CS335 Fall 2007
Graphics and Multimedia

Polygons:
Representation and Scan Conversion

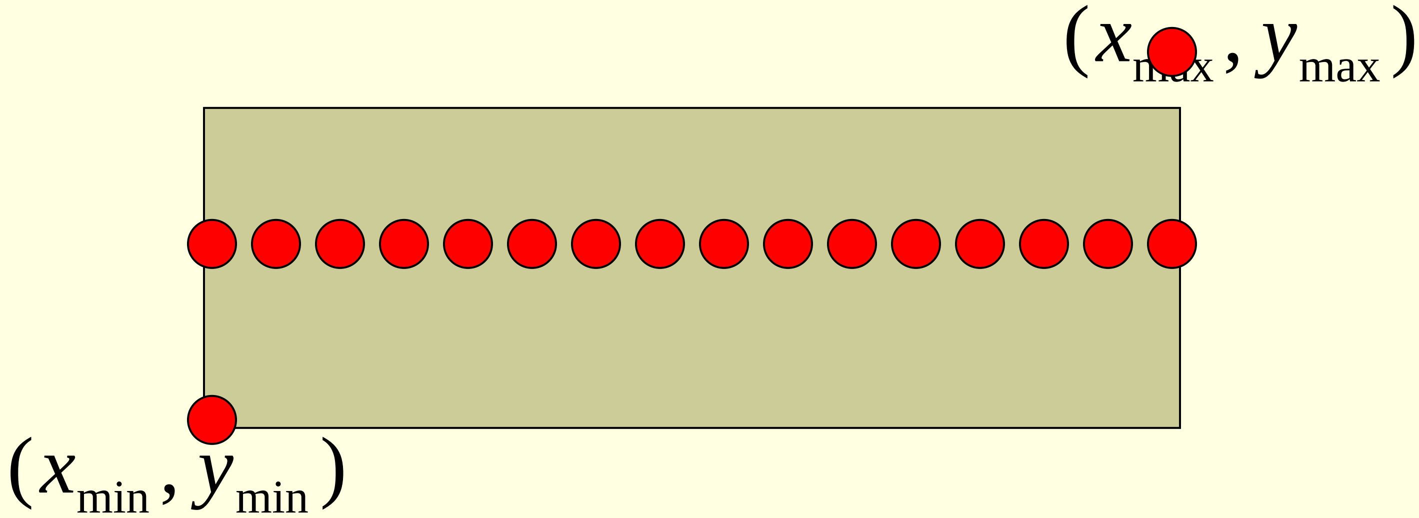
Scan Converting Regions

Basic algorithm (rectangles):

For $y = y_{\min}$ to $y_{\max}$

For $x = x_{\min}$ to $x_{\max}$

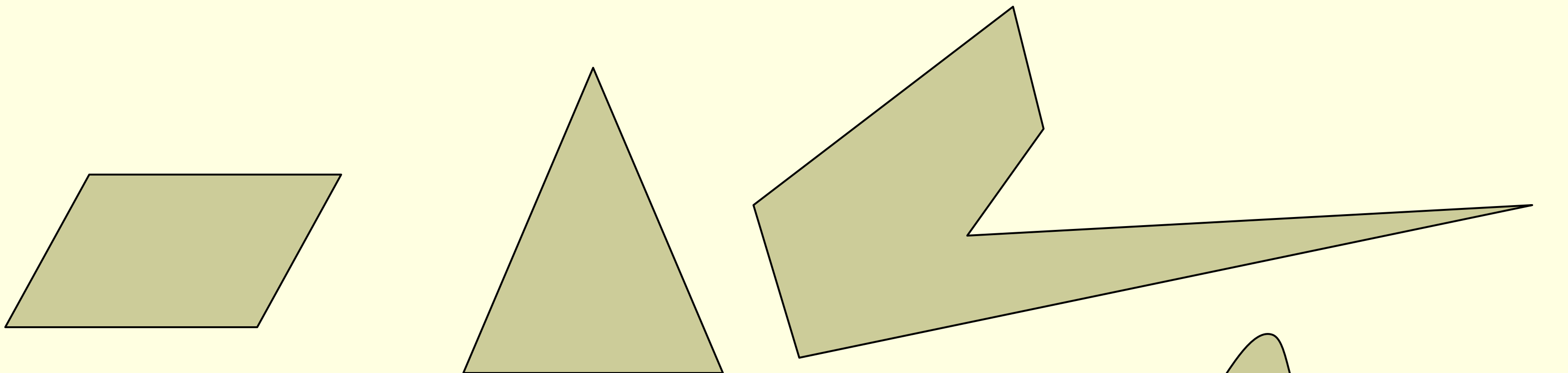
DrawPixel (x, y, color)



Vocabulary

- Spatial Coherence
  - values do not change or change slowly with changes in x and y
- Scan-line Coherence
  - adjacent scan lines are similar
- Edge Coherence
  - Edges of polygon don’t change rapidly with changes in x and y
- Temporal Coherence
  - Small changes with respect to time

Defining Polygons

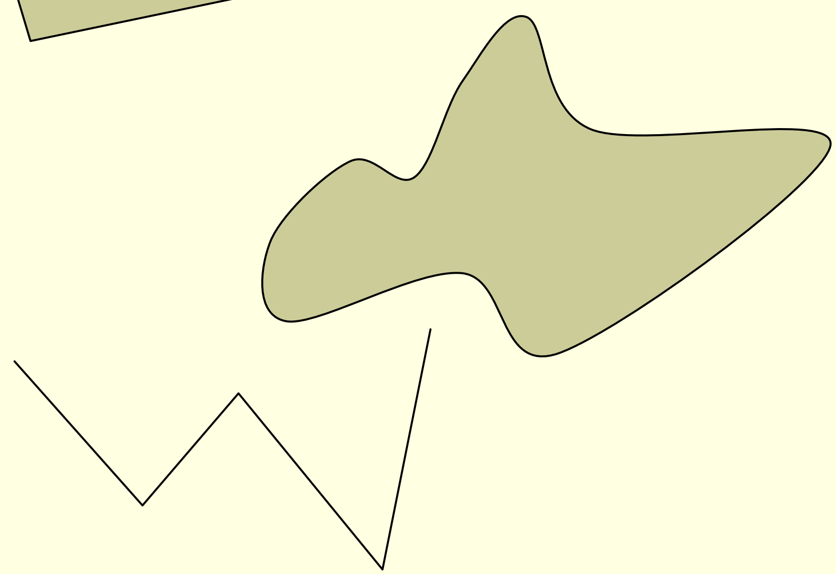


Polygon:

Closed curve

Linear edges

Ordered vertices



Representing Polygons

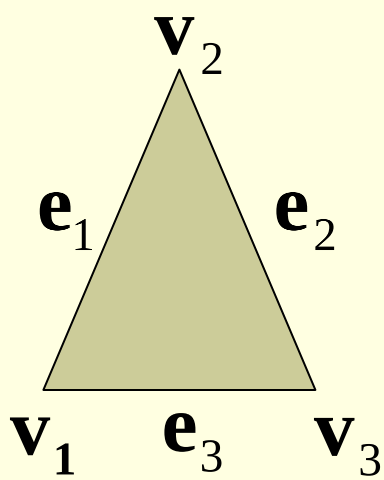
Ordered vertex/edge list:

$$\mathbf{v}_i = (x_i, y_i)$$

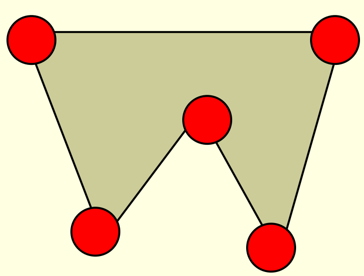
$$\mathbf{P}_{\mathbf{triangle}} = (\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}) \qquad \mathbf{e_i} = (\mathbf{v_i}, \mathbf{v_j})$$

