Accélération du Calcul de Droites 3D Discrètes

Vincent Boyer & Jean-Jacques Bourdin

Groupe de Recherche en Infographie et Synthèse d'images
Laboratoire d'Intelligence Artificielle
Université Paris 8
2, rue de la Liberté
93 526 Saint-Denis Cedex 02
France
boyer, jj@ai.univ-paris8.fr

Résumé : *Une modélisation courante de l'espace tridimensionnel est de le considérer comme un ensemble de voxels uniformes. Un objet est alors défini à l'intérieur d'un ensemble englobant de voxels. L'illumination de la scène passe par de longs calculs d'intersection entre des droites et les objets présents dans la scène. Nous proposons une méthode originale pour déterminer l'ensemble des voxels traversés par une droite. Cette méthode est ensuite comparée à celle de Kaufman.*

Mots-clés : Droite discrète, lancer de rayons, voxels, Informatique Graphique, Synthèse d'Images

1 Introduction

Conformément à Metge et Caubet [MC94] nous représentons l'espace tridimensionnel par des cubes élémentaires (voxels) uniformes. Un objet est alors contenu dans un volume englobant composé de voxels. Les méthodes d'illumination imposent de parcourir des droites. Il s'agit des rayons dans le lancer de rayons [FDFH90]. Ce sont les droites joignant les objets deux à deux pour la radiosité [Whi80]. Dans les deux cas l'intersection entre chaque droite et chaque voxel doit être détectée.

Déterminer rapidement l'intersection entre un rayon et un objet revêt une importance particulière. Whitted [Whi80] estime que le temps de calcul de cette opération représente 90% du temps total de calcul de la scène. Généralement les algorithmes utilisés s'inspirent de méthodes élaborées pour le calcul de droites discrètes 2D. Ainsi Stolte et Caubet [SC95], Cohen et Kaufman [CKBB90, CK91, Kau88] transposent la méthode DDA (Analyse Discrète Différentielle) présentée par Bresenham [Bre65] en 1965.

Des algorithmes plus récents ont amélioré cette méthode. C'est le cas de Rockne [RWW90] qui calcule la droite par «pas de deux», de Graham [GI94] qui en est une extension en «pas de trois» et de Angel [AM91] qui répète le motif obtenu avec la pente réduite par son pgcd. De nouvelles voies ont été explorées : Pitteway [Pit85], Castle [CP85], Dulucq [DGB90] et Reveilles [Rev90] utilisent une approche combinatoire. Bresenham lui–même [Bre85] apporte des modifications en effectuant un seul calcul pour déterminer tous les points de même ordonnée. Berstel réduit le problème à l'étude de fractions continues [Ber90].

Dernière avancée dans ce domaine, l'algorithme dû à Boyer et al. [BTB97] tire parti de ces différentes améliorations. Il est fondé sur le placement symétrique, par une méthode DDA, de plages calculées par une méthode combinatoire (voir plus loin). En réduisant l'espace de travail à l'hexadecant (symétrie du plan), il raccourcit considérablement les temps de calcul. Il est 21 fois plus rapide que celui de Bresenham [Bre65]. Ces performances représentent le gain en temps de calcul. Hélas en 2D, le temps d'affichage grève considérablement ces améliorations. En 3D, par contre, on n'a pas à afficher chacun des voxels parcourus. On doit, simplement, le détecter pour analyser les intersections éventuelles. Lorsqu'une intersection est détectée, elle impose, au lancer de rayons, une étude de ce que devient le rayon et, en radiosité, l'absence d'interaction entre les deux éléments de surface que l'on tentait de joindre. Les améliorations

venue de l'algorithmique 2D prennent donc un poids considérable en 3D. Nous présentons une adaptation à la synthèse d'images 3D des améliorations.

2 Problématique

Dans un espace discret composé de voxels uniformes, nous devons connaître, le plus rapidement possible, tous les voxels traversés par un rayon. Le voxel est le plus petit élément indivisible de l'espace tridimensionnel discret qui puisse être adressé [Udu91]. C'est le pendant du pixel dans une image à deux dimensions. Le voxel est assimilable à un cube d'arête unitaire. Il a donc six voisins immédiats. Nous pouvons également définir deux voisinages supplémentaires qui sont les 18 et 26 connexités.

