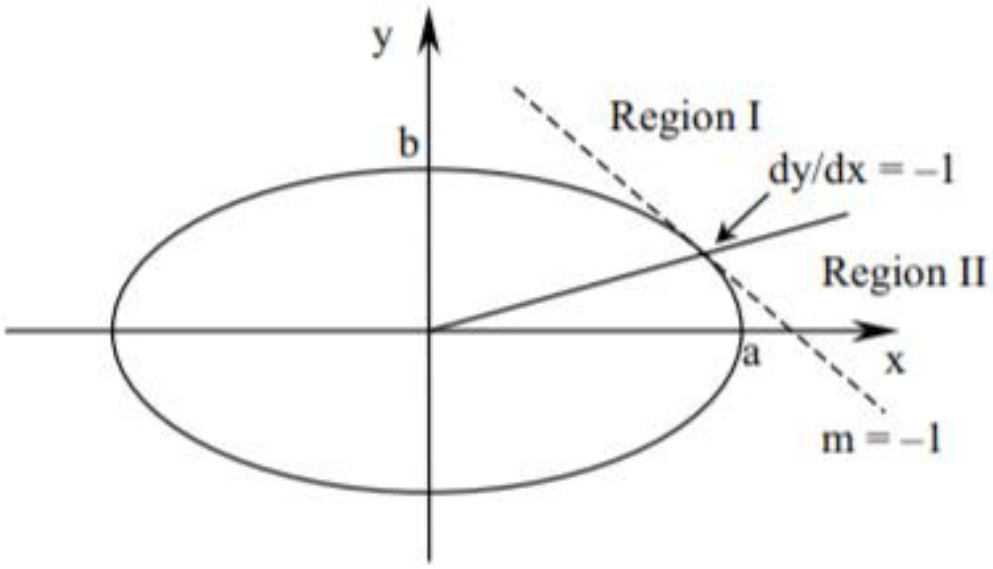
Experiment 3

Aim

To generate an ellipse using Mid-Point Ellipse Drawing Algorithm

Description/Theory

Midpoint ellipse algorithm is a method for drawing ellipses in computer graphics. This method is modified from Bresenham's algorithm. The advantage of this modified method is that only addition operations are required in the program loops. This leads to simple and fast implementation in all processors.



Algorithm

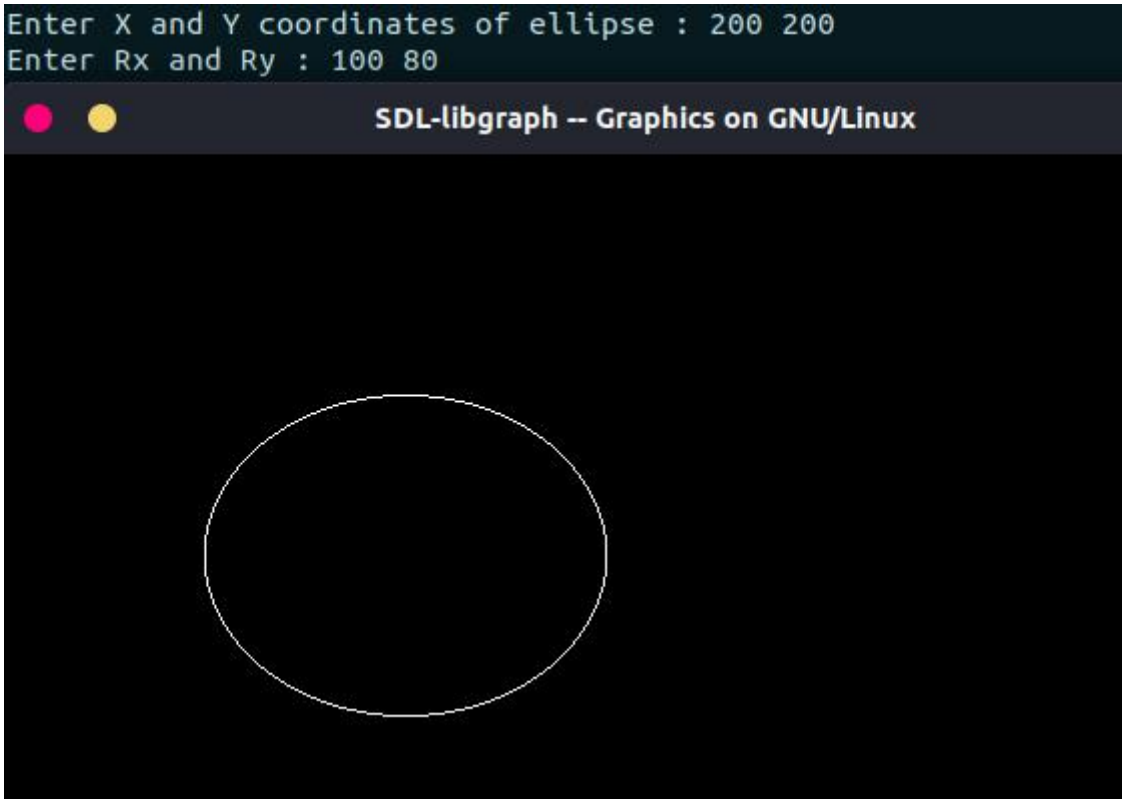
- Divide the quadrant into two regions n the boundary of two regions is the point at which the curve has a slope of -1.
- Process by taking unit steps in the x direction to the point P, then taking unit steps in the y direction.
- Apply midpoint algorithm.
- $f_{ellipse}(x,y) = r_y^2x^2 + r_x^2y^2 – r_x^2r_y^2$. The expression is:
  - < 0 inside the ellipse boundary
  - = 0 on the ellipse boundary
  - > 0 outside the ellipse boundary
- $p1_k = f_{ellipse}(x_k + 1, y_k – 1/2)$
  $p2_k = f_{ellipse}(x_k + 1/2, y_k – 1)$

