SCAN-LINE FILL ALGORITHMS

Acknowledgements

- Some of the material for the slides were adapted from College of Computer Information Science , Northeastern University
- Some of the images were taken from *Hearn, Baker, and Carithers, “Computer Graphics with OpenGL”*

Fill Algorithms

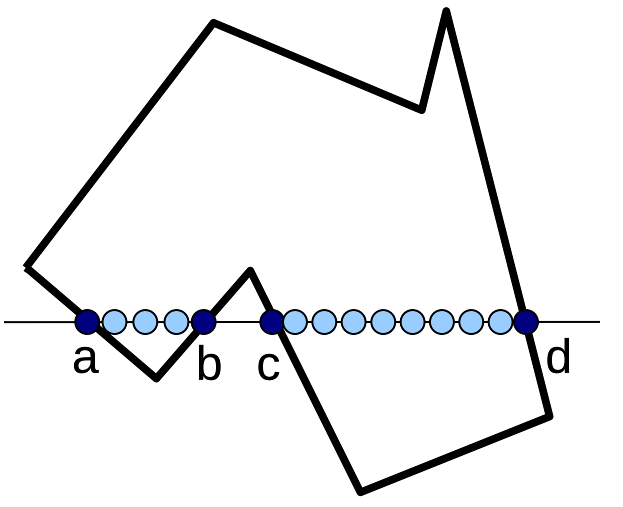
- Given the edges defining a polygon, and a color for the polygon, we need to fill all the pixels inside the polygon.
- Three different algorithms:
  - 1. Scan-line fill
  - 2. Boundary fill
  - 3. Flood fill

Filled Area Primitives

- Two basic approaches to area filling on raster systems:
  - Determine the overlap intervals for scan lines that cross the area (scan-line)
  - 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.

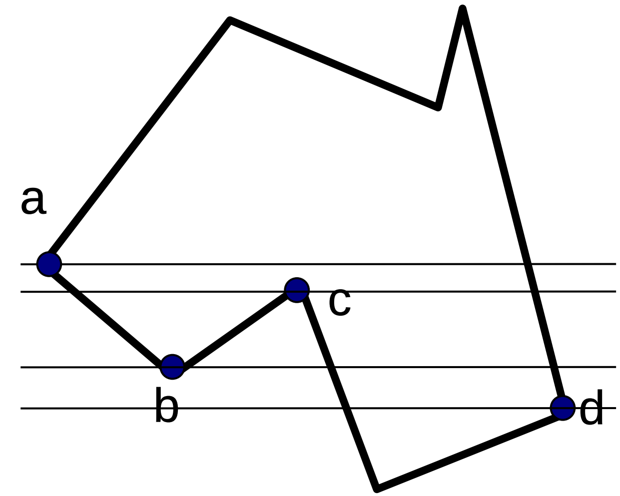
Scan-line Polygon Fill

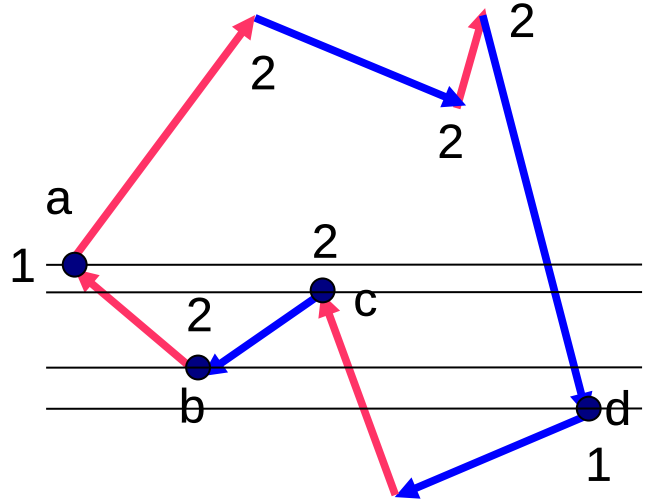
- For each scan-line:
  - Locate the intersection of the scan-line with the edges ($y=y_s$)
  - Sort the intersection points from left to right.
  - Draw the interiors intersection points pairwise. (a-b), (c-d)
- Problem with corners. Same point counted twice or not?



- *a*,*b*,*c* and *d* are intersected by 2 line segments each.
- Count *b*,c twice but *a* and *d* once. Why?

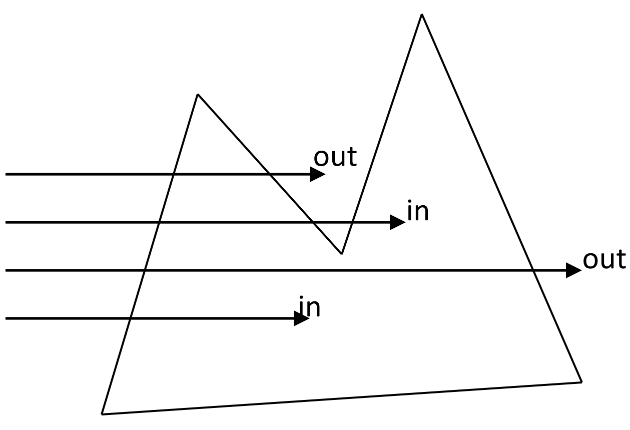
- Solution:
  Make a clockwise or counter-clockwise traversal on edges.
  Check if y is monotonically increasing or decreasing.
  If direction changes, double intersection, otherwise single intersection.





Scan Line Fill Algorithm

- Basic algorithm:
  - Assume scan line start from the left and is outside the polygon.
  - When intersect an edge of polygon, start to color each pixel (because now we're inside the polygon), when intersect another edge, stop coloring …
  - Odd number of edges: inside
  - Even number of edges: outside
- Advantage of scan-line fill: It does fill in the same order as rendering, and so can be pipelined.



Scan-Line Polygon Fill Algorithm

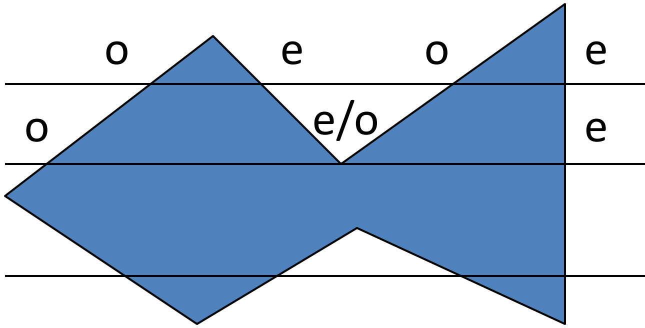
- Odd-parity rule

Calculate span extrema (intersections) on each scan line

Using parity rule to determine whether or not a point is inside a region

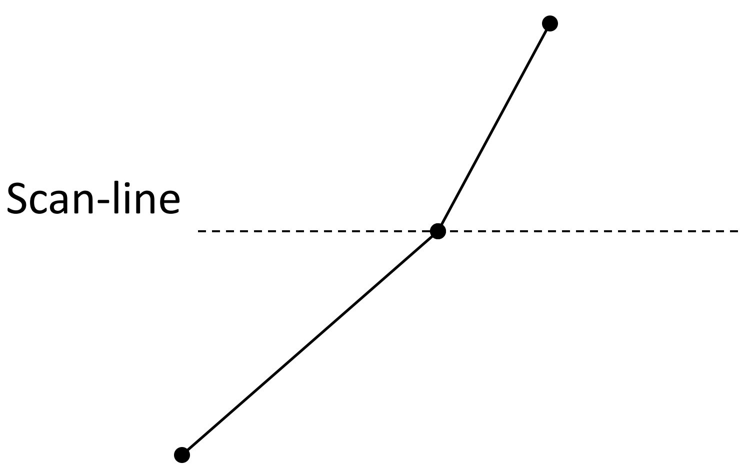
Parity is initially even
→ each intersection encountered thus inverts the parity bit

parity is odd → interior point (draw)
parity is even → exterior point (do not draw)

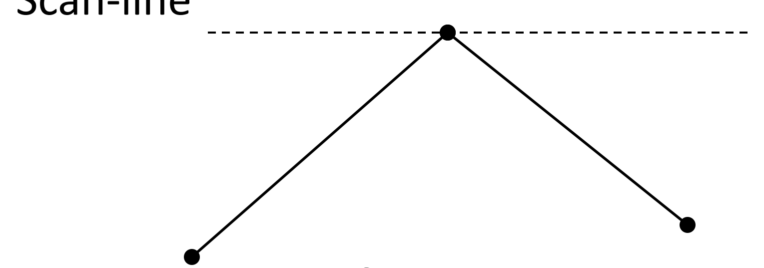


Scan Line Fill: What happens at edge end-point?

