Tracés de Droites Discrètes : le Juste Pas

Vincent Boyer & Jean-Jacques Bourdin
Groupe de Recherche en Infographie et Synthèse d'images
Laboratoire d'Intelligence Artificielle - Université Paris 8
93 526 Saint-Denis - France
boyer, jj@ai.univ-paris8.fr

1 Introduction

Le tracé de droites est une primitive fondamentale des palettes graphiques. Que la droite soit affichée indépendamment ou intégrée à une figure plus complexe (triangles, rectangles, ...), le calcul de droites discrètes est omniprésent. La rapidité de l'algorithme utilisé est primordial. Les algorithmes qui permettent de réaliser ce calcul sont nombreux et une très abondante littérature existe sur le sujet. On dénombre trois approches principales :

- l'ADD (Analyse Discrète Différentielle) présentée par J. Bresenham [1],
- les algorithmes basés sur des analyses combinatoires [2, 3, 4, 5],
- les algorithmes basés sur des analyses linguistiques [6, 7].

La plus utilisée reste l'ADD qui est étendue par une nouvelle classe d'algorithmes (les algorithmes à pas-de-N, pour lesquels nous utiliserons la dénomination usuelle de "*N-step*"). C'est cette classe qui est étudiée ici. Pour chacun des algorithmes N-step [8, 9, 10], une étude théorique est réalisée. Comme tous ces algorithmes calculent la meilleure approximation de la droite réelle, l'analyse porte sur la complexité en temps (la modicité de l'espace mémoire requis ne justifie pas une étude poussée en espace). L'analyse met en évidence un gain faible, surtout si on le compare aux objectifs affichés par les auteurs. Pour mener l'entreprise d'accélération à bien, de nouvelles propriétés sont présentées. Elles permettent l'élaboration et l'écriture d'un nouvel algorithme. La suite de l'article concerne son analyse. Enfin, les performances de l'ensemble de ces algorithmes sont comparées. L'analyse comme les performances montrent que le nouvel algorithme est plus de deux fois plus rapide que les précédents.

1.1 Notations

Avant de discuter plus en détail de ces algorithmes, nous introduisons des notations qui seront utilisées dans la suite de l'article :

- L'écran est assimilé à une partie du plan discret.

– "Droite" représente un segment discret joignant deux points de l'écran. Nous nommerons ces points $P(x_p, y_p)$ et $Q(x_q, y_q)$.

– La droite est décrite par sa pente (u, v) où $u = x_q - x_p$ et $v = y_q - y_p$.

Les coordonnées de P et Q étant entières, le segment à tracer est une translation exacte du segment $(0, 0)$, (u, v) [1]. Une droite peut être décrite par l'ensemble des points du plan discret la composant ou par son premier point et une série de déplacements. Pour une pente donnée, la série de déplacements est toujours composée d'au plus deux codes différents. Dans le premier octant ces codes sont :

– X : représentant un déplacement axial

– D : représentant un déplacement en diagonale

Remarquons que, à l'instar de Gill [10], nous n'optons par pour la représentation classique des codes de Freeman [11].

1.2 Principe de l'Analyse Différentielle Discrète

La méthode ADD est utilisée dans tous les algorithmes N-step. Elle permet de discrétiser des courbes en n'effectuant que des opérations entières. Elle consiste à déterminer, itérativement, dans un ensemble restreint de choix possibles, celui qui fait partie du segment. Dans l'algorithme d'origine [1], l'itération concerne un seul point à choisir parmi deux. Dans les autres algorithmes, il s'agit d'un motif (suite de points) à choisir parmi plusieurs motifs. Le choix s'effectue dans le premier cas par un test (dans les autres par une succession de tests) comparant un critère calculé à une (des) valeur(s) pré-définie(s). Ce processus est itéré pour déterminer l'ensemble des points (motifs) qui composent la droite.

1.3 Principes communs aux analyses

Les analyses que nous présentons portent sur la complexité en temps des différents algorithmes de la classe N-step. L'analyse porte sur le nombre d'opérations réalisées lors d'une itération du processus et sur le nombre d'itérations. La complexité est donnée par le produit : nombre d'itérations $\times$ nombre d'opérations par itération. Soient N le nombre de points composant un motif, C le nombre de motifs possibles, puis, pour chaque itération, a le nombre d'additions, i le nombre d'incrémentations du critère E et t le nombre de tests.

