Computer Graphics

Lab 1

Line Drawing

Introduction

- A **line** connects two end-points.
- It is a basic element in computer-graphics.
- To draw a line, you need two points between which you can draw a line.
- There are three line drawing algorithms,
  - **DDA**
  - **Bresenham's**
  - **Mid-point**

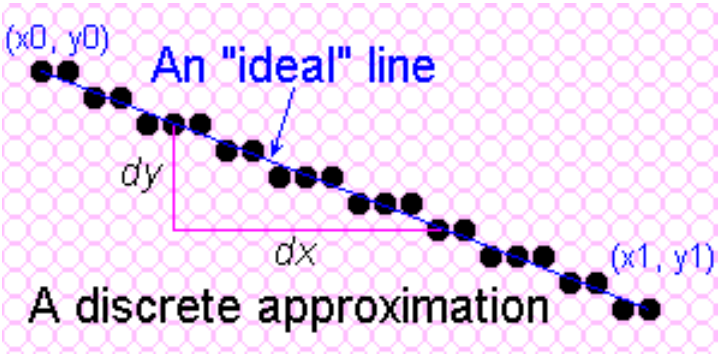
Introduction(Cont'd)

- Computer has to take care of two things while plotting any line on the computer screen:
  - **Pixels** to plot
  - **Computations** are required to calculate the pixel positions

- Line drawing algorithms
  - helps computer to find this things.

Towards the Ideal Line

- We can only do a discrete approximation
- Illuminate pixels as close to the true path as possible, consider bi-level display only
- Pixels are either lit or not lit



What is an *ideal* line

- Must appear straight and continuous
- Only possible axis-aligned and 45^{o} lines
- Must interpolate both defining end points
- Must be efficient, drawn quickly

Simple Line

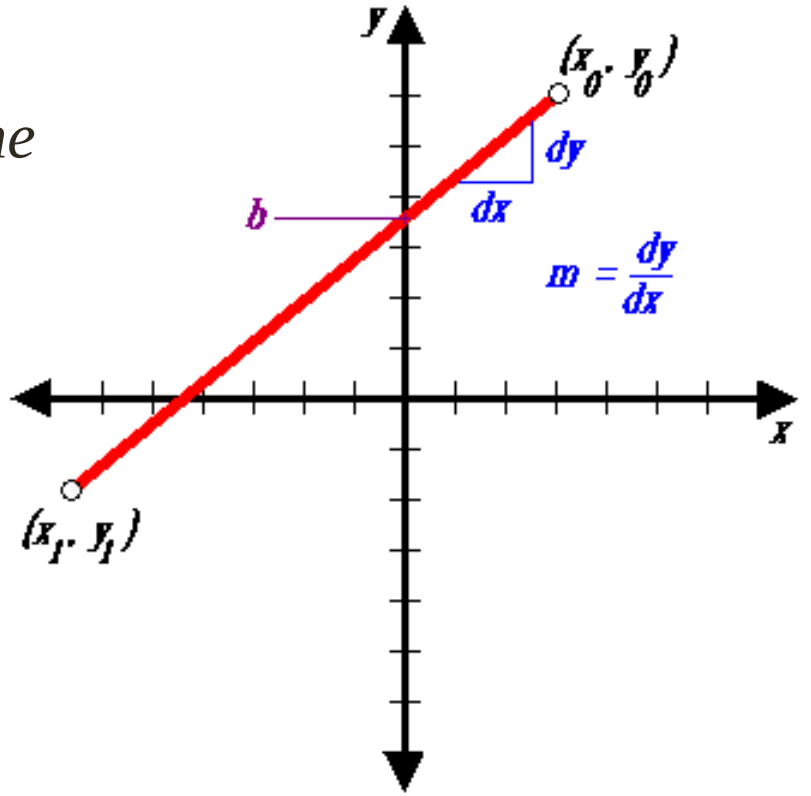
Based on *slope-intercept line equation* from algebra:

$$y = mx + b$$

Simple approach:

increment x, solve for y

Floating point arithmetic required

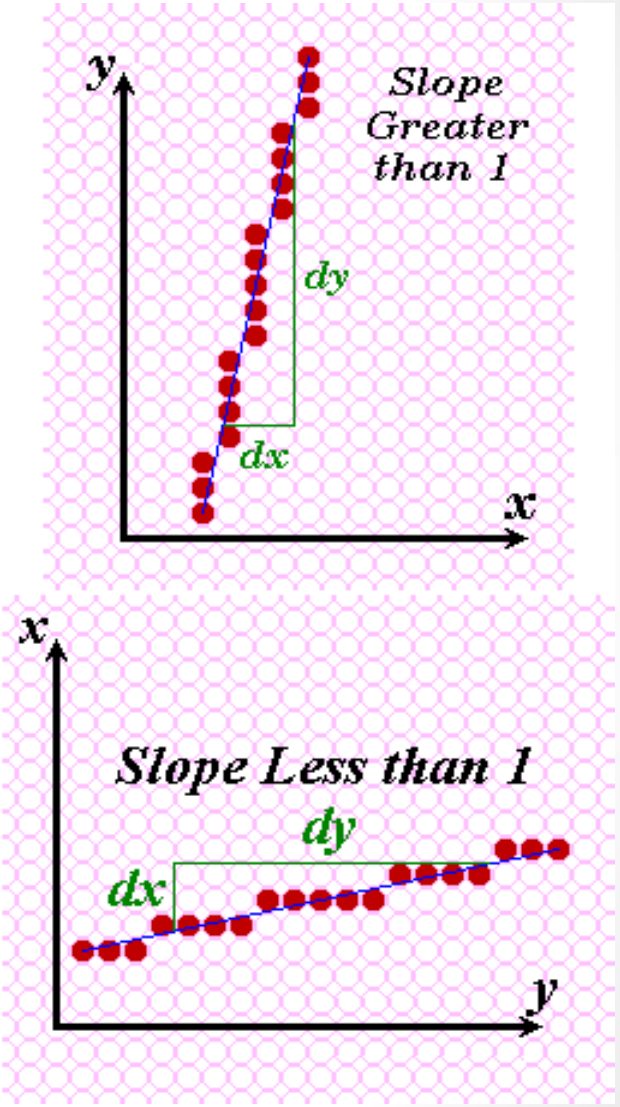


Does it Work?

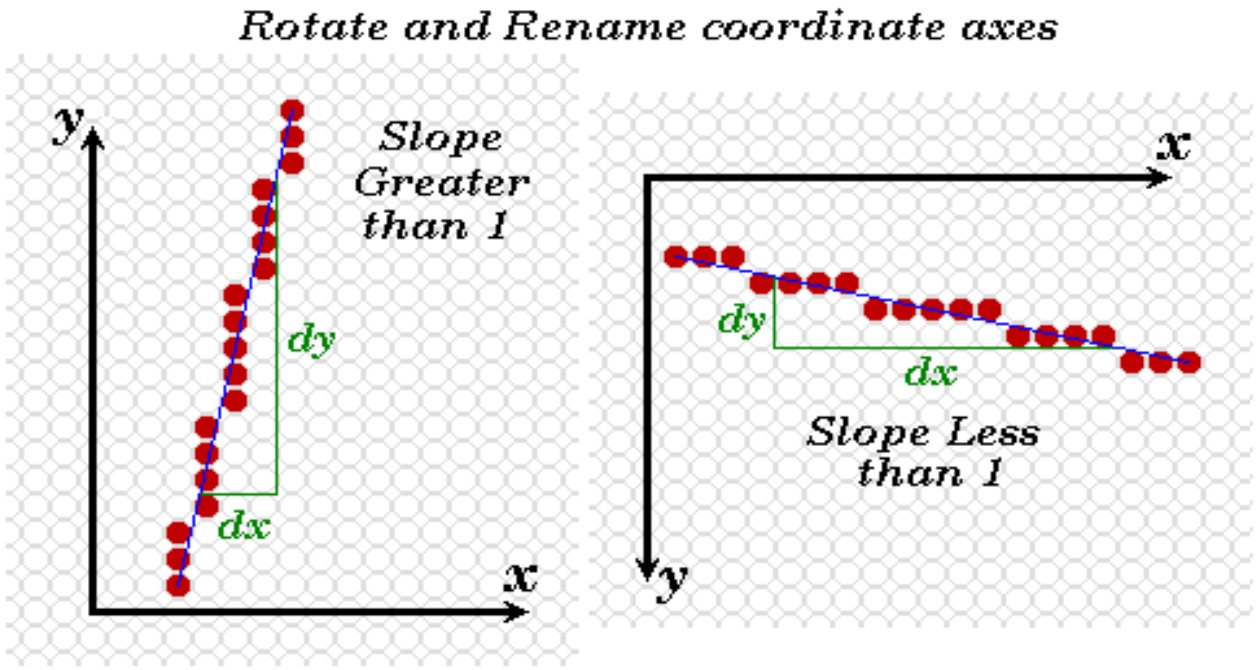
It seems to work okay for lines with a slope of 1 or less,