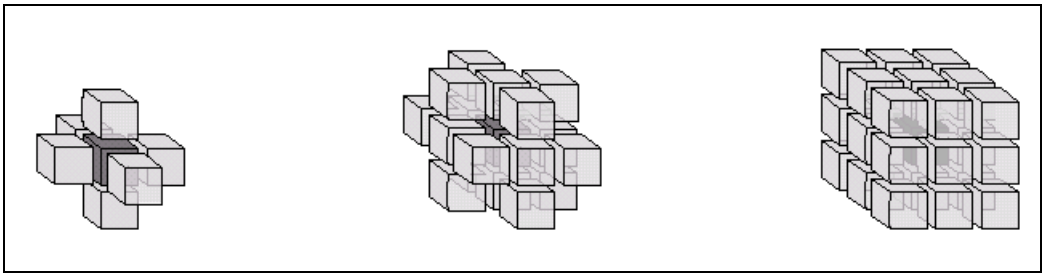

FIG. 1: Representation des voxels voisins du voxel central - en gris foncé. A gauche, les 6 voisins, au centre 18 voisins et à droite 26 voisins

3 Conventions

Nous utilisons la simplification courante de «droite» pour parler du segment discret joignant deux points. Notons $P(x_p, y_p, z_p)$ et $Q(x_q, y_q, z_q)$ les extrémités de la droite. Les coordonnées des points étant entières, le segment à calculer est une translation sans approximation de la droite $((0,0,0)(u,v,t))$, avec :

$$\begin{cases} u = x_q - x_p, \\ v = y_q - y_p, \\ t = z_q - z_p. \end{cases}$$

Afin de simplifier le problème et les calculs, nous restreignons le domaine de travail. Notre domaine est délimité par :

$$u \geq 2v \geq v \geq t \geq 0.$$

Toutes les autres parties de l'espace peuvent se déduire de celle–ci grâce à des translations ou des symétries. On appelle «déplacement élémentaire», le passage d'un voxel (V) à un voisin (V'). u, v, t étant des entiers positifs, comme $u \geq 2v$, les seuls déplacements élémentaires utilisés seront :

0 :	horizontal :	Si $V(x_v, y_v, z_v)$	alors	$V'(x_v + 1, y_v, z_v)$.
1 :	oblique :	Si $V(x_v, y_v, z_v)$	alors	$V'(x_v + 1, y_v + 1, z_v)$.
2 :	vertical :	Si $V(x_v, y_v, z_v)$	alors	$V'(x_v, y_v, z_v + 1)$.

En effet, nous considérons les mouvements verticaux comme devant être effectués à part des mouvements horizontaux. La 8–connexité est utilisée pour les mouvements effectués dans le plan horizontal. Une 2–connexité est employée pour les mouvements verticaux. En combinant les mouvements effectués dans ces deux plans, nous obtenons la 26–connexité. Définissons «l'-exposant du déplacement» comme étant le nombre de répétitions du déplacement. Par exemple

$0^2 = 00$. Nous appelons «chemin» la suite de déplacements élémentaires. La droite discrète 3D est le chemin le plus proche de la droite réelle joignant deux points qui la définissent.

4 Méthodes existantes

La méthode DDA de Bresenham [Bre65] est fondée sur le calcul d'un écart (δ) pour chaque point. Cet écart mesure la différence d'ordonnées entre un point de la droite réelle et un point du plan discret. Chercher, dans l'espace discret, le point le plus proche d'un point de la droite réelle revient à minimiser cet écart. La méthode DDA permet, par un calcul entier, de choisir parmi deux points du plan discret le plus proche de la droite réelle. Cette opération est répétée pour chaque point de la droite. Un système d'incréments (i_o et i_h) permet de calculer l'écart suivant.

