Computer Graphics

Filling

Introduction

fill problem:

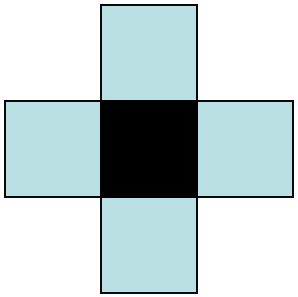
- input: (clipped) polygon given by either
  - analytic description (edges, vertices) or
  - pixel already rendered on a frame buffer
- task: fill the inside with pixels
- any ideas?

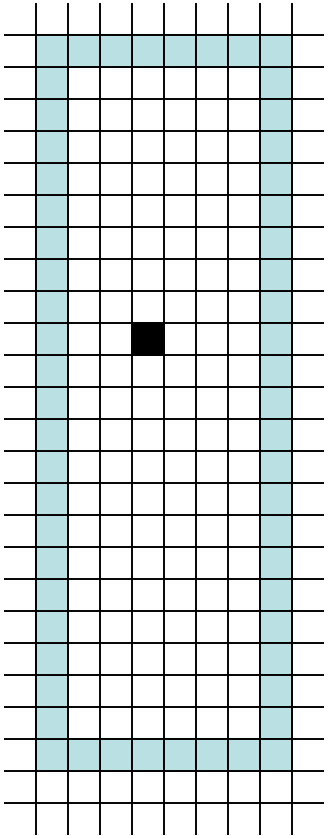
Filling

Flood-Fill

Flood Fill for Filling Pixel Outline

- given a start Pixel P_0
- starts filling recursively from P_0 using neighborhood scheme
  $\longrightarrow$ 4-neighborhood
  (top, right, bottom, left)



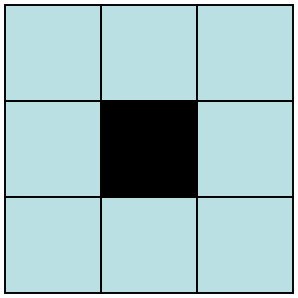


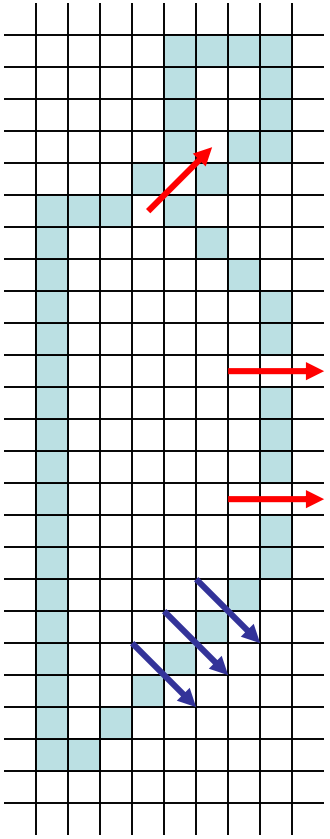
algorithm:

- check first if pixel set already
- if not set pixel and start recursion
- if pixel set abort

Problems with Flood Fill

- efficiency
  - recursion inefficient, O(n) deep
  - each pixel touched four times!
  - needs first pixel
- variety of possible polygons
  - problems with narrow tunnels
  - problems with holes
- partial solution: 8-neighborhood
  - new problems with diagonal lines
- better algorithm?



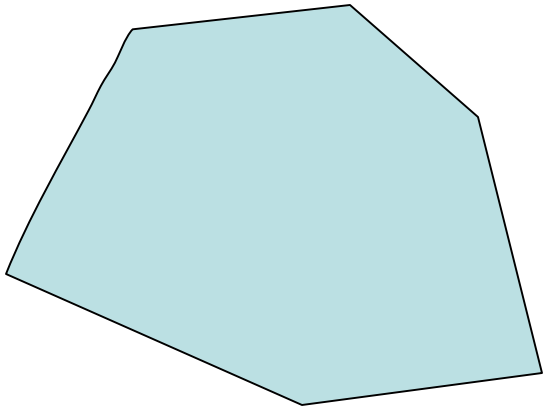


Filling

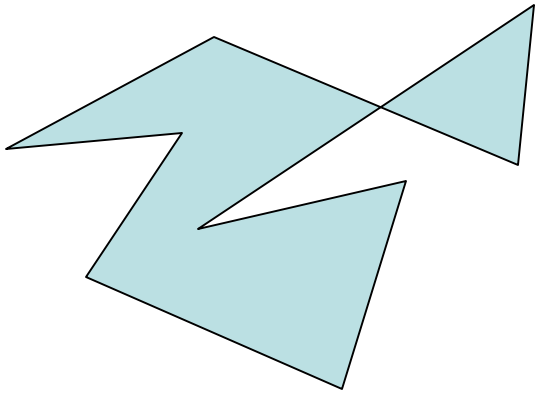
Edge Table Filling

Filling Using Analytic Descriptions

- input: polygon defined by:
  - lists of vertices {V} and edges {E}
  - edge from v_i to v_{i+1} with $1 \leq i < n$
  - edge from v_n to v_1 closes polygon



convex polygons
(if P_1 and $P_2 \in$ polygon, then all P_i between P_1 and $P_2 \in$ polygon)



