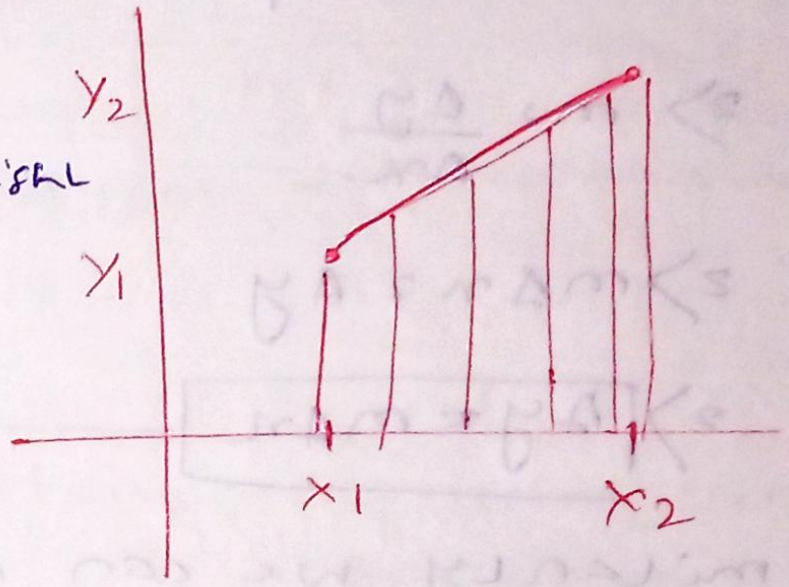
Output primitives:

Line drawing Algorithm

The cartesian slope intercept eqn for a straight line

$$y = m.x + b$$



With m represent the slope of the line and b is the y intercept.

Given that two end points of a line segment are specified at position (x_1, y_1) and (x_2, y_2) as shown in figure.

We can determine the value for the slope m and y intercept b with the following calculation

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{---------- (2)}$$

$$b = y_1 - mx_1 \quad \text{---------- (3)}$$

For any given x interval Δx along a line we can compute corresponding y interval Δy from eqn (2)

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

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

$$m \Delta x = \Delta y$$

$$\Delta y = m \Delta x \quad \text{---------- (4)}$$

Similarly we can obtain the corresponding x interval Δx corresponding to a specified Δy

$$\Delta x = \frac{\Delta y}{m} \quad \text{---------- (5)}$$

The eqn from the basic for determine the deflection voltage in analog device

For line with slope magnitude

$|m| < 1$ can be set proportional a small horizontal deflection voltage and the corresponding vertical deflection voltage is set proportional to Δy as calculated from eqn(4)

For line with slope magnitude

$|m| > 1$

Δy can be set proportional to a small vertical deflection voltage with corresponding horizontal deflection voltage set proportional to Δx as calculated from eqn(5)

For line with $m = 1$

$\Delta x = \Delta y$

the horizontal and vertical deflection voltage are equal.

DDA Algorithm:

The digital differential analyzer is a scan conversion line algorithm based on calculating either Δy or Δx.

Using eqn (4) and (5)

We sample the line at unit interval in one co-ordinate and determine corresponding integer value nearest the line path from the other co-ordinate

consider first a line with positive slope in the previous fig. If the slope is less than or equal to 1.

We sample at unit x interval ($\Delta x = 1$) and compute each successive y-value

$$y_{k+1} = y_k + m \quad \text{——— (1)}$$

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

$$\Rightarrow m = \frac{\Delta y}{\Delta x} = \frac{\Delta y}{1} \Rightarrow m = \Delta y$$

$$\Rightarrow m = y_2 - y_1$$

$$\Rightarrow y_2 = y_1 + m \quad \text{——— (2)}$$

For line with positive slope greater than 1 we reverse the role of x and y i.e we sample at unit y interval ($\Delta y = 1$) and calculate each successive x value as

$$x_{k+1} = x_k + 1/m \quad \text{——— (3)}$$

$$m = \frac{y_2 - y_1}{x_2 - x_1} \Rightarrow m = \frac{\Delta y}{\Delta x}$$

$$\Rightarrow m = \frac{1}{\Delta x} \Rightarrow m = \frac{1}{x_2 - x_1} \Rightarrow x_2 = x_1 + 1/m$$

Eqn (3) and (4) are based on assumption that lines are to processed from left end point and right end point.

If this processing is reversed so that the starting end point is at right then we have $\Delta x = -1$ and

$$y_{k+1} = y_k - m \quad \text{---(5)}$$

we have $\Delta y = -1$

$$x_{k+1} = x_k - 1/m \quad \text{---(6)}$$

Eqn (1) and (5) can also be used to calculate pixel position along a line with negative slope. If the absolute value of the slope is less than 1 and the start end point is at left we set $\Delta x = 1$ calculate y value when the starting end point is at right. We set $\Delta x = -1$

- similarly we have absolute value of a negative slope is greater than 1

$\Delta y = 1$ ~~or (2)~~

$\Delta y = -1$

This algorithm is summerized in the following procedure which accept as input the two end point pixel position.

- Horizontal and vertical difference bet the end point position are assigned to parameter Δx and Δy.

Advantage:

① It is easy to implement.

② DDA can avoid the floating point multiplication

Disadvantage:

① It is core accuracy at the end point.

② on the DDA floating point add is there which take more time than integer add

Algorithm:

1. $dx = |x_2 - x_1|$
2. $dy = |y_2 - y_1|$
3. If ($dx \geq dy$)
   length = dx
4. else
   length = dy

5. $x = x_1$

$y = y_1$ i = 0.

6. $\Delta x = \frac{x_2 - x_1}{length}$

7. $\Delta y = \frac{y_2 - y_1}{length}$

7. while ($i \leq length$)

$x = x + \Delta x$

$y = y + \Delta y$

$i = i + 1$

Q: Draw the line (10,8) and (0,0) using DDA algorithm

$dx = |x_2 - x_1| = |10 - 10| = 10$

$dy = |y_2 - y_1| = |8 - 8| = 8$

$dx > dy$ length $dx = 10$

$dy = 8$

$\Delta x = \frac{x_2 - x_1}{length} = \frac{0 - 10}{10} = -1$

$\Delta y = \frac{y_2 - y_1}{length} = \frac{0 - 8}{10} = -0.8$

	Plot round(x) round(y)	x + Δx	y + Δy
0	(10, 8)	10 - 1 2 9	8 - 0.8 2 7.2
1	(9, 7)	8	6.4
2	(8, 6)	7	5.6
3	(7, 6)	6	4.8
4	(6, 5)	5	4
5	(5, 4)	4	3.2
6	(4, 3)	3	2.4
7	(3, 2)	2	1.6
8	(2, 1)	1	1
9	(1, 0)	0	-0.2
10	(0, 0)	-1	-1

Q: Using DDA algorithm generate the point between the end point (6, 9) and (11, 19)

Ans: $(x_1, y_1) = (6, 9)$

$(x_2, y_2) = (11, 19)$

$dx = 11 - 6 = 5$

$dy = 19 - 9 = 10$

As $abs(dy) > abs(dx)$

$10 > 5$

step = abs(dy) = 10

xinc = 5/10 = 0.5

yinc = 10/10 = 1

x = 6 y = 9 is the initial point.
10 times calculated x and y
since step = 10

x = 6 + 0.5 = 6.5 y = y + 1 = 9 + 1 = 10

7	11
7.5	12
8.0	13
8.5	14
9.0	15
9.5	16
10	17
10.5	18
11	19

Step-1: Input the co-ordinate of two endpoints $A(x_1, y_1)$ and $B(x_2, y_2)$ for the line AB respectively. Note that point A and B are not equal. If they are equal then it is a point. Plot the point and return

Step-2: calculate dx and dy

dx = x2 - x1

& dy = y2 - y1

Step-3: calculate step.

if abs(dx) >= abs(dy)

then step = abs(dx)

else step = abs(dy)

Step-4: calculate the increment factor

$x_{inc} = \frac{dx}{step}$ $y_{inc} = dy/step$.

Step-5: initialize the initial point on the line and plot

x = x1 and y = y1

plot(x, y)

Step-6: Obtain the new pixel on the line and plot for k = 0 to step-1

do x = x + xinc

y = y + yinc

plot(round(x), round(y))

Step-7: Finish.

Bresenham Line drawing Algorithm

- An accurate and efficient raster line generation algorithm developed by Bresenham's convert line using only incremental integer calculation that can be adapted to display circles and other curves.
- It is an accurate line drawing algorithm that can used raster scan method for incremental integer calculation.
- Scan conversion process is used for drawing the straight line.
- For the co-ordinate (x_k, y)
- For (x_k, y)

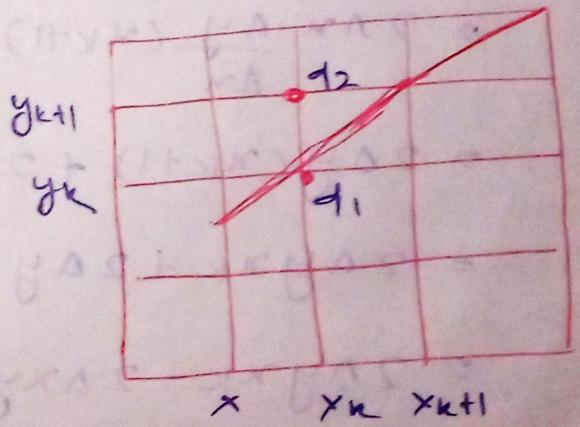
$$y = m(x_k) + b$$

- For (x_{k+1}, y)

$$y = m(x_{k+1}) + b$$

$d_1 = y - y_k$

$d_2 = y_{k+1} - y$



For the position (x_{k+1})

$d_1 = y - y_k$

$d_2 = y_{k+1} - y$

$d_1 = y - y_k$

$= m(x_{k+1}) + b - y_k$ ——— (1)

$d_2 = y_{k+1} - y$

$= y_{k+1} - m(x_{k+1}) - b$ ——— (2)

$d_1 - d_2 = m(x_{k+1}) + b - y_k - y_{k+1} + m(x_{k+1}) + b$

$= 2m(x_{k+1}) + 2b - 2y_k - 1.$

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

$= \Delta x [2m(x_{k+1}) + 2b - 2y_k - 1]$

$= \Delta x \left[2\frac{\Delta y}{\Delta x}(x_{k+1}) + 2b - 2y_k - 1\right]$

$= 2\Delta x \cdot \frac{\Delta y}{\Delta x}(x_{k+1}) + 2b\Delta x - 2y_k\Delta x - \Delta x$

$= 2\Delta y(x_{k+1}) + 2b\Delta x - 2y_k\Delta x - \Delta x$

$= 2\Delta y x_k + 2\Delta y + 2b\Delta x - 2y_k\Delta x - \Delta x$

$= 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2b\Delta x - \Delta x$

$p_2 = 2\Delta y x_k - 2\Delta x y_k + c$ ——— (2)

At (x_0, y_0)

$p_0 = c$

$\Rightarrow p_0 = 2\Delta y + 2\Delta x - \Delta x$

$\Rightarrow p_0 = 2\Delta y + \Delta x (2-1)$

The sign p_k is same as the sign of $d_1 - d_2$ since $\Delta x > 0$ for our ~~sample~~ example. parameter c is constant and has the value $2\Delta y + \Delta x(2-1)$ of pixel position. and will eliminated in the recursive calculation.

