Chapter-3
Scan Conversion

Scan Conversion:-

- A line is defined by its two endpoints and the line equation. where as a circle is defined by its radius, center position and the circle equation.
- It is the responsibility of the graphics system or the application program to convert each primitive from its geometric definition into a set of pixels that make up the primitive in the image space. The conversion is generally referred to as scan conversion or rasterization.

Scan Converting a Point:-

- A mathematical point (x, y), where $x, y \in \mathbb{R}$ and within an image area, needs to be scanned converted to a pixel at location (x', y').

⊛ This may be done by taking only the integer part or the floor of (x, y)

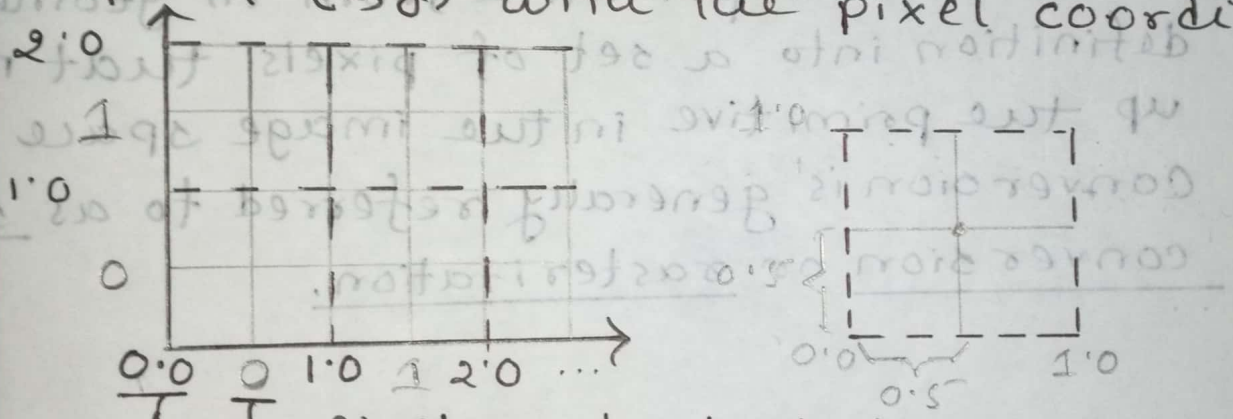
$x' = \lfloor x \rfloor$, $y' = \lfloor y \rfloor$

where, $x' \leq x < x'+1$ $y' \leq y < y'+1$

Example –

$(1.7, 0.8) \longrightarrow (1, 0)$
$(2.2, 1.3) \longrightarrow (2, 1)$
$(2.8, 1.9) \longrightarrow (2, 1)$

⊛ Another method is to align the integer values in the coordinate system for the point (x, y) with the pixel coordinate.



– Here, we can scan convert (x, y) by making

$x' = \lfloor x + 0.5 \rfloor$
$y' = \lfloor y + 0.5 \rfloor$

where, $x' - 0.5 \leq x < x' + 0.5$
$y' - 0.5 \leq y < y' + 0.5$

– This places the origin of the coordinate system for (x, y) at the center of pixel $(0, 0)$

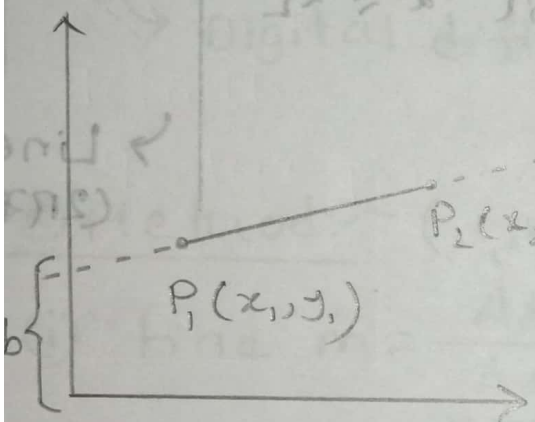
Example–

$(1.7, 0.8) \rightarrow (2, 1)$

$(2.8, 1.3) \rightarrow (2, 1)$

$(2.8, 1.9) \rightarrow (3, 2)$

Scan converting a line:–



$y = mx + b$

m = slope

b = y intercept

P_1 & P_2 = Two endpoints

Direct use of the line equation:–

Inputs:–

Two endpoints $P_1(x_1, y_1)$, $P_2(x_2, y_2)$

Output:–

Draw the line using input.

Method:–

(i) Find $m = \frac{y_2 - y_1}{x_2 - x_1}$

(ii) Find $b = y_1 - mx_1$

(iii) Set $x = x_1$

$y = y_1$

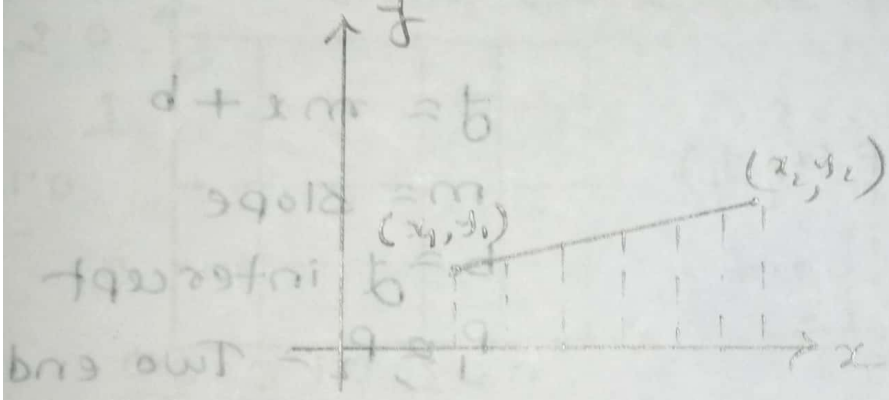
putpixel(x, y)

(iv) If $|m| \leq 1$,

$x = x + 1$

$y = mx + b$

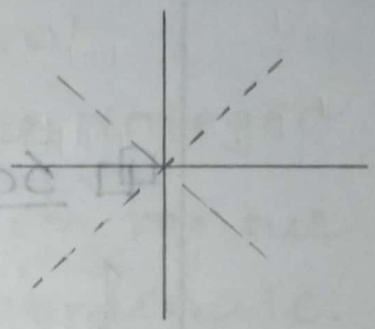
putpixel (x, y) until $x \leq x_2$



$|m| \leq 1$

$\Rightarrow -1 \leq m \leq 1$

$\Rightarrow -45^\circ \leq \theta \leq 45^\circ$



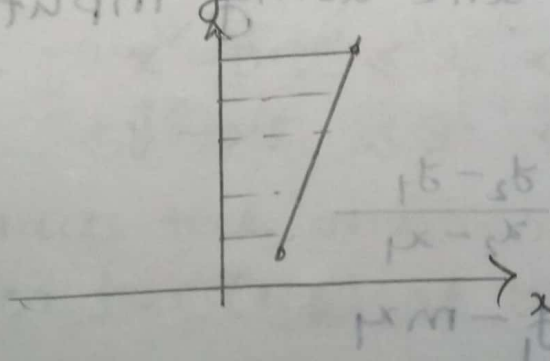
↳ Line x-axis (নিকট)

(v) Else if $|m| > 1$,

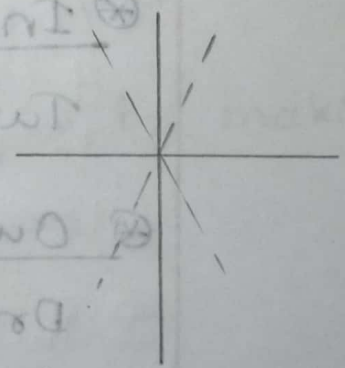
$y = y + 1$

$x = \frac{y - b}{m}$

putpixel (x, y) until $y \leq y_2$



$|m| > 1$



↳ Line y axis (নিকট)

⊛ Pros:-

Easy to implement.

⊛ Cons:-

Floating point multiplication and addition.

DDA Algoritum:-

↳ Digital differential analyzer

⊛ Method:-

(i) Find $m = \frac{\Delta y}{\Delta x}$

(ii) $x_i = x_1$, $y_i = y_1$ $(x_1 < x_2, y_1 < y_2)$

Putpixel (x, y)

(iii) If $|m| \leq 1$

$x_{i+1} = x_i + 1$

$y_{i+1} = y_i + m$

Putpixel (x_{i+1}, y_{i+1})

until $x_{i+1} \lesssim x_2$

$\Delta x = 1$
$m = \Delta y / \Delta x$
$\Rightarrow \Delta y = m \Delta x$
$\Rightarrow y_{i+1} - y_i = m \quad (\because \Delta x = 1)$
$\therefore y_{i+1} = y_i + m$

(iv) If $|m| > 1$

$y_{i+1} = y_i + 1$

$x_{i+1} = x_i + 1/m$, putpixel (x_i, y_i)

Repeat until $y_{i+1} \leq y_2$

$\Delta y = 1$

$m = \Delta y / \Delta x$

$\Rightarrow \Delta x = \Delta y / m$

$\Rightarrow x_{i+1} - x_i = 1/m$

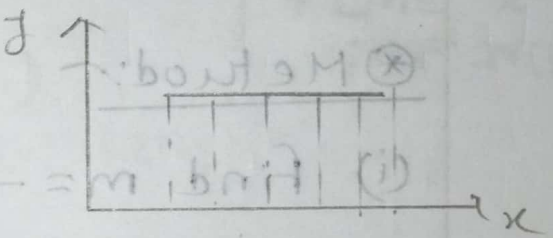
$\therefore x_{i+1} = x_i + 1/m$

(v) If $|m| = 0$

$$\frac{\Delta y}{\Delta x} = \frac{0}{1}$$

$x_{i+1} = x_i + 1$

$y_{i+1} = y_i$

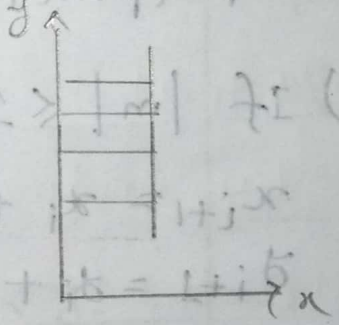


(vi) If $|m| = \infty$

$$\frac{\Delta y}{\Delta x} = \frac{\Delta y}{0}$$

$x_{i+1} = x_i$

$y_{i+1} = y_i + 1$

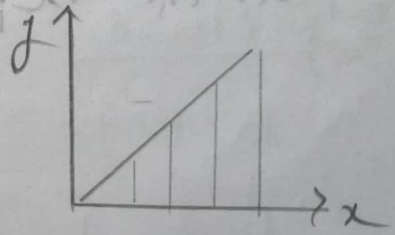


(vii) If $|m| = 1$

$$\frac{\Delta y}{\Delta x} = 1 \quad \therefore \Delta y = \Delta x$$

$y_{i+1} = x_{i+1}$

$x_{i+1} = x_i + 1$, putpixel (x_i, x_i)



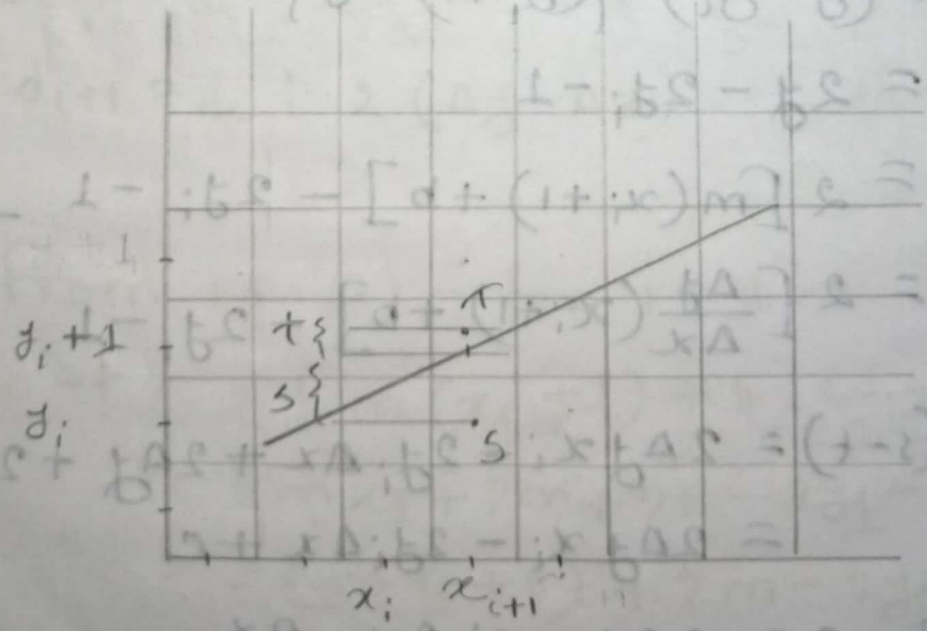
⊛ Pros:-

faster than using line equation. Because we don't need any floating point multiplication.

