Module - 2
Output Primitives

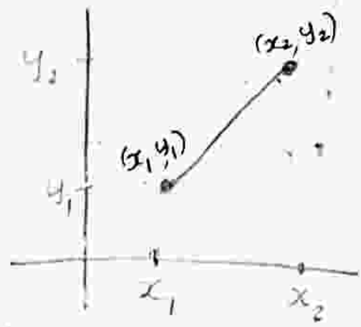
Line drawing is accomplished by calculating intermediate positions along the line path between two specified endpoint positions. An output device is then directed to fill in these positions between the endpoints.

Line Drawing Algorithms

The cartesian slope-intercept equation for a straight line is,

$$y = mx + b$$

where, m = slope of the line
b = y intercept



Given that the two endpoints of a line-segment are specified at positions (x_1, y_1) and (x_2, y_2)

here,

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

$$b = y - mx$$

For any given x interval Δx along a line, we can compute the corresponding y interval Δy

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

Similary we can obtain Δx corresponding to Δy as,

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

, These eqns form the basis for determinig deflection voltages in analog devices

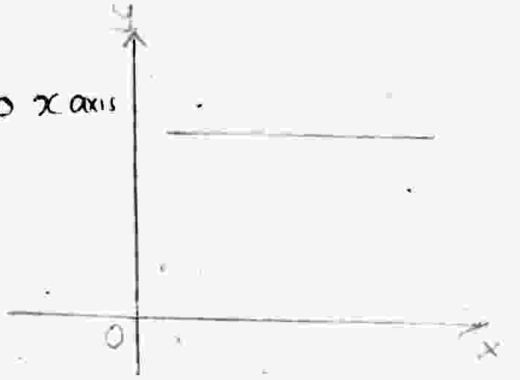
3 cases

I For lines with slope magnitudes $|m| < 1$, Δx can be set proportional to a small horizontal deflection voltage and the corresponding vertical defletion is then set proportional to Δy.

ie,

$|m| < 1$

→ parallel to x axis



II. For lines whose slopes have magnitudes $|m|>1$, Δy can be set proportional to a small vertical deflection voltage with the corresponding horizontal deflection voltage set proportional to Δx.

ie,

$|m| > 1$

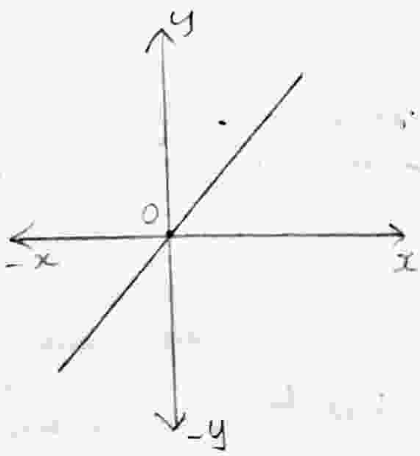
→ parallel to y axis

III. For lines with $m=1$, $\Delta x = \Delta y$ and the horizontal and vertical deflections voltages are equal. In each case, a smooth line with slope m is generated between the specified end points

ie,

$m=1$

→ $\Delta x = \Delta y$



DDA Algorithm

The digital differential analyzer (DDA) is a scan-conversion line algorithm.

Consider a line with +ve slope; ① if the slope is less than or equal to 1, we sample at unit x intervals ($\Delta x = 1$) and compute each successive y value as :-

ie, if $m \leq 1$ and ($\Delta x = 1$)

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

where,

→ k takes integer values from 1, for the first point and increases by 1 until the final endpoint is reached.

→ Since m can be any real number between 0 and 1, the calculated y values must be rounded to the nearest integer.