- Edge endpoint is duplicated.
- In other words, when a scan line intersects an edge endpoint, it intersects two edges.
- Two cases:
  - Case A: edges are monotonically increasing or decreasing
  - Case B: edges reverse direction at endpoint
- In Case A, we should consider this as only ONE edge intersection
- In Case B, we should consider this as TWO edge intersections



Case A



Case B

Speeding up Scan-Line Algorithm

- 1. Parallel algorithm: process each scan line in one processor. Each scan line is independent
- 2. From edge intersection with one scan line, derive edge intersection with next scan line
- 3. Active edge list

Scan-Line Polygon Fill Algorithm

- **Finding intersection pairs**:

Scan-line conversion method (for non-horizontal edges):

→ Taking advantage of the edge *coherence* property:
Incremental calculation between successive scan lines

→ Or using Bresenham's or mid-point scan line conversion algorithm on each edge
and keep a table of span extrema for each scan line

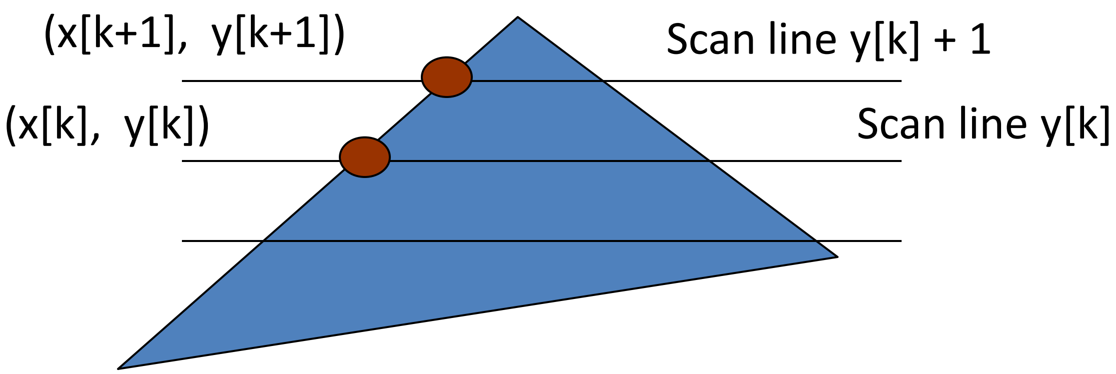
Scan-Line Polygon Fill Algorithm

- **Incremental scan line method**:

$m = (y[k+1] - y[k]) \;/\; (x[k+1] - x[k])$

$y[k+1] - y[k] = 1$

$x[k+1] = x[k] + 1/m \rightarrow x[k] = x[0] + k/m$



Derive next intersection

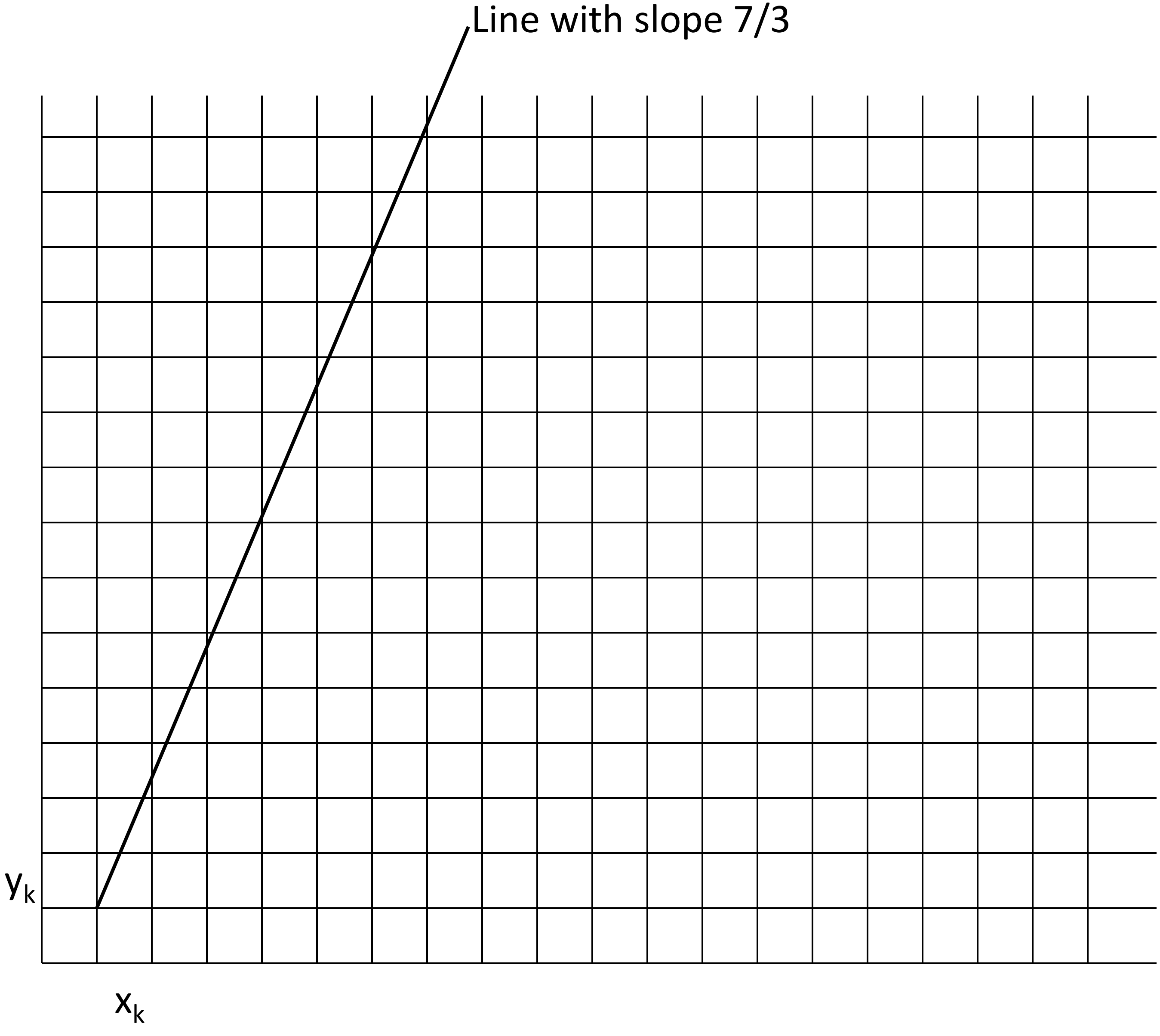
- Suppose that slope of the edge is

  $m = \boldsymbol{D}y/\boldsymbol{D}x$

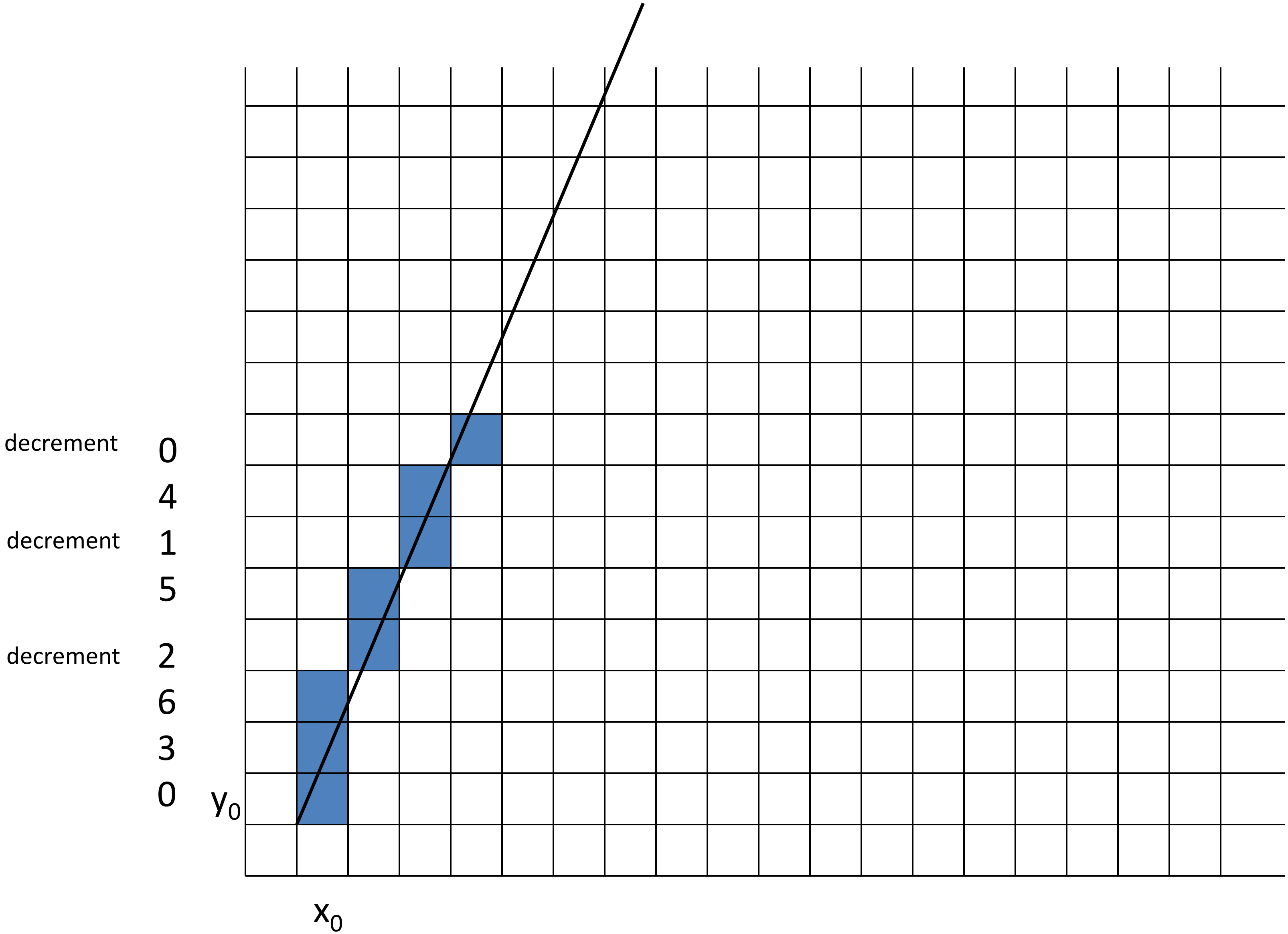
- Let x_k be the x intercept of the current scan line, and x_{k+1} be the x intercept of the next scan line, then

  $x_{k+1} = x_k + \boldsymbol{D}x/\boldsymbol{D}y$

- Algorithm:
  - Suppose m = 7/3
  - Initially, set counter to 0, and increment to 3 (which is $\boldsymbol{D}x$).
  - When move to next scan line, increment counter by increment
  - When counter is equal or greater than 7 (which is $\boldsymbol{D}y$), increment the x-intercept (in other words, the x-intercept for this scan line is one more than the previous scan line), and decrement counter by 7.



Line with slope 7/3
y_k
x_k



decrement 0
4
decrement 1
5
decrement 2
6
3
0
y_0
x_0

Scan-Line Polygon Fill Algorithm

- **Incremental scan line method**:

Bucket sorted edge table: containing all edges sorted by their smaller y coordinate.

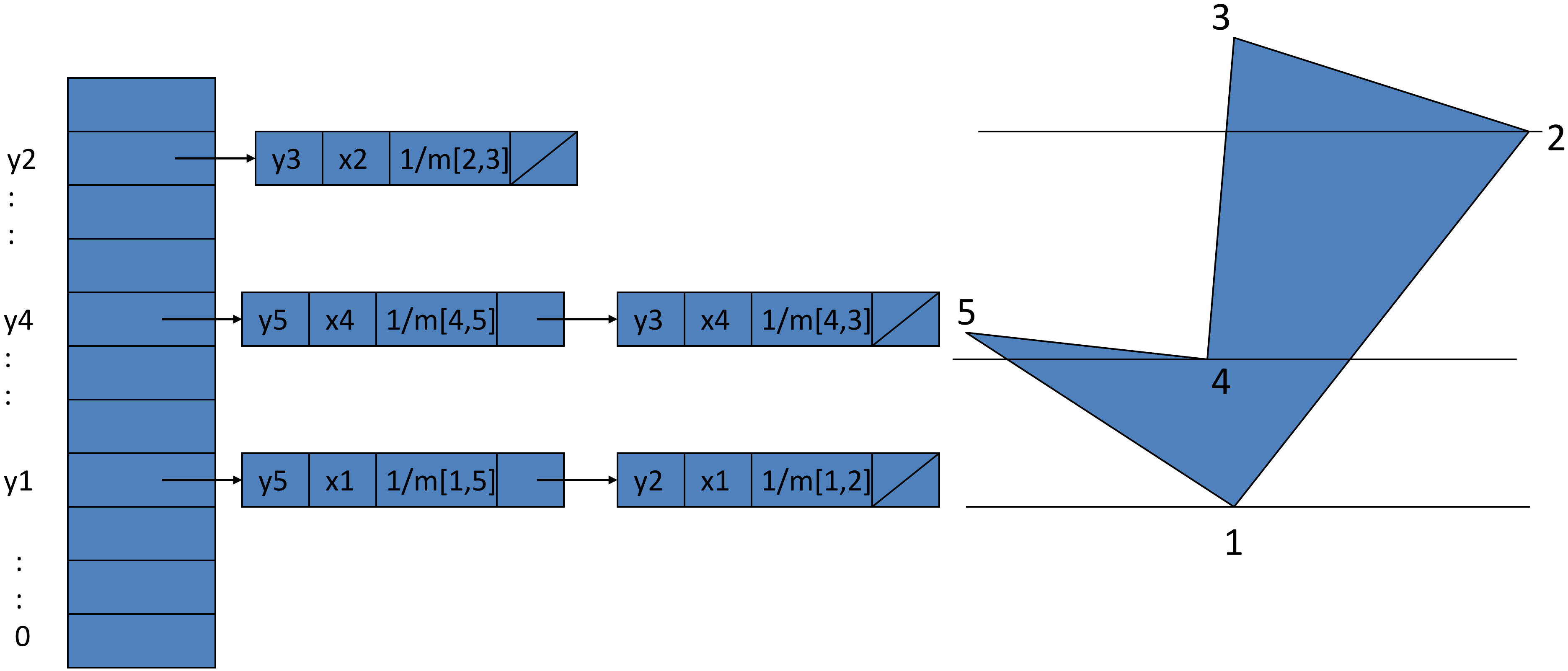
Each bucket: edges are recorded in order of increasing x coordinate of the endpoint.

Each entry of a bucket: y[max] of the edge, x[min] of lower endpoint and 1/m (x increment)

Active-edge table (or list): keep track of the set of edges that the scan line intersects and the intersection points in a data structure

Scan-Line Polygon Fill Algorithm

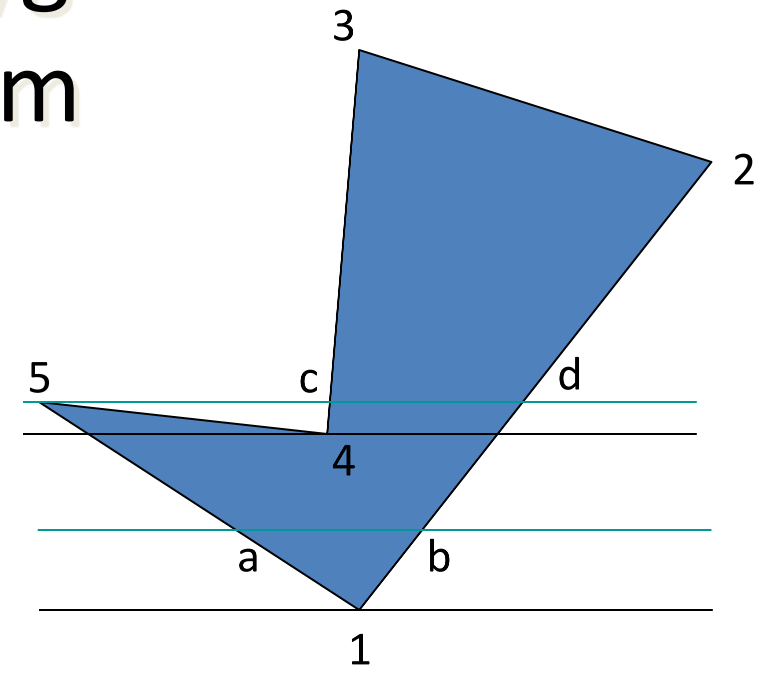
- **Example:**

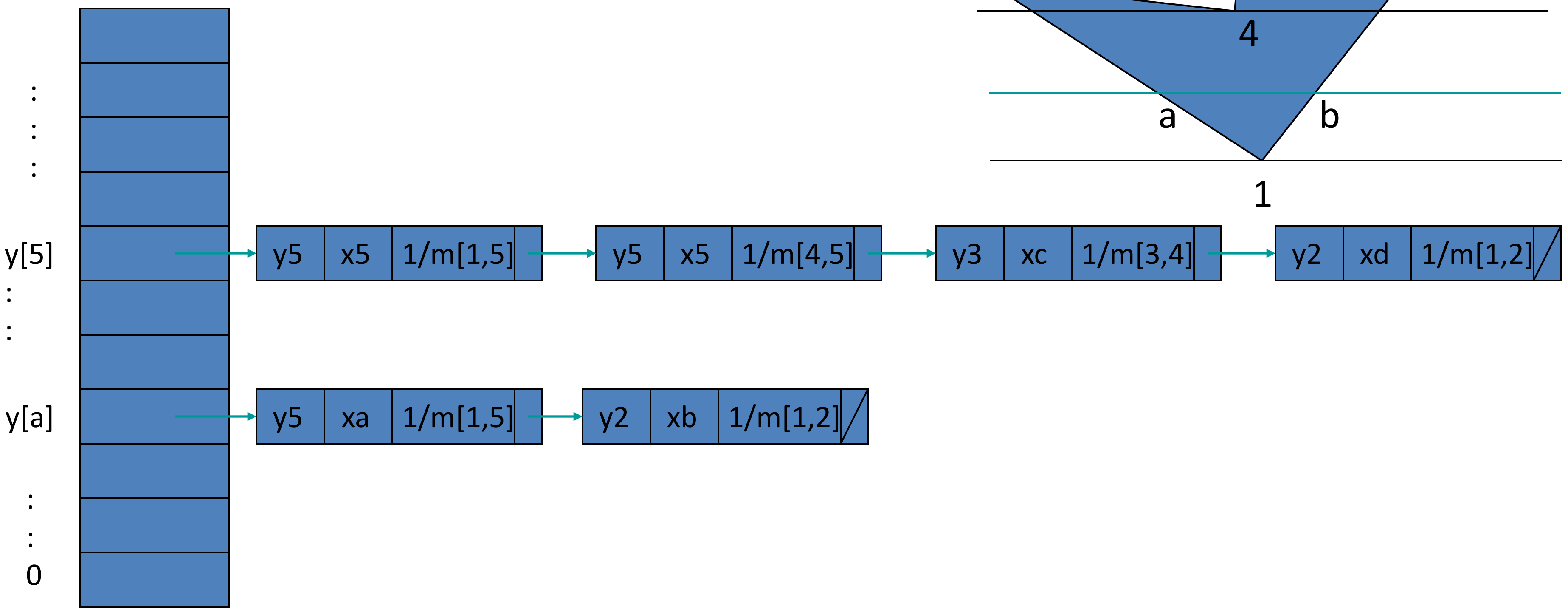


Sorted edge table (ET)

Scan-Line Polygon Fill Algorithm

- **Example:**





Active edge table (AET)

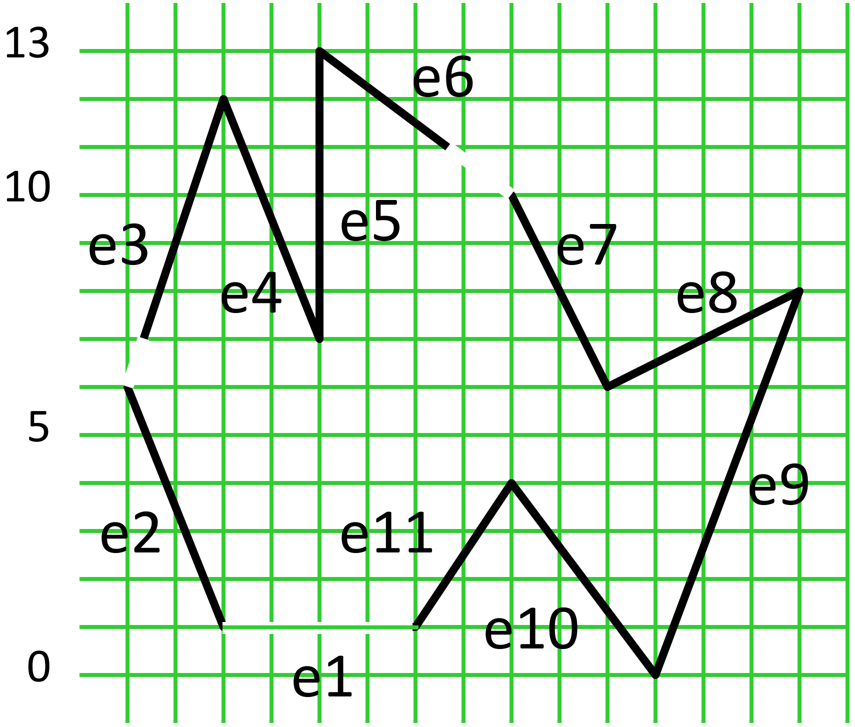
Active Edge List

- Start with the sorted edge table.
  - In this table, there is an entry for each scan-line.
  - Add only the non-horizontal edges into the table.
    - For each edge, we add it to the scan-line that it begins with (that is, the scan-line equal to its lowest y-value).
    - For each edge entry, store (1) the x-intercept with the scan-line, (2) the largest y-value of the edge, and (3) the inverse of the slope.
  - Each scan-line entry thus contains a sorted list of edges. The edges are sorted left to right. (To maintain sorted order, just use insertion based on their x value.)
- Next, we process the scan lines from bottom to top.
  - We maintain an active edge list for the current scan-line.
  - The active edge list contains all the edges crossed by that scan line. As we move up, update the active edge list by the sorted edge table if necessary.
  - Use iterative coherence calculations to obtain edge intersections quickly.

Polygon Data Structure
after preprocessing

Edge Table (ET) has a list of edges for each scan line.

13			
12			
11	→ e6		
10			
9			
8			
7	→ e3	→ e4	→ e5
6	→ e7	→ e8	
5			
4			
3			
2			
1	→ e2	→ e11	
0	→ e10	→ e9	



The Algorithm

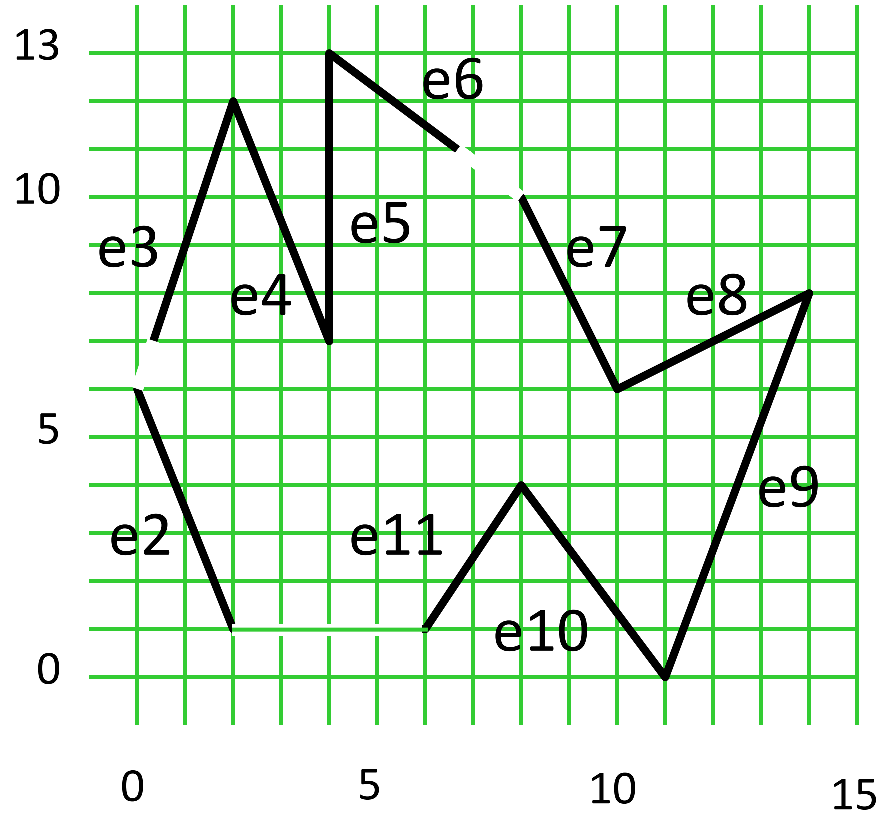
1. Start with smallest nonempty y value in ET.
2. Initialize SLB (Scan Line Bucket) to *nil*.
3. While current y $\leq$ top y value:
    a. Merge y bucket from ET into SLB; sort on xmin.
    b. Fill pixels between rounded pairs of x values in SLB.
    c. Remove edges from SLB whose ytop = current y.
    d. Increment xmin by 1/m for edges in SLB.
    e. Increment y by 1.

