Chapter – 2

Scan Conversion

2.1 Output Primitives

The basic geometric structures such as points, straight line segments, circles, conical sections, quadric surfaces, spline curves and surfaces, polygon color areas, which are used to describe a scène, are called output primitives.

The graphic programming packages describe the scene in terms of output primitives and to group sets of output primitives into more complex structures.

2.2 Line-Drawing Algorithm

The slope-intercept equation of a straight line is:

$y = mx + b$.............(1)

where

m = slope of line

b = y-intercept

For any two given points (x_1, y_1) and (x_2, y_2),

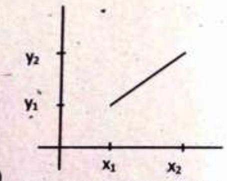


Fig. 2.1: Line path between two endpoint positions

$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{dy}{dx}$ ....(2)

$b = y_1 - m x_1$.................(3)

At any point (x_k, y_k)

$y_k = mx_k + b$.................(4)

At point (x_{k+1}, y_{k+1})

$y_{k+1} = mx_{k+1} + b$..............(5)

Subtracting equation (4) from (5)

$y_{k+1} - y_k = m(x_{k+1} - x_k)$

$(y_{k+1} - y_k)$ is increment in y as correspondingly increment in x.

For given x interval Δx along a line, we can compute the corresponding y interval Δy from the following equation.

$\Delta y = m\,\Delta x$............(6)

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

$\Delta x = \frac{\Delta y}{m}$ ....(7)

Case I: For lines with slope magnitudes $|m|<1$, Δx can be set proportional to a unit horizontal deflection voltage and the corresponding vertical deflection is then set proportional to Δy as calculated from equation 6.

Case II: For lines whose slopes have magnitude $|m|>1$, Δy can be set proportional to a unit vertical deflection voltage and the corresponding horizontal deflection voltage set proportional to Δx, calculated from equation 7.

Case III: For lines with $m=1$, $\Delta x=\Delta y$ and the horizontal and vertical deflection voltages are equal.

On raster systems, we must sample a line at discrete positions and determine the nearest pixel to the line at each sampled position.

2.2.1 Digital Differential Analyzer Algorithm

Digital differential analyzer (DDA) algorithm is scan conversion line algorithm based on calculating either Δx or Δy using the relation.

$\Delta y = m\Delta x$......... (i)

$\Delta x = \frac{\Delta y}{m}$.......... (ii)

We sample the line at unit interval in one co-ordinate and determine corresponding integer values nearest the path for the other co-ordinate.

Different cases:

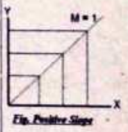
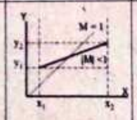
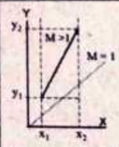
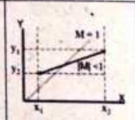
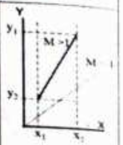
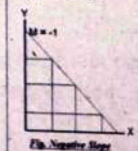
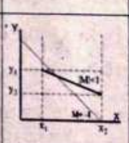
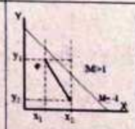
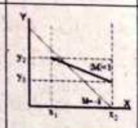
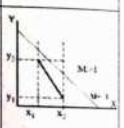
Case 1: Line with positive slope and magnitude less than or equal to 1 ($|m| \leq 1$)

Case 2: Line with positive slope and magnitude more than 1 (|m|>1)

Case 3: Line with negative slope and magnitude less than or equal to 1 ($|m| \leq 1$)

Case 4: Line with negative slope and magnitude more than 1 (|m|>1)

The possible combinations can be shown as follows:

	Moving Left to Right		Moving Right to Left	
	Slope (m) < 1	Slope (m) > 1	Slope (m) < 1	Slope (m) > 1
Positive Slope  Fig. Positive Slope	 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k + 1$	 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k - 1$
Negative Slope  Fig. Negative Slope	 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k - 1$	 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k + 1$

Line with a positive slope

Case I: If slope (m) ≤ 1, then sample at unit x intervals ($\Delta x = 1$) and compute each successive y value because the increment in x is more than increment in y.

So, set $\Delta x = 1$.

Now, from equation (1), we get

$\Delta y = m$

i.e., $x_{k+1} = x_k + \Delta x = x_k + 1$...............(3)

$y_{k+1} = y_k + \Delta y = y_k + m$..................(4)

where subscript k takes integer values starting from 1, for the first point, and increases by 1 unit until the final end point is reached. Since m can be any real number between 0 to 1, the calculated y values must be rounded to the nearest integer.

Case II: If m > 1, then the increment in x (Δx) is smaller than increment in y (Δy),

So, set $\Delta y = 1$.

Then,

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

That is,

$x_{k+1} = x_k + \frac{1}{m}$..........(5)

$y_{k+1} = y_k + 1$...........(6)

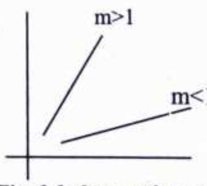


Fig. 2.2: Line with positive slope |m|<1 and |m|>1

Here, in the both case $m \leq 1$ and m>1, consider the algorithm (i.e., equations 3, 4, 5, 6) based on the assumption that lines are to be processed from the left end point to the right end point.

If this process is reversed (that is, lines are to be processed from **right to left**), relation required to change in both case.

Case I: If $m \leq 1$, $\Delta x = -1$, so, $\Delta y = -m$

i.e., $x_{k+1} = x_k - 1$

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

Case II: If m > 1, $\Delta y = -1$ and $\Delta x = -\frac{1}{m}$

i.e., $x_{k+1} = x_k - \frac{1}{m}$ ....(7)

$y_{k+1} = y_k - 1$ ....(8)

Line with negative slope

Case I: If $|m| \leq 1$, then assume start end point is the left.

$\Delta x = 1$ and $\Delta y = m$ (m is negative)

That is,

$x_{k+1} = x_k + 1$..............(9)

$y_{k+1} = y_k + m$.............(10)

If algorithm is required to proceed right to left, then set on

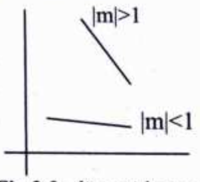


Fig 2.3: line with negative slope |m|<1 and |m|>1

$\Delta x = -1$ and $\Delta y = -m$

i.e., $x_{k+1} = x_k - 1$..............(11)

$y_{k+1} = y_k - m$.............(12)

Case II: If $|m| > 1$, then assume start end is at left and set $\Delta y = -1$

and $\Delta x = -\frac{1}{m}$

i.e., $x_{k+1} = x_k - \frac{1}{m}$..............(13)

$y_{k+1} = y_k - 1$..................(14)

If the algorithm is required to proceed **right to left**, then set

$\Delta y = 1$ and $\Delta x = \frac{1}{m}$.

i.e., $x_{k+1} = x_k + \frac{1}{m}$ ....(15)

$y_{k+1} = y_k + 1$..................(16)

DDA Algorithm

```
Step 1:  Start Algorithm.
Step 2:  Declare $x_1$, $y_1$, $x_2$, $y_2$, dx, step as integer variable and x, y,
         $x_{inc}$, $y_{inc}$ as floating point.
Step 3:  Enter value of $x_1$, $y_1$, $x_2$, $y_2$.
Step 4:  Calculate $dx = x_2 - x_1$.
Step 5:  Calculate $dy = y_2 - y_1$.
Step 6:  If absolute (dx) > absolute (dy)
         Then step = absolute (dx)
         Else step = absolute (dy)
Step 7:  $x_{inc} = \frac{dx}{step}$
         $y_{inc} = \frac{dy}{step}$
         assign $x = x_1$
         assign $y = y_1$
Step 8:  Set pixel (x, y)
Step 9:  $x = x + x_{inc}$
         $y = y + y_{inc}$
         Set pixels (Round (x), Round (y))
Step 10: Repeat step 9 until $x = x_2$
Step 11: End Algorithm
```

Advantages:

- It is faster method than direct line drawing equation y = mx+c for calculating pixel position as it eliminates multiplication.
- It avoids multiplication operation. It is simple to understand, and it doesn't require special knowledge to implement.

Disadvantages:

- The floating point addition is still needed in determining each successive point which is time consuming. The value of slope 'm' is usually stored in floating point number. So, there could be round off error.
- The line will move away from the true line path, especially when it is long due to successive round of error.
- Accumulation of round off error in successive additions of floating point increment.

Code in C to draw a line using DDA Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
     int gd=DETECT,gm;
     int x1,y1,x2,y2,stepsize,dx,dy,i;
     float x,y, xinc,yinc;
     initgraph(&gd,&gm,"c:\\tc\\bgi");
     printf("Digital Differntial Line Drawing Algorithm\n");
```

```
    printf("put the values of x1 and y1\n");
    scanf("%d %d",&x1,&y1);
    printf("put the values of x2 and y2\n");
    scanf("%d %d",&x2,&y2);
    dx = x2-x1;
    dy = y2-y1;
    x = x1;
    y = y1;
    if (abs(dy)>abs(dx))
    {
    stepsize=abs(dy);
    }
    else
    {
    stepsize=abs(dx);
    }
    xinc=dx/(float)stepsize;
    yinc=dy/(float)stepsize;
    putpixel(x,y,RED);
    for(i=0;i<stepsize;i++)
    {
     x=x+xinc;
     y=y+yinc;
     putpixel((int)(x+0.5),(int)(y+0.5),RED);
     }
    getch();
    closegraph();
}
```

Code in C to draw a checker box using DDA Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
```

```
#include<dos.h>
#include<math.h>
void dda (int, int, int, int);
void main()
{
      int gd = DETECT, gm;
      initgraph (&gd, &gm, "c:\\tc\\bgi");
      dda (100,100,200,100);
      dda (200,100,200,200);
      dda (200,200,100,200);
      dda (100,200,100,100);
      dda (100,100,200,200);
      dda (100,200,200,100);
      dda (100,125,200,125);
      dda (100,150,200,150);
      dda (100,175,200,175);
      dda (175,100,175,200);
      dda (175,100,150,200);
      dda (125,100,125,200);
      dda (100,150,150,100);
      dda (150,100,200,150);
      dda (200,150,150,200);
      dda (150,200,100,150);
       getch();
       closegraph();
       }
   void dda (int x1, int y1, int x2, int y2)
    {
     int i, stepsize, dx, dy;
     float x, y, xinc, yinc;
     dx = x2-x1;
dy = y2-y1;
x = x1;
y = y1;
if (abs(dy) > abs(dx))
{
stepsize = abs(dy);
}
else
{
stepsize = abs(dx);
}
xinc =  dx / (float) stepsize;
yinc=dy/ (float) stepsize;
putpixel (x, y, RED);
for (i=0; i < stepsize; i++)
{
  x = x + xinc;
  y = y + yinc;
  putpixel ((int) (x+0.5), (int) (y+0.5), RED);
  delay (10);
}
}
```

2.2.2 Bresenham's Line Algorithm

Bresenham's line algorithm is an accurate and efficient line drawing algorithm. It uses only integer arithmetic to find the next position to be plotted. It avoids incremental error. The major concept of Bresenham's algorithm is to determine the nearest pixel position. Great advantage of this algorithm is that it can be used to display circles and other curves.

In Bresenham's algorithm, we calculate the decision parameter which decides which pixel to select and which function is used for next decision parameter.

For positive slope and slope $|m| < 1$

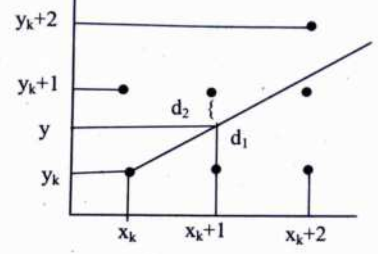


Figure 2.4: *Line with m<1*

- Pixel positions are determined by sampling at unit x intervals.
- Starting from left end position (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 closed to the line path.

Assuming the pixel at (x_k, y_k) to be displayed is determined, we next to decide which pixel 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 separations from the mathematical line path d_1 and d_2.

The y-co-ordinate on the mathematical at pixel column position $x_k + 1$ is calculation.

As,

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

Then,

$d_1 = y - y_k$

$d_1 = m (x_k + 1) + b - y_k$ .... (2)

And,

$d_2 = (y_k + 1) - y$

$d_2 = y_k + 1 - m (x_k+1) - b$ .... (3)

Now,

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

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

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

Defining decision parameter $p_k = \Delta x(d_1 - d_2)$

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

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

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

$= 2\Delta y x_k - 2\Delta x y_k + c$............(4)

Where, $c = 2\Delta y + \Delta x(2b - 1)$

Here, sign of p_k is same as the sign of $d_1 - d_2$ since $\Delta x > 0$.

Case I:

If $p_k \geq 0$, then $d_2 < d_1$, which implies that $y_k + 1$ is nearer than y_k. So, pixel at $(y_k + 1)$ is better to choose which reduce error than pixel at y_k. This determines next pixel co-ordinate to plot is $(x_k + 1, y_k + 1)$.

