Hidden Surface Removal
a.k.a Visible Surface Determination
Reminder - Pipeline

3D Model  →  Transformations

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

Polygon at [...]

"Pipeline"
Reminder - Pipeline

Object coordinates → Camera coordinates → Clip coordinates → Normalized device coordinates → Screen coordinates

\[
\begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  x_p \\
  y_p \\
  z_p \\
  1
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  x_s \\
  y_s
\end{pmatrix}
\]
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 \textit{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 → 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 \(P\)
\[ \text{Q} = \text{project}(P) \]
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 \textit{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
  \[ p_0, f_0; p_1, f_1 \]
  \[ p_1, f_1 = f(p_1) \]

• **Output**: value \( f(p) \) at any point \( p(t) \) on the line between them

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

\[ f(p(t)) = f(t) = tf_1 + (1-t)f_0 \]
Linear Interpolation
On a line

- **Input**: 2 points, 2 values
  \[ p_0, f_0; p_1, f_1 \]

- **Output**: value \( f(p) \) at any point \( p(t) \) on the line between them

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

\[
f(p) = f(t(p)) = tf_1 + (1 - t)f_0
\]

\[
t(p) = \frac{||p - p_0||}{||p_1 - p_0||}
\]
Computing \( \text{depth}(Q, x, y) \)

- Linear transformations preserve straight lines

\[
\begin{align*}
z_i &= t_i z_2 + (1 - t_i) z_1 \\
t_i &= \frac{|p_i - p_1|}{|p_2 - p_1|} \\
z_p &= t z_i + (1 - t) z_3 \\
t &= \frac{|p_s - p_3|}{|p_i - p_3|}
\end{align*}
\]
Linear Interpolation  
On a triangle

- **Input:** 3 points, 3 values
  \[ p_1, f_1; p_2, f_2; p_3, f_3 \]

- **Output:** value \( f(p) \) at any point \( p \)
in the triangle they generate

\[
p = p(\alpha_1, \alpha_2, \alpha_3) = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 \quad \text{with} \quad \alpha_1 + \alpha_2 + \alpha_3 = 1 \quad \alpha_i \geq 0
\]

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

\[\alpha_i(p) =?\]
Barycentric Coordinates

• Idea:
  – Use linear interpolation per line
    • Compute \( f_4, f_5 \)
    • Then \( f(p) \)
  – Unique?
Barycentric Coordinates

- Properties:
  - Unique!
  - Non-negative for interior points
  - Sum to 1
  - Centroid = \((1/3, 1/3, 1/3)\)

\[ \alpha_i = \frac{A_i}{A_1 + A_2 + A_3} \]
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

\begin{algorithm}
\textbf{ScanLineZBuffer} (Scene)
\begin{algorithmic}
\State \textbf{Scene2D} := \textbf{Project} (Scene);
\State Sort \textbf{Scene2D} into buckets of polygons \textbf{P} in increasing \textbf{YMin(P)} order;
\State \textbf{A} := \textbf{EmptySet};
\For{$y := \textbf{YMin(Scene2D)}$ to \textbf{YMax(Scene2D)} do}
\For{each pixel \textbf{(x, y)} in scanline \textbf{Y}=y do}
\State \textbf{PutZ(x, MaxZ)};
\State \textbf{A} := \textbf{A} + \{\textbf{P} \in \textbf{Scene} : \textbf{YMin(P)}<=y\};
\State \textbf{A} := \textbf{A} - \{\textbf{P} \in \textbf{A} : \textbf{YMax(P)}<y\};
\For{each polygon \textbf{P} in \textbf{A}}
\For{each pixel \textbf{(x, y)} in \textbf{P}'s span(s) on the scanline}
\State \textbf{z} := \textbf{Depth(P, x, y)};
\If{\textbf{z}<\textbf{GetZ(x)}}
\State \textbf{PutColor(x, y, Col(P))};
\State \textbf{PutZ(x, z)};
\EndIf
\EndFor
\EndFor
\EndFor
\EndFor
\end{algorithmic}
\end{algorithm}
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