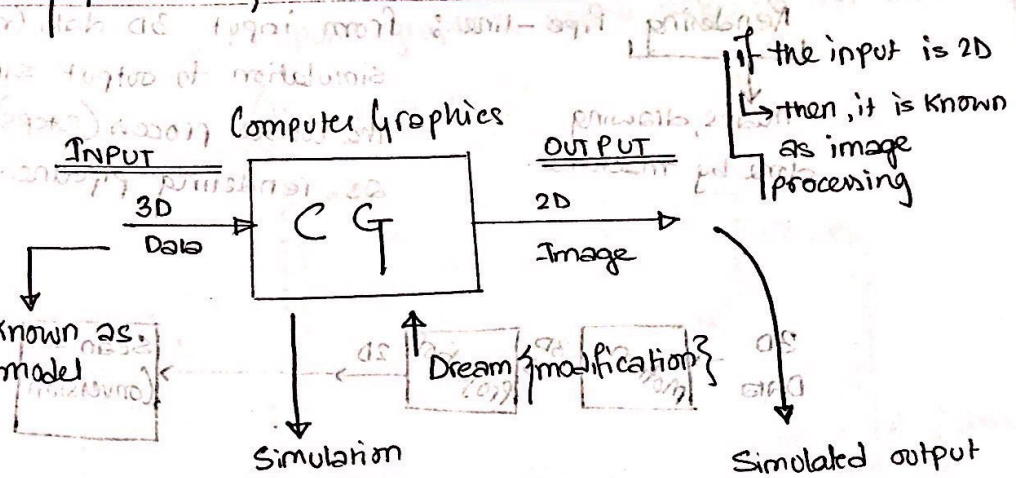
CSE 423 {GRAPHICS} lecture 1



The equation of the object to be simulated is called the model / 3D Data.

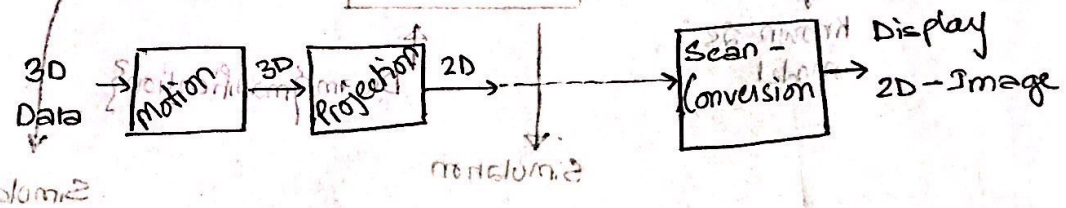
x, y)

y)

Rendering Pipe-line : from input 3D data (model) through simulation to output simulated 2D image.

→ means, drawing done by machine.

→ The whole process (steps) is known as rendering pipeline.

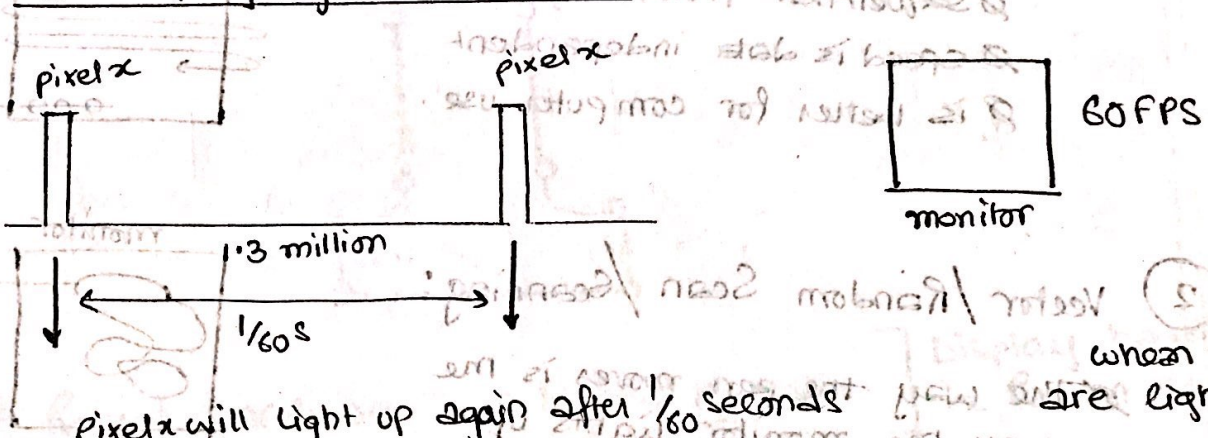


Hardware of display	Algorithm of display
∅ GPU	∅ GPU algorithm.
∅ Graphics card	

Display Hardware

- It is a part of computer graphics which takes in 2D input

Display System : (How it displays)



pixel x will light up again after 1/60 seconds when all pixels are lightened
→ known as one frame

- Since the pixels turn on and off so fast, it is not physically / visually perceptible to the naked eye; hence it feels like the monitor is always on.
→ 60 frames per second

Scanning :

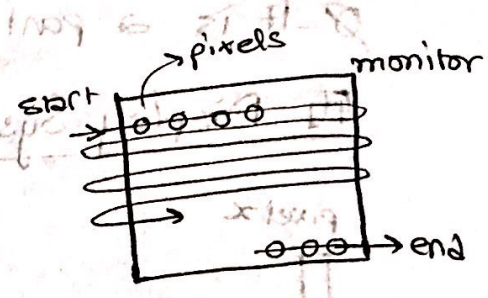
takes place → at the time of capturing & at the time of displaying

- a process that streams a 1D dot, from a 2D image.
- can use set of dots to create an image.

Scanning:

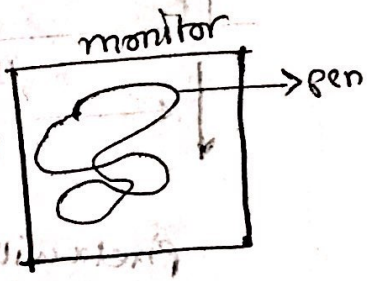
(1) Raster Scan / Scanning:

- sequential from start to end.
- speed is data independent
- is better for computer use.



(2) Vector / Random Scan / Scanning:

- The way the pen moves is the way the monitor lights up.
- Use of geometric equations to produce image.
- Vector scanning is good for line drawing, but not for field objects.
- It's speed is data dependent, for more data the speed slows down causing flickering.
- So, not good for computers.

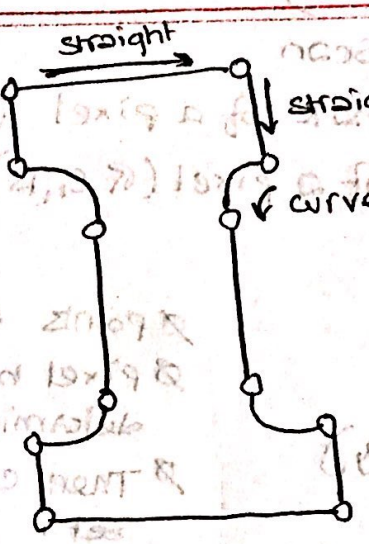


Advantages

- Ear is not dots
- it is relative
- if range is checked, we can manage size.
  ↳ The image won't be distorted when zoomed.

Example ⟶

Example:



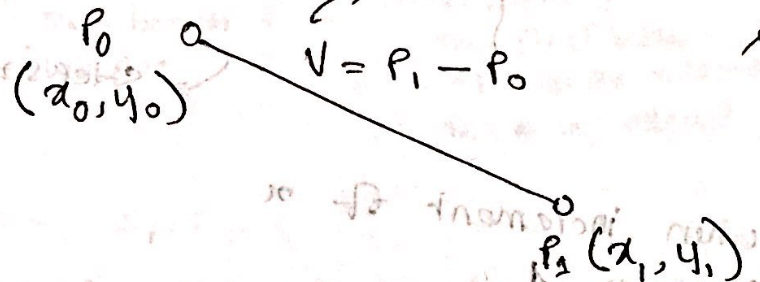
Creating boundary is work for vector graphics.

Recent hardware is a hybrid of both] Display hardware is Raster scan.

(Scan-conversion Algorithm :)
— Algo for Raster scan

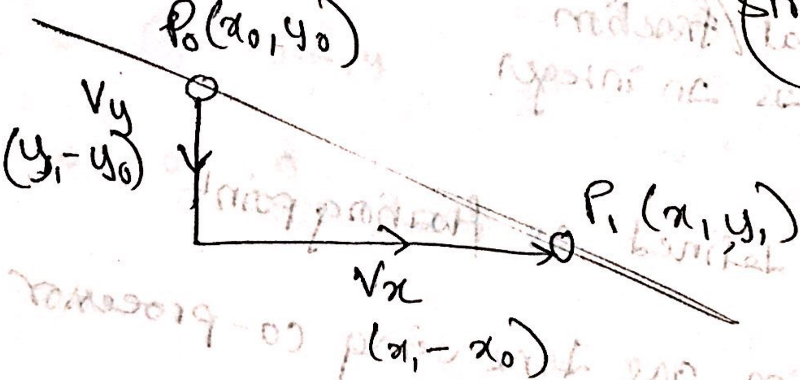
Output → 2 ज्ञा → ① Co-ordinate of a pixel (x,y)
② Color of a pixel.

Line-drawing Algorithm :



- points are 2D
- pixel have to be determined for line
- Then the color has to be set

- We need to set the start and end point
- The line we draw with computer is a vector, since direction is defined.



(Straight line and vector is same)

dx, dy { Carries direction and magnitude

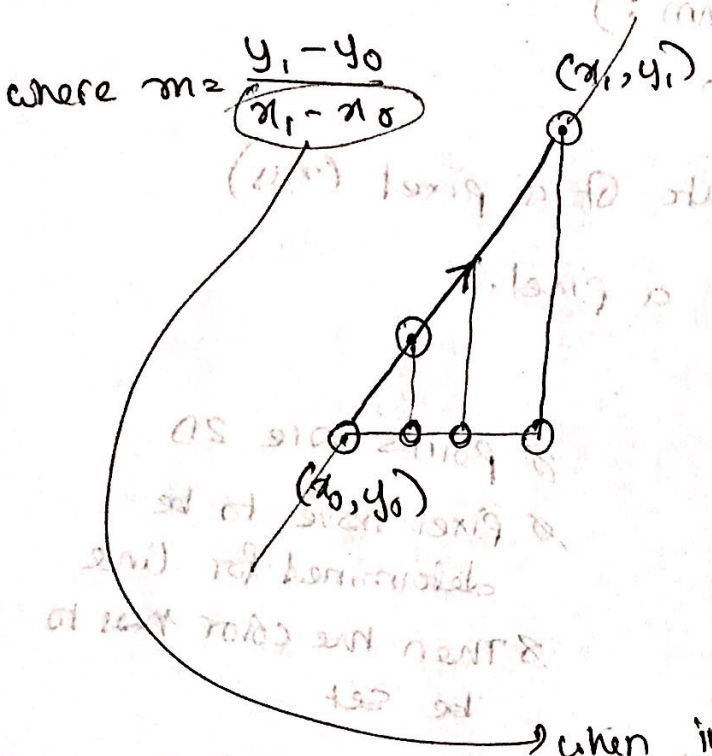
$dx \Rightarrow i|dx|$

$dy \Rightarrow j|dy|$

+ve /-ve defines the direction.

Vector and lines are synonyms.

DDA { Not Accepted as scan-conv Algo }.



when increment of x becomes 1,

then $m = y_1 - y_0$

so increment of y is equals m.

Line drawing Algo;

```
y = y0;
for (x = x0 to x <= x1) {
    draw pixel (x, y);
    y += m;
}
```

↳ step size = 1

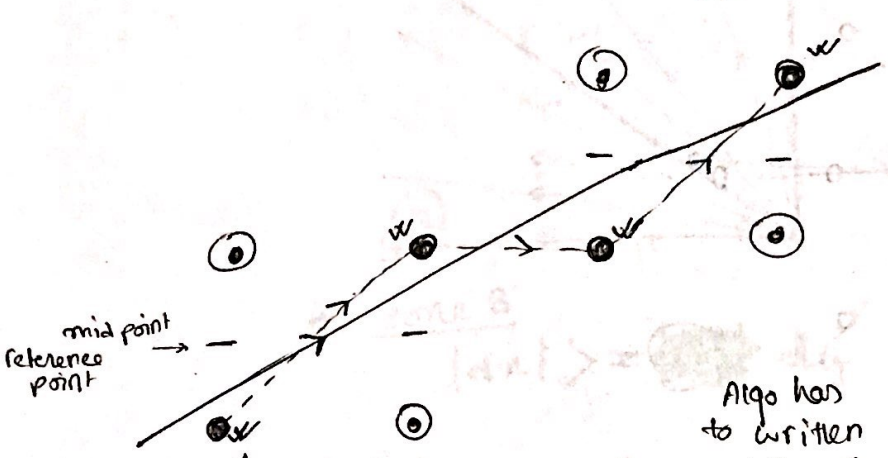
(1) m is decimal / fraction
→ y is taken as an integer
so
→ y must be defined as floating point
→ fp calculation are done using co-processor
→ and are expensive.

