Hidden Surface Removal

- A major research topic in computer graphics
- A whole variety of algorithms. We only cover a few:
  - Hidden Line Removal:
    - Back face culling
    - Quantitative Visibility (Appel)
  - Hidden Surface Removal
    - Depth Sort/Painter/BSP
    - Space Subdivision (Warnock)
    - 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

- Determine back & front faces using sign of inner product $\langle n, V \rangle$
  $$\langle n, V \rangle = n_x v_x + n_y v_y + n_z v_z = \|n\| \|V\| \cos \theta$$

- 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
Hidden Surface Removal

Quantitative Visibility

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?

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.

Quantitative Visibility

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

Depth Sort (object space)

- 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: 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

B S P Trees

- Different use of tests 3 & 4 in Depth Sort method
- Define:
  - $S_P$ – set of polygons
  - $P$  – $S_P$
  - $N_P$ - normal to $P$
  - $P$ is in plane $I_P$
- Subdivide $S_P$ into 3 groups:
  - Polygons in front of $I_P$ ($N_P$ direction)
  - Polygons behind $I_P$
  - Polygons intersecting $I_P$
- Split polygons in third class along $I_P$ into pieces and insert into the first 2 groups
**BSP 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

**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?

**Questions**

- How can one compute \( \text{Project}(P) \) and \( \text{Depth}(Q,x,y) \)?

**Z-Buffer**

```plaintext
ZBuffer(Scene)
For every pixel (x,y) do PutZ(x,y,MaxZ);
For each polygon P in Scene do
  Q := \text{Project}(P);
  For each pixel (x,y) in Q do
    z := \text{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

\[ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{d} \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ d \end{pmatrix} \]

- \( z \) is monotone with respect to \( z \) — use as depth to set order

Z-Buffer – Depth(Q, x, y)

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

Depth(Q, x, y) = \( \alpha_z z_4 + (1 - \alpha_z) z_3 \)

Z-Buffer Algorithm Properties

- Image space algorithm
- Data structure: Array of depth values
- Common in hardware (e.g. SGI/PC) due to simplicity
- Depth resolution of 32 bits is common
- Scene may be updated on the fly, adding new polygons

Z-Buffer hardware example

The Graphics Pipeline

- Hardware implementation of screen Z-buffer:
  - Polygons sent through pipeline one at a time
  - Display updated to reflect each new polygon

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

**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?

**Scan-Line Z-Buffer Algorithm**
- 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

**Freeform Hidden Surface Removal**
- Most visibility algorithms work only on polygons
- Can approximate freeform surface as (dense) set of polygons
- Or:
  ```
  FreeFormHiddenSrfRemove(S)
  If S covers less than one pixel then
  Draw pixel with Col(S) into Z-buffer; else
  begin
  Subdivide S into S1 and S2;
  FreeFormHiddenSrfRemove(S1);
  FreeFormHiddenSrfRemove(S2);
  end;
  ```
- The key question is the question of coverage.
- Another alternative is to use iso-parametric curves $S(u, v)$ 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.