② For line with +ve slope greater than 1, we sample at unit y intervals ($\Delta y = 1$) and calculate each succeding x value as,

ie, if $m > 1$ and ($\Delta y = 1$)

$$x_{k+1} = x_k + \frac{1}{m}$$

③ If the processing is reversed so that the starting endpoint is at the right, then either we have $\Delta x = -1$ and,

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

or we have $\Delta y = -1$ with

$$x_{k+1} = x_k - \frac{1}{m}$$

Here,

Horizontal and vertical differences between the endpoint positions are assigned to parameters Δx and Δy. The difference ~~bdw~~ 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. If the magnitude of Δx is greater than the magnitude of Δy and ~~xa~~ xa is less than xb, the values of the increments in the x and y directions are l and m, respectively. If the greater change is in the x direction, but xa is greater than xb, then the decrements -l and -m are used to generate each new point on the line. Otherwise, we use a unit increment in the y direction and an x increment of 1/m. The DDA algorithm is a faster method for calculating pixel positions.

program

```
#include * device.h*
# 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; t++)
    {
```

```
        x += xIncrement;
        y += yIncrement;
        setPixel (ROUND(x), ROUND(y));
    }
}
```

The ~~rounding algorithm~~ operations and floating-point arithmetic in procedure of line DDA are time-consuming. (We can improve the performance of the DDA algorithm by seperating ~~the~~ the increment m and 1/m into integer and fractional parts so that all calculations are reduced to integer operations.)

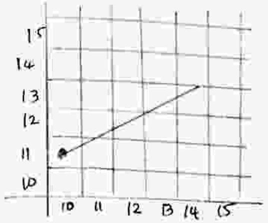
Drawbacks of DDL algorithm

1. Floating point values cannot be displayed on screen, so it needs to be rounded.
2. Processing time increases due to rounding of floating point numbers.
3. We can draw lines only on pixels

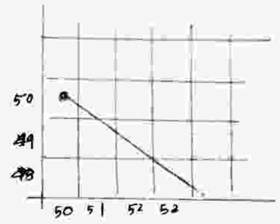
Bresenham's Algorithm

It is an accurate and efficient raster line-generating algorithm. Scan converts lines using only incremental integer calculations that can be adopted to display circles and other curves.

The vertical axes shows scan-line positions, and the horizontal axes shows identify pixel columns.



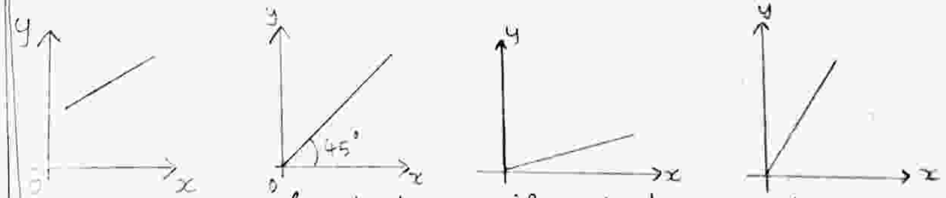
section of a screen where straight line segment is plotted



section of a screen, when a negative slope line segment is to be placed

Notes:

Basics



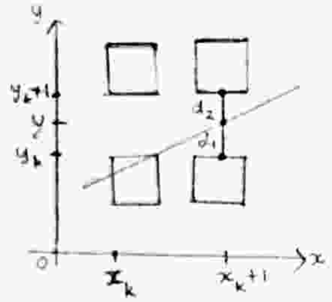
$y = mx + c$, $m = \frac{\Delta y}{\Delta x}$ | if $m = 1$ and $\Delta x = \Delta y$, angle $= 45°$ | if $m < 1$ and $\Delta x > \Delta y$, angle < 45 | if $m > 1$ and $\Delta x < \Delta y$, angle > 45

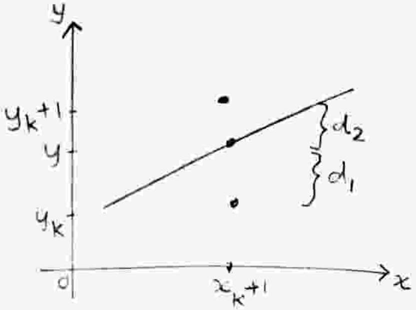
Rasterisation - Conversion of vector form to Raster form

