Title

Date

Mo Tu We Th Fr Sa Su

Notes

Output Primitives :→

Points and lines :→

→ Point is the fundamental element of picture representation.

→ It is the position in the plane defined as either the pair or triplet of numbers depending upon the dimension.

→ Two points represent line or edge & 3 or more points a polygon.

→ Curved lines are represented by short straight lines.

* Line drawing Algorithms :→

→ Slope - Intercept Equation

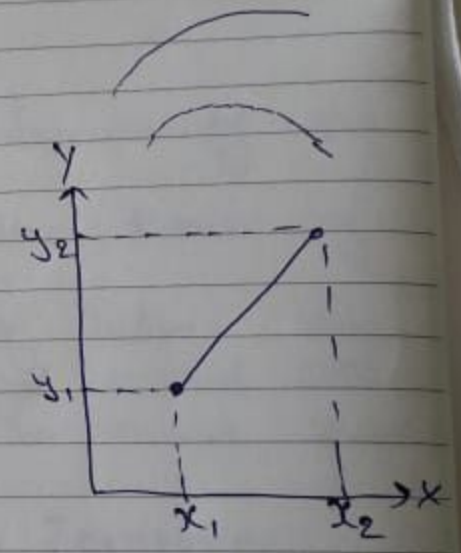
$$y = mx + b$$

→ Slope :→

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

→ Intercept :→

$$b = y_1 - mx_1$$



→ Interval Calculation :→

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

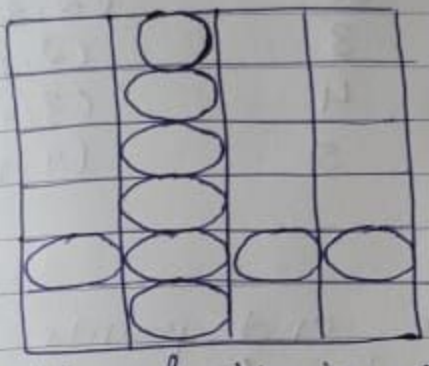
→ The process of 'turning on' the pixels of a line segment is called as vector generation or line generation & the algo for them are known as vector generation algorithm or line generation algorithm or drawing

Title 1, 2, 8, 10, 13, 14, 15, 16, 18, 24, 26, last ↑ 2, 3, 5, 7 Date

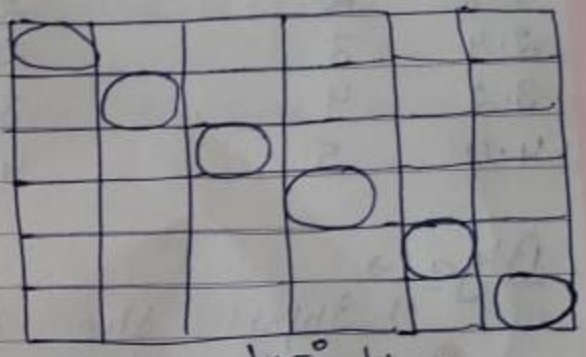
Mo Tu We Th Fr Sa Su

Notes

→ Desired characteristics of a line :→
- The line should appear as a straight line & it should start and end accurately.
- The line should be displayed with constant brightness along its length independent of its length and orientation.
- The line should be drawn rapidly.



Verticle & Horizontal lines



45° line.

→ DDA Algorithm :→ (Digital differential Analyzer)
- Sample the line at unit intervals in one coordinate.