(2) Casting needs to be done;

(Mid-point line-drawing Algorithm:)

~~Created by~~

-Bresenham's Algo

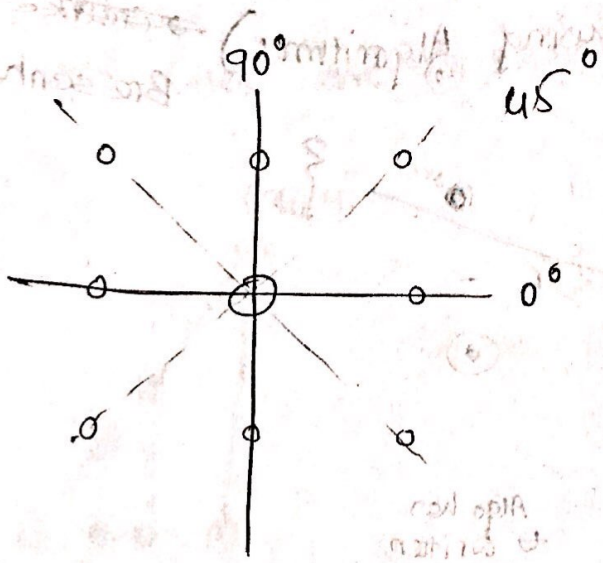


Algo has to written to define if the pixel below or over the midpoint has to be colored

For a pixel, what will be the next pixel?

A pixel has 8 neighbour pixel

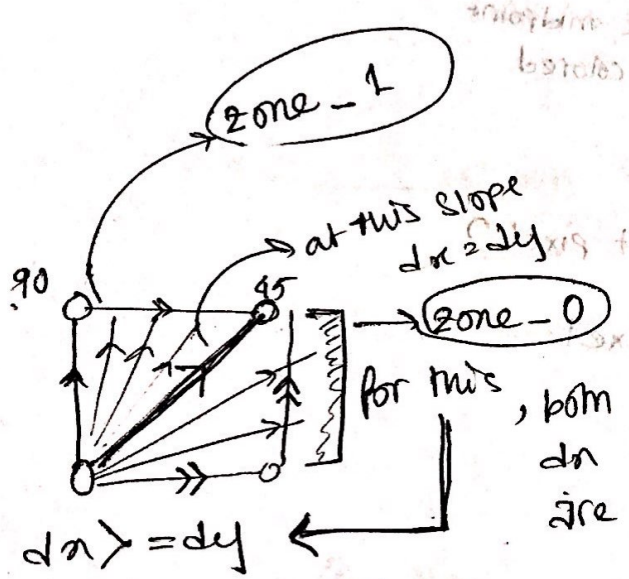
o	x,y	o+x,y
o		o+x,y
o	o	o



- −ve < 0
- +ve ≥ 0

0 to < 45 aka group

45 to < 90 aka group



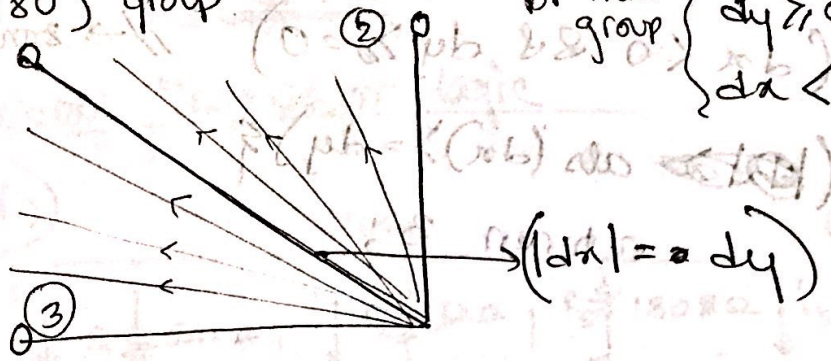
, both dx and dy are +ve

- $x_1 - x_0 = dx$
- $y_1 - y_0 = dy$

— has to be calculated from their value has to be determined the slope.

if dx >= dy ⇒
then zone 0;
else
zone 1;

Now, $(90-180)^\circ$ group



for NTS group: $dy \geq 0$, $dx < 0$ { +ve 0 is 0, -ve 0 is not 0 }

for zone 3

$|dx| >= dy$

else for zone 2

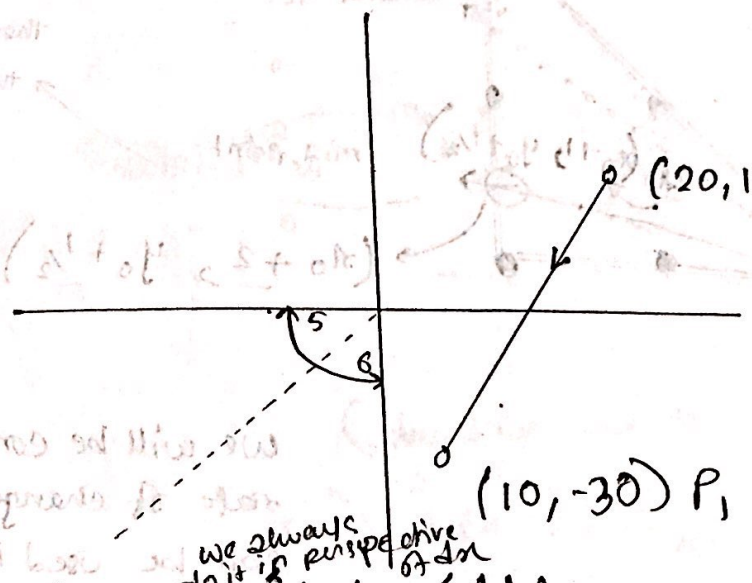
cond → $|dx| <= dy$

```
int find_Slope(int x0, int y0, int x1, int y1){
    int dx = x1 - x0, dy = y1 - y0;
    if (dx >= 0 && dy >= 0) { //(zone 0 & 1)
        if (dx >= dy){
            return 0 ;//→ we will actually call algos for all 8 zones to draw line
        }
        else {
            return 1
        }
        .
        .
        .
```

contd → ⊛

```
else if (dx < 0 && dy >= 0)        // → zone 2 & 3
    if (abs(dx) >= dy) {
        return 3;
    }
    else {
        return 2;
    }
}
```

CSE 423 (Graphics) Day 4



$dx = 10 - 20 = -10$

$dy = -30 - 10 = -40$

we always do it in perspective of dx

$|dx| < |dy|$

Abs(-10) < Abs(-40) } Hence, Zone 6.

Implicit form } Eq of a Line,

$y = mx + c$ ---(i) $\quad m = dy/dx$

Explicit form }

$dy\,x - dx\,y + dx\,c = 0$ } → multiply all with (dx).

↓

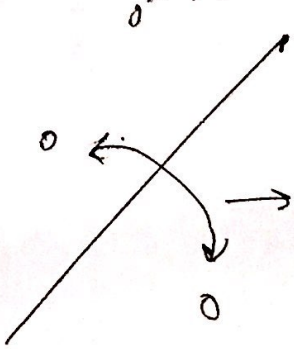
$Ax + By + C = 0$ ---(ii)

where,

$A = dy$

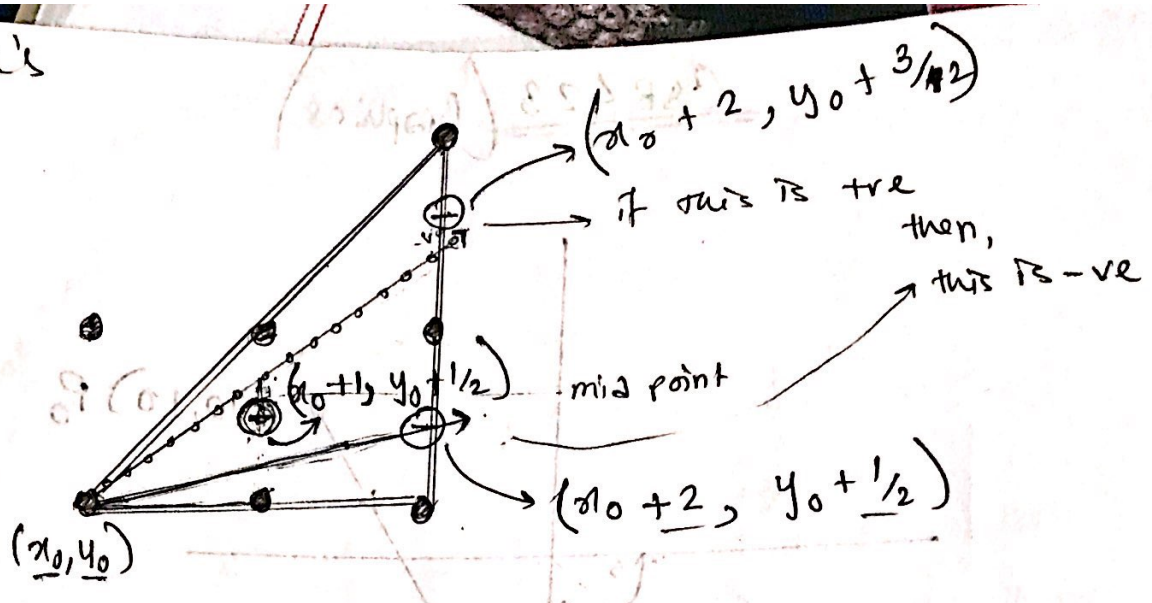
$B = -dx$

0 → This will be larger }



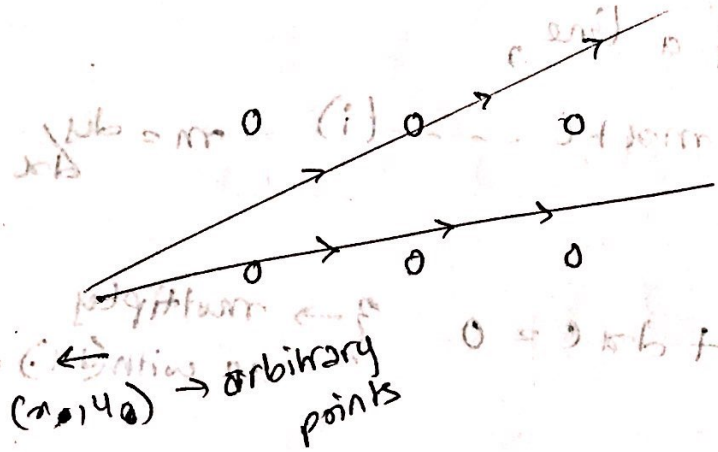
→ have opposite signs.

Bresenham's

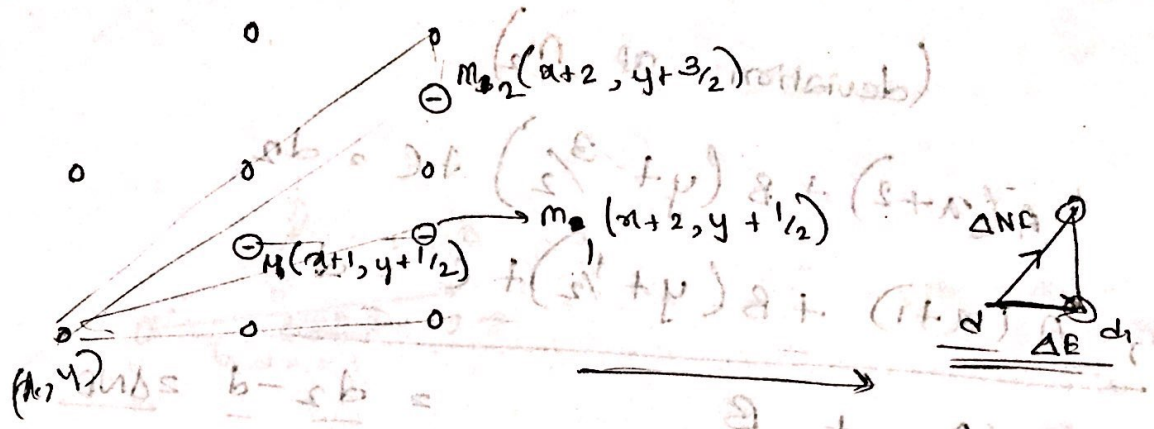


We will be considering rate of change which can be used to locate the pixel further in the path.

for,



(ii)

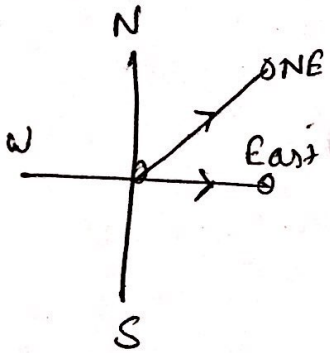


(deviation at M_1)

$$A(x+2) + B(y+1/2) + C = d_1 \quad \text{①}$$

at M; $A(x+1) + B(y+1/2) + C = d$

$$A + 0 + 0 = d_1 - d = \Delta E$$



for a horizontal movement the rate of change of pixel.

$$\therefore \boxed{\Delta E} = \boxed{A} = \boxed{dy}$$

(deviation at n_2)

