Contents

- Scan-line fill algorithm
- Boundary-fill (drop of ink)

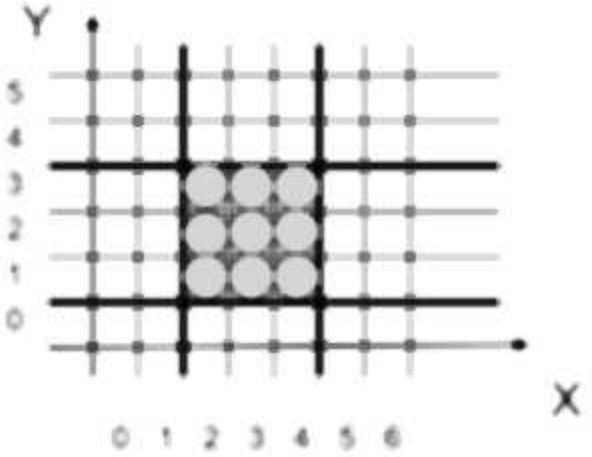
Objectives

- We are trying to fill the interior of a (polygon) area with a solid fill colour or pattern.
- There are two known approaches:
  1. Determine the overlap intervals of scan lines that cross the (polygon) area = scan line algorithm
  2. Start from a known interior point and flood the interior until you reach the boundary = boundary fill algorithm

Screen Grid vs. Pixel Coordinates

- Screen grid coordinates refer to a mathematically precise 2D cartesian coordinate system with infinitesimally small points: (x,y)
- Pixel coordinates (x_1,y_1) refer to finite areas bounded by the lines

$x - x_1 = 0,\ x - (x_2+1) = 0,\ y - y_1 = 0,\ y - (y_2+1) = 0$

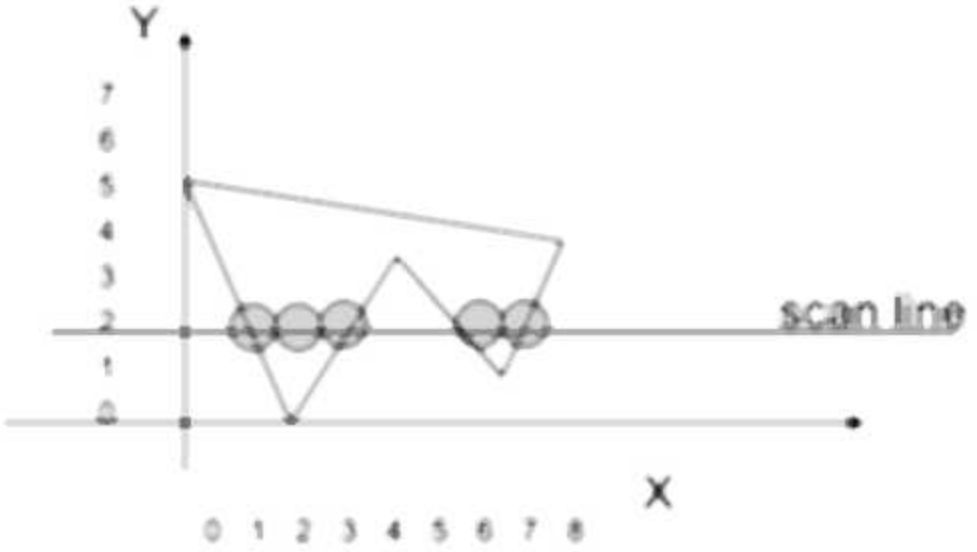


area bounded by

x=2, x=5, y=1, y=4

Scan-line algorithm

- We are trying to find the interior of a polygon area by locating the intersection points of a scan line with the edges of the polygon. Edge: set of connected finite length straight lines defining the perimeter of the polygon. Scan line: y = integer