⊛ Cons:-

- Floating point addition is still needed.
- Cumulative error due to limited precision in the floating point representation may cause calculated points to drift away from their true position when the line is relatively long.

Bresenham's Line Algoritum:-



- For S point, $x_{i+1} = x_i + 1$
  $y_{i+1} = y_i$
- For T point, $x_{i+1} = x_i + 1$
  $y_{i+1} = y_i + 1$

- The actual y coordinate of the line at $x = x_{i+1}$ is

$$y = m x_{i+1} + b$$
$$= m(x_i + 1) + b \quad [\because x_{i+1} = x_i + 1] \quad \text{---(1)}$$

- $s = y - y_i$
  $t = (y_i + 1) - y$

$$\therefore s - t = (y - y_i) - \{(y_i + 1) - y\}$$
$$= 2y - 2y_i - 1$$
$$= 2[m(x_i + 1) + b] - 2y_i - 1 \quad \text{---(2)}$$
$$= 2\left[\frac{\Delta y}{\Delta x}(x_i + 1) + b\right] - 2y_i - 1$$

$$\Rightarrow \Delta x(s - t) = 2\Delta y x_i - 2y_i \Delta x + 2\Delta y + 2b\Delta x - \Delta x$$
$$= 2\Delta y x_i - 2y_i \Delta x + c$$

where, $c = 2\Delta y + 2b\Delta x - \Delta x$

Let, $d_i = \Delta x(s-t)$

↳ decision variable

$d_i = 2\Delta y x_i - 2\Delta x y_i + c$

$d_{i+1} = 2\Delta y x_{i+1} - 2\Delta x y_{i+1} + c$

$= 2\Delta y (x_i + 1) - 2\Delta x y_{i+1} + c$

$d_{i+1} - d_i = 2\Delta y - 2\Delta x(y_{i+1} - y_i)$ ——— (3)

– For S point – $y_{i+1} = y_i$

$d_{i+1} - d_i = 2\Delta y$

$\therefore d_{i+1} = d_i + 2\Delta y$

– for T point – $y_{i+1} = y_i + 1$

$d_{i+1} - d_i = 2\Delta y - 2\Delta x$

$\therefore d_{i+1} = d_i + 2(\Delta y - \Delta x)$

$s - t \geq 0$

$\Rightarrow \Delta(s-t) \geq 0$

$\Rightarrow d_i \geq 0$ for T point

$d_i < 0$ for S point

– From eq. (2) – for $i = 1$ –

$s - t = 2[m(x_1 + 1) + b] - 2y_1 - 1$

$= 2mx_1 + 2m + 2b - 2y_1 - 1$

$= 2(mx_1 + b) + 2m - 2y_1 - 1$

$= 2y_1 + 2m - 2y_1 - 1$

$= 2m - 1$

$\Rightarrow s-t = \frac{\Delta y}{\Delta x} \cdot 2 - 1$

$\Rightarrow \Delta x(s-t) = 2\Delta y - \Delta x$

$\Rightarrow d_i = 2\Delta y - \Delta x$

$\therefore d_1 = 2\Delta y - \Delta x$

$dT = 2(\Delta y - \Delta x)$

$ds = 2\Delta y$

$$d_{i+1} = \begin{cases} d_i + 2(\Delta y - \Delta x) & , \text{if } d_i \geq 0 \\ d_i + 2\Delta y & , \text{if } d_i < 0 \end{cases}$$

$d_1 = 2\Delta y - \Delta x$

Input Points

$P_1'(x_1', y_1'), P_2'(x_2', y_2')$

⊛ Algoritum:-

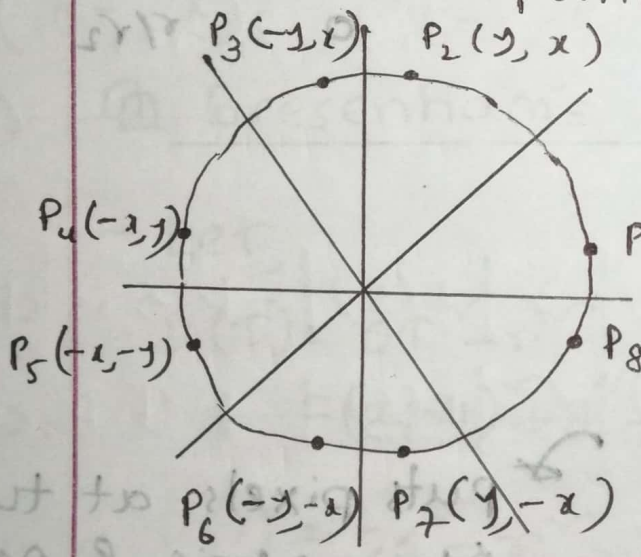
```
x = x1', y = y1';
dx = x2' - x1', dy = y2' - y1', dT = 2(dy - dx), ds = 2dy;
d = 2dy - dx;
setPixel(x, y);
while(x < x2'){
    x++;
    if(d < 0){
        d = d + ds;
    }
    else{
        d++;
    } d = d + dT;
```

setPixel (x, y);
}

- Highly efficient.
- Only integer addition, subtraction and multiplication.

Scan converting a circle:-

- A circle is a symmetric figure. Any circle-generating algorithm can take help of this 8-point symmetry of circle.
- 8-way symmetry is used by reflecting each calculated point around each 45° axis.



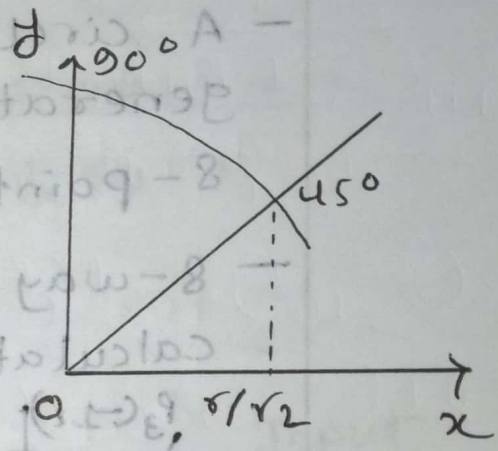
$P_1(x, y)$
$P_2(y, x)$
$P_3(-y, x)$
$P_4(-x, y)$
$P_5(-x, -y)$
$P_6(-y, -x)$
$P_7(y, -x)$
$P_8(x, -y)$

Using direct equation:-

- $x^2 + y^2 = r^2$

$\therefore y = \sqrt{r^2 - x^2}$

- Each x coordinate is in the sector from 90° to 45°.



At 90°, $x = r\cos\theta = r\cos 90°$
$= 0$

At 45°, $x = r\cos 45°$
$= r/\sqrt{2}$

Algorithm:-

```
while (x <= r/√2) {
    y = √(r² - x²);
    putPixel (x, y);
    x++;
}
```

→ puts pixels at the symmetric 8 points.
চার এলাকায় করে 8 টি করে মোট 32 টি পিক্সেল আসে।

Cons:-

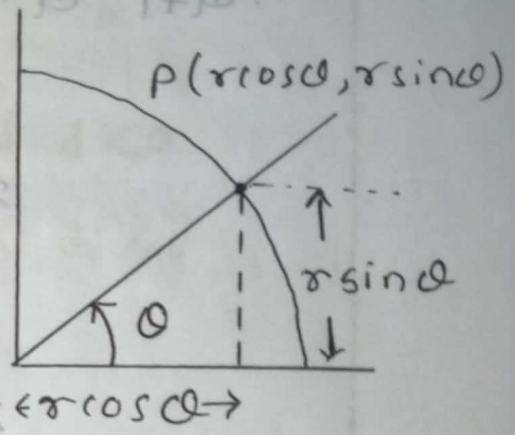
- very inefficient.
- for each point x and r needs to be squared, subtracted and find square root of them.

Using Trigonometric f^n s:-

θ is changed from 0 to $\pi/4$

$x = r\cos\theta$

$y = r\sin\theta$



Cons:-

- sinθ and cosθ are more time consuming than the previous method.

Bresenham's Circle Algorithm:-

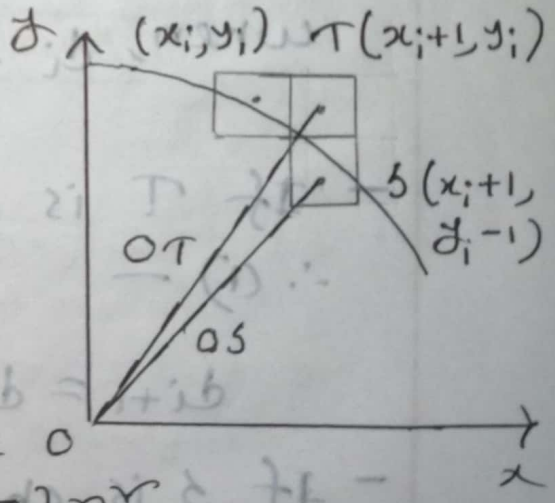
- Let,

$D(T) = OT^2 - r^2$

$= (x_i+1)^2 + y_i^2 - r^2$

$D(S) = OS^2 - r^2$

$= (x_i+1)^2 + (y_i-1)^2 - r^2$



$d_i = D(T) + DS$

$= 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 - 2r^2$

$d_{i+1} = 2(x_{i+1}+1)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2r^2$

$\therefore d_{i+1} - d_i = 2(x_{i+1}+1)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - y_i^2 - (y_i-1)^2$

$\because x_{i+1} = x_i + 1$

~~$\therefore d_{i+1} - d_i = 4x_i + 2(y_{i+1} - y_i) - 2(y_{i+1} - y_i) + 6$~~

$\therefore d_{i+1} - d_i = 2(x_i + 2)^2 + y_{i+1}^2 + y_{i+1} + 1 - 2y_{i+1}$
$- 2x_i^2 - 4x_i - 2 - y_i^2 - y_i + 2y_i - 1$

$= 2x_i^2 + 8x_i + 8 + y_{i+1}^2 + y_{i+1} + 1 - 2y_{i+1}$
$- 2x_i^2 - 4x_i - 2 - y_i^2 - y_i + 2y_i - 1$

$= 4x_i + 6 + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i)$

$\therefore d_{i+1} = d_i + 4x_i + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i) + 6$ ——— (1)

- when, $d_i \geq 0$, $|D(T)| \geq |D(S)| \rightarrow$ S is chosen
when, $d_i < 0$, $|D(T)| < |D(S)| \rightarrow$ T is chosen

- If T is chosen, ($x_{i+1} = x_i + 1$ and $y_{i+1} = y_i$)
$\therefore$ (1) —

$d_{i+1} = d_i + 4x_i + 6$

- If S is chosen, ($x_{i+1} = x_i + 1$, $y_{i+1} = y_i - 1$),
(1) —

$d_{i+1} = d_i + 4(x_i - y_i) + 10$

- Because, we will start at $\theta = 90°$, the first point will be $(0, r)$.