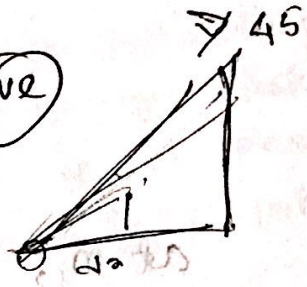
$$A(x+2) + B\left(y+\tfrac{3}{2}\right) + C = d_2$$

at m, $A(x+1) + B(y+1/2) + C = d$

$$A + B = d_2 - d = \Delta NE$$

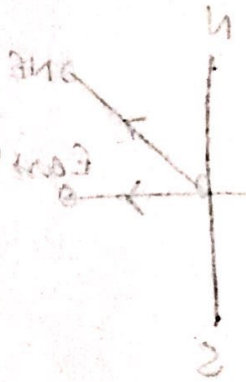
ΔNE

$\therefore \Delta NE = A + B = dy - dx$ → (-ve)



and ΔE

$\Delta E = dy$ → (+ve)



$$A(x_0+1) + B(y_0+1/2) + C = dinit$$

$$\Rightarrow \boxed{A + B/2 = dinit}$$

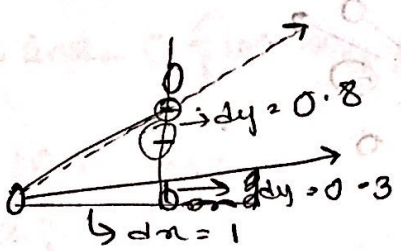
{ Since, $Ax_0 + By_0 + C = 0$ ∴

$$Ax_0 + A + By_0 + B/2 + C = dinit$$

$$\boxed{dinit = A + B/2 = dy - \frac{dx}{2}}$$

$$\Rightarrow A + B/2 + 0 = dinit$$

$$dinit = A + B/2$$



No matter how big this line is

for solid,

dinit = $dy - dx/2$

$= 0.3 - 0.5$

$= -0.2$

for dotted

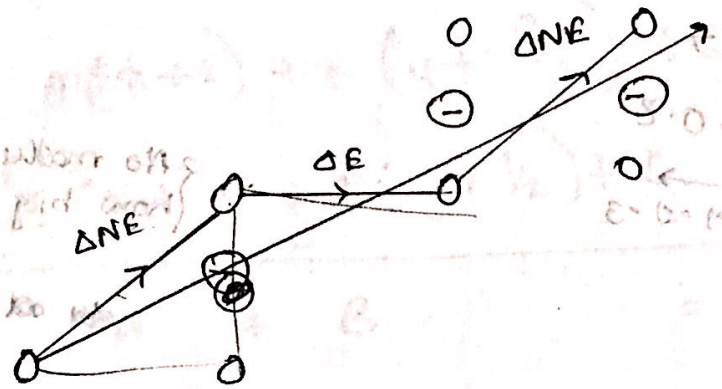
dinit = $0.8 - 0.5$

$= 0.3$

∴ if dinit is +ve then the next pixel is the upper pixel

if dinit is -ve

next pixel is the lower pixel




```
void drawline (int x0, int y0, int x1, int y1){
int dx = x1 - x0, dy = y1 - y0;
int d = 2dy - dx //dinit
int dE = 2dy, dNE = 2(dy - dx);
int y = y0, x = x0;
drawPixel (x,y); // the first pixel must be drawn outside of the loop bec
// while (x < x1) {
    if (d<0) { //ΔE
        x++;
        d += dE;
    else {
        y++;
        x++;
        d += dNE;
    }
    drawPixel (x,y);
}
```

Since for zone 0 x has definite change.

Q (30,50) to (40,54) { so 10 pixel by 10 pixel size.

void drawline_0 {int x_0, y_0, x_1, y_1}

$dx = 10$, $dy = 4$

$d = -2$;

$dE = +8$; $dNE = -12$

since only x++

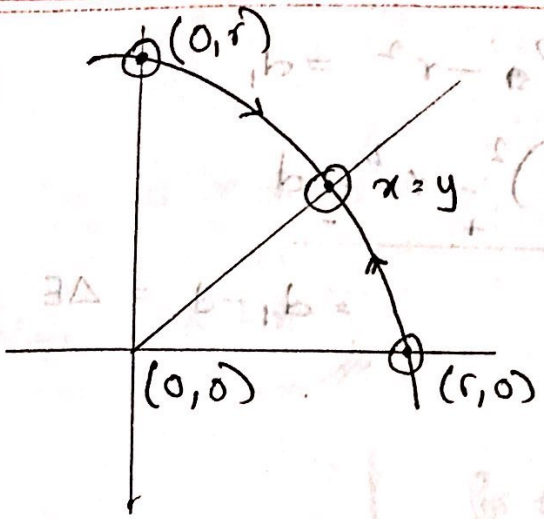
x	y	d	ΔE/ΔNE
30	50	-2	ΔE
31	50	6	ΔNE
32	51	-6	ΔE
33	51	2	ΔNE
34	52	-10	ΔE
35	52	-2	ΔE
36	52	-6	ΔNE
37	53	-6	ΔE
38	53	2	ΔNE
39	54	-10	ΔE
40	54	-2	ΔE

+ΔE

dinit = dinit + dE

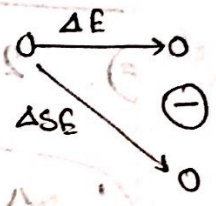
{ no redundant calculation.

Circle Drawing Algorithm — lecture 5

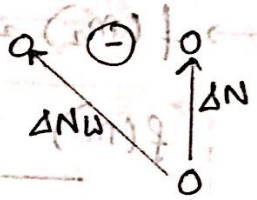


Operations

① Starting from (0,r)



② Starting from (r,0)



$$f(x,y) = x^2 + y^2 - r^2 = 0 \quad \text{---(1)}$$

② $m_2 (x-3/2, y+2)$ $m_1 (x-1/2, y+2)$ $m (x-1/2, y+1)$ (x,y)

$m_2 = (x-1/2, y+1)$

$m_1 = (x-1/2, y+2)$

$m_2 = (x-3/2, y+2)$

① (x,y) $m (x+1, y-1/2)$ $m_1 (x+2, y-1/2)$ $m_2 (x+2, y-3/2)$

$m = (x+1, y-1/2)$

$m_1 = (x+2, y-1/2)$

$m_2 = (x+2, y-3/2)$

$(a-b)^2 = a^2 - 2ab + b^2$

for Operation 1

$\rightarrow f(m_1) = (x+2)^2 + (y-1/2)^2 - r^2 = d_1$

$\rightarrow f(m) = (x+1)^2 + (y-1/2)^2 - r^2 = d$

$= d_1 - d = \Delta E$

$= 2x + 3$

$\therefore \Delta E = 2x + 3$

$\rightarrow f(m_2) = (x+2)^2 + (y-3/2)^2 - r^2 = d_2$

$f(m) = (x+1)^2 + (y-1/2)^2 - r^2 = d$

$= d_2 - d = \Delta SE$

$2x + 3 - 2y + 2$

$= 2x + 5 - 2y$

$= \Delta SE$

$\therefore \Delta SE = 2x - 2y + 5$

$dinit = (x_0+1)^2 + (y_0 - 1/2)^2 - r^2$

$= (x_0)^2 + 2x_0 + 1 + (y_0)^2 - y_0 + 1/4 - r^2$

$= 2x_0 + 1 - y_0 + 1/4$

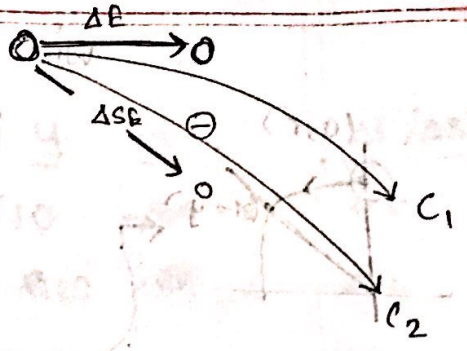
$= 2x_0 - y_0 + 5/4$

Since $x_0 = 0$ and $y_0 = r$

This, is bec start point is $(0, r)$

~~# for operation 2~~

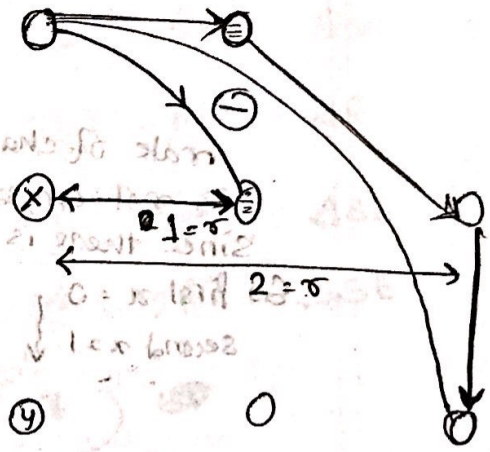
#



⇒ $5/4 - r = d_{init}$

⇒ $d_{init} = 5/4 - r$

ΔSE goes over mid point	for $r = 1$ → we get d_{init} +ve
ΔE goes underneath mid point	for $r = 2$ or greater → d_{init} = -ve



for $r = 1$ (small goes underneath)

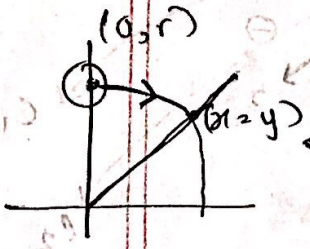
for $r = 2$ / greater (big goes over)

done to remove fraction.

```
void drawCircle_1 (int r){
int d = 5 - 4*r ;
int x = 0 , y = r ;
drawEight (x,y);
while (x<y) {
    if (d<0) {
        d+ = 4*(2x + 3);
    }
    else {
        d+ = 4*(2x - 2y + 5);
        x++;
        y--;
    }
    drawEight (x,y);
}//while()
}//end()
```

bec we mult by 4 to remove fraction other times d needs to (*4)

also :



rate of change is not constant since there is x.

So first x = 0

second x = 1

x increases 1

y decreases 1

start point

(0, 10) radius = 10

x	y	d	ΔE/ΔSE
0	10	-35	ΔE
1	10	-23	ΔE
2	10	-3	ΔE
3	10	25	ΔSE
4	9	-11	ΔE
5	9	33	ΔSE
6	8	21	ΔSE
7	7		

→ (7, 7) draw hoye sesh hoye jabe algo/

P.T.O →

* Operation 2 (start from (r,0))

$f(m_1) = (x-1/2)^2 + (y+2)^2 - r^2 = d_1$

$f(m) = (x-1/2)^2 + (y+1)^2 - r^2 = d$

$= d_1 - d = \Delta N$

$\Rightarrow 2y + 3$

$\therefore \Delta N = 2y + 3$ (x4)

$f(m_2) = (x-3/2)^2 + (y+2)^2 - r^2 = d_2$

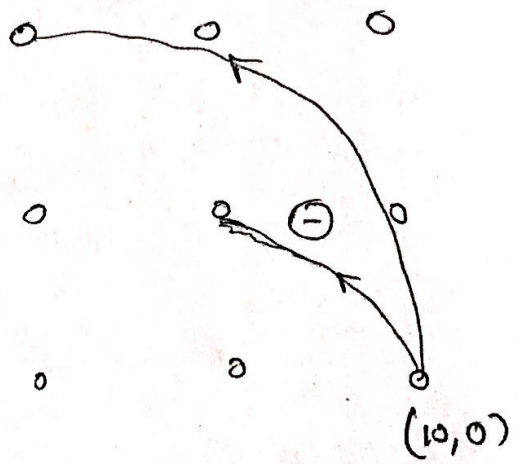
$f(m) = (x-1/2)^2 + (y+1)^2 - r^2 = d$

$= d_2 - d = \Delta NW$

$-2x + 2 \qquad 2y + 3$

2) $\Delta NW = 2y - 2x + 5$ (x4)

$\therefore$ dinit $= \left(5/4 - r\right)$ (x4)



So,

$r = 1 \longrightarrow$ d +ve $\rightarrow \Delta NW$ (y++, x--)

$r > 1 \rightarrow$ and, d -ve $\rightarrow \Delta N$ (y++)

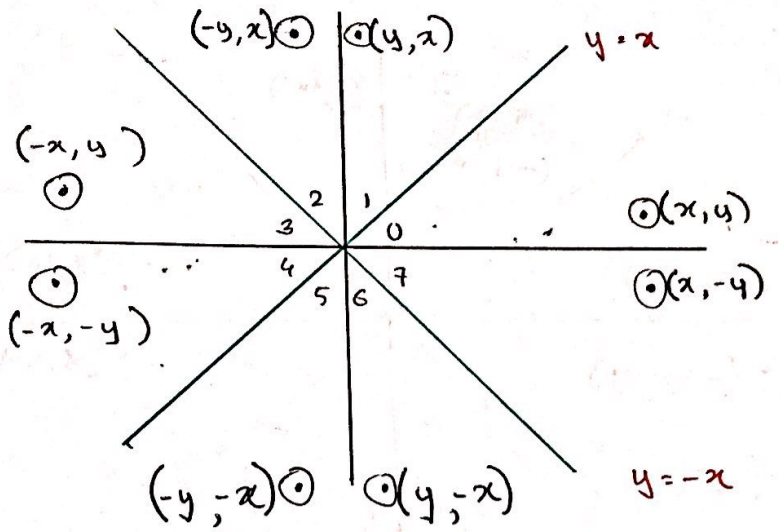
st. point (10, 0)

code

```
while (y<x) {
if (d<0) {//ΔN
    d+ = (ΔN)x4
    y++
}
else {ΔNW
    d+ = (ΔNW)x4
    y++
    x--
}//
```

x	y	d	ΔN/ΔNW
10	0	-35	ΔN
10	1	-23	ΔN
10	2	-3	ΔN
10	3	25	ΔNW
9	4	-11	ΔN
		33	
		21	

8-way Symmetry



Defn:

zone division

The line drawing code for each individual zone is not optimized, since, we need to write a lot of redundant lines of code.

