Straight Lines: a step by step method

CHALOPIN François-Pierre
BOURDIN Jean-Jacques
Département Informatique
Université Paris 8
2, rue de la Liberté
93526 Saint-Denis Cedex 2
[jj-chalopin]@ai.univ-paris8.fr

Abstract
Drawing straight lines is a major field in computer graphics. Most methods are improvements of the Discrete Differential Analysis method first presented by Bresenham [1]. Combinatory analysis method, as presented by Castle [2] or Dulucq [3] or Berstel [4] are not commonly used: they imply multiple string copies and are therefore slow. A new approach, combining combinatory analysis and DDA is presented. The DDA does not apply to each point but to a step computed by combinatory method. This algorithm is tested and proves to be more than four times faster than Bresenham's algorithm.

Key Words : Algorithms, Computer Graphics, incremental curve generation, line generators.

1. Introduction

To draw a line is one of the most used function of any graphics displays. It is done by drawing a path between two extremities with maximum linearity. The points of the path have to be the discrete points closest to the ones of the real line. The Bresenham's algorithm [1] is often used. Two reasons concurs to this: this algorithm is fast and is easy to program. Some other approaches of producing the linear path exist [4, 3, 5, 2] but they imply string copies and, therefore, are not efficiently implemented. When improved algorithms are presented, they are based on the DDA method [6, 7, 8]. Our work has been to combine these two major approaches to produce a new algorithm. The benchmarks prove its improvements significant.

2. Properties of the Straight line

Let p (x_p, y_p) and q (x_q, y_q) be the extremities of the line. We want to produce the linear path [p, q]. This path will be mentioned as the *line* from p to q. The *continuous line* will refer to the line segment of the real plane. The line is an exact translation of the line from (0, 0) to (u, v) where:

$$u = x_q - x_p$$
$$v = y_q - y_v.$$

The values of u and v give the *slope* of the line. Hence we will refer to the line of slope v/u rather than to the extreme points of the segment. We will limit our studies on the case where $u > v \geq 0$. It has been noted [9] that other cases are simple symmetries of this one.
Here there is only one y_x for each x of the segment.
Let the *trace word* of the line v/u be the series of letters c_x where

$$c_x = y_{x+1} - y_x$$

The trace words have been studied [5, 3, 10, 4, 11, 12]. Some important properties have been demonstrated there and will be used hereafter. First of all, for each line, there are three different way of computing the approximation of the continuous segment:

- Considering the best approximation as in [1]
- Considering the integer part of the real ordinate (the greater minor integer).
- Considering the upper integer (the littler major integer).

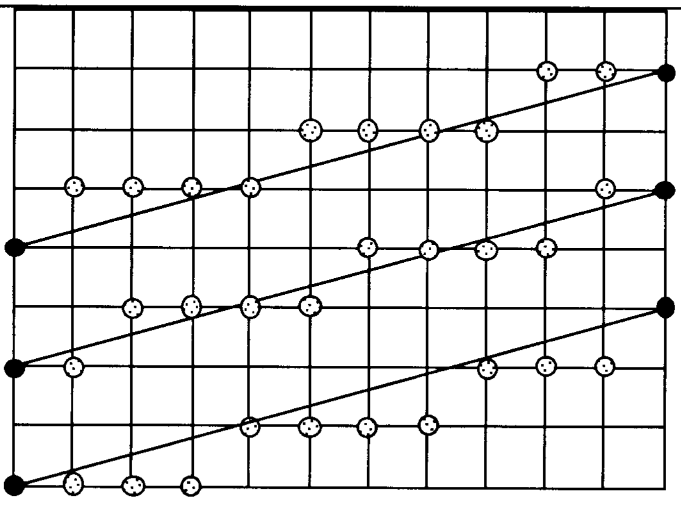

Figure 1 : Three lines for slope 3/11

Let w, w' and w" be the three respective trace words. These words are combined. For example, the three words of the line 3/11 are

w = 01000100010
w' = 00010001001
w" = 10010001000

Let n= 01, and ñ=10 the reverse of n, one can see that:

$$w.n = n.w'$$
$$\tilde{n}.w = w''.\tilde{n}$$

A left factor n of the trace word w exists in every case. Let l be the number of letters of the left factor n, Berstel [4] proves that l respects

$$v . l + [u/2] \equiv_u 0$$

Here l=2. This property permits the use of any of the three trace words. A simple translation will produce the best approximation from each of the two other words.