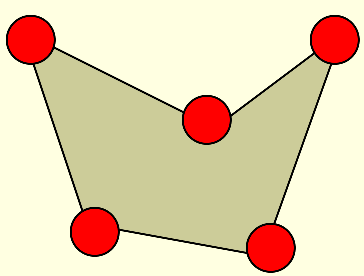
$$\mathbf{P}_{\mathbf{triangle}} = (\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3})$$

$$\mathbf{P}_{\mathbf{triangle}} = ((x_1, y_1), (x_2, y_2), (x_3, y_3))$$

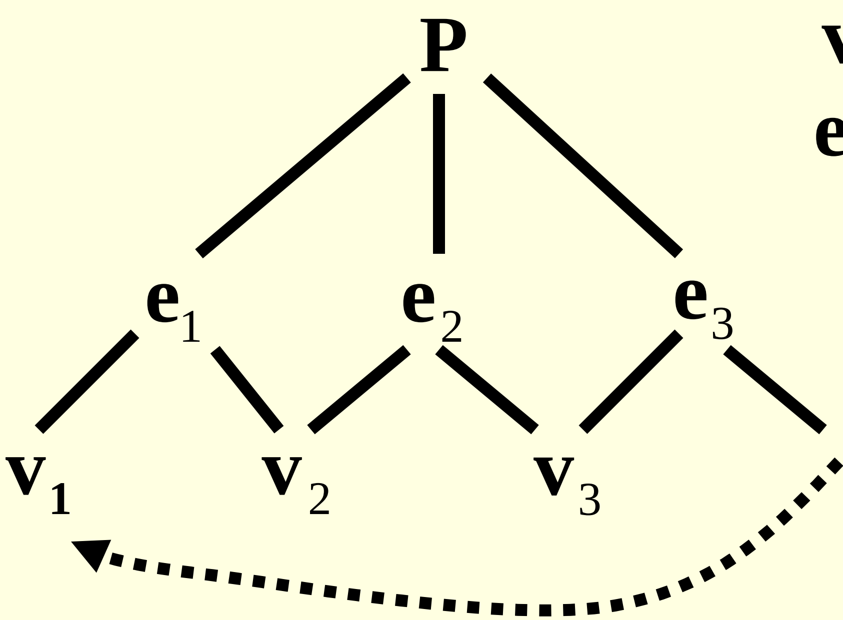


Order is important:

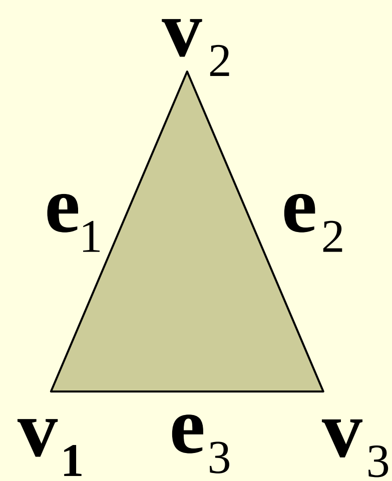




Winged-Edge Polygons

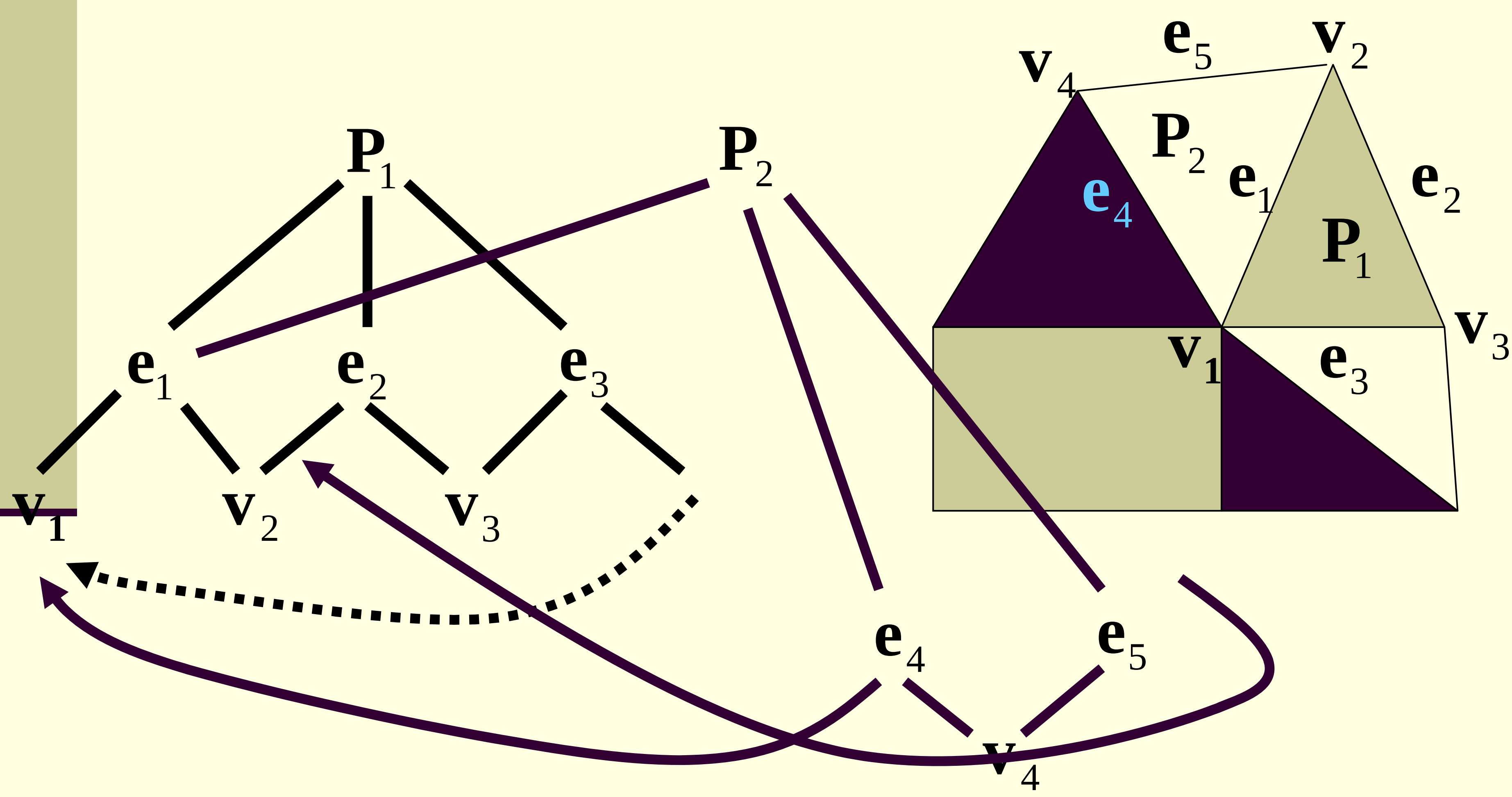


$$\mathbf{v}_i = (x_i, y_i)$$
$$\mathbf{e}_\mathbf{i} = (\mathbf{v}_\mathbf{i}, \mathbf{v}_\mathbf{j})$$



- Stores adjacency info
- Does not replicate data

Winged-Edge Polygons



Scan-Converting Polygons

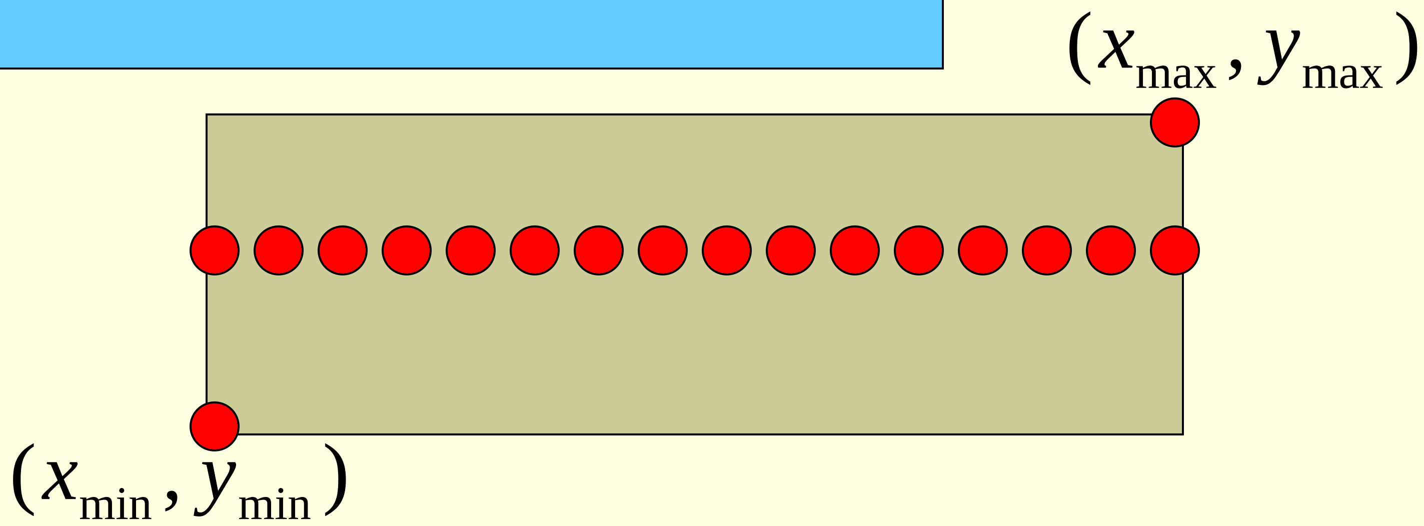
- Allowable polygons:
  - Convex
  - Concave
  - Self-intersecting
- Approach:
  - Calculate extrema of each scan-line span
  - Extrema come from intersection of scan-line with polygon
- Key Features:
  - Scan-line coherence yields incremental algorithm