$\therefore d_1 = 2(0+1)^2 + r^2 + (r-1)^2 - 2r^2$

$= 3 - 2r$

- $d_{i+1} = \begin{cases} d_i + 4x_i + 6 & , \text{if } d_i < 0 \\ d_i + 4(x_i - y_i) + 10 & , \text{if } d_i \geq 0 \end{cases}$

$d_1 = 3 - 2r$

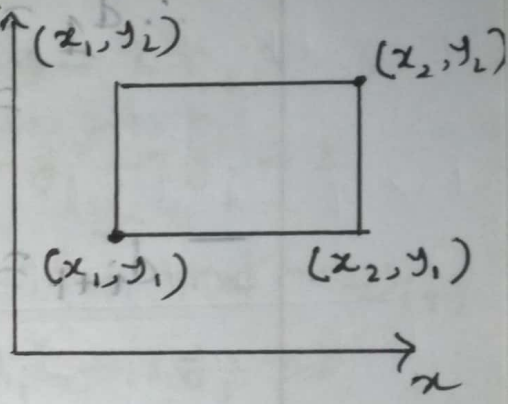
⊛ Algorithm:-

```
int x = 0, y = r, d = 3-2r;
while(x <= y){
    setPixel(x,y);
    if(d < 0){
        d = d + 4x + 6;
    }
    else{
        d = d + 4(x-y) + 10;
        y--;
    }
    x++;
}
```

8 points

Scan Converting a rectangle:-

```
drawRectangle(x1, y1, x2, y2){
    drawLine(x1, y1, x2, y1);
    drawLine(x2, y1, x2, y2);
    drawLine(x1, y2, x1, y2);
    drawLine(x1, y1, x1, y2);
}
```



↳ - No need of Bresenham's algorithm.
- Same for square drawing.

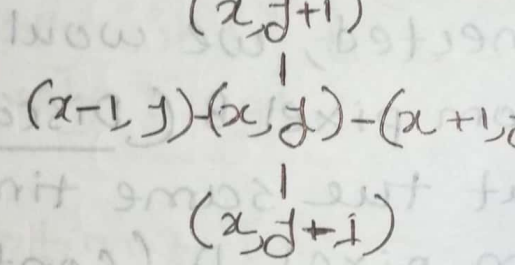
Region Filling:-

- Region filling is the process of coloring in a ~~defined~~ definite image area or region. Regions may be defined at the pixel or geometric level.
- At the pixel level, we describe a region either in terms of the bounding pixels that outline it or as the totality of pixels that comprise it.
- In the first case, the ~~bounda~~ region is called boundary-defined and algorithms used to fill it are called boundary-fill algorithm.

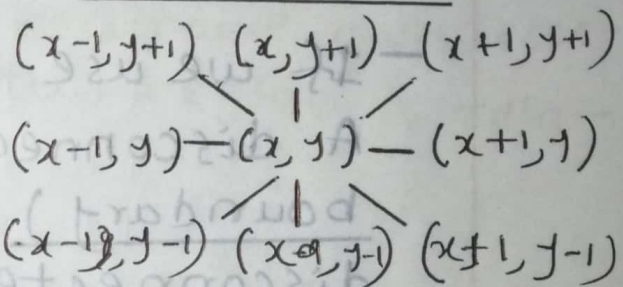
- The other type is called interior-defined region and the algorithms are called flood-fill algorithms.
- At the geometric level a region is defined or enclosed by such abstract contouring elements as connected lines and curves.

* 4-connected vs 8-connected :-

4-connected



8-connected



- Using 4-connected, the outside pixels in fig-1, do not define a region, because A and B are not connected.
- Using 8-connected, we identify a triangular region.

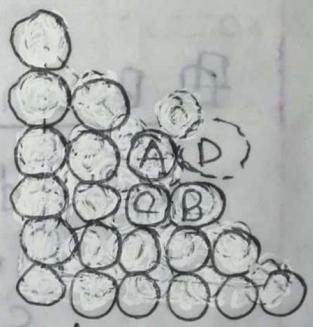


fig-1

- We can also decide if a region is connected to each other. Using 8-connected, we do not have an enclosed region, since interior pixel C is connected with exterior pixel D.

- Fig-1 is a boundary-defined region when we use 8-connected to boundary pixels and 4-connected for the interior pixels.
- Using same definition for both would result in contradiction.
- If we use 8-connected, we would have pixel A connected to pixel B (contiuous boundary) and at the same time pixel C connected to pixed D (Discontinuous boundary).
- If we use 4-connected, we would have pixel A disconnected from pixel B (Discontinuous boundary) and at the same time pixel C disconnected from pixel D (Continuous Boundary)

Boundary Fill Algoritum:-

```
Boundary fill (int x, int y, int Boundary_Color, int fill_color){
    int color;   seed point (inside the region)
    color = get_pixel (x, y);
    if (color != Boundary_Color && color != fill_Color){
        set_pixel (x, y, fill_color);
        Boundary fill (x+1, y, Boundary_color, fill_color);
        Boundary fill (x-1, y,  "  ,  "  );
        Boundary fill (x, y+1,  "  ,  "  );
        Boundary fill (x, y-1,  "  ,  "  );
    }
}
```

⊛ Pros:-

- The algoritum works elegantly on an arbitrarily shaped region by chasing and filling all non-boundary pixels that are connected to the seed either directly or indirectly.

⊛ Cons:-

- A straightforward implementation can take time and memory to execute due to the potentially high no. of recursive calls, especially when the size of the region is relatively large.

⊛ Optimization:-

- Variations can be made to limit the no. of recursive calls by structuring the order in which neighbouring pixels are processed. For example- we can first fill pixels to the left and right of the seed on the same scan line until boundary pixels are hit; we then inspect each pixel above and below the line just drawn- which can also be done with a loop- to see if it can be used as a new seed for the next horizontal line to fill.
- This way the number of recursive calls at any particular time is N when the current

line is N scan lines away from the initial seed.

A Flood fill Algorithm:-

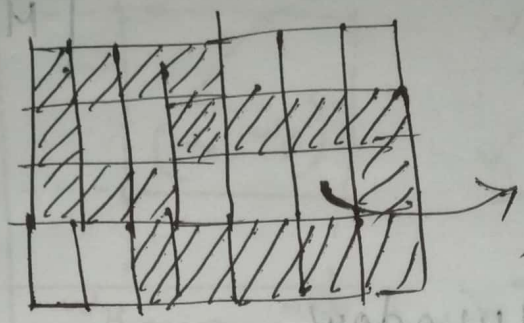
```
Flood fill ( x, y, original Color, fill Color){
    int color;
    color = get_pixel (x, y, fill Color);
    if (color == original Color){
        set_pixel (x, y, fill Color);
        Floodfill ( x+1, y, fill Color);
        Floodfill (x-1, y, fill color);
        Flood fill (x, y+1, fill color);
        Flood fill (x, y-1, fill color);
    }
}
```

Pros:-

- It is particularly useful when the region to be filled has no uniformly colored boundary. However, if the boundary is well defined but is itself multiply colored, it would be better to use boundary filled method.

Cons and optimization:-

↳ Same as boundary-fill alg.



4 point fails

if point works

Q. which algoritum when?

↳ (Hw)

