Assignment Number	: 2
Laboratory	: Computer Graphics
Batch	: E11
Roll Number	: 23310
Name	: Shashank Vijay Kapadnis

Title:

Understand and Implement DDA and Bresenham's Line Drawing Algorithms using OpenGL.

Problem Statement:

Implement DDA and Bresenham line drawing algorithm to draw: i) Simple Line ii) Dotted Line iii) Dashed Line iv) Solid line; using mouse interface Divide the screen in four quadrants with centre as (0, 0). The line should work for all the slopes positive as well as negative.

Theory:

A DDA (Digital Differential Analyzer) Line Drawing Algorithm

- In computer graphics, a digital differential analyzer (DDA) is hardware or software used for interpolation of variable over an interval between start and end point. DDAs are used for rasterization of lines, triangles and polygons. They can be extended to nonlinear functions, such as perspective correct texture mapping, quadratic curves, and traversing voxels.
- A linear DDA starts by calculating the smaller of dy or dx for a unit increment of the other. A line is then sampled at unit intervals in one coordinate and corresponding integer values nearest the line path are determined for the other coordinate.
- Considering a line with positive slope, if the slope is less than or equal to 1, we sample at unit x intervals (dx=1) and compute

successive y values as subscript k takes integer values starting from 0, for the 1st point and increases by 1 until endpoint is reached. y value is rounded off to nearest integer to correspond to a screen pixel.

- For lines with slope greater than 1, we reverse the role of x and y i.e., we sample at dy=1 and calculate consecutive x values as similar calculations are carried out to determine pixel positions along a line with negative slope. Thus, if the absolute value of the slope is less than 1, we set dx=1 if i.e., the starting extreme point is at the left.

Algorithm:

Given-

- Starting coordinates = (X_0, Y_0)
- Ending coordinates = (X_n, Y_n)

The points generation using DDA Algorithm involves the following steps-

Step-01:

Calculate ΔX, ΔY and M from the given input.

These parameters are calculated as-

- $\Delta X = X_n - X_0$
- $\Delta Y = Y_n - Y_0$
- $M = \Delta Y / \Delta X$

Step-02:

Find the number of steps or points in between the starting and ending coordinates.

if (absolute (ΔX) > absolute (ΔY))

Steps = absolute (ΔX);

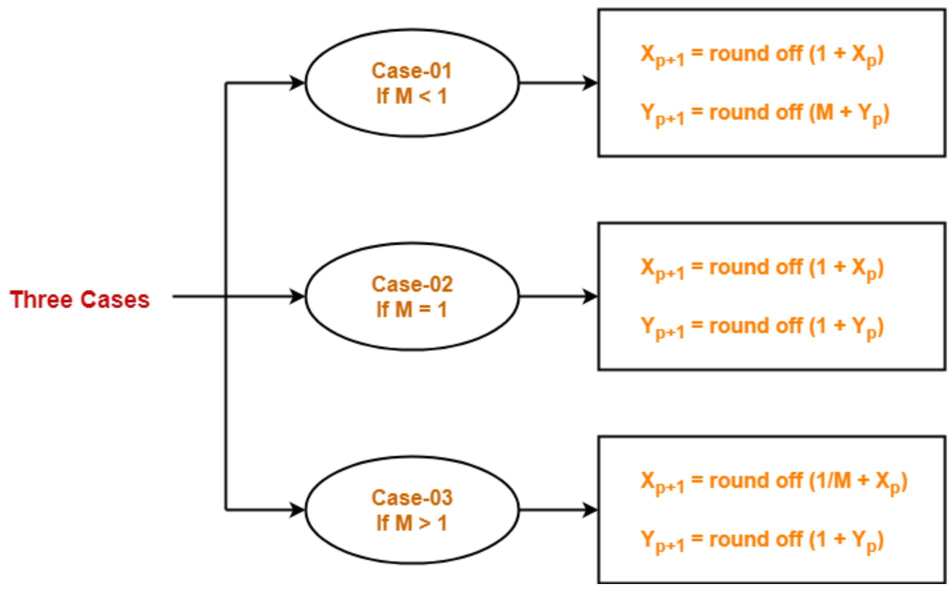
else

Steps = absolute (ΔY);

Step-03:

Suppose the current point is (X_p, Y_p) and the next point is (X_{p+1}, Y_{p+1}).

