Auto-Adaptive Step Straight-Line Algorithm

Vincent Boyer and Jean-Jacques Bourdin
Paris 8 University

One of the main functions of graphic displays is drawing straight lines. To draw lines quickly, the speed of the applied algorithm is critical. The following three different approaches exist:

- discrete differential analysis (DDA), introduced by Bresenham,[1]
- combinatory analysis,[2] and
- linguistic methods.[3,4]

The most famous approach remains the DDA, since extended by *N*-step algorithms. Here, we focus on this class of algorithms. Since three *N*-step algorithms have been published ($N = 2^5$, $N = 3^6$, $N = 4^7$), we analyzed them, studying only their time complexity because they compute the same approximation of the continuous line. Our analysis shows that improvements are small and don't support our objectives for speed.

We propose a new algorithm that uses other properties, some of them already presented.[8] We also compare the performances of these algorithms and present the theoretical analysis and benchmarks that prove the new algorithm is at least twice as fast as earlier ones.

Before discussing these algorithms in more detail, we introduce additional notations used throughout the article:

- "Line" denotes the discrete straight segment between two points $P(xp, yp)$ and $Q(xq, yq)$, and
- (u, v) describes the slope of lines $u = xq - xp$ and $v = yq - yp$.

Since the coordinates of P and Q are integers, the segment is an exact translation of line $(0, 0)(u, v)$. A line may be given by a set of points or by its first point and a set of moves. For slope (u, v), the set of moves is always composed using two different codes. In the first octant, the codes are

- X, representing an axial move, and
- D, representing a diagonal move.

N-step techniques

N-step algorithms use the DDA method. The main idea is to choose one pattern (a set of points) among different possible patterns. A succession of tests that compare predefined values and a term E realize this choice. Iterating this process determines the set of patterns composing the straight line.

A simple operation—number of iterations × number of instructions per loop—gives the time complexity. Let N be the number of points of one pattern. Let C be the number of possible patterns. Inside an iteration, let t be the number of tests, a the number of additions, and i the number of increments of E.

One operation (a sign test) realizes one of the tests; two operations (an addition and a sign test) realize the others.

For a given slope, the process is iterated p times ($p = u/N$). Each instruction (increment, sign test, addition) should take one unit of time. We calculate the theoretical evaluation performance (TEP) by

$$\text{TEP} = p(i + a + t)$$

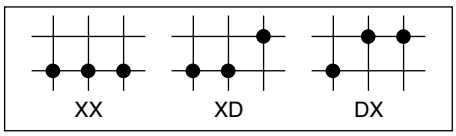
According to the chosen pattern, values i, t, and a may change.

We analyzed the worst and the best pattern cases. Based on an unproved equiprobability of all patterns, an average case completes the analysis.

We used four *N*-step algorithms: Bresenham's algorithm[1] ($N = 1$), the double-step[5] ($N = 2$), the triple-step[6] ($N = 3$), and the quad-step[7] ($N = 4$). For each algorithm we summarize the theoretical analysis in a table. All of these algorithms compute the line in a limited subset of the plane, mainly the octant or a subset of it. Throughout the article we call this limitation *workspace*.

Our new *N*-step algorithm for computing straight lines automatically chooses the *N* value. Benchmarks prove that the algorithm is at least twice as fast as those currently in use.

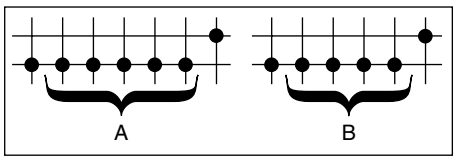
0272-1716/00/$10.00 © 2000 IEEE



1 Possible patterns of the double-step algorithm.

```
if (E>0) then
   pattern = X^A; E+=inc_XA;
else
   pattern = X^B; E+=inc_XB;
```

Figure 2. Auto-adaptive step algorithm.



3 Example of patterns X^A and X^B.

N = 1

Bresenham's straight-line algorithm[1] uses the DDA method and can be considered an *N*-step ($N = 1$) algorithm. In the first octant the possible patterns are X and D, so $C = 2$. Table 1 lists the results of this analysis.

