**Faculty of Computing**

**CS-361: Computer Graphics**

**Class: BSCS-12ABC & SE12AB**

**Lab 01: Introduction to Computer Graphics Setup and Line Drawing Algorithms**

**CLO-02:** Develop 2D and 3D graphical applications using programming libraries and tools.

**CLO-03:** Implement algorithms for rendering, transformations, and animations.

**Date: 28th Jan 2025 Time: 02:00 PM – 04:50 PM**

**Instructor: Dr. Sidra Sutana Lab Engineer: Mr. Aftab Farooq**

**Name:** Ahmed Mohiuddin Shah

**CMS ID:** 415216

**Section:** BSCS-12-A

**Lab:** 1

# Lab 01: Introduction to Computer Graphics Setup and Line Drawing Algorithms

**Lab Objective:**

The aim of this lab is to:

* Set up the environment for developing Computer Graphics applications using OpenGL.
* Understand and implement basic line drawing algorithms like DDA (Digital Differential Analyzer) and Bresenham’s Line Algorithm.

## Tools/Software Requirement:

* Operating System**:**
  + Windows / macOS / Linux (Ubuntu recommended)
* Development Environment**:**
  + Windows**:** Code::Blocks or Visual Studio
  + macOS**:** Xcode
  + Linux**:** GCC and g++ compilers
* Graphics Libraries**:**
  + OpenGL (built-in on macOS and Linux, available in Windows IDEs)
  + GLUT (OpenGL Utility Toolkit)
  + GLEW (OpenGL Extension Wrangler Library)
* Package Manager (for macOS/Linux)**:**
  + Homebrew (macOS): brew install freeglut glew
  + APT (Linux): sudo apt-get install freeglut3-dev glew-utils
* Compilers**:**
  + Windows**:** MinGW (for Code::Blocks) or Microsoft C++ Compiler (for Visual Studio)
  + macOS/Linux**:** GCC/G++

## Task :01 - Setting up OpenGL

* 1. *Prerequisites*

To begin Computer Graphics programming, you’ll need:

* + - A working knowledge of C/C++ programming.
    - Familiarity with an IDE such as Code::Blocks, Visual Studio, or Xcode.
    - OpenGL libraries installed.

## Installing OpenGL on Your System For Windows:

1. Install Code::Blocks or Visual Studio.
2. Install MinGW (if using Code::Blocks) and ensure OpenGL libraries are available.
3. Download GLUT and GLEW libraries and configure them in your IDE:
   * Place glut.h in the include/GL folder of your MinGW or Visual Studio.
   * Add libglut32.a and libglew.a to the lib directory.

## For macOS:

1. Install Xcode from the Mac App Store.
2. Open Terminal and install Command Line Tools:

xcode-select --install

1. Optionally, install FreeGLUT via Homebrew:

brew install freeglut

## For Linux (Ubuntu):

1. Install OpenGL development libraries:

sudo apt-get install freeglut3-dev glew-utils

## Writing Your First OpenGL Program

Write a simple program to open a window using OpenGL:

#include <GL/glut.h>

void display() { glClear(GL\_COLOR\_BUFFER\_BIT); glFlush();

}

int main(int argc, char\*\* argv) { glutInit(&argc, argv); glutInitDisplayMode(GLUT\_SINGLE); glutInitWindowSize(400, 400);

glutInitWindowPosition(100, 100); glutCreateWindow("OpenGL Setup"); glutDisplayFunc(display); glutMainLoop();

return 0;

}

* Compile**:** Use your IDE or Terminal to compile the program.
* Run**:** Execute the compiled program to open an OpenGL window.

## Task 2: Line Drawing Algorithms

* 1. *Introduction to Line Drawing*

In Computer Graphics, a line is a sequence of pixels drawn between two endpoints. The challenge is to decide which pixels should be set to approximate the appearance of a straight line.

* 1. *DDA (Digital Differential Analyzer) Algorithm*

The DDA algorithm increments the starting point of a line by small steps and calculates the intermediate points to form a line.

Algorithm Steps:

* + 1. Calculate the difference between the two points, *dx* and *dy*.
    2. Determine the number of steps required based on the maximum value of *dx* or *dy*.
    3. Increment *x* and *y* values by small increments in each step.
    4. Set the corresponding pixel.

## Example Code (C++):

#include <GL/glut.h> #include <math.h>

void drawLine(float x1, float y1, float x2, float y2) { float dx = x2 - x1;

float dy = y2 - y1;

float steps = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy); float xIncrement = dx / steps;

float yIncrement = dy / steps; float x = x1, y = y1;

glBegin(GL\_POINTS);

for (int i = 0; i <= steps; i++) { glVertex2i(round(x), round(y)); x += xIncrement;

y += yIncrement;

}

glEnd(); glFlush();

}

void display() { glClear(GL\_COLOR\_BUFFER\_BIT);

drawLine(50, 50, 200, 200); // Draw a line from (50, 50) to (200, 200)

}

int main(int argc, char\*\* argv) { glutInit(&argc, argv); glutCreateWindow("DDA Line Drawing"); glutDisplayFunc(display); glutMainLoop();

return 0;

}

* 1. *Bresenham’s Line Algorithm*

Bresenham’s Line Algorithm is more efficient than DDA because it uses only integer addition and subtraction, avoiding floating-point operations.

Algorithm Steps:

* + 1. Compute the differences *dx* and *dy*.
    2. Set the decision parameter *p=2dy−dx*
    3. For each pixel, if *p < 0*, increment *x* and keep *y* the same. Otherwise, increment both *x* and *y*, updating *p* accordingly.

## Example Code (C++):

#include <GL/glut.h>

void drawLine(int x1, int y1, int x2, int y2) { int dx = abs(x2 - x1), dy = abs(y2 - y1); int sx = (x1 < x2) ? 1 : -1;

int sy = (y1 < y2) ? 1 : -1; int err = dx - dy;

glBegin(GL\_POINTS);

while (x1 != x2 || y1 != y2) { glVertex2i(x1, y1);

int e2 = 2 \* err;

if (e2 > -dy) { err -= dy; x1 += sx; } if (e2 < dx) { err += dx; y1 += sy; }

}

glEnd(); glFlush();

}

void display() { glClear(GL\_COLOR\_BUFFER\_BIT);

drawLine(100, 100, 400, 300); // Draw a line from (100, 100) to (400, 300)

}

int main(int argc, char\*\* argv) { glutInit(&argc, argv); glutCreateWindow("Bresenham Line Drawing"); glutDisplayFunc(display);

glutMainLoop(); return 0;

}

## Lab Tasks:

1. **Environment Setup**:
   * Install OpenGL and the required libraries on your system.
   * Write a basic OpenGL program to open a window.

## Implement Line Drawing Algorithms:

* + Write code for the **DDA Line Algorithm**.
  + Write code for the **Bresenham’s Line Algorithm**.
  + Experiment with different starting and ending points for the lines.

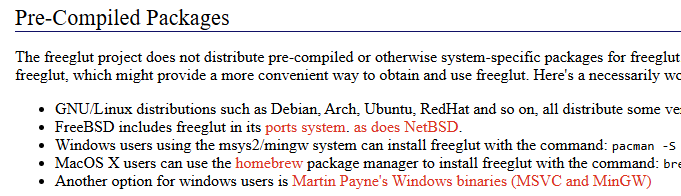
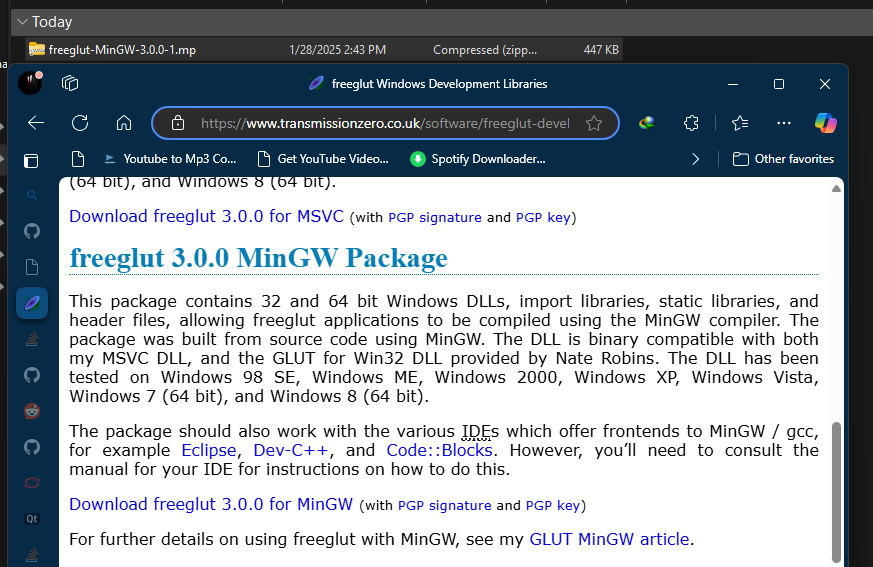
# Deliverables:

