Regularity and digital lines

F. Feschet[1], A. Fraboulet[2] and S. Bonnevay[1]
([1]) UMR 5823 - LASS, University Lyon 1, Bat 101,
43 bd du 11 Nov. 1918, Villeurbanne, France
([2]) L3I, Bat Blaise Pascal, INSA de Lyon,
20 Av Albert Einstein, Villeurbanne, France

May 29, 2001

Abstract

In this paper, we study and compare different drawing digital line algorithms. The first one draws lines pixel by pixel in an incremental way, the second one extends this incrementation to step by step and the last one proposes to automatically determine the length of the steps used for each incrementation. Here, we propose to develop the theorical approach of the third one in view to prove that the number of repetitions of steps for each incrementation can be computed. From this study an efficient algorithm is performed. A theoretical performance analysis of this algorithm and the others is displayed and compared to real experiments. Results show a difference between theoretical performances and true ones. Thus, we introduce a precise analysis of the influence of the machine hardware in order to explain the reasons of such a gap between theoretical and real performances. A chip point of view is also proposed to compare those algorithms in the context of an hardware implementation.

1 Introduction

In this paper, we present an arithmetical analysis of digital lower lines and a precise analysis of the performances of a new drawing algorithm. We propose an explanation to the difference between theoretical speedup and measured ones. Our approach is based upon a circuit study of three drawing algorithms. For a VLSI implementation, increasing the regularity of the drawing operations should lead to increase in the speed of the drawing. This conducts us to define the regularity of such lines by replication of groups of patterns thus extending preceeding works on drawing[4, 9, 2]. In a recent paper, Stephenson and Litow[10] proposed an approach to describe a digital line as a hierarchy, each level of the hierarchy consisting of groups of elements of a lower level. We provide such a decomposition as a consequence of our arithmetical analysis. Moreover, contrarily to their approach, our analysis permits to compute the groups of a level h with a succession of h Euclidean decompositions and not incrementaly as they proposed. Furthermore, we propose to study the impact of such a deep characterization of the structure of a line onto the performances of the drawing algorithms.

Many various approaches have been previously proposed to draw digital lines, using words[5, 8], combinatorics[6, 12, 11] or arithmetics[7]. The most famous one for drawing is the N-step algorithm [9] which extend the DDA (Discrete Differential Analysis) introduced by Bresenham[4]. As noticed by Boyer and Bourdin[3, 2], increasing the value of N leads to faster algorithms but the achieved speedup is limited due to an increase in the number of tests. They have thus presented a method which decide how many pixels belong to one horizontal step of lines in the first hexadecant. Their algorithm simply con-

sists in just deciding which of the two possible lengths for the step must be drawn. Our method propose a precise analysis of lower digital lines and we propose a new drawing algorithm which includes Boyer and Bourdin's one as a particular case.

Let us recall that given two points $O = (0,0)$ and $F = (u,v)$, we call line, the digital line passing through O and F. When restricted to digital segment, only the last step is cut to end the drawing at F. By integer translation of (t_x, t_y), it is possible to draw all lines passing through $A = (t_x, t_y)$ and $B = (u + t_x, v + t_y)$. In the first octant, all lines consist of a horizontal step (simply called a step in the sequel) separated by a diagonal move. It is well known that except for the first and the last step, the lengths of any step is either q or $q+1$ with $q \in \mathbb{N}$. This explains why the Boyer and Bourdin algorithm computes the length of the step, drawn it and repeat the process for the next one. It is clear that drawing a step is cheap by a memory set operation thus making their algorithm fast. However, this does not use the full regularity of lower lines. We begin this article by basic definitions which permits to easily obtain the well-known theorem that a digital segment is composed of steps of only two differents lengths (when not considering the first and the last one). Then we propose to precisely study the number of consecutive steps having the same length in a digital segment. This leads us to propose an algorithm in three parts. Then, after having computed the theoretical performances of our approach, we propose to study its real running-time. A deep analysis of the results is proposed showing that the machine has a great influences on the speedup of the algorithms. We then study the influence of a VLSI implementation of the compared algorithms. We finally gave our conclusions on the notion of *fast* drawing algorithms.

