An Insight to Breshenham Algorithm to Plot Straight Lines (with program in C++)

Prakash Gautam
B.Sc 3rd year
Tri-Chandra Multiple Campus
Ghantaghar Kathmandu Nepal

January 11 - 2014

pranphy@gmail.com http://pranphy.wordpress.com http://fb.com/pranphy

Breshenham Line Plotting Algorithm

It took me quite a while to appreciate the beauty of Brehsenham Line Drawing algorithm. Initially I thought It was just rounding off the decimal point to the nearest integer only in a zig-zag barcelona way but to my fascination it turned out to be tremendously interesting yet very very finely simple.

The equation of a straight line is $y=mx+c$. If two non-coincident points (x_0,y_0) and (x_1,y_1) be two given points. The equation of line passing through these two points will be.

$$y=y_0+\frac{y_1-y_0}{x_1-x_0}(x-x_0)---------(1)$$

$$y=y_0+\frac{\Delta y}{\Delta x}(x-x_0)$$
$$y=y_0+m(x-x_0)$$
$$y=mx+(y_0\text{-}mx_0) \quad \text{---------}(2)$$

Comparing it with $y=mx+c$ we see

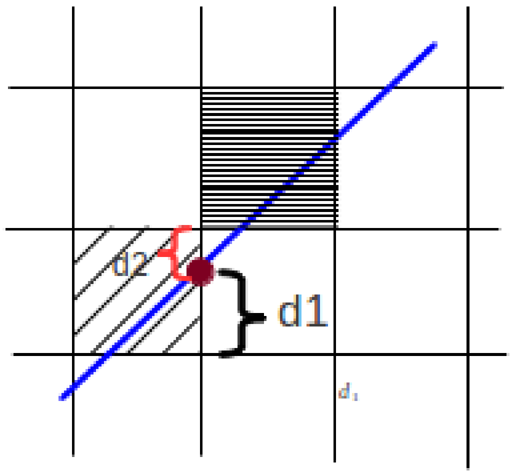
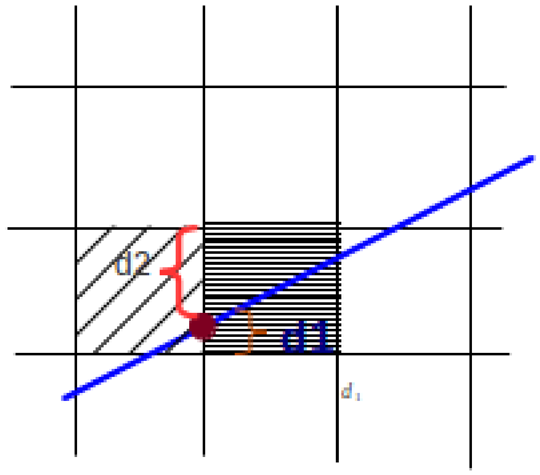
$$c=(y_0\text{-}mx_0)----(3)$$

It is not difficult to see if two points (x_0,y_0) and (x_1,y_1) are given then we can easily calculate the remaining points at any x with the use of equation (1)

Let us think how do we calculate the points with a computer program.
We would calculate the slope m and c Then start a loop from x_0 with step 1 and put value of x in equation 2 and simply find the values of y corresponding to the value of x .
The value of m is a floating point number. Computer needs bit more labour to calculate the result of floating point. Modern day VDU's are millions pixels in dimension. Just to draw a line across the VDU, poor CPU needs to do millions of floating point calculations which sucks more out of CPU. Lets fiddle around if we could somehow help the needy CPU here to avoid floating point multiplications.

Lets first concentrate on special case where the slope of line $0\leqslant m\leqslant 1$. If the first point is (x_0,y_0) and then with unit increase in x the value of y should increase by the value of slope m which is less than m . But m is a floating point number, and the pixel coordinates are always integers.

