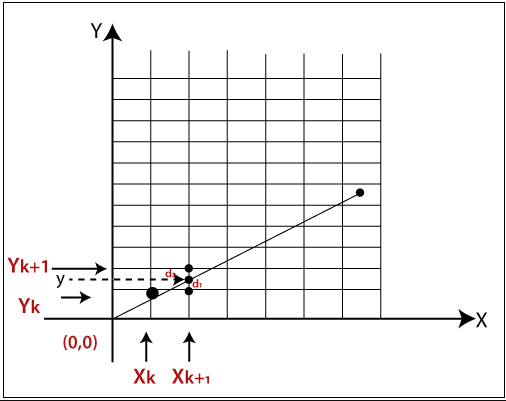
**TITLE: - IMPLEMENTATION OF BRESENHAM LINE DRAWING ALGORITHM**

**OBJECTIVE: -**

* To implement Bresenham Line Drawing Algorithm

**THEORY: -**

****Bresenham Line drawing Algorithm is more accurate and efficient line drawing algorithm developed by Bresenham. This algorithm can be further adopted to draw circle and other curves. In this method, next pixel is selected as the one which is at least distance from true line.

**CASE I: For slope m <1**

For this case, pixel positions along a line path are determined by sampling at unit x intervals starting from left end point (x0, y­0) of given line and then we step to each successive column x position and plot pixels whose y value is closer to line path.

Assuming, we have determined pixels at (xk, yk) to be displayed. Next, we need to decide which pixel to plot in column xk+1. There y value may be yk or yk+1. As in the figure, we label vertical pixels separations d1 and d2. The actual coordinate of y value at xk+1 is:

…………………………… (i)

Then, the distance d1 is:

And, the distance d2 is:

Thus, the distance between points d1 and d2= d2-d1

When (d1-d2) <0; d1 is closest pixel else d2 is the closest pixel.

Substituting m by ;

* **Where**
* Now,
* Then,
* Now, if is either zero (when or 1 when () depending upon the values of.

Again for the first parameter, we start from pixel position ( with.

* At, ;
* But,
* At
* **, which is the initial decision parameter.**
* Now, **if** ; We choose i.e.

So,

**Thus,**

* Else, if ; We choose i.e.

So,

**Thus,**

**CASE II: For Slope, m >1**

For this case, assuming the pixels is already plotted assuming that the sampling direction is along y axis.

i.e.

Similarly as above, the resulting decision parameters and results will be reversed on the basis of the axes.

**Advantages of Line Drawing Algorithm:**

It is more efficient and accurate as compared to the DDA algorithm. It has simplicity in implementation and calculations as it does not requires any rounding off up the floating coordinates. It gives optimized straight lines.

**Disadvantages of Bresenham Algorithm:**

It can draws the other curves but it requires further adaptations to its algorithm and it does not gives efficient support for drawing complex curves. It also leads to some aliasing effects. Thus, high slopes, jagged edges, stair-stepping etc. results during the implementation.

**Summary of Bresenham Algorithm (m<1):**

Step 1: Input two end points (x1,y1) and (x2,y2); and store left end point in (x0,y0).

Step 2: Plot the first point (x0,y0).

Step 3: Calculate constants dx and dy and find starting value for the decision parameter as .

Step 4: At each xk, along the line starting from k=0; perform the following tests:

* IF (**) the next point to plot is**  and
* Else If (; **The next point to plot is** and

Step 5: Repeat step y times.

**# Program to illustrate Bresenham Line Drawing Algorithm.**

#include<iostream.h>

#include<graphics.h>

#include<conio.h>

#include<string.h>

#include<math.h>

#include<stdlib.h>

#include<stdio.h>

int xc, yc;

float step;

int tablex, tabley, row\_height=20, column\_width=70;

void setcursor(int x, int y){

cout<<"\033["<<y<<";"<<x<<"H";

}

void drawtable(int row,int column){

int p\_row=tablex, p\_col=tabley;

for(int i=0; i<=row; i++){

line(p\_row,p\_col,p\_row+column\_width\*column,p\_col);

p\_col=p\_col+row\_height;

}

p\_row=tablex, p\_col=tabley;

for(int j=0; j<=column; j++){

line(p\_row,p\_col,p\_row,p\_col+row\_height\*row);

p\_row=p\_row+column\_width;

}

outtextxy(tablex+column\_width/2,tabley+row\_height/2,"k");

outtextxy(tablex+column\_width+column\_width/2,tabley+row\_height/2,"pk");

outtextxy(tablex+2\*column\_width+column\_width/2-10,tabley+row\_height/2,"xk+1");

outtextxy(tablex+3\*column\_width+5,tabley+row\_height/2," yk+1");

outtextxy(tablex+4\*column\_width+5,tabley+row\_height/2," pk+1");

