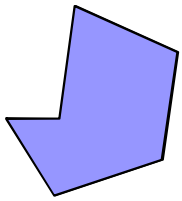
Rendering: Scan-line area fill

The algorithm
Applications

Scan-line algorithms

- **Purpose**: given a set of (2D) vertex coordinates for a polygon, fill the area surrounded by the polygon
- Polygons can be filled with a uniform colour or texture



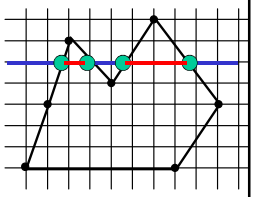
Scan-line algorithms

- Essential in rendering, i.e. conversion of geometric entities into image pixels
- Used, for example, in
  - Display of polygons
  - Hidden surface removal
  - Texture mapping

Scan-line algorithm - outline

For each scan line (each y-coordinate)

- Compute x coordinates of the intersections of the current scan line with all edges
- Sort these edge intersections by increasing x value
- Group the edge intersections by pairs (vertex intersections require special processing)
- Fill in the pixels on the scan line between pairs of values

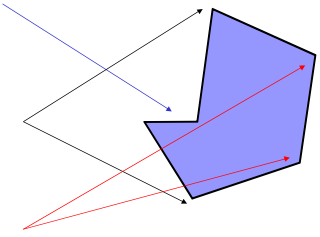


Vertex considerations

Do not include edges with zero slope

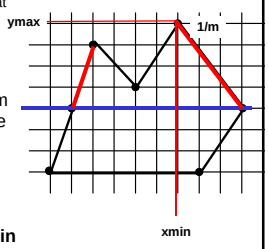
An extremum vertex - no additional processing

Non-extremum vertex - shorten a second edge by 1 unit in y direction



Computational structures

- Edge table
  - One table entry for each scan line which contains at least one “lower” vertex of a polygon edge
  - Properties of each edge are stored in a “bucket”
  - Table entry consists of an array of buckets
- Each bucket contains edges whose minimum y coordinate (**ymin**) starts at the bucket's line
- Each entry is a record containing:
  ymax **xmin** **increment**
- The array of buckets sorted according to **ymin**
- Records in a bucket sorted according to **xmin**



Polygon Data Structure

Edge record

xmin	ymax	1/m	• →
1	6	8/4	• →

(9, 6)

(1, 2)

xmin = x value at lowest y

ymax = highest y

Why 1/m?

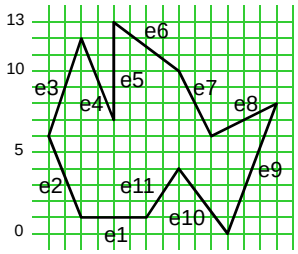
If y = mx + b, x = (y-b)/m.

x at y+1 = (y+1-b)/m = (y-b)/m + 1/m.

Polygon Data Structure

13
12
11
10 → e6
9
8
7 → e4 → e5
6 → e3 → e7 → e8
5
4
3
2
1 → e2 → e1 → e11
0 → e10 → e9

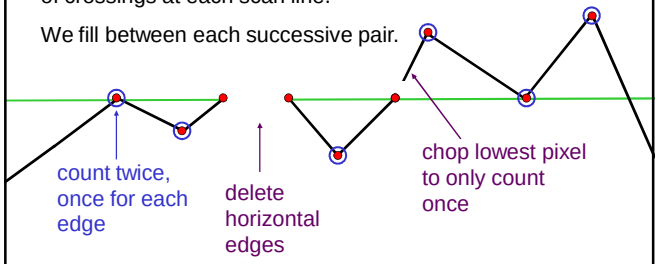
Edge Table (ET) has a list of edges for each scan line.



Preprocessing the edges

For a closed polygon, there should be an even number of crossings at each scan line.

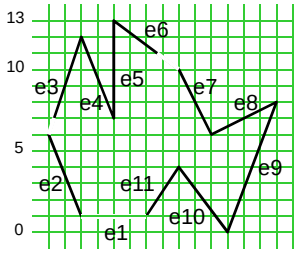
We fill between each successive pair.