Let us assume that a intermediate point (x_k, y_k) is plotted already, the next point (x_{k+1}, y_{k+1}) is either (x_k+1, y_k) or (x_k+1, y_k+1) (Note in both cases x increases by just 1). If we could somehow decide which one between these two to choose then our line would be known by induction.
Let us find out which one of the integer is closer to the actual line. The actual coordinate of the line at x_K+1 is $y=m(x_k+1)+c$ The distance between the actual position from the two points to be decided are.

	
In the first figure the actual line(blue) crosses the y at x_k+1 at more than $y_k+1/2$ so the point (x_k+1, y_k+1) is chosen.	But in the second case the actual line crosses the vertical line at y coordinate less than $y_k+1/2$ so the point (x_k+1, y_k) is chosen.

$$d_1=m(x_k+1)+c-y_k$$
$$d_2=(y_k+1)-m(x_k+1)-c$$

We will chose the next y coordinate y_k if d_1 is smaller than d_2 .

$$d_1-d_2=2\text{m}x_k-2\text{y}_k+2\text{m}+2\text{c}-1$$

if $(d_1-d_2)>0$ we'll chose (x_k+1, y_k+1) or we'll choose (x_k+1, y_k) as the next point.
Let us further simplify it by multiplying both sides by Δx

$$P_k=\Delta x(d_1-d_2)=2\Delta y x_k-2\Delta x y_k+2\Delta y+\Delta x(2c-1)---(4)$$

This expression above is the discriminant which decides which one between the two points in contention do we chose. Let us call this to be P_k -the

decision parameter.
Since for $0\leqslant m\leqslant 1$; $\Delta x>0$ and the sign of d_1-d_2 is preserved with P_k .

$$P_{k+1}=2\Delta y x_{k+1}-2\Delta x y_{k+1}+2\Delta y+\Delta x(2c-1)$$
$$P_{k+1}=P_k+2\Delta x-2\Delta y(y_{k+1}-y_k) \quad \text{----(5)}$$

We are down to just knowing the sign of P_k to decide whether the next point after (x_k,y_k) is (x_k+1,y_k) or (x_k+1,y_k+1) . The first point undoubtedly is (x_0,y_0) but what is the first decision parameter. We don't beforehand know what y_{k+1} is before knowing what P_k is.
For first decision parameter. Putting

$$x_k=x_0$$
$$y_k=y_0$$
$$c=y_0-mx_0----[Equation\,3\,above]$$

in equation 4 above we get initial decision parameter

$$P_{intial}=2\Delta y\,x_0-2\Delta x\,y_0+2\Delta y+\Delta x(2(y_0-mx_0)-1)$$
$$P_{initial}=2\Delta y x_0-2\Delta x\,y_0+2\Delta y+2\Delta x\,y_0-2\Delta x\frac{\Delta y}{\Delta x}x_0-\Delta x$$
$$P_{initial}=2\Delta y-\Delta x$$

Writing out algorithm for this:

1. Get the end points and calculate Δx and Δy and hence $P=P_{initial}=2\Delta y-\Delta x$
2. Initial point $x=x_0$ and $y=y_0$

Begin :=

3. Plot (x,y)
4. Store previous y . $t=y$
5. if $P>0$ then $y\leftarrow y+1$
6. Update $P\leftarrow P+2\Delta x-2\Delta y(y-t)$ from equation 5 above.
7. Increase x. $x\leftarrow x+1$
8. Loop Untill $x\leq x_{final}$

With this algorithm we can plot the line with solpe $0\leqslant m\leqslant 1$.

But line in real life are never always restricted to have the slope $0\leqslant m\leqslant 1$. How can we extend this method to find the points of line in all the situation where the value of slope m may be any number from the real axis?

Ok, lets begin the ride.

