Visibility

University of British Columbia

- view volume culling
- view volume clipping
- backface culling
- z-buffer occlusion test
- painter's algorithm & BSP trees
- occlusion culling
- raytracing

(1): view volume culling, view volume clipping

(2): backface culling, z-buffer occlusion test, painter's algorithm & BSP trees, occlusion culling

raytracing → (1), (2)

How to efficiently compute what can and cannot be seen.

(1) outside of view volume

(2) occluded by other objects

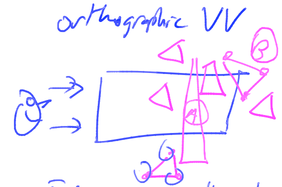
View Volume Culling

VV

University of British Columbia

polygons outside the view volume ?

orthographic VV



(B) If all vertices "outside" wrt any one plane then cull.

Idea: If all vertices "outside", then cull the polygon

Problem: fails sometimes (A), (B)

Fix: (1) For each vertex V

(2) For each plane P

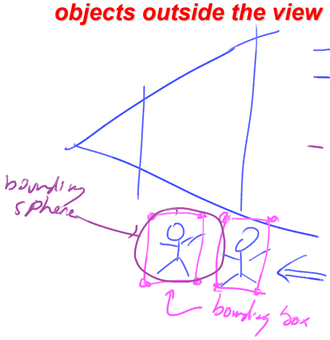
(3) compute vertex V is inside/outside wrt P

"outcodes"

View Volume Culling

University of British Columbia

objects outside the view volume ?

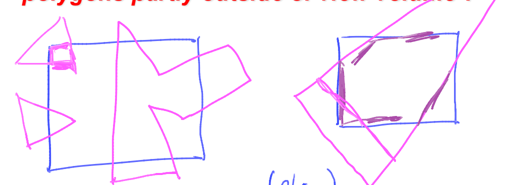


- inefficient to cull individual △s
- if all bbox vertices are "outside" wrt any plane, then cull
- if distance from sphere centre to the view volume > r, then cull.

View Volume Clipping

University of British Columbia

polygons partly outside of view volume ?



even a clipped triangle can have 7 edges in 2D

Idea: clip against one edge (plane) at a time

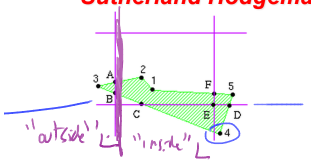
2D vertices → clip T → clip B → clip R → clip L → clipped polygon

3D → near → far → L → R → T → B →

View Volume Clipping

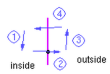
University of British Columbia

Sutherland Hodgeman clipping



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 ⟸ final
clip T : (unchanged)

```
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

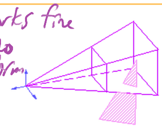
later: line-plane intersections.

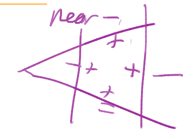
Clipping in VCS

University of British Columbia

VCS

– clipping here works fine
– but there is no canonical form





plane eq'ns for orthographic 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
```

Implicit plane equations for view frustum

```
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
```

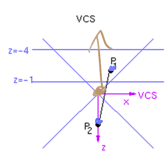
$F(x,y,z) \geq$ inside
$F(x,y,z) <$ outside

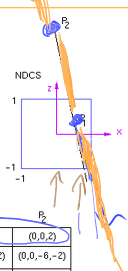
– canonical plane eq'ns ✓
– problem: points behind eye cause problems

Clipping in NDCS

University of British Columbia

NDCS





fix: clip against near in VCS first

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

	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)

– canonical frame ✓
– before /h ✓

Clipping in CCS

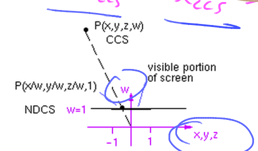
University of British Columbia

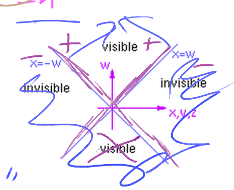
$\boxed{w \equiv h}$

Clipping usually done here

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

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





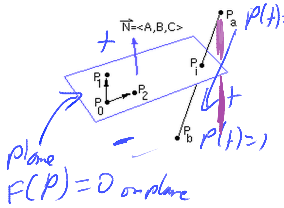
plane equations:

L $x + w = 0 = F(x,y,z,w)$
R $-x + w = 0 = F(\ )$
B $y + w = 0$
T $-y + w = 0$
N $z + w = 0$
F $-z + w = 0$

$F(x,y,z,w) \geq 0$ "inside"
$F(x,y,z,w) < 0$ "outside"

Line-Plane intersection

University of British Columbia



plane equation:

① $Ax + By + Cz + D = 0 = F(P)$

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

$N \cdot P + D = 0$

② $P(t) = P_a + t(P_b - P_a)$

subst. into to get intersect. pt.

