Date: 02/09/2025
Experiment No: 04
Experiment Name: Implement the Bresenham's line drawing algorithm for |m| >= 1 and |m| < 1.

Introduction

Bresenham's Line Drawing Algorithm is a fast and efficient way to draw straight lines on a computer screen. It works using only integer calculations, which makes it quicker than other methods like DDA. The algorithm decides which pixel gives the best path for the line between two points.

In this experiment, we implement Bresenham's algorithm for two cases:

1. |m| < 1 - line with a gentle slope
2. |m| ≥ 1 - line with a steep slope

This helps draw accurate and smooth lines in all directions on the screen.

Description:

The Bresenham's Line Drawing Algorithm is used to draw straight lines using only integer operations, making it fast and efficient for computer graphics.
To draw the letter "Z", we use this algorithm to draw three straight lines:

1. Top Horizontal Line:
   Draw a straight line from the top-left point to the top-right point of the "Z".

2. Diagonal Line:
   Connect the top-right point to the bottom-left point with a diagonal line.

3. Bottom Horizontal Line:
   Draw a straight line from the bottom-left point to the bottom-right point.

Each of these lines is plotted pixel by pixel based on the decision parameter (p) used in Bresenham's algorithm, which decides whether the next pixel moves horizontally or diagonally.

As a result, the three lines together form the letter "Z" with sharp and smooth edges on the screen.

Code:

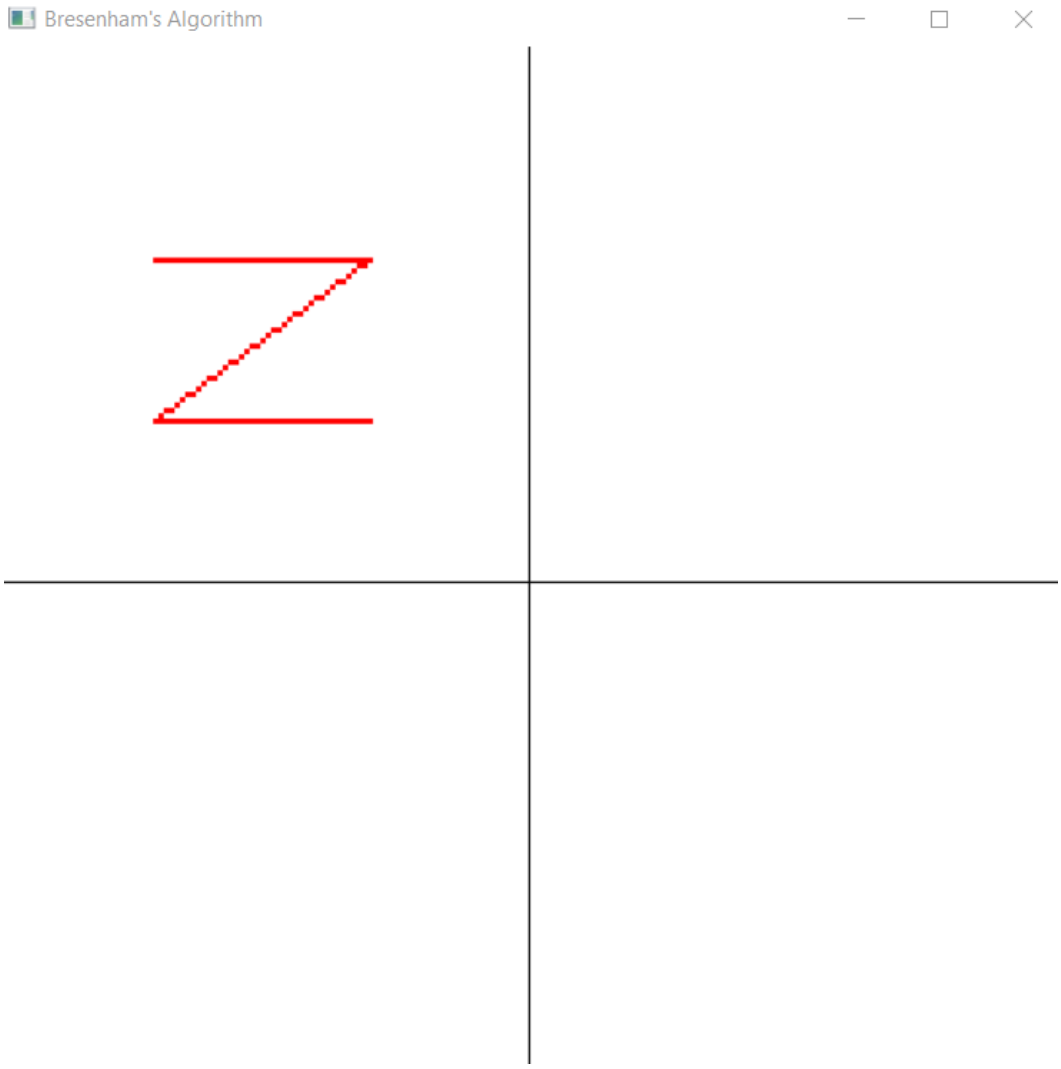
```
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <windows.h>
using namespace std;

void init(void)
```

```
{
   glClearColor(1, 1, 1, 1);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(-100, 100, -100, 100);
}

void bresenham(int x0, int y0, int x1, int y1){
   if (x0 > x1 || y0 > y1){
      swap(x0, x1);
      swap(y0, y1);
   }

   int dx = abs(x1 - x0);
   int dy = abs(y1 - y0);
   float m = dy / (float)dx;

   int x = x0;
   int y = y0;

   int x_inc = (x1 >= x0) ? 1 : -1;
   int y_inc = (y1 >= y0) ? 1 : -1;

   glBegin(GL_POINTS);

   if (m < 1){
      int p = 2 * dy - dx;

      while (x <= x1){
         glVertex2i(x, y);

         x += x_inc;

         if (p < 0){
            p += 2 * dy;
         }
         else
            y += y_inc;
            p += 2 * dy - 2 * dx;
         }
      }
   }
   else{
      int p = 2 * dx - dy;

      while (x <= x1){
         glVertex2i(x, y);
```

```
            y += y_inc;

            if (p < 0){
                p += 2 * dx;
            }
            else{
                x += x_inc;
                p += 2 * dx - 2 * dy;
            }
        }
    }

    glEnd();
}

void display(){
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_LINES);
    glColor3f(0, 0, 0);
    glVertex2f(100, 0);
    glVertex2f(-100, 0);
    glVertex2f(0, 100);
    glVertex2f(0, -100);
    glEnd();

    glColor3f(1, 0, 0);
    glPointSize(3);

    bresenham(-70, 60, -30, 60);
    bresenham(-30, 60, -70, 30);
    bresenham(-70, 30, -30, 30);

    glFlush();
}

int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Bresenham’s Algorithm");

    init();
    glutDisplayFunc(display);
```

```
    glutMainLoop();
    return 0;
}
```

Output:



Conclusion:

Bresenham's Line Drawing Algorithm efficiently draws straight lines on a pixel screen using only integer calculations. By handling both gentle slopes ($|m| < 1$) and steep slopes ($|m| \geq 1$), it can draw lines accurately in any direction. This makes it fast, precise, and very useful for creating shapes and letters in computer graphics.