

Experiment No.: 1

Title: Implementation of DDA Line Drawing Algorithm Calculation in OpenGL C.

Objective: The objective of this experiment is to implement the DDA line drawing algorithm in OpenGL C to draw a line between two points. The goal is to understand the working of the DDA algorithm and its application in computer graphics for rendering lines.

Algorithm:

1. **Initialize:** Start with the initial point (x0, y0) and the end point (x1, y1).
2. **Calculate Differences:** Compute the differences $\Delta x = x1 - x0$ and $\Delta y = y1 - y0$.
3. **Determine Steps:** Calculate the number of steps required to generate the line.
 $Steps = \max(|\Delta x|, |\Delta y|)$.
4. **Calculate Increment:** Calculate the increment values for x and y for each step.
 - $x_increment = \Delta x / Steps$
 - $y_increment = \Delta y / Steps$
5. **Draw Line:** Starting from (x0, y0), add the increment values to the current coordinates and plot the points until the end point is reached.

Code in OpenGL C++ :

```
#include<windows.h>
#include<GL/glut.h>
#include<stdio.h>

float x1, x2, y1, y2;
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    float dy, dx, step, x, y, i, Xin, Yin;
    dx = x2 - x1;
    dy = y2 - y1;
    if (abs(dx) > abs(dy)) {
        step = abs(dx);
    } else
        step = abs(dy);
    Xin = dx / step;
    Yin = dy / step;
    x = x1;
    y = y1;

    glBegin(GL_POINTS);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2i(x, y);

    glEnd();
    for (i = 0; i < step; i++) {
```

```

        x += Xin;
        y += Yin;
        glBegin(GL_POINTS);
        glColor3f(0.0, 1.0, 0.0);
        glVertex2i(x, y);
    glEnd();
}
glFlush();
}
void myInit (void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 100, 0, 100);
}
int main(int argc, char ** argv) {

    printf("Value of x1 : ");
    scanf("%f", &x1);
    printf("Value of y1 : ");
    scanf("%f", &y1);
    printf("Value of x2 : ");
    scanf("%f", &x2);
    printf("Value of y2 : ");
    scanf("%f", &y2);


    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("DDA Algorithm Without M");

    myInit ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}

```

Input:

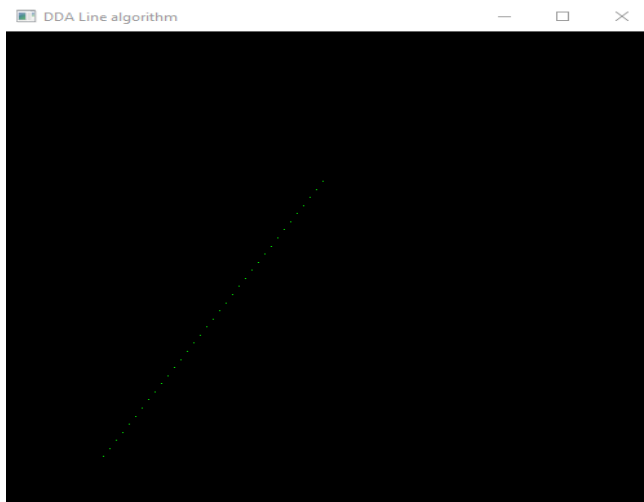


```

E:\4-2\Computer Graphics\Lab\DDA Line\bin\Debug\DDA Line.exe
Value of x1 : 15
Value of y1 : 10
Value of x2 : 50
Value of y2 : 70

```

Output:



Discussion:

During the implementation of the DDA line drawing algorithm using OpenGL in C, a few challenges were encountered:

Coordinate Mapping:

- **Problem:** The coordinate system used by OpenGL may not match the input coordinates, leading to unexpected line placements.
- **Solution:** The use of `gluOrtho2D` sets up a 2D orthographic viewing region, ensuring the input coordinates map correctly to the window dimensions. Adjust the range in `gluOrtho2D` if necessary.

Precision Handling:

- **Problem:** Floating-point arithmetic can introduce precision errors, causing the plotted points to be slightly off.
 - **Solution:** Using `round` function ensures that the coordinates are rounded to the nearest integer, which corresponds to pixel positions on the screen. However, ensure the math library is included by adding `#include <math.h>`.
- **Visualization:**
 - **Problem:** The display may not refresh correctly if the buffer is not cleared or if `glFlush` is not called.
 - **Solution:** Clearing the buffer using `glClear(GL_COLOR_BUFFER_BIT)` and flushing the commands using `glFlush()` ensures the display updates correctly.

