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 $P$
Flood Fill

\[
\text{FloodFill}(\text{Polygon } P, \text{int } x, \text{int } y, \text{Color } C) \\
\text{if not (OnBoundary}(x, y, P) \text{ or Colored}(x, y, C)) \\
\begin{array}{l}
\text{PlotPixel}(x, y, C) \\
\text{FloodFill}(P, x+1, y, C) \\
\text{FloodFill}(P, x, y+1, C) \\
\text{FloodFill}(P, x+1, y+1, C) \\
\text{FloodFill}(P, x-1, y, C) \\
\end{array}
\text{end ;}
\]
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

```plaintext
ScanConvert(Polygon P, Color C)
For y := 0 to ScreenYMax do
    T := Points of intersections of edges of P with line Y = y;
    Sort T 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 $v_{i,n}$, $V$ in $(0,n-1)$,
in increasing order of $YMin(v_{i,n})$;
$A$ = $\emptyset$
For $y$ from $0$ to ScreenYMax do
  if $(v_{i,n} \cap A$ and $YMin(v_{i,n}) < y)$
    $A = A \cup \{v_{i,n}\}$;
  if $(v_{i,n} < A$ and $YMax(v_{i,n}) < y)$
    $A = A \setminus \{v_{i,n}\}$;
  $\Pi$ = Points of intersection of $A$ members with line $y$
  Sort $\Pi$ 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” $\Rightarrow$ 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 \textit{GetPixelVal} 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>