N = 2

Rokne et al.[5] consider the two hexadecants (half of an octant) separately. We present the analysis for the first one. In this workspace the possible patterns are XX, XD, and DX, so $C = 3$ (see Figure 1). The double-step algorithm presents an average speedup of 15-percent on Bresenham's method (see Table 2).

N = 3

In the first hexadecant, the different possible patterns are XXX, DXD, XXD, XDX, and DXX, so $C = 5$. The triple-step algorithm presents an average speedup of 20 percent on Bresenham's method (see Table 3).

N = 4

In the quad-step algorithm[7] the octant is divided into six parts. The average workspace is one sixth of the octant. *C* always equals 5. The tests used for the choice of the workspace aren't included in the analysis. The quad-step algorithm presents an average speedup of 30 percent on Bresenham's method (see Table 4).

The main idea of these algorithms is to increase *N* to reduce the number of instructions and therefore the complexity. The main flaw of this analysis is that whenever *N* increases, *C* and the number of instructions increase. In addition, the performance improvement is often overestimated. For example, the double-step algorithm is considered to be two times faster than Bresenham's algorithm, but analysis and tests show only a 15 percent speedup.

Table 1. Analysis of the single-step algorithm.

Cases	*t*	*a*	*i*	*p*	TEP
Best	1	0	1	*u*	2*u*
Worst	1	0	1	*u*	2*u*
Average	1	0	1	*u*	2*u*

Table 2. Analysis of the double-step algorithm.

Cases	*t*	*a*	*i*	*p*	TEP
Best	1	0	1	*u*/2	*u*
Worst	2	1	1	*u*/2	2*u*
Average	5/3	2/3	1	*u*/2	5*u*/3

Table 3. Analysis of the triple-step algorithm.

Cases	*t*	*a*	*i*	*p*	TEP
Best	1	0	1	*u*/3	2*u*/3
Worst	4	3	1	*u*/3	8*u*/3
Average	14/5	1	1	*u*/3	8*u*/5

Table 4. Analysis of the quad-step algorithm.

Cases	*t*	*a*	*i*	*p*	TEP
Best	1	0	1	*u*/4	*u*/2
Worst	4	3	1	*u*/4	2*u*
Average	14/5	9/5	1	*u*/4	7*u*/5

Table 5. Analysis of the AAS algorithm.

Cases	*t*	*a*	*i*	*p*	TEP
Best	1	0	1	*v*	2*v*
Worst	1	0	1	*v*	2*v*
Average	1	0	1	*v*	2*v*

Auto-adaptive step algorithm

Our new algorithm, shown in Figure 2, uses the following properties:

1. All lines can be computed in the first half of the first octant (the hexadecant). Hereafter, we consider the lines to be in this hexadecant.
2. In the first hexadecant, a line is composed of axial moves X separated by one and only one diagonal move D:

$$L = X^{n_1}DX^{n_2}DX^{n_3}D\ldots X^{n_{m-1}}DX^{n_m}$$

   where X^n denotes *n* times pattern X, and n_j is the length of the *j*th step.
3. The length of the step is almost constant. If $A = \max_{j=2}^{m-1}(n_j)$ and $B = A - 1$, then

$$\forall j \in]2, m-1[, n_j \in \{A, B\}$$

Moreover $n_1 + n_m \in \{A,B\}$. Consequently, values n_1 and n_m are calculated separately. In the first hexadecant we

have $A = (u - v)/v$. An iteration consists in choosing one of two different patterns, X^A and X^B ($C = 2$). Figure 3 presents an example with $A = 5$ and $B = 4$. A simple sign test (see Figure 2) helps with the choosing. Table 5 presents the analysis of the auto-adaptive step (AAS) algorithm. Note that with this method the number of iterations p equals the number of Ds that are v. In the first hexadecant $v < u/2$; therefore the speedup is at least 2.