Increment Calculation:

- **Problem:** Incorrect increment calculation can result in an incomplete or distorted line.
- **Solution:** Ensure that `steps`, `x_increment`, and `y_increment` are calculated correctly as

```
float steps = fabs(dx) > fabs(dy) ? fabs(dx) : fabs(dy);  
float x_next = dx / steps;  
float y_increment = dy / steps;
```

Experiment No.: 2

Title: Implementation of DDA (Digital Differential Analyzer) Line Drawing Algorithm with Slope Calculation in OpenGL C

Objective: The objective of this experiment is to implement the DDA line drawing algorithm in OpenGL C to draw a line between two points and calculate the slope (m) of the line. The goal is to understand the working of the DDA algorithm and its application in computer graphics for rendering lines.

Algorithm:

1. Initialize: Start with the initial point (x0, y0) and the end point (x1, y1).
2. Calculate Differences: Compute the differences $\Delta x = x1 - x0$ and $\Delta y = y1 - y0$.
3. Determine Steps: Calculate the number of steps required to generate the line. Steps = $\max(|\Delta x|, |\Delta y|)$.
4. Calculate Increment: Calculate the increment values for x and y for each step.

$$x_increment = \Delta x / \text{Steps}$$

$$y_increment = \Delta y / \text{Steps}$$

5. Draw Line: Starting from (x0, y0), add the increment values to the current coordinates and plot the points until the end point is reached.
6. Calculate Slope: Slope (m) = $\Delta y / \Delta x$.

Code in OpenGL C++ :

```
#include<stdio.h>
#include <GL/gl.h>
#include <GL/glut.h>
float x1,y1,x2,y2,m;
float dx =0,dy= 0;
void display(void)
{
/* clear all pixels */
glClear (GL_COLOR_BUFFER_BIT);
glColor3f (0.0, 1.0, 0.0);
glBegin(GL_POINTS);
if(m>0 && m<=1)
{
while(x1<=x2 && y1<=y2)
{
x1=x1+1;
y1=y1+m;
```

```

        glVertex3f(x1/100,y1/100,0.0);
        printf("%f %f",x1,y1);
    }
}
else if(m>1)
{
    while(x1<=x2 && y1<=y2)
    {
        x1=x1+(1/m);
        y1=y1+1;
        glVertex3f(x1/100,y1/100,0.0);
        printf("%f %f",x1,y1);
    }
}
else if(m>-1 && m<=0){
    while(x1>=x2 && y1>=y2)
    {
        x1=x1-1;
        y1=y1-m;
        glVertex3f(x1/100,y1/100,0.0);
        printf("%f %f",x1,y1);
    }
}
else if(m<-1)
{
    while(x1>=x2 && y1>=y2)
    {
        x1=x1-(1/m);
        y1=y1-1;
        glVertex3f(x1/100,y1/100,0.0);
        printf("%f %f",x1,y1);
    }
}
glEnd();
glFlush ();
}

void init (void) {
/* select clearing (background) color */
glClearColor (0.0, 0.0, 0.0, 0.0);
/* initialize viewing values */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

```

```

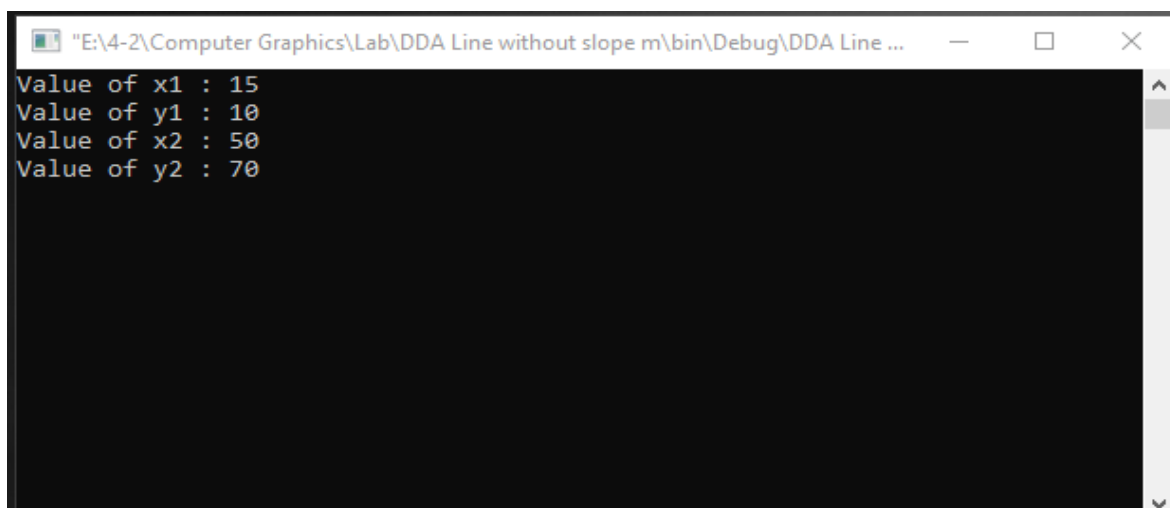
int main(int argc, char** argv){
    printf("Enter value of X1 :");
    scanf("%f",&x1);
    printf("Enter value of y1 :");
    scanf("%f",&y1);
    printf("Enter value of X2 :");
    scanf("%f",&x2);
    printf("Enter value of Y2 :");
    scanf("%f",&y2);
    dx=x2-x1;
    dy=y2-y1;
    m=dy/dx;

    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("DDA Line With Slope(m)");
    init();
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}

```

Input:



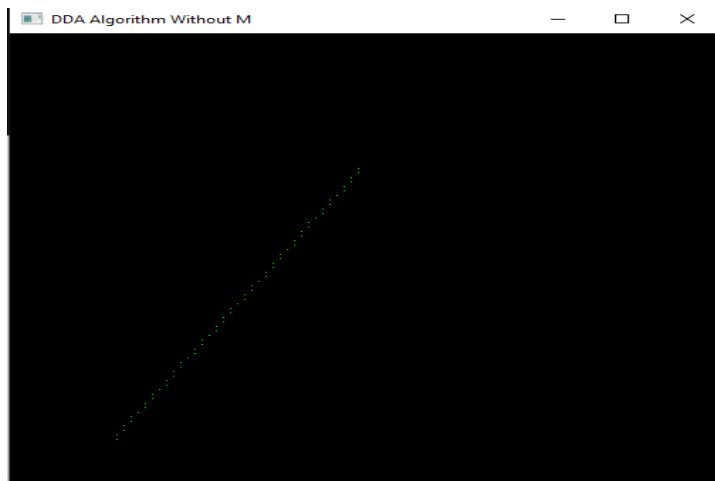
The screenshot shows a Windows command prompt window with the title bar: "E:\4-2\Computer Graphics\Lab\DDA Line without slope m\bin\Debug\DDA Line ...". The window contains the following text:

```

Value of x1 : 15
Value of y1 : 10
Value of x2 : 50
Value of y2 : 70

```

Output:



Discussion:

During the implementation of the DDA line drawing algorithm using OpenGL in C, a few challenges were encountered:

Incorrect Logic for Slope Handling:

- **Problem:** The logic for determining which case to use based on the slope mmm might not handle all possible line directions correctly. Specifically, it does not account for all negative slopes or cases where the line goes from higher to lower values of xxx or yyy.
- **Solution:** Simplify the logic by calculating steps based on the largest difference and then updating xxx and yyy incrementally.

Coordinate Transformation Issues:

- **Problem:** Using `glVertex3f(x1 / 100, y1 / 100, 0.0)` assumes the input coordinates are within a specific range and scales them. This can lead to inaccuracies or issues if coordinates are outside the expected range.
- **Solution:** Adjust the viewport and coordinate system setup to correctly map input coordinates to the screen without arbitrary scaling. Use `glVertex2i` for integer coordinates and proper viewport setup.

☐ Incorrect Loop Conditions:

- **Problem:** The conditions in the while loops (`while(x1 <= x2 && y1 <= y2)`) may not correctly handle lines that go from a higher to a lower coordinate.
- **Solution:** Ensure the loop conditions correctly account for the direction of the line.

☐ No Line for Vertical or Horizontal Slopes:

- **Problem:** The code does not handle cases where the line is perfectly vertical or horizontal, as these have undefined or zero slopes.
- **Solution:** Add special handling for these cases within the DDA logic.

Experiment No.: 3

Title: Implementation of Bresenham's Line Drawing Algorithm using OpenGL in C

Objective: The objective of this experiment is to implement Bresenham's line drawing algorithm to draw a line between two points using OpenGL in C. The goal is to understand the working of Bresenham's algorithm and its application in computer graphics for rendering lines efficiently.

Algorithm:

1. **Initialize:** Start with the initial point (x0, y0) and the end point (x1, y1).
2. **Calculate Differences:** Compute the differences $\Delta x = |x1 - x0|$ and $\Delta y = |y1 - y0|$.
3. **Determine Decision Parameters:**
 - o Determine the direction of the line (increment or decrement for x and y).
 - o Set the initial decision parameter ppp based on Δx and Δy .
4. **Plot Points:**
 - o Starting from (x0, y0), plot the points based on the decision parameter ppp.
 - o Update ppp and the current coordinates (x, y) at each step until the end point is reached.

Code :

```
#include <stdio.h>
#include <GL/gl.h>
#include <GL/glut.h>
float x1,y1,x2,y2,m,i,j,p;
int dx=0,dy=0;
void display(void)
{
    /* clear all pixels */
    glClear (GL_COLOR_BUFFER_BIT);

    glEnd();

    glColor3f (0.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    p=(2*dy)-dx;
    for(i=x1,j=y1;i<=x2,j<=y2; ){
        if(p>=0){
            i=i+1;
            j=j+1;
            if((i>x2)||(j>y2)){
                break;
            }
            printf("%0.2f %0.2f\n",i,j);
            glVertex3f ((i/100), (j/100), 0.0);
            p=p+(2*dy)-(2*dx);
        }
        else if(p<0){
            i=i+1;
```



```

        if((i>x2)||(j>y2)){
            break;
        }
        printf("%0.2f %0.2f\n",i,j);
        glVertex3f ((i/100), (j/100), 0.0);
        p=p+(2*dy);
    }
}
glEnd();
glFlush ();
}
void init (void)
{
    /* select clearing (background) color */
    glClearColor (0.0, 0.0, 0.0, 0.0);
    /* initialize viewing values */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
int main(int argc, char** argv)
{
    printf("Value of x1 : ");
    scanf("%f", &x1);
    printf("Value of y1 : ");
    scanf("%f", &y1);
    printf("Value of x2 : ");
    scanf("%f", &x2);
    printf("Value of y2 : ");
    scanf("%f", &y2);
    dx=x2-x1;
    dy=y2-y1;
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("Bresenham's Line Drawing");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();

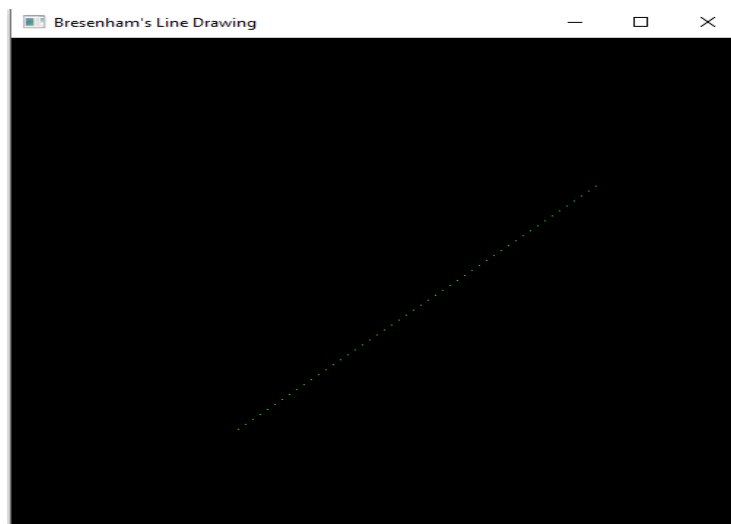
    return 0;
}

```

Input

```
"E:\4-2\Computer Graphics\Lab\Bresenham's Line\bin\Debug\Bresenham's Line.exe"
Value of x1 : 30
Value of y1 : 20
Value of x2 : 80
Value of y2 : 90
31.00 21.00
32.00 22.00
33.00 23.00
34.00 24.00
35.00 25.00
36.00 26.00
37.00 27.00
38.00 28.00
39.00 29.00
40.00 30.00
41.00 31.00
42.00 32.00
43.00 33.00
44.00 34.00
```

Output:



Discussion:

Incorrect Loop Conditions:

- **Problem:** The loop condition for `(i = x1, j = y1; i <= x2, j <= y2;)` uses a comma, which causes both conditions to be evaluated together. It might terminate early or run infinitely.
- **Solution:** Use two separate conditions to ensure the loop correctly updates `i` and `j` independently.

```
for (i = x1, j = y1; i <= x2 && j <= y2;) { }
```

Missing Handling for Different Slopes:

- **Problem:** The code only handles lines with positive slopes. Lines with negative slopes or horizontal/vertical lines are not handled correctly.

- **Solution:** Extend the logic to handle all cases, including lines with negative slopes, and separate the handling for the main octants of the Cartesian plane.

Coordinate Transformation Issues:

- **Problem:** The transformation `glVertex3f((i / 100), (j / 100), 0.0)` scales the coordinates arbitrarily.
- **Solution:** Use proper scaling and setup of the coordinate system using `'glOrtho'`