Therefore, in our works we focused on the word w'. It is the word our algorithm produces in the first place. The word w can be deduced from it.

3. Lines drawing algorithms

We present now two main algorithms to perform the drawing of a line. The first one [1] is the base of a whole field in computer graphics, the incremental, DDA, algorithms. The other one has not been presented in this form before, but is a rewriting of [3] or [2].

3.1. Bresenham's algorithm

The error done when one choose y_X as approximation of the value $\frac{v . x}{u}$ can not be greater than $\frac{1}{2}$. Moreover, the measure of the gap between the real value and the approximation is given by:

$$\text{gap}(x, y_X) = u.y_X - v.x$$

Let delta be the sum

$$\text{delta}(x, y_X) = \text{gap}(x, y_X) + \text{gap}(x, y_X+1)$$

The knowledge of the sign of delta is enough to know which one of y_X+1 or y_X is the best approximation of the real value. In C language algorithm 1 presents Bresenham's .

3.2. Recursive algorithm

Trace words are recurrences of 0 and 1 that form steps. For example if u>2.v the number of 0 (u—v) is more important than the number of 1 (v). Therefore the trace word is a long string of 0 separated by some 1. Let a *step* be a set of subsequent 0. As the word describes a straight line, the steps are almost of the same length. As each 1 is a separator of two different steps, there are v different steps. The average length of a step is then : $\frac{u - v}{v}$. That is, if this value is not an integer, some steps are $[\frac{u}{v}]$ wide, and others are $[\frac{u}{v}] - 1$ wide. The second kind of these steps are more oblique and the long kind are more horizontal. The distribution of the two kinds of steps is the most equitable. Then there are $(u-v) - v\frac{u-v}{v}$ long (horizontal) steps. The same method may be used to divide up the two kinds of steps. The method is therefore recursive. Inthe algorithm 2 the trace word of slope v/u is named s (u, v).

```
void Bresenham (int xp, yp, xq, yq)
{
   int x, y, delta, incobl, inchor;
   delta = 2* (yq - yp) - (xq - xp);
   incobl = 2* (yq - yp) - 2* (xq - xp);
   inchor = 2* (yq - yp);
   y = yp;
   for (x=xp; x<=xq; x++)
   {
      plot (x, y);
      if (delta > 0)
      {
         y++;
         delta = delta + incobl;
      }
      else
      {
         delta = delta + inchor;
      }
   }
}
```

Algorithm 1 : Bresenham's

If $v == 0$	then	$s(u, v) = 0^u$
If $u == v$	then	$s(u, v) = 1^u$
If $v == 1$	then	$s(u, v) = 0^{u-1}\,1$
If $[\frac{u}{v}] <= 1$	then	$s(u, v) = F_m (s(p, q))$, where
		$p = u - v$ (i.e. u mod v)
		$q = v \bmod (u-v)$
		$m = v \operatorname{div} (u-v)$ (Rq $m>=1$)
If $[\frac{u}{v}] > 1$	then	$s(u, v) = G_m (s(p, q))$, where
		$p = v$
		$q = v - (u \bmod v)$
		$m = (u - v) \operatorname{div} v$ (Rq $m>=1$)
Where		$F_m : \begin{cases} 0 \longrightarrow 0\,1^m \\ 1 \longrightarrow 0\,1^{m+1} \end{cases}$
and		$G_m : \begin{cases} 0 \longrightarrow 0^{m+1}\,1 \\ 1 \longrightarrow 0^m\,1 \end{cases}$

Algorithm 2 : recursive method

3.3. More remarks

If Bresenham's algorithm is the most used of these algorithms, it is first because it computes the best approximation of the line and second because it is the quickest. Even if there are a lot of improvements [6, 7, 8] the gain in speed is not realised in practice because of the inevitable pixel write operations. Otherwise algorithms as algorithm 2 are slowed down by the many string copies. We develop now a new method that combines the improvements of both methods by using their main characteristics.

4. Step DDA method

We are now to combine these two different approaches and implement a new method. The principle of our work is to consider the steps to be distributed as the letters of a straight line. The recursive function is therefore called only once. The algorithm is achieve with the use of the DDA algorithm. The DDA algorithm is rebuild to accept the steps as arguments.

