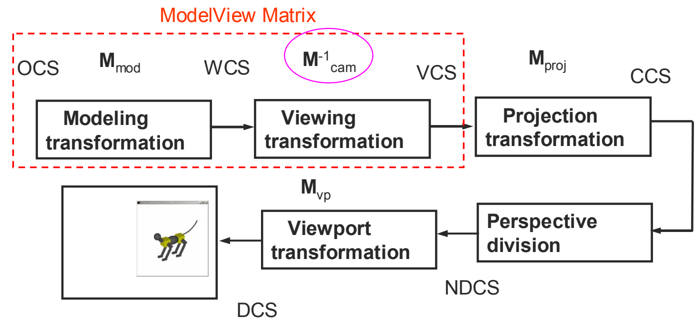
Projections (contd.)
&
Viewport Transformations

CS 174A

Transformations in the Pipeline



Defining $\mathbf{M}_{cam}$

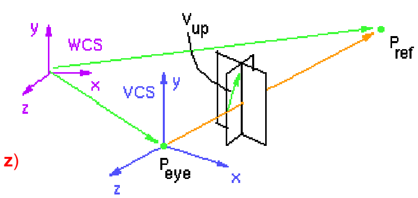
Given:

Eye point P_{eye}

Reference point P_{ref}

Up vector $\mathbf{v}_{up}$

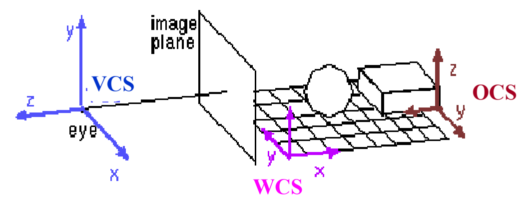
($\mathbf{v}_{up}$ is not necessarily orthogonal to **z**)



To build $\mathbf{M}_{cam}$ we need to define a camera coordinate system (O, **i**, **j**, **k**)

Camera Transformation

Represents objects with respect to camera coordinates

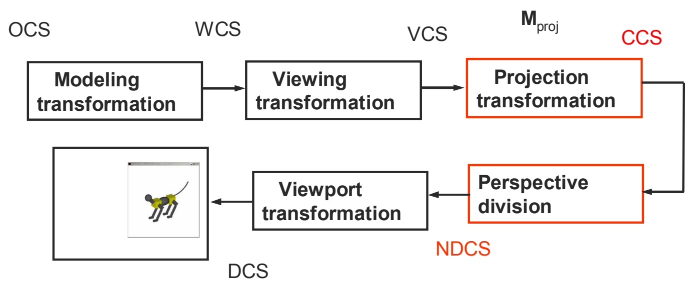


$$P_{\text{WCS}} = \text{M}_{\text{mod}} P_{\text{OCS}}$$

$$P_{\text{WCS}} = \text{M}_{\text{cam}} P_{\text{VCS}} \quad \Rightarrow \quad P_{\text{VCS}} = \text{M}_{\text{cam}}^{-1} P_{\text{WCS}}$$

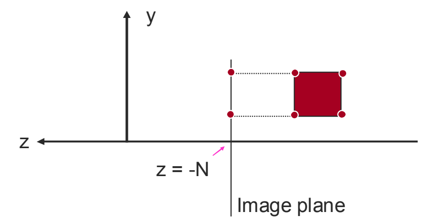
$$P_{\text{VCS}} = \underbrace{\text{M}_{\text{cam}}^{-1} \text{M}_{\text{mod}}}_{\text{Modelview Transformation}} P_{\text{OCS}}$$

Graphics Pipeline



A Basic Orthographic Projection

$P'_x = P_x$
$P'_y = P_y$
$P'_z = -N$

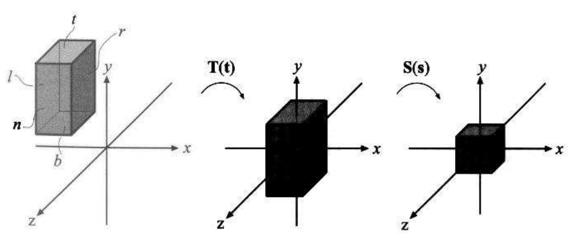


Matrix Form:

$$\begin{bmatrix} P'_x \\ P'_y \\ P'_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -N \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

Alternatively Transform to a Normalized Cuboid

- We have a cuboid that we want to map to the normalized or square cube from [-1, +1] in all axes
- We have parameters of cuboid (l,r ; t,b; n,f)

