Implementation II

Ed Angel
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts
University of New Mexico
Objectives

• Introduce clipping algorithms for polygons
• Survey hidden-surface algorithms
Polygon Clipping

• Not as simple as line segment clipping
  - Clipping a line segment yields at most one line segment
  - Clipping a polygon can yield multiple polygons

• However, clipping a convex polygon can yield at most one other polygon
Tessellation and Convexity

• One strategy is to replace nonconvex (concave) polygons with a set of triangular polygons (a tessellation)
• Also makes fill easier
• Tessellation code in GLU library
Clipping as a Black Box

- Can consider line segment clipping as a process that takes in two vertices and produces either no vertices or the vertices of a clipped line segment.
Pipeline Clipping of Line Segments

• Clipping against each side of window is independent of other sides
  - Can use four independent clippers in a pipeline
Pipeline Clipping of Polygons

• Three dimensions: add front and back clippers
• Strategy used in SGI Geometry Engine
• Small increase in latency
Bounding Boxes

- Rather than doing clipping on a complex polygon, we can use an *axis-aligned bounding box* or *extent*
  - Smallest rectangle aligned with axes that encloses the polygon
  - Simple to compute: max and min of x and y
Bounding boxes

Can usually determine accept/reject based only on bounding box

accept

reject

requires detailed clipping
Clipping and Visibility

• Clipping has much in common with hidden-surface removal
• In both cases, we are trying to remove objects that are not visible to the camera
• Often we can use visibility or occlusion testing early in the process to eliminate as many polygons as possible before going through the entire pipeline
Hidden Surface Removal

- Object-space approach: use pairwise testing between polygons (objects)

  partially obscuring  
  can draw independently

- Worst case complexity $O(n^2)$ for $n$ polygons
Painter’s Algorithm

- Render polygons a back to front order so that polygons behind others are simply painted over.
- B behind A as seen by viewer
- Fill B then A
Depth Sort

• Requires ordering of polygons first
  - \(O(n \log n)\) calculation for ordering
  - Not every polygon is either in front or behind all other polygons

• Order polygons and deal with easy cases first, harder later

Polygons sorted by distance from COP
Easy Cases

• A lies behind all other polygons
  - Can render

• Polygons overlap in z but not in either x or y
  - Can render independently
Hard Cases

Overlap in all directions but can one is fully on one side of the other

cyclic overlap

penetration
Back-Face Removal (Culling)

- face is visible iff \( 90 \geq \theta \geq -90 \)
equivalently \( \cos \theta \geq 0 \)
or \( \mathbf{v} \cdot \mathbf{n} \geq 0 \)

- plane of face has form \( ax + by + cz + d = 0 \)
but after normalization \( \mathbf{n} = (0 0 1 0)^T \)

- need only test the sign of \( c \)

- In OpenGL we can simply enable culling
but may not work correctly if we have nonconvex objects
Image Space Approach

• Look at each projector \((nm)\) for an \(n \times m\) frame buffer and find closest of \(k\) polygons
• Complexity \(O(nmk)\)
• Ray tracing
• \(z\)-buffer
z-Buffer Algorithm

• Use a buffer called the z or depth buffer to store the depth of the closest object at each pixel found so far
• As we render each polygon, compare the depth of each pixel to depth in z buffer
• If less, place shade of pixel in color buffer and update z buffer
Efficiency

- If we work scan line by scan line as we move across a scan line, the depth changes satisfy $a \Delta x + b \Delta y + c \Delta z = 0$

Along scan line

$\Delta y = 0$

$\Delta z = - \frac{a}{c} \Delta x$

In screen space $\Delta x = 1$
Scan-Line Algorithm

• Can combine shading and hsr through scan line algorithm

scan line i: no need for depth information, can only be in no or one polygon

scan line j: need depth information only when in more than one polygon
Implementation

• Need a data structure to store
  - Flag for each polygon (inside/outside)
  - Incremental structure for scan lines that stores which edges are encountered
  - Parameters for planes
Visibility Testing

- In many realtime applications, such as games, we want to eliminate as many objects as possible within the application
  - Reduce burden on pipeline
  - Reduce traffic on bus
- Partition space with Binary Spatial Partition (BSP) Tree
Simple Example

consider 6 parallel polygons

The plane of A separates B and C from D, E and F
BSP Tree

- Can continue recursively
  - Plane of C separates B from A
  - Plane of D separates E and F
- Can put this information in a BSP tree
  - Use for visibility and occlusion testing