- If the pixel y_k is closer to the line path that the pixel at y_{k+1} (i.e $d_1 < d_2$) then decision parameter is negative. In this case we plot the lower pixel Otherwise we plot upper pixel.
- co-ordinate changes along the line occur in unit steps in either the x and y direction. Therefore we can obtain the successive decision parameter using incremental integer calculation.

The decision parameter is evaluated from eqn (3)

$$P_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c \quad \text{---(4)}$$

Substracting eqn(3) from eqn(4)

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

$$= 2\Delta y x_{k+1} + 2\Delta y - 2\Delta x y_{k+1} - 2\Delta y x_k + 2\Delta x y_k$$

$$= 2\Delta y - 2\Delta x (y_{k+1} - y_k) \quad \text{---(5)}$$

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x (y_{k+1} - y_k) \quad \text{---(6)}$$

where the term $y_{k+1} - y_k$ is either 0 or 1 depending on the sign of parameter P_k.

- The recursive calculation of the decision parameter is performed at each integer calculation. starting at the left co-ordinate end point of the line.
- The first parameter P_0 is evaluated at from eqn (3) at starting pixel position (x_0, y_0) with m evaluated as $\frac{\Delta y}{\Delta x}$

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

$$\frac{\Delta y}{\Delta x}\left[P_k = 2\Delta y x_k - 2y_k\Delta x + 2\Delta y + \Delta x(2i-1)\right.$$

$$= 2\Delta y x_k^0 - 2y\Delta x + 2\Delta y + \Delta x^0$$

$$\left. = 2\Delta y - \Delta x\right]$$

Algorithm:

1. Input the two line end point and the left end point (x_0, y_0)
2. Load (x_0, y_0) into the frame buffer that is plot the 1st point
3. Calculate constant 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)

$$p_{k+1} = p_k + 2\Delta y$$

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.

Q: Illustrate the Bresenham's line drawing algorithm with end point (20,10) and (30,18)

Ans: $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{18-10}{30-20} = 0.8 < 1$

$\Delta x = 30 - 20 = 10$

$\Delta y = 18 - 10 = 8$

$2\Delta y - 2\Delta x = 2 \cdot 8 - 2 \cdot 10 = -4$

$2\Delta y = 16$

$P0 = 2\Delta y - \Delta x = 16 - 10 = 6$

We plot the initial point $(x_0, y_0) = (20,$ determine the successive pixel along the line path

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)

Mid point circle algorithm:

The eqn for a circle

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

where r is the radius of circle whose centre is at (0,0)

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

From this we have

$y_0 = \sqrt{20^2 - 0^2} \simeq 20$

$y_1 = \sqrt{20^2 - 1^2} \simeq 20.$

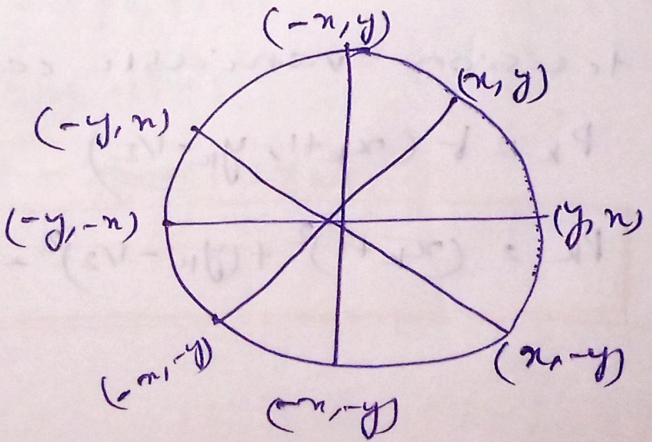
$y_2 = \sqrt{20^2 - 2^2} \simeq 20$

$y_{19} = \sqrt{20^2 - 19^2} \simeq 6$

$y_{20} = \sqrt{20^2 - 20^2} \simeq 0$

This is an efficient way of finding the value of y at unit x interval. Because of the following reasons.

- The ~~circle~~ resulting circle has large gap.
- The calculation are not very efficient as it has square and square root operation
- Mid point circle drawing is an efficient algorithm
- To make circle drawing algorithm more efficient we have to center the circle at (0,0)
- By doing this the circle will have eight way symmetry
- In mid point circle algorithm we use eight way symmetry.



So only even calculating the point from the top right eight of a circle and use symmetry to get the rest of the point

- Developed by John Bresenham
- Assume that we have just plotted point (x_k, y_k)
- the next point is a choice between

  (x_k+1, y_k) and (x_k+1, y_k-1)

- We like to choose the point that is nearest to the actual circle.
- The eqn of the circle be written as

$$f(x,y) = x^2 + y^2 - r^2$$

- The eqn evaluated as follows

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

- By evaluating this function at the midpoint between

  (x_k+1, y_k) and (x_k+1, y_k-1) we can make our decision.

The decision variable can be defined as

$$P_k = f(x_k+1, y_k - 1/2)$$

$$P_k = (x_k+1)^2 + (y_k - 1/2)^2 - r^2$$

If $P_k < 0$ the mid point $(x_{k+1}, y_k - 1/2)$ is inside the circle i.e (x_k+1, y_k) is closer to the circle.

ELSE the mid point $(x_{k+1}, y_k - 1/2)$ is outside the circle i.e (x_k+1, y_k-1) is closer.

