Chapter 3
Raster Graphics Algorithm

Raster Graphics Algorithm and Scan Conversion

- In a raster graphics system, picture definition is stored in a memory called frame buffer or refresh buffer.
- Hence the process of determining the appropriate pixels for representing a picture (or graphics object) in raster i.e. pixels' format is called rasterization.
- The process of conversion of the rasterized picture i.e. picture definition, stored in the frame buffer into a set of pixel intensity values for the rigid display system is called scan conversion. This is done by display processor.
- In a raster display system, a picture is specified by set of intensities for the pixel positions in the display.
- We generate images by turning the pixels ON or OFF, so it is necessary to represent an object(continuous) as a collection of discrete pixels.

Line Scan Conversion Algorithm

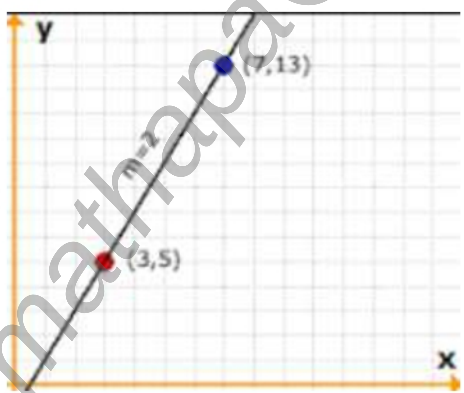
Digital Differential Analyzer(DDA) Algorithm

- It is a scan conversion line algorithm based on calculation either $\triangle x$ or $\triangle y$ using equation

  $m=\triangle y/\triangle x$……………………….i)
- We sample the line at unit interval in one direction (X direction if $\triangle x$ is greater than $\triangle y$, otherwise in y direction) and determine the corresponding integer values nearest to the line path for the other coordinate.

Derivation / Algorithm

Consider a line with **positive slope** as shown in the figure below



The equation of the line is given,

$Y = m.x+b$ ……………………(ii)

$m = (y2-y1)/(x2-x1)$ …………………(iii)

Case I:

If $|m| \leq 1$, we sample at unit x interval i.e $\triangle x = 1$.

$x_{k+1} = x_k+1$ ……………………(iii)

Then we compute each successive y-values using equation i by setting $\triangle y = m$

(since, $m=\triangle y/\triangle x$, $\triangle x=x_{k+1} - x_k = 1$, $\triangle y= y_{k+1} - y_k$).

So, $y_{k+1} = y_k + m$ …………………….(iv)

The calculated y value must be rounded to the nearest integer. Repeat $\triangle x$ times.

Case II:

If |m| > 1, we sample at unit y-interval i.e $\triangle y = 1$

$$y_{k+1} = y_k + 1 \ldots\ldots\ldots\ldots\ldots\ldots (v)$$

Then we compute each successive x-values using equation i by setting $\triangle x = 1/m$

(Since, $m=\triangle y/\triangle x$ and $\triangle y=\triangle y= y_{k+1} - y_k = 1$, $\triangle x=x_{k+1} - x_k$)

So, $x_{k+1} = x_k + 1/m \ldots \ldots\ldots\ldots (vi)$

The calculated x value must be rounded to the nearest integer. Repeat $\triangle y$ times.

- *The obtained equations also can be used to calculate the pixel position along a line with **negative slope.***

Advantage of DDA

1. It is simple to understand.
2. It is faster method than direct use of line equation y=mx+c. (Removes multiplication operation and only involve increments operation of x or y direction)
3. It requires no special skills for implementation

Disadvantages of DDA

1. A floating point addition is still needed in determining each successive point which is time consuming.
2. The accumulation of round off error in successive addition of the floating point increments may cause the calculated pixel position to drift away from the true line path for long line segment.

C-Implementation

```
void DDA(int x1,int y1,int x2, int y2)
{
        int dx,dy,xplot,yplot,i,steps;
        float slope,x,y;

        dx=x2-x1;
        dy=y2-y1;
        slope = (float)dy /(float)dx;

        x=x1;
        y=y1;
        xplot=x;
        yplot=y;

        if(fabs(slope)<=1)
        {
                for(i=0;i<abs(dx);i++)
                {
                        putpixel(xplot,yplot,RED);
                        x = x + 1;
                        y= y + slope;
                        xplot = x;
                        yplot=round(y);
                }
        }
```

```
        else
        {
                for(i=0;i<abs(dy);i++)
                {
                        putpixel(xplot,yplot,RED);
                        y=y+1;
                        x=x + (1.0/slope);
                        xplot=round(x);
                        yplot=y;
                }
        }
}
```

Numerical Examples

1. **Consider a line from (2,1) to (8,3). Using simple DDA algorithm, rasterize this line.**

Solution:

Given Starting Point :(x1, y1)= (2,1) and Ending Point: (x2,y2) = (8,3)

Now Slope m=(y2-y1)/(x2-x1)= (3-1)/(8-2) =(1/3) = 0.333

Since |m|<1, From DDA algorithm we have

$\mathbf{X_{k+1} = X_k + 1}$

$\mathbf{Y_{k+1} = Y_k + (M)}$

Repeat $\triangle x$=(8-2) = 6 times

Iteration	X	Y	X-Plot	Y-Plot	(X,Y)
0	2	1	2	1	(2,1)
1	3	1.33	3	1	(3,1)
2	4	1.66	4	2	(4,2)
3	5	1.993	5	2	(5,2)
4	6	2,326	6	2	(6,2)
5	7	2.659	7	3	(7,3)
6	8	2.999	8	3	(8,3)

Therefore, the digitized coordinates are (2,1), (3,1), (4,2), (5,2), (6,2), (7,3), (8,3).

