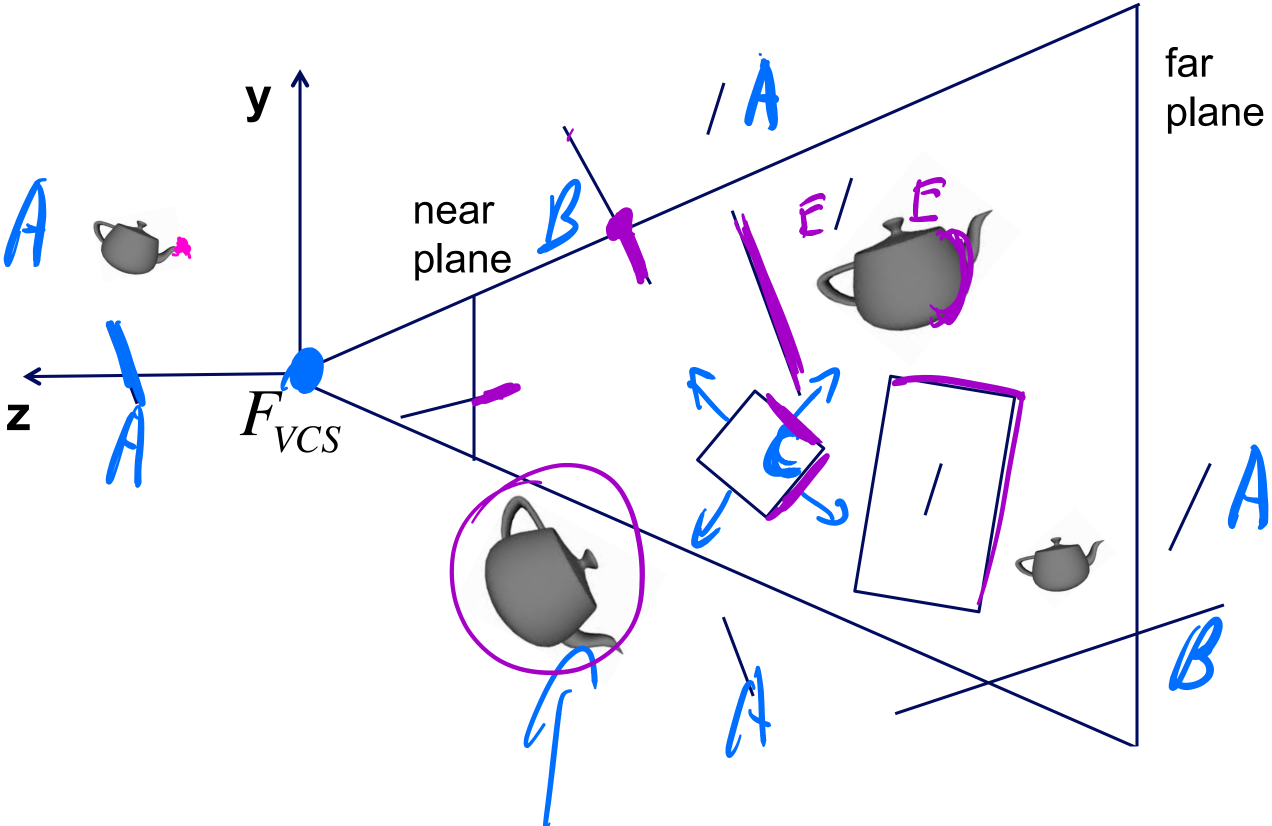
Visibility

Determining which objects / triangles / pixels can be seen



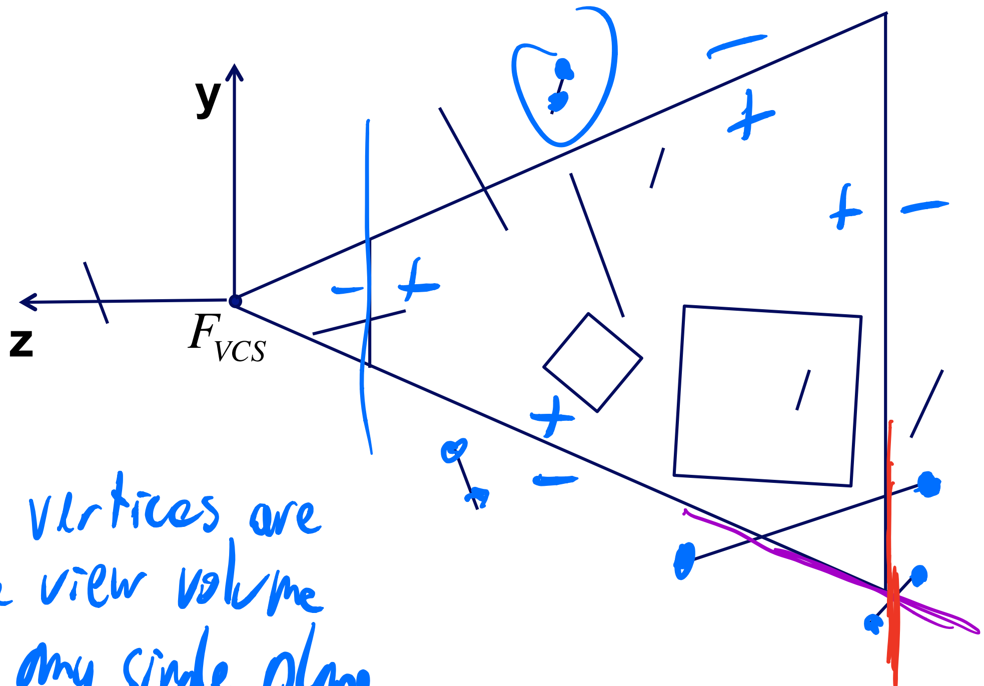
Visibility

Methods

- A view volume culling → triangles or objects
- B view volume clipping
- C backface culling
- D occlusion: z-buffer test → pixel level
