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
(1) View Volume Culling (for triangles)

(1) View Volume Culling (for objects)

bounding sphere:

bounding box:
(2) View Volume Clipping

general polygon clipping:

triangles with bounding-box scan conversion:

Clipping in VCS

Plane equations

<table>
<thead>
<tr>
<th>Orthographic View Volume</th>
<th>Perspective View Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>left: $x - \text{left} = 0$</td>
<td>left: $x + \text{left} \times z / \text{near} = 0$</td>
</tr>
<tr>
<td>right: $-x + \text{right} = 0$</td>
<td>right: $-x - \text{right} \times z / \text{near} = 0$</td>
</tr>
<tr>
<td>bottom: $y - \text{bottom} = 0$</td>
<td>top: $-y - \text{top} \times z / \text{near} = 0$</td>
</tr>
<tr>
<td>top: $-y + \text{top} = 0$</td>
<td>bottom: $y + \text{bottom} \times z / \text{near} = 0$</td>
</tr>
<tr>
<td>front: $-z + \text{near} = 0$</td>
<td>front: $-z - \text{near} = 0$</td>
</tr>
<tr>
<td>back: $z + \text{far} = 0$</td>
<td>back: $z + \text{far} = 0$</td>
</tr>
</tbody>
</table>
Clipping in NDCS (?)

NDCS

Canonical plane equations:

- left: \( x + h = 0 \)
- right: \( -x + h = 0 \)
- bot: \( y + h = 0 \)
- top: \( -y + h = 0 \)
- near: \( z + h = 0 \)
- far: \( -z + h = 0 \)

Clipping in CCS

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

<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 transform:

\[
P_{vcs} \xrightarrow{M_{proj}} P_{ccs} \xrightarrow{\lambda h} P_{ndcs}
\]
Line-Plane intersection

(3) Backface Culling in VCS
(3) 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:

(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

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

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
(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