2. **Digitized the line with end points (0,0) and (4,5) using DDA.**

Given Starting Point :(x1, y1)= (0,0) and Ending Point: (x2,y2) = (4,5)

Now Slope m=(y2-y1)/(x2-x1)= (5-0)/(5-0) =(5/4) = 1.25

Since |m|>1, From DDA algorithm we have

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

$x_{k+1} = x_k + (1/m)$

Repeat △y=(5-0) = 5 times

Iteration	X	Y	X-Plot	Y-Plot	(X,Y)
0	0	0	0	0	(0,0)
1	0.8	1	1	1	(1,1)
2	1.6	2	2	2	(2,2)
3	2.4	3	2	3	(2,3)
4	3.2	4	3	4	(3,4)
5	4	5	4	5	(4,5)

Therefore, the digitized coordinates are (0,0), (1,1), (2,2), (2,3), (3,4), (4,5).

3. Digitized the line with end points (3,7) and (8,3) using DDA.

Given Starting Point :(x1, y1)= (3,7) and Ending Point: (x2,y2) = (8,3)

Now Slope m=(y2-y1)/(x2-x1)= (3-7)/(8-3) =(-4/5) = -0.8

Since |m|<1, From DDA algorithm we have

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k + (M)$

Repeat △x=(8-3) = 5 times

Iteration	X	Y	X-Plot	Y-Plot	(X,Y)
0	3	7	3	7	(3,7)
1	4	7 - 0.8 = 6.2	4	6	(4,6)
2	5	5.4	5	5	(5,5)
3	6	4.6	6	5	(6,5)
4	7	3.8	7	4	(7,4)
5	8	3	8	3	(8,3)

4. Consider a line from (0,0) to (5,5). Using simple DDA algorithm, rasterize this line.

Solution:

Given Starting Point :(x1, y1)= (0,0) and Ending Point: (x2,y2) = (5,5)

Now Slope m=(y2-y1)/(x2-x1)= (5-0)/(5-0) =(/55) = 1

Since |m|<=1, From DDA algorithm we have

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k + (M)$

Repeat △x=(5-0) = 5 times

Iteration	X	Y	X-Plot	Y-Plot	(X,Y)
0	0	0	0	0	(0,0)
1	1	1	1	1	(1,2)
2	2	2	2	2	(2,2)
3	3	3	3	3	(3,3)
4	4	4	4	4	(4,4)
5	5	5	5	5	(5,5)

Bresenham's Line Algorithm

- An accurate (avoids round off as in DDA) and efficient raster line generating algorithm, developed by Bresenham, scan converts lines using only incremental integer calculations and also can be adapted to display circles and other curves.
- It introduces the integer decision parameter to select the next appropriate pixel point.
- Advantage of BLA over DDA
  1. In DDA algorithm, each successive point is computed in floating point, so it requires more time and more memory space but in BLA each successive point is calculated in integer value or whole number. So it requires less time and less memory space.
  2. In DDA, since the calculated point value is floating point number, it should be rounded at the end of calculation but in BLA it doesn't need to round, so there is no accumulation of rounding error.
  3. Due to rounding error, the line drawn by DDA algorithm is not accurate, while BLA is accurate.
  4. DDA algorithm can't be used in other application except line drawing but BLA can be implemented in other application such as drawing circle, ellipse and other curves.

Derivation

See class notes for Detail Derivation of Decision Parameter (both case)

Bresenham's Algorithm for 0<m≤1:

- Step 1: Input the line endpoints and store the left endpoint in (x0, y0).
- Step 2: Load (x0, y0) in to the frame buffer, i.e., plot the first point.
- Step 3: Calculate constants Δx, Δy, 2Δy -2Δx, and obtain the initial decision parameter as;

  $p0 = 2\Delta y - \Delta x$
- Step 4: At each xk along the line, starting at k = 0, perform the following test;
- If pk < 0

  Plot pixel(xk+1, yk), and

  Set pk+1 = pk +2Δy
- Otherwise,

  Plot pixel(xk+1, yk+1), and

  Set $p_{k+1} = p_k + 2\Delta y - 2\Delta x$
- Step 5: Repeat step 4, Δx times.
- Step 6: End.

Bresenham's Algorithm for m>1

- Step 1: Input the line endpoints and store the left endpoint in (x0, y0).
- Step 2: Load (x0, y0) in to the frame buffer, i.e., plot the first point.
- Step 3: Calculate constants Δx, Δy, 2Δx -2Δy, and obtain the initial decision parameter as;

  $p0 = 2\Delta x - \Delta y$
- Step 4: At each yk along the line, starting at k = 0, perform the following test;

- If pk < 0

  Plot pixel(x_k, y_{k+1}), and

  Set p_{k+1} = pk +2Δx
- Otherwise,
  - Plot pixel(x_{k+1}, y_{k+1}), and
  - Set $p_{k+1} = p_k + 2\Delta x - 2\Delta y$
- Step 5: Repeat step 4, Δy times.
- Step 6: End

Note:

- Horizontal lines (**Δy = 0,** vertical lines (**Δx = O),** and diagonal lines with **Δx** =Δy each can be loaded directly into the frame buffer without processing them through the line-plotting algorithm.
- **For negative slopes, the procedures are similar, except that now one coordinate decreases as the other increases.**

C-Implementation

```
void Bresenham(int x1, int y1, int x2, int y2)
{

        float m;
        int dx,dy,p,xplot,yplot,i,x,y;

        dx=abs(x2-x1);
        dy=abs(y2-y1);

        m=(float) dy/dx;

        x=x1;
        y=y1;
```

```
        if(m<=1)
        {
                p= 2 * dy -dx;

                for (i=0;i<abs(dx);i++)
                {
                        if(p<0)
                        {
                                putpixel(x,y,RED);
                                x=x+1;
                                y=y;
                                p= p + 2*dy;
                        }

                        else
                        {
                                putpixel(x,y,RED);
                                x=x+1;
                                y=y+1;
                                p=p + 2 * dy - 2* dx;
                        }
                }
        }
        else
        {
                p= 2 * dx - dy;

                for (i=0;i<abs(dy);i++)
                {
                        if(p<0)
                        {
                                putpixel(x,y,RED);
                                x=x;
                                y=y+1;
                                p= p + 2*dx;
                        }
                        else
                        {
                                putpixel(x,y,RED);
                                x=x+1;
                                y=y+1;
                                p=p + 2 * dx - 2* dy;
                        }
                }
        }

}
```


Questions:

1. Apply Bresenham's algorithm to draw a line from (-3,0) and end point is (4,4)

Solution

Starting Point (x0,y0) = (-3, 0) Ending Point (x_n, y_n) = (4,4) such that scanning takes place from left to right

Slope m = (4-0)/ (4+3) = 4/5 = 0.57 i.e. $|M| \leq 1$

Here: $\Delta x = 4 + 3 = 7$ $\Delta y = 4 - 0 = 4$ $2\Delta x = 2 * 7 = 14$ $2\Delta y = 2 * 4 = 8$

Now from Bresenhams Algorithm for |M|<1, and scanning from left to right

Initial Decision Parameter: $P_0 = 2\Delta y - \Delta x$
if(p_k<0) then
$X_{k+1} = X_k + 1$ and
$Y_{k+1} = Y_k$ and
$P_{k+1} = P_k + 2\Delta y$
otherwise
$X_{k+1} = X_k+1$ and
$Y_{k+1} = Y_k+1$ and
$P_{k+1} = P_k + 2\Delta y - 2\Delta x$
Repeat Δx times

K	P_k	$(X_{k+1}, Y_{k+1}$
0	(2 * 4 -7)=1	(-3+1, 0+1) = (-2,1)
1	(1 + 8-14)= -5	(-2+1, 1)=(-1,1)
2	(-5+8)=3	(-1+1, 1+1)=(0,2)
3	-3	(1,2)
4	5	(2,3)
5	-1	(3,3)
6	7	(4,4)

Therefore the digitized coordinates are (x0, y0) = (-3, 0), (x1, y1) = (-2, 1),……(x7,y7) = (4,4)

2. Apply Brenham's algorithm to draw a line from (3,7) and end point is (8,3)

Solution

Starting Point (x_0,y_0) = (3, 7) Ending Point (x_n, y_n) = (8, 3) such that scanning takes place from left to right

Slope m = (3-7)/ (8-3) = -0.8 i.e. |M|≤1

Here: $\Delta x = |8-3| = 5$ $\Delta y = |3-7| = 4$ $2\Delta x = 2 * 5 = 10$ $2\Delta y = 2 * 4 = 8$

Now from Bresenhams Algorithm for |M|<=1, and scanning from left to right

Initial Decision Parameter: $P_0 = 2\Delta y - \Delta x$
if(p_k<0) then
$X_{k+1} = X_k + 1$ and
$Y_{k+1} = Y_k$ and
$P_{k+1} = P_k + 2\Delta y$
otherwise
$X_{k+1} = X_k+1$ and
$Y_{k+1} = Y_k$-1 and
$P_{k+1} = P_k + 2\Delta y - 2\Delta x$
Repeat Δx times

K	P_k	$(X_{k+1}, Y_{k+1}$
0	8-5=3	(4,6)
1	3 + 8 – 10 = 1	(5,5)
2	1 + 8 – 10 = -1	(6, 5)
3	-1 + 8 = 7	(7,4)
4	7 + 8 – 10 = 5	(8,3)

Therefore the digitized coordinates are (3,7), (4,6) , (5,5) , (6,5) , (7,4) , (8,3)

3. Apply Bresenham's algorithm to draw a line from (20,15) and end point is (30,30)

Solution

Starting Point $(x_0,y_0) = (20, 15)$ Ending Point $(x_n, y_n) = (30, 30)$ such that scanning takes place from left to right

Slope m = (30-15)/ (30-20) = 1.5 i.e. |M|>1

Here: $\Delta x = 30-20 = 10$ $\Delta y = 30-15 = 15$ $2\Delta x = 2 * 10 = 20$ $2\Delta y = 2 * 15 = 30$

Now from Bresenhams Algorithm for |M|>1, and scanning from left to right

Initial Decision Parameter: $P_0 = p0 = 2\Delta x - \Delta y$
if(p_k<0) then
$X_{k+1} = X_k$ and
$Y_{k+1} = Y_k+1$ and
$P_{k+1} = P_k + 2\Delta x$
otherwise
$X_{k+1} = X_{k+1}$ and
$Y_{k+1} = Y_k+1$ and
$P_{k+1} = P_k + 2\Delta x - 2\Delta y$
Repeat Δy times

