Module - II

Line Drawing Algorithm ..

1. DDA Algorithm

→ Digital Differential Analyzer is a scan conversion line algorithm,

Algorithm

```
#define ROUND(a) ((int)(a+0.5))
void lineDDA(int xa, int ya, int xb, int yb)
{
    int dx = xb-xa, dy = yb-ya, steps, k;
    float xIncrement, yIncrement, x=xa, y=ya;
    if (abs(dx) > abs(dy))
        steps = abs(dx);
    else
        steps = abs(dy);
    xIncrement = dx / (float) steps;
    yIncrement = dy / (float) steps;
    setPixel (ROUND(x), ROUND(y));
    for (k=0; k<steps; k++)
    {
        x += xIncrement;
        y += yIncrement;
        setPixel (ROUND(x), ROUND(y));
    }
}
```

Initially the algorithm accepts as input the two endpoint pixel positions.

Horizontal and vertical differences b/w the end point positions are assigned to parameters dx and dy.

→ The difference with the greater magnitude determines the value of parameter steps.

→ Starting with pixel position (x_a, y_a), we determine the offset needed at each step to generate the next pixel position along the line path.

→ We loop through this process steps times.

→ If the magnitude of dx is greater than the magnitude of dy and x_a is less than x_b, the values of the increments in the x and y directions are 1 and m.

Advantages

1. Faster method for calculating pixel positions.
2. Eliminates the multiplication.

Disadvantages

1. Round off error in successive additions

can cause the calculated pixel positions to drift away from the true line path for long line segments.

2. Rounding operations and floating point arithmetic are still time consuming.

2. Bresenham's Line Algorithm

Algorithm for $|m| < 1$.

① Input the two line endpoints and store the left endpoint in (x_0, y_0).

② Load (x_0, y_0) into the frame buffer, i.e. plot the first point.

③ Calculate constants Δx, Δy, $2\Delta y$ and $2\Delta y - 2\Delta x$ and obtain the starting value for the decision parameter as

$$P_0 = 2\Delta y - \Delta x$$

④ At each x_k along the line, starting at $k = 0$, perform the following test:
if $P_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and

$$P_{k+1} = P_k + 2\Delta y$$

Otherwise, the next point to plot is (x_k+1, y_k+1) and

$$P_{K+1} = P_K + 2\Delta y - 2\Delta x$$

(5) Repeat step 4 Δx times.

Eg:

Endpoints

(20, 10) and (30, 18)