Running the Algorithm

ET

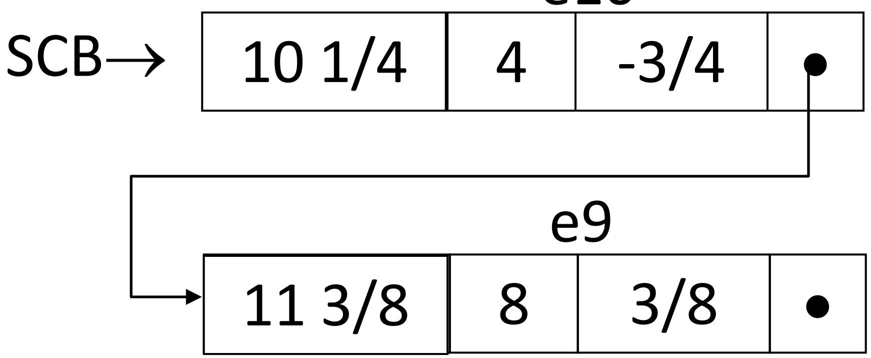
13
12
11 $\rightarrow$ e6
10
9
8
7 $\rightarrow$ e3 $\rightarrow$ e4 $\rightarrow$ e5
6 $\rightarrow$ e7 $\rightarrow$ e8
5
4
3
2
1 $\rightarrow$ e2 $\rightarrow$ e11
0 $\rightarrow$ e10 $\rightarrow$ e9

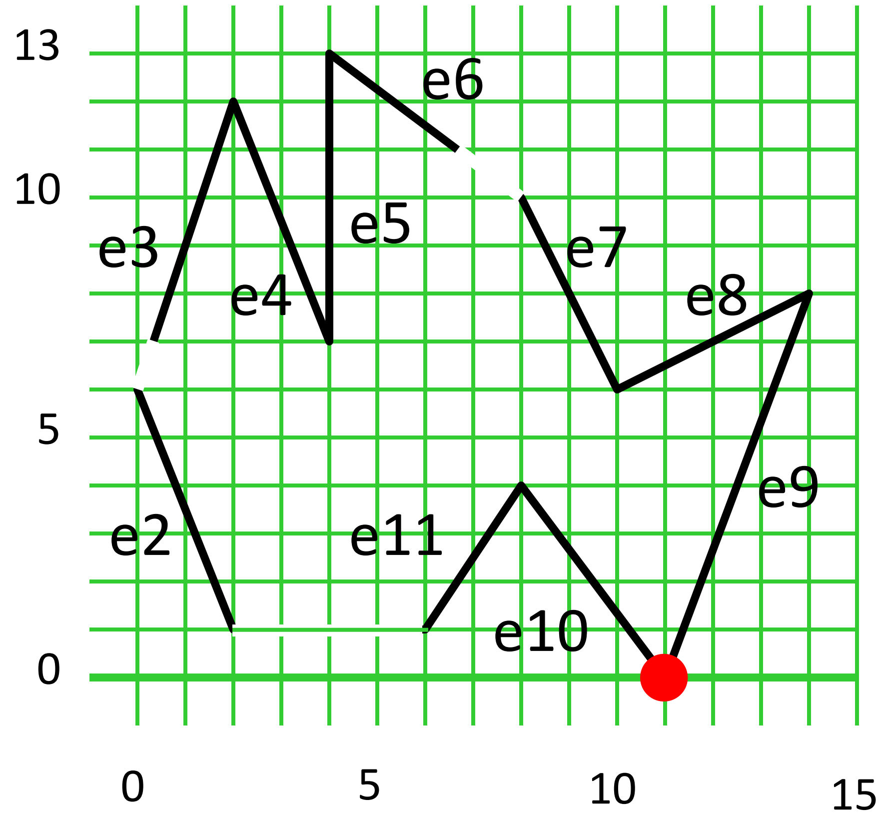
	xmin	ymax	1/m
e2	2	6	-2/5
e3	1/3	12	1/3
e4	4	12	-2/5
e5	4	13	0
e6	6 2/3	13	-4/3
e7	10	10	-1/2
e8	10	8	2
e9	11	8	3/8
e10	11	4	-3/4
e11	6	4	2/3



