CS 430
Computer Graphics

Polygon Clipping and Filling
Week 3, Lecture 5
David Breen, William Regli and Maxim Peysakhov
Department of Computer Science
Drexel University

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 should be 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-Cipping 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:
  – \( v_0, v_1, \ldots, v_n \) the vertices defining the polygon
  – Single infinite clip edge w/ inside/outside info
• Output:
  – \( v'_0, v'_1, \ldots, v'_{m} \) vertices of the clipped polygon
• Do this 4 (or \( n \)) 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 v1 to vn and back to v1
- Check vi, vi+1 wrt the clip edge
- Need vi, vi+1's inside/outside status
- Add vertex one at a time. There are 4 cases:
  1. Output vi+1 to P''
  2. Output intersection point to P''
  3. No output
  4. Output intersection point & vi+1 to P''

**Sutherland-Hodgman Algorithm**

- Given 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 vi+1 to P''
        » Case 2: Output intersection point to P''
        » Case 3: No output
        » Case 4: Output intersection point & vi+1 to P''
    - P' = P''
  }

**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

Linked List Data Structure

Intersection Nodes

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

Example: Union

(V1, V2, V3, P0, V8, V4, P3, V0), (V6, P1, P2)

Example
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 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

Example: Intersection

{P1, V7, P0}, {P3, V5, P2}

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

Point P Inside a Rectangle?

• Just re-use code from Cohen-Sutherland algorithm
• If a vertex’s code equals zero, it’s inside
  • Else, it’s outside
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 : \((P_2, P_3)\)
• \(E_0 = P_0 + t_0*(P_1-P_0)\)
• \(E_2 = P_2 + t_2*(P_3-P_2)\)
• \(P_0 + t_0*D_0 = P_2 + t_2*D_2\)
• \(x_0 +dx_0 * t_0 = x_2 +dx_2 * t_2\)
• \(y_0 +dy_0 * t_0 = y_2 +dy_2 * t_2\)

Intersecting Two Edges (2)

• Solve for \(t\)’s
• \(t_0 = \frac{(x_0 - x_2) * dy_2 + (y_2 - y_0) * dx_2}{dy_0 * dx_2 - dx_0 * dy_2}\)
• \(t_2 = \frac{(x_2 - x_0) * dy_0 + (y_0 - y_2) * dx_0}{dx_2 * dy_0 - dy_2 * dx_0}\)
• See http://www.vb-helper.com/howto_intersect_lines.html for derivation
• Edges intersect if \(0 \leq t_0, t_2 \leq 1\)
• Edges are parallel if denominator = 0

Examples

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 \(x_{min}\) to \(x_{max}\)
  – Fill from \(y_{min}\) to \(y_{max}\)
• 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
  - Bp initially even (off)
  - Invert at each intersect
  - Draw when 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

Scan-Filling a Polygon
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 \( y_{\text{max}} \) (open) for BC. \( b_p \) doesn’t change.
• Ignore CD. D is \( y_{\text{max}} \) (closed) for DE. \( b_p \) becomes even. Stop drawing.
• I is \( y_{\text{max}} \) (open) for IJ. No drawing.
• Ignore IH. H is \( y_{\text{min}} \) (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 \( y_{\text{max}} \) and \( y_{\text{min}} \)
    • If edge is horizontal, ignore it
    • If \( y_{\text{max}} \) on scan-line, ignore it
    • If \( y_{\text{max}} < y < y_{\text{min}} \) add edge to scan-line \( y \)'s edge list
  – For each scan-line between polygon’s \( y_{\text{max}} \) and \( y_{\text{min}} \)
    • Calculate intersections with edges on list
    • Sort intersections in \( x \)
    • Perform parity-bit scan-line filling
    • Apply ceiling on first xsect and floor on second xsect
    • Check for double intersection special case
  – Clear scan-lines’ edge list

Example

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 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

```c
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

 Plain PBM Image files

- There is exactly one image in a file
- File begins with "magic number" "P1"
- Next line specifies pixel resolution
- Each pixel is represented by a byte containing ASCII '1' (black) or '0' (white)
- All fields/values separated by whitespace characters
- No line longer than 70 characters

 Plain PBM Image Example

```
P1
WIDTH 7
HEIGHT 5
DEPTH 1

0 1 1 1 1 1 1
1 0 0 0 0 1 1
0 1 1 1 1 1 1
1 0 0 0 0 1 1
0 1 1 1 1 1 1
```

Plain PBM Image File

- There is exactly one image in a file
- File begins with "magic number" "P1"
- Next line specifies pixel resolution
- Each pixel is represented by a byte containing ASCII '1' (black) or '0' (white)
- All fields/values separated by whitespace characters
- No line longer than 70 characters?
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
    • DeCasteljeau, Subdivision, De Boor

Homework #2

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

Programming assignment 3

• Input PostScript-like file.
• Output B/W PBM.
• Implement viewports.
• Use HW2 for polygon clipping.
• Implement scanline polygon filling. (You can not use flood filling algorithms)