→ Determine the corresponding integer values nearest the line path in another coordinate.

$$\text{Slope} \Rightarrow m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} = \frac{\Delta y}{\Delta x}$$

$|m| < 1$

→ $\Delta y < \Delta x$

→ $x_{k+1} = x_k + 1$

→ $y_{k+1} = y_k + m$

$|m| > 1$

→ $\Delta y > \Delta x$

→ $y_{k+1} = y_k + 1$

→ $x_{k+1} = x_k + \frac{1}{m}$

$|m| = 1$

$\Delta y = \Delta x$

→ $y_{k+1} = y_k + 1$

→ $x_{k+1} = x_k + 1$

Title

Date

Mo Tu We Th Fr Sa Su

Notes

Example → Draw a line b/w (0,0) to (4,5)

Sol :→ $m = \frac{5-0}{4-0} > 1$ $\quad \frac{1}{m} = \frac{4}{5} \Rightarrow 0.8$

X	Y	X_{Plot}	Y_{Plot}	(X,Y)
0	0	0	0	(0,0)
0.8	1	1	1	(1,1)
1.6	2	2	2	(2,2)
2.4	3	2	3	(2,3)
3.2	4	3	4	(3,4)
4.0	5	4	5	(4,5)

Algo :→

1) Input the two line end points & store the left endpoint in (x_0, y_0).
2) Plot the first point (x_0, y_0)
3) Calculate Constants Δx & Δy.
4) If $|\Delta x| > |\Delta y|$ steps $= \Delta x$ else steps $= \Delta y$
5) Calculate $x_{inc} = |\Delta x| / steps$ &

$y_{inc} = |\Delta y| / steps$.

6) At each x_k along the line, starting at $k=0$, plot the next pixel at $(x_k + x_{inc}, y_k + y_{inc})$
7) Repeat step 6 Steps ~~several~~ times.

Example → ? draw a line b/w (10, 15) & (15, 30).

Title

Date

Mo Tu We Th Fr Sa Su

Notes

Advantages of DDA Algorithm :→

1) It is the simplest algorithm & it does not require special skills for implementation.
2) It is faster method for calculating pixel positions than the direct use of equation $y=mx+b$. It eliminates the multiplication in the equation by making use of raster characteristics, so that appropriate increment are applied in the x or y directions to find the pixel position along the line path.

Disadvantages :→

1) Floating point arithmetic in DDA is still time consuming.
2) The algorithm is orientation dependent. So the end point accuracy is poor.

Code :→

```
#include <graphics.h>
#include <stdio.h>
#include <math.h>
#include <dos.h>

void main()
{ float x, y, x1, y1, x2, y2, dx, dy, step;
  int i, gd=DETECT, gm;

  initgraph(&gd, &gm, "C:\\turboc3\\bgi");

printf("Enter the value of x1 & y1:");
```

Title 1, 2, 4, 8, 10, 13, 14, 15, 16, 18, 22, 23 Date

Mo Tu We Th Fr Sa Su 24, 26, 28, later -> 1, 3, 7,

Notes

☆ Bresenham's Algorithm :→

→ Uses only incremental integer calculations.

→ For $|m| < 1$

- Start from left end point (x_0, y_0) step to each successive column (x samples) & plot the pixel whose scan line y value is closest to the line path.

→ After (x_k, y_k) the choice could be (x_{k+1}, y_k) or (x_{k+1}, y_{k+1}).

