Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report - I

[COMP 342]

Submitted by:

Abhijeet Poudel (44)

Submitted to:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date: 18/12/2022

TABLE OF CONTENTS

Chapter 1: Introduction 3

1.1 Language Primitives and Graphics Library 3

Chapter 2: Helper Functions 3

2.1 toNVC() Function 3

2.2 toNVC2() Function 4

2.3 altList() Function 4

Chapter 3: Digital Differential Analyzer (DDA) Line Drawing Algorithm 4

3.1. Algorithm for DDA 4

3.2 Source Code 5

3.3 Output 8

Chapter 4: Bresenham's Line Drawing Algorithm 9

4.1. Algorithm for Bresenham's Line Drawing 9

4.2. Source Code 10

4.3. Output 13

4.3.1. For slope >1 (Joining points (-500,-500) & (250,250)) 13

4.3.1. For slope <1 (Joining points (-500, 500) & (250, -250)) 14

Chapter 5: Mid-point Line Drawing Algorithm 14

5.1. Algorithm for Mid- point Line Drawing: 14

5.2. Source Code 16

5.3. Output 19

5.3.1 For slope >1 (Joining points (-500,-500) & (250,250)) 19

5.3.2 For slope <1 (Joining points (-500,500) & (250,-250)) 20

Chapter 6: Conclusion 21

Chapter 1: Introduction

1.1 Language Primitives and Graphics Library

Graphics Library: PyOpenGL 3.1.6

Programming Language: Python 3.10.5

Window Context: GLFW

Helpers: Numpy

Chapter 2: Helper Functions

Helper functions can help to make code more concise and easier to read by breaking up complex tasks into smaller tasks. This can also help to improve code maintainability and reduce the risk of errors by eliminating the need to repeat code. Additionally, helper functions can help to abstract away complex logic, making the code easier to understand and debug.

In this report we have made use of 3 different helper funtions namely 1 toNVC(), altList() anf toNVC2()

2.1 toNVC() Function

```
def toNVC(xList, yList, resolution):
  for i in range(len(xList)):
    xList[i] = (xList[i]) / (resolution)
    yList[i] = (yList[i]) / (resolution)
  coordinateList = altList(xList,yList)
  return coordinateList
```

This code is used to convert a given list of coordinates from their original values to normalized values. It takes in two lists of x and y coordinates, as well as a resolution

value. It then divides each coordinate by the resolution value, creating a normalized coordinate list. Finally, it returns the normalized coordinate list.

2.2 toNVC2() Function

```
def toNVC2(lst,resolution):
  for i in range(len(lst)):
    lst[i] = (lst[i]) / (resolution)

  return lst
```

This code takes a list of numbers (lst) and a resolution value and divides each item in the list by the resolution value. It returns the list with the new values. This code could be used for normalizing a list of data for a specific resolution.

2.3 altList() Function

```
def altList(lst1, lst2):
    return [sub[item] for item in range(len(lst2))
                      for sub in [lst1, lst2]]
```

This code is used to create an alternate list using two lists. The function altList takes in two lists, lst1 and lst2, and returns a list with the items of lst1 and lst2 alternating. For example, altList([1,2,3], [4,5,6]) would return [1,4,2,5,3,6].

Chapter 3: Digital Differential Analyzer (DDA) Line Drawing Algorithm

3.1. Algorithm for DDA

Step 1: Consider initial point (x0, y0) and final point (x1, y1,) as the starting and end points

Step 2: Calculate dx, dy and slope(m) of the given points as

dx= x1-x0

dy= y1-y0

m=dy/dx

Step 3: Calculate the number of steps required for the whole process as

If (|dx| >|dy|) then steps= |dx|

Else steps= |dy|

Step 4: Calculate increment in x and y with the formula:

X_increament= dx/ steps

Y_ increament= du/steps

Step5: Calculate successive points start from x0 and y0 adding increament to the current value as

x=x+ X_increament

y=y+ Y_increament

Step 6: Repeat step 5 until x=x1

3.2 Source Code

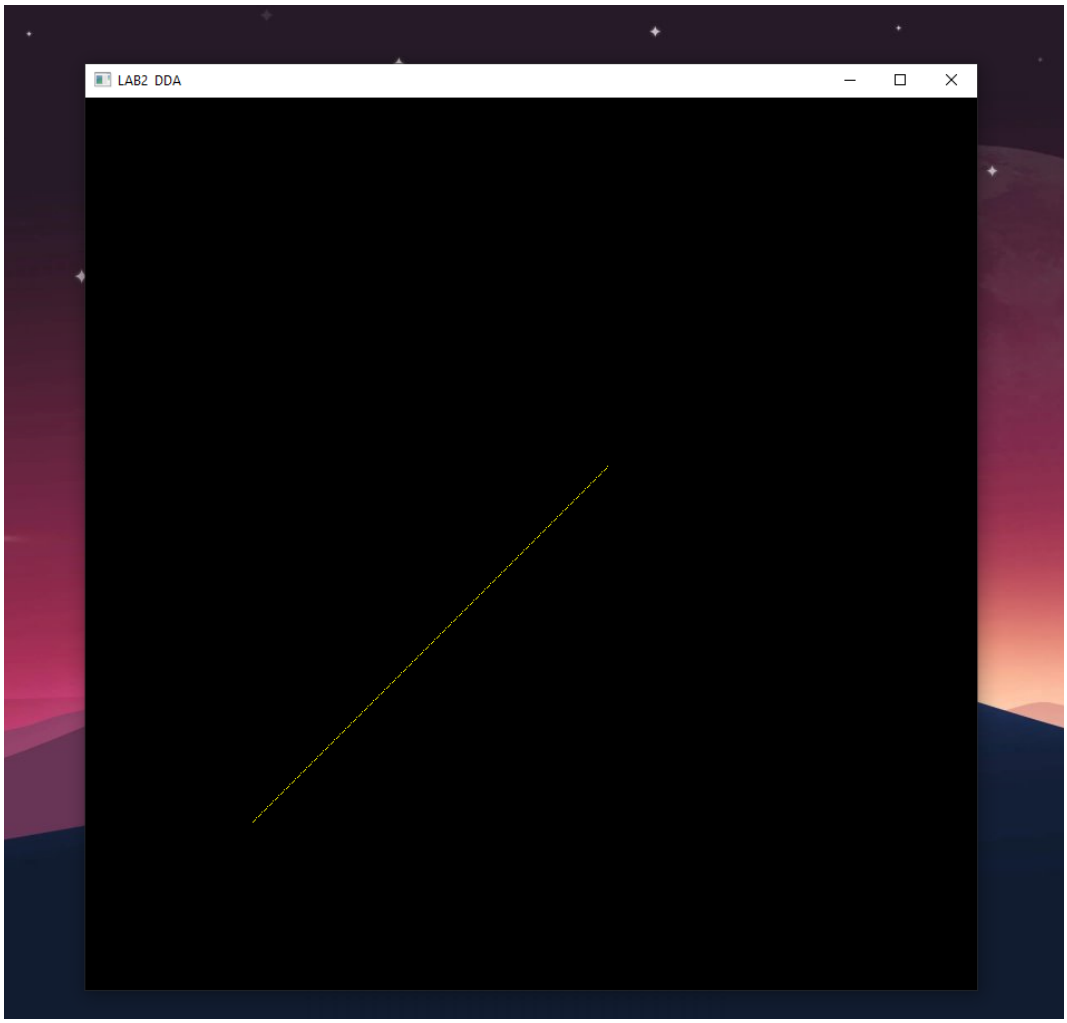
```
import glfw
import numpy as np
from OpenGL.GL import *
from OpenGL.GL.shaders import compileProgram, compileShader
from helpers import toNVC


RESOLUTION = 800


def window_resize(window, width, height):
    glViewport(0, 0, width, height)


def dda(start_x,start_y,end_x,end_y, resolution):
  x_points = []
  y_points = []
  dx=end_x-start_x
  dy=end_y-start_y
```

```
  step=abs(dy)
  if abs(dx)>abs(dy):
    step=abs(dx)
  new_x=start_x
  new_y=start_y
  x_inc=dx/step
  y_inc=dy/step

  for i in range(step):
    x_points.append(new_x)
    y_points.append(new_y)
    new_x += x_inc
    new_y += y_inc

  return toNVC(x_points,y_points,resolution)


def main():

    vertex_src = """
   #version 330

   layout(location=0) in vec2 aPos;

   void main(){

    gl_Position =vec4(aPos,0.0f,1.0f);

   }

  """

    fragment_src = """

  #version 330

  out vec4 FragColor;

  void main(){
    FragColor =vec4 (1.0f,1.0f,0.0f,1.0f);

  }

  """
    if not glfw.init():
        raise Exception("glfw cannot be initialised")

    window = glfw.create_window(RESOLUTION, RESOLUTION, "LAB2", None,
None)
```

```
    if not window:
        glfw.terminate()
        raise Exception("glfw window cannot be created!")

    # glfw.set_window_pos(window,100,100)

    glfw.set_window_size_callback(window, window_resize)
    glfw.make_context_current(window)

    temp = dda(-500, -500, 250, 250, RESOLUTION)

    vertices = np.array(temp, dtype=np.float32)

    render_count = round(len(temp))

    print(temp)
    indices = np.array([i for i in range(1, render_count + 1)],
dtype=np.uint32)

    shader = compileProgram(
        compileShader(vertex_src, GL_VERTEX_SHADER),
        compileShader(fragment_src, GL_FRAGMENT_SHADER),
    )

    vertex_buffer_object = glGenBuffers(1)
    glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer_object)
    glBufferData(GL_ARRAY_BUFFER, vertices.nbytes, vertices,
GL_STATIC_DRAW)

    element_buffer_object = glGenBuffers(1)
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, element_buffer_object)
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.nbytes, indices,
GL_STATIC_DRAW)

    glEnableVertexAttribArray(0)
    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 00,
ctypes.c_void_p(0))

    glUseProgram(shader)

    print(render_count)

    while not glfw.window_should_close(window):
        glfw.poll_events()

        glDrawElements(GL_POINTS, len(indices), GL_UNSIGNED_BYTE, None)

        glfw.swap_buffers(window)
```

```
    glfw.terminate()

main()
```