}

void main(){

int gd=DETECT, gm=0;

initgraph(&gd, &gm, "..\\bgi");

xc=getmaxx()/2;

yc=getmaxy()/2;

tablex=xc-318;

tabley=yc-120;

float x1,y1,x2,y2; //Line end point coordinates

int k=0;

rectangle(xc+40,yc-110,getmaxx(),yc-43);

rectangle(xc+40,yc-45,getmaxx(),yc+100);

outtextxy(xc+150,yc-107,"INPUT");

line(xc+40,yc-97,getmaxx(),yc-97);

outtextxy(xc+150,yc-40,"OUTPUT");

line(xc+40,yc-30,getmaxx(),yc-30);

setcursor(47,10); cout<<"Enter Point to start: ";

setcursor(68,10); cin>>x1;

setcursor(78,10); cin>>y1;

setcursor(47,11); cout<<"Enter the end point: ";

setcursor(68,11); cin>>x2;

setcursor(78,11); cin>>y2;

int dx=(x2-x1), dy=(y2-y1);

int steps=abs(dx);

int p0=2\*dy-dx, pk=p0, pk1;

char x\_value[10], y\_value[10],p\_k[10], p\_k1[10],SN[10];

int text\_x=tablex+column\_width/2-10;

int text\_y=tabley+row\_height+row\_height/2;

while(k<=dx){

if(pk<=0)

pk1=pk+2\*dy;

else

pk1=pk+(2\*dy)-(2\*dx);

itoa(x1,x\_value,10);

itoa(y1,y\_value,10);

itoa(pk,p\_k,10);

itoa(pk1,p\_k1,10);

itoa(k,SN,10);

outtextxy(text\_x,text\_y,SN);

outtextxy(text\_x+column\_width,text\_y,p\_k);

outtextxy(text\_x+column\_width\*2,text\_y,x\_value);

outtextxy(text\_x+column\_width\*3,text\_y,y\_value);

outtextxy(text\_x+column\_width\*4,text\_y,p\_k1);

putpixel(xc+120+x1,yc+y1,WHITE);

text\_y=text\_y+row\_height; //Displays the output pixels

if(pk<=0){

x1=x1+1;

y1=y1;

pk=pk+2\*dy;

}

else{

x1=x1+1;

y1=y1+1;

pk=pk+(2\*dy)-(2\*dx);

pk1=pk;}

k++;

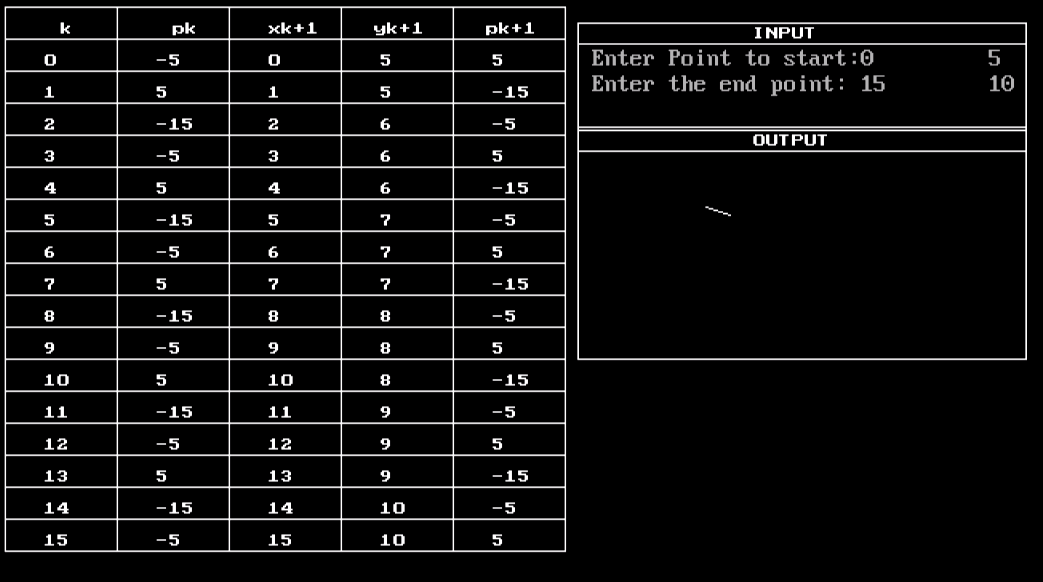
}

drawtable(steps+2,5);

getch();

closegraph();

}

**OUTPUT**: -

**CONCLUSION**:

In this way, we implement Bresenham algorithm through writing code in C++ programming language and analysed its precision and way of calculation.