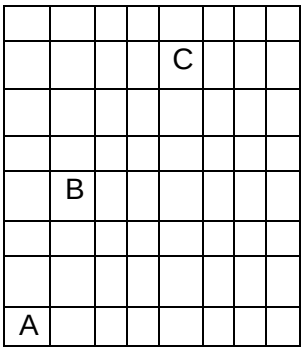
3. Graphics Output primitive

Contents

3.1 Point

- Pixel is a unit square area identified by the coordinate of its lower left corner.
- Each pixel on the display surface has a finite size depending on the screen resolution & hence a pixel can't represent a single mathematical number.
- Origin of the reference coordinate system being located of the lower left corner of the display surface.
- The each pixel is accused by non-negative integer coordinate pair(x, y).
- The x values start at the origin &increase from left to right along a scan line & y values start at the bottom & increase upwards.
- In the above diagram the coordinate of pixel A:0,0 ,B:1,4 , C:4,7.C:4,7.
- A coding position (4. 2, 7. 2) is represented by C.



- Whereas (1.5, 4.2) is represented by B.
- In order to half a pixel on the screen we need to round off the coordinate to a nearest integer.

Line Drawing

- Line drawing is accomplished by calculating intermediate point coordinates along the line path between two given end points.
- Screen pixel are referred with integervalues, plotted positions may only approximate the calculate coordinates, what is pixel which are intensified are those which lie very close to the line path.
- In a high resolution system the adjacent pixels are so closely spread that the approximated line pixels lievery close to actual line path and hence the plotted lines appear to be much smooth-almost like straight line drown on paper.
- In low resolution system the same approximation technique causes to display with stair step appearance that is not smooth.

Line Drawing Algorithm

- The equation of a straight line is

  Y=mX + b

Where **m** representing slope of the line and **b** as the y intercept.

- Given two end points of a line segment are (x_1, y_1)&(x_2, y_2)
- Then the straight line can be written as $y_1 = \frac{y_2 - y_1}{x_2 - x_1} x_1 + b$ (1)
- We can determine the volume for the slope 'm' & y intercept 'b' with the following calculation.

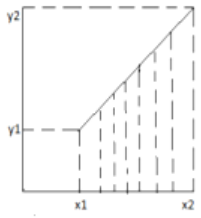
$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad (2)$$

$$b = y_1 - mx_1 \quad (3)$$

$$\Rightarrow y_1 = mx_1 + b$$

For any given x interval Δx along a line, we can compute the corresponding y interval Δy from the equ-2 $m = \frac{\Delta y}{\Delta x} \Rightarrow \Delta y = m\Delta x$

- Similarly we can obtained the x interval Δx corresponding to a specified Δy as $\Delta x = \frac{\Delta y}{m}$
- For a line with slope magnitude |m|<1, Δx can be set proportional to a small horizontal deflection voltage & the corresponding vertical deflection is then set proportional to Δy as calculate from the equation $\Delta y = m\,\Delta x$.
- For a line 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 Δxcalculate from the equation $\Delta x = \frac{\Delta y}{m}$



  - For a line with m=1,then $\Delta x = \Delta y$and vertical & horizontal deflection voltages are equal.
  - In each case a smooth line with slope m is generated between specified end point.

3.2DDA Algorithm(Digital Differential Analyzer)

- DDA is a scan-conversion line algorithm base on either Δx or Δy.
- We sample the line at unit interval in one coordinate & determined the corresponding integer value nearest the line path for the other coordinate.
- Consider four cases for the DDA line drawing.

Case-1

Consider a line with +veslope,If the slope is less than or equal to 1.

If m<1, then x is incrementing faster.

The value of x=1 increment every time, compute& round they value.

As $\Delta x = 1$

Then $\Delta y = m$

$\Rightarrow \quad y_{k+1} - y_k = m$

$\Rightarrow \quad y_{k+1} = y_k + m$

Where k takes integer value starting from 1,for the first point b increases by one until the final end point is reached.

From the above equation we will get

$y_{k+1} = m + y_k$,Then$x = x_0, y = y_0$

Illuminate the pixel (x,round (y))

$x = x_0 + 1$

$y = y_0 + 1 \times m$ Then illuminate Pixel(x,round (y))

Until it reaches at the end$x == x_1$

Case-2

In this case Y increment faster in m>1

The step is y=1 increment, compute& round the value of x

Then $\Delta y = 1, \Delta x = \frac{1}{m}$

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

If$x = x_0$&$y = y_0$

Then illuminate the pixel(round(x), y))

$y = y_0 + 1, x = x_0 + \frac{1}{m}$.