$\Delta x = 10 \quad \Delta y = 8$

$2\Delta y = 16 \quad 2\Delta y - 2\Delta x$

$2\Delta x = 20 \quad = -4$

$P_0 = 2\Delta y - \Delta x$

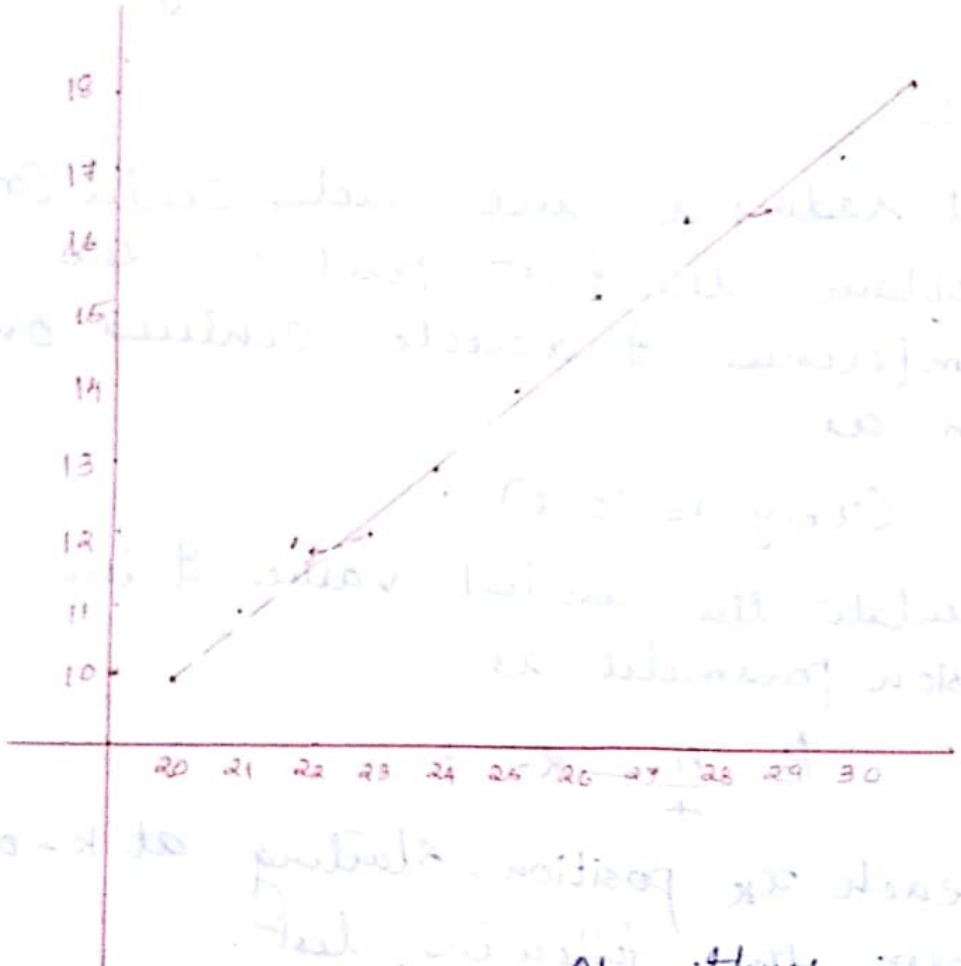
$= 16 - 10 = 6$

$x_0 = 20$

$y_0 = 10$

K	P_K	(x_{K+1}, y_{K+1})	
0	6	(21, 11)	$P_1 = P_0 + 2\Delta y - 2\Delta x$
1	2	(22, 12)	$= 6 + -4 = 2$
2	-2	(23, 12)	$P_2 = 2 + -4$
			$= -2$
3	14	(24, 13)	$P_3 = -2 + 16$
			$= 14$
4	10	(25, 14)	$P_4 = 14 + -4 = 10$
5	6	(26, 15)	$P_5 = 10 - 4 = 6$
6	2	(27, 16)	$P_6 = 6 - 4 = 2$

k	P_k	(x_{k+1}, y_{k+1})	
7	−2	(28, 16)	$P_7 = 2 - 4 = -2$
8	14	(29, 17)	$P_8 = -2 + 16 = 14$
9	10	(30, 18)	$P_9 = 14 - 4 = 10$



Circle Generation Algorithms

① Mid point circle algorithm.

To apply the midpoint method, we define a circle fn

$f_{circle}(x, y) = x^2 + y^2 - r^2$

r – radius.

In general, the relative position of any point (x,y) can be determined by checking the sign of the circle fn:

$$f_{circle}(x,y) = \begin{cases} <0, & \text{if } (x,y) \text{ is inside the circle boundary} \\ =0, & \text{if } (x,y) \text{ is on the circle boundary} \\ >0, & \text{if } (x,y) \text{ is outside the circle boundary} \end{cases}$$

Algm :

1. Input radius r and circle center (x_c, y_c) and obtain the first point on the circumference of a circle centered on the origin as

$$(x_0, y_0) = (0, r)$$

2. Calculate the initial value of the decision parameter as

$$P_0 = \frac{5}{4} - r$$

3. At each x_k position, starting at $k=0$, perform the following test:

If $P_k < 0$, the next point along the circle centered on $(0,0)$ is (x_{k+1}, y_k) and

$$P_{k+1} = P_k + 2x_{k+1} + 1$$

otherwise, the next point along the circle is (x_k+1, y_k-1) and

$$P_{k+1} = P_k + 2x_{k+1} + 1 - 2y_{k+1}$$

where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$.

4. Determine symmetry points in the other seven octants

5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c, \quad y = y + y_c$$

6. Repeat steps 3 through 5 until $x \geq y$.

Eg:

Radius $r = 10$,

$(x_c, y_c) = (0, 0)$.

$(x_0, y_0) = (0, 10)$.

$P_0 = 1 - r = 1 - 10 = \underline{-9}$.

k	P_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$	P_{k+1}
0	-9	(1, 10)	2	20	$-9+2+1 = \underline{-6}$
1	-6	(2, 10)	4	20	-1
2	-1	(3, 10)	6	20	6
3	6	(4, 9)	8	18	-3
4	-3	(5, 9)	10	18	8
5	8	(6, 8)	12	16	5
6	5	(7, 7)	14	14	

② Bresenham's incremental circle algm for the first quadrant.

All variables are assumed integer

initialize the variables

$x_i = 0$

$y_i = R$.

$\Delta i = 2(1-R)$

Limit = 0.

while $y_i \geq$ Limit

call setpixel (x_i, y_i)

determine if case 1 or 2, 4 or 5, or 3.

if $\Delta_i < 0$ then

$\delta = 2\Delta i + 2y_i - 1$

determine whether case 1 or 2.

if $\delta \leq 0$ then

call mh (x_i, y_i, Δ_i)

else

call md (x_i, y_i, Δ_i)

end if.

else if $\Delta_i > 0$ then

$\delta' = 2\Delta_i - 2x_i - 1$

determine whether case 4 or 5.

if $\delta' \leq 0$ then

call md (x_i, y_i, Δ_i)

else

call mv (x_i, y_i, Δ_i)

end if

else if $\Delta_i = 0$ then

call md(x_i, y_i, Δ_i)

end if

end while

finish.

* move horizontally.

Subroutine mh(x_i, y_i, Δ_i)

$x_i = x_i + 1$

$\Delta_i = \Delta_i + 2x_i + 1$

end sub.

* move diagonally

Sub routine md(x_i, y_i, Δ_i)

$x_i = x_i + 1$

$y_i = y_i - 1$

$\Delta_i = \Delta_i + 2x_i - 2y_i + 2$

end sub.

* move vertically.

Subroutine mv(x_i, y_i, Δ_i)

$y_i = y_i - 1$

$\Delta_i = \Delta_i - 2y_i + 1$

end sub.

Eg.

Consider the origin centered circle of radius 8.

Initial Calculations.

$x_i = 0$, $y_i = 8$

$\Delta_i = 2(1 - R) = 2 \times -7 = -14$.

Limit = 0.

