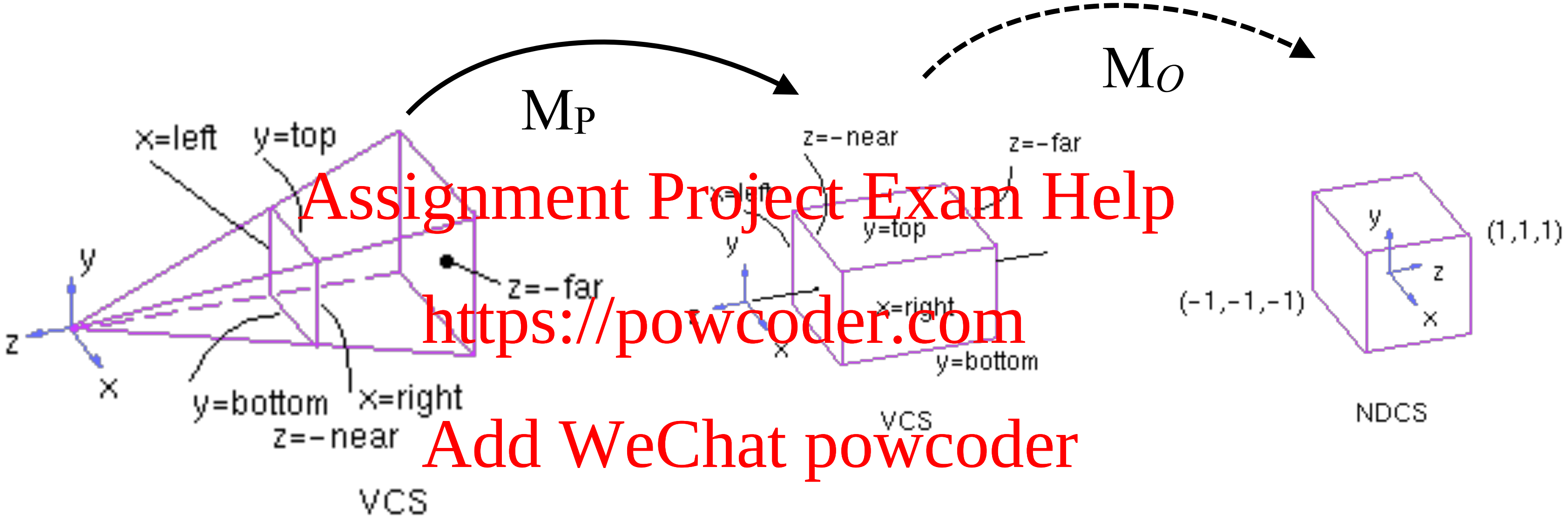
Interpretation of the Perspective transformation

Warps the view volume and the objects in it

- Eye becomes a point at infinity, and the projection rays become parallel lines (orthographic projection)
- We also want to keep z