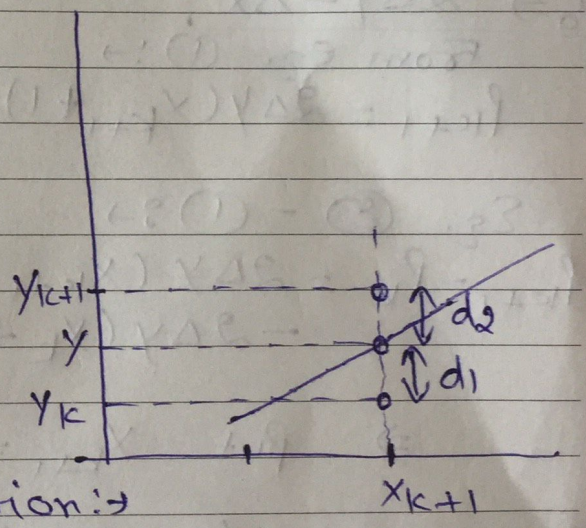
$$y = m(x_k + 1) + c$$

$$d_1 = y - y_k$$

$$d_1 = m(x_k + 1) + c - y_k$$

$$d_2 = y_k + 1 - y$$

$$d_2 = y_k + 1 - m(x_k + 1) - c$$



Difference b/w seperation :→

$$d_1 - d_2 = m(x_k + 1) + c - y_k - y_k - 1 + m(x_k + 1) + c$$

$$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2c - 1$$

Defining Decision parameter :→

$$P_k = \Delta x (d_1 - d_2)$$

Putting the value of $(d_1 - d_2)$:→

$$P_k = \Delta x [2m(x_k + 1) - 2y_k + 2c - 1]$$

$$\Rightarrow \Delta x \left[2\frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2c - 1\right]$$

$$\Rightarrow 2\Delta y(x_k + 1) - 2\Delta x y_k + 2\Delta x c - \Delta x$$

$$\Rightarrow 2\Delta y(x_k + 1) - 2\Delta x y_k + \Delta x(2c - 1) \quad -(1)$$

Title

Mo Tu We Th Fr Sa Su

Notes

As, $y = mx_k + c$

$c = y_k - mx_k$

$c = y_k - \frac{\Delta y}{\Delta x} \cdot x_k$

Put the value of c :→

$\Rightarrow 2\Delta y(x_k+1) - 2\Delta x y_k + \Delta x\left(2y - 2x\frac{\Delta y}{\Delta x} - 1\right)$

$\Rightarrow 2\Delta y(x_k+1) - 2\Delta x y_k + 2\Delta x\, y_k - 2x_k\Delta y - \Delta x$

$\Rightarrow 2\Delta y x_k + 2\Delta y - 2\Delta x y_k + 2\Delta x y_k - 2x_k\Delta y - \Delta x$

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

From eq. (1) :→

$P_{k+1} = 2\Delta y(x_{k+1}+1) - 2\Delta x y_{k+1} + \Delta x(2c-1) - (2)$

Eq. (2) − (1) :→

$P_{k+1} - P_k = 2\Delta y(x_{k+1}+1) - 2\Delta x y_{k+1} + \Delta x(2c-1)$
$- 2\Delta y(x_k+1) + 2\Delta x y_k - \Delta x(2c-1)$

$\Rightarrow$ Put $x_{k+1} = x_k + 1$

$\Rightarrow 2\Delta y(x_k+2) - 2\Delta x y_{k+1} + \Delta x(2c-1)$
$- 2\Delta y(x_k+1) + 2\Delta x y_k - \Delta x(2c-1)$

$\Rightarrow 2\Delta y x_k + 4\Delta y - 2\Delta x y_{k+1}$
$- 2\Delta y x_k - 2\Delta y + 2\Delta x y_k$

$\Rightarrow 2\Delta y - 2\Delta x(y_{k+1} - y_k)$

$P_{k+1} = P_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$

If $P_0 \geq 0$

$P_{k+1} = P_k + 2\Delta y - 2\Delta x$ $\quad \because y_{k+1} - y_k = 1$

If $P_0 < 0$, $P_{k+1} = P_k + 2\Delta y$ $\quad \because y_{k+1} - y_k = 0$

Algorithm :→

1) Input the two line end points & store the left end point in (x_0, y_0).
2) Plot first point (x_0, y_0).
3) Calculate constants Δx, Δy, $2\Delta y$, $2\Delta y - 2\Delta x$ & obtain $p_0 = 2\Delta y - \Delta x$.
4) At each x_k along the line, starting at $k=0$ perform the following test :

If $p_k < 0$, the next point plot is (x_k+1, y_k) & $p_{k+1} = p_k + 2\Delta y$

otherwise, the next point to plot is (x_k+1, y_k+1)
$p_{k+1} = p_k + 2\Delta y - 2\Delta x$

5) Repeat step 4 Δx times.

