Experiment No. 3
Topic : Implement Midpoint Circle Algorithm
Name: Kiran Vishnu Dhuri
Roll Number: 07
Date of Performance:
Date of Submission:

Experiment No. 3

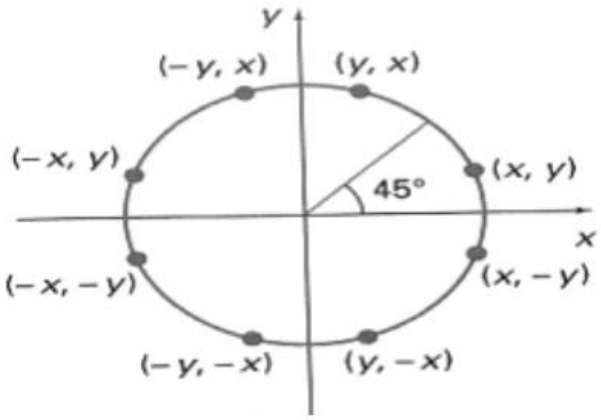
Aim: To implement midpoint circle algorithm.

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.



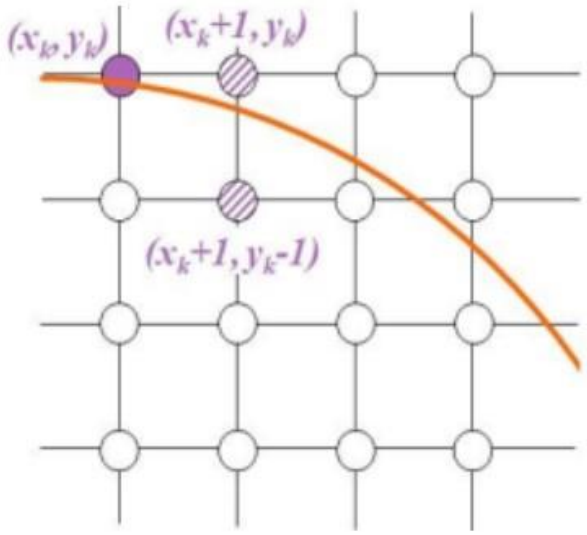
The equation of circle with center at origin is x 2 + y 2 = r 2

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: `Repeatstepswhilex≤y`

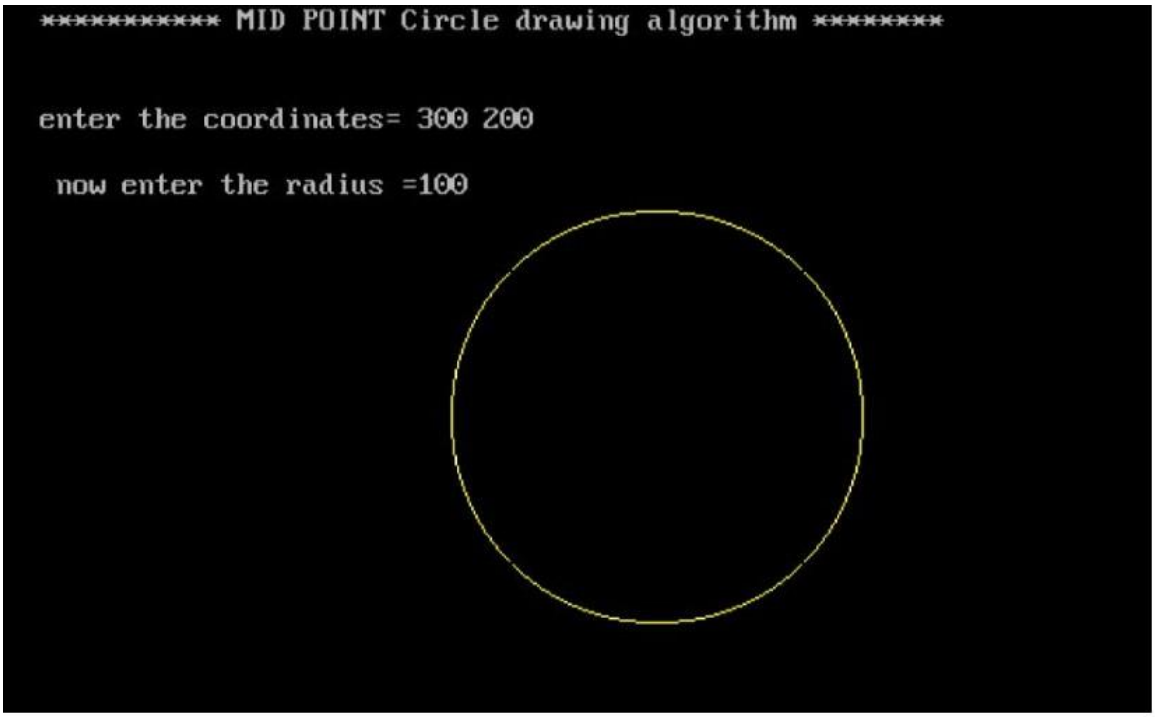
```
    Plot(x,y)
    If(p<0)
Thensetp=p+2x+3
Else
    p=p+2(x−y)+5
    y=y−1(endif)
    x=x+1(endloop)
```

Step3: End

Program –

```
#include<graphics.h>
#include<conio.h>
#include<stdio.h>
void main()
{
int x,y,x_mid,y_mid,radius,dp;
int g_mode,g_driver=DETECT;
clrscr();
initgraph(&g_driver,&g_mode,"C:\\TURBOC3\\BGI");
printf("********** MID POINT Circle drawing algorithm ********\n\n");
printf("\nenter the coordinates= "); scanf("%d %d",&x_mid,&y_mid);
printf("\n now enter the radius =");
scanf("%d",&radius);
x=0; y=radius; dp=1-
radius;
do
{
putpixel(x_mid+x,y_mid+y,YELLOW);
putpixel(x_mid+y,y_mid+x,YELLOW); putpixel(x_mid-
y,y_mid+x,YELLOW); putpixel(x_mid-x,y_mid+y,YELLOW);
putpixel(x_mid-x,y_mid-y,YELLOW); putpixel(x_mid-y,y_mid-
x,YELLOW); putpixel(x_mid+y,y_mid-x,YELLOW);
putpixel(x_mid+x,y_mid-y,YELLOW); if(dp<0) { dp+=(2*x)+1;
} else{
y=y-1;
dp+=(2*x)-(2*y)+1;
}
x=x+1;
}while(y>x);
getch();
}
```

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

Fast or slow: The Midpoint Circle Algorithm is relatively fast compared to other algorithms for drawing circles, such as the Bresenham's circle drawing algorithm. It involves only integer arithmetic operations, making it efficient for implementation on devices with limited computational capabilities. However, it's worth noting that the execution speed may still depend on the hardware and implementation specifics.

Draw one arc only and repeat the process in 8 quadrants: The Midpoint Circle Algorithm indeed draws one-eighth of the circle (either the first or the last 45 degrees, depending on the implementation) and then replicates this in the other octants by exploiting the symmetry of the circle. This approach significantly reduces the computational effort needed to draw a complete circle.

Difference with line drawing method: The Midpoint Circle Algorithm differs from line drawing algorithms, such as Bresenham's line algorithm, primarily in the way it calculates the points on the circumference of the circle. While line drawing algorithms determine which pixels to turn on or off along a straight line, the Midpoint Circle Algorithm focuses on determining the points along the circular path using a midpoint-based approach, considering the symmetry and the incremental computation of the coordinates.