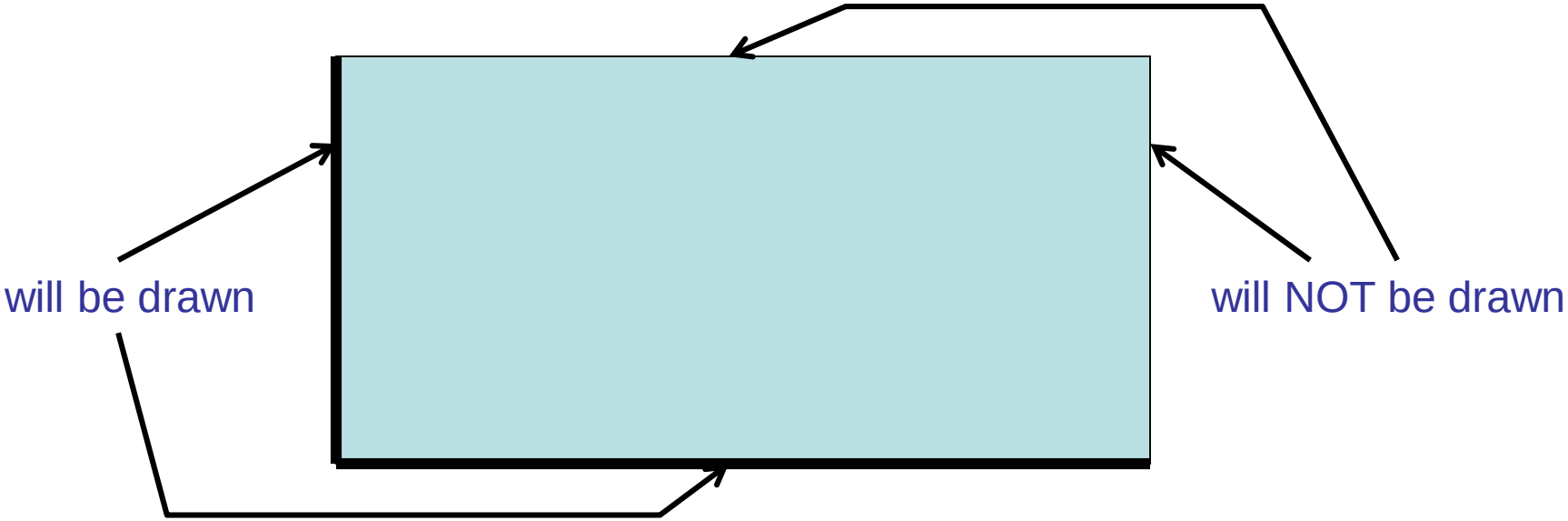
concave polygons
possibly with self-intersections

Definition

- pixels exactly on polygon edges:
  - are part of the polygon if they are below or left of the polygon
  - are not part of the polygon if they are above or right of the polygon