Example :→ Endpoints : (20, 10) & (30, 18)

Sol :→ Slope :→ $\frac{18-10}{30-20} = \frac{8}{10} = 0.8 < 1$

$\Delta x = 30 - 20 = 10$ $\quad p_0 = 2 \times 8 - 10 = 6$

$\Delta y = 18 - 10 = 8$ $\quad 2\Delta y - 2\Delta x = -4$

Plot $(x_0, y_0) = (20, 10)$

k	p_k	(x_{k+1}, y_{k+1})
0	6	(21, 11)
1	2	(22, 12)
2	-2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)
5	6	(26, 15)
6	2	(27, 16)
7	-2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)

Example → 2

(15, 18) to (10, 15)

Notes

Advantage :→ Involves only the Calculation of constants Δx, Δy, $2\Delta y$, $2\Delta y - 2\Delta x$ once & integer addition & subtraction in each step.

✡ Circle Generating Algorithm :→

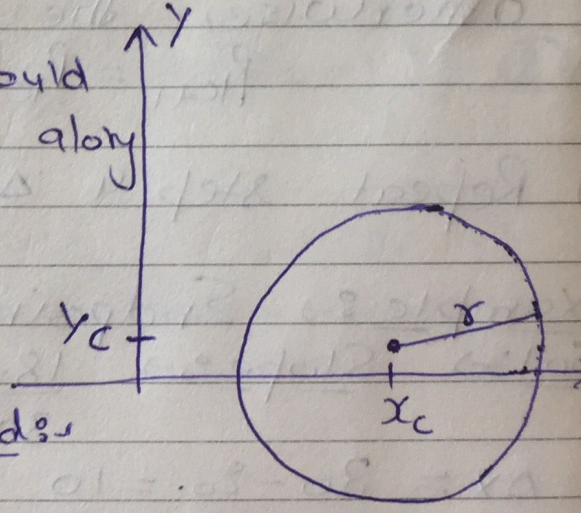
Basic foundation :→

Circle Equation →

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

Point along circumference could be calculated by stepping along x-axis :→

$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$$



Problem in above method :→

→ Computational Complexity.
→ Spacing : Non uniform spacing of plotted pixe

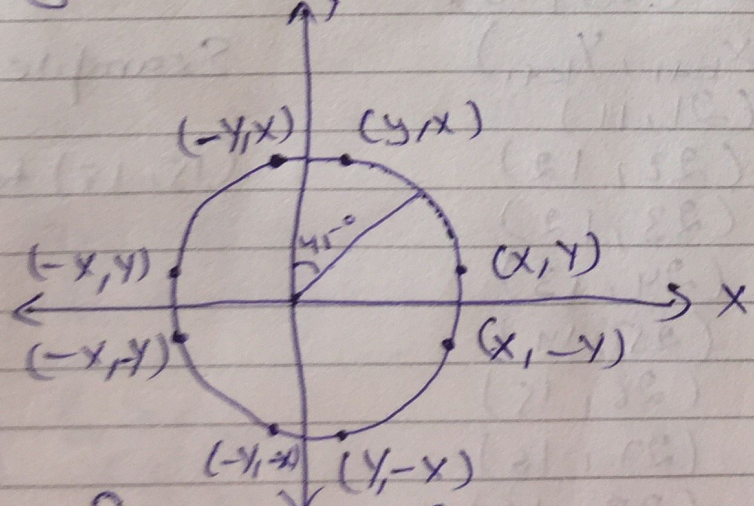


Fig: Circle Symmetry

Title 2, 4, 8, 10, 13, 14, 15, 16, 18, 23, 26, 28 Date Lab 1, 3, 7
24.

Notes

Mo Tu We Th Fr Sa Su

Mid-Point circle Algorithm :→

→ Circle function defined as :→

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

→ Any point (x, y) satisfy the following conditions :→

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

→ Decision parameter in the circle function :→

x - Unit intervals.

y_k or y_{k-1}

So the plotting points will be either (x_{k+1}, y_k) or (x_{k+1}, y_{k-1}).