Remarquons que parmi les tests l'un d'entre eux est effectué en une opération (test du signe de E), les autres nécessitant deux opérations (addition et test de signe).

Pour une droite donnée, le processus est itéré p fois, avec $p = u/N$. Nommons EP l'évaluation théorique des performances, en considérant que toutes les opérations élémentaires sont réalisées en une unité de temps, il sort $EP = p\,(i+t+a)$.

En pratique les valeurs i, t et a varient selon le motif à choisir. Notre étude détaille donc les meilleurs et pires cas. Un cas moyen, basé sur une très hypothétique équiprobabilité des motifs, complète l'ensemble.

2 Algorithmes N-step

Les quatre algorithmes N-step, étudiés ici, sont : l'algorithme de Bresenham [1] (N=1), du pas-de-deux [8] (N=2), du pas-de-trois [9] (N=3) et du pas-de-quatre [10] (N=4). La généralisation présentée par Gill n'est pas reprise ici compte tenu de ses mauvaises performances. Pour chacun des quatre algorithmes, l'analyse est résumée dans un tableau. La plupart de ces algorithmes ramène (par exemple grâce à des symétries) la droite à calculer dans un sous-ensemble du plan, l'octant chez Bresenham, un sous-octant dans les autres. Cet ensemble est nommé espace de travail dans la suite de l'article.

2.1 N = 1

L'algorithme de Bresenham [1] utilise la méthode ADD et peut être considéré comme de classe N-step avec $N = 1$. Dans le premier octant les motifs possibles sont X et D donc $C = 2$ (cf. figure 1). Le choix du motif à afficher est réalisé par un unique test de signe (cf. algorithme 1). Le tableau 1 présente les résultats de l'analyse de l'algorithme de Bresenham.

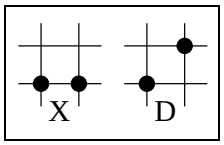

FIG. 1: N = 1 : motifs possibles

```
if (E>0) then
  motif = D ; E+=inc_D ;
else
  motif = X ; E+=inc_X ;
```

Algorithme 1: Simple pas

Cas	meilleur	pire	moyen
t	1	1	1
a	0	0	0
i	1	1	1
p	u	u	u
EP	2u	2u	2u

TAB. 1: Analyse du simple pas

2.2 N = 2

L'algorithme présenté par Rokne et al. [8] traite séparément les deux premiers hexadécants. On peut donc considérer que l'espace de travail utilisé est l'hexadécant (moitié d'un octant). Dans le premier hexadécant, les motifs possibles sont XX, XD et DX et, donc, $C = 3$ (cf. figure 2). Le motif à afficher est déterminé par deux tests (dont un de signe) au plus (cf. algorithme 2). Le pas-de-deux permet une accélération moyenne de 15% par rapport au simple pas (cf. tableau 2). On est loin de l'accélération revendiquée par les auteurs de

l'article (ils annoncent qu'en ne calculant qu'une itération tous les deux points les calculs sont divisés par deux).

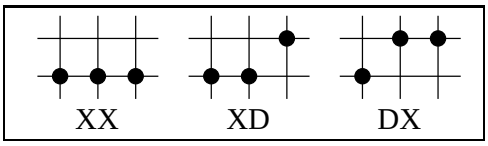

FIG. 2: N = 2 : motifs possibles

```
if (E<0)
  motif = XX ; E+=inc_XX ;
else if (E<v)
  motif = XD ; E+=inc_XD ;
else
  motif = DX ; E+=inc_DX ;
```

Algorithme 2: Pas-de-deux

Cas	meilleur	pire	moyen
t	1	2	5/3
a	0	1	2/3
i	1	1	1
p	u/2	u/2	u/2
EP	u	2u	5u/3

TAB. 2: Analyse du pas-de-deux

2.3 N = 3

Cette amélioration de la méthode précédente est présentée avec plus de modestie par ses auteurs [9]. Ils revendiquent seulement un gain de 15% par rapport à la méthode précédente. Leur méthode impose de clarifier la séparation en hexadécants. Dans le premier hexadécant, les différents motifs possibles sont XXX, XXD, XDX, DXX et DXD et $C = 5$ (cf. figure 3). La valeur de C est la même dans le second hexadécant. Le pas-de-trois permet une accélération moyenne de 20% par rapport au simple pas (cf. tableau 3).