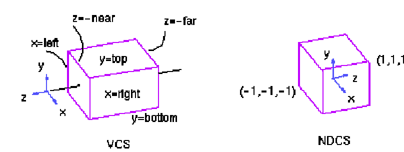


Alternatively Transform to Normalized Cuboid

Scaling and translation

Define:

top = *t* bottom = *b*
left = *l* right = *r*
-near = *n* -far = *f*



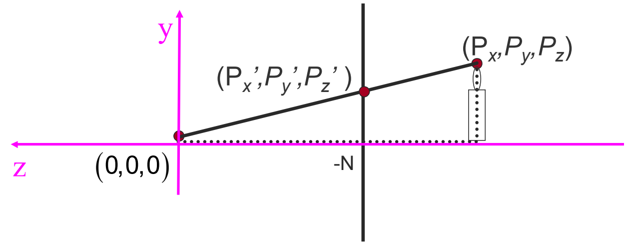
$$M_{O'} = \underbrace{\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{Scaling}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{Translation}}$$

Final Result

$$
\mathbf{M}_O = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad glm::ortho = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \boxed{\frac{-2}{f-n}} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

- Looking down –z, f and n are negative (n > f)
- OpenGL convention: positive n, f, negate internally

Perspective Projection



Looks like we've got some nice similar triangles here?

$$
P_z' = -N \qquad \frac{P_x}{P_z} = \frac{P_x'}{P_z'} \quad \Rightarrow \quad P_x' = \frac{NP_x}{-P_z}
$$

$$
\frac{P_y}{P_z} = \frac{P_y'}{P_z'} \quad \Rightarrow \quad P_y' = \frac{NP_y}{-P_z}
$$

$$
\boxed{\begin{bmatrix} P_x' \\ P_y' \\ P_z' \\ 1 \end{bmatrix} = \begin{bmatrix} P_xN/(-P_z) \\ P_yN/(-P_z) \\ -N \\ 1 \end{bmatrix}}
$$

In Homogeneous Matrix Form

Reminder:

$$
\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \rightarrow \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \xrightarrow[\times w]{} \begin{bmatrix} wP_x \\ wP_y \\ wP_z \\ w \end{bmatrix} \xrightarrow[\div w]{\text{homogenize}} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}
$$

(a line in 4D space)

Perspective projection:

$$
\begin{bmatrix} P_x' \\ P_y' \\ P_z' \\ 1 \end{bmatrix} = \begin{bmatrix} P_xN/(-P_z) \\ P_yN/(-P_z) \\ -N \\ 1 \end{bmatrix}
$$

In Homogeneous Matrix Form

Reminder:

$$
\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \rightarrow \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \xrightarrow[\times w]{} \begin{bmatrix} wP_x \\ wP_y \\ wP_z \\ w \end{bmatrix} \xrightarrow[\div w]{\text{homogenize}} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}
$$

(a line in 4D space)

Perspective projection:

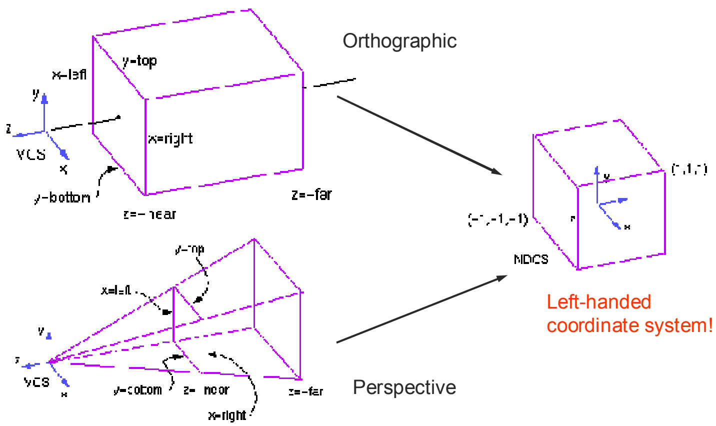
$$
\begin{bmatrix} P_x' \\ P_y' \\ P_z' \\ 1 \end{bmatrix} = \begin{bmatrix} P_xN/(-P_z) \\ P_yN/(-P_z) \\ -N \\ 1 \end{bmatrix} \xrightarrow[-P_z/N]{\times} \boxed{\begin{bmatrix} P_x \\ P_y \\ P_z \\ -P_z/N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/N & 0 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}}
$$

Therefore:

