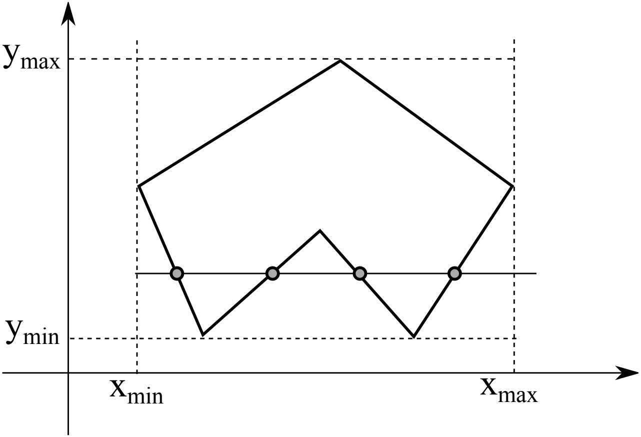
Computer Graphics (CS7.302)

Output Primitives

Raghavendra G S

Apr 16th, 2025

Scan Line Polygon fill

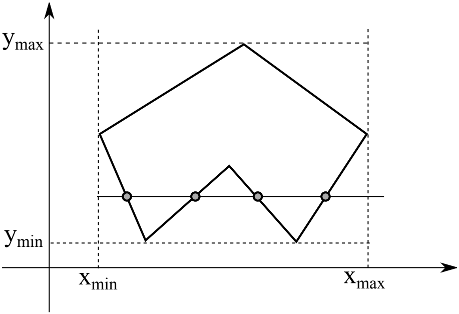


Scan Line Polygon fill

We we first see the steps and then the details

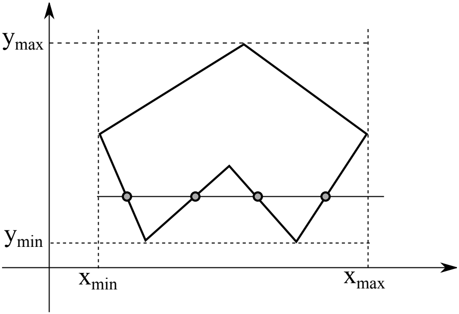
- find Bounding Box/minimum enclosed rectangle.
- number of scan lines $n = y_{max} - y_{min} + 1$
- for each scan line do
  - obtain intersection point of the scan line with the polygon edges.
  - sort intersections from left to right.
  - form intersection pairs from the list*
  - fill within pairs
  - intersection points are updated with each scan line
- stop when you reach y_{max}

Check if a point is within a polygon



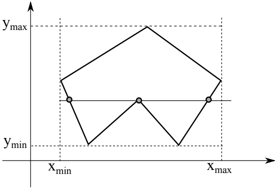
- Normal Case
    - check number of intersection points towards right/left
    - odd number means interior else exterior

Check if a point is within a polygon

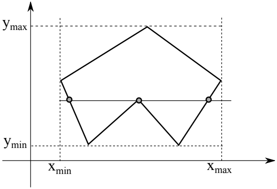


- Normal Case
  - check number of intersection points towards right/left
  - odd number means interior else exterior
- Special Case
  - if the scan line intersects a vertex.
  - two cases.

Case 1

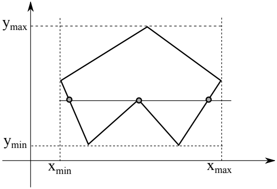


Case 1



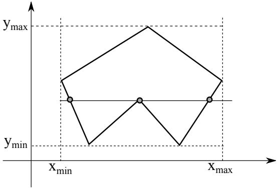
- two edges are meeting at the intersection point two.

Case 1



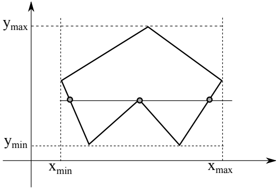
- two edges are meeting at the intersection point two.
- add one more intersection point there and increase the count.

