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)
View Volume Culling (for triangles)
View Volume Culling (for objects)

bounding sphere:

bounding box:
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</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>
View Volume Clipping

general polygon clipping:

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

Plane equations

Othographic View Volume

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

Perspective View Volume

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

NDCS

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

<table>
<thead>
<tr>
<th></th>
<th>P₁</th>
<th>P₂</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:
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

\[ \vec{N} = \langle A, B, C \rangle \]
Backface Culling in VCS
Backface Culling in NDCS
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

• 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 \( i,j \) {
    \text{Depth}[i,j] = \text{MAX\_DEPTH}
    \text{Image}[i,j] = \text{BACKGROUND\_COLOUR}
}

for all polygons \( P \) {
    project vertices into screen-space, i.e., DCS
    for all pixels in \( P \) {
        if (\text{Z\_pixel} < \text{Depth}[i,j]) \{  // closer?
            \text{Image}[i,j] = \text{C\_pixel}  // overwrite pixel
            \text{Depth}[i,j] = \text{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