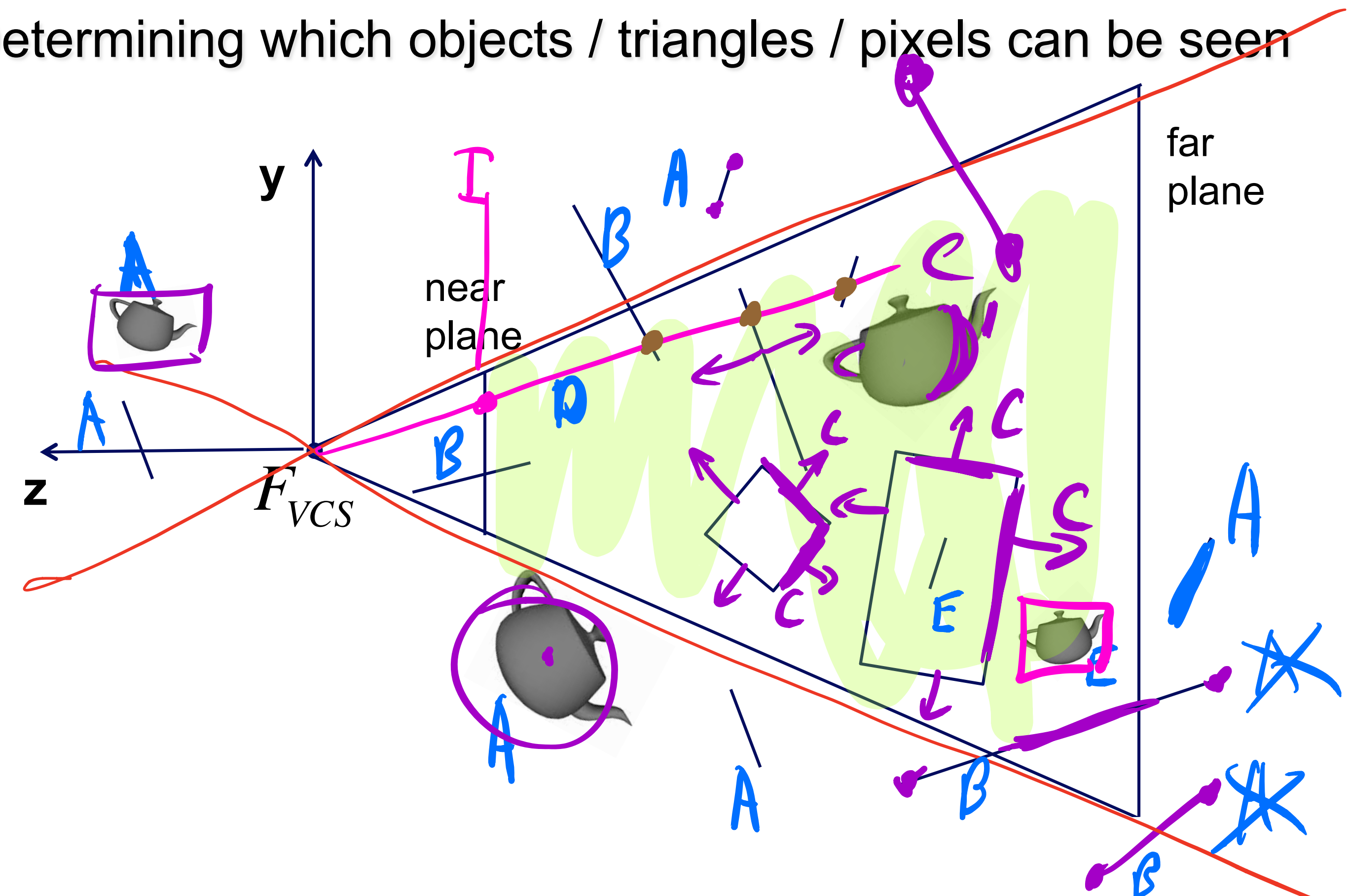
Visibility

Determining which objects / triangles / pixels can be seen



Visibility

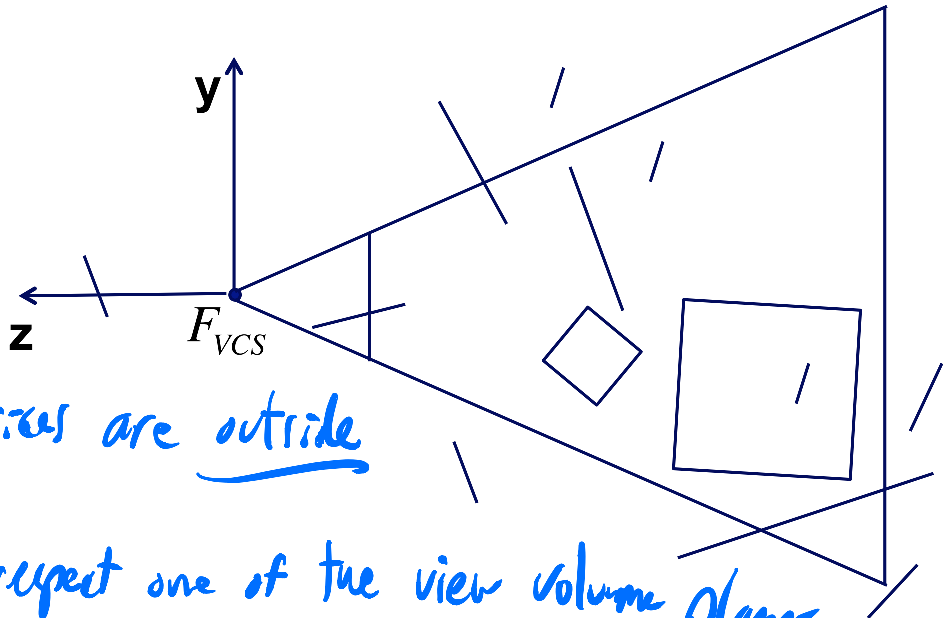
Methods

- A • view volume culling — removes triangles or objects outside of view volume
- B • view volume clipping — triangles or objects that cross into the VV
- C • backface culling
- D • occlusion: z-buffer test
- E • occlusion: object culling — test a fake render of the bounding cube to see if any pixels would have been visible
- • raycasting (and raytracing) — later.

three.js : A for objects

WebGL/OpenGL/DirectX/GPU : A B C D for triangles.