Case 1



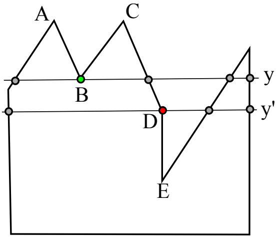
- two edges are meeting at the intersection point two.
- add one more intersection point there and increase the count.
- are we done?

Case 1

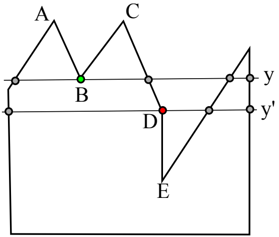


- two edges are meeting at the intersection point two.
- add one more intersection point there and increase the count.
- are we done?
  - no, there is one more case we need to consider.

Case 2

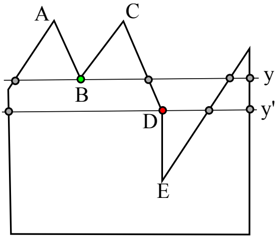


Case 2



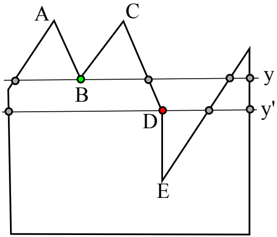
- for scanline y for green vertex we have case 1

Case 2



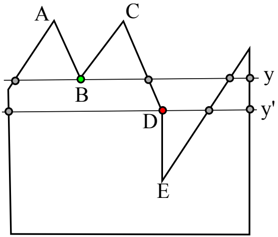
- for scanline y for green vertex we have case 1
  - note edges on same side of scanline

Case 2



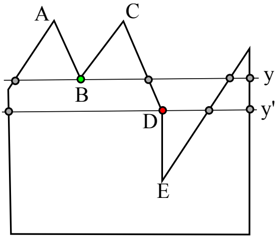
- for scanline y for green vertex we have case 1
  - note edges on same side of scanline
- for scanline y' we have red vertex which is case 2

Case 2



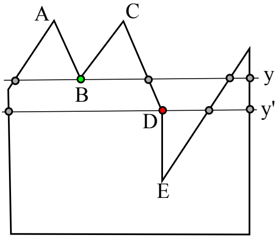
- for scanline y for green vertex we have case 1
  - note edges on same side of scanline
- for scanline y' we have red vertex which is case 2
  - shouldn't increment intersection points

Case 2



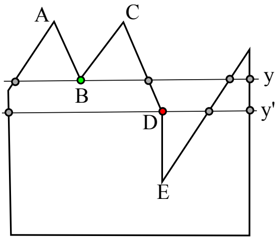
- for scanline y for green vertex we have case 1
  - note edges on same side of scanline
- for scanline y' we have red vertex which is case 2
  - shouldn't increment intersection points
  - note edges on different side of scanline

Case 2



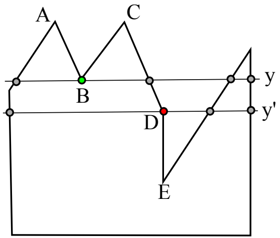
- To count vertices on a scanline

Case 2



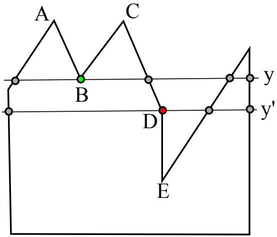
- To count vertices on a scanline
  - traverse along polygon boundary clockwise/counter-clockwise
  - observe relative change in y value of edges on either side of vertex

Case 2



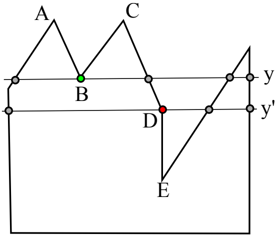
- To count vertices on a scanline
    - traverse along polygon boundary clockwise/counter-clockwise
    - observe relative change in y value of edges on either side of vertex
- If endpoint y-values of consecutive edges is monotonically increasing/decreasing