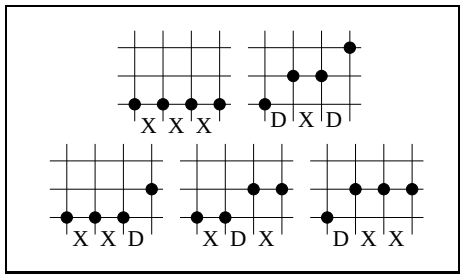

FIG. 3: N = 3 : motifs possibles

Cas	meilleur	pire	moyen
t	1	4	14/5
a	0	2	1
i	1	1	1
p	u/3	u/3	u/3
EP	2u/3	7u/3	8u/5

TAB. 3: Analyse du pas-de-trois

2.4 N = 4

Pour sa méthode, Gill [10] divise l'espace en de multiples sous-octants. Par exemple, en pas-de-quatre, il faut 6 sous-octants et pour chacun d'entre eux

nous avons $C = 5$. Le pas-de-quatre permet une accélération moyenne de 30% par rapport au simple pas (cf. tableau 4). Notons que les opérations d'initialisation sont longues : il faut trouver le sous-octant auquel correspond la droite et initialiser un nombre significatif de variables. Le tableau d'analyse n'en tient malgré tout pas compte.

Cas	meilleur	pire	moyen
t	1	4	14/5
a	0	3	9/5
i	1	1	1
p	u/4	u/4	u/4
EP	u/2	2u	7u/5

TAB. 4: Analyse du pas-de-quatre

Basés sur une même idée, augmenter la valeur de N et donc diminuer le nombre d'itérations, les algorithmes de cette classe souffrent du même défaut, le nombre C de motifs à envisager augmente avec N. De ce fait, le temps nécessaire pour choisir le motif correct augmente significativement. Le gain en temps est faible, à peine plus de 30%. Cette valeur doit être comparée aux assertions des auteurs qui l'estiment à près de 50% (cf. Rokne et al. [8]).

3 Le meilleur pas : l'auto-détermination

Notre nouvelle méthode repose sur le principe de l'auto-détermination du pas. Ainsi, au lieu de choisir un pas mal adapté à la droite à tracer, nous commençons par calculer le pas le plus adéquat. Pour cela, considérons les propriétés suivantes :

Propriété 1 *Toute droite peut être calculée dans le premier hexadécant.*

Démontrons cette propriété importante. Rappelons d'abord que, depuis J. Bresenham [1], tout le monde calcule toute droite dans le premier octant. Il faut donc maintenant trouver une relation entre une droite du deuxième hexadécant et une droite du premier. Posons d'abord deux droites symétriques de part et d'autre de la frontière entre les deux premiers hexadécants (droite de pente $(2,\ 1)$). Soient la droite $D^{(1)}$ de pente $(u,\ v)$, avec $0 \leq v < 2\,v \leq u$, et la droite $D^{(2)}$ de pente $(u,\ u - v)$. Notons également :

$$\begin{aligned} y_x^{(1)} &= \left[\frac{v\,x}{u}\right] \\ y_x^{(2)} &= \left[\frac{(u - v)\,x}{u}\right] \end{aligned}$$

où $[\alpha]$ est l'entier le plus proche de α. Ces valeurs sont celles des points de chacune des deux droites pour l'abscisse x. Nous affirmons que la seconde de

ces valeurs peut être déduite de l'autre. En effet :

$$y_x^{(1)} + y_x^{(2)} = x \quad \forall\, x \in [0, u] \tag{1}$$

Ceci se montre aisément :

$$\begin{aligned} y_x^{(1)} + y_x^{(2)} &= \left[\frac{v\,x}{u}\right] + \left[\frac{(u - v)\,x}{u}\right] \\ y_x^{(1)} + y_x^{(2)} &= \left[\frac{v\,x}{u}\right] + \left[x + \frac{-v\,x}{u}\right] \end{aligned}$$