Polygon Data Structure
after preprocessing

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

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



Algorithm

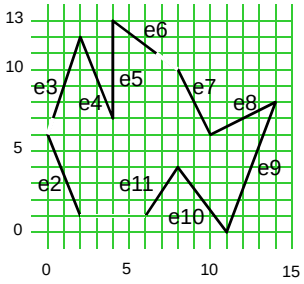
1 Build edge table
  Initialise scan-line list
2 Find first non-empty bucket **(ymin)** in the edge table
3 While the current **ymin** not greater than topmost y coordinate of the polygon
  3.1 merge the edge table for the current **ymin** with scan-line list, sort on increasing **xmin**
  3.2 fill pixels between pairs of rounded **xmin**-s
  3.3 Remove from the scan-line list the edges whose **ymax** = current scanline
  3.4 increment **xmin** by the **increment** (1/m) for remaining edges in the scan list
  3.5 re-sort scan line list on increasing **xmin**
  3.6 increment current scanline **ymin** to give next scanline

ET

11 → e6
7 → e3 → e4 → e5
6 → e7 → e8
1 → e2 → e11
0 → e10 → 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



Running the Algorithm

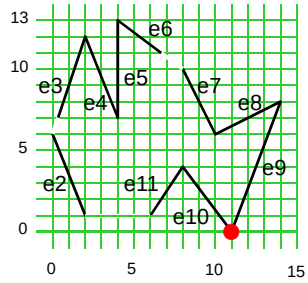
y=0

SCB→ | 10 1/4 | 4 | -3/4 | •

e10

e9

| 11 3/8 | 8 | 3/8 | • |



Running the Algorithm

y=1

SLB→ | 1 3/5 | 6 | -2/5 | •

e2

e11

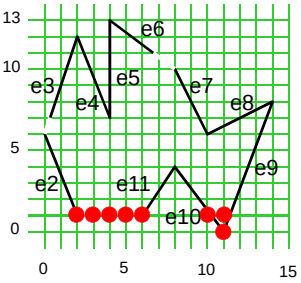
| 6 2/3 | 4 | 2/3 | •

e10

| 9 1/2 | 4 | -3/4 | •

e9

| 11 6/8 | 8 | 3/8 | •



Running the Algorithm

y=2

SLB→ | 1 1/5 | 6 | -2/5 | •

e2

e11

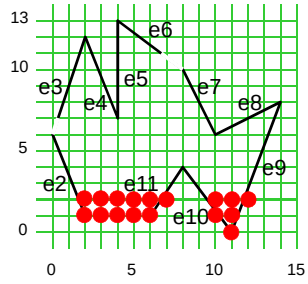
| 7 1/3 | 4 | 2/3 | •

e10

| 8 3/4 | 4 | -3/4 | •

e9

| 12 1/8 | 8 | 3/8 | •



Running the Algorithm

y=3

SLB→ | 4/5 | 6 | -2/5 | •

e2

e11

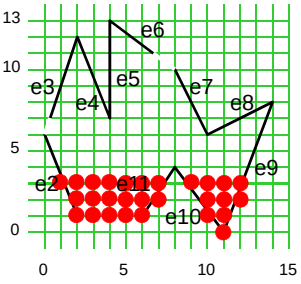
| 8 | 4 | 2/3 | •

e10

| 8 | 4 | -3/4 | •

e9

| 12 4/8 | 8 | 3/8 | •



Running the Algorithm

y=4

SLB→ | 4/5 | 6 | -2/5 | •

e2

e11

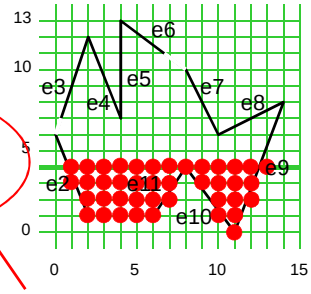
| 8 | 4 | 2/3 | •

e10

| 8 | 4 | -3/4 | •

e9

| 12 4/8 | 8 | 3/8 | •



Remove these edges.