Code

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>

void ellipse(int xc,int yc,int rx,int ry)
{
  int gm=DETECT,gd;
  int x, y, p;
  initgraph(&gm,&gd,"");
  x=0;
  y=ry;
  p=(ry*ry)-(rx*rx*ry)+((rx*rx)/4);
```

```
  while((2*x*ry*ry)<(2*y*rx*rx))
  {
       putpixel(xc+x,yc-y,WHITE);
       putpixel(xc-x,yc+y,WHITE);
       putpixel(xc+x,yc+y,WHITE);
       putpixel(xc-x,yc-y,WHITE);
       if(p<0)
       {
                  x=x+1;
                  p=p+(2*ry*ry*x)+(ry*ry);
       }
       else
       {
                  x=x+1;
                  y=y-1;
                  p=p+(2*ry*ry*x+ry*ry)-(2*rx*rx*y);
       }
  }
  p=((float)x+0.5)*((float)x+0.5)*ry*ry+(y-1)*(y-1)*rx*rx-rx*rx*ry*ry;
  while(y>=0)
  {
       putpixel(xc+x,yc-y,WHITE);
       putpixel(xc-x,yc+y,WHITE);
       putpixel(xc+x,yc+y,WHITE);
       putpixel(xc-x,yc-y,WHITE);
       if(p>0)
       {
         y=y-1;
                  p=p-(2*rx*rx*y)+(rx*rx);
       }
       else
       {
                  y=y-1;
                  x=x+1;
                  p=p+(2*ry*ry*x)-(2*rx*rx*y)-(rx*rx);
       }
  }
  getch();
  closegraph();
}