Setpixel	Δi	δ ($2\Delta i + 2y_{i-1}$)	δ' ($2\Delta i - 2x_{i-1}$)	x	y
(0,8)	-14			0	8
	-11	-13		1	8
(1,8)	-6	-7		2	8
(2,8)	-12	3		3	7
(3,7)	-3	-11		4	7
(4,7)	-3	7		5	6
(5,6)	1	5		6	5
(6,5)	9		-11	7	4
(7,4)	4		3	7	3
(7,3)	18		-7	8	2
(8,2)	17		19	8	1

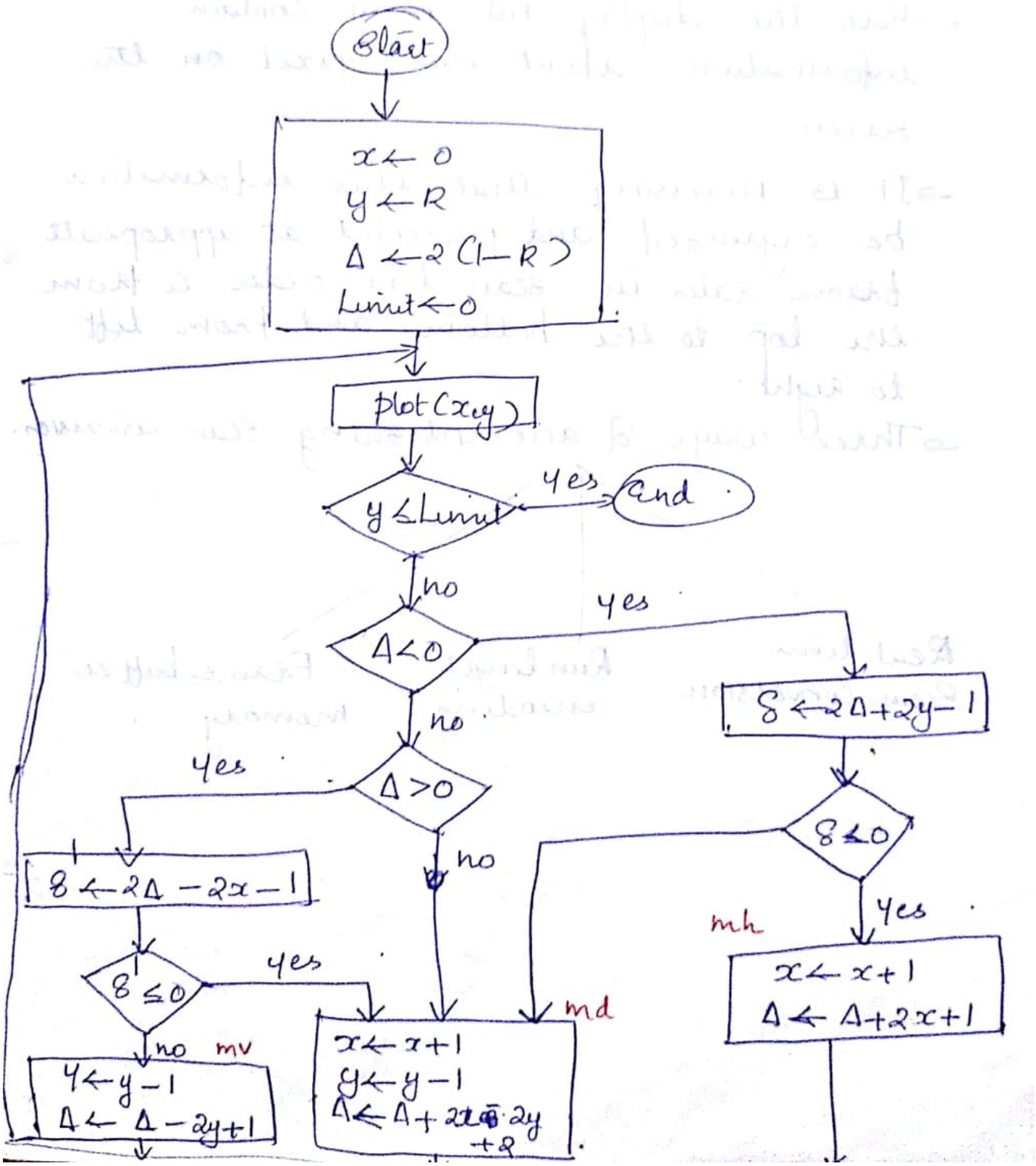
Setpixel Ai δ δ' x y

(8,1)

18 17 8 0

(8,0)