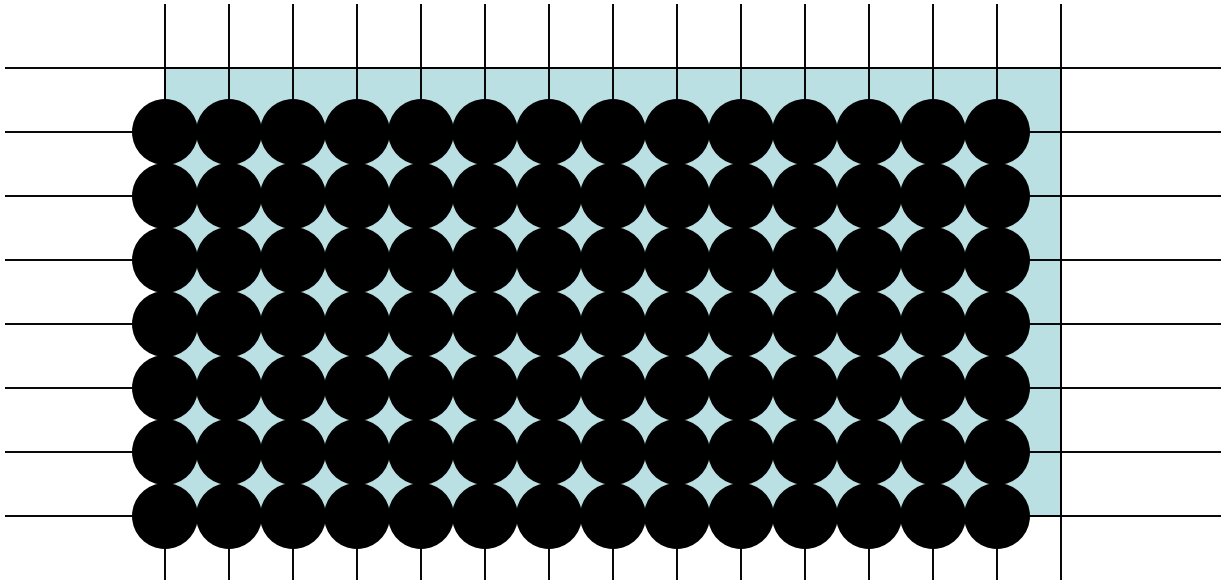
Definition

- pixels exactly on polygon edges:
  - are part of the polygon if they are below or left of the polygon
  - are not part of the polygon if they are above or right of the polygon



Preliminary Idea – Filling Rectangles

- iterate over y from y_{min} to y_{max}-1
- for each y_i:
  fill all pixels in between x_{min} and x_{max}-1

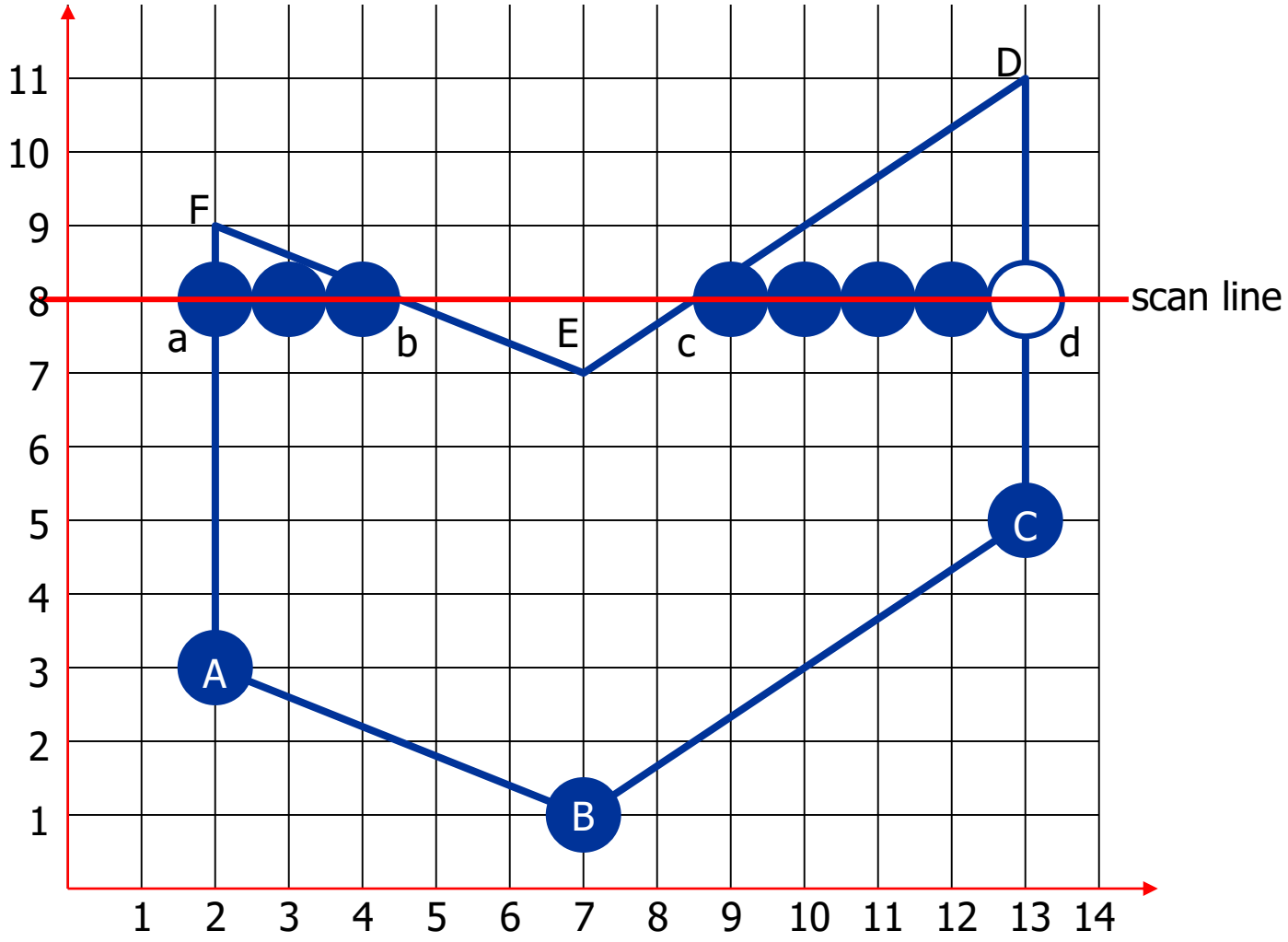


Generalization to Arbitrary Polygons

iterative algorithm:

- scan polygon by pixel rows bottom-up
- find all intersections with scan line
- sort intersections by *x*-value
- fill according to parity
  - parity is initially 0
  - parity changes between 0 and 1 at each intersection between scan line and polygon
  - fill where parity is 1

Example


11
10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14
F
a
b
E
c
D
d
scan line
A
B
C

Iterative Filling: Thoughts

- Which pixel is “inside” for non-Integer intersections?

  if we are “inside” (parity 1), draw left pixel;
  if we are “outside” (parity 0), draw right pixel
- How to treat intersections at Integer pixel coordinates?