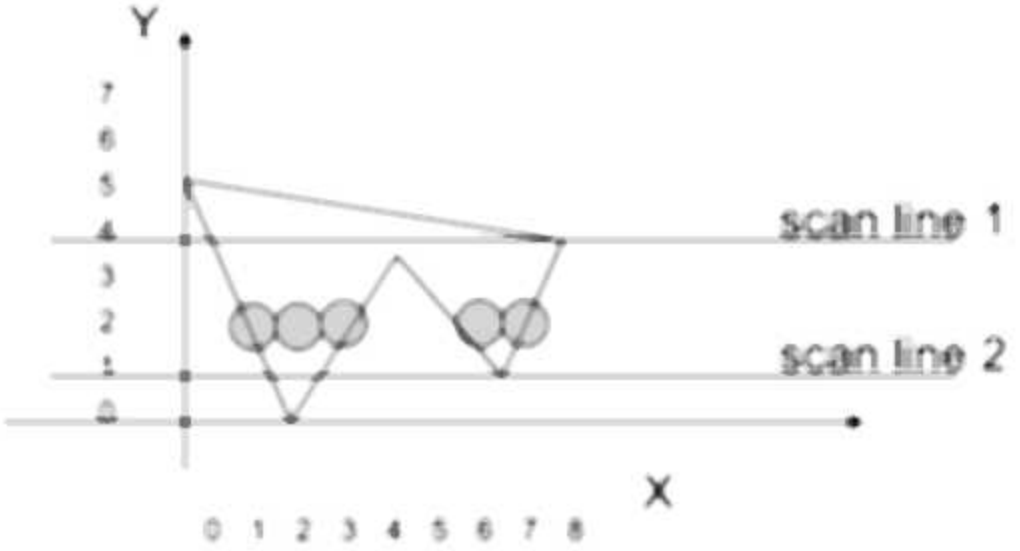


Scan-line algorithm

- This is a mathematical procedure with the basic result of a collection of intersection points with the edges of a polygon.
- Warning: We do not know which intersection points are starting points for the intervals and which are end points!
- Sort the intersection points and obtain stretches (intervals) of the scan line inside the polygon.
- Next task: draw these intervals!

Scan-line algorithm

- How many intersection points are generated?
- For scan line 1 three!!
- For scan line 2, four!

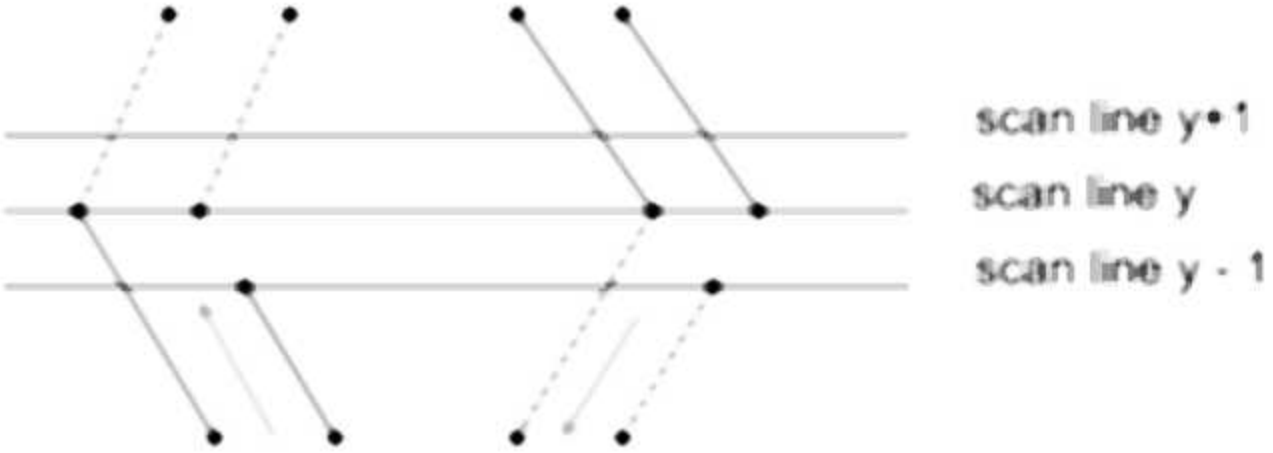


Scan-line algorithm

- Scan line 2 is not problematic, the second interval has zero length and can be dealt with by the line drawing algorithm
- Scan line 1 is "topologically" different:
  - it should generate one intersection point but it generates two
  - it passes an intersection point where the edges lie on both sides of the scan line

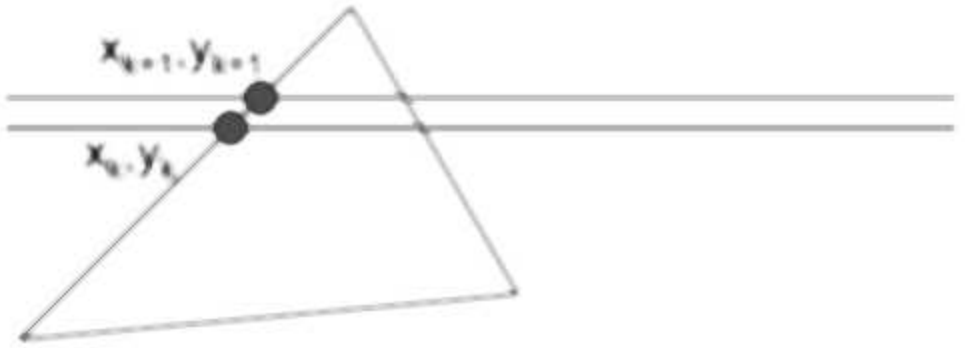
Scan-line algorithm

- The problem appears when the end point of an edge is the starting point of the next edge and y is monotonic (non-decreasing or non-increasing). This could happen very often!
- The problem is solved by shortening one of the edges:



Scan-line algorithm

- We now have a collection of edges, partially shortened, for which we can calculate the intersection points with the scan lines.
- For any given edge the successive intersection points are close to each other:



Scan-line coherence

- **Coherence**: the processing of the intersection points along an edge can be simplified from the knowledge that consecutive intersection points are simply related
- The slope of the edge m is given by two consequtive interception points as

$$m = (y_{k+1} - y_k)/(x_{k+1} - x_k)$$

Scan-line coherence

- For two consequtive interception points

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

- Hence

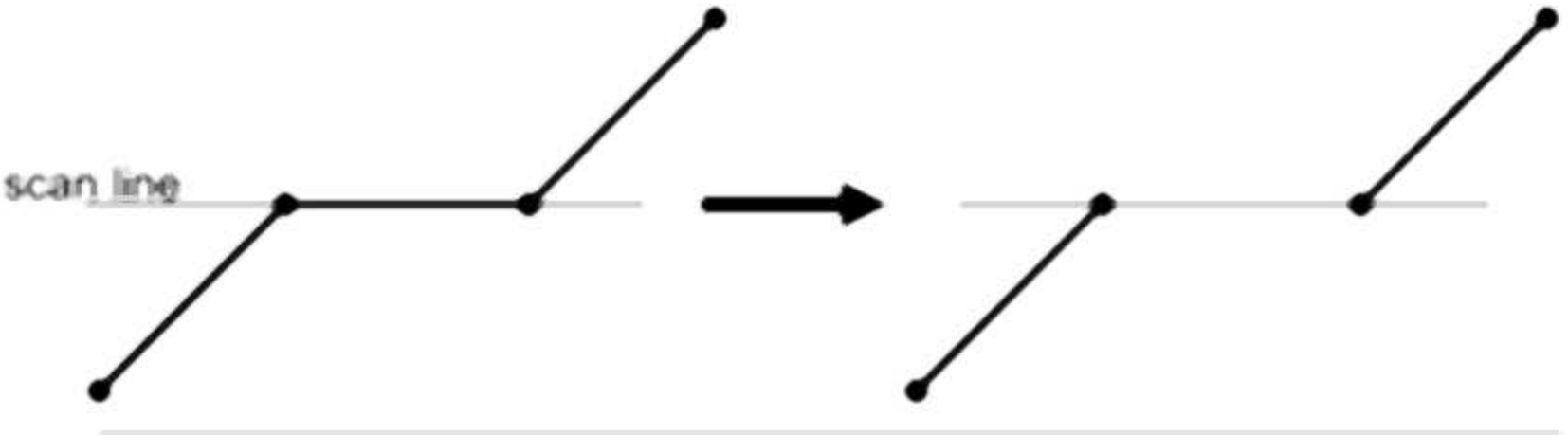
$$x_{k+1} = x_k + 1/m$$

- and for the k:th scan line above the first scan line

$$x_k = x_0 + k/m$$

Scan-line algorithm

- What happens with **horizontal edges**?
- In principle an infinite number of interception points!
- Horizontal edges can actually be treated very simply: ignore them!



Scan-line algorithm

- For an efficient polygon fill we should organize our edges such that they are immediately available
- Use a sorted edge array
  - all edges point upwards (no horizontal edges)
  - for each scan line y value, list the edges starting from that y-value
  - for each edge give y_{max}, x_{start} and 1/m
  - sort the edges according to m: -0 -> $-\infty$, $+\infty$ -> +0