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

$$P_{k+1} = (x_k+1+1)^2 + (y_{k+1} - 1/2)^2 - r^2$$

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

$$= (x_k+1)^2 + 2(x_k+1) + 1 - (x_k+1)^2 + (y^2_{k+1} - y_{k+1} + 1/4) - (y^2_k - y_k + 1/4)$$

2> $$P_{k+1} = P_k + 2(x_{k+1}) + (y^2_{k+1} - y^2_k) - (y_{k+1} - y_k) + 1$$

Case-1: $P_k < 0$ so $y_{k+1} = y_k$

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

$P_k > 0$ so $y_{k+1} = y_k - 1$

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

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

2> $$P_{k+1} = P_k + 2x_{k+1} - 2(y_k - 1) + 1$$

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

calculation of successive decision parameters

$P_k = b(x_{k+1}, y_k - 1/2)$

putting $k = 0$ $\quad b(x_0+1, y_0 - 1/2)$

$= b(0+1, \text{boundary } r - 1/2)$

we start the point in co-ordinate $(0, r)$

$P_0 = b(1, r - 1/2)$

$= 1 + (r - 1/2)^2 - r^2$

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

$= \frac{5}{4} - r \approx 1 - r$

$\Rightarrow \boxed{P_0 = 1 - r}$

Q: Draw a circle centered at $(0,0)$ of radius

Sol: $(x_0, y_0) = (0, 10)$

$P_0 = \frac{5}{4} - 10 \approx 1 - 10 = -9$

$k = 0$ so next point to be plotted is $(1, 10)$

Since $P_0 < 0$

$P_1 = P_0 + 2 \cdot 1 + 1 = -9 + 2 + 1 = -6$

$k = 1$ next point to be plotted is $(2, 10)$ since $P_1 < 0$

$P_2 = P_1 + 2 \cdot 2 + 1 = -6 + 4 + 1 = -1$

$k = 2$ next point is $(3, 10)$ since $P_2 < 0$

$P_3 = P_2 + 2 \cdot 3 + 1 = -1 + 6 + 1 = 6$

k=3 next point is (4,9) since $p_3 > 0$

~~$p_{k+1} = p_3 + 2 \cdot 4 + 1 - 2 \cdot 9$~~

$p_4 = p_3 + 2 \cdot 4 + 1 - 2 \cdot 9 = 6 + 8 + 1 - 18 = -3$

k=4 next point is (5,9) since $p_4 < 0$

$p_5 = p_4 + 2 \cdot 5 + 1 = -3 + 10 + 1 = 8$

k=5 next point is (6,8) since $p_5 > 0$

$p_6 = p_5 + 2 \cdot 6 + 1 - 2 \cdot 8 = 8 + 12 + 1 - 16 = 5$

k=6 next point is (7,7) since $p_6 > 0$

$p_7 = p_6 + 2 \cdot 7 + 1 - 2 \cdot 7 = 5 + 14 + 1 - 14 = 6$

Now $x > = y$ so stop.

Note that for each of the points ~~to/to~~ (1,10), (2,10) .... (7,7) determine its symmetry points and plot.

Advantage:

Eight-way symmetry can hugely reduce the work in drawing a circle.

Algorithm:

1. Input radius r and circle center (x_c, y_c) then set the co-ordinate for the first point on the circumference of a circle center at origin as

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

Step-2: calculate the initial value of the decision parameter

$$P_0 = \frac{5}{4} - r \simeq 1 - r$$

Step-3: Starting with k=0 at each position x_k perform the following test

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

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

Else the next point along the circle is (x_{k+1}, y_{k-1})

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

Step-4: Determine symmetry point on other seven point

Step-5: Move each calculated pixel position (x, y) onto the circular path centered at (x_c, y_c) to plot the co-ordinate value

$$x = x + x_c$$
$$y = y + y_c$$

Step-6: Repeat step 3 to 5 until $x > y$ not satisfied (Repeat while $x < y$)

polygon filling

- A chain of connected line segment often called: polygons.
- polygon are closed polylines where starting and ending vertices are same.
- polygon may be either convex or concave

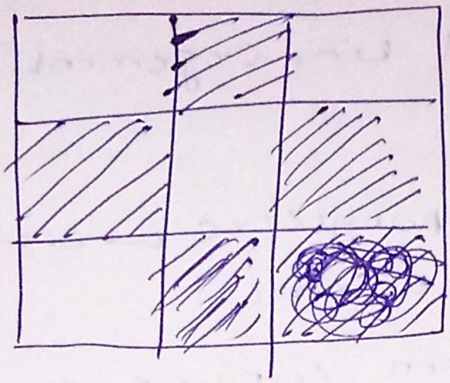
filling polygon

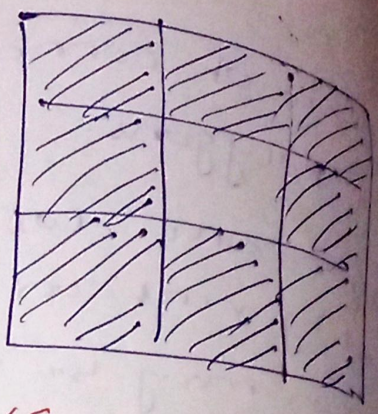
* which pixel to fill
* what to fill them

- Fill algorithm deal with pixel defined region
- A pixel defined region is a group of pixel with same color that are connected to one another.
- Two pixel are connected where there is an unbroken path of adjusting pixel connecting them.
- Two commonly of adjacent pixel.

4-adjacent: two pixel are 4-adjacent if they lie next to each other horizontally or vertically but not diagonally.

8-adjacent: Two pixel are 8-adjacent if they lie next to each other horizontally, vertically or diagonally.





(4-connected) (8-connected)

Seed-fill: this technique work by assuming a point called seed point has to be inside the polygon and search adjacent point can fill that are inside the polygon.

→ If a newly discover adjacent point is found inside the polygon then it become the new seed and the algorithm continue
→ If the adjacent point found is not inside the polygon then the boundary has been found.

- this algorithm only applicable to raster devices.

Seed Fill: Boundary Fill:

- It is used to fill an area with a specified colour until the specified boundary colour is encountered.
- this algorithm starts from a specified

Flood fill:

This algorithm is used when an area defined with multiple colour boundaried.

