Une accélération du tracé de droites

Vincent BOYER
Jamel TAYEB
Jean-Jacques BOURDIN

Groupe de Recherche en Infographie et Synthèse d'images
Département Informatique
Université PARIS 8
2, rue de la liberté
93256 Saint-Denis Cedex 2
Tel : (33) 01 49 40 64 00
Fax : (33) 01 49 40 67 83
[boyer-jamel-jj]@ai.univ-paris8.fr

Résumé

Le tracé d'une droite en synthèse d'images est le plus souvent réalisé à l'aide d'une méthode basée sur l'Analyse Discrète Différentielle (DDA) comme l'algorithme de Bresenham [Bresenham 1965]. Des méthodes plus récentes améliorent la rapidité de cet algorithme. Nous adoptons l'amélioration permettant de réduire le segment à son plus petit motif (la plage). Dans un deuxième temps, une symétrie est recherchée et démontrée permettant de ne plus calculer que la moitié du segment. Enfin ces deux améliorations sont appliquées au tracé par plage. Un algorithme est écrit et testé pour ses qualités en temps.

Mots clé :[Informatique graphique] : Synthèse d'Images, Approximation de Droites, Algorithmes.

1 - Introduction

En synthèse d'images, le tracé de droites est la primitive la plus utilisée. Le tracé consiste en l'éclairage des points représentant un chemin linéaire joignant deux extrémités données. Par abus de langage, le terme de "droites" est utilisé pour parler de cette suite discrète de points pris sur un maillage orthogonal représentant l'écran graphique. De nombreuses méthodes sont connues qui permettent de réaliser ce tracé, mais la plupart d'entre elles reprennent l'algorithme de Bresenham [Bresenham 1965] et tentent de l'améliorer [Gardner 1975], [Rockne 1990], [Angel 1991], [Kuzmin 1995].

Seules quelques méthodes issues des mathématiques discrètes et de la combinatoire [Berstel 1990], [Reveilles 1990], [Troesch 1993], ou, au moins, utilisant des propriétés particulières des nombres [Pitteway 1982], [Castel 1985], [Pitteway 1985] ouvrent des perspectives différentes. Ces méthodes associent le segment à un mot de trace basé sur le code de Freeman [Freeman 1974] des connections entre deux points consécutifs. Le tracé de droites est alors ramené à un calcul de mot. Mais ces méthodes se heurtent à la lenteur des recopies des chaînes de caractères qui en grèvent lourdement les performances, les rendant moins efficaces que l'algorithme de Bresenham.

Enfin une dernière approche consiste à répartir le tracé par groupe de points de même ordonnée (ou abscisse) que nous nommons des plages [Bresenham 1985], [Chalopin 1996]. Dans cette approche, nous ne travaillons plus point par point, mais par plage, ce qui réduit considérablement les temps de calcul. C'est cette perspective que nous adoptons. En effet le travail par plage est plus efficace et plus rapide, mais en plus il est bien adapté aux ordinateurs actuels qui comprennent des primitives d'écriture de segments horizontaux ou verticaux.

Nous améliorons les méthodes précédentes en présentant deux modifications :

- Utilisation de la « pente réduite » permettant de travailler sur un segment plus court qu'il s'agira de répéter.
- Démonstration d'une symétrie interne au segment permettant de ne calculer que la première moitié du segment réduit.

2 - Notations

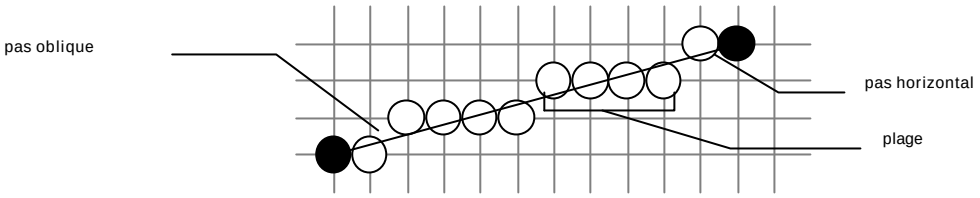

Figure 1 - Visualisation du mot de la droite de pente (11, 3).

La droite (le segment de droite) est définie par deux points initiaux P(x_p, y_p) et Q(x_q, y_q). Soient u et v les différences d'abscisses et d'ordonnées.

$$u = x_q - x_p, v = y_q - y_p$$

On note v/u la pente de la droite.

Remarquons que les symétries les plus usuelles (par rapport aux axes et aux diagonales principales) permettent de travailler dans l'octant de son choix. De plus, comme les coordonnées des points sont entières, le segment tracé est une translation sans approximation du segment ((0, 0), (u, v)). On peut alors remarquer que la droite se ramène à l'équation :

$$y = \frac{v\,x}{u}$$

On peut prendre le segment en coordonnées entières en choisissant soit la **meilleure approximation**, soit la **partie entière** (utilisée par la suite et notée $\lfloor\ \rfloor$) soit **l'entier supérieur** à la valeur réelle de y.

Nous choisissons de travailler dans l'octant $0 \le v \le u$. Les points successifs du segment de droite étant connexes, dans cet octant, ils sont séparés soit par un pas oblique (x et y croissent de 1) soit par un pas horizontal (seul x croit de 1). On peut symboliser alors un segment de droite par la suite des évolutions de y. Le mot 01000100010 représente la meilleure approximation du segment de pente 3/11. Avec l'approximation à l'entier inférieur nous aurions obtenu : 00010001001. Dans tous les cas nous pouvons considérer que les mots sont constitués de plages (à lettres identiques) séparées par une lettre différente [Chalopin 1996]. Il existe deux types de plages (plutôt horizontales H, plutôt oblique O). Dans tout l'hexadécant $0 \le v \le 2v \le u$, ce sont des plages de 0 séparées par des 1. Nous ne travaillerons plus que dans cet héxadécant et nous travaillerons sur la partie entière.

3 - Utilisation du PGCD

On remarque que le segment de pente $k\,v'/k\,u'$ avec $k \in N$ est k fois la duplication du segment de pente v'/u'. De même on peut réduire le mot à calculer à son motif le plus petit qui sera réecrit pgcd (u, v) fois. Par exemple la droite 1024, 256 est équivalente à 256 fois la droite 4,1.

Nous appelons pente réduite [Berstel 1990], la pente v'/u' si :

$$u = u'k$$
$$v = v'k$$
$$k = \gcd(u,v)$$

Le mot calculé pour la pente réduite sera dupliqué k fois pour obtenir le mot de la pente v/u

4 - Placement symétrique des lettres

Désormais, nous considérons que v/u est une pente réduite (i.e u et v sont premiers entre eux). Ainsi si nous utilisons l'algorithme de tracé de Bresenham [Bresenham 1965] adapté pour l'approximation à la partie entière, la droite (0, 0) - (11, 3) nous donne les valeurs suivantes : u = 10 et v = 3.

x	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***
y	***0***	***0***	***0***	***0***	***1***	***1***	***1***	***1***	***2***	***2***	***2***	***3***
delta	-16	-9	-3	3	-13	-7	-1	5	-11	-5	1	inutile
lettre	0	0	0	1	0	0	0	1	0	0	1	inutile

Figure 2 - Résultat de l'exécution de la fonction Dessin

Nous constatons sur cet exemple l'existence d'une symétrie par rapport au centre sauf aux extrémités. En effectuant le calcul et le tracé de la première moitié du segment, nous pouvons sans le calculer tracer la seconde p artie du segment. Remarquons que seule la première lettre constitue une exception propre au calcul de la droite inférieure. Notre algorithme est donc simplifié et devient :

```
void DessinSymetrique(int u, int v) {
	int increment_oblique, increment_horizontal, delta ;
	increment_oblique=2*v - 2*u ; increment_horizontal=2*v ;
	delta=2*v - 2*u ; afficher(0) ;
	afficherfin(1) ;
	u=(u-1)/2 ;                                    /*opération supplémentaire*/
	for (x=0; x<u; x++) {
		if(delta>=0) {
			afficher(1) ; afficherfin(1) ;
			delta+=increment_oblique ;
		} else {
			afficher(0) ; afficherfin(0) ;
			delta+=increment_horizontal ;
		}
	}
}
```

Il n'y a aucun ajout d'opération par rapport à la méthode de Bresenham si ce n'est une division par 2 (décalage) à l'initialisation. Cette méthode permet de diviser par 2 le nombre de tests et d'opérations à effectuer dans la boucle.