Example: Rectangle

For $y = y_{\min}$ to $y_{\max}$

  For $x = x_{\min}$ to $x_{\max}$

    DrawPixel (x, y, color)



Scan-line coherence: extrema are always the same

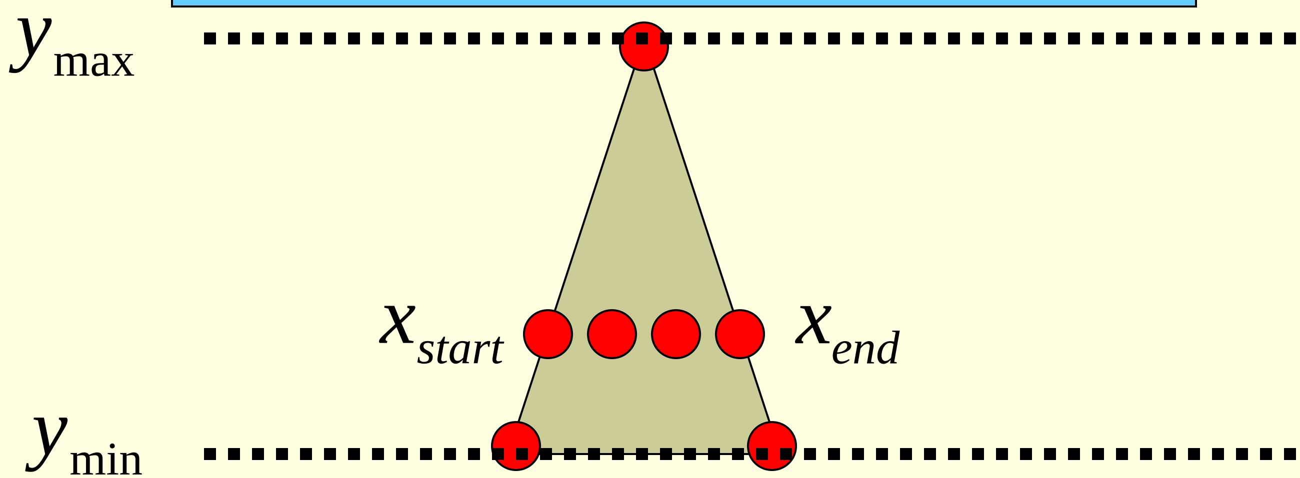
Example: Triangle

For $y = y_{\min}$ to $y_{\max}$

Compute x_{start} and x_{end}

For $x = x_{start}$ to x_{end}

DrawPixel (x, y, color)

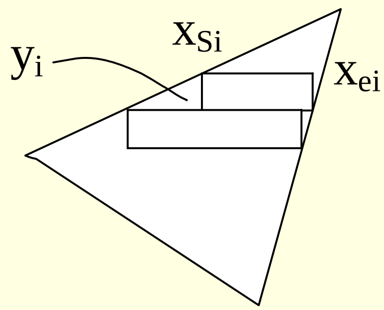


Scan-line coherence: extrema change slowly

Triangle Fill Scan Conversion

Algorithm Input : Triangle end points (x_1,y_1), (x_2,y_2), (x_3,y_3)

Algorithm Output : List of horizontal line segments indexed by scan line



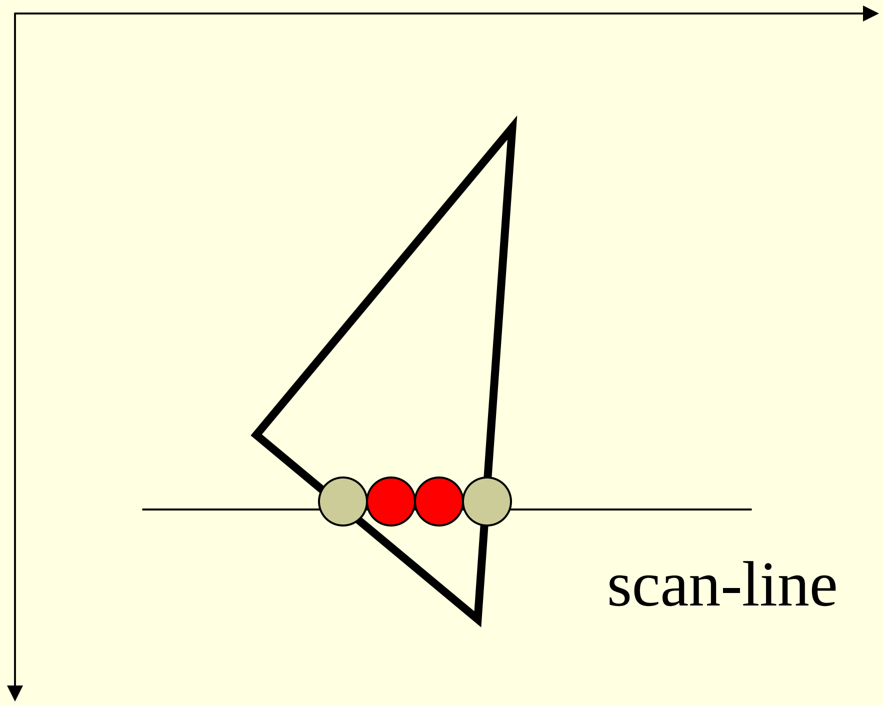
$$\{(y_i, x_{s_i}, x_{e_i})\}$$

Motivation :

Just as we approximate curves by small straight lines, we often approximate areas and surfaces by small triangles

All polygons can be decomposed into triangles if you work hard enough ! (computational geometry !)

Fill Algorithm

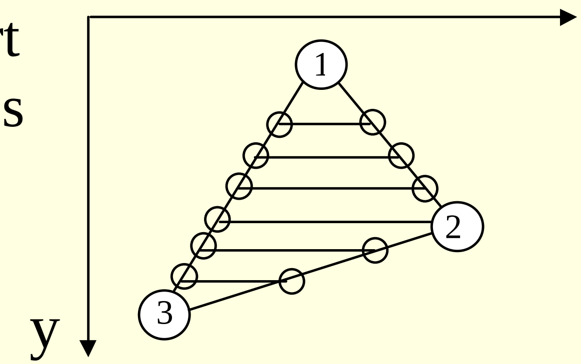


Idea: step an intersection ray α from low y-values to high values (vertical axis) and maintain a list of intersection points

Basic Idea

Sort (x_1,y_1), (x_2,y_2), (x_3,y_3) such that $y_1 \le y_2 \le y_3$

Order distinct lines on y-axis, convert y coordinate to discrete integer values

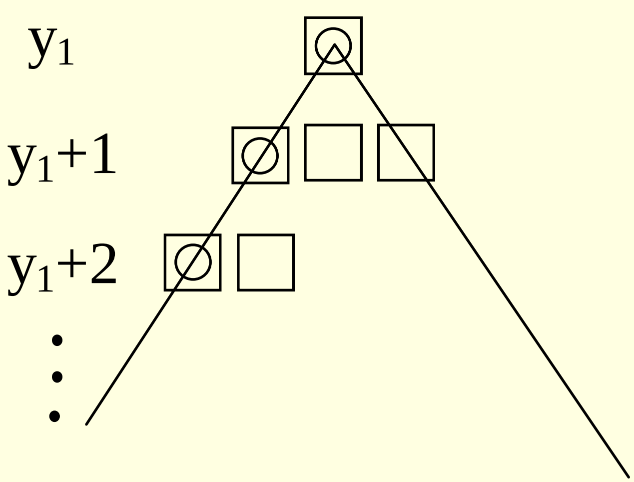


Equation of line :

$$y_k = y_1 + k$$

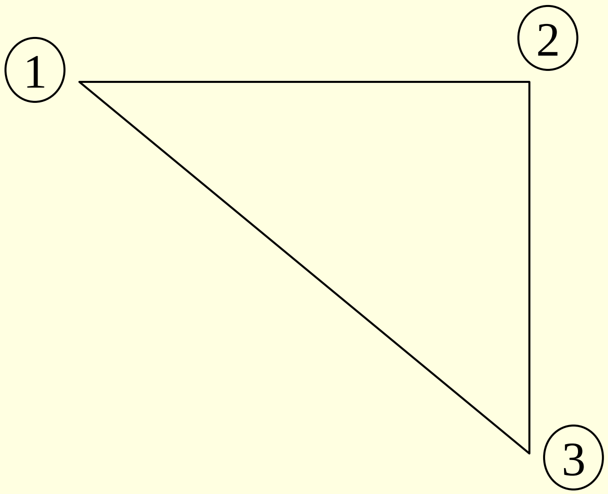
$$x_k = x_1 + \frac{x_2 - x_1}{y_2 - y_1} k$$

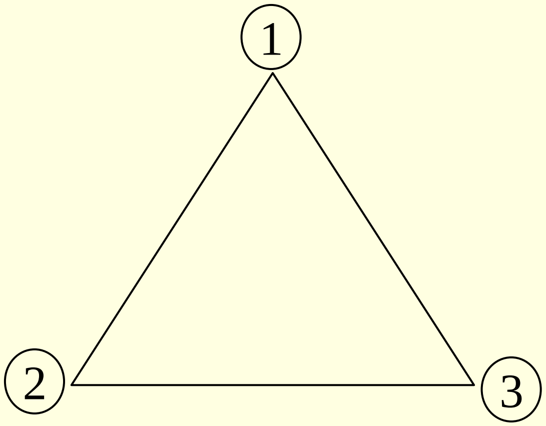
$$= x_1 + \frac{1}{m} k$$



give change in x per unit change in y.

Special Cases

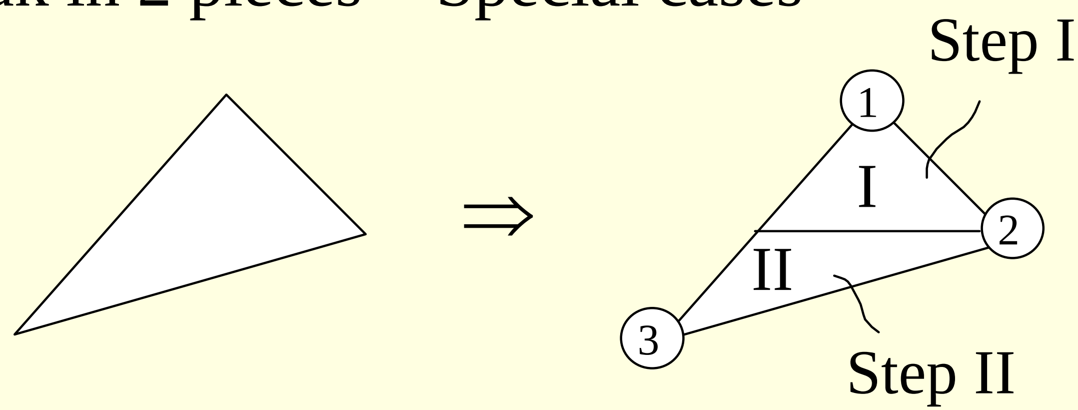




Careful ! $\Delta y = 0$, $m = \infty$

Techniques

1. Sort 1,2,3
2. Break in 2 pieces = Special cases



<u>Special Cases</u> :

$y_1 = y_2$ no step I

$y_2 = y_3$ no step II

Techniques (con't)

3. Ignore direction if we use line primitive to fill the scan line in either direction

4. 3 equations : as long as $y_i = mx + b$

we have

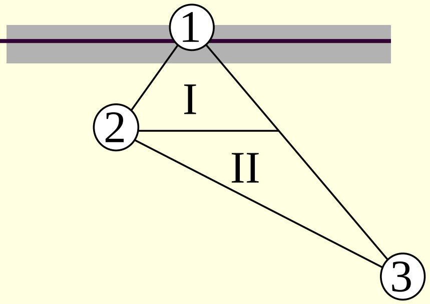
$$\Delta y = 1 \qquad \Rightarrow \qquad \Delta x = \frac{\Delta y}{m} = \frac{1}{m}$$

Since

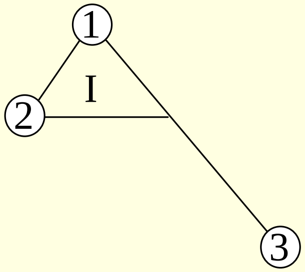
$$\frac{1}{m} = \frac{\Delta x}{\Delta y}$$

so we are OK if we handle $\Delta y = 0$ separately

Triangle Fill Procedure (Rendering)



1. Sort endpoints in y such that $y_1 \le y_2 \le y_3$



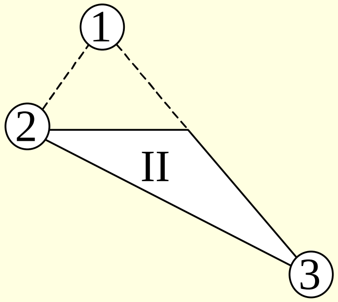
2. Unless $y_1 = y_2$, do part I , that is,

loop :

step $y = y_1, \dots, y_2 - 1$

use 2 slopes :

$$m_{2,1}^{-1} = \frac{x_2 - x_1}{y_2 - y_1} \quad \text{and} \quad m_{3,1}^{-1} = \frac{x_3 - x_1}{y_3 - y_1}$$



Compute :

$$X_{start} = X_1 + k(m_{2,1})^{-1}$$

$$X_{end} = X_1 + k(m_{3,1})^{-1}$$

Triangle Fill Procedure (con't)

3. After $y = y_2 - 1$, update X_{start} and X_{end} change slope :

$$m_{2,1}^{-1} \quad \rightarrow \quad m_{2,3}^{-1} = \frac{x_3 - x_2}{y_3 - y_2}$$

4. Continue part II

loop step $y = y_2, \ldots, y_3$,

$$X_{start} = X_2 + k(m_{2,3})^{-1}$$

$$X_{end} = X_1 + k(m_{3,1})^{-1}$$

Triangle Fill Procedure (con't)

5. Special cases :

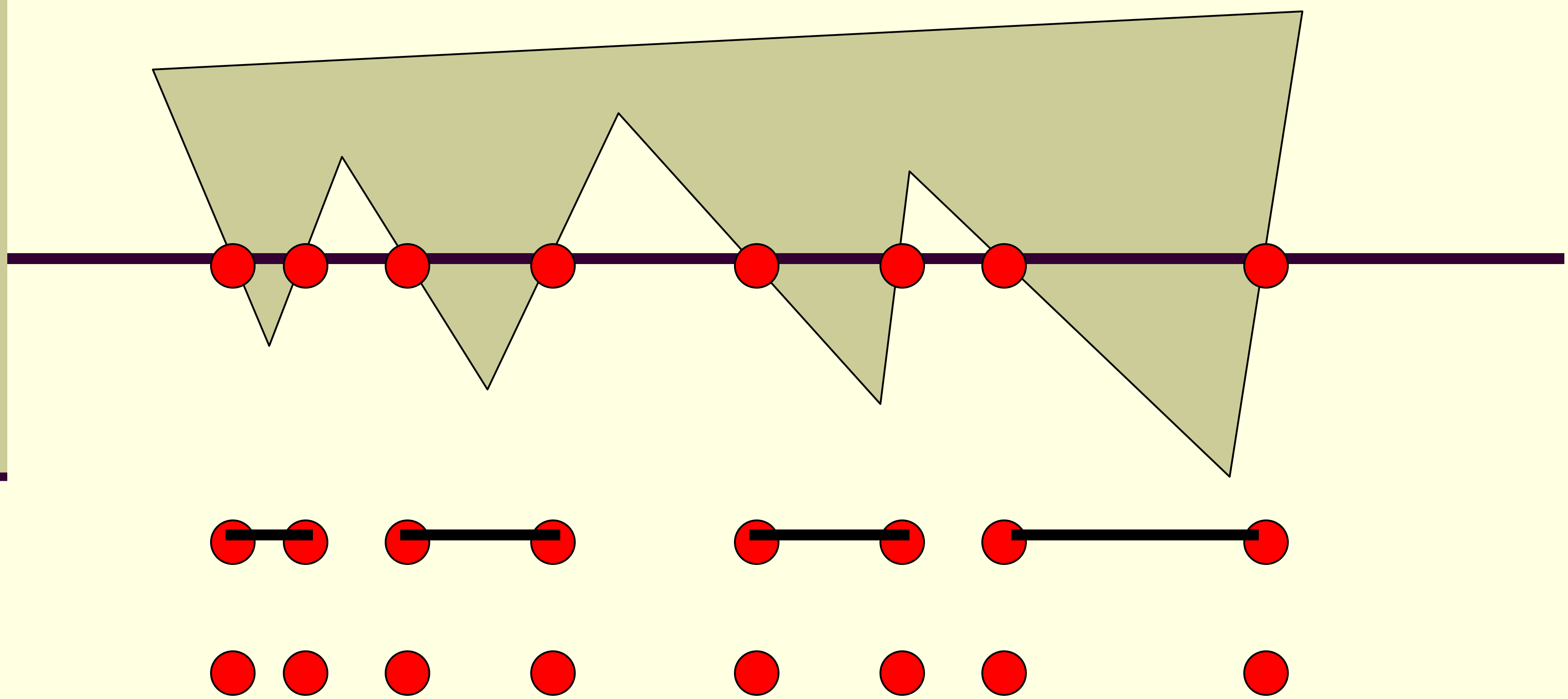
   if $y_1 = y_2$, skip to here and do part II

   if $y_2 = y_3$, stop here before starting II (draw at $y = y_2$ first!)