2 Arithmetical analysis

2.1 Methodology

Let us suppose that $u, v \in \mathbb{R}$ and that $0 < v \leq u$. The real line through O and F has the equation: $y = \frac{v}{u}x$. We decompose u by v: $\exists q \in \mathbb{N}^* \exists r \in [0, v[, u = q.v + r$. This can be repeated to v and r when $r \neq 0$: $\exists p \in \mathbb{N}^* \exists s \in [0, r[, v = p.r + s$.

definition 1 *Let $n \in \mathbb{N}$. We call* barrier *of level n, the lowest integer $h(n)$ such that: $\forall x \in \mathbb{R}, x \geq h(n) \Rightarrow \frac{v}{u}x \geq n$.*

Clearly, using $v \neq 0$, $\frac{v}{u}x \geq n \Leftrightarrow x \geq n\frac{u}{v}$ and thus using $u = q.v + r$ leads to $x \geq nq + n\frac{r}{v}$ such that $h(n) = nq + \lceil n\frac{r}{v} \rceil$.

definition 2 *Let $n \in \mathbb{N}^*$. We call* step *of level n, the integer number $l(n)$ defined by, $l(n) = h(n) - h(n-1)$.*

Clearly, using the definition of $h()$, we get $l(n) = q + \lceil n\frac{r}{v} \rceil - \lceil (n-1)\frac{r}{v} \rceil$. The geometrical meaning of these notions is presented in Fig. 1.

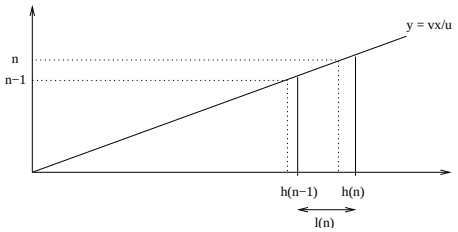

Figure 1: Geometrical meaning of barrier and step notions

property 1 $\forall n \in \mathbb{N}^*,\ l(n) \in \{q, q+1\}$

Let $n \in \mathbb{N},\ n \geq 2$ then $\exists k \in \mathbb{N}^*,\ k < (n-1)\frac{r}{v} \leq k+1$. Using $0 \leq \frac{r}{v} < 1$, we get $k \leq k + \frac{r}{v} < n\frac{r}{v} \leq k + 1 + \frac{r}{v} < k + 2$. Thus, $\lceil n\frac{r}{v} \rceil \in \{k+1, k+2\}$ and $\lceil (n-1)\frac{r}{v} \rceil = k+1$, which imply the result.

This result is well known and justified the search for the alternating sequence of steps of length q and $q+1$. The step length variations are just given by $\Delta(n) = \lceil n\frac{r}{v} \rceil - \lceil (n-1)\frac{r}{v} \rceil$.

property 2 *If $\frac{2r}{v} \leq 1$ then $\forall n \in \mathbb{N}^*$,*

$$\Delta(n) = 1 \Rightarrow \Delta(n+1) = 0$$

Let us suppose $\Delta(n) = 1$ with $n \in \mathbb{N}^*$. Let us call k the number $\lceil (n-1)\frac{r}{v} \rceil$ thus it is clear that $\lceil n\frac{r}{v} \rceil =$

$k+1$. We also have, $\Delta(n+1) = \lceil (n+1)\frac{r}{v} \rceil - \lceil n\frac{r}{v} \rceil$. But, since $(n+1)\frac{r}{v} = (n-1)\frac{r}{v} + 2\frac{r}{v}$ and $\frac{2r}{v} \leq 1$, using $k \geq (n-1)\frac{r}{v}$ leads to $\lceil (n+1)\frac{r}{v} \rceil = k+1$ and thus to $\Delta(n+1) = 0$.

A similar analysis leads to the following property.

property 3 *If* $\frac{2r}{v} \geq 1$ *then* $\forall n \in \mathbb{N}^*$,

$$\Delta(n) = 0 \Rightarrow \Delta(n+1) = 1$$

In the previous properties, we can deduce nothing from the value $\Delta(n) = 0$ when $\frac{2r}{v} \leq 1$ and $\Delta(n) = 1$ when $\frac{2r}{v} \geq 1$. Thus, it is possible to have a repetition of steps of length q in the first case and $q+1$ in the later one. However, a $q+1$ step is always followed by a q step in the former case and a q step is always followed by a $q+1$ in the later one. To describe the regularity of digital line, we might stop the analysis here leading to Boyer and Bourdin's algorithm. However, we suspect that more regularity can be given by a carefull analysis of the number of consecutive steps of the same length. This analysis is done in section 2.4. Let us notice that when $r = 0$, there exists only steps of length q which are thus simply repeated $\lfloor v \rfloor$ times.

Let us also remark that from the previous analysis, it is obvious to see that the probabilities of appearance of q step and of a $q+1$ step are not equiprobable, one is prefered to the other depending on the value of $\frac{r}{v}$.

