Synthèse d'image par lancer de rayon sur un hypercube

Thierry Priol, Kadi Bouatouch

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N°752

SYNTHESE D'IMAGE PAR LANCER DE RAYON SUR UN HYPERCUBE

Thierry PRIOL
Kadi BOUATOUCH

NOVEMBRE 1987

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

SYNTHESE D'IMAGE PAR LANCER DE RAYON SUR UN HYPERCUBE*

Thierry PRIOL
Kadi BOUATOUCH

IRISA
Campus de Beaulieu
35042 RENNES CEDEX
FRANCE

Publication Interne n° 376 - Octobre 1987 - 10 pages

Résumé :

Nous décrivons dans cet article les travaux que nous menons dans le domaine de la synthèse d'image par lancer de rayon sur une machine multiprocesseurs de type hypercube. Notre but est de concevoir des algorithmes plus performants que ceux actuellement utilisés sur des machines parallèles telles que la machine CRISTAL du CCETT . Notre démarche consiste à implémenter sur un hypercube iPSC plusieurs algorithmes de lancer de rayon et d'en comparer les performances.

RAY TRACING ON AN iPSC HYPERCUBE

Abstract :

We describe in this paper, the implementation of our ray tracing algorithms on an iPSC hypercube multi-processors.

Our goal is to propose parallel algorithms more efficient than those implemented on machines like CRISTAL designed at the CCETT. Our approach consists in implementing on a hypercube, several algorithms and comparing their performances.

Synthèse d'image par lancer de rayon sur un hypercube *

Thierry Priol
Kadi Bouatouch

IRISA †
Campus de Beaulieu
35042 Rennes Cedex
France

Résumé

Nous décrivons dans cet article les travaux que nous menons dans le domaine de la synthèse d'image par lancer de rayon sur une machine multiprocesseurs de type hypercube. Notre but est de concevoir des algorithmes plus performants que ceux actuellement utilisés sur des machines parallèles telles que la machine CRISTAL du CCETT [1]. Notre démarche consiste à implémenter sur un hypercube iPSC plusieurs algorithmes de lancer de rayon et d'en comparer les performances.

1 Introduction

La méthode de synthèse d'image par lancer de rayon est incontestablement celle qui produit des images les plus réalistes. Cependant elle présente l'inconvénient d'être coûteuse en temps et demande plusieurs heures de calcul sur un mini-ordinateur. Depuis quelques mois sont commercialisées des machines multiprocesseurs offrant un bon rapport coût-performance. Il est donc intéressant d'utiliser ce type de machine pour la synthèse d'image par lancer de rayon. Après une introduction à la méthode du lancer de rayon, nous décrivons dans le paragraphe 3 les travaux effectués à l'IRISA sur une machine hypercube iPSC.

*Ces travaux sont réalisés dans le cadre de la convention no 86ME46 entre le CCETT et l'IRISA et avec le soutien de C^3.

†Institut de Recherche en Informatique et Systèmes Aléatoires.

[1] Centre Commun d'Etudes de Télédiffusion et Télécommunications

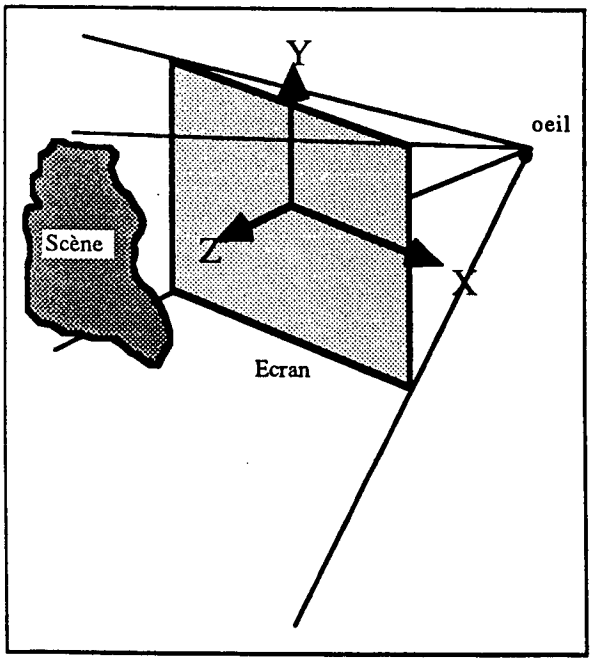

Figure 1: Principe du lancer de rayon.