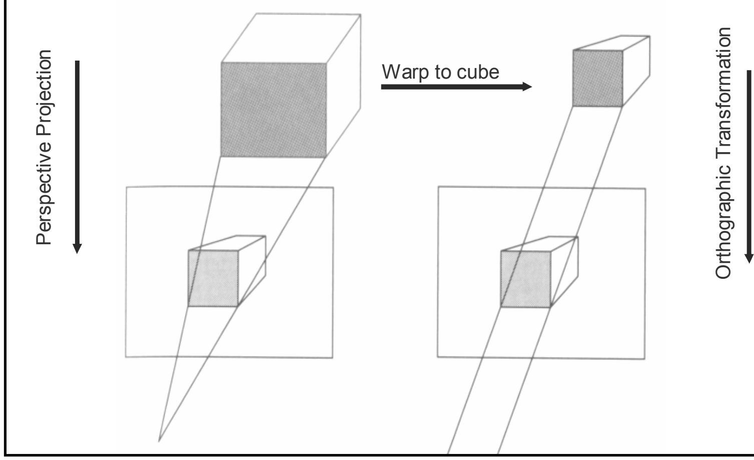
$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/N & 0 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \xrightarrow[\div -P_z/N]{\text{and then: homogenize}} \begin{bmatrix} P_x' \\ P_y' \\ P_z' \\ 1 \end{bmatrix}
$$

Homogenization step:
"Perspective Division"
(divide by $w = -P_z/N$)

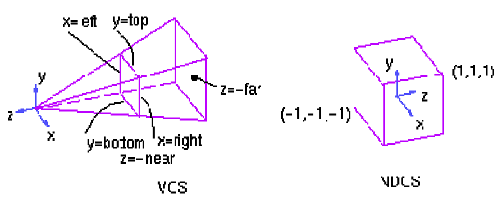
View Volumes



Perspective Projection vs Perspective Warp



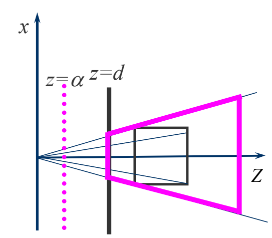
Derivation: Perspective Transformation

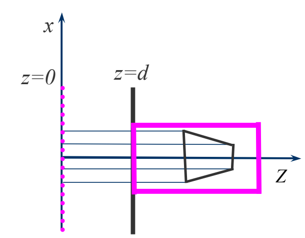


It is basically a mapping of planes

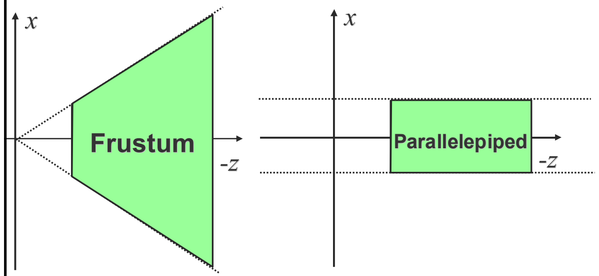
Projection Normalization

- warp perspective view volume to orthogonal view volume
  - render all scenes with orthographic projection!
  - aka perspective warp





Derivation: Perspective Transformation Matrix



$$\begin{bmatrix} \frac{P_x n}{P_z} \\ \frac{P_y n}{P_z} \\ n+f-\frac{fn}{P_z} \\ 1 \end{bmatrix}$$

Therefore:

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

Perspective Transformation Matrix

$$\mathbf{M}_P = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z\frac{n+f}{n} - f \\ \frac{P_z}{n} \end{bmatrix} \xrightarrow[\div\left(h=\frac{P_z}{n}\right)]{\text{homogenize}} \begin{bmatrix} \frac{P_x n}{P_z} \\ \frac{P_y n}{P_z} \\ n+f-\frac{fn}{P_z} \\ 1 \end{bmatrix}$$

The Projection Matrix

$$\mathbf{M}_{\text{proj}} = \mathbf{M}_O \mathbf{M}_P = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

