LAB 5

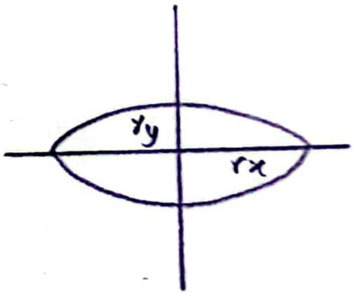
Mid-Point Ellipse algorithm

Objective:

To implement the mid-point ellipse algorithm to draw ellipse.

Theory

In an ellipse, we use mid-point algorithm for two regions ie region 1 and region 2. And later use the symmetry to find all the other remaining coordinates.



$P_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$ [Region 1]

if $(P_0 < 0)$ => $x_{k+1} = x_k + 1$

$y_{k+1} = y_k$

$P_{k+1} = P_k + 2r_y^2 x_{k+1} + r_y^2$

Else => $x_{k+1} = x_k + 1$

$y_{k+1} = y_k + 1$

$P_{k+1} = P_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$

continue until $\boxed{2r_y^2 x \geq 2r_x^2 y}$

$P_0 = r_y^2 (x_0 + \frac{1}{2})^2 + r_y^2 (y_0 - 1)^2 - r_x^2 r_y^2$ [region 2]

if $(P_0 > 0)$ => $x_{k+1} = x_k$

$y_{k+1} = y_k - 1$

$P_{k+1} = P_k - 2r_x^2 y_{k+1} + r_x^2$

else => $x_{k+1} = x_k + 1$

$y_{k+1} = y_k - 1$

$P_{k+1} = P_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$

continue untile $\boxed{y = 0}$

The use symmetry logic to other 3 ~~points.~~ quadrants of each region.

Algorithm

1) Input r_x, r_y and centre (x_c, y_c) and obtain the first point on ellipse centered at origin as

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

2) Calculate the initial value of decision parameter in region 1 as $P_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$

3) At each step x_k position in region 1, starting at $k=0$, perform following.

If $(P_k < 0) \Rightarrow (x_k+1, y_k)$ and $P_{k+1} = P_k + 2r_y^2 x_{k+1} + r_y^2$

else $\Rightarrow (x_{k+1}, y_k - 1)$ and $P_{k+1} = P_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$

with $2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2$ and $2r_x^2 y_{k+1} = 2r_x^2 y_k + 2r_x^2$

and continue until $2r_y^2 x \geq 2r_x^2 y$

4) Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as

$$P_0 = r_y^2(x_0 + \frac{1}{2})^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$$

5) At each step y_k position in region 2, starting at $k=0$, perform following.

If $(P_k > 0) \Rightarrow (x_k, y_k - 1)$ and $P_{k+1} = P_k - 2r_x^2 y_{k+1} + r_x^2$

else $\Rightarrow (x_k+1, y_k-1)$ and $P_{k+1} = P_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$

continue until $y = 0$

6) For both regions, determine symmetry points in the other three quadrants.

7) Move each calculated position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values.

$$x = x + x_c$$

$$y = y + y_c$$

Source code

```
#include <graphics.h>
#include <math.h>
#include <iostream>
using namespace std;
int main()
{
  int gd = DETECT, gm;
  initgraph(&gd, &gm, (char*)" ");
  int Xr, Yr, x1, y1, p, k=0;
  cout << "Enter the x-radius of ellipse: ";
   cin >> Xr;
  cout << "Enter the y-radius of ellipse: ";
  cin >> Yr;
  cout << "Enter the centre coordinates of ellipse: ";
  cin >> x1 >> y1;
  P = pow(Yr, 2) - pow(Xr, 2)* Yr + 1/4 * pow(Xr, 2);
  int x=0, y=Yr;
  while (2*Yr*Yr*x < 2*Xr*Xr*y)
  {
    putpixel(x+x1, y+y1, 1);
    putpixel(-x+x1, y+y1, 2);
    putpixel(x+x1, -y+y1, 3);
    putpixel(-x+x1, -y+y1, 4);
    if (P<0)
    {
      x=x+1;
      P=P+2*pow(Yr, 2)*x + pow(Yr, 2);
    }
    else
    { x= x+1;
      y=y-1;
      P=P+2*pow(Yr, 2)*x - 2*pow(Xr, 2)*y + pow(Yr, 2);
    }
    delay(50);
  P= Yr*Yr*(x+1/2)*(x+1/2) + Xr*Xr*(y-1)*(y-1) -
        Xr*Xr*Yr*Yr ;
while (y>=0)
{  putpixel (x+x1, y+y1 ,2);
   putpixel (-x+x1, y+y1 , 2);
   putpixel (x+x1 , -y+y1, 3);
   putpixel (-x+x1, -y+y1, 4);
   if (p>0)
   {  y=y-1;
      p=p-2*pow(xr, 2)*y + pow(xr, 2);
   }
   else
   {
     x=x+1;
     y=y-1;
     p=p+2*pow(yr, 2)* x - 2* pow(xr, 2)* y + pow(xr, 2);
   }
   delay (50);
}
getch ();
closegraph();
}
```

Output

Discussion and Conclusion

We implemented the mid-point ellipse drawing algorithm and wrote a c++ graphics program and tested a bunch of coordinates to draw multiple ellipse.

This has helped us to strengthen our knowledge on mid-point ellipse drawing algorithm.

LAB 6

Boundary Fill Algorithm

Objective:

To implement the boundary fill algorithm and fill polygon shapes.

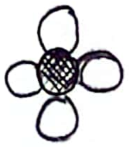
Theory:

Boundaryfill algorithm is a seed fill algorithm meaning it requires a point of the polygon shape to be filled. Its parameters are p(x, y) points, fill-color and boundary-color.

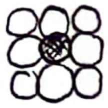
Starting from point p(x, y). test is performed to determine whether the neighbouring pixel is already filled or boundary is reached. If not then boundary pixels are filled will fill color and their neighbours are tested. This process is repeated till boundary is reached.

Two types.

- 4 connected
- 8 connected.



4-connected



8-connected

Note: This algorithm is used only when the boundary is of single color.

Algorithm

```
void boundaryFill(int x, int y, int bcolor, int fcolor)
{
 int value = getpixel(x, y);
  if(value!=bcolor && value!=fcolor)
  { putpixel(x, y, fcolor);
    boundaryFill(x-1, y, bcolor, fcolor)
    boundaryFill(x+1, y, bcolor, fcolor)
    boundaryFill(x, y-1, bcolor, fcolor)
    boundaryFill(x, y+1, bcolor, fcolor)
    // 8 point starts here
    boundaryFill( x-1, y-1, bcolor, fcolor)
    boundaryFill( x-1, y+1, bcolor, fcolor)
    boundaryFill( x+1, y-1, bcolor, fcolor)
    boundaryFill( x+1, y+1, bcolor, fcolor)
  }
}
```

CS CamScanner

Source code

```
#include <graphics.h>
#include <iostream>
#include <dos.h>
Using namespace std;
void boundaryFill (int x, int y, int fill_color, int b_color)
{
 if (getpixel(x,y) != b_color && getpixel(x,y) != fill_color)
 {
   putpixel(x, y, fill_color);
   boundaryFill (x+1, y, fill_color, bcolor);
   boundaryFill (x-1, y, fill_color, bcolor);
   boundaryFill (x, y+1, fill_color, bcolor);
   boundaryFill (y, y-1, fill_color, bcolor);
   // 8 points
   boundaryFill (x+1, y+1, fill_color, b_color);
   boundaryFill (x+1, y-1, fill_color, b_color);
   boundaryFill (x-1, y+1, fill_color, b_color);
   boundaryFill (x-1, y-1, fill_color, b_color);
 }
}
int main()
{
  int gd = DETECT, gm;
  initgraph(&gd, gm, " ");
  setcolor(15);
  rectangle(100, 100, 300, 300);
  boundaryFill(220, 220, 4, 15);
  getch();
  closegraph();
}
```

Output

Disiussion and Conclusion

We implemented the boundory fill algorithm and wrote a C++ graphics program and tested a bunch of polygon shapes and filled them using the algerithm.

This has helped us to strengthen our knowledge on boundory fill algorithm.

CamScanner