SCAN LINE ALGORITHM

Given the edges defining a polygon, and a color for the polygon, we need to fill all the pixels inside the polygon.

- Three different algorithms:

1. Scan-line fill
2. Boundary fill
3. Flood fill

Two basic approaches to area filling on raster systems:

1)Determine the overlap intervals for scan lines that cross the area (scan-line)

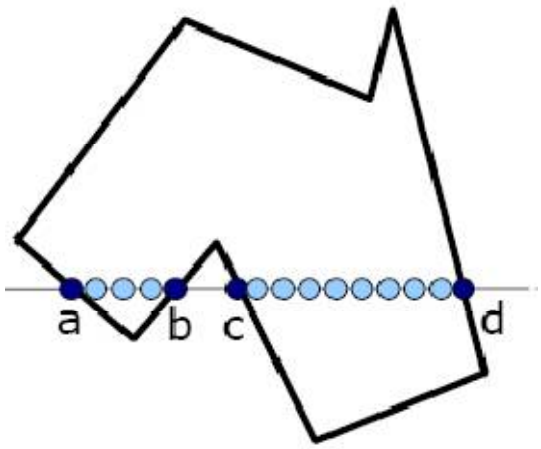
2)Start from an interior position and point outward from this point until the boundary condition reached (fill method)

- Scan-line: simple objects, polygons, circles,..

- Fill-method: complex objects, interactive fill.

Main Theme:

- A scan-line algorithm locates the intersection points of the scanline with each edge of the area to be filled.
- It proceeds from left to right, pairing all the intersections and the intervening pixels are set to be specified fill intensity or color.



- A scan line that intersects a polygon vertex may produce an even number of intersections or an odd number of intersections.
- Even number of intersections can be paired correctly , identify interior pixel but odd number of intersections do not correspond to the polygon interior.

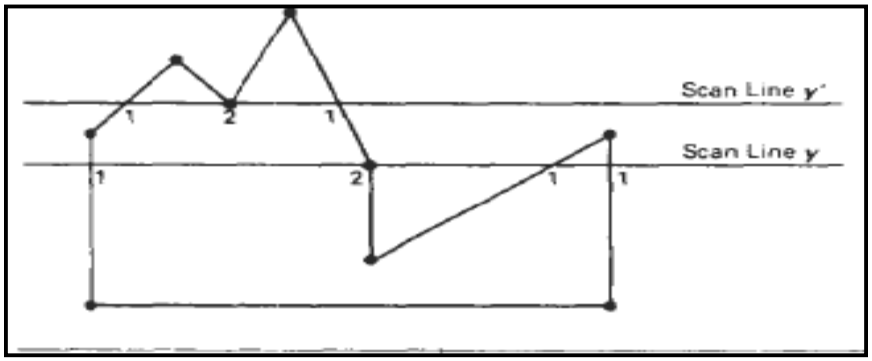
- In following eg. two scan lines at positions y and y' that intersect edge endpoints.

- Scan line y intersects five polygon edges.

- Scan line y', however, intersects an even number of edges although it also passes through a vertex.

- Intersection points along scan line y' correctly identify the interior pixel spans.

- But with scan line y, we need to do some additional processing to determine the correct interior points.



Two scan lines cutting polygon edges [Ref. Baker]

-The topological difference between scan line y and scan line y' in above figure is identified by noting

the position of the intersecting edges relative to the scan line.

-For scan line y, the two intersecting edges sharing a vertex are on opposite sides of the scan line. But for scan line y', the two intersecting edges are both above the scan line.

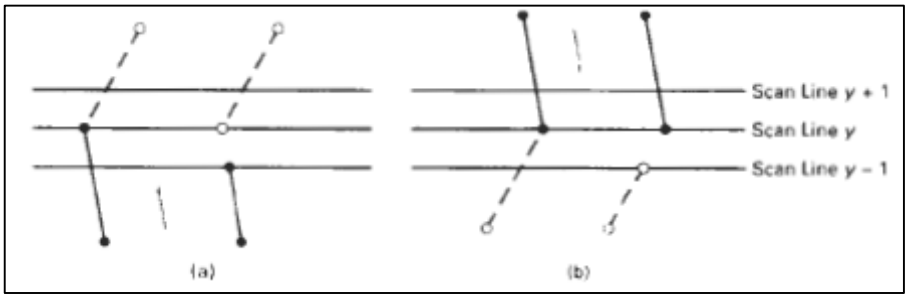
-Thus, the vertices that require additional processing are those that have connecting edges on opposite sides of the scan line. ….
* [We can identify these vertices by tracing around the polygon boundary either in clockwise or counterclockwise order and observing the relative changes in vertex y coordinates as we move from one edge to the next. If the endpoint y values of two consecutive edges monotonically increase or decrease, we need to count the middle vertex as a single intersection point for any scan line passing through that vertex. Otherwise, the shared vertex represents a local extremum (minimum or maximum) on the polygon boundary, and the two edge intersections with the scan line passing

through that vertex can be added to the intersection list.]

