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)

removes triangles or objects outside of view volume

triangle or objects that cross into the RV

test a fake render of the boundary cube to see if any pixels would have been visible

three.js: A for objects
WebGL/OpenGL/DirectX/GPU: ABCCD for triangles
later.
View Volume Culling (for triangles)

Idea: Cull if all vertices are outside the view volume 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 \( \text{dist}(C, \text{plane}) > r \) for at least one of the view volume plane.

bounding box: cull if all 8 vertices are "outside" for at least one plane.
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</th>
<th>second</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>point</td>
<td>point</td>
<td>point(s)</td>
</tr>
<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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>and second point</td>
</tr>
</tbody>
</table>
View Volume Clipping

general polygon clipping:

Original 3D vertex list → (clip near) → (clip far) → (clip top) → (clip back) → (clip front) → (clip right) → (clip left) → clipped vertex list.

tor triangles with bounding-box scan conversion:

vertex list → (clip near) → (clip far) → (clip boundary box) → clipped vertex list.
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: \( x + \text{left} \times z / \text{near} = 0 \)
right: \( -x - \text{right} \times z / \text{near} = 0 \)
top: \( -y - \text{top} \times z / \text{near} = 0 \)
bottom: \( y + \text{bottom} \times z / \text{near} = 0 \)
front: \( -z - \text{near} = 0 \)
back: \( z + \text{far} = 0 \)

Note: clipping in VCS works, but the equations still depend on user-defined variables.
Clipping in NDCS (?)

NDCS

VCS

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

\(M_{maj}\)

+ Canonical plane equations problematic for regions that cross \(z=0\) plane

\((\text{don't call in NDCS})\)
Clipping in CCS

NDCS:
-1 ≤ \( \frac{x_{\text{ccs}}}{h_{\text{ccs}}} \) ≤ 1

CCS:
-\( h_{\text{ccs}} \leq x_{\text{ccs}} \leq h_{\text{ccs}} \) 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:** \( \vec{N} = (\vec{P}_2 - \vec{P}_0)(\vec{P}_1 - \vec{P}_0) \)

**Line Eqn:** \( P(t) = \vec{P}_a + t(\vec{P}_b - \vec{P}_a) \)

For \( t=0 \):

\[ \vec{P}_a \]

For \( t\in[0,1] \):

\[ \vec{P}_b \]

For \( t=1 \):

\[ \vec{P}(1) = \vec{P}_b \]

Substitute here to compute \( \vec{P}(t) \)

Substitutions:

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

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

\[ t = -\frac{N \cdot P_a - D}{N \cdot P_b - N \cdot P_a} = \frac{-F(P_a)}{F(P_b) - F(P_0)} \]
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 \( \text{Peye} \) is below the plane:

\[
\vec{N} \cdot \vec{P} + D = 0 = F(\text{P})
\]

\[
\vec{N} \cdot \text{Peye} + D = F(\text{Peye})
\]

\( D < 0 \) => cull if \( D < 0 \)
Backface Culling in NDCS

\[ \text{P}_\text{eye} (0, 0, 0) \]

\[ \text{P}_\text{eye} (0, 0, -\infty) \]

\[ \text{cull if eye below plane} \]

\[ \text{cull if } \text{N}_z > 0 \]

projectors

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-buffer

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

\[
Z_{ncs} = \frac{Z_{nocs} + 1}{2}
\]

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 \((Z_{pixel} < \text{Depth}[i, j])\) { // 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