int main()
{
   int xc,yc,rx,ry;
   printf("Enter the coordinates of the centre of the ellipse: ");
   scanf("%d %d",&xc,&yc);
   printf("Enter the x-radius and y-radius: ");
   scanf("%d %d",&rx,&ry);
   ellipse(xc,yc,rx,ry);
   getch();
   return 0;
}
```

Result



Discussion

The output in the image above shows the drawing of ellipse which is drawn by using Mid-Point Ellispe Drawing Algorithm.

Finding and learning

From the above output, we observe that we can use symmetry of ellipse to draw an ellipse in Computer Graphics. Here, we calculated pixel points for one quadrant and used those points to draw the whole figure. That is, suppose the calculated pixel point is (x, y), then pixel point in other quadrants will be (-x, y), (x, -y) and (-x, -y).

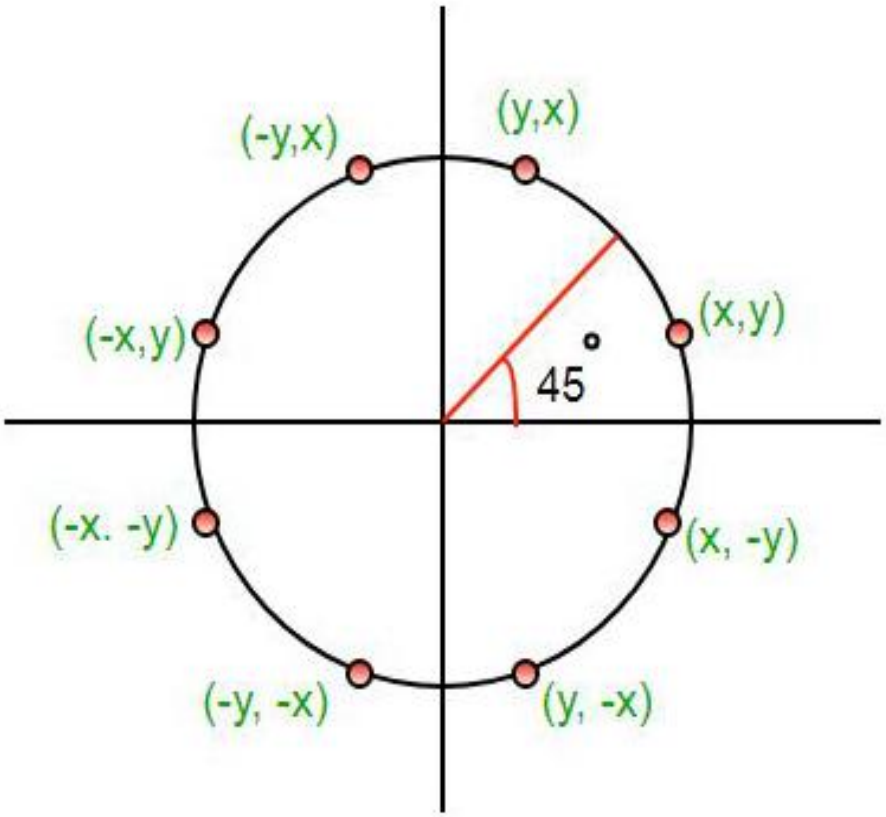
Experiment 4

Aim

To generate a circle using Mid-Point Circle Drawing Algorithm

Description/Theory

We need to plot the perimeter points of a circle whose centre co-ordinates and radius are given using the Mid-Point Circle Drawing Algorithm.
We use the above 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. This will work only because a circle is symmetric about its centre.



The algorithm is very similar to the Mid-Point Line Generation Algorithm. Here, only the boundary condition is different.

Algorithm

- Input radius **r** and circle centre $(x_c, y_c)(x_c, y_c)$ and obtain the first point on the circumference of the circle centred on the origin as

$$(x_0, y_0) = (0, r)$$

- Calculate the initial value of decision parameter as

$$P_0 = 5/4 - r$$

- At each XKXK position starting at K=0, perform the following test:-

  If $P_K < 0$ then next point on circle (0,0) is (X_{K+1},Y_K) and

  $$P_{K+1} = P_K + 2X_{K+1} + 1$$

  Else

  $$P_{K+1} = P_K + 2X_{K+1} + 1 - 2Y_{K+1}$$

  Where, $2X_{K+1} = 2X_{K+2}$ and $2Y_{K+1} = 2Y_{K-2}$.

- Determine the symmetry points in other seven octants.

- Move each calculate pixel position (X, Y) onto the circular path centred on (X_C, Yc) and plot the coordinate values.

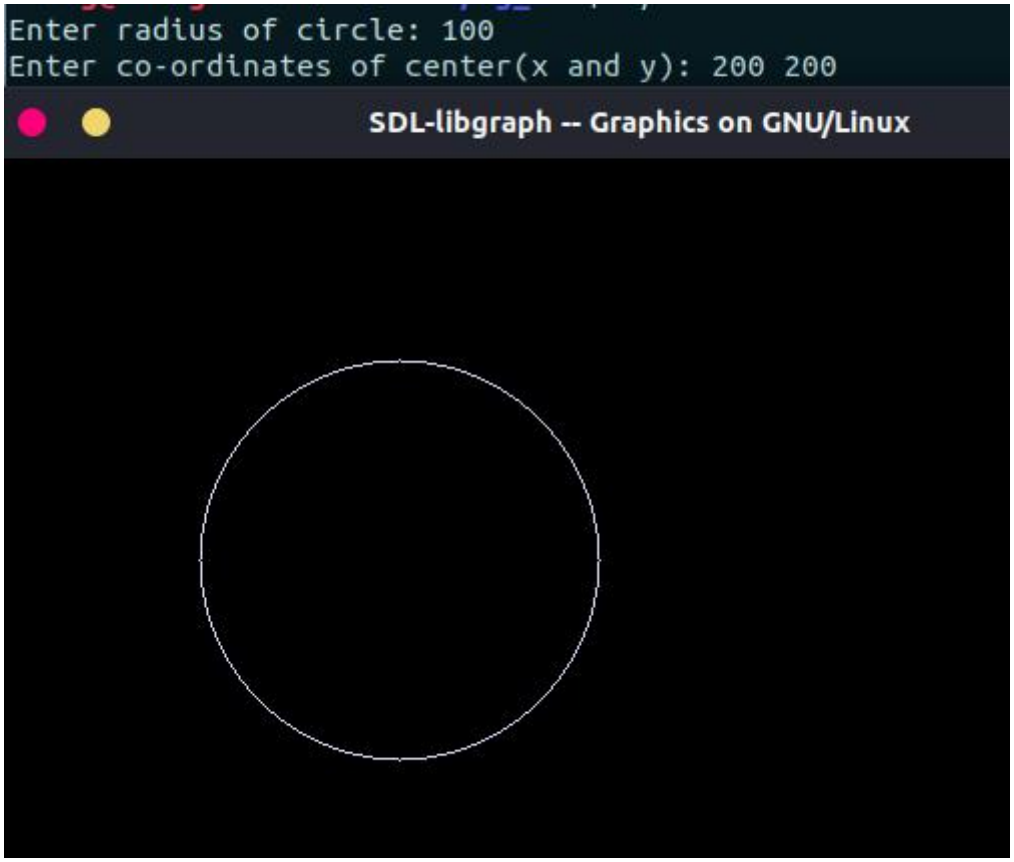
$$X = X + X_C, \quad Y = Y + Y_C$$

- Repeat step-3 through 5 until X >= Y.

Code