Case 2



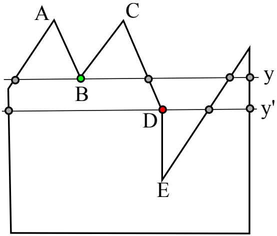
- To count vertices on a scanline
  - traverse along polygon boundary clockwise/counter-clockwise
  - observe relative change in y value of edges on either side of vertex
- If endpoint y-values of consecutive edges is monotonically increasing/decreasing
  - ⇒ case 2

Case 2



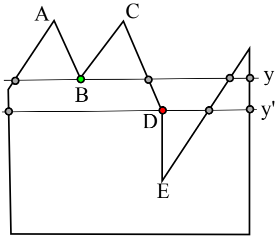
- To count vertices on a scanline
  - traverse along polygon boundary clockwise/counter-clockwise
  - observe relative change in y value of edges on either side of vertex
- If endpoint y-values of consecutive edges is monotonically increasing/decreasing
  - ⇒ case 2
  - don't increment intersection points

Case 2



- To count vertices on a scanline
  - traverse along polygon boundary clockwise/counter-clockwise
  - observe relative change in y value of edges on either side of vertex
- If endpoint y-values of consecutive edges is monotonically increasing/decreasing
  - ⇒ case 2
  - don't increment intersection points
- else vertex represents local extrema
  - ⇒ case 1

Case 2



- To count vertices on a scanline
  - traverse along polygon boundary clockwise/counter-clockwise
  - observe relative change in y value of edges on either side of vertex
- If endpoint y-values of consecutive edges is monotonically increasing/decreasing
  - ⇒ case 2
  - don't increment intersection points
- else vertex represents local extrema
  - ⇒ case 1
  - increment intersection points

Important features

Important features

Scanline coherence

Important features

Scanline coherence

- coverage/visibility of a face doesn't change much from one scanline to next

Important features

Scanline coherence

- coverage/visibility of a face doesn't change much from one scanline to next

Edge coherence

Important features

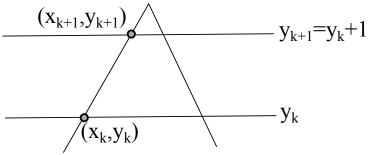
Scanline coherence

- coverage/visibility of a face doesn't change much from one scanline to next

Edge coherence

- edge intersected by scanline i is typically intersected by scanline $i+1$ too

Handling edge intersections



- $y_{k+1} = y_k + 1$
- $x_{k+1} = x_k + \frac{1}{m}$
- again we meet our familiar foe i.e. round off error.
- how to convert it to use only integer arithmetic?

Integer Arithmetic

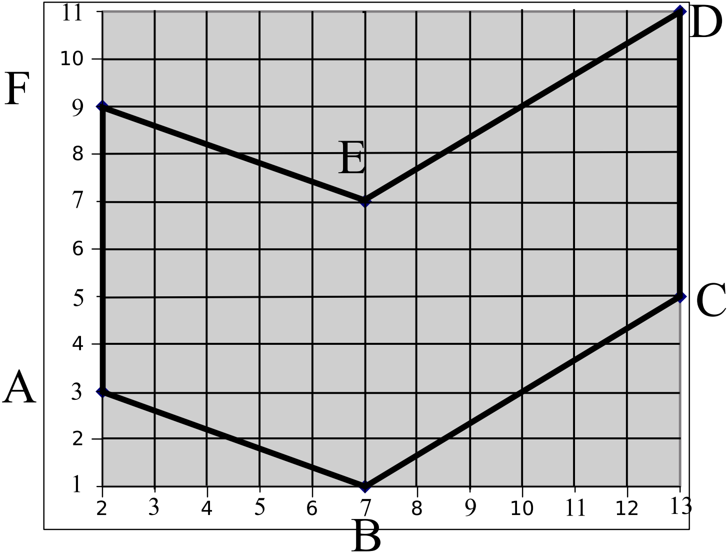
Lets take an example

- $m = \frac{\Delta y}{\Delta x} = \frac{7}{3}$
- let initial value of counter $C = 0$
- let counter increment $\Delta C = \Delta x = 3$
- for next successive scanlines $C = 3, 6, 9$
- third scanline $C > \Delta y$
- compute $C = C - \Delta y = 2$ and and increment x by 1.
- continue the same process till $y_k = y_{max}$.