Algorithm

To illustrate Bresenham's approach, consider scan-conversion process for lines with +ve slope less than 1. Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left end point (x_0, y_0) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path. Assuming we have determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel ~~pos~~ to plot in column x_{k+1}. Our choices are the pixels at positions (x_k+1, y_k) and (x_k+1, y_k+1)

At sampling position x_k+1, we label vertical pixel seperation from the mathematical line path as d_1 and d_2.





The y point on the mathematical line at pixel column positions x_k+1 is calculated as,

$y = mx + b$

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

$x_{next} = x_k + 1$

$y_{next} = y_k$ or y_k+1

$d_1 = \text{actual } y - y_k$

$d_1 = y - y_k$

$= m(x_k+1) + b - y_k$

$d_2 = (y_k+1) - \text{actual } y$

$= (y_k+1) - y$

$= y_k + 1 - m(x_k+1) - b$

Conditions,

if $(d_1-d_2)<0$
→ d_1 is small
→ y_k is small

ie, $d_1 < d_2$
so, pixel 1 is selected
$x_{i+1} = x_i + 1$
$y_i = y_i$

if $(d_1-d_2)>0$
→ d_2 small
→ y_k is incremented

ie, $d_1 > d_2$
So, pixel 2 is selected
ie, $x_{i+1} = x_i + 1$
$y_{i-1} = y_i - 1$

The difference between two seperations. ie, d_1 and d_2 is,

$$d_1 - d_2 = [m(x_k+1)+b-y_k] - [y_k+1-m(x_k+1)-b]$$

$$= m(x_k+1)+b-y_k-y_k-1+m(x_k+1)+b$$

$$= 2m(x_k+1) + 2b - 2y_k - 1 \quad —①$$

A decision parameter P_k for the k^{th} step in the line algorithm can be obtained by rearranging. It involves only integer calculations. We accomplish this by substituting $m = \frac{\Delta y}{\Delta x}$ where, Δy and Δx are the vertical and horizontal seperations of the endpoint positions and defining, (multiplying with Δx)

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

$$\Delta x(d_1-d_2) = \Delta x\left[2\frac{\Delta y}{\Delta x}(x_k+1) + 2b - 2y_k - 1\right]$$

$$P_k = \Delta x \times 2 \frac{\Delta y}{\Delta x}(x_k+1) + \Delta x \times 2b - \Delta x\, 2y_k - \Delta x$$

$$P_k = 2\Delta y(x_k+1) + 2\Delta x b - 2\Delta x y_k - \Delta x$$

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

Here this 2 terms varies and others are constants, so we avoid them.

$$P_k = 2\Delta y x_k - 2\Delta x y_k$$

The sign of P_k is the same as the sign of d_1-d_2, since $\Delta x>0$
Parameter c is constant and has the value $2\Delta y - \Delta x$(

which is independent of pixel position and will be eliminated in the recursive calculations for P_k. If the pixel at y_k is closer to the line path than the pixel at y_k+1 (ie, $d_1<d_2$) then decision parameter P_k is negative. In that case, we plot the lower pixel; otherwise we plot the upper level.

Coordinates change along the line occur in unit steps in either the x or y directions. Therefore, we can obtain the values of successive decision parameters using incremental integer calculations. At step $k+1$, the decision parameter is evaluated as,

$$P_{k+1} = 2\Delta y\, x_{k+1} - 2\Delta x\, y_{k+1} \quad -②$$

eqn② − eqn①

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

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

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

[But $x_{k+1} = x_k + 1$, so that] [if $P_{k+1} - P_k < 0$]

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

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

where the term $y_{k+1} - y_k$ is either 0 or 1, depending on the sign of parameter P_k. The first parameter P_k is evaluated at the starting pixel position (x_0, y_0) and with m evaluated as $\frac{\Delta y}{\Delta x}$,