View Volume Culling (for triangles)



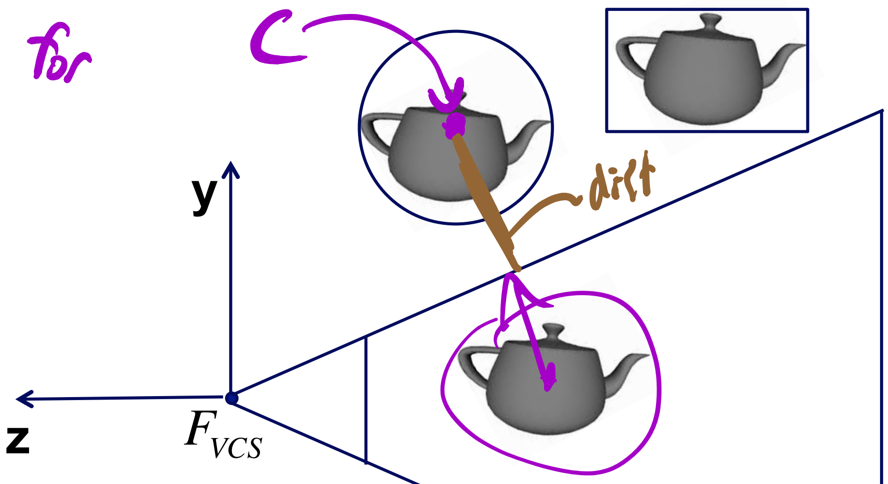
Idea: Cull if all vertices are outside the viewvolume

→ outside with respect one of the view volume planes

^ needs to be

View Volume Culling (for objects)

Idea: fast cull test for
entire objects



bounding sphere:

cull if dist(C, plane) > r for at least one of the view volume planes

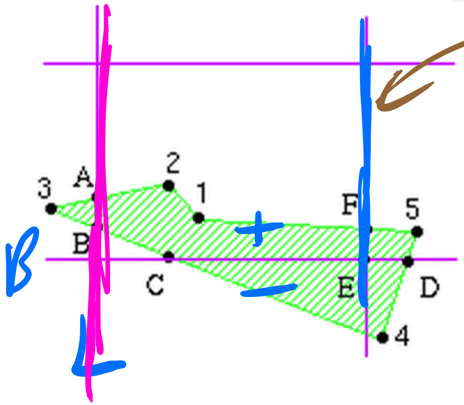
"outside"

bounding box:

cull if all 8 vertices are "outside" for at least one plane.

2D Clipping

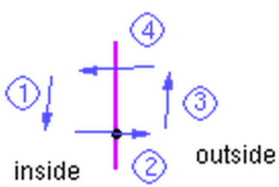
Sutherland Hodgeman algorithm



display region / viewport.