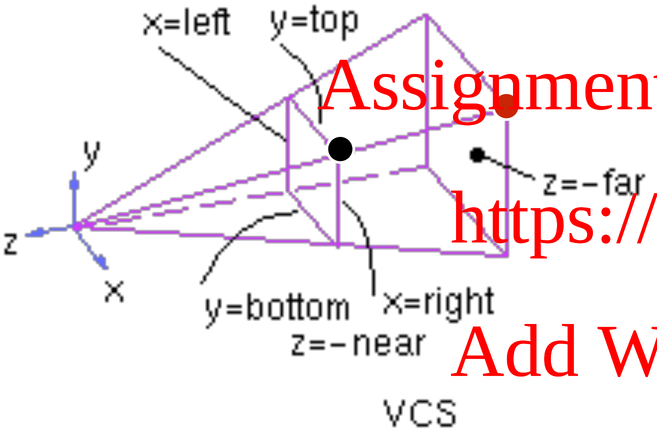


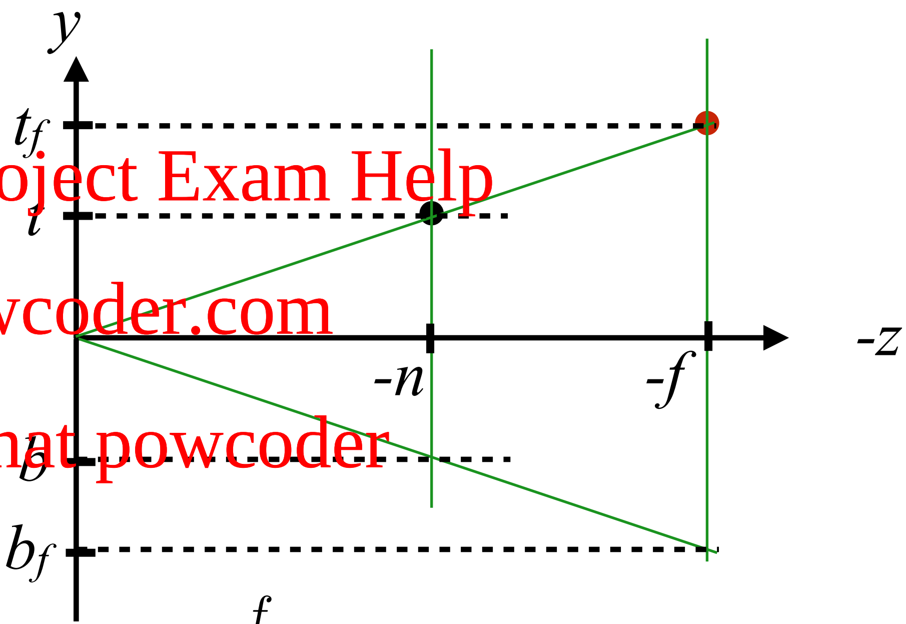
Two step derivation



- Start with the Canonical Perspective Projection matrix
- Adjust CPP to scale z so that after the application of CPP z = -near maps to z'=-near and z=-far maps to z'=-far)
- We now have the standard orthographic view volume
- Use the previously derived orthographic projection matrix M_O

First: What are the view volume points in viewing coordinates?



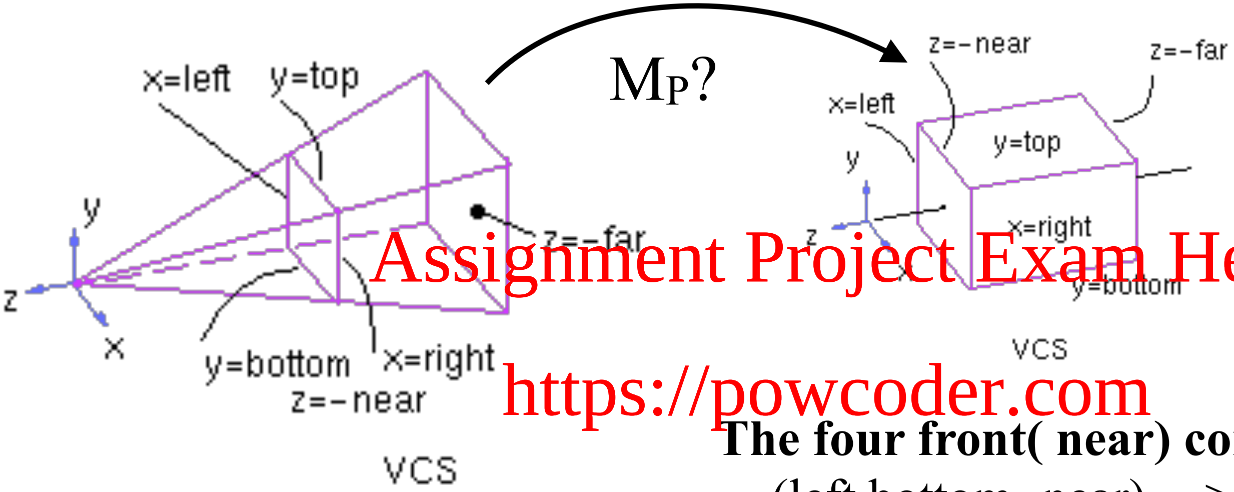


From similar triangles: $\frac{t}{-n} = \frac{t_f}{-f} \rightarrow t_f = t\frac{f}{n}$

Similarly $y = b$, and for $x = r$ and $x = l$

So for example: black point: $(r, t, -n) \rightarrow$ Red point: $(rf/n, tf/n, -f)$

First step: What does the CPP produce?