Flow chart



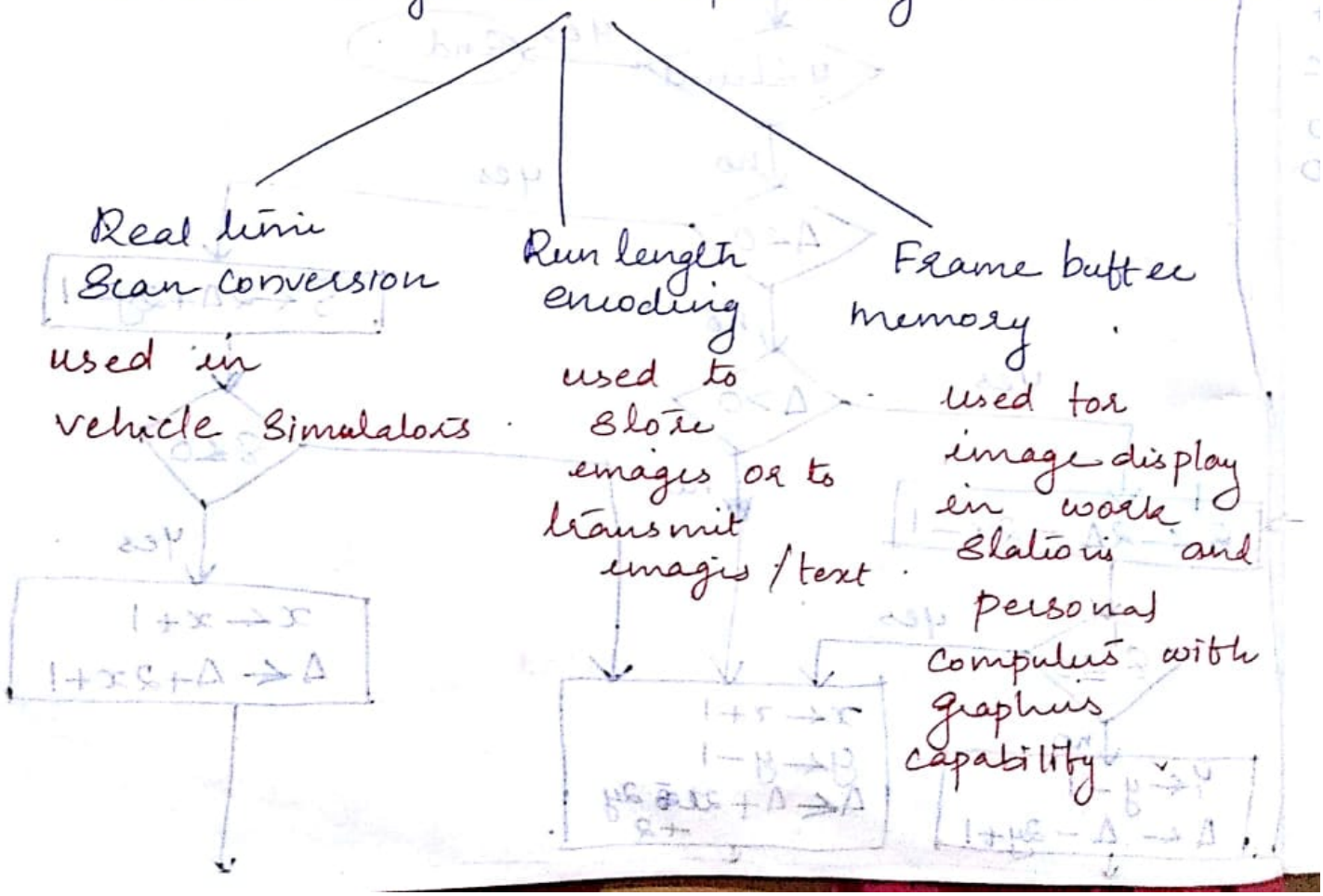
Scan Conversion - Generation of the Display.

→ Is the process of organizing the picture into precise pattern required by the graphics display.

→ Here the display list must contain information about every pixel on the screen.

→ It is necessary that this information be organized and presented at appropriate frame rates in scan line order, ie from the top to the bottom and from left to right.

→ Three ways of accomplishing scan conversion.



Real time Scan Conversion

→ Here the picture is randomly represented in terms of visual attributes and geometric properties.

→ Visual attributes are color, shade and intensity.

→ Geometric properties are xy coordinates, slopes and text.

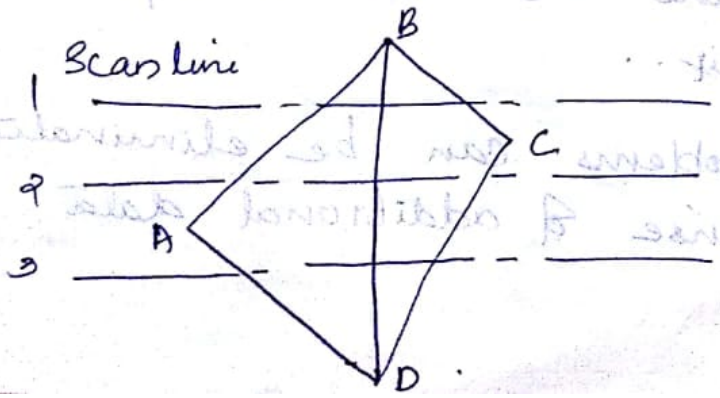
→ The simplest implementation for real-time scan conversion processes the entire display list to obtain the intersections of each line in the display list with a particular scan line each time a scan line is displayed.

1. A simple Active Edge List using pointers.

→ Maintains the active edge list using 2 floating pointers into the y sorted list.

→ **begin pointer**
Used to indicate the beginning of the active edge list.