2.2 Repetition of steps of length q

We now suppose that $0 < \frac{2r}{v} \leq 1$. Let n be an integer, $n \neq 0$ such that $\Delta(n) = 1$, thus knowing that $\Delta(n+1) = 0$. Let us define $k = \lceil (n-1)\frac{r}{v} \rceil$ and thus $\delta = k - (n-1)\frac{r}{v}$ belongs to $[0, 1[$. We know that $\exists a \in [0, r[, \delta = \frac{a}{v}$ from the fact that $\lceil n\frac{r}{v} \rceil = k+1$, which implies $k < n\frac{r}{v} \Leftrightarrow r > vk - (n-1)r$.

If we denote by β the number $n\frac{r}{v} - k$, then we have $\beta = \frac{r-a}{v}$. As we know that $\Delta(n+1) = 0$, we look for the greatest integer m such that $\lceil (n+m)\frac{r}{v} \rceil = k+1$. But, $(n+m)\frac{r}{v} \leq k+1 \Leftrightarrow m\frac{r}{v} + \beta \leq 1$ which is then equivalent to $mr \leq v - (r-a)$. Thus by using that $v = pr + s$ and that $r \neq 0$, we deduce the following equivalence

$$\lceil (n+m)\frac{r}{v} \rceil = k+1 \Longleftrightarrow m \leq p - 1 + \frac{a+s}{r}$$

Since m is an integer, $m = p - 1 + \lfloor \frac{a+s}{r} \rfloor$. Moreover, since $a < r$ and $s < r$, we have $\lfloor \frac{a+s}{r} \rfloor < 2$ and thus $m \in \{p, p-1\}$. The decision process depends only on the difference between a and $r - s$,

1. $m = p \Longleftrightarrow a \geq r - s$
2. $m = p - 1 \Longleftrightarrow a < r - s$

It is also possible to express the evolution of δ and a computing $\beta + m\frac{r}{v}$,

1. case $m = p$, $\beta + m\frac{r}{v} = 1 - \frac{a-(r-s)}{v}$ thus δ becomes $\frac{a-(r-s)}{v}$ and a becomes $a - (r-s)$,
2. case $m = p-1$, $\beta + m\frac{r}{v} = 1 - \frac{a+s}{v}$ thus δ becomes $\frac{a+s}{v}$ and a becomes $a + s$

The previous analysis produces the following algorithm for drawing a digital line,

1. $\Delta \leftarrow r - s$
2. $a \leftarrow 0$
3. fill$(q+1)$ *// a step of length* $q+1$
4. if $(a < \Delta)$ then
5. rep $\leftarrow p - 1$
6. $a \leftarrow a + s$
7. else
8. rep $\leftarrow p$
9. $a \leftarrow a - \Delta$
10. end if
11. for $i = 1$ to rep do
12. fill(q)
13. end for
14. if not at F then goto 3

In order to draw a segment, the call *fill(q + 1)* is modified for the step containing F. Moreover, when $\frac{2r}{v} = 1$ there is an alterning sequence of steps of length $q+1$, q.

2.3 Repetition of steps of length $q+1$

In this part we suppose that $\frac{2r}{v} > 1$. Let n be a natural number, $n \neq 0$, such that $\Delta(n) = 0$. We thus deduce that $\lceil n\frac{r}{v} \rceil = \lceil (n-1)\frac{r}{v} \rceil = k$. We also know that $\Delta(n+1) = 1$. As in the previous case, we denote by δ the value $k - n\frac{r}{v}$ and $\exists a \in [0, v[$ such that $\delta = \frac{a}{v}$. Let us now note that for all natural i then $\Delta(n+i) = 1 \iff \lceil (n+i)\frac{r}{v} \rceil = k+i$ is a necessary and sufficient condition. Hence, rewriting $(n+i)\frac{r}{v}$ as $k + i - \frac{is+a}{v}$ permits to conclude that $\lceil (n+i)\frac{r}{v} \rceil = k+i \iff \frac{is+a}{v} < 1$. However, since $v = pr+s$ and $v < 2r$ and $0 < r < v$, $v = r+s$. Then, using $s \neq 0$ and $a \geq r$, we obtain that $i < 1 + \frac{r-a}{s}$ which is equivalent to $i < \frac{v-a}{s}$. Thus, $\Delta(n+i) = 1 \iff i \leq \lfloor \frac{v-a}{s} \rfloor$.

