Polygon Clipping and Filling
Week 3, Lecture 5

David Breen, William Regli and Maxim Peysakhov
Geometric and Intelligent Computing Laboratory
Department of Computer Science
Drexel University
http://gicl.cs.drexel.edu
Outline

- Polygon clipping
  - Sutherland-Hodgman,
  - Weiler-Atherton
- Polygon filling
  - Scan filling polygons
  - Flood filling polygons
- Introduction and discussion of homework #2
Polygon

- Ordered set of vertices (points)
  - Usually counter-clockwise
- Two consecutive vertices define an edge
- Left side of edge is inside
- Right side is outside
- Last vertex implicitly connected to first
- In 3D vertices are co-planar
Polygon Clipping

- Lots of different cases
- Issues
  - Edges of polygon need to be tested against clipping rectangle
  - May need to add new edges
  - Edges discarded or divided
  - Multiple polygons can result from a single polygon
The Sutherland-Hodgman Polygon-Clipping Algorithm

- Divide and Conquer
- Idea:
  - Clip single polygon using single infinite clip edge
  - Repeat 4 times
- Note the generality:
  - 2D convex n-gons can clip arbitrary n-gons
  - 3D convex polyhedra can clip arbitrary polyhedra
Sutherland-Hodgman Algorithm

• Input:
  - \( v_1, v_2, \ldots, v_n \) the vertices defining the polygon
  - Single infinite clip edge w/ inside/outside info

• Output:
  - \( v'_1, v'_2, \ldots, v'_m \), vertices of the clipped polygon

• Do this 4 (or \( n_e \)) times

• Traverse vertices (edges)
• Add vertices one-at-a-time to output polygon
  - Use inside/outside info
  - Edge intersections
Sutherland-Hodgman Algorithm

- Can be done incrementally
- If first point inside add. If outside, don’t add
- Move around polygon from $v_1$ to $v_n$ and back to $v_1$
- Check $v_i, v_{i+1}$ wrt the clip edge
- Need $v_i, v_{i+1}$’s inside/outside status
- Add vertex one at a time. There are 4 cases:
Sutherland-Hodgman Algorithm

• foreach polygon $P$  $P' = P$
  – foreach clipping edge (there are 4) {
    • Clip polygon $P'$ to clipping edge
      – foreach edge in polygon $P'$
        » Check clipping cases (there are 4)
          » Case 1 : Output $v_{i+1}$
          » Case 2 : Output intersection point
          » Case 3 : No output
          » Case 4 : Output intersection point $v_{i+1}$

& $v_{i+1}$}
Sutherland-Hodgman Algorithm
Sutherland-Hodgman Algorithm

Animated by Max Peysakhov @ Drexel University
Final Result

Note: Edges XY and ZW!
Issues with Sutherland-Hodgman Algorithm

- Clipping a concave polygon
- Can produce two CONNECTED areas
Weiler-Atherton Algorithm

- General clipping algorithm for concave polygons with holes
- Produces multiple polygons (with holes)
- Make linked list data structure
- Traverse to make new polygon(s)
Weiler-Atherton Algorithm

• Given polygons A and B as linked list of vertices (counter-clockwise order)
• Find all edge intersections & place in list
• Insert as “intersection” nodes
• Nodes point to A & B
• Determine in/out status of vertices
Intersection Special Cases

• If “intersecting” edges are parallel, ignore
• Intersection point is a vertex
  – Vertex of A lies on a vertex or edge of B
  – Edge of A runs through a vertex of B
  – Replace vertex with an intersection node
Weiler-Atherton Algorithm: Union

- Find a vertex of A outside of B
- Traverse linked list
- At each intersection point switch to other polygon
- Do until return to starting vertex
- All visited vertices and nodes define union’ed polygon
Weiler-Atherton Algorithm: Intersection

• Start at intersection point
  – If connected to an “inside” vertex, go there
  – Else step to an intersection point
  – If neither, stop
• Traverse linked list
• At each intersection point switch to other polygon and remove intersection point from list
• Do until return to starting intersection point
• If intersection list not empty, pick another one
• All visited vertices and nodes define and’ed polygon
Boolean Special Cases

If polygons don’t intersect

– Union
  • If one inside the other, return polygon that surrounds the other
  • Else, return both polygons

– Intersection
  • If one inside the other, return polygon inside the other
  • Else, return no polygons
Point P Inside a Polygon?