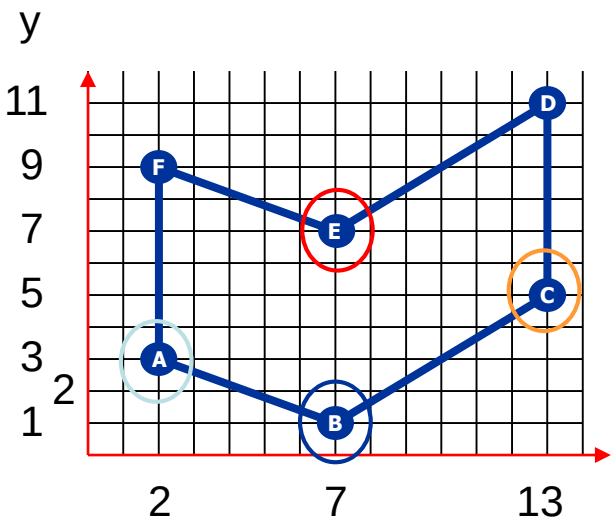
  draw if it is the starting pixel (smaller x),
  discard if it is the ending pixel (bigger x)
- How to treat intersections where two edges meet?

  treat edges with y_{min} at the intersection as usual,
  discard edges with y_{max} at the intersection
- How to treat horizontal edges?

  render bottom edges but discard top edges

Iterative Filling: General Idea

- edge table (ET) with needed edge data:
  - y_{min}: lowest *y*-value
  - x_{start}: *x*-value for vertex with y_{min}
  - y_{max}: highest *y*-value
  - $t=^{dx}/_{dy}=^{1}/_{m}$: edge offset between two scan lines



edge	y_{min}	x_{start}	y_{max}	dx/dy	
AB	1	7	3	-5/2	← (2-7)/(3-1)
BC	1	7	5	6/4	←(13-7)/(5-1)
FA	3	2	9	0	← 0/6
CD	5	13	11	0	← 0/6
EF	7	7	9	-5/2	← (2-7)/(9-7)
DE	7	7	11	6/4	← (13-7)/11-7)

Iterative Filling

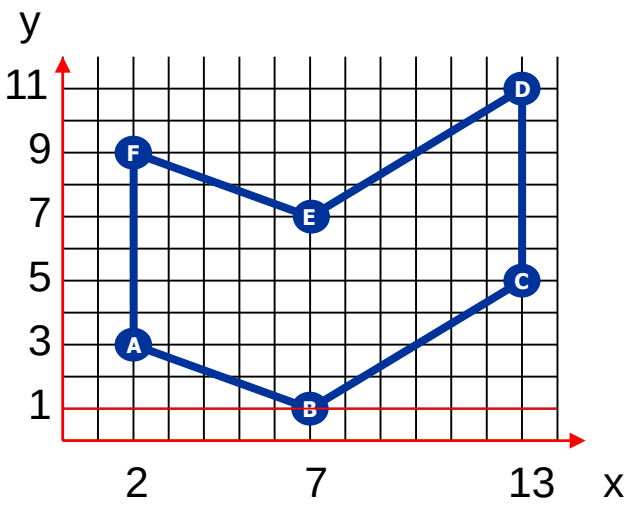
general algorithm:

- iterate over scan lines and maintain active edge table (AET)
- AET contains intersection point sorted by growing *x*-value
- parts between each two pairs are filled
- AET is updated for each new scan line
- algorithm terminates when AET is empty

Iterative Filling: Detailed Algorithm

- initialize y with smallest *y*-value in ET
- initialize AET as empty
- repeat until ET and AET are empty:
  - move all entries with y_{min} = y from ET to AET
  - sort AET by x_{start}
  - fill in scan line y pair-wise according to AET
  - remove all entries with y_{max}=y from AET
  - increment y
  - for all entries in AET, update x_{start} value by adding dx/dy

Iterative Filling: Detailed Example



edge table (ET)

edge	y_{min}	x_{start}	y_{max}	dx/dy
AB	1	7	3	-5/2
BC	1	7	5	3/2
FA	3	2	9	0
CD	5	13	11	0
EF	7	7	9	-5/2
DE	7	7	11	3/2

AET (scan line 1)