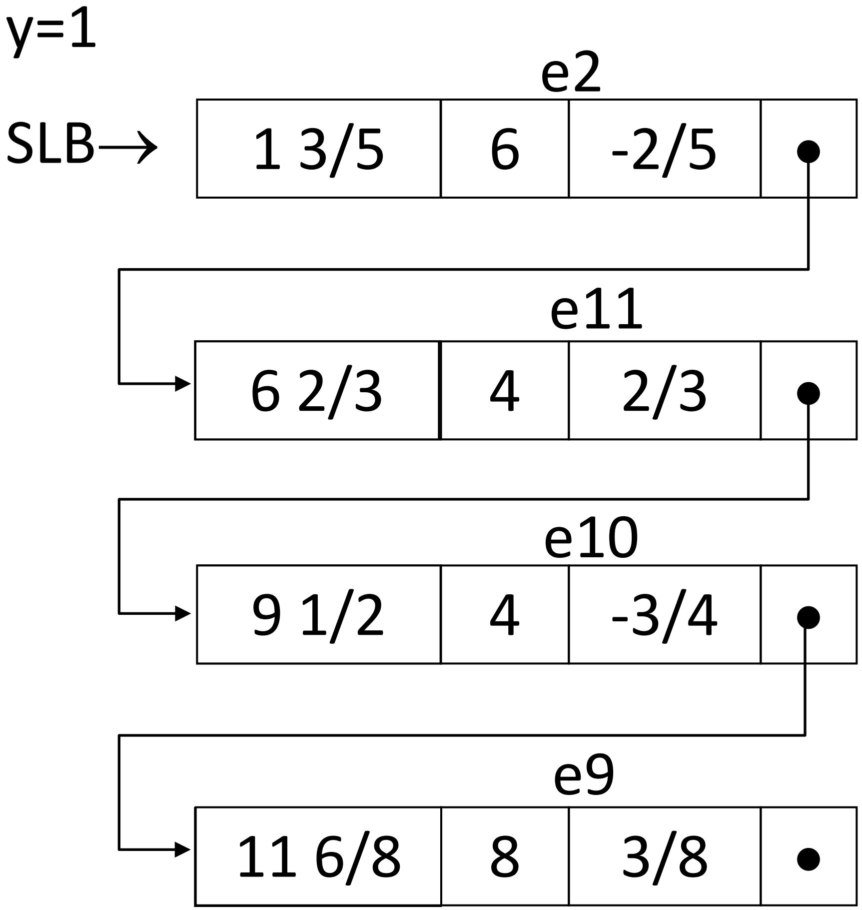
Running the Algorithm

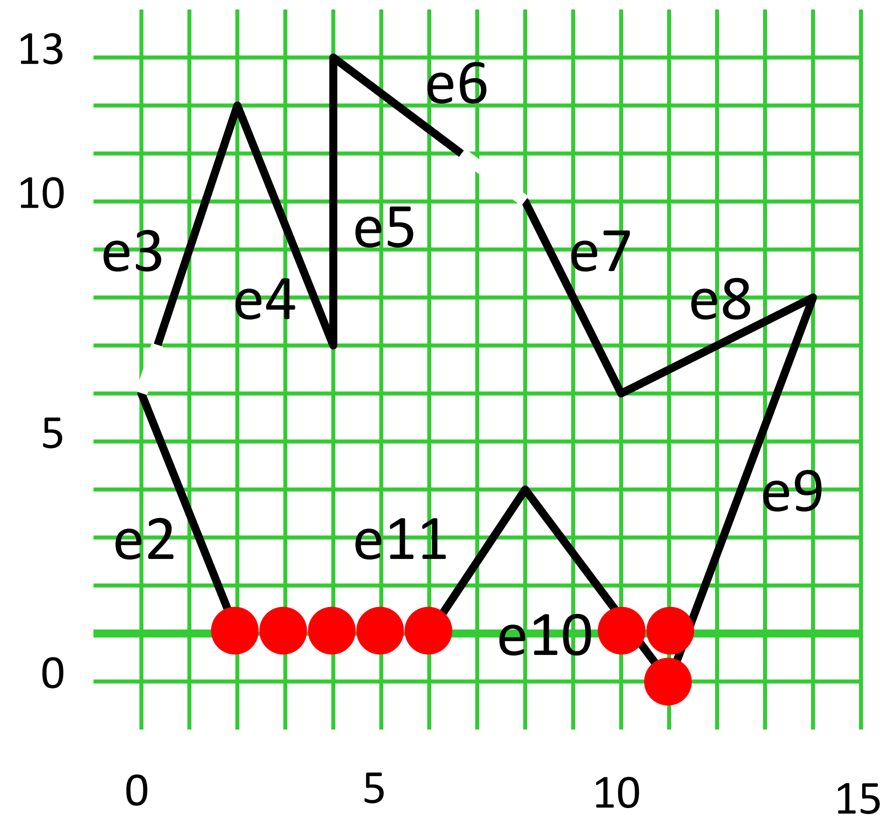
y=0





Running the Algorithm



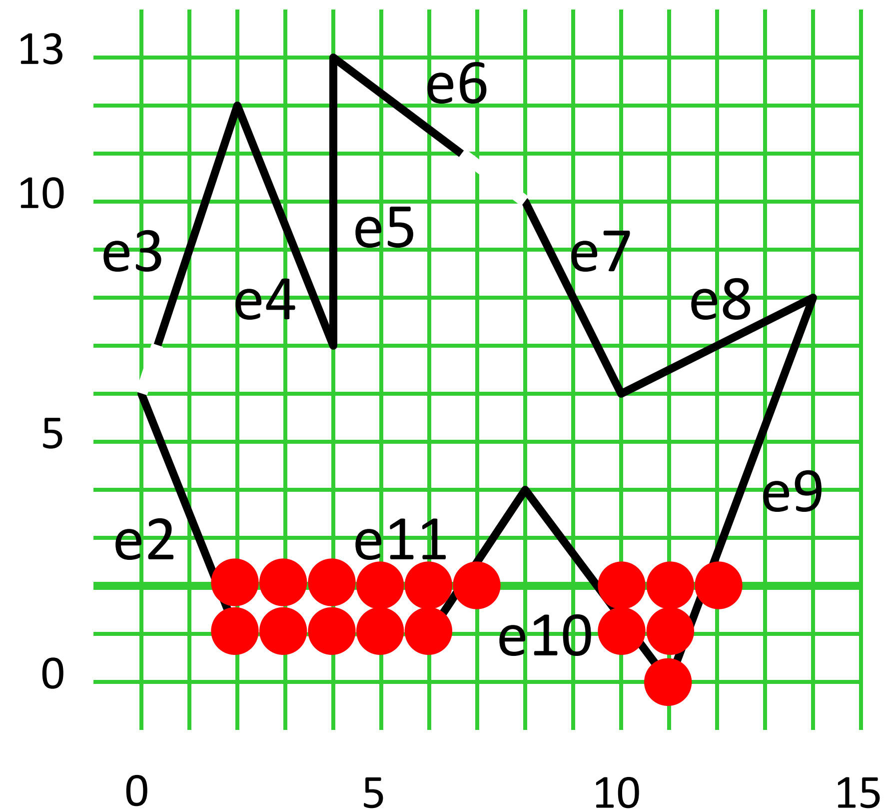


Running the Algorithm

y=2

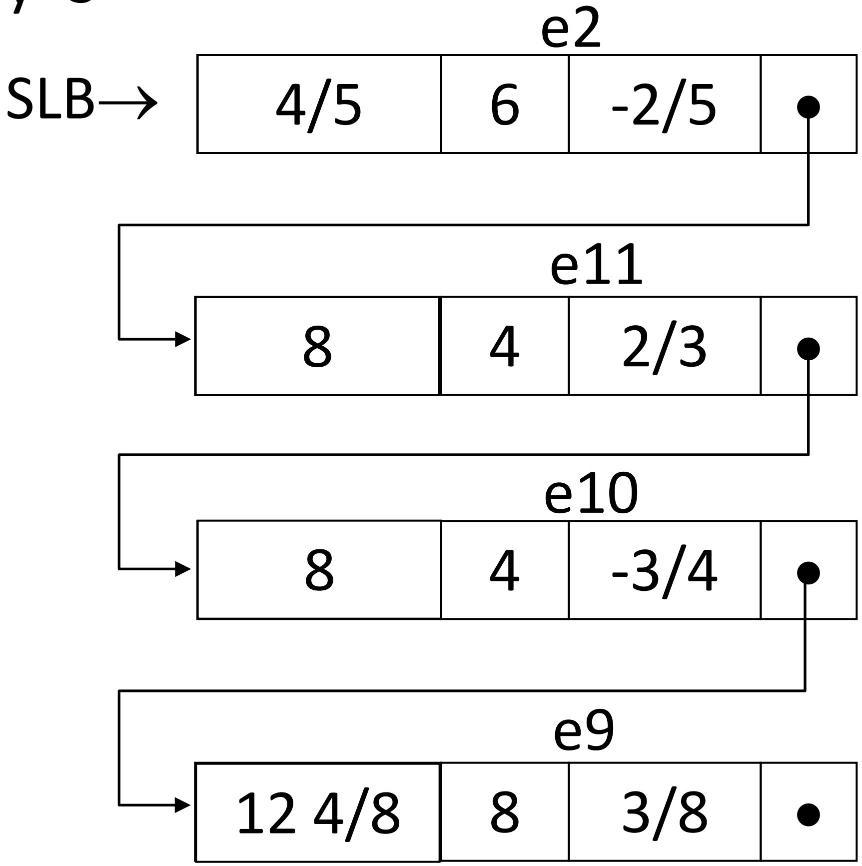
SLB→

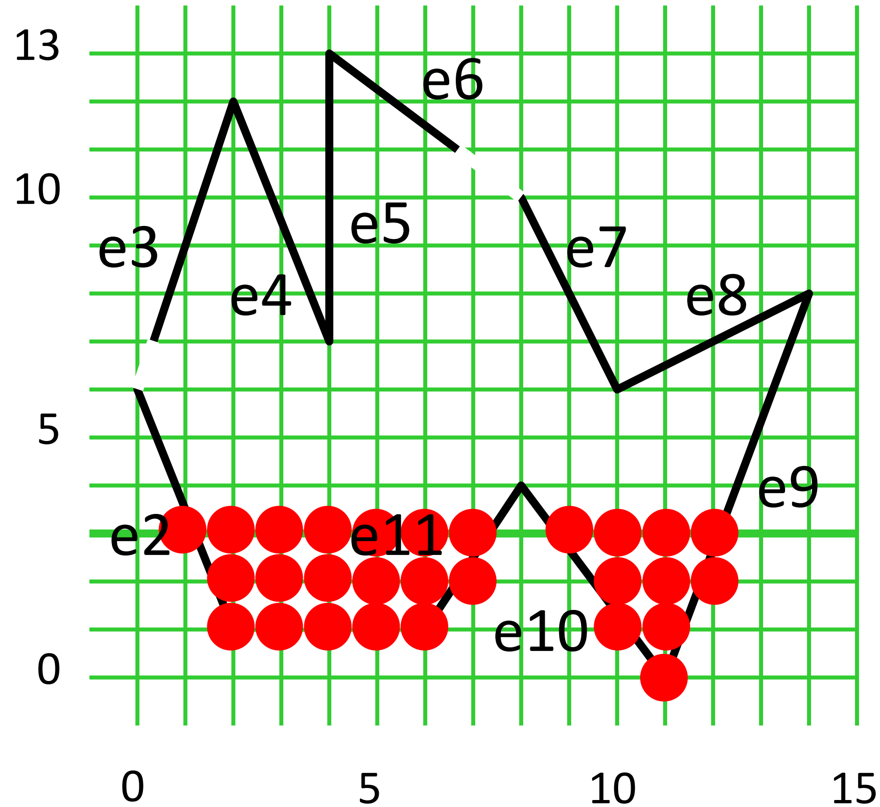
e2	1 1/5	6	-2/5	•
e11	7 1/3	4	2/3	•
e10	8 3/4	4	-3/4	•
e9	12 1/8	8	3/8	•