2 La méthode du lancer de rayon

L'idée développée par Goldstein et Nagel [9] était de simuler le fonctionnement d'un appareil photographique en sens inverse. Pour chaque pixel de l'écran, on lance un rayon partant du point de vue et passant par ce pixel afin d'identifier les surfaces visibles (figure 1). Connaissant les positions des sources

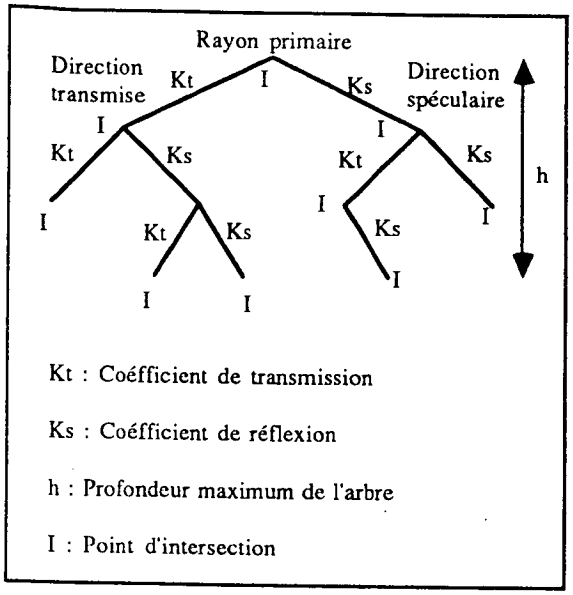

Figure 3: Arbre photométrique

objets intersectés par un rayon. Des améliorations ont été apportées afin de diminuer cette recherche. Elle consiste à attribuer à chaque feuille (objets) et à chaque noeud un volume englobant sphérique et parallélépipèdique. Cette technique permet, à un moment donné dans la recherche, d'éliminer un sous-arbre quand le rayon n'intersecte pas le volume englobant associé à ce sous-arbre. Malgré cela , l'algorithme de Roth est peu efficace dans le cas où des images possèdent quelques milliers d'objets.

2.3 Algorithmes de subdivision spatiale

De nouvelles techniques, basées sur la subdivision spatiale, permettent d'obtenir un gain de l'ordre de 20 par rapport à l'algorithme que nous venons de décrire. Ces algorithmes consistent à subdiviser l'espace 3D, contenant la scène à synthétiser en régions parallélépipèdiques ne contenant qu'un nombre minimum de primitives [8,11,3]. Un rayon pénétrant dans une région n'intersecte que peu de primitives. Cette technique permet de diminuer considérablement le nombre d'intersections et le temps de synthèse devient relativement indépendant du nombre de primitives contenues dans la scène. Nous avons mis au point un nouvel algorithme de subdivision spatiale basée sur un découpage irrégulier de la scène tout en utilisant la modélisation CSG [1].

Cet algorithme consiste en deux phases :

1. La phase de prétraitement permet d'effectuer un partitionnement binaire de l'espace, appelé BSP [3] [7], sur les volumes englobants minimaux projetés sur l'écran (figure 4).

   Chaque région de l'espace ainsi créée est découpée en profondeur par les faces des volumes englobants minimaux perpendiculaires à l'axe des z afin de minimiser le nombre de primitives dans chaque région. Ce découpage n'est pas suffisant car il ne permet pas aux rayons de pouvoir passer d'une région à l'autre. Lors du découpage une structure de donnée appelée *pointeurs de connexité* est mise à jour, permettant ainsi le calcul de la région suivante dans la direction du rayon dans la phase de synthèse (figure 5).

