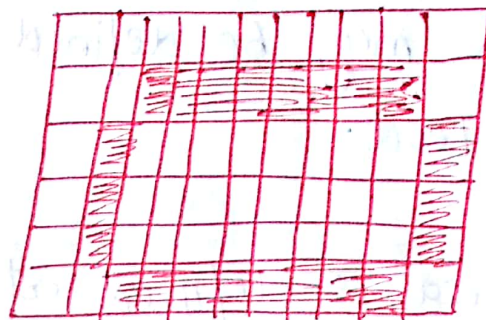


Polygon Filling

- Process of colouring the area of polygon.
- An area or region is defined as a collection of pixels.
- Area/region are of 2 types:-

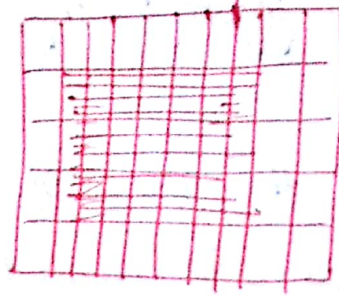
1) Boundary defined Region (BDR)

- Pixels that marks the boundary/outline of the region are called BDR pixels.
- Unique color not same as color of interior pixels.
- Algorithms that are used to fill the boundary defined region are called as Boundary fill alg.



2) Interior Defined Region

- Collection of same color continuous pixels.
- Pixels exterior to a region have different colors.
- Algorithms used to fill interior defined region are called Interior/Flood fill algorithms.



Both these algorithms are together called as

Seed-fill algorithms.

↓
(Becoz both start from a pixel within a region)
(Pixel where you start)

→ Principle of seed fill algorithms.

"Start at a sample pixel called as seed pixel from the area, fill color value and the boundary color value."

→ Area/Region may be defined at pixel level or Geometric level.

→ when the area is defined at pixel level, we have 4 different algorithms:-

- 1) Boundary fill
- 2) Flood "
- 3) Edge "
- 4) Fence "

→ when defined a geometric level know as scanline approach.

Boundary Fill Algorithms

- Recursive algorithm, begins with starting point inside the region. (seed)
- Algo checks whether this is a boundary pixel or has been filled already.
 - If answer is No, it fills pixel and make recursive call to itself using each & every neighbouring pixel as a new seed.
 - If answer is YES, algo. simply return to caller.

Adv:- works well for arbitrary shaped region

DisAdv:- Takes time & memory becoz of so many recursive calls.

```
void boundary Fill (int x, int y, int fill, int boundary)
{
    int current = get pixel(x, y)
    if (current != boundary && current != fill)
    {
        set color (fill);
        set pixel (x, y);
        boundary fill (x+1, y, fill, Boundary)
        boundary fill (x-1, y, fill, Boundary)
        boundary fill (x, y+1, fill, Boundary)
        boundary fill (x, y-1, fill, Boundary)
    }
}
```

Flood fill Algorithm

- Begins with seed inside the region.
- checks, if pixel has region's original color.

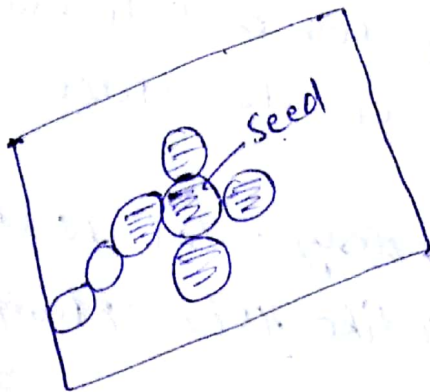
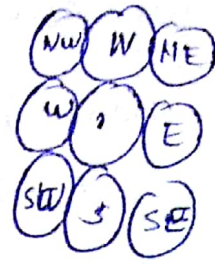
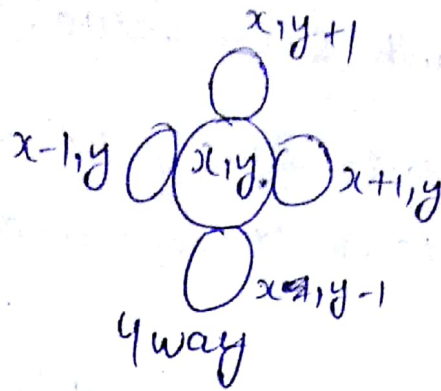
If yes :- it fills pixel with a new color & uses each of pixel's neighbour as a new seed in a recursive call.

If No :- it returns to caller.

Adv:- useful if region has non-uniform coloured boundary.

Disadv:- same as Boundary fill.

Boundary filling



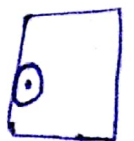
Boundary-fill (x, y , b -color, b -color)
 { seed coordinates
 new background (0) black

if (get pixel $(x, y) \neq b$ -color && get pixel $(x, y) \neq b$ -color)

[put pixel $(x, y, b$ -color)

Boundary-fill $(x+1, y, b$ -color, b -color)

" $(x, y+1, \text{" "})$
 " $(x-1, y, \text{" "})$
 " $(x, y-1, \text{" "})$



Flood-fill

- we start with seed value.
- then we inspect all 8 points surrounding the seed. If these points aren't on boundary, each will be filled.