Here $y_{k+1} = y_k$,

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

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

if $P_{k+1} - P_k \geq 0$ Here $y_{k+1} = y_k + 1$

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

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

if $P_k < 0$,

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

if $P_k \geq 0$,

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

Initial value

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

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

$$P_1 = 2\Delta y\, x_1 + 2\Delta y - 2\Delta x\, y_1 + 2\Delta x b - \Delta x \quad -③$$

$$y_1 = m x_1 + b$$

$$b = y_1 - m x_1$$

$$b = y_1 - \frac{\Delta y}{\Delta x} x_1 \quad -④$$

Sub ④ in ③

$$P_1 = 2\Delta y\, x_1 + 2\Delta y - 2\Delta x\, y_1 + 2\Delta x\left[y_1 - \frac{\Delta y}{\Delta x} x_1\right] - \Delta x$$

$$= 2\Delta y\, x_1 + 2\Delta y - 2\Delta x\, y_1 + 2\Delta x\, y_1 - 2\cancel{\Delta x}\frac{\Delta y}{\cancel{\Delta x}} x_1 - \Delta x$$

$$= \cancel{2\Delta y\, x_1} + 2\Delta y - \cancel{2\Delta x\, y_1} + \cancel{2\Delta x\, y_1} - \cancel{2\Delta y\, x_1} - \Delta x$$

$$\boxed{P_1 = 2\Delta y - \Delta x} \text{ initial value}$$

Bresenham's line Drawing Algorithm for $|m|<1$

Step 1: Input the two lines endpoints and store the left end point in (x_1, y_1)

Step 2: Load (x_1, y_1) into the frame buffer, ie; plot the 1st point

Step 3: Calculate constants $\Delta x, \Delta y, 2\Delta y, 2\Delta x$ and obtain the starting value for the decision parameter (P_k) as,

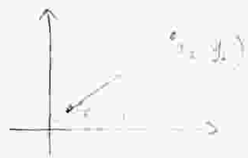
$$P_1 = 2\Delta y - \Delta x$$

Step 4: At each x, 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

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

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

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



Step 5: Repeat step 4 in Δx times

Illustration

$(x_1, y_1) = (1, 1)$

$(x_2, y_2) = (8, 5)$

$x = 1$

$y = 1$

$\Delta x = x_2 - x_1 = 8 - 1 = 7$

$\Delta y = y_2 - y_1 = 5 - 1 = 4$

$P_1 = 2\Delta y - \Delta x = 2\times4 - 7 = \underline{1}$

```
{ x = x1
  y = y1
  Δx = x2 - x1;
  Δy = y2 - y1;
  P1 = 2Δy - Δx,
  while (x <= x2)
  {
     putpixel(x,y)
     x++;
     if(p<0)
       p = p+2Δy;
     else
     {
       p = p+2Δy-2Δx
       y++;
     }
  }
}
```

while $(1 <= 8)$
putpixel (1, 1)
x++; // x = 2
if (1 < 0) ✗
else
$p = p + 2\Delta y - 2\Delta x$
$= 1 + 2\times4 - 2\times7$
$= 1 + 8 - 14$
$= 1 - 6 = \underline{-5}$
y++; // y = 2

while $(2 <= 8)$
putpixel (2, 2)
x++; // x = 3
if $(-5 < 0)$ ✓
$p = p + 2\Delta y$
$= -5 + 2\times4$
$= -5 + 8 = \underline{3}$
y = 2 itself (not incremented)

while $(3 <= 8)$
putpixel (3, 2)
x++, // x = 4
if (3 < 0) ✗
else
$p = p + 2\Delta y - 2\Delta x$
$= 3 + 2\times4 - 2\times7$
$= 3 + 8 - 14 = 11 - 14 = \underline{-3}$
y++; // y = 3