Polygon Scan-Conversion

- Basic algorithm: For all Scan-lines
  - Find intersection of scan line with polygon
  - Do this **incrementally**
  - Draw pixels inside of polygon
- Special concerns:
  - Concave polygons - intersection is a set of segments, not just one segment
  - Edge intersections (vertices)

Concave Polygons

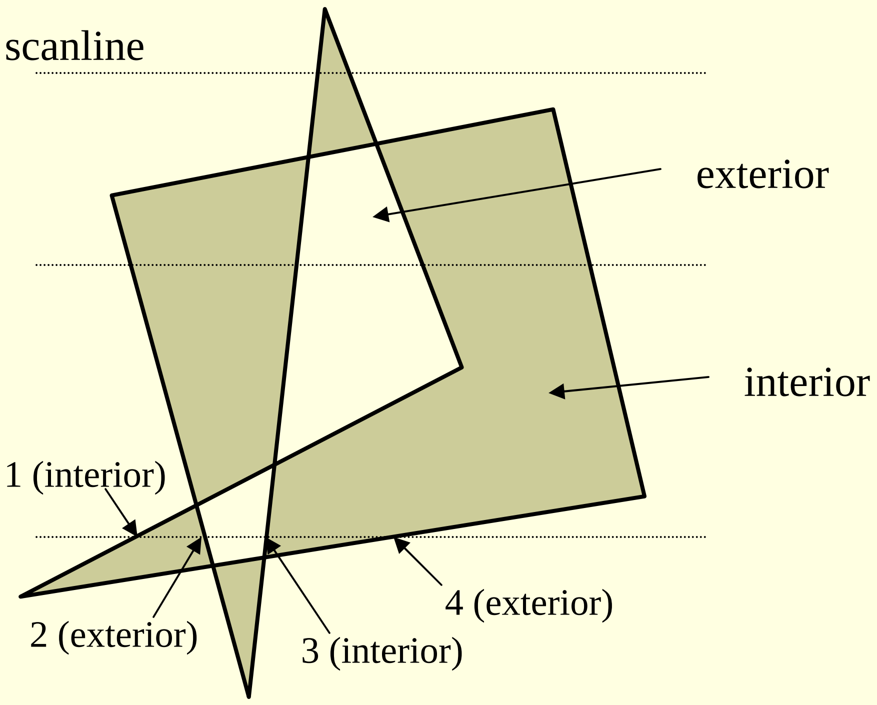


Inside or Outside

Count Intersections:

- Start ray outside polygon
- Count all intersections
- Inside/outside changes with each intersection

If the number of crossed edges is odd
Then we are in the interior



Problem points:

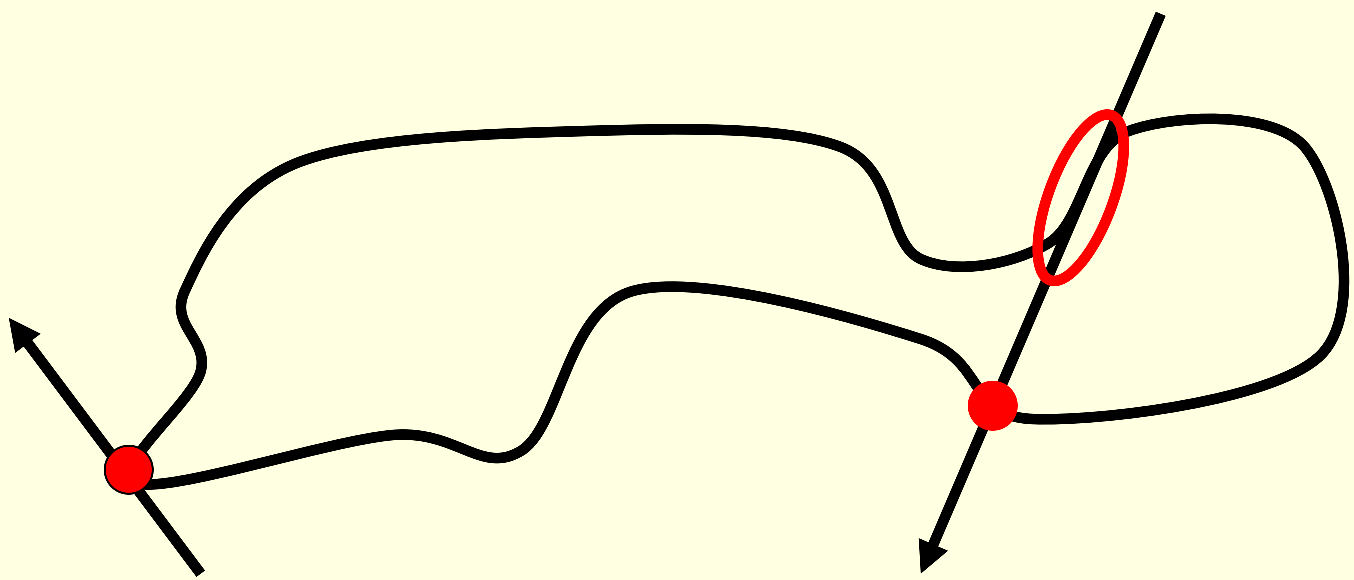
- Tangents
- Multiple intersections at vertices
- Starting ray outside

Steps for Scan Conversion

- Find intersections of scan-line with primitive
- Sort intersections in x coordinate
- Draw parts of scan-line that are inside primitive using an inside/outside counter

Special Cases to Manage

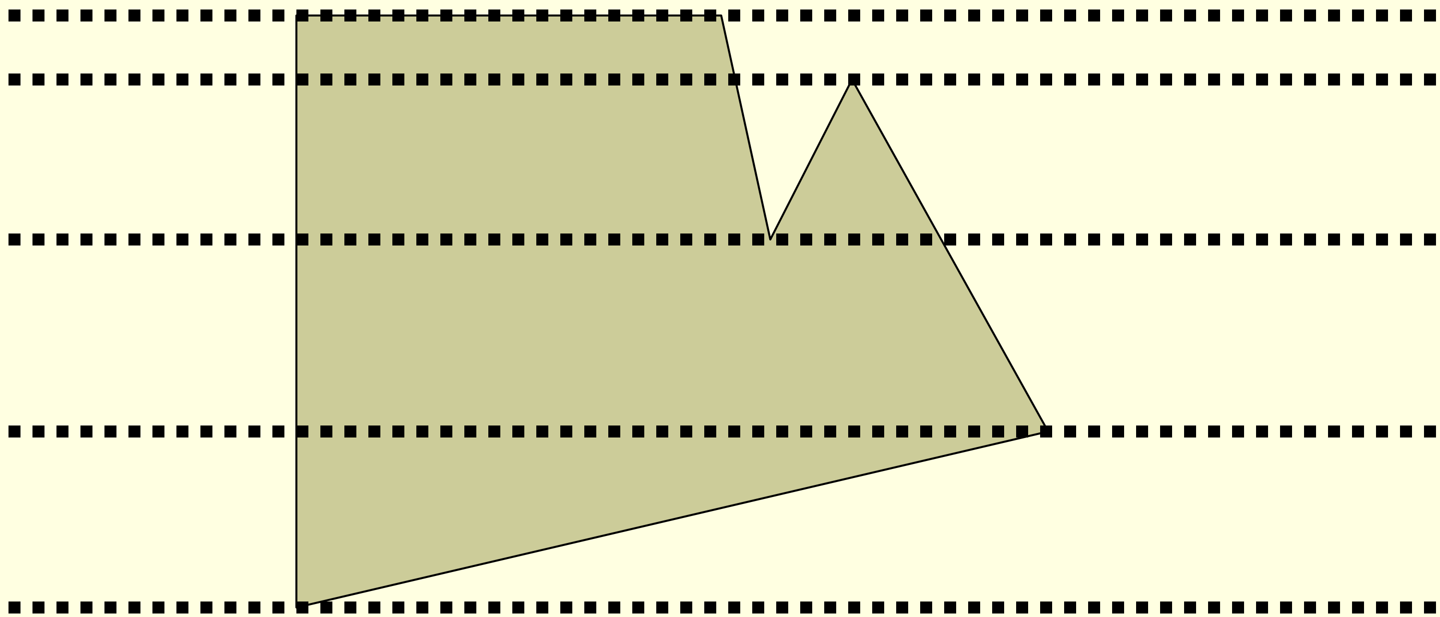
- Fractional intersection
- Intersection at a vertex
- Intersection with a horizontal edge



