Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

LAB-III

[Course : COMP 342]

Submitted by

Sudip Bhattarai

Roll no:11

Submitted to

Mr.Dhiraj Shrestha
Department of Computer Science and Engineering

May 17, 2024,

Using Mid-Point Algorithm

- To draw a circle:

Algorithm:

1. Input radius r and circle center (X_C, Y_C) and obtain the first point on the circumference of a circle centered on the origin as

   $(X_0, Y_0)=(0,r)$

2. Calculate the initial value of the decision parameter as

   $Po=5/4-1$

   $Po=1-r$

3. At each X_k position, starting at k = 0, perform the following test:

   If $Pk< 0$

   $X_{k+1} = X_k+1$

   $Y_{k+1} = Yк$

   $P_{k+1} = P_k+ 2X_{k+1}+ 1$

   Else

   $X_{k+1} = X_k+1$

   $Y_{k+1} = Yк-1$

   $P_{k+1} = P_k + 2X_{k+1}+ 1-2Y_{k+1}$

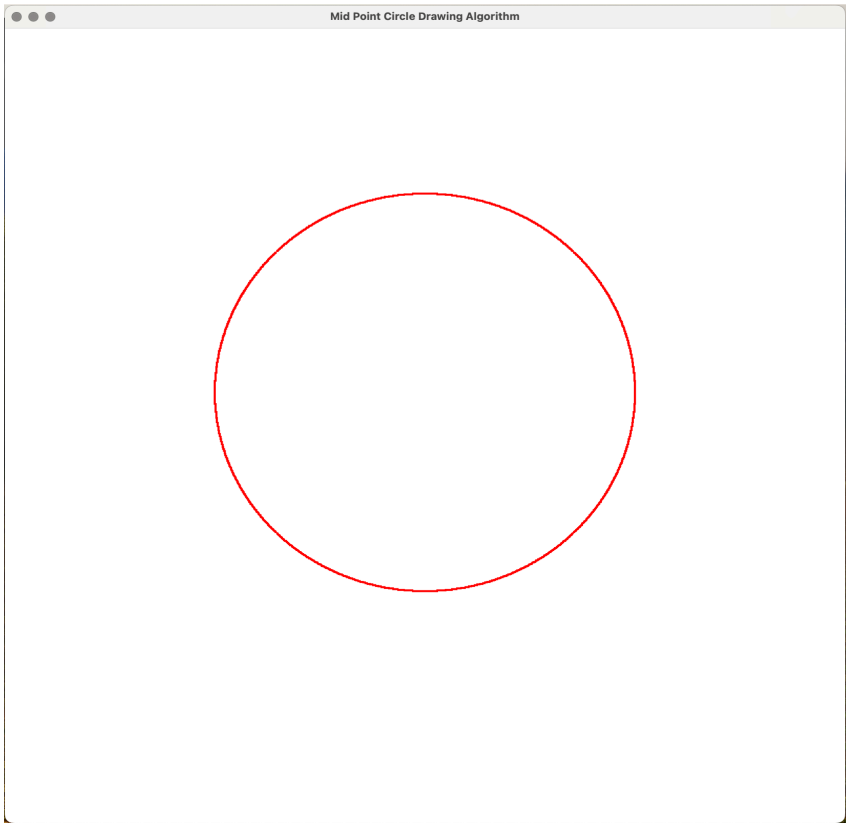
4. Determine the symmetry points in the other seven octants.
5. Move each calculated pixel position (x,y) onto the circular path centered on (X_C, Y_C) plot the co-ordinate values
6. Repeat steps 3 through 5 until $x \geq y$

Code Implementation:

```
def drawCircle(radius,x_center,y_center):
    x=0;
    points=[]
    y=radius;
    p=1-radius;
    while x<y:
        points.extend(calculateSymmetricPoints(x,y,x_center,y_center))
        if(p<0):
            x=x+1;
            p=p+2*x+1;
        else :
            x=x+1;
            y=y-1;
            p=p+2*x+1-2*y
    return points

def calculateSymmetricPoints(x,y,x_center,y_center):
    symmetric_coordinates = [
        (x + x_center, y + y_center), (y + x_center, x + y_center),
        (-y + x_center, x + y_center), (-x + x_center, y + y_center),
        (-x + x_center, -y + y_center), (-y + x_center, -x +
y_center(y + x_center, -x + y_center), (x + x_center, -y + y_center)
    ]
    return symmetric_coordinates
```

Output:



- To draw Ellipse

Algorithm:

1. Input r_x, r_y and the ellipse center (X_C, Y_C) and obtain the first point on an ellipse centered on origin as

   $(X_0, Y_0)=(0, r_y)$

2. Calculate the initial value of the decision parameter in region 1 as

   $P1_0=r_y^2 - r_x^2 r_y+(1/4)* r_x^2$

3. At each X_k position in region 1, starting at k = 0, perform the following test:

   If P1<O

   $X_{k+1} = X_k+1$

   $Y_{k+1} =$ Ук

   $P1_{k+1} = P1_k + 2r_y^2 X_{k+1} + r_y^2$

   Else

   $X_{k+1} = X_k+1$

   $Y_{k+1} =$ Ук-1

   $P1_{k+1} = P1_k + 2r_y^2 X_{k+1} + r_y^2 - 2r_x^2 Y_{k+1}$

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

4. Calculate the initial value of decision parameter in region 2 using the last point(X_0, Y_0) calculated in region 1 as

   $P2_0=r_y^2(X_0+½)^2 + r_x^2 (Y_0-1)^2 - r_x^2 r_y^2$

5. At each Y_k position in region 2, starting at k = 0, perform the following test:

   If P2≤0

   $X_{k+1} = X_k+1$

   $Y_{k+1} =$ Ук-1

   $P2_{k+1} = P2_k + 2r_y^2 X_{k+1} - 2r_x^2 Y_{k+1} + r_X^2$

   Else

   $X_{k+1} = X_k$

   $Y_{k+1} =$ Ук-1

   $P2_{k+1} = P2_k + r_x^2 - 2r_x^2 Y_{k+1}$

4. Using the same incremental calculations for x and y as in region 2 continue until y=0.
6. Determine the symmetry points in the other three quadrants.