while $(4 <= 8)$
putpixel (4, 3)
x++, // x = 5
if $(-3 < 0)$ ✓
$p = p + 2\Delta y$
$= -3 + 2\times4$
$= -3 + 8$
$= \underline{5}$
y = 3 itself

x	y	P
1	1	1
2	2	-5
3	2	3
4	3	-3
5	3	5
6	4	-1
7	4	7
8	5	1

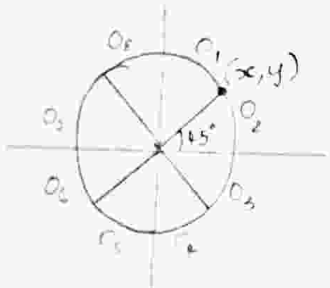
Circle Generating Algorithm

Circle

A circle is defined as the set of points that are all at a given distance r from a center position.

Equation of circle,

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \text{ where } (x_0, y_0) \text{ is the origin}$$



Calculation of a circle point (x, y) in one octant yields the circle points shown for the other seven octants.

Mid-point circle algorithm

Equation of a circle at midpoint (0,0):-

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

Here, x - increases in unit interval

y - can increase or decrease.

Next point can be :-

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

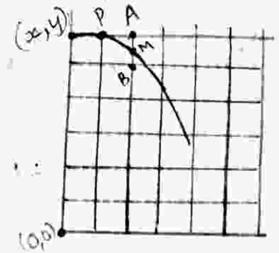
the point we choose depends on the decision parameter(P_k)

mid point is :- $\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right)$

ie, $= \left(\frac{(x_k+1)+(x_k+1)}{2}, \frac{y_k+y_k-1}{2}\right)$

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

$= \left[x_k+1, \frac{2y_k-1}{2}\right]$



$= x_k+1, \frac{2y_k}{2} - \frac{1}{2}$

$= \left[(x_k+1), (y_k - \frac{1}{2})\right]$

$P_k = f\left((x_k+1), (y_k - \frac{1}{2})\right)$

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

put value of x and y of P_k in $f(x,y)$

$(x_k+1)^2 + (y_k - \frac{1}{2})^2 - r^2 = 0$

$\longrightarrow P_k = (x_k+1)^2 + (y_k - \frac{1}{2})^2 - r^2$

2 cases,

if $P_k < 0$,

$d_1 < d_2$

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k$

midpoint is inside the circle; y_k is closer

if $P_k > 0$,

$d_1 > d_2$

$x_{k+1} = x_k + 1$

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

midpoint is outside the circle, y_{k-1} is closer

Similarly, we need successive decision parameters,

$$P_{(k+1)} = f\left((x_{k+1}+1),\left(y_{k+1}-\tfrac{1}{2}\right)\right)$$

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

$$P_{(k+1)} - P_k = ((x_k+1)+1)^2 + \left(y_{k+1}+\tfrac{1}{2}\right)^2 - r^2 - (x_k+1)^2 - \left(y_k-\tfrac{1}{2}\right)^2 + r^2$$

$$= \cancel{(x_k+1)^2} + 2(x_k+1) + 1 + (y_{k+1})^2 + 2\times(y_k+1)\times\tfrac{1}{2} + \tfrac{1}{4}$$

$$- \cancel{(x_k+1)^2} - \left[y_k^2 - 2\times y_k \times \tfrac{1}{2} + \tfrac{1}{4}\right]$$

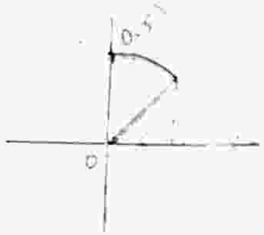
$$= 2(x_k+1) + (y_{k+1})^2 + (y_k+1) - y_k^2 - y_k + 1 + \cancel{\tfrac{1}{4}} - \cancel{\tfrac{1}{4}}$$