Contrary to the previous case, since the first step has a $q+1$ length it is necessary to study the influence of the first repetition onto the whole procedure. It is sufficient to set $a = 0$ thus leading to $\lfloor \frac{v}{s} \rfloor$ repetition. Since $v = r+s$, we check $\lfloor \frac{v}{s} \rfloor = 1 + \lfloor \frac{r}{s} \rfloor$. We thus decompose r in function of s: $\exists c \in \mathbb{N}, d \in [0, s[, r = cs + d$. Thus expressing $1 + \lfloor \frac{r}{s} \rfloor$ as $1 + c$. Then, we note that

1. $c\frac{r}{v} = c - \frac{r-d}{v}$, which implies that $\lceil c\frac{r}{v} \rceil = c$,
2. $(c+1)\frac{r}{v} = c + \frac{d}{v}$, which implies that $\lceil (c+1)\frac{r}{v} \rceil \in \{c, c+1\}$ (depending whether $d = 0$ or not),
3. $(c+2)\frac{r}{v} = c + 1 - \frac{s-d}{v}$, which implies that $\lceil (c+2)\frac{r}{v} \rceil = c+1$,

For the second case, we further have: $d = 0 \Rightarrow a = 0$ and $d \neq 0 \Rightarrow a = s - d$.

Let us now focus on the repetition of the step of length $q+1$. The greatest value for i is $\lfloor \frac{v-a}{s} \rfloor$ which is equal to $\lfloor \frac{(c+1)s+d-a}{s} \rfloor$. Thus, it implies that the greatest value for i is bounded by $c+1$ depending on d is greater or not than a. Let us recall that $\lceil n\frac{r}{v} \rceil = \lceil (n-1)\frac{r}{v} \rceil = k$ and that $a = \delta v$ with $\delta = k - n\frac{r}{v}$. Thus $a = kv - nr$. Moreover, $a + r = v(k - (n-1)\frac{r}{v}$ thus leading to $a + r < v$. As a consequence, $s > 0$ implies that $\frac{r}{s} < \frac{v-a}{s}$ which leads to $\lfloor \frac{r}{s} \rfloor \geq \lfloor \frac{v-a}{s} \rfloor$. If we add that $c = \lfloor \frac{r}{s} \rfloor$ we deduce that $\lfloor \frac{v-a}{s} \rfloor \in \{c, c+1\}$. We then reduce the computation of the greatest i to the following,

1. if $d > a$ then $i = c+1$ and $(n+c+1)\frac{r}{v} = k + c + \frac{d-a}{v}$ and $(n+c+2)\frac{r}{v} = k + c + 1 - \frac{s-d+a}{v}$ thus leading to a new value for a equal to $a+s-d$,
2. if $d \leq a$ then $i = c+1-\lfloor \frac{a-d}{s} \rfloor$. The case $a - d \geq s$ is impossible thus $a - d < s$ and $i = c+1$. Then, $(n+c+1)\frac{r}{v} = k + c - \frac{a-d}{v}$ and $(n+c+2)\frac{r}{v} = k + c + 1 - \frac{s+a-d}{v}$. Hence, the new value for a is equal to $a - d$.

The previous analysis leads to the following algorithm,

1. $\Delta \leftarrow s - d$
2. $a \leftarrow 0$
3. if $(a < d)$ then
4. rep $\leftarrow c + 1$
5. $a \leftarrow a + \Delta$
6. else
7. rep $\leftarrow c$
8. $a \leftarrow a - d$
9. end if
10. for $i = 1$ to rep do
11. fill$(q+1)$
12. end for
13. fill(q)
14. if not at F then goto 3

2.4 Further analysis

In this part, we present three examples, two with short segments and a third one with a not so short one. In figure 2, a digital lower segment is drawn for the case $\frac{2r}{v} < 1$. In that case, all the points are drawn in just one pass of the proposed algorithm. The choosen segment is in the first hexadecant.

As we have seen before, the decision process depends on the comparison between $\frac{2r}{v}$ and 1. However, this does not depend on the first hexadecant

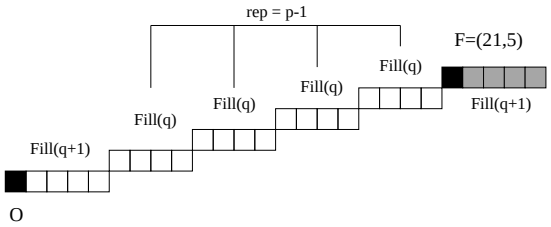

Figure 2: An example of segment when $\frac{2r}{v} < 1$

as shown in figure 3 where another segment - corresponding the the second case $\frac{2r}{v} > 1$ - of this first hexadecant is drawn.