Cette méthode effectue un suivi point par point. Rockne et al. [RWW90] modifient la méthode DDA en effectuant un suivi par paire de points. Le calcul d'un écart permet le plus souvent l'affichage de deux points. Le nombre d'opérations nécessaires au calcul est ainsi réduit. L'utilisation du double pas permet un gain en temps de calcul allant jusqu'à 40%.

Les méthodes combinatoires [Pit85] assimilent les «déplacements élémentaires» à des lettres. Un chemin peut être représenté par une suite de lettres appelée «mot». Pour une droite donnée d'équation :

$$y = \frac{vx}{u}$$

la façon d'approximer y construit trois mots différents :

mot inférieur :	valeur approchée à l'entier inférieur de y
mot juste :	valeur la plus proche de y
mot supérieur :	valeur approchée à l'entier supérieur de y

Angel [AM91] définit la notion de «pente réduite». v/u est une pente réduite si u et v sont premiers entre eux. Le mot de la pente réduite est appelé motif. Le mot de pente kv/ku est k fois le mot de pente v/u. Le calcul ne se fait plus sur toute la longueur du mot mais uniquement sur son motif. Lors de l'affichage, ce dernier sera dupliqué k fois ($k = \mathrm{PGCD}(u, v)$). Ce calcul est utile dans 40% des cas. Malheureusement le temps de calcul du PGCD de 2 nombres est important. Le gain en temps de calcul est d'environ 20%.

Bresenham en 1985 [Bre85] propose une méthode déterminant tous les points de même ordonnée. Le terme de «plages» est introduit par Bourdin [CB96] pour définir ce regroupement. Une plage est une suite de lettres consécutives identiques. Les plages sont séparées par un caractère différent (l'autre caractère de notre alphabet). Dans notre domaine de travail, les plages seront constituées de 0 et séparées par un 1. Nous considérons maintenant que le caractère séparateur, ici 1, fait partie de la plage. Chaque mot comporte u lettres. Chaque plage étant composée d'un 1, un mot est composé de v plages. La taille moyenne de chaque plage est donc u/v. Cette valeur n'est pas un entier si la pente est réduite. Elle est encadrée par deux valeurs entières. Il existe donc deux types de plages :

O	plage oblique :	sa taille sera la valeur approchée à l'entier inférieur de v/u
H	plage horizontale :	sa taille sera la valeur approchée à l'entier supérieur de v/u

Le mot juste de la droite est une suite de plages encadrée par des lettres isolées. Par contre, le mot inférieur n'est formé que de plages entières. Les plages sont disposées de façon homogène. On peut les répartir par une méthode DDA (calcul d'un écart δ, d'incréments i_h, i_o).

Enfin, une symétrie au milieu du mot a été démontrée [BTB97]. Cette symétrie exclut les lettres extrêmes prédéterminées. Grâce à cette propriété, seule une moitié de la droite est calculée. Les plages étant formées de lettres, on peut étendre cette propriété au calcul de mot par plages.

5 Adaptation du calcul 2D à la 3D

Nous présentons ici l'adaptation de l'algorithme de Boyer et al. [BTB97] à la 3D. Nous considérons que v/u et t/v sont des «pentes réduites» (i.e le pgcd de u, v et de t, v est égal à 1). La taille des plages est obtenue en calculant $m = (u - v)/v$. La plage horizontale (H) est composée de $0^{m+1}1$. La plage oblique (O) est formée de 0^m1. Les plages seront réparties de façon homogène par la méthode DDA. Les déplacements verticaux (2) seront placés lors de cette répartition. Pour cela, nous utilisons une seconde valeur d'écart δ_z. Elle mesure l'écart entre la droite réelle et la droite discrète de pente t/v. Nous faisons évoluer cet écart grâce à un incrément vertical i_v. Prenons comme exemple la droite ((0,0,0) (21,11,4)). Nous obtenons :

$$u = 27, v = 11, t = 4, m = 1, \mathrm{H} = 001, \mathrm{O} = 01$$

$$i_o = -5, i_h = 6, i_v = -7$$

$$\delta = -5, \delta_z = -7$$

i	11	10	9	8	7	6	5	4	3	2	1
δ	-5	1	-4	2	-3	3	-2	4	-1	5	0
δ_z	-7	-3	1	-6	-2	2	-5	-1	3	-4	0
mot	H	O	H2	O	H	O2	H	O	H2	O	O2