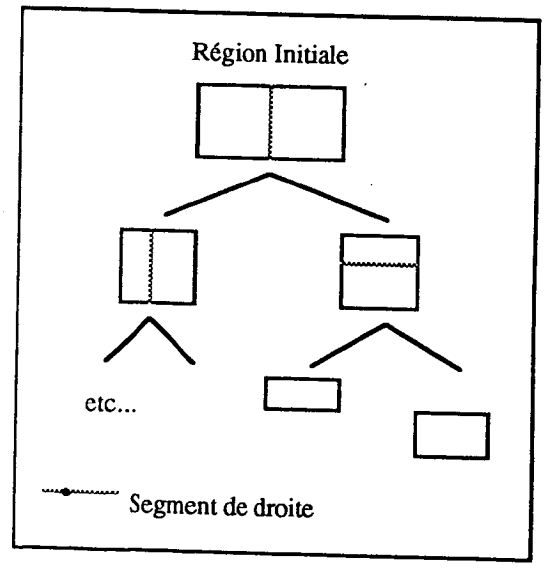

Figure 4: Découpage par un BSP.

2. Dans la phase de synthèse, les rayons primaires sont envoyés dans les régions ayant, en projection sur l'écran, des primitives. Quand un rayon quitte une région, le calcul de la région suivante est réalisé grâce aux pointeurs de connexité.

   Un mécanisme de boîte aux lettres permet d'éviter de calculer plusieurs fois l'intersection entre un rayon et une même primitive partagée par deux régions différentes.

[3] Binary Space Partionning

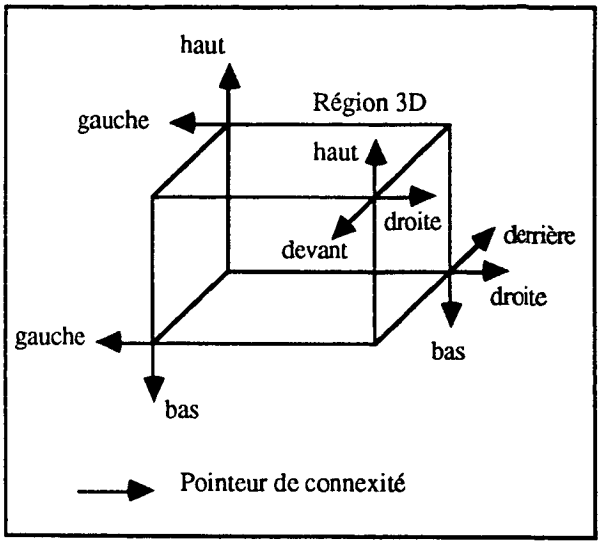

Figure 5: Pointeurs de connexité.

Nous allons décrire dans la chapitre suivant la parallélisation de deux algorithmes de lancer de rayon. Le premier est basé sur l'algorithme de Roth tandis que le deuxième utilise la technique exposée dans la section précédente. Nous employons pour ce travail une machine multi-processeurs de type iPSC.

3 Utilisation d'un hypercube iPSC pour le lancer de rayon

3.1 iPSC et topologie hypercube

L'iPSC est une machine multi-processeurs commercialisée par Intel et qui résulte de recherches effectuées au Caltech [4]. Les processeurs de l'iPSC sont reliés entre eux selon une topologie hypercube qui est en fait un cube de dimension N, où N représente le nombre de processeurs connectés directement à un processeur (figure 6).

Cette topologie permet de simuler des architectures très diverses telles que les anneaux, les réseaux maillés 2D ou 3D, les arbres, etc... La configuration que nous avons à notre disposition possède 64 processeurs 80286 ($N = 6$) ayant chacun 4.5 Megaoctets de mémoire centrale et un co-processeur flottant 80287. Ces performances en crête sont de l'ordre de 4 Mflops et 50 Mips. L'ensemble de ces processeurs est relié à un processeur frontal fonctionnant sous UNIX et permettant le chargement et l'exécution des programmes sur l'iPSC.

[4]California Institute of Technology

3.2 Parallélisation de l'algorithme de Roth

La parallélisation la plus courante de l'algorithme de Roth résulte directement du principe même de l'appareil photographique où la pellicule est impressionnée simultanément par plusieurs rayons lumineux. Chaque pixel de l'écran peut donc se traiter comme une entité indépendante. La parallélisation consiste donc à dupliquer l'arbre CSG initial dans chacun des processeurs et répartir l'écran afin d'associer à chacun d'eux un groupe de pixels à calculer. Cette allocation est réalisée par un superviseur. Le groupe de pixels peut être soit un pixel soit une ligne de pixels; cela dépend des caractéristiques du lien de communication entre les processeurs et la mémoire d'image. Une des premières réalisations de ce type d'algorithme a été effectuée sur la machine LINKS par Nishimura et al [14]. En France, des travaux efectués par le CCETT ont abouti à la réalisation de la machine CRISTAL. Cette machine multi-processeurs comporte 10 processeurs NS32032 avec 512k de mémoire ainsi qu'un opérateur flottant. Sa topologie consiste en plusieurs grappes de processeurs connectées sur un bus commun. Un superviseur alloue à chaque processeur un pixel à calculer. Les performances de cette machine permettent d'obtenir des images d'une centaine de primitives en quelques dizaines de minutes.

A l'IRISA, nous avons nous même mis en oeuvre ce type d'algorithme sur un hypercube iPSC afin d'avoir une base de comparaison avec les futurs algorithmes de lancer de rayon. Les premiers résultats montrent un gain de performance de 16 par rapport au même algorithme fonctionnant sur une machine séquentielle de type SUN3/160.

Si les performances de cet algorithme sont intéressantes sur des images ayant moins d'une centaine de primitives, celui-ci devient très coûteux pour des images comportant plusieurs milliers de primitives. En effet le temps de recherche des objets intersectés augmente considérablement dès que le nombre de primitives devient important.

Il serait intéressant de paralléliser un algorithme de lancer de rayon utilisant la subdivision spatiale et de le comparer à l'algorithme de Roth. Nous présentons dans la section suivante les premiers travaux que nous avons effectués dans ce domaine.