3.3 Output



Chapter 4: Bresenham's Line Drawing Algorithm

4.1. Algorithm for Bresenham's Line Drawing

Step 1: Consider initial point (x0, y0) and final point (x1, y1,) as the starting and end points

Step 2: Calculate dx and dy of the given points as

dx= x1-x0

dy= y1-y0

Step 3: Calculate initial decision Parameter as pk= 2* dy – dx

Step 4: At each x_k along the line, starting from k=0, perform the following test:

If pk < 0 then pk=pk+ 2 * dy

Else Check the slope of the line

If m>0 (i.e., positive slope) y0=y0+1

If m<0 (i.e., negative slope) y0=y0-1

And, pk = pk + 2 * dy – 2 * dx

Step 5: Change the value of x as

If x0 < x1 then x0=x0+1

Else x0 = x0 -1

Step 6: Repeat step 4 and 5 for dx number of times

4.2. Source Code

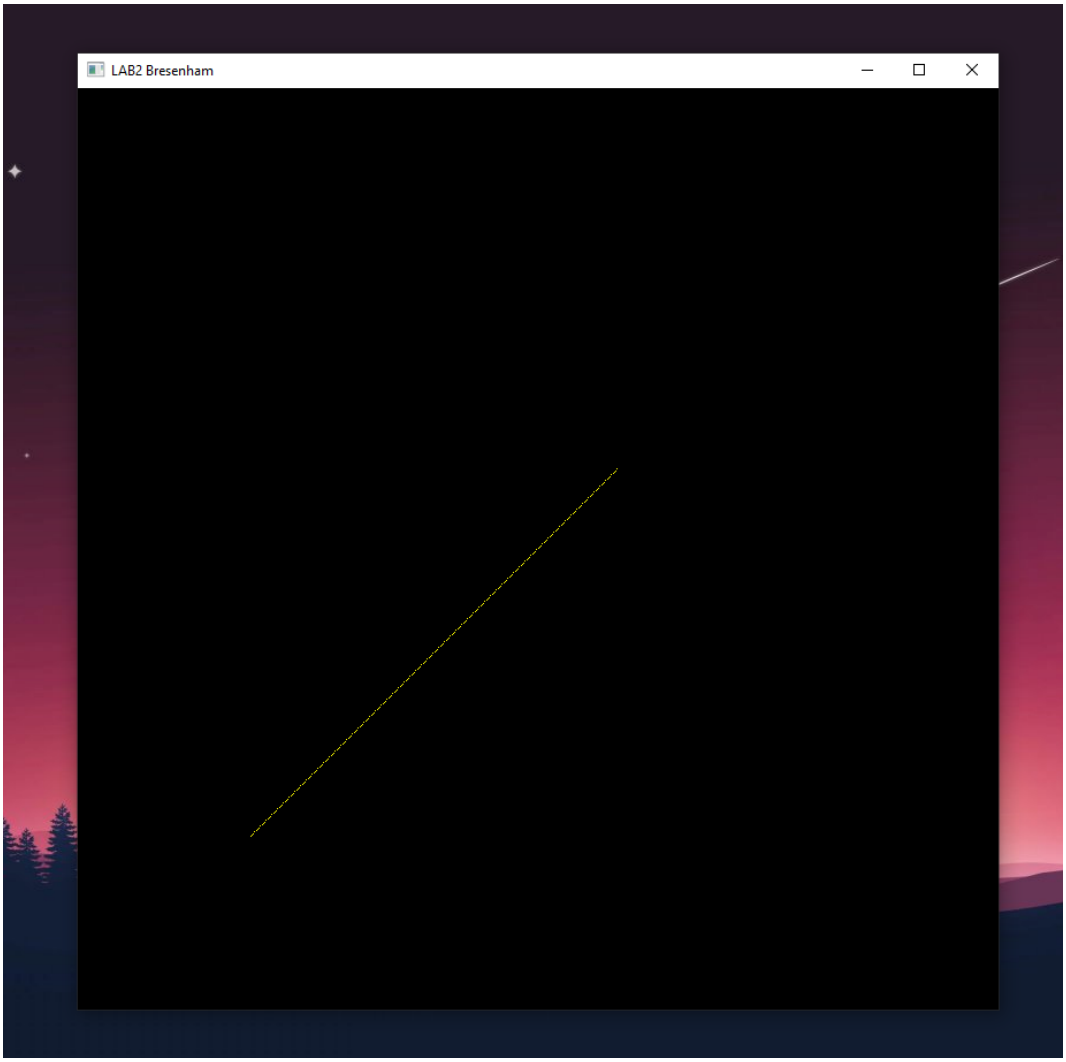
```
import glfw
import numpy as np
from OpenGL.GL import *
from OpenGL.GL.shaders import compileProgram, compileShader
from helpers import toNVC


RESOLUTION = 800


def window_resize(window, width, height):
    glViewport(0, 0, width, height)

def bh(x_start, y_start, x_end, y_end, res):
    dx = abs(x_end - x_start)
    dy = abs(y_end - y_start)
    pk = 2 * dy - dx
    x_coordinates = np.array([])
    y_coordinates = np.array([])
    for i in range(0, dx + 1):
        x_coordinates = np.append(x_coordinates, x_start)
        y_coordinates = np.append(y_coordinates, y_start)
        if x_start < x_end:
            x_start = x_start + 1

        else:
            x_start = x_start - 1
        if pk < 0:
            pk = pk + 2 * dy
        else:
            if y_start < y_end:
                y_start = y_start + 1
            else:
                y_start = y_start - 1
            pk = pk + 2 * dy - 2 * dx
    return toNVC(x_coordinates, y_coordinates, res)


def main():

    vertex_src = """
   #version 330

   layout(location=0) in vec2 aPos;
```

```
   void main(){

    gl_Position =vec4(aPos,0.0f,1.0f);

   }

  """

    fragment_src = """

  #version 330

  out vec4 FragColor;

  void main(){
    FragColor =vec4 (1.0f,1.0f,0.0f,1.0f);

  }

  """
    if not glfw.init():
        raise Exception("glfw cannot be initialised")

    window = glfw.create_window(RESOLUTION, RESOLUTION, "LAB2
Bresenham", None, None)

    if not window:
        glfw.terminate()
        raise Exception("glfw window cannot be created!")

    # glfw.set_window_pos(window,100,100)

    glfw.set_window_size_callback(window, window_resize)
    glfw.make_context_current(window)

    temp = bh(-500, -500, 250, 250, RESOLUTION)

    vertices = np.array(temp, dtype=np.float32)

    render_count = round(len(temp))

    print(temp)
    indices = np.array([i for i in range(1, render_count + 1)],
dtype=np.uint32)

    shader = compileProgram(
        compileShader(vertex_src, GL_VERTEX_SHADER),
        compileShader(fragment_src, GL_FRAGMENT_SHADER),
    )
```

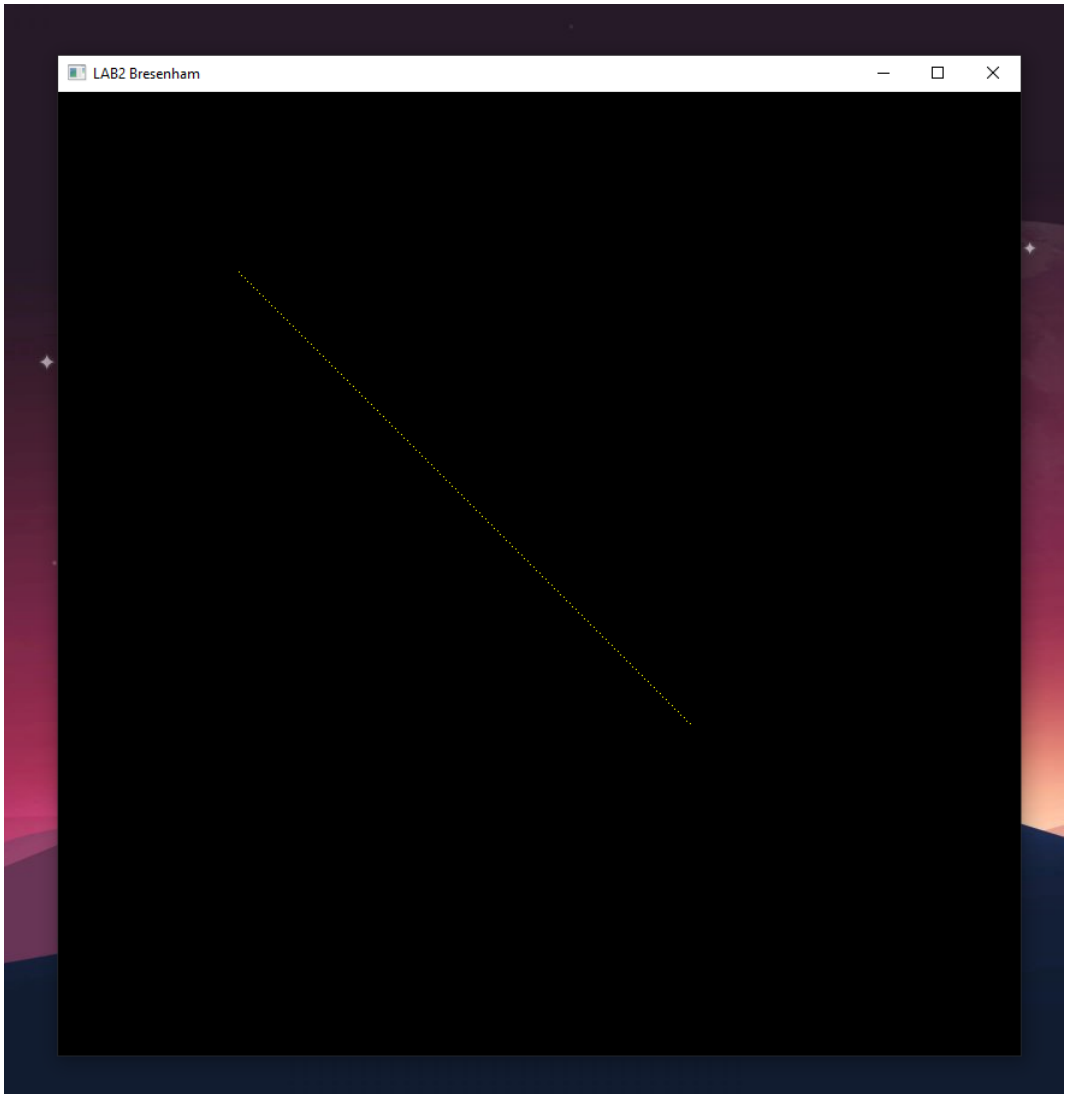
```
    vertex_buffer_object = glGenBuffers(1)
    glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer_object)
    glBufferData(GL_ARRAY_BUFFER, vertices.nbytes, vertices,
GL_STATIC_DRAW)

    element_buffer_object = glGenBuffers(1)
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, element_buffer_object)
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.nbytes, indices,
GL_STATIC_DRAW)

    glEnableVertexAttribArray(0)
    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0,
ctypes.c_void_p(0))

    glUseProgram(shader)


    while not glfw.window_should_close(window):
        glfw.poll_events()

        glDrawElements(GL_POINTS, len(indices), GL_UNSIGNED_BYTE, None)

        glfw.swap_buffers(window)

    glfw.terminate()


main()
```

4.3. Output

4.3.1. **For slope >1** (Joining points (-500, -500) & (250,250))



4.3.1. **For slope <1** (Joining points (-500, 500) & (250, -250))



Chapter 5: Mid-point Line Drawing Algorithm

5.1. Algorithm for Mid- point Line Drawing:

Step 1: Consider initial point (x0, y0) and final point (x1, y1,) as the starting and end points

Step 2: Calculate dx and dy of the given points as

dx= x1-x0

dy= y1-y0

Step 3: Check if slope of the line is positive or negative by:

If dx> dy and dy != 0

Decide=0

Initial decision parameter (pk) = dx – (dy / 2)

Else

Decide= 1

Initial decision parameter (pk) = dy – (dx / 2)

Step 4: At each point along the line the value of decide is checked as

a. If decide =1
    i. x = x + 1
    ii. if pk < 0
        pk= pk + dy
    iii. else
        pk = pk + (dy – dx)
        y = y + 1
b. Else
    i. Y = y – 1
    ii. If pk < 0
        Pk = pk + dx
    iii. Else
        Pk = pk + (dx – dy)
        X = x+1

Step 5: Repeat Step 4 until the following condition is fulfilled

a. If decide =1 then

Condition: x< x1

b. Else

Condition: y > y1

5.2. Source Code

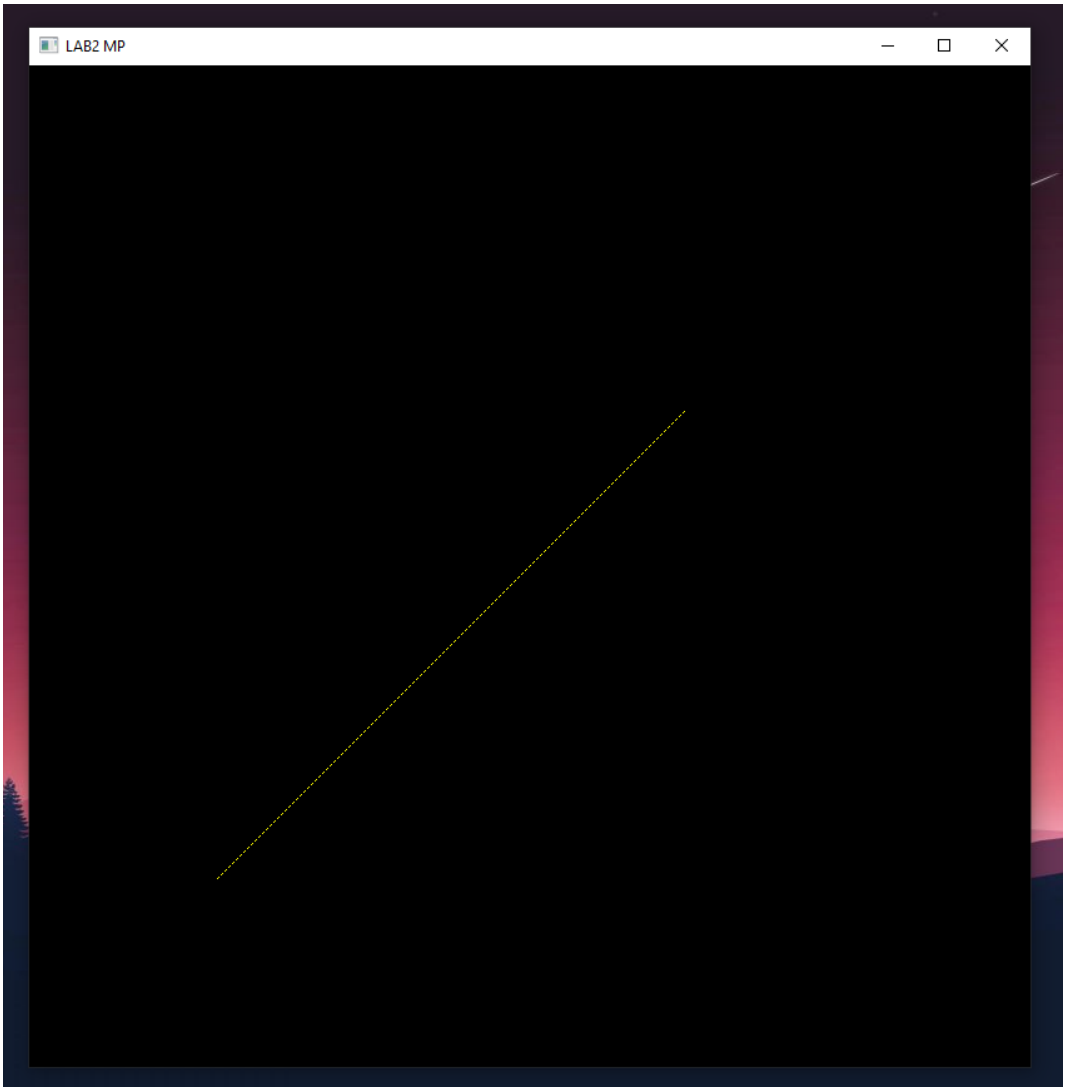
```
import glfw
import numpy as np
from OpenGL.GL import *
from OpenGL.GL.shaders import compileProgram, compileShader
from helpers import toNVC


RESOLUTION = 800


def window_resize(window, width, height):
    glViewport(0, 0, width, height)
def mp(x0, y0, x1, y1, res):
  dx = x1 - x0
  dy = y1 - y0
  x = x0
  y = y0
  if dx > dy and dy != 0:
   decide = 0
   pk = dx - (dy / 2)
  else:
    decide = 1
    pk = dy - (dx / 2)
  x_coordinates = np.array([])
  y_coordinates = np.array([])
  print(y > y1)
  while (x < x1) if (decide) else (y > y1):
    print("hi")
    x_coordinates = np.append(x_coordinates, x)
    y_coordinates = np.append(y_coordinates, y)
    if decide:
      x = x + 1
      print("hi")
      if pk < 0:
       pk = pk + dy
      else:
        pk = pk + (dy - dx)
```

```
        y = y + 1
    else:
      y = y - 1
      if pk < 0:
       pk = pk + dx

      else:
        pk = pk + (dx - dy)
        x = x + 1
  return toNVC(x_coordinates, y_coordinates, res)


def main():

    vertex_src = """
   #version 330

   layout(location=0) in vec2 aPos;

   void main(){

    gl_Position =vec4(aPos,0.0f,1.0f);

   }

  """

    fragment_src = """

  #version 330

  out vec4 FragColor;

  void main(){
    FragColor =vec4 (1.0f,1.0f,0.0f,1.0f);

  }

  """
    if not glfw.init():
        raise Exception("glfw cannot be initialised")

    window = glfw.create_window(RESOLUTION, RESOLUTION, "LAB2 MP", None,
None)

    if not window:
        glfw.terminate()
        raise Exception("glfw window cannot be created!")
```