Hence, we use 8-way symmetry.

Explanation

If a point is selected at zone 0, as (x, y) then we can derive its corresponding coordinates in the other zones by using reflection in the lines $(y = x)$ & $(y = -x)$

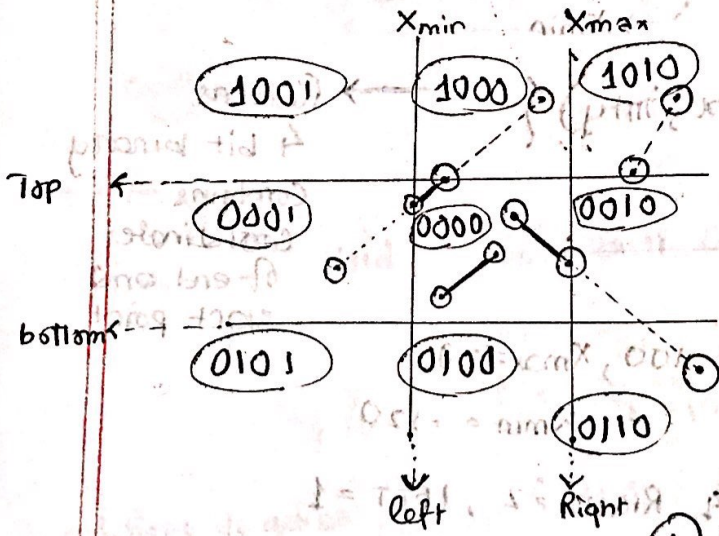
From the above figure it can be determined that any point in another zone can be traced back to zone 0 (x,y) pattern, by going through the corresponding rearrangement of the x & y coordinates in the other zones.

How algo works.

So, if we can derive the mid point line drawing algorithm for zone 0. We can use it draw a line in any other zone, just by rearrangement of the coordinates in any other zone as shown in figure above.

So we transform the points to zone 0. find the next pixel and by rearrangement, draw the pixel in its actual zone.

Line - Clipping Algorithm :-

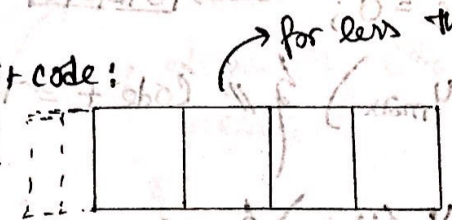


Only the line remaining inside the frame is drawn

for z axis (3D) has 27 layers

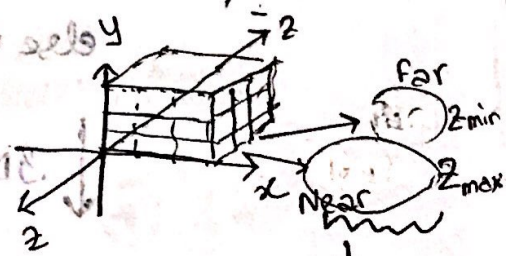
(1) Cohen-Sutherland LCA

Region Outcode



has 2 more bits meaning 6 bits for 3D modelling

for greater than ymax then → 1 else → 0

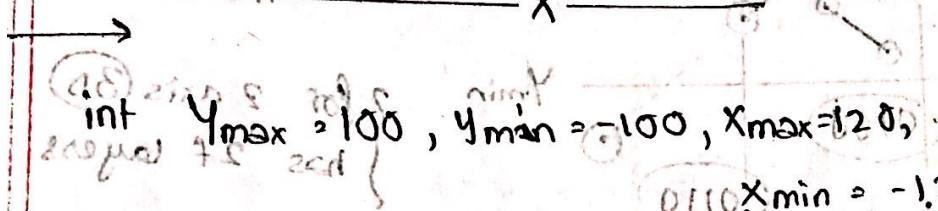


→ Ymax → Ymin

```
int TOP = 100, Bottom = -100
    RIGHT = 120, LEFT = -120
              → Xmax   → Xmin
```

```
int MakeCode (int x, int y) {
    int Code = 0;
```

→ Returns 4 bit binary contains coordinates of end and start point




```
int Ymax = 100, Ymin = -100, Xmax = 120;
```

Xmin = -120

```
int TOP = 8, Bottom = 4, RIGHT = 2, LEFT = 1

int MakeCode (int x, int y) {
    int Code = 0;
```

→ 0 0 0 0

1 0 0 0

```
    if (y > Ymax) { // Code += TOP;
```

→ meaning 4th maximum bit will become 1

```
    else if (y < Ymin) { // Code += Bottom;
```

→ 0 1 0 0

```
    if (x > Xmax) { // Code += RIGHT;

    else if (x < Xmin) { // Code += LEFT;
```

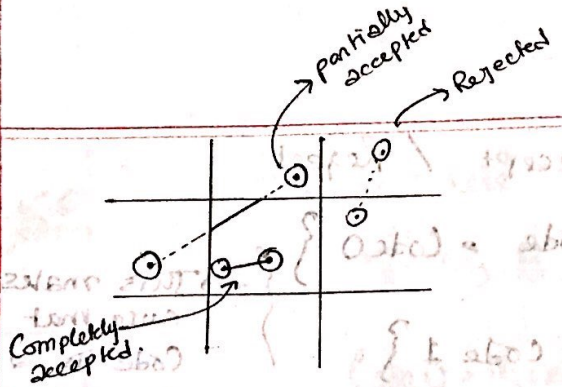
3D note {near & far 2 bits will be added}

```
    return Code;
```

32 → Zmax

16 → Zmin

}



Code →

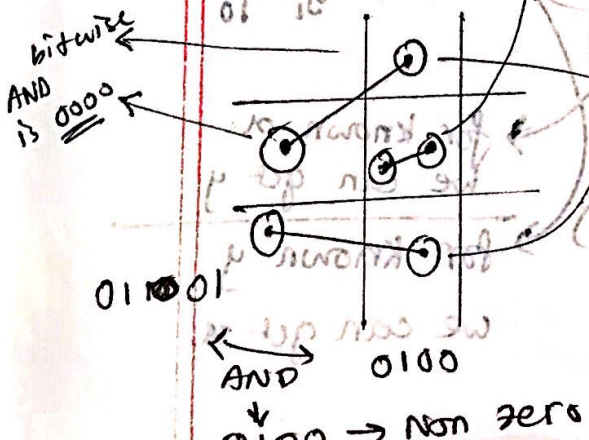
```
Void Cohen_Sutherland (int x0, int y0, int x1, int y1)
{
    int Code, Code0, Code1;
    int x, y;
    Code0 = MakeCode (x0, y0)   → start point
    Code1 = MakeCode (x1, y1)   → end point
    while(1){                   → bit wise OR
        if (Code0 | Code1 == 0){ // complet accept
            drawLine (x0, y0, x1, y1)
            break;
        }
                                → bit wise AND
        else if (Code0 & Code1){ // true hoile complet reject
            break;
        }
        else { // Partially Accept / Reject
```

3 Conditions to get out of loop:

① Complete accept → drawline & break

② Complete reject → Do nothing & break

③ Partial → Clip, draw line & break



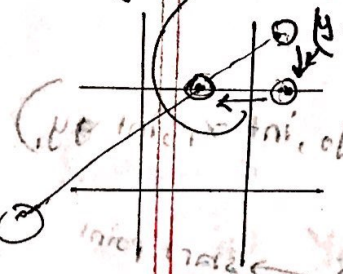
(*) Cont'd → pto

else {//Partially Accept / Reject

if (Code 0){Code = Code 0} → This makes sure that Code is non-zero value.

else {code = code 1}

now we need an x value which is on the line



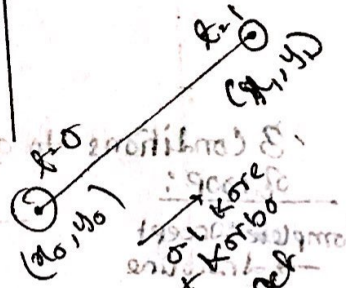
if (Code & TOP) {//y > ymax

y = ymax;

x = x0 + (ymax - y0)/(y1 - y0) (x1 - x0);

x = x0 + t(x1 - x0)

y = y0 + t(y1 - y0)



else if (Code & BOTTOM) {//y < ymin

y = ymin;

x = x0 + (ymin - y0)/(y1 - y0) (x1 - x0); t's value is put in → y = y0 + (x - x0)/(x1 - x0) (y1 - y0)

}

else if (Code & RIGHT) {//x > xmax

x = xmax;

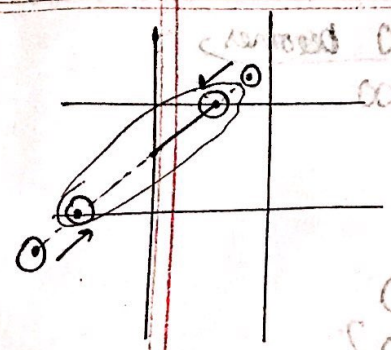
y = y0 + t(xmax - x0)/(x1 - x0) (y1 - y0)

} else {//

→ same for xmax & xmin → x = x0 + (y - y0)/(y1 - y0) (x1 - x0)

for known x we can get y

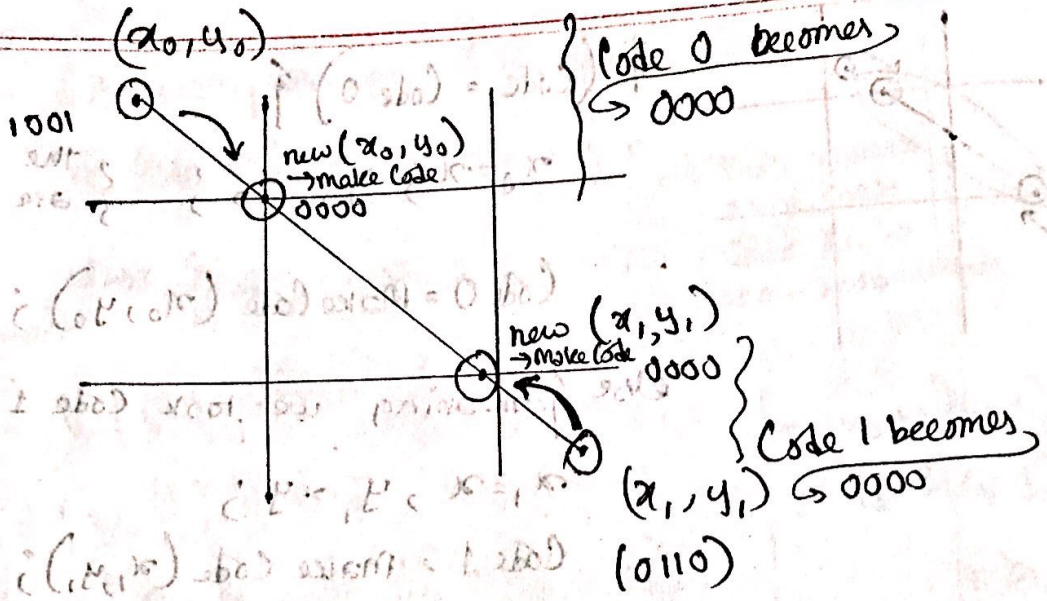
for known y we can get x



```
if (Code1 = Code0) {
    x0 = x; y0 = y;        } the x and y we got are the new x0 & y0

    Code0 = MakeCode(x0, y0);
else {//meaning we took Code 1
    x1 = x, y1 = y;
    Code1 = MakeCode(x1, y1);
}

}//else → the partial else

}//while

}//end
```

Summary;

- → will be clipping and making code again to make it fit inside frame
- → once inside frame code becomes 0 and now becomes fully accepted.