The first part of our algorithm is done by the function **line** that is mainly a switch. For example, the line 3/11 is obtained by the used of the function **line (11, 3)**. This function computes the calling to the function **drawing.** here the call is **drawing (3, 1, 0001, 001).** The word 00010001001 is drawn by the putrow functions.

For example the trace word of line 31/97 is:

H H H H H H H O H H H H H H H O H H H H H H O H H H H H H H O

This is obtained by the call of the function **drawing (31, 4, 0001, 001)**

delta	−4	23	19	15	11	7	3	−1	26	22	18	14	10	6	2	
Step	h	o	o	o	o	o	o	h	o	o	o	o	o	o	o	
delta	−2	25	21	17	13	9	5	1	−3	24	20	16	12	8	4	0
Step	h	o	o	o	o	o	o	o	h	o	o	o	o	o	o	o

As soon as the putrow primitive is hardware implemented, for example on SGI Elan, the algorithm proves a good improvement in speed. When Bresenham's algorithm computes and draw the line pixel per pixel with a putpixel primitive, when Rokne's algorithm [7] computes the pixels two by two, our algorithm uses the putrow primitive and draw the line step by step.

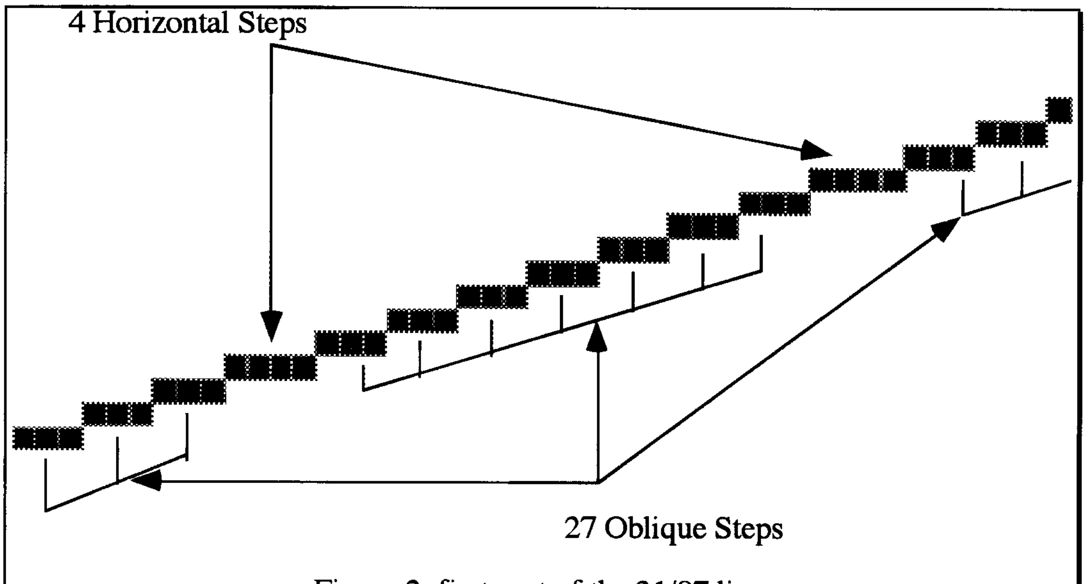

Figure 2: first part of the 31/97 line

```
drawing ( u, v, H, O) {
    incobl = v — u ;
    inchor = v ;
    delta  = v — u ;
    while (u — —) {
        if (delta ≥ 0) {
            putrow (O) ;
            delta += incobl ;
        }
        else {
            putrow (H) ;
            delta += inchor ;
        }
    }
}
```

algorithm 4: drawing function

```
line (u, v) {
    if (u > 2 * v && v > 0 ) {
        m = (u — v ) / v ;
        w = u % v ;
        H = $0^{m+1}1$;
        O = $0^{m}1$;
        drawing ( v, v — w, H, O) ;
    }
    elsif ( u > 0 && v > 0 ) {
        m =  v / (u — v)  ;
        w = v % (u — v ) ;
        H = $01^{m}$;
        O = $01^{m+1}$;
        drawing ( v, v — w, H, O) ;
    }
}
```

Algorithm 3: line function

The complexity of the algorithm is not o(u) but o(v). The minor width of a step is 2 (when $u \approx 2.v$) and then the speed up is only of a factor 2. In other cases, the speed up increases.

