Implement Digital Differential Analyzer Line drawing algorithm

Algorithm:

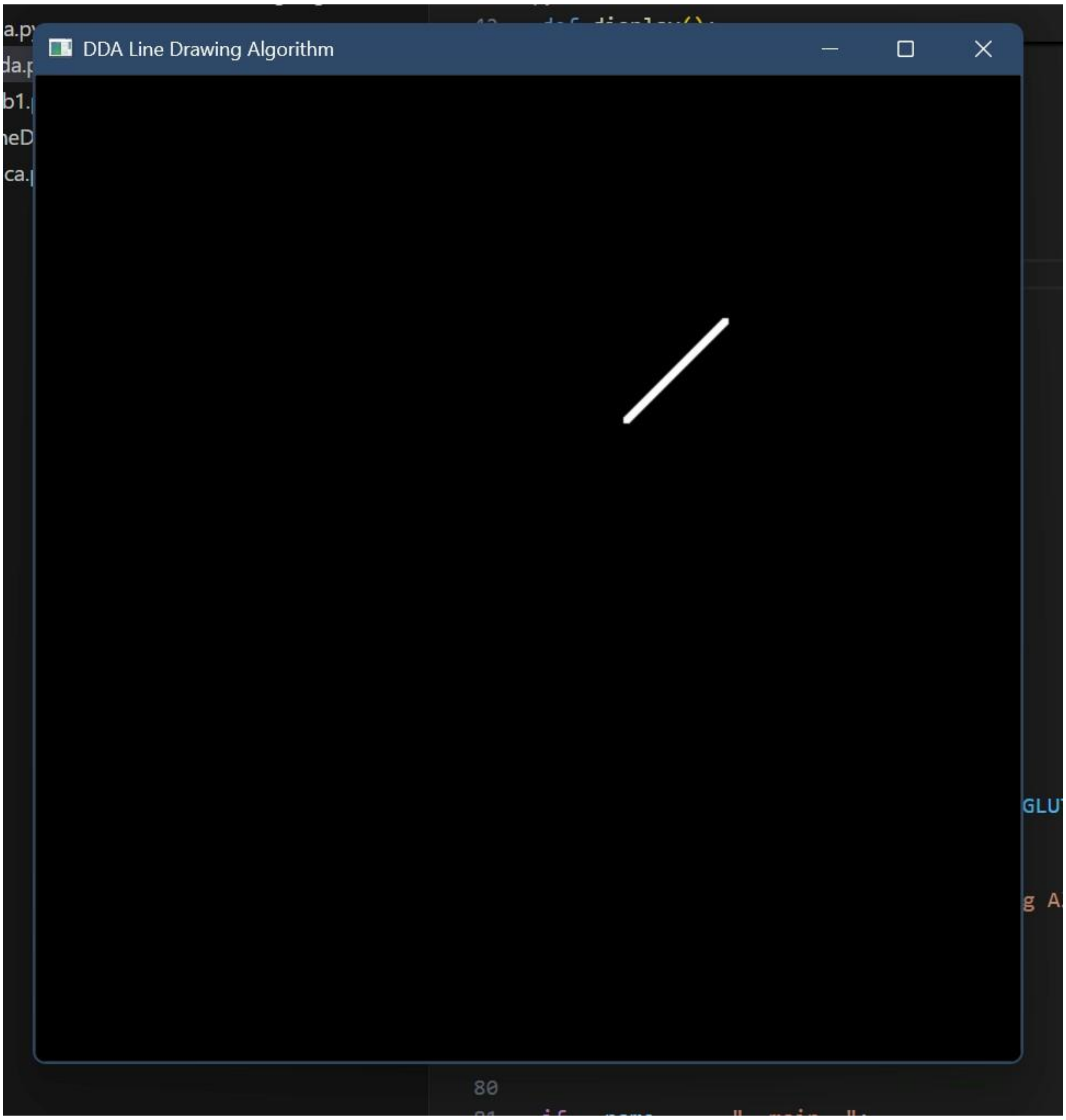
1. Get starting point(x1,y1) and ending point(x2,y2)
2. Calculate dx=x2-x1 and dy=y2-y1
3. If /dx/ >= /dy/ , step size = /dx/
   Else if /dx/ < /dy/ , step size = /dy/
4. Find increment factor as xinc = dx/stepsize and yinc = dy/stepsize
5. Set x = x1 and y=y1 and plot (x,y)
6. Calculate co-ordinates iteratively till stepsize is reached. Increase x and y w.r.t. xinc and yinc and plot them.

Source code:

```
from OpenGL.GL import * from
OpenGL.GLU import *

from OpenGL.GLUT import *

import sys

import math


x1, y1, x2, y2 = 0, 0, 0, 0

points = []


def dda_algorithm():

    global points

    points = []


    dx = x2 - x1

    dy = y2 - y1


    if abs(dx) >= abs(dy):

        steps = abs(dx)

    else:

        steps = abs(dy)


    xinc = dx / steps
yinc = dy / steps


    x = x1

    y = y1
    print("\nPlotted Points (DDA):")
    print("(x , y)")
```

```
    print("-------")

    for i in range(int(steps) + 1):
        px = math.floor(x+0.5)
py = math.floor(y+0.5)

        points.append((px, py))
        print(f"({px}, {py})")

        x += xinc
        y += yinc


def display():
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(1.0, 1.0, 1.0)
    glPointSize(4)

    glBegin(GL_POINTS)
for (x, y) in points:

        glVertex2i(x, y)
    glEnd()

    glFlush()

 def
init():

    glClearColor(0.0, 0.0, 0.0, 1.0)
    gluOrtho2D(-500, 500, -500, 500)


def main():
    global x1, y1, x2, y2

    x1 = int(input("Enter x1: "))
    y1 = int(input("Enter y1: "))
    x2 = int(input("Enter x2: "))
    y2 = int(input("Enter y2: "))
    dda_algorithm()

    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
    glutInitWindowSize(600, 600)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"DDA Line Drawing Algorithm")

    init()
    glutDisplayFunc(display)
    glutMainLoop()


if __name__ == "__main__":
    main()
```

Output:



Implement Bresenham Line Drawing algorithm for both slopes(|m|<1 and |m|>=1)

Algorithm:

1. Get starting point(x1,y1) and ending point(x2,y2)
2. Set x=x1 and y=y1 and plot (x,y)
3. Calculate dx=x2-x1 and dy=y2-y1 And also calculate P0=2dy-dx
4. At each xk along the line starting at k=0 perform the following test If Pk<0 a) Plot pixel (xk+1,yk) b) Pk+1 = Pk+2dy Else a) Plot pixel (xk+1,yk+1) b) Pk+1 = Pk+2dy-2dx
5. Repeat step 4 for dx times

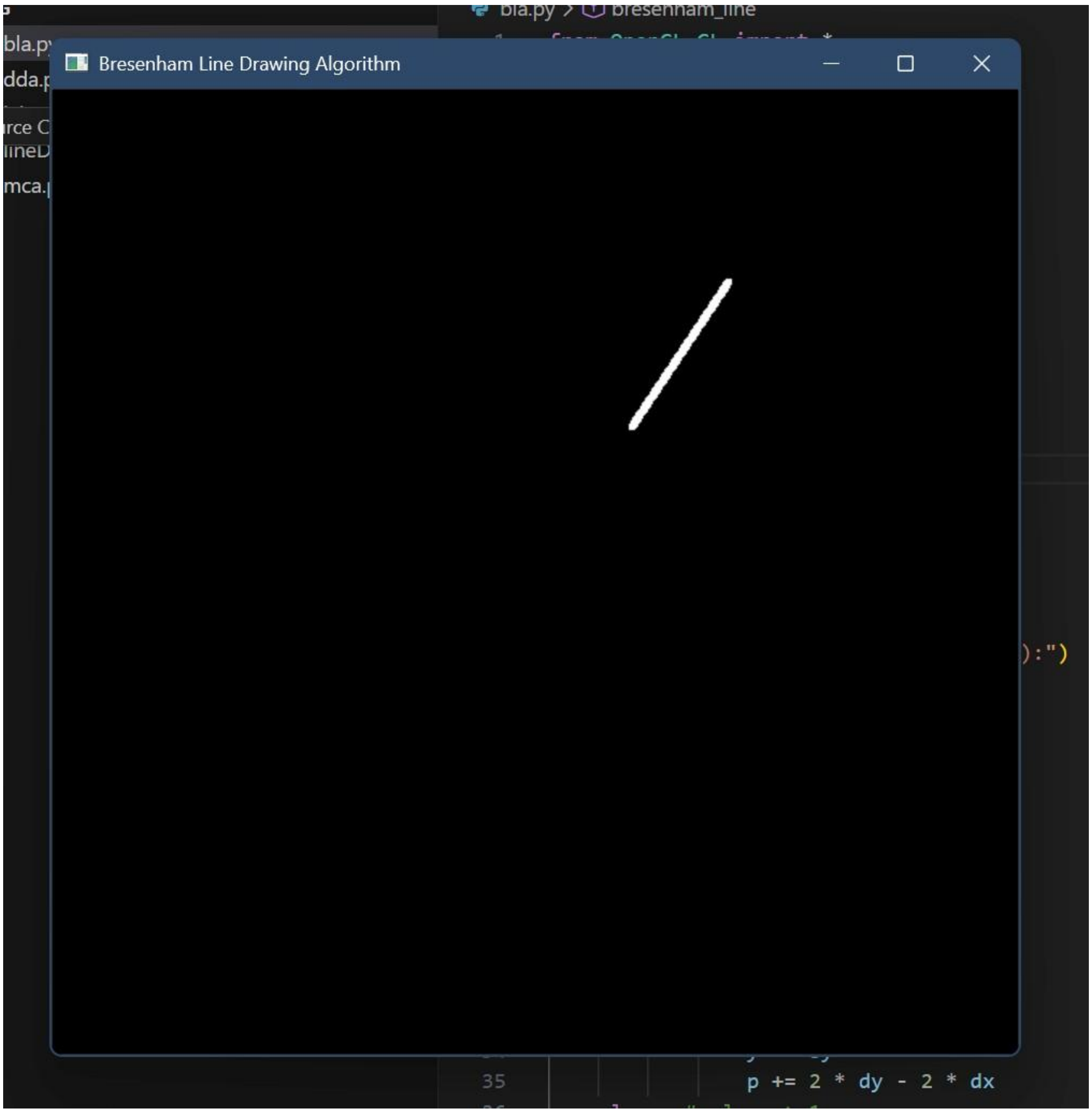
Source code:

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import sys

x1, y1, x2, y2 = 0, 0, 0, 0
points = []

def bresenham_line(x1, y1, x2, y2):
    points = []

    dx = abs(x2 - x1)
dy = abs(y2 - y1)


    sx = 1 if x2 > x1 else -1
    sy = 1 if y2 > y1 else -1

    x = x1
    y = y1

    print("\nPlotted Points (Bresenham):")
    print("(x , y)")
    print("-------")

    if dx >= dy:  # slope <= 1
        p = 2 * dy - dx
for i in range(dx + 1):
```

```
            points.append((x, y))
            print(f"({x}, {y})")
            x += sx
            if p < 0:
                p += 2 * dy
            else:
                y += sy
                p += 2 * dy - 2 * dx
    else:  # slope > 1
        p = 2 * dx - dy
        for i in range(dy + 1):
points.append((x, y))
```

```
            print(f"({x}, {y})")
            y += sy
            if p < 0:
                p += 2 * dx
            else:
                x += sx
                p += 2 * dx - 2 * dy

    return points

def display():
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(1.0, 1.0, 1.0)
glPointSize(4)


    glBegin(GL_POINTS)
    for (x, y) in points:
        glVertex2i(x, y)
    glEnd()

    glFlush()

def init():
    glClearColor(0.0, 0.0, 0.0, 1.0)
    gluOrtho2D(-500, 500, -500, 500)

def main():
    global x1, y1, x2, y2, points

    x1 = int(input("Enter x1: "))
    y1 = int(input("Enter y1: "))
    x2 = int(input("Enter x2: "))
    y2 = int(input("Enter y2: "))

    points = bresenham_line(x1, y1, x2, y2)

    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
    glutInitWindowSize(600, 600)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"Bresenham Line Drawing Algorithm")

init()
    glutDisplayFunc(display)
    glutMainLoop()

if __name__ == "__main__":
    main()
```

Output:

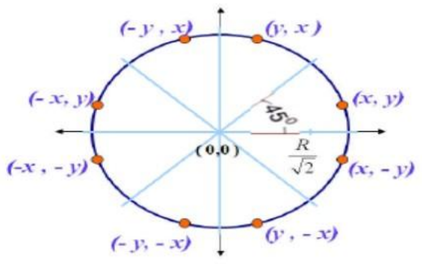

bla.py
dda.p
rce C
lineD
mca.
bla.py > bresenham_line
Bresenham Line Drawing Algorithm
):")

```
p += 2 * dy - 2 * dx
```

Write a Program to implement mid- point Circle Drawing Algorithm

Algorithm:

1. Input radius(r) and circle center(xc,yc) and obtain the first point on the circumference of a circle centered on the origin as (x0,y0) = (0,r)
2. Calculate the initial value of the decision parameter as P0=5/4 - r
3. At each xk position, starting at k=0, perform the following test: If Pk<0 */next point (xk+1,yk-1)/*
   xk+1 = xk + 1 yk+1 =
   yk
   Pk+1 = Pk + 2xk+1 + 1
   else */next point(xk+1,yk-*
   1)/ xk+1 = xk + 1 yk+1
   = yk - 1
   Pk+1 = Pk + 2xk+1 + 1 - 2yk+1 where
   2xk+1 = 2xk + 2 and 2yk+1 = 2yk -2



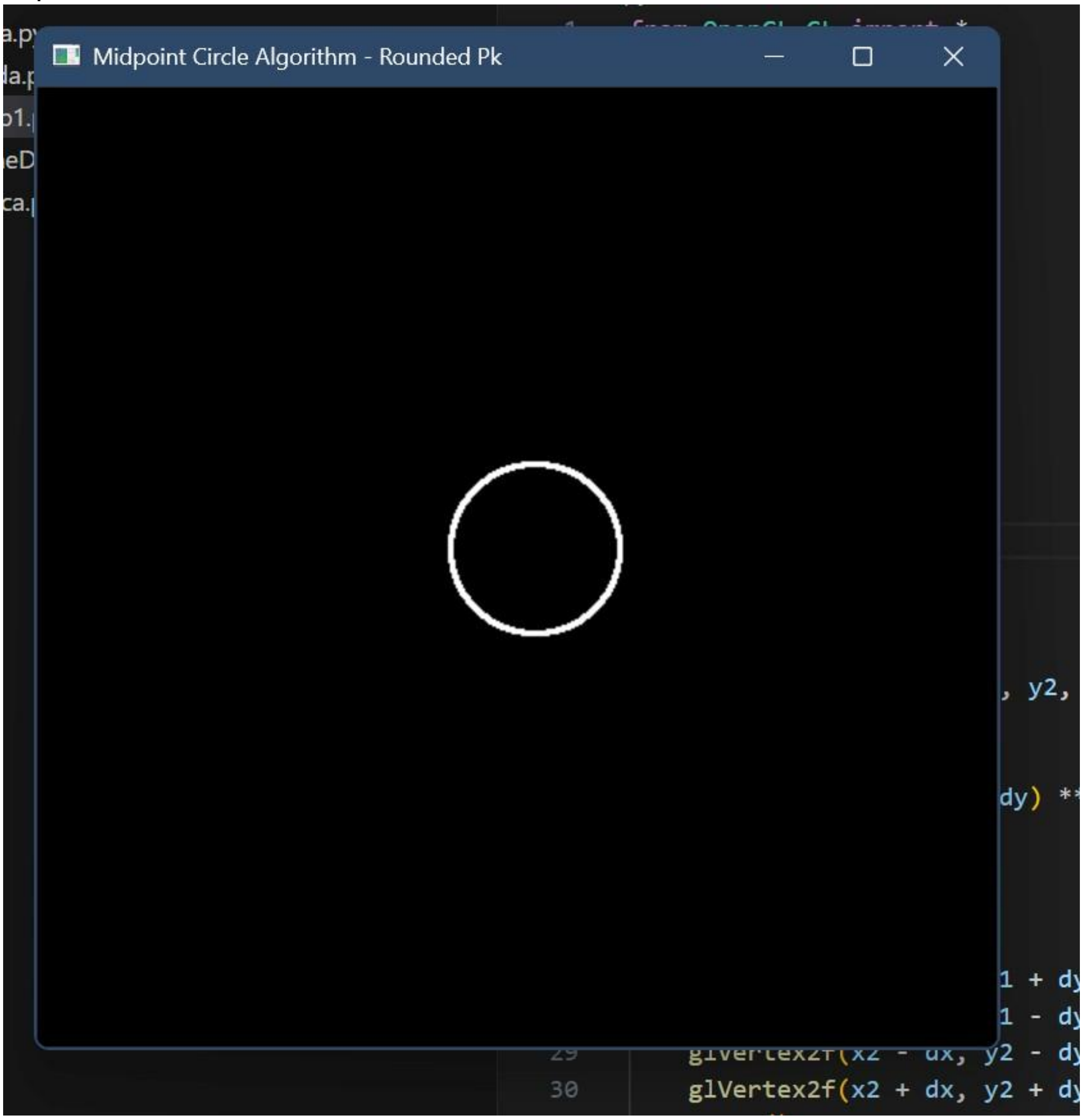
4. Determine the symmetry points in the other seven octants.
5. Move each calculated pixel position (x,y) onto the circular path centered on (xc,yc) and plot the co-ordinate values:
   x = x + xc, y = y + yc
6. Repeat step 3 through 5 until x >+ y

Source Code:

```
from OpenGL.GL import *
from OpenGL.GLU import *

from OpenGL.GLUT import *


circle_points = []

xc, yc, r = 0, 0, 0


def plot_symmetric_points(xc, yc, x, y, points):
"""Calculate 8 symmetric points and shift by circle center"""

    octant_points = [

        (xc + x, yc + y),

        (xc - x, yc + y),
        (xc + x, yc - y),
```

```
        (xc - x, yc - y),
        (xc + y, yc + x),
        (xc - y, yc + x),
        (xc + y, yc - x),
        (xc - y, yc - x)
    ]
    points.extend(octant_points)

def midpoint_circle(xc, yc, r):
    points = []
    x = 0
y = r
    P = round(5/4 - r)
    k = 0

    print(f"{'k':<5}{'Pk':<10}{'(xk, yk)':<12}{'(xk+1,
yk+1)':<15}")
    print("-" * 45)

    plot_symmetric_points(xc, yc, x, y, points)

    while x <= y:
        x_next = x + 1
        if P < 0:
            y_next = y                P_next =
round(P + 2 * x_next + 1)
        else:
            y_next = y - 1
            P_next = round(P + 2 * x_next + 1 - 2 * y_next)

        print(f"{k:<5}{P:<10}{(x, y)!s:<12}{(x_next,
y_next)!s:<15}")

        x = x_next
        y = y_next
        P = P_next
        k += 1
        plot_symmetric_points(xc, yc, x, y, points)

    return points

def display():
    glClear(GL_COLOR_BUFFER_BIT)
    glPointSize(3)
    glBegin(GL_POINTS)
    for px, py in circle_points:
        glVertex2i(px, py)
    glEnd()
    glFlush()

def get_int_input(prompt):
    while True:
        value = input(prompt)
try:
            return int(value)
        except ValueError:
            print("Please enter a valid integer.")

def main():
    global xc, yc, r, circle_points

    r = get_int_input("Enter radius of circle: ")
    xc = get_int_input("Enter x-coordinate of center: ")
    yc = get_int_input("Enter y-coordinate of center: ")

    circle_points = midpoint_circle(xc, yc, r)

    glutInit()
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
    glutInitWindowSize(500, 500)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"Midpoint Circle Algorithm - Rounded Pk")
    glClearColor(0.0, 0.0, 0.0, 1.0)
    glColor3f(1.0, 1.0, 1.0)     gluOrtho2D(-
250, 250, -250, 250)

    glutDisplayFunc(display)
    glutMainLoop()

if __name__ == "__main__":
    main()
```

Output:



Implement the Line Function (DDA/BLA) for generating a line graph of a given set of data

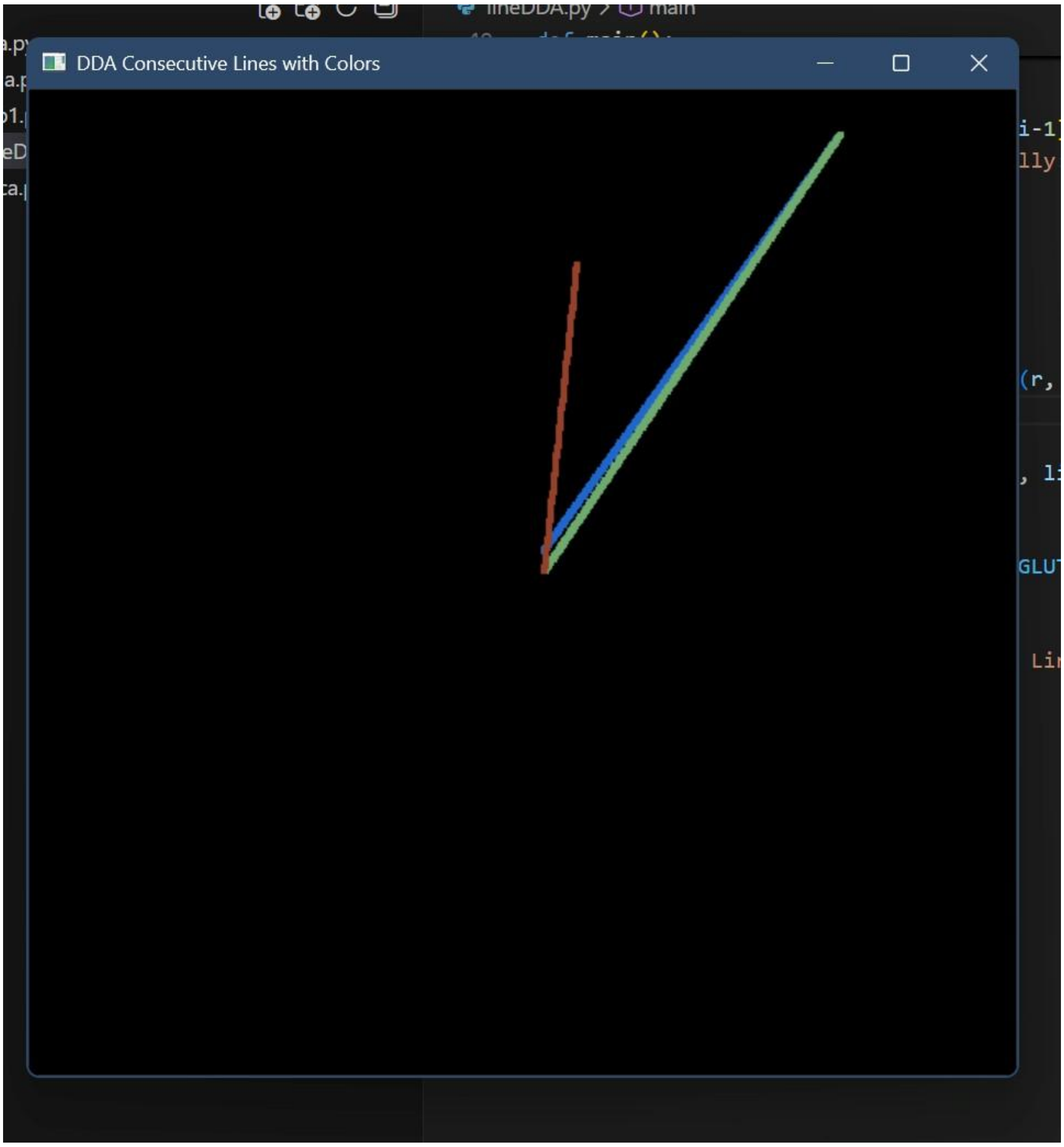
Source Code:

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import sys import
math
import random

lines = []
points = []

def dda_algorithm(x1, y1, x2, y2, color):
    global points

    dx = x2 - x1
    dy = y2 - y1

    steps = abs(dx) if abs(dx) >= abs(dy) else abs(dy)

    xinc = dx / steps
    yinc = dy / steps

    x = x1
    y = y1

    for i in range(int(steps) + 1):
px = math.floor(x + 0.5)
        py = math.floor(y + 0.5)
        points.append(((px, py), color))

        x += xinc
        y += yinc
```

```
 def
display():

    glClear(GL_COLOR_BUFFER_BIT)
    glPointSize(4)

    glBegin(GL_POINTS)
    for (x, y), (r, g, b) in points:
```

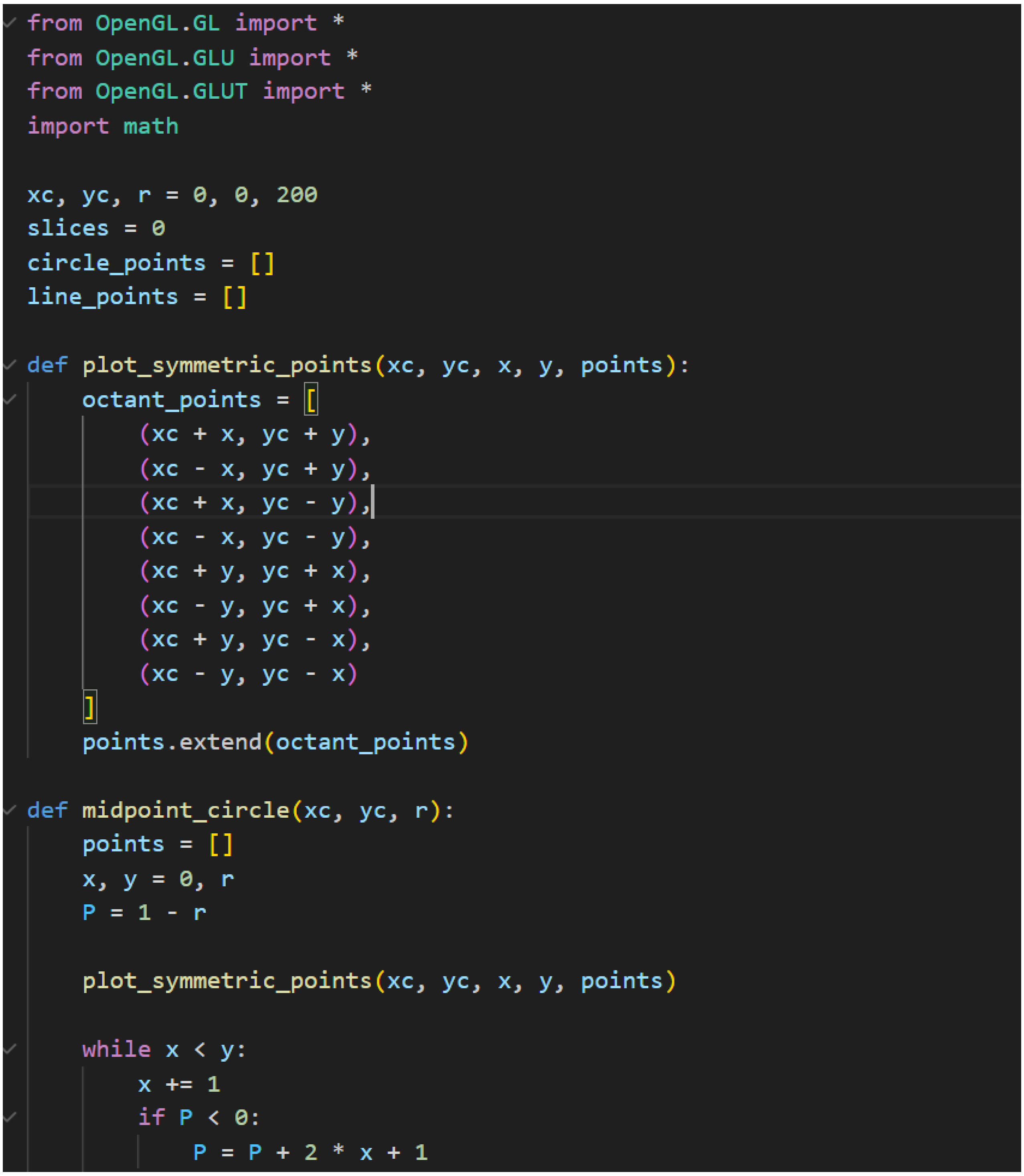
```
        glColor3f(r, g, b)
        glVertex2i(x, y)
    glEnd()

    glFlush()

def init():
    glClearColor(0.0, 0.0, 0.0, 1.0)
    gluOrtho2D(-500, 500, -500, 500)

def main():
    global lines, points

    n = int(input("Enter number of lines: "))

    print("\nLine 1")
    x1 = int(input("Enter x1: "))
    y1 = int(input("Enter y1: "))
x2 = int(input("Enter x2: "))
    y2 = int(input("Enter y2: "))

    r = random.random()
    g = random.random()
    b = random.random()
    lines.append((x1, y1, x2, y2, (r, g, b)))

    for i in range(1, n):
        print(f"\nLine {i + 1}")
        x1, y1 = lines[i-1][2], lines[i-1][3]
        print(f"Start point automatically: ({x1}, {y1})")
        x2 = int(input("Enter x2: "))
        y2 = int(input("Enter y2: "))

        r = random.random()
        g = random.random()
        b = random.random()
        lines.append((x1, y1, x2, y2, (r, g, b)))

    for line in lines:
        dda_algorithm(line[0], line[1], line[2], line[3],
line[4])

    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
    glutInitWindowSize(600, 600)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"DDA Consecutive Lines with Colors")

    init()
    glutDisplayFunc(display)
    glutMainLoop()

if __name__ == "__main__":
    main()
```

Output:

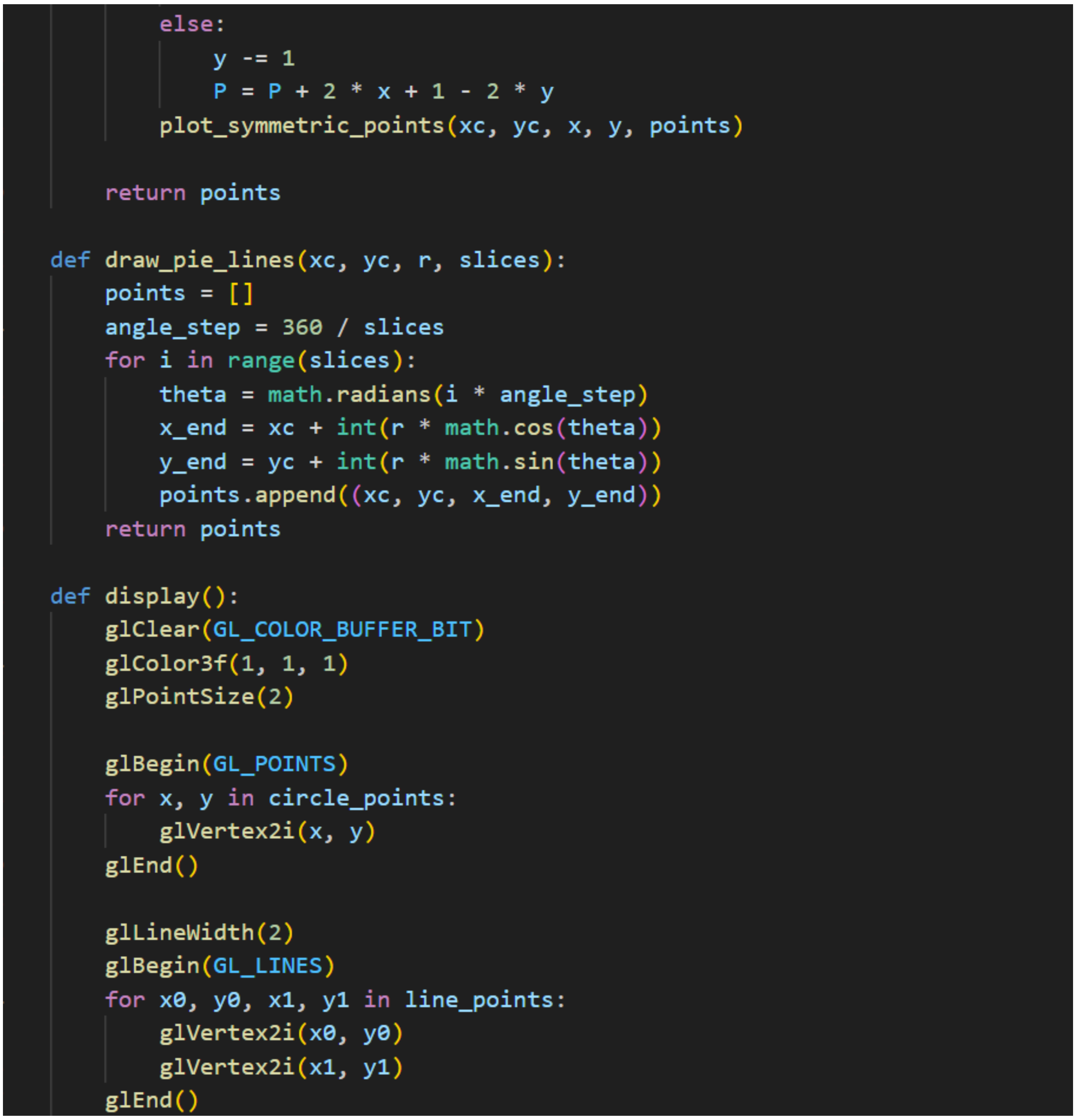


Implement the Pie chart

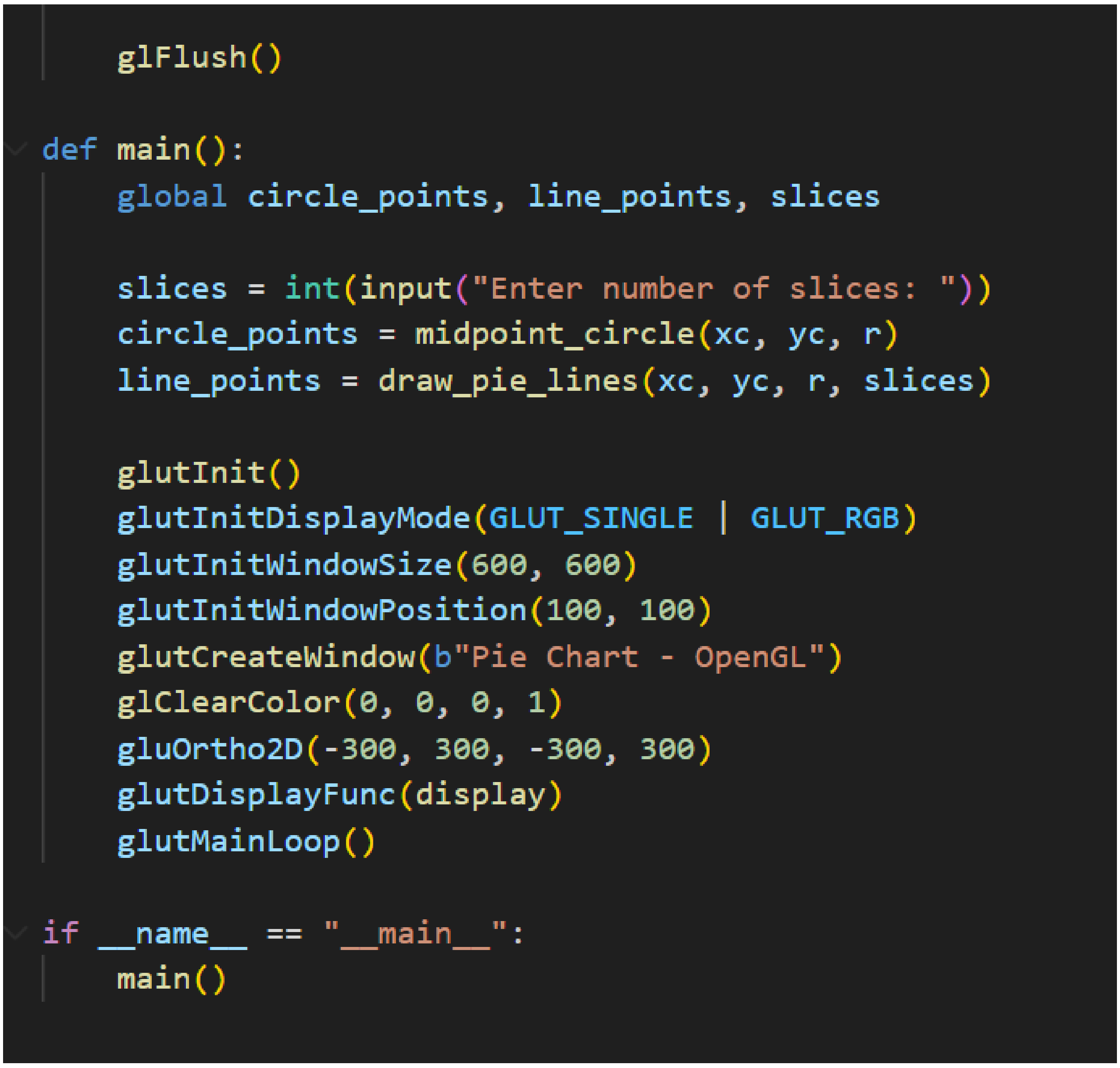
Source Code:



```python
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import math

xc, yc, r = 0, 0, 200
slices = 0
circle_points = []
line_points = []

def plot_symmetric_points(xc, yc, x, y, points):
    octant_points = [
        (xc + x, yc + y),
        (xc - x, yc + y),
        (xc + x, yc - y),
        (xc - x, yc - y),
        (xc + y, yc + x),
        (xc - y, yc + x),
        (xc + y, yc - x),
        (xc - y, yc - x)
    ]
    points.extend(octant_points)

def midpoint_circle(xc, yc, r):
    points = []
    x, y = 0, r
    P = 1 - r

    plot_symmetric_points(xc, yc, x, y, points)

    while x < y:
        x += 1
        if P < 0:
            P = P + 2 * x + 1
```

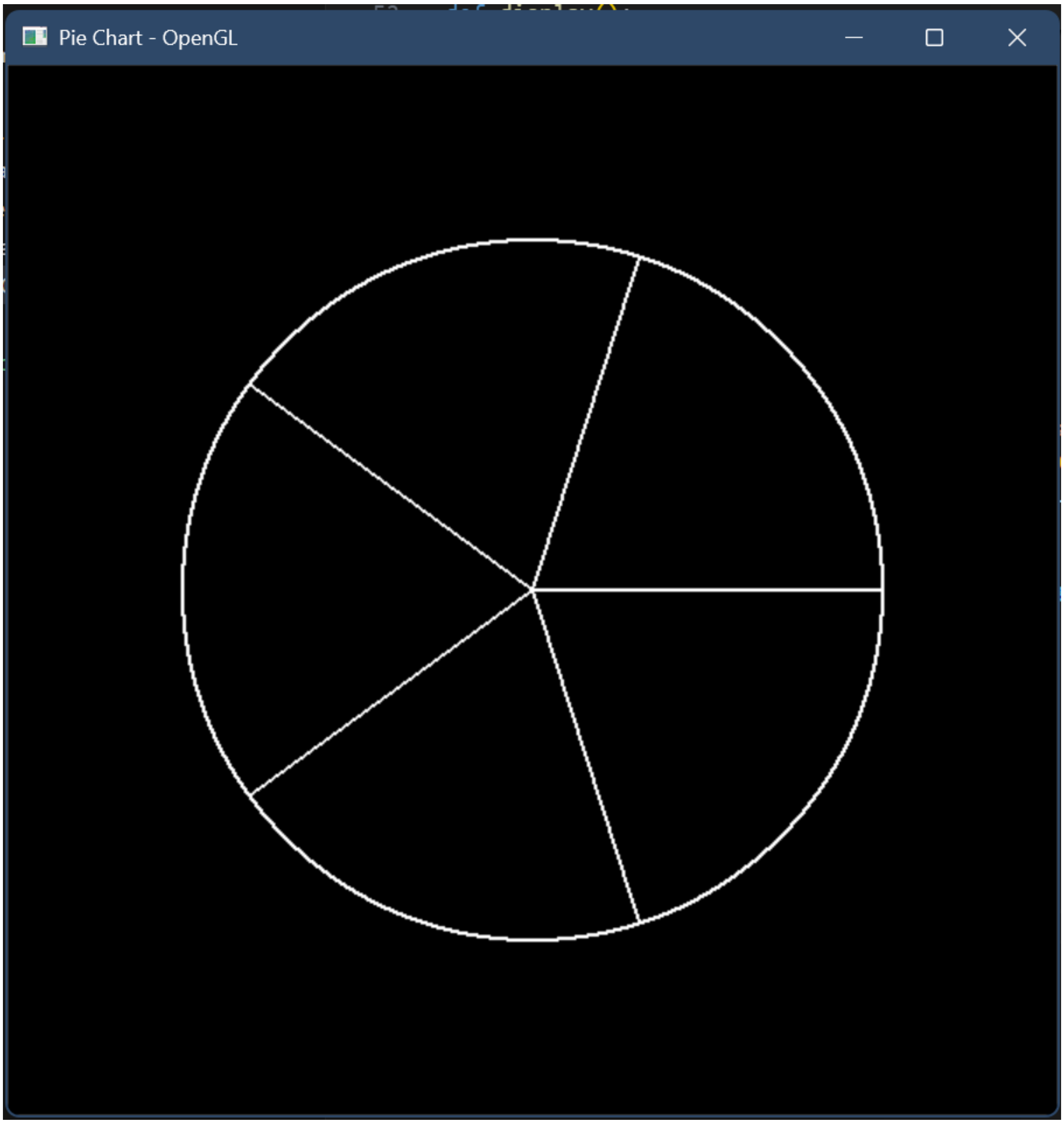


```python
        else:
            y -= 1
            P = P + 2 * x + 1 - 2 * y
        plot_symmetric_points(xc, yc, x, y, points)

    return points

def draw_pie_lines(xc, yc, r, slices):
    points = []
    angle_step = 360 / slices
    for i in range(slices):
        theta = math.radians(i * angle_step)
        x_end = xc + int(r * math.cos(theta))
        y_end = yc + int(r * math.sin(theta))
        points.append((xc, yc, x_end, y_end))
    return points

def display():
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(1, 1, 1)
    glPointSize(2)

    glBegin(GL_POINTS)
    for x, y in circle_points:
        glVertex2i(x, y)
    glEnd()

    glLineWidth(2)
    glBegin(GL_LINES)
    for x0, y0, x1, y1 in line_points:
        glVertex2i(x0, y0)
        glVertex2i(x1, y1)
    glEnd()
```



```python
    glFlush()

def main():
    global circle_points, line_points, slices

    slices = int(input("Enter number of slices: "))
    circle_points = midpoint_circle(xc, yc, r)
    line_points = draw_pie_lines(xc, yc, r, slices)

    glutInit()
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
    glutInitWindowSize(600, 600)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"Pie Chart - OpenGL")
    glClearColor(0, 0, 0, 1)
    gluOrtho2D(-300, 300, -300, 300)
    glutDisplayFunc(display)
    glutMainLoop()

if __name__ == "__main__":
    main()
```

Output:


Pie Chart - OpenGL