Line Drawing

Point plotting is accomplished by converting a single coordinate position by an application program into approximate operation for the output device in use.

CRT electron beam is turned on to illuminate the screen phosphor at selected location.

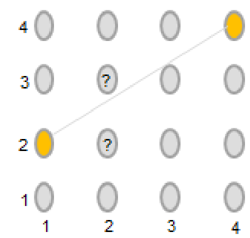
In random scan system, point plotting commands are stored in display list and coordinate values in these instructions are converted to deflection voltages that position the electron beam at that screen location to be plotted during each refresh cycle.

In case of black and white raster scan system a point is plotted by setting the bit value corresponding to specified screen position within frame buffer to 1.

For drawing lines, we need to calculate intermediate positions along the line path between two end points e.g. 10.45 is rounded off to 10 (causes stair cases or jaggies to be formed)

To load intensity value into frame buffer at position x, y use setpixel (x, y, intensity)

To retrieve current frame buffer intensity value for specified location use getpixel(x,y)



Digital Differential Analyzer Algorithm (DDA)

A scan conversion algorithm for lines computes the coordinates of the pixels that lie on a near an ideal infinitely thin straight line imposed on a 2D Raster grid.

It traces out successive (x,y) values by simultaneously incrementing x and y by small steps proportional to the first derivative of x and y.

Slope intercept equation of a straight line is $y = m x + c$

For points (x1, y1) and (x2,y2) slope m = y2 – y1/x2 - x1 or $\Delta y/\Delta x$

Algorithms for displaying lines depend on these equations for given interval x we can compute y interval

$\Delta y = m . \Delta x$ ‘x’ interval Δx is obtained by $\Delta x = \Delta y/m$

These equations form the basis for determining deflection voltages in analog devices

For lines with slope m = 1, Δy and Δx are equal, DDA scan conversion algorithm is based on calculating Δx, Δy

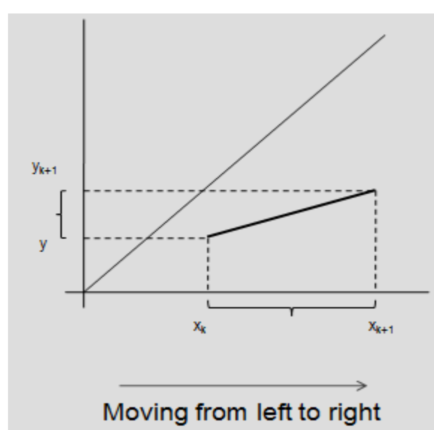
We sample line at unit interval in one coordinate and determine corresponding integer values nearest the line path for other coordinate

For Lines with Positive Slope

i. For Lines with |m| <=1 (Moving from Left to Right)

The simplest strategy for scan conversion of lines is to compute the slope ‘m’ as $\Delta y/\Delta x$ to increment x by 1 pixel starting with the leftmost point to calculate the next value of y_i, for each x and to intensify the pixel at (x_i,|y_i|). Then

$$m = y_{k+1} - y_k / \quad x_{k+1} - x_k$$
$$y_{k+1} = \quad y_k + m.(x_{k+1} - x_k)$$



And if $\Delta x = 1$ then $y_{k+1} = y_k + m$. Thus the unit change in x changes y by m , which is the slope of the line. For all points (x_k, y_k) on the line, we know that if $x_{k+1} = x_k + 1$ then $y_{k+1} = y_k + m$ i.e. the values of x and y are defined in terms of their previous values

Similarly

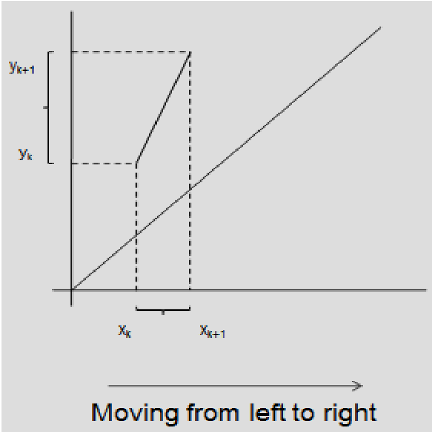
ii. For Lines with slope > 1 (Moving from Left to Right)

