BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

Lab Report

Title	Marks
Course Code: CSE-342 **Course Title**: Computer Graphics **Lab no**:	

Submitted By	Submitted To
Name: Saba-E-Zannat **ID**: 21225103304 **Intake**: 49 **Section**: 05 Department of CSE Bangladesh University of Business & Technology	**Name**: Md. Khairul Islam **Designation**: Lecturer Department of CSE Bangladesh University of Business & Technology

Submission Date

Teacher’s Signature

1. **Title**: Bresenham's Line Drawing Algorithm

2. **Objective**:

The objective of this report is to implement and understand Bresenham’s Line Drawing Algorithm for rendering lines on a pixel-based display.

3. **Environment**:

- ● Operating System: Windows
- ● Programming Language: Python
- ● Editor: Jupyter Notebook
- ● Graphics Library: matplotlib
- ● Hardware: Standard computer system with basic graphics capabilities

4. **Introduction**:

In computer graphics, Bresenham’s Line Algorithm is an efficient way to draw a straight line on a raster display using only integer calculations. Developed by Jack Bresenham in 1962, this algorithm determines the pixels that should be illuminated to best approximate a straight line between two given points. Unlike traditional methods that rely on floating-point arithmetic, Bresenham’s algorithm uses only integer additions and subtractions, making it computationally efficient for graphics applications.

5. **Algorithm**:

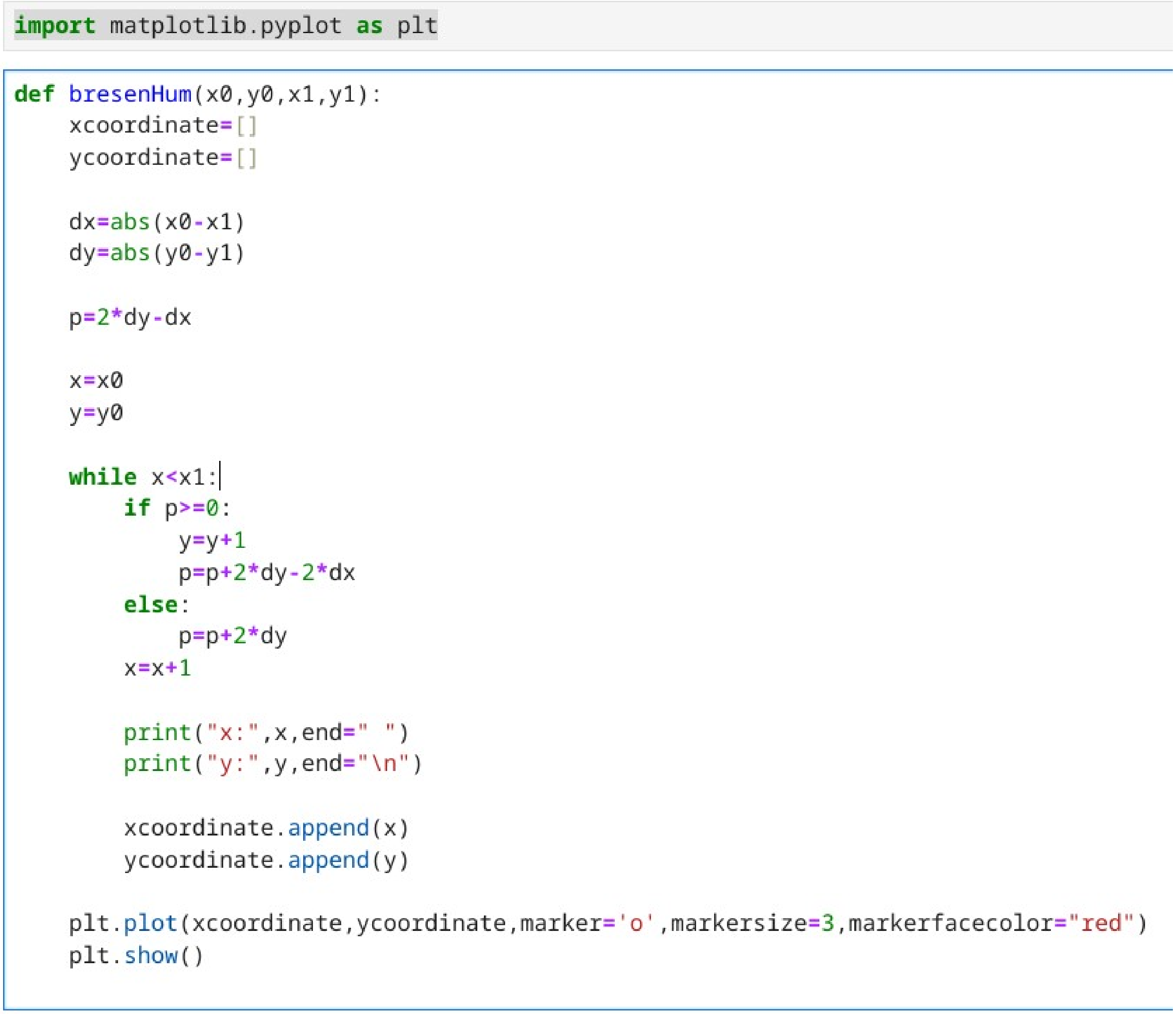
Digital Differential Analyzer (DDA) Algorithm

Steps:

Start with the initial pixel at – (x0, y0).