K	P_k	$(X_{k+1}, Y_{k+1}$
0	5	(21,16)
1	-5	(21,17)
2	15	(22,18)
3	5	(23,19)
4	-5	(23,20)
5	15	(24,21)
6	5	(25,22)
7	-5	(25,23)
8	15	(26,24)
9	5	(27,25)
10	-5	(27,26)
11	15	(28,27)
12	5	(29,28)
13	-5	(29,29)
14	15	(30,30)

Therefore the digitized coordinates are (x0, y0) = (20,15), (x1, y1) = (21,16),……..(x15, y15) = (30,30)

4. Apply Bresenham's algorithm to draw a line from (6, 12) and end point is (10, 5)

Solution

Starting Point (x0,y0) = (6, 12) Ending Point $(x_n, y_n) = (10,5)$ such that scanning takes place from left to right

Slope m = (5-12)/ (10-6) = -7/4 = -1.75 i.e. |M|>1
Here: $\Delta x = |10-6| = 4$ $\Delta y = |5-12| = 7$ $2\Delta x = 2 * 4 = 8$ $2\Delta y = 2 * 7 = 14$

Now from Bresenhams Algorithm for |M|>1, and scanning from left to right

Initial Decision Parameter: $P_0 = p0 = 2\Delta x - \Delta y$
if(p_k<0) then
$X_{k+1} = X_k$ and
$Y_{k+1} = Y_k$-1 and
$P_{k+1} = P_k + 2\Delta x$
otherwise
$X_{k+1} = X_k$+1 and
$Y_{k+1} = Y_k$-1 and
$P_{k+1} = P_k + 2\Delta x - 2\Delta y$
Repeat Δy times

K	P_k	$(X_{k+1}, Y_{k+1}$
0	2 * 4 -7=1	(6+1, 12-1)=(7,11)
1	1 + 8 – 14=-5	(7, 11-1)=(7,10)
2	-5+8=3	(7+1, 10-1)=(8,9)
3	3+8-14=-3	(8,9-1)=(8,8)
4	-3+8=5	(8+1, 8-1)=(9,7)
5	5+8-14=-1	(9,7-1)=(9,6)
6	-1+8=7	(9+1, 6-1)=(10,5)

Therefore the digitized coordinates are (x0, y0) = (6, 12), (x1, y1) = (7, 11),…….(x7,y7) = (10,5)

5. Apply Bresenham's algorithm to draw a line from (16,20) and end point is (20,15)
Solution
Starting Point (x_0,y_0) = (16, 20) Ending Point (x_n, y_n) = (20, 15) such that scanning takes place from left to right
Slope m = (15-20)/ (20-16) = -1.25 i.e. |M|>1
Here: $\Delta x = |20-16| = 4$ $\Delta y = |15-20| = 5$ $2\Delta x = 2 * 4 = 8$ $2\Delta y = 2 * 5 = 10$

Now from Bresenhams Algorithm for |M|>1, and scanning from left to right

Initial Decision Parameter: $P_0 = p0 = 2\Delta x - \Delta y$
if(p_k<0) then
$X_{k+1} = X_k$ and
$Y_{k+1} = Y_k$-1 and
$P_{k+1} = P_k + 2\Delta x$
otherwise
$X_{k+1} = X_{k+1}$ and
$Y_{k+1} = Y_k$-1 and
$P_{k+1} = P_k + 2\Delta x - 2\Delta y$
Repeat Δy times

K	P_k	$(X_{k+1}, Y_{k+1}$
0	8-5=3	(17,19)
1	3 + 8 – 10 = 1	(18,18)

2	1 + 8 – 10= -1	(18,17)
3	-1 + 8 = 7	(19,16)
4	7 + 8 – 10 = 5	(20, 15)

Therefore the digitized coordinates are (x0, y0) = (20,15), (x1, y1) = (21,16),……(x15, y15) = (30,30)

Midpoint Circle Generating algorithm

- Since a circle is a symmetrical figure, midpoint circle generating algorithm takes this advantage of symmetry property to plot right points for each value that the algorithm calculates.

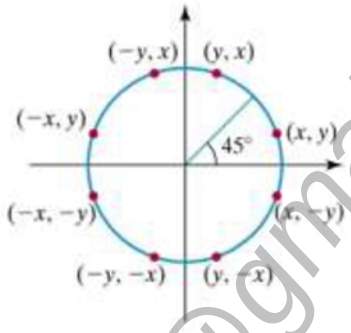


Figure 3-18

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

- Eight-way symmetry is used by reflecting each calculated point around each 45-degree axis.
- Suppose a calculated point is (x,y), then we can obtain eight different point as:

  (x,y), (x,-y), (-x,y), (-x,-y), (y,x), (-y,x), (y,-x), (-y,-x)
- So using this symmetry principal, we can reduce the amount of computation.