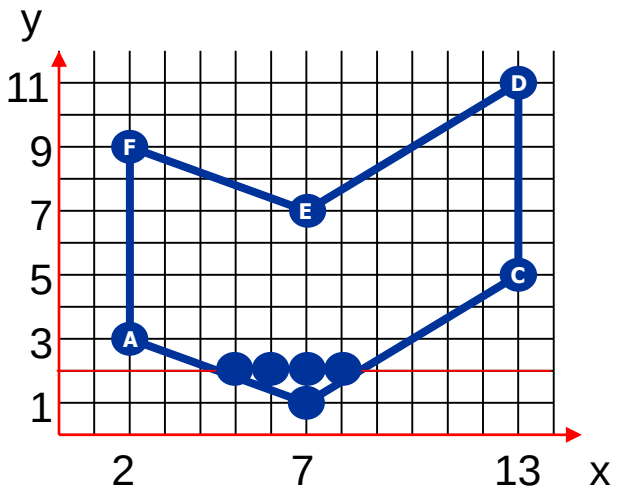
edge	y_{min}	x_{start}	y_{max}	dx/dy
AB	1	**7**	3	-5/2
BC	1	**7**	5	3/2

y_{min}-value of first pair is 1, thus, draw

AET (scan line 2)

edge	y_{min}	x_{start}	y_{max}	dx/dy
AB	1	**9/2**	3	-5/2
BC	1	**17/2**	5	3/2

fill between x-values of
- (9/2, 2) round up (outside) (5,2)
- (17/2, 2) round down (inside) (8,2)



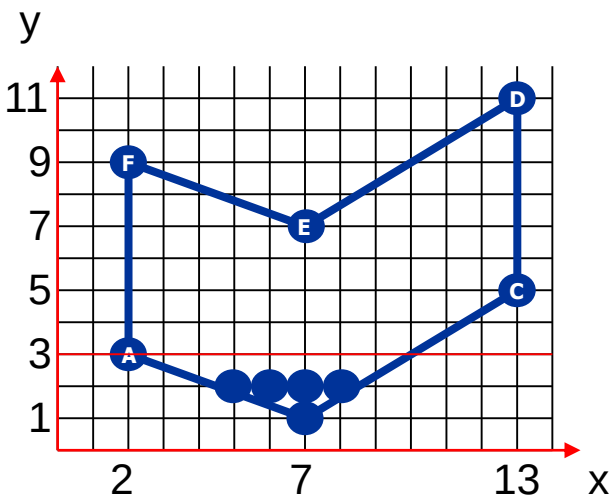
edge table (ET)

edge	y_{min}	x_{start}	y_{max}	dx/dy
FA	3	2	9	0
CD	5	13	11	0
EF	7	7	9	-5/2
DE	7	7	11	3/2

AET (going to scan line 3)

edge	y_{min}	x_{start}	y_{max}	dx/dy
AB	1	**4/2**	3	-5/2
BC	1	**20/2**	5	3/2

Iterative Filling: Detailed Example



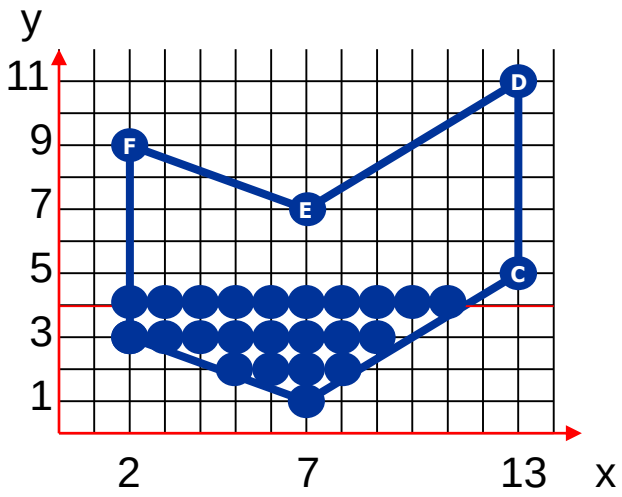
edge table (ET)

edge	y_{min}	x_{start}	y_{max}	dx/dy
FA	3	2	9	0
CD	5	13	11	0
EF	7	7	9	-5/2
DE	7	7	11	3/2

AET (scan line 3)

edge	y_{min}	x_{start}	y_{max}	dx/dy
AB	1	4/2	3	-5/2
FA	3	**2**	9	0
BC	1	**20/2**	5	3/2

- remove edge AB
- move edge FA from ET to AET
- fill between 2 and 20/2
  first pixel is inside and
  last pixel is outside



edge table (ET)