Find the next point by following the below three cases-


Three Cases
Case-01
If M < 1
X_{p+1} = round off (1 + X_p)
Y_{p+1} = round off (M + Y_p)
Case-02
If M = 1
X_{p+1} = round off (1 + X_p)
Y_{p+1} = round off (1 + Y_p)
Case-03
If M > 1
X_{p+1} = round off (1/M + X_p)
Y_{p+1} = round off (1 + Y_p)

Step-04:

Keep repeating Step-03 until the end point is reached or the number of generated new points (including the starting and ending points) equals to the steps count.

B Bresenham’s Line Drawing Algorithm

- The Bresenham’s Line drawing algorithm is a scan line algorithm used to determine the intermediate points which determine the line segment between a given initial point and a given final point.
- This algorithm is faster than the DDA algorithm and more efficient as it only involves integer addition, subtraction, multiplication and division.
- These integer calculations have a higher calculation speed than the floating-point calculations and hence, the line is plotted at a higher speed.
- A decision parameter is used to determine the position of the next pixel.
- 2 points are considered while plotting the next ($i+1^{th}$) point. The actual point value is determined by the slope point form equation which can be determined by the 2 given points. The distance value affects the decision parameter and hence the actual distance is not calculated.
- The distance between the actual point and the 2 assumed points is compared. The point which is closer to the actual point is considered for plotting.
- This process is repeated till the final given point is reached.
- The algorithm was initially determined for a line of slope less than 1, which can be generalized by altering a few parameters.

Algorithm:

```
1 START
2 Get Initial Co-ordinates (xi, yi) and final coordinates (xf, yf).
3 Initialize dx and dy to abs (xi, xf) and abs (yi, yf) respectively where abs
  () represents absolute difference between passed values.
4 Initialize x_change and y_change values on the basis of following
  conditions.
  a If xi > xf then x_change = -1 else x_change = 1
  b If yi > yf then y_change = -1 else y_change = 1
5 If dx is 0, plot vertical line and exit.
6 If dy is 0, plot horizontal line and exit.
7 Initialize x = xi and y = yi.
8 Initialize decision parameter P.
9 If dx > dy:
    a Set P = 2*dy – dx
    b Initialize loop variable i to 0.
    c If P > 0, y = y + y_change, P = P + 2*(dy - dx).
    d Else, P = P + 2*dy
    e x = x + x_change
```

```
        f   Plot vertex (x, y).
        g   i = i+1
        h   If i < dx, Go To step (c).
    10 Else:
        a   Set P = 2*dx – dy
        b   Initialize loop variable i to 0.
        c   If P > 0, x = x + x_change, P = P + 2*(dx - dy)
        d   Else, P = P + 2*dy
        e   y = y + y_change
        f   Plot vertex (x, y)/
        g   i = i + 1.
        h   If i < dy, Go to Step (c).
    11 STOP
```

Program:

<u>DDA</u>;-

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

#include<stdio.h>

#include<stdlib.h>


float X1,X2,Y1,Y2,xmax,ymax;

int choice;


void init()

{

   glClearColor(1.0,0.0,0.0,0.0);

   glColor3f(1.0,1.0,1.0);

   gluOrtho2D(0,1024,0,768);

}

void SimpleLine(int x1,int y1,int x2,int y2){

   glBegin(GL_LINES);

   glVertex2f(x1, y1);
```

```
    glVertex2f(x2, y2);
    glEnd();
    glFlush();
}
void primitives(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    xmax=glutGet(GLUT_WINDOW_WIDTH);
    ymax=glutGet(GLUT_WINDOW_HEIGHT);
    SimpleLine(X1+512,Y1+384,X2+512,Y2+384);
    glFlush();
}
int main(int argc,char **argv)
{
    printf("Enter the value of x1 : ");
    scanf("%f",&X1);
    printf("Enter the value of y1 : ");
    scanf("%f",&Y1);
    printf("Enter the value of x2 : ");
    scanf("%f",&X2);
    printf("Enter the value of y2 : ");
    scanf("%f",&Y2);
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE);
    glutInitWindowPosition(0,0);
    glutInitWindowSize(1024,768);
    glutCreateWindow("DDA Line Pattern");
```

```
   init();
   glutDisplayFunc(primitives);
   glutMainLoop();
}
```

<u>Bresenham</u>:-

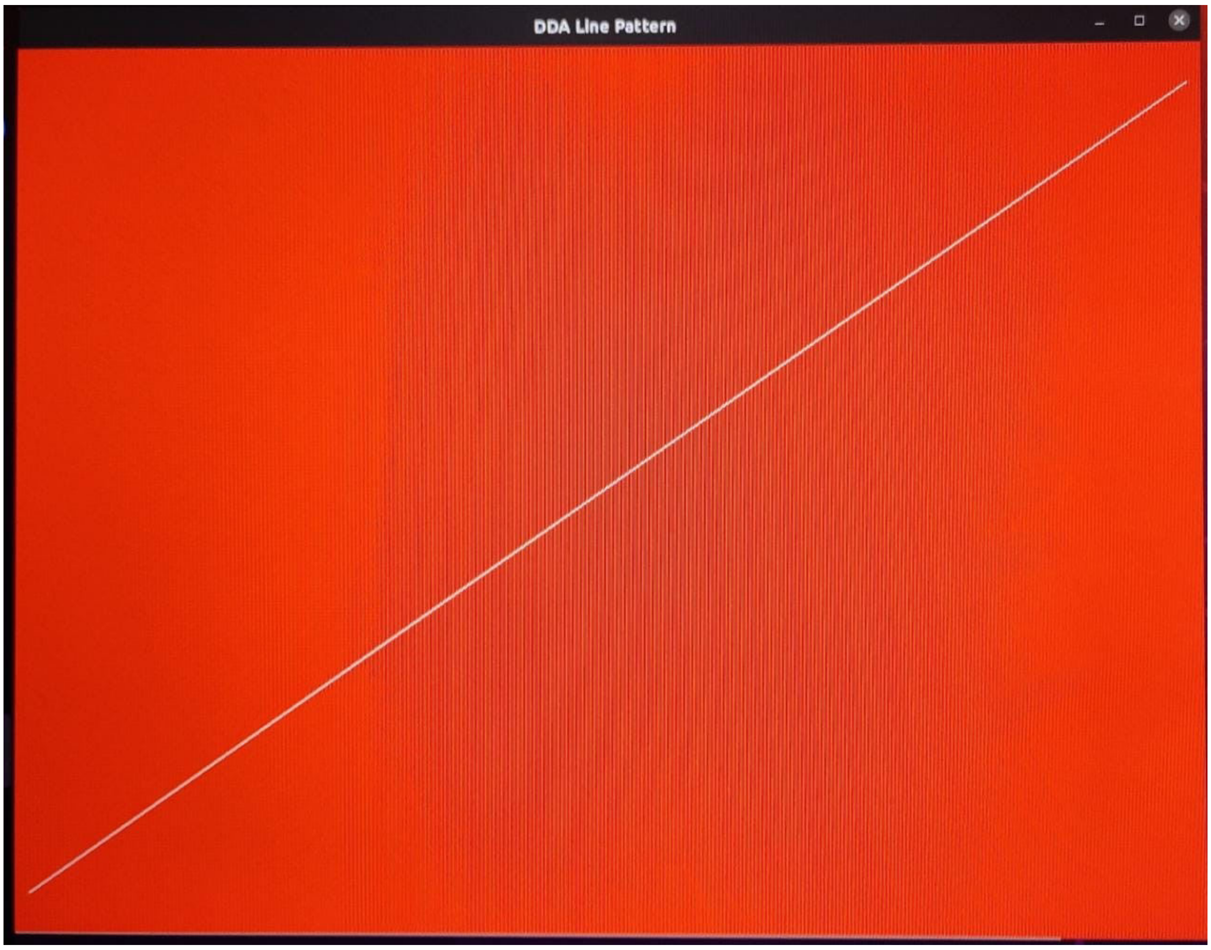
```
#include <GL/freeglut.h>
#include <GL/gl.h>
#include<math.h>
#include<cstdio>

float X1,X2,Y1,Y2,xmax,ymax;
int choice;

void init()
{
   glClearColor(1.0,0.0,0,0.0);
   glColor3f(1.0,1.0,1.0);
   gluOrtho2D(0,1024,0,768);
}
void SimpleLine(int x1,int y1,int x2,int y2){
   glBegin(GL_LINES);
   glVertex2f(x1, y1);
   glVertex2f(x2, y2);
   glEnd();
   glFlush();
}
void renderFunction()
```

```
{

   glClear(GL_COLOR_BUFFER_BIT);
   SimpleLine(X1+512,Y1+384,X2+512,Y2+384);
   glFlush();
}


int main(int argc, char** argv)
{
   printf("Enter the value of x1 : ");
   scanf("%f",&X1);
   printf("Enter the value of y1 : ");
   scanf("%f",&Y1);
   printf("Enter the value of x2 : ");
   scanf("%f",&X2);
   printf("Enter the value of y2 : ");
   scanf("%f",&Y2);
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition(0,0);
   glutInitWindowSize(1024,768);
   glutCreateWindow("Bresenham Line pattern");
   init();
   glutDisplayFunc(renderFunction);
   glutMainLoop();
   return 0;
```

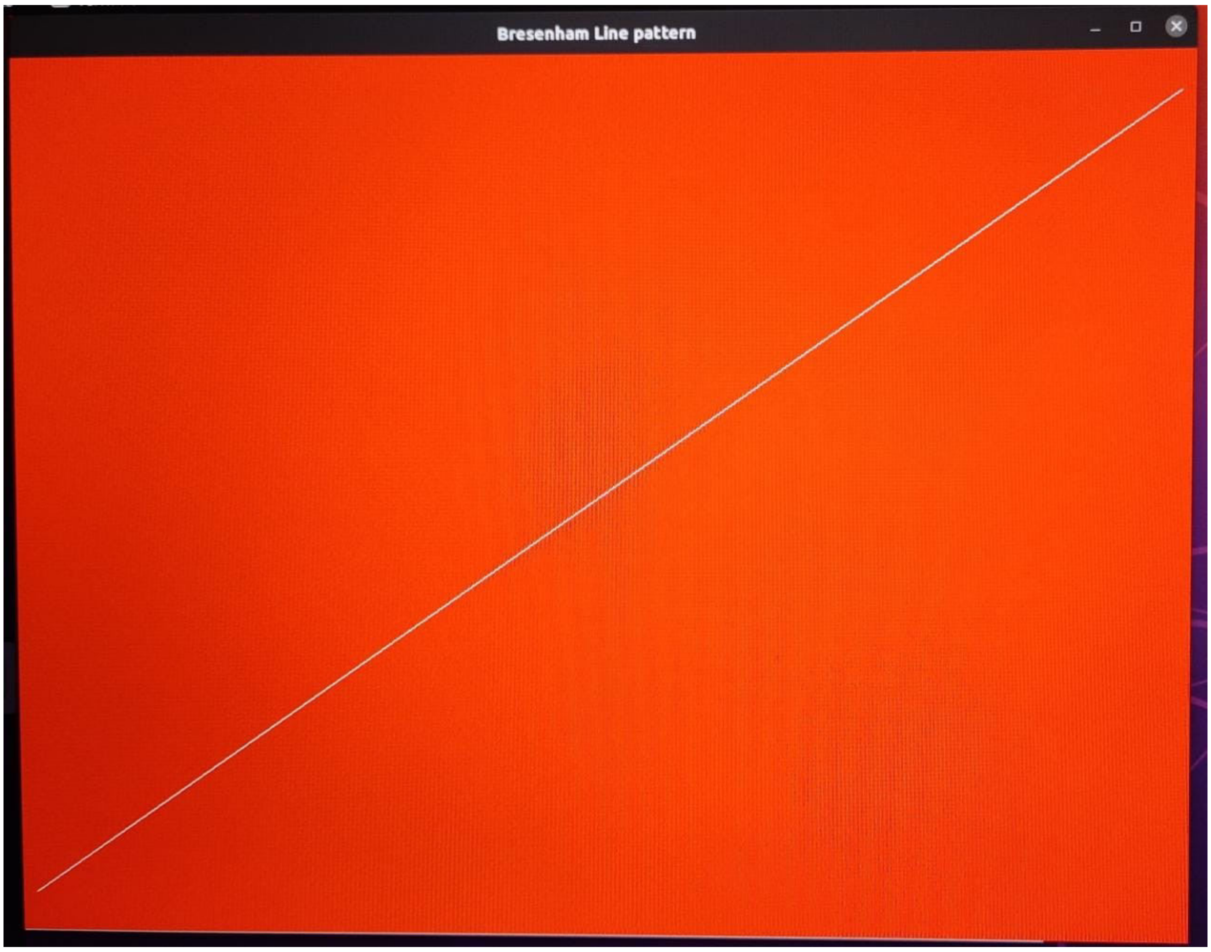
}

Screenshots:

DDA:-



<u>Bresenham</u>:-



Conclusion:

1. DDA and Bresenham's Algorithm for line drawing were studied and analyzed mathematically.
2. Above algorithms were implemented using OpenGL library and mouse interface for trapping co-ordinates and assigning menu was used.