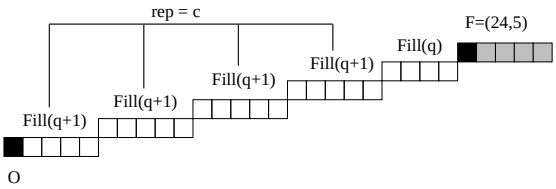

Figure 3: An example of segment when $\frac{2r}{v} > 1$

As we can easily see on those first two examples, the speedup of the proposed algorithm is high due to high values for p and c. When increasing the slope of the digital lines, those both values decrease as in figure 4. However, even in that case, we achieved a non neglectable speedup because most of the drawing is done without any test, addition or comparison.

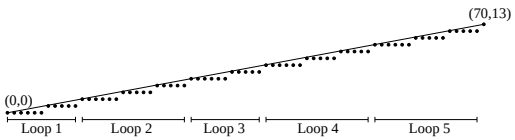

Figure 4: An example of long segment with $\frac{2r}{v} < 1$

One might notice that it is possible to also study the repetitions of the repetitions with a similar method as the one proposed in this paper. We have not done it because it generates a new series of euclidean divisions whose costs seem not to be hidden by speedup in the process. Moreover, it is also possible to use a symmetry property such that only half of the line must be effectively drawn. Since we want to study the repetition factors and their influences on the running time, we do not use this property during the experiments.

3 Experiments

In this part, we propose two kinds of performance analysis, a theoretical one and a truly experimental one. Our goal is to compare the true speedups that can be reached in practice. In the next section, we will focus on an hardware analysis, here we only compare software implementations on a general purpose computer (Pentium III 500 Mhz with 128 Mb under Debian).

We propose to compare the theoretical performances and the true performances of the Bresenham, Boyer and Bourdin and our own algorithm. Let us notice that in the true experiment, running time of Bresenham algorithm is similar to the one of Boyer and Bourdin[3]. However, as we will see in the second part of this section, this does not mean that their algorithm is much faster than Bresenham one. The protocol of the comparison is given on each subsection, but for the Boyer and Bourdin algorithm, we used the code proposed in their Eurographics 99 paper[2]. Note that their algorithm contains many misprints such as the behaviour when $v = 0$ or $v = 1$. Moreover, some lines are incorrect. For instance when $u = 1000$ and $v = 63$, they generate 15, 15, 16, 16 as a series of moves strictly inside the lines, but this is forbidden. A smaller line having this problem can be obtained when $u = 70$ and $v = 26$. This problem comes from the symmetry of their drawing algorithm. We decide to keep it because their performances are based on a symmetrical version of the drawing.

3.1 Theoretical performance analysis

To study the efficiency of the proposed algorithms, we compare the cost of one iteration of our method with the algorithm proposed by Boyer and Bourdin. Their algorithm is mainly described by the following

code:

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

Thus, one iteration costs : 1 test, 1 affectation and 1 addition, plus the drawing of one step. In our case, the cost is the same. We must add the cost of the loop but we save one test and one affectation per loop iteration. Moreover, since we have proved the alternating sequence of steps of length q and $q+1$, changing from one length to another costs nothing. Thus in the worst case, we save the cost of all the changes in length that are scheduled by the alternative sequence. Thus in the worst case, we are as fast as Boyer and Bourdin's one with a little speedup. To quantify this speedup, we compute, with u fixed and v varying, the distribution of the repetition factors p and c. The results are summarized in figure 5 where the mean value of p and c are plotted relatively to segments lengths.

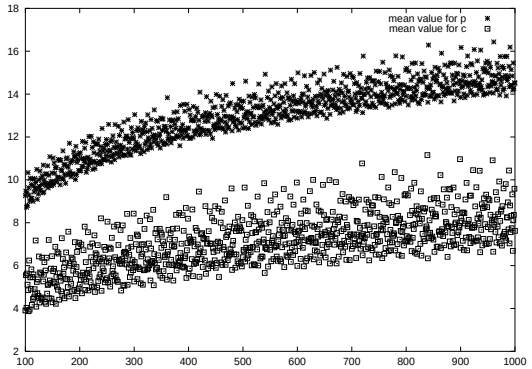

Figure 5: Mean values for p and c relatively to length of segments

In this experiments, we drawned all lines (u, v) with u the value in axis and v varying from 1 to u. On all those lines, we computed the probability of appearance of a p and a c value and then deduced the mean. We clearly see that a speedup of 9 to 14 can be achieved for p when the lengths increase. This is of course a theoretical speedup that must be confirmed by drawing experiments. In the case of c, the speedup is more limited because it varies from 4 to 7. However, the proportion of the use of c is 0.6 time the one of using p. Thus, it only decreases the speedup partially.

