Visibility

Determining which objects / triangles / pixels can be seen
Visibility

Methods

- view volume culling
- view volume clipping
- backface culling
- occlusion: z-buffer test
- occlusion: object culling
- raycasting (and raytracing)

OpenGL/Unreal/DirectX support for triangles

Triangles or objects

Pixel level
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 $\text{dist}(C, \text{plane}) > r$

Bounding box:
- Cull if all 8 vertices are "outside" with respect to one of the frustum planes
2D Clipping

Sutherland Hodgeman algorithm

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

<table>
<thead>
<tr>
<th>case</th>
<th>first point</th>
<th>second point</th>
<th>output point(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>inside</td>
<td>inside</td>
<td>second point</td>
</tr>
<tr>
<td>2</td>
<td>inside</td>
<td>outside</td>
<td>intersection point</td>
</tr>
<tr>
<td>3</td>
<td>outside</td>
<td>outside</td>
<td>none</td>
</tr>
<tr>
<td>4</td>
<td>outside</td>
<td>inside</td>
<td>intersection point and second point</td>
</tr>
</tbody>
</table>

E.g., vertex A
general polygon clipping:

clip against each of 6 planes in turn

original vertex list → clipped vertex list.

for triangles with bounding-box scan conversion:
Clipping in VCS

Plane equations

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

\[ P_{\text{ces}} = M_{\text{proj}} P_{\text{ces}} \]

\[
\begin{bmatrix}
1 \\ 1 \\ -5/3 \\ -8/3 \\ -1
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th></th>
<th>( P_1 )</th>
<th>( P_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCS</td>
<td>( (1, 0, -2) )</td>
<td>( (0, 0, 2) )</td>
</tr>
<tr>
<td>CCS</td>
<td>( (1, 0, 2/3, 2) )</td>
<td>( (0, 0, -6, -2) )</td>
</tr>
<tr>
<td>NDCS</td>
<td>( (1/2, 0, 1/3) )</td>
<td>( (0, 0, 3) )</td>
</tr>
</tbody>
</table>
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: \(-z + h = 0\)
Line-Plane intersection

Line equation:
\[ \mathbf{p}(t) = \mathbf{p}_a + t(\mathbf{p}_b - \mathbf{p}_a) \]

Plane equation:
\[ \mathbf{N} \cdot (\mathbf{p} - \mathbf{p}_0) + D = 0 = F(\mathbf{p}) \]

\[ D = -\mathbf{N} \cdot \mathbf{p}_0 \]

\[ N \cdot [\mathbf{p}_a + t(\mathbf{p}_b - \mathbf{p}_a)] + D = 0 \]

\[ N \cdot \mathbf{p}_b - N \cdot \mathbf{p}_a = \frac{-F(\mathbf{p}_a)}{F(\mathbf{p}_b) - F(\mathbf{p}_a)} = \frac{da}{da + db} \]
Backface Culling in VCS

**Bad**

Idea: cull if $N_z < 0$

Correct VCS backface culling:

Cull if $\mathbf{N} \cdot \mathbf{P} + D < 0$, where $\mathbf{N}$ is the normal of the polygon and $\mathbf{P}$ is the position of the point.
Backface Culling in NDCS

Cull in NDCS if $N_z$ points away
\[ N_z > 0 \]
Transforming Normals

Using $h=0$

Problem (in the case of non-uniform scaling) skip the translation.
Transforming Normals

consider a plane, before and after transformation:

\[ N = \begin{bmatrix} A \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = 0 \]

\[ \begin{bmatrix} A & B & C & D \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = 0 \]

Write this as:

\[ N^T \cdot p = 0 \]

\[ N'^T \cdot p' = 0 \]

\[ (M \cdot N)^T (M \cdot p) = 0 \]

\[ N^T (M^T M) p = 0 \]
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

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\([i,j]\)) { // closer?
            Image\([i,j]\) = C_pixel // overwrite pixel
            Depth\([i,j]\) = Z_pixel // overwrite z
        }
    }
}

\[ Z_{Nocs} \in [-1, 1] \]
\[ Z_{DCS} \in [0, 1] \]
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.

Standard: call fragment shader, then test Z.

Pros: potential computational savings.
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