7. Move each calculated pixel position (x,y) onto the elliptical path centered on (X_C, Y_C) and plot the co-ordinate values:

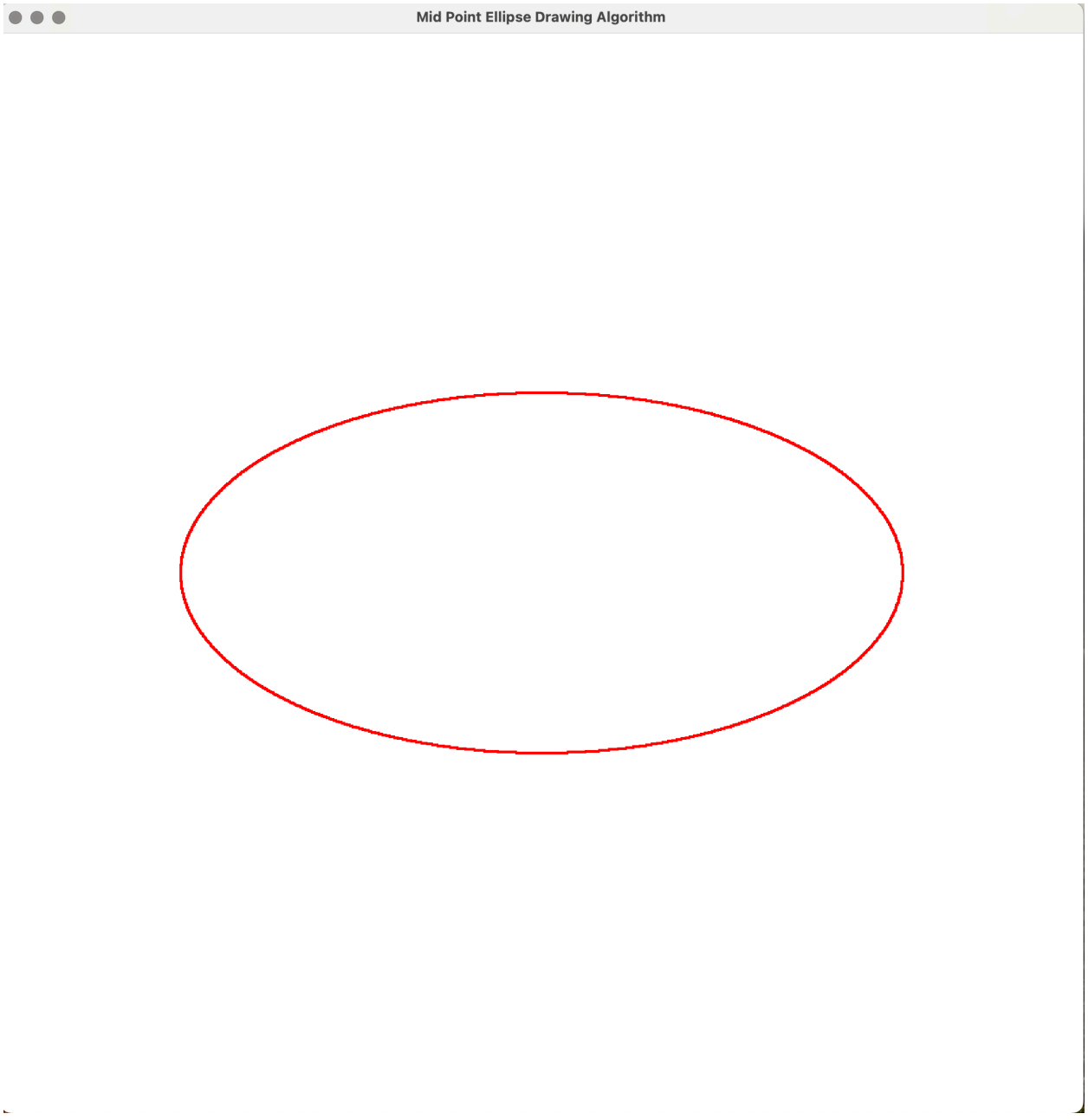
Code Implementation:

```
def drawEllipse(x_center,y_center,r_x,r_y):
    points=[]
    x=0
    y=r_y
    r_x_2=pow(r_x,2)
    r_y_2=pow(r_y,2)
    p1=r_y_2-r_x_2*r_y+(1/4*r_x_2)
    while 2*r_y_2*x<2*r_x_2*y:

 points.extend(calculateSymmetricPoints(x,y,x_center,y_center))
            if p1<0:
                x=x+1
                p1=p1+2*r_y_2*x+r_y_2
            else:
                x=x+1
                y=y-1
                p1=p1+2*r_y_2*x-2*r_x_2*y+r_y_2
    p2=r_y_2*(pow((x+1/2),2))+r_x_2*pow((y-1),2)-r_x_2*r_y_2
    while y>=0:

 points.extend(calculateSymmetricPoints(x,y,x_center,y_center))
            if p2>0:
                x=x+1
                y=y-1
                p2=p2+2*r_y_2*x-2*r_x_2*y+r_x_2
            else:
                y=y-1
                p2=p2-2*r_x_2*y+r_x_2
    return points
def calculateSymmetricPoints(x,y,x_center,y_center):
    symmetric_coordinates = [
        (x + x_center, y + y_center), (-x + x_center, y + y_center),
        (-x + x_center, -y + y_center), (x + x_center, -y + y_center)
    ]
    return symmetric_coordinates
```