(x0, y0)
1001
new (x0, y0)
→ make code
0000
Code 0 becomes
0000
new (x1, y1)
→ make code
0000
Code 1 becomes
0000
(x1, y1)
(0110)

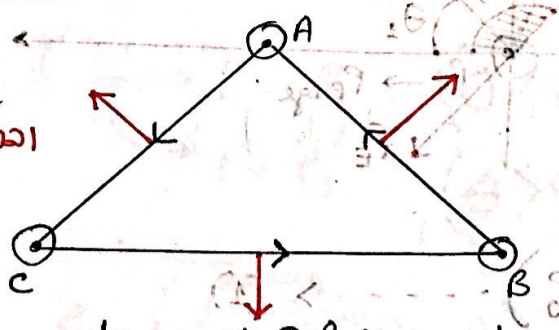
Lecture 7

Line Clipping Algorithm

1. Cohen, Sutherland LCA → uses makeCode
2. Cyrus - Beak → uses &

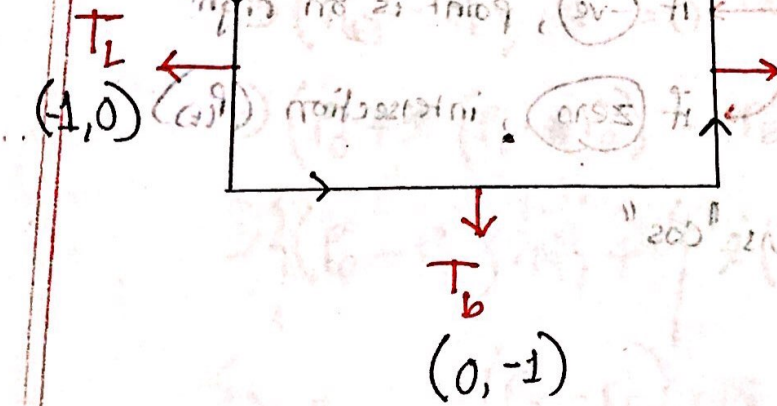
Cyrus Beak Line Clipping Algorithm:

Gradient vector OR Surface normal



* Considering the direction of the edge, anything on the left is inside & right is outside

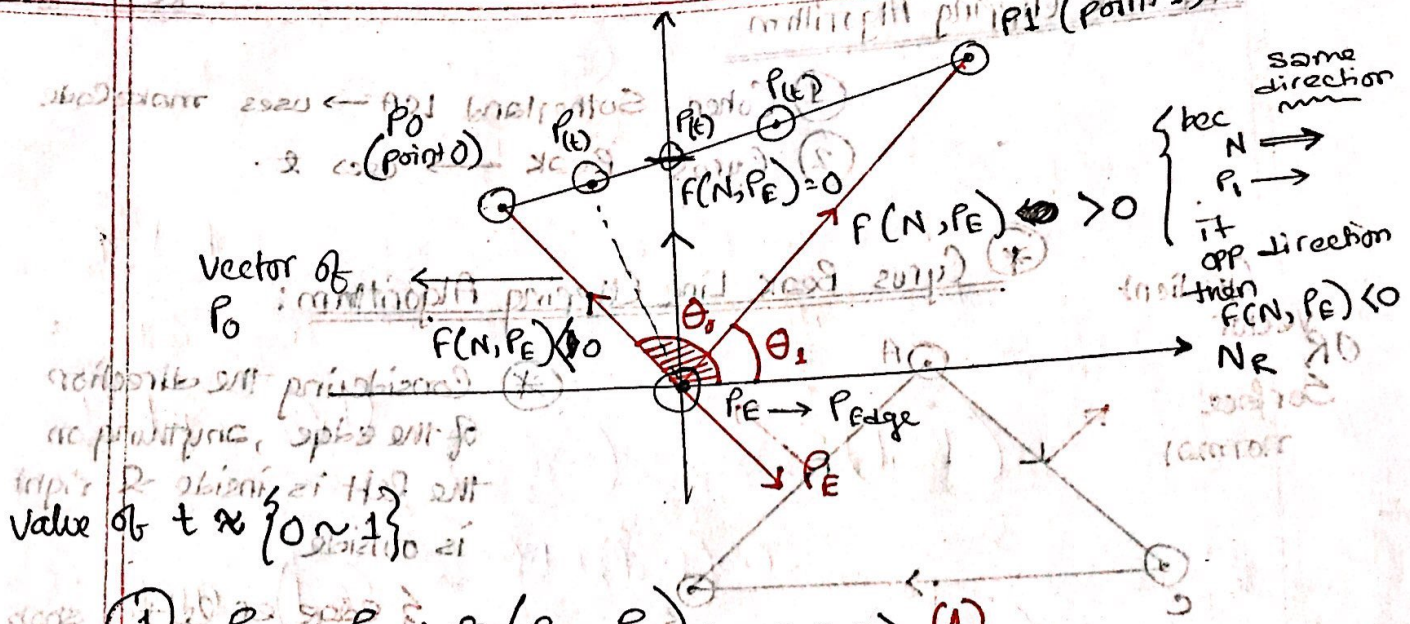
{ edge er left e shob inside, anything on the right will be considered to be outside.



	Values	
T_T	(0,1)	top
T_B	(0,-1)	bottom
T_L	(-1,0)	left
T_R	(1,0)	right

$f(N, P_E) = N \cdot (P - P_E)$ ← dot prod

θ = Angle between vector & N



Value of $t \approx \{0 \sim 1\}$

(1) $P(t) = P_0 + t(P_{0_1} - P_0)$ ----- (1)

↳ parametric eqn of a line in the view plane.

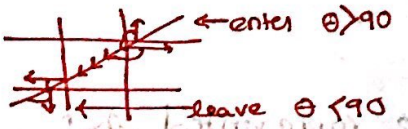
at $t=0$, $P(t) = P_0$ & at $t=1$, $P(t) = P_1$

(2) $(P - P_E) \cdot N =$ if (+ve), Point is on left
if (-ve), Point is on right
if (zero), intersection ($P(t)$)

(0,1) (0,1-) (1-,0)

dot product considers "cos"

(1-,0)



(Considerations)

$(P-P_E)\cdot N > 0$ (i) When θ is less than 90 $(\theta < 90^\circ)$ → LEAVING FRAME

$(P-P_E)\cdot N < 0$ (ii) When θ is greater than 90 $(\theta > 90^\circ)$ → ENTERING FRAME

$(P-P_E)\cdot N = 0$ (iii) When θ is equal to 90 $(\theta = 90^\circ)$ → On the boundary line

if → $P = P_1$ $(\theta < 90^\circ)$

$P = P_0$ $(\theta > 90^\circ)$

⋮ → intersection of boundary

$$P_{(t)} = P_0 + t(P_1 - P_0) \quad \text{(i)}$$

when $P = P_{(t)}$ → $(P-P_E)\cdot N = 0$ (ii)

substituting $P = P_{(t)}$

$$\Rightarrow (P_{(t)} - P_E)\cdot N = 0$$

$$\Rightarrow (P_0 + t(P_1 - P_0) - P_E)\cdot N = 0$$

$$\Rightarrow \{(P_0 - P_E)\cdot N\} + \{t(P_1 - P_0)\cdot N\} = 0$$

$$\Rightarrow t = \frac{-(P_0 - P_E)\cdot N}{(P_1 - P_0)\cdot N}$$

→ the equation used to find the point of intersection of line & edge

t, can be represented as:

$$t = \frac{(P_E - P_0)\cdot N}{(P_1 - P_0)\cdot N}$$

but well consider the other eqn:

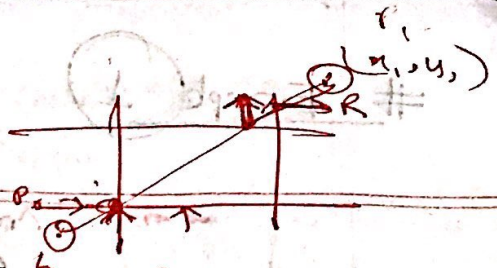
(*) List of t for all possible boundaries:

	Boundary	N	(*) $(P_0 - P_E)\cdot N$	$(P_1 - P_0)\cdot N$	t (*) (-ve matt)
1	TOP	$(0,1)$	$(x_0 - x_E)\cdot 0 + (y_0 - y_{max})\cdot 1$ $\rightarrow (y_0 - y_{max})$	$(x_1 - x_0)\cdot 0 + (y_1 - y_0)\cdot 1$ $\rightarrow (y_1 - y_0)$	$\frac{y_{max} - y_0}{y_1 - y_0}$
2	RIGHT	$(1,0)$	$(x_0 - x_{max})\cdot 1 + (y_0 - y_E)\cdot 0$ $\rightarrow (x_0 - x_{max})$	$\rightarrow (x_1 - x_0)$	$\frac{x_{max} - x_0}{x_1 - x_0}$
3	BOTTOM	$(0,-1)$	$\rightarrow -1(y_0 - y_{min})$	$-1(y_1 - y_0)$	$\frac{y_{min} - y_0}{y_1 - y_0}$
4	LEFT	$(-1,0)$	$\rightarrow -1(x_0 - x_{min})$	$-1(x_1 - x_0)$	$\frac{x_{min} - x_0}{x_1 - x_0}$

(-ve 1 will create no effect.)

(*) $(P_0 - P_E)\cdot N = (x_0 - x_E)\cdot N_x + (y_0 - y_E)\cdot N_y$

(*) $(P_1 - P_0)\cdot N = (x_1 - x_0)\cdot N_x + (y_1 - y_0)\cdot N_y$



(*) Explanation for derivation of t

$$(P_E - P_0) \cdot N$$

For top, $N = (0,1)$ & $P_E = x, y_{max}$

$$\Rightarrow [(x, y_{max}) - (x_0, y_0)] \cdot (0,1)$$

$$\Rightarrow (x - x_0) \cdot 0 + (y_{max} - y_0) \cdot 1$$

$$\Rightarrow (y_{max} - y_0)$$

Also $(P_1 - P_0) \cdot N \Rightarrow (x_1 - x_0) \cdot 0 + (y_1 - y_0) \cdot 1$

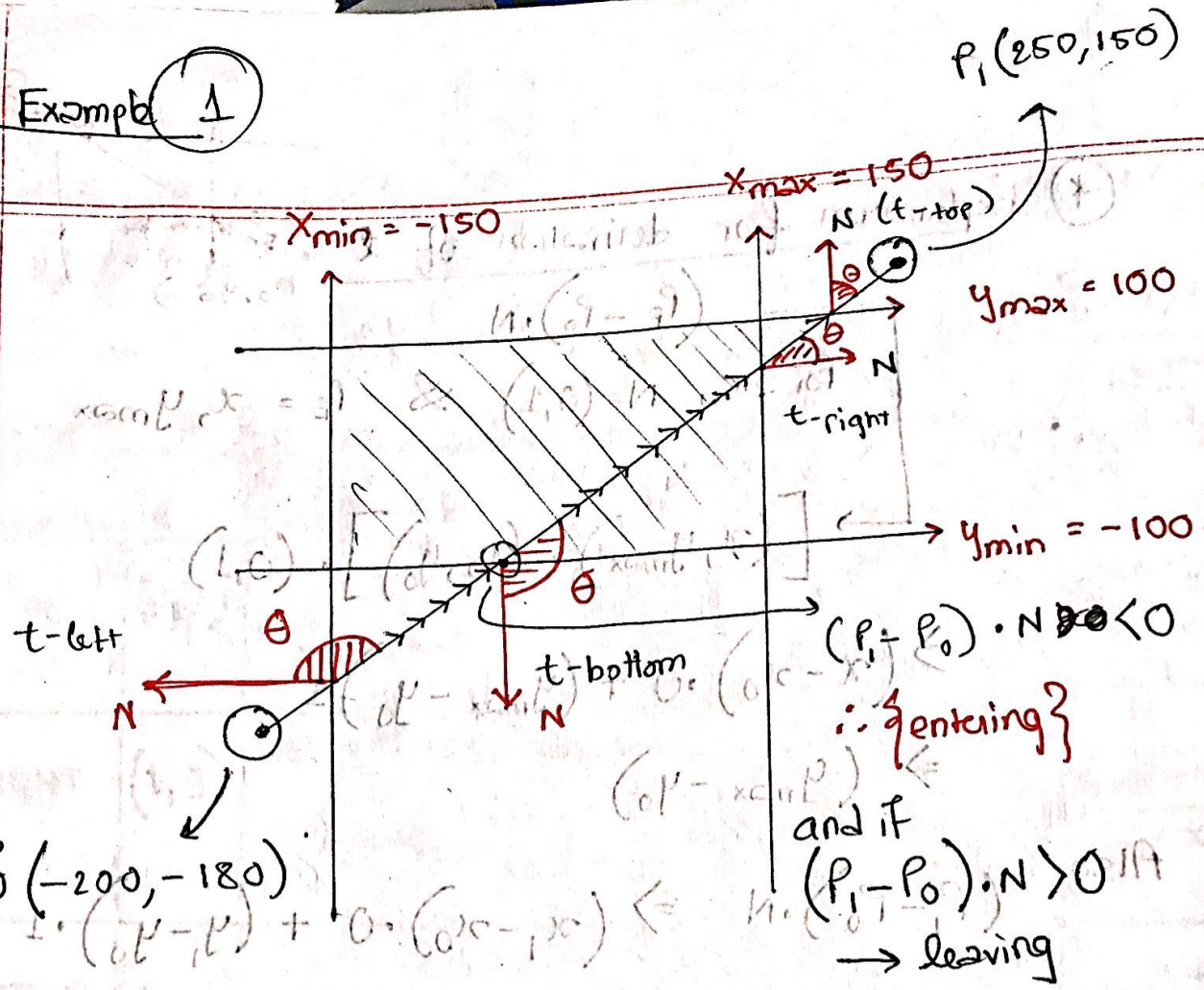
$$\Rightarrow (y_1 - y_0)$$

Hence,

$$t = \frac{y_{max} - y_0}{y_1 - y_0}$$

When, calculating for bottom & left the -1 won't create any change except for showing that P_E is either $\rightarrow y_{min}$ / $\rightarrow x_{min}$.