Midpoint Circle Algorithm

1. Input radius r and circle center (x_c, y_c), and obtain the first point on the circumference of a circle centered on the origin as

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

2. Calculate the initial value of the decision parameter as

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

3. At each x_k position, starting at $k = 0$, perform the following test: If $p_k < 0$, the next point along the circle centered on $(0, 0)$ is (x_{k+1}, y_k) and

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

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

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

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

4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c, \qquad y = y + y_c$$

6. Repeat steps 3 through 5 until $x \geq y$.

//See your class note for detail derivation of Decision Parameter

C-Implementation

```
void mpcircle(int xc, int yc, int rad)
{
        int x,y,p,i;
        //print first pixel
        x=0;
        y=rad;
        applysymmetry(x,y,xc,yc);
```

```
        p=1-rad;
        while(x<y)
        {
                if(p<0)
                {
                        x=x+1;
                        y=y;
                        p=p + 2 * x +1;
                }
                else
                {
                    x=x+1;
                    y=y-1;
                    p = p+ 2*x + 1 - 2 * y;
                }
            applysymmetry(x,y,xc,yc);
        }
}

void applysymmetry(int x,int y,int xc,int yc)
{
        putpixel(x+xc,y+yc,RED);
        putpixel(x+xc,-y+yc,RED);
        putpixel(y+xc,-x+yc,RED);
        putpixel(-y+xc,-x+yc,RED);
        putpixel(-x+xc,-y+yc,RED);
        putpixel(-x+xc,y+yc,RED);
        putpixel(-y+xc,x+yc,RED);
        putpixel(y+xc,x+yc,RED);
}
```

Numerical

1. **Digitized the circle with radius 10. (Digitized a circle $X^2 + Y^2 = 100$ on first octant.)**

   Solution:

   Initial Point = (Xo, Yo) = (o, r) = (0,10) and origin = (0,0)

   Initial decision parameter (p_0) = $1 - r = 1 - 10 = -9$

   From midpoint circle algorithm we have

   If $P < 0$

   Plot $(x_k + 1, y_k)$

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

   Else $(P \geq 0)$

   Plot $(x_k + 1, y_k - 1)$

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

   Thus,

   Now, the successive decision parameter values and position along the circle path are determined as follow.

K	P_k	X_{k+1}	Y_{K+1}	$(X_{K+1}, Y_{k=1})$
0	-9	1	10	(1,10)
1	-9*2*1+1=-6	2	10	(2,10)
2	-6 + 2*2+1=-1	3	10	(3,10)
3	-1 + 2 * 3 + 1 = 6	4	9	(4,9)
4	6 + 2 * 4 – 2 * 9 + 1 =-3	5	9	(5,9)
5	-3 + 2 * 5 + 1= 8	6	8	(6,8)
6	8 + 2 * 6 – 2 * 8 + 1 = 5	7	7	(7,7)
7	5+ 2 * 7-2 * 7 + 1 =6	8	6	[stop and discard this point since x>y

Hence, the digitized coordinates for circle in first octant are (x0,y0) = (0, 10), (x1,y1) = (1,10).....(x7,y7)=(7,7).

Note: For each calculated point we need to apply symmetry principle to obtain seven other different coordinates one for each of the seven different octants.

2. Digitized a circle with radius r=12 and centered at (250,300)

Solution:

Note: When center is not origin, we first calculate the octants points of the circle in the same way as the center at origin then add the given circle center on each calculated pixels.

Here, Initial Point (x0,y0) (0,r) = (0,12) and origin=(0,0)

Initia l decision parameter (p_0) = 1 – 12 = 1 – 12 = -11

From midpoint circle algorithm we have

If $P < 0$

Plot $(x_k + 1, y_k)$

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

Else $(P \geq 0)$

Plot $(x_k + 1, y_k - 1)$

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

Thus,

Now, the successive decision parameter values and position along the circle path are determined as follow.

K	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1}) at (0,0)	(X_{k+1}, Y_{k+1}) at (250,300)
0	-11	1	12	(1,12)	(250+1, 300+12)=(251,312)
1	-11 + 2* 1 + 1 = -8	2	12	(2,12)	(250+2,300+12) =(252,312)
2	-8 + 2 * 2 + 1 = -3	3	12	(3,12)	(250+3,300+12) =(253,312)
3	-3 + 2 * 3 + 1=4	4	11	(4,11)	(250+4,300+11) =(254,311)
4	4 + 2 * 3 – 2 * 11 + 1= -9	5	11	(5,11)	(250+5,300+11) =(255,311)
5	-9 + 2 * 5 + 1 = 2	6	10	(6,10)	(250+6,300+10) =(256,310)
6	2 + 2*6 – 2* 10 + 1 = -5	7	10	(7,10)	(250+7,300+10) =(257,310)
7	-5 + 2 * 7 + 1 = 10	8	9	(8,9)	(250+8,300+9) =(258,309)
8	10+ 16 – 18 + 1= 9	9	8	(Discard and stop since X>Y)	

Therefore the digitized coordinates are (Xo,Yo) = (250+0, 300+12) = (250,312), (X1,Y1)= (251,312)..... (X8,Y8)= (258,312)

- **Derive Decision parameter for midpoint circle generating algorithm taking octant y=0 to y<=x and moving in anti-clock direction.**

Note: Here we need to scan convert a circle when starting point is (r,0) and moving in anticlockwise direction. For this we take the octant y=0 to y<=x (here, $|m| > 1$). Hence, we step in unit interval in y direction i.e. $y_{K+1} = Y_k + 1$ and find each successive X i.e. X_k or X_{k-1}

Midpoint Ellipse Algorithm

Midpoint Ellipse Algorithm

1. Input r_x, r_y, and ellipse center (x_c, y_c), and obtain the first point on an ellipse centered on the origin as

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

2. Calculate the initial parameter in region 1 as

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. At each x_i position, starting at $i = 0$, if $p1_i < 0$, the next point along the ellipse centered on (0, 0) is $(x_i + 1, y_i)$ and

$$p1_{i+1} = p1_i + 2r_y^2 x_{i+1} + r_y^2$$

otherwise, the next point is $(x_i + 1, y_i - 1)$ and

$$p1_{i+1} = p1_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_y^2$$

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

Midpoint Ellipse Algorithm

4. (x_0, y_0) is the last position calculated in region 1. Calculate the initial parameter in region 2 as

$$p2_0 = r_y^2 (x_0 + \tfrac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

5. At each y_i position, starting at $i = 0$, if $p2_i > 0$, the next point along the ellipse centered on (0, 0) is $(x_i, y_i - 1)$ and

$$p2_{i+1} = p2_i - 2r_x^2 y_{i+1} + r_x^2$$

otherwise, the next point is $(x_i + 1, y_i - 1)$ and

$$p2_{i+1} = p2_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_x^2$$

Use the same incremental calculations as in region 1. Continue until $y = 0$.

6. For both regions determine symmetry points in the other three quadrants.
7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values

$$x = x + x_c, \qquad y = y + y_c$$

//see class notes for derivation of Decision Parameter

Numerical

1. The input ellipse parameters are $r_x = 8$ and $r_y = 6$. Using midpoint method, rasterize this ellipse.

Solution:

Given $r_x = 8$ and $r_y = 6$

For Region 1

The initial point for the ellipse on the origin (xo, yo) = (0, r_y) = (0,6)

Calculation the initial decision parameter

$P_{10} = r^2_y + (r^2_x / 4) - r_y r^2_x$

$= 6^2 + (8^2 / 4) - 6 * (8)^2$

$= -332$

From midpoint algorithm, for Region 1 we know,

If(P_k<0) then

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k$

$P_{K+1} = P_{1k} + r^2_y + 2r^2_y(X_{K+1})$

Else (i.e. P_k >=0)

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k - 1$

$P_{K+1} = P_k + r^2_y + 2r^2_y(X_{K+1}) - 2r^2_x(Y_{K+1})$

And stop when $2r^2_y X_{k+1} >= 2r^2_x Y_{k+1}$

Now successive decision parameter values and positions along the ellipse path are calculated using midpoint method as

K	P_{1k}	$(X_{k+1}, Y_{K=1})$ at (o, o)	$2r^2_y X_{k+1}$	$2r^2_x Y_{k+1}$
0	-332	(1, 6)	72	768
1	-224	(2,6)	144	768
2	-44	(3,6)	216	768
3	208	(4,5)	288	640
4	-108	(5,5)	360	640
5	288	(6,4)	432	512
6	244	(7,3)	504	384

Since $2r^2_y X_{k+1} >= 2r^2_x Y_{k+1}$, so we stop here for region 1.

(note: (7, 3) is the starting point for region 2)

Now for Region-2,

From midpoint algorithm, for Region 2 we know,

Initial Decision Parameter:

$P_{20} = (X_k + (1/2))^2 r_y^2 + (Y_k - 1)^2 r_x^2 - r_x^2 r_y^2$ (is calculated for the point when it enters region 2)

If(P_k<=0) then

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k - 1$

$P_{K+1} = p_k + 2r_y^2(x_{K+1}) - 2r_x^2(Y_{K+1}) + r_x^2$

Else (i.e. $P_k > 0$)

$X_{k+1} = X_k$

$Y_{k+1} = Y_k - 1$

$P_{K+1} = p_k - 2r_x^2(Y_{K+1}) + r_x^2$

So, the initial point for region 2 is $(x_0, y_0) = (7, 3)$ and the initial decision parameter is

$P_{20} = (X_k + (1/2))^2 r_y^2 + (Y_k - 1)^2 r_x^2 - r_x^2 r_y^2$

$= 36 * (7 + 0.5)^2 + 64 * (3-1)^2 - 64 * 36$

$= -23$

The remaining pixels in region two for first quadrant are than calculated as

K	P_{2k}	$(X_{k+1}, Y_{K=1})$ at (o,o)
0	-23	(8,2)
1	361	(8,1)
2	297	(8,0) [$(r_x,0)$ so stop]

So remaining points in other quadrants can be calculated using the symmetry property of ellipse.

Filled Area Primitives

- 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.
  - i. One way to fill an area is to determine the overlap intervals for scan lines that crosses the area.
    1. Scan Line Polygon Fill Algorithm
    2. Scan Line Fill of Curve Boundary Area
  - ii. Another method for area filling is to start from a given interior position and point outward from this until a specified condition is met.
    1. Boundary Fill Algorithm
    2. Flood Fill Algorithm

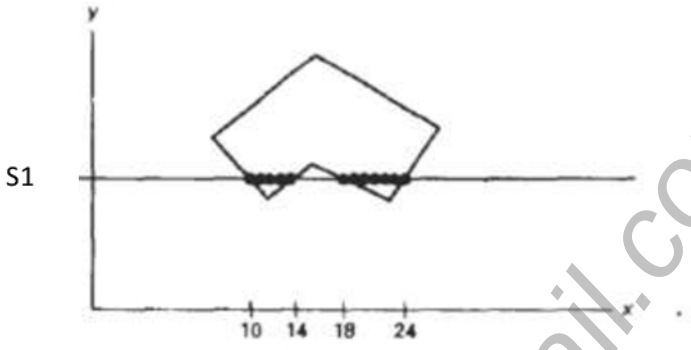
Scan Line Polygon Fill Algorithm

iii. Polygon is an ordered list of vertices as shown in the following figure.

iv. For filling polygons with particular colors, we need to determine the pixels falling on the border of the polygon and those which fall inside the polygon.

v. One way to fill an area is to determine the overlap intervals for scan lines that crosses the area.

vi. In this 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 fill color.

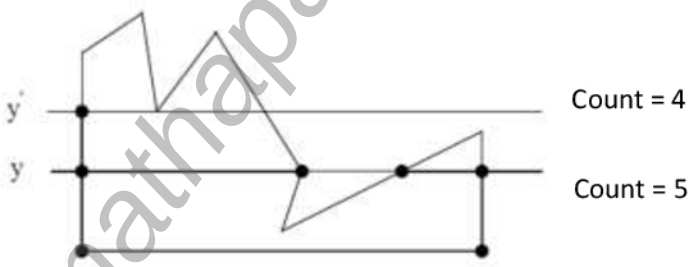


vii. In the above figure, for the scanline S1, there are four intersection points: X_{10}, X_{14}, X_{18}, and X_{24}, thus giving two intersection Pairs: Pair1 = (X_{10}, X_{14}) and Pair2 = (X_{18}, X_{24}).

viii. Some scan line intersections at polygon requires extra processing when the scan line passes through vertex (An intersection point that connect two edge).

ix. In this method, the scan line intersection points count must be **even** so for vertex the intersection point must be count **twice.**

x. Figure below shows two scan line y and y^1 that intersects at polygon edges.



xi. For scan line y', as it passes through vertex we count that intersection point twice and hence total intersection point is even i.e. count = 4. This strategy worked as expected.

xii. But this consideration also may create problem for some instant, like scan line y. Here total intersection point is odd i.e. count=5. So, we need to do some additional processing to determine the correct interior points.

xiii. The difference between scan line y and scan line y' is the position of the intersecting edges relative to the scan line.

xiv. For scan line y, the two intersecting edges sharing a vertex are on opposite sides of the scan line. But for scan line y', the two intersecting edges are both above the scan line.

xv. Thus for vertices that have connecting edges on opposite sides of the scan line, the intersection point must be counted as **once not twice** so that we get total count of **four i.e. even** for scan line y.

Scan line fill of Curved Boundary area

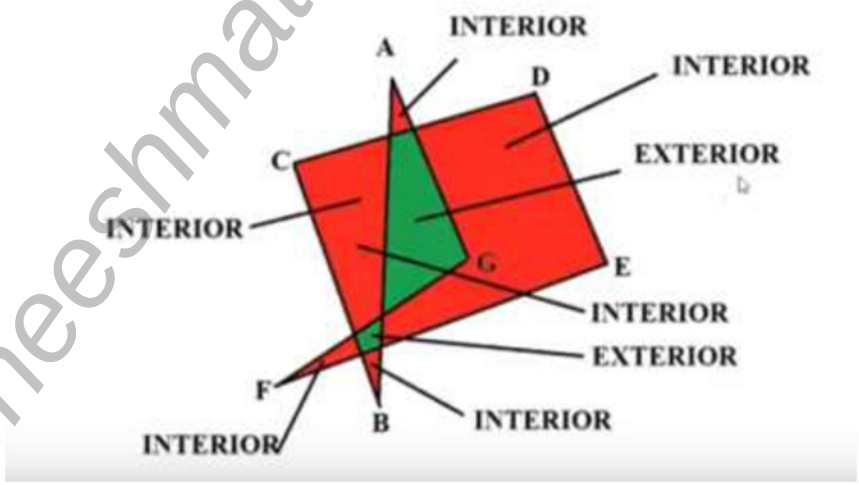
- In general, scan-line fill of regions with curved boundaries requires more work than polygon filling, since intersection calculation now involve nonlinear boundaries.
- For simple curves such as circles or ellipses, performing a scan-line fill is a straightforward process. We only need to calculate the two scan-line Intersection on opposite sides of the curve. This is the same as generating pixel positions along the curve boundary, and we can do that with the midpoint method-
- Then we simply fill in the horizontal pixel spans between the boundary points on opposite side of the curve

Fig: Interior fill of an elliptical arc

Inside outside Test

- Area-filling algorithms and other graphics processes often need to identify interior regions of objects.
- A polygon can be self-intersecting, meaning edges cross other edges. (The points of intersection are not vertices.)
- For a polygon with no self-intersection, identifying the interior regions of such polygon is straightforward. However, when a polygon has intersecting edges then it is difficult to identify which regions are inside and which are outside.
- So for this, there are two techniques to find out if the region is interior or exterior
  1. Odd even rule
  2. Non zero winding number

1. Odd even rule
   - One of the method for performing inside outside test is odd even rule/odd parity rule or even odd rule.
   - In this technique a line is drawn from any position P to a distant point **outside the coordinate extents** of the object and then counting the number of edge crossings along the line.
   - If the number of polygon edges crossed by this line is **odd, then P is an interior point**. Otherwise, P is an exterior point.
   - To obtain an accurate edge count, we must be sure that the line path we choose does not intersect any **polygon vertices.**
   - **The scan-line polygon fill algorithm discussed in the previous section is an example of area filling using the odd-even rule**

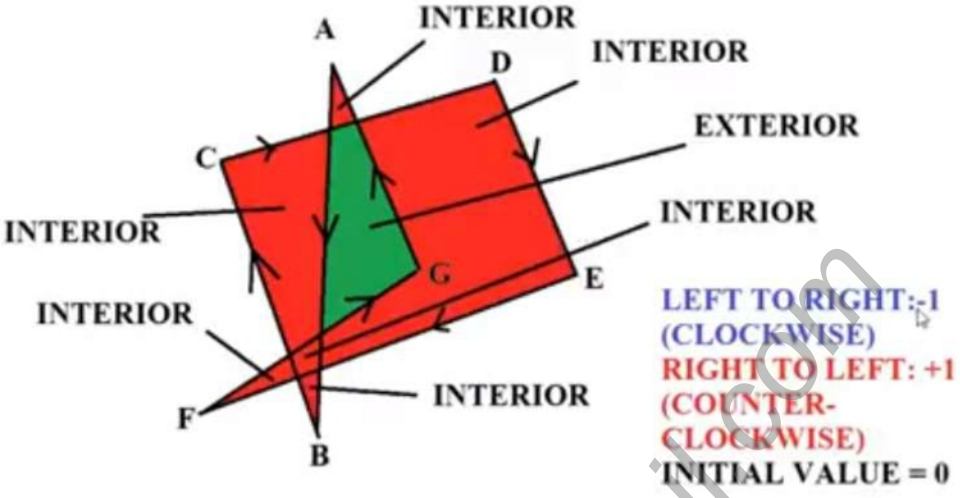


Example: Odd even rule

2. Non zero winding number
   - The winding number is number of the times the polygon edges wind around a particular point in the **counterclockwise direction.**
   - To apply non zero winding number rule, first the winding number is **initialized to zero**.
   - Then we imagine drawing a line from point **p to outside the polygon** which **doesn't pass through any vertex.**
   - Now we add one to the winding number every time we intersect the polygon edge that crosses the line

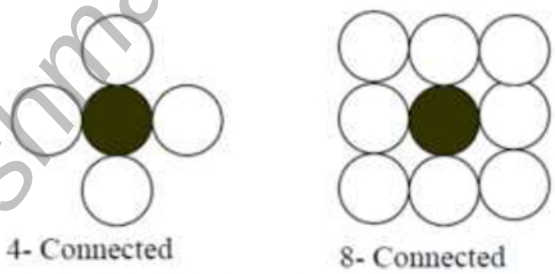
form **right to left, and** subtract one every time we intersect and edge that crosses from **left to right.**

- The **interior points are that which have a non-zero value** for the winding number and **exterior points are those that have zero value** for winding number.



Boundary Fill Algorithm

- In Boundary filling algorithm, we start at a point called seed pixel inside a region (example: inside **ellipse**) 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.
- A boundary-fill procedure accepts as input the co-ordinates of an interior point (x,y), a fill color, and a boundary color. Starting from (x,y), the procedure tests neighbouring positions to determine whether they are of boundary color. If not, they are painted with the fill color, and their neighbours are tested. This process continues until all pixel up to the boundary color area have tested.
- The neighbouring pixels from current pixel are proceeded by two methods:



  - 4- connected: if they are adjacent horizontally and vertically.
  - 8- connected: if they adjacent horizontally, vertically and diagonally.
- Since this procedure requires considerable stacking of neighboring points, more efficient methods are generally employed
- Recursive boundary-fill algorithm may not fill 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.
- It can be used in a graphics tablet or other interactive device, by an artist or designer to sketch a figure outline, select a fill color or pattern from a color menu, and pick an interior point.

C-Implementation

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);
        }
}
```

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. i.e. coloring of the pixels of a region is done with the fill color until we keep getting the old interior color.
- Using either a 4-connected or 8-connected approach, we then step through pixel positions until all interior points have been repainted. The following procedure flood fills a 4-connected region recursively, starting from the input position.
- It is used in **bucket fill** tool of paint program.

C-Implementation / 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);
       }
}
```

Flood Fill Vs Boundary Fill Algorithm

Boundary Fill Algorithm	Flood Fill Algorithm
Area filling is started inside a point with in a boundary region and fill the region with in the specified color until it reaches the boundary.	Area filling is started from a point and it replaces the old color with the new color
Boundary can be defined with single color.	Boundary can be defined with multiple color.
Need to deal with boundary color	No need to deal with boundary color.
The interior points are replaced with new color.	The old color is replaced with new color.
It is less time consuming	It consumes more time
It searches for boundary.	It searches for old color

Assignment-3

1. Apply Bresenham's algorithm to draw a line from (0,40) and end point is (40,0)
2. **Apply Bresenham's algorithm to draw a line between the endpoints (-2,-4) and (-6,-9)**
3. Apply Midpoint circle algorithm to draw a circle of radius 9 centered at (6,7).

Reference Book:
Computer Graphics, Donald D. Hearn

Note

- Send the scanned file of handwritten document in pdf/doc format in this address thapa_bheeshma@hotmail.com
- Clearly specify your College name, full name, college roll number and assignment number in subject field of email.

Last date of submission: .....