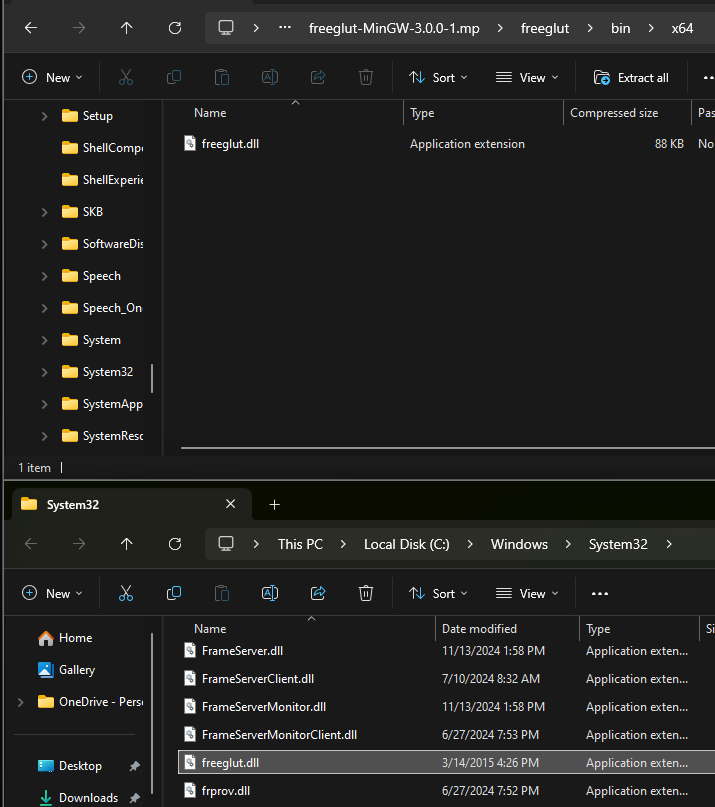
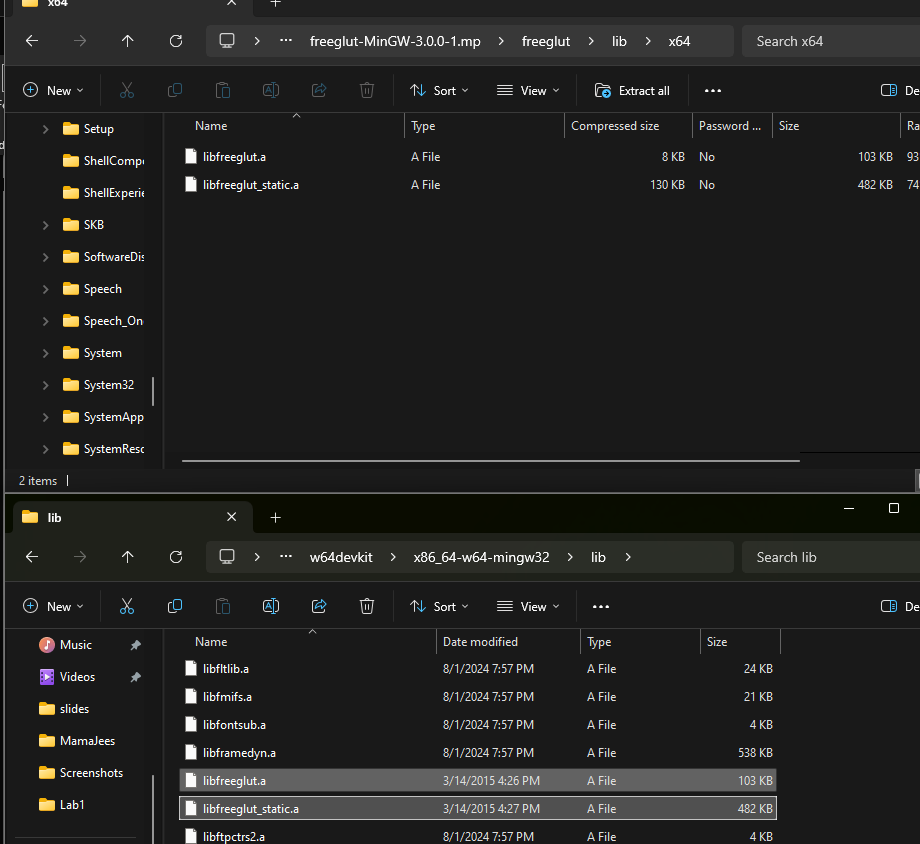
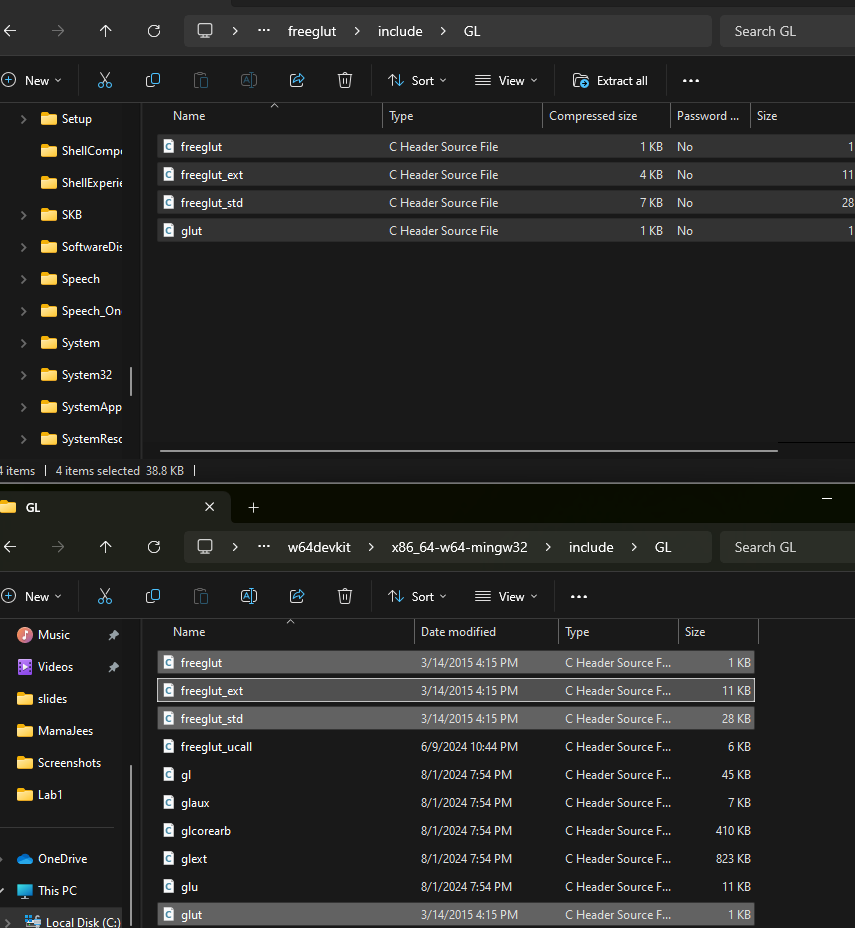
* Compile a single word document by filling in the solution part and submit this Word file on LMS
* Submit your code files (.cpp) for both the DDA and Bresenham’s algorithms.
* Include screenshots of the program outputs.
* Submit your Lab Word File and code files seperately on submission link.

