NUMERO D'ORDRE : 1357

USTL

ANNEE : 1994

LIFL Laboratoire d'Informatique Fondamentale de Lille

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

S.DEGRANDE

Définition d'une machine cellulaire pour la mise en oeuvre d'un lancer de rayon massivement parallèle

Thèse soutenue le 13 Juillet 1994, devant la commission d'examen :

Président :	V. CORDONNIER	Université de Lille 1
Directeur de Thèse :	M. MERIAUX	Université de Lille 1
Rapporteurs :	R. CAUBET	Université Paul Sabatier - Toulouse
	G. MAZARE	Université de Grenoble
Examinateurs :	G. ATAMENIA	Université de Lille 1

UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE
U.F.R. d'I.E.E.A. Bât M3. 59655 Villeneuve d'Ascq CEDEX
Tél. 20.43.47.24 Fax. 20.43.65.66

UNIVERSITE DES SCIENCES
ET TECHNOLOGIES DE LILLE

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M. H. LEFEBVRE, M. PARREAU

PROFESSEURS HONORAIRES DES ANCIENNES FACULTES DE DROIT ET SCIENCES ECONOMIQUES, DES SCIENCES ET DES LETTRES

MM. ARNOULT, BONTE, BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, DECUYPER, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, GERMAIN, GLACET, GONTIER, KOURGANOFF, LAMOTTE, LASSERRE, LELONG, LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, PEREZ, ROIG, ROSEAU, ROUELLE, SCHILTZ, SAVARD, ZAMANSKI, Mes BEAUJEU, LELONG.

PROFESSEUR EMERITE

M. A. LEBRUN

ANCIENS PRESIDENTS DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. M. PARREAU, J. LOMBARD, M. MIGEON, J. CORTOIS, A.DUBRULLE

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

M. P. LOUIS

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CHAMLEY Hervé	Géotechnique
M. CONSTANT Eugène	Electronique
M. ESCAIG Bertrand	Physique du solide
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. LABLACHE COMBIER Alain	Chimie
M. LOMBARD Jacques	Sociologie
M. MACKE Bruno	Physique moléculaire et rayonnements atmosphériques

M. MIGEON Michel	EUDIL
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie appliquée

PROFESSEURS - 1ère CLASSE

M. BACCHUS Pierre	Astronomie
M. BIAYS Pierre	Géographie
M. BILLARD Jean	Physique du Solide
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean Pierre	Chimie-Physique
M. BOSCQ Denis	Probabilités
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BRASSELET Jean Paul	Géométrie et topologie
M. BREZINSKI Claude	Analyse numérique
M. BRIDOUX Michel	Chimie Physique
M. BRUYELLE Pierre	Géographie
M. CARREZ Christian	Informatique
M. CELET Paul	Géologie générale
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. CROSNIER Yves	Electronique
Mme DACHARRY Monique	Géographie
M. DAUCHET Max	Informatique
M. DEBOURSE Jean Pierre	Gestion des entreprises
M. DEBRABANT Pierre	Géologie appliquée
M. DECLERCQ Roger	Sciences de gestion
M. DEGAUQUE Pierre	Electronique
M. DESCHEPPER Joseph	Sciences de gestion
Mme DESSAUX Odile	Spectroscopie de la réactivité chimique
M. DHAINAUT André	Biologie animale
Mme DHAINAUT Nicole	Biologie animale
M. DJAFARI Rouhani	Physique
M. DORMARD Serge	Sciences Economiques
M. DOUKHAN Jean Claude	Physique du solide
M. DUBRULLE Alain	Spectroscopie hertzienne
M. DUPOUY Jean Paul	Biologie
M. DYMENT Arthur	Mécanique
M. FOCT Jacques Jacques	Métallurgie
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie structurale
M. FRONTIER Serge	Ecologie numérique
M. GLORIEUX Pierre	Physique moléculaire et rayonnements atmosphéri
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie-Physique
M. GRANELLE Jean Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILBAULT Pierre	Physiologie animale
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. HENRY Jean Pierre	Génie mécanique
M. HERMAN Maurice	Physique spatiale
M. LACOSTE Louis	Biologie Végétale
M. LANGRAND Claude	Probabilités et statistiques

M. LATTEUX Michel	Informatique
M. LAVEINE Jean Pierre	Paléontologie
Mme LECLERCQ Ginette	Catalyse
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique atomique et moléculaire
M. LEROY Jean Marie	Spectrochimie
M. LHENAFF René	Géographie
M. LHOMME Jean	Chimie organique biologique
M. LOUAGE Françis	Electronique
M. LOUCHEUX Claude	Chimie-Physique
M. LUCQUIN Michel	Chimie physique
M. MAILLET Pierre	Sciences Economiques
M. MAROUF Nadir	Sociologie
M. MICHEAU Pierre	Mécanique des fluides
M. PAQUET Jacques	Géologie générale
M. PASZKOWSKI Stéfan	Mathématiques
M. PETIT Francis	Chimie organique
M. PORCHET Maurice	Biologie animale
M. POUZET Pierre	Modélisation - calcul scientifique
M. POVY Lucien	Automatique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. RAMAN Jean Pierre	Sciences de gestion
M. SALMER Georges	Electronique
M. SCHAMPS Joël	Spectroscopie moléculaire
Mme SCHWARZBACH Yvette	Géométrie
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
M. SLIWA Henri	Chimie organique
M. SOMME Jean	Géographie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. THIEBAULT François	Sciences de la Terre
M. THOMAS Jean Claude	Géométrie - Topologie
M. THUMERELLE Pierre	Démographie - Géographie humaine
M. TILLIEU Jacques	Physique théorique
M. TOULOTTE Jean Marc	Automatique
M. TREANTON Jean René	Sociologie du travail
M. TURRELL Georges	Spectrochimie infrarouge et raman
M. VANEECLOO Nicolas	Sciences Economiques
M. VAST Pierre	Chimie inorganique
M. VERBERT André	Biochimie
M. VERNET Philippe	Génétique
M. VIDAL Pierre	Automatique
M. WALLART Françis	Spectrochimie infrarouge et raman
M. WEINSTEIN Olivier	Analyse économique de la recherche et développement
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ABRAHAM Francis	
M. ALLAMANDO Etienne	Composants électroniques
M. ANDRIES Jean Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BALL Steven	Génétique
M. BART André	Biologie animale
M. BASSERY Louis	Génie des procédés et réactions chimiques
Mme BATTIAU Yvonne	Géographie
M. BAUSIERE Robert	Systèmes électroniques
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique atomique et moléculaire
M. BERNAGE Pascal	Physique atomique, moléculaire et du rayonnement
M. BERTHOUD Arnaud	Sciences Economiques
M. BERTRAND Hugues	Sciences Economiques
M. BERZIN Robert	Analyse
M. BISKUPSKI Gérard	Physique de l'état condensé et cristallographie
M. BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie végétale
M. BOHIN Jean Pierre	Biochimie métabolique et cellulaire
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie civil
M. BOIVIN Jean Claude	Spectrochimie
M. BOUCHER Daniel	Physique
M. BOUQUELET Stéphane	Biologie appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BROCARD Jacques	Chimie
Mme BROUSMICHE Claudine	Paléontologie
M. BUISINE Daniel	Mécanique
M. CAPURON Alfred	Biologie animale
M. CARRE François	Géographie humaine
M. CATTEAU Jean Pierre	Chimie organique
M. CAYATTE Jean Louis	Sciences Economiques
M. CHAPOTON Alain	Electronique
M. CHARET Pierre	Biochimie structurale
M. CHIVE Maurice	Composants électroniques optiques
M. COMYN Gérard	Informatique théorique
Mme CONSTANT Monique	Composants électroniques et optiques
M. COQUERY Jean Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique nucléaire et corpusculaire
M. COUTURIER Daniel	Chimie organique
M. CRAMPON Norbert	Tectolique géodynamique
M. CURGY Jean Jacques	Biologie
M. DANGOISSE Didier	Physique théorique
M. DE PARIS Jean Claude	Analyse
M. DECOSTER Didier	Composants électroniques et optiques
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean Paul	Informatique
M. DELORME Pierre	Physiologie animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
Mme DEMUYNCK Claire	Physique atomique, moléculaire et du rayonnemen
M. DENEL Jacques	Informatique
M. DEPREZ Gilbert	Physique du solide - christallographie

M. DERIEUX Jean Claude	Microbiologie
M. DERYCKE Alain	Informatique
M. DESCAMPS Marc	Physique de l'état condensé et cristallographie
M. DEVRAINNE Pierre	Chimie minérale
M. DEWAILLY Jean Michel	Géographie humaine
M. DHAMELINCOURT Paul	Chimie physique
M. DI PERSIO Jean	Physique de l'état condensé et cristallographie
M. DUBAR Claude	Sociologie démographique
M. DUBOIS Henri	Spectroscopie hertzienne
M. DUBOIS Jean Jacques	Géographie
M. DUBUS Jean Paul	Spectrométrie des solides
M. DUPONT Christophe	Vie de la firme
M. DUTHOIT Bruno	Génie civil
Mme DUVAL Anne	Algèbre
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FARVACQUE Jean Louis	Physique de l'état condensé et cristallographie
M. FAUQUEMBERGUE Renaud	Composants électroniques
M. FELIX Yves	Mathématiques
M. FERRIERE Jacky	Tectonique - Géodynamique
M. FISCHER Jean Claude	Chimie organique, minérale et analytique
M. FONTAINE Hubert	Dynamique des cristaux
M. FORSE Michel	Sociologie
M. GADREY Jean	Sciences économiques
M. GAMBLIN André	Géographie urbaine, industrielle et démographie
M. GOBLOT Rémi	Algèbre
M. GOURIEROUX Christian	Probabilités et statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLOT Jean	Chimie organique
M. GUELTON Michel	Chimie physique
M. GUICHAOUA André	Sociologie
M. HAIMAN Georges	Modélisation,calcul scientifique, statistiques
M. HOUDART René	Physique atomique
M. HUEBSCHMANN Johannes	Mathématiques
M. HUTTNER Marc	Algèbre
M. ISAERT Noël	Physique de l'état condensé et cristallographie
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et statistiques
M. JEAN Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KOENIG Gérard	Sciences de gestion
M. KOSTRUBIEC Benjamin	Géographie
M. KREMBEL Jean	Biochimie
Mme KRIFA Hadjila	Sciences Economiques
M. LANGEVIN Michel	Algèbre
M. LASSALLE Bernard	Embryologie et biologie de la différenciation
M. LE MEHAUTE Alain	Modélisation,calcul scientifique,statistiques
M. LEBFEVRE Yannic	Physique atomique,moléculaire et du rayonnement
M. LECLERCQ Lucien	Chimie physique
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Marc	Composants électroniques et optiques
M. LEFEVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
M. LEGRAND Michel	Astronomie - Météorologie
M. LEGRAND Pierre	Chimie
Mme LEGRAND Solange	Algèbre
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne

M. LE MAROIS Henri	Vie de la firme
M. LEMOINE Yves	Biologie et physiologie végétales
M. LESCURE François	Algèbre
M. LESENNE Jacques	Systèmes électroniques
M. LOCQUENEUX Robert	Physique théorique
Mme LOPES Maria	Mathématiques
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU François	Sciences économiques
M. MAHIEU Jean Marie	Optique - Physique atomique
M. MAIZIERES Christian	Automatique
M. MANSY Jean Louis	Géologie
M. MAURISSON Patrick	Sciences Economiques
M. MERIAUX Michel	EUDIL
M. MERLIN Jean Claude	Chimie
M. MESMACQUE Gérard	Génie mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MOCHE Raymond	Modélisation,calcul scientifique,statistiques
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie organique
M. MORE Marcel	Physique de l'état condensé et cristallographie
M. MORTREUX André	Chimie organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
M. NIAY Pierre	Physique atomique,moléculaire et du rayonnement
M. NICOLE Jacques	Spectrochimie
M. NOTELET Francis	Systèmes électroniques
M. PALAVIT Gérard	Génie chimique
M. PARSY Fernand	Mécanique
M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. PERTUZON Emile	Physiologie animale
M. PETIT Daniel	Biologie des populations et écosystèmes
M. PLIHON Dominique	Sciences Economiques
M. PONSOLLE Louis	Chimie physique
M. POSTAIRE Jack	Informatique industrielle
M. RAMBOUR Serge	Biologie
M. RENARD Jean Pierre	Géographie humaine
M. RENARD Philippe	Sciences de gestion
M. RICHARD Alain	Biologie animale
M. RIETSCH François	Physique des polymères
M. ROBINET Jean Claude	EUDIL
M. ROGALSKI Marc	Analyse
M. ROLLAND Paul	Composants électroniques et optiques
M. ROLLET Philippe	Sciences Economiques
Mme ROUSSEL Isabelle	Géographie physique
M. ROUSSIGNOL Michel	Modélisation,calcul scientifique,statistiques
M. ROY Jean Claude	Psychophysiologie
M. SALERNO Françis	Sciences de gestion
M. SANCHOLLE Michel	Biologie et physiologie végétales
Mme SANDIG Anna Margarette	
M. SAWERYSYN Jean Pierre	Chimie physique
M. STAROSWIECKI Marcel	Informatique
M. STEEN Jean Pierre	Informatique
Mme STELLMACHER Irène	Astronomie - Météorologie
M. STERBOUL François	Informatique
M. TAILLIEZ Roger	Génie alimentaire
M. TANRE Daniel	Géométrie - Topologie
M. THERY Pierre	Systèmes électroniques
Mme TJOTTA Jacqueline	Mathématiques
M. TOURSEL Bernard	Informatique
M. TREANTON Jean René	Sociologie du travail

M. TURREL Georges	Spectrochimie infrarouge et raman
M. VANDIJK Hendrik	
Mme VAN ISEGHEM Jeanine	Modélisation,calcul scientifique,statistiques
M. VANDORPE Bernard	Chimie minérale
M. VASSEUR Christian	Automatique
M. VASSEUR Jacques	Biologie
Mme VIANO Marie Claude	
M. WACRENIER Jean Marie	Electronique
M. WARTEL Michel	Chimie inorganique
M. WATERLOT Michel	géologie générale
M. WEICHERT Dieter	Génie mécanique
M. WERNER Georges	Informatique théorique
M. WIGNACOURT Jean Pierre	
M. WOZNIAK Michel	Spectrochimie
Mme ZINN JUSTIN Nicole	Algèbre

A tous ceux qui d'une manière ou d'une autre ont permis que soient écrits ces mots :

A tous ceux qui d'une manière ou une autre ont permis que soient écrits ces mots :

A tous ceux qui d'une manière ou une autre ont permis que soient écrits ces mots :

A tous ceux qui d'une manière ou une autre ont permis que soient écrits ces mots :

Remerciements

Cette page est, m'avait-on dit, la plus difficile à écrire; et je ne déroge pas à la règle. Il y a tant de choses que j'aimerais exprimer et si peu que j'arriverais à poser. C'est pourtant une des rares occasions qui m'est donnée de pouvoir dire ma gratitude. Même si les mots ne suffisent pas, merci à tous ceux qui ont permis que je puisse tenter de l'écrire aujourd'hui.

Je remercie particulièrement Mr le Professeur V. CORDONNIER pour avoir accepté de présider ce jury. C'est pour moi un grand honneur.

Mr M. MERIAUX, Professeur à l'EUDIL, a encadré ces travaux. En m'accordant son soutien, il m'a permis de les mener à terme. Je lui dois beaucoup.

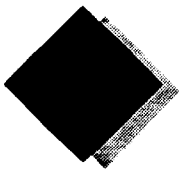
Mr R. CAUBET, Professeur à l'Université Paul Sabatier, et Mr G. MAZARE, Professeur à l'Université de Grenoble, ont eu la gentillesse, et la patience, de rapporter ce texte. Je les remercie vivement du temps qu'ils ont bien voulu m'accorder.

Mr G. ATAMENIA, Maître de Conférences, m'a guidé durant mes débuts. Il a, avec attention, examiné ce travail, et m'a conduit à améliorer les premières versions de ce texte. Je l'en remercie chaleureusement.

Depuis mon arrivée au LIFL, l'équipe GRAPHIX a connue une explosion démographique telle que je ne pourrais faire une liste exhaustive de tous ses membres sans provoquer l'ennui du lecteur. Chacun avec sa personnalité a permis de créer une ambiance chaleureuse, parfois orageuse; travailler avec eux et parmi eux est chaque jour un plaisir nouveau et une surprise renouvelée.

Mes remerciements vont également à l'équipe technique du laboratoire. De par sa qualité et son efficacité, elle assure un cadre de travail des plus agréables.

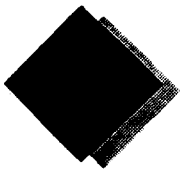
Mr H. GLANC a effectué la reproduction de ces quelques feuillets, et de bien d'autres..., avec une célérité sans égale. Je ne l'en remercierai jamais assez.



Sommaire

Annexes

Introduction



Lancer de Rayon

L'homme a de tout temps cherché à reproduire les spectacles visuels que la nature lui propose, afin de fixer le moment présent sur un support moins *dégradable* que la mémoire, de partager les images avec ses congénères.

Il utilisa tout d'abord le dessin, qui lui permit non seulement la reproduction d'images, mais également leur production. Puis, la photographie donna à chacun la possibilité de saisir l'image de façon fidèle. La cinématographie et la vidéoscopie lui apportèrent en plus la faculté de voir, et revoir, l'image captée avec sa dimension temporelle.

Aussi, lorsque l'homme disposa d'une machine de calcul programmable et d'un dispositif de visualisation dynamique, chercha-t-il naturellement à les utiliser pour répondre à ce besoin. Depuis le début des années 50, il s'est attaché à résoudre le problème, apparemment simple pour l'artiste peintre ou le photographe, de produire une image reflétant le plus exactement possible une réalité visuelle.

Les débuts de l'*imagerie numérique* permettaient la visualisation d'objets tridimensionnels dans un mode dit *fil de fer*, où n'apparaissent que les contours des objets. Les objets complexes étaient découpés en facettes planes, afin de permettre l'utilisation d'algorithmes simples d'élimination des parties cachées. Mais lorsque les *informaticiens de l'image* voulurent colorier les objets tracés, ils se heurtèrent à un problème qui n'était pas du seul domaine de la géométrie.

En effet, la couleur produite par un objet est le résultat, capté par l'oeil humain, des interactions entre les ondes lumineuses et les molécules composants le volume des objets, en fonction de leurs dispositions, comme de leurs natures. La prise en compte d'un tel effet étant bien trop complexe, des modèles empiriques d'*éclairement* furent établis à partir de la fin des années 60. Nous disposons actuellement de modèles permettant la création d'images de qualité visuelle proche de la réalité.

L'un de ces modèles connu sous le nom de *lancer de rayon*[1], ou *tracer de rayon*, permet de rendre les effets de transparence, d'ombres et de réflexion sur les objets. Mais malgré la puissance des machines d'aujourd'hui, l'utilisation de ce procédé demande encore quelques dizaines de minutes de calculs, ce qui reste trop long pour les utilisateurs de l'*infographie*[2], pour lesquels le temps de production d'une image est aussi important que sa qualité.

1. Méthode qui s'apparente au mécanisme de la vision.
2. Outils de création d'images artificielles à des fins artistiques.

Machine Voxel

Comme nous venons de l'évoquer, nous désirons produire une image entièrement calculée par un ordinateur. Pour cela, nous indiquons les objets que nous désirons afficher, leurs caractéristiques (couleur, nature du matériau...) et leur position dans l'espace. A partir de cette définition, la machine doit produire un ensemble de données correspondant à une image. Classiquement, une image est représentée par un ensemble de points colorés, ou *pixels*[1], que l'on affiche sur un écran, un peu à la manière des peintres tachistes.

On dispose donc en début de chaîne d'objets définis, par exemple, mathématiquement (plans, sphères, surfaces gauches...), et l'on obtient un ensemble bidimensionnel de pixels, à afficher. Nous appellerons ce mode de fonctionnement *organisation pixel*, puisqu'il vise avant tout à produire une image 2D[2], mémorisée comme un tableau de points.

Pour les besoins de l'*imagerie médicale*, un nouveau type d'organisation, que nous qualifierons d'*organisation voxel*[3], a été définie. Ces machines, destinées au traitement et à l'affichage d'images obtenues par coupes scanner ou résonance magnétique nucléaire, mémorisent directement les objets sous formes de volumes élémentaires. Ils sont construits par assemblage de *briques*, à la manière des jeux de construction par emboîtement.

Massivement parallèle

Pour répondre aux besoins gigantesques de puissance de calculs (plusieurs millions, voire milliards, d'opérations par seconde), dont les algorithmes d'*infographie* ne sont qu'un exemple, les architectes d'ordinateurs ont développé des machines dites *parallèles*, permettant l'exécution simultanée de plusieurs calculs. Lorsque le degré de parallélisme[4] est inférieur à quelques centaines, on parle de machines *multi-processeurs*, alors que lorsqu'il est supérieur à un millier, la machine est dite *massivement parallèle*.

Dans le premier cas, les machines sont constituées d'un assemblage de processeurs généraux puissants (quelques dizaines, et aujourd'hui centaines, de MIPS[5]), permettant une utilisation souple d'algorithmes complexes. Elles peuvent être utilisées, soit en découpant le problème en tâches différentes, exécutées en parallèle sur chacun des processeurs (à la manière d'une répartition par secteurs d'activité dans une industrie), soit en effectuant la même tâche sur chacun des processeurs, mais avec des données différentes (à la manière d'une répartition de la production au sein d'une équipe).

Dans le second cas de parallélisme, les machines sont constituées d'un assemblage de *cellules* de calculs, généralement de faible puissance et de petite taille, afin de pouvoir en utiliser un très grand nombre pour un coût et un volume raisonnable, mais au dépend de la souplesse de programmation (bien qu'il existe actuellement des projets de *monstres de calculs* possédant plusieurs milliers de processeurs généraux, occupant une surface de quelques dizaines de mètres carrés, et pesant plusieurs tonnes). Nous ne disposons pas encore des outils permettant d'utiliser efficacement ces machines par découpage de tâches; elles sont donc principalement adaptées aux problèmes de complexité simple, présentant une possibilité élevée de découpage de données.

Machine Voxel massivement parallèle pour le lancer de rayon

Dans le cadre du Lancer de Rayon, toutes les manipulations sur les objets se font dans l'espace 3D, la plus coûteuse en temps étant la détermination de la présence d'objets en un point donné de l'espace. Dans le cas de l'organisation pixel, cela oblige à utiliser la définition mathématique de la scène, et nécessite donc des calculs relativement complexes. Par contre, l'organisation voxel permettrait de résoudre très simplement ce problème, puisqu'il suffirait de lire l'emplacement mémoire 3D correspondant au point de l'espace pour déterminer s'il est occupé

1. Picture Element : Elément graphique correspondant à un point écran.
2. Abréviation de Dimension-2. Indique une organisation planaire.
3. Volume Element : analogie, dans l'espace, d'un Pixel.
4. Nombre de calculs exécutables simultanément.
5. Million Instructions Per Seconds - Million d'instructions par secondes.

ou non. Ce mode de représentation de l'espace ne permet plus l'utilisation des algorithmes habituels du monde continu, mais nécessite la mise en oeuvre des méthodes de la géométrie discrète. Nous parlerons dans ce cas de *Lancer de Rayon Discret*.

Enfin, le Lancer de Rayon, de par son fonctionnement intrinsèque, permet un parallélisme, par découpage de données, très élevé. Il autorise même plusieurs types de répartition des données et des calculs.

Nous nous proposons de définir une *Machine Voxel*, pour implémenter la méthode du *Lancer de Rayon*, qui par son organisation *massivement parallèle* permettra de diminuer le temps nécessaire à la création d'une image. Cette machine est appelée **RC**^{2} (*Réseau Cellulaire pour le Ray Casting*[1]).

La définition d'une machine spécialisée sous-entend un certain nombre d'études visant à en déterminer le fonctionnement tant algorithmique qu'architectural. Il faut tout d'abord faire le choix de l'organisation générale de la machine, au vu des tâches que l'on désire y implémenter. Il faut ensuite affiner la description de l'implémentation des algorithmes, pour aboutir à la définition détaillée de l'architecture. C'est la démarche que nous avons adoptée dans ce texte.

Dans une première partie introductive, nous détaillerons les principales méthodes d'éclairements, et plus particulièrement le Lancer de Rayon. Nous indiquerons les différentes propositions existantes pour améliorer le temps de calcul de cette méthode. Nous définirons également les différentes possibilités de fonctionnement des machines parallèles et massivement parallèles.

La deuxième partie sera consacrée à l'étude du fonctionnement du Lancer de Rayon Discret sur une Machine Voxel. Nous définirons plus précisément la notion de Machine Voxel, et nous proposerons et étudierons l'implémentation des différents traitements permettant d'aboutir à l'image synthétisée. Le but de cette partie est d'affiner la spécification de la machine **RC**^{2}.

La troisième et dernière partie conclura ce texte en proposant un résumé des spécifications et un schéma général de la cellule du réseau **RC**^{2}.

1. Ray Casting : Lancer de Rayon.

1ère Partie

Notions de Base

Il nous faut commencer ce document en reprenant les notions qui servent de base au développement du projet RC^2. Celui-ci a pour but la définition d'une machine permettant l'affichage rapide d'images dont le réalisme soit satisfaisant.

