Polygon Filling (Rasterization)

Point-in-polygon test

How do we tell if a point is inside or outside a polygon?

Odd-even rule: count the number of times a line from the point of interest to a point known to be outside crosses the edges of the polygon.

- Odd = inside
- Even = outside

Non-zero winding number rule: Consider the number of times the polygon edges wind around the point of interest (counter-clockwise direction).

- Define edge direction to be counter-clockwise.
- Define edge normal as the clockwise normal.
- Choose a "far" point, outside polygon, and cast a ray from the point of interest to that point.
- Calculate a winding number: for each edge crossed:
  - If $\text{angle}(\text{ray, normal}) < 90$ → increase counter by 1.
  - If $\text{angle}(\text{ray, normal}) > 90$ → decrease the count by 1.
- If the winding number is non-zero the point is inside the polygon.

Point-in-polygon test

Which interior corresponds to which rule?

Winding number

Odd-even rule
Scan-Line Polygon Fill

The scanline “span”:
A span is a group of adjacent pixels on a
scanline, that are considered to be interior
to the polygon.

Two scan-line spans:
from x=2 through 4
from x=9 through 13

Polygon Fill-Scan-line algorithm

For each scan line crossing the polygon:
- Find intersections with polygon edges
- Sort from left to right
- Fill interior spans.

Polygon Fill-Scan-line algorithm steps

Find the intersection of the scan line with all
edges of the polygon.
Sort the intersections by increasing x
coordinate.
Fill all pixels between pairs of intersections
that lie interior to the polygon, using the odd
parity rule:
- Parity is initially even, and each intersection
  inverse the parity bit.
- Draw when parity is odd.
- Do not draw when parity is even.

Scan-Line Polygon Fill: span extrema

MidPoint algorithm edge pixels
Problem: draws pixels outside polygon
Only Interior pixels

Polygon Fill - scanline algorithm
Special cases

Q: Given an intersection with an arbitrary
fractional x value, which pixel on either side
of intersection is interior ?

A: If we approach a fractional intersection to
the right and are inside polygon, we round
down the x coordinate to be inside polygon;
and vice versa.

Polygon Fill - scanline algorithm
Special cases

Q: How do we deal with intersection at integer
pixel coordinate (think of 2 polygons sharing
such pixel- to whom does it belong) ?

A:
Leftmost pixels of a span are considered to be
interior.
Rightmost pixels, are considered to be exterior.
**Polygon Fill - scanline algorithm**

### Special cases

**Q: How do we deal with intersection at integer pixel coordinate which is also a shared vertex?**

**A:** We will count only the Ymin vertex of an edge in the parity calculation, but not the Ymax.

**Q: How do we deal with intersection at integer pixel coordinate where the vertices also define a horizontal line?**

**A:** We ignore horizontal edges in intersection calculations.

### Scan-Line Coherence

**Scan-line coherence** means that the interior spans corresponding to two adjacent scan lines are usually very similar.

We use scan-line coherence in order to compute these spans incrementally, rather than intersecting each scan line with the polygon.

In particular, if a polygon edge intersects a scan line $y=k$ at $x_k$, the intersection with $y=k+1$ is

$$x_{k+1} = x_k + \frac{1}{m}$$

(where $m$ is the slope of the edge).

### Scan-Line Algorithm - Data structures

**Active Edge Table (AET):**

* Maintain an *Active Edge* list that contains only the list of edges crossing the current scan line. Therefore, the edges held by the AET are updated each new scanline.
* In this table each edge element holds the following information: $Y_{max}$, slope, (all taken from the ET), and the $x$ coordinate of intersection point between the edge and current scanline (this should be updated for each new scanline).
* The edges in this table are sorted by the $x$ coordinate of the intersection points (left to right).
**Scan-Line Algorithm**

**ET+AET Example (AET is given for scanlines 9 & 10):**

<table>
<thead>
<tr>
<th>Scanlines</th>
<th>AET:</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:</td>
<td><img src="image1" alt="Scan Lines" /></td>
</tr>
<tr>
<td>10:</td>
<td><img src="image2" alt="Scan Lines" /></td>
</tr>
</tbody>
</table>

Note that this AET example is general, and don’t handle special cases (such as the intersection with edge DC, that according to our rules should be taken as x=12 rather than 13).

**Scan-Line Algorithm steps**

1. **Initialize the ET.**
2. **Set \( y' \) (the current scanline) to the first non-empty entry in the ET.**
3. **Repeat until the AET and ET are empty:**
   - Move new edges from ET to AET: take all edges in entry \( y \) of ET (recall that these edges has \( y_{min} = y \)).
   - Update the x coordinate (intersection point with current scanline) of each edge in the AET.
   - Re-sort the AET list, if necessary.
   - Fill interior spans according to the edges on the AET list.
   - Remove from AET those edges with \( y_{max} = y \) (will not intersect the next scanline).
   - Increment \( y \) by 1 (move to next scanline).

**Exercise:**
Run the algorithm to fill the polygon below:

**Result:**

![Scan Lines](image3)