- $P'_x = n\, P_x / P_z$
- $P'_y = n\, P_y / P_z$
- $P'_z = -n$

The four front(near) corners map as follows:

(left,bottom,-near) --> (left,bottom,-near)
(left,top,-near) --> (left,top,-near)
(right,bottom,-near) --> (right,bottom,-near)
(right,top,-near) --> (right,top,-near)

The four back (far) corners map as follows:

(l*f/n,b*f/n, -f) --> (l, b,-n) instead of (l,b,-f)
(l*f/n,t*f/n,-f) --> (l, t, -n) instead of (l,t,-f)
(r*f/n,b*f/n, -f) --> (r,b,-n) instead of (r,b,-f)
(r*f/n,t*f/n,-f) --> (r,t,-n) instead of (r,t,-f)

So we just have to adjust the matrix for z

First step:Adjust the matrix to keep and scale Z

- We want to keep z and
- We want $P'_z = -n$ for $P_z = -n$ and $P'_z = -f$ for $P_z = -f$ **after** the perspective division
- Where do we do the changes?

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1}{n} & 0 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

First step: With some intuition...

Reminder: n,f are positive

$$\mathbf{M}_P \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z \frac{n+f}{n} + f \\ -\frac{P_z}{n} \end{bmatrix} \xrightarrow{\texttt{Homogenize with } h=-P_z/n} \begin{bmatrix} \frac{-P_x n}{P_z} \\ \frac{-P_y n}{P_z} \\ -(n+f) - \frac{fn}{P_z} \\ 1 \end{bmatrix}$$

Therefore

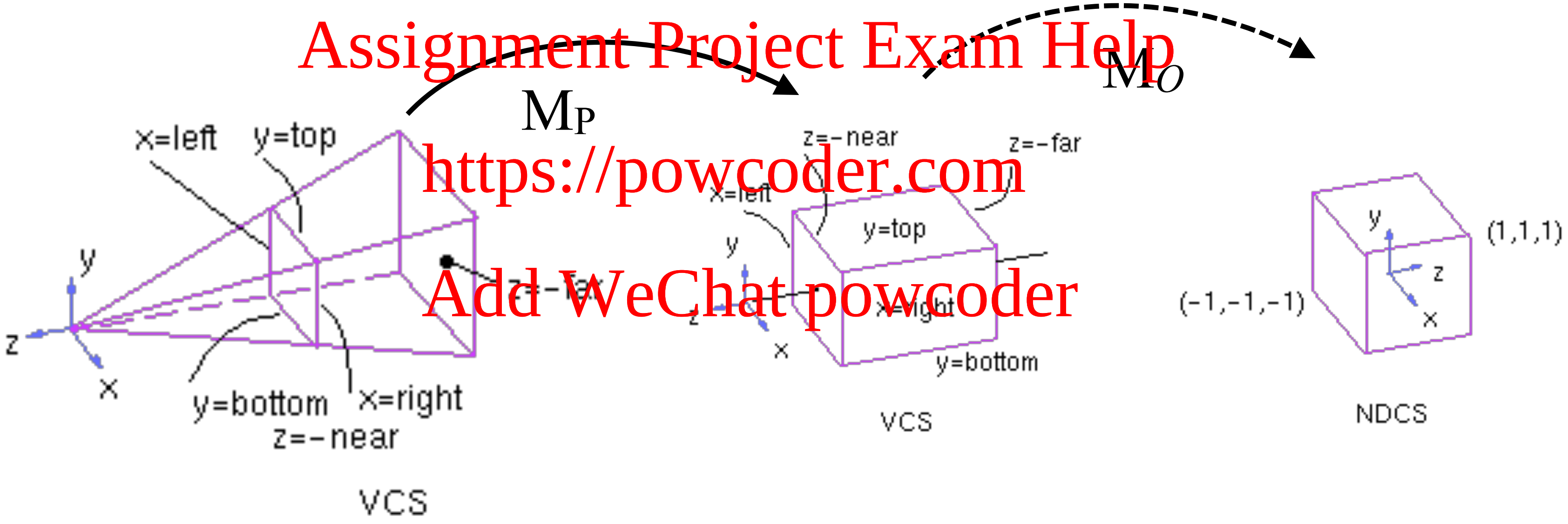
$$\mathbf{M}_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{n+f}{n} & f \\ 0 & 0 & \frac{-1}{n} & 0 \end{bmatrix} \quad \text{or} \quad \mathbf{M}_P = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & fn \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Sanity check

- (l*f/n,b*f/n,-f) --> (l,b,-f)
- (l,b,-n) --> (l,b,-n)