→ Now, each of these points become seed. The algo continues to work in this fashion until all points surrounding all seeds enter into border.

→ Some cases where boundary color is diff. from fill color. For situations like this flood fill is used.

↳ Here instead of matching color with a boundary color, a specified color is matched.

→ flood-fill(x, y , old-color, new-color)

{
 putpixel(x, y , new-color)

flood fill ($x+1, y$, old-c, nc)

($x-1, y$,

($x, y+1$

($x, y-1$

($x+1, y+1$

($x-1, y-1$

($x+1, y-1$

($x-1, y+1$

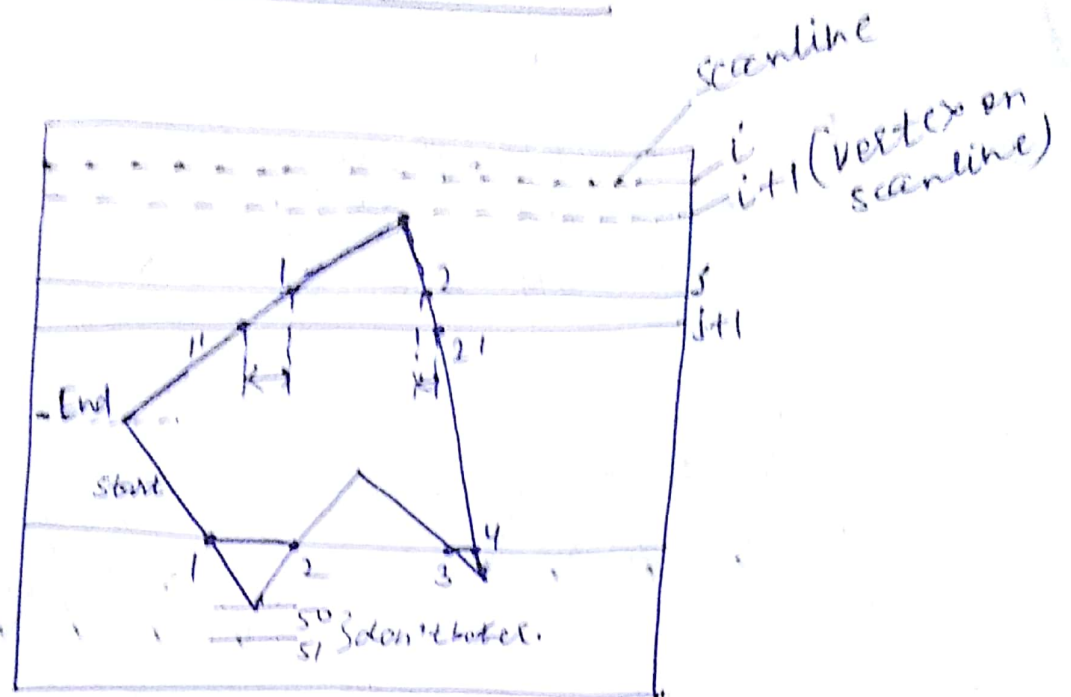
}

}

→ doesn't work for
large polygons.

→ If an inside pixel
is in some other color
then fill terminates &
polygon remain unfilled.

Scan Conversion



- Pixels within the polygon are displayed
- Row of pixels are called scanline.
- All pixel b/w 1-2 are displayed
 " " " 1/2 / 3-4 " "
- moving from 1SL to another SL.

→ 1 ON 1' modified
 2 — 2'

→ Point 1 is modified by the slope of line

→ when we move from 1SL to next the position of edge should be an amt $= \frac{dy}{dx} = \frac{\Delta y}{\Delta x}$

From 1SL to another :-

- 1) Intersection pt. change by $\Delta y / \Delta x$
- 2) edges can start / end.

Intersection points.

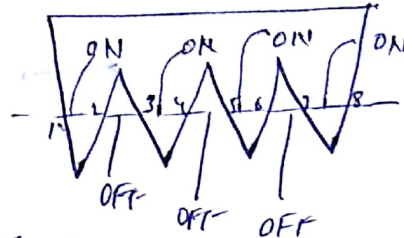
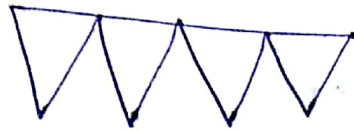
I_1, I_2, \dots, I_n

B/w 1 & 2 — ON — Pixels.

2 & 3 — OFF

3 & 4 — ON

→



→ Every scanline keep track of intersection pts.

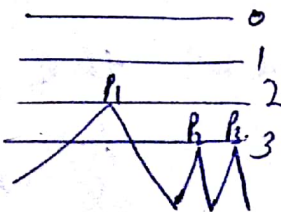
Algorithm

Given:- A polygon as a list of points.

Todo:- To fill the polygon.