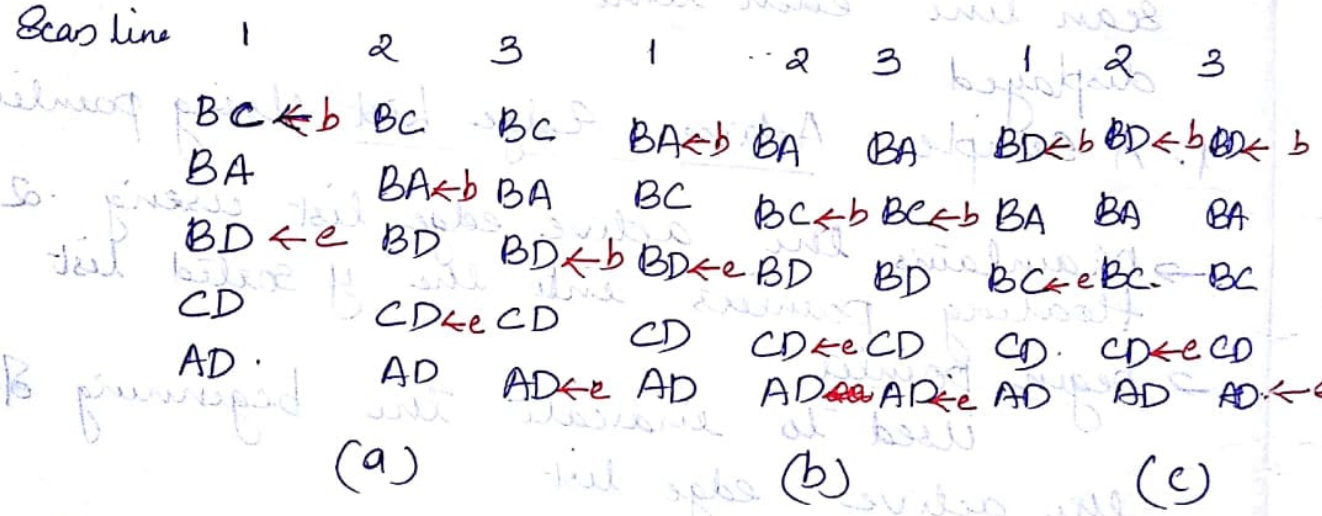
→ **end pointer**
Used to indicate the end of the active edge list.



→ The begin pointer is initially set at the beginning of the list, i.e at BC.

→ The end pointer is set at the last line in the list that begins above the scan line under consideration, i.e at BD.

→ As the scan moves down the picture, the end pointer is moved down to include those new lines which now start on or above the current scan line and the begin pointer is moved down to eliminate lines which end above the current scan line.



→ Fig (b) and (c) illustrate a problem with this algm.

→ In (c) the line BC never drops of the active edge list..

So these problems can be eliminated at the expense of additional data structures.

2. A Sorted Active Edge List.

→ Any bucket sort of all the lines in the picture is performed. For this we create a storage location or bucket for each scan line.

→ As the lines in the display list are examined, information about each line is placed in the bucket corresponding to the largest y value of the line.

→ For simple monochrome (black & white) line drawings, only the x intercept on the bucket scan line, Δx - change in x intercept and Δy (no. of scan lines crossed by the line) are recorded.

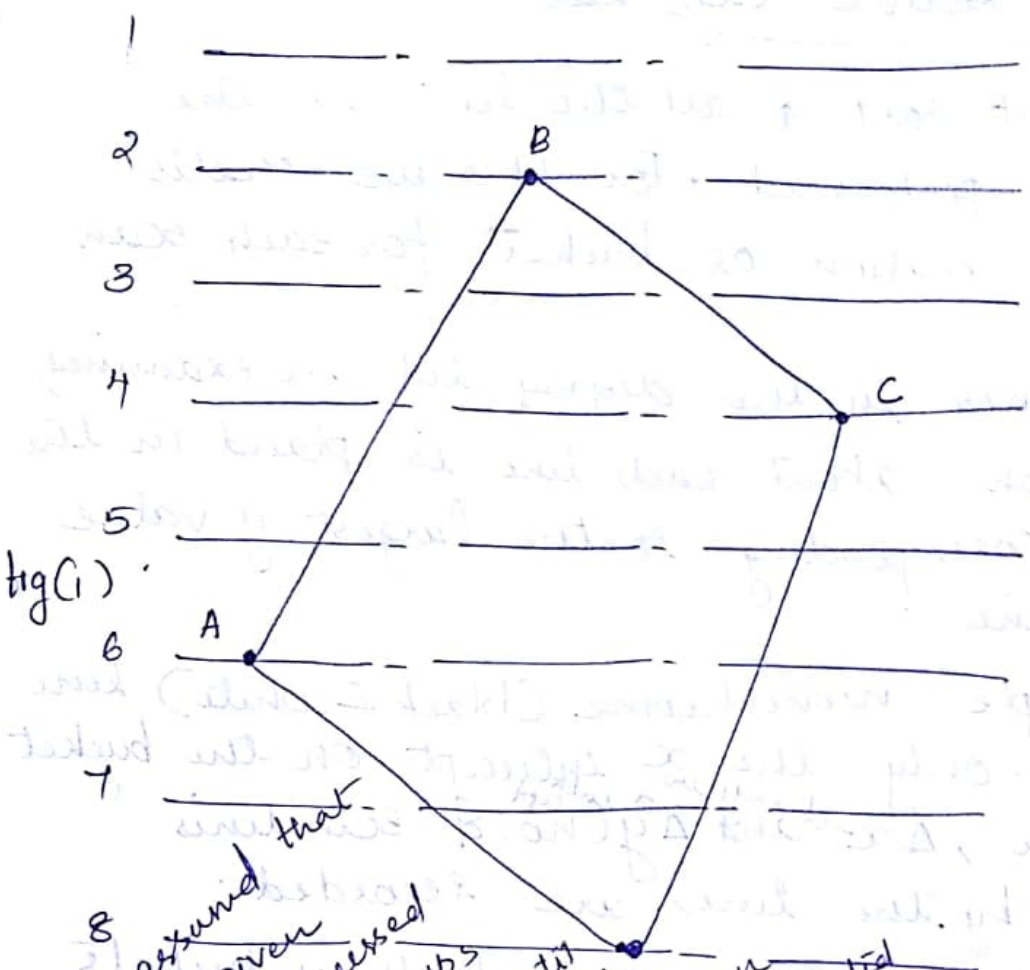
→ For simple images most of the y buckets are empty.