3.2 Experimental analysis

In this part, we compare the behaviour of the three algorithms in the context of a real implementation. For a fixed u, we drawn every lines for every v either in $[0; u/2]$ (hexadecant) or $[0; u]$ (octant). The measured time correspond to the difference of the processor clock between the beginning and the end of the execution. This method is reliable in the contrary to the unix `time` command which might not include memory accesses when they are performed by a system call. In order to make the algorithms comparable we drawn the lines in a memory array. This is important for the Bresenham algorithm because in this case, both the coordinates of the pixels and the memory addresses are incremental. Using a function to store, on a screen, a pixel in the position given by an incremental analysis is catastrophic since the job is done twice, once for computing the pixel and the second one for recomputing the address where to set the pixel. This might explains its lack of performances in the first Boyer and Bourdin comparison[2] which gave a Bresenham implementation of one mega pixels per second which is extrememly slow. In our experiment, we refer to the second experiment of Boyer and Bourdin[3] which seems far more reliable.

We propose two measurements. One call *without memory* and the second one *with memory*. The first case refer to the fact that we ran the algorithm but suppressing all the memory accesses to the buffer. In the second case, lines are effectively drawn in the buffer. Notice for the Boyer and Bourdin algorithm there is some memory management in an auxilliary buffer and that we also take this into account because this really corresponds to the cost of the algorithm.

When drawing steps, we prefer to use a simple `for` loop. This is more efficient than a `memset` operation in `C` and more under control. Thus, the cost of this operation is thus the cost of pointers arithmetic.

The time given are in second and have been reproduced several time in order to check their significance.

Let us now simply focus on the line drawing *without memory accesses* in both case of hexadecant and octant (see Fig. 6 and Fig. 7).

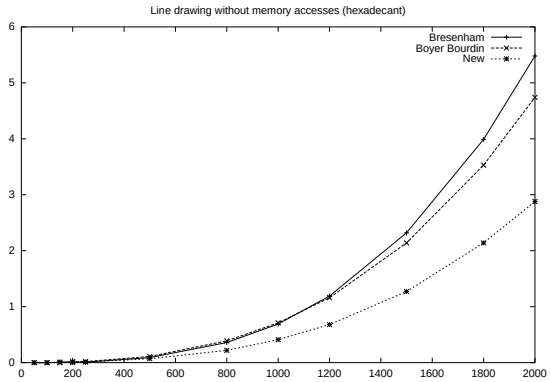

Figure 6: Hexadecant performances without memory access

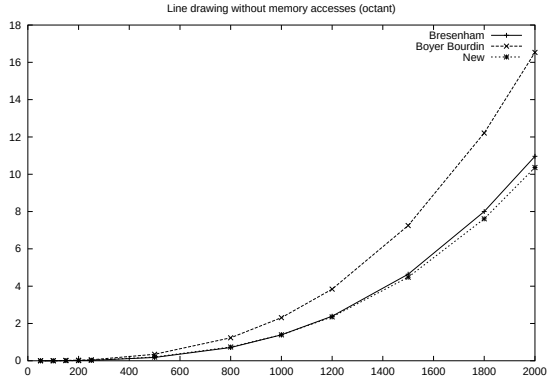

Figure 7: Octant performances without memory access

Boyer Bourdin method uses an extra buffer to draw a virtual line that will be replicated an integer number of times to draw the real line. This extra buffer kills the optimization as far as we consider only the number of operation needed to draw the line without doing the actual plotting. This overhead becomes qui significant as the number of small segments increases in the second hexadecant of the first octant (see Fig. 7.)

We now focus on the same experiments but with all memory accesses (see Fig. 8 and Fig. 9). The first result is that our new method performs best, followed by Boyer and Bourdin. Notice however that the speedup is far smaller than what was expected by a theoretical analysis. This is another step to the conclusion that at a so small level of abstraction, instructions are not all equivalent such that a single memory access can reduced all optimizations to nothing. It is also interesting to note that a simple steps by steps methods[1] called *RunLength* in the graphics performs better than Boyer and Bourdin. Theses algorithms are nearly equivalent but the last one has a high cost to the management of an internal buffer and the use of gcd calculations.

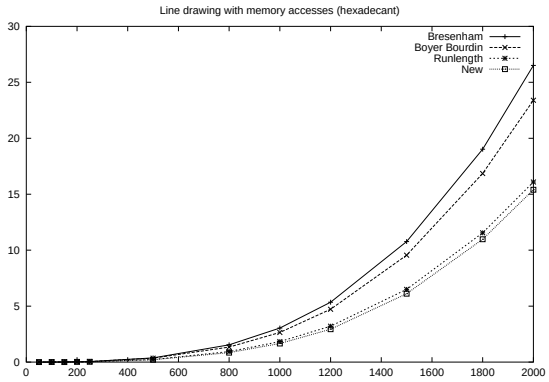

Figure 8: Hexadecant performances with memory access

In that case, we make distinct versions of our new algorithm. This comes from the remark that the distribution of q is not uniform such that we wrote a specific treatment of the case $q = 1$ and/or $q = 2$. This leads to other versions. When comparing these new versions with the original one (see Fig. 10 and Fig. 11), we can remark that a special treatment of the case $q = 1$ leads to an increase in the performances while adding the case $q = 2$ does not have a

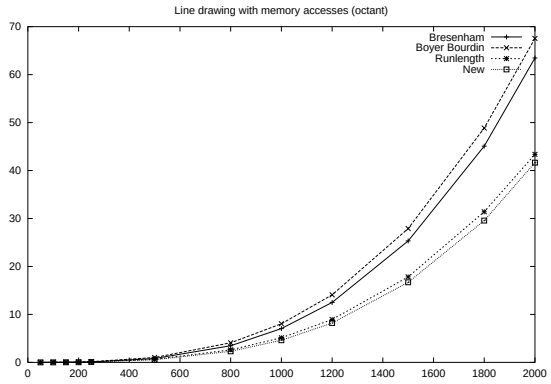

Figure 9: Octant performances with memory access

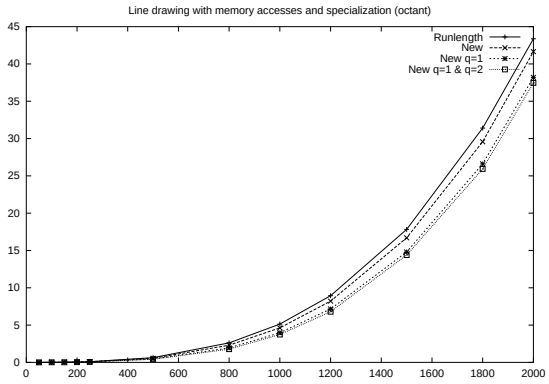

Figure 11: Octant performances with memory access and specialization

real impact. Remember that in this software implementation, the machine and the compiler both have a great influence on the performances and that in a VLSI implementation, we would not be confronted with such problems.

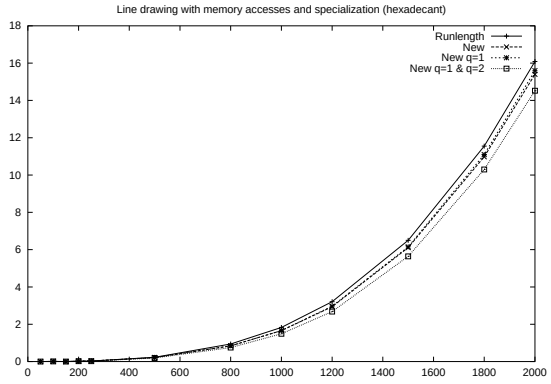

Figure 10: Hexadecant performances with memory access and specialization

In order to check this influence of the machine, we also tested the same code (compiled with the same gcc compiler) on a sparc station under Solaris 7.0. The results with memory access are given in Fig. 12. The main conclusion of that experiment is that we can reach no gain after a deep optimization since the Bresenham method performs best. Theses performances are mainly due to the memory management in the sparc architecture and a precise analysis has to be done to conclude. This is under study but necessitate an assembly simulation to have every parameters under control.

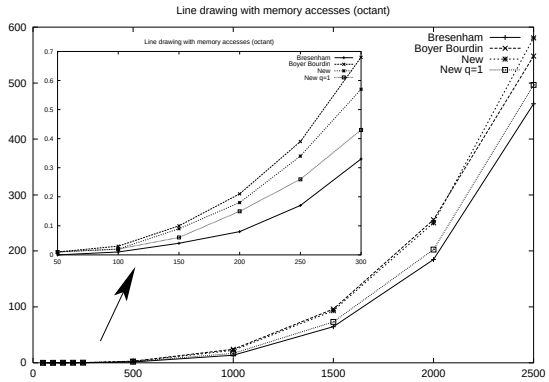

Figure 12: Octant performances with memory access on a sparc station

4 Hardware Point of View

Although the previous tests have been made on machines with powerful generic processors, either RISC or CISC, the main bottleneck does not resides in the computation but in memory accesses. We can see that nearly 75% of the time is spent for memory accesses for the new method (see Fig. 7 and Fig. 9.)

