EXPERIMENT NO :1

OBJECTIVE: TO LEARN AND PERFORM GRAPHICS OPERATION ON DIGITAL DIFFERENTIAL ANALYZER ALGORITHM.

THEORY:

The Digital Differential Analyzer (DDA) is a simple and widely used algorithm for generating lines, primarily in computer graphics. It's used to rasterize a line by incrementally stepping along the line path and determining which pixels to turn on to approximate the line on a discrete grid or screen.

The basic idea behind the DDA algorithm is to calculate the incremental change in the x and y coordinates, and then use these increments to draw each pixel along the line path.

ALGORTIHM FOR DDA:

1. Input: You have two endpoints of the line, (x1, y1) and (x2, y2).

2. Calculate slope: Calculate the differences between x2 - x1 and y2 - y1 to determine the slope of the line. The slope indicates the rate of change in the y-coordinate for each unit change in the x-coordinate.

3. Determine Number of Steps: Determine the number of steps required to traverse from (x1, y1) to (x2, y2). This is typically done by finding the maximum difference between the x-coordinates and y-coordinates and using it as the number of steps.

4. Calculate Increment: Calculate the incremental change in x (xIncrement) and y (yIncrement) coordinates for each step. These increments are used to move along the line path.

5. Draw Pixels: Starting from the first endpoint (x1, y1), draw the pixel at the current coordinates (x, y), then update x by adding xIncrement and update y by adding yIncrement. Repeat this process for the determined number of steps.

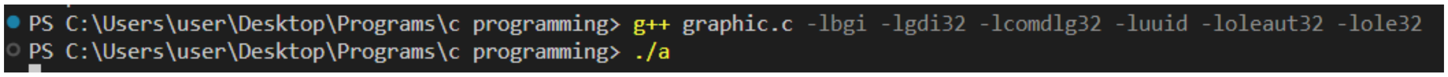
6. Rounding: During the calculations, you might need to round the x and y coordinates to the nearest integer, as screen pixels are discrete. This can be done using rounding functions or by adding 0.5 and then truncating the result.

The DDA algorithm is straightforward and efficient, but it can suffer from precision issues when the slope is very steep or close to zero. In such cases, the algorithm may miss or duplicate pixels. However, it provides a good introduction to line rasterization concepts and is often used as a teaching tool.

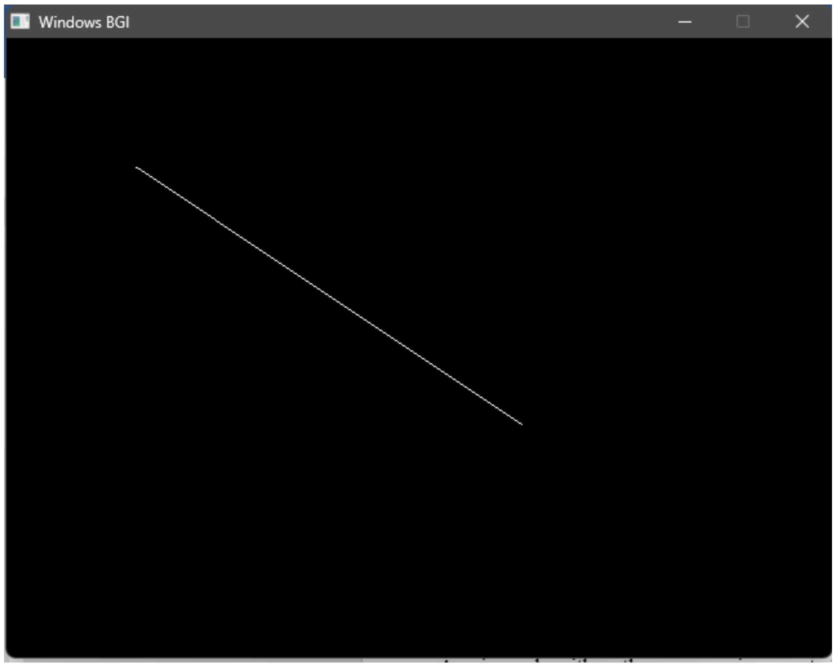
Implementation in C:

```
//Program

#include <stdio.h>
#include <graphics.h>

void drawLineDDA(int x1, int y1, int x2, int y2) {
    int dx = x2 - x1;
    int dy = y2 - y1;

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

    float x = x1, y = y1;

    for (int i = 0; i < steps; i++) {
        putpixel(x, y, WHITE);
        x += xIncrement;
        y += yIncrement;
    }
}

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, NULL);

    int x1 = 100, y1 = 100, x2 = 400, y2 = 300;

    drawLineDDA(x1, y1, x2, y2);

    getch();
    closegraph();
    return 0;
}
```