Illuminate the pixel(round(x), y)

Continue until $y == y_1$

- If the processing is reversed, then we havefollowing two cases .

Case -1

If m<1, then x is incrementing faster

The value of x=1 decrement every time, compute & round the y value.$\Delta x = -1$

Then $\Delta y = -m$

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

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

Where k takes integer value starting from 1, for the first point b increases by one until the final end point is reached.

From the above equation we will get$y_{k+1} = -m + y_k$

Then $x = x_1 , y = y_1$

Illuminate the pixel (x,round(y))

$x = x_1 - 1$

$y = y_1 - 1 \times m$ Then illuminate Pixel(x,round (y))

Until it reaches at the end$x == x_0$

Case-2

If m>1then $\Delta y = -1, \Delta x = -\frac{1}{m}$

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

$x = x_1 , y = y_1$

Illuminate the pixel (round (x), y)

$y = y_1 - 1 , x = x_1 - \frac{1}{m}$

Illuminate the pixel (round (x), y)

Until you reaches at the endpoint $y == y_0$

Advantages

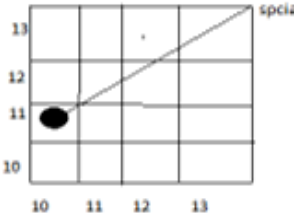
- DDA algorithm is a faster method for calculating pixel position then the equationof apixel position. Y = mx + b

Disadvantages

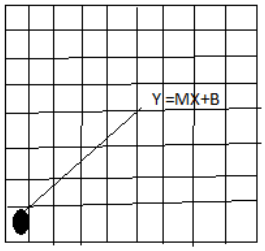
- Accumulation of round of error is successive addition of the floating point increments is used to find the pixel position but it take lot of time to compute the pixel position.

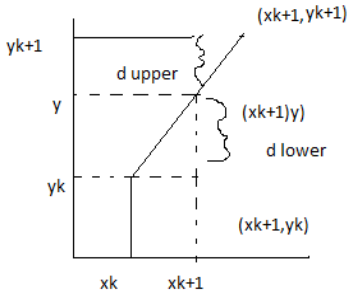
3.3 Bresenham'sLine Drawing Algorithm

- An accurate & efficient raster line– generating algorithm developed by Bresenham.
- Scan converts lines using onlyincrementalinteger calculation that can be adapted to display circle & other curves.
- The vertical axis show scans line position & horizontally axis Identify pixel columns.



- We need to decide which of two possible pixel position is closer to the line path at each sample step.
- W e first consider this scan conversion process for lines with +ve slope <1.
- Pixel position along line path are then determined by sampling x interval.
- Starting from left end point (x_0 ,y_0) of a given line.
- We step to each successivecolumn(x position) &plot the pixel whose scan-line y value is closest to the line path.
- Consider the following diagram.





- From the above diagram figure-2 the pixel at (x_k ,y_k) is the starting point of the line.
- We next do decide whichpixel to plot in columnx_{k+1}.
- Our choices are the pixel at position($x_k + 1$,y_k) and ($x_k + 1$,$y_k + 1$)
- At sapling position$x_k + 1$, we label vertical pixel separation from the mathematical line path as $d_1$and d_2.
- The y –coordinate on the mathematical line at pixel column position $\mathrm{x_k} + 1$ is calculated as $y = m(\mathrm{x_k} + 1)$ +b
- Then $\mathrm{d_1} = \mathrm{y} - \mathrm{y_k}$

$$\Rightarrow m(\mathrm{x_k} + 1) + \mathrm{b} - \mathrm{y_k}$$

And $\mathrm{d_2} = (\mathrm{y_k} + 1) - \mathrm{y}$

$$= \mathrm{y_k} + 1 - \mathrm{m}(\mathrm{x_k} + 1) - \mathrm{b}$$

The deference between thesetwo separations is

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

$$\Rightarrow 2m(\mathrm{x_{k+1}}) + 2\mathrm{b} - \mathrm{y_k} - \mathrm{y_k} - 1$$

$$\Rightarrow 2m(\mathrm{x_{k+1}}) - 2\mathrm{y_k} + 2\mathrm{b} - 1 \qquad (1)$$

Suppose we use decision parameter p_k ,k^{th} step is the line algorithm by rearranging the above equation, so that it in values integer calculation.