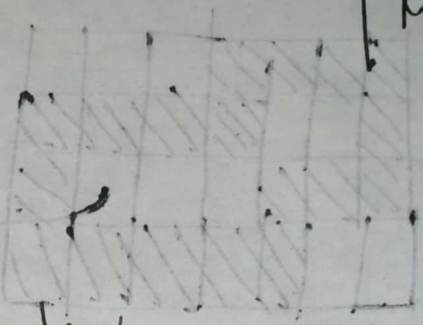
(CT) → Saturday

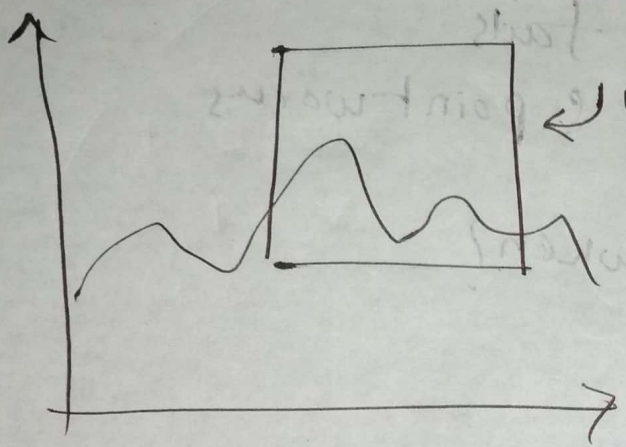
ch-3

19.07.22
CG
M71 Sir

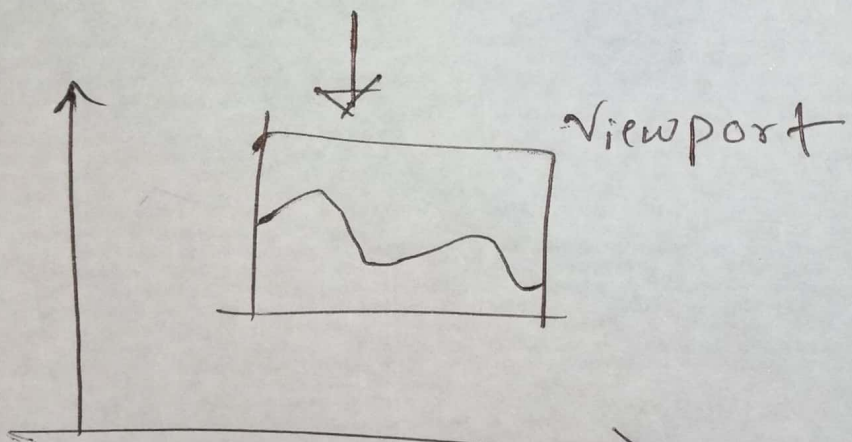
Ch-5

Viewing and clipping

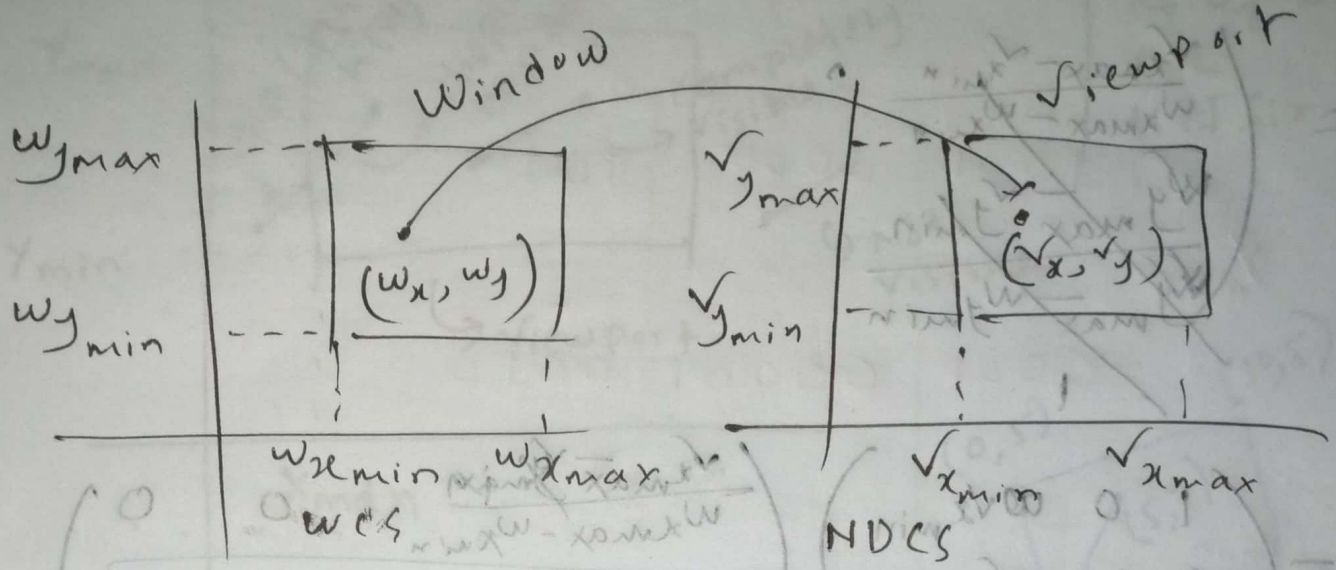




WCS
↳ real world



NDCS (Device)
↳ Normalized
Device coordinate
System



$$\frac{w_x - w_{x_{min}}}{w_{x_{max}} - w_{x_{min}}} = \frac{v_x - v_{x_{min}}}{v_{x_{max}} - v_{x_{min}}}$$

$$\therefore v_x = \frac{v_{x_{max}} - v_{x_{min}}}{w_{x_{max}} - w_{x_{min}}} (w_x - w_{x_{min}}) + v_{x_{min}}$$

$$\therefore v_y = \frac{v_{y_{max}} - v_{y_{min}}}{w_{y_{max}} - w_{y_{min}}} (w_y - w_{y_{min}}) + v_{y_{min}}$$

$$\begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} = N \begin{pmatrix} w_x \\ w_y \\ 1 \end{pmatrix}$$

$\rightarrow N = ?$

$$= \begin{pmatrix} 1 & 0 & v_{x_{min}} \\ 0 & 1 & v_{y_{min}} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{v_{x_{max}} - v_{x_{min}}}{w_{x_{max}} - w_{x_{min}}} & 0 & 0 \\ 0 & \cdots & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

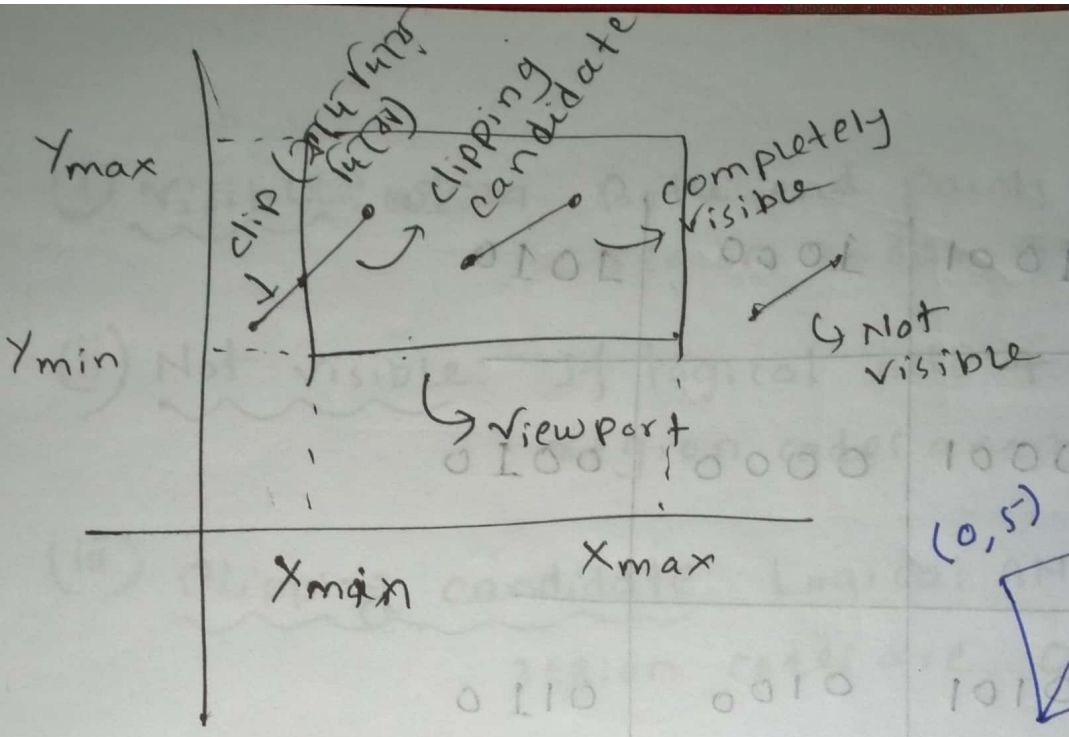
$$v_x = \frac{v_{x_{max}} - v_{x_{min}}}{w_{x_{max}} - w_{x_{min}}} (w_x - w_{x_{min}}) + v_{x_{min}}$$

$$v_y = \frac{v_{y_{max}} - v_{y_{min}}}{w_{y_{max}} - w_{y_{min}}} (w_y - w_{y_{min}}) + v_{y_{min}}$$

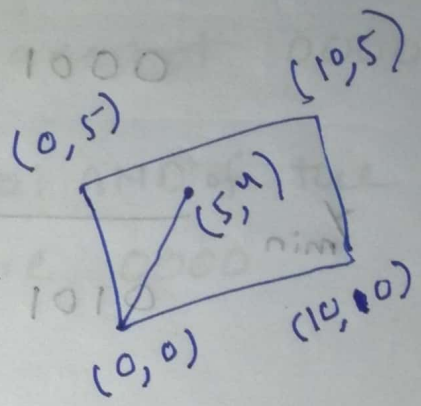
$$= \begin{bmatrix} 1 & 0 & v_{x_{min}} \\ 0 & 1 & v_{y_{min}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{v_{x_{max}} - v_{x_{min}}}{w_{x_{max}} - w_{x_{min}}} & 0 & 0 \\ 0 & \frac{v_{y_{max}} - v_{y_{min}}}{w_{x_{max}} - w_{y_{min}}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -w \\ 0 & 1 & -w \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{pmatrix} w \\ 1 \end{pmatrix} H = \begin{pmatrix} v \\ 1 \end{pmatrix}$$

$C = H$



26.07.22
CG
MZI Sir



Sutherland Algorithm:-

Region code নির্ণয় করবো → Visible/Not visible/Clipping candidate নির্ণয় করবো
(4-bits)

↳ Minimum 2টি point নিতে হবে।

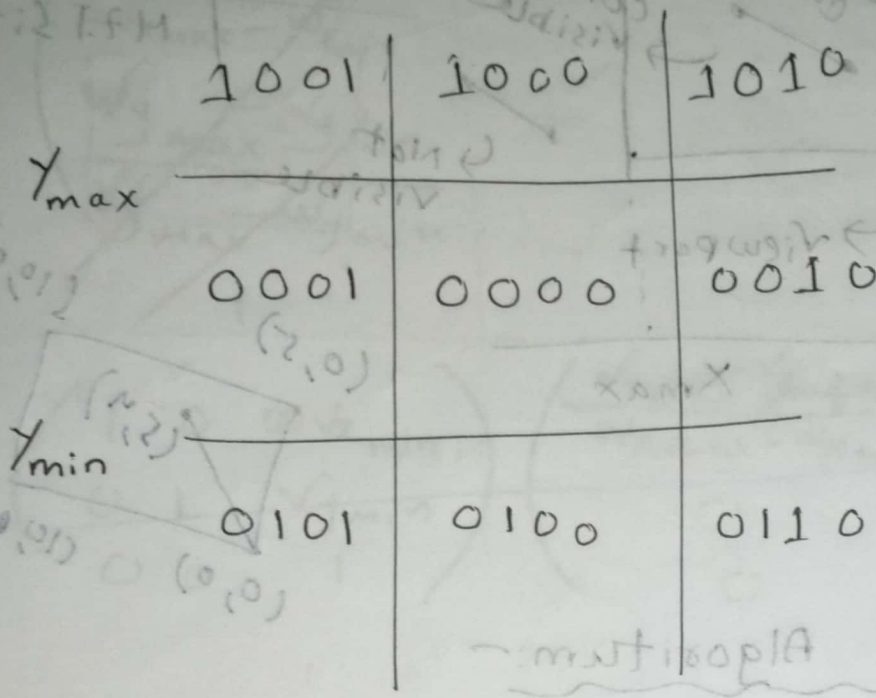
(i) bit-1: The point above of the y_{max} $= sign(y - y_{max})$

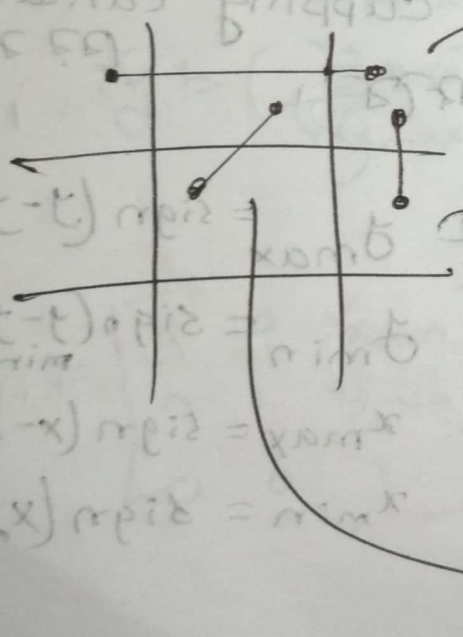
(ii) bit-2: The point below of " y_{min} $= sign(y - y_{min})$

(iii) bit-3: " " right " " $x_{max} = sign(x - x_{max})$

(iv) bit-4: " " left " " $x_{min} = sign(x_{min} - x)$

$$sign(a) = \begin{cases} 1, & \text{if } a \text{ is positive} \\ 0, & \text{otherwise} \end{cases}$$





