Area Fill Algorithms

- Boundary/Flood Fill
- Scanline Polygon Fill
- Scanline Boundary Fill
- Pattern Fill

Scanline Polygon Fill Algorithm

- Look at individual scan lines
- Compute intersection points with polygon edges
- Fill between alternate pairs of intersection points

More specifically:

- For each scanline spanning the polygon:
  - Find int. pts. with all edges scanline cuts
  - Sort intersection points by increasing x
  - Turn on all pixels between alternate pairs of intersection points
- But--
  - Look at intersection points that are polygon vertices
Vertex intersection points that are not local max or min must be preprocessed!

### Preprocessing non-max/min intersection points
- Move lower endpoint of upper edge up by one pixel
- i.e., $y \leftarrow y + 1$
- What about $x$?
  - $m = \Delta y / \Delta x$, so $\Delta x = (1/m) \times \Delta y$
  - But $\Delta y = 1$, so:
    - $x \leftarrow x + 1/m$
Active Edge
- A polygon edge intersected by the current scanline
- As polygon is scanned, edges will become active and inactive.
- Criterion for activating an edge:
  \( y_{sl} = \text{ymin of the edge} \)
  (Here \( y_{sl} = \text{y of current scanline} \))
- Criterion for deactivating an edge:
  \( y_{sl} = \text{ymax of the edge} \)

Vertical & Horizontal Coherence
- Moving from one scanline to next:
  \( y = y + 1 \)
- If edge remains active, new intersection point coordinates will be:
  \( y_{new} = y_{old} + 1 \)
  \( x_{new} = x_{old} + \frac{1}{m} \)
  \( \frac{1}{m} = \text{inverse slope of edge} \)

Scanline Polygon Fill Algorithm Input
- List of polygon vertices \((x_i, y_i)\)
Scanline Polygon Fill Algorithm Data Structures

1. Edge table:
   – For each edge: edge #, ymin, ymax, x, 1/m
2. Activation Table:
   – (y, edge number activated at y)
     • Provides edge(s) activated for each new scanline
     • Constructed by doing a "bin" or "bucket" sort
3. Active Edge List (AEL):
   – Active edge numbers sorted on x
     • A dynamic data structure

Bin Sort for Activation Table

Scanline Polygon Fill Algorithm

1. Set up edge table from vertex list; determine range of scanlines spanning polygon (miny, maxy)
2. Preprocess edges with nonlocal max/min endpoints
3. Set up activation table (bin sort)
4. For each scanline spanned by polygon:
   – Add new active edges to AEL using activation table
   – Sort active edge list on x
   – Fill between alternate pairs of points (x,y) in order of sorted active edges
   – For each edge e in active edge list:
     If (y != ymax[e]) Compute & store new x (x+=1/m)
     Else Delete edge e from the active edge list
Scanline Polygon Fill Algorithm Example

```
Scanline Poly Fill Alg. (with example Data)

<table>
<thead>
<tr>
<th>Edge Table (as Algorithm Executes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

Active Edge List (as it develops)

<table>
<thead>
<tr>
<th>y</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Edges</td>
<td>0.5</td>
<td>0.5</td>
<td>0.4</td>
<td>0.124</td>
<td>0.124</td>
<td>0.123</td>
<td>2.0</td>
<td>2.3</td>
</tr>
<tr>
<td>Fill between</td>
<td>1-1</td>
<td>1-10</td>
<td>2-10</td>
<td>2-5.5-10</td>
<td>2-2.6-9</td>
<td>7-9</td>
<td>8-9</td>
<td></td>
</tr>
</tbody>
</table>
```

Video of BALSA Scanline Poly Fill Algorithm Animator

- Brown University Algorithm Simulator and Animator
- Mark Brown and Bob Sedgewick

- Scanline Fill Algorithms can be fast if sorting is done efficiently
Demo of Scanline Polygon Fill Algorithm vs. Boundary Fill Algorithm

Adapting Scanline Polygon Fill to other primitives
- Example: a circle or an ellipse
  - Use midpoint algorithm to obtain intersection points with the next scanline
  - Draw horizontal lines between intersection points
  - Only need to traverse part of the circle or ellipse

Scanline Circle Fill Algorithm
Modify midpoint circle algorithm for each step draw 4 horizontal lines

```plaintext
Line(xh,yh,xh,yh); // 1
Line(xh,yk,xh,yk); // 2
Line(-xh,yk,yh,xk); // 3
Line(-xh,xk,yk,yh); // 4
```