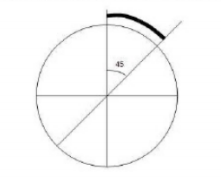
**Practical No 3**

**A) Implementation of Bresenham circle drawing algorithm.**

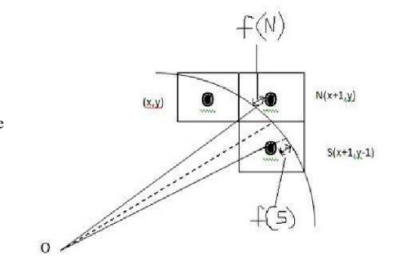
**Aim: Write a program to implementation of Bresenhams circle drawing algorithm.**

**Theory:**

Circles have the property of being highly symmetrical, which is handy when it comes to drawing them on a display screen. We know that there are 360 degree in a circle. First, we see that circle is symmetrical about the x axis, so only the first 180 degrees need to be calculated.



Next, we see that it’s also symmetrical about the y axis, so now we only need to calculate the first 90 degrees. Finally, we see the circle is also symmetrical about 45 degree diagonal axis, so we only need to calculate the first 45 degrees. Bresenham’s circle algorithm also calculates the locations of the pixels in the first 45 degrees. It assumes that the circle is cantered on the origin shifting the original center coordinates (centerx, centery). So, for every pixel (x, y) it calculates, we draw a pixel in each of the 8 octants of the circle.

At any point (x, y) we have two choices to choose the pixel on east of it or the south-east pixel. To choose the pixel, we first need to determine the errors involved with both N and S which are f(N) and f(S) respectively and whichever gives the lesser error, that the pixel we need to choose.

Let di=f(N)+f(S), where d is called as decision parameter, so that if (di<=0),

then N (x+1, y) will be chosen as the next pixel i.e., xi+1=xi+1 and yi+1=yi, and if (di>0), then S (x+1, y-1) is to be chosen as next pixel. i.e., xi+1=xi+1 and yi+1=yi-1.

We know that for a circle x2+y2=r2, where r represents the radius of circle.

**Algorithm:**

Here xc and yc denote the x-coordinate and y-coordinate of the center of circle. R is denoted as the radius of the circle.

1. Set X = 0 and Y = R
2. Set D = 3 – 2R
3. Repeat While (X < Y)
4. Call Draw Circle(Xc, Yc, X, Y)
5. Set X = X + 1
6. If (D < 0) Then
7. D = D + 4X + 6
8. Else
9. Set Y = Y – 1
10. D = D + 4(X – Y) + 10 [End of If]
11. Call Draw Circle(Xc, Yc, X, Y) [End of While]
12. Exit

* Now it will run the drawcircle funtion
* Draw Circle (Xc, Yc, X, Y):
* Call PutPixel(Xc + X, Yc, + Y)
* Call PutPixel(Xc - X, Yc, + Y)
* Call PutPixel(Xc + X, Yc, - Y)
* Call PutPixel(Xc - X, Yc, - Y)
* Call PutPixel(Xc + Y, Yc, + X)
* Call PutPixel(Xc - Y, Yc, + X)
* Call PutPixel(Xc + Y, Yc, - X)
* Call PutPixel(Xc - Y, Yc, - X)
* Exit

**Conclusion: We have implemented Bresenhams circle drawing algorithm.**

**Code:**

#include <graphics.h>

#include <stdlib.h>

#include <stdio.h>

#include <iostream.h>

#include <conio.h>

#include <math.h>

#include <dos.h>

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

{

putpixel(x+xc,y+yc,15);

putpixel(x+xc,-y+yc,15);

putpixel(-x+xc,-y+yc,15);

putpixel(-x+xc,y+yc,15);

putpixel(y+xc,x+yc,15);

putpixel(y+xc,-x+yc,15);

putpixel(-y+xc,-x+yc,15);

putpixel(-y+xc,x+yc,15);

}

void BCircle(int xc,int yc,int r)

{

int x=0,y=r,d=3-(2\*r);

DCircle(xc,yc,x,y);

while(x<=y)

{

if(d<=0)

{

d=d+(4\*x)+6;

}

else

{

d=d+(4\*x)-(4\*y)+10;

y=y-1;

}

x=x+1;

delay(100);

DCircle(xc,yc,x,y);

}

}

int main(void)

{

clrscr();

int xc,yc,r,gdriver = DETECT, gmode, errorcode;

initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

cout<<"Bresenham's circle drawing algorithm:\n";

cout<<"Enter the values of xc and yc: ";

cin>>xc>>yc;

cout<<"Enter the value of radius: ";

cin>>r;

BCircle(xc,yc,r);

getch();

closegraph();

return 0;

}

**Output:**