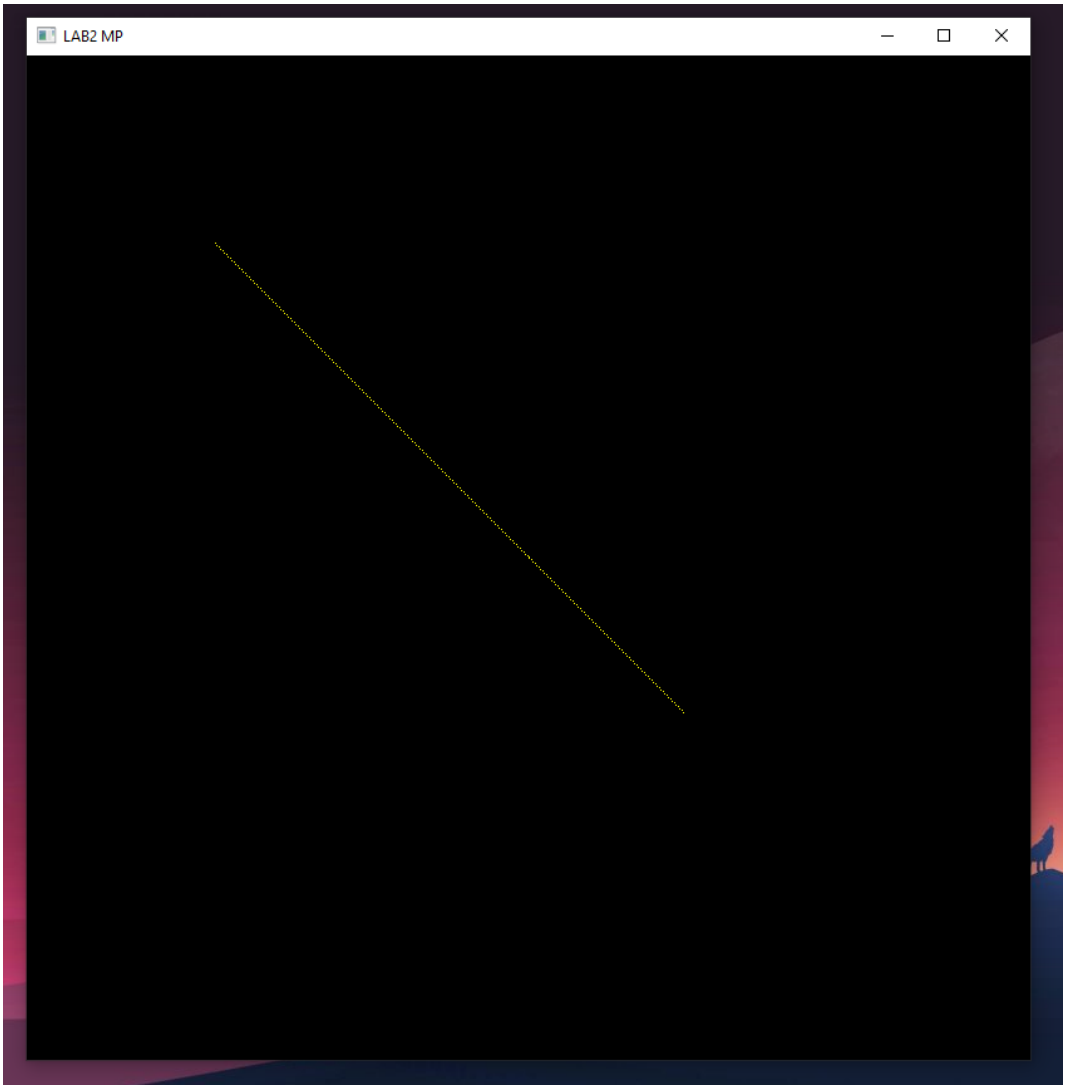
```
    # glfw.set_window_pos(window,100,100)

    glfw.set_window_size_callback(window, window_resize)
    glfw.make_context_current(window)

    temp = mp(-500, -500, 250, 250, RESOLUTION)

    vertices = np.array(temp, dtype=np.float32)

    render_count = round(len(temp))

    print(temp)
    indices = np.array([i for i in range(1, render_count + 1)],
dtype=np.uint32)

    shader = compileProgram(
        compileShader(vertex_src, GL_VERTEX_SHADER),
        compileShader(fragment_src, GL_FRAGMENT_SHADER),
    )

    vertex_buffer_object = glGenBuffers(1)
    glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer_object)
    glBufferData(GL_ARRAY_BUFFER, vertices.nbytes, vertices,
GL_STATIC_DRAW)

    element_buffer_object = glGenBuffers(1)
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, element_buffer_object)
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.nbytes, indices,
GL_STATIC_DRAW)

    glEnableVertexAttribArray(0)
    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0,
ctypes.c_void_p(0))

    glUseProgram(shader)


    while not glfw.window_should_close(window):
        glfw.poll_events()

        glDrawElements(GL_POINTS, len(indices), GL_UNSIGNED_BYTE, None)

        glfw.swap_buffers(window)

    glfw.terminate()


main()
```

5.3. Output

5.3.1 **For slope >1** (Joining points (-500,-500) & (250,250))



5.3.2 **For slope <1** (Joining points (-500,500) & (250,-250))



Chapter 6: Conclusion

Through the completion of this lab work, we were able to gain a better understanding of how the three-line drawing algorithms (DDA, Bresenham and Mid-Point) work and how to implement them into GLFW and OpenGL in order to draw lines on the screen. To do that, we had to convert the co-ordinates obtained from the algorithms into normalized viewing coordinates (NVC) and then use those coordinates to draw the lines. Normalizing the coordinates was necessary as any coordinate exceeding either -1 or 1 in any direction is automatically clipped by OpenGL. Therefore, normalizing the coordinates helped us to draw lines without any problem.. In the end, we successfully implemented the three algorithms and were able to draw lines on the screen.