- Connect P with another point P` that you know is outside polygon
- Intersect segment PP` with polygon edges
- Watch out for vertices!
- If # intersections is even (or 0) → Outside
- If odd → Inside
Edge clipping

• Re-use line clipping from HW1
  – Similar triangles method
  – Cyrus-Beck line clipping

• Yet another technique
Intersecting Two Edges (1)

- Edge 0 : \((P_0, P_1)\)
- Edge 2 : \((P_2, P_3)\)
- \(E_0 = P_0 + t_0 \cdot (P_1 - P_0)\) \(D_0 \equiv (P_1 - P_0)\)
- \(E_2 = P_2 + t_2 \cdot (P_3 - P_2)\) \(D_2 \equiv (P_3 - P_2)\)
- \(P_0 + t_0 \cdot D_0 = P_2 + t_2 \cdot D_2\)
- \(x_0 + dx_0 \cdot t_0 = x_2 + dx_2 \cdot t_2\)
- \(y_0 + dy_0 \cdot t_0 = y_2 + dy_2 \cdot t_2\)
Intersecting Two Edges (2)

• Solve for t’s
  • \( t_0 = \frac{((x_0 - x_2) \cdot dy_2 + (y_2 - y_0) \cdot dx_2)}{(dy_0 \cdot dx_2 - dx_0 \cdot dy_2)} \)
  • \( t_2 = \frac{((x_2 - x_0) \cdot dy_0 + (y_0 - y_2) \cdot dx_0)}{(dy_2 \cdot dx_0 - dx_2 \cdot dy_0)} \)

• See [http://www.vb-helper.com/howto_intersect_lines.html](http://www.vb-helper.com/howto_intersect_lines.html) for derivation

• Edges intersect if 0 ≤ \( t_0, t_2 \leq 1 \)
• Edges are parallel if denominator = 0
Filling Primitives: Rectangles, Polygons & Circles

• Two part process
  – Which pixels to fill?
  – What values to fill them with?

• Idea: **Coherence**
  – **Spatial**: pixels are the same from pixel-to-pixel and scan-line to scan-line;
  – **Span**: all pixels on a span get the same value
  – **Scan-line**: consecutive scan lines are the same
  – **Edge**: pixels are the same along edges
Scan Filling Primitives: Rectangles

- Easy algorithm
  - Fill from $x_{\text{min}}$ to $x_{\text{max}}$
  - Fill from $y_{\text{min}}$ to $y_{\text{max}}$

- Issues
  - What if two adjacent rectangles share an edge?
  - Color the boundary pixels twice?
  - Rules:
    - Color only interior pixels
    - Color left and bottom edges
Scan Filling Primitives:
Polygons

• Observe:
  – FA, DC intersections are integer
  – FE, ED intersections are not integer

• For each scan line, how to figure out which pixels are inside the polygon?
Scan Filling Polygons

- Idea #1: use midpoint algo on each edge, fill in between extrema points
- Note: many extrema pixels lie outside the polygon
- Why: midpoint algo has no sense of in/out
Scan Filling Polygons

• Idea #2: draw pixels only strictly inside
  – Find intersections of scan line with edges
  – Sort intersections by increasing $x$ coordinate
  – Fill pixels on inside based on a parity bit
    • $B_p$ initially even (off)
    • Invert at each intersect
    • Draw with odd, do not draw when even
Scan Filling Polygons

• Issues with Idea #2:
  – If at a fractional x value, how to pick which pixels are in interior?
  – Intersections at integer vertex coordinates?
  – Shared vertices?
  – Vertices that define a horizontal edge?
How to handle vertices?