Mid point $\Rightarrow \left(\frac{x_k + 1 + x_k + 1}{2}, \frac{y_k + y_k - 1}{2}\right)$

$\Rightarrow (x_k + 1, y_k - \frac{1}{2})$

$\Rightarrow x^2 + y^2 = r^2$

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

Put the values of x & y :→

Title

Notes

Date

Mo Tu We Th Fr Sa Su

$$P_{1c} = (x_k+1)^2 + (y_k - \tfrac{1}{2})^2 - r^2$$

$$P_{k+1} = (x_{k+1}+1)^2 + (y_{k+1} - \tfrac{1}{2})^2 - r^2$$

$$\Rightarrow ((x_k+1)+1)^2 + (y_{k+1} - \tfrac{1}{2})^2 - r^2$$

$$P_{k+1} - P_k \Rightarrow ((x_{1c}+1)+1)^2 + (y_{k+1} - \tfrac{1}{2})^2 - r^2 - (x_{1c}+1)^2 - (y_{1c} - \tfrac{1}{2})^2 + r^2$$

$$\Rightarrow (x_k+1)^2 + 1 + 2(x_k+1) - (x_k+1)^2 - (y_{1c} - \tfrac{1}{2})^2 + (y_{k+1} - \tfrac{1}{2})^2$$

$$\Rightarrow 2(x_k+1) + y_{k+1}^2 + \tfrac{1}{4} - y_{k+1} - y_k^2 - \tfrac{1}{4} + y_k + 1$$

$$\Rightarrow 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

$$P_{1c+1} = P_k + 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

Initial decision parameter :-

$x_{1c} = 0, \; y_k = r$

$$P_0 = (0+1)^2 + (r - \tfrac{1}{2})^2 - r^2$$

$$= 1 + r^2 + \tfrac{1}{4} - 2 \cdot \tfrac{r}{2} - r^2$$

$$\Rightarrow \tfrac{5}{4} - r$$

$P_0 \Rightarrow 1 - r$

$P_k \geq 0$

$y_{k+1} = y_k - 1$

$(x_{1c+1}, y_{1c} - 1)$

$P_k < 0$

$y_{k+1} = y_k$

(x_{k+1}, y_k)

Title

Date

Notes

Mo Tu We Th Fr Sa Su

Example: Plot a circle of radius 8 cm using midpoint algorithm.

Sol:→ $r = 8$

$P_0 = 1 - r$

$P_{k+1} = P_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$

k	(x_k, y_k)	P_k	(x_{k+1}, y_{k+1})
0	(0, 8)	−7	(1, 8)
1	(1, 8)	−4	(2, 8)
2	(2, 8)	+1	(3, 7)
3	(3, 7)	−6	(4, 7)
4	(4, 7)	+2	(5, 6)
5	(5, 6)	+1	(6, 5) → (x, y) stop

(x, y)	(−x, y)	(−x, −y)	(x, −y)
Q1	Q2	Q3	Q4
(0, 8)	(0, 8)	(0, −8)	(0, −8)
(1, 8)	(−1, 8)	(−1, −8)	(1, −8)
(2, 8)	(−2, 8)	(−2, −8)	(2, −8)
(3, 7)	(−3, 7)	(−3, −7)	(3, −7)
(4, 7)	(−4, 7)	(−4, −7)	(4, −7)
(5, 6)	(−5, 6)	(−5, −6)	(5, −6)
(6, 5)	(−6, 5)	(−6, −5)	(6, −5)
(7, 4)	(−7, 4)	(−7, −4)	(7, −4)
(7, 3)	(−7, 3)	(−7, −3)	(7, −3)
(8, 2)	(−8, 2)	(−8, −2)	(8, −2)
(8, 8)	(−8, 1)	(−8, −1)	(8, −1)
(8, 0)	(−8, 0)	(−8, 0)	(8, 0)