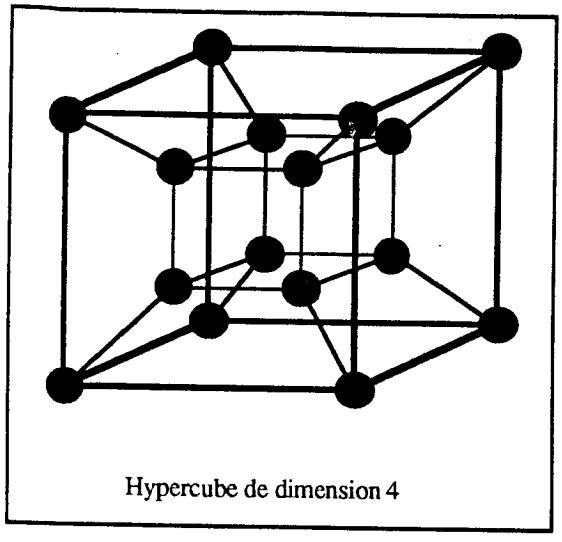

Figure 6: Topologie hypercube

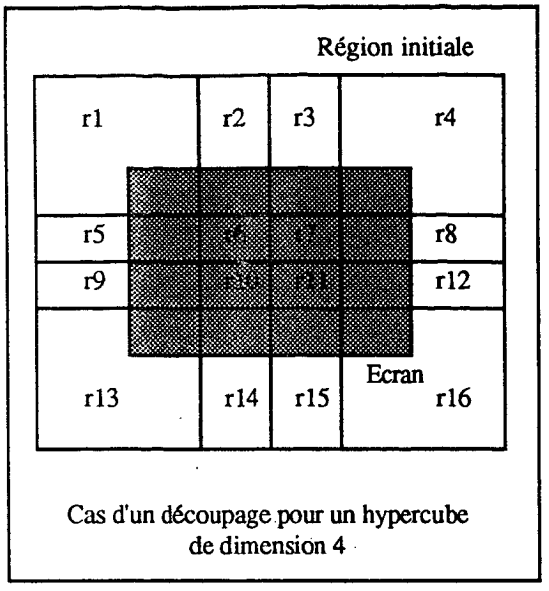

Figure 7: Subdivision régulière de l'écran

3.3 Hypercube et subdivision spatiale

Des méthodes sur la parallélisation des algorithmes utilisant la subdivision de l'espace ont été proposées. On pourra citer notamment les travaux de Nemoto et al [13]. L'algorithme s'exécute sur un réseau maillé 3D. Les auteurs abordent essentiellement l'aspect du contrôle de la charge des processeurs afin d'obtenir un comportement de l'algorithme qui utilise au mieux les ressources de calcul de la machine. La technique présentée est fondée sur une redistribution dynamique des sous-espaces associés à chaque processeur. Cette méthode est intéressante dans le cas où les objets sont modélisés par un ensemble de polygones plans mais son usage dans le cas d'une modélisation de type CSG reste prohibitif car il faudrait à chaque redistribution des sous-espaces, défaire et refaire les sous-arbres CSG. Cleary et al [5] comparent les performances d'un algorithme de subdivision spatiale implémenté sur un réseau 2D et 3D et montrent que le réseau 2D est plus performant que le 3D.

Notre objectif est de concevoir un algorithme de lancer de rayon utilisant notre algorithme de subdivision spatiale sur une machine multi-processeurs. Cet algorithme doit pouvoir synthétiser des scènes ayant 500 à 1000 primitives et avoir un facteur de rapidité important par rapport à l'algorithme de Roth sur la même machine. La technique de subdivision spatiale que nous employons offre l'avantage d'être rapide mais elle consomme beaucoup de mémoire (environ 2 Mega- octet pour une image ayant une centaine de primitives). Il n'est donc pas possible de dupliquer les informations dans chacun des processeurs si l'on veut obtenir des images complexes. Il faut alors partitionner la base de donnée initiale et associer une partie à chaque processeur.

La topologie hypercube permet d'émuler un grand nombre d'architectures (anneau, réseau maillé 2D et 3D, arbre, etc...). Nous avons choisi d'utiliser un réseau maillé 2D car il permet de minimiser les communications en rendant le calcul des rayons primaires local aux processeurs. Cette architecture est simulée sur l'iPSC en utilisant les codes de Gray ce qui élimine tout routage dans les communications générées par l'algorithme. On associe à chaque processeur une région de l'espace, appelée macro-région, suivant le découpage illustré par la figure 7. Chaque processeur a le même nombre de pixels à calculer. La taille des macro-régions est plus importante à la périphérie de la grille 2D car on doit tenir compte des objets qui ne sont pas vus directement par l'observateur. En effet, ces objets peuvent être visibles par des réflexions sur d'autres objets.

Le modèle photométrique nécessite l'usage de procédures récursives. C'est à dire que pour calculer la valeur de la couleur d'un pixel, il faut avoir évalué tous les apports dans les directions transmise et réfléchie s'il y a lieu. L'implémentation du

modèle photométrique n'est pas simple car celui-ci doit tenir compte des informations pouvant être calculées ultérieurement par d'autres processeurs. En effet un rayon peut s'échapper d'un processeur, il sera alors traité après un temps non déterminé. Il faut que le processeur ayant reçu ce rayon envoie en retour l'apport lumineux au processeur ayant émis ce rayon. Cette technique a été proposé par Cleary [5]. Elle a le désavantage de créer un grand nombre de messages dans la machine et nécessite pour son implémentation une structure de donnée volumineuse. Or il est possible d'éviter cela. En effet, si l'on examine l'arbre photométrique, l'apport lumineux calculé à un noeud de cet arbre est multiplié par le produit des coefficients de réflexion K_s et de transmission K_t du chemin entre la racine et le noeud considéré (figure 3). Il suffit donc de rajouter, dans la structure de donnée associée au rayon, suffisamment d'informations pour qu'un processeur recevant un rayon soit à même de calculer l'intensité à ajouter à la valeur du pixel et cela indépendemment des autres processeurs ayant déjà réalisé des calculs pour ce même pixel. Ces informations sont le produit des K_s et K_t depuis la racine de l'arbre photométrique et les coordonnées du pixel, auquel il faut cumuler l'apport lumineux calculé. Un processeur recevant un rayon peut alors calculer l'apport lumineux et l'envoyer au frontal de l'iPSC qui sera chargé de le cumuler dans la mémoire d'image à la bonne adresse.