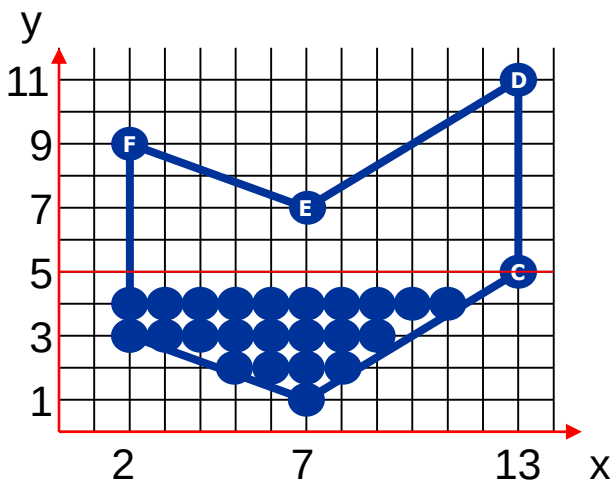
edge	y_{min}	x_{start}	y_{max}	dx/dy
CD	5	13	11	0
EF	7	7	9	-5/2
DE	7	7	11	3/2

AET (scan line 4)

edge	y_{min}	x_{start}	y_{max}	dx/dy
FA	3	**2**	9	0
BC	1	**23/2**	5	3/2

- fill between both entries
  round off (23/2) to (11)

Iterative Filling: Detailed Example



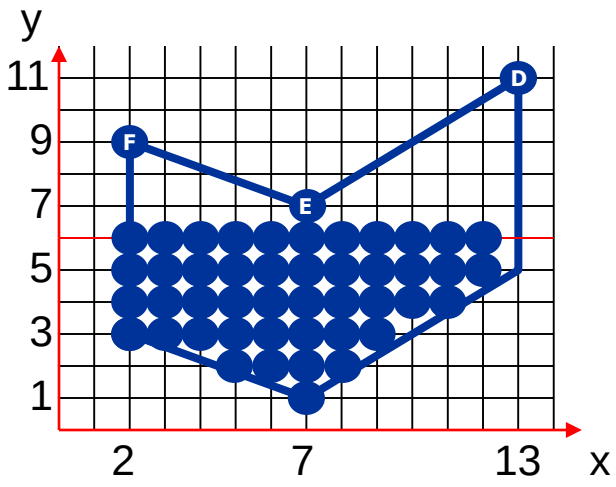
edge table (ET)

edge	y_{min}	x_{start}	y_{max}	dx/dy
CD	5	13	11	0
EF	7	7	9	-5/2
DE	7	7	11	3/2

AET (scan line 5)

edge	y_{min}	x_{start}	y_{max}	dx/dy
FA	3	**2**	9	0
CD	5	**13**	11	0
BC	1	26/2	5	3/2

- remove edge BC
- move edge CD from ET to AET
- fill between 2 and 26/2
  with 26/2 considered to be outside



edge table (ET)

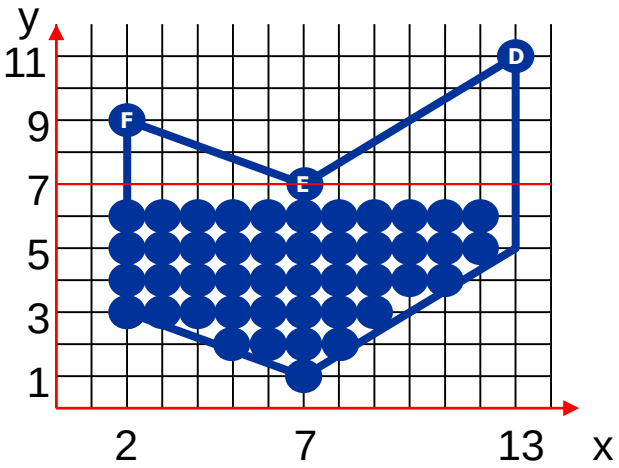
edge	y_{min}	x_{start}	y_{max}	dx/dy
EF	7	7	9	-5/2
DE	7	7	11	3/2

AET (scan line 6)

edge	y_{min}	x_{start}	y_{max}	dx/dy
FA	3	**2**	9	0
CD	5	**13**	11	0

- fill between both entries with
  (13, 6) considered to be outside

Iterative Filling: Detailed Example



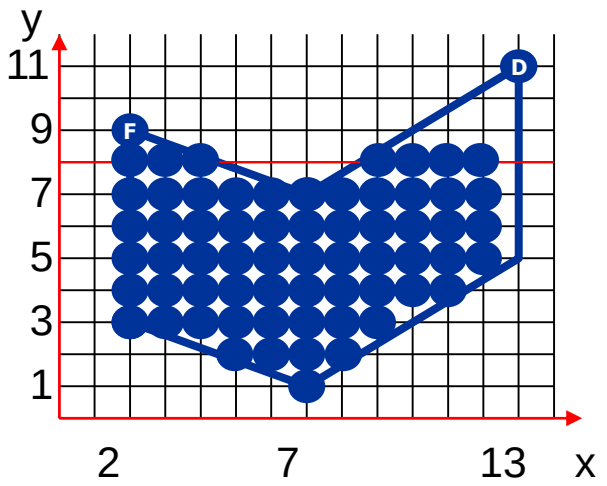
edge table (ET)

edge	y_{min}	x_{start}	y_{max}	dx/dy
EF	7	7	9	-5/2
DE	7	7	11	3/2

AET (scan line 7)

edge	y_{min}	x_{start}	y_{max}	dx/dy
FA	3	**2**	9	0
EF	7	**7**	9	-5/2
DE	7	**7**	11	3/2
CD	5	**13**	11	0