-Thus, If two intersecting edges sharing a vertex are on opposite side of scan line then we store only one intersection point (I.P) for that vertex.
* [One way to resolve the question as to whether we should count a vertex as one intersection or two is to shorten some polygon edges to split those vertices that should be counted as one intersection. We can process non horizontal edges around the polygon boundary in the order specified, either clockwise or counterclockwise.

As we process each edge, we can check to determine whether that edge and the next non horizontal edge have either monotonically increasing or decreasing endpoint y values. If so, the lower edge can be shortened to ensure that only one intersection point is generated for the scan line going through the common vertex joining the two edges.]

[Following Figure illustrates shortening of an edge.



Edge Shortening [Baker]

-When the end point y coordinates of the two edges are increasing, the y value of the upper endpoint for the current edge is decreased by 1, as in (a). When the end point y values are monotonically decreasing, as in (b) of above figure, we decrease they coordinate of the upper endpoint of the edge following the current edge.]

- Here it is necessary to calculate X - I. P. for scan line with every polygon side.

-Coherence property is used here to simplify these calculations. It is defined as property of scene by which we can relate one part of the scene with the other part of the scene.

-Slope of an edge is used as a coherence property as slope of the edge is constant from one scan line to next.

-By using this property, we can find out the x-intersection value on the lower scan line if the x-intersection value for current scan line is known.

$$X_{i+1} = x_i - (1/m)$$

-As we scan from top to bottom, value of y coordinates between the two scan line changes by 1. i.e. $y_{i+1} = y_i - 1$

[

Following figure shows two successive scan lines crossing a left edge of a polygon.

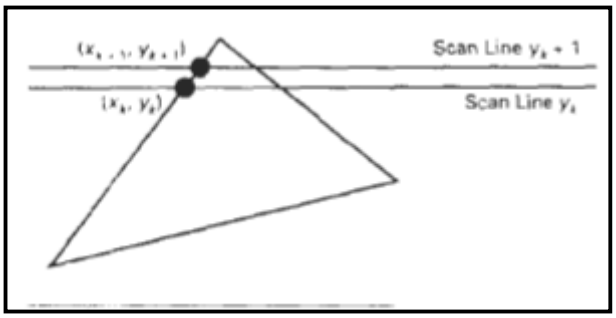


Figure 1Two Successive scan lines intersecting a polygon boundry [Baker]

The slope of this polygon boundary line can be expressed in terms of the scan-line intersection coordinates:

$m = (y_{k+1} - y_K) / (x_{k+1} - x_K)$

Since the change in y coordinates between the two scan lines is simply –

$$y_{k+1} - y_K = 1$$

The x-intersection value x_{k+1} on the upper scan line can be determined from the x-intersection value x_k on the preceding scan line as

$$x_{k+1} = x_K + (1/m)$$

]

-Many times it is not necessary to compute the x-intersections for scan line with every polygon side. It is easier to identify which polygon sides should be tested for x-intersection if we first sort the sides in order of their maximum y-value.

-Once the sides are sorted we can process the scanlines from the polygon top to its bottom producing active edge list (or simply active edges) for each scan line crossing the polygon boundaries.

-This active edge list for a scan line contains all edges crossed by that scan line.
-Each successive x intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer.
-Thus, incremental calculations of x intercepts along an edge for successive scan lines can be expressed as –

$$x_{k+1} = x_K + (dx/dy)$$

-To efficiently perform a polygon fill, we can first store the polygon boundary in a sorted edge table that contains a11 the information necessary to process the scan lines efficiently.
- Proceeding around the edges in either a clockwise or a counterclockwise order, we can use a bucket sort to store the edges, sorted on the smallest y value of each edge, in the correct scan line positions.
-Only non horizontal edges are entered into the sorted edge table.
-As the edges are processed, we can also shorten certain edges to resolve the vertex – intersection question.

-Each entry in the table for a particular scan line contains the maximum y value for that
edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.
-For each scan line, the edges are in sorted order from left to right.