First step: M_P

- Creates the previous orthographic view volume



- Which can then be transformed to NDCS with M_O

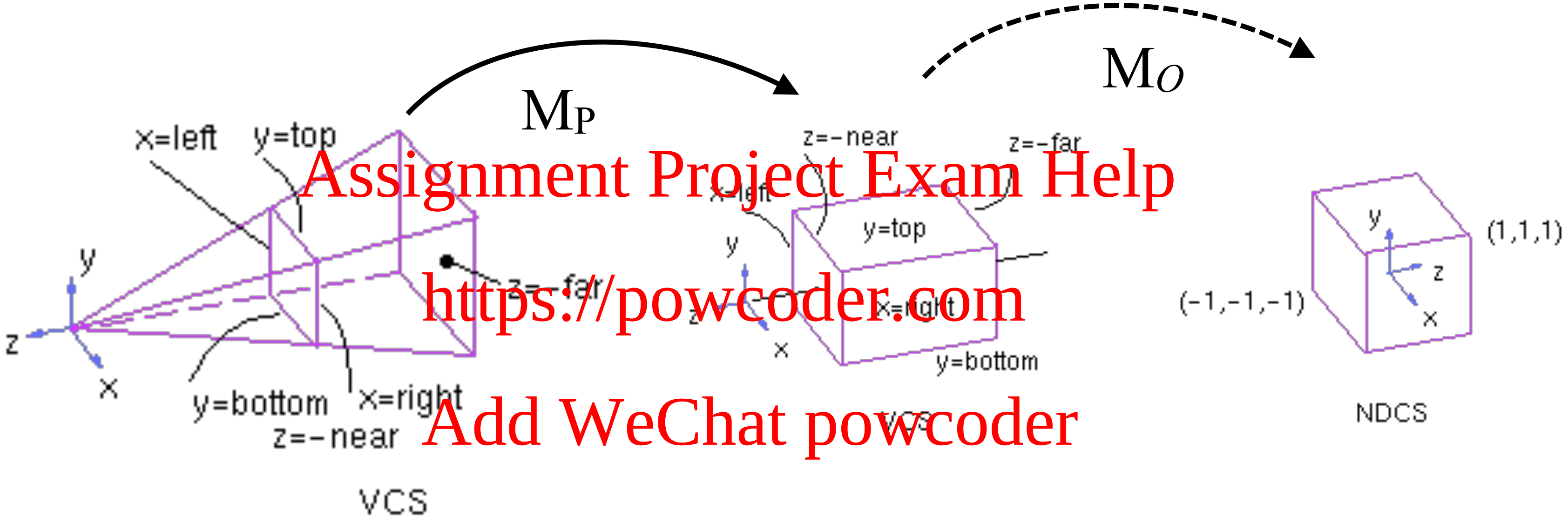
Second Step: Combine with Orthographic Projection Matrix

Now all we need to do is an orthographic transformation

- We can use matrix M_O that transforms an orthographic view volume to a normalized one (NDCS)

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

Second Step: Combine with Orthographic Projection Matrix



$$\mathrm{M}_{\texttt{prom}} = \mathrm{M}_O \mathrm{M}_P$$

OpenGL Perspective Matrix

Old form

- Still widely used

$$\mathrm{M}_{\mathtt{OpenGL}} = \begin{bmatrix} \frac{2|n|}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2|n|}{t-b} & \frac{b+t}{t-b} & 0 \\ 0 & 0 & \frac{|n|+|f|}{|n|-|f|} & \frac{2|f||n|}{|n|-|f|} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Projections in OpenGL (mimicking the old way)

New way

```
projMat = frustum(-left, right, bottom, top, near, far);
projMat = ortho(-left, right, bottom, top, near, far) ;
projMat = Perspective(fov, aspect, near far) ;
```

near plane at z = -near

far plane at z = -far

Matrix Order

Normally projection has to apply to all objects (i.e. the entire scene) thus it must pre-multiply the modelview matrix

- $M = M_{proj}M_{modelview}$ or
- $M = M_{proj}M_{view}M_{model}$

Important

Projection parameters are given in CAMERA Coordinate system (Viewing).

So if camera is at z = 50, is aligned with the world CS, and you give near = 10 where is the near plane with respect to the world?

Important

Projection parameters are given in CAMERA Coordinate system (Viewing).