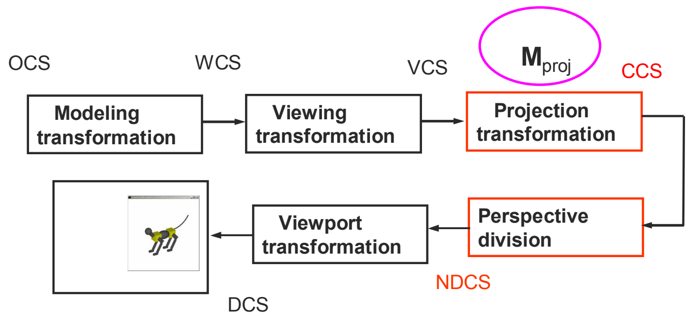
$$\mathbf{M}_{\text{proj}} = \mathbf{M}_{\text{O}} \mathbf{M}_{\text{P}} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The Projection Matrix

$$\mathbf{M}_{\text{proj}} = \mathbf{M}_{\text{O}} \mathbf{M}_{\text{P}} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- ***Followed by a perspective division in order to create homogeneous coord. and***
- ***then drop the z-coord.***

Graphics Pipeline



Orthographic Projection in OpenGL

glMatrixMode(GL_PROJECTION) ;

glLoadIdentity() ;

Followed by one of:

glOrtho(left, right, bottom, top, near, far) ;

near plane at z = -near

far plane at z = -far

gluOrtho2D(left, right, bottom, top) ;

assumes near = 0 far = 1

Perspective Projection in OpenGL

glMatrixMode(GL_PROJECTION) ;

glLoadIdentity() ;

Followed by one of:

glFrustrum(left, right, bottom, top, near, far) ;

near plane at z = -near

far plane at z = -far

gluPerspective(fovy, aspect, bottom, top) ;

fov (field of view) is measured in degrees and center at 0

Nonlinearity of Perspective Transformation

tracks in VCS:

left x=-1, y=-1

right x=1, y=-1

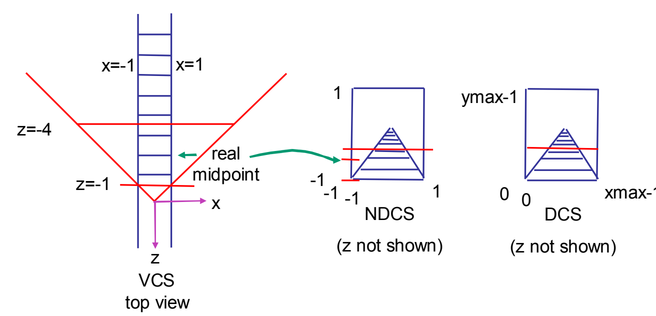
z=-inf, inf

view volume

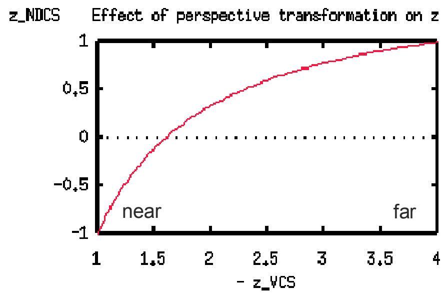
left = -1, right = 1

bot = -1, top = 1

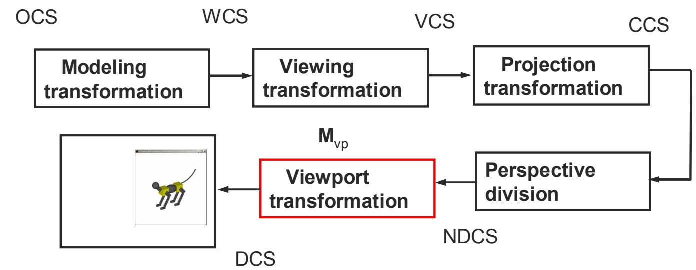
near = 1, far = 4



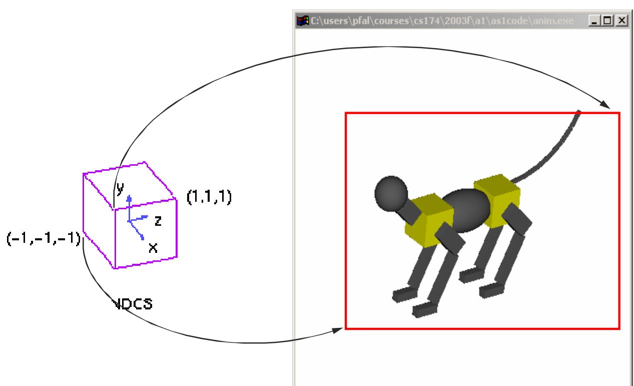
Z in NDCS vs –Z in VCS



Viewport Transformation

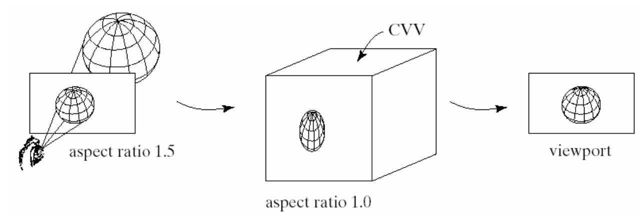


Viewport



Why Viewports?