(1) Suppose we want to color the entries area whose original color is interior colour and replace it with bill colour

(2) then we start with a point in this area then colour all surrounding point until we set a pixel that is not interior colour.

(3) Start at a point inside a region

(4) Replace a specific interior colour (old colour will bill colour)

(5) Fill the 4-connected and 8-connected region until all interior point being placed

Character generation:

→ We can display letters and numbers in variety of size and style.

→ The overall design style for the set of character is called typeface.

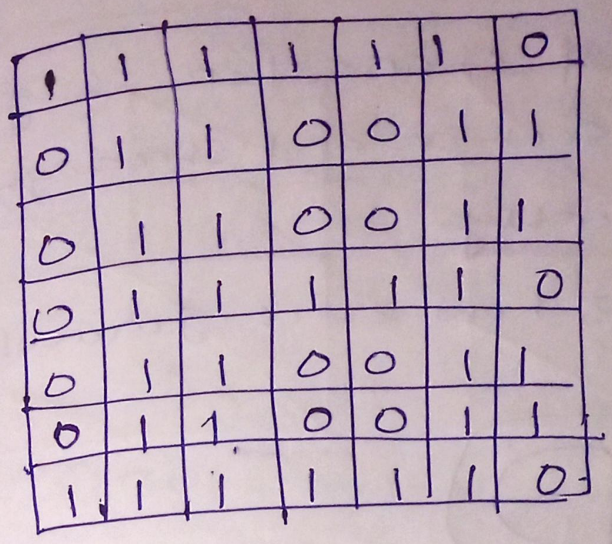
→ Today large number of typeface are available for computer application for example Helvetica, and New York Palatino etc.

→ originally the term font referred to set of cast metal character forms in particular size and format such as 10 point courier italic or 12-point palatino Bold. NOW the term font and typeface are often used interchangeable since printing is no longer done with cast metal form.

→ Two diff. representation are used for storing computer font

Bitmap font:

→ A simple method for representing the character shapes in a particular typeface is to use rectangular grid patterns.



→ When the pattern is copied to the area of frame buffer, the 1 bits designate which pixel position are to displayed on the monitors.

→ Bit map font are simplest to define and display as character grid only need to be mapped to a frame buffer position.

→ Bit map font requires more space bcoz each variation must be stored in a font cache.

→ It is possible to generate diff. size and other variation from one set but this usually does not produce good result.

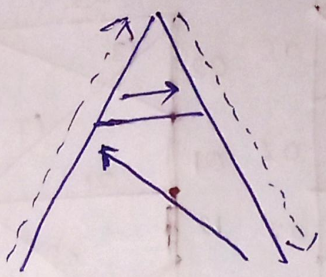
Outline font:

- In this method character is generated using curve section and straight line as combine assembly.
- Fig below shows how it is generated



- To display the character we need to tell interior region of the character
- This method required less storage since each variation does not required a distinct font cache.
- We can produce boldface or different size by manipulating the curve def^n for the character outline.
- But this will take more time to process the outline fonts bcoz they must be scan converted into the frame buffer

Stroke method:



Stroke method for letter A

-> It uses small line segement to generate a character

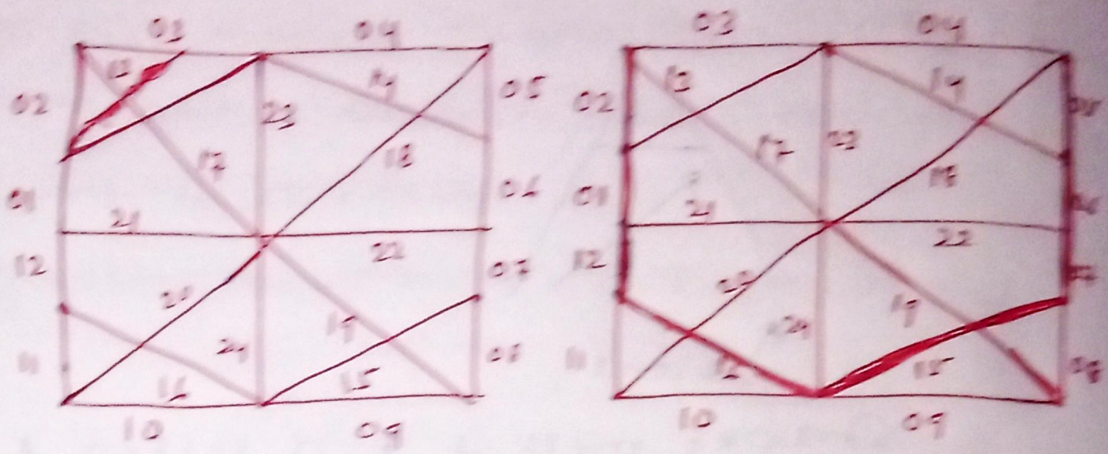
-> The small series of line segement are drawn like a stroke of a pen to form a character.

-> We can generate our own stroke method by calling line drawing algorithm

-> Here it is necessary to decide which line segement are needs for each character and then draw that line to display character.

-> It support scaling by changing length of line segement.

Starburst Method:



- In this method a fix pattern of line segment are used to generate character.
- There are 24 line segments.
- We highlight those line which are necessary to draw a particular character.
- Pattern for particular character is stored in the form of 24 bit code. in which each bit represents corresponding line having that number.
- That code contain 0 or 1 based on line segment need to highlight line and 0 for other line.
- Code for letter V is

  110011100001001100000000

- This technique is not used now a days bcoz it requires more memory to store 24 bits code for single character.