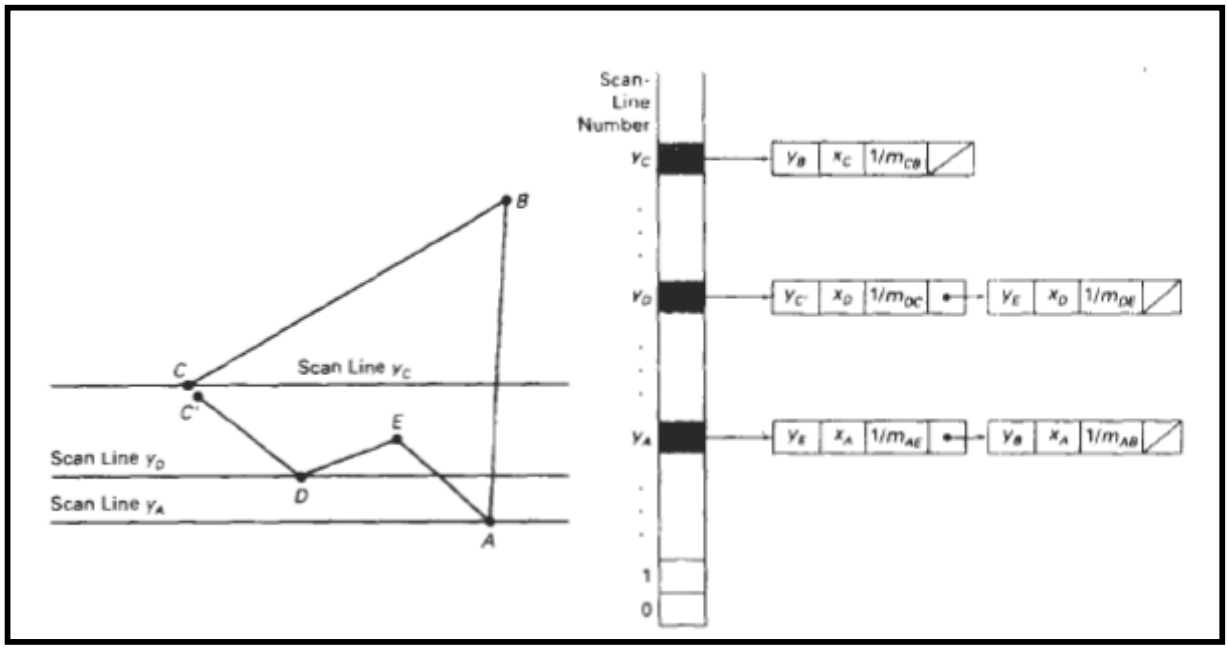


Figure 2 Polygon , Active Edge List and Shortened edge [Baker]

- Next, we process the scan lines from the bottom of the polygon to its top, producing an active edge

list for each scan line crossing the polygon boundaries.

-The active edge list for a scan line contains all edges crossed by that scan line, with iterative coherence calculations used to obtain the edge intersections.

-Implementation of edge-intersection calculations can also be facilitated by storing dx and dy values in the sorted edge table.

- For each scan line, we fill in the pixel spans for each pair of x-intercepts starting from the leftmost x-intercept value and ending at one position before the rightmost x intercept. And each polygon edge can be shortened by one unit in the y direction at the top end point.

-These measures also guarantee that pixels in adjacent polygons will not overlap each other.

- Aforesaid steps are summarized as below –

1	Sort the polygon slides on largest y-value
2	Start with the largest y-value and scan down the polygon
3	For each y, determine which sides can be intersected and find the x values of these

	intersection points
4	Sort, pair and pass the x-values to draw the S. Lines (line drawing routine)
	Algorithm for Complete Scan Line
1	Read the number of vertices of polygon *n*
2	Read x and y co-ordinates of all vertices in array x[n] and y[n].
3	Find the minimum and maximum value of y and store as y_{min} and y_{max}
4	Store the initial x-value (x1 and x2), y-values (y1 and y2) for two end points and x-increment by 1/m from scanline to line for each edge in the array edges. For each edge check that y1 > y2, if not then interchange(x1,y1) with (x2,y2) so that for each edge, y1 represents its maximum y-coordinate and y2 represents its minimum y-coordinate
5	Sort array edges in descending order of y.
6	Set y = y_{max}

7	Find the active edges and update active edge list
	If (y >= y1 and y1 < y2)
	{ edge is active}
	Else

	{ edge is not active }
8	Compute the x-intersects for all active edges for current y-value and x-intersects for successive y-values can be given as – $X_{new} = x_{old} – (1/m)$
9	Sort the x-intersections in ascending order. If x-intersect is vertex ….x-intersect = x1, and y=y1 then apply vertex test to check whether to consider one intersect or two intersects.
10	Extract pairs of intersects from the sorted x-intersection
11	Pass pairs of x-values to the line drawing routine to draw corresponding line segments.
12	Set y = y - 1
13	Repeat steps 7 to 12 until $y >= y_{min}$
14	Stop