| Experiment No. 3 |
| --- |
| Implement midpoint Circle algorithm. |
| Name: Jaffari Mohammed Ali Sayyed Naqi Ali |
| Roll Number: 16 |
| 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.
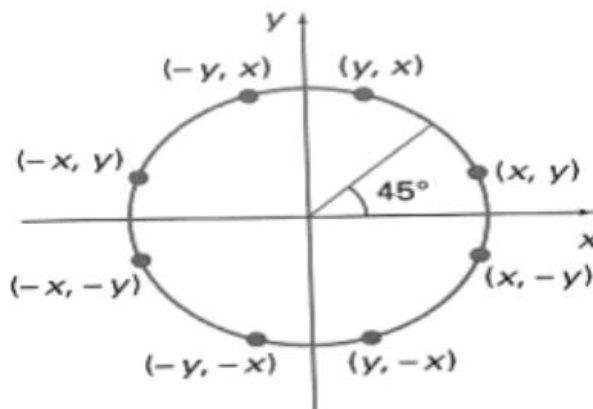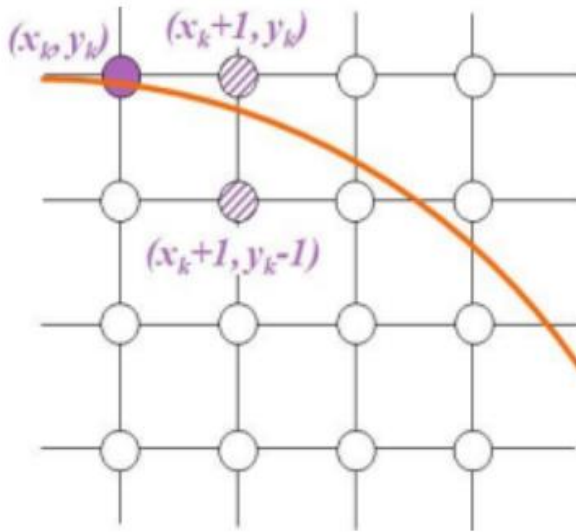

The equation of circle with center at origin is $x^2 + y^2 = r^2$

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 $(x_k + 1, y_k)$ or $(x_k + 1, y_k - 1)$ whichever is closer the circle boundary.

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

If $p_k < 0$, the midpoint is inside the circle and the pixel at y_k is closer to the circle boundary.

Otherwise, the midpoint is outside or on the circle boundary and the pixel at $y_k - 1$ is closer to the circle boundary.

Algorithm – The Midpoint Circle Algorithm is a simple and efficient method for drawing a circle on a pixel grid in computer graphics. It uses the concept of the "midpoint" to determine which pixels should be part of the circle. Here is the step-by-step derivation of the Midpoint Circle Algorithm:

Assumptions:

1. You have a grid of pixels, and each pixel is identified by its coordinates (x, y), where (0,0) is the center of the grid.

Algorithm:

1. Start with the initial point at $(x, y) = (0, r)$, where r is the radius of the circle.

2. Calculate the initial decision parameter: $P = 5/4 - r$ (i.e., $P_0 = 5/4 - r$).

3. Initialize $x = 0$ and $y = r$.

4. At each step, plot the points (x, y), (-x, y), (x, -y), and (-x, -y) to take advantage of the circle's symmetry.

5. Compute the next decision parameter Pk for the next pixel position (xk+1, yk) as follows:

   - If Pk < 0, choose the pixel to the right: xk+1 = xk + 1 and Pk+1 = Pk + 2*xk + 3.

   - If Pk ≥ 0, choose the pixel to the lower-right: xk+1 = xk + 1 and yk+1 = yk - 1, and Pk+1 = Pk + 2*xk - 2*yk + 5.

6. Repeat steps 4 and 5 until x is greater than or equal to y. At this point, you've completed one-eighth of the circle.

7. For each point plotted, reflect it in all eight octants to complete the full circle.

Here's a more detailed explanation:The algorithm starts at the point (0, r), which is chosen because it's on the circle's perimeter, and it's one of the points that minimizes the error when calculating the midpoint. The decision parameter P is initialized as P0 = 5/4 - r.

