**Shudhanshu Shrotriya**

**500086221**

**Batch-4**

**Experiment – 4 & 5**

**Computer Graphics Lab**

**Implementation of DDA and Bresenham’s Algorithm**

**DDA Algorithm:**

DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of lines. In this method, the calculation is performed at each step but by using the results of previous steps.

**Steps to use DDA Algorithm:**

**Step1:** Start Algorithm

**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:** If ABS (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:** End Algorithm

**CODE:**

#include <iostream>

#include <math.h>

#include <GL/glut.h>

using namespace std;

double X1, Y1, X2, Y2;

float round\_value(float v)

{

return floor(v + 0.5);

}

void LineDDA(void)

{

double dx = (X2 - X1);

double dy = (Y2 - Y1);

double steps;

float xInc, yInc, x = X1, y = Y1;

/\* Find out whether to increment x or y \*/

steps = (abs(dx) > abs(dy)) ? (abs(dx)) : (abs(dy));

xInc = dx / (float)steps;

yInc = dy / (float)steps;

/\* Clears buffers to preset values \*/

glClear(GL\_COLOR\_BUFFER\_BIT);

/\* Plot the points \*/

glBegin(GL\_POINTS);

/\* Plot the first point \*/

glVertex2d(x, y);

int k;

/\* For every step, find an intermediate vertex \*/

for (k = 0; k < steps; k++)

{

x += xInc;

y += yInc;

/\* printf("%0.6lf %0.6lf\n",floor(x), floor(y)); \*/

glVertex2d(round\_value(x), round\_value(y));

}

glEnd();

glFlush();

}

void Init()

{

/\* Set clear color to white \*/

glClearColor(1.0, 1.0, 1.0, 0);

/\* Set fill color to black \*/

glColor3f(0.0, 0.0, 0.0);

/\* glViewport(0 , 0 , 640 , 480); \*/

/\* glMatrixMode(GL\_PROJECTION); \*/

/\* glLoadIdentity(); \*/

gluOrtho2D(0, 640, 0, 480);

}

void main(int argc, char\*\* argv)

{

cout << "Enter two end points of the line to be drawn:\n";

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout << "\nEnter Point1( X1 , Y1):\n";

cin >> X1 >> Y1;

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout << "\nEnter Point1(X2, Y2) :\n";

cin >> X2 >> Y2;

/\* Initialise GLUT library \*/

glutInit(&argc, argv);

/\* Set the initial display mode \*/

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

/\* Set the initial window position and size \*/

glutInitWindowPosition(0, 0);

glutInitWindowSize(640, 480);

/\* Create the window with title "DDA\_Line" \*/

glutCreateWindow("DDA\_Line");

/\* Initialize drawing colors \*/

Init();

/\* Call the displaying function \*/

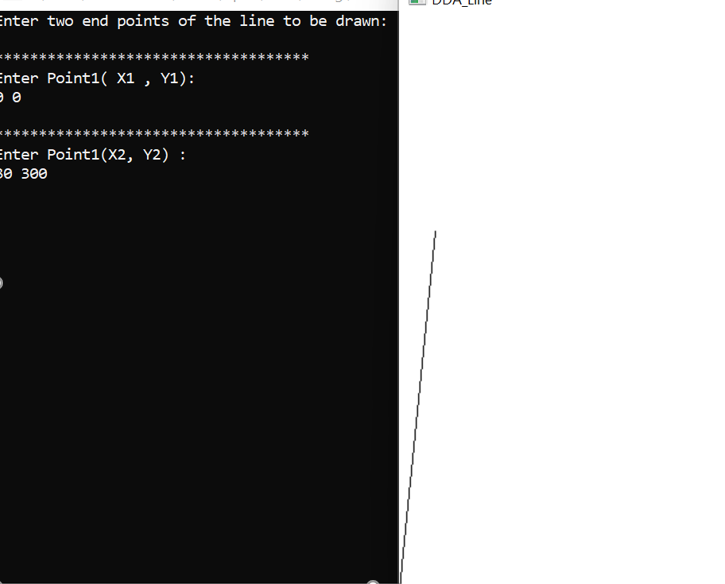
glutDisplayFunc(LineDDA);

/\* Keep displaying untill the program is closed \*/

glutMainLoop();

}

**OUTPUT:**



**Bresenham’s Algorithm**

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.

In this method, next pixel selected is that one who has the least distance from true line.

## Bresenham's Line Algorithm:

**Step1:** Start Algorithm

**Step2:** Declare variable x1,x2,y1,y2,d,i1,i2,dx,dy

**Step3:** Enter value of x1,y1,x2,y2  
                Where x1,y1are coordinates of starting point  
                And x2,y2 are coordinates of Ending point

**Step4:** Calculate dx = x2-x1  
                Calculate dy = y2-y1  
                Calculate i1=2\*dy  
                Calculate i2=2\*(dy-dx)   
                Calculate d=i1-dx

**Step5:** Consider (x, y) as starting point and xendas maximum possible value of x.  
                If dx < 0  
                        Then x = x2  
                        y = y2  
                          xend=x1  
                If dx > 0  
                    Then x = x1  
                y = y1  
                        xend=x2

**Step6:** Generate point at (x,y)coordinates.

**Step7:** Check if whole line is generated.  
                If x > = xend  
                Stop.

**Step8:** Calculate co-ordinates of the next pixel   
                If d < 0  
                    Then d = d + i1  
                If d ≥ 0  
          Then d = d + i2  
                Increment y = y + 1

**Step9:** Increment x = x + 1

**Step10:** Draw a point of latest (x, y) coordinates

**Step11:** Go to step 7

**Step12:** End of Algorithm

**Code:**

#include <GL/glut.h>

#include <stdio.h>

#include <iostream>

using namespace std;

int x1, y3, x2, y2;

void myInit()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.0, 0.0, 0.0, 1.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 500, 0, 500);

}

void draw\_pixel(int x, int y)

{

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

void draw\_line(int x1, int x2, int y3, int y2)

{

int dx, dy, i, e;

int incx, incy, inc1, inc2;

int x, y;

dx = x2 - x1;

dy = y2 - y3;

if (dx < 0) dx = -dx;

if (dy < 0) dy = -dy;

incx = 1;

if (x2 < x1) incx = -1;

incy = 1;

if (y2 < y3) incy = -1;

x = x1; y = y3;

if (dx > dy)

{

draw\_pixel(x, y);

e = 2 \* dy - dx;

inc1 = 2 \* (dy - dx);

inc2 = 2 \* dy;

for (i = 0; i < dx; i++)

{

if (e >= 0)

{

y += incy;

e += inc1;

}

else

e += inc2;

x += incx;

draw\_pixel(x, y);

}

}

else

{

draw\_pixel(x, y);

e = 2 \* dx - dy;

inc1 = 2 \* (dx - dy);

inc2 = 2 \* dx;

for (i = 0; i < dy; i++)

{

if (e >= 0)

{

x += incx;

e += inc1;

}

else

e += inc2;

y += incy;

draw\_pixel(x, y);

}

}

}

void myDisplay()

{

draw\_line(x1, x2, y3, y2);

glFlush();

}

int main(int argc, char\*\* argv)

{

cout <<"Enter (x1, y3, x2, y2)\n";

cin >> x1 >> y3 >> x2 >> y2;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Bresenham's Line Drawing");

myInit();

glutDisplayFunc(myDisplay);

glutMainLoop();

return 0;

}

**Output:**