1001
1010
1000 → bitwise and

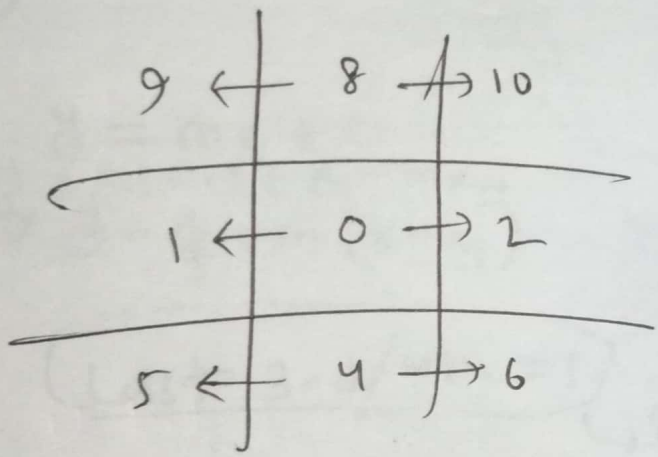
1010
0010
0010

1000
0000
0000

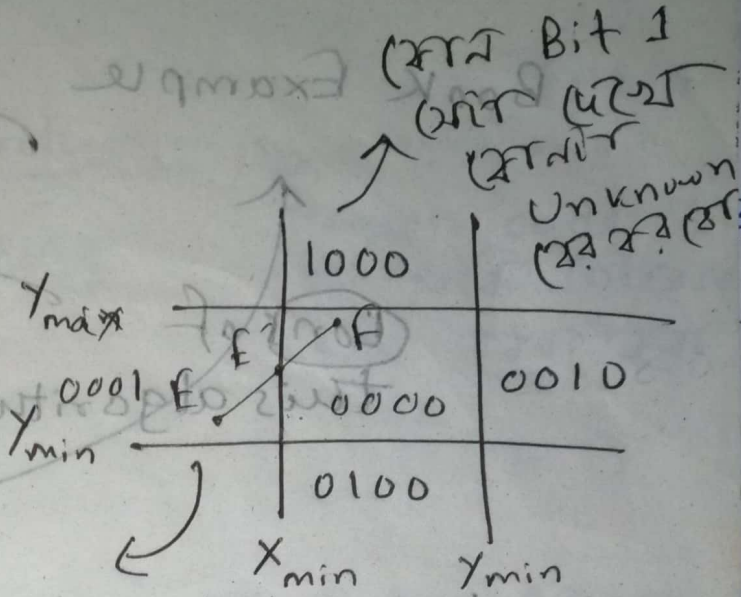
(i) Visible: Both end points region code are 0000

(ii) Not visible: If logical AND of the region codes are not 0000

(iii) Clipping candidate: Logical AND of the region codes are 0000



13.08.22
CG
MH Sir

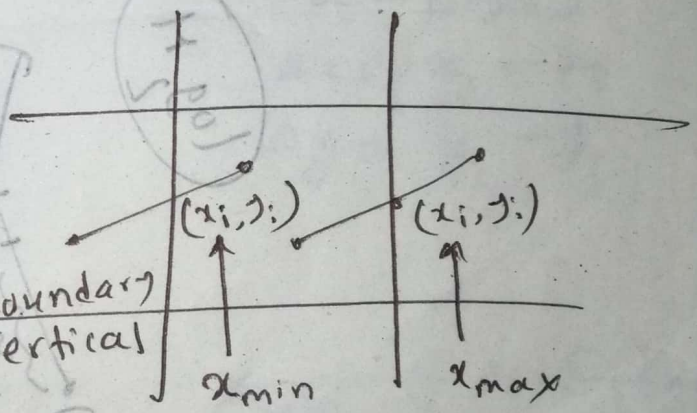


$E'(x_{min}, ?)$

(*)

$y = mx + c$
$\Rightarrow y = mx + y_1 - mx_1$
$\Rightarrow y = y_1 + m(x - x_1)$