- E occlusion: object culling →
- raycasting (and raytracing)

OpenGL / WebGL / DirectX support
A B C D for triangles

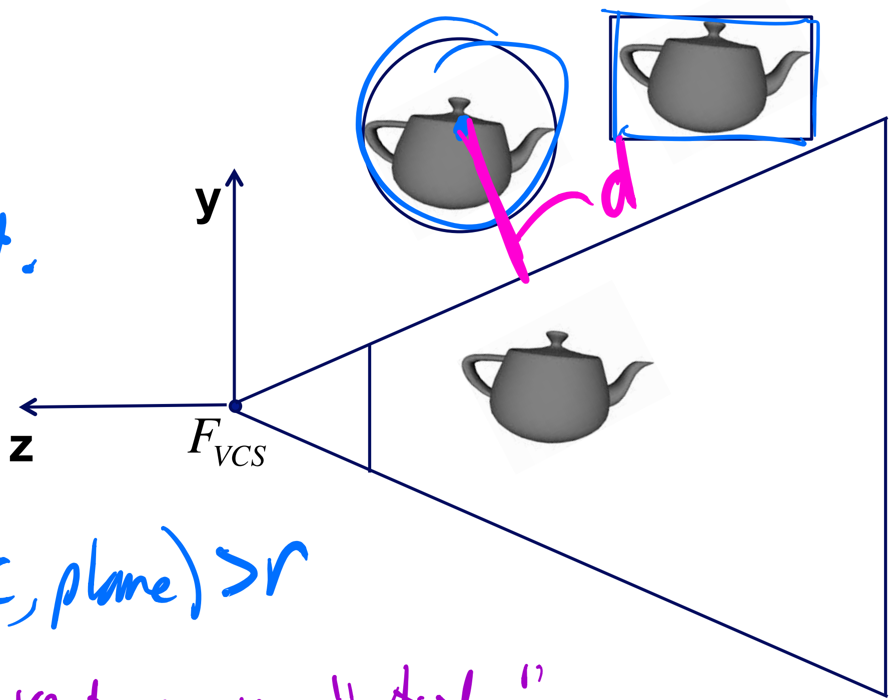
View Volume Culling (for triangles)



Idea: cull if all vertices are outside the view volume with respect any single plane.

