

## Lab Report : 02



**Title: Computer Graphics Lab**  
**Course code: CSE-304**  
**3rd Year 1st Semester**

**Date of Submission: 9.07.2023**

**Submitted to-**

<p><b>Prof. Dr. Mohammad Shorif Uddin</b> <i>Professor</i> <i>Department of Computer</i> <i>Science and Engineering</i> <i>Jahangirnagar University</i> <i>Savar, Dhaka-1342</i></p>	<p><b>Dr. Morium Akther</b> <i>Associate Professor</i> <i>Department of Computer</i> <i>Science and Engineering</i> <i>Jahangirnagar University</i> <i>Savar, Dhaka-1342</i></p>
--	--

Sl	Class Roll	Registration Number	Name
01	388	20200650758	Md.Tanvir Hossain Saon

## Experiment No.05

**Name Of the Experiment:** Scan Conversion of a circle using Mid Point Algorithm.

### Introduction:

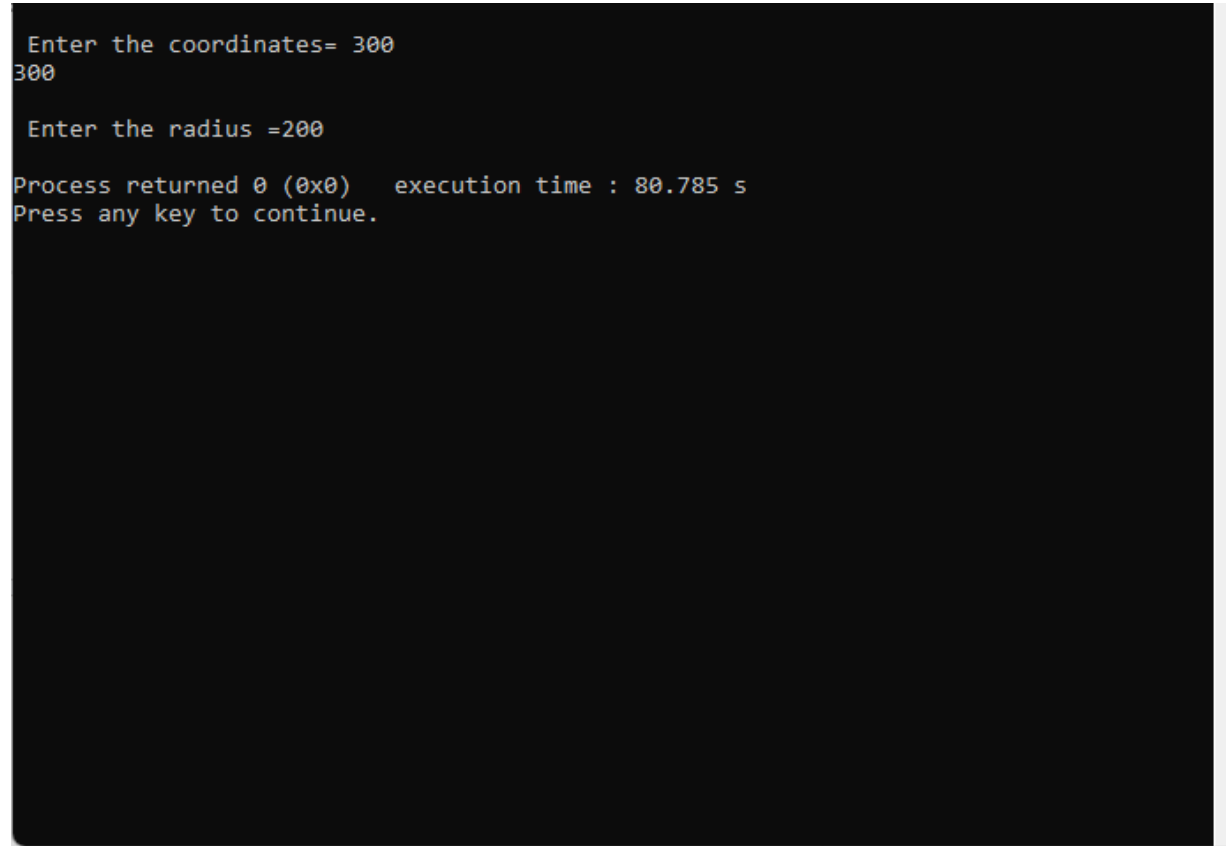
Scan conversion is the process of converting a geometric object, such as a circle, from its mathematical representation into a discrete pixel-based representation on a computer screen or display. The midpoint algorithm, also known as Bresenham's circle algorithm, is a popular method for scan converting circles efficiently. In this discussion, we will explore the scan conversion of a circle using the midpoint algorithm.