Last 3rd/4th = 1 → Boundary Vertical



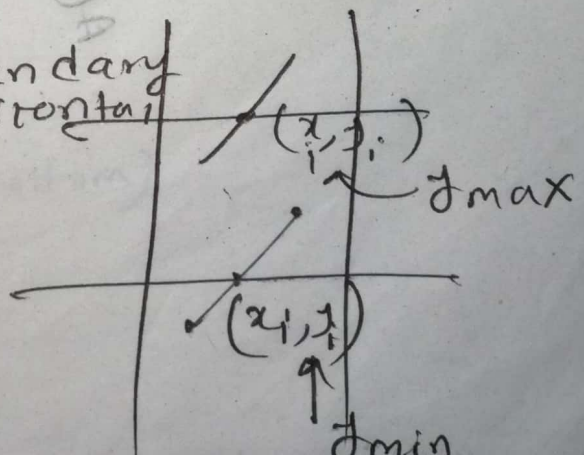
$x_i = x_{min}$ or x_{max}

$y_i = y_1 + m(x_i - x_1)$

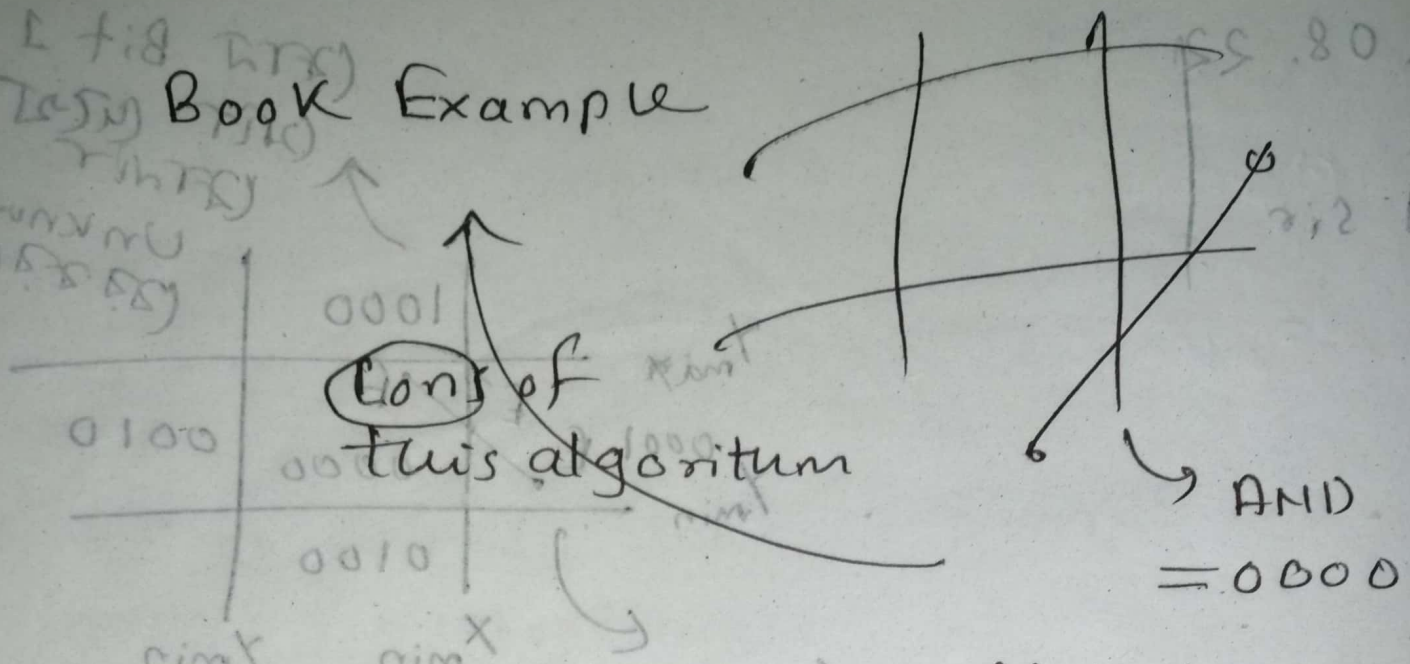
(*) 1st/2nd = 1 → Boundary Horizontal

$y_i = y_{min}$ or y_{max}

$x_i = x_1 + \frac{y_i - y_1}{m}$

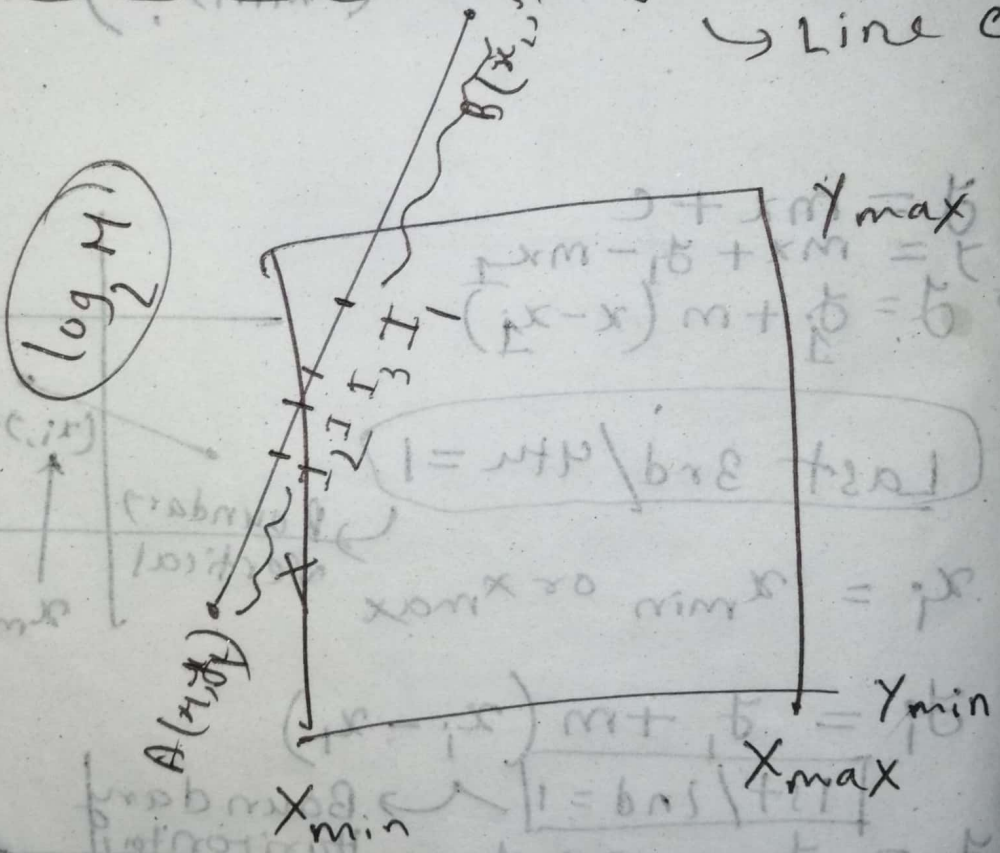


Book Example



Midpoint Sub-division Algoritum:-

↳ Line clipp

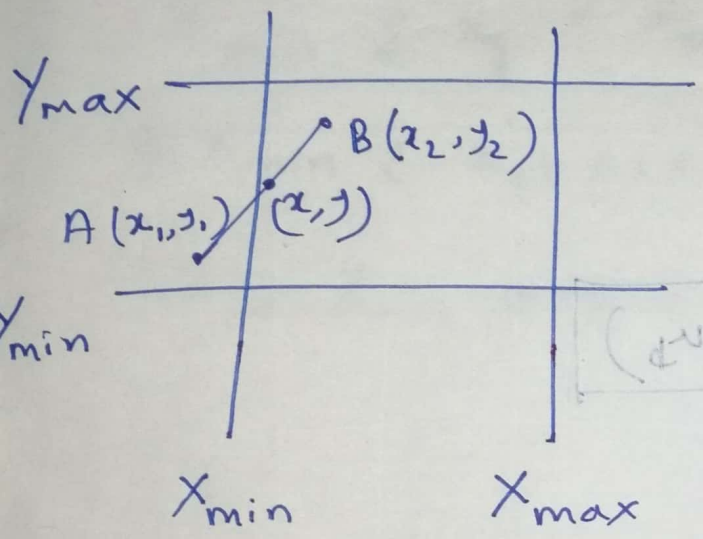


16.08.22
CG
MTJ Sir

Liang Barsky Algoritum:-

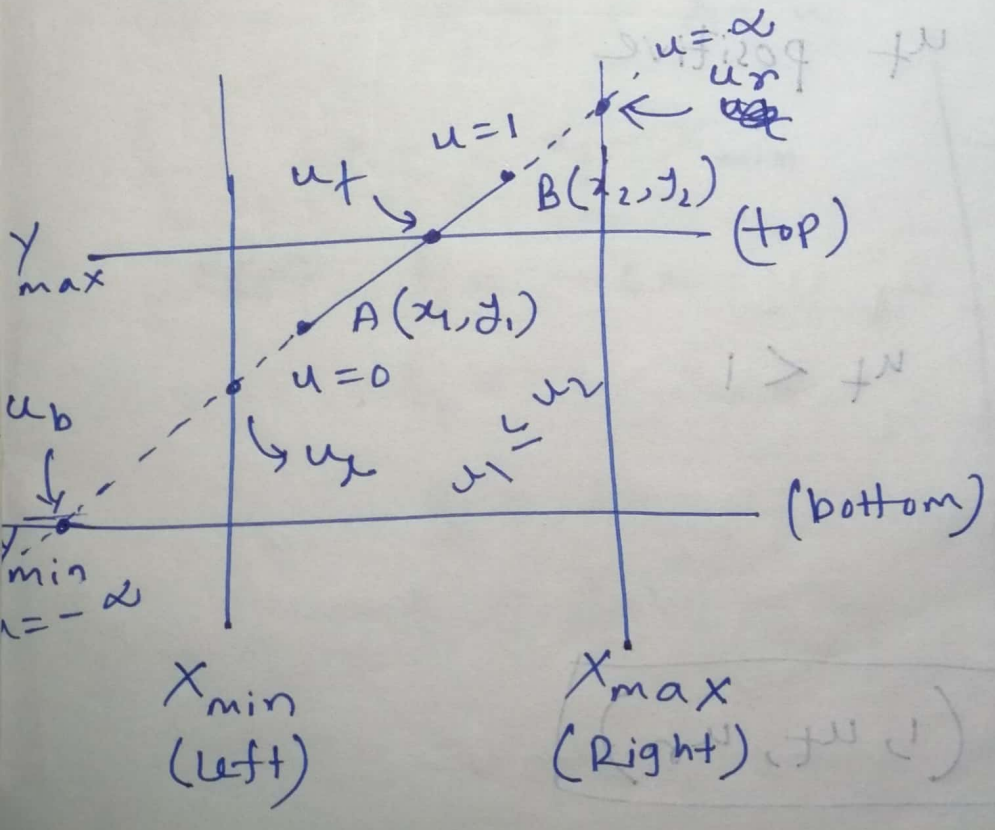
↳ For line clipping

↳ Region code চেক্ করে অবশ্যই Intersection point বের করা

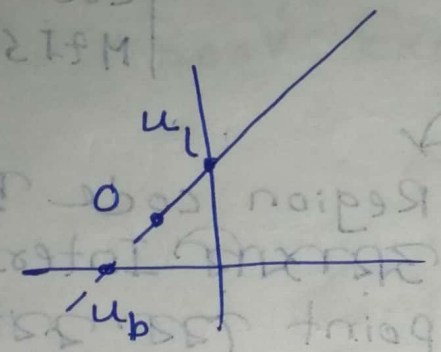


$$x = x_1 + \Delta x \cdot u$$
$$y = y_1 + \Delta y \cdot u$$

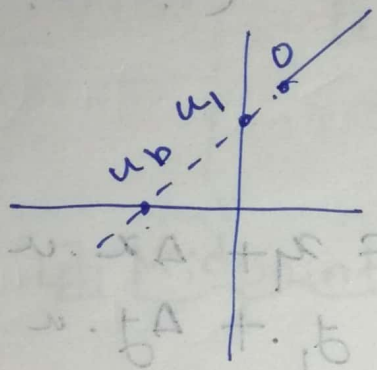
$$u = [0, 1]$$
$$\Delta x = x_2 - x_1$$
$$\Delta y = y_2 - y_1$$



