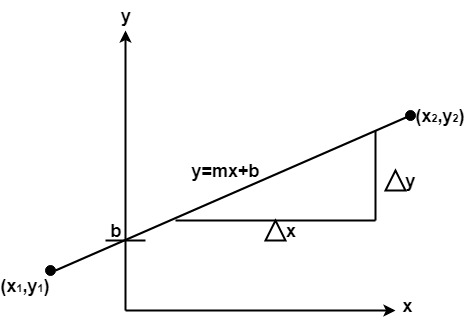
**Scan Converting a Straight Line**

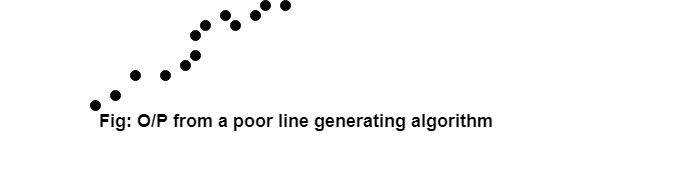
A straight line may be defined by two endpoints & an equation. In fig the two endpoints are described by (x1,y1) and (x2,y2). The equation of the line is used to determine the x, y coordinates of all the points that lie between these two endpoints.



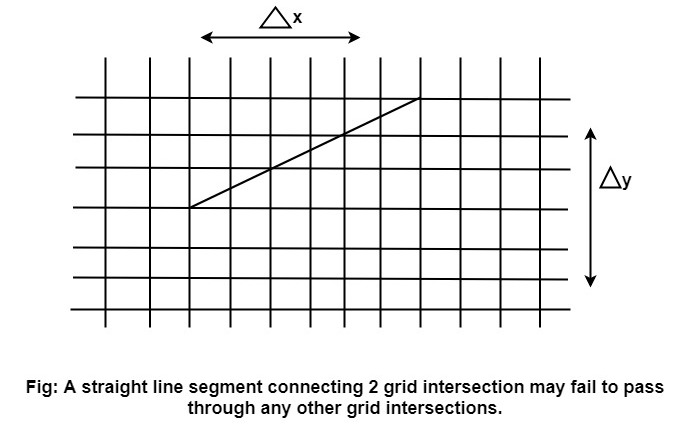
Using the equation of a straight line, y = mx + b where m = Scan Converting a Straight Line& b = the y interrupt, we can find values of y by incrementing x from x =x1, to x = x2. By scan-converting these calculated x, y values, we represent the line as a sequence of pixels.

**Properties of Good Line Drawing Algorithm:**

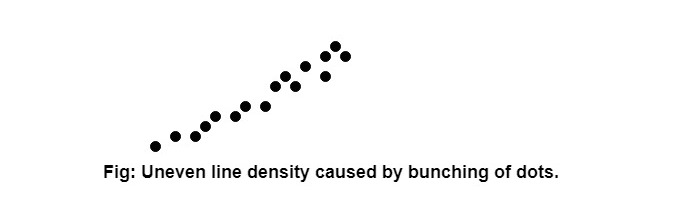
**1. Line should appear Straight:** We must appropriate the line by choosing addressable points close to it. If we choose well, the line will appear straight, if not, we shall produce crossed lines.



The lines must be generated parallel or at 45° to the x and y-axes. Other lines cause a problem: a line segment through it starts and finishes at addressable points, may happen to pass through no another addressable points in between.



**2. Lines should terminate accurately:** Unless lines are plotted accurately, they may terminate at the wrong place.



**3. Lines should have constant density:** Line density is proportional to the no. of dots displayed divided by the length of the line.

To maintain constant density, dots should be equally spaced.

**4. Line density should be independent of line length and angle:** This can be done by computing an approximating line-length estimate and to use a line-generation algorithm that keeps line density constant to within the accuracy of this estimate.

**5. Line should be drawn rapidly:** This computation should be performed by special-purpose hardware.

**Algorithm for line Drawing:**

1. Direct use of line equation
2. DDA (Digital Differential Analyzer)
3. Bresenham's Algorithm

**Direct use of line equation:**

It is the simplest form of conversion. First of all scan P1 and P2 points. P1 has co-ordinates (x1',y1') and (x2' y2' ).

Then       m = (y2',y1')/( x2',x1') and b = Scan Converting a Straight Line

If value of |m|≤1 for each integer value of x. But do not consider Scan Converting a Straight Line

If value of |m|>1 for each integer value of y. But do not consider Scan Converting a Straight Line

**Example:** A line with starting point as (0, 0) and ending point (6, 18) is given. Calculate value of intermediate points and slope of line.