3.3.1 Fonctionnement de l'algorithme sur l'iPSC

Nous allons décrire dans ce paragraphe, un algorithme de subdivision spatiale implémenté sur l'hypercube iPSC. Le fonctionnement de l'algorithme se résume en quatre étapes :

1. Le frontal de l'iPSC lit l'arbre CSG et découpe la scène et transmet à chaque processeur la macro-région qui lui est destinée ainsi que le sous-arbre CSG associé à cette macro-région.

2. Chaque processeur effectue un découpage irrégulier de sa macro-région suivant la méthode d' Arnaldi [1]. Le résultat est un ensemble de régions.

3. Quand l'opération de découpage est terminé, le processeur avertit le frontal de l'iPSC et attend la réponse de celui-ci pour commencer l'étape suivante. Quand tous les processeurs ont terminé leur découpage, le frontal émet un message à tous les processeurs pour qu'ils débutent l'étape de synthèse.

4. Les processeurs débutent la phase de synthèse. Chaque processeur possède une liste des régions où il est nécessaire de lancer des rayons issus du point de vue. Ces rayons génèrent à leur tour des rayons lumineux et secondaires qui seront traités soit par lui-même soit par d'autres processeurs. La communication de ces rayons entre processeurs s'effectue par l'envoi et la réception de message. L'échange des messages se fait par l'intermédiaire d'une file d'attente FIFO afin d'éviter un blocage temporaire quand un rayon est émis vers un autre processeur. Cette file d'attente est gérée par un processus, distinct de celui réalisant l'évaluation des rayons et est associé à chacun des noeuds de l'hypercube.

Le comportement du processus d'évaluation des rayons est décrit par l'algorithme suivant :

Algorithme (en pseudo-C)

```
main()
{
  while(image_non_termine()) {

     /* Traitement des rayons primaires */

     generation_rayon_primaire(r);
     evaluation_rayon(r);

     /* Traitement des rayons provenant  */
     /* des autres processeurs           */

     while(rayon_a_lire()) {
        lire_rayon(r);
        calcul_region_entree(r);
        evaluation_rayon(r);
     }
   }
}
```

evaluation_rayon(r) Cette procédure évalue un rayon en fonction de son type. Des rayons secondaires et lumineux peuvent être émis.

calcul_region_entree(r)
Cette procédure détermine la région d'entrée d'un rayon provenant d'un autre processeur.

image_non_terminee() Cette fonction met en oeuvre l'algorithme de détection de la terminaison.

3.3.2 Evaluation des rayons

Le traitement des rayons s'effectue en fonction de leurs types. L'évaluation des rayons issus de

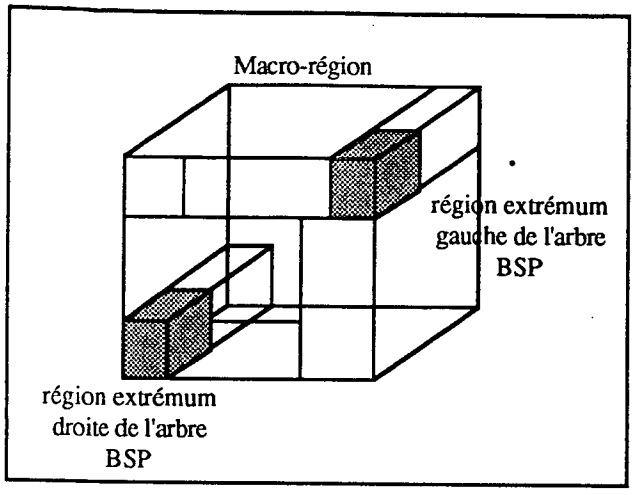

Figure 8: Calcul de la région d'entrée

l'observateur (rayon primaire) s'effectue localement au processeur. Grâce à l'usage de deux repères [1], un rayon primaire restera toujours dans la macro-région associée au processeur. Par contre, les rayons secondaires (effet miroir et transparent) et les rayons lumineux peuvent nécessiter un traitement par plusieurs processeurs. Ce sont donc ces rayons qui sont transférés entre les processeurs.