$$X_{min} \leq x \leq X_{max}$$

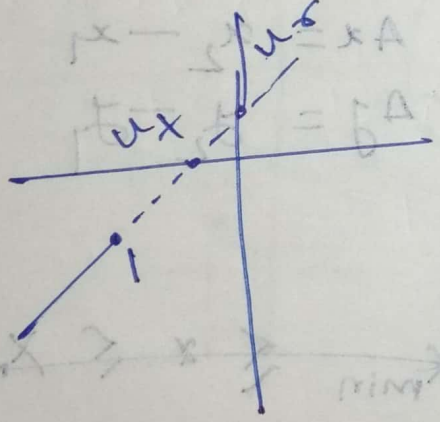


u_l positive

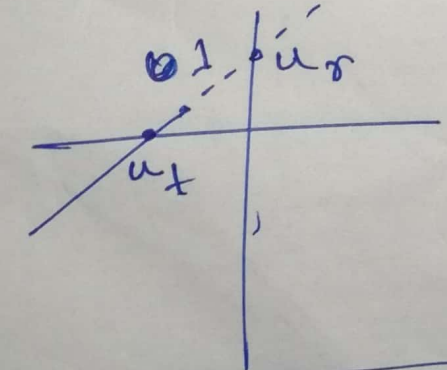


u_l negative

$$\therefore \boxed{u_l = \max(0, u_l, u_b)}$$



u_t positive



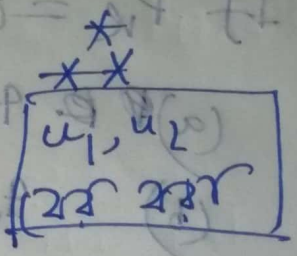
$u_t < 1$

$$\therefore \boxed{u_t = \min(1, u_t, u_r)}$$

$u_1 \rightarrow$ clipping start point

$u_2 \rightarrow$ u end

$\therefore u_1 \leq u_2$



$$x_{min} \leq x_1 \leq x_{max}$$

$$\Rightarrow x_{min} \leq x_1 + \Delta x \cdot u \leq x_{max}$$

And, $$y_{min} \leq y_1 + \Delta y\, u \leq y_{max}$$

$$p_k \cdot u \leq q_k$$

$k = 1, 2, \ldots$

$\therefore u \leq q_k / p_k$

$$x_{min} \leq x_1 + \Delta x\, u$$

$$\Rightarrow -\Delta x\, u \leq x_1 - x_{min}$$

$p_1 = -\Delta x$

$q_1 = x_1 - x_{min}$

$p_2 = \Delta x$

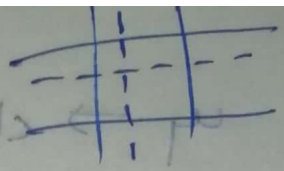
$q_2 = x_{max} - x_1$

$p_3 = -\Delta y$

$q_3 = y_1 - y_{min}$

$p_4 = \Delta y$

$q_4 = y_{max} - y_1$



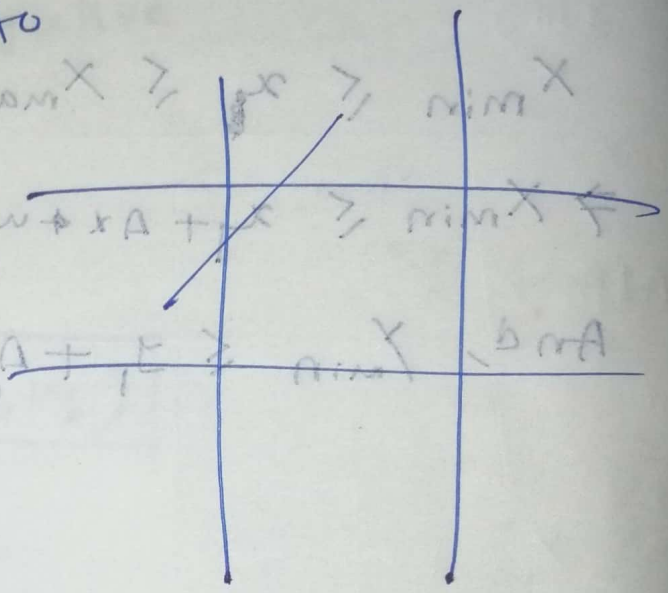
(i) If $P_k = 0 \Rightarrow$ Parallel

(a) $q_k < 0 \rightarrow$ window এর বাইরে $\rightarrow X$

(b) $q_k \geq 0 \rightarrow$ Inside window $\rightarrow \checkmark$

(ii) $P_k < 0 \rightarrow$ outside to inside

(iii) $P_k > 0 \rightarrow$ Inside to outside



(iv) $P_k \neq 0$

~~when $P_k \neq 0$~~

$$r_k = q_k / P_k$$

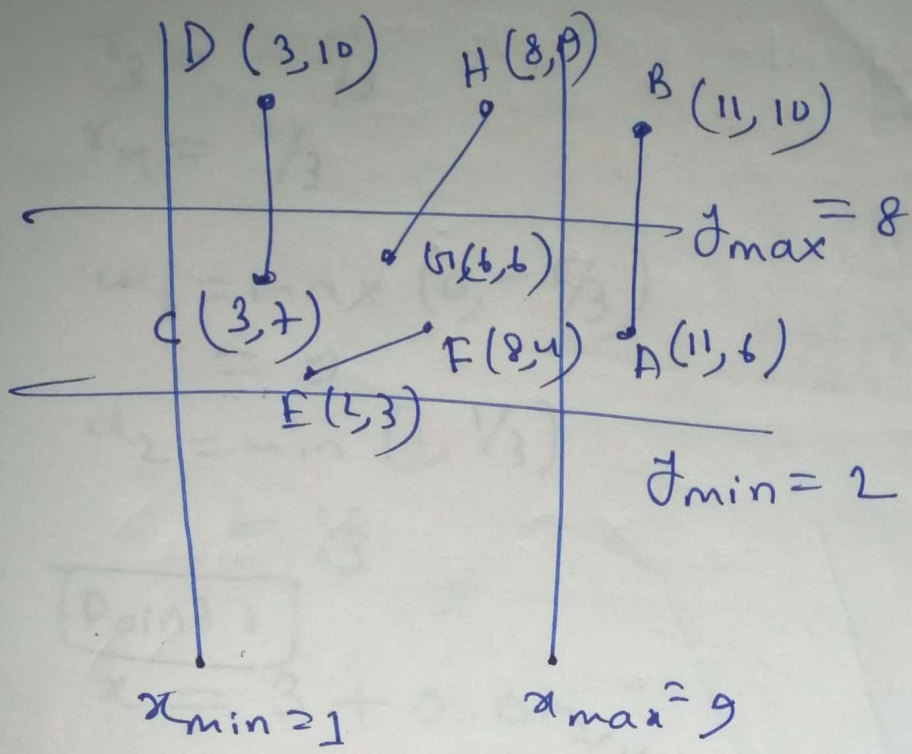
(a) $u_1 = \max(0, u_l, u_b)$

$= \max(0, r_1, r_3)$

(b) $u_2 = \min(1, u_r, u_t)$

$= \min(1, r_2, r_4)$

(c) If $u_1 > u_2$ then discard



For Line AB

$P_1 = 0,\ q_1 = 10$

$P_2 = 0,\ q_2 = -2$

$P_3 = -4,\ q_3 = 4$

$P_4 = +4,\ q_4 = 2$

→ any $P_k = 0$ ✓
any $q_k < 0$ ✓

→ ✗ Discard

For Line CD

$P_1 = 0,\ q_1 = 2$

$P_2 = 0,\ q_2 = 6$

$P_3 = -3,\ q_3 = 5$

$P_4 = 3,\ q_4 = 1$

→ Any $P_k = 0$ ✓
any $q_k < 0$ ✗
$P_1, P_2 = 0$ ✗ dont calc. α_1, α_2

$r_3 = -5/3$

$r_4 = 1/3$

$u_1 = \max(0, -5/3)$

$= 0$

$u_2 = \min(1, 1/3)$

$= 1/3$

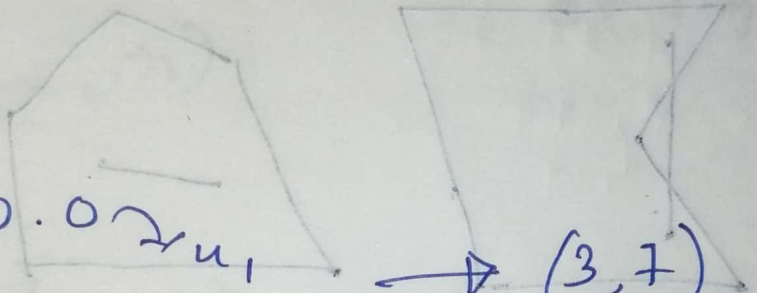
Point 1

$x = 3 + 0.0 \times u_1$

$= 3$

$y = 7 + 3.0 \times u_1$

$=$



$\rightarrow (3, 7)$

Point 2

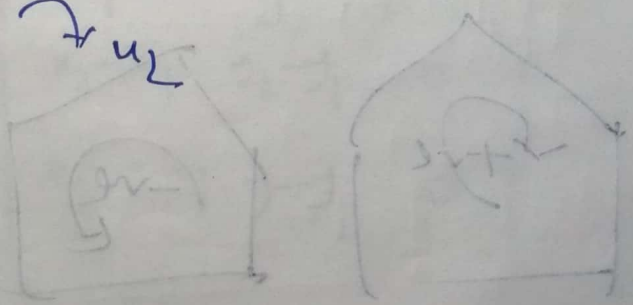
$x = 3 + 0.1/3 \times u_2$

$= 3$

$\rightarrow (3, 8)$

$y = 7 + 3.1/3 \times u_2$

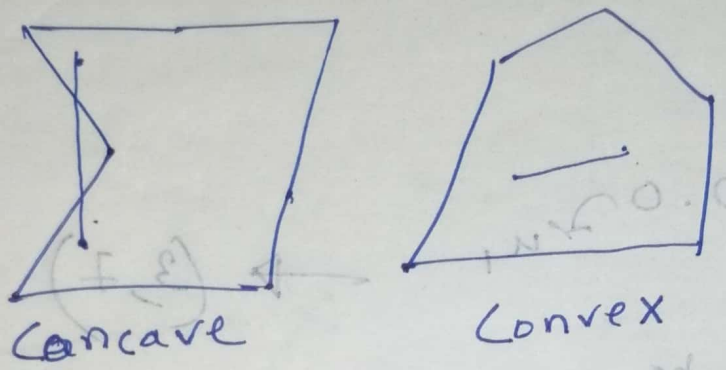
$= 8$



Polygon Clipping

(i) Convex polygon

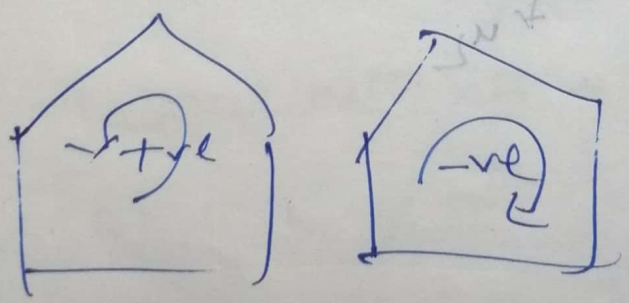
(ii) Concave polygon

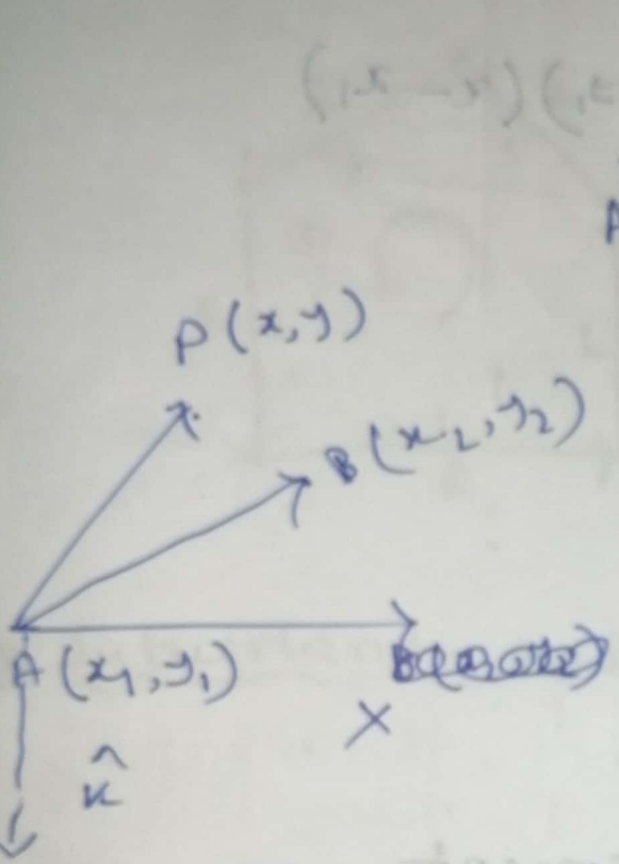


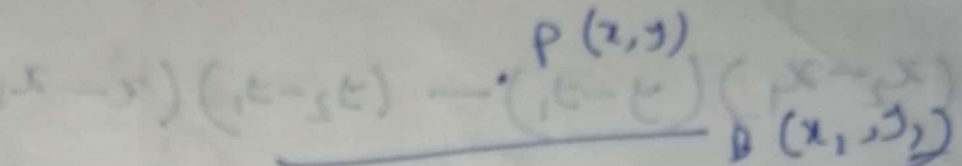
Orientation

(i) Positive ↷ Counterclockwise

(ii) Negative ↷ Clockwise







↳ P left of AB ?, or right ?

$$\overline{AB} = (x_2 - x_1)\,\hat{i} + (y_2 - y_1)\,\hat{j}$$

$$\overline{AP} = (x - x_1)\,\hat{i} + (y - y_1)\,\hat{j}$$

$$\overline{AB} \times \overline{AP} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_2 - x_1 & y_2 - y_1 & 0 \\ x - x_1 & y - y_1 & 0 \end{vmatrix}$$

$$= \hat{i}(0) + \hat{j}(0) + \hat{k}\{(x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)\}$$

$$c = |\overline{AB} \times \overline{AC}|$$

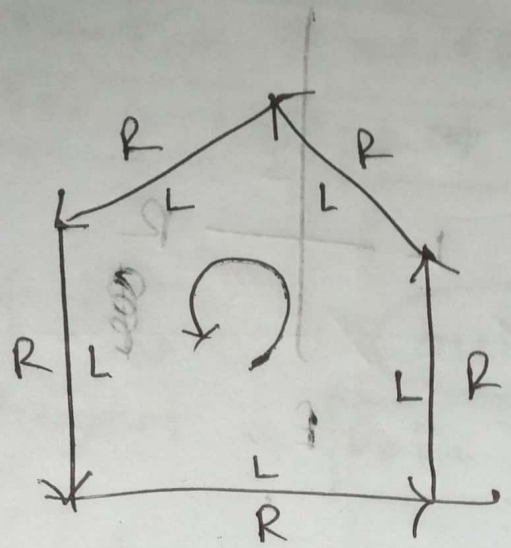
$$\therefore c = (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)$$