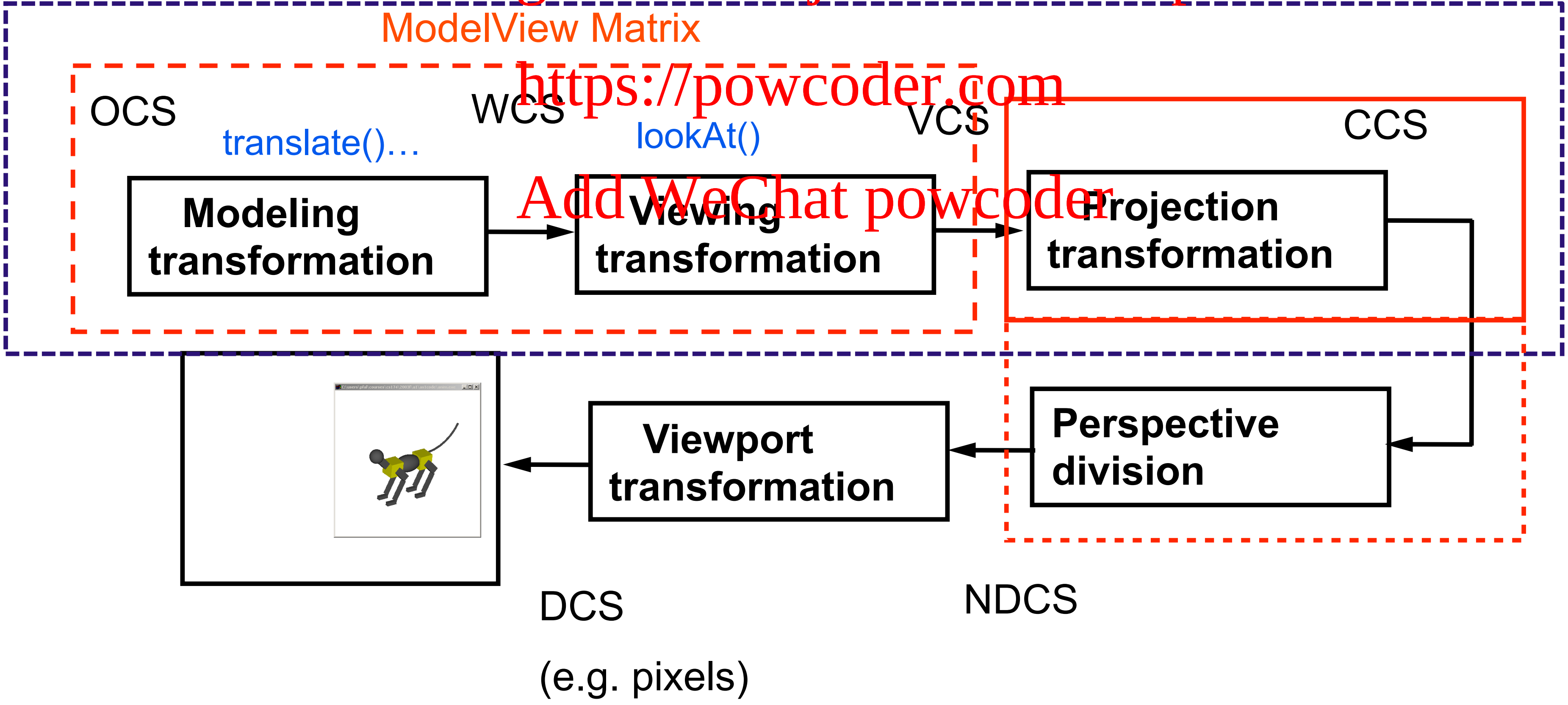
So if the camera is at z = 50, is aligned with the world CS, and you give |near| = 10 where is the near plane with respect to the world?

- Transformed by inverse(Mvcs)
- i.e. (0,0,40)

Perspective Division in Pipeline

The perspective division is done automatically

- Typically the vertex shaders writes CCS in gl_Position.



ModelView Matrix

OCS

translate()…

WCS

lookAt()

VCS

CCS

Modeling transformation

Viewing transformation

Projection transformation

Perspective division

Viewport transformation

DCS

(e.g. pixels)

NDCS

Perspective Division in Pipeline

However, we can do it ourselves if we want.

The vertex shader has total freedom on how to deal with projections