→ The x intercepts are sorted into scan line order and the active edge list is scan-converted.

→ After the active edge list is scan converted, Δy for each line on the active edge list is decremented by one.

→ If $\Delta y < 0$, the line is deleted from the active edge list.

→ Finally, the x intercepts for the new scan line are obtained by adding Δx to the previous values for each line on the active edge list. And the process is repeated for all scan lines.



It is assumed that edges for a given scan line are accessed in groups of 3, until a null or D completion is indicated.

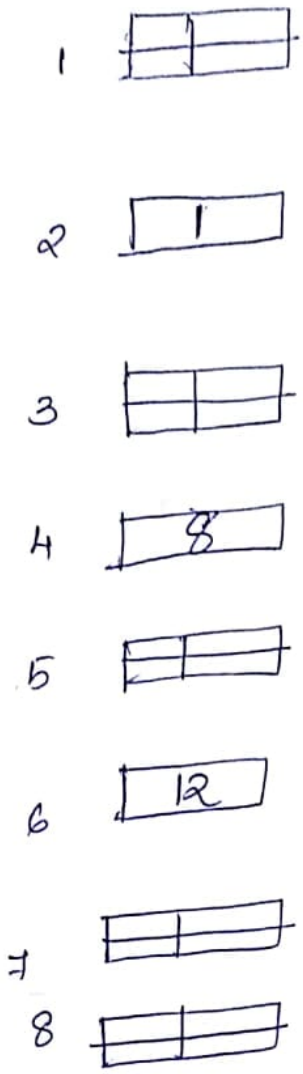
Active Edge list.

Scan line 3: $x_{BA} + \Delta x_{BA}, \Delta x_{BA}, \Delta y_{BA} - 1,$
$x_{BC} + \Delta x_{BC}, \Delta x_{BC}, \Delta y_{BC} - 1$

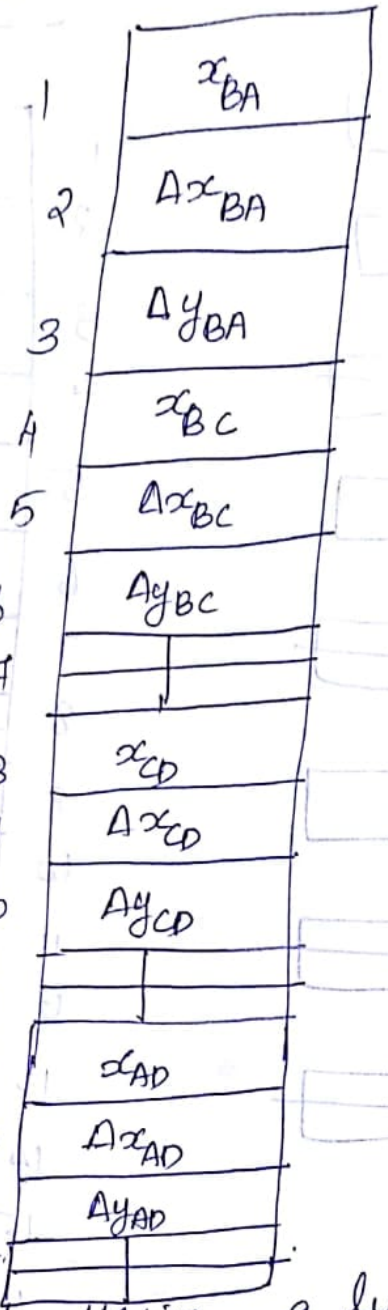
Scan line 5: $x_{BA} + 3\Delta x_{BA}, \Delta x_{BA}, \Delta y_{BA} - 3,$
$x_{CD} + \Delta x_{CD}, \Delta x_{CD}, \Delta y_{CD} - 1$

Scan line 7: $x_{CD} + 3\Delta x_{CD}, \Delta x_{CD}, \Delta y_{CD} - 3,$
$x_{AD} + \Delta x_{AD}, \Delta x_{AD}, \Delta y_{AD} - 1$

y bucket



Indexed List



An active edge list using a linked list.

→ In order to conveniently add and delete lines to the display, a linked list data structure is used.

→ The end of each data group and the location of the next data group on that scan line as well as the completion of the link are required.

→ If a new edge is added, the information about the line is added at the end of

The data list :

trig(d)	y bucket		Linked list	
1			1	x_{BA}
2	1		2	Δx_{BA}
			3	Δy_{BA}
3			4	5
4	9		5	x_{BC}
			6	Δx_{BC}
5			7	Δy_{BC}
			8	
6	13		9	x_{CD}
			10	Δx_{CD}
7			11	Δy_{CD}
			12	
8			13	x_{AD}
			14	Δx_{AD}
			15	Δy_{AD}
			16	

If edge BD is added,

8		16	
		17	x_{BD}
		18	Δx_{BD}
		19	Δy_{BD}
		20	

If edge BC is deleted,

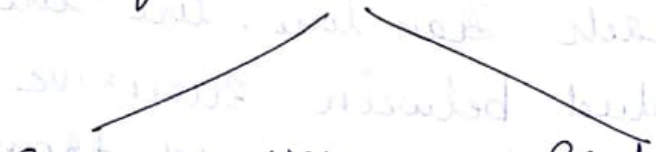
	17

Solid Area Scan conversion.