## Solution:

### How Did I Install The glut library

I am using w64devkit for my C/C++ and VSCode As my Development Environment.

1. First I downloaded the binaries .zip from [freeglut Windows Development Libraries](https://www.transmissionzero.co.uk/software/freeglut-devel/) for MinGW. (<https://www.transmissionzero.co.uk/software/freeglut-devel/>). I got this link from the original freeglut website and followed the link for the windows pre-compiled binaries download. 
2. 
3. I then Extracted the files into a folder. A black screen with white text

   Description automatically generated
4. I then Copied the x64 version of the .dll file in the bin folder to the Windows/System32 folder. We can also put the dll inside our code folder. 
5. I Copied the x64 version of the lib folder files in to the mingw lib folder. 
6. And finally I copied the include/GL folder contents of the freeglut folder into the include/GL folder of the Mingw. 
7. After that I ran the test code that was provided in the doc. A screen shot of a computer program

   Description automatically generatedA screen shot of a computer program

   Description automatically generated
8. The code works with both the *glut.h*  and *freeglut.h* headers.
9. 
10. I did notice to link the proper libraries and header I needed to give the following flags: 
11. *-lfreeglut -lopengl32 -lglu32*

### Writing First OpenGL Program:

This program displays a simple black screen:

**Code:**

#include <GL/glut.h>

void display()

{

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glFlush();

}

int main(int argc, char \*\*argv)

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE);

    glutInitWindowSize(500, 500);

    glutInitWindowPosition(100, 100);

    glutCreateWindow("OpenGL Test");

    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

**Output:**

**A screenshot of a computer

Description automatically generated**

### Using Digital Differential Analyzer Algorithm to draw a line:

The Code provided in the original doc was not displaying line correctly so I had to modify the code a bit.