Comme x est une valeur entière, on a donc :

$$\begin{aligned} y_x^{(1)} + y_x^{(2)} &= x + \left[\frac{v\,x}{u}\right] + \left[\frac{-v\,x}{u}\right] \text{ soit,} \\ y_x^{(1)} + y_x^{(2)} &= x \end{aligned}$$

Notons $C_x^{(i)}$ le x-ième code de la droite $D^{(i)}$. Comme ce code est une valeur d'un alphabet binaire, notons *not* la fonction qui prend le code inverse.

Propriété 2 *À chaque pas, le code de la droite $D^{(2)}$ est l'inverse du code de la droite $D^{(1)}$:*

$$C_x^{(2)} = not\,(C_x^{(1)})$$

En effet, le code de chacune des droites est donné par la relation :

$$\begin{aligned} y_{x+1}^{(i)} - y_x^{(i)} = 1 &\iff C_x^{(i)} = D \\ y_{x+1}^{(i)} - y_x^{(i)} = 0 &\iff C_x^{(i)} = X \end{aligned}$$

Grâce à l'égalité 1, nous avons aussi :

$$y_x^{(1)} + y_x^{(2)} = x$$

et

$$y_{x+1}^{(1)} + y_{x+1}^{(2)} = x + 1$$

Donc,

$$\begin{aligned} (y_{x+1}^{(1)} + y_{x+1}^{(2)}) - (y_x^{(1)} + y_x^{(2)}) &= (x + 1) - x && \text{soit} \\ y_{x+1}^{(1)} - y_x^{(1)} + y_{x+1}^{(2)} - y_x^{(2)} &= 1 && \iff \\ C_x^{(1)} = X \iff & \quad C_x^{(2)} = D && \text{et} \\ C_x^{(1)} = D \iff & \quad C_x^{(2)} = X \end{aligned}$$

Donc, pour calculer une droite du deuxième hexadécant, on prendra sa symétrique dont tous les codes seront simplement inversés. Ainsi l'espace de travail est divisé par deux. Cela permet de se focaliser pour la suite sur le premier hexadécant.

Notons X^n la chaîne formée de n répétitions de la chaîne X.

Propriété 3 *Dans le premier hexadécant, la droite est formée de suites de X séparées par un D. On peut l'écrire :*

$$X^{n_1}DX^{n_2}DX^{n_3}D...X^{n_{m-1}}DX^{n_m}$$

Propriété 4 *À l'exception des deux extrêmes, ces suites sont de la même longueur, à un près. Nommons A cette longueur et prenons $B = A + 1$:*

$$\forall j \in]n_1,\ n_m\ [, \qquad n_j \in \{A, B\} \qquad et$$
$$n_1 + n_m \in \{\ A,\ B\}$$

Ces deux propriétés sont à la base de la méthode de Castle et Pitteway [2] qui reprend l'algorithme d'Euclide pour tracer la droite.

Dans l'algorithme, les valeurs n_1 et n_m sont calculées séparément. Dans le premier hexadécant, nous avons $A = \lfloor (u - v)/v \rfloor$ (c'est-à-dire la partie entière de la valeur). À chaque itération nous devons choisir entre deux motifs X^A et X^B (donc nous avons : $C = 2$). La figure 4 présente un exemple avec $A = 4$ et $B = 5$. Un simple test de signe permet de choisir parmi ces deux possibilités (cf. algorithme 3). Le tableau 5 présente l'analyse du pas auto-déterminé. Remarquons que p le nombre d'itérations est donné par le nombre v de pas obliques. Or dans le premier hexadécant $v < u/2$. En conséquence, le gain en temps par rapport au simple pas est d'au moins 50%.