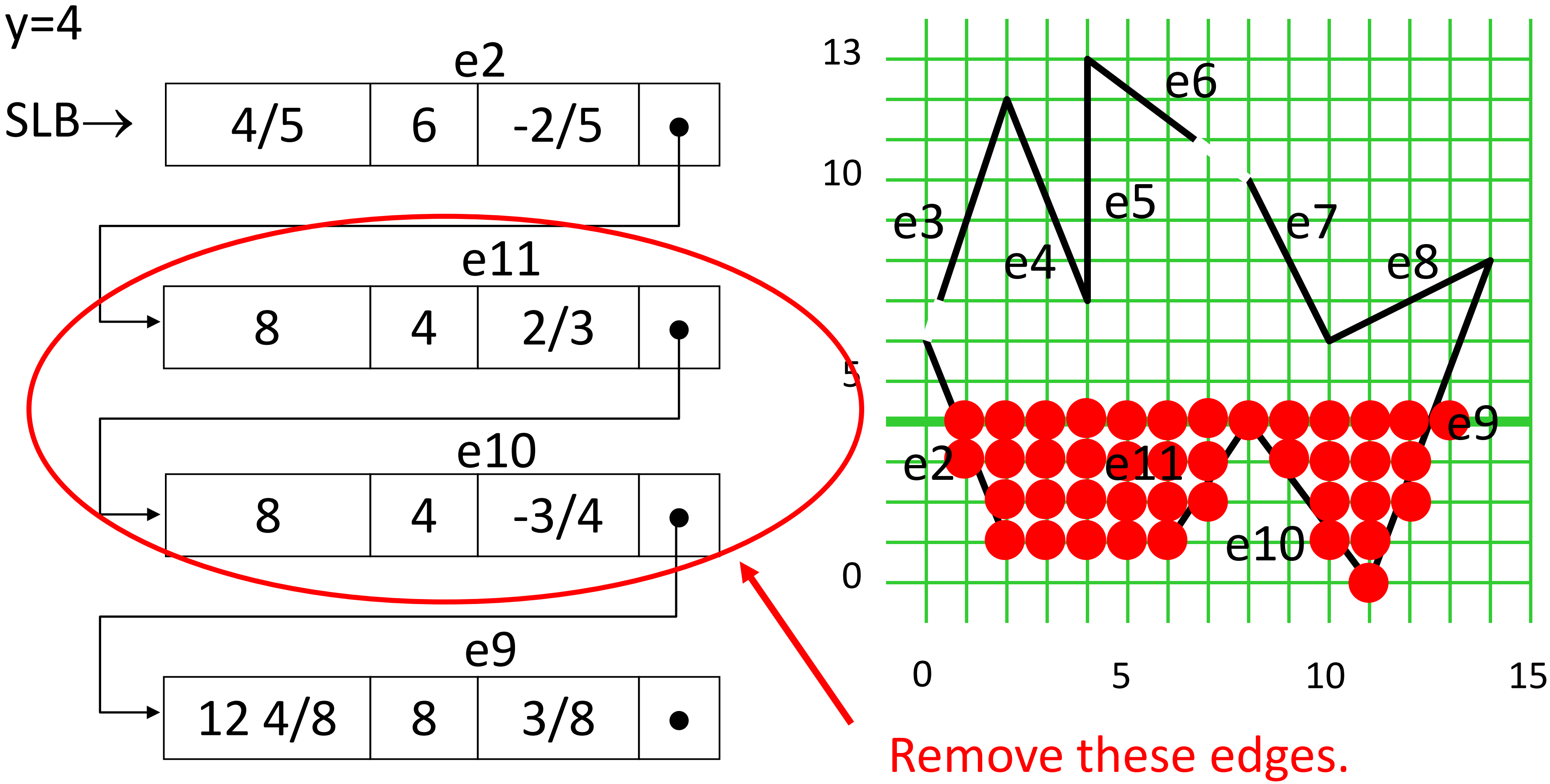
Running the Algorithm

y=3



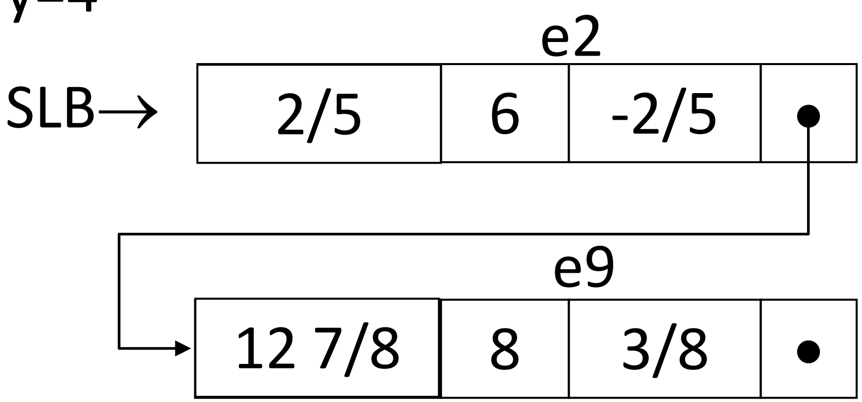


Running the Algorithm

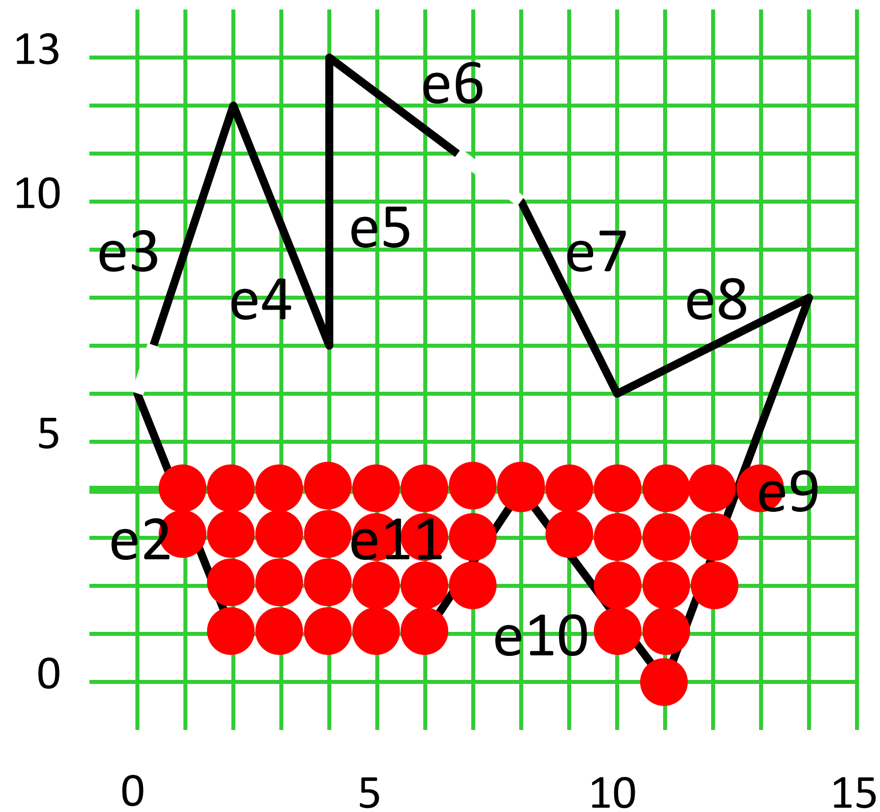


Running the Algorithm

y=4



e11 and e10 are removed.