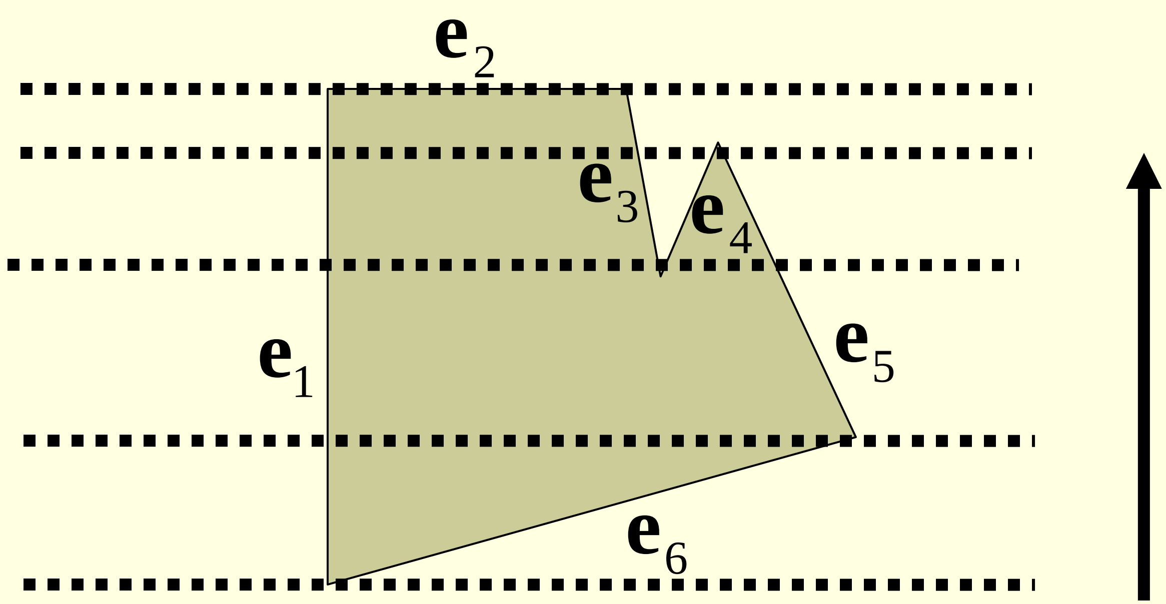
Conventions for Polygons

- Fractional Intersection: Inside moving right - round down. Outside moving right - round up
- Vertex intersection:
  - if a vertex is a minimum for an edge (lower vertex), count it as an intersection
  - if a vertex is a maximum for an edge (upper vertex), do not count it
- Horizontal edges: do not count vertices of horizontal edges in the inside/outside count

Example



GET: Global Edge Table



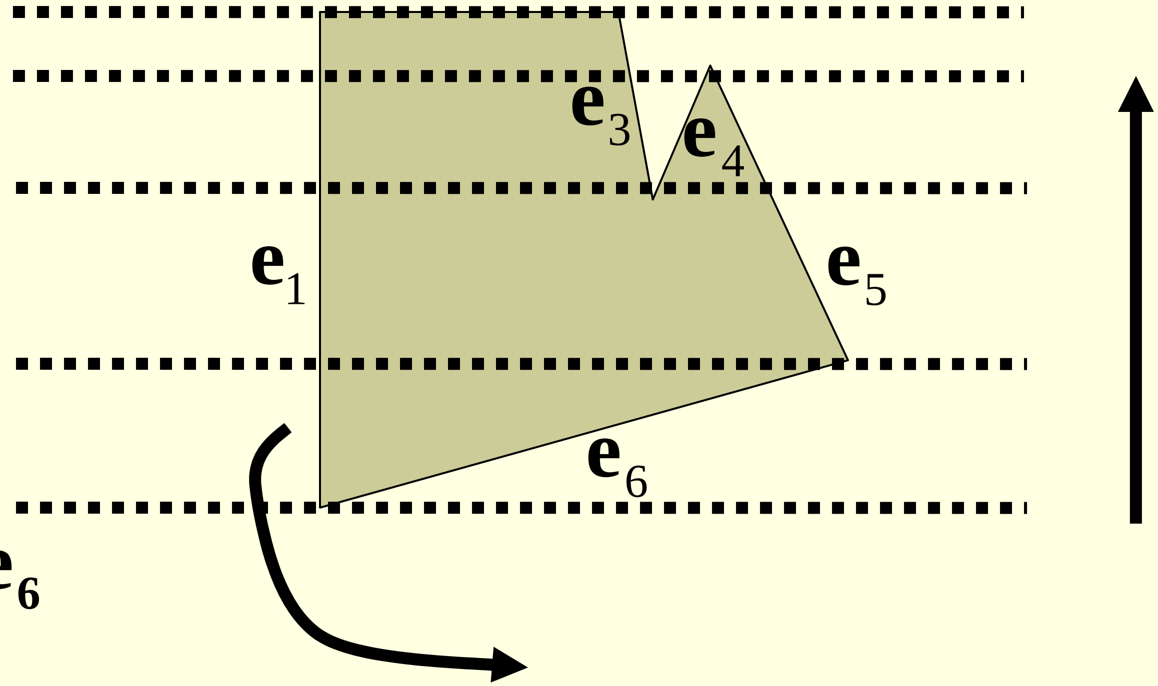
$\mathbf{e_1, e_6}$

$\mathbf{e}_5$

$\mathbf{e}_4, \mathbf{e}_3$

$\mathbf{e}_2$

AET: Active Edge Table



$\mathbf{e}_1, \mathbf{e}_6$

$\mathbf{e}_5$

$\mathbf{e}_4, \mathbf{e}_3$

$\mathbf{e}_2$

Using Edge Coherence

Goal: Compute intersection of scan-line with polygon edge as efficiently as possible

For scan-line i $\qquad x_{i+1} = x_i + \frac{1}{m}$

$$y = mx_i + b \Rightarrow x_i = \frac{y_i - b}{m}$$

$x_{i+1} = \dfrac{y_{i+1} - b}{m}$ but $y_{i+1} = y_i + 1$ so

$$x_{i+1} = \frac{y_i + 1 - b}{m} = \frac{y_i - b}{m} + \frac{1}{m} = x_i + \frac{1}{m}$$

Using Edge Coherence

$$m = \frac{y_{\max} - y_{\min}}{x_{\max} - x_{\min}} \Rightarrow \frac{1}{m} = \frac{x_{\max} - x_{\min}}{y_{\max} - y_{\min}}$$

Can maintain the intersection coordinate x-value as 3 integers: the integer x value, the numerator and denominator of 1/m

Each scan line causes the numerator to be incremented:

$$\frac{1}{m} + \frac{1}{m} = \frac{(x_{\max} - x_{\min}) + (x_{\max} - x_{\min})}{y_{\max} - y_{\min}}$$

Using Edge Coherence

When the numerator exceeds the denominator, add 1 to the x value and reset the numerator to (num-denom):

