Date: 26/08/2025
Experiment No: 03
Experiment Name: Implementing and Visualizing the DLD and DDA Algorithm.

Introduction:

In computer graphics, drawing straight lines on a screen is an essential task. Two commonly used algorithms for this purpose are the Digital Line Drawing (DLD) algorithm and the Digital Differential Analyzer (DDA) algorithm. These algorithms use mathematical calculations to decide which pixels should be turned on to best represent a straight line between two points. Implementing and visualizing these methods helps us understand how computers approximate lines and how efficiency and accuracy can differ between algorithms.

Description:

Digital Differential Analyzer (DDA):

The Digital Differential Analyzer (DDA) algorithm is a line drawing method used in computer graphics. It works by calculating the step size in the x-direction and y-direction between the starting point and the ending point of a line. Then, it gradually plots the points in small steps to make the line appear straight on the screen.

Steps of DDA Algorithm

1. Input the coordinates of the starting point (x1, y1) and ending point (x2, y2).
2. Calculate dx and dy:

    a) dx = x2 – x1
    b) dy = y2 – y1

3. Find the number of steps needed:

    a) steps = maximum of |dx| or |dy|
    (This ensures the line is plotted smoothly.)

4. Calculate increments for each step:

    a) x_inc = dx / steps
    b) y_inc = dy / steps

5. Initialize starting point:

    a) x = x1, y = y1

6. Plot the points:

    a) For i = 1 to steps:

```
i.   Plot(x, y)
ii.  x = x + x_inc
iii. y = y + y_inc
```

Digital Line Drawing (DLD) Algorithm:
The Digital Line Drawing (DLD) algorithm is one of the basic methods to draw a straight line on a computer screen. It uses the standard equation of a line (y = mx + c) and plots the pixels one by one. Depending on the slope (m) of the line, the algorithm either increments x and calculates y, or increments y and calculates x.

Although it is simple and easy to understand, it requires floating-point multiplication and rounding

<u>Steps of DLD Algorithm</u>

1. Take input for the two endpoints of the line (x1, y1) and (x2, y2).
2. Check if the line is vertical (x1 = x2)
   a) If yes, just plot all points from y1 to y2 at the same x-value.
3. Find the slope (m):
   a) m = (x2−x1) / (y2−y1)

4. Find the intercept (c):
   a) b = y1 − m·x1
5. If the slope is small (|m| ≤ 1):

a) Move along x-axis (increase x by 1 each step).
b) For each x, calculate y = m·x + b.
c) Plot the pixel at (x, round(y)).