Une des méthodes, permettant de prendre en compte un certain nombre des effets naturels d'interaction entre la lumière et les objets d'une scène, est l'utilisation du lancer de rayons. Cependant, comme nous pourrons le voir, ce procédé est assez complexe à mettre en oeuvre et nécessite un grand nombre de calculs.

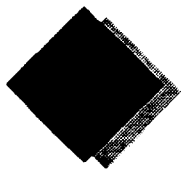
La plupart des implémentations existantes se font sur des machines disposant de processeurs généraux, afin d'en utiliser la souplesse. La puissance de ces processeurs permet, aujourd'hui, de calculer une image en une poignée de minutes. Cependant l'ensemble des calculs doit être réeffectué lors d'un changement de position d'un objet, d'une source lumineuse ou de l'observateur.

Pour tenter de résoudre, ou de diminuer, les phénomènes qui impliquent cette complexité, il nous a semblé intéressant de reprendre la démarche d'un algorithme bien connu, le *tampon en profondeur* (ou *z-Buffer*), proposé par Catmull [Catm75]. Cette méthode a repoussé le problème de l'élimination des parties cachées après le processus de discrètisation, là où l'ensemble des pixels composant les objets est déterminé. Il suffit donc d'une simple comparaison entre les profondeurs des pixels devant apparaître au même endroit pour déterminer celui qui sera visible par l'observateur. Le grand progrès de ce principe est qu'il ne nécessite plus aucun tri en profondeur des objets, et ne pose donc plus aucun problème dans le cas d'objets se chevauchant. Il allie simplicité et puissance, mais au détriment d'un besoin de capacité mémoire importante. Si cela pouvait entraîner quelques réticences en 1975, la capacité et le prix des composants mémoires actuels permettent de disposer d'un tampon en profondeur câblé à moindre frais.

L'application du même principe pour éliminer certains calculs utilisés dans le lancer de rayons nous a amené à étudier l'utilisation de l'organisation voxel. Le parallélisme massif intrinsèque obtenu dans ce cas a permis de définir une machine spécialisée, qui nécessite un très grand nombre de processeurs élémentaires. Si une telle architecture n'a pas encore de réalité commerciale, nous pouvons espérer que les progrès de la technologie la rendront un jour viable, comme il en a été pour le z-Buffer.

Ce travail s'appuyant sur certains algorithmes de synthèse d'images, et sur les principes des machines massivement parallèles, nous présentons ces deux aspects dans ces deux premiers chapitres d'introduction.

CHAPITRE



Méthodes associées à la synthèse d'images réalistes

On peut résumer de manière très simple l'ensemble des traitements permettant d'aboutir à une image de synthèse : il s'agit de calculer la couleur à afficher en chaque pixel d'un écran. Cela passe par la détermination de l'objet visible en un pixel et de la couleur que prend cet objet en fonction de son environnement (sources lumineuses et autres objets).

Nous commençons, dans ce chapitre, par la présentation de modèles empiriques d'éclairement, puis nous présentons une technique de rendu réaliste : le Lancer de Rayon.

Ce dernier est très coûteux en temps de calcul, et plusieurs accélérations algorithmiques ont été proposées, que nous résumons. Enfin, nous exposons les divers schémas de parallélisation de cette méthode de rendu. Nous proposons une caractérisation des gains et accélérations obtenus par les diverses implémentations, afin de dégager les potentialités et avantages d'une approche voxel massivement parallèle.

1.1 Présentation des techniques de rendu

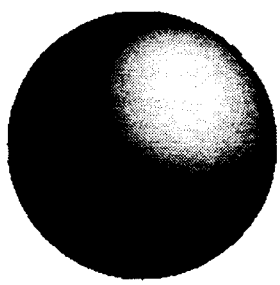
On entend par ce terme l'ensemble des techniques qui permettent de donner un aspect réaliste à une image. Nous n'en présenterons que les plus couramment utilisées.

1.1.1 Modèles empiriques d'éclairement simple

Ces modèles ne prennent en compte que l'interaction entre la lumière et un objet donné. Des lois empiriques ont été proposées pour exprimer les effets lumineux produits. Ce sont ces modèles que nous présentons dans un premier temps.

Expression de l'intensité lumineuse en un point

L'observation d'un objet éclairé permet de déterminer 2 phénomènes d'éclairement :

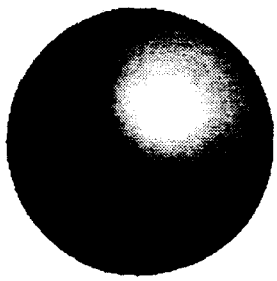


Un objet *mat* présente un éclairement *diffus* diminuant en fonction de l'écart entre la normale au point considéré et la direction de la source lumineuse.

Ce phénomène provient de l'absorption des photons qui provoquent un changement de niveau d'énergie des atomes constituant la surface de l'objet. Pour retrouver leur position d'équilibre, les atomes réémettent un rayonnement qui se trouve à la même longueur d'onde que le rayonnement initial, de par les lois de la physique quantique.

Lorsque les photons ne possèdent pas l'énergie juste suffisante pour créer l'ionisation des atomes de l'objet, ils sont absorbés et leur énergie transformée en chaleur.

La direction d'émission est aléatoire, et l'on obtient donc un éclairement diffus uniformément réparti dans l'espace. La valeur de l'intensité est cependant dépendante de la nature du matériau, de l'angle d'incidence et de la longueur d'onde du faisceau incident.



Un objet *brillant* présente une zone, ou tâche, *spéculaire*, correspondant à la réflexion de la lumière en direction de l'observateur.

Cet effet est dû à une réflexion des photons sur la surface de l'objet, réflexion provoquée par un choc des photons sur les mailles cristallines composant la matière de l'objet. Cet effet n'est possible que si les mailles sont *serrées*, et donc sur les objets *durs* (métaux, verre...).

Suivant l'organisation des mailles, cette réflexion se produit dans une direction unique, ou aléatoirement dans une portion de l'espace autour de cette direction principale. Visuellement, cela se traduit par une tâche claire plus ou moins étendue.

De manière générale, les deux effets se superposent en proportions plus ou moins importantes. L'intensité résultante en un point d'un objet est donc donnée par :

$$I_{tot} = I_{diff} + I_{spec} \qquad \textbf{(Eq. 1.1)}$$

avec I_{diff} : Intensité de lumière diffuse.
I_{spec} : Intensité de lumière spéculaire.

Dans le cas où l'objet est éclairé par plusieurs sources, on effectue une sommation des intensités produites par chacune des sources :

$$I_{tot} = \sum_{n=1}^{nbl} [I_{diff}(n) + I_{spec}(n)] \qquad \textbf{(Eq. 1.2)}$$

avec nbl : Nombre de sources lumineuses.
$I_x(n)$: Intensité produite par la source n.

Différentes expressions de l'intensité diffuse et spéculaire ont été proposées. Nous utiliserons les notations de la Figure 1.1 pour les décrire.

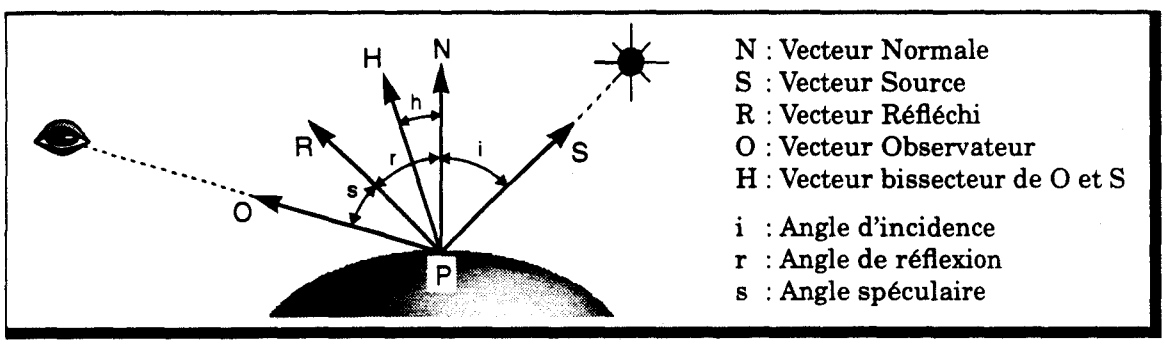


Figure 1.1 ***Notations utilisées pour les équations d'éclairement.*** *Les vecteurs S, N et R sont coplanaires, de même pour O, H et S*

Eclairement Diffus

Lambert

La loi régissant l'éclairement diffus a été définie par les spécialistes de l'optique géométrique par la *loi de Lambert*, également appelée *loi cosinus* :

$$I_{diff} = k_d \cdot \cos(i) \cdot I_S = k_d \cdot (\hat{N} \bullet \hat{S}) \cdot I_S \qquad \textbf{(Eq. 1.3)}$$

avec k_d : Coefficient de réflexion diffuse, permettant de définir la proportion d'intensité diffuse.
I_S : Intensité de la source lumineuse.

L'intensité produite par cette loi est maximale pour $i = 0$, et diminue au fur et à mesure que l'angle i augmente. Elle est indépendante de la position de l'observateur; l'émission est donc bien uniforme dans le demi-espace éclairé.

Atténuation

Warnock [Warn69] a utilisé cette loi en y incluant une diminution de l'intensité en fonction de la distance d_O entre l'observateur et l'objet (Eq. 1.4). De cette manière, des objets identiques situées à des profondeurs différentes ne sont plus confondus.

$$I_{diff} = k_d \cdot \frac{\cos(i)}{d_O} \cdot I_S \qquad \textbf{(Eq. 1.4)}$$

D'après les lois physiques, l'atténuation d'intensité est fonction du carré de la distance. Il faudrait donc utiliser :

$$I_{diff} = k_d \cdot \frac{\cos(i)}{d_O^2} \cdot I_S$$

Cependant la variation induite est trop faible pour les sources situées à grande distance, et trop importante pour les sources proches. L'atténuation suivante est souvent préférée [Fole90] :

$$I_{diff} = k_d \cdot \frac{\cos(i)}{\max\left(c_1 + c_2 d_O + c_3 d_3^2,\ 1\right)} \cdot I_S$$

c_1 permet de garder une valeur non nulle; la valeur est minorée à 1, pour garantir qu'il y a toujours atténuation.

Eclairage ambiant

En général, un objet est situé dans une scène au sein de laquelle se produisent de nombreuses réflexions (sur les murs, les sols...). Il n'est pas possible de modéliser facilement ces sources de lumière *parasite* qui éclairent uniformément les objets. On utilise donc un terme constant d'intensité ambiante, appliqué à l'ensemble de l'objet :

$$I = k_a \cdot I_{amb} \qquad \textbf{(Eq. 1.5)}$$

Eclairement Spéculaire

Warnock

Pour rendre l'aspect de l'éclairement spéculaire, Warnock [Warn69] a proposé d'utiliser un $\cos^n(i)$. Cette approximation permet de *régler* la tâche spéculaire à l'aide de la valeur de l'exposant spéculaire n. Elle a cependant l'inconvénient d'être indépendante de la position de l'observateur. Or, lorsque l'on regarde un objet brillant, la zone spéculaire se déplace en fonction de notre position. La proposition de Warnock n'est valable, mathématiquement, que dans le cas où l'observateur et la source sont situés dans la même direction.

Phong

Phong [Phon75] a repris les travaux de Warnock, et a défini l'intensité spéculaire par :

$$I_{spec} = W(i) \cdot \cos^n(s) \cdot I_S = W(i) \cdot (\vec{R} \bullet \vec{O})^n \cdot I_S \qquad \textbf{(Eq. 1.6)}$$

avec $W(i)$: fonction de réflectance spéculaire.
n : exposant spéculaire.

La fonction de réflectance est une caractéristique du matériau composant l'objet (Figure 1.2). Elle indique l'importance de l'effet spéculaire en fonction de l'angle d'incidence.

En général, on remplacera la fonction de réflectance difficile à calculer par une simple constante :

$$I_{spec} = k_s \cdot (\vec{R} \bullet \vec{O})^n \cdot I_S$$

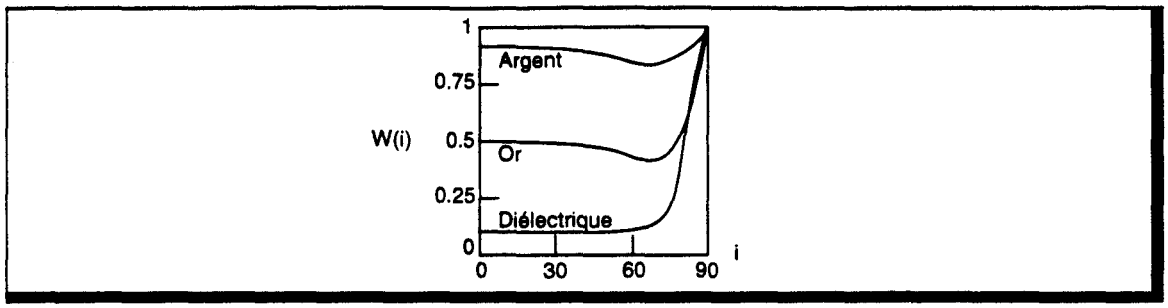


Figure 1.2 ***Courbes de réflectance spéculaire.*** *Valeur de la fonction de réflectance en fonction de l'angle d'incidence*

Calcul du vecteur réfléchi $\vec{R}$

De par la loi de Descartes, les vecteurs $\vec{S}$, $\vec{N}$ et $\vec{R}$ sont situés dans le même plan. On a donc $\vec{R} = \alpha\vec{S} + \beta\vec{N}$. De plus, l'angle de réflexion r est égal à l'angle d'incidence i.

Les conditions précédentes permettent de définir $\vec{R}$ par l'équation [Glas89] :

$$\vec{R} = 2\vec{N} \cdot (\vec{N} \bullet \vec{S}) - \vec{S}$$

Blinn

Le vecteur réfléchi $\vec{R}$ est indépendant de la position de l'observateur, mais son calcul est coûteux. Blinn [Blin77] a proposé d'utiliser un autre vecteur $\vec{H}$, bissecteur de $\vec{O}$ et $\vec{S}$, appelé direction d'éclairement maximum. En effet, l'intensité spéculaire est maximale lorsque $\vec{H}$ coïncide avec $\vec{N}$, c'est à dire lorsque $\vec{O}$ coïncide avec $\vec{R}$.

On a alors : $\vec{H} = \frac{\vec{O} + \vec{S}}{2}$, et $I_{spec} = W(i) \cdot \cos^m(h) \cdot I_S = W(i) \cdot (\vec{O} \bullet \vec{H})^m \cdot I_S$

Remarque : L'angle h n'est pas identique à l'angle s. Par conséquent la valeur de m ne peut être égale à la valeur de n (exposant spéculaire défini par Phong).

Prise en compte de la couleur

Les valeurs d'éclairements, telles que nous venons de les définir, sont en fait des fonctions spectrales, c'est à dire qu'elles définissent une valeur d'intensité pour chaque longueur d'onde. L'équation d'intensité (Eq. 1.1) devient alors :

$$I_{tot}(\lambda) = I_{diff}(\lambda) + I_{spec}(\lambda) \qquad \textbf{(Eq. 1.7)}$$

Les études photométriques et psychophysiques de l'oeil ont montré une sensibilité différente de celui-ci dans trois zones du spectre visible : le bleu, le vert et le rouge (ce qui est la base du choix de fonctionnement des écrans couleurs à tubes cathodiques) [Fole90]. Les mesures effectuées ont permis de construire trois fonctions d'appariement, définissant les puissances nécessaires sur chaque primaire pour couvrir le spectre visible (Figure 1.3). Par mélange de ces trois primaires, il est possible de reproduire l'*impression* visuelle d'une couleur de densité spectrale donnée.

Soit $P(\lambda)$ le spectre de la couleur à afficher. On calculera les valeurs des trois primaires par convolution avec les fonctions d'appariement :

$$R = \int_{-\infty}^{\infty} P(\lambda)\, r_\lambda\, d\lambda, \quad V = \int_{-\infty}^{\infty} P(\lambda)\, v_\lambda\, d\lambda, \quad B = \int_{-\infty}^{\infty} P(\lambda)\, b_\lambda\, d\lambda \qquad \textbf{(Eq. 1.8)}$$

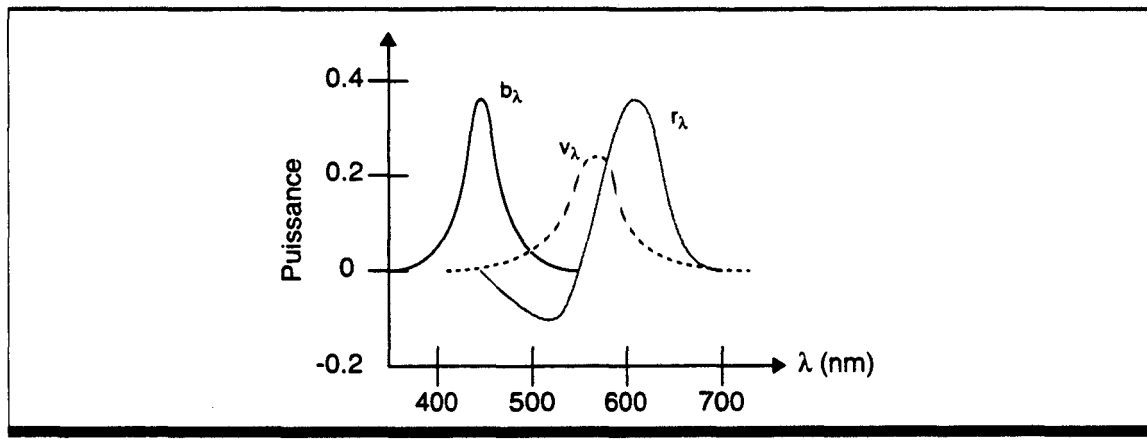


Figure 1.3 ***Fonctions d'appariement.***

Expression simplifiée de la couleur

L'utilisation de fonctions de densité spectrale étant bien trop coûteuse, on effectue en général une discrètisation de celles-ci, en effectuant les calculs pour un certain nombre de longueur d'ondes, les convolutions (Eq. 1.8) étant effectuées pour déterminer les valeurs des trois primaires d'affichage.

L'expression minimale de la couleur (qui est la plus fréquemment utilisée) consiste à ne calculer le spectre de la couleur qu'en trois longueurs d'ondes (rouge, vert et bleu) et d'utiliser des fonctions de Dirac en remplacement des trois fonctions d'appariement, ce qui élimine les calculs de convolutions.

On calculera donc un triplet $[I_R\ I_V\ I_B]$ pour chaque point à afficher, plutôt que de calculer la fonction $I(\lambda)$. Pour des modèles de rendu simple, tels que ceux présentés ci dessus, l'utilisation de plus de 3 longueurs d'onde ne permet pas d'augmenter sensiblement la qualité de l'image, ce qui n'est pas le cas avec des modèles plus évolués.

Eclairement diffus

Nous avons vu que le phénomène de diffusion est tributaire de l'énergie nécessaire à l'ionisation d'une molécule. Or l'énergie d'un photon est donné par :

$$E = h \cdot f = h \cdot \frac{c}{\lambda}$$

avec
- E : Energie en Joules.
- h : Constante de Planck (~ 6.63×10^{-34} J.s).
- f : Fréquence du photon (s^{-1}).
- c : Vitesse de la lumière dans le vide (~ 3×10^{8} $m.s^{-1}$).
- λ : Longueur d'onde du photon (m^{-1}).

Cette énergie est proportionnelle à la longueur d'onde du photon, c'est à dire à la couleur du rayonnement qu'il génère.

La couleur visible d'un objet (rouge, par exemple), éclairé par une source de lumière blanche, est par conséquent due à la rediffusion des photons de longueur d'onde *rouge* (ou, plus exactement, dans le spectre de lumière rouge), car ce sont les seuls possédant l'énergie suffisante pour ioniser les molécules de l'objet.

De plus, pour que notre objet puisse apparaître rouge, il est nécessaire que la source qui l'éclaire émette dans le rouge. Dans le cas contraire, il paraîtra noir, puisqu'il absorbe toutes les autres longueurs d'ondes.

L'intensité de l'éclairement diffus est donc fonction du spectre de la source, et d'une fonction de réflectance diffuse. Cette dernière sera approximée par un triplet $[C_R\ C_V\ C_B]$, représentant la couleur de l'objet, c'est à dire le pourcentage de chacune des composantes de couleurs R, V, B; la répartition spectrale de la source est notée $[I_{sR}\ I_{sV}\ I_{sB}]$, l'intensité de la source ambiante étant $[I_{ambR}\ I_{ambV}\ I_{ambB}]$.

On obtient donc :

$$I_{diff_\lambda} = k_a \cdot C_\lambda \cdot I_{amb\lambda} + k_d \cdot C_\lambda \cdot I_{s\lambda} \cdot (\vec{N} \bullet \vec{S}) \text{ , pour chaque couleur primaire}$$

Eclairement Spéculaire

Dans le cas spéculaire, il n'y a pas d'absorption et rediffusion de photons. L'énergie de celui-ci, et donc la couleur associée, n'entre de ce fait pas en compte. On utilisera un triplet $[k_{sR}\ k_{sV}\ k_{sB}]$, pour approximer le pouvoir spéculaire de la matière.

On obtient alors :

$$I_{spec_\lambda} = k_{s\lambda} \cdot I_{s\lambda} \cdot (\vec{R} \bullet \vec{O})^n, \quad \text{pour } \lambda = R, V, B$$

Equation générale

En prenant en compte tous les résultats précédents, l'équation d'intensité (Eq. 1.1) devient :

$$I_{tot\lambda} = k_a \cdot C_\lambda \cdot I_{amb\lambda} + I_{s\lambda} \cdot \left[k_d \cdot C_\lambda \cdot (\vec{N} \bullet \vec{S}) + k_{s\lambda} \cdot (\vec{R} \bullet \vec{O})^n\right], \quad \text{pour } \lambda = R, V, B \qquad \textbf{(Eq. 1.9)}$$

Amélioration de la modélisation des sources lumineuses

Dans les paragraphes précédents, les sources étaient considérées comme ponctuelles et rayonnant uniformément dans tout l'espace. Les seuls contrôles possibles sont leur position et leur intensité.

Sources ponctuelles directionnelles

Warn [Warn83] a proposé un modèle permettant de contrôler la direction et la concentration d'une source lumineuse, comme le fait un photographe en studio. Il est possible, de cette façon, d'obtenir les effets d'éclairage désirés pour mettre en valeur un objet.

Warn utilise une source ponctuelle fictive (Figure 1.4) produisant une réflexion spéculaire en un point d'un plan réflecteur, orienté perpendiculairement à la direction de cette source.

En utilisant le modèle spéculaire de Phong, on obtient une intensité émise dans la direction $-\vec{S}$ valant :

$$I_{s\lambda} = I_{S_V\lambda} \cdot \cos^p(l) = I_{S_V\lambda} \cdot (\vec{L} \bullet -\vec{S})^p$$

puisque $\vec{L}$ est confondu avec $\vec{N}$, et donc $\vec{R}$, sur le plan réflecteur.

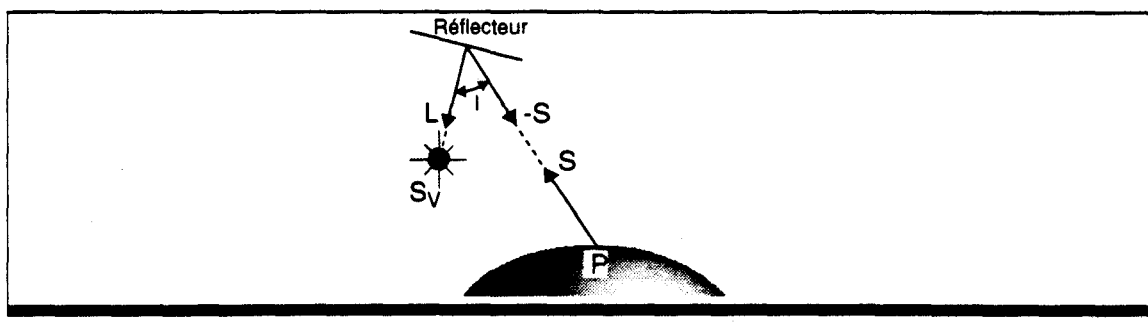


Figure 1.4 ***Modèle de Warn.*** *La source S_V est fictive.*

On peut également représenter l'ensemble source fictive-point réflecteur par une source directionnelle (Figure 1.5). La zone grisée représente la distribution spatiale de l'intensité de la source, en fonction de l'angle l entre la direction principale et la direction du point éclairé. Elle est fonction de l'exposant spéculaire p du réflecteur. La valeur de cet exposant permet d'ajuster

la concentration du faisceau, pour obtenir un éclairage du type *spot* (cône étroit) ou du type *flood* (cône large).