The algorithm then proceeds by incrementing x and decrementing y while repeatedly calculating the next decision parameter Pk. The choice of the next pixel depends on whether Pk is less than 0 or greater than/equal to 0.

The algorithm continues until x is greater than or equal to y. At this point, one-eighth of the circle is drawn, and the other seven eighths can be generated by reflecting the points in each octant.

The Midpoint Circle Algorithm is efficient because it minimizes the number of calculations and operations needed to draw the circle, making it suitable for use in real-time graphics and situations where performance is important.

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);
```
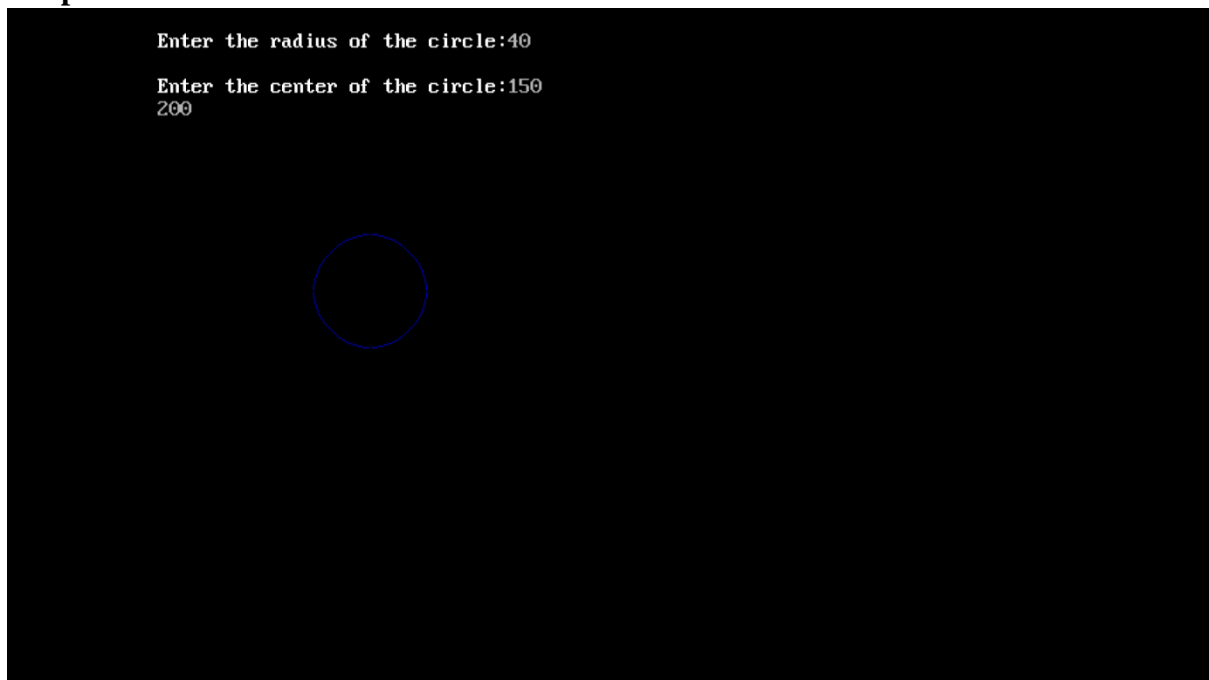
```
putpixel(y+xc,x+yc,BLUE);

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,"C:\\TurboC3\\BGI");

printf("\nEnter the radius of the circle:");

scanf("%d",&r);

printf("\nEnter 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();

return 0;

}
```

Output –

Conclusion:

The Midpoint Circle Algorithm plays a pivotal role in the realm of computer graphics, providing a streamlined and accurate approach to circle rendering. In contrast to line drawing algorithms, it tailors its methodology specifically for circles, capitalizing on their inherent symmetry to enhance efficiency.This algorithm efficiently computes circle points by progressively navigating along the circle's periphery while fine-tuning a decision parameter. Leveraging this symmetry considerably lessens the computational burden when compared to line drawing methods, which must accommodate various angles and slopes.The algorithm follows a systematic process by iteratively identifying points within each octant and then mirroring them to complete the entire circle. Its efficiency is quite remarkable, boasting a time complexity of O(n) for circle drawing, where 'n' symbolizes the radius.