4 - La démonstration

Dans ce paragraphe, nous démontrerons que les lettres sont disposées de façon symétrique.
Propriété : lettre (x) = lettre (u - x).
Nous allons démontrer cette propriété en deux étapes :

- D'abord trouver une condition nécéssaire et suffisante à l'existance d'un pas oblique de part et d'autre de l'axe de symétrie
- Ensuite démontrer que ces deux conditions sont équivalentes

Nous avons par définition les relations suivantes :

$d(x,y) = vx - uy$

$y = \left\lfloor \frac{vx}{u} \right\rfloor$

$d(x,y) = vx \bmod u$

(mod = reste de la division entière de vx par u)

Donc $0 \le d(x,y) < u$

Remarque : $\forall x, \exists ! y \ / \ 0 \le d(x,y) < u$

Calculons la valeur symétrique de $d(x,y)$

$d(u-x,y') = v(u-x) - uy'$ où $y' = \left\lfloor \frac{v(u-x)}{u} \right\rfloor$

$d(u-x,y') = vu - vx - uv - u\left\lfloor \frac{-vx}{u} \right\rfloor$

$= -vx - u\left\lfloor \frac{-vx}{u} \right\rfloor$

Etudions les cas où y va varier d'abord au début du segment, ensuite dans sa dernière partie :

- Si $d(x,y) \ge u - v$ alors puisque $d(x+1,y) = d(x,y) + v$, nous obtenons $d(x+1,y) \ge u$, donc y croit de 1.
  Par conséquent si $d(x,y) \ge u - v$ alors la lettre associée est 1.
  Réciproquement :
  (1) : $0 \le d(x+1,y+1) < u$
  comme : $d(x+1,y+1) = d(x,y) + v - u$
  (1) s'écrit $0 \le d(x,y) + v - u < u$
  nous obtenons $d(x,y) \ge u - v$

- Si $0 \le d(u-x,y') < v$ alors puisque $d(u-x-1,y'-1) = d(u-x,y') - v + u$,
  nous avons : $0 \le u - v \le d(u-x-1,y'-1) < u$
  Pour u - x - 1, le seul y possible est y' - 1.
  Dons si x croit et si $d(u-x,y') < v$ alors le pas précédent est 1

Démontrons que $d(x,y) \ge u - v$ correspond à $d(u-x,y') \le v$.

Pour cette démonstration, nous aurons besoin d'étudier la somme de deux deltas symétriques soit : $d(x,y) + d(u-x,y')$

$$d(x,y) + d(u-x,y') = vx - u\left\lfloor \frac{vx}{u} \right\rfloor - vx - u\left\lfloor \frac{-vx}{u} \right\rfloor$$

$$= -u\left(\left\lfloor \frac{vx}{u} \right\rfloor + \left\lfloor \frac{-vx}{u} \right\rfloor \right)$$

Deux cas sont possibles en fonction de la valeur de $b = vx \bmod u$.

- si $b = 0$ alors $d(x, y) + d(u - x, y') = 0$ car $\left\lfloor \frac{vx}{u} \right\rfloor = \frac{vx}{u}$

  si $b \neq 0$ alors $\left\lfloor \frac{-vx}{u} \right\rfloor = -\left\lceil \frac{vx}{u} \right\rceil = -\left\lfloor \frac{vx}{u} \right\rfloor + 1$

$$d' où : d(x, y) + d(u - x, y') = u\left(\left\lfloor \frac{vx}{u} \right\rfloor - \left\lfloor \frac{vx}{u} \right\rfloor + 1\right)$$

$$= u$$

La somme de deux deltas symétriques est soit égale à u si $b \neq 0$, soit égale à 0 si $b = 0$.

Revenons maintenant au calcul :

Deux cas sont possibles en fonction de la valeur de b (modulo de vx et u).

- Si $b \neq 0$ alors nous savons que $d(x, y) + d(u - x, y') = u$
  - $d(x, y) \geq u - v$

    $\Leftrightarrow d(u - x, y') + d(x, y) \geq d(u - x, y') + u - v$

    $\Leftrightarrow u \geq d(u - x, y') + u - v$

    $\Leftrightarrow d(u - x, y') \leq v$

    - si $b = 0$, comme $b = d(x, y)$ alors on a $d(x, y) = 0$

      comme u et v et $x \leq u$ sont premiers entre eux, c'esdt possible pour $x = y = 0$ et $x = u, y = v$, cas particulier sans symétrie.

Nous avons démontré que les pas sont disposés symétriquement. De ce fait à un pas horizontal correspondra en symétrie un pas horizontal, et à un pas oblique correspondra en symétrie un pas oblique. Seule la première lettre à tracer aura un symétrique différent car nous sommes dans le cas où x = y = 0(à ce 0 correspondra un 1).