The performance gap between Bresenham method and ours also increases when memory transfers are considered. Memory accesses are made by group in our method and are more regular. However this benefit is only a cache artifact as every pixels are still drawn one after the other in our implementation as well as in Bresenham's one. The real benefit of memory access regularity can be obtained by using 2 pixels wide, or more, memory transfers.

We can see that there is a severe performance degradation when memory accesses are turned on for a given algorithm. Memory considerations can largely change the choice of the best algorithm. The Boyer Bourdin method is clearly left behind on Fig. 9 because it handles memory in a way that causes many more cache misses than the other methods (it writes the line both ways using symmetrical properties.)

VLSI Implementation and Memory Accesses

Real graphic hardware does not access video memory through caches. A chip on a graphic card has a direct access to the whole video memory. However, these accesses are still faster if they are performed in a short period of time on a contiguous memory block than at complete random. It is the same effect that can be seen on our experimentations where the performance gap between Bresenham's method and ours also increases when memory transfers are considered.

A VLSI implementation combined with a specific hardware memory interface would improve a lot further the memory transfer speed.

For instance, the specialized version with $q = 1$ made for the experiments can be extended to deal with other specific values of $q = 2, 3, 4$. Fixed values for q provide hardwired segment size within the implementation. Thus all the accesses made to draw a line are done to the memory by group of two or more pixels using the full memory interface width. The `for` loop used so far to write pixels would be transformed to a wide memory access combining q or $q+1$ pixels.

This is only feasible by using methods like Boyer Bourdin or ours and not Bresenham's one or it's Run-length variant such as the one that can be found in Abrash[1]. In such methods the length of a segment is only know at run-time and does not allow a full optimization for memory transfers. Experimental results (not shown in this paper) show that the Run-length method has an average performance curve located just above our (non specialized) new method, although we used pixel wide memory transfers.

Boyer Bourdin method is not well suited for a VLSI implementation because as it has some severe drawbacks such as the need of extra memory and more complex arithmetical operations (mainly a `gcd` computation).

5 Conclusion

In this paper we have studied and compared different drawing digital line algorithms. The algorithm we propose achieves the best performance compared to other methods and is well suited for a VLSI implementation.

It should be noted that a mixed solution combining several algorithms, each suited for a different hexadecant could be well suited for a highly tuned implementation.

The main conclusion is that, in a software implementation, the reduction in the number of instructions does not imply a reduction in the running-time. Thus, the use of an asm code is necessary to be adapted to the architecture of the machine on which the algorithm should run.

Also, a VLSI simulation is the subject of a future work in order to measure the impact of the optimizations when the memory transferts are under control.

References

[1] M. Abrash. *Zen of Graphics Programming*. Keith Weiskamp, 1995. ISBN 1–883577–08–X.

[2] V. Boyer and J.J. Bourdin. Fast lines: A span by span method. *Computer Graphics Forum (Eurographics 99 Proc)*, 18(3):C377–C384, 1999.

[3] V. Boyer and J.J. Bourdin. Auto-adaptative step straight-line algorithm. *IEEE Computer Graphics and Applications*, 20(5):67–69, 2000.

[4] J.E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, Jul 1965.

[5] R. Brons. Linguistic methods for description of a straight line on a grid. *Computer Graphics and Image Processing*, 3:48–62, 1974.

[6] C.M.A. Castle and M.L.V. Pitteway. An application of euclid's algorithm to drawing straight lines. In R.A. Earnshaw, editor, *Fundamental Algorithms for Computer Graphics*, volume 17 of *F*, pages 135–139. Springer-Verlag, Berlin, NATO Advanced Study Inst., 1985.

[7] I. Debled and J.P. Reveilles. A linear algorithm for segmentation of digital curves. In *Parallel Image Analysis: Theory and Applications*, pages 73–100. World Scientific Publishing Company, 1995.

[8] S. Dulucq and D. Goyou-Beauchamps. Sur les facteurs des suites de sturm. *Theoretical Computer Science*, (71):381–400, 1990.

[9] G. Gill. N-step incremental straight-line algorithms. *IEEE Computer Graphics and Applications*, 14(3):66–72, May 1994.

[10] P. Stephenson and B. Litow. Why step when you can run ? *IEEE Computer Graphics and Applications*, 20(6):76–84, 2000.

[11] A. Troesch. Interpretation geometrique de l'algorithme d'euclide et reconnaissance de segments. *Theoretical Computer Science*, (115):291–319, 1993.

[12] L. Wu. On the chain code of a line. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(3):347–353, 1982.