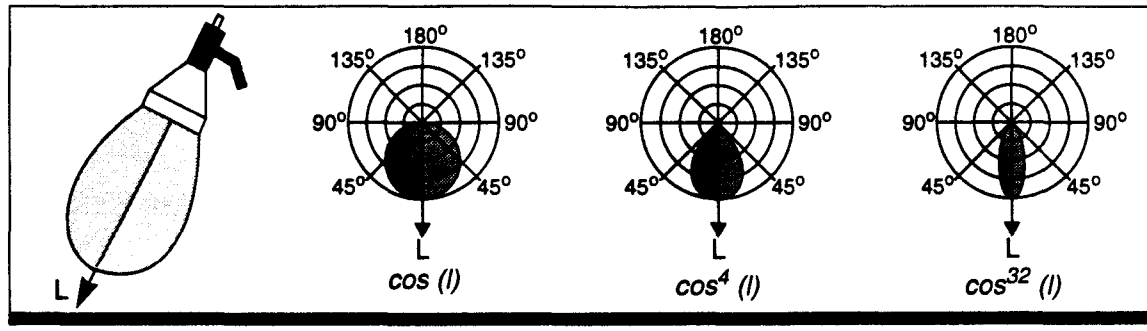


Figure 1.5 ***Modèle équivalent de source directionnelle.*** *Distribution spatiale de l'intensité en fonction de la valeur de p [Fole90].*

Sources non ponctuelles

D'autres modèles ont été proposés pour prendre en compte les sources non ponctuelles, qui sont plus réalistes, tels les tubes néons ou les surfaces lumineuses [Verb84][Nish85][Poul90]. Ces sources génèrent également des zones de pénombres dont il faut intégrer l'effet dans le modèle d'éclairement.

L'intensité en un point éclairé par une source non ponctuelle nécessite de calculer l'intégrale sur les éléments surfaciques de la source :

$$I = \int_{\text{Surface}} I_x \left[k_d \cdot (\vec{N} \bullet \vec{L}_x) + k_s \cdot (\vec{O} \bullet \vec{R}_x)^n \right] dx$$

Verbeck [Verb84] propose de remplacer la source non ponctuelle par un ensemble de sources ponctuelles, ce qui permet de remplacer l'intégrale par une somme. Cette technique nécessite d'utiliser un grand nombre de sources pour éviter les problèmes de sous échantillonnage.

Dans certains cas particuliers (sources linéaires dirigées parallèlement ou perpendiculairement à la surface éclairée), il existe une solution analytique simple à l'intégrale [Nish85]. Mais dans le cas général, la résolution nécessite le calcul d'une intégrale d'un cos^n [Poul90].

1.1.2 Modèle basé sur les lois physiques

Les modèles précédents sont basés sur l'observation visuelle de l'éclairement d'un objet, et considèrent l'effet spéculaire comme un effet miroir sur une surface lisse. Pour améliorer le rendu d'objets rugueux, un modèle plus proche de la physique a été défini.

Ce modèle est basé sur l'étude du transfert d'énergie lumineuse se produisant entre deux surfaces. Il définit l'énergie que peut émettre une surface en fonction de l'intensité de lumière qu'elle reçoit depuis une source lumineuse. Une fonction appelée *réflectance bidirectionnelle* est utilisée pour exprimer l'intensité réfléchie par la surface en fonction de l'énergie qu'elle possède (pour plus de détails, voir l'annexe A).

On peut définir ce modèle sous la forme simplifiée :

$$I_R = \rho \cdot I_S \cdot (\vec{N} \bullet \vec{S})$$

avec : I_R : Intensité réfléchie par la surface
I_S : Intensité reçue depuis la source
ρ : Fonction de réflectance bidirectionnelle

La réflectance bidirectionnelle représente, en fait, la manière dont une surface réémet la lumière. Elle est donc fonction des caractéristiques optiques du matériau. Cette fonction peut être approximée par deux composantes, une composante diffuse et une composante spéculaire :

$$\rho = k_d \cdot \rho_d + k_s \cdot \rho_s$$

Différentes expressions de ρ_s ont été proposées [Blin77][Cook81], basées sur les travaux de Torrance et Sparrow [Torr67].

Ces modèles permettent d'obtenir des effets lumineux proches de la réalité, mais au prix de calculs très complexes.

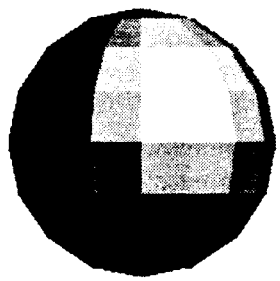
1.1.3 Ombrage[1] d'objets facettisés

L'utilisation des modèles d'éclairement nécessite la connaissance du vecteur normal au point à éclairer. Or, pour simplifier les traitements d'affichage, la plupart des systèmes graphiques convertissent les objets en un ensemble de facettes planes. Il n'est alors plus possible de connaître la normale exacte en tout point de l'objet.

Trois méthodes ont été proposées pour ombrer un objet facettisé. La première calcule une intensité constante pour chaque facette. Les deux autres approximent l'éclairement par interpolation, dans le but de *lisser* l'ombrage de l'objet.

Ombrage polygonal

Principe



Le découpage en facettes de l'objet étant défini, il est possible de déterminer la normale à chaque polygone. On peut alors appliquer une méthode d'éclairement au centre de chaque facette. L'intensité obtenue est appliquée à toute la facette.

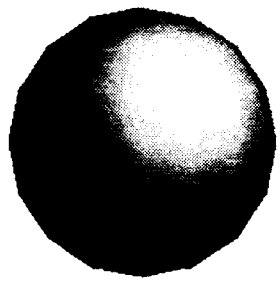
Cet ombrage n'est exact que sous les deux conditions suivantes :
- l'objet est réellement constitué de facette.
- la source et l'observateur sont situés à l'infini (auquel cas, les vecteurs source et observateur sont constants sur tout l'objet).

Inconvénient

L'inconvénient majeur évident de cette méthode d'ombrage est de laisser apparaître le maillage dans le cas d'objets facettisés.

Ombrage de Gouraud

Principe



Pour rendre l'ombrage continu (de degré C^0) sur l'objet facettisé, Gouraud [Gour71] fait les 2 remarques suivantes :
- l'intensité calculée en un sommet doit être la même pour toutes les facettes contenant ce sommet. Il faut pour cela utiliser la même valeur de normale. Gouraud propose d'effectuer la moyenne des normales aux surfaces adjacentes au sommet, ou de calculer la vraie normale dans le cas où cela est possible (la définition mathématique de l'objet étant connue en ce point). Cette normale permet de calculer une couleur *exacte* en chaque sommet
- l'intensité au sein d'une facette sera obtenue par interpolation afin d'assurer une continuité de l'ombrage.

1. L'ombrage correspond au calcul de l'intensité lumineuse en tout point de l'objet.

Utilisation de l'éclairement spéculaire

L'ombrage de Gouraud s'effectuant par interpolation d'intensité, il n'est valable que lorsque la variation des intensités aux sommets est relativement faible. En particulier, cette méthode ne peut traiter correctement l'utilisation de l'éclairement spéculaire, car il induit une forte variation sur une faible surface. Suivant les positions de l'objet, de la source et de l'observateur, le reflet spéculaire peut-être occulté ou au contraire amplifié (Figure 1.6).

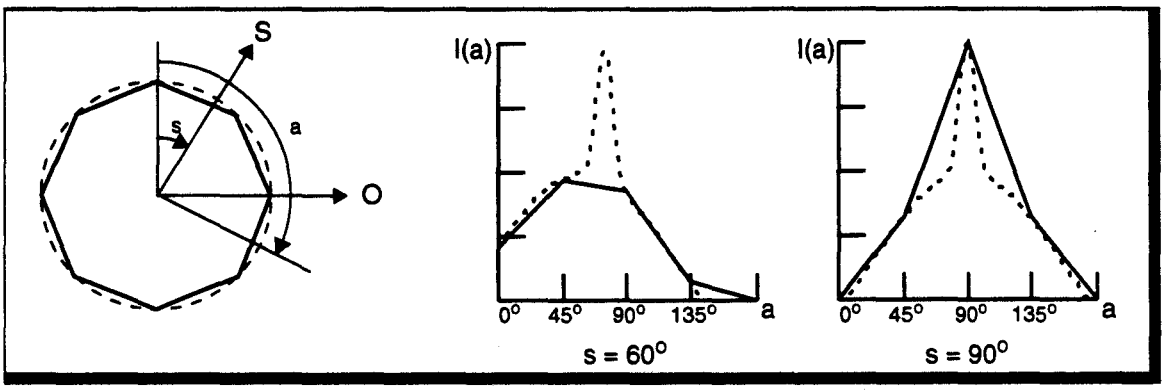
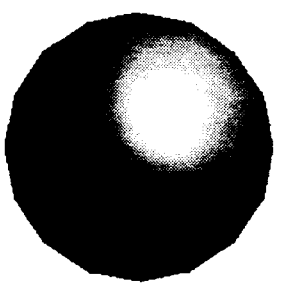


Figure 1.6 ***Ombrage de Gouraud avec effet spéculaire.*** *Eclairement obtenu sur la sphère discrétisée, en fonction de la position de la source. La courbe pointillée représente l'ombrage que l'on obtiendrait sur une vraie sphère.*

Cet effet peut cependant être diminué en utilisant des facettes de petites tailles, et donc en facettisant finement les objets.

Ombrage de Phong

Principe



Pour calculer correctement la valeur de l'intensité spéculaire, il est nécessaire de connaître le vecteur normal en tout point.

Phong [Phon75] propose d'utiliser une double interpolation linéaire sur les normales des sommets, pour obtenir le vecteur normal en tout point de la facette.

La formule d'éclairement de Phong (incluant le spéculaire) est ensuite calculée en tout point de la facette.

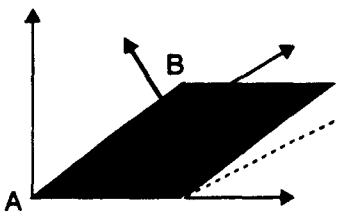
Inconvénients de l'interpolation

L'utilisation de l'interpolation, que ce soit de couleurs ou de normales, présente un certain nombre d'inconvénients. Nous citerons :

Distorsion de perspective

Cette anomalie est due au fait que l'interpolation est effectuée après la transformation perspective de l'objet, et non dans l'espace de définition de l'objet. En effet, cette transformation ne conserve pas la notion de distance (Figure 1.7). On comprend bien qu'en perspective, il y a *tassement* des distances au fur et à mesure que l'on s'éloigne de l'observateur. Cet effet ne peut pas être pris en compte par une interpolation linéaire. Il n'est cependant perceptible que sur des facettes de grande taille s'étendant vers le point de fuite de la projection. Il est par conséquent possible de diminuer les effets de distorsion en limitant la taille des facettes.

Ce problème est résolu en effectuant les interpolations en coordonnées homogènes (espace 4D (x, y, z, w)), la mise en perspective étant effectuée après l'interpolation (par division par w).



Soient les points A et B, de la figure, de coordonnées :

$$A = [0\ 0\ 0] \text{ et } B = [0\ 10\ 100] .$$

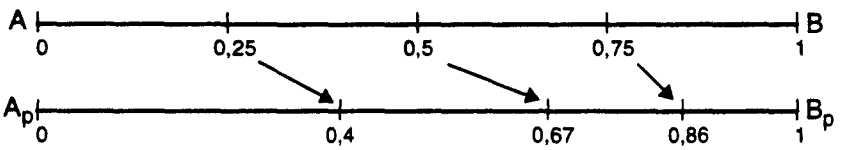
Les formules de transformation perspective sont :

$$X_p = (D \cdot X)/(D+Z) , \quad Y_p = (D \cdot Y)/(D+Z) \text{ et}$$
$$Z_p = (D \cdot Z)/(D+Z)$$

En prenant D, distance Observateur, à 100, on obtient les projections de A et B :

$$A_p = [0\ 0\ 0] \text{ et } B_p = [0\ 5\ 50]$$

En calculant les projetés des points situés au quart, à la moitié et au trois-quart du segment AB, on obtient :



(Les chiffres situés en dessous des segments représentent les positions relatives des points sur le segment AB)

Figure 1.7 ***Tassement des distances en projection perspective.***

Dépendance à l'orientation

L'interpolation effectuée sur un polygone comporte l'inconvénient d'être dépendante de l'orientation de celui-ci. Sur la Figure 1.8.a, l'intensité au point P est $I_P = (I_B + I_D)/2$, et sur la Figure 1.8.b, elle vaut $I_p = (I_A + I_C)/2$, ce qui est certainement une valeur différente.

La solution est d'utiliser des facettes triangulaires plutôt que des polygones, l'interpolation étant alors indépendante de l'orientation.

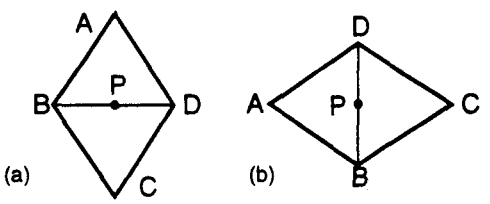


Figure 1.8 ***Dépendance de l'interpolation à l'orientation du polygone.***

Malgré ces inconvénients, les facettes sont toujours largement utilisées, car elles permettent d'approximer toutes les surfaces gauches, et, surtout, elles sont facilement et rapidement affichables.

1.1.4 Effets inter-objets

Les modèles précédents prennent en compte l'effet que produit une source lumineuse lorsqu'elle éclaire la surface d'un objet.