L'utilisation de la pente réduite avec recopie de motif [Angel1991], du calcul de droites par plages [Bresenham 1985] [Chalopin 1996], de la symétrie et du double pas [Rockne 1990] permettent d'améliorer significativement les temps de calcul de droites.

5 - Tests et Performances

Pour comparer l'efficacité des différents algorithmes, nous avons effectué un ensemble de tests sur une station Silicon Graphics Elan.

Protocole utilisé pour les tests : nous calculons toutes les droites partant de l'origine et décrivant la surface du triangle rectangle ((0,0),(x,0),(x,x/2)). Le temps indiqué dans le tableau qui suit correspond à la valeur « user » de la commande Unix « time ». Ces tests effectués n'incluent aucunement le tracé correspondant au mot formé, seul les temps de calcul de mots sont donnés.

Bresenham	Algorithme de Bresenham
ChaBou	Algorithme de Chalopin & Bourdin avec double pas et pgcd
JaBo	Nouvel algorithme avec double pas et pgcd

Figure 3 - Contenu des algorithmes évalués.

x	***Bresenham***	***ChaBou***	***JaBo***

500	16,46	2,52	1,81
1000	131,69	16,14	10,46
1500	443,99	49,56	30,23
2000	1052,99	111,47	65,76
2500	2055,82	210,45	121,27
3000	3560,57	355,16	200,54
4000	8433,12	815,65	451,01
6000	28458,15	2664,76	1432,00
10000	131922,93	11971,24	6278,75

Figure 4 - Les durées de calcul sont données en secondes.

Les temps entre les divers algorithmes nous donnent une accélération d'un ordre 2 par rapport à Chalopin Bourdin et 20 par rapport à Bresenham.

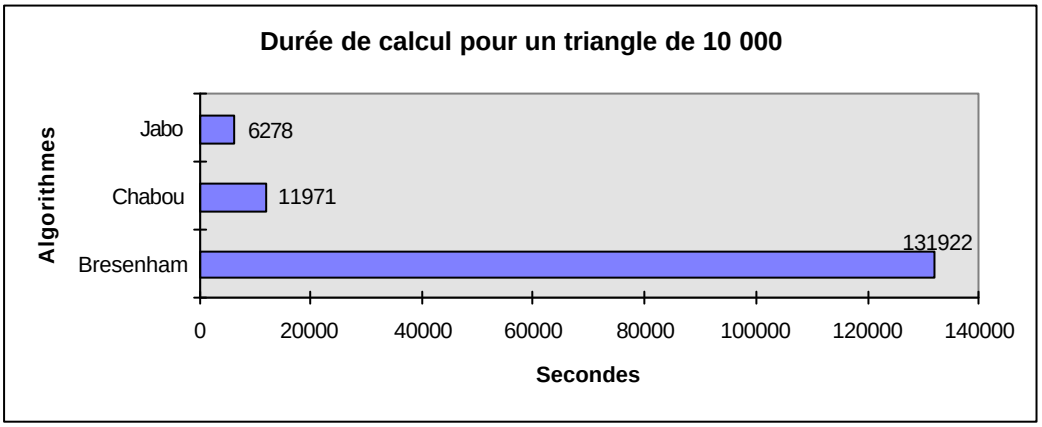

Figure 5 - Performances comparatives des algorithmes.

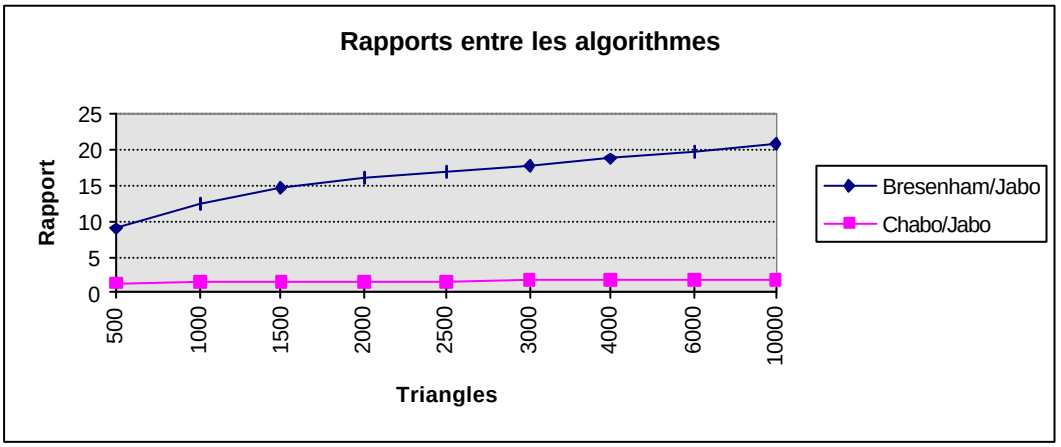

Figure 6 - Rapports de performances entre notre algorithme et les méthodes existantes.

