Scan Conversion:
Drawing Polygons on Raster Display
The Problem’s Statement

- **Problem:**
  Given a closed two-dimensional polygon, fill its interior with specified color on graphics display

- **Assumptions:**
  - polygon is simple. I.e. no self intersections
  - polygon is simply connected

- **Solutions:**
  - Flood fill
  - Scan conversion
Flood Fill Algorithm

- Let $P$ be a polygon with $n$ vertices $v_0$ to $v_{n-1}$ ($v_n = v_0$).
- Let $C$ be the color to paint the polygon.
- Let $p = (x,y) \in P$ be a point inside $P$. 

![Diagram of a polygon with points marked]
Flood Fill

FloodFill (Polygon $P$, int $x$, int $y$, Color $C$)
if not (OnBoundary $(x, y, P)$ or Colored $(x, y, C)$)
begin
  PlotPixel $(x, y, C)$;
  FloodFill $(P, x+1, y, C)$;
  FloodFill $(P, x, y+1, C)$;
  FloodFill $(P, x, y-1, C)$;
  FloodFill $(P, x-1, y, C)$;
end ;
Flood Fill - Discussion

- What is the cost per pixel?
- What is the worst case stack size needed?
- Where is this algorithm used/useful?
- How do we get a seed point?
Scan Conversion

- Let $C$ be a simple closed curve in plane $P$.

- **Jordan theorem:** $P - C$ has exactly two connected components $W_1$ and $W_2$, for which $C$ is the common boundary.

- By intersecting the boundary of the polygon (a simple closed curve) with a set of horizontal lines pixel wide apart, one can detect and set only the pixels inside the polygon.
Basic Scan Conversion Algorithm

- Let $P$ be a polygon with $n$ vertices $v_0$ to $v_{n-1}$ ($v_n = v_0$)
- Let $C$ be the color to paint this polygon
- Each intersection of straight line with boundary moves into/out-of the polygon
- Detect (and set) pixels inside the polygon boundary
Basic Scan Conversion

ScanConvert (Polygon P, Color C)
For y := 0 to ScreenYMax do
    I ⊆ Points of intersections of edges of P with line Y = y;
    Sort I in increasing X order and
    Fill with color C alternating segments;
end;
Optimized Scan Conversion

- Maintain list of *active* edges $A$
- $A$ contains edges that intersect current scan line

**ScanConvert**($P, C$)

Sort all edges $v_i v_{i+1}$, $\forall i \in \{0, n-1\}$,
in increasing order of $Y_{\text{Min}}(v_i v_{i+1})$;
$A \leftarrow 0$;

For $y$ from 0 to $\text{ScreenYMax}$ do

- if ($v_i v_{i+1} \notin A$ and $Y_{\text{Min}}(v_i v_{i+1}) \leq y$)
  
  $A \leftarrow A \cup \{v_i v_{i+1}\}$;

- if ($v_i v_{i+1} \in A$ and $Y_{\text{Max}}(v_i v_{i+1}) < y$)
  
  $A \leftarrow A \setminus \{v_i v_{i+1}\}$;

$I \leftarrow$ Points of intersection of $A$ members with line $Y = y$;
Sort $I$ in increasing $X$ order and
fill with color $C$ alternating segments;

end
More Enhancements

Exploit coherence of adjacent horizontal lines:

- Determine the intersections of horizontal line $L_{i+1}$ from the intersections of horizontal line $L_i$

- Assume there exist no vertex $v$ between $L_i$ and $L_{i+1}$ (vertices are only on lines)

- Use midpoint or Bresenham scan-line conversion on each edge – minimize computation from $L_i$ to $L_{i+1}$
Special Cases

- **Comment:** Errors in both algorithms can have global effects or “leaks” ⇒ need very stable implementation

- **Question:** Any advantage to making the flood fill in eight instead of four directions? Any disadvantages?
## Comparison

<table>
<thead>
<tr>
<th>Flood Fill</th>
<th>Scan Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very simple</td>
<td>More complex</td>
</tr>
<tr>
<td>Discrete algorithm in screen space</td>
<td>Discrete algorithm in object and/or screen space</td>
</tr>
<tr>
<td>Requires <code>GetPixelVal</code> system call</td>
<td>Device independent</td>
</tr>
<tr>
<td>Requires a seed point</td>
<td>No seed point required</td>
</tr>
<tr>
<td>Requires very large stack</td>
<td>Requires small stack</td>
</tr>
<tr>
<td>Common in paint packages</td>
<td>Used in image rendering</td>
</tr>
<tr>
<td>Unsuitable for line-based Z-buffer</td>
<td>Suitable for line-based Z-buffer</td>
</tr>
</tbody>
</table>