Hidden Surface Removal
Hidden Surface Removal

- A whole variety of algorithms. We cover a few:
  - Hidden Line Removal:
    - Back face culling
    - Quantitative Visibility (Appel)
  - Hidden Surface Removal
    - Depth Sort/Painter/BSP
    - Z-Buffer
  - Advanced methods
    - Freeform Hidden Surface/Line Removal
    - Ray Tracing

- Algorithm types
  - Object space – operates in world/object space
  - Image space – operates in screen space
Hidden Surface Removal for Polygonal Scenes

- Input: Set of polygons in three-dimensional space + a viewpoint

- Output: A two-dimensional image of projected polygons, containing only visible portions
Back Face Culling (object space)

- In closed polyhedron you don’t see object “back” faces
- Assumption
  - Normals of faces point out from the object (could clearly always point in)
- Object space algorithm
Back Face Culling

- Determine back & front faces using sign of inner product $<n, V>$

\[
\langle n, V \rangle = n_x v_x + n_y v_y + n_z v_z = \|n\| \cdot \|V\| \cos \theta
\]

- In a convex object:
  - Invisible back faces
  - All front faces entirely visible $\Rightarrow$ solves hidden surfaces problem

- In non-convex object:
  - Invisible back faces
  - Front faces can be visible, invisible, or partially visible
Back Face Culling
Back Face Culling
Quantitative Visibility

First general hidden line algorithm, by Appel 1967

**Definition:** Every edge has a non-negative *Quantitative visibility* value $Q_v$, which corresponds to the number of times the edge is obscured. If $Q_v = 0$ the edge is visible.
Quantitative Visibility

- **Definition:** An active edge is a boundary edge of an open object and/or silhouette edge.
- **Question:** What is a boundary/silhouette edge?
- **Question:** How can we compute boundary/silhouette edges in $O(n)$, $n$ number of triangles?

- **Definition:** A passive edge is an interior edge.
- **Question:** What is the motivation of the distinction between active and passive edges?
Quantitative Visibility

Consider projected edges in the projection/image plane:

- **Observation**: The visibility of an edge can change only where it intersects another active edge in the projection plane.

- If an edge does not intersect any active edge, its *visibility* is **homogeneous**.
**HiddenLine** (Objects)

For all objects
- compute set of all edges \( E \);
- compute set of active edges, \( A \);

For every edge \( e \) in \( E \) do
  - \( \{e_i\} := e \) subdivided at all locations
    - \( e \) intersects an edge in \( A \);
  - \( E := (E - \{e\}) \cup \{e_i\} \);
end;

For every edge \( e \) in \( E \) do
  - Compute \( Q_v \) of \( e \);
  - If \( (Q_v = 0) \) then
    - output \( e \);
  end;
end;
Comments:

- Finding all the intersection of \( n \) segments in the plane is trivially an \( O(n^2) \) problem.
- Could be improved to \( O(n \log n) \) using plane sweep.
- Can the number of intersections \( k \) be larger than \( n \log n \)?
Quantitative Visibility

More Comments:

- The computation of the $Q_v$ of each subdivided edge can be conducted in several ways:
  - Selection of a single point (what is a good point selection!?) on the edge and testing how many polygons obscure it.
  - Exploiting coherence from the edge’s neighbors, any time it intersects an active edge.
  - What is the change of the $Q_v$ when crossing a boundary? Crossing a silhouette?

- A vertex can share edges with different $Q_v$.
- This is an object space algorithm
Question: Given a set of polygons, is it possible to:
- sort them (by depth)
- then paint them back to front (over each other) to remove the hidden surfaces?

Answer: In general, no

Works for special cases
- E.g. polygons with constant z
  (where do we have polygons with constant z!?)
Depth Sort (object space)

- While simple sorting can approximate the hidden surface removal process it will fail for:
  - Intersecting polygons
  - Mutually occluding polygons

- We need to find ways to resolve these cases:
Depth Sort by Splitting

- Given two polygons, $P$ and $Q$, we can order them in $z$ if:
  1. $P$ and $Q$ do not overlap in their $x$ extents
  2. Or $P$ and $Q$ do not overlap in their $y$ extents
  3. Or $P$ is totally on one side of $Q$’s plane
  4. Or $Q$ is totally on one side of $P$’s plane
  5. Or $P$ and $Q$ do not intersect in projection plane

- Can we always resolve the relation between $P$ and $Q$ using steps 1-5?
Depth Sort by Splitting

- What can be done if steps 1-5 between $P$ and $Q$ all failed?

- Split $P$ ($Q$) along:
  - the intersection with $Q$ ($P$) into two smaller polygons – see below (how could one compute this intersection!?)
  - the intersection of $P$ ($Q$) with the plane containing $Q$ ($P$).

- Object Space Algorithm

$P < Q < R$
BSP Trees

- Different use of tests 3 & 4 in Depth Sort method
- Define:
  - $S_p$ – set of polygons
  - $P \in S_p$
  - $N_p$ - normal to $P$
  - $P$ is in plane $L_p$
- Subdivide $S_p$ into 3 groups:
  - Polygons in front of $L_p$ ($N_p$ direction)
  - Polygons behind $L_p$
  - Polygons intersecting $L_p$
- Split polygons in third class along $L_p$ into pieces and insert into the first 2 groups
B S P Trees

- After subdivision
  - Polygons behind $L_p$ can’t obscure $P \Rightarrow$ draw first
  - $P$ can’t obscure polygons in front of $L_p \Rightarrow$ draw $P$
  - Draw polygons in front of $L_p$

- Recursively subdivide and draw front & back sets

- BSP – Binary Space/Spatial Partition

Hidden Surfaces - Center for Graphics and Geometric Computing, Technion
B S P Trees

- Convention: Right sibling in $N_p$ direction
- BSP Tree is **view independent**
- Constructed using only object geometry
- Can be used in hidden surface removal from multiple views
- How can one choose what is visible for a given view?
B S P Trees

- Given view direction $V$, perform recursive tree traversal
  - Visit back side tree (from this view)
  - Draw current node’s polygon
  - Visit front side tree

- To decide which side is back/front for given view check sign of $<V, N_p>$

- Object Space Algorithm
Z-Buffer Algorithm (image space)

- Basic Idea: resolve the visibility at the pixel level, using depth sort.

- For each image-pixel save both the color and the current depth $z$

- Instead of always painting the pixels while scan-converting a polygon, do so only if polygon’s depth is less than current $z$ depth at that pixel

- New color will replace current one only if closer in $z$

- Can the Z-buffer handle mutually-occluding/intersecting polygons?
Example
Example
Example
Example
Z-Buffer

Questions:
How can one compute Project(P) and Depth(Q,x,y)?

ZBuffer(Scene)
For every pixel (x,y) do PutZ(x,y,MaxZ);
For each polygon P in Scene do
  Q := Project(P);
  For each pixel (x,y) in Q do
    z := Depth(Q,x,y);
    if (z < GetZ(x,y)) then
      PutZ(x,y,z);
    PutColor(x,y,Col(P));
    end;
  end;
end;
Z-Buffer – The Depth map

A simple three dimensional scene

Z-buffer representation
Z-Buffer - Project(P)

- Use of regular perspective lose depth
  - Need to store separately

- Alternative: perspective warp

\[
(x, y, z, 1) \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \frac{d}{d-\alpha} & 1 \\
0 & 0 & \frac{-\alpha d}{d-\alpha} & 0
\end{bmatrix} = \left(x, y, \frac{(z-\alpha)d}{d-\alpha}, \frac{z}{d}\right)
\]

\[
(x_p, y_p, z_p) = \left(\frac{x}{z/d}, \frac{y}{z/d}, \frac{d^2}{d-\alpha}\left(1 - \frac{\alpha}{z}\right)\right)
\]

- \(z_p\) monotone with respect to \(z\) – use as depth to set order
Z-Buffer – Depth(Q, x, y)

\[ z_4 = \alpha_1 z_1 + (1 - \alpha_1) z_2 \]

\[ z_5 = \alpha_2 z_1 + (1 - \alpha_2) z_3 \]

scanline \( Y = y \)

\[
\text{Depth}(Q, x, y) = \alpha_3 z_4 + (1 - \alpha_3) z_5
\]
Z-Buffer Algorithm Properties

- Image space algorithm
- Data structure: Array of depth values
- Common in hardware (e.g. NVIDIA/ATI) due to simplicity
- Depth resolution of 32 bits is common
- Scene may be updated on the fly, adding new polygons
Z-Buffer hardware example
Hardware implementation of screen Z-buffer:
- Polygons sent through pipeline one at a time
- Display updated to reflect each new polygon
Z Fighting

For (almost) coplanar polygons.
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 weighing the different list elements, in order
CSG Object Z-Buffer

- Extension to the basic Z-buffer algorithm
- Allows the visual computation of CSG (Constructive Solid Geometry)
- How can we extend the Z-buffer to support CSG?

Hidden Surfaces - Center for Graphics and Geometric Computing, Technion
In software implementations - amount of memory required for screen Z-buffer used to 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.

Question: What is the memory size needed for full screen Z-Buffer? For scan-line Z-Buffer?

Entire scene must be available a-priori

Image cannot be updated incrementally
Scan-Line Z-Buffer Algorithm

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

ScanLineZBuffer(Scene)

Scene2D := Project(Scene);
Sort Scene2D into buckets of polygons P in increasing YMin(P) order;
A := EmptySet;
For y := YMin(Scene2D) to YMax(Scene2D) do
  For each pixel (x, y) in scanline Y=y do  PutZ(x, MaxZ);
  A := A + {P in Scene : YMin(P)<=y};
  A := A - {P in A : YMax(P)<y};
  For each polygon P in A
    For each pixel (x, y) in P’s spans on the scanline
      z := Depth(P, x, y);
      if (z<GetZ(x)) then
        PutZ(x, z);
        PutColor(x, y, Col(P));
      end;
    end;
  end;
end;
Freeform Hidden Surface Removal

- Most visibility algorithms work only on polygons
- Can approximate freeform surface as (dense) set of polygons
- Or:

\[
\text{FreeFormHiddenSrfRemove}(S)
\]

If \( S \) covers less than one pixel then

- Draw pixel with \( \text{Col}(S) \) into Z-buffer;
else

begin

- Subdivide \( S \) into \( S_1 \) and \( S_2 \);
- \text{FreeFormHiddenSrfRemove}(S_1);
- \text{FreeFormHiddenSrfRemove}(S_2);
end;
Freeform Hidden Surface Removal

- The key question is the question of **coverage**.
- Another alternative is to use iso-parametric curves $S(u_0, v)$ of surface $S(u, v)$.
- These F-16’s were rendered using iso-parametric curves.
Freeform Hidden Line Removal

- Extends Appel’s algorithm to freeform surfaces.
- Silhouette extraction, curve-curve and line-surface intersections must be developed.