Example 2
$r = 10$

Note

If $P_k < 0$, $y_{k+1} = y_k$

If $P_k \geq 0$, $y_{k+1} = y_k - 1$

$P_{k+1} = P_k + 2(x_k + 1)$

$P_{k+1} = P_k + 2(x_k + 1) + 1 - 2y_{k+1}$

Notes

Algorithm :→

1) Input radius r and circle center (x_0, y_0) and obtain the first point of the circumference of 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 at $(0,0)$ is (x_{k+1}, y_k) &

$$P_{k+1} = P_k + 2(x_k + 1) + 1$$

Otherwise, the next point along the circle centered at $(0,0)$ is (x_{k+1}, y_{k-1}). &

$$P_{k+1} = P_k + 2(x_k + 1) - 2y_{k+1} + 1$$

4) Determine the symmetry points in the other seven octants.

5) Move each calculated pixel position (x, y) on to the circular path

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

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

Notes

Inside Outside Test :-

⇒ Odd Even Rule
→ Odd Edge means interior.

⇒ Non zero winding Number rule
→ Non zero winding number means interior.

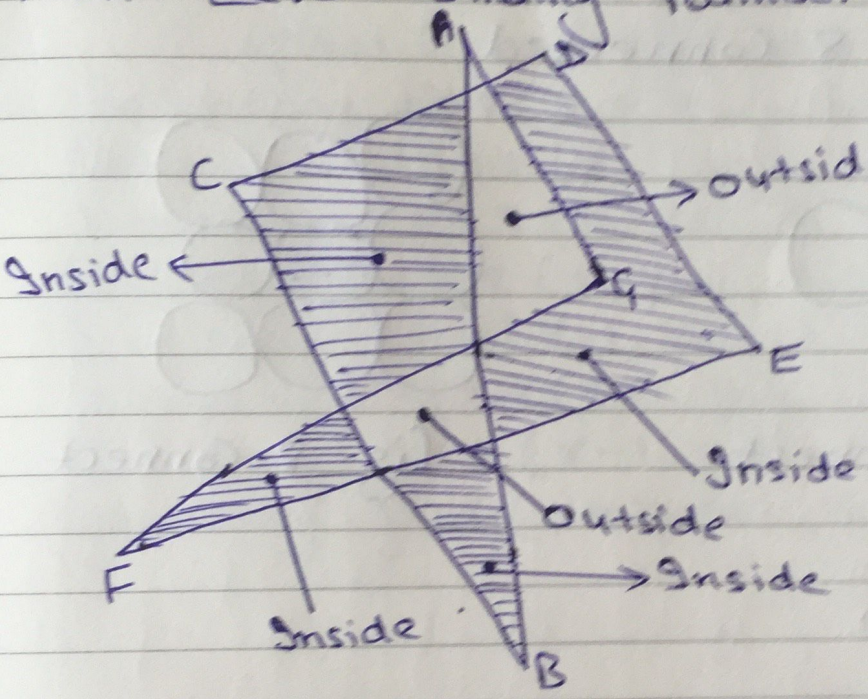
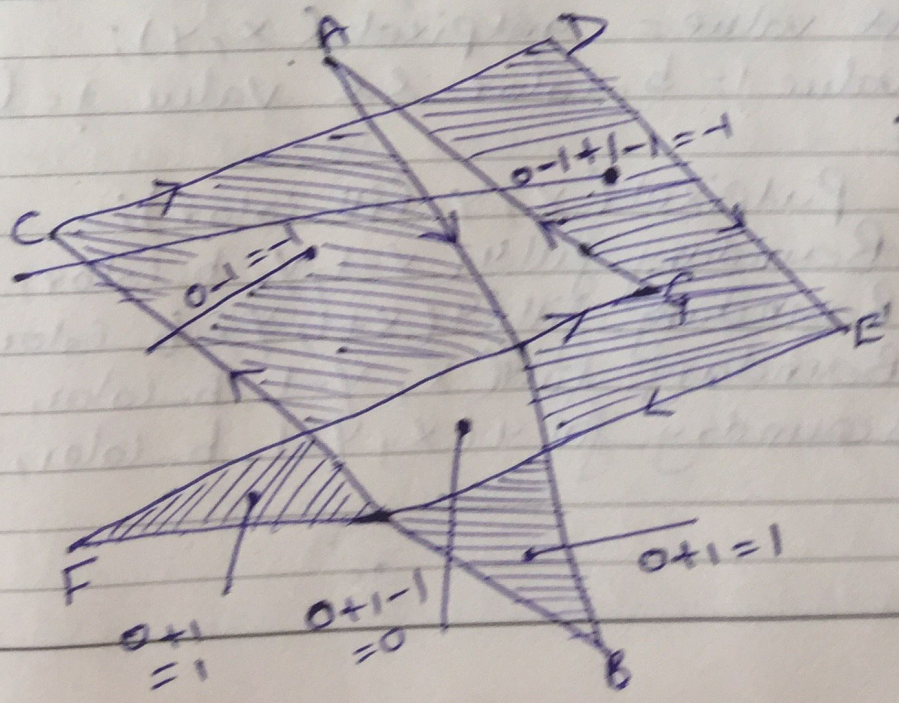


