Hidden Surface Removal

a.k.a Visible Surface Determination
Reminder - Pipeline

3D Model  →  Transformations

Polygon at [(2,9), (5,7), (8,9)]

Polygon at [...]
Reminder - Pipeline

Object coordinates

\[
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
\]

Camera coordinates

Clip coordinates

Normalized device coordinates

Screen coordinates

Model-view \rightarrow Projection \rightarrow Perspective division \rightarrow Viewport

\[
\begin{pmatrix}
x_p \\
y_p \\
z_p
\end{pmatrix}
\]

\[
\begin{pmatrix}
x_s \\
y_s
\end{pmatrix}
\]
Hidden Surface Removal

- **Input**: Polygons in normalized device coordinates

- **Output**: 2D image of projected polygons containing only visible portions
Hidden Surface Removal Algorithms

- Back face culling
- Painter’s algorithm (depth sort)
- Z-Buffer
- Scan-line Z-Buffer
Back Face Culling
Object Space

In a closed polyhedron back faces are not visible
Back Face Culling
Determining back faces

- In a closed polyhedron back faces are not visible
- Assume normals point *out*

face visible if: $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$

$\cos\theta \geq 0$

$v \cdot n = |v||n|\cos\theta \geq 0$
Back Face culling

When will it work?

Closed, convex

Open

Closed, non convex
Back Face culling

• Closed convex objects
  – Invisible back faces
  – All front faces visible
  → Visibility problem solved

• Open objects
  – Back faces possibly visible

• Closed non-convex objects
  – Invisible back faces
  – Front faces can be visible, invisible or partially visible
Painter’s Algorithm
Object space

• Which **color** to draw?
• Draw everything $\rightarrow$ which **order** to draw?
Painter’s Algorithm

- Sort polygons by depth values
- Paint back to front
- When will this fail?

Intersecting polygons

Cyclic occlusion
Painter’s Algorithm

• Works in special cases
  – E.g. polygons with constant z
  – Where do we have polygons with constant z?
Painter’s Algorithm
How do we fix it?

• Depth sort per **polygon** doesn’t work

• Depth sort per **pixel**
  – Image space algorithm
Z-Buffer Algorithm
Image Space

Resolve visibility at the pixel level

Store color + current z per pixel

Put new color only if depth < than current z
Z-Buffer Algorithm

The Z-Buffer
Z-Buffer Algorithm
The Algorithm

For every pixel \((x, y)\)
\[
\text{putZ}(x, y, \text{MaxZ})
\]

For each polygon \(P\)
\[
Q = \text{project}(P)
\]
for each pixel \((x, y)\) in \(Q\)
\[
z = \text{depth}(Q, x, y)
\]
if \(z < \text{getZ}(x, y)\)
\[
\text{putColor}(x, y, \text{col}(P))
\]
\[
\text{putZ}(x, y, z)
\]
end
end
end
Z-Buffer Algorithm
The Algorithm

For every pixel \((x, y)\)
\[
\text{putZ}(x, y, \text{MaxZ})
\]

For each polygon \(P\)
\[
Q = \text{project}(P)
\]
for each pixel \((x, y)\) in \(Q\)
\[
z = \text{depth}(Q, x, y)
\]
if \(z < \text{getZ}(Q, x, y)\)
\[
\text{putColor}(x, y, \text{col}(P))
\]
\[
\text{putZ}(x, y, z)
\]
end
end
end
Z-Buffer Algorithm

The Algorithm

For every pixel \((x, y)\)

\[
\text{putZ}(x, y, \text{MaxZ})
\]

For each polygon \(P\)

\[
Q = \text{project}(P)
\]

for each pixel \((x, y)\) in \(Q\)

\[
z = \text{depth}(Q, x, y)
\]

if \(z < \text{getZ}(Q, x, y)\)

\[
\text{putColor}(x, y, \text{col}(P))
\]

\[
\text{putZ}(x, y, z)
\]

end

end

end
Computing $\text{depth}(Q, x, y)$

- Have $z$ coordinate at 3 vertices
- How do we compute $z$ at pre-image of projected point?
Computing $\text{depth}(Q, x, y)$

- We know
  - 3D coordinates at vertices
  - 2D coordinates at vertices
  - 2D coordinates at $p_s$

- We need
  - 3D coordinates at $p$
Computing $\text{depth}(Q, x, y)$

- Linear transformations preserve straight lines

1. Compute $z_i$

2. Compute $z_p$
Linear Interpolation
On a line

- Input: 2 points, 2 values
- Output: value at any point $p$ on the line $L(t)$ between them

\[ L(t) = tp_1 + (1 - t)p_0 \]

\[
\begin{align*}
L(0) &= p_0 \\
L(1) &= p_1 \\
L(0.5) &= \frac{p_0 + p_1}{2}
\end{align*}
\]

$p_1, f_1, p_0, f_0, p$

\[
t = \frac{||p - p_0||}{||p_1 - p_0||}
\]

\[
f(p) = f(t) = tf_1 + (1 - t)f_0
\]
Computing \( \text{depth} (Q, x, y) \)