Example 1



We know that,

$\theta > 90 \Rightarrow$ entering

$\theta < 90 \Rightarrow$ leaving

Here,

(t_E) the entering t's are → {t_{left} & t_{bottom}}

(t_L) the leaving t's are → {t_{top} & t_{right}}

(*) Cyrus beck said that the line is drawn from

$t_E(max)$ to $t_L(min)$

$t_{E(max)} = 0.2424$ → $t_{1(min)} = 0.778$

t_{bottom} t_{right}

So,

$$t_{(top)} = \frac{y_{max} - y_0}{y_1 - y_0} = \frac{100-(-180)}{150-(-180)} = 0.8484$$

$t_{emax} \rightarrow t_{imin}$

$t_{bottom} \rightarrow t_{right}$

leave

$$t_{(bottom)} = \frac{y_{min} - y_0}{y_1 - y_0} = \frac{-100-(-180)}{150-(-180)} = 0.2424$$

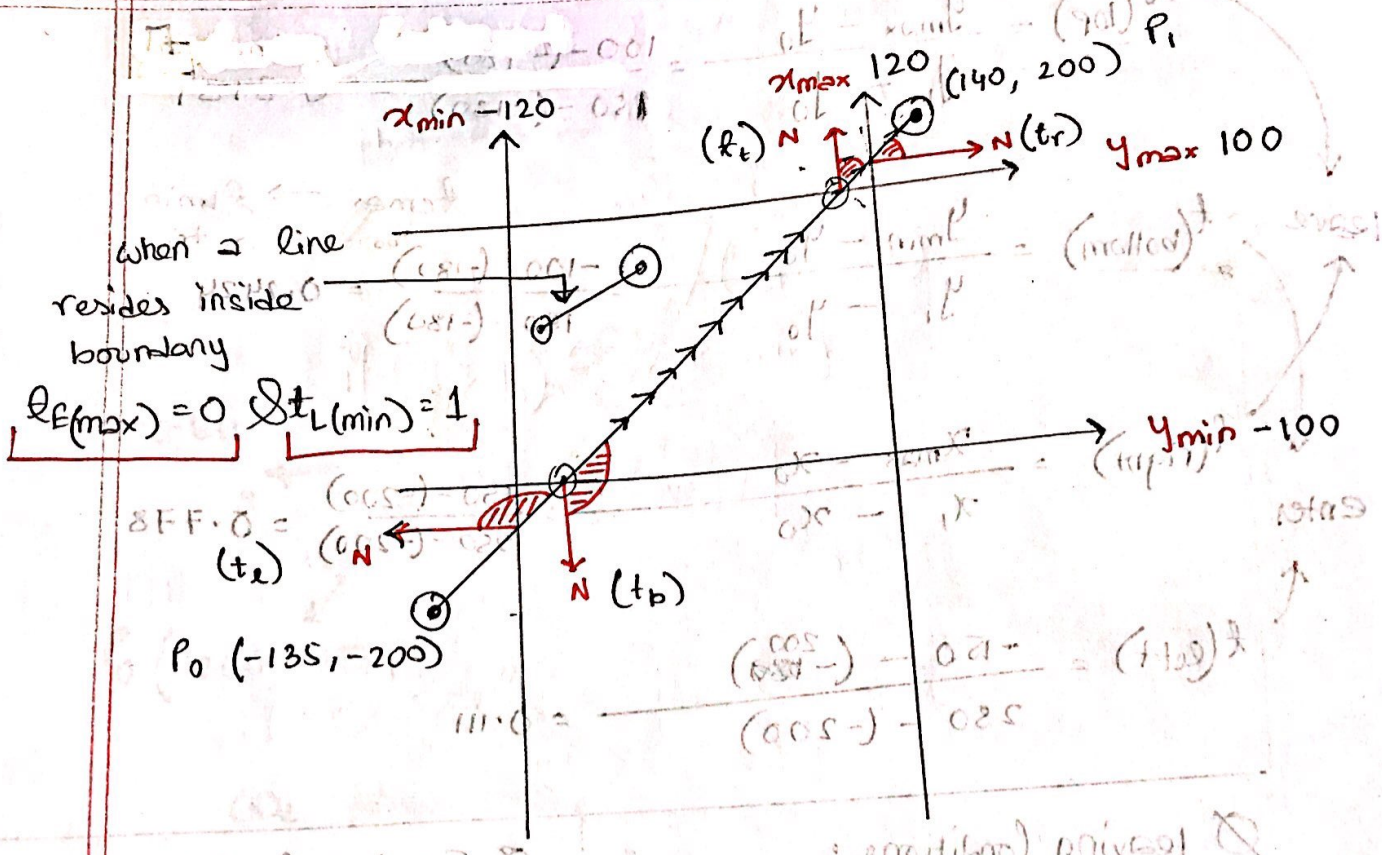
enter

$$t_{(right)} = \frac{x_{max} - x_0}{x_1 - x_0} = \frac{150-(-200)}{250-(-200)} = 0.778$$

$$t_{(left)} = \frac{-150-(-200)}{250-(-200)} = 0.111$$

∅ leaving Conditions :	∅ Entering Conditions :
$(P_1 - P_0) \cdot N > 0$ {Positive}	$(P_1 - P_0) \cdot N < 0$ {negative}
Right $x_1 > x_0$	Right $x_1 < x_0$
left $x_0 > x$	left $x_0 < x_1$
Top $y_1 > y_0$	Top $y_1 < y_0$
Bottom $y_0 > y_1$	Bottom $y_0 < y_1$

Example 2



Here,

$t_E \rightarrow \{ t_l \ \& \ t_b \}$ t_{Ent}

$t_L \rightarrow \{ t_r \ \& \ t_t \}$ $t_{leaving}$

So,

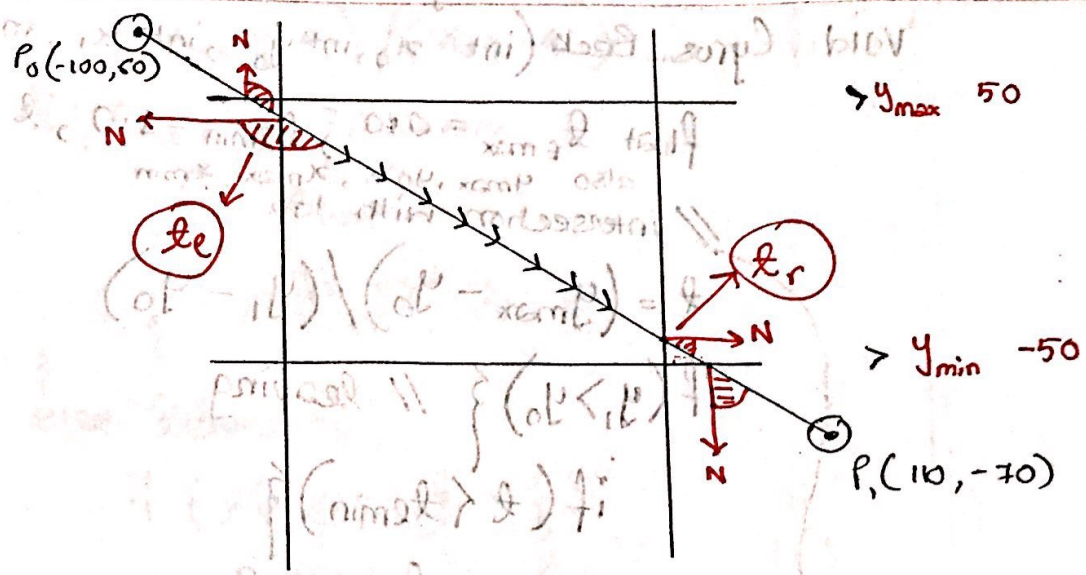
t_{enter} $\begin{cases} t_l^{(-1,0)} = \dfrac{-120-(-135)}{140-(-135)} = 0.05454 \\ t_b^{(0,-1)} = \dfrac{-100-(-200)}{200-(-200)} = 0.25 \rightarrow t_b = t_{F(max)} = 0.25 \end{cases}$

t_{leave} $\begin{cases} t_r^{(1,0)} = \dfrac{120-(-135)}{140-(-135)} = 0.9273 \\ t_t^{(0,1)} = \dfrac{100-(-200)}{200-(-200)} = 0.75 \rightarrow t_t = t_{L(min)} = 0.75 \end{cases}$

line

Example # 3

$x_{min} = -60$ $\quad$ $x_{max} = 60$



Here,

$t_{Entering} \longrightarrow \{ t_t \text{ & } t_e \}$

$t_{leaving} \longrightarrow \{ t_{bo} \text{ & } t_r \}$

So,

$$t_{Ent} \begin{cases} t_{top} = \dfrac{50 - (60)}{-70 - (60)} = 0.0769 \\ t_{left} = \dfrac{-60 - (-100)}{110 - (-100)} = 0.1905 \end{cases} \longrightarrow t_e \rightarrow t_E(max) \rightarrow (0.1905)$$

$$t_{leave} \begin{cases} t_{bottom} = \dfrac{-50 - 60}{-70 - 60} = 0.84615 \\ t_{right} = \dfrac{60 - (-100)}{110 - (-100)} = 0.7619 \end{cases}$$

$\rightarrow t_e(min) \rightarrow t_{right} \rightarrow (0.7619)$

line

CODE

Cyrus Beck Algorithm

```
Void Cyrus_Beck (int x0, int y0, int x1, int y1){
    float tEmax = 0.0, temin = 1.0, t;
    also ymax, ymin, xmax, xmin
    // intersection with top
    t = (ymax - y0)/(y1 - y0)
    if (y1 > y0){ // leaving
        if(t < temin){
            temin = t;
        }
    else { //entering
        if (t > temax){
            temax = t;
        }
    }
    t = (ymin - y0)/(y1 - y0);
    if (y1 > y0){ //entering
        if(tEmax < t){
            tEmax = t
        }
    }
    else { // leaving
        if (t1min > t)
            temin = t;
    }
}
```

TOP

BOTTOM

RIGHT

```
t = (x_max - x_0) / (x_1 - x_0)
if (x_01 > x_0) { // leaving
    if (t < t_1min)
        t_1min = t;
}
else { //entering
    if (t > t_Emax)
        t_Emax = t;
}
```

LEFT

```
t = (x_min - x_0) / (x_1 - x_0)
if (x_1 > x_0) { // entering
    if (t > t_emax) {
        t_Emax = t;
    }
}
else { // leaving
    if (t < t_1min) {
        t_1min = t;
    }
}
```

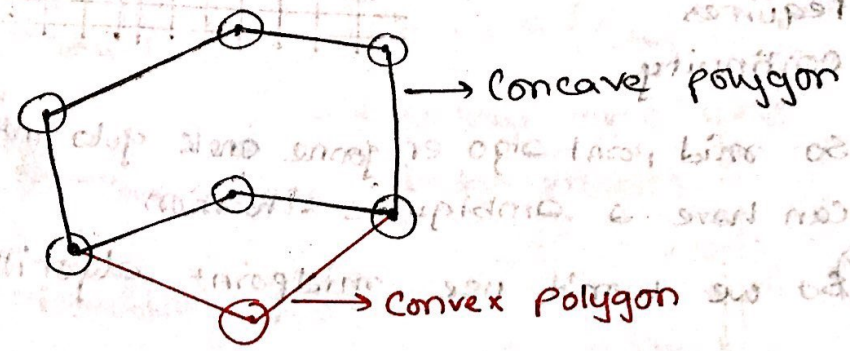
FINAL COND TO DRAW THE LINE

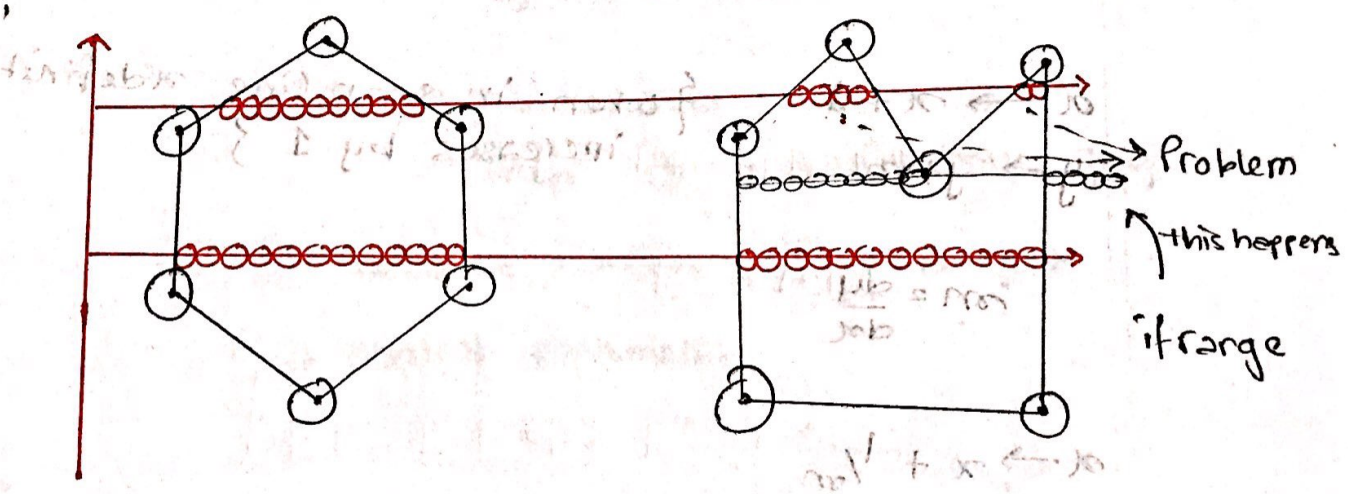
```
if (temin >= temax ) {//acceptable condition.
    new P0 = Point (temax);
    new P1 = Point (temin);
    drawline(newP0.x, newP0.y, newP1.x, newP1.y);
}
POI   newP0, newP1;
POI   Point (float t)
    POI.x = P0.x + t (P1.x - P0.x);
    POI.y = P0.y + t (P1.y - P0.y);
    return POI;
}
```

Quiz 1

- Cohen Sutherland → 6bit
- Cyrus Beck LCA → values of (t)

Polygon Filling Algorithm: (Scan line algorithm)

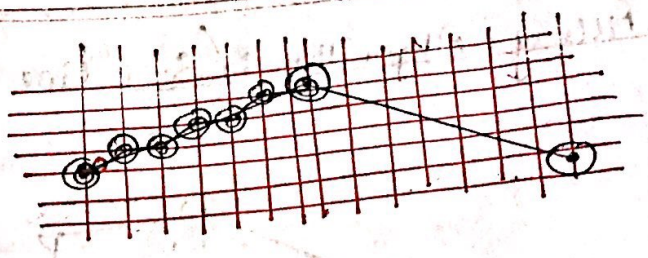




filling starts from

y_{min} to y_{max} (iteration) → and fills from (x_{min} to x_{max})

parity-based filling → we check if even no. of intersections.



requires continuity

So mid point algo er jonne onek gulo intersection. can have a ambiguous situation

(W) So we won't use midpoint algorithm.