Case II:

If $p_k < 0$ then $d_1 < d_2$ which implies pixel at y_k is nearer than pixel at $(y_k + 1)$.So, pixel at y_k is better to choose which reduce error than pixel at $(y_k + 1)$.This determines next pixel co-ordinate to plot is $(x_k + 1, y_k)$

Now, similarly, pixel as $(x_k + 2)$ can be determined whether it is $(x_k + 2, y_k + 1)$ or $(x_k + 2, y_k + 2)$ by looking the sign of deciding parameter p_k+1 assuming pixel as (x_k+1) is known.

$p_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c$ where c is same as in p_k

Now,

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

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

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

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

This implies that decision parameter for the current column can be determined if the decision parameter of the last column is known.

Here, $(y_{k+1} - y_k)$ could either 0 or 1 which depends on sign of p_k.

If $p_k \geq 0$ (i.e.,$d_2 < d_1$), $y_{k+1} = y_k + 1$ which implies $(y_k + 1 - y_k) = 1$

That is, at $p_k \geq 0$, the pixel to plot is $(x_k + 1, y_k + 1)$ and

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

If $p_k < 0$ (i.e.,$d_1 < d_2$), $y_{k+1} = y_k$ which implies $(y_{k+1} - y_k = 0)$

i.e., at $p_k < 0$, then pixel to be plotted is (x_k+1, y_k)

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

Initial decision parameter (p_0)

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

$p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + \Delta x(2b - 1)$

But $b = y_0 - m x_0 = y_0 - \frac{\Delta y}{\Delta x} x_0$

$= 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x y_0 - 2\Delta y x_0 - \Delta x$

$= 2\Delta y - \Delta x$

BLA Algorithm

```
Step 1. Start
Step 2. Declare variables $x_1, y_1, x_2, y_2, l_x, l_y, \Delta x, \Delta y, p_0, p_k, p_{k+1}$
Step 3. Read values of $x_1, y_1, x_2, y_2$
Step 4. Calculate $\Delta x$ = absolute $(x_2 - x_1)$
        $\Delta y$ = absolute $(y_2 - y_1)$
Step 5. If $(x_2 > x_1)$
        assign $l_x = 1$
        else
        assign $l_x = -1$
Step 6. if $(y_2 > y_1)$
        assign $l_y = 1$
        else
        assign $l_y = -1$
```

```
Step 7. Plot $(x_1, y_1)$
Step 8. if $\Delta x > \Delta y$(i.e., $|m| < 1$)
        compute $p_0 = 2\Delta y - \Delta x$
        starting at $k = 0$ to $\Delta x$ times, repeat
        if $(p_k < 0)$
        $x_{k+1} = x_k + 1$
        $y_{k+1} = y_k$
        $p_{k+1} = p_k + 2\Delta y$
        else
        $x_{k+1} = x_k + 1$
        $y_{k+1} = y_k + 1$
        $p_{k+1} = p_k + 2\Delta y - 2\Delta x$
        plot$(x_{k+1}, y_{k+1})$
        else
        calculate $p = 2\Delta x - \Delta y$
        starting at $k = 0$ to $\Delta y$ times, repeat
        if$(p_k < 0)$
        $x_{k+1} = x_k$
        $y_{k+1} = y_k + 1$
        $p_{k+1} = p_k + 2\Delta x$
        else
        $x_{k+1} = x_k + 1$
        $y_{k+1} = y_k + 1$
        $p_{k+1} = p_k + 2\Delta x - 2\Delta y$
        Plot$(x_{k+1}, y_{k+1})$
Step 9. Stop
```

Advantage of BLA over DDA:

- In DDA algorithm, each successive point is computed in floating point. So, it requires more time and more memory space. While in BLA, each successive point is calculated in integer value. So, it requires less time and less memory space.

- In DDA, since the calculated point value is in floating point number, it should be rounded at the end of calculation. But in BLA, round off is not necessary. So, there is no accumulation of rounding error.
- Due to rounding error, the line drawn by DDA algorithm is not accurate, while by BLA algorithm, line is accurate.
- DDA algorithm cannot be used in other application except line drawing, but BLA can be implemented in other application such as circle, ellipse, and other curves.

Comparison between DDA and BLA

Base of comparison	**Digital differential analyzer line drawing algorithm**	**Bresenham's line drawing algorithm**
Arithmetic	DDA algorithm uses floating points	BLA uses integer
Operations	DDA algorithm uses multiplication and division in its operations	BLA uses only subtraction and addition in its operation
Speed	DDA algorithm is slower than BLA because it uses floating points	BLA is faster than DDA because it uses subtraction, addition and integer only
Accuracy and Efficiency	DDA algorithm is not as accurate and efficient as BLA as error and error accumulation occurs on it	BLA is more efficient and much accurate than DDA algorithm
Drawing	DDA algorithm can't draw curves and circles as accurate as by BLA	BLA can draw curves and circles much more accurately
Round off	DDA algorithm round off the co-ordinates to integer that is nearest to the line	BLA does not round off but takes the incremental value in its operation

Base of comparison	Digital differential analyzer line drawing algorithm	Bresenham's line drawing algorithm
Expensive	DDA is expensive as it uses floating point	BLA is cheaper than DDA as it uses only addition and subtraction

Code in C to draw a line using BLA

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<math.h>
void main()
{
 int gd = DETECT, gm, x1, y1, x2, y2, lx, ly, dy, dx, pk, i;
 initgraph (&gd, &gm, "c:\\tc\\bgi");
 printf ("put the values of x1 and y1\n");
  scanf ("%d %d",&x1, &y1);
  printf ("put the values of x2 and y2\n");
  scanf ("%d %d",&x2, &y2);
  dx = abs (x2-x1);
  dy = abs (y2-y1);
  if (x2 > x1)
 {
     lx=1;
     }
 else
     {
     lx=-1;
     }
if (y2 > y1)
     {
      ly=1;
```

```
        }
else
        {
         ly=-1;
        }
putpixel (x1,y1,RED);
if (dx > dy)
        {
        pk=2*dy-dx;
        for(i=0; i<dx; i++)
        {
        if (pk < 0)
        {
        x1 = x1 + lx;
        y1 = y1;
        pk = pk + 2*dy;
        }
else
        {
        x = x1 + lx;
        y1 = y1 + ly;
        pk = pk + 2*dy - 2*dx;
        }
putpixel (x1, y1, RED);
        }
}
else
        {
pk=2*dx-dy;
for(i=0; i<dy;i++)
{
```