Running the Algorithm

y=5

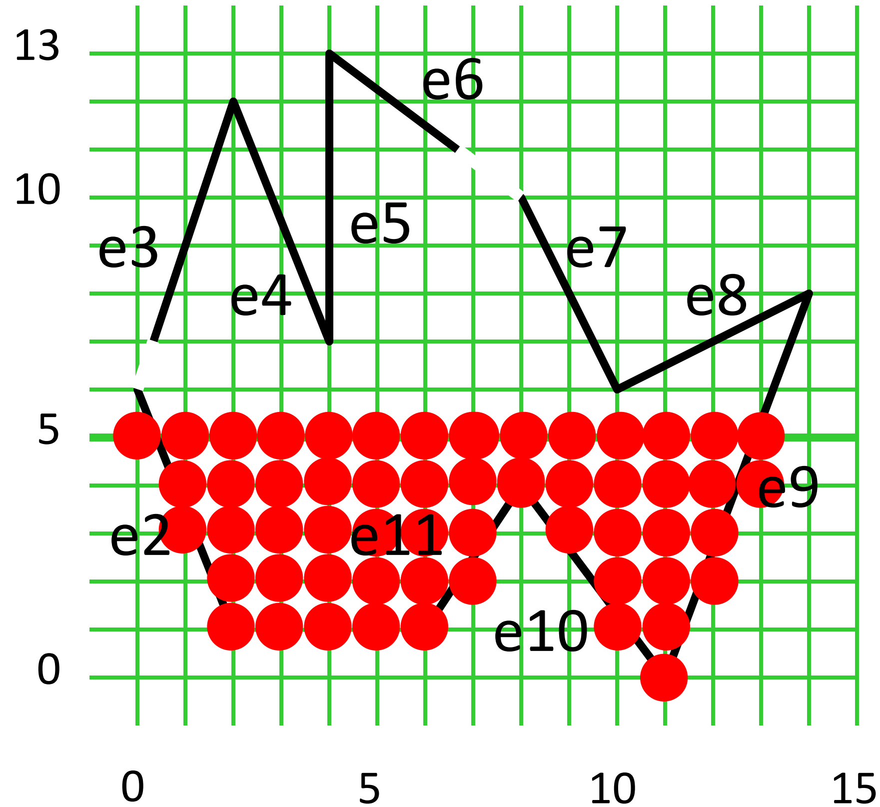
SLB→

e2

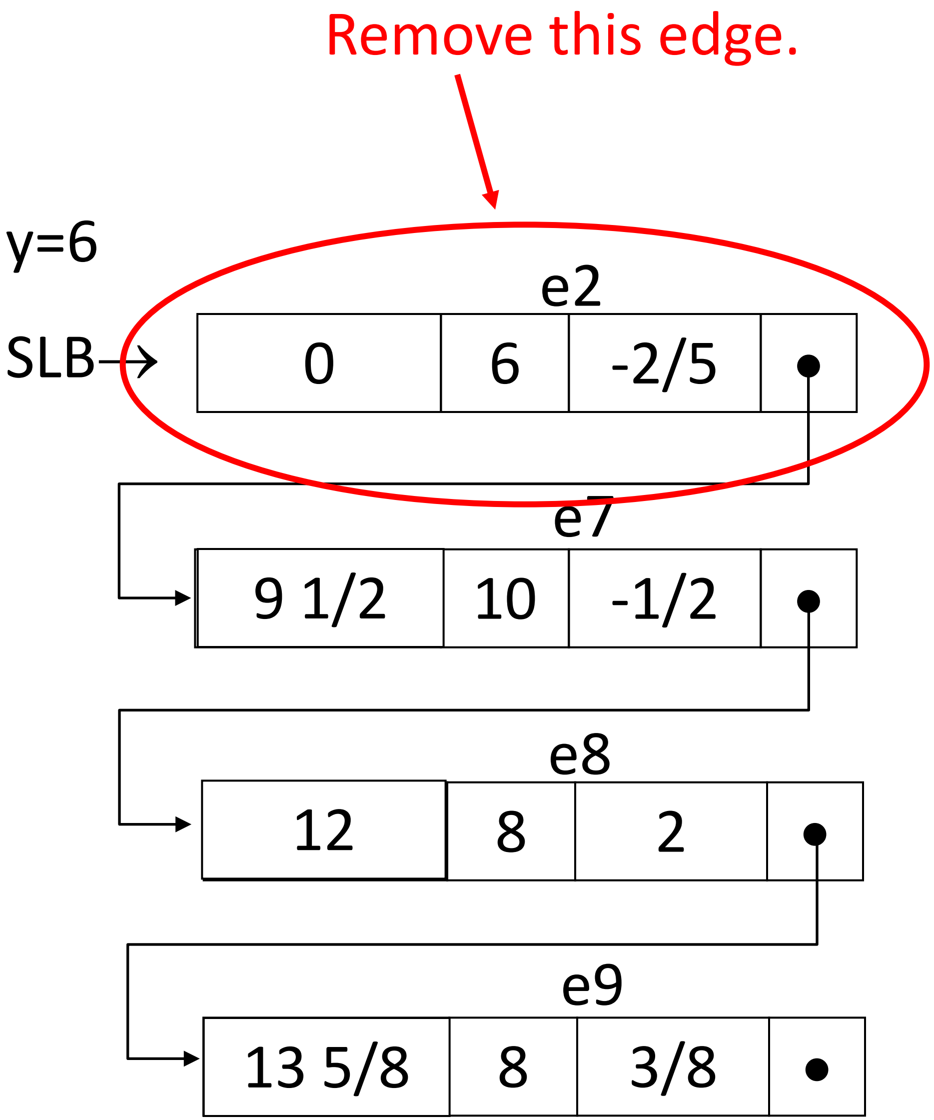
0	6	-2/5	•

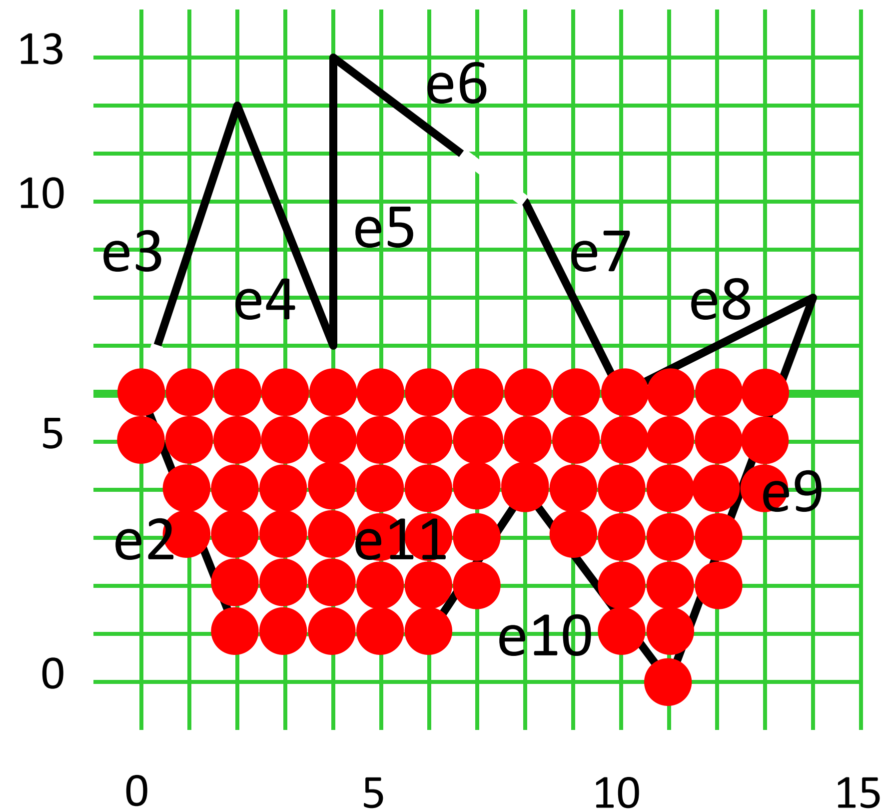
e9

13 2/8	8	3/8	•



Running the Algorithm





Running the Algorithm

Add these edges.