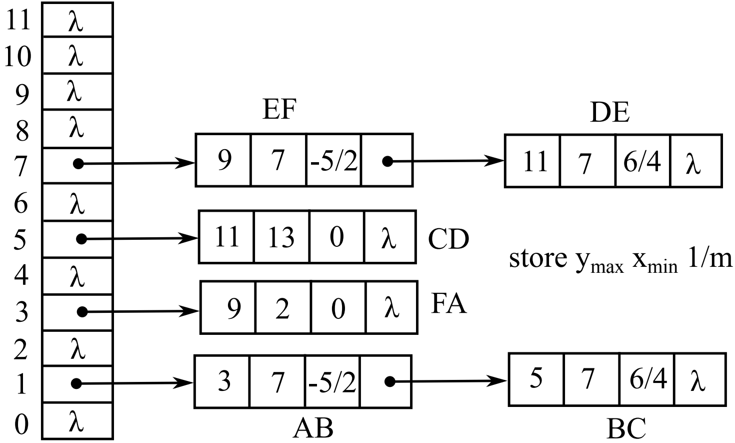
Data Structures used

- Sorted Edge Table (SET)
    - built using bucket sort where number of buckets equal to number of scanlines.
    - edges sorted by y_{min} with separate y bucket for each scanline.
    - within each bucket edges are sorted by increasing x of y_{min} point.
    - only non-horizontal edges are stored.
- Edge Structure stored for each edge in scanline.It contains
    - $y_{max}, x_{min}, \Delta x, \Delta y$ and pointer to next edge.
- Active Edge List/Table (AET)
    - contains all edges crossed by current scanline.
    - sorted by increasing x coordinates.