Inside - Outside Test

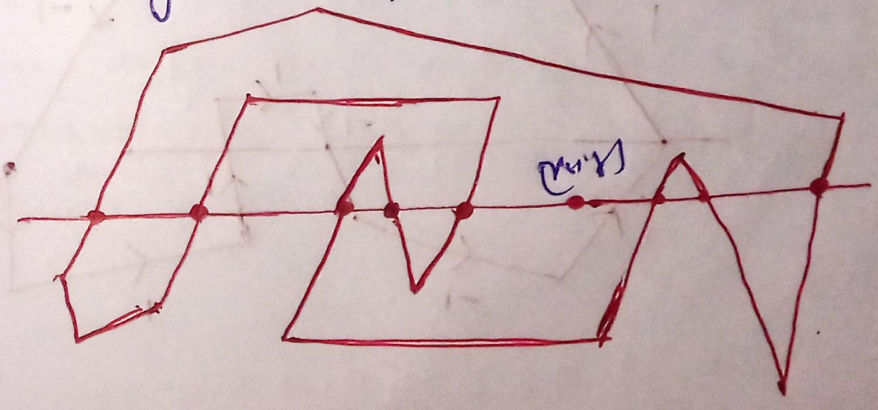
This method is also known as counting number method.

While filling an object we often need to identify whether particular point is inside the object or outside it.

- There are two methods by which we can identify whether particular point is inside an object or outside.

Odd-Even Rule:

- In this technique we will count the edge crossing along the line from any point x, y to infinity.
- If the number of interactions is odd then the point x, y is an interior point. and if the number of interaction is even then the point x, y is an exterior point.
- The following example depicts this concept

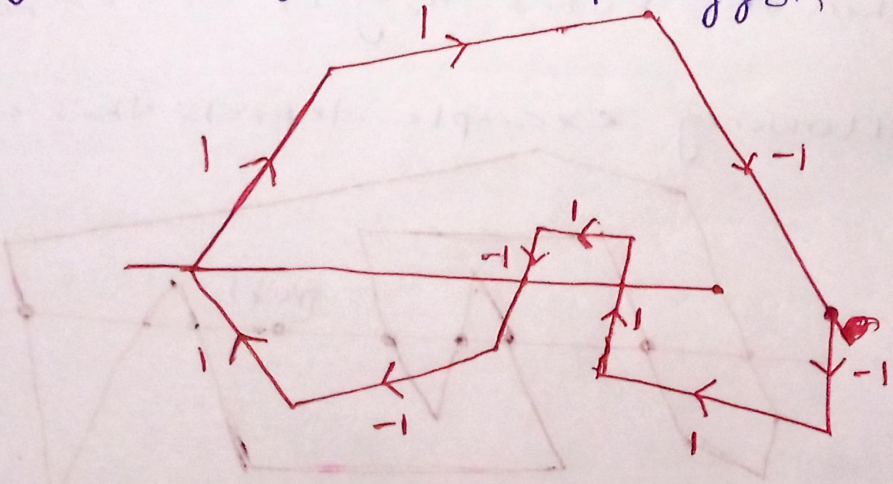


From the above figure we can see that from the point x, y, the number of intersection point on the left side is 1 and the right side is 3.

- From both ends the number of intersection point is odd, so the point is considered within the object.

NONZERO winding NUMBER rule:

- This method is also used with the simple polygon to test the given point is interior or not.
- It can simply understood with the help of a pin and a rubber band.
- Fix up the pin on one of the edge of the polygon and tie-up the rubber band with it and then stretch the rubber band along the edge of the polygon



- when all the edges of the polygon are covered by the rubber band check out the pin which has been fixed up at the point to test.
- If we find at least one wind at the point consider it within the polygon else we can say that the point is outside the polygon.
- In another alternative method give directions to all the edge of the polygon Draw a scan line from the point to be test towards ~~from~~ the left most of x direction.

i) Give the value 1 to all the edges which are going to upward direction and all other -1 as direction value.

ii) Check the edge direction values from which the scan line is passing and sum up them

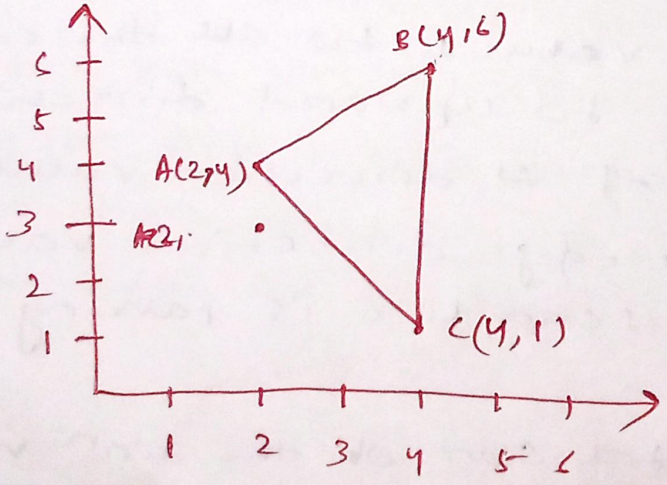
iii) If the total sum of the dirn value is non zero, then this point to be tested is an interior point otherwise it is an exterior point.

iii) In the above figure we sum up the direction value from which the scan line is passing then the total is 1-1+1=1 which is non zero. So the point is said to be an interior point.

Problem on scanline polygon filling:

Explain scan line polygon filling algorithm. Determine the content of the actual edge table to fill the polygon with vertices ~~A(2,4) B(2,7) and C(4,4)~~ A(2,4) B(4,6) and C(4,1) for y 1 to y6

Soln:



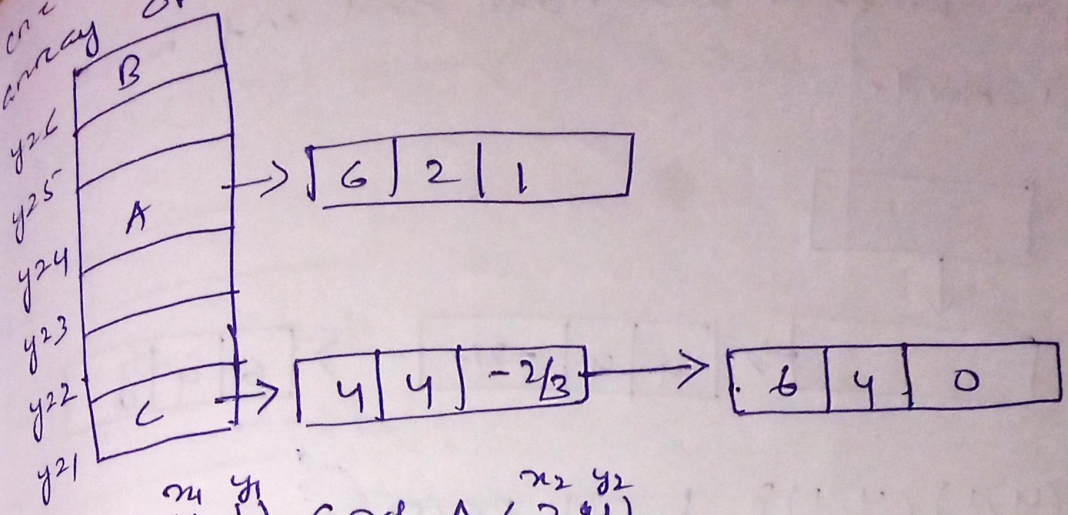
Sort the edge from ymin → ymax

A(2,4) B(4,6) and C(4,1)

Sort ymin → ymax

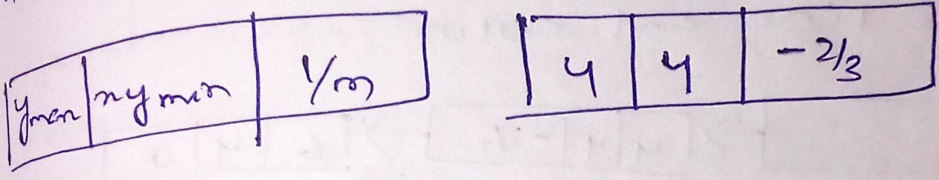
C(4,1) A(2,4) and B(4,6)

create a global edge table. It is an array of linked list



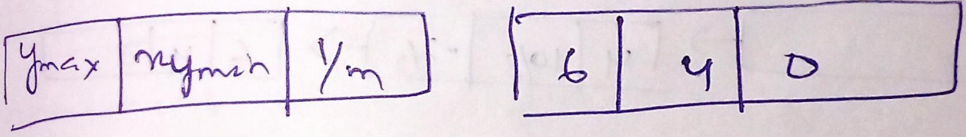
CA = C(4,1) and A(2,4)

(x_1 y_1) (x_2 y_2)



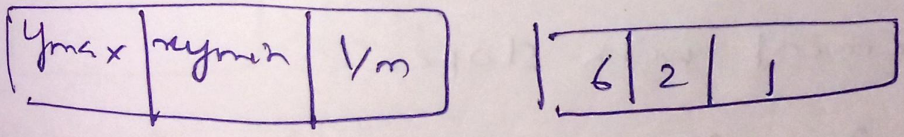
$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{4-1}{2-4} = 3/-2$$

(II) CB = C(4,1) and C(4,6)



$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{6-1}{4-4} = 5/0$$

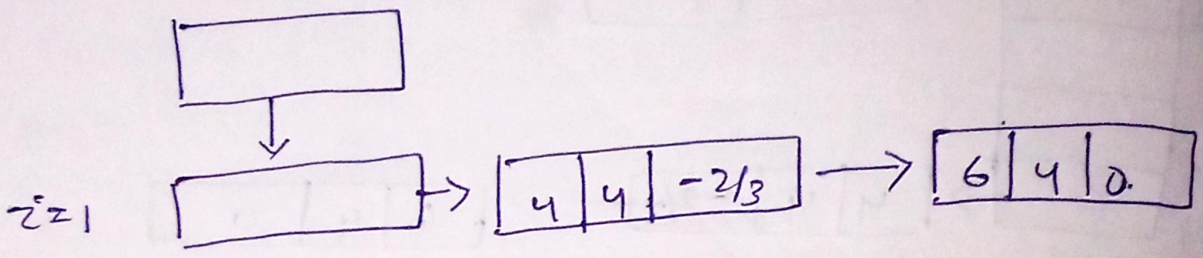
(III) AB = A(2,4) and B(4,6)



$$m = \frac{6-4}{4-2} = 2/2 = 1$$

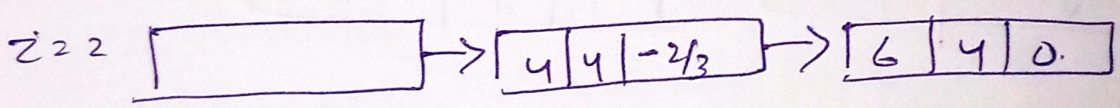
create an active edge table

Begin with null node and end with null node



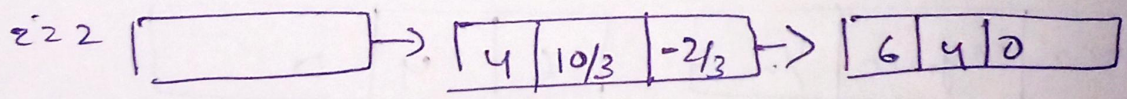
{(4,1) (4,1)} first pair

Draw a line and color



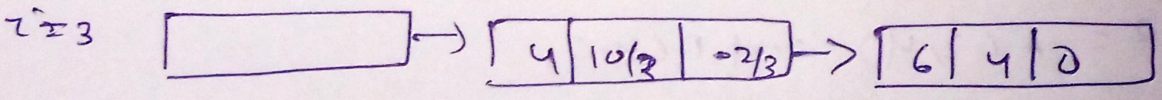
Increment with slope.