```
#include<stdio.h>
#include<graphics.h>

void drawcircle(int x0, int y0, int radius)
{
   int x = radius;
   int y = 0;
   int err = 0;

   while (x >= y)
   {
   putpixel(x0 + x, y0 + y, 7);
   putpixel(x0 + y, y0 + x, 7);
   putpixel(x0 - y, y0 + x, 7);
   putpixel(x0 - x, y0 + y, 7);
   putpixel(x0 - x, y0 - y, 7);
   putpixel(x0 - y, y0 - x, 7);
   putpixel(x0 + y, y0 - x, 7);
   putpixel(x0 + x, y0 - y, 7);
         delay(50);
   if (err <= 0)
   {
      y += 1;
      err += 2*y + 1;
   }

   if (err > 0)
   {
      x -= 1;
      err -= 2*x + 1;
   }
   }
}

int main()
{
   int gdriver=DETECT, gmode, error, x, y, r;
   printf("Enter radius of circle: ");
   scanf("%d", &r);
   printf("Enter co-ordinates of center(x and y): ");
   scanf("%d%d", &x, &y);
   initgraph(&gdriver, &gmode, "");
   drawcircle(x, y, r);
   getch();
   return 0;
}
```

Result



Discussion

From the above code and output, we can observe that we can draw more points, and eventually, the complete circle using the property of circle being symmetric about the centre.

Finding and learning

We use the symmetricity of a circle to get the points in other octants. If there is a situation when the centre of the circle is not at origin, we can use the concept of “Shifting of Origin” to get the required outcome.