```
if(pk<0)
{
x1=x1;
y1=y1+ly;
pk=pk+2*dx;
}
else
{
x1= x1 + lx;
y1= y1 + ly;
pk = pk + 2*dx-2*dy;
}
putpixel (x1,y1,RED);
}
}
getch();
closegraph ();
}
```

2.3 Circle

The equation of circle in Cartesian form is

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

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

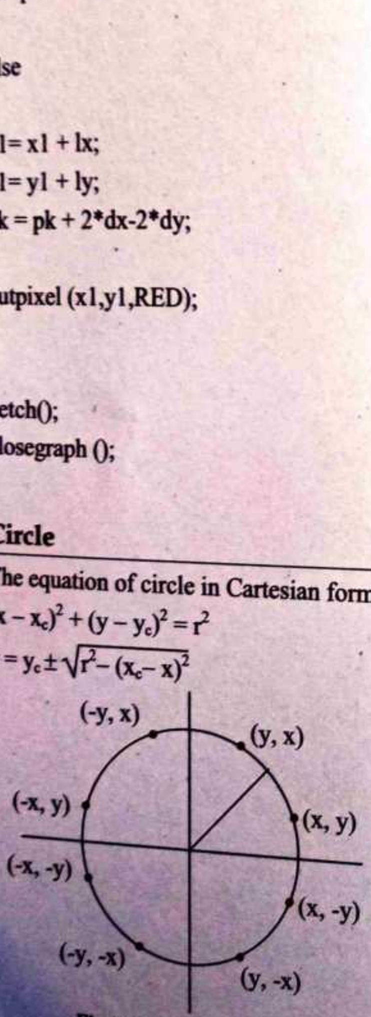


Figure 2.5: Circle with symmetry points

- We sample at unit intervals and determine the closest pixel position to the specified circle at each steps.
- Non uniform spacing of plotted pixel is a problem.
- Interchange the rate to x and y wherever the absolute value of slope of the circle tangent greater than1.
- To solve the computational complexity, we use symmetry of circle i.e., calculate for one octant and use symmetry for others.
- Bresenham's line algorithm for raster displays is adapted for circle generation by setting up decision parameters for finding the closest pixel to the circumference at each sampling step.
- We test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary.

2.3.1 Midpoint Circle Algorithm (Derivation)

The equation of circle is $x^2 + y^2 = r^2$

With center (0, 0) and radius r, let's define a circle function as circle $(x, y) = x^2 + y^2 - r^2$.

The distance of pixel to adjacent pixel is unit.

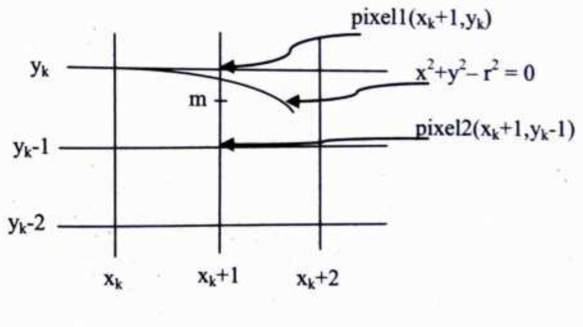


Figure 2.6: Mid-point circle pixels

In figure, length between pixel 1 and pixel 2 is $(y_k - y_{k+1}) = 1$ and half the length $= \frac{1}{2}$.

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

- Assume (x_k, y_k) is plotted, then next point close to the circle is $(x_{k+1,}y_k)$ or (x_k+1, y_k-1)
- Decision parameter is the circle function evaluated at the midpoint between these two points.

$$p_k = f_{circle}(x_k+1, y_k-\frac{1}{2})$$

$$p_k = (x_k+1)^2 + (y_k-\frac{1}{2})^2 - r^2 \quad \text{..........(2)}$$

So, if $p_k<0$, then midpoint is inside the circle and y_k is closer to circle boundary. Else, midpoint is outside the circle. And y_k-1 is closer to the circle boundary

- Successive decision parameters are obtained using incremental calculations i.e., next decision parameter is obtained at

$x_{k+1}+1 = x_k + 1 + 1$ and $y_{k+1} - \frac{1}{2}$

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

$$p_{k+1} = (x_k+1+1)^2 + (y_{k+1}-\frac{1}{2})^2 - r^2 \quad \text{........(3)}$$

Subtracting equation (2) from (3)

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

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

$$(x_k+1)^2 - y_k^2 + 2y_k\times\frac{1}{2} - \frac{1}{4}$$

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

Where y_{k+1} is either y_k or $y_k - 1$ depending on sign of p_k

If $p_k < 0$, then next pixel is at $(x_k + 1, y_k)$

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

If $p_k \geq 0$, then next pixel is at $(x_k + 1, y_k - 1)$

$$p_{k+1} = p_k + 2x_{k+1} + [(y_k - 1)^2 - y_k^2] - (y_k - 1 - y_k) + 1$$

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

$$= p_k + 2x_{k+1} - 2y_k + 1 + 1 + 1$$

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

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

Initial decision parameter

The initial decision parameter is obtained by evaluating the circle function at starting point $(x_0, y_0) = (0,r)$.

$$p_0 = f_{circle}(1, r-\frac{1}{2})$$

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

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

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

If the radius r is specified as an integer, we can simply round to $p_0 = 1 - r$

Mid-point circle algorithm

Step 1. Start

Step 2. Declare variables x_c, y_c, r, x_0, y_0, p_0, p_k, p_{k+1}.

Step 3. Read values of x_c, y_c, r.

Step 4. Initialize the x_0 and y_0 i.e., set the co-ordinates for the first point on the circumference of the circle centered at origin as

$x_0 = 0$

$y_0 = r$

Step 5. Calculate initial value of decision parameter

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

Step 6. At each x_k position, starting from $k = 0$