### Source Code:

```
#include<graphics.>
#include<conio.h>
#include<stdio.h>
int 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("\n Enter the coordinates= ");
    scanf("%d %d",&x_mid,&y_mid);
    printf("\n Enter the radius =");
    scanf("%d",&radius);
    x=0;
    y=radius;
    dp=1-radius;
    do
    {
        putpixel(x_mid+x,y_mid+y,GREEN);
        putpixel(x_mid+y,y_mid+x,GREEN);
        putpixel(x_mid-y,y_mid+x,GREEN);
        putpixel(x_mid-x,y_mid+y,GREEN);
        putpixel(x_mid-x,y_mid-y,GREEN);
        putpixel(x_mid-y,y_mid-x,GREEN);
        putpixel(x_mid+y,y_mid-x,GREEN);
        putpixel(x_mid+x,y_mid-y,GREEN);
        if(dp<0)
        {
            dp+=(2*x)+1;
        }
    }
```

```
        else
        {
            y=y-1;
            dp+=(2*x) - (2*y) ;
        }
        x=x+1;
    }
    while(y>x);
    getch();
}
```

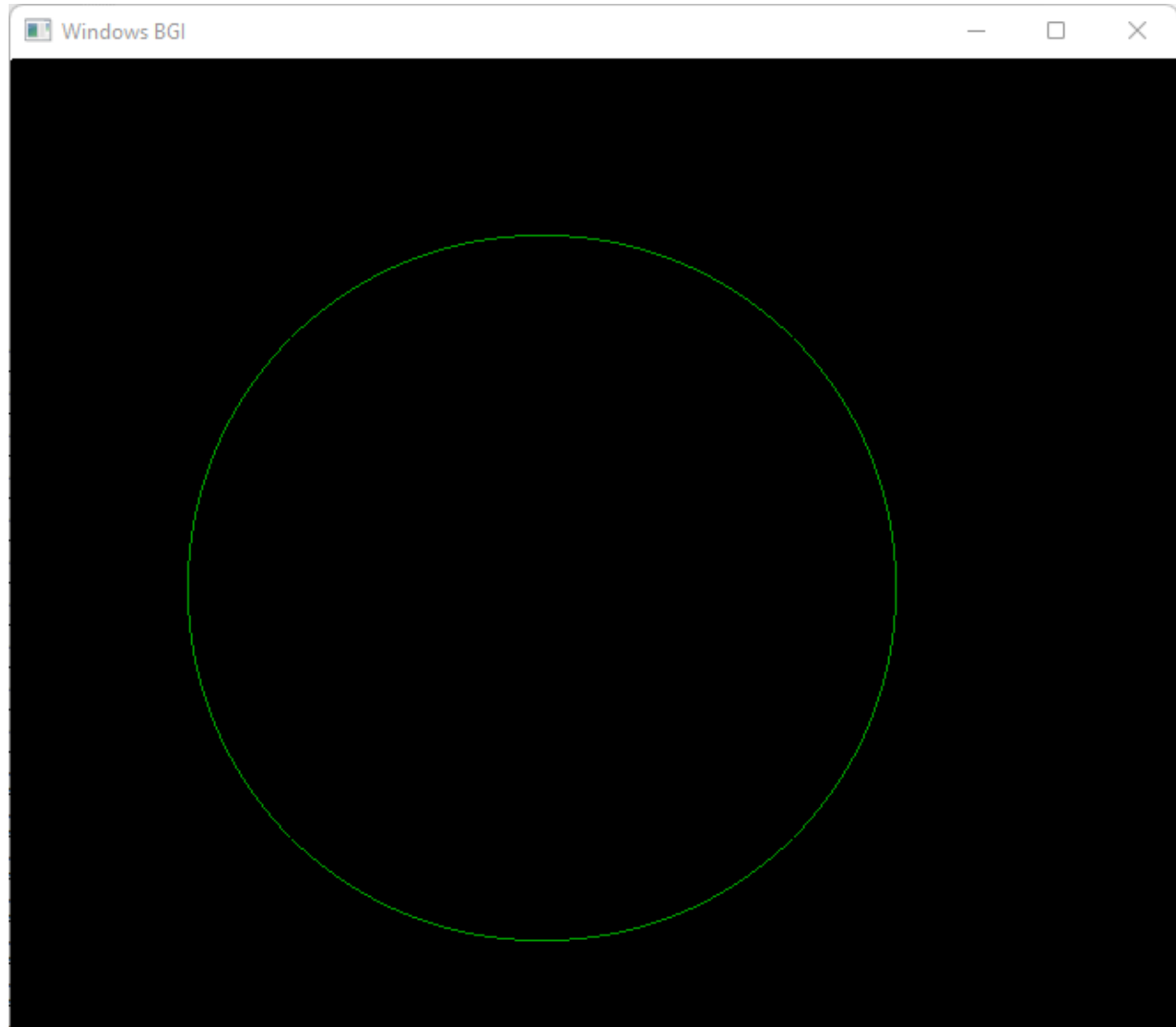
## OUTPUT:

A screenshot of a terminal window with a black background and white text. The text shows the program's execution flow: it prompts for coordinates, receives the input '300', prompts for a radius, receives '200', and then displays the execution time and a message to press a key to continue.

```
Enter the coordinates= 300
300

Enter the radius =200

Process returned 0 (0x0)   execution time : 80.785 s
Press any key to continue.
```



### **Discussion:**

The midpoint algorithm for circle scan conversion determines the pixels to be activated or filled in order to approximate a circle on a pixel grid. The algorithm takes advantage of the symmetry of a circle to minimize the number of calculations needed. To begin the scan conversion of a circle using the midpoint algorithm, we need the following inputs:

- Center coordinates  $(h, k)$  of the circle
- Radius  $r$  of the circle

The midpoint algorithm iteratively determines which pixels to activate, starting from the topmost point of the circle and moving clockwise. The basic steps of the algorithm are as follows:

1. Initialize the variables:

- Set  $x = 0$  and  $y = r$  (starting point at the top of the circle).
- Compute the initial decision parameter,  $P = 1 - r$ .

2. Loop until  $x \leq y$ :

- At each iteration, activate the pixels  $(h + x, k + y)$ ,  $(h - x, k + y)$ ,  $(h + x, k - y)$ , and  $(h - x, k - y)$  to draw the four symmetric points of the circle.

- If the decision parameter  $P < 0$ , update  $P$  and  $x$ :

$$- P := P + 2x + 1$$

$$- x := x + 1$$

- If the decision parameter  $P \geq 0$ , update  $P$ ,  $x$ , and  $y$ :

$$- P := P + 2x - 2y + 1$$

$$- x := x + 1$$

$$- y := y - 1$$

By repeating these steps until  $x > y$ , we will have scan converted the circle using the midpoint algorithm. The algorithm efficiently determines the pixels to activate by utilizing the symmetry of the circle and only calculating the decision parameter based on incremental changes in  $x$  and  $y$ .

It's important to note that the midpoint algorithm assumes the circle is centered at the origin  $(0, 0)$ . If the circle is not centered at the origin, you can translate the circle to the desired center point before applying the algorithm.

In conclusion, the midpoint algorithm is an efficient method for scan converting circles. By following the steps outlined above and activating the appropriate pixels, we can approximate a circle on a pixel grid using the midpoint algorithm.

## Experiment No.06

### Name of the Experiment:

Scan and conversion of Ellipse.

