Visibility

Determining which objects / triangles / pixels can be seen

- Projective rendering
  1. view volume culling
  2. view volume clipping
  3. backface culling
  4. occlusion: z-buffer test
  5. occlusion: object culling

- Raytracing
  For each pixel in image, compute a ray and see what it hits in the scene.
(1) View Volume Culling (for triangles)

Idea: If vertices are outside view volume then cull.

Revised Idea:
Cull iff all vertices are outside wrt the same view volume plane.

(1) View Volume Culling (for objects)

Idea: Fast test to cull entire object.

Bounding sphere:
Cull if \( \text{dist}(C, \text{plane}) < -r \)

Bounding box:
Cull if all 8 vertices are "outside" wrt any one of the view volume planes.
(2) View Volume Clipping

general polygon clipping:

Clip against each of the 6 planes in fun.

vertex list

Clip near

Clip front

clip polygon

Clip against each of the 6 planes in fun.

Vertex list

Clip near

clip polygon

tor triangles with bounding-box scan conversion:

vertex list

Clip near

Clip front

Clip polygon

Clipping in VCS

Plane equations

Othographic View Volume

left: \( x - \text{left} = 0 \)
right: \( -x + \text{right} = 0 \)
bottom: \( y - \text{bottom} = 0 \)
top: \( -y + \text{top} = 0 \)
front: \( -z - \text{near} = 0 \)
back: \( z + \text{far} = 0 \)

Perspective View Volume

left: \( x + \text{left} \cdot z / \text{near} = 0 \)
right: \( -x - \text{right} \cdot z / \text{near} = 0 \)
top: \( -y - \text{top} \cdot z / \text{near} = 0 \)
bottom: \( y + \text{bottom} \cdot z / \text{near} = 0 \)
front: \( -z - \text{near} = 0 \)
back: \( z + \text{far} = 0 \)
Clipping in NDCS (?)

Fails for primitives that span from $+z$ to $-z$

Clipping in CCS

NDCS: $-1 \leq x_{NDCS} \leq 1$

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

canonical plane equations:

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

Would be convenient because the plane equations remain fixed

No!

This is where the $P_2$ line segment connectivity gets lost

P_{CCS} \rightarrow M_{proj} \rightarrow P_{NDCS}

$P_{VCS}$ $P_{CCS}$ $P_{NDCS}$
Line-Plane intersection

Plane equation
\[ \vec{N} = (\vec{P}_2 - \vec{P}_0) \times (\vec{P}_1 - \vec{P}_0) \]
\[ A x + B y + C z + D = 0 \]
\[ \langle A, B, C \rangle \cdot \langle x, y, z \rangle + D = 0 \]
\[ \vec{N} \cdot \vec{P} + D = 0 = F(\vec{P}) \]

where \( D = -\vec{N} \cdot \vec{P}_0 \)
\[ \vec{N} \cdot (\vec{P}_0 + t(\vec{P}_b - \vec{P}_0) + D = 0 \]
\[ \vec{N} \cdot \vec{P}_0 + t(n \cdot (\vec{P}_b - \vec{N} \cdot \vec{P}_0)) + D = 0 \]

(3) Backface Culling in VCS

Idea: cull if \( N_z < 0 \)

Correct VCS culling rule:
Cull if \( \vec{P}_v \text{eye}(0,0,0) \) is below the plane of the polygon.
where “below” = side of the Normal
\[ \vec{N} \cdot \vec{P} + D = 0 = F(x,y,z) \quad \text{Cull if } F(0,0,0) < 0 \quad \text{i.e. cull if } D < 0 \]
(3) Backface Culling in NDCS

Cull if $N_z$ in NDCS is $> 0$

Transforming Normals

Using $h=0$

Problem (occurs for non-uniform scaling)

Scale $(x, y)$

Wrong answer $N(2, 1)$

Defined answer $N'$

Skip the translate

Origin
Transforming Normals

consider a plane, before and after transformation:

Before:

\[ p' = Mp \]

\[ \vec{N} \]

\[ \vec{N}' = Q \cdot \vec{N} \]

After:

\[ p' = Mp \]

\[ \vec{N}' \]

\[ M : \text{transforms points in the usual way} \]

\[ Q : \text{transforms normals — what should it be?} \]

Let’s write a plane equation for \( \mathbf{A} \)

\[ Ax + By + Cz + D = 0 \]

or \( \begin{bmatrix} A & B & C \end{bmatrix} \begin{bmatrix} x \vline y \vline z \end{bmatrix}^T = 0 \]

Let’s write this as \( N^T \cdot p = 0 \)

Similarly, for \( \mathbf{B} \), we can write \( N^T \cdot p' = 0 \)

Now substitute for \( N' \):

\[ p' = (QN)^T \cdot (MP) \]

\[ \Rightarrow N^T (QN)^T MP = 0 \]

\[ Q^T M = I \Rightarrow Q = (M^{-1})^T \]

(4) Occlusion: Z-buffer

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

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 \(Z_{\text{pixel}} < \text{Depth}[i, j]\) { // closer?
            Image\([i, j]\) = C_{\text{pixel}} // overwrite pixel
            Depth\([i, j]\) = Z_{\text{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

- resolution issues in \(z\): “z-fighting”
- results can still depend on rendering order
  for cases of identical depths for a fragment
(5) Occlusion: Object 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
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

![Raytracing Diagram](image-url)