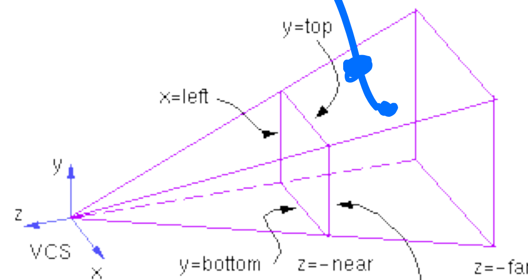
original: 1, 2, 3, 4, 5, 1
clip L: 1, 2, A, B, 4, 5, 1
clip B: 1, 2, A, B, C, D, 5, 1
clip R: 1, 2, A, B, C, E, F, 1
clip T: (same)

```
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

View Volume Clipping

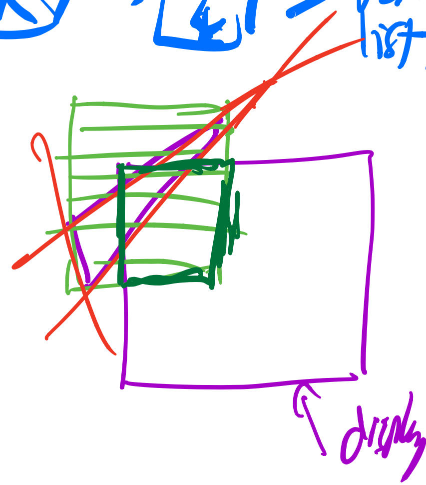


general polygon clipping:

Original 3D Vertex list → clip near → clip far → clip top → clip bot → clip R → clip L → Clipped vertex list.

tor triangles with bounding-box scan conversion:

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



Clipping in VCS

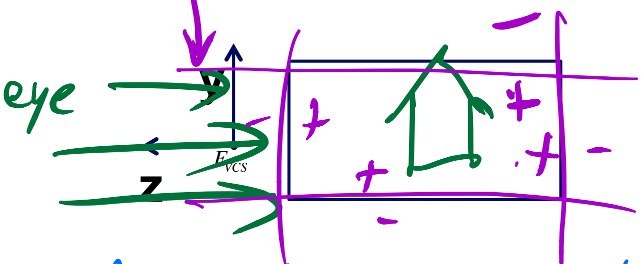
Plane equations

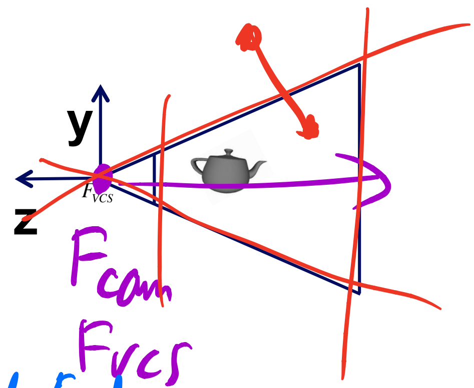
Othographic View Volume

```
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
```





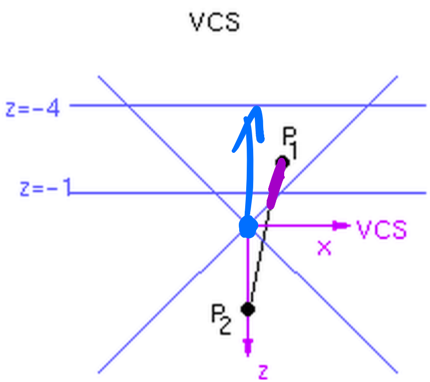
Note: clipping in VCS works, but the equations still depend on user-defined variables

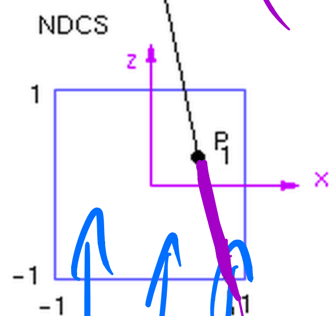
Clipping in NDCS (?)

+ canonical plane equations
− problematic for segments that cross z=0 plane

(don't cull in NDCS)

NDCS top view





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

M_{proj}

	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)

M_{proj}

/h

Clipping in CCS

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

clipping coordinate system

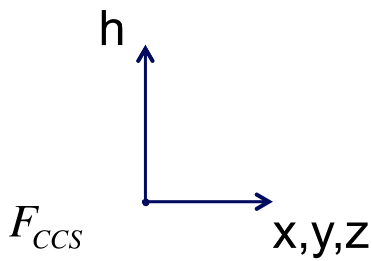
NDCS: $-1 \le \dfrac{x_{ccs}}{h_{ccs}} \le 1$

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

use this

canonical plane equations:

```
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

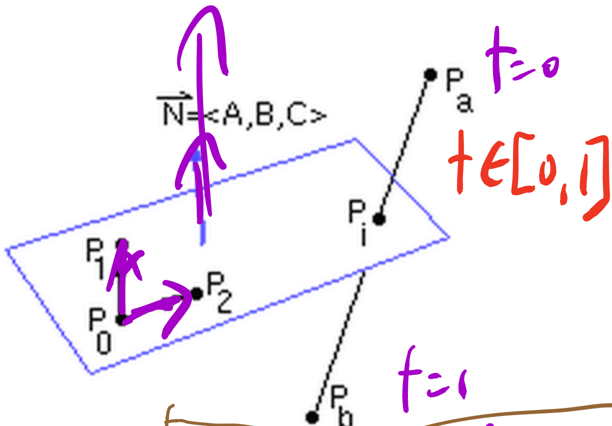
Plane Eqn: $\vec{N} = (P_2 - P_0)(P_1 - P_0)$

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

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

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

$$D = -\vec{N} \cdot \vec{P_0}$$



Line eq'n: $\boxed{P(t) = P_a + t(P_b - P_a)}$

substitute here to compute P(t)

Substitutions:

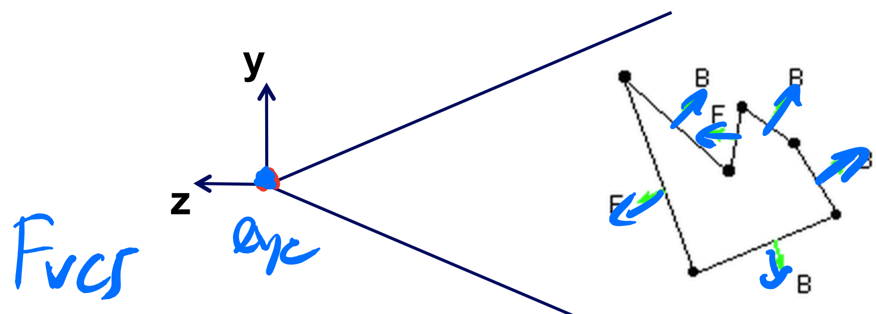
$$N \cdot [P_a + t(P_b - P_a)] + D = 0$$

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

+D − D

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

Backface Culling in VCS



Idea: cull if $N_z < 0$

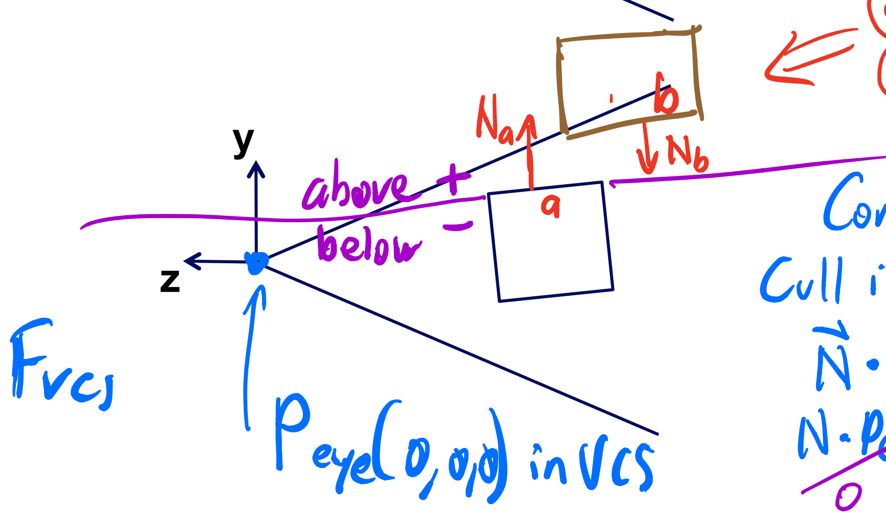
B = back facing

F = front facing

Problematic cases

(a) should be culled, but it is not.

