Experiment No. 3

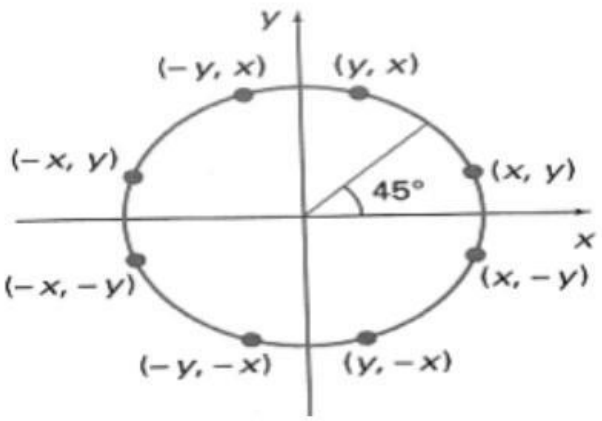
Aim: To implement midpoint circle algorithm.

Objective:

Draw a circle using mid-point circle drawing algorithm by determining the points needed for rasterizing a circle. The mid-point algorithm to calculate all the perimeter points of the circle in the first octant and then print them along with their mirror points in the other octants.

Theory:

The shape of the circle is similar in each quadrant. We can generate the points in one section and the points in other sections can be obtained by considering the symmetry about x-axis and y-axis.



The equation of circle with center at origin is x 2 + y 2 = r 2

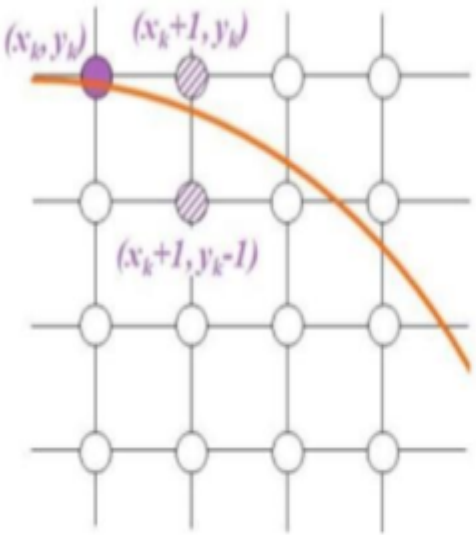
Let the circle function is f circle (x, y) is <

0, if (x, y) is inside circle boundary, is = 0, if

(x, y) is on circle boundary, is > 0, if (x,

y) is outside circle boundary.

Consider the pixel at (xk, yk) is plotted,



Now the next pixel along the circumference of the circle will be either (xk + 1, yk) or (xk + 1, yk – 1) whichever is closer the circle boundary.

Let the decision parameter pk is equal to the circle function evaluate at the mid-point between two pixels.

If pk < 0, the midpoint is inside the circle and the pixel at yk is closer to the circle boundary.

Otherwise, the midpoint is outside or on the circle boundary and the pixel at yk – 1 is closer to the circle boundary.

Algorithm –

Step1: Put x =0, y =r in equation 2
We have p=1-r

Step2: Repeat steps while x ≤ y
Plot (x, y)
If (p<0)
Then set p = p + 2x + 3
Else
p = p + 2(x-y)+5
y =y - 1 (end if)
x =x+1 (end loop)

