Topic 8: Visibility

- Elementary visibility computations:
  - Clipping
  - Backface culling

- Algorithms for visibility determination:
  - Z-Buffering
  - Painter’s algorithm
  - BSP Trees
Visibility Problem

What is NOT visible?
Visibility Problem

What is NOT visible?

- primitives outside of the field of view
- back-facing primitives
- primitives occluded by other objects closer to the camera
Polygon Clipping (wrt to a single plane)

Input edge \( (p_k, p_{k+1}) \)
- in, in
- in, out
- out, out
- out, in

Output
- \( p_{k+1} \)
- \( p_{\text{intersect}} \)
- \( p_{\text{intersect}}, p_{k+1} \)
Polygon Clipping (wrt to a volume)

Clip with respect to each plane of the volume in sequence!

Does the order of the planes matter?

Does it work for concave polygons?

Does it work for concave volumes?
Polygon Clipping (when to clip?)

Diagram showing the transformation process from OCS to CCS through modelling, viewing, projection, viewport, and perspective transformations.
Backface culling
Backface culling
Backface culling
N.V > 0 is a back face?
Backface culling

\[ \mathbf{N} \cdot (\mathbf{P} - \mathbf{E}) > 0 \]
Backface culling (when to cull?)

Where in the graphics pipeline can we do backface culling?

canonical projection
Does backface culling always determine visibility completely for a single object?
Occluded faces

In typical scenes some polygons will overlap, we must determine which portion of each polygon is visible to eye!
Painters Algorithm

Sort primitives in $Z$.
Draw primitives back to front (CBA).
Painters Algorithm

Problems

• Large faces
• Intersecting faces
• Cycles
BSP tree
Visibility Problem

Z-Buffer
Scanline
Rasterization takes shapes like triangles and determines which pixels to fill.
Filling Polygons

First approach:

1. Polygon Scan-Conversion
   - Rasterize a polygon scan line by scan line, determining which pixels to fill on each line.
Second Approach:

2. Polygon Fill

- Select a pixel inside the polygon. Grow outward until the whole polygon is filled.
Coherence

Scan-line

Edge

Span
Polygon Scan Conversion

- Intersection Points
- Other points in the span
Polygon Scan Conversion

Process each scan line

1. Find the intersections of the scan line with all polygon edges.
2. Sort the intersections by $x$ coordinate.
3. Fill in pixels between pairs of intersections using an odd-parity rule.
   - Set parity even initially.
   - Each intersection flips the parity.
   - Draw when parity is odd.
Special Cases: Vertices

How do we count the intersecting vertex in the parity computation?

Count it zero or two times.
Special Cases: Vertices

What about:

Here it counts once.
Computing Intersections

For each scan line, we need to know if it intersects the polygon edges.

It is expensive to compute a complete line-line intersection computation for each scan line.

After computing the intersection between a scan line and an edge, we can use that information in the next scan line.
Always convex: No matter how you rotate a triangle, it only has one span per scan line.

Any polygon can be decomposed into triangles.
Visibility Problem

A-buffer
Visibility Problem

Image space algorithms
- Operate in display terms pixels, scanlines
- Visibility resolved to display resolution
- Examples: Z-buffer, ray-tracing
- $O(n \times \text{resolution})$

Object Space algorithms
- Analytically compute visible fragments
- Examples: painters algorithm, BSP
- $O(n^2)$