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

Where Δx ,Δy vertical & horizontal separation of the end point position then

$$\Rightarrow d_1 - d_2 = 2\frac{\Delta y}{\Delta x}(x_{k+1}) - 2y_k + 2b - 1$$

We know that $x_{k+1} = (x_k + 1)$

$\Rightarrow p_k = \Delta x(d_1 - d_2) = 2\Delta y(x_k + 1) - 2\Delta x y_k - \Delta x(2b - 1)$

$\Rightarrow p_k = 2\Delta y x_k - 2\Delta y - 2\Delta x y_k - \Delta x(2b - 1)$

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

The sign of p_k is the same an the sign of d_1 -d_2 since Δx> 0. C is an constant and c=2 Δy + Δx (2b -1) , which is independent pixel at y_k is closer to the line path then pixel at y_{k+1} (d_1<d_2) the decision parameter in -ve. In this case we plot lower pixel otherwise we plot upper pixel.

Coordinate changes along the line occur in unit steps either the x or y direction.

Therefore we can obtain the value of successive parameter using increment integer calculation at step $k + 1$ the decision parameter is evaluated at

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

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

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

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

The y_{k+1} - y_k is either 0 or 1 depending on the sign of parameter p_k

First parameter p_0 is evaluated from the equ-2,At starting point (x_0 ,y_0)& m evaluated as $\frac{\Delta y}{\Delta x}$,then $p_0 = 2\Delta y - \Delta x$

Algorithm

Step 1 : Input the two line end points &store left end point in (x_0 ,y_0)

Step 2 : load (x_0 ,y_0) into the from buffer plot the first point.

Step 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$p_0 = 2\Delta y - \Delta x$

Step 4 : At each x_k along the line starting at k=0, perform the following last.

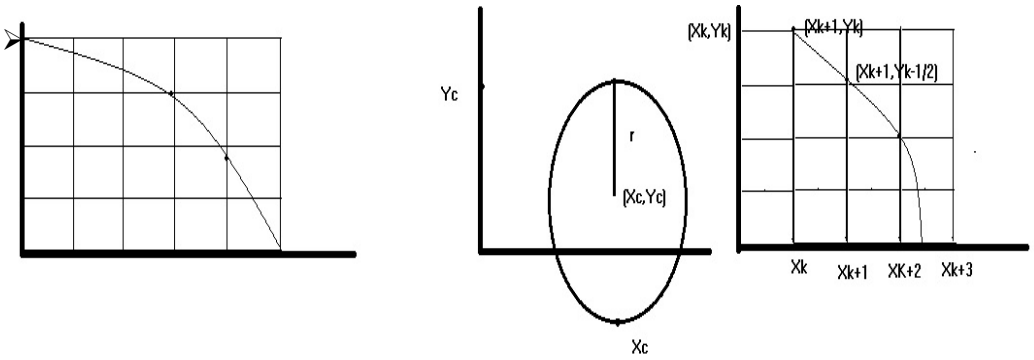
If p_k <0 the next point to plot is $(x_k + 1, y_k)$&$p_{k+1} = p_k + 2\Delta y$

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

p_{k+1} = p_k +2 Δy - 2 Δx

3.4 Midpoint Circle Algorithm

- A circle is defined as a set of points that are all a given distance r from a canter Position (x_c, y_c) then we express the equation as $(x-x_c)^2+(y-y_c)^2 - r^2$.
- If f(X,Y)=0 then the point lies on the circle boundary.
- If f(X,Y)<0 then the point lies inside of in the circle boundary.
- If f(X,Y) > 0 then the point lies on outside circle boundary .
- In mid point circle algorithm the decision parameter of the K^{th} step in the circle function evaluated using co-ordinate of the midpoint of the two pixel.
- Centers which are the next possible pixel position to be plotted.
- Let assume that we are giving unit increments to x in the plotting process & determining the y position using this algorithm.