1. Compute the differences:
   - dx = x1 - x0
   - dy = y1 - y0
2. Initialize decision parameter:
   - p = 2dy - dx
3. Iterate through x from x0 to x1:
   - Plot the pixel at (x, y)
   - If p < 0:
     - p = p + 2dy
   - Else:
     - y = y + 1
     - p = p + 2dy - 2dx
4. Repeat until the endpoint (x1, y1) is reached.

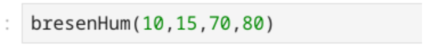
6. **Code**:



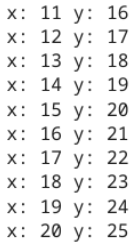
```python
import matplotlib.pyplot as plt
```

```python
def bresenHum(x0,y0,x1,y1):
    xcoordinate=[]
    ycoordinate=[]

    dx=abs(x0-x1)
    dy=abs(y0-y1)

    p=2*dy-dx

    x=x0
    y=y0

    while x<x1:
        if p>=0:
            y=y+1
            p=p+2*dy-2*dx
        else:
            p=p+2*dy
        x=x+1

        print("x:",x,end=" ")
        print("y:",y,end="\n")

        xcoordinate.append(x)
        ycoordinate.append(y)

    plt.plot(xcoordinate,ycoordinate,marker='o',markersize=3,markerfacecolor="red")
    plt.show()
```

7.**Snapshot(Input & Output)**:

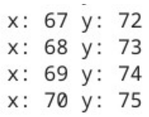


```
bresenHum(10,15,70,80)
```

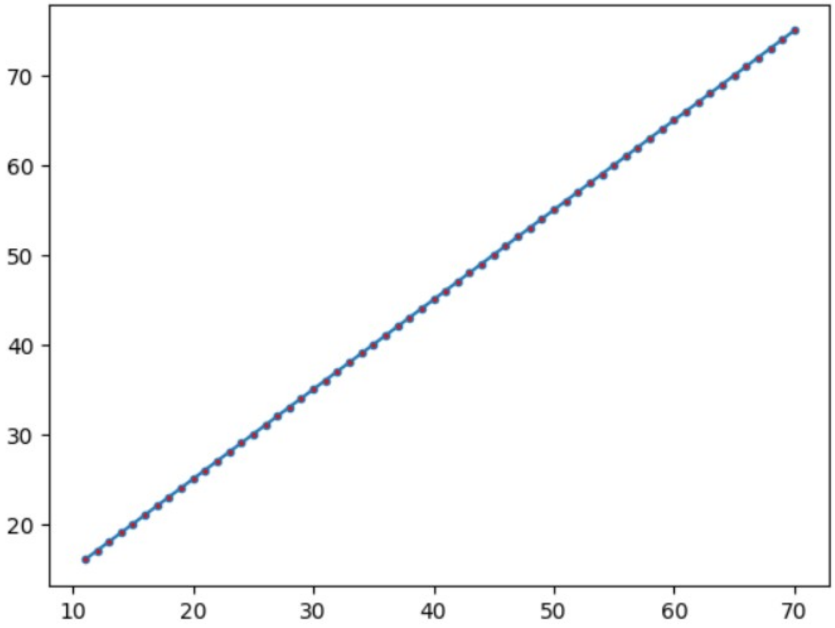


```
x: 11 y: 16
x: 12 y: 17
x: 13 y: 18
x: 14 y: 19
x: 15 y: 20
x: 16 y: 21
x: 17 y: 22
x: 18 y: 23
x: 19 y: 24
x: 20 y: 25
```

…..



```
x: 67 y: 72
x: 68 y: 73
x: 69 y: 74
x: 70 y: 75
```



8. **Discussion & conclusion**:

Bresenham’s Line Algorithm is widely used in graphical applications due to its simplicity and speed. Unlike other line-drawing methods that rely on floating-point arithmetic, this algorithm efficiently uses integer calculations, making it ideal for real-time rendering. One of its main advantages is its ability to minimize computational overhead while maintaining accuracy. However, the algorithm is limited to lines with a slope between 0 and 1 in its basic form, requiring adaptations for other cases. Additionally, anti-aliasing techniques may be needed to smooth jagged edges in high-resolution displays.