Computer Graphics

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

Lab 01

Implementation of Line drawing algorithms: DDA Algorithm, Bresenham's Algorithm

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

AIM

This lab assignment focuses on the implementation and visualization of two popular line drawing algorithms: DDA (Digital Differential Analyzer) and Bresenham's Algorithm. 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 drawing algorithm (DDA or Bresenham), input fields for specifying line coordinates, a canvas for drawing and a log display for calculation logs
- DDA Algorithm: The DDA line drawing algorithm is implemented in the **dda_algorithm** function. It calculates and logs the points along the line and the relevant calculations
- Bresenham Algorithm: The Bresenham line drawing algorithm is implemented in the **bresenham_algorithm** function.
- Logging: Calculation logs for both algorithms are displayed on the GUI. This includes details such as the values of dx, dy, slope, steps, Xinc, Yinc and the decision parameters at each point
- 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 (DDA or Bresenham) by clicking the respective buttons
- Input the starting and ending coordinates of the line in the provided entry fields
- Click the "Submit" button to execute the selected algorithm and draw the line
- View the calculation logs displayed on the GUI

DDA Line Drawing Algorithm

Algorithm

```
Input:

- (Xstart, Ystart): Starting point coordinates

- (Xend, Yend): Ending point coordinates

- Desired Color: The color of the line


Output:

- Draws a line from (Xstart, Ystart) to (Xend, Yend)


m = (Yend - Ystart) / (Xend - Xstart)


if (abs(m) > 1) and (Ystart > Yend):

       Swap(Xstart, Xend)

       Swap(Ystart, Yend)

X = Xstart

Y = Ystart

SetPixel(X, Y, DesiredColor)

if (abs(m) > 1):

       m = 1 / m

       Y = Y + 1


while (Y <= Yend - 1):

       X = X + m

       RoundedX = Round(X)

       SetPixel(RoundedX, Y, DesiredColor)

       Y = Y + 1

       SetPixel(Xend, Yend, DesiredColor)
```

Code

```
def dda_algorithm(startX, startY, endX, endY):
       points = []
       dx = endX - startX
       dy = endY - startY

       steps = max(abs(dx), abs(dy))

       Xinc = dx / float(steps)
       Yinc = dy / float(steps)

       x, y = startX, startY
       points.append((x, y))

       for _ in range(steps):
           x += Xinc
           y += Yinc
           points.append((round(x), round(y)))

       return points
```

Bresenham's Line Drawing Algorithm

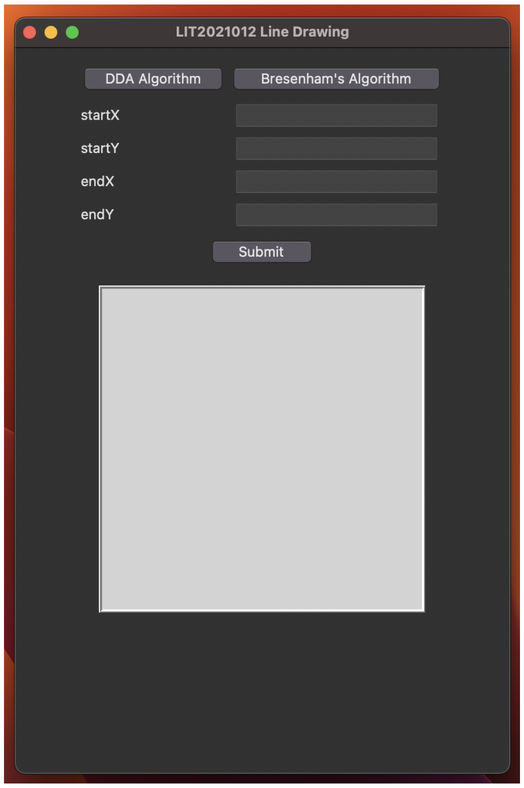
Algorithm

```
Input:
- (Xstart, Ystart): Starting point coordinates
- (Xend, Yend): Ending point coordinates
- Desired Color: The color of the line

Output:
- Draws a line from (Xstart, Ystart) to (Xend, Yend)

x0 = Xstart
y0 = Ystart
Δx = abs(x1 - x0)
Δy = abs(y1 - y0)

2Δy = 2 * Δy
2Δy - 2Δx = 2 * Δy - 2 * Δx

P0 = 2Δy - Δx

for k = 1 to Δx:
       if P < 0:
       x = x + 1
       P = P + twoΔy
else:
       x = x + 1
       y = y + 1
       P = P + 2Δy - 2Δx
```