3.3.3 Calcul de la région d'entrée

Puisque chaque processeur n'a pas la connaissance de toute la base de donnée en mémoire, l'évaluation d'un rayon ne peut pas être effectué totalement par le même processeur. Quand un rayon quitte la macro-région associée à un processeur, celui-ci le transmet au processeur possèdant la macro-région suivante dans la direction du rayon. Le processeur destinataire possède un ensemble de petites régions liées entre elles par des pointeurs (figure 5) et résultant du découpage de la macro-région. Il faut donc déterminer celle dans laquelle pénètre le rayon. Il suffit lors du découpage de la macro-région de mémoriser deux régions situées aux extrêma gauche et droit de l'arbre BSP utilisé pour le découpage. Ces deux régions peuvent être déterminées facilement lors du découpage et permettent de retrouver dans tous les cas la région d'entrée en utilisant les pointeurs de connexité associés à ces deux régions (figure 8).

3.3.4 Détection de la terminaison

Chaque processeur ne peut savoir s'il a terminé son travail car il peut recevoir à tout moment un rayon à calculer d'un processeur voisin. Pour détecter la terminaison de l'algorithme, nous utilisons l'algorithme du jeton proposé par Dijkstra [6]. Pour ce faire, nous utilisons un anneau virtuel sur lequel circule un jeton ayant une couleur noire ou blanche. A l'initialisation, le processeur 0 possède le jeton de couleur blanche. Si après un tour de l'anneau le jeton est resté blanc, la terminaison de l'algorithme sera alors effective. Sinon, il suffit d'émettre un nouveau jeton de couleur blanche.

Ce jeton n'est pas envoyé au processeur suivant immédiatement. Dans le processus de synthèse, nous pouvons dégager deux étapes : la première consiste à générer les rayons primaires tout en consommant les rayons provenant des autres processeurs; la deuxième étape débute quand il n'y a plus de rayons primaires à calculer. A ce stade, le processeur à potentiellement terminé son travail et il ne lui reste qu'à consommer les rayons provenant des autres processeurs. C'est dans cette étape que le jeton va circuler de processeur en processeur sur l'anneau virtuel. Chaque processeur ne fait circuler le jeton que s'il est lui même rendu dans cette étape. La conséquence de ce mécanisme est de minimiser le nombre de jetons pour détecter la terminaison effective de l'algorithme. Les premiers résultats ont montrés qu'une vingtaine de jetons sont nécessaires pour détecter la terminaison de l'algorithme.

3.3.5 Premiers résultats

Les premiers tests ont été effectués sur une image possèdant une centaine de primitives et une résolution de 256 x 256 pixels. Le temps de synthèse est supérieur à ceux obtenus avec l'algorithme de Roth. En fait, l'algorithme est très efficace dans un premier temps puis son efficacité diminue rapidement. Une analyse du comportement de l'algorithme montre que l'algorithme converge vers un état où 75% des processeurs sont bloqués car les files d'attente sont saturées. Or si l'on examine l'algorithme , il y a priorité à la consommation afin d'éviter que les files d'attente ne saturent. En réalité, cette saturation est dûe à au moins deux raisons :

1. *Propriété de convergence des rayons lumineux*

   Tous les rayons lumineux issus des autres processeurs convergent vers les processeurs les plus près des sources lumineuses ce qui provoque une saturation des files d'attente de ces processeurs

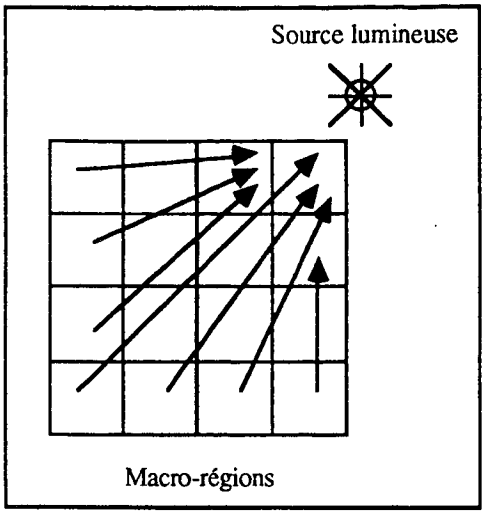

Figure 9: Convergence des rayons lumineux

et de proche en proche la saturation des files d'attentes des autres processeurs (figure 9).