View Volume Culling (for objects)

Idea: fast test
for entire object.



bounding sphere:

Cull if dist(c, plane) > r

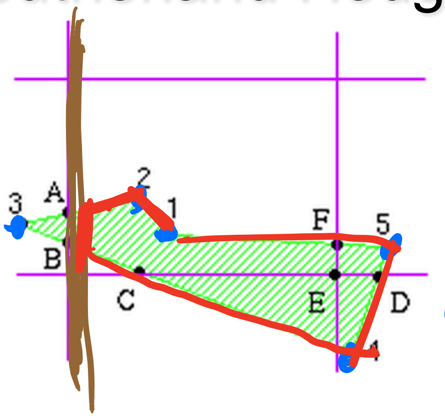
bounding box:

Cull if all 8 vertices are "outside"
with respect to one of the frustum planes

2D Clipping

vertices → clip L → clip B → clip R → clip T → clipped polygon vertex list

Sutherland Hodgeman algorithm



Original: 1, 2, 3, 4, 5, 1

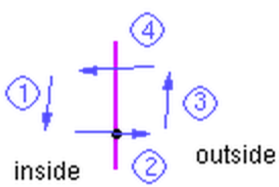
clip L: 1, 2, A, B, 4, 5, 1

clip B: 1, 2, A, B, C, D, 5, 1

clip R: 1, 2, A, B, C, E, F, 1

clip T: (same)

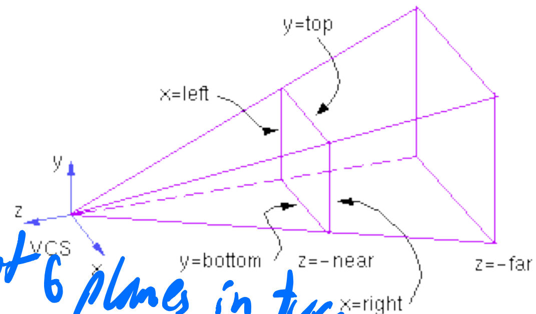
```
for each side of clipping window
    for each edge of polygon
        output points based upon the following table
```



case #	first point	second point	output point(s)
1	inside	inside	second point
2	inside	outside	intersection point
3	outside	outside	none
4	outside	inside	intersection point and second point

e.g., vertex A

View Volume Clipping in VCS (✓ works)



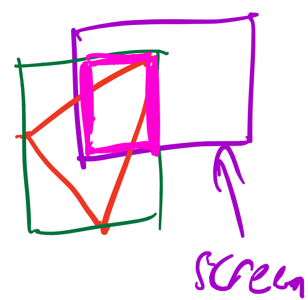
general polygon clipping:

clip against each of 6 planes in turn

original vertex list → → → → → → clipped vertex list.

tor triangles with bounding-box scan conversion:

vertex list → clip near → clip far → clip bounding box →



Clipping in VCS

and Culling

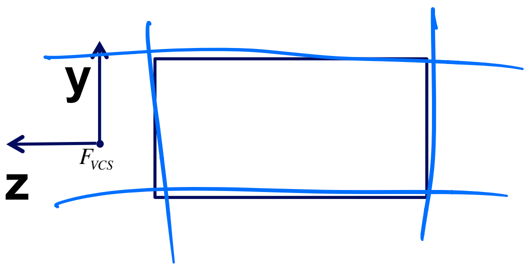
Plane equations

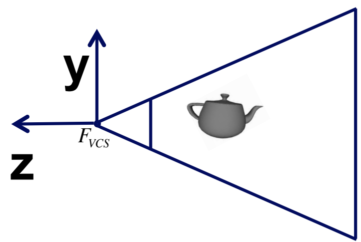
Othographic View Volume

```
left:     x - left = 0
right:   -x + right = 0
bottom:   y - bottom = 0
top:     -y + top = 0
front:   -z - near = 0
back:     z + far = 0
```

Perspective View Volume

```
left:     x + left*z/near = 0
right:   -x - right*z/near = 0
top:     -y - top*z/near = 0
bottom:   y + bottom*z/near = 0
front:   -z - near = 0
back:     z + far = 0
```



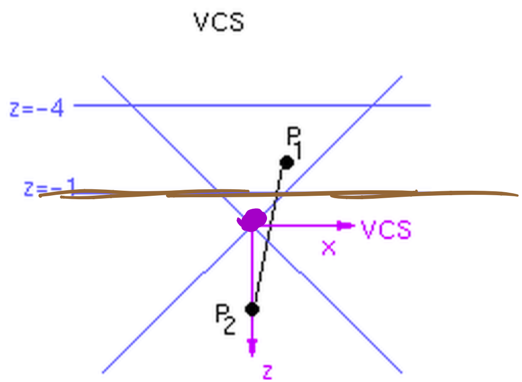


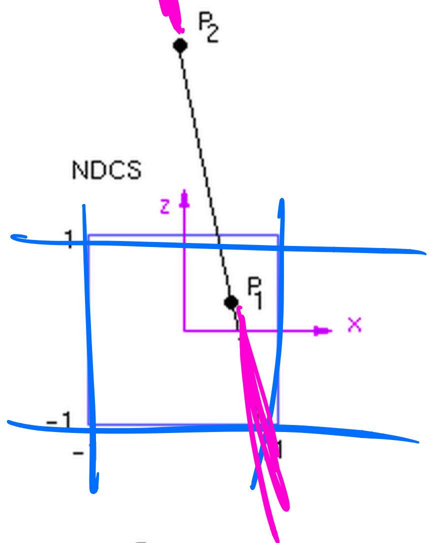
$P_{vcs} \rightarrow \boxed{M_{proj}} \xrightarrow{P_{ccs}} \boxed{/h} \rightarrow P_{ndcs}$

Clipping in NDCS (?)

NDCS

I





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

	P_1	P_2
VCS	(1, 0, −2)	(0,0,2)
CCS	(1, 0, 2/3, 2)	(0,0,−6,−2)
NDCS	(1/2, 0, 1/3)	(0, 0, 3)

$P_{ccs} = M_{proj} P_{vcs}$

Clipping in CCS

$$P_{VCS} \rightarrow \boxed{M_{proj}} \xrightarrow{P_{CCS}} \boxed{/h} \xrightarrow{P_{NDCS}}$$

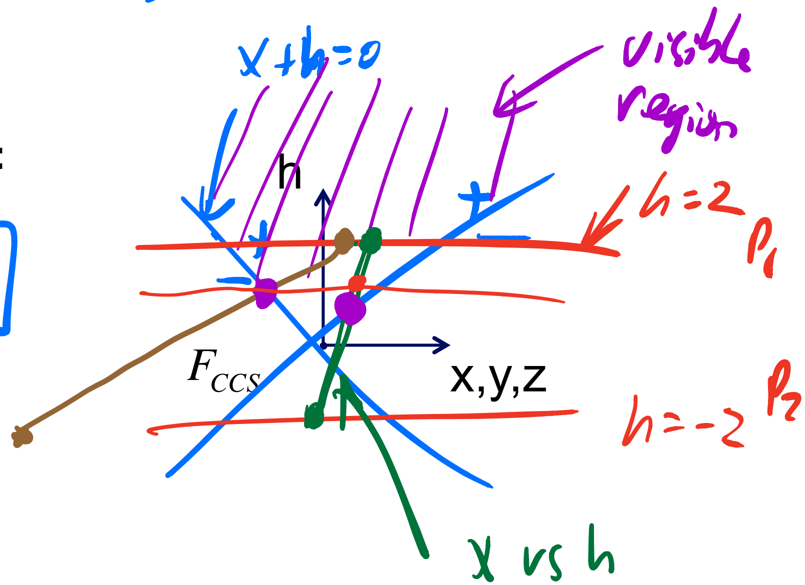
NDCS: $-1 \le x_{NDCS} \le 1 \qquad -1 \le \frac{x_{ccs}}{h_{ccs}} \le 1$

CCS: $-h_{ccs} \le x_{ccs} \le h_{ccs}$

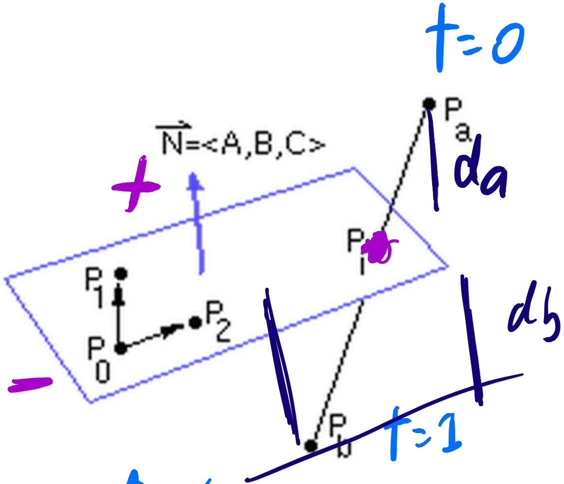
canonical plane equations:

$F(x,y,z,h) > 0$ inside

```
left:    x + h = 0
right:  -x + h = 0
bot:     y + h = 0
top:    -y + h = 0
near:    z + h = 0
far:    -z + h = 0
```



Line-Plane intersection



Plane eqn:

$$\vec{N} = (P_2 - P_0) \times (P_1 - P_0)$$

$$Ax + By + Cz + D = 0$$

$$\langle A, B, C \rangle \cdot \langle x, y, z \rangle + D = 0$$

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

$$D = -N \cdot P_0$$

Line equation:

$$\boxed{P(t) = P_a + t(P_b - P_a)}$$

$$N \cdot [P_a + t(P_b - P_a)] + D = 0$$

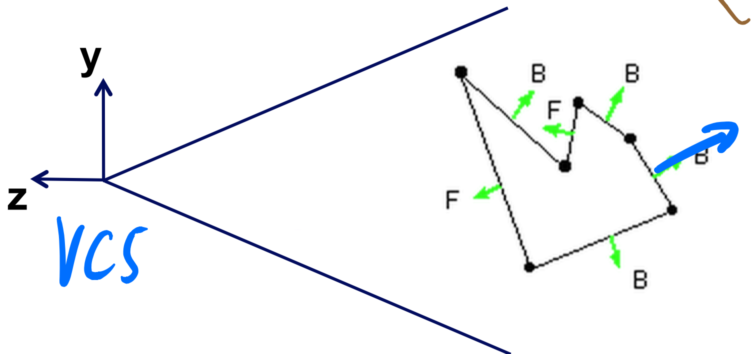
$$= N \cdot P_a + t(N \cdot P_b - N \cdot P_a) + D = 0$$

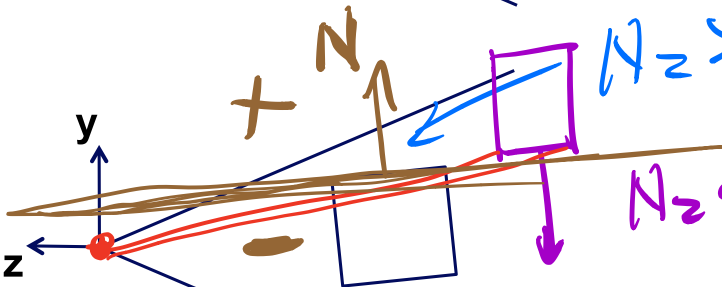
$$t = \frac{-N \cdot P_a - D}{\underset{+D}{N \cdot P_b} - \underset{-D}{N \cdot P_a}} = \frac{-F(P_a)}{F(P_b) - F(P_a)} = \frac{d_a}{d_a + d_b}$$

Backface Culling in VCS

Bad

Idea: cull if $N_z < 0$





$N_z > 0$ but it is invisible

$N_z < 0$ but it is visible

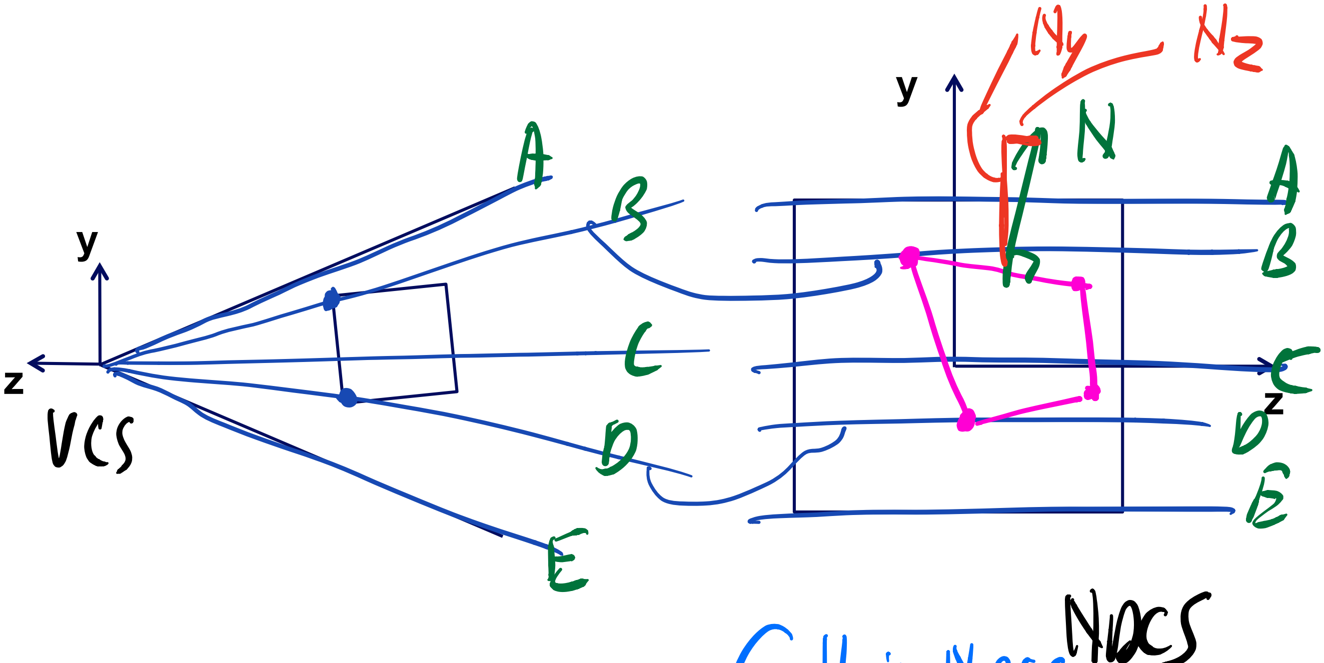
Correct VCS backface culling:

Cull if $P_{eye}(0,0,0)$ is below the plane of the polygon

$\vec{N} \cdot P + D > 0$

cull if $D < 0$

Backface Culling in NDCS



Cull in NDCS if
N_z points away
$\equiv N_z > 0$

Transforming Normals

Scales, rotate, shear or basis vectors.

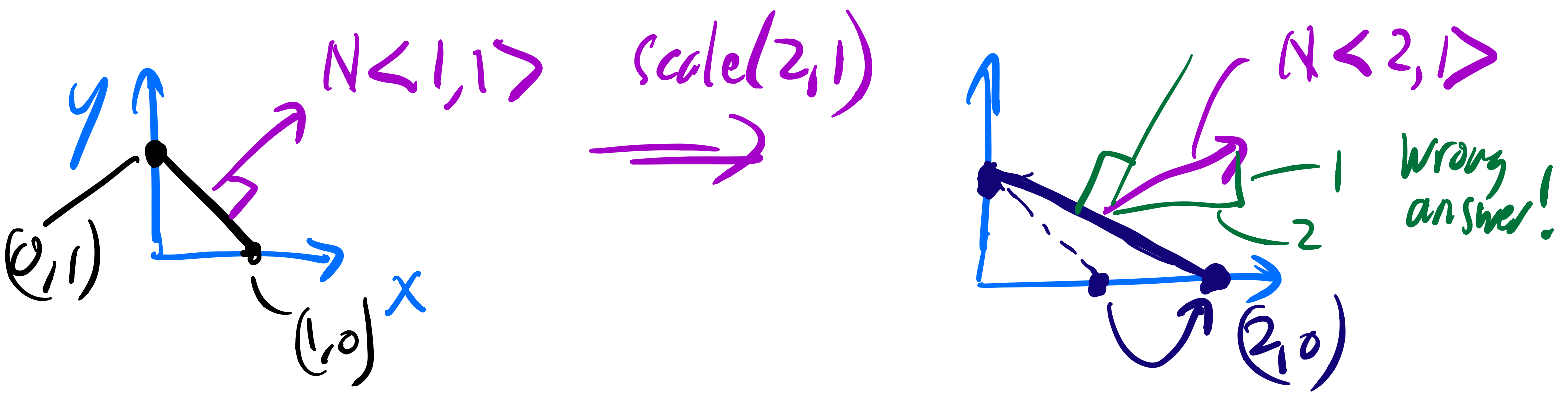
Translation or Origin

Using h=0

$$\begin{bmatrix} i & j & k & \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} N_x \\ N_y \\ N_z \\ 0 \end{bmatrix}$$

skip the translation.

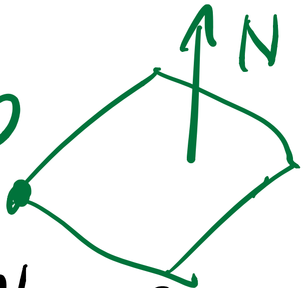
Problem (in the case of non-uniform scaling)



Transforming Normals

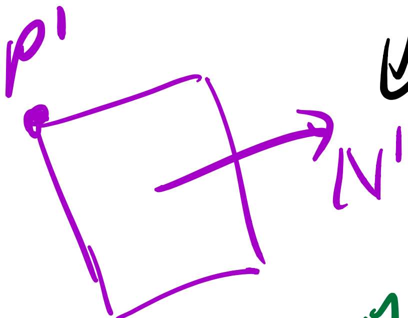
$N = \langle A, B, C \rangle \leftarrow D$

consider a plane, before and after transformation:



$$P' = MP$$

$$N' = \hat{M} N$$



$$Ax + By + Cz + D = 0$$

$$\begin{bmatrix} A & B & C & D \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

Write this as

$$N^T \cdot P = 0$$

$$N'^T \cdot P' = 0$$

$$(\hat{M} N)^T (M \cdot P) = 0$$

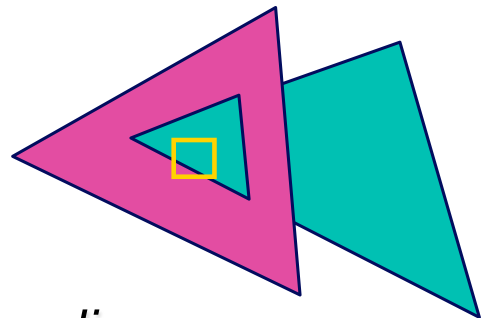
$$N^T (\hat{M}^T M) P = 0$$

$$\hat{M}^T \cdot M = I$$

$$\hat{M} = (M^{-1})^T$$

Occlusion

view occluded by objects in front of a given pixel or polygon ?



- image space algorithms:
  - *operate on pixels or scan-lines*
  - *visibility resolved to the precision of the display*
  - *e.g.: Z-buffer*
- object space algorithms:
  - *explicitly compute visible portions of polygons*
  - *painter's algorithm: depth-sorting, BSP trees*

Z-buffer

$Z_{NDCS} \in [-1,1]$

store (r,g,b,z) for each pixel

1.0

$Z_{DCS} \in [0,1]$

```
for all i,j {
 Depth[i,j] = MAX_DEPTH
 Image[i,j] = BACKGROUND_COLOUR
}
for all polygons P {
  project vertices into screen-space, i.e., DCS
  for all pixels in P {
    if (Z_pixel < Depth[i,j]) {     // closer?
      Image[i,j] = C_pixel   // overwrite pixel
      Depth[i,j] = Z_pixel   // overwrite z
    }
  }
}
```

Z-buffer

- hardware support
- extra memory
- jaggies, i.e., steps along intersections
- poor performance for high depth complexity scenes;
  - use occlusion culling to mitigate this

"early z-test": do z-buffer test, then call fragment shader
pro: potential computational savings

standard: call fragment shader, then test z
pro: fragment shader can modify z

Occlusion Culling

- occlusion queries
  - virtual render of bounding box

do not change pixel values,
just count # pixels
that pass the z-buffer
test.

- precomputed visibility tables
  - *store a list of visible cells*
- horizon maps
  - *for terrain models*

Visibility in Practice: WebGL, OpenGL

Commonly supported by hardware & OpenGL / DirectX

- view volume culling (for triangles)
- view volume clipping
- backface culling
- z-buffer occlusion test

Software, i.e., on your own

- view volume culling (for objects)
- occlusion culling

Raycasting and Raytracing

alternative to projective rendering

- for each pixel p
  - *construct ray r from eye through p*
  - *intersect r with all polygons or objects*
  - *color p according to closest surface*