```
If $p_k < 0$
    $x_{k+1} = x_k+1$
    $y_{k+1} = y_k$
    $p_{k+1} = p_k+2x_{k+1}+1$
else
    $x_{k+1} = x_k+1$
    $y_{k+1} = y_k-1$
    $p_{k+1} = p_k+2x_{k+1}-2y_{k+1}+1$
```

Step 7. Determine the symmetry in other seven octants.

Step 8. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c)

Step 9. Plot the co-ordinates values

$x = x + x_c$

$y = y + y_c$

Step 10. Repeat steps 6 to 9 until $x \geq y$.

Step 11. Stop

2.4 Ellipse

An ellipse is defined as the set of points such that the sum of the distances from two fixed point/positions (foci) is same for all points

Major axis: The straight line segment extending from one side of the ellipse to the other through foci.

Minor axis: The shorter dimension of the ellipse, bisecting the major axis at the halfway position (ellipse center) between the two foci.

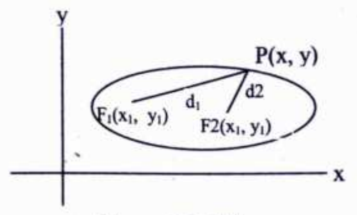


Figure 2.7: *Ellipse*

General equation: $\frac{(x - x_c)^2}{r_x^2} + \frac{(y - y_c)^2}{r_y^2} = 1$

In polar form,

$x = x_c + r_x\cos\theta$

$y = y_c + r_y\sin\theta$

2.4.1 Mid-Point Ellipse Algorithm

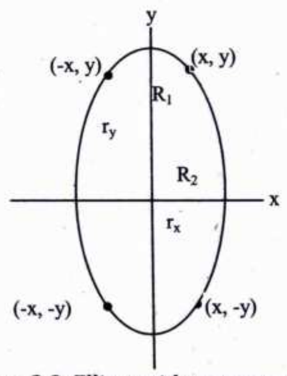


Figure 2.8: *Ellipse with symmetry points*

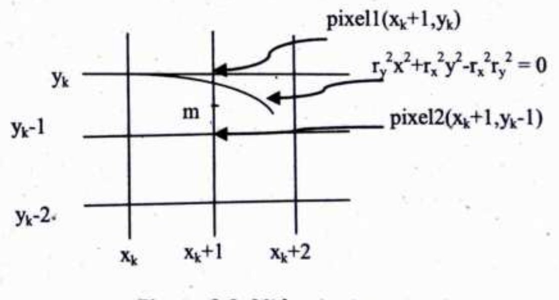


Figure 2.9: *Mid-point in region 1*

- Applied throughout the 1st quadrant in two parts
- Figure shows the division of 1st quadrant according to the slope of an ellipse with $r_x<r_y$
- Process this quadrant by taking unit steps in x-direction where slope of curve in y-magnitude less than 1, and taking unit step in y-direction where slope has magnitude greater than 1. Region 1 and 2 can be processed in different ways.

i) Start at position $(0, r_y)$ and step clock wise along the elliptical path in first quadrants, shifting from unit step in x to unit step in y when slope becomes greater than -1

ii) Alternatively, start at position $(r_x, 0)$ and select points in counter clockwise, shifting from unit step in y to unit step in x when the slope becomes less than-1.

Here, we start at position $(0, r_y)$

-we define an ellipse function with $(x_c, y_c) = (0,0)$

$$f_{ellipse}(x, y) = r_y^2x^2 + r_x^2y^2 - r_x^2r_y^2 \quad \text{........(i)}$$

With properties

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

Thus ellipse function serves as the decision parameter.

- at each sampling position, we select the next pixel along the ellipse path according to the sign of the ellipse function evaluated at the midpoint between the two candidate pixels.
- at each step, we test the value of the slope of the curve,
- slope can be calculated as: $r_y^2x^2 + r_x^2y^2 = r_x^2r_y^2$

Differentiating with respect to x,

$$2r_y^2x + 2r_x^2y\frac{dy}{dx} = 0$$

$$\therefore \frac{dy}{dx} = -\frac{2r_y^2x}{2r_x^2y} \quad \text{........... (ii)}$$

At the boundary between region 1 and region 2, slope = −1.

So,

$$\frac{dy}{dx} = -1 = \frac{-2r_y^2x}{2r_x^2y}$$

or, $2r_y^2x = 2r_x^2y$

We move out of region 1, whenever

$2r_y^2x > 2r_x^2y$

We move out from region 2, whenever

$2r_y^2x < 2r_x^2y$

Assuming (x_k, y_k) has been illuminated (selected) we determine the next position along the ellipse path by evaluating the decision parameter at the midpoint $(x_k + 1, y_k - \frac{1}{2})$.

We have to determine the next point is $(x_k + 1, y_k)$ or $(x_k + 1, y_k - 1)$

We define decision parameter at mid-point as

$$p_{1k} = f_{ellipse}(x_k + 1, y_k - \frac{1}{2})$$

$$p_{1k} = r_y^2(x_k+1)^2 + r_x^2\left(y_k - \frac{1}{2}\right)^2 - r_x^2r_y^2 \text{.............(iii)}$$

If $p_{1k}<0$, the mid-point is inside the ellipse and the pixel on scan line y_k is closer to the ellipse boundary

Otherwise, the mid-point is outside or on the boundary and we select the pixel on scan line y_k-1

At the next sampling position $(x_{k+1}+1 = x_k+2)$, the decision parameter for region 1 is evaluated as

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

$$p_{1k+1} = r_y^2[(x_k + 1) + 1]^2 + r_x^2(y_{k+1} + \frac{1}{2})^2 - r_x^2r_y^2 \text{.......(iv)}$$

Subtracting equation (iii) from (iv),

$$p_{1k+1} - p_{1k} = r_y^2[(x_k + 1 + 1)^2 - (x_k+1)^2] + r_x^2\left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right] - r_x^2r_y^2 + r_x^2r_y^2$$

$$= r_y^2[(x_k + 1)^2 + 2(x_k + 1) + 1 - (x_k+1)^2] + r_x^2[y^2_{k+1} - 2y_{k+1}\times\frac{1}{2}+\frac{1}{4} - y_k^2 + 2y_k\times\frac{1}{2} - \frac{1}{4}]$$

$$= 2r_y^2(x_k + 1) + r_y^2 + r_x^2 + r_x^2\,[(y^2_{k+1} - y_k^2) - (y_{k+1} - y_k)]$$

$$= 2r_y^2(x_k + 1) + r_y^2 + r_x^2[(y^2_{k+1} - y_k^2) - (y_{k+1} - y_k)]$$

Where y_{k+1} is either y_k or y_k-1, depending on the sign of p_{1k}

If $p_{1k} \le 0$ i.e., $y_{k+1} = y_k$, then decision parameter is

$$p_{1k+1} = p_{1k} + 2r_y^2(x_k + 1) + r_y^2$$

If $p_{1k} > 0$ i.e., $y_{k+1} = y_k - 1$, then decision parameter is

$$p_{1k+1} = p_{1k} + 2r_y^2(x_k + 1) + r_y^2 - 2r_x^2 y_{k+1}$$

The initial decision parameter is evaluated at start position $(x_0, y_0) = (0, r_y)$ as

$$p_{10} = f_{ellipse}\left(1,\ r_y - \frac{1}{2}\right)$$

$$= r_y^2 + r_x^2\left(x_y - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$

$$p_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2 \text{ ....(v)}$$

Region 2

We sample at unit steps in the negative y direction and the midpoint is now taken between horizontal pixels at each step. The decision parameter is

$$p_{2k} = f_{ellipse}\left(x_k + \frac{1}{2},\ y_k - 1\right)$$

$$p_{2k} = r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2 \text{ ....(vi)}$$