Fig. filling using odd Even Rule



LTR ⇒ -1
RTL ⇒ +1
Initial ⇒ 0

✱ Boundary Fill Algorithm :→

→ Fill the region starting from the interior point until the appropriate color boundary is reached.

→ Two ways :
- → 4 Connected
- → 8 Connected

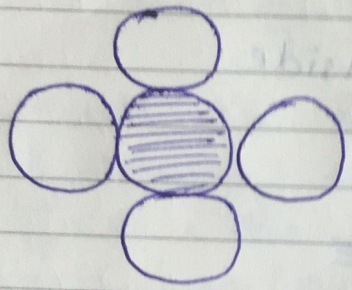


Fig. 4 Connect

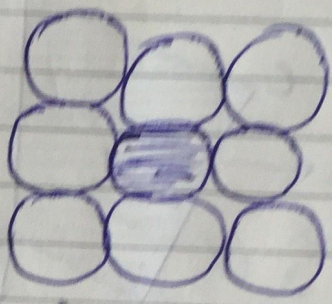


Fig 8 Connect

4 Connected :→

```
void Boundary_fill4 (int x, int y, int
                     b_color, int fill_color)
{   int value = getpixel (x, y);
    if (value != b_color && value != fill_color)
    {
        Putpixel (x, y, fill_color);
        Boundary_fill4 (x-1, y, b_color, fill_color);
        Boundary_fill4 (x+1, y, b_color, fill_color);
        Boundary_fill4 (x, y-1, b_color, fill_color);
        Boundry_fill4 (x, y+1, b_color, fill_color);
    }
}
```

6. Read :→ 11, 18, 44, 62, 33, 61, 31, 46, 34, 40, 10, 52,

Title Date

Notes Mo Tu We Th Fr Sa Su

8 Connected :→

```
void Boundary_fill8(int x, int y, int b_color, int
                                    fill_color)
{ int current = getpixel(x, y);
  if(current != b_color && current != fill_color)
    {
      Putpixel(x, y, fill_color);
    Boundary_fill8(x-1, y, b_color, fill_color);
    Boundary_fill8(x+1, y, b_color, fill_color);
    Boundary_fill8(x, y-1, b_color, fill_color);
    Boundary_fill8(x, y+1, b_color, fill_color);
    Boundary_fill8(x-1, y-1, b_color, fill_color);
    Boundary_fill8(x+1, y-1, b_color, fill_color);
    Boundary_fill8(x-1, y+1, b_color, fill_color);
    Boundary_fill8(x+1, y+1, b_color, fill_color);
    }
}
```

Drawbacks :→

1) If any interior pixel is already colored with the fill_color then the region may not be colored properly.
2) It makes lot of use of stack so memory inefficient.
3) For some figures 4-Connected method may not work fine.