(b) should not be culled, but it is



Correct VCS culling:

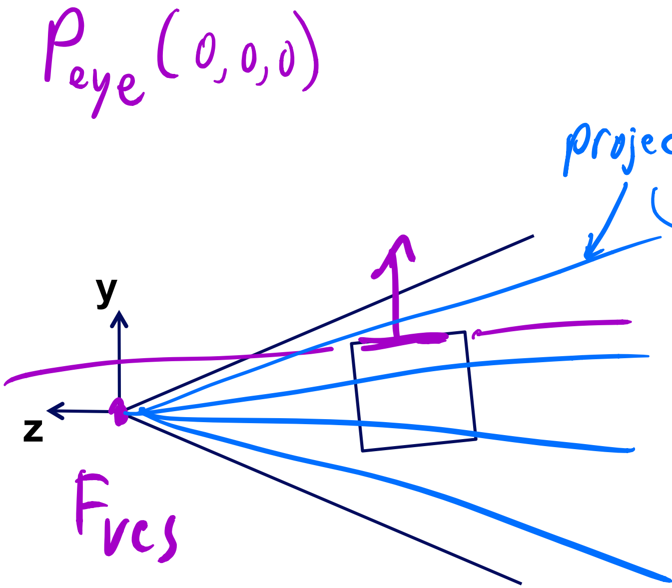
Cull if P_{eye} is below the plane

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

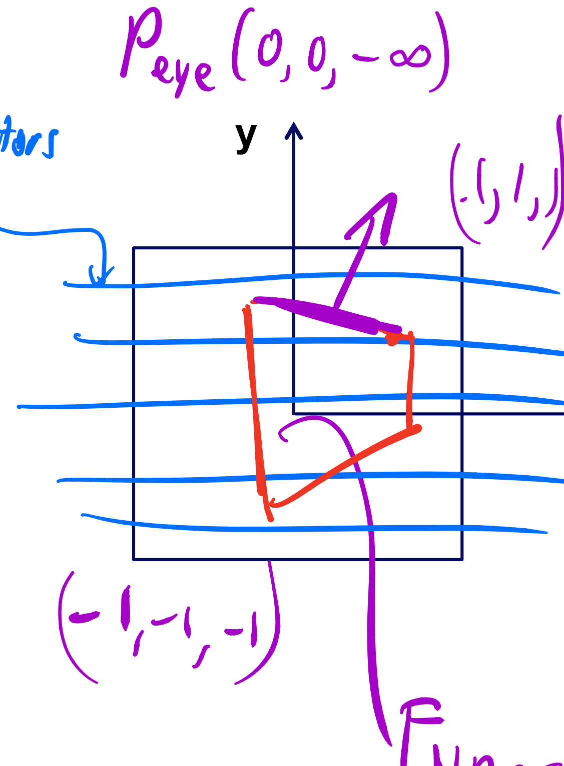
$$\underbrace{\cancel{N\cdot P_{eye}}}_{0} + D = F(P_{eye})$$

$\Rightarrow$ cull if $D < 0$

Backface Culling in NDCS



cull if eye below plane.



cull if $N_z > 0$

left handed coordinate system (z has been flipped)

Transforming Normals

Using h=0

$$\begin{bmatrix} & & & \\ & & & \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

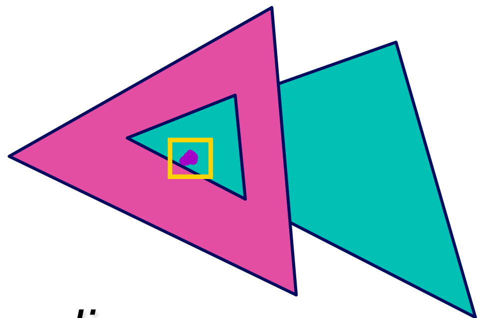
Problem

Transforming Normals

consider a plane, before and after transformation:

Occlusion

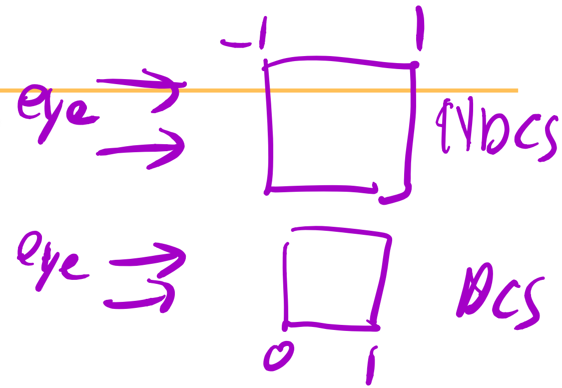
view occluded by objects in front of a given pixel or polygon ?



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

$$z_{DCS} = \frac{z_{NDCS} + 1}{2}$$

Z-buffer



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

z_{DCS}

```
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

Occlusion Culling

- occlusion queries
  - virtual render of bounding box
- 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*