Algo:- 1.

Sort vertices as per y-coordinate and place them in bucket for that co-ordinate.



Buc[i] →

B[0] →

B[1] →

B[2] — P₁ (vertices).

B[3] — P₂, P₃

If y coordinate is i
we keep track of Buc[i].

→ A polygon defined by a set of vertices $v_1 - v_n$
and color of polygon color.
for all polygon edges do

calculate $\frac{dx}{dy}$ for each edge.

end for

for all scanlines in polygon (top bottom) do
for all polygon edges do

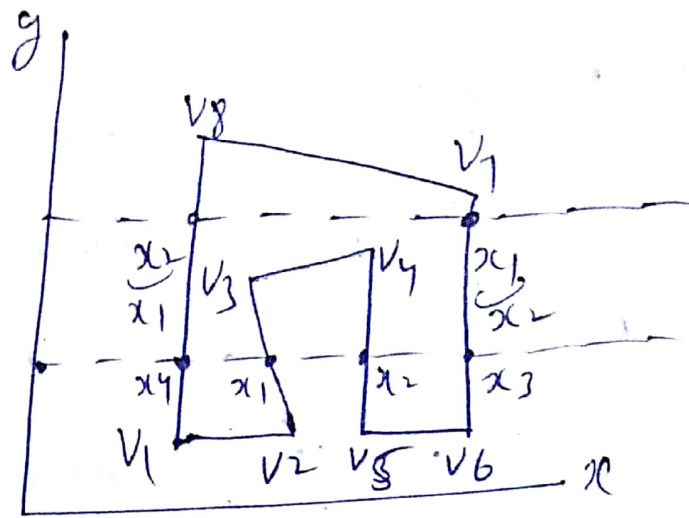
if edge intersects with scanline then
cal. x co-ordinate of scan extrema

end if

end for

sort the scan extrema in ascending order
illuminate pixels b/w scan extrema pairs
end for.

-
- A scanline is a horizontal row of pixels.
 - The SL starts from top of polygon and goes through row of pixels starting at top of polygon.
 - Points where polygon edge intersects scanline are called scan extrema with
 - If an edge intersects with scanline x-coordinates of scan extrema are called x-coordinates LE.



edge $V_1 - V_2$ doesn't intersect scanline.

$V_2 - V_3$

$V_3 - V_4$

1)

$V_4 - V_5$

$V_5 - V_6$

$V_6 - V_7$ — intersects scanline at x_1

$V_7 - V_8$ — no.

$V_8 - V_1$ — n n n x_2

→ Scan extends sorted in ascending order.

→ $V_1 - V_2$

$V_2 - V_3 - x_1$

$V_3 - V_4$

$V_4 - V_5 - x_2$

$V_5 - V_6$

$V_6 - V_7 - x_3$

$V_7 - V_8$

$V_8 - V_1 - x_4$

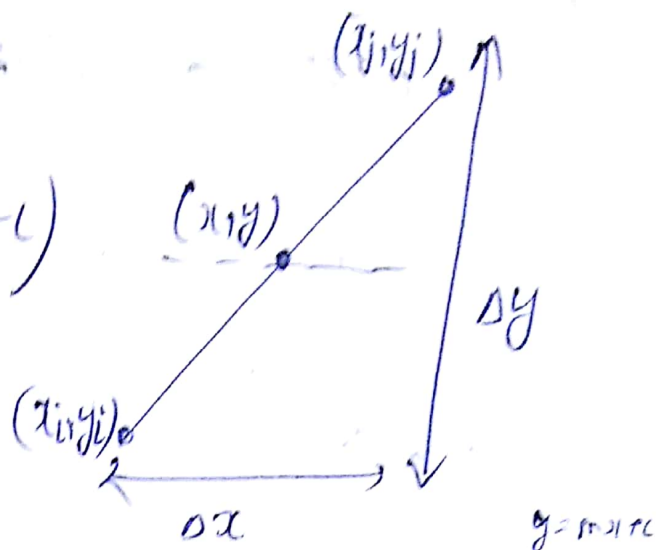
Linear Interpolation

→ Cartesian eq. of line is

$$y = \frac{\Delta y}{\Delta x} x + c \quad (y = mx + c)$$

Given (x_i, y_i) is on line
then

$$c = y_i - \frac{\Delta y}{\Delta x} x_i$$



→ Substitute c into eq. of a straight line.

$$\boxed{x = x_i + \frac{\Delta x}{\Delta y} (y - y_i)}$$

Scan extremes

Co-ordinate of scan extremes are:

$$y = y_i - 1$$
$$x = x_i + \frac{\Delta x}{\Delta y} (y - y_i)$$

$\frac{\Delta x}{\Delta y}$ = constant along edge, \therefore it can be pre-cal.

If an edge is horizontal $\Delta y = 0$

$$m = \begin{cases} \frac{\Delta x}{\Delta y} & \text{if } \Delta y \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

→ Scan lines are sorted in ascending order of x and all pixels b/w pairs of scan lines are filled.