Output:



Using Polar Coordinates

- To draw a circle:

Algorithm:

1. Input radius r and circle center (X_C,Y_C) and obtain the first point on the circumference of a circle centered on the origin as
   $(X_0,Y_0)=(0,r)$
2. Calculate:
   step_size=1/r
   theta_end=(pi/4)
   theta=0
3. Compute
   x= rcos(theta), y= rsin(theta)
4. Determine the symmetry points in the other seven octants.
5. Move each calculated pixel position (x,y) onto the circular path centered on (X_C,Y_C) plot the co-ordinate values
6. Repeat steps 3 through 5 until theta ≥ theta_end

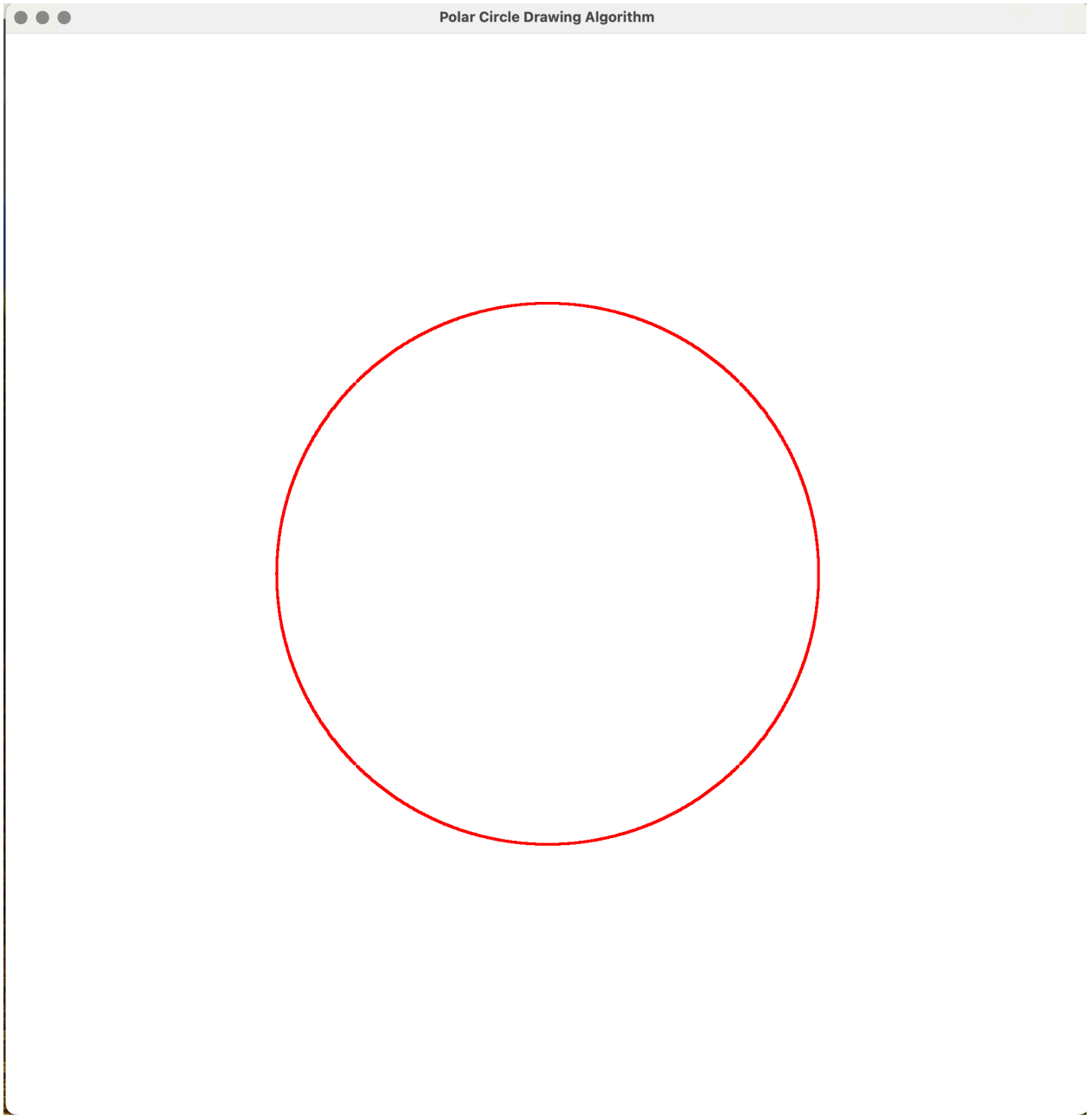
Code Implementation:

```
def drawCircle(x_center,y_center,radius):
    points=[]
    x=0
    y=radius
    step_size=1/radius;
    theta_end=np.pi/4;
    theta=0;

    while theta<theta_end:
        points.extend(calculateSymmetricPoints(x,y,x_center,y_center))
        x=radius*np.cos(theta)
        y=radius*np.sin(theta)
        theta+=step_size
    return points;

def calculateSymmetricPoints(x,y,x_center,y_center):
    symmetric_coordinates = [
        (x + x_center, y + y_center), (y + x_center, x + y_center),
        (-y + x_center, x + y_center), (-x + x_center, y + y_center),
        (-x + x_center, -y + y_center), (-y + x_center, -x +
y_center)(y + x_center, -x + y_center), (x + x_center, -y + y_center)
    ]
    return symmetric_coordinates
```

Output:



The above shown outputs are drawn using a simple program draw shape, and the coordinates returned by the above algorithm, which code snippet is shown below:

```
import glfw
import math
from OpenGL.GL import *
X_BASE=500
Y_BASE=500
def drawShape(all_points,label,pointsize=5):
    if not glfw.init():
        print("Failed to initialize GLFW")
        return -1

    window = glfw.create_window(1000, 1000, label, None,
None)if not window:
        glfw.terminate()
        return -1
    glfw.make_context_current(window)

    while not glfw.window_should_close(window):
        glClearColor(1.0, 1.0, 1.0, 1.0)
        glClear(GL_COLOR_BUFFER_BIT)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        glOrtho(0, 1200 ,1200, 0, 0, 1)
        glColor4fv([1.0, 0.0, 0.0, 1.0],)
        glPointSize(pointsize)
        glBegin(GL_POINTS)
        for point in all_points:
            glVertex2f(point[0]+X_BASE, point[1]+Y_BASE)
        glEnd()
        glfw.swap_buffers(window)
        glfw.poll_events()

    glfw.terminate()
```