Notes

Mo Tu We Th Fr Sa Su

Flood fill Algo :→

Drawback of Boundary fill Algo :→

→ If the boundary is of more than one color then region may not be colored properly.

→ Flood fill Algo works based on the previous color of the pixels in the region we want to fill color.

Algo :→

1) Get a pixel inside the region we want to color.

Current = getPixel(x, y);

2) Check if the color of the pixel (Current) is old color. Then if fill the pixel with fill_color.

3) Using 4 Connected / 8 Connected find the neighbours of the current pixel & repeat ~~step~~ 2 for every pixel.

```
void flood_fill 4 ( int x, int y, int old_color,
                    int fill_color )
{
    int current = get_pixel(x, y);
  if (current == old_color)
  {
      Putpixel(x, y, fill_color);
    Flood_fill 4 (x-1, y, fill_color, old_color);
    Flood-fill 4(x+1, y, fill_color, Old_color);
```

```
    Flood_fill4(x, y-1, fill_color, old_color);
    Flood_fill4(x, y+1, fill_color, old_color);
  }
}
```

Drawbacks :→

1) If an inside pixel is in some other color then the fill terminates & the polygon remains unfilled.

✱ Cell Array :→

→ A computer graphics output primitive defined by a rectangular grid of equal size rectangular cells each having a single color.

✱ Character Generation :→

→ In Computer graphics character can be generated using software.

→ In Hardware implementation of character generation limited faces of character can be generated.

→ A wide variety of faces of character can be generated with software implementation.

→ Three methods :→ 1) Stroke method
2) Vector method / Bitmap method
3) Star burst method.

⇒ Stroke method :→

→ In this method we use a sequence of line drawing function & arc function to generate characters.

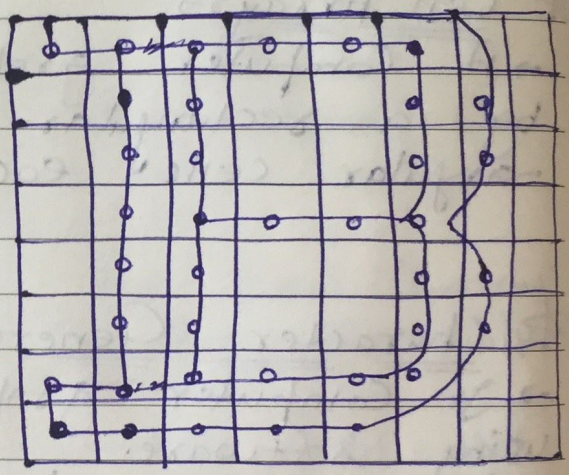
Notes

Mo Tu We Th Fr Sa Su

→ we can generate a sequence of characters by assigning starting and end point of line or arc.
→ By using this method various faces of character can be generated by changing the values in line & Arc function.
→ The main disadvantage of this method is when we draw a diagonal line it produce aliased character.

⇒ Bitmap Method :→

1	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0



→ This method is suitable for producing various character.
→ Font size of character can be increased by increasing the size of array.
→ The main drawback of this method is that this method produce aliased character.

⇒ Starbust Method :→
→ In this method a fixed pattern of line is used to generate the character.
→ In this method we use a segment

Combination of 24 bit line segments:
→ In 24 bit line segment code each bit represent a single line.
→ To highlight a line we put corresponding bit 1 in 24 bit line segment code & 0 otherwise.

Drawbacks:→
1) 24 bit segment code is required to put in memory for generating character. Hence extra memory is required in the method.
2) Character quality is poor due to limited face.