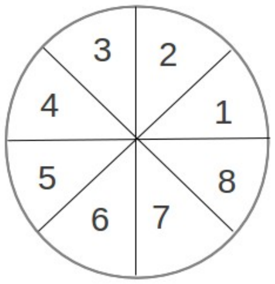
Lets us divide the Cartesian plane into 8 equal octant. The lines with slope with multiples of $\frac{\pi}{4}$ will suffice our division of the planes into octants. From

our journey so far we have completed the calculation of points of line anywhere in the first octant. Now the job is to conquer the rest of the octants. There is a nice symmetry between the rest of the octant with the first. There exist a unique transformation matrix that transforms the line back and forth between first and the rest octant. Now the job is almost over.
Identify the octant and transform it to first octant, calculate the points with the artifice we have done so far, and transform back to the original octant to plot.
Bravo!! we conquered everest.
Lets do the mathematics of the idea then.
Let us, for our ease, transform the co-ordinate system with axes parallel but the origin transferred to the initial point of line. We do this because we would not then have to worry about two points to determine a line, only the final point would describe the line completely as the first point will be the origin of our new coordinate system.
It is easy to determine the octant, as we have divided above, which the line lies in, simply by finding out which octant the final point lies in.

If the x coordinate is positive there are four candidate for the octant namely 1,2,8,7. Among these four the candidates are down to 1 and 2 if the y coordinate is +ve as well. And whether the y coordinate is greater than x determines whether the line lies on 2nd quadrant or first. Similarly the rest of the octants are determined.

The table below shows how to figure how the octant of the line with the coordinate of the end point.

$x \geq 0$	$y \geq 0$	$x \geq y$	1
		$x < y$	2
	$y < 0$	$\lvert x \rvert \geq \lvert y \rvert$	8
		$\lvert x \rvert < \lvert y \rvert$	7
$x < 0$	$y \geq 0$	$\lvert x \rvert \geq \lvert y \rvert$	4
		$\lvert x \rvert < \lvert y \rvert$	3
	$y < 0$	$\lvert x \rvert \geq \lvert y \rvert$	5
		$\lvert x \rvert < \lvert y \rvert$	6



After we have found out the octant of the line with this. Our task is transform every line into first octant then find the corresponding points on first octant and transform back to original octant to plot.
What transforms the points back and forth between the given octant and the octant number 1. All the points are transformed to first octant where the coordinate of the point is (x, y)

Octant	Original Point	Matrix to transform to 1st octant	Procedure	Matrix to transform from 1st octant.	Procedure
1	(x,y)	$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$	Do nothing	$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$	Do nothing
2	(y,x)	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$	Reverse x and y	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$	Swap coordinates.
3	$(-y,x)$	$\begin{bmatrix}0 & 1\\-1 & 0\end{bmatrix}$	Swap first and change sign of **second** coordinate	$\begin{bmatrix}0 & -1\\1 & 0\end{bmatrix}$	Swap and change sign of **first** coordinate.
4	$(-x,y)$	$\begin{bmatrix}-1 & 0\\0 & 1\end{bmatrix}$	Just reverse the sign of first coordinate.	$\begin{bmatrix}-1 & 0\\0 & 1\end{bmatrix}$	Just reverse the sign of first coordinate.
5	$(-x,-y)$	$\begin{bmatrix}-1 & 0\\0 & -1\end{bmatrix}$	Reverse the sign of both coordinates.	$\begin{bmatrix}-1 & 0\\0 & -1\end{bmatrix}$	Reverse the sign of both coordinates.
6	$(-y,-x)$	$\begin{bmatrix}0 & -1\\-1 & 0\end{bmatrix}$	Swap coordinates then change sign of both.	$\begin{bmatrix}0 & -1\\-1 & 0\end{bmatrix}$	Swap coordinates then change sign of both.
7	$(y,-x)$	$\begin{bmatrix}0 & -1\\1 & 0\end{bmatrix}$	Swap and change the sign of **first** coordinate.	$\begin{bmatrix}0 & 1\\-1 & 0\end{bmatrix}$	Swap first and change sign of **second** coordinate
8	$(x,-y)$	$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$	Just reverse the sign of second coordinate.	$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$	Just reverse the sign of second coordinate.