6. If the slope is large (|m| > 1):

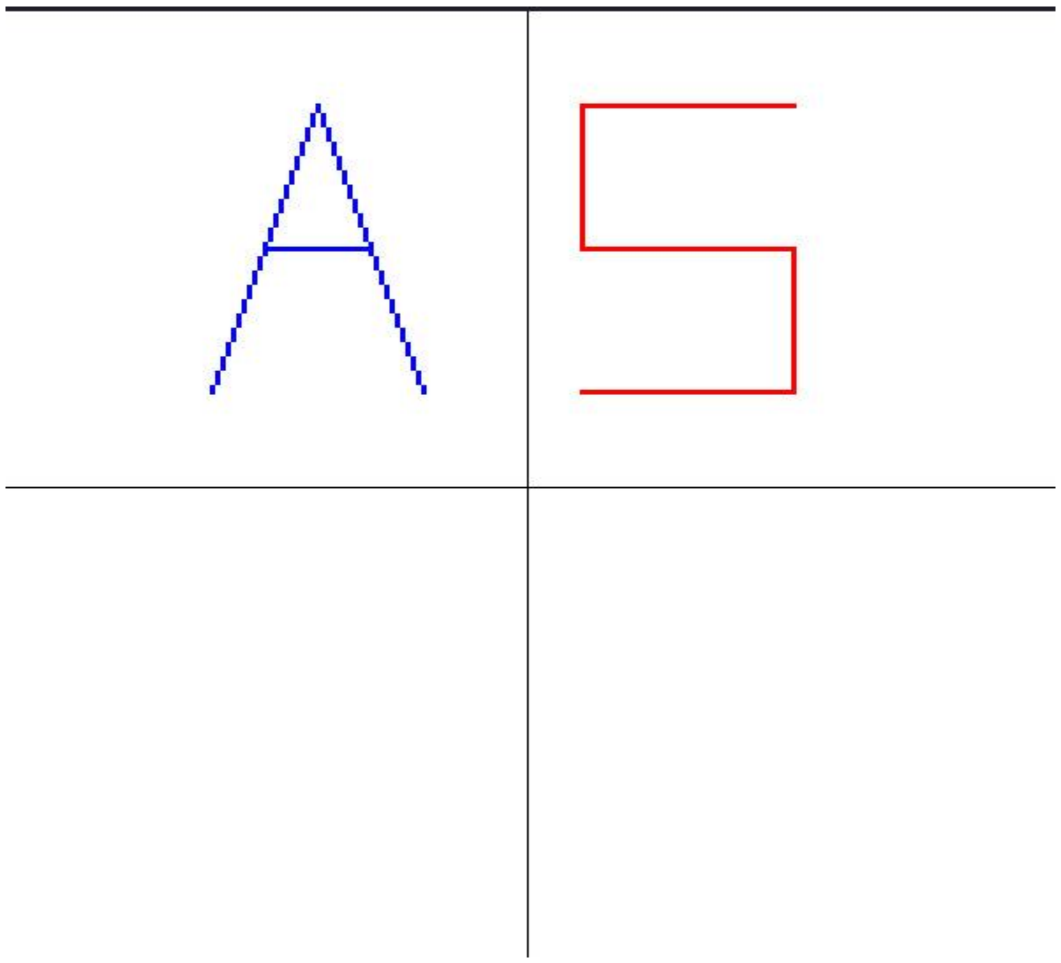
a) Move along y-axis (increase y by 1 each step).
b) For each y, calculate x = (y – b) / m.
c) Plot the pixel at (round(x), y)

<u>Code:</u>

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

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

```
// DDA Algorithm
void DDA(float x1, float y1, float x2, float y2) {
    float dx = x2 - x1;
    float dy = y2 - y1;

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

    float x_inc = dx / steps;
    float y_inc = dy / steps;

    float x = x1;
    float y = y1;

    glPointSize(3.0);
    glBegin(GL_POINTS);
    for (int i = 0; i <= steps; i++) {
        glVertex2f(x, y);
        x += x_inc;
        y += y_inc;
    }
    glEnd();
}

// DLD Algorithm
void DLD(float x1, float y1, float x2, float y2) {
    float dx = x2 - x1;
    float dy = y2 - y1;

    float m = dy / dx;
    float b = y1 - m * x1;

    glPointSize(3.0);
    glBegin(GL_POINTS);

    if(fabs(m) <= 1) {
        float start = (x1 < x2) ? x1 : x2;
        float end   = (x1 < x2) ? x2 : x1;

        for(float x = start; x <= end; x += 1.0) {
            float y = m * x + b;
            glVertex2f(x, round(y));
        }
    } else {
        float start = (y1 < y2) ? y1 : y2;
        float end   = (y1 < y2) ? y2 : y1;
        for(float y = start; y <= end; y += 1.0) {
            float x = (y - b) / m;
            glVertex2f(round(x), y);
        }
    }

    glEnd();
}

//display
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);
    DDA(10, 80, 50, 80);
    DDA(10, 80, 10, 50);
    DDA(10, 50, 50, 50);
    DDA(50, 50, 50, 20);
    DDA(50, 20, 10, 20);


    glColor3f(0, 0, 1);

    DLD(-60, 20, -40, 80);
    DLD(-20, 20, -40, 80);
    DLD(-50, 50, -30, 50);

    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("DDA & DLD");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Output:



Conclusion:

In this experiment, we implemented and visualized the DDA and DLD line drawing algorithms. Both algorithms help draw straight lines on a computer screen by calculating which pixels to turn on. DDA uses incremental calculations for smooth plotting, while DLD uses the line equation directly. Visualizing these algorithms helps us understand how computers approximate lines and compare their efficiency and accuracy in graphics.