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

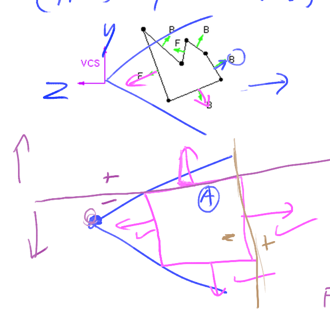
$N\cdot P_a + t(N\cdot P_b - N\cdot P_a) + D = 0$

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

Backface Culling in VCS

University of British Columbia

(applies only to "closed" objects, or objects that have well defined front/back faces)



Idea: cull if $N_z < 0$

Problem: polygon A would not be culled.

Fix: test to see if eye is below plane. If below, then cull.

Plane: $Ax + By + Cz + D = F(x,y,z)$

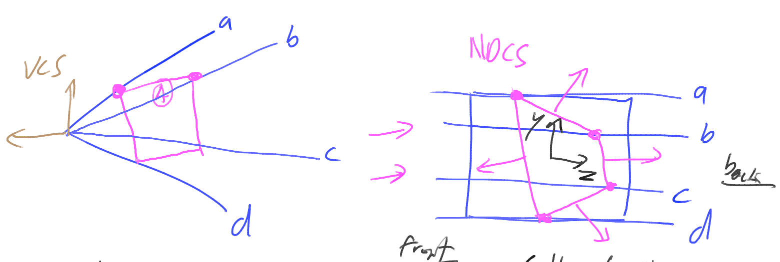
$N \cdot P + D = 0$

For eye: $P(0,0,0)$ $D = -N\cdot P_1$

If $D<0$ then cull, ie, if $-N\cdot P_1 < 0$ then cull

Backface Culling in NDCS

University of British Columbia

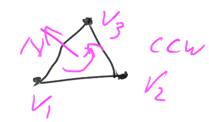


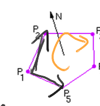
Cull if $N_z > 0$

can cull in either VCS (plane check) or NDCS (N_z check)

Computing Surface Normals

University of British Columbia





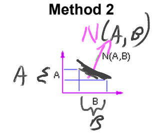
clockwise numbering when seen from the front side

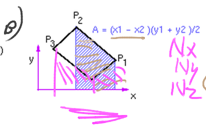
(more often use CCW)

Method 1

cross product $N = (P_3 - P_1)\times(P_2 - P_1)$

Method 2





$N_x = \sum_{i=1}^{n}(y_i - y_j)(z_i + z_j)/2$ = area projected onto yz-plane

$N_y = \sum_{i=1}^{n}(z_i - z_j)(x_i + x_j)/2$ = area on xz plane

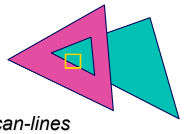
$N_z = \sum_{i=1}^{n}(x_i - x_j)(y_i + y_j)/2$ = area on xy plane.

Occlusion

University of British Columbia

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

most common, OpenGL



- 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

classic projective rendering, ie OpenGL

University of British Columbia

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

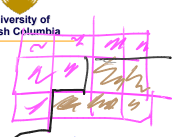
```
for all i,j {
 Depth[i,j] = MAX_DEPTH
 Image[i,j] = BACKGROUND_COLOUR
}
for all polygons P {
  project vertices into screen-space, i.e., DCS
  for all pixels in P {
    if (Z_pixel < Depth[i,j]) {
      Image[i,j] = C_pixel
      Depth[i,j] = Z_pixel
    }
  }
}
```

overwrite pixel & update depth

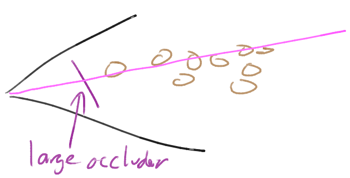
Z-buffer

University of British Columbia

- hardware support
- limitations:



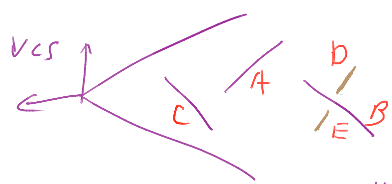
- extra memory
- jaggies ie, steps along intersecting geometry
- poor performance for high depth complexity scenes



solution: occlusion culling

Painter's Algorithm

University of British Columbia



Draw the polygons from back to front

eg: B, A, C

⇒ sort by z →

problems: - which z value to use ?
- cyclic overlap ?
- intersecting geometry ?

Fix: use BSP tree "Binary Space Partitioning"

Binary Space Partition (BSP) trees

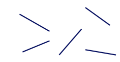
University of British Columbia

- object-space method
- produces a back-to-front ordering
- build the BSP tree once
- traverse the BSP in a view-dependent fashion

(skip)

BSP trees (example)

University of British Columbia



Building a BSP tree

University of British Columbia

```
BSPtree *BSPmaketree(polygon list) {
  choose a polygon as the tree root
  for all other polygons {
    if polygon is in front, add to front list
    if polygon is behind, add to behind list
    else split polygon and add one part to each list
  }
  BSPtree = BSPcombinetree(BSPmaketree(front list),
            root, BSPmaketree(behind list) )
}
```


Using a BSP tree

University of British Columbia

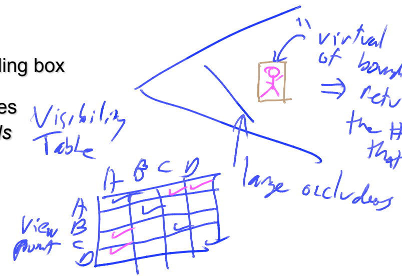
producing a back-to-front ordering

```
DrawTree(BSPtree) {
  if (eye is in front of root) {
    DrawTree(BSPtree->behind)
    DrawPoly(BSPtree->root)
    DrawPoly(BSPtree->front)
  } else {
    DrawTree(BSPtree->front)
    DrawPoly(BSPtree->root)
    DrawTree(BSPtree->behind)
  }
}
```

Occlusion Culling

University of British Columbia

- occlusion queries
  - virtual render of bounding box
- precomputed visibility tables
  - *store a list of visible cells*
- horizon maps
  - *for terrain models*



"virtual render" of bounding box ⇒ returns the # pixels that would have appeared.

Visibility in Practice: WebGL, OpenGL

University of British Columbia

OpenGL support? ⟶ no need to add in shader

- view volume culling ⟶ polygons get culled anyhow, but no bbox checks, etc.
- view volume clipping
- backface culling
- z-buffer occlusion test
- painter's ~~algorithm~~ & BSP trees — need to do this yourself, if desired
- occlusion culling ⟶ virtual render of a bbox to see if pixels would have appeared on screen. (sometimes supported)

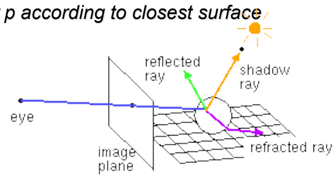
Raycasting and Raytracing

University of British Columbia

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*

(skip for now)



Transforming Normals

University of British Columbia

rotations & scales

translations

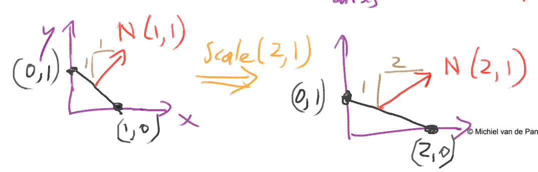
Using h=0

$$\begin{bmatrix}N_x'\\N_y'\\N_z'\\0\end{bmatrix} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ 0&0&0&1\end{bmatrix}\begin{bmatrix}N_x\\N_y\\N_z\\0\end{bmatrix}$$

this often works

h=1 for points

Problem non-uniform scales

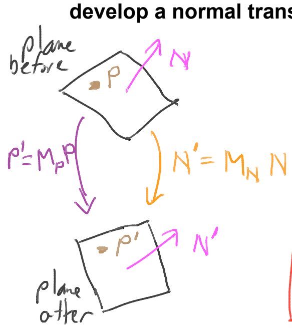


$M_N = (M_p^{-1})^T$ ← matrix to transform normals.

Transforming Normals

University of British Columbia

develop a normal transformation matrix:



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

$$\underbrace{[A \quad B \quad C \quad D]}_{N^T} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0 \quad \leftarrow p$$

$N^T p = 0$ before

$N'^T p' = 0$ after

$$(M_N N)^T M_p p = 0$$

$$N^T \underline{M_N^T M_p} \, p = 0$$

deduce $M_N^T M_p = I$

$M_N = (M_p^{-1})^T$