Experiment 2

Name: Yash Jarang

Class: D6AD Roll no: 47.

Course Outcome: LO1.

DOP: 7/9/2021

DOS: 24/9/2021

Teacher's Sign:

Student's Sign:

①

Aim : Implementation of DDA line drawing algorithm.

~~T~~Theory : Digital Differential Analyzer (DDA):

The DDA starts by calculating the smaller of dy or dx for a unit increment of the other. A line is then sampled at unit increment intervals in one coordinate and corresponding integer values nearest the line path are determined for the other co-ordinate.

Consider a line with positive slope, If the slope is less than or equal to 1, we sample at unit x intervals ($dx=1$) and compare successive y values as : $y_{k+1} = y_k + m$.

Subscript k takes integer values starting from 0, for the 1st point and increases by 1 until end point is reached. If value is rounded off ~~by~~ to nearest integer to correspond to a screen pixel.

For lines with slope greater than 1, we reverse the whole of x and y i.e we sample at $dy=1$ and calculate consecutive x values at $x_{k+1} = x_k + \frac{1}{m}$.

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.

(2)

Example: If a line is drawn from (6,9) to (11,12). Rasterize it.

—

(6,9) to (11,12) $x_1 = 6$ & $x_2 = 11$.
$y_1 = 9$ & $y_2 = 12$

$dx = x_2 - x_1 = 5$
$dy = y_2 - y_1 = 3$.

$\frac{dy}{dx} = \frac{3}{5} = m$. $\therefore$ Steps $= 5$.

$x_{inc} = dx/\text{steps} = 5/5 = 1$.
$y_{inc} = dy/\text{steps} = 3/5 = 3/5$.
F = float.

i	x_{new}	y_{new}	Plot (F(x_{new}), F(y_{new}))		
0	6	9	" (F(6+0.5), F(9+0.5))	6	9
1	7	9.6	(F(7+0.5), F(9.6+0.5))	7	10
2	8	10.2	(F(8+0.5), F(10.2+0.5))	8	10
3	9	10.8	(F(9+0.5), F(10.8+0.5))	9	11
4	10	11.4	(F(10+0.5), F(11.4+0.5))	10	11
5	11	12	(F(11+0.5), F(12+0.5))	11	12

(3)

Digital Differential Analyzer Algorithm:

Step 1: Start.

Step 2: Declare $x_1, y_1, x_2, y_2, dx, dy, x, y$ as integers.

Step 3: Enter the values of $x_1, y_1; x_2, y_2$.

Step 4: Calculate $dx = x_2 - x_1$

Calculate $dy = y_2 - y_1$

Step 5: If abs (dx) > abs (dy)

then steps = abs (dx).

else

steps = abs(dy).

Step 6: $x_{inc} = dx / step$

$y_{inc} = dy / step$.

assign $x = x_1$, $y = y_1$.

Step 7: Set pixel (x, y).

Step 8: $x = x + x_{inc}$.

$y = y + y_{inc}$.

Step 9: Set pixels (round(x), round(y))

Step 10: Repeat Step 8, 9 until $x = x_2$.

Step 11: End.

Rasterize the line AB using DDA where A(0,0) and B (8,4).

→ $x_1 = 0$, $x_2 = 8$, $y_1 = 0$, $y_2 = 4$.

$d_x = x_2 - x_1 = 8$. $\frac{dy}{dx} = \frac{4}{8} = \frac{1}{2}$.

$d_y = y_2 - y_1 = 4$.

④

i	x_{new}	y_{new}	plot $(F(x_{new}, y_{new}))$		
0	0	0	plot (f(0+0.5), f(0+0.5))	0	0
1	1	0.5	plot (f(1+0.5), f(0.5+0.5))	1	1
2	2	1	plot (f(2+0.5), f(1+0.5))	2	1
3	3	1.5	plot (f(3+0.5), f(1.5+0.5))	3	2
4	4	2	plot (f(4+0.5), f(2+0.5))	4	2
5	5	2.5	plot (f(5+0.5), f(2.5+0.5))	5	3
6	6	3	plot (f(6+0.5), f(3+0.5))	6	3
7	7	3.5	plot (f(7+0.5), f(3.5+0.5))	7	4
8	8	4	plot (f(8+0.5), f(4+0.5))	8	4

Conclusion: Advantages of DDA

i) DDA is the simplest algorithm and it does not require special skills for implementation.

ii) It is a faster method for calculating pixel positions.

iii) Eliminates ~~not~~ multiplication by using screen characteristics.

Disadvantages of DDA.

i) Floating point arithmetic in DDA alogrithm is still time consuming

ii) The algorithm is orientation dependant. Hence end point accuracy is poor.

iii) Rounding down takes time.

Followed by the program code:

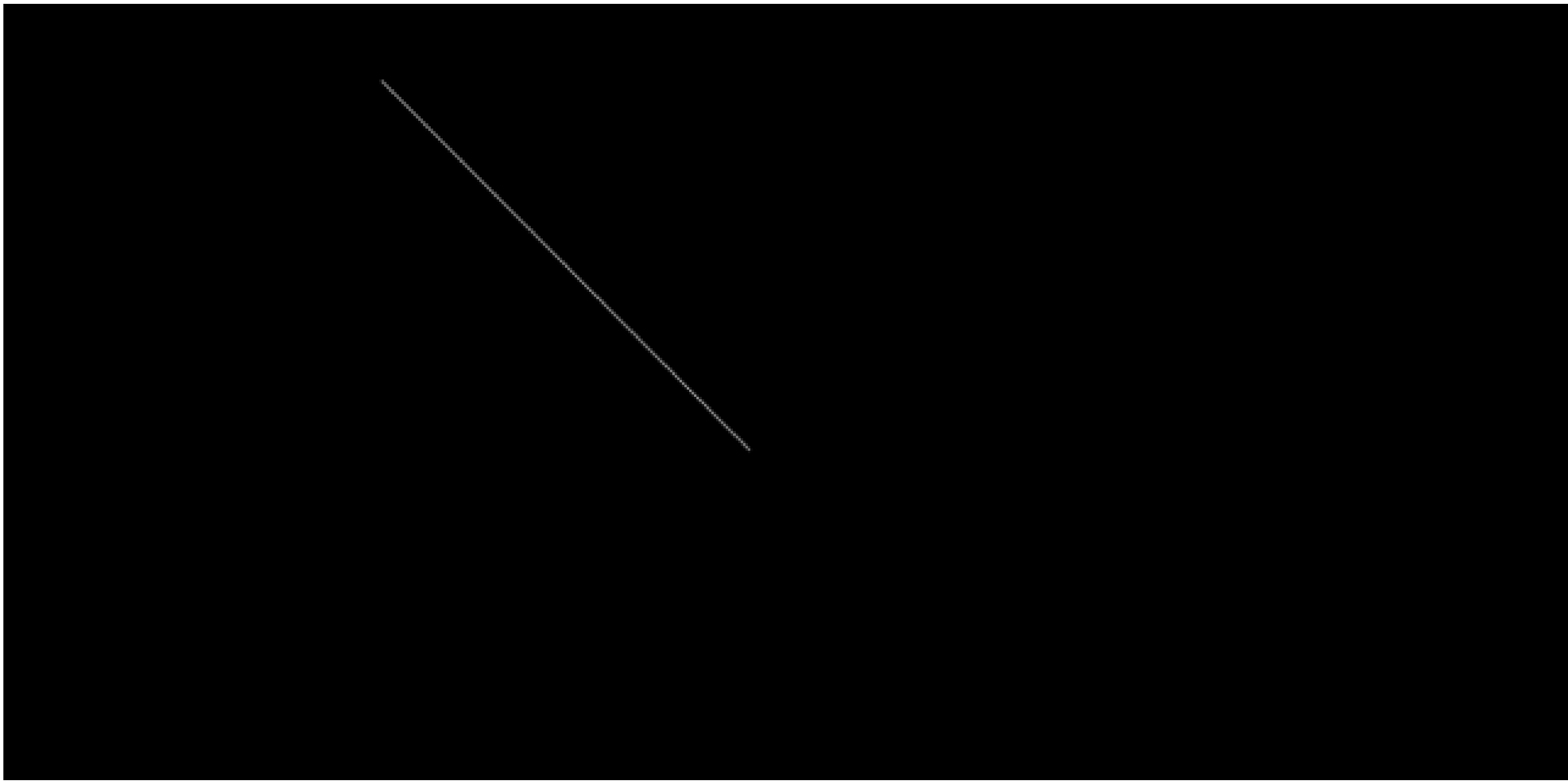
CODE:

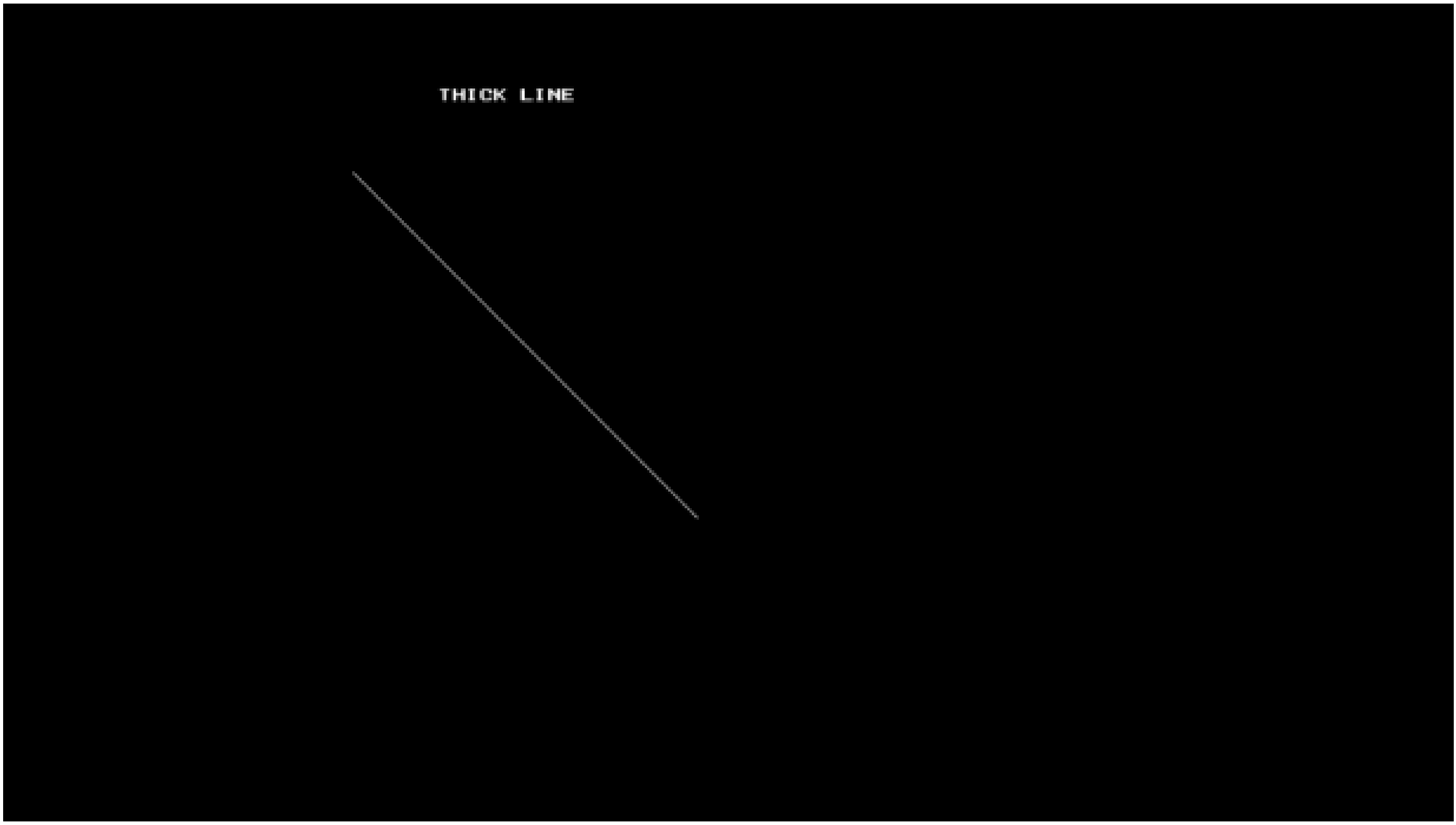
```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void main()
{
	int gd=DETECT,gm,i,errorcode;
	float x,y,dx,dy;
	int steps,r;
	int x0,x1,y0,y1;
	int color_val;
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
	errorcode=graphresult();
	if(errorcode!=0)
	{
		printf("Graphics error:%s\n",grapherrormsg(errorcode));
		printf("press any key to halt:");
		getch();
		exit(1);
	}
	setbkcolor(BLACK);
	x0=0,y0=0,x1=8,y1=4;
	dx=(float)(x1-x0);
	dy=(float)(y1-y0);
	steps=0;

	if(dx>=dy)
	{
		steps=dx;
	}

	else
	{
		steps=dy;
	}

	dx=dx/steps;
	dy=dy/steps;
	x=x0;
	y=y0;
	i=1;

	while(i<=steps)
	{
		putpixel(x,y,RED);
		x+=dx;
		y+=dy;
```

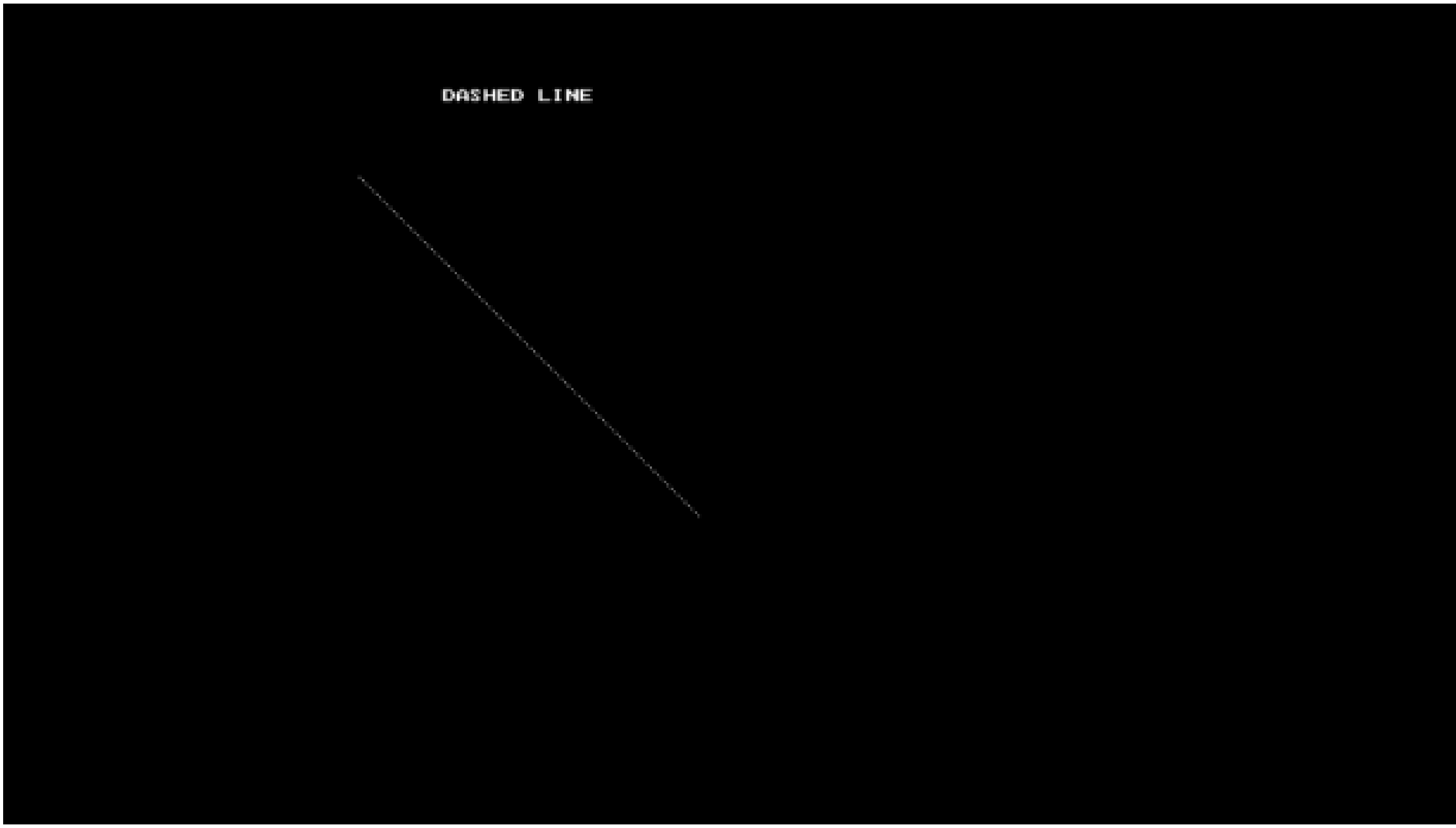
```
        i+=1;
}

//displaying thick lines
x=x0;
y=y0;
getch();
cleardevice();
outtextxy(150,50,"THICK LINE");

for(steps;steps>0;steps--)
{
        x=x+dx;
        y=y+dy;
        delay(20);
        putpixel(floor(x+0.5),floor(y+0.5),WHITE);
        putpixel(floor(x+1.5),floor(y+1.5),WHITE);
        putpixel(floor(x-1.5),floor(y-1.5),WHITE);
}

//displaying dashed lines
x=x0;
y=y0;
getch();
cleardevice();
outtextxy(150,50,"DASHED LINE");

for(steps;steps>0;steps--)
{
        x=x+dx;
        y=y+dy;
        delay(20);

        if(steps%2==0)
        {
                putpixel(floor(x+0.5),floor(y+0.5),WHITE);
        }
}
//displaying colored lines
x=x0;
y=y0;
color_val=0;
getch();
cleardevice();
outtextxy(150,50,"COLOR LINE");

for(steps;steps>0;steps--)
{
        x=x+dx;
        y=y+dy;
        delay(20);
        putpixel(floor(x+0.5),floor(y+0.5),color_val);
        color_val++;
```

```
            if(color_val==15)
                    color_val=0;
    }
    getch();
    closegraph();
}
```

OUTPUT:






DASHED LINE


COLOR LINE