$4 - 2/3 = \frac{12-2}{3} = 10/3$



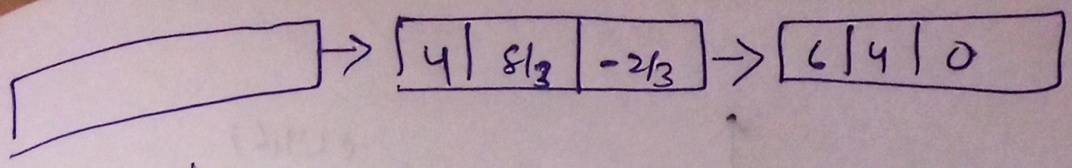
{(3.3, 2) (4,2)} Second pair.

Draw a line and color

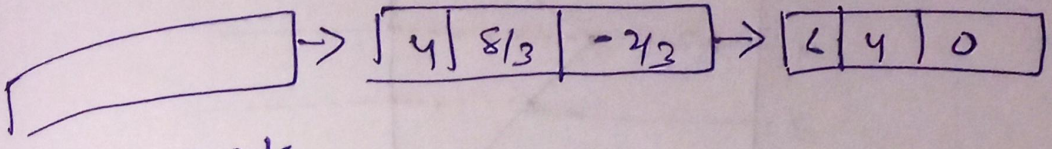


Increment with slope

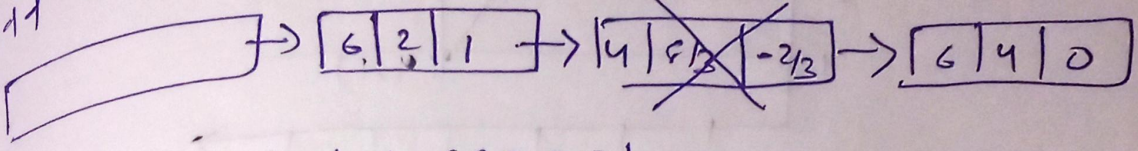
$\frac{10}{3} + \frac{2}{3} = 6/3$



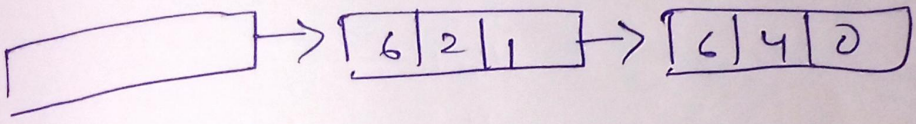
{(2,6,3),(4,3)} Third pair



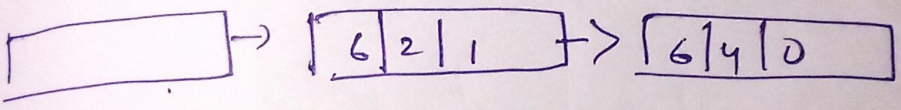
Add the node



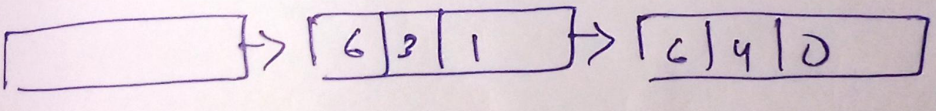
with 3 nodes we cannot make a pair so delete node whose first value and i value is same.



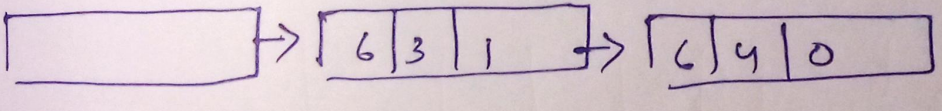
{(2,4)(4,4)} Fourth pair.



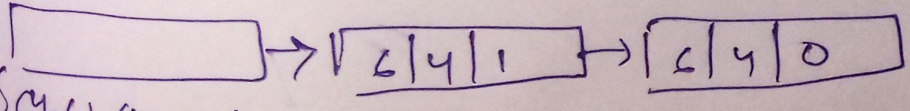
Increment with slope.



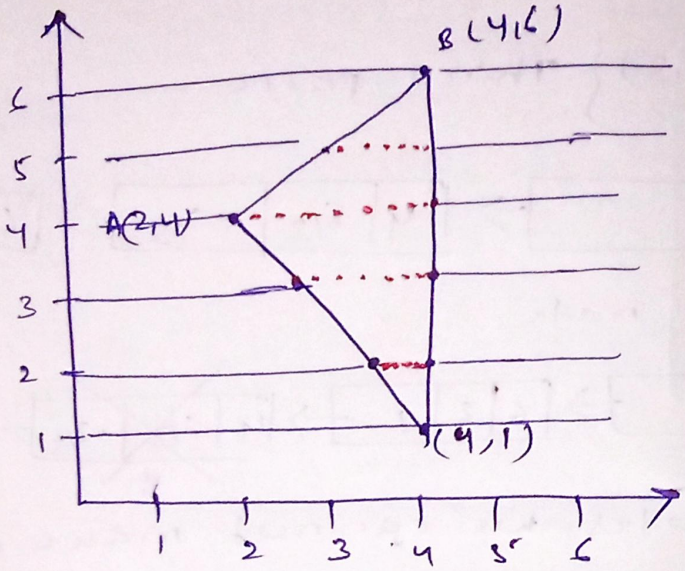
{(3,5)(4,5)} Fifth pair



Increment with slop



{(4,6)(4,6)} Sixth pair


B(4,6)
6
5
4
A(2,4)
3
2
1
(4,1)
1
2
3
4
5
6