2. *Cohérence objet*

   Entre deux pixels voisins les directions des rayons primaires passant par ces pixels sont peu différentes. Par conséquent les rayons lumineux ou secondaires auront eux aussi des directions à peu près identiques et auront donc tendance à emprunter les mêmes chemins. Ce phénomène appelé *cohérence* permet de réutiliser les calculs précédents pour le calcul d'un nouveau rayon lors d'une implémentation séquentielle des algorithmes mettant en oeuvre la subdivision spatiale. Dans le cas d'une architecture parallèle, cette propriété entraîne la saturation des ressources de communication et de calcul sur un sous ensemble restreint des processeurs.

4 Conclusion

Nous venons de montrer que la parallélisation d'un algorithme de lancer de rayon utilisant la technique de subdivision spatiale mettait en jeu de nombreux problèmes que l'on peut trouver en algorithmique distribuée. Nous travaillons actuellement sur la minimisation du nombre de messages entre processeurs, grâce notamment au calcul des rayons lumineux traités par le processeur qui les a créés. Nous pensons que ceci permettrait de supprimer la propriété de convergence des rayons, donc d'éliminer la saturation des files d'attente et permettrait l'équilibre des tâches associées à chaque processeur.

Bibliographie

[1] B. Arnaldi, T. Priol, and K. Bouatouch. A new space subdivision method for ray tracing csg modelled scenes. *The Visual Computer*, 3(2):98–108, August 1987.

[2] K. Bouatouch, B. Arnaldi, and T. Priol. Lgrc: a language for image synthesis by ray-casting. *TSI*, 6:475–489, Novembre 1986.

[3] K. Bouatouch, M.O Madani, T. Priol, and B. Arnaldi. A new algorithm of space tracing using a csg model. In *EUROGRAPHICS'87 Conference Proceeding*, pages 65–78, Centre for Mathematics and Computer Science, August 1987.

[4] J. Clark. Designing surfaces in 3d. *CACM*, 19(8), August 1976.

[5] J.G Cleary, B. Wyvill, G.M. Birtwistle, and R. Vatti. *Multiprocessor Ray Tracing*. Research Report 83/128/17, University of Calgary, October 1983.

[6] E.W Dijkstra, W.H.J Feijen, and A.J.M Van Gasteren. Derivation of a termination detection algorithm for distributed computation. *Inf. Proc. Letters*, 16:217–219, June 1983.

[7] H. Fuchs. On visible surface generation by a priori tree structure. In *SIGGRAPH'80 Conference Proceeding*, pages 149–158, July 1980.

[8] A. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[9] R.A. Goldstein and R. Nagel. 3d modeling with synthavision simulation. *Simulation*, 16(1):25–31, January 1971.

[10] A. Roy Hall and Donald P. Greenberg. A testbed for realistic image synthesis. *IEEE Computer Graphics and Applications*, 3(8):10–20, November 1983.

[11] M. R. Kaplan. Space-tracing, a constant time ray tracer. In *SIGGRAPH'85 tutorial on the uses of spatial coherence in ray tracing*, 1985.

[12] Benoit Mandelbrot. *Les objets fractals*. Flammarion, 1984.

[13] K. Nemoto and T. Omachi. An adaptative subdivision by sliding boundary surfaces for fast ray tracing. *Graphics Interface*, 43–48, 1986.

[14] H. Nishimura, H. Ohno, T. Kawata, I. Shirakawa, and K. Omuira. Links-1: a parallel pipelined multimicrocomputer system for image creation. In *Proc. of the 10th Symp. on Computer Architecture*, pages 387–394, 1983.

[15] B.T. Phong. Illumination model for computer generated images. *Communications of the ACM*, 18:311–317, June 1975.

[16] A.A. Requicha. Representation for rigid solids : theory, methods, and systems. *ACM Computing Surveys*, 12(4):437–464, December 1980.

[17] S.D Roth. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18(2):109–144, February 1982.

[18] R.B Tilove and A.A.G Requicha. Closure of boolean operations on geometric entities. *Computer Aided Design*, 12(5):219–220, September 1980.

[19] K.E. Torrance and E.M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of Optical Society of America*, 57(9):1105–1114, September 1967.

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

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique