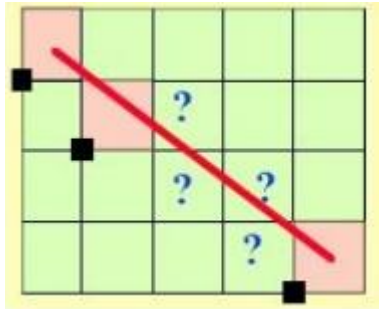


Experiment No 2

Title 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 center as (0, 0). The line should work for all the slopes positive as well as negative.

Problem Statement: Write a program to implement following algorithms : DDA Line Drawing Algorithm Bresenham Line Drawing Algorithm



Objective:

To understand the different algorithms for line & circle generation and their implementation.

Problem definition:

Rasterization : Process of determining which pixels provide the best approximation to a desired line on the screen.

Scan Conversion: Combination of rasterization and generating the picture in scan line order.

General requirements

Straight lines must appear as straight lines.

They must start and end accurately.

Lines should have constant brightness along their length

Lines should drawn rapidly.

Line drawing algorithm

Programmer specifies (x,y) values of end pixels

Need algorithm to figure out which intermediate pixels are on line path

Pixel (x,y) values constrained to integer values

Actual computed intermediate line values may be floats

Rounding may be required. E.g. computed point (10.48, 20.51) rounded to (10, 21)

Rounded pixel value is off actual line path.

Sloped lines end up having jaggies

Vertical, horizontal lines, no jaggies

```

#include<iostream>
#include<GL/glut.h>
using namespace std;
int Algo,type;
void Init()
{
    glClearColor(0,0,0,0);
    glColor3f(0,1,0);
    gluOrtho2D(0,640,0,480);
    glClear(GL_COLOR_BUFFER_BIT);
}
int sign(float a){
    if(a==0){
        return 0;
    }
    if(a>0){
        return 1;
    }
    return -1;
}
void B_Line(int x_1,int y_1,int x_2,int y_2,int t){
    float dy, dx, m , P;
    dy = y_2 - y_1;
    dx = x_2 - x_1;
    m = dy/dx;
    P = 2*dy - dx;
    int x = x_1, y = y_1;
    cout<<"\n x1 = "<<x<<" y1 = "<<y;
    if(m<1){
        int cnt=1;
        for(int i=0; i<=dx;i++){

```

```
if(t == 1){  
    glBegin(GL_POINTS);  
    glVertex2i(x,y);  
    glEnd();
```

```
}
```

```
if(t == 2){  
    if(i%2==0){  
        glBegin(GL_POINTS);  
        glVertex2i(x,y);  
        glEnd();
```

```
}
```

```
}
```

```
if(t == 3){  
    if(cnt <= 10){  
        glBegin(GL_POINTS);  
        glVertex2i(x,y);  
        glEnd();
```

```
}
```

```
cnt++;  
if(cnt == 15){  
    cnt =1;
```

```
}
```

```
}
```

```
if(P<0){  
    x = x +1;  
    y =y;
```

```

P = P + 2*dy;

}

else{
x= x+1;
y = y+1;
P = P + 2*dy
- 2*dx;

}

}

}

else{
int cnt = 1;
for(int i=0;i<=dy;i++){
if(t == 1){
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();

}

if(t == 2){
if(i%2==0){
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();

}

}

}

```

```

if(t == 3){
if(cnt <= 10){
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
cnt++;
if(cnt == 15){
cnt =1;
}
}
if(P<0){
x = x;
y =y+1;
P = P + 2*dx;
}
else{
x= x+1;
y = y+1;
P = P + 2*dx - 2*dy;
}
}
}
cout<<"\n xlast = "<<x<<" ylast = "<<y;
glFlush();
}

void DDA_LINE(int x_1,int y_1,int x_2,int y_2, int t){
float dx,dy,length;
dx = x_2-x_1;
dy = y_2-y_1;
if(abs(dx) >= abs(dy)){
length = abs(dx);

```

```

}
else{
length = abs(dy);
}
float xin, yin;
xin = dx/length;
yin = dy/length;
float x,y;
x = x_1 + 0.5 * sign(xin);
y = y_1 + 0.5 * sign(yin);
int i=0;
int cnt =1;
while(i<=length){
if(t == 1){
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
if(t == 2){
if(i%2==0){
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
}
if(t == 3){
if(cnt <= 10){
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
}
cnt++;

```

```

if(cnt == 15){
    cnt =1;
} }
x = x + xin;
y = y + yin;
i++ ;
}
glFlush();
}
void display()
{
    DDA_LINE(0,240,640,240,1);
    B_Line(320,0,320,640,1);
    glFlush();
}
void mymouse(int b,int s, int x, int y)
{
    static int x_s,y_s,x_e,y_e,pt=0;
    if(b==GLUT_LEFT_BUTTON && s==GLUT_DOWN)
    {
        if(pt==0)
        {
            x_s =x;
            y_s =480 - y;
            pt++;
            glBegin(GL_POINTS);
            glVertex2i(x_s,y_s);
            glEnd();
        }
        else
        {
            x_e=x;

```

```

y_e=480-y;
cout<<"\n x_1_click "<<x_s<<" y_1_click "<<y_s;
cout<<"\n x_2_click "<<x_e<<" y_2_click "<<y_e<<"\n";

glBegin(GL_POINTS);
glVertex2i(x_e,y_e);
glEnd();

if(Algo == 1){
DDA_LINE(x_s,y_s,x_e,y_e,type);
}

if(Algo == 2){
B_Line(x_s,y_s,x_e,y_e,type);
}

}

else if(b==GLUT_RIGHT_BUTTON && s==GLUT_DOWN)
{
pt=0;
}

glFlush();
}

int main(int argc ,char **argv)
{
cout<<"\n Select the Algorithm \n 1. DDA \n 2. Bresenham's \n";
cin>>Algo;

cout<<"Select the Line Type \n 1. Simple Line \n 2. Dotted Line\n 3. Dashed Line \n";
cin>>type;

if((Algo == 1 || Algo == 2)&&(type==1 || type==2 || type==3)){
}

else{
cout<<"\n Option enter are wrong \n";
return 0;
}
}

```



```

glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowPosition(100,100);
glutInitWindowSize(640,480);
glutCreateWindow("DDA-Line");
Init();
glutDisplayFunc(display);
glutMouseFunc(mymouse);
glutMainLoop();
return 0;
}

```

OUTPUT

```

g++ filename.cpp -lGL -lGLU -lglut
./a.out

```