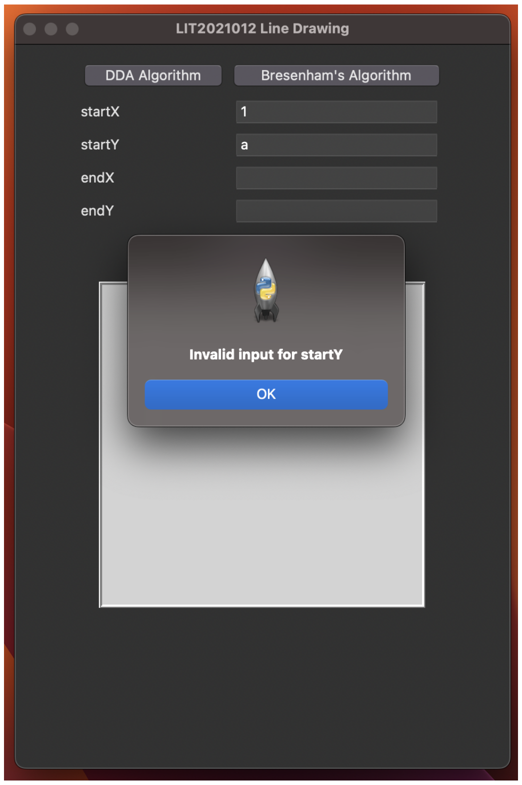
Code

```
def bresenham_algorithm(startX, startY, endX, endY):

points = []


dx = endX - startX

dy = endY - startY


x, y = startX, startY

points.append((x, y))


if abs(dx) > abs(dy):

       steps = abs(dx)

else:

       steps = abs(dy)


Xinc = dx / float(steps)

Yinc = dy / float(steps)


p = 2 * abs(dy) - abs(dx)

for _ in range(steps):

       if p < 0:

           p += 2 * abs(dy)

       else:

           p += 2 * abs(dy) - 2 * abs(dx)

           y += Yinc

           x += Xinc

points.append((round(x), round(y)))

return points
```

ScreenShots

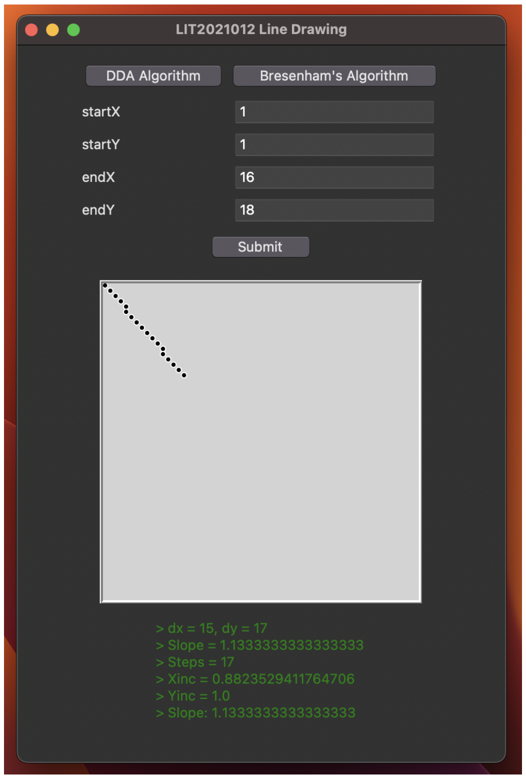

LIT2021012 Line Drawing

startX	
startY	
endX	
endY	


LIT2021012 Line Drawing

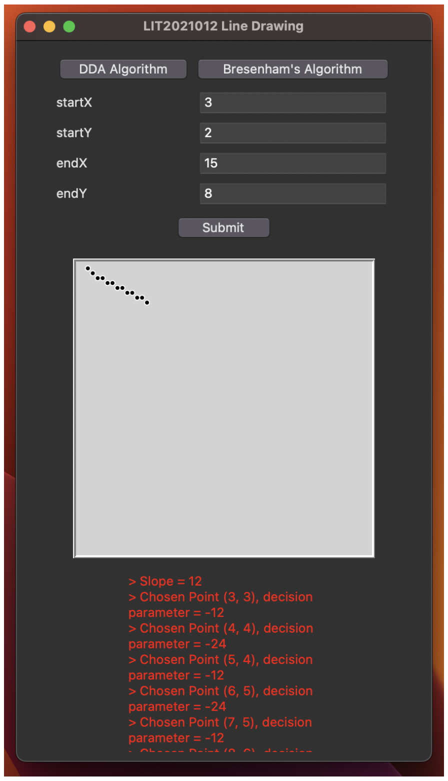
startX	1
startY	a
endX	
endY	

Invalid input for startY


LIT2021012 Line Drawing

startX	1
startY	1
endX	16
endY	18

```
> dx = 15, dy = 17
> Slope = 1.1333333333333333
> Steps = 17
> Xinc = 0.8823529411764706
> Yinc = 1.0
> Slope: 1.1333333333333333
```


LIT2021012 Line Drawing

startX	3
startY	2
endX	15
endY	8

```
> Slope = 12
> Chosen Point (3, 3), decision
parameter = -12
> Chosen Point (4, 4), decision
parameter = -24
> Chosen Point (5, 4), decision
parameter = -12
> Chosen Point (6, 5), decision
parameter = -24
> Chosen Point (7, 5), decision
parameter = -12
```