**Solution:** P1 (0,0) P7 (6,18)

              x1=0  
              y1=0  
              x2=6  
              y2=18  
              Scan Converting a Straight Line

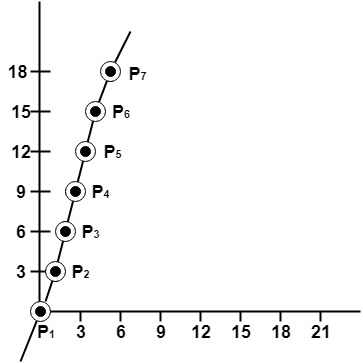
We know equation of line is  
              y =m x + b  
              y = 3x + b..............equation (1)

put value of x from initial point in equation (1), i.e., (0, 0) x =0, y=0  
              0 = 3 x 0 + b  
              0 = b ⟹ b=0

put b = 0 in equation (1)  
              y = 3x + 0  
              y = 3x

Now calculate intermediate points  
    Let x = 1 ⟹ y = 3 x 1 ⟹ y = 3  
    Let x = 2 ⟹ y = 3 x 2 ⟹ y = 6  
    Let x = 3 ⟹ y = 3 x 3 ⟹ y = 9  
    Let x = 4 ⟹ y = 3 x 4 ⟹ y = 12  
    Let x = 5 ⟹ y = 3 x 5 ⟹ y = 15  
    Let x = 6 ⟹ y = 3 x 6 ⟹ y = 18

So points are P1 (0,0)  
              P2 (1,3)  
              P3 (2,6)  
              P4 (3,9)  
              P5 (4,12)  
              P6 (5,15)  
              P7 (6,18)



**Algorithm for drawing line using equation:**

**Step1:** Start Algorithm

**Step2:** Declare variables x1,x2,y1,y2,dx,dy,m,b,

**Step3:** Enter values of x1,x2,y1,y2.  
              The (x1,y1) are co-ordinates of a starting point of the line.  
              The (x2,y2) are co-ordinates of a ending point of the line.

**Step4:** Calculate dx = x2- x1

**Step5:** Calculate dy = y2-y1

**Step6:** Calculate m = Scan Converting a Straight Line

**Step7:** Calculate b = y1-m\* x1

**Step8:** Set (x, y) equal to starting point, i.e., lowest point and xendequal to largest value of x.

              If dx < 0  
                  then x = x2  
              y = y2  
                        xend= x1  
        If dx > 0  
              then x = x1  
                  y = y1  
                        xend= x2

**Step9:** Check whether the complete line has been drawn if x=xend, stop

**Step10:** Plot a point at current (x, y) coordinates

**Step11:** Increment value of x, i.e., x = x+1

**Step12:** Compute next value of y from equation y = mx + b

**Step13:** Go to Step9.

**Program to draw a line using LineSlope Method**

1. #include <graphics.h>
2. #include <stdlib.h>
3. #include <math.h>
4. #include <stdio.h>
5. #include <conio.h>
6. #include <iostream.h>
8. class bresen
9. {
10. float x, y, x1, y1, x2, y2, dx, dy, m, c, xend;
11. public:
12. void get ();
13. void cal ();
14. };
15. void main ()
16. {
17. bresen b;
18. b.get ();
19. b.cal ();
20. getch ();
21. }
22. Void bresen :: get ()
23. {
24. print ("Enter start & end points");
25. print ("enter x1, y1, x2, y2");
26. scanf ("%f%f%f%f",sx1, sx2, sx3, sx4)
27. }
28. void bresen ::cal ()
29. {
30. /\* request auto detection \*/
31. int gdriver = DETECT,gmode, errorcode;
32. /\* initialize graphics and local variables \*/
33. initgraph (&gdriver, &gmode, " ");
34. /\* read result of initialization \*/
35. errorcode = graphresult ();
36. if (errorcode ! = grOK)    /\*an error occurred \*/
37. {
38. printf("Graphics error: %s \n", grapherrormsg (errorcode);
39. printf ("Press any key to halt:");
40. getch ();
41. exit (1); /\* terminate with an error code \*/
42. }
43. dx = x2-x1;
44. dy=y2-2y1;
45. m = dy/dx;
46. c = y1 - (m \* x1);
47. if (dx<0)
48. {
49. x=x2;
50. y=y2;
51. xend=x1;
52. }
53. else
54. {
55. x=x1;
56. y=y1;
57. xend=x2;
58. }
59. while (x<=xend)
60. {
61. putpixel (x, y, RED);
62. y++;
63. y=(x\*x) +c;
64. }
65. }

**OUTPUT:**

Enter Starting and End Points

Enter (X1, Y1, X2, Y2) 200 100 300 200