**Code:**

#include <GL/glut.h>

#include <math.h>

void drawLine(float x1, float y1, float x2, float y2)

{

    float dx = x2 - x1;

    float dy = y2 - y1;

    float steps = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy);

    float xIncrement = dx / steps;

    float yIncrement = dy / steps;

    float x = x1, y = y1;

    glBegin(GL\_POINTS);

    for (int i = 0; i <= steps; i++)

    {

        glVertex2i(round(x), round(y));

        x += xIncrement;

        y += yIncrement;

    }

    glEnd();

    glFlush();

}

void display()

{

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(1.0f, 0.0f, 0.0f); // Set line color to red

    drawLine(50, 50, 200, 200); // Draw a line from (50, 50) to (200, 200)

}

int main(int argc, char \*\*argv)

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(500, 500); // Here we set the window size

    glutCreateWindow("DDA Line Drawing");

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black

    glMatrixMode(GL\_PROJECTION); // Switch to projection matrix

    glLoadIdentity(); // Reset the matrix

    gluOrtho2D(0, 500, 0, 500); // Set orthographic projection

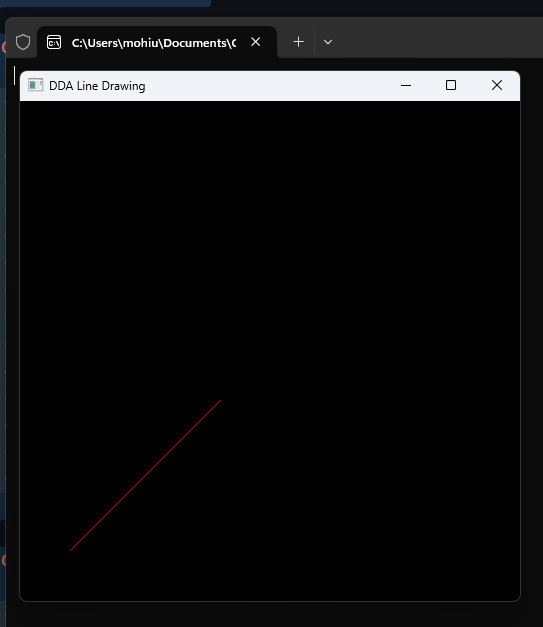
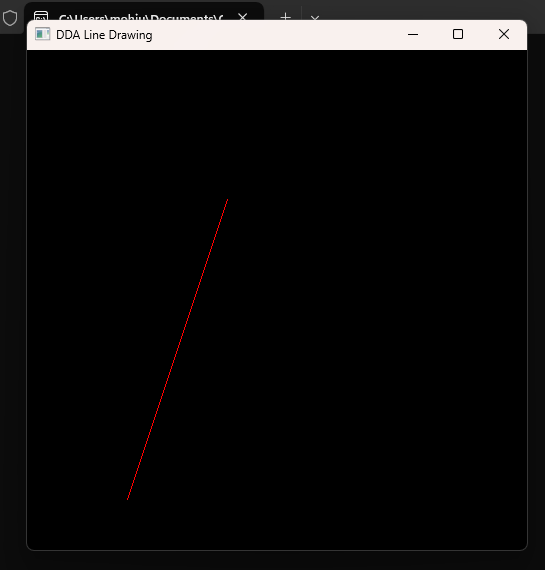
    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

**Output:**

****

# Bresenham’s Line Algorithm

This code also had troubles in running so I had to modify it a bit.

**Code:**

#include <GL/glut.h>

void drawLine(int x1, int y1, int x2, int y2)

{

    int dx = abs(x2 - x1), dy = abs(y2 - y1);

    int sx = (x1 < x2) ? 1 : -1;

    int sy = (y1 < y2) ? 1 : -1;

    int err = dx - dy;

    glBegin(GL\_POINTS);

    while (x1 != x2 || y1 != y2)

    {

        glVertex2i(x1, y1);

        int e2 = 2 \* err;

        if (e2 > -dy)

        {

            err -= dy;

            x1 += sx;

        }

        if (e2 < dx)

        {

            err += dx;

            y1 += sy;

        }

    }

    glEnd();

    glFlush();

}

void display()

