Outline

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

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 bounding the polygon
  - A single infinite clip edge
- Output:
  - \( v'_1, v'_2, \ldots, v'_n \), vertices of the clipped polygon
- Do this 4 (or \( n \)) times
Sutherland-Hodgman Algorithm

• Clipping of the concave polygon
• Can produce two CONNECTED areas

Issues with Sutherland-Hodgman Algorithm

Weiler-Atherton Algorithm

• General clipping algorithm for concave polygons with holes
• Produces multiple polygons (with holes)

Weiler-Atherton Algorithm

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

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
• 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
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 (0 is even) => Outside
- If odd => Inside

Do two edges intersect?

Cyrus-Beck Algorithm

Line \( P(t) = P_0 + t(P_1 - P_0) \)

Point on the edge \( P_{ED} \)

\[
N \cdot \left[ (P_0 - P_{ED}) \right] = 0 \\
N \cdot \left[ (P_1 - P_{ED}) - (P_0 - P_{ED}) \right] = 0 \\
N \cdot \left[ (P_1 - P_0) \right] + N \cdot t(P_1 - P_0) = 0 \\
\]

Let \( D = (P_1 - P_0) \)

\[
t = \frac{N \cdot (P_1 - P_0)}{-N \cdot D} \\
\]

Make sure

1. \( D = 0 \), or \( P_1 = P_0 \)
2. \( N \cdot D = 0 \) lines are not parallel

Is \( 0 \leq t \leq 1 \) for both edges?

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, keeping track of 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
    • Invert at each intersect
    • Draw with odd, do not draw when even

How to handle vertices?

• Problem:
  – vertices are counted twice
• Solution:
  – If both endpoints at a vertex are on the same side of the scan line, count it twice
  – If both endpoints are on different sides of a scan line, count it once
  – Compare current y value with y value of other endpoint

How to handle horizontal edges?

• Idea: don’t count vertices
  – On AB, A is at \( y_{min} \) for JA; AB does not contribute, \( B_p \) is odd and draw AB
  – Edge BC has \( y_{min} \) at B, but AB does not contribute, \( B_p \) becomes even and drawing stops
  – At J, IJ has \( y_{min} \) but JA does not, so \( B_p \) becomes odd and span drawn to BC
  – The span that goes from IJ to C sees no change at C,
    – etc.

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 Polygon

• Issues with Idea #2:
  – If at a fractional x value, how to pick which pixels are in interior?
  – Intersections at integer pixel coordinates?
  – Shared vertices?
  – Vertices that define a horizontal edge?
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)
- Check neighbors for filled or border color
- Color neighbors.
- 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

Pattern Filling

- Via Scan Conversion
  - Copy pixel in pattern template to location in polygon
  - The entire screen is 'tiled' and the polygon is a window
- Without Scan Conversion
  - Create patterned template in a rectangular work area (off the screen)
  - Copy this template as needed
- Handle overwrites with masking

Homework #2

- Modify homework #1
- Add "moveto" and "lineto" commands
- They will define closed polygons
- Perform union and intersection operations on polygons
- Display edges with Bresenham code
Course Status

So far everything is a straight line!

• 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