L'augmentation du rapport de vitesse de tracé entre l'algorithme de Bresenham et JaBo en notre faveur, s'explique par le recours au pgcd qui conduit au calcul d'un mot dont la longueur varie non pas seulement en fonction de u, mais aussi de v. Voilà pourquoi la variation du rapport Bresenham / JaBo est non linéaire.

6 - Conclusion

Nous avons mis en évidence et démontré une nouvelle symétrie permettant d'améliorer les performences du tracé de droites dans un espace discret bi-dimentionnel. Cette amélioration semble ne pas être limitée à la bi-dimentionnalité, mais peut être transposée au tracé de droites dans un espace tri-dimentionnel.
Dans le futur, nous comptons tester notre méthode en affichant les droites. Les machines actuelles permettant l'affichage par adressage des plages horizontales et verticales, nous espérons conserver une accélération importante par rapport à l'algorithme de Bresenham. Remarquons que l'utilisation de machine avec adressage des diagonales automatiques permettrait d'augmenter le gain. Néanmoins, le bilan est déjà très positif car l'intégration des différentes méthodes nous donne une accélération très significative.

7 - Bibliographie

1. [ANGEL 1991] : ANGEL E & MORRISON D. *Speeding up Bresenham's algorithm*, IEEE Computer Graphics & Applications, Vol 11 pp 16-17(Novembre 1991).

2. [BERSTEL 1990] : BERSTEL J. *Tracé de droites, fractions continues et morphismes itérés*, M. Lothaire, *Mots*, Mélanges offerts à M.P. SCHÜTZENBZEREGER, Hermes (1990).

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

4. [BRESENHAM 1982] : BRESENHAM J.E. *Incremental line Compaction*. The Computer Journal (fevrier 1982), Vol. 2, N°1.

5. [BRONS 1974] : BRONS R. *Linguistic Methods for the description of a straight line on a gird*. Computer Graphics and Image Processing (1974) Vol 3, N°1, pp 48-62.

6. [CASTLE 1985] : CASTLE C.M.A. *An application of Euclid's algorithm to drawing straigth lines*. Fundamental Algorithms in Computer Graphics, Springer-Verlag (1985), pp 135-139.

7. [CHALOPIN 1996] : CHALOPIN F.P & BOURDIN J.J. *Drawing straight lines Faster*. Cours de maitrise de Paris8 (1996), pp 150-158.

8. [DULUC 1990] : DULUC S & GOYOU-BEAUCHAMPS D. *Sur les facteurs des suites de Sturm*. Theorical Computer Science 71 (1990), pp 381-400.

9. [FREEMAN 1970] : FREEMAN H. *Boundary encoding and processing in Picture and Processing and Psychopictrorics*. LIPKIN B.S. & ROSENFELD A, Academie Press, New-York (1970), pp 241-266.

10. [FOLEY 1990] : FOLEY J.D, VAN DAM A, FEINER S, HUGHES J. *Computer Graphics, Principes and Practices, second edition*. Addison Wesley (1990) .

11. [GARDNER 1975] : GARDNER P.L. *Gardner Modifications of Bresenham's algorithm for display*, IBM Tech. Disclosure Bull.18 (1975), pp 1595-1596.

12. [PITTEWAY 1985] : PITTEWAY M.L.V. *Euclid's algorithm and lines drawing*. Fundamental Algorithms in Computer Graphics, Springer-Verlag (1985), pp 101-106.

13. [REVEILLES 1990] : REVEILLES J.P. *Droites discrètes et fractions continues*. ULP département d'informatique (janvier 1990), R90/01.

14. [ROKNE 1990] : ROCKNE J.G & WYVILL B. *Fast line scan-conversion*. ACM Transactions on Graphics, Vol 9, N°4 (Octobre 1990), pp 376-388.

15. [TROESCH 1993] : TROESCH A. Interprétation géométrique de l'algorithme d'Euclide et reconnaissance de segments. Theoretical Computer Science 115 (1993), pp 291-319.