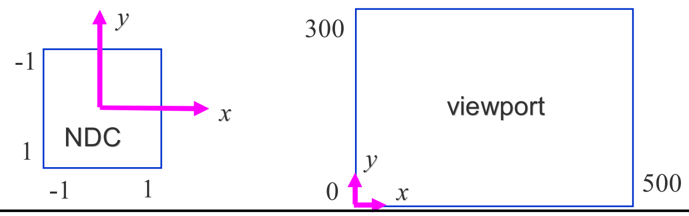
Undo the distortion of the projection transformation



NDC to Device Transformation

- map from NDC to pixel coordinates on display
  - NDC range is x = -1...1, y = -1...1, z = -1...1
  - typical display range: x = 0...500, y = 0...300
    - maximum is size of actual screen
    - z range max and default is (0, 1), use later for visibility

glViewport(0,0,w,h);
glDepthRange(0,1); // depth = 1 by default



Viewport Matrix

- Leave *z* coordinates unchanged
- Transform x,y coordinates to a viewport of size n_x x n_y square pixels (assume pixel size: 1.0x1.0), from (0,0) at lower left; thus, viewport is [-0.5, n_x-0.5] x [-0.5, n_y-0.5]

$$\mathrm{M}_{VP} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \frac{n_x-1}{2} \\ 0 & 1 & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Translation}} \underbrace{\begin{bmatrix} \frac{n_x}{2} & 0 & 0 & 0 \\ 0 & \frac{n_y}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Scaling}} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

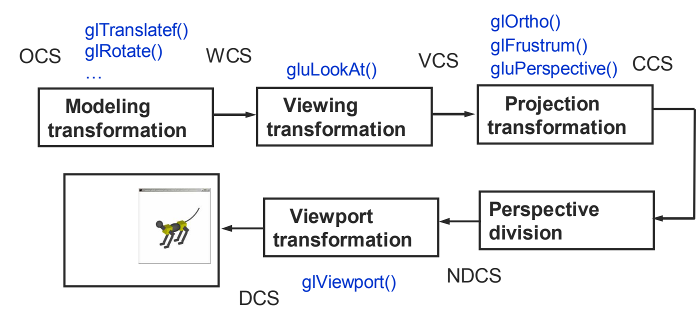
- What would change if *y* were increasing downward?
  - Answer: -n_y/2 in scaling matrix

Viewport in OpenGL

glViewport(Glint x, GLint y, GLsizei width, GLsizei height) ;

x,y: lower left corner of viewport rectangle in pixels
width, height: width and height of viewport.

GL Functions in the Pipeline

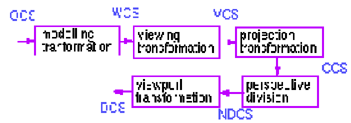


3D Clipping

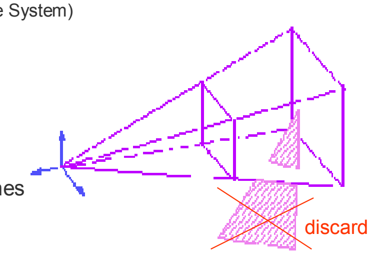
Keep only what is visible

We can clip

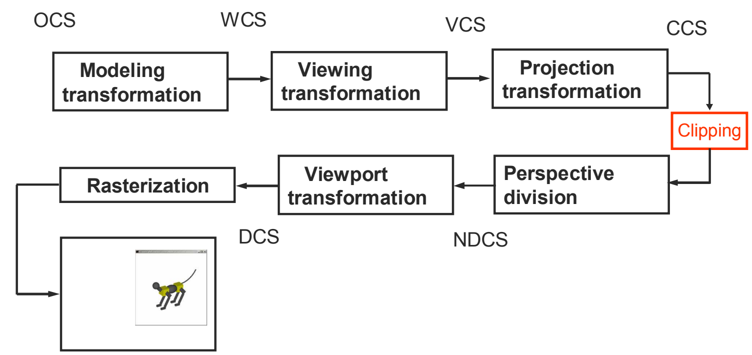
1. in the VCS
   - What are the six plane equations?
2. in the CCS (a.k.a. Clipping Coordinate System)
   - Clipping in homogeneous coords
   - 4D volume bounded by 3D planes
   - Still simple and efficient
   - **Usually done here (!)**
3. in the NDCS
   - Singularity at $P_z = 0$