- Assuming we have justplotted the pixel at (x_k, y_k).
- We next need to determine whether the pixel at the position $(x_k + 1, y_k)$ or the one at position$(x_k + 1, y_k - 1)$ is closer to the circle.
- Our decision parameter$p_k$at the Kth step is the circle function evaluated at the midpointbetween these two pixels.
- Let us consider the pixel (x_k+1, y_k-1/2) ,Then

$$p_k = f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right) \Rightarrow (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2 \qquad (1)$$

If the $p_k < 0$, this midpoint is inside the circle and the pixel on the scan line y_k is closer to the circle boundary .Otherwise, the midpoint is outside or on the circle boundary, and we select the pixel on the scan line $y_k - 1$

- Successive decision parameters areobtained using incremental calculation.
- Avoiding a lot of computation at each step we obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position

$$x_{k+1} + 1 = x_k + 2$$

$$p_{k+1} = f_{circle}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

$$p_{k+1} = [(x_k + 1) + 1)]^2 + (y_{k+1} - \frac{1}{2})^2 - r^2$$

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

$$p_{k+1} - p_k = 2(x_k + 1) + ({y_{k+1}}^2 - {y_k}^2 - (y_{k+1} - y_k) + 1 \qquad (3)$$

$$p_{k+1} = p_k + 2(x_k + 1) + ({y_{k+1}}^2 - {y_k}^2 - (y_{k+1} - y_k) + 1 \qquad (4)$$

- The value of y_{k+1} is y_k or y_k+1

  Put y_k in place of y_{k+1}

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

$$\Rightarrow p_k + 2(x_k + 1) + 1$$

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

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

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

$$p_{k+1} = p_k + 2(x_k + 1) - 2y_k - 2$$

The initial decision parameter is obtain by evaluating the circle function at the start position (x_0, y_0) = (0,r) then we will get another point (1, r-½)

$$p_0 = 1^2 + (r - \tfrac{1}{2})^2 - r^2$$

$$p_0 = 1^2 + r^2 - r + \tfrac{1}{4} - r^2$$

$$p_0 = \frac{5}{4} - r$$

Algorithm

Step1: Input radius 'r' & circle canter (x_c, y_c) and obtain the first point on the circumference of a circle cantered on the origin as (x_0, y_0) = (0, r).

Step2: Calculate the initial value of the decision parameter as $p_0 = \frac{5}{4} - r$.

Step3: At each x_k position, starting at k=0, Perform the following test: If $p_k < 0$, the next point along the circle centred on (0,0) is $(x_k + 1, y_k)$ &

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$&

$p_{k+1} = p_k + 2x_{k+1+1} - 2y_{k+1}$

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

Step4:Determine symmetry points in the other seven octants.

Step5: Move each calculated pixel position (x,y) on the circular path centered on (x_c,y_c)& plot the coordinate values; x = x+x_c, y= y+y_c.

Step6: Repeat step3 through 5unit x>=y.

3.5 Boundary Fill Algorithm

- There are two types of filling area i.e 4-connected region and 8-connected region
- 4-connected region: From a given pixel, the region that you can get to by a series of 4 way moves (N, S, E and W).
- 8-connected region: From a given pixel, the region that you can get to by a series of 8 way moves (N, S, E, W, NE, NW, SE, and SW).
- Start at a point inside a region
- Paint the interior outward to the edge
- The edge must be specified in a single color
- Fill the 4-connected or 8-connected region
    - 4-connected fill is faster, but can have problems.

Algorithm

```
void BoundaryFill4(int x, int y, int newcolor, int edgecolor)
{
  int current;
  current = ReadPixel(x, y);
  if(current != edgecolor && current != newcolor)
  {
     BoundaryFill4(x+1, y, newcolor, edgecolor);
     BoundaryFill4(x-1, y, newcolor, edgecolor);
     BoundaryFill4(x, y+1, newcolor, edgecolor);
```

```
        BoundaryFill4(x, y-1, newcolor, edgecolor);
    }}
```

Flood Fill Algorithm

- Used when an area defined with multiple color boundaries
- Start at a point inside a region
- Replace a specified interior color (old color) with fill color
- Fill the 4-connected or 8-connected region until all interior points being replaced

Algorithm

```
void flood_fill4(intx,inty,intfill_color,intold_color)
{
int current;
    current=getpixel (x,y);
    if (current==old_color)
    {
putpixel (x,y,fill_color);
        flood_fill4(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);
        flood_fill4(x+1,y, fill_color, old_color);
}}
```