Perform unit increment in y direction $\Delta y = 1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = 1$ and compute successive x value as

$$x_{k+1} - x_k = 1/m$$

or $x_{k+1} = x_k + 1/m$

The x value computed must be rounded off to the nearest whole number



iii. For Lines with slope < 1 (Moving from Right to Left)

Perform unit increment in x direction $\Delta x = -1$ as $\Delta x > \Delta y$ i.e. $x_{k+1} - x_k = -1$ and compute successive y value as

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

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

The y value computed must be rounded off to the nearest whole number

iv. For Lines with slope > 1 (Moving from Right to Left)

Perform unit increment in y direction $\Delta y = -1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = -1$ and compute successive x value as

$$x_{k+1} - x_k = -1/m$$

or $x_{k+1} = x_k - 1/m$

The x value computed must be rounded off to the nearest whole number

For Lines with Negative Slope

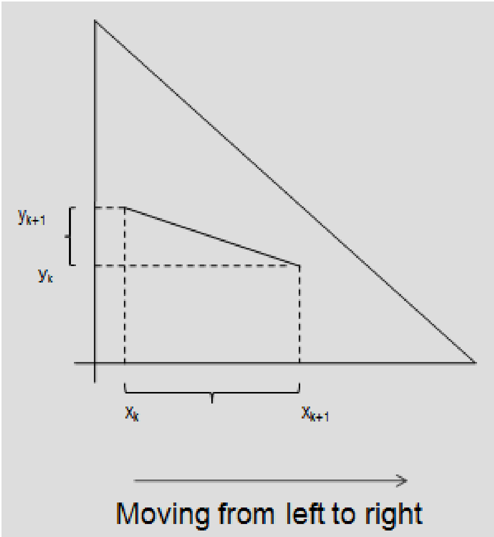
v. For Lines with |m| <=1 (Moving from Left to Right)

Perform unit increment in x direction $\Delta x = 1$ as $\Delta x > \Delta y$ i.e. $x_{k+1} - x_k = 1$ and compute successive y value as

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

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

The y value computed must be rounded off to the nearest whole number



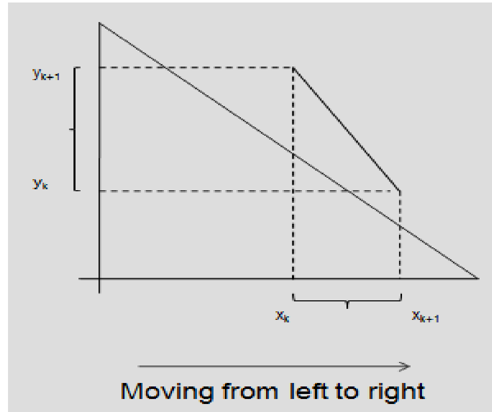
vi. For Lines with |m| > 1 (Moving from Left to Right)

Perform unit increment in y direction $\Delta y = -1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = -1$ and compute successive x value as

$$x_{k+1} - x_k = -1/m$$

or $x_{k+1} = x_k - 1/m$

The x value computed must be rounded off to the nearest whole number



vii. For Lines with |m| < 1 (Moving from Right to Left)

Perform unit increment in x direction $\Delta x = -1$ as $\Delta x > \Delta y$ i.e. $x_{k+1} - x_k = -1$ and compute successive y value as

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

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

The y value computed must be rounded off to the nearest whole number

viii. For Lines with |m| > 1 (Moving from Right to Left)

Perform unit increment in y direction $\Delta y = 1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = 1$ and compute successive x value as

$$x_{k+1} - x_k = 1/m$$

or $x_{k+1} = x_k + 1/m$

The x value computed must be rounded off to the nearest whole number

This algorithm is based on floating point arithmetic so it is slower than Bresenham's line drawing algorithm for drawing lines as Bresenham's line drawing algorithm is based on integer arithmetic approach.

Algorithm:

```
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=O; k<steps; k++){
         x += xIncrement;
         y += yIncrement;
        setpixel (ROUND(x), ROUND(y));
      }
}
```