- move edges DE and EF into AET
- fill between first 2 entries; (7, 7) is outside
- fill between last 2 entries; (13, 7) is outside



edge table (ET)

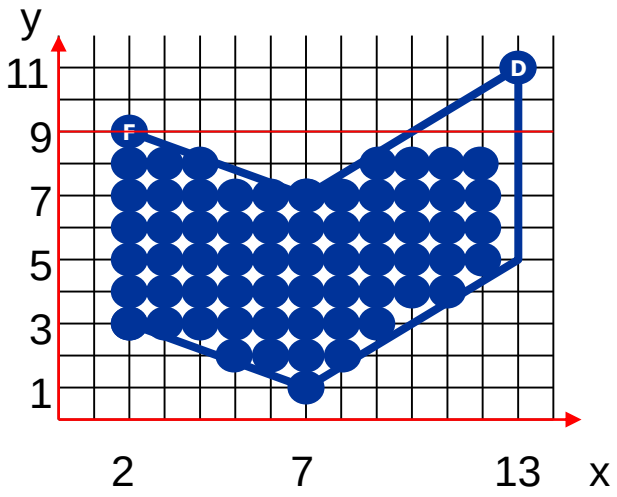
edge	y_{min}	x_{start}	y_{max}	dx/dy
--	--	--	--	--

- fill between first two entries (9/2, 8) rounded down to (4, 8)
- fill between last two entries; (17/2, 8) rounded down to (9, 8)

AET (scan line 8)

edge	y_{min}	x_{start}	y_{max}	dx/dy
FA	3	**2**	9	0
EF	7	**9/2**	9	-5/2
DE	7	**17/2**	11	3/2
CD	5	**13**	11	0

Iterative Filling: Detailed Example



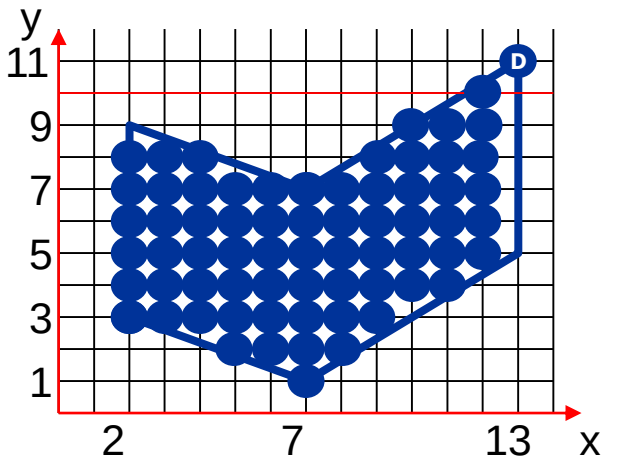
edge table (ET)

edge	y_{min}	x_{start}	y_{max}	dx/dy
--	--	--	--	--

AET (scan line 9)

edge	y_{min}	x_{start}	y_{max}	dx/dy
FA	3	**2**	9	0
EF	7	**4/2**	9	-5/2
DE	7	**20/2**	11	3/2
CD	5	**13**	11	0

- fill between fist two enties; (2, 9) is not set because it is ymax of two edges
- fill between last two entries; (13, 9) is outside
- remove FA and EF



edge table (ET)

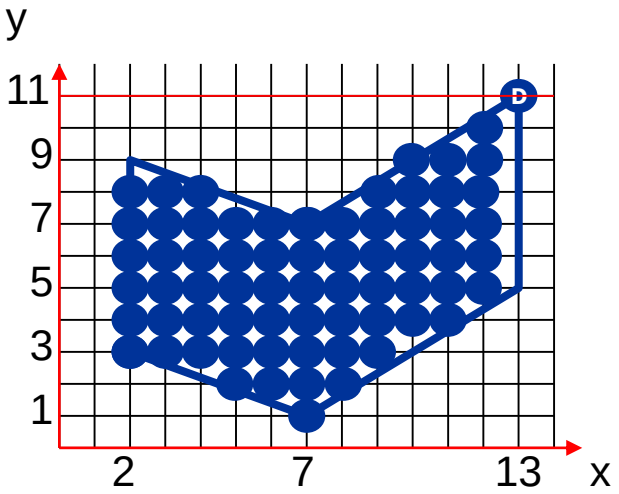
edge	y_{min}	x_{start}	y_{max}	dx/dy
--	--	--	--	--

AET (scan line 10)

edge	y_{min}	x_{start}	y_{max}	dx/dy
DE	7	**23/2**	11	3/2
CD	5	**13**	11	0

-fill between both entries; (23/2, 10) is rounded up to (12, 10); (13, 10) is considered to be outside