Put k = 0 for initial decision parameter for region 2

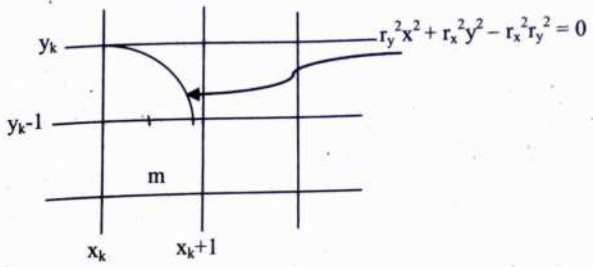


***Figure 2.10:** Midpoint in region 2*

If $p_{2k} > 0$, the midpoint is outside the ellipse boundary and we select the pixel x_k. If $p_{2k} < 0$, the midpoint is inside or on the ellipse boundary and we select pixel position x_k+1.

Now, at next sampling position $y_{k+1} - 1 = y_k - 2$

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

$$p_{2k+1} = r_y^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x^2(y_{k+1} - 1)^2 - r_x^2 r_y^2 \text{ ....(vii)}$$

Subtracting equation (vi) from (vii)

$$p_{2k+1} = p_{2k} + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right] + r_x^2\left[(y_k - 1 - 1)^2 - (y_k - 1)^2\right]$$

$$= p_{2k} + r_x^2\left[(y_k - 1)^2 - 2(y_k - 1) + 1 - (y_k - 1)^2\right] + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right]$$

$$= p_{2k} + r_x^2[-2(y_k - 1) + 1] + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right]$$

If $p_{2k} > 0$, then

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

$$p_{2k+1} = p_{2k} - 2r_x^2 y_{k+1} + r_x^2$$

If $p_{2k} < 0$, then

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

$$p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

Mid-point ellipse algorithm

Step 1. Start

Step 2. Declare variables x_c, y_c, r_x, r_y, x, y, p_0, p_k, p_{k+1}

Step 3. Read Values of x_c, y_c, r_x, r_y.

Step 4. Obtain the first point on an ellipse centered on origin (x, y) by initializing the x and y as

$x = 0$

$y = r_y$

Step 5. Calculate the initial value of the decision parameter in region 1 as

$$p_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

Step 6. For each x_k position in region 1, starting at k = 0, perform the following test.

If $p_{1k}<0$,

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

$y_{k+1} = y_k$

$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2$

else

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

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

$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$

and continue until $2r_y^2 x \geq 2r_x^2 y$

Step 7. Calculate the initial decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as

$$p_{20} = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0-1)^2 - r_x^2 r_y^2$$

Step 8. At each y_k position in region 2, starting at k = 0, perform the following test.

If $p_{2k}>0$,

the next point along the ellipse centered on (0,0) is (x_k, y_k-1) and

$p_{2k+1} = p_{2k} - 2r_x^2 y_{k+1} + r_x^2$

else

the next point along the ellipse is

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

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

$p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$

Use the same incremental calculations for x and y as in region 1.

Step 9. Determine the symmetry points in the other three quadrants.

Step 10. Move each calculated pixel position(x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values.

$x = x+ x_c$

$y = y+ y_c$

Step 11. Repeat the steps for region 2 until y<0.

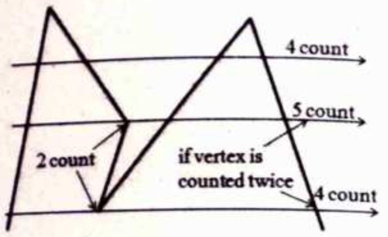
Step 12. Stop.

2.5 Filled Area Primitive

A standard output primitive in general graphics is solid color or patterned polygon area. Other kinds of area primitives are sometimes available, but polygons are easier to process since they have linear boundaries. The main idea behind the 2D or 3D object filling procedure is that it provides us more realism on the object of interest. There are two basic approaches to area filling in raster systems.

- One way to fill an area is to determine the overlap intervals for scan lines that crosses the area.
- Another method for area filling is to start from a given interior position and point outward from this until a specified boundary is met.

2.5.1 SCAN-LINE Polygon Fill Algorithm:

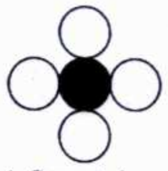


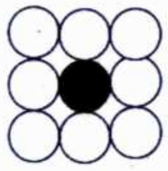
In scan-line polygon fill algorithm, for each scan-line crossing a polygon, it locates the intersection points of the scan line with the polygon edges. These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified color. In this method, the scan line must be even. At two edge connecting point called the vertex, we count the scan line two to handling the problem for scan line passing through vertex, but this consideration also may creates problem for some instant as shown in figure. To handle such problem shown by count-5, we keep the vertex blank and hence the count become 1, so that overall count in that scan line become even as shown in figure

2.5.2 Boundary-fill Algorithm:

In Boundary filling algorithm starts at a point called seed pixel inside a region and paint the interior outward the boundary. If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by until the boundary color is reached. Starting from (x, y), the procedure tests neighboring positions to determine whether they are of boundary color. If not, they are painted with the fill color, and their neighbors are tested. This process continues until all pixel up to the boundary color area have tested. The neighboring pixels from current pixel are proceeded by two method:

- 4-Connected (if they are adjacent horizontally and vertically.)
- 8-Connected (if they adjacent horizontally, vertically and diagonally.)


4- Connected


8- Connected

Algorithm for Boundary fill 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);
        Boundary_fill4(x,y+1, b_color, fill_color);
        }
}
```

Algorithm for Boundary fill 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);
        }
}
```

Recursive boundary-fill algorithm not fills regions correctly if some interior pixels are already displayed in the fill color. Encountering a pixel with the fill color can cause a recursive branch to terminate, leaving other interior pixel unfilled. To avoid this we can first change the color of any interior pixels that are initially set to the fill color before applying the boundary fill procedure.

2.5.3 Flood-fill Algorithm:

Flood_fill Algorithm is applicable when we want to fill an area that is not defined within a single color boundary. If fill area is bounded with different color, we can paint that area by replacing a specified interior color instead of searching of boundary color value. This approach is called flood fill algorithm. We start from a specified interior pixel (x,y) and reassign all pixel values that are currently set to a given interior color with desired fill_color. Using either 4-connected or 8-connected region recursively starting from input position, The algorithm fills the area by desired color.

Algorithm:

```
void flood_fill4(int x,int y,int fill_color,int old_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);
        }
}
```

SOLVED NUMERICAL AND DERIVATIONS

1. ***Consider a line from (3,7) to (8,3). Using simple DDA algorithm, rasterize this line.***

Solution:

Here,

Starting point $(x_1, y_1) = (3,7)$

Ending point $(x_2, y_2) = (8,3)$

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

$$= \frac{3-7}{8-3}$$

$$= -0.8$$

Since $|m|<1$, from DDA Algorithm we get,

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

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

k	x_k	y_k	x_{plot}	y_{plot}	(x_{k+1}, y_{k+1})
1	3	7	3	7	(3,7)
2	4	6.2	4	6	(4,6)
3	5	5.4	5	5	(5,5)
4	6	4.6	6	5	(6,5)
5	7	3.8	7	4	(7,4)
6	8	3	8	3	(8,3)

2. ***Use Bresenham's algorithm to scan convert a straight line connecting the end points (20, 10) and (30, 18).***

Solution:

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

$\Delta x = |x_2 - x_1| = 10$

$\Delta y = |y_2 - y_1| = 8$

$p_0 = 2\Delta y - \Delta x$ (since $\Delta x > \Delta y$ i.e., $m < 1$)

$= 6$

For $\Delta x>\Delta y$,

if $p_k< 0$,

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

$y_{k+1} = y_k$

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

if $p_k> 0$,

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

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

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

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)

3. ***Determine the raster location along the circle octant in the first quadrant for a circle with radius 10.***

Solution:

$r = 10$

Initial decision parameter,

$p_0 = 1-r = 1-10 = -9$

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

$2x_0 = 0,\ 2y_0 = 20$

k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	–9	(1, 10)	2	20
1	–6	(2, 10)	4	20
2	–1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	–3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7,7)	14	14

4. ***Using midpoint circle algorithm, calculate the co-ordinate on the first quadrant of a circle having radius 6 and centre at (20, 10).***

Solution:

$r = 6$

$(x_2, y_2) = (20, 10)$

$p_0 = 1-r = 1-6 = -5$

$(x_0, y_0) = (0,6)$

If $p_k<0$

$(x_{k+1}, y_{k+1}) = (x_{k+1}, y_k)$

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

Else

$(x_{k+1}, y_{k+1}) = (x_k+1, y_k-1)$

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

For first octant,

k	p_k	x_{k+1}, y_{k+1}	$2x_{k+1}$	$2y_{k+1}$	Pixel to plot $x_{k+1}+x_c, y_{k+1}+y_k$
		(0,6)	0	12	(20,16)
0	–5	(1,6)	2	12	(21,16)
1	–2	(2,6)	4	12	(22,16)
2	0	(3,5)	6	10	(23, 15)

k	p_k	x_{k+1}, y_{k+1}	$2x_{k+1}$	$2y_{k+1}$	**Pixel to plot** $x_{k+1}+x_c, y_{k+1}+y_k$
3	–3	(4,5)	8	10	(24, 15)
4	9	(5,4)	10	9	x>y, not necessary to plot pixels

For second octant, using symmetry (x,y)→(y,x)

(0,6)→(6,0), (1,6) →(6,1),(2,6) →(6,2),(3,5) →(5,3) and (4,5) → (5,4).

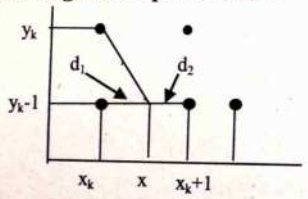
At last we need to shift the co-ordinates in the center (20, 10) so,

(6, 0) → (26, 10), (6, 1)→(26, 11), (6, 2)→(26, 12), (5, 3) → (25, 13) and (5, 4)→(25, 14)

5. ***Derive the Bresenham's decision parameter to draw a line moving from left to right and having negative slope. State the condition to identify you are in the second region of the ellipse using midpoint algorithm.*** *[2072 Kartik]*

Solution:

i) **For line with negative slope and |m| >1**



- Pixel positions are determined by sampling at unity intervals.
- Starting from left end position (x_0, y_0) of a given line, we step to each successive row (y position) and plot the pixel whose x value is closest to the line path.
- Assuming the pixel at (x_k, y_k) to be displayed is determined, we next need to decide which pixel to plot in row y_k–1.

- The candidate pixels are at position $(x_k, y_k–1)$ and $(x_k+1, y_k–1)$
- At sampling position y_k–1,we label horizontal pixel separation from the mathematical line path d_1 and d_2.
- The x-coordinate on the mathematical line at pixel row position y_k–1 is calculated as

$y_k - 1 = mx + b$

or, $x = \frac{y_k - 1 - b}{m}$

where $m = -\frac{\Delta y}{\Delta x}$ since m is negative.

Then,

$d_1 = x - x_k$..................(ii)

$d_2 = x_k + 1 - x$.............(iii)

And,

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

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

Now, defining decision parameter $p_k = \Delta y(d_1 - d_2)$

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

$\therefore p_k = -2\Delta x y_k - 2\Delta y x_k + c$..............(iv)

Where $C = 2\Delta x + 2\Delta x b - \Delta y$ (All are constant)

Here, the sign of p_k is same as the sign of $d_1 - d_2$, as $\Delta y > 0$

Case I:

If $p_k \geq 0$, then $d_1 > d_2$ which implies that x_k+1 is nearer than x_k, so the next pixel to choose is (x_k+1, y_k-1)

Case II:

If $p_k < 0$, then $d_1 < d_2$ which implies that x_k is nearer than x_k+1, so the next pixel to select is (x_k, y_k-1)

Now,

Similarly, pixel at (y_k-2) can be determined whether it is (x_k+1, y_k-2) or (x_k+2, y_k-2) by looking the sign of deciding parameter p_{k+1} assuming pixel at (y_k-1) is known.

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

where C is same as in p_k.

Now,

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

This implies that decision parameter for the current row can be determined if the decision parameter of the last row is known.

Here (x_k+1-x_k) could either 0 or 1 which depends on sign of p_k.

If $p_k \geq 0$ (i.e., $d_1 > d_2$), $x_{k+1} = x_k+1$

Then, $p_{k+1} = p_k + 2\Delta x - 2\Delta y$

If $p_k < 0$, then $x_{k+1} = x_k$

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

Initial decision parameter

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

Now,

$p_0 = -2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x b - 2\Delta x - \Delta y$

$$= -2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x\left(y_0 + \frac{\Delta y}{\Delta x}x_0\right) + 2\Delta x - \Delta y$$

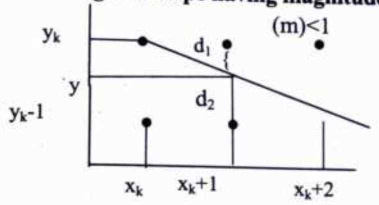
$$\left[\because y = mx + b, b = y - mx, m = -\frac{\Delta y}{\Delta x}\right]$$

$$= -2\Delta x y_0 - 2\Delta y x_0 + \Delta x\left(\frac{2\Delta x y_0 + 2\Delta y x_0}{\Delta x}\right) + 2\Delta x - \Delta y$$

$$= -2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta x - \Delta y$$

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

ii) For line with negative slope having magnitude less than 1.



$d_1 = y_k - y$

$d_2 = y - (y_k - 1) = y - y_k + 1$

$d_1 - d_2 = y_k - y - y + y_k - 1 = 2y_k - 2y - 1$

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

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

$m = -\dfrac{\Delta y}{\Delta x}$ ($\because$ m is negative)

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

$$= \frac{2yk * \Delta x + 2\Delta y x k + 2\Delta y - \Delta x(2b) - \Delta x}{\Delta x}$$

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

$p_k = 2\Delta x y_k + 2\Delta y x_k + c$

where $c = 2\Delta y - \Delta x(2b) - \Delta x$

For next decision parameter,

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

Now,

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

if $pk < 0$, $d1<d2$

so,

$(x_{k+1}, y_{k+1}) = (x_k+1, y_k)$

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

if $pk \geq 0$, $d1 \geq d2$

so, $(x_{k+1}, y_{k+1}) = (x_k+1, y_k-1)$

so, $p_{k+1} = p_k + 2\Delta y - 2\Delta x$

Now, for initial decision parameter,

$p_k = p_0,\ x_k = x_0,\ y_k = y_0$

$$p_0 = 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta y - \Delta x(2\left(y_0 - \left(-\frac{\Delta y}{\Delta x}\right)x_0\right) + 1)$$

$$= 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta y - \Delta x\left(\frac{2\Delta x y_{0+2\Delta y x_{0+\Delta x}}}{\Delta x}\right)$$

$$= 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta y - 2\Delta x y_0 - 2\Delta y x_0 - \Delta x$$

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

6. ***Determine the pixel positions of following curve in first quadrant using mid-point algorithm.***

$$\frac{x^2}{64} + \frac{y^2}{36} = 1$$ *[2076 Ashwin Back]*

Solution:

Here, $r_x^2 = 64$, $r_y^2 = 36$

Here, Half of major axis $(r_x) = 8$

Half off minor axis $(r_y) = 6$

First point of the ellipse starting with $(0, r_y)$ is

$x_0 = 0$

$y_0 = r_y$

So, $x_0 = 0$, $y_0 = 6$

Now,

Initial decision parameter in region 1 is

$$P_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

$$= 6^2 - 8^2 \times 6 + \frac{1}{4} \times 8^2 = -332$$

Now, for each x_k position in region 1, starting at $k = 0$, we perform the following test

If $P_{1k} < 0$,

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

$y_{k+1} = y_k$

$P_{1k+1} = P_{1k} + 2r_y^2 x_{k+1} + r_y^2$

Else

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

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

$P_{1k+1} = P_{1k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$

and will continue until $2r_y^2 x \geq 2r_x^2 y$

Similarly, we calculate the initial decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as

$$P_{20} = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

Now, at each y_k position in regions, starting at $k = 0$, we perform following test

If $P_{2k} > 0$,

$x_{k+1} = x_k$

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

$P_{2k+1} = P_{2k} - 2r_x^2 y_{k+1} + r_x^2$

Else

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

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

$P_{2k+1} = P_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$

We repeat the steps for region 2 until $y < 0$

(x_k, y_k)	P_{1k}	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$	P_{1k+1}
(0,6)	–332	(1,6)	72	768	–224
(1,6)	–224	(2,6)	144	768	–44
(2,6)	–44	(3,6)	216	768	208
(3,6)	208	(4,5)	288	640	–108
(4,5)	–108	(5,5)	360	640	288
(6,5)	288	(6,4)	432	512	244
(6,4)	244	(7,3)	504	384	

The point (7, 3) will be the initial point for region 2. The initial decision parameter for region 2 is P_{2k}

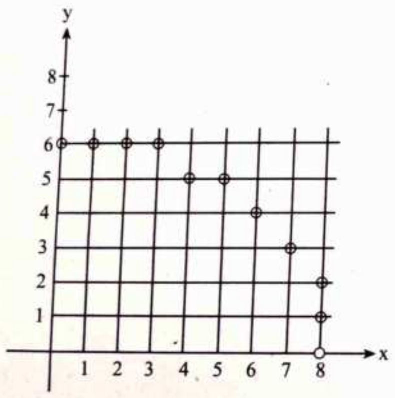
$$= 6^2\left(7+\frac{1}{2}\right)^2 + 8^2(3-1)^2 - 8^2 * 6^2$$

$= -23$

(x_k, y_k)	P_{2k}	(x_{k+1}, y_{k+1})	P_{2k+1}
(7,3)	−23	(8,2)	361
(8,2)	361	(8,1)	297
(8,1)	297	(8,0)	

So, the pixel positions or points to plot of the given curve in first quadrant using mid-point algorithm are

(0,6), (1,6), (2,6), (3,6), (4,5), (5,5), (6,4), (7,3), (8,2), (8,1) and (8,0)



Chapter-3

Two-Dimensional Transformations

3.1 Introduction

Complex picture can be treated as a combination of straight line, circles, ellipse, etc. and if we are able to generate these basic figures, we can also generate combinations of them. Transformation means changing the graphics by changing the position, orientation or size of the original graphics by applying rules. When transformation occurs in 2D plane then it is called 2D transformations. Geometrical transformation is a mathematical procedure which changes/alters orientation, size, and shape of objects i.e., the co-ordinate description of objects.

3.2 Basic Transformations

The basic transformations are:

- Translation
- Scaling
- Rotation

3.2.1 Translation / Shifting

Translation repositions an object along a straight line path from one co-ordinate location to another. A two dimensional point can be translated by adding translation distances t_x and t_y to the original co-ordinate position to move the point to a new position (x', y')

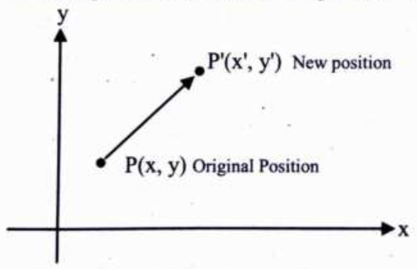


Figure 3.1: Translation of a point