1. $c = 0$ → on the line
2. $c > 0$ → leftside
3. $c < 0$ → right

Subjective measurement
↳ যেমন মনে হচ্ছে
এমন হতে পারে

Objective measurement
↳ based on some measurement

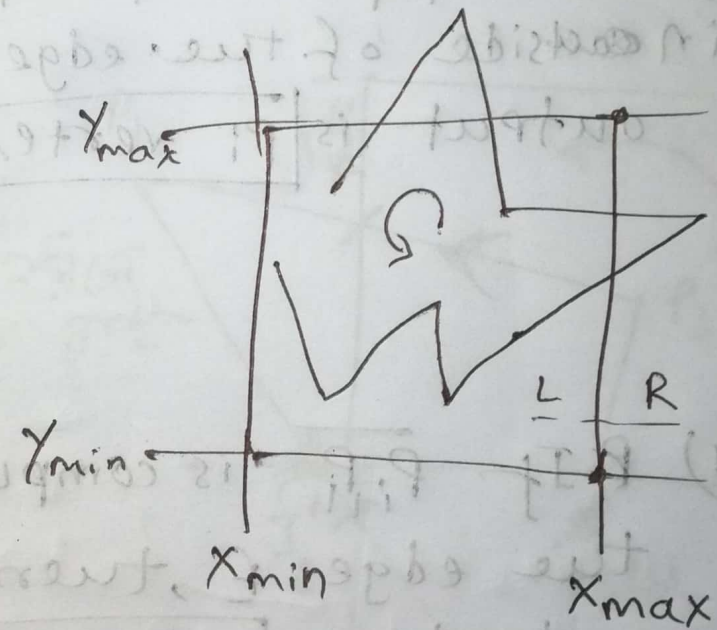
17.08.22
CG
M7I Sir

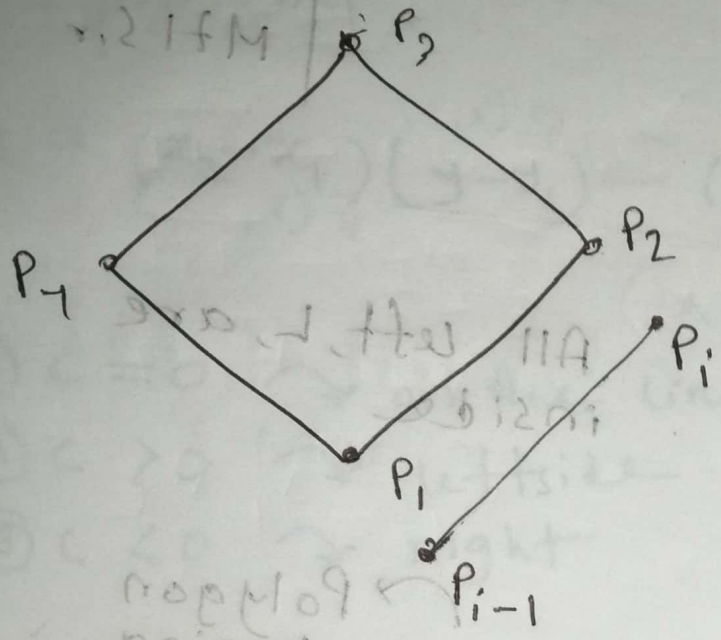


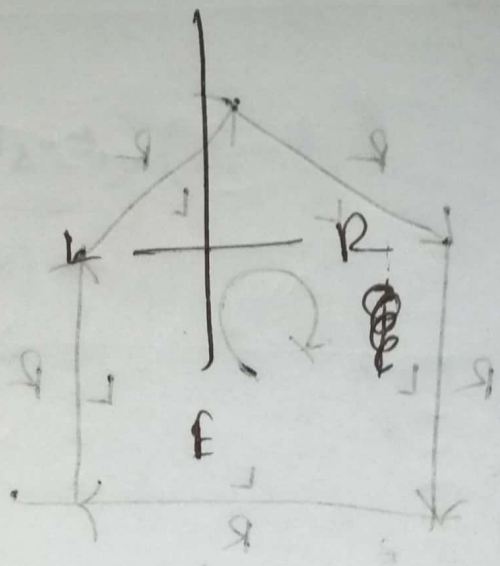
All left, L, are inside

↳ Polygon clipping

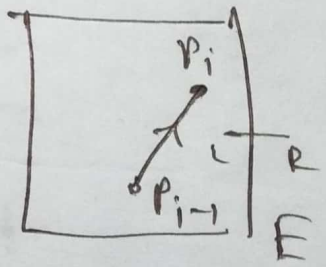
Sutherland - Hodgeman Algorithm



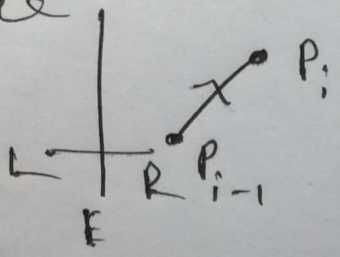




(i) If P_{i-1} and P_i, $\overline{P_{i-1}P_i}$, is completely inside of the edge E, then the output is [P_i vertex]

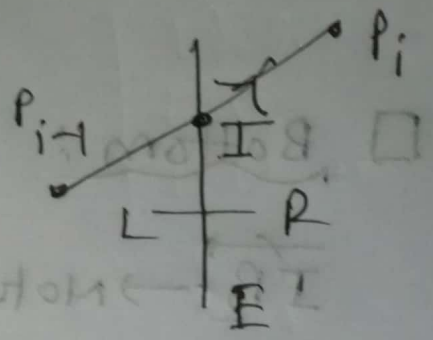


(ii) If $\overline{P_{i-1}P_i}$ is completely outside of the edge E, then the output is [nothing]



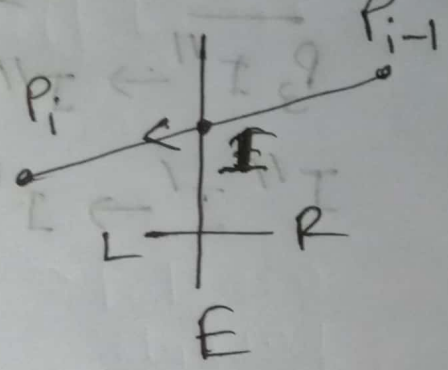
(iii) If $\overline{P_{i-1}P_i}$ is (in → out), output is I

intersection point I



(iv) If $\overline{P_{i-1}P_i}$ is (out → in)

Output is both I and P_i



✱ Example

□ Right edge

$\overline{P_1P_2} \rightarrow P_2$
$\overline{P_2P_3} \rightarrow P_3$
$\overline{P_3P_1} \rightarrow P_1$

নতুন করে draw করলে change নাই

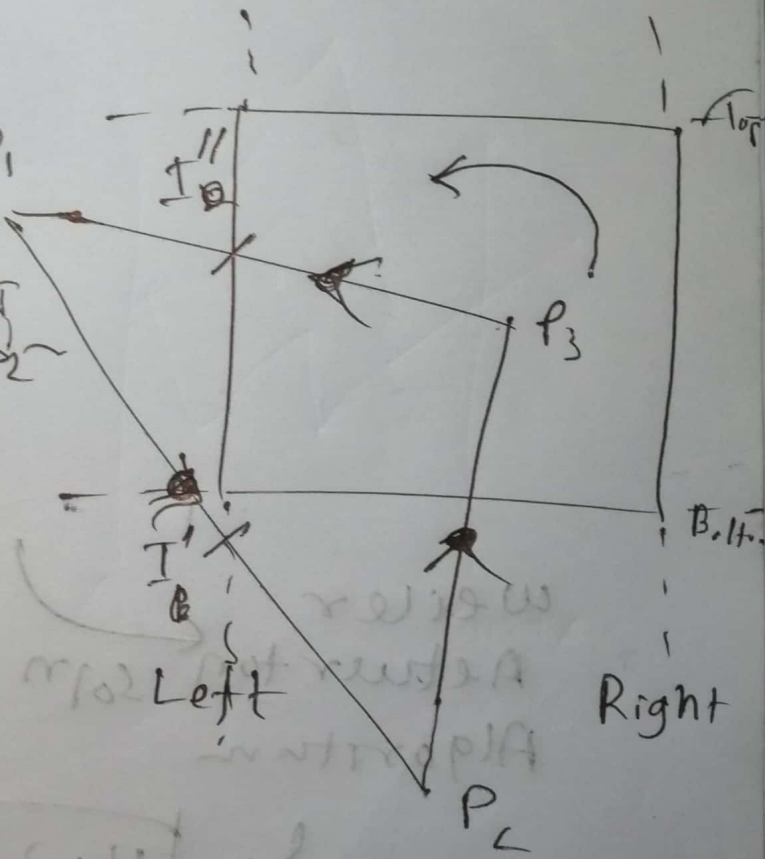
□ Top edge

same

□ Left edge

$\overline{P_1P_2} \rightarrow I'_B P_2$
$\overline{P_2P_3} \rightarrow P_3$
$\overline{P_3P_1} \rightarrow P_3, I''_B$

change



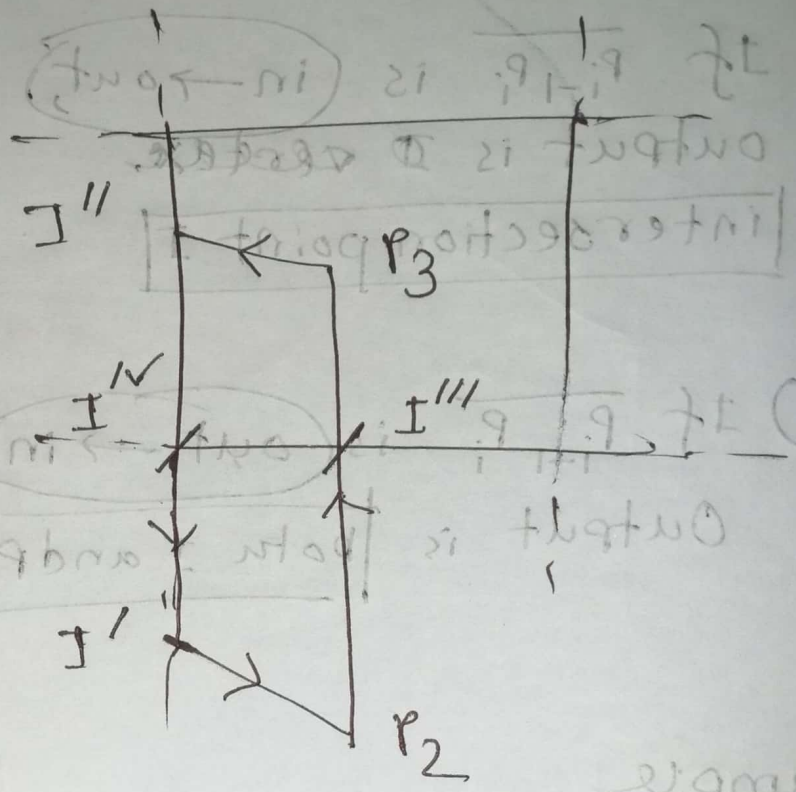
□ Bottom:

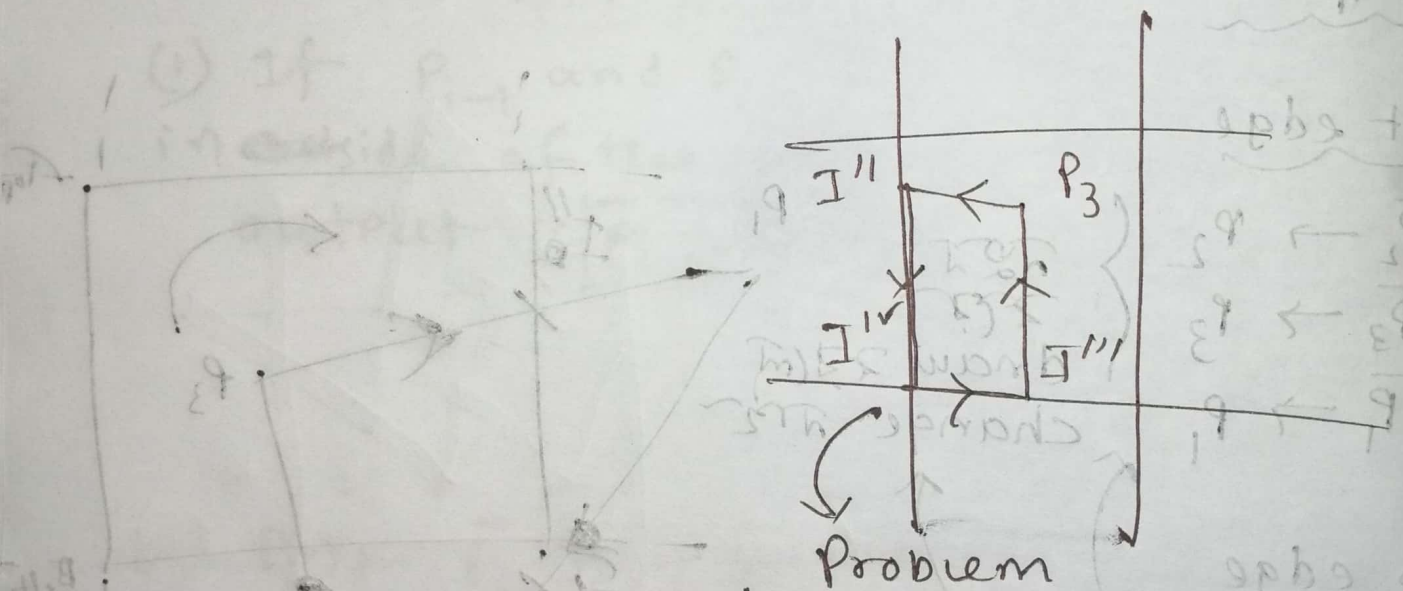
$\overrightarrow{I P_2}$ → Nothing

$\overline{P_2 P_3}$ → I''' – P_3

$\overline{P_3 I''}$ → I''

$I'' I'$ → I^{IV}





Problem
Extra Edge

Weiler
Atherton
Algorithm

soln

↳ HW

↳ Important