Iterative Filling: Detailed Example



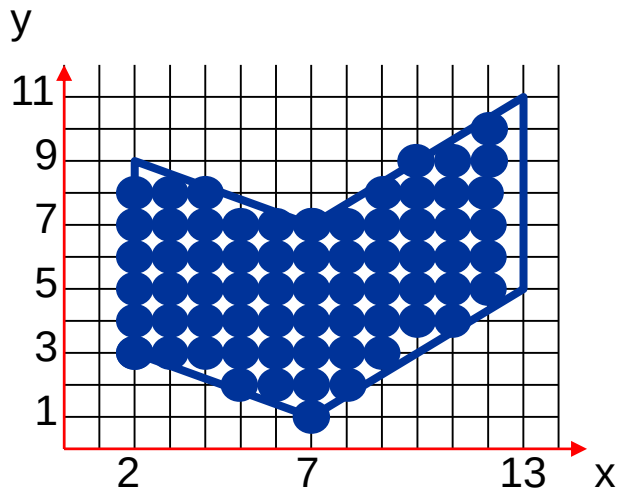
edge table (ET)

edge	y_{min}	x_{start}	y_{max}	dx/dy
--	--	--	--	--

AET (Scanline 11)

edge	y_{min}	x_{start}	y_{max}	dx/dy
DE	7	**26/2**	11	3/2
CD	5	**13**	11	0

- pixel (13, 11) is not set because it is y_{max} of two edges
- remove edges DE and CD



edge table (ET)

edge	y_{min}	x_{start}	y_{max}	dx/dy
--	--	--	--	--

AET

edge	y_{min}	x_{start}	y_{max}	dx/dy
--	--	--	--	--

- ET and AET are empty
- we are done, algorithm terminates

Note 1: ET/AET Sorting & Efficiency

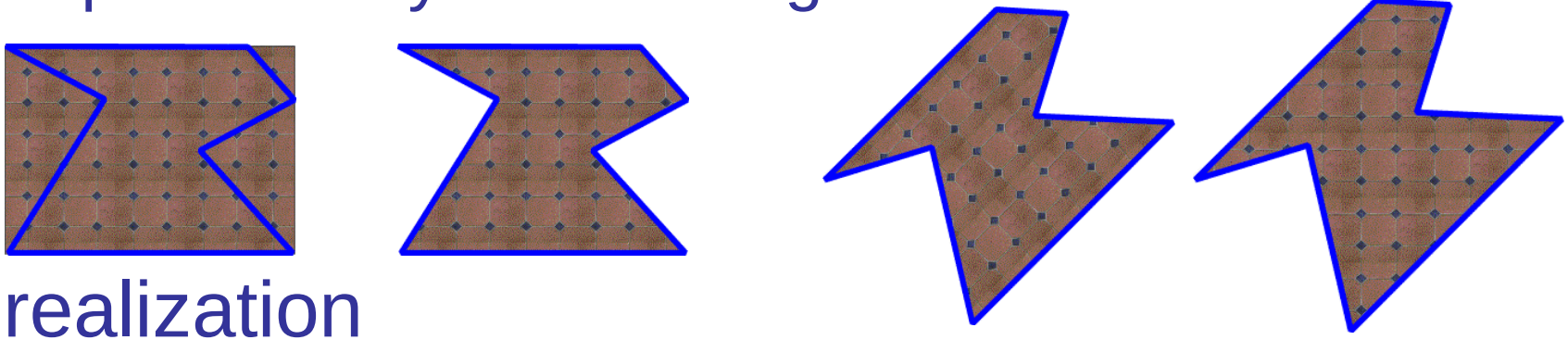
- which sorting algorithm to use?
  - depends on the average complexity of the polygons to fill
- complexity of sorting?
  - Bubble-Sort: $O(N^2)$
  - Merge-Sort: $O(N \log N)$
- how many operations needed?
  - Bubble-Sort: $k_1 N^2 + c_1$
  - Merge-Sort: $k_2 N \log N + c_2$
  - depends on number of items to sort!!!

Note 1: ET/AET Sorting & Efficiency

- which sorting algorithm to use in special cases, e.g., having to fill only quads?
  - 4 points to sort per line maximum!
  - elaborate sorting algorithms use elaborate data structures that need to be built
  - use “hard-coded” Bubble Sort: $O(n^2)$ but maximum 6 comparisons!

Note 2: Iterative Filling with Patterns

- two possible techniques:
  - pattern moves with polygon
  - pattern stays with background



- realization
  - first technique: interpolation (various methods)
  - second technique: logic AND with set pixel (1) /not set pixel (0) and pattern pixel

Iterative Filling: Summary

- can handle arbitrary polygons (convex and concave, with self-intersections)
- efficient algorithm
  - no complex computations, no multiplications
  - increments pre-computed
  - easy sorting during iteration in AET only
- simplification for convex polygons possible (one pair in AET only)