- **Problem:**
  - vertices are counted twice

- **Solution:**
  - If both neighboring vertices are on the same side of the scan line, don’t count it
  - If both neighboring vertices are on different sides of a scan line, count it once
  - Compare current y value with y value of neighboring vertices
How to handle horizontal edges?

• Idea: don’t count their vertices

• Apply open and closed status to vertices to other edges
  – $y_{\text{min}}$ vertex closed
  – $y_{\text{max}}$ vertex is open

• On AB, A is at $y_{\text{min}}$ for JA; AB does not contribute, $B_p$ is odd and draw AB

• Edge BC has $y_{\text{min}}$ at B, but AB does not contribute, $B_p$ becomes even and drawing stops
How to handle horizontal edges?

- Start drawing at IJ ($B_p$ becomes odd).
- C is $y_{max}$ (open) for BC. $B_p$ doesn’t change.
- Ignore CD. D is $y_{min}$ (closed) for DE. $B_p$ becomes even. Stop drawing.
- I is $y_{max}$ (open) for IJ. No drawing.
- Ignore IH. H is $y_{min}$ (closed) for GH. $B_p$ becomes odd. Draw to FE.
- Ignore GF. No drawing.
Polygon Filling Algorithm

• For each polygon
  – For each edge, mark each scan-line that the edge crosses by examining its $y_{\text{min}}$ and $y_{\text{max}}$
    • If edge is horizontal, ignore it
    • If $y_{\text{max}}$ on scan-line, ignore it
    • If $y_{\text{min}} \leq y < y_{\text{max}}$ add edge to scan-line $y$‘s edge list
  – For each scan-line between polygon’s $y_{\text{min}}$ and $y_{\text{max}}$
    • Calculate intersections with edges on list
    • Sort intersections in $x$
    • Perform parity-bit scan-line filling
    • Check for double intersection special case
  – Clear scan-lines’ edge list
How to handle slivers?

- When the scan area does not have an “interior”
- Solution: use anti-aliasing
- But, to do so will require softening the rules about drawing only interior pixels
Scan-Filling a Polygon
Scan Filling Curved Objects

- Hard in general case
- Easier for circles and ellipses.
- Use midpoint Alg to generate boundary points.
- Fill in horizontal pixel spans
- Use symmetry
Boundary-Fill Algorithm

- Start with some internal point \((x,y)\)
- Color it
- Check neighbors for filled or border color
- Color neighbors if OK
- Continue recursively
4 Connected Boundary-Fill Alg

Void BoundaryFill4( int x, int y, int fill, int bnd)
{
    If Color(x,y) != fill and Color(x,y) != bnd
    {
        SetColor(x,y) = fill;
        BoundaryFill4(x+1, y, fill, bnd);
        BoundaryFill4(x, y +1, fill, bnd);
        BoundaryFill4(x-1, y, fill, bnd);
        BoundaryFill4(x, y -1, fill, bnd);
    }
}
Boundary-Fill Algorithm

• Issues with recursive boundary-fill algorithm:
  – May make mistakes if parts of the space already filled with the Fill color
  – Requires very big stack size

• More efficient algorithms
  – First color contiguous span along one scan line
  – Only stack beginning positions of neighboring scan lines
Course Status

So far everything straight lines!

• How to model 2D curved objects?
  – Representation
    • Circles
    • Types of 2D Curves
    • Parametric Cubic Curves
    • Bézier Curves, (non)uniform, (non)rational
    • NURBS
  – Drawing of 2D Curves
    • Line drawing algorithms for complex curves
    • DeCasteljau, Subdivision, De Boor
Homework #2

• Modify homework #1
• Add “moveto” and “lineto” commands
• They define closed polygons
• Clip polygons against window with Sutherland-Hodgman algorithm
• Display edges with HW1 line-drawing code