- Linear transformations preserve straight lines

\[ z_i = t_i z_2 + (1 - t_i) z_1 \]

\[ t_i = \frac{|p_i - p_1|}{|p_2 - p_1|} \]

\[ z_p = t z_i + (1 - t) z_3 \]

\[ t = \frac{|p_s - p_3|}{|p_i - p_3|} \]
Linear Interpolation
On a triangle

• Input: 3 points, 3 values

• Output: value at any point \( p \) in the triangle they generate

\[
p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3
\]

\[
\alpha_1 + \alpha_2 + \alpha_3 = 1
\]

\[
\alpha_i \geq 0
\]

\[
f(p) = f(\alpha_1, \alpha_2, \alpha_3) = \alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3
\]
Barycentric Coordinates

$$\alpha_i = \frac{A_i}{A_1 + A_2 + A_3}$$

The Barycentric coordinates of a point in a triangle are unique.

B.C. of all interior points are ≥ 0.

Triangle centroid = (1/3, 1/3, 1/3).
\[ f(t) = tf(0) + (1-t)f(1) \]

\[ f_4 = t_1 f_1 + (1-t_1) f_2 \]

\[ f_5 = t_2 f_1 + (1-t_2) f_3 \]

\[ f(x, y) = t_3 f_4 + (1-t_3) f_5 = t_3 (t_1 f_1 + (1-t_1) f_2) + (1-t_3) (t_2 f_1 + (1-t_2) f_3) = \alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3 \]
Z-Buffer Algorithm

- Image space algorithm
- Data structure: Array of depth values
- Implemented in hardware due to simplicity
- Depth resolution of 32 bits is common
- *Scene may be updated on the fly, adding new polygons*
Z Fighting

When Z-buffer has low precision and/or \( \alpha \) is not chosen correctly
Transparency Z-Buffer

How can we emulate transparent objects?
Transparency Buffer

• Extension to the basic Z-buffer algorithm
• Save *all* pixel values
• At the end – have list of polygons & depths (order) for each pixel
• Simulate transparency
  by weighting the different
  list elements, in order
• Do we need all pixel values?
The Graphics Pipeline

• Hardware implementation of screen Z-buffer:
  – Polygons sent through pipeline one at a time
  – Display updated to reflect each new polygon
Scan-Line Z-Buffer Algorithm

A={ }  
A={a,d}  
A={c,d,b}  
A={b}  
A={ }
Scan-Line Z-Buffer Algorithm

• In software implementations - amount of memory required for screen Z-buffer may be prohibitive

• Scan-line Z-buffer algorithm:
  – Render the image one line at a time
  – Take into account only polygons affecting this line

• Combination of polygon scan-conversion & Z-buffer algorithms
• Only Z-buffer the size of scan-line is required.
• Entire scene must be available a-priori
• Image cannot be updated incrementally
Scan-Line Z-Buffer Algorithm

**ScanLineZBuffer**\(\text{(Scene)}\)

\[
\begin{align*}
\text{Scene2D} &:= \text{Project}(\text{Scene}); \\
\text{Sort Scene2D into buckets of polygons P in increasing YMin(P) order;} \\
A &:= \text{EmptySet}; \\
\text{for } y := \text{YMin(Scene2D)} \text{ to } \text{YMax(Scene2D)} \text{ do} \\
&\quad \text{for each pixel } (x, y) \text{ in scanline } Y=y \text{ do } \text{PutZ}(x, \text{MaxZ}); \\
&\quad A := A + \{P \text{ in Scene : } \text{YMin}(P) \leq y\}; \\
&\quad A := A - \{P \text{ in A : } \text{YMax}(P)<y\}; \\
&\quad \text{for each polygon } P \text{ in A} \\
&\quad \quad \text{for each pixel } (x, y) \text{ in P’s span(s) on the scanline} \\
&\quad \quad \quad z := \text{Depth}(P, x, y); \\
&\quad \quad \quad \text{if } (z<\text{GetZ}(x)) \text{ then} \\
&\quad \quad \quad \quad \text{PutColor}(x, y, \text{Col}(P)); \\
&\quad \quad \quad \quad \text{PutZ}(x, z); \\
&\quad \quad \quad \quad \text{end;} \\
&\quad \quad \text{end;} \\
&\quad \text{end;} \\
\end{align*}
\]

A={ }  A={}  A={a,d}  A={c,d,b}  A={b}  A={ }
Improvement – Incremental Z

- Advance simultaneously in NDC and screen coordinates

\[ ax + by + cz + d = 0. \]

\[ \Delta x = x_2 - x_1, \]

\[ \Delta y = y_2 - y_1, \]

\[ \Delta z = z_2 - z_1, \]

\[ a \Delta x + b \Delta y + c \Delta z = 0. \]

\[ \Delta z = -\frac{a}{c} \Delta x. \]
References

• Interactive CG, Angel, 6-th edition, Chapter 6.11