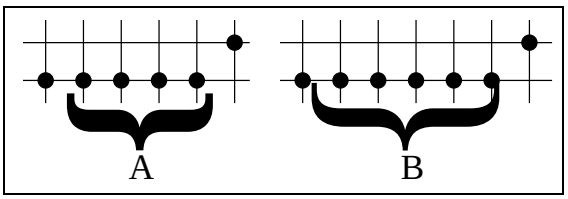

FIG. 4: Exemple de deux motifs

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

Algorithme 3: Pas auto-déterminé

Cas	meilleur	pire	moyen
t	1	1	1
a	0	0	0
i	1	1	1
p	v	v	v
EP	2v	2v	2v

TAB. 5: Analyse du pas auto-déterminé

4 Performances

Pour valider ces résultats, nous avons procédé à des tests sur machine. Pour cela nous avons implémenté les cinq algorithmes, ADD, pas-de-deux, pas-de-trois, pas-de-quatre et pas-auto-déterminé en langage **C**. Toutes les droites

partant de l'origine et comprises dans le triangle ($(0, 0)$, $(N, 0)$, $(N, N/2)$) sont calculées. Le champ "temps total" du programme *profile* de l'exécution est donné (cf. tableau 6). Ces tests ont été effectués sur une station de type Compaq Professional Workstation XP1000 (processeur : alpha 21264 (500 MHz)). La dernière ligne du tableau présente le ratio entre l'algorithme testé et celui de Bresenham. Bien sûr ces résultats expérimentaux dépendent beaucoup de facteurs extérieurs : qualité du compilateur, non-rupture de la séquence d'exécution,... Ceci étant, la convergence entre les résultats théoriques et pratiques est indéniable.

N	1	2	3	4	auto-déterminé
500	0.2559	0.2256	0.1973	0.2031	0.0742
1000	2.0088	1.7793	1.6445	1.5030	0.5605
2000	16.1641	13.8379	12.9570	11.42	4.3555
4000	130.3613	110.7432	100.46	87.47	34.51
%	100	84.95	77.06	67.00	26.47

TAB. 6: Performances

5 Conclusion

Nous avons réalisé un algorithme avec un pas auto-déterminé. Cet algorithme est une amélioration de la méthode ADD de Bresenham et des algorithmes de la classe N-step. Ne fixant plus le pas arbitrairement, mais le calculant en fonction de la pente, nous avons diminué le nombre de cas à traiter. Comme Bresenham, nous choisissons toujours le motif parmi deux et uniquement deux motifs possibles ce qui diminue les opérations à réaliser dans le processus. De plus, comme les autres algorithmes, nous augmentons la taille du pas ce qui a pour effet de diminuer le nombre d'itérations nécessaires. Qu'il s'agisse d'évaluation analytique ou de tests sur machine, cette nouvelle méthode est significativement plus efficace que les précédentes.

Références

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

[2] C.M.A. Castle and M.L.V. Pitteway. An Application of Euclid's Algorithm to Drawing Straight Lines. In *Fundamental Algorithms in Computer Graphics*, pages 135–139. Springer-Verlag, 1985.

[3] J. Berstel. *Mots, Mélanges offerts à MP. Schützenberger*, chapter Tracés de droites, fractions continues et morphismes itérés. Hermès, 1990.

[4] J.P. Reveillès. Droites discrètes et fractions continues. Technical Report R90/01, ULP Département d'Informatique, Strasbourg, France, Janvier 1990.

[5] A. Troesch. Interprétation géométrique de l'algorithme d'Euclide et reconnaissance de segments. *Theoretical Computer Science*, 115 :291–319, 1993.

[6] R. Brons. Linguistic methods for description of a straight line on a grid. *CG&IP*, 3 :48–62, 1974.

[7] L. Wu. On the Chain Code of a Line. *IEEE Transactions on Patterns analysis and machine intelligence*, PAMI-4(3) :347–353, 1982.

[8] J.G. Rokne, B. Wyvill, and X. Wu. Fast line scan-conversion. *ACM TOG*, 9(4) :376–388, October 1990.

[9] P. Graham and S.S. Iyengar. Double and triple step incremental generation of lines. In *IEEE CG&A*, pages 49–53, May 1994.

[10] G. Gill. N-Step Incremental Straight-Line Algorithms. *IEEE CG&A*, pages 66–72, May 1994.

[11] H. Freeman. Boundary Encoding and Processing. In *Picture Processing and Psychopictorics*, pages 241–266. B.S. Lipkin and A. Rosenfeld, Eds, New-York : Academic, 1970.