Polygons vs. Line Loops
- Consider these code fragments:
  ```
  glBegin( GL_LINE_STRIP );
  glVertex2i( 10, 20 );
  glVertex2i( 50, 20 );
  glVertex2i( 50, 40 );
  glVertex2i( 30, 45 );
  glEnd();
  ``
  ```
  glBegin( GL_LINE_LOOP );
  glVertex2i( 10, 20 );
  glVertex2i( 50, 20 );
  glVertex2i( 50, 40 );
  glVertex2i( 30, 45 );
  glEnd();
  ```
- The resulting figures look the same, but aren't:
  - `GL_POLYGON` is closed; `GL_LINE_STRIP` isn't
  - `GL_LINE_LOOP` is "closed", but isn't a polygon

Filling Polygons
- In general, we fill in spans of pixels on scan lines which intersect the polygon
- Exploits spatial coherence:
  - Pixels on one scan line set to same value show span coherence
  - Also see scan-line coherence between adjacent scan lines
- Easy if it's an aligned rectangle
  ```
  for( y = miny to maxy )
  for( x = minx to maxx )
  setPixel( x, y );
  ```

Parity Algorithm
- Harder if it's not an aligned rectangle
- The algorithm we will implement is a parity algorithm
- It is based on a mathematical principle:
  - Odd-even rule (a.k.a. even-odd rule)
  - Odd parity rule
  - Inside-outside test
- Suppose we need to know if a particular point is inside or outside of a polygon
- One method
  - Construct a line segment between the point in question and a point known to be outside the polygon
  - Count how many intersections occur between the line segment and the polygon boundary
  - If there are an odd number, then the point in question is inside; an even number indicates that it is outside
### Filling Polygons

- Basic algorithm:
  - For each scan line,
  - Find polygon/scan-line intersections
  - Sort intersections by increasing x value
  - Fill pixel runs between pairs of x values

- In this example:
  - Scan line 3 intersects two edges
    - We’ll draw one span
  - Scan line 9 intersects 4 edges
    - We’ll draw two spans

### Edge Sharing

- Consider two rectangles which share an edge
- If we scan-convert each rectangle separately, we’ll draw the shared pixels twice
- A simple rule to avoid this:
  - A boundary pixel is not considered part of the primitive if the halfplane defined by that edge which contains the primitive lies below a non-vertical edge, or to the left of a vertical edge
- In effect, this means that a rectangle “owns” its left and bottom edges, but not its right and top edges

### Issues When Filling Spans

- Four issues to deal with:
  - Given an intersection with arbitrary fractional x value, how do we determine which pixel on either side of that intersection is interior?
  - How do we deal with the special case of intersections at integer pixel coordinates?
  - How do we deal with the special case in (2) for shared vertices?
  - How do we deal with the special case in (3) in which the vertices define a horizontal edge?

### Intersection at Fractional x Value

- How do we determine which pixel on either side of that intersection is interior?
- Assume we approach from the left:
  - If outside \( \rightarrow \) inside, round \( x \) up to find the interior pixel
  - If inside \( \rightarrow \) outside, round \( x \) down to find the interior pixel
- Example:
  - \( A = (2.3) \) \( B = (7.1) \) \( C = (13.5) \) \( D = (13, 11) \) \( E = (7, 7) \) \( F = (2, 9) \)
  - Point a: round up
  - Point b: round down

### Intersection at Integer Coordinates

- We can handle this the same way we handled shared edges
  - Leftmost pixel has integer x coordinate \( \rightarrow \) define it as interior
  - Rightmost pixel has integer x coordinate \( \rightarrow \) define it as exterior
- Example:
  - \( A = (2, 3) \) \( B = (7, 1) \) \( C = (13, 5) \) \( D = (13, 11) \) \( E = (7, 7) \) \( F = (2, 9) \)
  - Point A: interior
  - Point C: exterior
Shared Vertices at Integer Coordinates

• How should we count a vertex if our counting line passes through each of the vertices below?
• What are the fundamental differences between the vertices at A, J, and E?

Shared Vertices at Integer Coordinates

• Rules:
  • We count a \( y_{\text{min}} \) vertex in the parity calculation
  • We do not count a \( y_{\text{max}} \) vertex
  • Note that a \( y_{\text{max}} \) vertex will be drawn if it is also the \( y_{\text{min}} \) vertex for the adjacent edge

Shared Vertices at Integer Coordinates

• Applying these rules to our example:
  • Count a \( y_{\text{min}} \) vertex
  • Do not count a \( y_{\text{max}} \) vertex

  • Example:
    A = (2,3) B = (7,1) C = (13,5) D = (13,11) E = (7,7) F = (2,9)
  • Not counted: D F
  • Counted once: A C
  • Counted twice: B E

Shared Vertices Define Horizontal Edge

• Desired effect is that bottom edges are drawn, but not top edges
• This happens automatically if we don’t count the edge’s vertices

  • Example:
    A = (2,3) B = (7,3) C = (13,5) D = (13,11) E = (7,9) F = (2,9)
  • Edge AB is drawn
  • Edge FE is not drawn

Applying Our Rules

• Consider scan line 8 here:
  A = (2,3) B = (7,1) C = (13,5)
  D = (13,11) E = (7,7) F = (2,9)
• Scan line 8 hits no vertices
• Intersection points:
  a = (2.8) b = (4.5,8)
  c = (8.5,8) d = (13,8)
• Intersections with AF and CD are at integer coordinates
  • Not true for FE and ED
• We reach b while inside
  • Round down to (4,8)
• We reach c while outside
  • Round up to (9,8)

Horizontal Edges

• Consider this figure
• Edge AB is a bottom edge
  • A is the \( y_{\text{min}} \) for JA, so it is counted
    • Also, it is drawn, as it is an outside → inside crossing
  • Parity remains odd, and AB is drawn
• Edge BC is drawn
• Parity becomes even, and the span is ended
• Can also consider the spans at G and I this way
Implementation

• Finding intersections can be very slow, adversely affecting performance
• Can take advantage of edge coherence
  • Many edges intersecting scan line $k$ will also intersect scan line $k+1$
• Between scan lines, we know that $\Delta y$ is 1
• We can figure out the intersection for scan line $k+1$ by applying a delta from the intersection for scan line $k$
• Equation:
  $$x_{k+1} = x_k + \frac{1}{m}$$
• So, all we need to do is find a fast way to do this!