From above table we see that same matrix transforms back and forth between all the octants except for octant number 3 and 7. It seems that we have to keep track of whether we are transferring from octant 3 to 1 or 1 to 3. And similarly for octant 7.

We can do this by using one transformation to transform from given octant to the first. And the other to transform from 1st to the given octant. For all but 3 and 7 both the transformation would be same. Just two of these odd octants are trying to force us to make two transformations for the well behaved 6 of the rest. Can we get over the odds??
Luckily yes.
We get extremely lucky here, there is a symmetry between the odd ones. Once the point is transferred from 3rd to the 1st octant the point transforms back to 3rd octant exactly as the point is transferred from 7th to the first. And also once the point is transferred from 7th to 1st octant the point transfers back to the 7th octant exactly as the point is transferred from 3rd to 1st.
Lets go fooling around. First transfrom the point normally from the given octant. If the octant is one of 7 or 3 then fool the system by changing the octant index to 3 or 7 respectively. Then do normally.

Below is a computer program written in C++ implementing the algorithm.

```
#include<iostream>
using namespace std;
struct Point {int x,y;};

ostream& operator<<(ostream&,Point);
istream& operator>>(istream&,Point&);

Point operator+(Point,Point);
Point operator-(Point,Point);
int FindOctant(Point);
Point ProperPoint(Point,int);
void BreshenhamLine(Point,Point);

template <typename DataType>
void Swap(DataType&,DataType&);

int main()
{
    Point InitialPoint,FinalPoint;
    cin>>InitialPoint>>FinalPoint;
    BreshenhamLine(InitialPoint,FinalPoint);
    return 0;
}

void BreshenhamLine(Point InitialPoint,Point FinalPoint)
{
    Point P=FinalPoint-InitialPoint; //transfer origin
    int OctantIndex=FindOctant(P);
    P=ProperPoint(P,OctantIndex); //transform point
    OctantIndex=(OctantIndex==7?3:(OctantIndex==3)?7:OctantIndex);
//fooling around you see?
    Point CurrentPoint={0,0};
    int p=2*P.y-P.x;
    do
    {
        cout<<InitialPoint+ProperPoint(CurrentPoint,OctantIndex)<<endl;
//transfer origin back
        int t=CurrentPoint.y;
         if(p>0) CurrentPoint.y++;
         p=p+2*P.y-2*P.x*(CurrentPoint.y-t);
         CurrentPoint.x++;
     }
     while(CurrentPoint.x<=P.x);
 }

 int FindOctant(Point P)
 {
     if(P.x>=0)
         if(P.y>=0)
             return (P.y>P.x?2:1);
         else
             return (-P.y<P.x?8:7);
     else
         if(P.y>0)
             return (P.y>-P.x?3:4);
         else
             return(-P.y>-P.x?6:5);

 }

 Point ProperPoint(Point P,int d)
 {
     if(d==2||d==3||d==6||d==7)
          Swap(P.x,P.y);
     if(d==7||d==4||d==5||d==6)
          P.x*=-1;
    if(d==3||d==8||d==5||d==6)
        P.y*=-1;
    return P;
}

Point operator+(Point P1,Point P2)
{
    Point P;
    P.x=P1.x+P2.x;
    P.y=P1.y+P2.y;
    return P;
}

Point operator-(Point P1,Point P2)
{
    Point P;
    P.x=P1.x-P2.x;
    P.y=P1.y-P2.y;
    return P;
}

ostream& operator<<(ostream& op,Point p)
{
    cout<<"("<<p.x<<","<<p.y<<")";
    return op;
}

istream& operator>>(istream&ip,Point&p)
{
    cin>>p.x>>p.y;
    return ip;
}

template <typename DataType>
void Swap(DataType& p1,DataType&p2)
{
    DataType p;
    p=p1;
    p1=p2;
    p2=p;
```