→ A unique characteristics of a raster scan device is the ability to present solid areas.

→ The generation of solid areas from simple edge or vertex descriptions is called Solid area scan conversion, polygon filling or contour filling.

→ Several techniques are used to fill a contour



Scan conversion

→ Attempt to determine in scan line order, whether or not a point is inside a polygon or contour.

→ The algm generally proceed from top of the polygon or contour to the bottom.

Seed fill.

→ Assume that some point inside the closed contour is known.

→ The algm then proceed to search for points adjacent to the seed point that are inside the contour.

→ If the adjacent point is not inside the

contour, then a boundary of the contour has been found.

→ If the adjacent point is inside that contour, then it becomes a new seed point and the search continues recursively.

Polygon Filling Algorithms

Scan conversion

I Scan line Polygon Fill Algo

→ For each scan line within the vertical extents of the polygon, an active edge list is set up and edge intersections are calculated.

→ Across each scan line, the intensity fill is then applied between successive pairs of edge intersections processed from left to right.

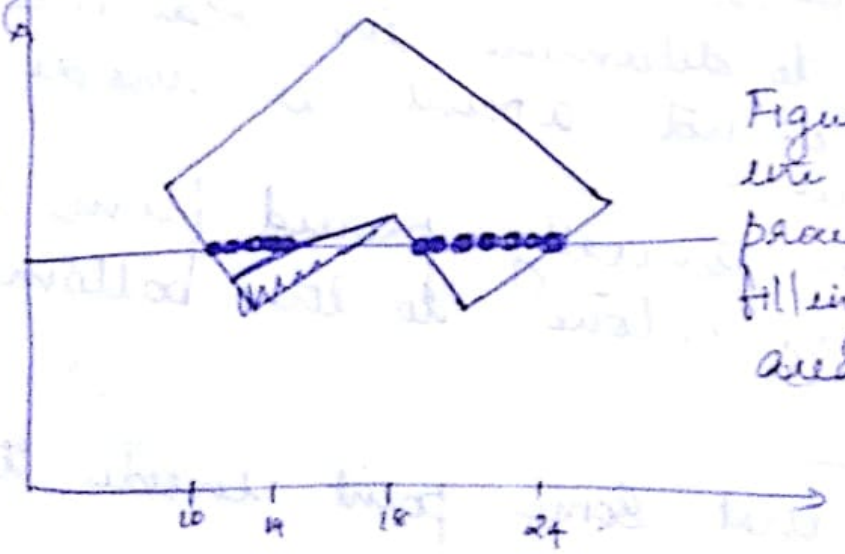


Figure illustrates the scan line procedure for solid filling of polygon areas.

→ For each scan line crossing a polygon, the area-fill algm locates the intersection points of the scan line with the polygon edges.

→ These intersection points are then sorted from left to right and the corresponding

frame-buffer positions b/w each intersection pair are set to the specified fill color.

Spatial Coherence

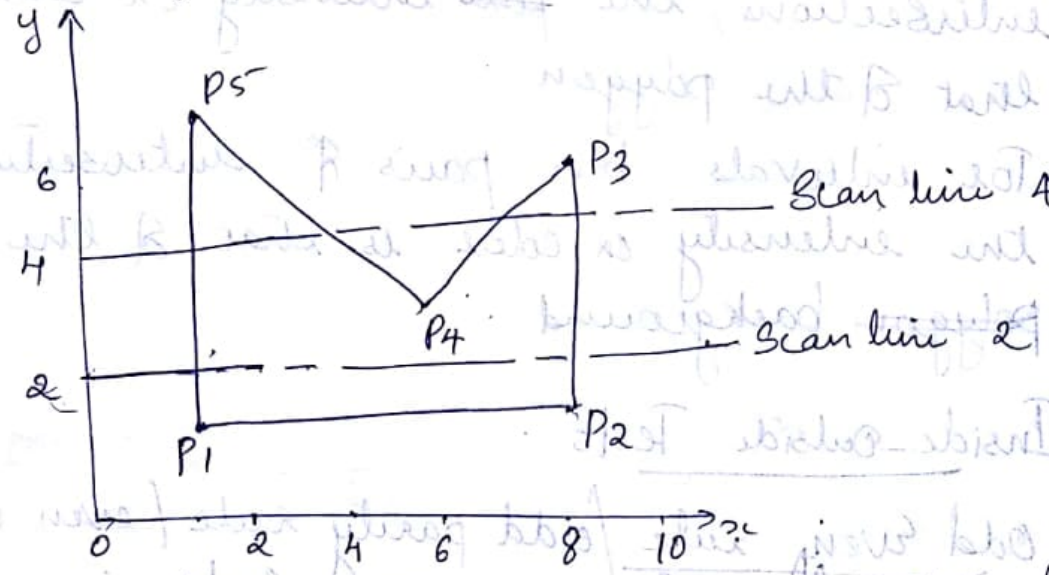
Except at boundary edges, adjacent pixels are likely to have the same characteristics.

Scan line Coherence

Adjacent pixels on a scan line are likely to have same characteristics.

→ The characteristics of pixels on a given scan line change only where a polygon edge intersects the scan line.

→ These intersections divide the scan line into regions



→ For the simple polygon, the scan line labelled 2 intersects the polygon at $x = 1$ and $x = 8$.

