Introduction to Computer Graphics
(C S 6 0 2)
Lecture 05
Line Drawing Techniques

5.1 Line

A **line**, or **straight line**, is, roughly speaking, an (infinitely) thin, (infinitely) long, straight geometrical object, i.e. a curve that is long and straight. Given two points, in Euclidean geometry, one can always find exactly one line that passes through the two points; this line provides the shortest connection between the points and is called a straight line. Three or more points that lie on the same line are called ***collinear***. Two different lines can intersect in at most one point; whereas two different planes can intersect in at most one line. This intuitive concept of a line can be formalized in various ways.

A line may have three forms with respect to slope i.e. it may have slope = 1 as shown in following figure (a), or may have slope < 1 as shown in figure (b) or it may have slope > 1 as shown in figure (c). Now if a line has slope = 1 it is very easy to draw the line by simply starting form one point and go on incrementing the x and y coordinates till they reach the second point. So that is a simple case but if slope < 1 or is > 1 then there will be some problem.

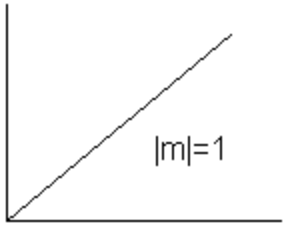


figure (a)

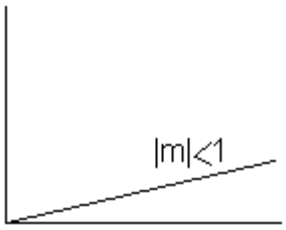


figure (b)

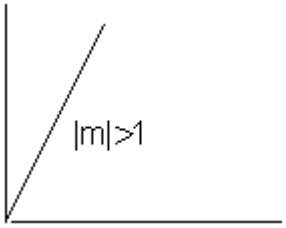


figure (c)

There are three techniques to be discussed to draw a line involving different time complexities that will be discussed later. These techniques are:

- Incremental line algorithm
- DDA line algorithm
- Bresenham line algorithm

5.2 Incremental line algorithm

This algorithm exploits simple line equation $y = m x + b$
Where $m = dy / dx$
and $b = y – m x$

Now check if |m| < 1 then starting at the first point, simply increment x by 1 (unit increment) till it reaches ending point; whereas calculate y point by the equation for

each x and conversely if |m|>1 then increment y by 1 till it reaches ending point; whereas calculate x point corresponding to each y, by the equation.

Now before moving ahead let us discuss why these two cases are tested. First if |m| is less than 1 then it means that for every subsequent pixel on the line there will be unit increment in x direction and there will be less than 1 increment in y direction and vice versa for slope greater than 1. Let us clarify this with the help of an **example**:

Suppose a line has two points p1 (10, 10) and p2 (20, 18)
Now difference between y coordinates that is dy = y2 – y1 = 18 – 10 = 8
Whereas difference between x coordinates is dx = x2 – x1 = 20 – 10 = 10
This means that there will be 10 pixels on the line in which for x-axis there will be distance of 1 between each pixel and for y-axis the distance will be 0.8.

Consider the case of another line with points p1 (10, 10) and p2 (16, 20)
Now difference between y coordinates that is dy = y2 – y1 = 20 – 10 = 10
Whereas difference between x coordinates is dx = x2 – x1 = 16 – 10 = 6

This means that there will be 10 pixels on the line in which for x-axis there will be distance of 0.6 between each pixel and for y-axis the distance will be 1.

Now having discussed this concept at length let us learn the algorithm to draw a line using above technique, called incremental line algorithm:

Incremental_Line (Point p1, Point p2)

```
dx = p2.x – p1.x
dy = p2.y – p1.y
m = dy / dx
x = p1.x
y = p1.y
b = y – m * x
if |m| < 1
        for counter = p1.x to p2.x
                drawPixel (x, y)
                x = x + 1
                y = m * x + b
else
        for counter = p1.y to p2.y
                drawPixel (x, y)
                y = y + 1
                x = ( y – b ) / m
```

Discussion on algorithm:
Well above algorithm is quite simple and easy but firstly it involves lot of mathematical calculations that is for calculating coordinate using equation each time secondly it works only in incremental direction.

We have another algorithm that works fine in all directions and involving less calculation mostly only addition; which will be discussed in next topic.