5. Benchmarks

We performed two series of benchmarks. The first one is done with drawing of the lines, without a putrow primitive. The drawing is achieved for 10 000 lines of random lines in the range given. Even in that case, the new algorithm is proved to be an improvement of previous algorithms.

	Bresenham's	**Castle & Pitterway's**	**recursive algorithm**	**New Algorithm**
(10-50)	1314	1651	1991	905
(50-100)	2417	2945	3125	1539
(100-150)	3457	4469	4633	2212
(150-200)	4957	5665	6157	2821
(200-250)	6051	7210	7422	3481
(250-300)	7159	8776	8965	4299
(300-350)	8485	9938	10002	5207
(350-400)	10125	11479	11785	5608
(400-450)	11077	12855	13294	6415
(450-500)	12461	14355	14543	7002

The second benchmark was done by testing only the two fastest algorithms above: Bresenham's and ours. For this benchmark the putrow primitive is considered and the results are even better. For each value u in the range, every value v is used. For the longest lines, the drawing was not possible as soon as the length of the line is greater than the screen definition. This does not change the ratio of time.

Sizes	Bresenham's	New Algorithm
1-200	221	55
200-400	1552	366
400-600	4218	972
600-800	8218	1875
800-1000	13555	3074
1000-1200	20223	4574
1200-1400	28228	6367
1400-1600	37569	8456
1600-1800	47244	10844
1800-2000	60255	13531
2000-2200	73590	16511
2200-2400	88269	19786
2400-2600	104270	23355
2600-2800	121638	27231
2800-3000	140319	31398

These results have been obtained on a Silicon Graphics International Indigo Elan. Other tests are in progress on other computer graphics hardware. But the ratio of speed up seems to be a little greater than 4 and remains constant.

6. Conclusion

A new algorithm for the drawing of straight lines have been presented. It is proved to be 4 times faster than previous algorithms. It seems to be very important to implement such an algorithm on hardware for any computer graphics device. Our further works consist in the implementation of this method to perform directly antialiased lines. We shall also try to mix the algorithm of Rokne [6] to this work. It could be another improvement.

References

[1] J.E. Bresenham, Algorithm for computer control of a digital plotter, IBM System Journal, Vol 4, n°1,p. 25-30, 1965.
[2] C.M.A. Castel, M.L.V. Pitteway, An Application of Euclid's Algorithm to Drawing Straight Lines, in Fundamental Algorithms in Computer Graphics, Springer-Verlag 1985, pp. 135-139.
[3] S. Dulucq, Cours de DEA Informatique, Université de Bordeaux, 1987.
[4] J. Berstel, Tracé de droites, fractions continues et morphismes itérés, M. Lothaire, "*Mots*", *Mélanges offerts à M.-P. Schützenberger*, Hermès 1990.
[5] M.L.V. Pitteway, The relationship between Euclid's algorithm and run-length encoding, in Fundamental Algorithms in Computer Graphics, Springer-Verlag 1985, pp. 105-112.
[6] P.L. Gardner, Modfications of Bresenham's algorithm for displays, IBM Tech. Disclosure Bull. 18, 1595-1596, 1975.
[7] J. G. Rokne, B. Wyvill, Xiaolin Wu, Fast Line Scan-Conversion, ACM Transactions on Graphics, Vol 9, N°4, October 1990.
[8] E. Angel, D. Morrison, Short Note: Speeding Up Bresenham's Algorithm, IEEE CG&A, Vol. 11, November 1991.
[9] J.D. Foley, A. Van Dam, S. Feiner, J. Hughes, *Computer Graphics, Principles and Practices, second edition*, Addison Wesley.
[10] S. Dulucq, D. Goyou-Beauchamps, Sur les facteurs des suites de Sturm, Theoretical Computer Science 71 (1990) 381-400.
[11] J.-P. Reveilles, Droites discètes et fractions continues, ULP Département d'Informatique, R90/01, Janvier 1990.
[12] A. Troesch, Interprétation géométrique de l'algorithme d'Euclide et reconnaissance de segments, Theretical Computer Science 115 (1993) 291-319.