→ These intersections divide the scan line into 3 regions:

$x < 1$ outside the polygon

$1 \leq x \leq 8$ inside the polygon.

$x > 8$ outside the polygon.

Similarly the scanline 4 is divided into 5 regions.

$x < 1$	outside
$1 \le x \le 4$	inside
$4 \le x \le 6$	outside
$6 \le x \le 8$	inside
$x > 8$	outside

→ First the intersections are sorted.

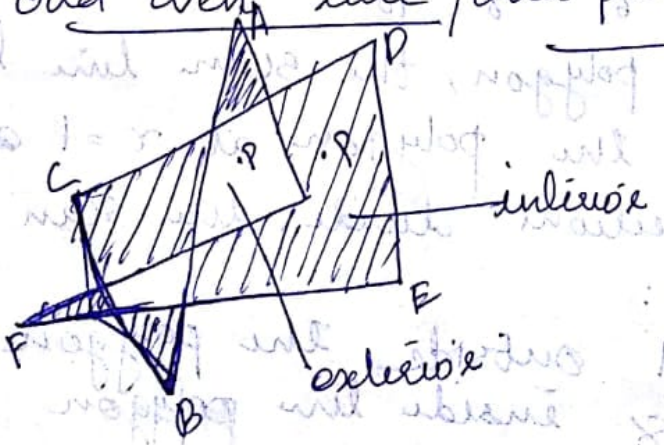
→ Then the sorted intersections are considered in pairs.

→ For each interval formed by a pair of intersections, the ~~pixel~~ intensity or color is that of the polygon.

→ For intervals b/w pairs of intersections the intensity or color is that of the ~~polygon~~ background.

* Inside-Outside Tests.

Odd Even rule / odd parity rule / even odd rule.

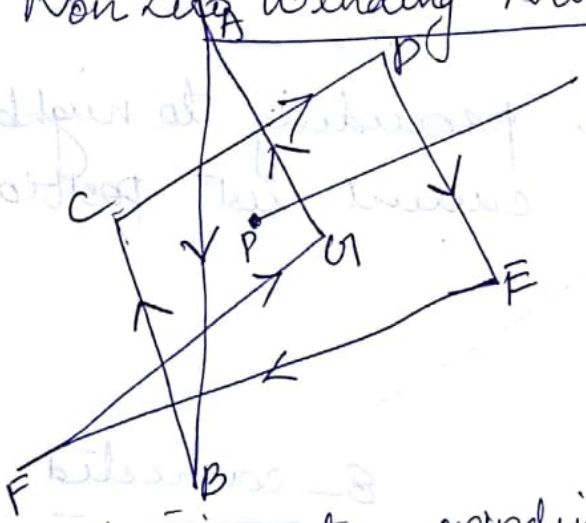


→ Draw a line from any position P to a distant point outside the coordinate extents of the object without intersecting vertices and count the no: of edge crossings along the line.

→ If the no: of polygon edges crossed by this line is odd, then P is an interior point.

→ Otherwise, P is an exterior point.

Nonzero winding Number Rule



→ Initialize the winding number to 0.

→ Draw a line from any position P to a distant point outside the coordinate extents and count the no: of edges cross the line each direction.

→ we add 1 to the winding number every time we intersect a polygon edge that crosses the line from right to left.

→ we subtract 1 when we intersect an edge that crosses from left to right.

→ If the winding number is nonzero, P is defined to be an interior point otherwise P is an exterior point.

Boundary Fill Algm : Seed Fill

→ Start at a point inside a region and paint the interior outward toward the boundary.

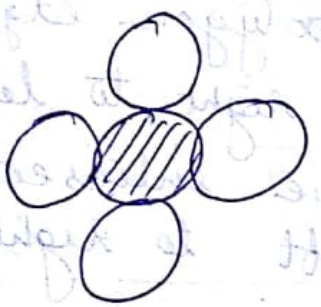
→ If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered.

→ 2 methods for proceeding to neighbouring pixels from the current list position.

4-connected

→ Four neighbouring points are listed.

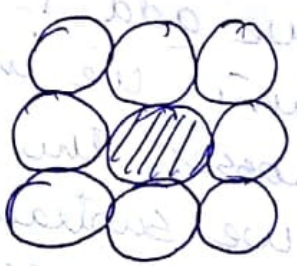
→ Right, left, above & below the current pixel.



→ Produces partial fill

8-connected

→ Neighbouring positions to be listed includes the four diagonal pixels.



→ Produces complete fill

```
void boundaryFill (int x, int y, int fill,
                   int boundary)
{
    int current;
    current = getPixel(x,y);
    if ((current != boundary) && (current != fill))
    {
        setColor(fill);
        setPixel(x,y);
        boundaryFill (x+1, y, fill, boundary);
        boundaryFill (x-1, y, fill, boundary);
        boundaryFill (x, y+1, fill, boundary);
        boundaryFill (x, y-1, fill, boundary);
    }
}
```

III Flood Fill Algm (Seed fill)

→ Used to fill or recolor an area.

```
void floodFill (int x, int y, int fillcolor,
                int oldcolor)
{
    if (getPixel(x,y) == oldcolor)
    {
        setColor(fillColor);
        setPixel(x,y);
        floodFill (x+1, y, fillcolor, oldcolor);
        floodFill (x-1, y, fillcolor, oldcolor);
        floodFill (x, y+1, fillcolor, oldcolor);
    } floodFill (x, y-1, fillcolor, oldcolor);
}
```