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

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

Initial decision Parameter :-

$x_k = 0$, $y_k = r$



$$P = (x_k+1)^2 + \left(y_k - \tfrac{1}{2}\right)^2 - r^2$$

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

$$= 1^2 + \left(r^2 - 2r\times\tfrac{1}{2} + \tfrac{1}{4}\right) - r^2$$

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

$$= 1 + \tfrac{1}{4} - r$$

$$P_0 = \tfrac{5}{4} - r \qquad \tfrac{5}{4} \approx 1$$

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

if $(P_k \geq 0)$,

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

coordinate is :- (x_k+1, y_k-1)

if $(P_k < 0)$,

$y_{k+1} = y_k$

coordinate is :- (x_k+1, y_k)

so,

$$\boxed{\begin{aligned} P_k &= 1 - r \\ P_{k+1} &= P_k + 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1}-y_k) + 1 \end{aligned}}$$

Example :

if $r = 8$ (given) ie, $(x, y) = (0, r)$

$P_k = 1 - r$

① $P_0 = 1 - 8 = -7$

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

② $P_1 = -7 + 2(0+1) + (8^2 - 8^2) - (8-8) + 1$

$= -7 + 2 + 0 + 0 + 1 = -7 + 3 = \underline{\underline{-4}}$

③ $P_2 = -4 + 2(1+1) + 0 - 0 + 1$

$= -4 + 4 + 1 = \underline{\underline{1}}$

④ $P_3 = 1 + 2(2+1) + (7^2 - 8^2) - (7-8) + 1$

$= 1 + 6 + 49 - 64 + 1 + 1$

$= 7 + -15 + 1 + 1$

$= \underline{\underline{-6}}$

⑤ $P_4 = -6 + 2(3+1) + 0-0 + 0-0 + 1$

$= -6 + 8 + 1 = \underline{3}$

⑥ $P_5 = 3 + 2(4+1) + (6^2 - 7^2) - (6-7) + 1$

$= 3 + 10 + 36 - 49 + 1 + 1$

$= \underline{2}$

k iteration	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	3	5	6
5	5	6	2	6	5

Here $x \geq y$, it indicate the end point of half octet.

whenever $x \geq y$, end of ~~first~~ half octet

Quadrant 1; (x, y)		Quadrant 2; $(-x, y)$	
(0,8)	(7,4)	(0,8)	(-7,4)
(1,8)	(7,3)	(-1,8)	(-7,3)
(2,8)	(8,2)	(-2,8)	(-8,2)
(3,7)	(8,1)	(-3,7)	(-8,1)
(4,7)	(8,0)	(-4,7)	(-8,0)
(5,6)		(-5,6)	
(6,5)		(-6,5)	

Character Generation

Letters, numbers and other characters can be displayed in a variety of sizes and styles. The overall design style for a set of characters is called a type face.

Type faces (or fonts) can be divided into two broad groups: serif and sans serif.

Serif type has small lines or accents at the end of the main character strokes, while sans serif type does not have accents. For example, the text in our text books is set in a serif font. But this sentence is printed in a sans-serif font. Serif type is generally more readable; ie, it is easier to read in longer blocks of text. On the other hand, the individual characters in sans-serif type are easier to recognize. For this reason, sans-serif type is said to be more legible. Since sans-serif characters can be quickly recognized, this type face is good for labeling and short headings.

Computer Fonts

Two different representations are used for storing computer fonts. They are:

- Bitmap font
- Outline font

Bitmap font

- A simple method for representing the character shapes is using rectangular grid patterns ~~[illegible]~~ and this character sets are reffered to ~~as~~ as bitmap font or (bitmapped font)

- When the pattern is represented in bitmap font and is copied to an area of the frame buffer, the 1 bits designate which pixel positions are to be displayed on the monitor.

- Bitmap fonts require more space, because ~~for~~ each variation (size and format) must be stored in a font cache.

figure -

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

Outline Font

- In this scheme, the character shapes are described/represented using straight lines and curve sections. It is more flexible than bitmap font.

- When the pattern/character shape is displayed in outline font, the interior of the character outline must be filled using the scan-line fill procedure.

- Outline fonts require less storage since each variation does not require a distinct font cache.

figure:-

Disadvantage

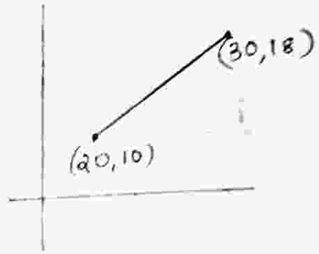
It does take more time to process the outline fonts, because they must be scan converted into the frame buffer.

Assignment

Plot a straight line with end points (20,10) and (30,18)

$\Rightarrow (x_1, y_1) = (20,10)$

$(x_2, y_2) = (30,18)$



Consider the algorithm,

$x = x_1 = 20$

$y = y_1 = 10$

$\Delta x = x_2 - x_1 = 30 - 20 = \underline{10}$

$\Delta y = y_2 - y_1 = 18 - 10 = \underline{8}$

$P_1 = 2\Delta y - \Delta x$

$= 2 \times 8 - 10 = 16 - 10 = \underline{6}$

i=0

while (20<=30) ✓

putpixel (20,10)

x++; // x=21

if (6<0) ✗

else

$p = p + 2\Delta y - 2\Delta x$

$= 6 + 2\times 8 - 2\times 10$

$= 6 + 16 - 20 = \underline{2}$

y++; // y= 11

i=1

while (21<=30)

putpixel (21,11)

x++; // x=22

if (2<0) ✗

else

$p = p + 2\Delta y - 2\Delta x$

$= 2 + 2\times 8 - 2\times 10$

$= 2 + 16 - 20 = \underline{-2}$

y++; // y=12

i=2

while (22<=30)

Algorithm steps

```
{
  x = x1;
  y = y1;
  Δx = x2 - x1;
  Δy = y2 - y1;
  P1 = 2 Δy - Δx
  while (x <= x2)
  {
     putpixel (x,y)
     x++;
     if (p<0)
       p = p + 2Δy;
     else
     {
       p = p + 2Δy - 2Δx
       y++;
     }
  }
}
```

putpixel (22,12)
x++; //x=23
if(-2<0) ✓
p=p+2Δy
= -2+2×8=16-2=14
y=12; (No change)

i=3
while (23<=30)
putpixel (23,12)
x++; //x=24
if (14<0) ×
else
p=p+2Δy-2Δx
=14+2×8-2×10=10
y++; //y=13

i=4
while (24<=30)
putpixel (24,13)
x++; //x=25
if (10<0) ×
else
p=p+2Δy-2Δx
=10×2×8-2×10=6
y++; //y=14

i=5
while (25<=30)
putpixel (25,14)
x++; //x=26
if (6<0) ×
else
p=p+2Δy-2Δx
=6+2×8-2×10=2
y++; //y=15

i=6
while(26<=30)
putpixel (26,15)
x++; //x=27
if(2<0) ×
else
p=p+2Δy-2Δx
=2+2×8-2×10
=-2
y++; //y=16

i=7
while (27<=30)
putpixel (27,16)
x++; //x=28
if (-2<0) ✓
p=p+2Δy
=-2+2×8=14
y=16 (No change)

i=8
while (28<=30)
putpixel (28,16)
x++; //x=29
if (14<0) ×
else
p=p+2Δy-2Δx
=14+16-20=10
y++; //y=17

i=9
while (29<=30)
putpixel (29,17)
x++; //x=30
if (10<0) ×
else
p=p+2Δy-2Δx
=10+16-20=6
y++; y=18

x	y	p
20	10	6
21	11	2
22	12	-2
23	12	14
24	13	10
25	14	6
26	15	2
27	16	-2
28	16	14
29	17	10
30	[illegible]	