Implementation

void leftEdgeScan( int xmin, int ymin, int xmax, int ymax ) {
  int x, y, numerator, denominator, increment;
  x = xmin;
  numerator = xmax – xmin;
  denominator = ymax – ymin;
  increment = 0;
  for( y = ymin; y < ymax; ++y ) {
    drawDot( x, y );
    increment += numerator;
    if( increment > denominator ) {
      /* round up to next pixel */
      ++x;
      /* pull out the 1*denominator */
      increment -= denominator;
    }
  }
}

Comments

• Must do this for all edges and all scan lines, not just one edge
• Can generalize this to handle arbitrarily complex polygons using an edge table
• The ET contains all edges, sorted by $y_{min}$
• We move edges from ET into an active edge table as we encounter them
• Note: our text calls AET the active edge list (AEL)

Building the ET

• Our figure’s edges:
  • AB, BC, CD, DE, EF, FA
• We create a bucket list for each scan line which indicates which edge(s) intersect that scan line
• Each edge is listed only the first time it is crossed
  • E.g., FA will appear for scan line 3, but not for lines 4-9
• Bucket chains are sorted by increasing $x$ of the lower endpoint
  • What if multiple buckets have the same $x$ endpoint?

Bucket Contents

• Each bucket contains the following things:
  • $y_{min}$ of the edge
  • Current $x$ of the edge
    • Initially, $x$ of the vertex with the $y_{min}$ coordinate
  • The $x$ increment ($1/m$)
  • A link to the next bucket
• We build the ET as an array of pointers to buckets, indexed by scan line number
ET Build Example

- \( y_{\text{max}} \) of the edge
- \( x \) of the \( y_{\text{max}} \) vertex
- \( x \) increment (1/m)

Fill Algorithm

- Form the ET
- Initialize AET to empty
- Set \( y \) to the smallest \( y \) in the ET which has buckets
- Repeat until both ET and AET are empty:
  - Remove AET entries where \( y = y_{\text{max}} \)
  - Move from ET\([y]\) to AET when \( y_{\text{min}} = y \)
  - Sort AET on \( x \)
  - Fill pixels on scan line \( y \) using pairs of \( x \) coords from AET
  - Increment \( y \)
  - For each non-vertical edge in AET, update \( x \) for new \( y \)
    - How to detect non-vertical edge?
      - Look at inverse slope - if \( 1/m \neq 0 \), have non-vertical edge

Other Issues

- Can still be fooled by complex figures
  - Interior vertices can cause a hole in the polygon
  - Apply parity test to verify the fill is correct
  - Which vertices are drawn?
- Basic problem: concave polygons
- Can we eliminate them?

Detecting Concave Polygons

- Convex polygons contain only angles < 180°
- Concave polygons contain at least one angle > 180°

Two detection methods:
- Comparing cross products of vectors representing edges
  - Differing signs \( \rightarrow \) concave polygon
  - Compare vertex positions to extensions of edges
    - If have vertices on both sides of edge extension line \( \rightarrow \) concave
- Cross Products
  - Consider this polygon:
  - Edge vectors:
    - \( AB = (3,0,0) \)
    - \( BC = (3,3,0) \)
    - \( CD = (3,-3,0) \)
    - \( DE = (-9,0,0) \)
    - \( EF = (0,9,0) \)
    - \( FA = (0,-9,0) \)
  - Cross products:
    - \( AB \times BC = (0,0,9) \)
    - \( BC \times CD = (0,0,-18) \)
    - \( CD \times DE = (0,0,27) \)
    - \( DE \times EF = (0,0,81) \)
    - \( EF \times FA = (0,0,81) \)
    - \( FA \times AB = (0,0,27) \)
  - Because the \( BC \times CD \) vector has a negative \( z \), we extend the \( BC \) edge
Translation and Rotation

- Another idea
  - Uses transformations, which we’ll get to later

- Translate polygon so that vertex $B$ is at the origin, then rotate so that $BC$ coincides with the $x$ axis

- Vertex $D$ is below the axis, so we extend $BC$

Filled Polygons in OpenGL

- Can only fill convex polygons
  - Can get strange results if trying to fill concave polygon

- Can draw using standard closed-figure types
  - `GL_POLYGON`, `GL_TRIANGLES`, `GL_TRIANGLE_FAN`, etc.

- Can also draw aligned rectangles using `glRect*()`

- In 3D, can use GLU support for 3D shapes
  - `gluCylinder()`, `gluSphere()`, etc.

- GLUT also supports 3D shapes
  - `glutSolidTeapot()`, `glutSolidSphere()`, etc.

- See OpenGL/GLU/GLUT function index in text, page 856