|  |  |
| --- | --- |
| **Name** | RAHUL RAMMILAN YADAV |
| **Roll No:** |  |
| **Class/Sem:**  **Experiment No.:**  **Title:**  **Date of**  **Performance:**  **Date of Submission:** | DSE/III  1  Digital Differential Analyzer Line Drawing Algorithm |
|  |
| **Marks:** |  |
| **Sign of Faculty:** |  |

**Experiment No. 1**

**Aim** : Write a program to implement Digital Differential Analyzer Line Drawing Algorithm in C.

**Objective:** To implement DDA line drawing algorithm for drawing a line segment between two pointsA (x1, y1) and B (x2, y2)

**Theory :** DDA stands for Digital Differential Analyzer. It is an incrementalmethod of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps.

**DDA Working Mechanism**

Case 1: Slope < 1 (m<1)(theta<45)x coordinates increase fast

dx is set to unit interval dx=1; dy is computed; m = dy/dx therefore; dy = m Calculation for next pixel for line processed from left to right

xk+1 = x k + dx = xk + 1 yk+1 = yk + dy = yk + m

Calculation for next pixel for line processed from right to left xk+1 = x k + dx = xk - 1

yk+1 = yk + dy = yk - m

Case 2: Slope > 1 (m>1) theta>45 y coordinates increase fast

dy is set to unit interval dy=1; dx is computed; m = dy/dx therefore; dx = 1/m Calculation for next pixel for line processed from left to right

xk+1 = x k + dx = xk + 1/m yk+1 = yk + dy = yk + 1

Calculation for next pixel for line processed from right to left xk+1 = x k + dx = xk - 1/m

yk+1 = yk + dy = yk - 1

Case 3: Slope =1 (m=1) theta=45

dx and dy is set to unit interval dx=1 and dy=1

Calculation for next pixel for line processed from left to right xk+1 = x k + dx = xk + 1

yk+1 = yk + dy = yk + 1

Calculation for next pixel for line processed from right to left xk+1 = x k + dx = xk - 1

yk+1 = yk + dy = yk - 1

**Algorithm**

**DDA Algorithm:**

**Step1:** StartAlgorithm

**Step2:** Declare x1,y1,x2,y2,dx,dy,x,y as integer variables. **Step3:** Enter value of x1,y1,x2,y2.

**Step4:** Calculate dx = x2-x1 **Step5:** Calculate dy = y2-y1 **Step6:** IfABS (dx) >ABS (dy)

Then step = abs (dx) Else

**Step7:** xinc=dx/step yinc=dy/step assign x = x1 assign y = y1

**Step8:** Set pixel (x, y) **Step9:** x = x + xinc

y = y + yinc

Set pixels (Round (x), Round (y)) **Step10:** Repeat step 9 until x = x2 **Step11:** EndAlgorithm

Code :

#include <stdio.h>

#include <graphics.h>

void drawLineDDA(int x1, int y1, int x2, int 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;

    float y = y1;

    putpixel(x, y, WHITE);

    for (int i = 1; i <= steps; i++) {

        x += xIncrement;

        y += yIncrement;

        putpixel(round(x), round(y), WHITE);

    }

}

int main() {

    int gd = DETECT, gm;

    initgraph(&gd, &gm, "");

    int x1, y1, x2, y2;

    printf("Enter the coordinates of the first point (x1 y1): ");

    scanf("%d %d", &x1, &y1);

    printf("Enter the coordinates of the second point (x2 y2): ");

    scanf("%d %d", &x2, &y2);

    drawLineDDA(x1, y1, x2, y2);

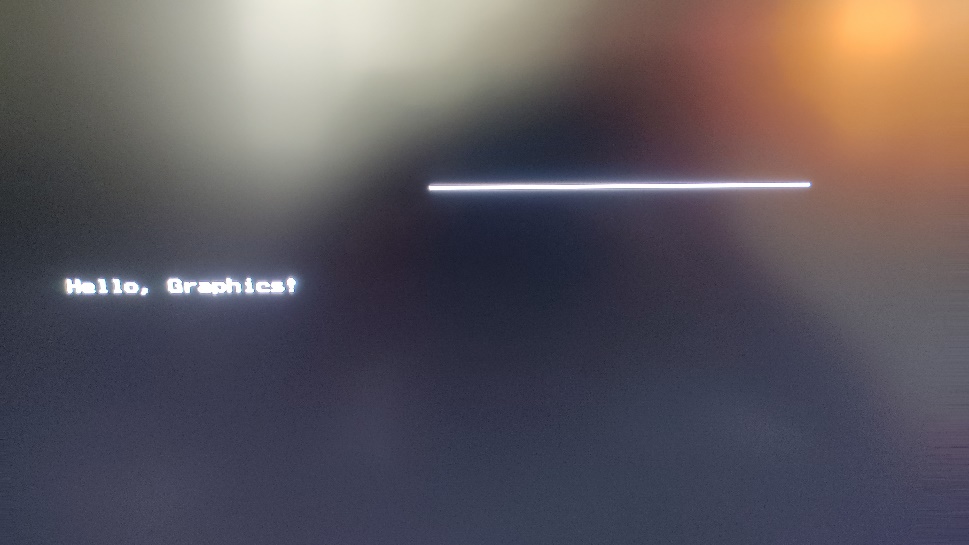
    delay(5000);

    closegraph();

    return 0;

}

**Output :**

****

**Conclusion :**

**The Digital Differential Analyzer (DDA) algorithm is a straightforward and efficient method for drawing lines on a computer screen. It is based on linear interpolation using floating-point arithmetic to determine the pixel positions along the line. The algorithm is easy to understand and implement, making it suitable for simple graphics applications.**

**However, the DDA algorithm has some limitations. It relies on floating-point arithmetic, which may not be ideal for systems with limited computational resources. Moreover, rounding errors during calculations can accumulate, leading to slight deviations from the ideal line. Additionally, the DDA algorithm is sensitive to the choice of the number of steps, and an inappropriate number of steps may result in a loss of precision or graphical artifacts.**