In any case, we must clip against planes



Z-Buffer Graphics Pipeline



Background (Reminder)

Plane equations

- **Implicit**

$F(x, y, z) = Ax + By + Cz + D = \mathbf{N} \cdot P + D$

Points on Plane $F(x, y, z) = 0$

- **Parametric**

$Plane(s,t) = P_0 + s(P_1 - P_0) + t(P_2 - P_0)$

P_0, P_1, P_2 not collinear

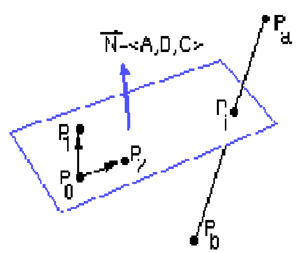
or

$Plane(s,t) = (1 - s - t)P_0 + sP_1 + tP_2$

$Plane(s,t) = P_0 + sV_1 + tV_2$, where V_1, V_2 are basis vectors

- **Explicit**

$z = -(A/C)x - (B/C)y - (D/C), \quad C \neq 0$



Intersection of Line and Plane

Implicit equation for the plane:

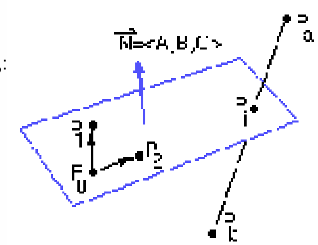
$$F(P) = \mathbf{N} \cdot P + D = 0$$

Parametric equation for the line from P_a to P_b:

$$L(t) = P_a + t(P_b - P_a)$$

Plug $L(t)$ into $F(P)$ and solve for $t = t_i$:

$$\mathbf{N} \cdot [P_a + t_i(P_b - P_a)] = -D$$



Intersection of Line and Plane

Implicit equation for the plane:

$$F(P) = \mathbf{N} \cdot P + D = 0$$

Parametric equation for the line from P_a to P_b:

$$L(t) = P_a + t(P_b - P_a)$$

Plug $L(t)$ into $F(P)$ and solve for $t = t_i$:

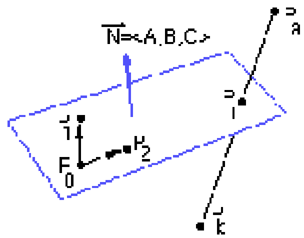
$$\mathbf{N} \cdot [P_a + t_i(P_b - P_a)] = -D$$

Therefore,

$$t_i = \frac{-D - \mathbf{N} \cdot P_a}{\mathbf{N} \cdot P_b - \mathbf{N} \cdot P_a} = \frac{-F(P_a)}{F(P_b) - F(P_a)}$$

Finally, evaluate $L(t_i)$ for intersection point P_i:

$$P_i = P_a - \frac{F(P_a)(P_b - P_a)}{F(P_b) - F(P_a)} = \frac{P_a F(P_b) - P_b F(P_a)}{F(P_b) - F(P_a)}$$



Culling

When an entire triangle lies outside the view volume, it can be “culled”

- i.e., eliminated from the pipeline
- Especially helpful when many triangles are grouped into an object with an associated bounding volume (e.g., a bounding sphere) which lies outside the view volume (signed distance to plane > sphere radius)

A Line Rendering Algorithm

Compute $\mathbf{M}_{mod}$
Compute $\mathbf{M}^{-1}_{cam}$
Compute $\mathbf{M}_{modelview} = \mathbf{M}^{-1}_{cam}\,\mathbf{M}_{mod}$
Compute $\mathbf{M}_{O}$
Compute $\mathbf{M}_{P}$ // disregard $\mathbf{M}_P$ here and below for orthographic-only case
Compute $\mathbf{M}_{proj} = \mathbf{M}_O\,\mathbf{M}_P$
Compute $\mathbf{M}_{vp}$
Compute $\mathbf{M} = \mathbf{M}_{vp}\,\mathbf{M}_{proj}\,\mathbf{M}_{modelview}$
for each line segment *i* between points P_i and Q_i **do**
  $P = \mathbf{M}P_i$; $Q = \mathbf{M}Q_i$
  drawline(P_x/w_P, P_y/w_P, Q_x/w_Q, Q_y/w_Q) // w_P, w_Q are 4th coords of P, Q
end for