Output:



```
PS C:\Users\user\Desktop\Programs\c programming> g++ graphic.c -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32
PS C:\Users\user\Desktop\Programs\c programming> ./a
```



DISCUSSION:

In this lab, we implemented the Digital Differential Analyzer (DDA) algorithm in C to draw a line on a graphics window. The DDA algorithm calculates the coordinates of the line using incremental steps based on the differences in x and y coordinates. We defined a function drawLineDDA that takes the endpoints of the line as arguments and iterates through each step to plot the pixels along the line. After plotting all points, we closed the graphics window. While the DDA algorithm provides a basic method for line drawing, other algorithms like Bresenham's may offer better efficiency in certain cases.

CONCLUSION:

The above experiment in C programming, we have understood the concept of Digtal Differential Analyzer(DDA) Algorithm and how it work.

EXPERIMENT NO :2

OBJECTIVE: TO LEARN AND PERFORM GRAPHICS OPERATION ON BRESENHAM LINE ALGORITHM.

THEORY:

The Bresenham Line Algorithm is a widely used and efficient method for drawing a line between two points on a computer screen or in a digital image. It was developed by Jack E. Bresenham in 1962. The algorithm is designed to determine the coordinates of the pixels that should be turned on to approximate a straight line between two given points. It avoids the need for floating-point arithmetic, which can be computationally expensive, by using integer arithmetic operations like addition and subtraction.

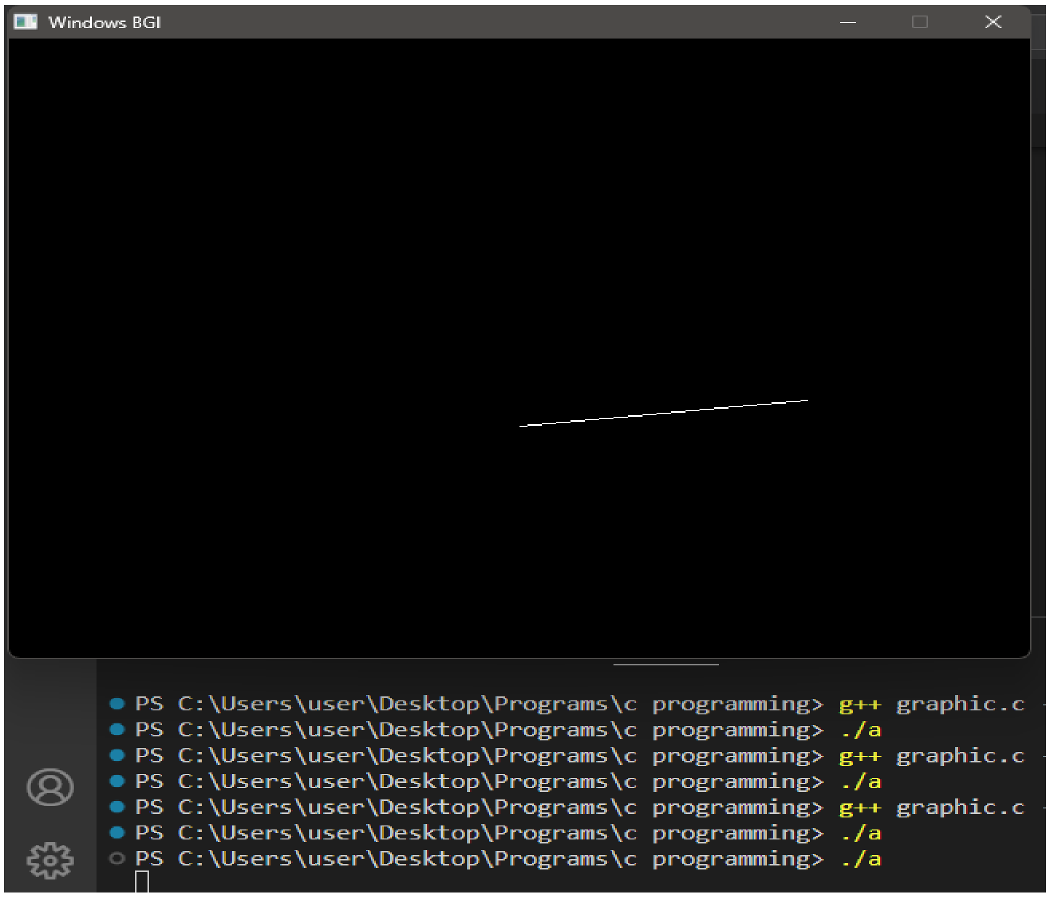
ALGORTIHM FOR BLA:

- Input: Two endpoints of the line, (x1, y1) and (x2, y2).
- Calculate Differences: Calculate differences dx = x2 - x1 and dy = y2 - y1.
- Determine Increments: Determine the increments increment_x and increment_y based on the sign of dx and dy. If dx and dy are positive, increments are 1; otherwise, -1.
- Initial Decision Parameter: Calculate the initial decision parameter p as 2 * dy - dx if the absolute value of dy is less than the absolute value of dx, otherwise p = 2 * dx - dy.
- Drawing: For each step from 1 to dx (or dy, depending on the slope):
    - Plot the pixel at (x, y).
    - If p < 0, increment x by increment_x and update p as p + 2 * dy.
    - Else, increment both x and y by their respective increments and update p as p + 2 * dy - 2 * dx.
- Termination: Repeat step 5 until the line is completely drawn.

Implementation in C:

```
//Program
#include <stdio.h>
#include <graphics.h>
void drawLineBresenham(int x1, int y1, int x2, int y2) {
   int dx = x2 - x1;
   int dy = y2 - y1;
   int increment_x = (dx > 0) ? 1 : -1;
   int increment_y = (dy > 0) ? 1 : -1;
   dx = abs(dx);
   dy = abs(dy);
   int p;
   if (dy <= dx)
        p = 2 * dy - dx;
   else
      p = 2 * dx - dy;
   int x = x1, y = y1;
   for (int i = 0; i <= dx; i++) {
      putpixel(x, y, WHITE);
      if (dy <= dx) {
            x += increment_x;
            if (p < 0)
            p += 2 * dy;
            else {
                y += increment_y;
                p += 2 * (dy - dx);
            }
        } else {
            y += increment_y;
            if (p < 0)  p += 2 * dx;
            else {
                x += increment_x;
                p += 2 * (dx - dy);
            }}
}
}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, NULL);
    int x1 = 500, y1 = 280, x2 = 320, y2 = 300;
    drawLineBresenham(x1, y1, x2, y2);
    getch();
    closegraph();
    return 0;}
```

OUTPUT:



```
PS C:\Users\user\Desktop\Programs\c programming> g++ graphic.c
PS C:\Users\user\Desktop\Programs\c programming> ./a
PS C:\Users\user\Desktop\Programs\c programming> g++ graphic.c
PS C:\Users\user\Desktop\Programs\c programming> ./a
PS C:\Users\user\Desktop\Programs\c programming> g++ graphic.c
PS C:\Users\user\Desktop\Programs\c programming> ./a
PS C:\Users\user\Desktop\Programs\c programming> ./a
```

DISCUSSION:

In this lab, we implemented Bresenham's Line Algorithm in C to draw a straight line efficiently between two endpoints on a graphics window. The algorithm calculates differences in x and y coordinates to determine increments for drawing in the correct direction. It initializes a decision parameter based on the absolute differences in coordinates and iterates through steps, updating pixel positions according to the parameter's evaluation. By avoiding floating-point arithmetic and rounding operations, Bresenham's algorithm ensures accurate and smooth line rendering, making it suitable for real-time graphics applications. This implementation showcases the integration of graphics functions in C, demonstrating the practical application of Bresenham's algorithm for line rasterization.

CONCLUSION:

The above experiment in C programming, we have understood the concept of Bresenham Line Algorithm (BLA) and how it work.