5.3 Digital Differential Analyzer (DDA) Algorithm:

DDA abbreviated for digital differential analyzer has very simple technique. Find difference dx and dy between x coordinates and y coordinates respectively ending points of a line. If |dx| is greater than |dy|, than |dx| will be step and otherwise |dy| will be step.

```
if |dx|>|dy| then
        step = |dx|
else
        step = |dy|
```

Now very simple to say that step is the total number of pixel required for a line. Next step is to divide dx and dy by step to get xIncrement and yIncrement that is the increment required in each step to find next pixel value.

```
xIncrement = dx/step
yIncrement = dy/step
```

Next a loop is required that will run step times. In the loop drawPixel and add xIncrement in x1 by and yIncrement in y1.

To sum-up all above in the algorithm, we will get,

DDA_Line (Point p1, Point p2)

```
dx = p2.x – p1. x
dy = p2.y – p1. y
x1=p1.x
y1=p1.y
if |dx|>|dy| then
        step = |dx|
else
        step = |dy|
xIncrement = dx/step
yIncrement = dy/step
for counter = 1 to step
        drawPixel (x1, y1)
        x1 = x1 + xIncrement
        y1 = y1 + yIncrement
```

Criticism on Algorithm:

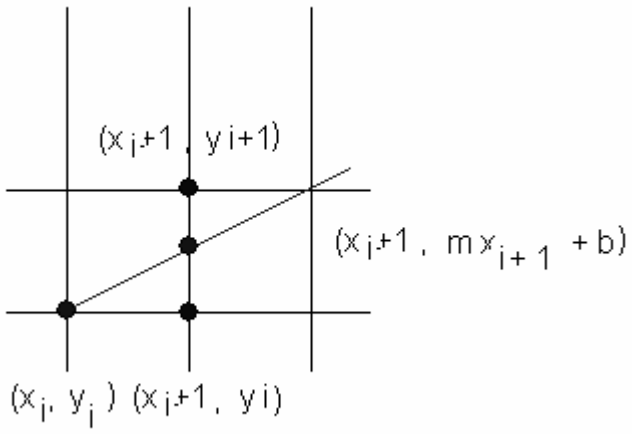
There is serious criticism on the algorithm that is use of floating point calculation. They say that when we have to draw points that should have integers as coordinates then why to use floating point calculation, which requires more space as well as they have more computational cost.

Therefore there is need to develop an algorithm which would be based on integer type calculations. Therefore, work is done and finally we will come up with an algorithm "Bresenham Line Drawing algorithm" which will be discussed next.

5.4 Bresenham's Line Algorithm

Bresenham's algorithm finds the closest integer coordinates to the actual line, using only integer math. Assuming that the slope is positive and less than 1, moving 1 step in the x direction, y either stays the same, or increases by 1. A decision function is required to resolve this choice.

If the current point is (x_i, y_i), the next point can be either (x_i+1,y_i) or (x_i+1,y_i+1) . The actual position on the line is $(x_i+1, m(x_i+1)+c)$. Calculating the distance between the true point, and the two alternative pixel positions available gives:



$$
\begin{aligned}
d_1 &= y - y_i \\
&= m * (x+1)+b-y_i \\
d_2 &= y_i + 1 - y \\
&= y_i + 1 - m (x_i + 1) - b
\end{aligned}
$$

Let us magically define a decision function p, to determine which distance is closer to the true point. By taking the difference between the distances, the decision function will be positive if d1 is larger, and negative otherwise. A positive scaling factor is added to ensure that no division is necessary, and only integer math need be used.

$$
\begin{aligned}
p_i &= dx\,(d_1-d_2) \\
p_i &= dx\,(2m * (x_i+1) + 2b - 2y_i-1) \\
p_i &= 2\,dy\,(x_i+1) - 2\,dx\,y_i + dx\,(2b-1) \quad \text{---------- (i)} \\
p_i &= 2\,dy\,x_i - 2\,dx\,y_i + k \quad \text{---------- (ii)}
\end{aligned}
$$

where $k=2\,dy + dx\,(2b-1)$

Then we can calculate pi+1 in terms of pi without any x_i , y_i or k .

p_{i+1} = 2 dy x_{i+1} – 2 dx y_{i+1} + k
p_{i+1} = 2 dy (x_i + 1) - 2 dx y_{i+1} + k since x_{i+1}= x_i + 1
p_{i+1} = 2 dy x_i + 2 dy- 2 dx y_{i+1} + k ---------- (iii)
Now subtracting (ii) from (iii), we get
p_{i+1} - p_i = 2 dy - 2 dx (y_{i+1} - y_i)
p_{i+1} = p_i + 2 dy - 2 dx (y_{i+1} - y_i)

If the next point is: (x_i+1,y_i) then

d_1<d_2 => d_1-d_2<0
=> p_i<0
=> **p_i+1= p_i + 2 dy**

If the next point is: (x_i+1,y_i+1) then

d_1>d_2 => d_1-d_2>0
=> p_i>0
=> **p_i+1= p_i + 2 dy - 2 dx**

The p_i is our decision variable, and calculated using integer arithmetic from pre-computed constants and its previous value. Now a question is remaining how to calculate initial value of p_i. For that use equation (i) and put values (x_1, y_1)

p_i = 2 dy (x_1+1) – 2 dx y_i + dx (2b-1)
where b = y – m x implies that
p_i = 2 dy x_1 +2 dy – 2 dx y_i + dx (2 (y_1 – mx_1) -1)
p_i = 2 dy x_1 +2 dy – 2 dx y_i + 2 dx y_1 – 2 dy x_1 - dx
p_i = 2 dy x_1 +2 dy – 2 dx y_i + 2 dx y1 – 2 dy x1 - dx

there are certain figures will cancel each other shown in same different colour

p_i = 2 dy - dx

Thus Bresenham's line drawing algorithm is as follows:

```
dx = x2-x1
dy = y2-y1
p  = 2dy-dx
c1 = 2dy
c2 = 2(dy-dx)
x  = x1
y  = y1
plot (x,y,colour)
while (x <  x2 )
     x++;
```

```
if (p < 0)
     p = p +  c1
else
     p = p +  c2
y++
plot (x,y,colour)
```

Again, this algorithm can be easily generalized to other arrangements of the end points of the line segment, and for different ranges of the slope of the line.

5.5 Improving performance

Several techniques can be used to improve the performance of line-drawing procedures. These are important because line drawing is one of the fundamental primitives used by most of the other rendering applications. An improvement in the speed of line-drawing will result in an overall improvement of most graphical applications.

Removing procedure calls using macros or inline code can produce improvements. Unrolling loops also may produce longer pieces of code, but these may run faster.

The use of separate x and y coordinates can be discarded in favour of direct frame buffer addressing. Most algorithms can be adapted to calculate only the initial frame buffer address corresponding to the starting point and to replaced:

X++ with Addr++
Y++ with Addr+=XResolution

Fixed point representation allows a method for performing calculations using only integer arithmetic, but still obtaining the accuracy of floating point values. In fixed point, the fraction part of a value is stored separately, in another integer:

M = Mint.Mfrac
Mint = Int(M)
Mfrac = Frac(M)× MaxInt

Addition in fixed point representation occurs by adding fractional and integer components separately, and only transferring any carry-over from the fractional result to the integer result. The sequence could be implemented using the following two integer additions: ADD Yfrac,Mfrac ; ADC Yint,Mint

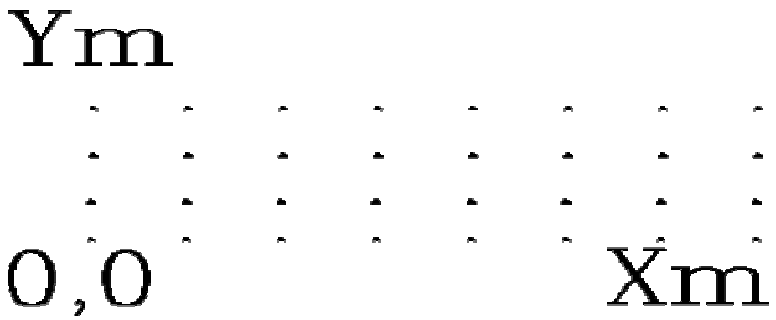
Improved versions of these algorithms exist. For example the following variations exist on Bresenham's original algorithm:

5.6 Symmetry (forward and backward simultaneously)

Segmentation (divide into smaller identical segments - GCD(D x,D y))
Double step, triple step, n step.

5.7 Setting a Pixel

Initial Task: Turning on a pixel (loading the frame buffer/bit-map). Assume the simplest case, i.e., an 8-bit, non-interlaced graphics system. Then each byte in the frame buffer corresponds to a pixel in the output display.



To find the address of a particular pixel (X,Y) we use the following formula:

addr(X, Y) = addr(0,0) + Y rows * (Xm + 1) + X (all in bytes)

addr(X,Y) = the memory address of pixel (X,Y)
addr(0,0) = the memory address of the initial pixel (0,0)
Number of rows = number of raster lines.
Number of columns = number of pixels/raster line.

Example:
For a system with 640 × 480 pixel resolution, find the address of pixel X = 340, Y = 150

addr(340, 150) = addr(0,0) + 150 * 640 (bytes/row) + 340
= base + 96,340 is the byte location

Graphics system usually have a command such as set_pixel (x, y) where x, y are integers.