Running the Algorithm

y=4

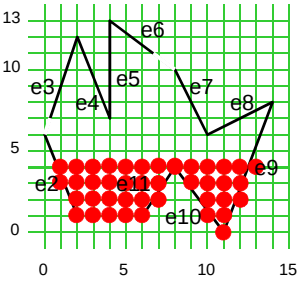
SLB→ | 2/5 | 6 | -2/5 | •

e2

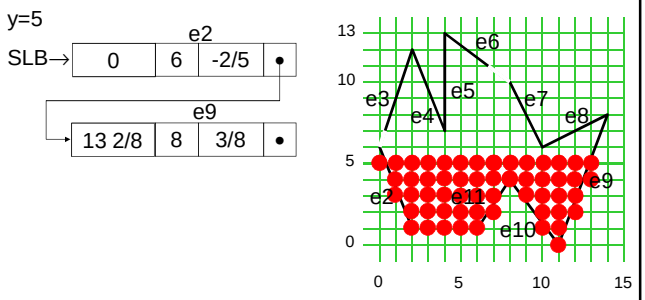
e9

| 12 7/8 | 8 | 3/8 | •

e11 and e10 are removed.

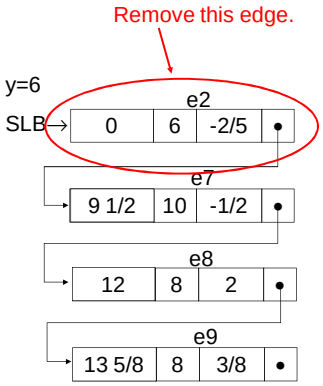


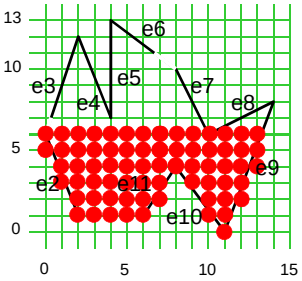
Running the Algorithm



Running the Algorithm

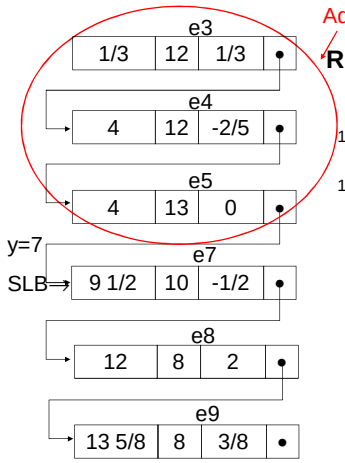
Remove this edge.

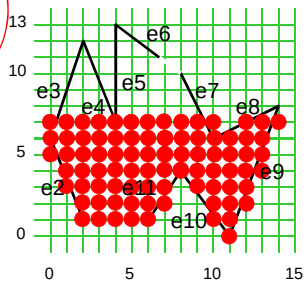




Add these edges.

Running the Algorithm





Scan line algorithm

- The algorithm provides a method for computing **locations** of individual pixels in a 2D display
  - E.g. hidden surface removal methods
- The pixel **colours** are computed using other algorithms, e.g.
  - Shading (flat, Gouraud, Phong)
  - Texturing
- The algorithm is efficient because it incorporates some coherence properties of the polygons (i.e. direction and slope of each edge)

Polygon display

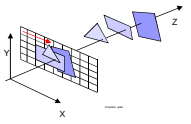
Flat shading

Phong shading

Hidden surface removal

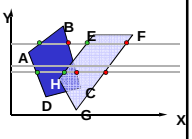
Z-buffer algorithm

```
for x increments, along each
scan line x+1, y):
   z' = z - A/C
for y increments,
for each new scan line (x, y+1):
   z" = z + B/C
```

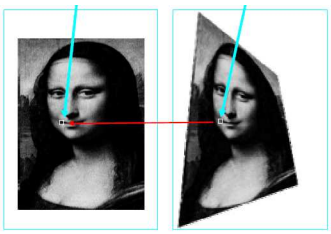


Scan-line method

•Each scan line is processed
•list of edges (of **ALL** polygons) crossing a current line sorted in order of increasing x
• ...



Texture mapping



Backward mapping algorithm:

for (r,c) = polygon pixel

$u = (a_{11}r+a_{12}c+a_{13})/(a_{31}r+a_{32}c+1)$

$v = (a_{21}r+a_{22}c+a_{23})/(a_{31}r+a_{32}c+1)$

copy pixel at source (u,v)
to destination (r,c)

Credits

The animations copied from Prof. Harriet Fell's lecture slides, College of Computer and Information Science, Northeastern University

Next lecture

Hidden surface removal