### Introduction:

Scan conversion refers to the process of converting a geometric object, such as an ellipse, from its mathematical representation into a discrete pixel-based representation on a computer screen or display. The scan conversion of an ellipse involves determining which pixels to activate or fill to approximate the shape of the ellipse. There are several algorithms available for scan converting ellipses, including the midpoint algorithm and the polar coordinate algorithm. In this discussion, we will explore the scan conversion of an ellipse using the midpoint algorithm.

### Source Code:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
#include<bits/stdc++.h>
using namespace std;
void disp();
float x,y;
int xc,yc;
int main()
{
    int gd=DETECT,gm,a,b;
    float p1,p2;
    //clrscr();
    initgraph(&gd,&gm,"c:\\turboc3\\bgi");
    printf("*** Ellipse Generating Algorithm\n");
    printf("Enter the value of Xc\t");
    scanf("%d",&xc);
    printf("Enter the value of yc\t");
    scanf("%d",&yc);
    printf("Enter X axis length\t");
    scanf("%d",&a);
    printf("Enter Y axis length\t");
    scanf("%d",&b);
    x=0;
    y=b;
    disp();
    p1=(b*b)-(a*a*y*y)+(a*a)/4;
```

```

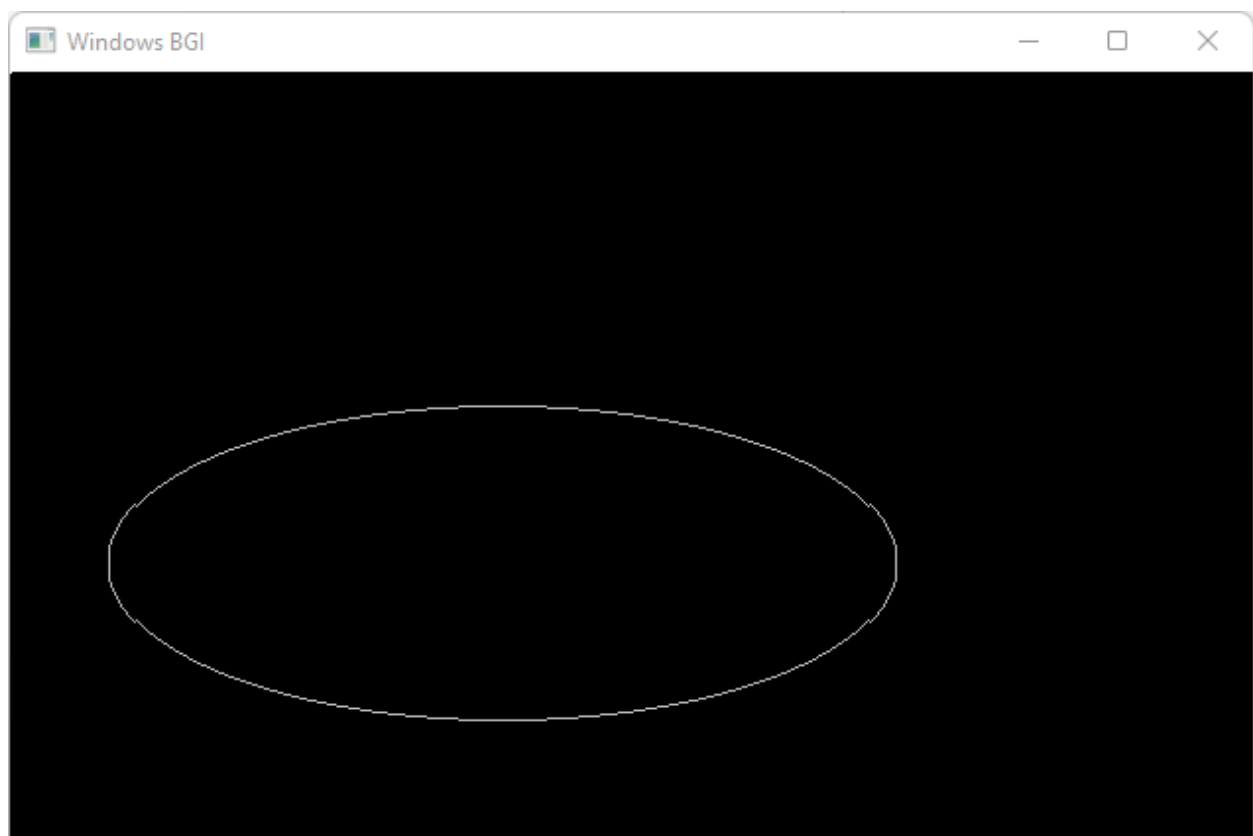
while((2.0*b*b*x)<=(2.0*a*a*y))
{
    x++;
    if(p1<=0)
        p1=p1+(2.0*b*b*x)+(b*b);
    else
    {
        y--;
        p1=p1+(2.0*b*b*x)+(b*b)-(2.0*a*a*y);
    }
    disp();
    x=-x;
    disp();
    x=-x;
    delay(50);
}
x=a
;
y=0
;
disp();
p2=(a*a)+2.0*(b*b*a)+(b*b)/4;
while((2.0*b*b*x)>(2.0*a*a*y))
{
    y++;
    if(p2>0)
        p2=p2+(a*a)-(2.0*a*a*y);
    else
    {
        x--;
        p2=p2+(2.0*b*b*x)-(2.0*a*a*y)+(a*a);
    }
    disp();
    y=-y;
    disp();
    y=-y;
    delay(50);
}
getch();
closegraph();
}
void disp()
{
    putpixel(xc+x,yc+y,7);
}

```

```
    putpixel(xc-x,yc+y,7);  
    putpixel(xc+x,yc-y,7);  
    putpixel(xc+x,yc+y,7);  
}
```

### OUTPUT:

```
*** Ellipse Generating Algorithm ***  
Enter the value of Xc    250  
Enter the value of yc    250  
Enter X axis length      200  
Enter Y axis length      80
```



### Discussion:

The midpoint algorithm for ellipse scan conversion is an extension of the midpoint algorithm used for circle scan conversion. It determines the pixels to be activated or filled to approximate an ellipse on a pixel grid. The algorithm takes advantage of the symmetry of an ellipse to minimize the number of calculations needed.



To begin the scan conversion of an ellipse using the midpoint algorithm, we need the following inputs:

- Center coordinates (h, k) of the ellipse
- Semi-major axis a of the ellipse
- Semi-minor axis b of the ellipse

The midpoint algorithm iteratively determines which pixels to activate, starting from a specific point on the ellipse and moving in a clockwise direction. The basic steps of the algorithm are as follows:

1. Initialize the variables:

- Set  $x = 0$  and  $y = b$  (starting point at the topmost point of the ellipse).
- Compute the initial decision parameter,  $P = b^2 - a^2 * b + 0.25 * a^2$ .

2. Loop until  $2 * a^2 * y \leq 2 * b^2 * x$ :

- At each iteration, activate the pixels  $(h + x, k + y)$ ,  $(h - x, k + y)$ ,  $(h + x, k - y)$ , and  $(h - x, k - y)$  to draw the four symmetric points of the ellipse.

- If the decision parameter  $P < 0$ , update P and x:

$$- P := P + b^2 * (2 * x + 3)$$

$$- x := x + 1$$

- If the decision parameter  $P \geq 0$ , update P, x, and y:

$$- P := P + b^2 * (2 * x + 3) + a^2 * (2 - 2 * y)$$

$$- x := x + 1$$

$$- y := y - 1$$

3. Repeat steps 2, but this time loop until  $y = 0$ :

- Activate the pixels  $(h + x, k + y)$ ,  $(h - x, k + y)$ ,  $(h + x, k - y)$ , and  $(h - x, k - y)$  for the four symmetric points of the ellipse.

By repeating these steps until  $y = 0$ , we will have scan converted the ellipse using the midpoint algorithm. The algorithm efficiently determines the pixels to activate by utilizing the symmetry of the ellipse and only calculating the decision parameter based on incremental changes in x and y.

It's important to note that the midpoint algorithm assumes the ellipse is centered at the origin (0, 0). If the ellipse is not centered at the origin, We can translate the ellipse to the desired center point before applying the algorithm. In conclusion, the midpoint algorithm is a commonly used method for scan converting ellipses.