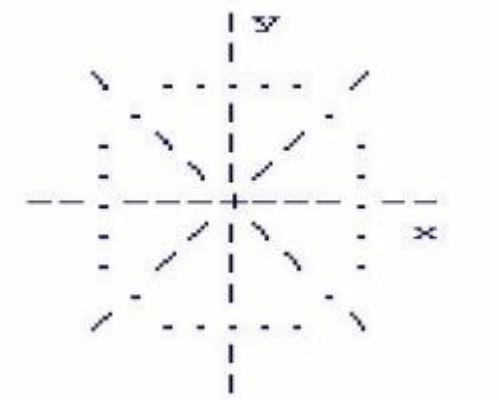
Experiment No 3

Title Implement Bresenham circle drawing algorithm to draw any object. The object should be displayed in all the quadrants with respect to center and radius

Bresenham's Circle Algorithm 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 degrees in a circle. First we see that a circle is symetrical about the x axis, so only the first 180 degrees need to be calculated. Next we see that it's also symetrical about the y axis, so now we only need to calculate the first 90 degrees. Finally we see that the circle is also symetrical about the 45 degree diagonal axis, so we only need to calculate the first 45 degrees. 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 8 octants of the circle :

PutPixel(CenterX + X, Center Y + Y)

PutPixel(CenterX + X, Center Y - Y)

PutPixel(CenterX - X, Center Y + Y)

PutPixel(CenterX - X, Center Y - Y)

PutPixel(CenterX + Y, Center Y + X)

PutPixel(CenterX + Y, Center Y - X)

PutPixel(CenterX - Y, Center Y + X)

PutPixel(CenterX - Y, Center Y - X)

Algorithm :

Given a radius for the circle we perform this initialisation:

d := 3 - (2 * RADIUS)

x := 0

y := RADIUS

Now for each pixel we do the following operations until x = y :

Draw the 8 circle pixels

if d < 0 then

d := d + (4 * x) + 6

else

begin

d := d + 4 * (x - y) + 10

y := y - 1;

end;

Bresenham's Circle Generation Algorithm

Input to the Algorithm

(xc, yc) Integer Centre Co-ordinates and „r" radius.

Output of the Algorithm

Turning ON the pixels on circular boundary.

Data Variables

(xc,yc) : Integer, representing center co-ordinates.

r : Integer, radius of the circle

d : Integer, decision variable.

x,y : Integer, points used for turning ON the pixel.

Algorithm BresenhamCircleGen (xc,yc,r: Integer)

Step 1. Start

Step 2. x=0

Step 3. y = r; { Initial point on the circle}

Step 4. d = 3 - 2 * r; { Initialise decision variable}

Step 5. if (x > y) go to step no 17

Step 6. Plot(xc + x, yc +y);

Step 7. Plot(xc - x, yc + y);

Step 8. Plot(xc + x, yc - y);

Step 9. Plot(xc - x, yc - y);

Step 10.Plot(xc + y, yc + x);

Step 11.Plot(xc - Y,yc + x);

Step 12.Plot(xc +y, yc - x);

Step 13.Plot(xc - Y,yc - x); { Plot eight symmetric points}

Step 14.If (d > = 0) Then

{

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

y = y-1;

}

else

{

d = d + 4 * x + 6;

}

Step 15.x=x+1

Step 16.go to step no 5

Step 17.stop.

```
#include<GL/glut.h>
#include<iostream>
using namespace std;
int r;
void E_way(int x, int y){
 glBegin(GL_POINTS);
 glVertex2i(x+320,y+240);
 glVertex2i(y+320,x+240);
 glVertex2i(y+320, -x+240);
 glVertex2i(x+320, -y+240);
 glVertex2i(-x+320,-y+240);
 glVertex2i(-y+320,-x+240);
 glVertex2i(-y+320,x+240);
 glVertex2i(-x+320,y+240);
 glEnd();
```

```
 glFlush();
}
void B_circle(){
 float d;
 d = 3 - 2*r;
 int x,y;
 x = 0 ;
 y = r ;
 do{
 E_way(x,y);
 if(d<0){
 d=d+4*x+6;
 }
 else{
 d= d+4*(x-y)+10;
 y=y-1;
 }
 x=x+1;
 }while(x<y);
}
void init(){
 glClearColor(1,1,1,0);
 glColor3f(1,0,0);
 gluOrtho2D(0,640,0,480);
 glClear(GL_COLOR_BUFFER_BIT);
}
int main(int argc, char **argv){
 cout<<"\n Enter Radius \t ";
 cin>>r;
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
 glutInitWindowPosition(100,100);
```

```
 glutInitWindowSize(640,480);
 glutCreateWindow("Circle");
 init();
 glutDisplayFunc(B_circle);
 glutMainLoop();
 return 0;
}
```

OUTPUT

g++ filename.cpp -lGL -lGLU -lglut

./a.out