{

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(0.0f, 1.0f, 0.0f);  // Set Line Color to green

    drawLine(100, 100, 400, 300); // Draw a line from (100, 100) to (400, 300)

}

int main(int argc, char \*\*argv)

{

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(500, 500);

    glutCreateWindow("Bresenham Line Drawing");

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(0.0, 500.0, 0.0, 500.0);

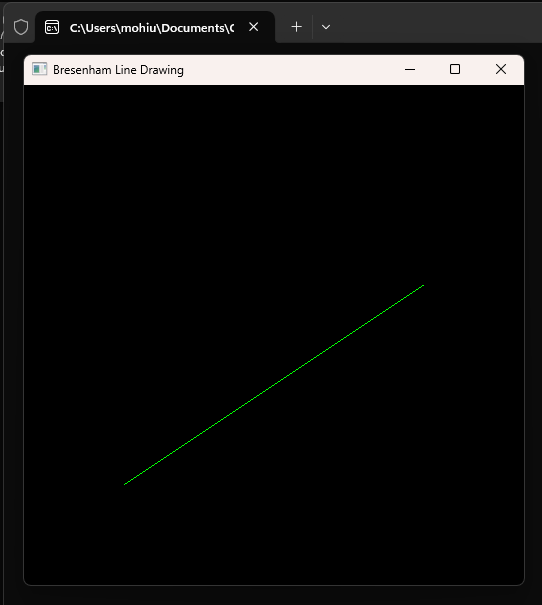
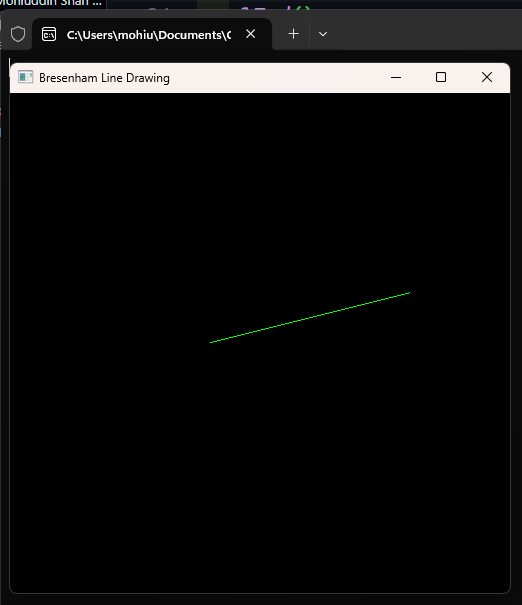
    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}

**Output:**

**** ****

# Lab Rubrics

**Your Lab 1 will be graded out of 5 for each rubric according to the following rubrics**

Lab Rubrics for Lab 1(Introduction to Computer Graphics Setup and Line Drawing Algorithms)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **Assessment** | **Unacceptable (0 Marks)** | **Does Not Meet Expectation s (1/2**  **Marks)** | **Meets Expectations(3/4 Marks)** | **Exceeds Expectations(5 Marks)** |
| **1** | **Illustrating the basic understanding of semantics and syntax**  **(CLO2, PLO3)** |  | The student is unable to demonstrate the understanding of syntax of C language and is unable towrite an  executable code. | The student demonstrates some understanding of syntax of C language and is able to write a code with few errors. | The student demonstrates good understanding of syntax of C language and is able to write executable code without help |
|  |  |  | The student is not able to understand the structure ofa program at  all. | The student is able to understand the structure but still learning the syntax. | The student is able to understand the structure and is able to identify problems in the code when introduced |
| **2** | **Software Tool Usage**  **(CLO3-PLO5)** | The student did not submit any work.  OR  The student plagiarized the solution and/or used unfair means. | The student demonstrates alack of understanding of tool usage.  Implementation has syntax/semantic/runtime errors, and the student is unable to debug and correctthe errors.  The code has inadequate comments and variable names and does not adhereto the coding standards.  No Error handling has beenperformed.  Documentation is poorlystructured. | The student demonstrates some understanding of tool usage.  The codes are correct in terms of their syntax, however, the program output is not always correct in all test cases.  The code has limited comments and inconsistent variable names and may not adhere to the coding standards.  Some Error handling has been performed.  Documentation is adequately structured. | The student demonstrates a good understanding of tool usage.  Furthermore, his/her coding is complete and functional, and the program output is correct in all test cases.  The code has sufficient comments and consistent variable names and reasonably adhere to the coding standards.  Adequate Error handling has been performed.  Documentation is well structured. |