$x \rightarrow x+1$ { when in scan line x definitely increases by 1 }

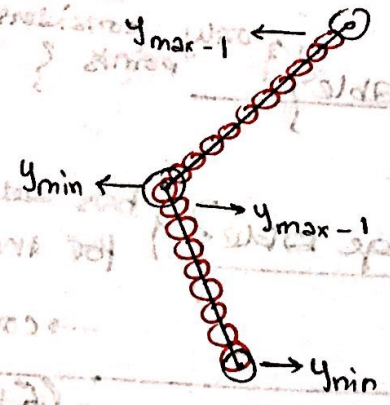
$y \rightarrow y+m$

$m = \frac{dy}{dx}$

$x \rightarrow x + 1/m$

$y \rightarrow y+1$ { when y definitely increases by 1 }

So { we won't use mid point algo since it creates multiple horizontal and vertical.



→ no pixel in y_{max}

we draw from (y_{min}) to $(y_{max}-1)$

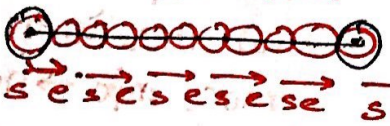
→ so that there is no double overlaps

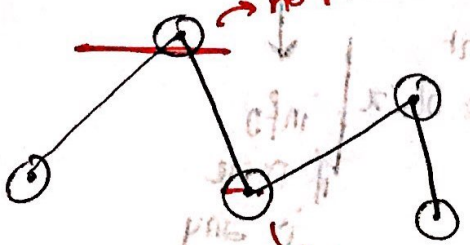
for boundary pixel horizontal lines are not filled

fill pixel er shomoi abar horizontal fill hoye jaye

this is because boundary pixel is just a control parameter.

So even pixels are required otherwise ∞ hoye jaye





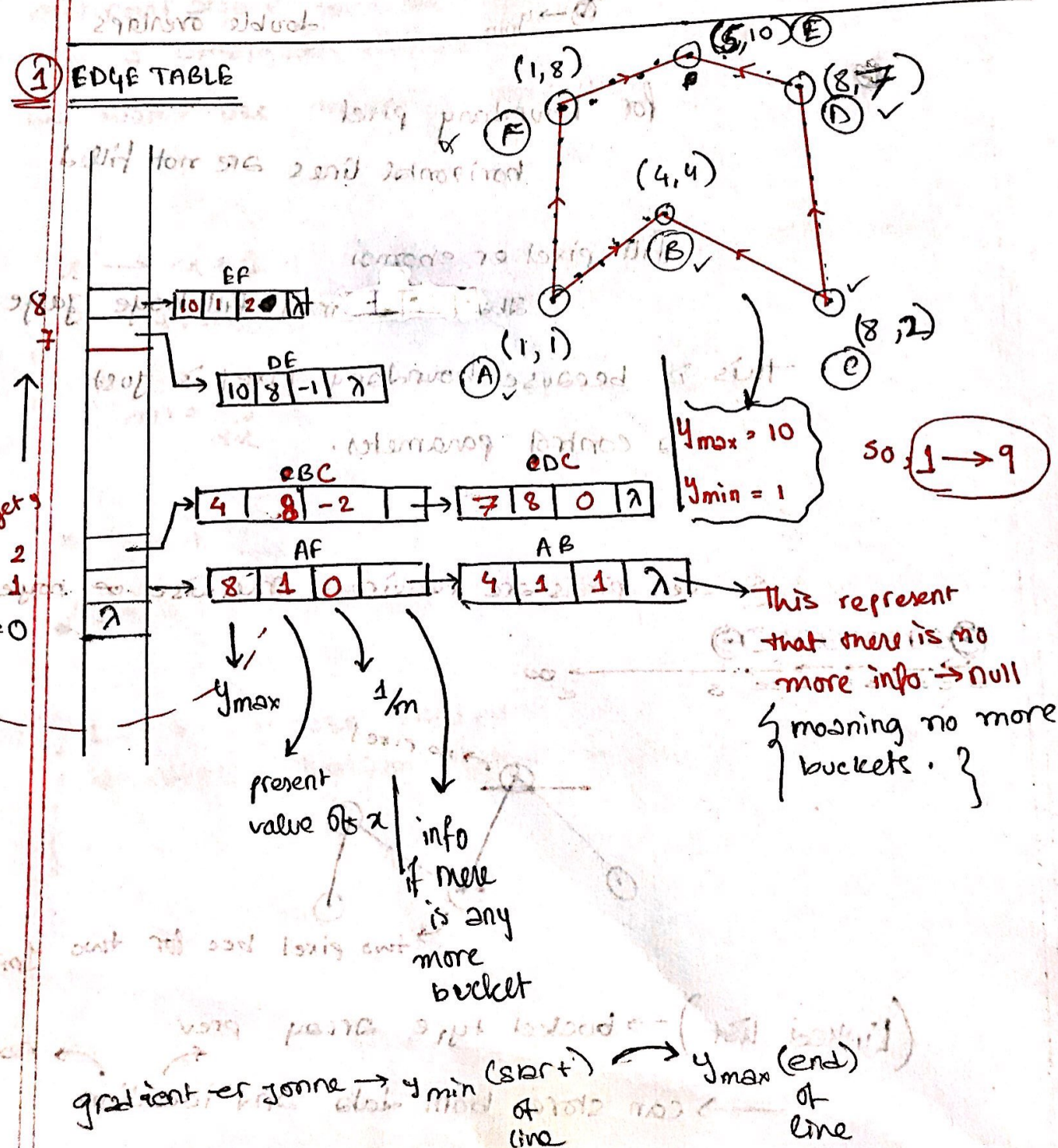
(Linked list) → bucket type array prev ← → next

→ can store both data and location

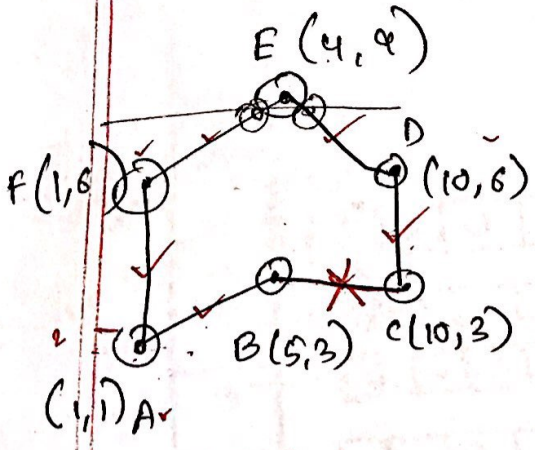
{Edge Table} } only considers the corner points }

{Active Edge table} } has set values of x for the y's }
→ considers everything

(1) EDGE TABLE



gradient ek jonne → y_{min} (start of line) → y_{max} (end of line)

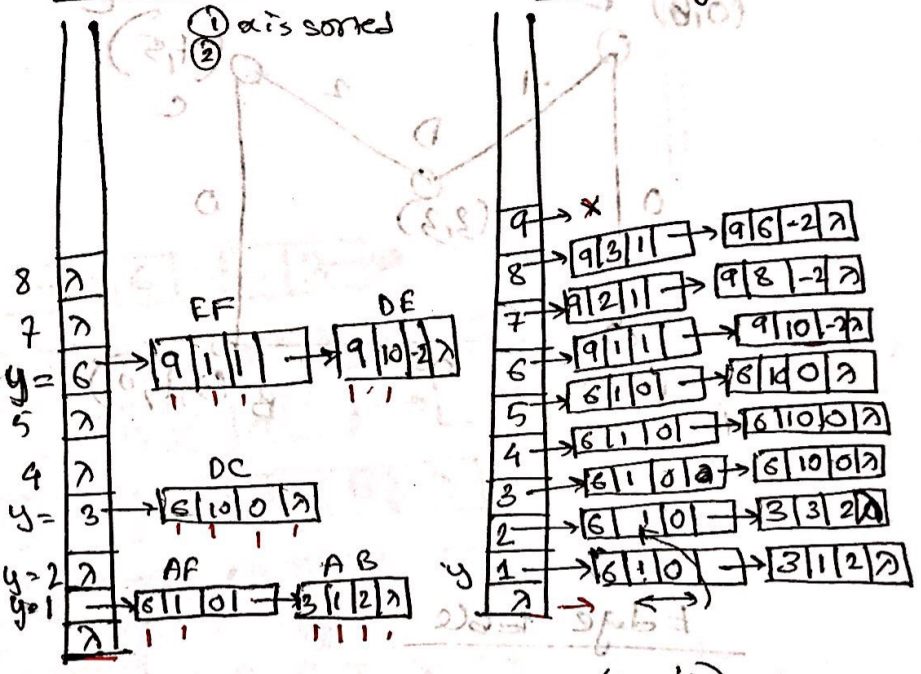


corner points

all boundaries pixel

Edge table

Active Edge table

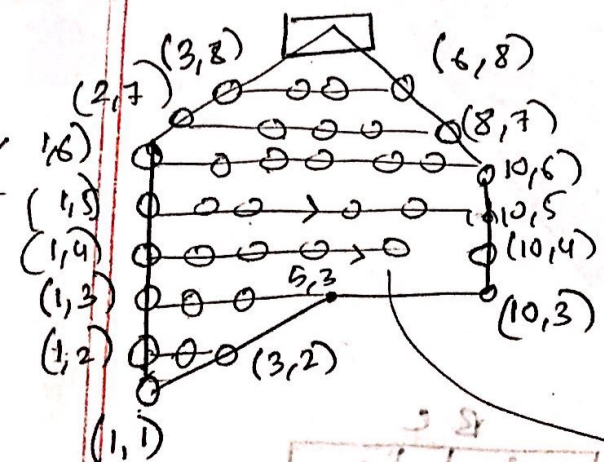


when we go up for edge table

→ we change $(x+1/m)$ for upper new x.

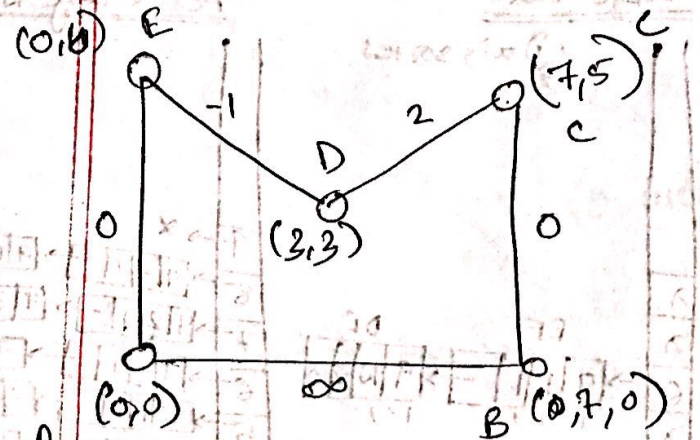
→ we check for ymax with edge value of y

