**LAB 3**

**OBJECTIVE:**

TO IMPLEMENT MIDPOINT CIRCLE GENERATION ALGORITHM OR BRESENHAM’S CIRCLE ALGORITHM FOR DRAWING A CIRCLE OF GIVEN CENTER (x,y) AND RADIUS r.

**THEORY:**

Circle have the property of being highly symmetrical, which is handy when it comes to drawing them of a display screen.

* We know that there are 360 degrees in a circle. First we see that a circle is symmetrical about the x axis, so only the first 180 degrees need to be calculated.
* Next we see that its also symmetrical about the y axis, so now we only need to calculate the first 90 degrees.
* Finally we see that the circle is also symmetrical about the 45 degree diagonal axis, so we only need to calculate the first 45 degrees.
* We only need to calculate the values on the boarder of the circle in the first octant. The other values may be determined symmetry.

Bresenham’s circle algorithm calculates the locations of the pixels in the first 45 degrees. It assumes that the circle is centered on the origin. So for every pixel (x,y) it calculates, we draw a pixel in each of the eight octants of the circle. This is done till when the value of the y coordinate equals the x coordinate. The pixel positions for determining symmetry are given in the below algorithm.

The algorithm for Midpoint Circle Generation is given in following steps:

1. Input radius r and circle center (xc,yc), then set the coordinates for the first point on the circumference of a circle centered on the origin as:

(xc,yc) = (0,r)

1. Calculate the initial value of the decision parameter as:

1. Starting with k = 0 at each position xk, perform the following test. If pk < 0, 0, the next point along the circle centered on (0,0) is (xk+1,yk) and:

Otherwise the next point along the circle (xk+1,yk-1) and:

1. Determine symmetry points in the other seven octants
2. Move each calculated pixel position (x,y) onto the circular path centered (xc,yc) to plot the coordinate values:
3. Repeat steps 3 to 5 until x>=y.

**SOURCE CODE:**

#include<stdio.h>

#include<graphics.h>

#include<math.h>

#include<conio.h>

#include<dos.h>

void draw\_circle(int,int,int);

void symmetry(int,int,int,int);

int main()

{

int xc,yc,R;

int graphicsdriver=DETECT,graphicsmode;

initgraph(&graphicsdriver,&graphicsmode,"C:\\TURBOC3\\BGI");

printf("Enter the center of the circle:\n");

printf("Xc=");

scanf("%d",&xc);

printf("Yc=");

scanf("%d",&yc);

printf("Enter the radius of the circle:\n");

scanf("%d",&R);

draw\_circle(xc,yc,R);

getch();

closegraph();

return 0x1337;

}

void draw\_circle(int xc, int yc, int rad)

{

int x=0;

int y = rad;

int p = 1-rad;

symmetry(x,y,xc,yc);

for(x=0;y>=x;x++)

{

if(p<0)

p+=2\*x+3;

else {

p+=2\*(x-y)+5;

y--;

}

symmetry(x,y,xc,yc);

delay(50);

}

}

void symmetry(int x, int y, int xc, int yc)

{

putpixel(xc+x,yc+y,GREEN);

// printf("%d%d\n",(xc+x),(yc+y));

delay(50);

putpixel(xc+y,yc+x,GREEN);

delay(50);

putpixel(xc-y,yc+x,GREEN);

delay(50);

putpixel(xc-x,yc+y,GREEN);

delay(50);

putpixel(xc-x,yc-y,GREEN);

delay(50);

putpixel(xc-y,yc-x,GREEN);

delay(50);

putpixel(xc+y,yc-x,GREEN);

delay(50);

putpixel(xc+x,yc-y,GREEN);

delay(50);

}

OUTPUT: