Computer Graphics

Karthik S
LIT2021012
5th Semester - Information Technology
Mail: lit2021012@iiitl.ac.in

Table of Contents

> All the code to this lab can be found in my github repository : https://github.com/KarthikS373/graphics

Lab 02

Implementation of Circle drawing algorithms:

1. Bresenham's Algorithm
2. Mid-Point Algorithm

AIM

The aim of this lab is to implement and compare two popular algorithms for drawing circles - Bresenham's Algorithm and Midpoint Algorithm. This involves understanding the theory behind these algorithms, coding them, and evaluating their performance. The assignment provides a Python GUI application that allows users to draw lines using these algorithms interactively

Submission

Overview

Here's an overview of the key components and functionality:

- App Class: This class represents the main application and serves as the GUI frontend. It contains UI elements for selecting the circle drawing algorithm (Bresenham's or Midpoint), input fields, a canvas for drawing and a log display for calculation logs
- Bresenham Algorithm: The Bresenham's circle drawing algorithm is implemented in the **bresenham_algorithm** function
- Mid-Point Algorithm: The Midpoint circle drawing algorithm is implemented in the **midpoint_circle_algorithm** function
- Logging: Detailed calculation logs for both algorithms are displayed on the GUI.
- Error Handling: The application validates user input to ensure that it consists of valid integer values for coordinates. It displays an error message if invalid input is detected

How to Run

- Install poetry from [here] if not already installed
- Clone the project and install the dependencies using poetry install
- Run the project using `poetry run python path_to_main.py`

How to Use

- Launch the application
- Select the drawing algorithm by clicking the respective buttons
- Input the center coordinates and radius of the circle in the provided entry fields
- Click the "Submit" button to execute the selected algorithm and draw the circle
- View the calculation logs displayed on the GUI

Bresenham’s Circle Drawing Algorithm

Algorithm

```
Input:
       - (Xc, Yc): Center coordinates of the circle
       - R: Radius of the circle
       - Desired Color: The color of the line

Output:
       - Draws a circle with center at (Xc, Yc) and the specified radius

Initialize X and Y to 0.
Calculate the initial decision parameter: P = 3 - 2 * R

For each point (X, Y), do the following:
       SetPixel(Xc + X, Yc + Y, DesiredColor)
       SetPixel(Xc - X, Yc + Y, DesiredColor)
       SetPixel(Xc + X, Yc - Y, DesiredColor)
       SetPixel(Xc - X, Yc - Y, DesiredColor)
       SetPixel(Xc + Y, Yc + X, DesiredColor)
       SetPixel(Xc - Y, Yc + X, DesiredColor)
       SetPixel(Xc + Y, Yc - X, DesiredColor)
       SetPixel(Xc - Y, Yc - X, DesiredColor)

If P is less than 0, increment X and update P as follows:
       P = P + 4 * X + 6
If P is greater than or equal to 0, increment X and decrement Y, then update P as follows:
       P = P + 4 * (X - Y) + 10
```

Code

```
def bresenham_circle_algorithm(x_center, y_center, radius):
    x = radius
    y = 0
    points = []
    logs = []
    P = 3 - (radius << 1)

    points.append((x_center + x, y_center - y))

    if radius > 0:
        points.append((x_center - x, y_center - y))
        points.append((x_center + y, y_center + x))
        points.append((x_center - y, y_center + x))

    while y <= x:
        y += 1
        if P <= 0:
            P += (y << 1) + 1
        else:
            x -= 1
            P += ((y - x + 1) << 1)
        if x < y:
            break
        points.append((x_center + x, y_center - y))
        points.append((x_center - x, y_center - y))
        points.append((x_center + x, y_center + y))
        points.append((x_center - x, y_center + y))
        if x != y:
            points.append((x_center + y, y_center - x))

            points.append((x_center - y, y_center - x))

            points.append((x_center + y, y_center + x))

            points.append((x_center - y, y_center + x))

    return points
```

Midpoint Circle Drawing Algorithm

Algorithm

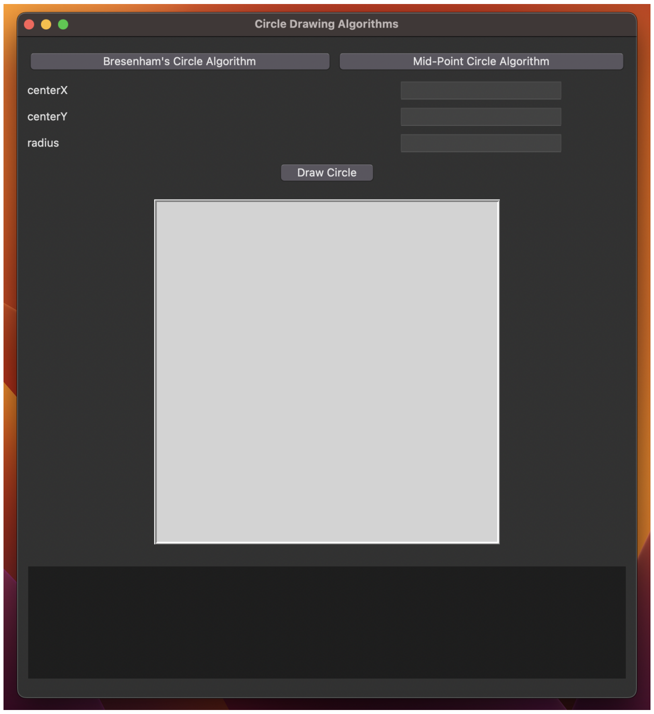
```
Input:

- (Xc, Yc): Center coordinates of the circle

- R: Radius of the circle

- Desired Color: The color of the line


Output:

- Draws a circle with center at (Xc, Yc) and the specified radius


Initialize variables:

    x to 0

    y to r

Calculate the initial decision parameter:

    P0 = 5/4 - r


Loop until x >= y


Plot the pixels at eight symmetric positions:

    SetPixel(xc + x, yc + y, DesiredColor)

    SetPixel(xc - x, yc + y, DesiredColor)

    SetPixel(xc + x, yc - y, DesiredColor)

    SetPixel(xc - x, yc - y, DesiredColor)

    SetPixel(xc + y, yc + x, DesiredColor)

    SetPixel(xc - y, yc + x, DesiredColor)

    SetPixel(xc + y, yc - x, DesiredColor)

    SetPixel(xc - y, yc - x, DesiredColor)
```

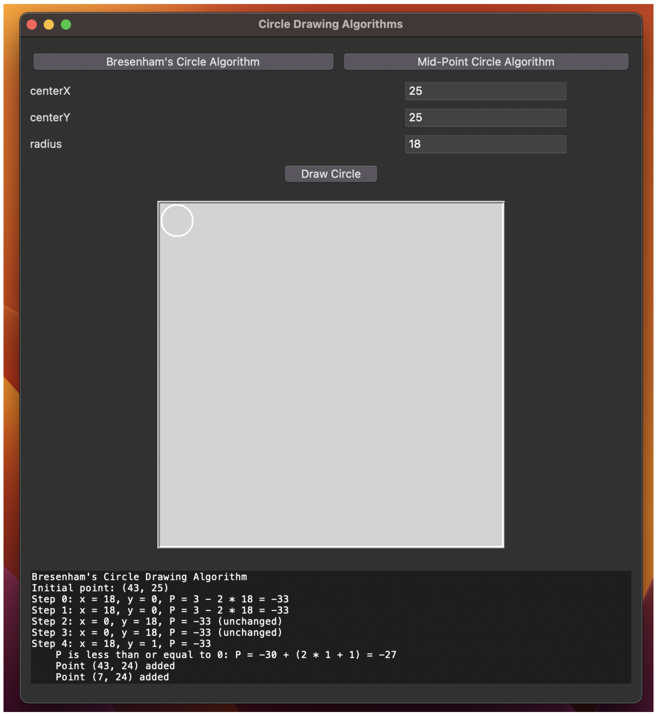
```
Calculate the next decision parameter Pk+1:
    If Pk is less than 0:
        Pk+1 = Pk + 2xk+1 + 1
    If Pk is greater than or equal to 0:
        Pk+1 = Pk + 2xk+1 + 1 - 2yk+1

Increment x by 1
Update the increment terms:
    2xk+1 = 2xk+1 + 2
    2yk+1 = 2yk+1 - 2

Determine which position is closer to the circle path:
    If Pk+1 is negative:
        select the pixel at (xk+1, yk)
    If Pk+1 is non-negative:
        select the pixel at (xk+1, yk-1)

Continue the loop for all points (x, y) along the circumference of the circle

Move each calculated pixel position (x, y) onto the circular path centered at (xc, yc) and plot the
coordinate values:
    X = x + xc
    Y = y + yc
```

Code

```
def midpoint_circle_algorithm(x_center, y_center, radius):
    x = radius
    y = 0
    points = []
    logs = []

    points.append((x_center + x, y_center - y))
    P = 1 - radius

    while x > y:
        y += 1

        if P <= 0:
            P = P + (2 * y + 1)
        else:
            x -= 1
            P = P + (2 * y - 2 * x + 1)

        if x < y:
            break
        points.append((x_center + x, y_center - y))
        points.append((x_center - x, y_center - y))
        points.append((x_center + x, y_center + y))
        points.append((x_center - x, y_center + y))
        if x != y:
            points.append((x_center + y, y_center - x))
            points.append((x_center - y, y_center - x))
            points.append((x_center + y, y_center + x))

            points.append((x_center - y, y_center + x))


    return points
```

ScreenShots


Circle Drawing Algorithms
Bresenham's Circle Algorithm
Mid-Point Circle Algorithm
centerX
centerY
radius
Draw Circle

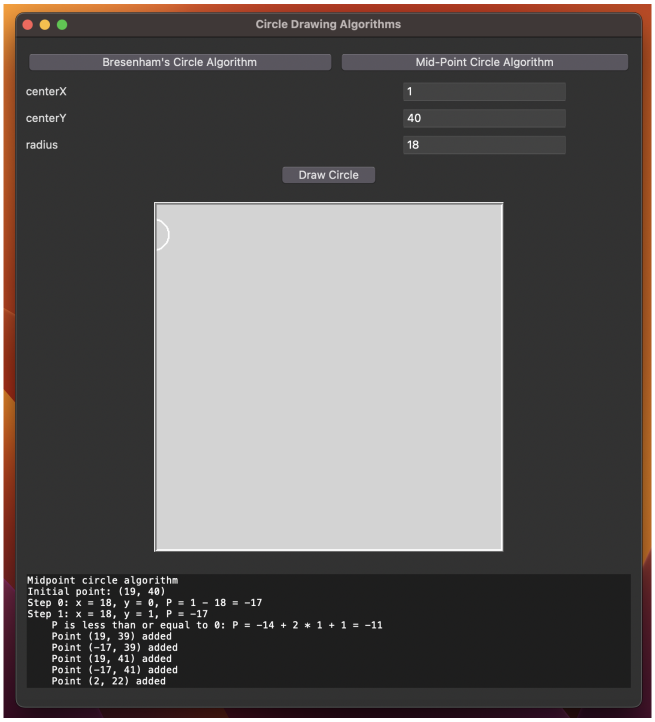

Circle Drawing Algorithms
Bresenham's Circle Algorithm
Mid-Point Circle Algorithm
Draw Circle

centerX 25

centerY 25

radius 18

```
Bresenham's Circle Drawing Algorithm
Initial point: (43, 25)
Step 0: x = 18, y = 0, P = 3 - 2 * 18 = -33
Step 1: x = 18, y = 0, P = 3 - 2 * 18 = -33
Step 2: x = 0, y = 18, P = -33 (unchanged)
Step 3: x = 0, y = 18, P = -33 (unchanged)
Step 4: x = 18, y = 1, P = -33
    P is less than or equal to 0: P = -30 + (2 * 1 + 1) = -27
    Point (43, 24) added
    Point (7, 24) added
```

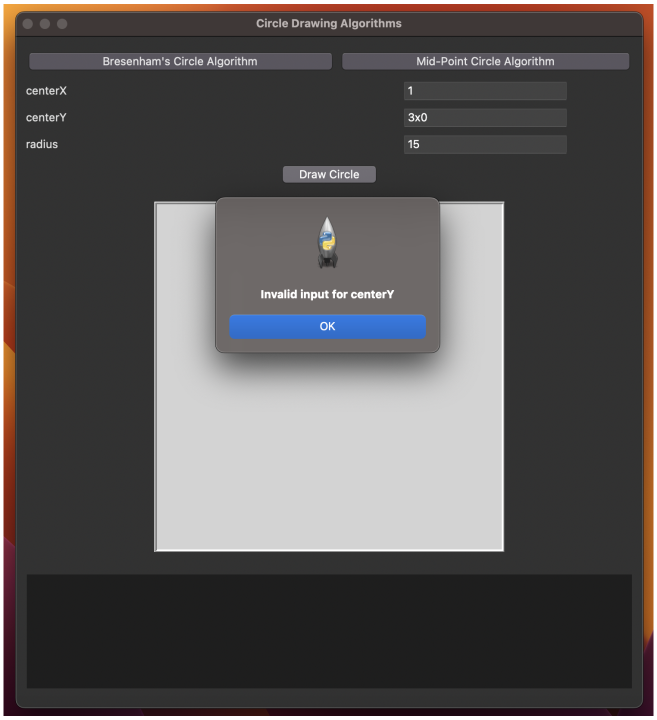

Circle Drawing Algorithms
Bresenham's Circle Algorithm
Mid-Point Circle Algorithm
Draw Circle

centerX 1

centerY 40

radius 18

```
Midpoint circle algorithm
Initial point: (19, 40)
Step 0: x = 18, y = 0, P = 1 - 18 = -17
Step 1: x = 18, y = 1, P = -17
    P is less than or equal to 0: P = -14 + 2 * 1 + 1 = -11
    Point (19, 39) added
    Point (-17, 39) added
    Point (19, 41) added
    Point (-17, 41) added
    Point (2, 22) added
```


Circle Drawing Algorithms
Bresenham's Circle Algorithm
Mid-Point Circle Algorithm
Draw Circle
OK

centerX 1

centerY 3x0

radius 15

Invalid input for centerY