x(4,9) → E

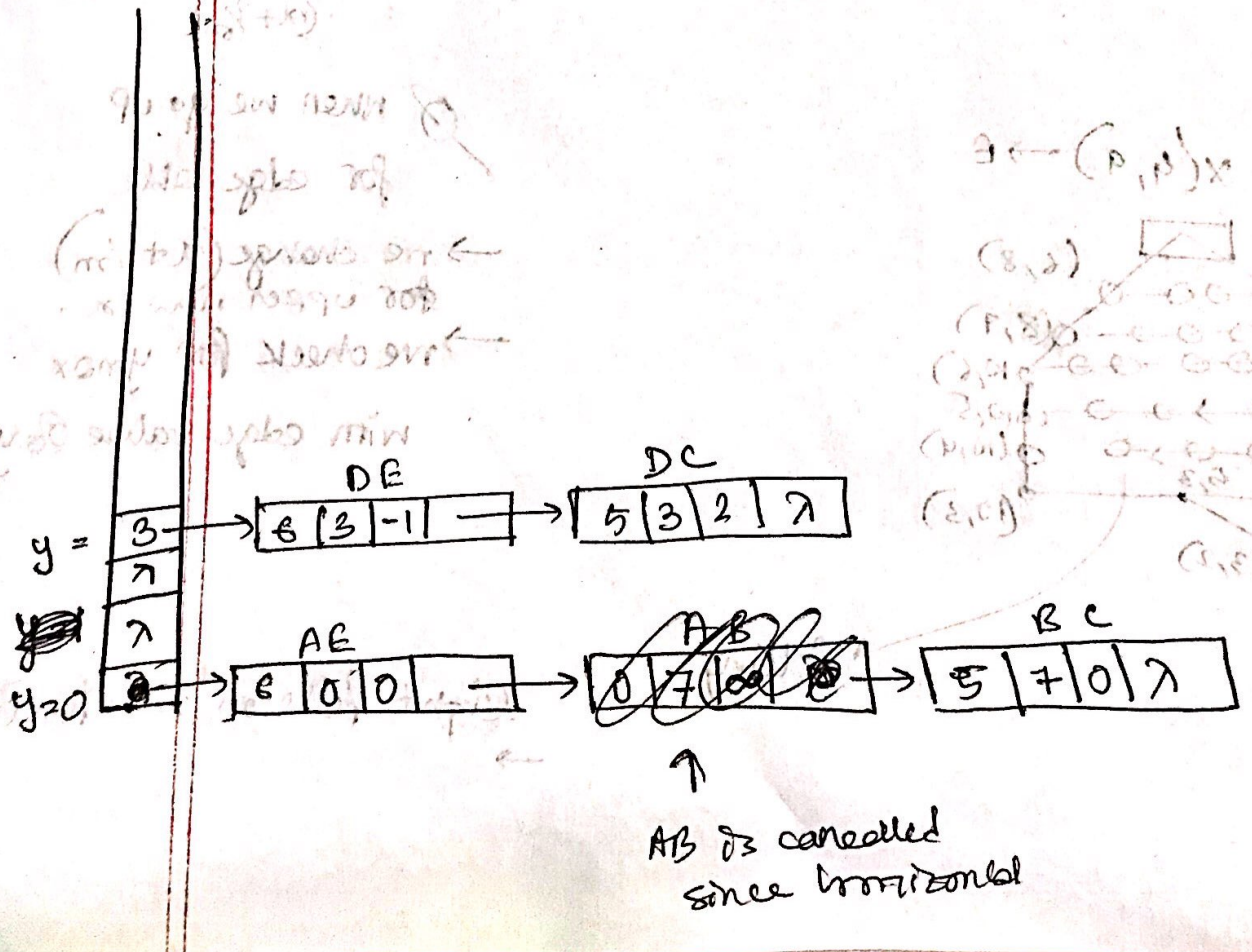


Filling

Light falls on "on" face

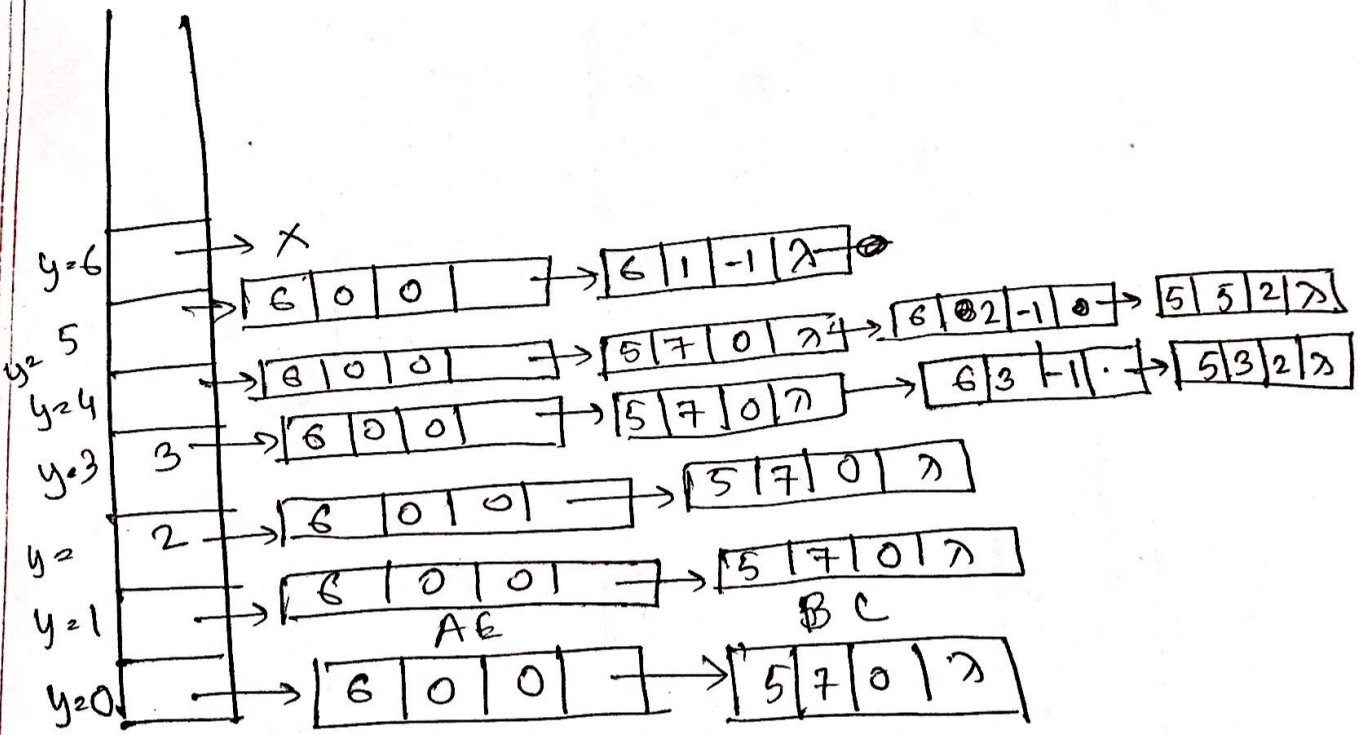


Edge table



AB is cancelled since horizontal

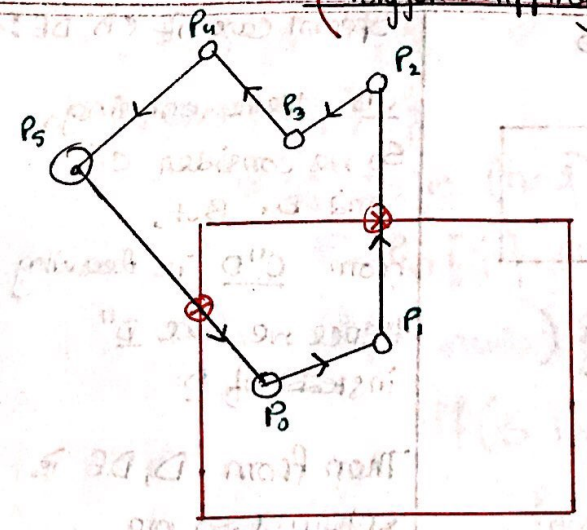
Active Edge



lecture 9

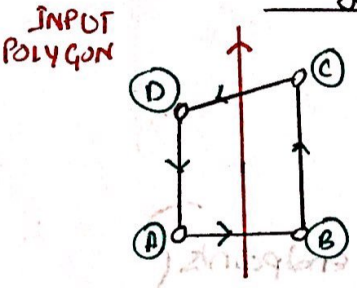
29.06.19

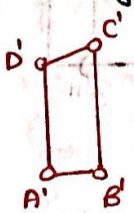
Polygon Clipping Algorithm



left inside
right outside.

Sutherland – Hodgeman PCA





OUTPUT POLYGON

4 cases

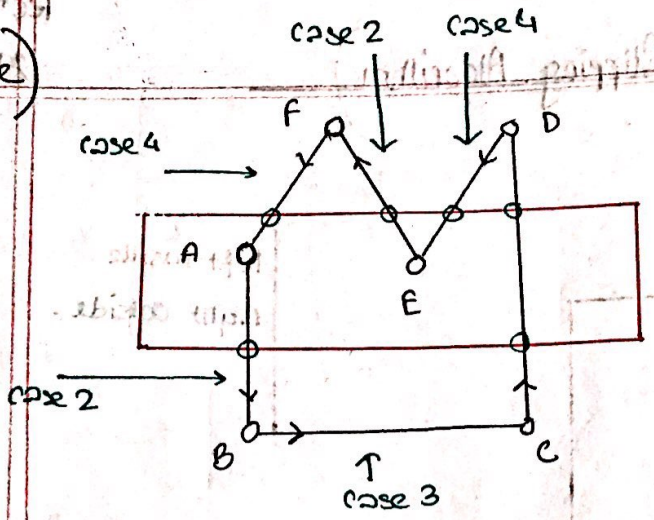
(1) If both the start and end points are inside, (in case of AD) then keep the end points only. [Both inside]*

(2) If one is inside and another outside (in case of AB) → leaving
then keep the intersection only. [P outside, S inside]*

(3) If both points are outside, then we ignore that (BC's case)
[both outside]

(4) If one is inside and another is outside, (CD's case) → entering
then keep the intersection & end points [S, outside, P inside]

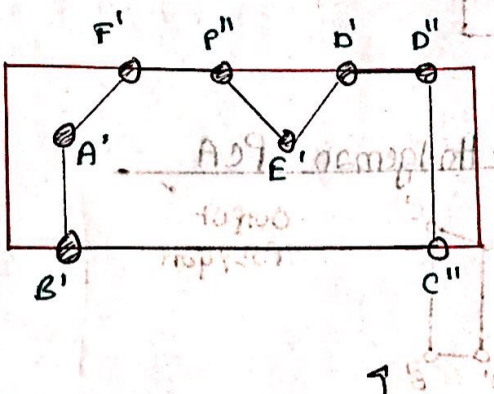
(Special Case)



Special case of CD, DE :-

CD make entering, So we consider C" and D. But, from C"D is leaving Hence we take D" instead of D

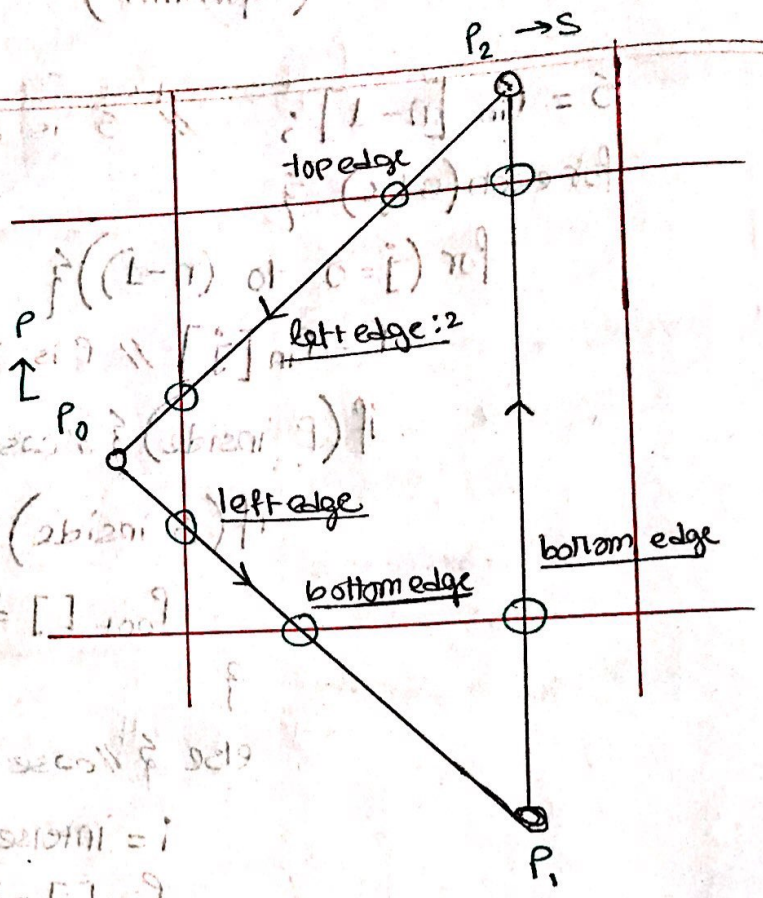
Then from D, DE is entering so we consider E" and E.



So the question remains { what are the endpoints & new intersections ? }

(Algorithm)

```
S = Pin [n-1];   // S is a point {start Point}
for each (edge) {
    for (j=0 to (n-1)){
        P = Pin [j] // P is the end point
        if(P inside) { // case 1 & case 4
            if(S inside) {//case 1
                Pout [] = P; // when we put the value it goes to next like stack.
            }
            else {//case 4
                i = intersect (S,P);
                Pout [] = i;
                Pout [] = P;
            }
        }
        else {//case 2 & 3
            if (S inside) {//case 2
                i = intersect (S,P);
                Pout [] = i;
            }
            else {//case 3 ≈ both points outside.
                //do nothing
            }
        } //else ends
        S=P; → start point will be prev end-point
    }//end of for loop
}//end of loop for each edge.
```



If we think about it a little

1st scenario

S = P2 ;
P = P0 ;

else e jabo,
Then else er else e jabo {case 3}
Do noming.