$$m = \tfrac{3}{2} \Rightarrow \tfrac{1}{m} = \tfrac{2}{3}$$

$$x_{\min} = 5$$

$$5, \quad 5\tfrac{2}{3}, \quad 5\tfrac{4}{3} \;=\; 6\tfrac{1}{3}$$

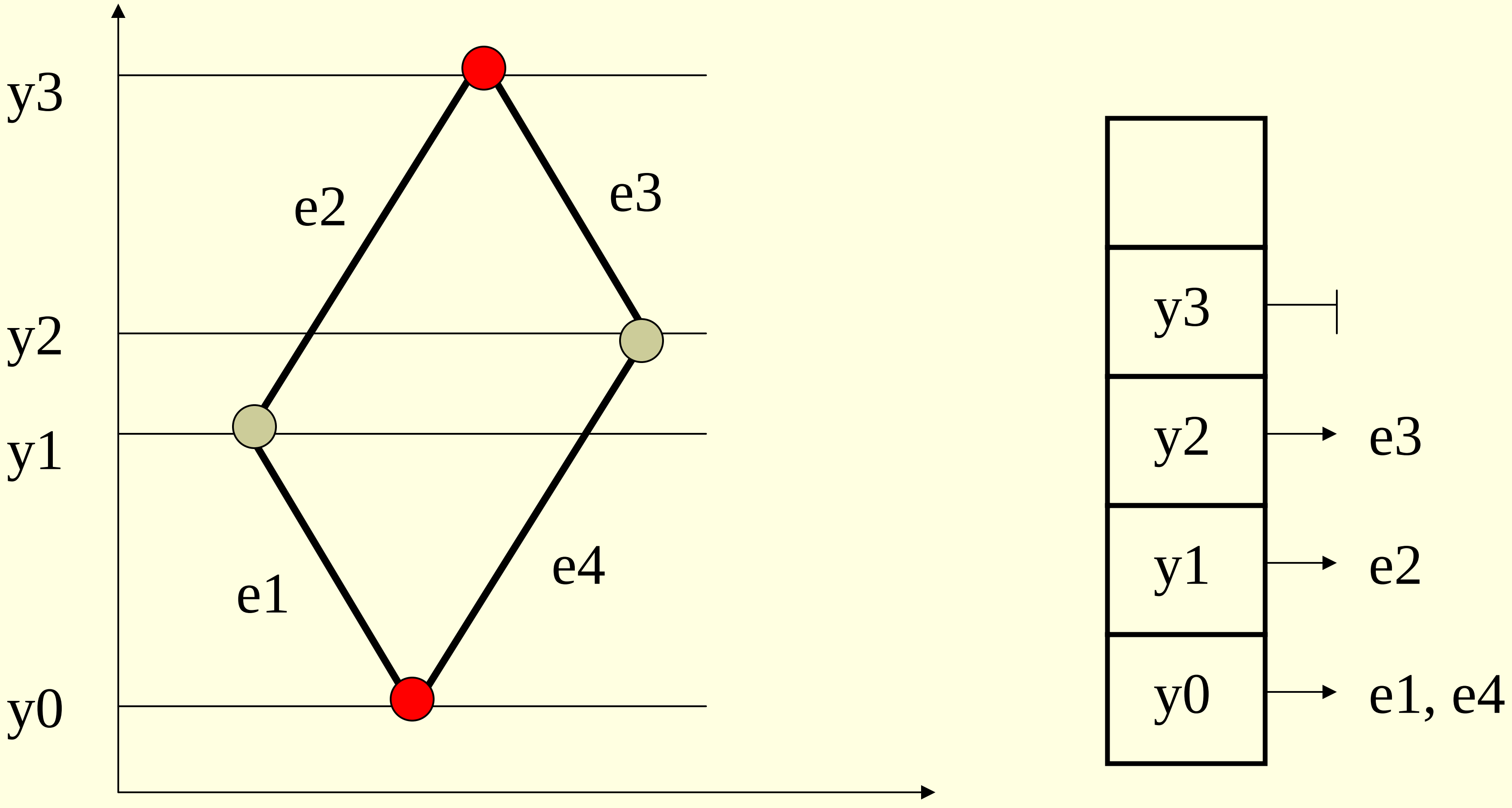
Scan Conversion Algorithm

- Make the Global Edge Table (GET)
- Set y=ymin from the GET
- Initialize the AET to empty
- Repeat until both the GET and AET are empty:

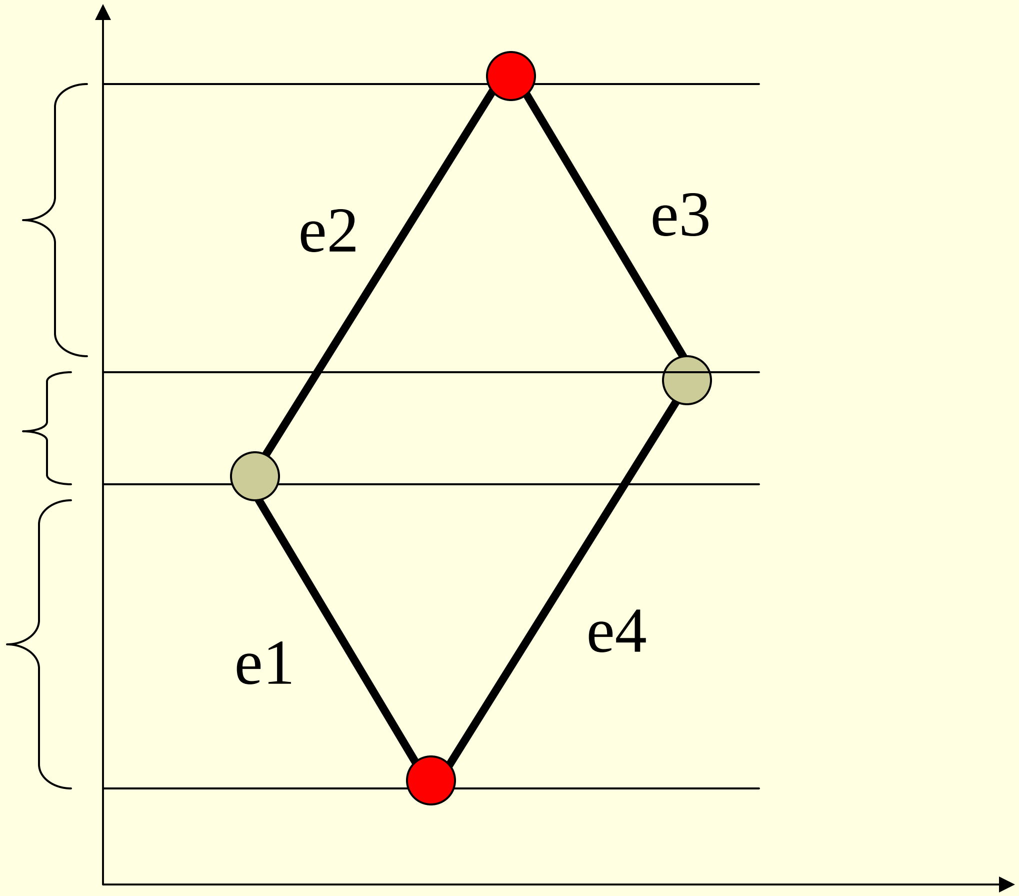
Scan Conversion Algorithm

- Move edges from GET to AET when ymin = y; maintain the AET sorted by x
- Scan convert scan line using inside/outside count and x coords in AET
- Delete from AET those edges that will not take part in the next scan line (when y = ymax)
- Move counter to next scan line (y++)
- Update next x value for each edge in AET
- Resort AET

Global Edge Table



Convex Polygon

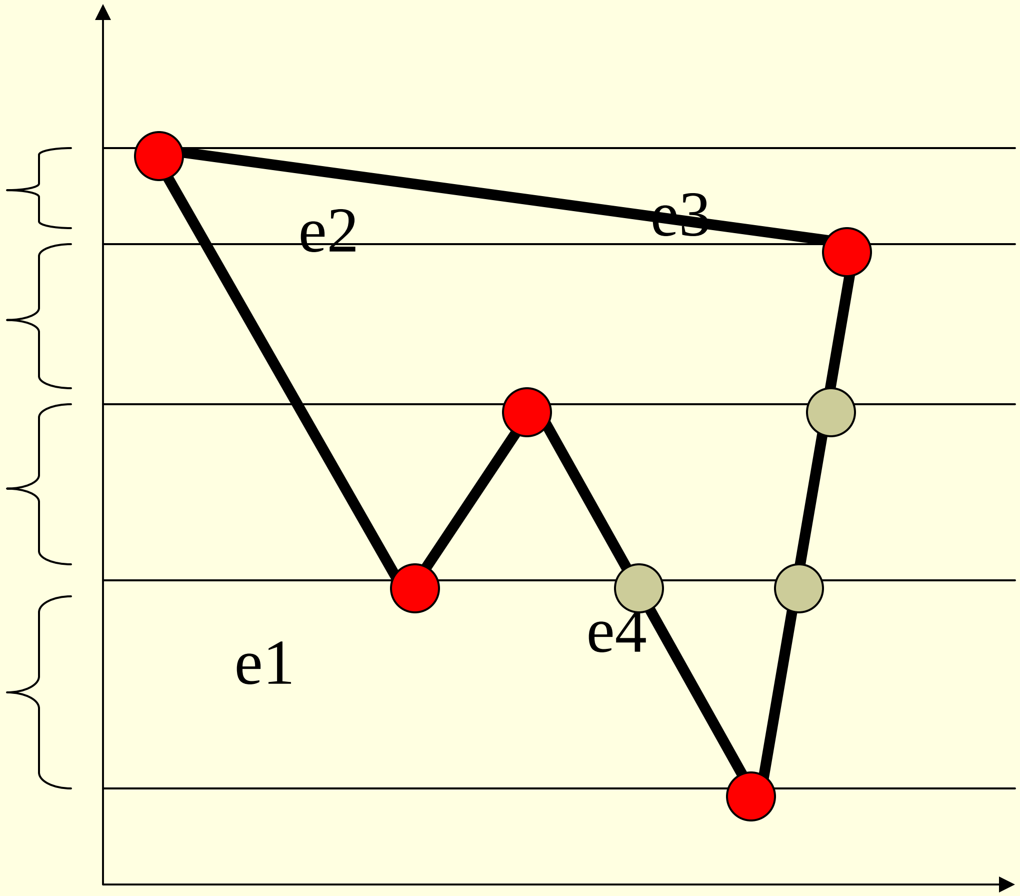


Region 1:

Region 2:

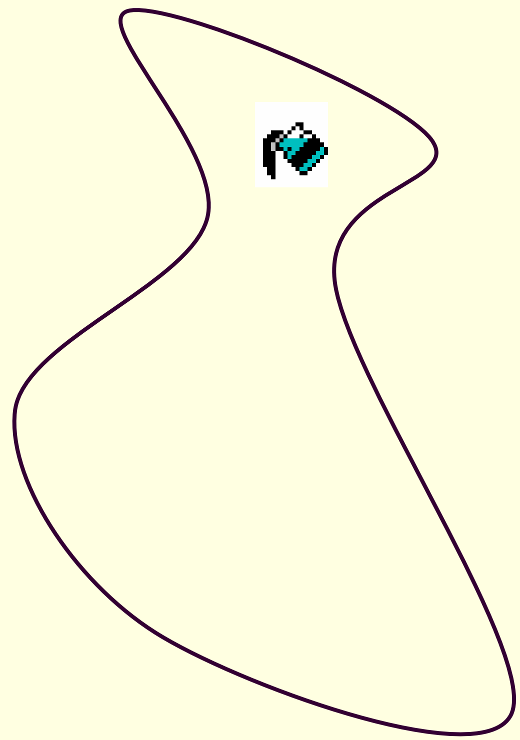
Region 3:

General (non-convex) Case



Concavities: Cause multiple segments to be drawn. This is when the inside/outside test becomes very important

Fill Algorithms



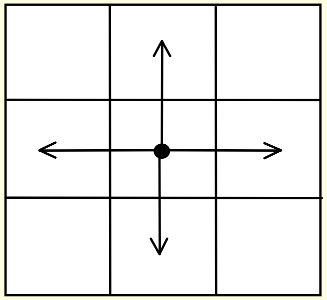
Fill in the border of a closed geometric primitive.

Must define what we mean by border.

[paint-brush demo]

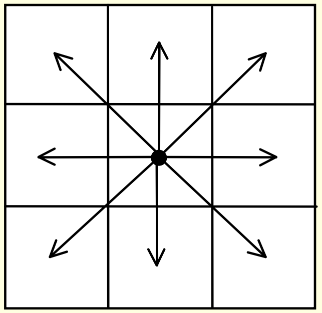
Connectivity

4 connected



$x, y \rightarrow$ $x\pm1, y$
$x, y\pm1$

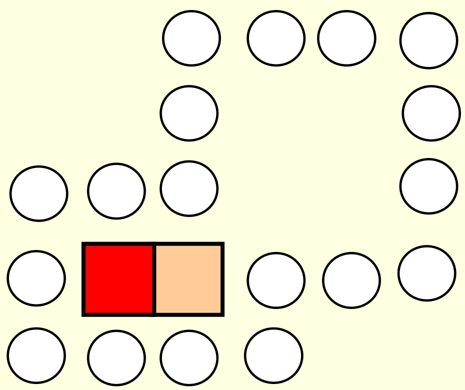
8 -connected



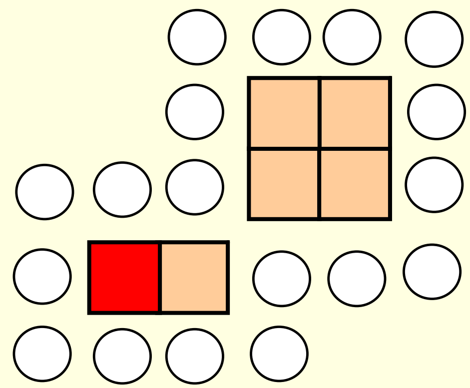
$(x, y) \rightarrow$ $x\pm1, y$
$x, y\pm1$
$x\pm1, y+1$
$x\pm1, y-1$

Examples

4-connect



8-connect



Note : every 4-connected region is also 8-connected.

4-Connected Boundary Fill Algorithm

Task : Given boundary, paint 4 connected interior

Input :

1. Raster area with boundary of 4-connected area, boundary pixel values are equal to B
2. Internal start point (x,y)
3. Flood fill value F

Output : Filled raster area, stopped when 4-connected to boundary.

Algorithm : 4B_Fill (x,y,F,B)

```
Function 4B_Fill (x,y,F,B)
    Value = get_pixel (x,y)
    If Value≠B and Value≠F
            set_pixel (x,y,F)
            4B_Fill (x+1,y,F,B)
            4B_Fill (x-1,y,F,B)
            4B_Fill (x,y+1,F,B)
            4B_Fill (x,y-1,F,B)
```

Note : Recursion is not necessarily a good computational strategy, unless compiler is very smart.

[use a stack structure]

Two types of fills

- Boundary Fill
  - It can fill the interior region with different colors, except the boundary color.
  - It requires a given boundary
- Flood Fill
  - Sometimes we want to fill a region that is ***not*** defined by a ***single*** boundary color
  - It fills region with the same interior color

Flood Fill of 4-Connected Region

Input :

1. Area pixels marked by value "A"
2. Seed (x,y)
3. Fill value "F"

Output : Raster with "A" replaced by "F"
whenever 4-connected to (x,y)

Algorithm : 4F_Fill (x,y,A,F)

```
Function 4F_Fill (x,y,A,F)
    Pixvalue = get_pixel (x,y)
    If  Pixvalue = A
        set_pixel (x,y,F)
        4F_Fill (x+1,y,A,F)
        4F_Fill (x-1,y,A,F)
        4F_Fill (x,y+1,A,F)
        4F_Fill (x,y-1,A,F)
```

Note : Both Boundary fill and Flood fill could iterative procedure rather than recursion.

8-Connected Versions

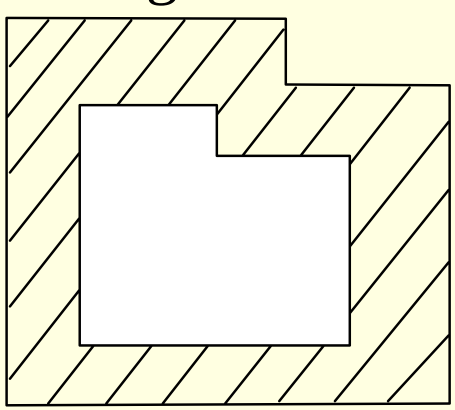
8-Boundary-Fill

Same procedure as 4-connected but add 4 more cases :

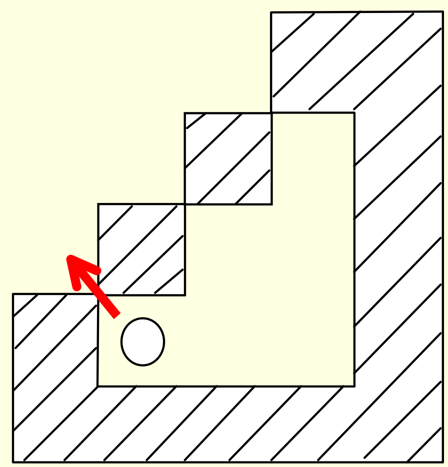
(x+1,y+1) , (x-1,y+1) , (x-1,y-1) , (x+1,y-1)

to the list of calls to 8B_Fill

No diagonal Boundary gaps allowed



OK



NO !
Bleeding out !

8-Flood Fill

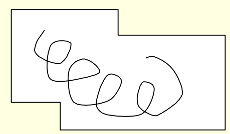
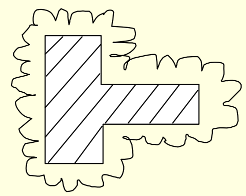
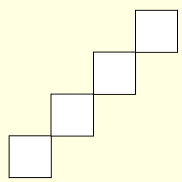
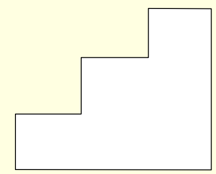
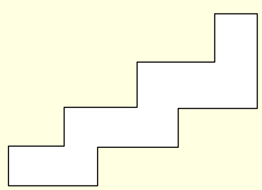
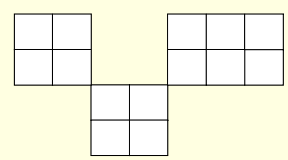
Same procedure but add 4 more cases :

(x±1 , y±1)

8 total calls to **8F_Fill**

Diagonally adjacent pixel groups will be joined :

Fill Summary

	Boundary Fill	**Flood Fill**
parameters	Seed x,y boundary color fill color	Seed x,y fill color get color at seed pixel
property	Ignore existing non-boundary colors 	Replace seed color, ignores boundary color 
4-connected	Bresenham stairstep boundary OK 	Fills “stairs” only 
8-connected	Must have no diagonal gaps in boundary 	Fills diagonally adjacent 

Summary

- Scan-conversion polygons
  - Triangles
  - Polygons
    - Inside/outside test
- Fill
  - Boundary Fill
  - Flood Fill
- Example exercises:
  - Hearn pp 140-141, ex3-2, 3-9, 3-19