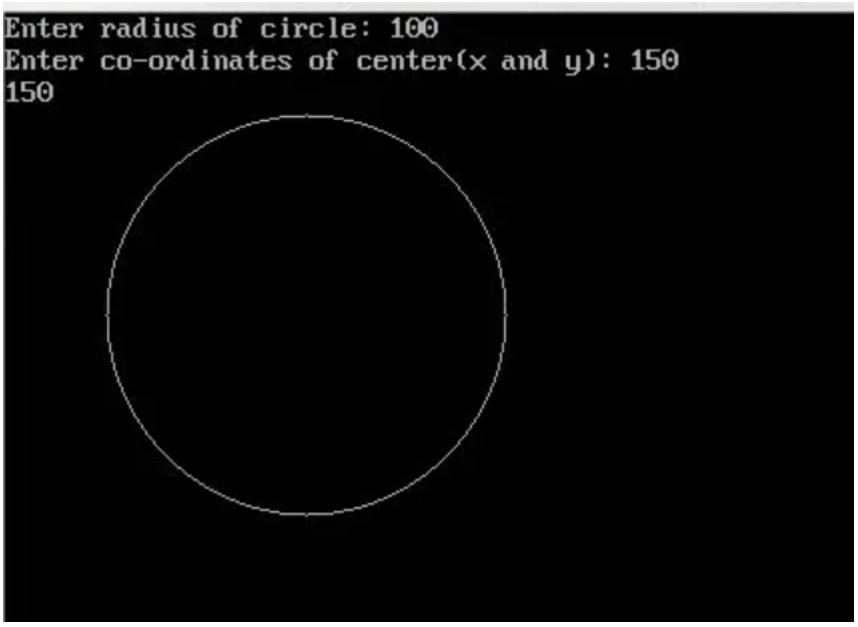
Step3: End

Program –

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void pixel(int x,int y,int xc,int yc)
{
putpixel(x+xc,y+yc,BLUE);
putpixel(x+xc,-y+yc,BLUE);
putpixel(-x+xc,y+yc,BLUE);
putpixel(-x+xc,-y+yc,BLUE);
putpixel(y+xc,x+yc,BLUE);
putpixel(y+xc,-x+yc,BLUE);
putpixel(-y+xc,x+yc,BLUE);
putpixel(-y+xc,-x+yc,BLUE);
}
main()
{
int gd=DETECT,gm=0,r,xc,yc,x,y;
float p;
//detectgraph(&gd,&gm);
initgraph(&gd,&gm," ");
printf("\n Enter the radius of the circle:");
scanf("%d",&r);
printf("\n Enter the center of the circle:");
scanf("%d %d",&xc,&yc);
```

```
y=r;
x=0;
p=(5/4)-r;
while(x<y)
{
        if(p<0)
        {
                x=x+1;
                y=y;
                p=p+2*x+3;
        }
        else
        {
                x=x+1;
                y=y-1;
                p=p+2*x-2*y+5;
        }
        pixel(x,y,xc,yc);
}
getch();
closegraph();
}
```

output –



Conclusion: Comment on

1. Fast or slow

2. Draw one arc only and repeat the process in 8 quadrants

3. Difference with line drawing method

1. The midpoint circle algorithm is considered a fast and efficient method for drawing circles in computer graphics. It uses integer arithmetic and decision parameters to determine the pixels to be plotted, avoiding computationally expensive floating-point calculations. Due to its simplicity and optimized approach, this algorithm operates quickly, making it suitable for devices with limited computational power or real-time applications like gaming, where rapid rendering of circles is necessary. Its efficiency lies in its ability to generate accurate circle shapes with minimal computational overhead, enabling swift rendering of circles on raster-based displays.

2. To draw an arc and repeat the process in all eight quadrants, you can follow the midpoint circle algorithm for each quadrant while ensuring that you only plot the points within the desired angle range. Here's a general outline of how to do this:

A. Initialize the circle drawing process by specifying the center coordinates (x0, y0), the radius (r), and the start and end angles of the arc.
B. For each of the eight quadrants (0-45 degrees, 45-90 degrees, and so on), do the following: a. Set the initial point within the current quadrant, based on the start angle and the current quadrant. For example, if you're working on the first quadrant (0-45 degrees), you might start at (x0 + r, y0).
b. Calculate the decision parameter (P) as described in the midpoint circle algorithm, and proceed to plot the points according to the decision parameter, ensuring that you stay within the current quadrant and the specified angle range.
c. Continue plotting points until the angle exceeds the end angle of the arc or until you complete the entire quadrant.
C. Repeat steps 2 for all eight quadrants, adjusting the initial point and angle range as needed for each quadrant.
By following this approach, you can draw an arc within each quadrant while controlling the start and end angles. This allows you to create a complete circular arc by repeating the process in all eight quadrants, ensuring that only the desired portion of the circle is plotted.

3. The midpoint circle algorithm and line drawing algorithms, such as Bresenham's line algorithm, are fundamental tools in computer graphics, but they serve distinct purposes. The primary difference lies in the shapes they are designed to draw. The midpoint circle algorithm is tailored for efficiently rendering circles or circular shapes. It utilizes integer arithmetic and a decision parameter to plot points along the circumference of a circle, minimizing computational complexity by avoiding costly trigonometric calculations. On the other hand, line drawing algorithms are employed to create straight lines between two endpoints. These algorithms determine which pixels should be part of the line's path, making them conceptually simpler than circle drawing algorithms. In summary, the midpoint circle algorithm is specialized for circular shapes, while line drawing algorithms excel at rendering straight-line segments, both playing crucial roles in various aspects of computer graphics.