but doesn't work well for lines with slope greater than 1 – lines become more discontinuous in appearance and we must add more than 1 pixel per column to make it work.

Solution? - use *symmetry.*



Modify algorithm per octant



OR, increment along x-axis if dy<dx else increment along y-axis

The challenge is to find a way to calculate the next x,y position by previous one as quickly as possible.

DDA algorithm

- DDA = Digital Differential Analyser
- It is an incremental scan-conversion method.
- Such an approach is characterized by performing calculations at each step using results from the preceding step.

DDA algorithm

- **Step 1** – Get the input of two end points (X0,Y0) and (X1,Y1).
- **Step 2** – Calculate the difference between two end points.

  $dx = X_1 - X_0$ $\quad\quad$ $dy = Y_1 - Y_0$
- **Step 3** – Based on the calculated difference in step-2, you need to identify the number of steps to put pixel.

  If dx > dy, then you need more steps in x coordinate;

  otherwise in y coordinate.

  if (absolute(dx) > absolute(dy)) Steps = absolute(dx);

  else Steps = absolute(dy);

DDA algorithm

- **Step 4** – Calculate the increment in x coordinate and y coordinate.

  Xincrement = dx / (float) steps;

  Yincrement = dy / (float) steps;

- **Step 5** – Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
for(int v=0; v < Steps; v++)
{
        putpixel(Round(x), Round(y));
        x = x + Xincrement;
        y = y + Yincrement;
}
```

Example 1:

- **Scan convert a line having end points (3,2) & (4,7) using DDA.**

➢ **Solution:**

dx= x2 - x1 = 4-3 = 1

dy= y2 - y1 = 7-2 = 5

As, dx < dy then

steps = absolute(y2-y1)= 5

Xincrement = dx / (float) steps

= 1/5 =0.2

Yincrement = dy / (float) steps

= 5/5 =1

X	Y	X plotted	Y plotted
3	2	3	2
3.2	3	3	3
3.4	4	3	4
3.6	5	4	5
3.8	6	4	6
4	7	4	7

Example 2:

- **Scan convert a line having end points (10,3) & (6,8) using DDA.**

➢ **Solution:**

???

dx= 6-10 = -4, dy = 8-3 = 5

n.Steps = 5

Xinc = dx/n.steps = -4/5 = -0.8

Yinc = dy/n.steps = 5/5 = 1

DDA algorithm

```
line(int x1, int y1, int x2, int y2)

{
float x,y;
int dx = x2-x1, dy = y2-y1;
int n = abs(dx)> abs(dy)? abs(dx) : abs(dy);
float XInc = dx/n, YInc = dy/n;
       x = x1;
       y = y1;
       while( n-- ) {
              point(round(x),round(y));
              x += XInc;
              y += YInc;
       }
}
```