Performances

We've implemented and tested these algorithms so that we can compare them. We computed every line, starting from (0, 0) and ending in the first hexadecant. The Max value corresponds to the maximal length of the lines. We tested all five algorithms.

Table 6 presents the times (in seconds) obtained by the total time field of the Unix profile program. We used a Compaq Professional Workstation XP1000 with a 500-MHz Alpha 21264 CPU to realize these benchmarks. The percent value is the ratio between the current algorithm and Bresenham's algorithm.

Conclusion

The AAS algorithm to compute straight lines improves Bresenham's DDA and the *N*-step algorithms. It determines the better step according to the slope of the line. The number of instructions remains constant per iteration. Like Bresenham's, this method always chooses one pattern from two possible patterns. Moreover, like the other *N*-step algorithms, the step's length increases, and the number of iterations decreases with it. Both the theoretical evaluations and the benchmarks prove that this new method is more efficient than previous ones. ■

Table 6. *N*-step algorithm performances in seconds.

Max	1	2	3	4	AAS
500	0.2559	0.2256	0.1973	0.2031	0.0742
1,000	2.0088	1.7793	1.6445	1.5030	0.5605
2,000	16.1641	13.8379	12.9570	11.42	4.3555
4,000	130.3613	110.743	100.46	87.47	34.51
Percent	100	84.95	77.06	67.00	26.47

References

1. J.E. Bresenham, "Algorithm for Computer Control of a Digital Plotter," *IBM System J.*, Vol. 4, No. 1, 1965, pp. 25-30.
2. C.M.A. Castle and M.L.V. Pitteway, "An Application of Euclid's Algorithm to Drawing Straight Lines," *Fundamental Algorithms in Computer Graphics*, Springer-Verlag, Berlin, 1985, pp. 135-139.
3. R. Brons, "Linguistic Methods for Description of a Straight Line on a Grid," *Computer Graphics and Image Processing (CGIP)*, Vol. 3, 1974, pp. 48-62.
4. L. Wu, "On the Chain Code of a Line," *IEEE Trans. Patterns Analysis and Machine Intelligence (PAMI),* Vol. 4, No. 3, 1982, pp. 347-353.
5. J.G. Rokne, B. Wyvill, and X. Wu, "Fast Line Scan Conversion," *ACM Trans. Graphics*, Vol. 9, No. 4, Oct. 1990, pp. 376-388.
6. P. Graham and S.S. Iyengar, "Double and Triple Step Incremental Generation of Lines," *IEEE Computer Graphics and Applications*, Vol. 14, No. 3, May 1994, pp. 49-53.
7. G. Gill, "*N*-Step Incremental Straight-Line Algorithms," *IEEE Computer Graphics and Applications,* Vol. 14, No. 3, May 1994, pp. 66-72.
8. V. Boyer and J.-J. Bourdin, "Fast Lines: a Span by Span Method," *Proc. Eurographics 99,* Computer Graphics Forum Series, Blackwell Publishers, Oxford, U.K., Vol. 18, No. 3, Sept. 1999, pp. 377-384.

Vincent Boyer *is a PhD student in the Computer Science Department at Paris 8 University. His research interest is in high-performance algorithms and, particularly, fast 2D primitives. He introduced a new model for color shading and improved the methods for the generation of nonparametric curves, antialiasing computation, and 3D discrete lines computation. He is a member of the Group for Research in Image Synthesis (GRIS).*

Jean-Jacques Bourdin *is an assistant professor in the Computer Science Department of Paris 8 University. He founded the GRIS, a group that focuses on fast algorithms for paint boxes. His academic interest is to lead students from the undergraduate level to a PhD thesis. Bourdin received a PhD in computer science from Bordeaux University and a French postdoctoral degree (HDR) from Paris 8 University. He is a member of ACM and, Eurographics, and serves on the AFIG board (the French association of computer graphics).*

Readers may contact the authors at Groupe de Recherche en Infographie et Synthèse d'images, Laboratoire d'Intelligence Artificielle, Université Paris8, 93 526 Saint-Denis, France, e-mail {boyer, jj} @ai.univ-paris8.fr.