TAB. 1: Résultat de l'exécution du calcul par plages

On remarque que les lettres sont placées symétriquement. La symétrie démontrée pour le calcul de droites discrètes bidimensionnelles est conservée pour le calcul de droites discrètes tridimensionnelles. Seules les lettres extrêmes : $i = v$ et $i = 1$ sont différentes. Par définition, nous savons que toutes les droites inférieures commencent par un H et finissent par O2. Les autres lettres étant placées de façon symétrique, nous pouvons diviser le nombre de tests et d'opérations par 2. Le calcul de la première moitié du segment produit la seconde. Le temps d'exécution linéaire par rapport à v est divisé par 2. Nous présentons ici l'algorithme sous la forme d'une fonction en langage C. Les paramètres de cette fonction sont :

$i = v$	représentant le nombre total de plages dans le mot
$j = v - u\%v$	représentant le nombre de plages obliques
$k = t$	représentant le nombre de déplacements en hauteur
H	la plage horizontale
O	la plage oblique

C'est cette fonction qui a été utilisée pour les mesures de performances exposées dans la section suivante.

afficher et afficherfin sont des fonctions chargées de l'affichage d'une plage à l'écran respectivement au début et à la fin du segment. Le «pas de deux» [RWW90] a été intégré à cet algorithme afin d'améliorer encore ses performances. Remarquons de plus que le calcul de la droite inférieure, associé à une 2–connexité verticale est bien adaptée aux calculs d'intersections droite/voxels. En effet les voxels parcourus sont alors toujours contenus dans la liste ainsi déterminée.

6 Tests et performances

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

Protocole utilisé pour les tests : Nous calculons toutes les droites partant de l'origine et décrivant le volume de la pyramide suivante : ((0,0,0), (x,0,0), (x,x/2,0), (x,x/2,x/2), (x,0,x/2)).

```
void DessinSymetrique(int i, int j, int k, int H[], int O[]) {
  int io, ih, iv;
  int d, dz;

  io = j-i;
  ih = j;
  iv = k-i;
  d = 2*j-i;
  dz = 2*k-i;

  afficher(H); affichерfin(2); affichерfin(O);
  i = (i-1)/2;
  while (i--) {
    if (delta>=0) {
      afficher(O); affichерfin(O);
      delta+=io;
    } else {
      afficher(H); affichерfin(H);
      delta+=ih;
    }
    if (dz>=0){
      afficher(2); affichерfin(2);
      delta+=iv;
    } else
      delta+=k;
  }
}
```

FIG. 2: Fonction en C permettant le plaement symétrique des plages

x	1 : Kaufman	2 : Nouvel algorithme	3 : 2+Double pas
100	2,50	0,37	0,32
200	40,92	5,33	4,42
300	208,60	25,90	20,98
400	667,22	79,82	63,89
500	1621,93	192,50	152,58
600	3361,29	395,74	311,86
700	6234,93	729,03	571,24
800	10769,19	1237,89	969,00
900	17252,16	1975,25	1552,78
1000	26123,60	3001,48	2359,23

TAB. 2: Comparaison des différents algorithmes

Le temps indiqué dans le tableau qui suit correspond à la valeur «user» de la commande Unix time. Ces temps sont donnés en seconde 2.

Ces résultats ne comprennent pas les algorithmes utilisant le pgcd. En effet ce calcul nécessite, en moyenne, plus de temps qu'il n'en rapporte. Dans un souci de simplicité nous avons considéré que les pentes v/u et t/v étaient des pentes réduites. Le rapport entre l'ancien algorithme et le nôtre est de l'ordre de 11. Le double pas permet d'améliorer les temps de calcul de 1,27 fois.

7 Conclusion

Notre méthode a donc permis une amélioration très significative des temps de calcul d'une droite tridimensionnelle dans un espace discret composé de voxels uniformes. En effectuant les calculs plage par plage et non plus point par point, nous avons diminué le nombre d'opérations nécessaires au calcul de droites. Entre l'algorithme de Kaufman et le nôtre, nous divisons le temps de calcul par 7 au moins. Plus la pente d'une droite est faible, plus le gain est important. De plus, plus la droite est longue, plus l'accélération augmente : pour une droite comportant 1000 points, le facteur est alors de 11. Cette méthode permet donc de connaître plus rapidement tous les voxels traversés par un rayon. En l'intégrant, nous devrions ainsi optimiser notablement les algorithmes d'illumination. Ceci permettra par exemple d'augmenter le nombre d'objets gérables ou d'améliorer l'interactivité entre l'utilisateur et la machine.

Références

[AM91] E. Angel and D. Morrison. Speeding up bresenham's algorithm. *IEEE Coputer Graphics & Applications*, 11 :16–17, November 1991.

[Ber90] J. Berstel. *Mots, Mélanges offerts à MP. Schützenbzereger*, chapter Tracé de droites, fractions continues et morphismes itérés. Hermès, 1990.

[Bre65] JE. Bresenham. An incremental algorithm for digital plotting. *IBM System Journal*, 4(1) :25–30, 1965.

[Bre85] JE. Bresenham. Run length slice algorithm for incremental lines. In *Fundemental Algorithms in Computer Graphics*, pages 59–102. Springer-Verlag, 1985.

[BTB97] V. Boyer, J. Tayeb, and JJ. Bourdin. Une accélération du tracé de droites. In *I.A.C : Intelligence Artificielle et Compléxité*, pages 142–149, February 1997.

[CB96] FP. Chalopin and JJ. Bourdin. Straight lines : a step by step method. In *Winter School in Computer Graphics*, Plzen, February 1996.

[CK91] D. Cohen and A. Kaufman. Scan-conversion algorithm for linear and quadratic objects. *Volume Visualisation*, pages 280–300, 1991.

[CKBB90] D. Cohen, A. Kaufman, R. Bakalash, and S. Bergman. Real-time discrete shading. *Visual Computer*, 6 :16–27, 1990.

[CP85] CMA. Castle and MLV. Pitteway. An application of euclid's algorithm to drawing straigth lines. In *Fundemental Algorithms in Computer Graphics*, pages 135–139. Springer-Verlag, 1985.

[DGB90] S. Dulucq and D. Goyou-Beauchamps. Sur les facteurs des suites de sturm. *Theorical Coputer Science*, 71 :381–400, 1990.

[FDFH90] JD. Foley, A. Van Dam, S. Feiner, and J. Hughes. *Computer graphics, principes and practice*. Addison Wesley, 1990.

[GI94] P. Graham and SS. Iyengar. Double and triple step incremental generation of lines. In *IEEE Computer Graphics & Application*, pages 49–53, May 1994.

[Kau88] A. Kaufman. Efficient algorithms for scan-converting 3d polygons. *Computer & Graphics*, 12(2) :213–219, 1988.

[MC94] C. Metge and R. Caubet. Une structure de données discrète pour les scènes de radiosité. In *Deuxièmes journées de l'AFIG*, pages 43–51, Toulouse, 1994.

[Pit85] MLV. Pitteway. Euclid's algorithm and lines drawing. In *Fundemental Algorithms in Computer Graphics*, pages 101–106. Springer-Verlag, 1985.

[Rev90] JP. Reveilles. Droites discretes et fonctions continues, January 1990.

[RWW90] JG. Rockne, B. Wyvill, and X. Wu. Fast line scan-conversion. *ACM Transaction on Graphics*, 9(4) :376–388, October 1990.

[SC95] N. Stolte and R. Caubet. Discrete ray-tracing of huge voxel spaces. In *Eurographics'95*, volume 14 of *3*, pages 383–394, 1995.

[Udu91] JK. Udupa. 3d discrete space, 1991. University of Pensylvania.

[Whi80] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6) :343–349, 1980.