Example

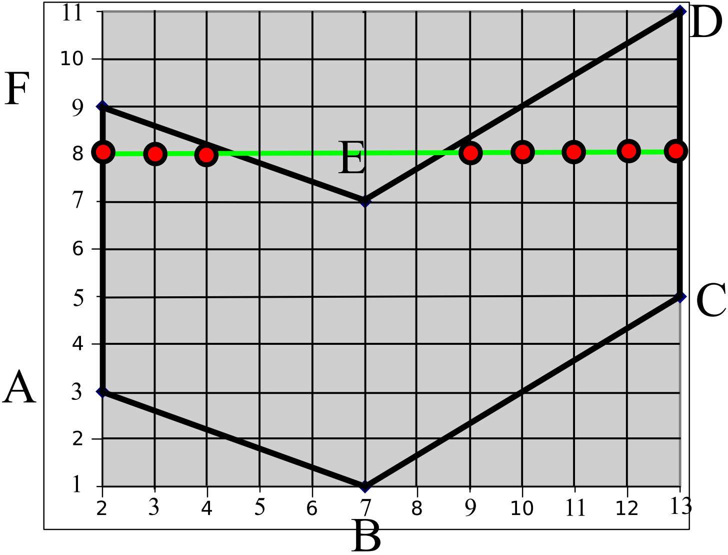


Bucket Sorted Edge Table for the example

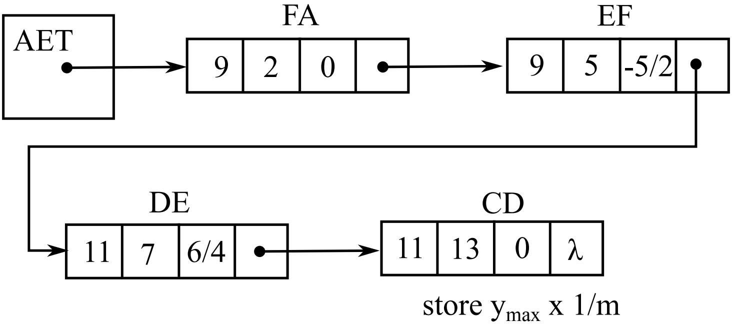


store y_{max} x_{min} $1/m$

example at Scanline 8

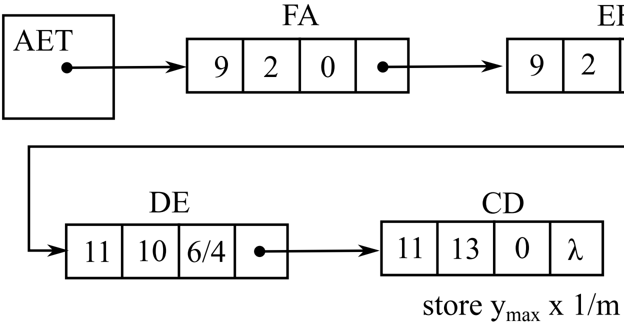


AET at Scanline 8

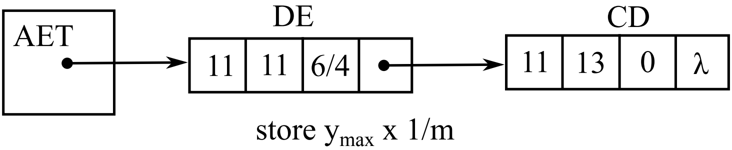


store y_{max} x 1/m

AET at Scanline 9



AET at Scanline 10



store y_{max} x 1/m

Processing Steps

- Set y to be smallest y or first non-empty bucket in SET entry
- initialize AET to be empty.
- repeat until both SET and AET are empty.
    - move from SET bucket y to AET edges whose $y_{min} = y$
    - sort AET on x (generally done before itself while in SET)
    - fill pixels on scanline using x intersection pairs from AET.
    - increment scanline by one.
    - remove edges from AET when $y = y_{max}$
    - for each non-vertical edge update x for new y.

Other Filling Algorithms

- Boundary Filling
- Flood Filling

Boundary Filling

- Start with an interior point
- Paint the interior outward towards the boundary
- Input is interior point (x, y)
- Test neighbouring positions to determine boundary color or not.
- If not paint them and test their neighbours.
- Continue till whole area is filled.
- Checking neighbours
  - Four-connected
  - Eight-connected

Algorithm

```
void boundaryFill4 (int x, int y, int fillColor, int borderColor)
   {
      int interiorColor;
      /* Set current color to fillColor, then perform following operations.
      */
      getPixel (x, y, interiorColor);
      if ((interiorColor != borderColor) && (interiorColor != fillColor))
      {
         setPixel (x, y);    // Set color of pixel to fillColor.
         boundaryFill4 (x + 1, y , fillColor, borderColor);
         boundaryFill4 (x - 1, y , fillColor, borderColor);
         boundaryFill4 (x , y + 1, fillColor, borderColor);
         boundaryFill4 (x , y - 1, fillColor, borderColor);
      }
   }
```

Boundary Filling

- Recursive algortihm might not fill correctly if by chance one interior pixel is already in fillcolor.
- Occurs because we check both fill color and boundary color.
- Stacking/unstacking might be costly.
- Process entire scanline instead of pixels.
  - **TODO:** show corresponding image.

Flood Filling

- We might want to color something which has different boundary colors.
- In such case we check only for interior color
- Again there are two variants
  - Four-connected
  - Eight-connected
- Again we can process entire scanline instead of pixels

Algorithm

```
void floodFill4 (int x, int y, int fillColor, int interiorColor)
   {
      int color;
      /* Set current color to fillColor, then perform following operations.
      */
      getPixel (x, y, color);
      if (color = interiorColor) {
         setPixel (x, y);    // Set color of pixel to fillColor.
         floodFill4 (x + 1, y, fillColor, interiorColor);
         floodFill4 (x - 1, y, fillColor, interiorColor);
         floodFill4 (x, y + 1, fillColor, interiorColor);
         floodFill4 (x, y - 1, fillColor, interiorColor);
      }
   }
```

The End