Lorsque plusieurs objets sont éclairés, d'autres effets apparaissent :

- l'existence d'ombres, dues au fait qu'un objet peut masquer, totalement ou partiellement, une source lumineuse, et donc empêcher qu'un autre objet soit éclairé,
- la transparence, qui permet de voir un objet à travers un autre,
- la réflexion, un objet pouvant être vu de façon indirecte par réflexion sur un autre objet.

Des méthodes ont été proposées pour chacun de ces effets. La technique du lancer de rayon prenant directement et implicitement en compte ces phénomènes, nous ne les détaillerons pas ici. Nous en expliquerons simplement les causes et les effets.

Ombres portées

Un des facteurs permettant à l'homme d'estimer les positions des objets dans l'espace est l'existence des ombres générées par les objets les uns sur les autres. Il est donc important de les inclure dans le processus de synthèse d'une image, pour la rendre réaliste et compréhensible.

Il existe deux types d'ombres, suivant le type de la source les produisant :

1. Les sources ponctuelles, et les sources situées *à l'infini*, créent des ombres au contour franc, correspondant à la projection de l'objet éclairé sur l'objet à l'ombre (Figure 1.9.a).
2. Les sources non ponctuelles, ou deux sources proches l'une de l'autre, créent une zone de pénombre autour d'une zone d'ombre (Figure 1.9.b). Cette zone provient du fait qu'une partie seulement de la lumière émise atteint la surface.

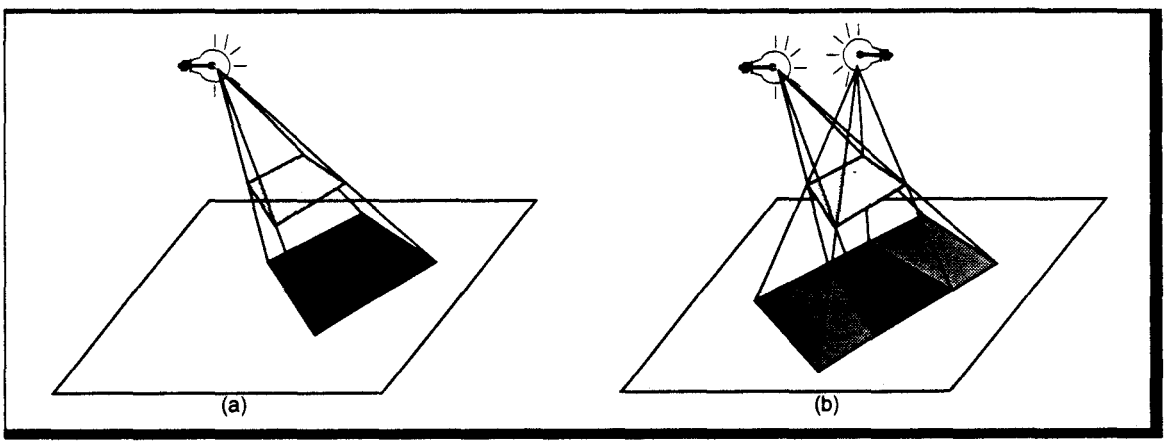


Figure 1.9 ***Ombres portées.*** *(a)- Ombre créée par une source ponctuelle. (b) Ombre + Pénombre créées par deux sources ponctuelles.*

L'équation d'éclairement devient :

$$I_{tot} = I_{amb} + \sum_{l} S_l \cdot [I_{diff}(l) + I_{spec}(l)] \qquad \textbf{(Eq. 1.10)}$$

avec l : Indice sur les sources lumineuses,
S_l : Drapeau d'ombrage :
$S_l = 0$: point à l'ombre, $S_l = 1$: point éclairé.

Transparence

La transparence est un effet moins indispensable, mais qui permet d'améliorer le réalisme lorsqu'elle est utilisée.

Transparence réfractive

La transparence fait intervenir les lois de réfraction de l'optique géométrique (Figure 1.10) définies par Descartes : lorsqu'un rayon passe d'un milieu A à un milieu B, il subit une déviation, fonction des indices de réfraction des 2 milieux, définie par :

$\vec{L}, \vec{N}, \vec{T}$ sont coplanaires, et $n_b \cdot \sin(t) = n_A \cdot \sin(i)$

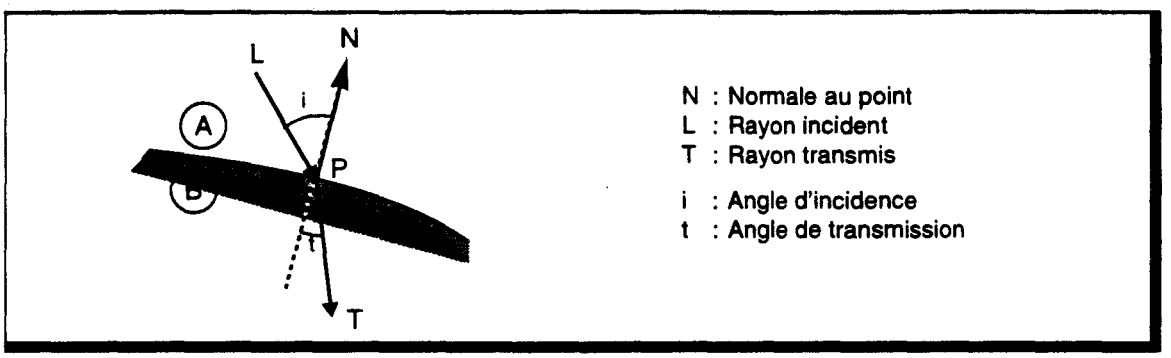


Figure 1.10 ***Géométrie de la réfraction.***

L'intensité vue au point P est obtenue par [Whit80] :

$$I_{tot} = I_{loc} + k_t \cdot I_T \qquad \textbf{(Eq. 1.11)}$$

avec I_{loc} : Intensité locale, telle que définie par le modèle d'éclairement (Eq. 1.9),

k_t : Coefficient de transparence du milieu B,

I_T : Intensité provenant de la direction du rayon transmis.

Remarques :

- L'indice de réfraction d'un milieu est fonction de la longueur d'onde du rayon incident[1].
- Si $n_b < n_A$, alors on a $i < t$. Il y a donc un angle d'incidence maximum appelé *incidence critique*, obtenu lorsque $t = \pi/2$: $i_C = \text{asin}(n_B/n_A)$. Lorsque l'incidence est supérieure à l'incidence critique, il se produit une réflexion interne dans le milieu A, et non une réfraction.

La réfraction est difficile à implémenter dans le cas d'un rendu classique, car il faut modifier la trajectoire de la direction de visée (Figure 1.11) et calculer l'intensité du point vu par transparence, qui peut lui-même appartenir à un objet transparent...

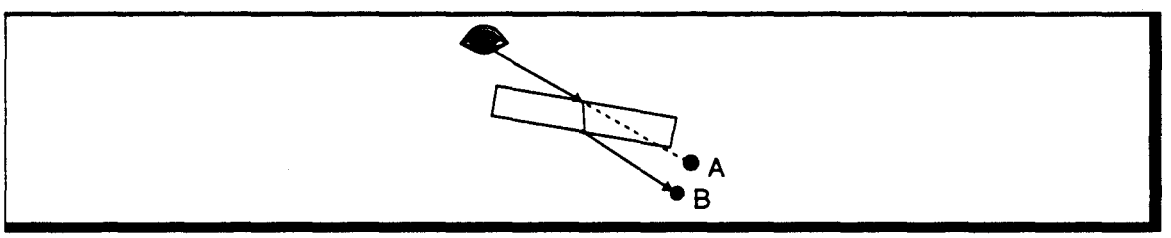


Figure 1.11 ***Modification de trajectoire.*** *De par la réfraction, c'est l'objet au point B qui est vue par transparence, et non l'objet au point A.*

Transparence non réfractive

Pour permettre l'utilisation de transparence dans un système de rendu classique par projection, on effectue la simplification consistant à considérer qu'il n'y a pas de modification de la direction des rayons lumineux lors de la traversée d'un objet transparent. Cette supposition n'est vraie que dans le cas de l'incidence normale[2], ou lorsque $n_B = n_A$, ou également lorsque l'objet traversé est d'épaisseur négligeable.

1. Ce qui crée la décomposition spectrale lorsqu'un faisceau de lumière blanche traverse un prisme.
2. $i = 0$

L'intensité vue est alors obtenue par interpolation entre l'intensité de l'objet transparent et l'intensité de l'objet vu par transparence (Figure 1.12) :

$$I_{tot} = (1 - k_{t1}) \cdot I_1 + k_{t1} \cdot I_2$$

avec k_{t1} : Coefficient de transparence du matériau 1.

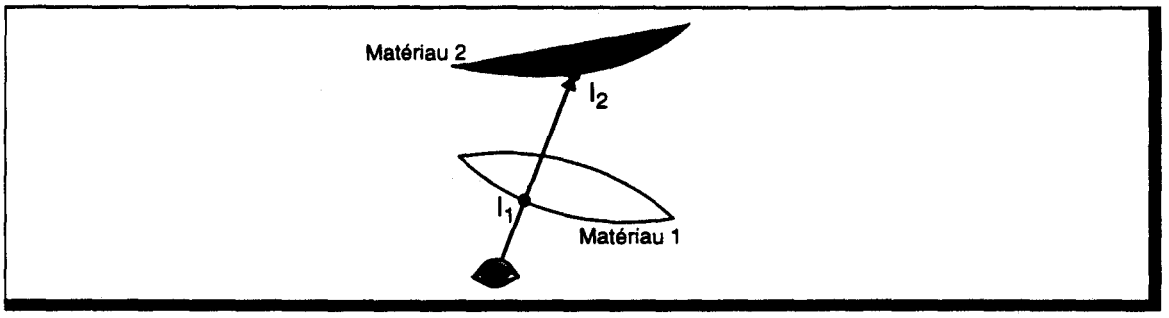


Figure 1.12 ***Transparence non réfractive.*** *Le rayon n'est pas dévié.*

Réflexion

La réflexion d'un objet sur un autre répond également aux lois de l'optique géométrique. Selon les lois de Descartes (Figure 1.13) :

$\vec{L}, \vec{N}, \vec{R}$ sont coplanaires, et $r = i$

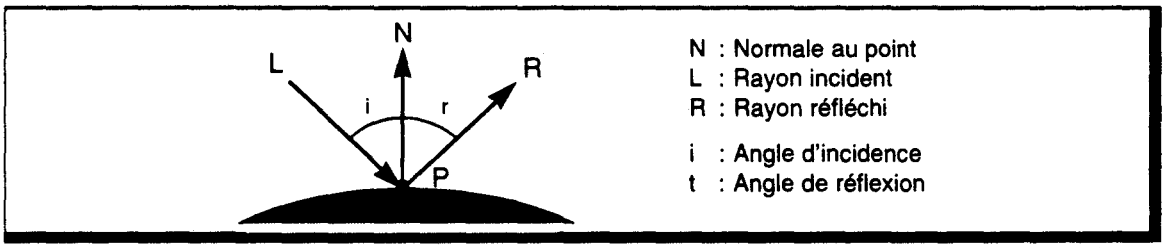


Figure 1.13 ***Géométrie de la réflexion.***

L'intensité vue au point P est obtenue par [Whit80] :

$$I_{tot} = I_{loc} + k_r \cdot I_R \qquad \textbf{(Eq. 1.12)}$$

avec I_{loc} : Intensité locale, telle que définie par le modèle d'éclairement (Eq. 1.9),

k_r : Coefficient de réflexion de l'objet,

I_R : Intensité provenant de la direction du rayon réfléchi.

L'utilisation de la réflexion n'est pas possible facilement dans le cas d'un schéma de rendu classique, pour les mêmes raisons que celles évoquées dans le cas de la transparence.

1.1.5 Technique avancée de rendu : Lancer de Rayon

Dès les débuts de la synthèse d'image, les chercheurs en infographie ont essayé de calquer le fonctionnement de l'oeil humain en appliquant les techniques des opticiens, qui utilisent la notion de rayons lumineux dont ils suivent les trajectoires dans l'espace. Ils se sont heurtés au manque de puissance des machines de l'époque, et ont plutôt essayé de résoudre les problèmes un à un. Un ensemble d'algorithmes a alors vu le jour, pour éliminer les parties cachées, effectuer le calcul d'ombre, de transparence... Cependant ces techniques ne sont généralement pas utilisables sur des scènes quelconques, et elles sont de plus difficilement intégrables l'une avec l'autre.

L'augmentation de la puissance des machines a vu renaître la technique du lancer de rayon, au début des années 80, qui est devenu un des outils de synthèse les plus répandus.

Cependant, quelques effets d'interaction lumineuse ne peuvent être traités par cette technique, ce qui a amené l'utilisation d'une autre classe de méthodes, appelée *radiosité* [Gora84], plus proche de la réalité physique de l'éclairement. Pour cela un calcul d'échange d'énergie lumineuse est effectué entre tous les objets composant la scène. Nous ne détaillerons pas ce principe.

Présentation du lancer de rayon

Le lancer de rayon est une méthode de rendu en marge de la technique classique de rendu par projection (pour laquelle on convertit les objets en pixels avant de calculer leur éclairement). Pour le lancer de rayon, il n'y a pas de conversion explicite des objets. Seule une information d'éclairement est recherchée en tout pixel de l'écran.

Principe

Le premier *capteur d'images*, créé par l'homme, fut l'appareil photographique. Le lancer de rayon est basé sur une simulation du fonctionnement de celui-ci [Glas89].

Un film photographique est placé dans le fond d'une enceinte opaque (Figure 1.14). Lorsque la lumière atteint le film, il se produit une modification chimique de l'émulsion photosensible déposée sur le film. L'utilisation d'un produit révélateur puis d'un produit fixateur permet l'obtention de l'image sur le film.

Si le film était placé directement devant la scène (lorsqu'il n'y a pas de face avant à l'enceinte), un point de l'émulsion recevrait de la lumière depuis l'ensemble de l'espace l'environnant. L'impact de tous ces rayons lumineux résulteraient en une teinte grisâtre. L'image formée seraient donc complètement grise.

Pour éviter cela, on perce la face avant de l'enceinte, afin de réaliser un cache ne permettant qu'à un seul rayon, ou tout du moins à un nombre très restreint de rayons, de parvenir à un point donné du film. Sur les appareils photographiques actuels, cet orifice est remplacé par un objectif réalisant une focalisation des rayons lumineux.

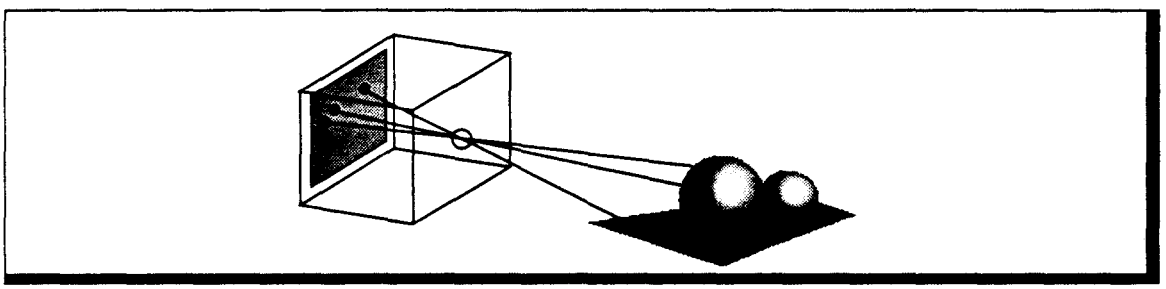


Figure 1.14 ***Modèle de l'appareil photographique.*** *Ce modèle est la base du lancer de rayon.*

Dans le cas du Lancer de Rayon., il est plus simple de placer le plan du film devant l'oeil, ce dernier remplaçant le trou de visée (Figure 1.15). On garde, avec ce principe, la nécessité de n'avoir qu'un rayon par point du film.

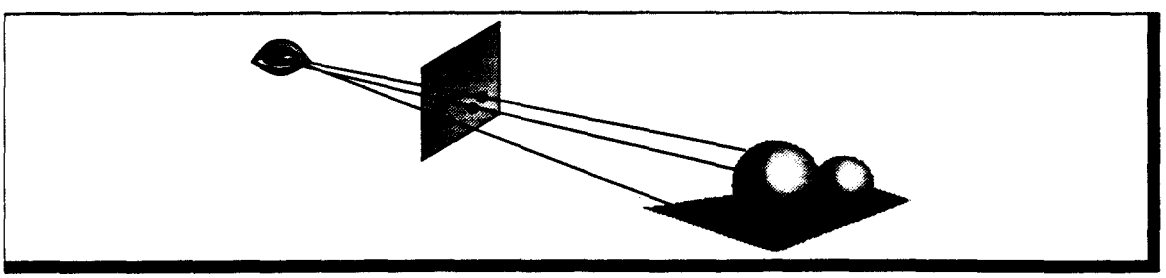


Figure 1.15 ***Modèle utilisé en lancer de rayon.***

Le plan du film étant le support de l'image photographiée, son équivalent en synthèse d'images est l'écran. La visualisation d'une image nécessite la connaissance de la couleur à afficher en chacun des pixels de l'écran. Il est, par conséquent, nécessaire de connaître l'ensemble des rayons traversant le plan écran et arrivant à l'oeil.

Lancer de rayons direct

Nous venons d'indiquer qu'il nous faut connaître la couleur apportée par chaque rayon passant par l'écran, et provenant des objets de la scène. Une première approche consiste à *suivre* les rayons générés par les sources lumineuses (Figure 1.16). Ces rayons peuvent suivre trois types de parcours :

- Les rayons du type *R1* partent de la source et arrivent à l'oeil après une réflexion ou une transmission sur un objet. La couleur du pixel traversé par le rayon est alors celle de l'objet au point d'impact. Cette couleur est déterminée par un des modèles d'éclairement vu précédemment (Phong, Cook-Torrance...)
- Les rayons du type *R2* subissent plusieurs réflexions et/ou transmissions avant de parvenir à l'oeil. Il faut, dans ce cas, appliquer récursivement le modèle d'éclairement de Whitted pour déterminer la couleur vue par l'observateur. Il faut combiner en chaque point d'impact la valeur de l'intensité directe, éclairement dû aux sources, et la valeur de l'intensité indirecte, éclairement provenant des autres objets.
- Les rayons du type *R3* ne parviennent pas à l'oeil. Ils ne contribuent donc pas à l'image vue par l'observateur.

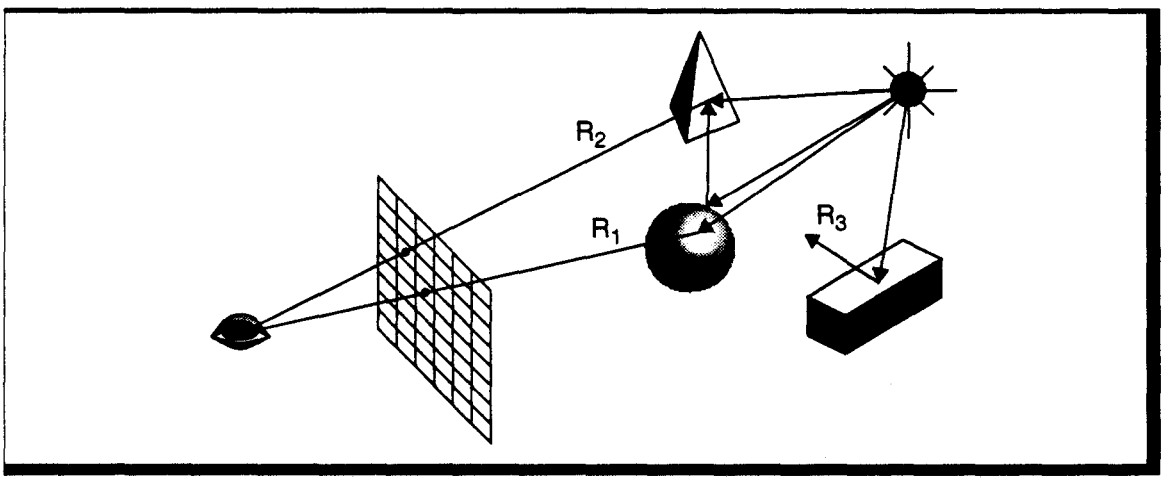


Figure 1.16 ***Lancer de rayon direct.*** *Les trois types de parcours de rayons pouvant exister.*

Ce procédé, qui permet de simuler le comportement des rayons lumineux, comporte deux inconvénients :

1. Lorsqu'un rayon atteint une surface, on ne prend en compte que la composante spéculaire de la réflexion ou de la transmission, puisque l'on choisit une direction privilégiée. La prise en compte de la réflexion diffuse (réflexion uniforme dans tout l'espace de la quantité de lumière reçue) n'étant pas possible de par le principe du suivi d'*un* rayon lumineux, une autre méthode, appelée *radiosité*, a été développée [Gora84].
2. Sur l'ensemble des rayons émis par les sources lumineuses, seul un petit nombre parviendra à l'observateur. Cette méthode est donc peu efficace. Pour y remédier, un autre procédé est utilisé, permettant de ne calculer que les rayons utiles : le *lancer de rayon inverse*.

Lancer de rayons inverse

Pour limiter les temps de calcul, on désire ne prendre en compte que les rayons lumineux contribuant à l'image. Autrement dit, il ne faut considérer que les rayons qui traversent l'écran, à destination de l'observateur. Cela ne peut se faire a priori en partant des sources. On effectue

donc le parcours inverse, consistant à *lancer* les rayons, depuis l'oeil, à travers chaque pixel de l'écran, en direction de la scène. (Figure 1.17)

Ces rayons, appelés *rayons primaires*, intersectent (P_1, P_2) ou non (P_3) un des objets de la scène. Lorsqu'il y a impact, il faut tout d'abord déterminer si l'objet est éclairé par les sources lumineuses. Pour cela, on *envoie* un rayon, appelé *rayon d'ombrage*, vers chacune des sources (L_1, L_2, L_3, L_4). Si le rayon d'ombrage n'est pas intercepté par un autre objet (ce qui n'est pas le cas de L_2), on calcule la valeur de l'intensité directe. On émet ensuite un rayon dans la direction transmise et/ou réfléchie (R_1, R_2), afin de déterminer l'intensité indirecte. Ces rayons, appelés *rayons secondaires*, pouvent, à leur tour, rencontrer un objet. On émet alors les rayons d'ombrages, réfléchis, transmis (R_3)...

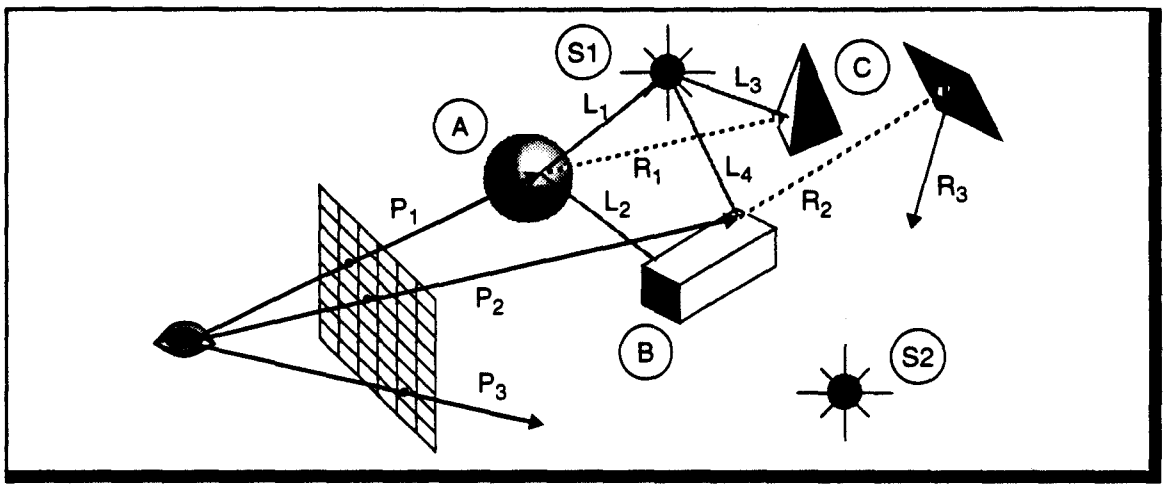


Figure 1.17 ***Lancer de rayon inverse.*** *Représentation d'une partie des rayons.*

Ce procédé permet donc de ne calculer que les rayons lumineux traversant l'écran. Il présente cependant un inconvénient par rapport à la technique de lancer direct. Dans la plupart des implémentations, le parcours des rayons d'ombrage est stoppé dès qu'il y a intersection avec un autre objet opaque. Il n'y a pas de prise en compte d'une éventuelle réflexion sur cet objet, ce qui empêche l'éclairage indirect. Par exemple, l'intensité du point d'impact du rayon L_2 n'est pas utilisée pour le calcul de l'intensité directe de l'objet A. D'autre part, on considère généralement que les rayons d'ombrage ne sont pas déviés lorsqu'ils traversent un objet transparent. Là aussi, on obtient une perte de réalisme. Ces deux phénomènes pourraient, certes, être simulés, mais au pris d'une large augmentation du temps de calcul, pour un accroissement de réalisme minime, par rapport à celui déjà obtenu par le lancer de rayon vis-à-vis des méthodes classiques.

Forêt de rayons

Comme nous venons de le voir, le lancer de rayon est un procédé purement récursif, qui peut donc être représenté par un arbre (Figure 1.18).

Pour chacun des pixels écran, on lance un rayon primaire qui génère des rayons secondaires et des rayons d'ombrage... En chaque point d'intersection, il y a à déterminer les rayons à lancer, et ensuite à calculer l'intensité lumineuse du point, en fonction des intensités portées par chaque rayon émis. Cette valeur est renvoyée au *père* du rayon, pour être combinée avec les autres intensités.

L'arbre est construit en descendant, par création des rayons. Lors de la remontée, il y a accumulation de la valeur d'intensité.

Dans certains cas (miroirs en vis à vis, par exemple), la profondeur de l'arbre pourrait être infinie. On choisit en général de limiter artificiellement cette profondeur, afin d'éviter ce phénomène. La profondeur maximale est souvent fixée en dessous de 10.

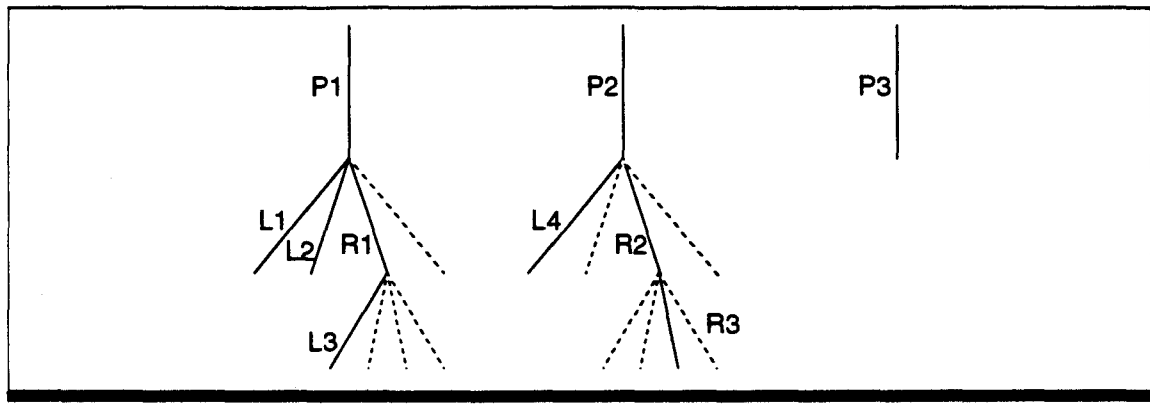


Figure 1.18 ***Forêt de rayons.*** *Ne sont représentés que les rayons de la Figure 1.17.*

Construction de l'arbre

Un rayon est défini par sa position initiale (l'oeil pour un rayon primaire, le point d'impact pour un rayon secondaire ou d'ombrage) et par sa direction. *Lancer* un rayon consiste à déterminer un point d'intersection entre ce rayon et l'objet le plus proche sur sa trajectoire.

Il n'est pas possible de connaître a priori le point d'impact du rayon. Il faut donc effectuer un calcul algébrique d'intersection entre le segment support du rayon et l'ensemble des objets de la scène. On trie ensuite la liste des points obtenus, pour déterminer celui qui est le plus proche du point de départ du rayon.

Le calcul d'intersection est, en général, effectué en utilisant une définition paramétrique du rayon. Cela simplifie grandement les opérations nécessaires. Des méthodes de calculs ont été développées pour les différents types d'objets habituellement utilisés : le plan, les polygones, les quadriques, les surfaces gauches.

Algorithme

Le pseudo-algorithme du lancer de rayon pourrait être :

```
Pour chaque pixel écran
|    Couleur_Pixel <- TRACER (rayon primaire)
FPour

Fonction TRACER (rayon)
|    Rechercher intersection la plus proche
|    Si (non_intersection OU profondeur_maxi) alors
|    |    Couleur <- Couleur_Fond
|    Sinon
|    |    Couleur <- Couleur_ambiante
|    |    Pour chaque source
|    |    |    Si (non intersection rayon d'ombrage) alors
|    |    |    |    Couleur <- Couleur + Intensité_directe
|    |    |    Fsi
|    |    FPour
|    |    Si (réflexion) alors
|    |    |    Couleur <- Couleur + TRACER (rayon réfléchi)
|    |    Fsi
|    |    Si (transmission) alors
|    |    |    Couleur <- Couleur + TRACER (rayon transmis)
|    |    Fsi
|    Fsi
|    Retourner (Couleur)
Fin
```

1.1.6 En conclusion

De nos jours, le réalisme visuel de l'image présente autant d'importance que le temps nécessaire pour la générer. Un logiciel de synthèse d'images doit donc inclure les méthodes d'amélioration du rendu pour être commercialement et scientifiquement valable. Les machines de rendu classique de dernière génération intègrent de base ces techniques, grâce aux puissances qu'elles ont atteintes. Les algorithmes de Lancer de Rayon les prennent également en compte depuis de nombreuses années.

Une architecture spécifique pour le Lancer de Rayon n'a de sens que si elle permet de supporter ces méthodes. Nous nous attacherons donc à proposer une solution pour les principales d'entre elles.

1.2 Améliorations algorithmiques du Lancer de Rayon

En reprenant l'algorithme général du lancer de rayon, nous pouvons en déduire les trois facteurs qui en font une méthode coûteuse. Nous pouvons établir que, *grossièrement*,

$$T_{\text{Lancer\_Rayon}} = K \cdot Nb_{\text{Rayons\_Lancés}} \cdot Nb_{\text{Calcul\_Intersection}} \cdot T_{\text{Calcul\_Intersection}}$$

(K est un facteur de proportionnalité)

On peut donc distinguer trois stratégies d'accélération, chacune essayant de limiter l'un ou l'autre des facteurs de cette expression.

Nous poserons dans la suite du texte : $T_{IT} = Nb_{\text{Calcul\_Intersection}} \cdot T_{\text{Calcul\_Intersection}}$. Cette valeur représente le coût de la détermination de l'intersection entre un rayon et les objets de la scène, le tri sur les intersections n'étant pas pris en compte.

1.2.1 Diminution du temps de calcul d'une intersection

Il est, bien entendu, nécessaire d'optimiser les algorithmes permettant de déterminer l'intersection entre un rayon et un objet. On dispose de méthodes spécifiques pour chacun des types d'objets (intersection avec une quadrique, avec une quartique, avec une surface spline...). On a tout intérêt à utiliser les objets de plus haut niveau possible, même si la détermination du point d'intersection est plus complexe. En effet, la facettisation d'une surface gauche, par exemple, nécessite l'utilisation de plusieurs dizaines, voir centaines, de polygones. Le coût peut alors être bien plus élevé, étant donné le nombre de calculs d'intersection à effectuer.

Utilisation de volumes englobants

Un calcul d'intersection pouvant être très complexe, on définit un volume englobant pour chacun des objets de la scène, qui permettra d'effectuer un tri des objets susceptibles d'être sur le chemin d'un rayon. L'englobant est choisi de forme *simple,* afin de pouvoir déterminer rapidement s'il est intersecté ou non par un rayon. Dans le cas positif, il faut, bien sûr, vérifier ensuite s'il y a intersection réelle avec l'objet.

Un calcul rapide va nous permettre d'établir le gain réalisé :

Soit N le nombre d'objets de la scène, et T_{CO} le temps moyen de calcul d'intersection sur ces objets. Le temps nécessaire à la détermination de l'objet intersectant un rayon est alors donné par (sans tenir compte du temps nécessaire au tri) :

$$T_{IT} = T_{\text{Sans\_Volume\_Englobant}} = N \cdot T_{CO}$$

On pose T_{CV} le temps de calcul d'intersection avec un volume englobant, α le nombre moyen d'objets par volume englobant, et η le pourcentage d'intersections réussies avec les volumes englobants. On définit :

$$T_{IT} = T_{\text{Avec\_Volume\_Englobant}} = N_{\text{Volumes\_Englobants}} \cdot T_{CV} + N_{\text{Objets\_A\_Tester}} \cdot T_{CO}$$

On a : $N_{\text{Volumes\_Englobants}} = N/\alpha$

et : $N_{\text{Objets\_A\_Tester}} = N_{\text{Volumes\_Englobants\_Intersectés}} \cdot N_{\text{Objets\_Par\_Volume\_Englobant}} = (N/\alpha) \cdot \eta \cdot \alpha = N \cdot \eta$

On en déduit donc que :

$$T_{IT} = N\left(\frac{1}{\alpha} \cdot T_{CV} + \eta \cdot T_{CO}\right)$$

Le gain obtenu est, par conséquent :

$$\frac{1}{Gain} = \frac{T_{\text{Avec\_Volume\_Englobant}}}{T_{\text{Sans\_Volume\_Englobant}}} = \frac{1}{\alpha} \cdot \frac{T_{CV}}{T_{CO}} + \eta$$

Un gain important est atteint si

→ d'une part $T_{CV} \ll \alpha \cdot T_{CO}$, autrement dit si le volume englobant est très simple du point de vue du calcul d'intersection, ou s'il regroupe un grand nombre d'objets (Whitted propose d'utiliser des sphères [Whit80]),

→ d'autre part $\eta \ll 1$, c'est-à-dire si le pourcentage de volumes englobants intersectés est faible, ce qui nécessite que l'englobant soit le plus petit possible, et donc le plus proche possible des objets qu'il contient.

Ces deux conditions sont contradictoires, ce qui impose un compromis.

Exemple : Prenons un englobant sphérique contenant un ensemble composé d'une centaine de facettes ($\alpha = 100$). On a $T_{CO} \approx T_{CV}$ (cf. Annexe B). Si l'on suppose $\eta = 1/10$, on obtient alors $G \approx 10$.

Si le résultat est intéressant, il reste néanmoins que l'algorithme est toujours en temps linéaire par rapport au nombre d'objets.

1.2.2 Diminution du nombre de calculs d'intersections

Dans la méthode de base, il est nécessaire de *passer en revue* chacun des objets de la scène lorsque l'on lance un rayon. Une autre méthode visant à améliorer le lancer de rayon est de modifier l'algorithme afin qu'il soit en temps logarithmique, voire constant, par rapport au nombre d'objets.

Hiérarchie de volumes englobants

Rubin et Whitted [Rubi80] ont introduit la notion de hiérarchie de volumes englobants. Chaque englobant d'objets est lui-même intégré dans un volume regroupant plusieurs englobants, de manière hiérarchique. Si un volume *père* n'est pas intersecté, alors il est inutile de tester ses *descendants*. On aboutit ainsi à une recherche en temps logarithmique de l'intersection avec un rayon.

Là aussi, un compromis est à trouver. En effet, plus l'arité de la hiérarchie est importante, et donc plus l'on regroupe d'objets, plus l'accélération serait importante. Mais en contre-partie, les englobants *pères* sont plus volumineux, ce qui implique un taux d'intersection, η, plus élevé.

Subdivision spatiale.

La méthode précédente consiste en une organisation hiérarchique des objets, basée sur leur volume, construite depuis le bas vers le haut de l'arbre.

La subdivision spatiale, dont le but est également de minimiser le nombre de tests d'intersection, procède de manière opposée. On divise l'espace 3-D *complet* de la scène, de façon uniforme ou non, et l'on affecte à chaque subdivision les objets s'y trouvant entièrement, ou partiellement (Figure 1.19). La démarche est descendante : chaque subdivision est elle-même découpée en fonction des objets l'occupant. On obtient donc une hiérarchie de volumes basée sur la forme des objets.

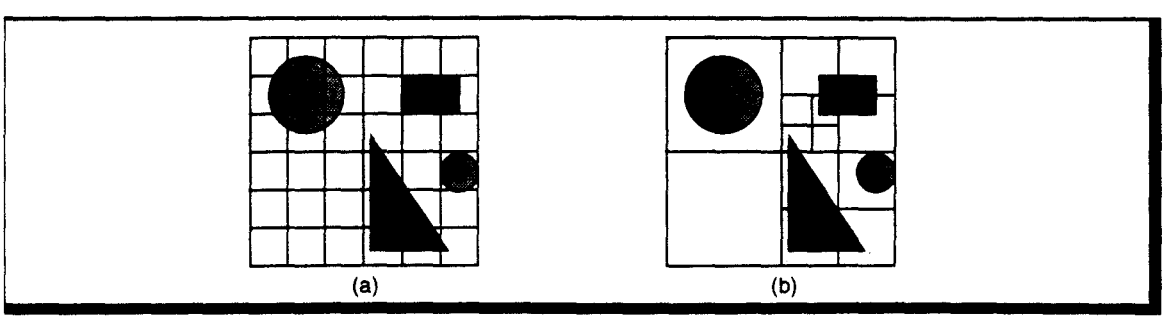


Figure 1.19 ***Subdivision Spatiale (exemple en 2 dimensions).*** *(a) - Uniforme. (b) - Non uniforme.*

La recherche d'intersection s'effectue en testant l'un après l'autre les volumes traversés par le rayon (Figure 1.20). Un algorithme de suivi du rayon permet de déterminer, en fonction du point de sortie du volume courant, le volume suivant sur le trajet.

Le pseudo-algorithme de recherche est :

```
Recherche_Intersection(Rayon)
|     Volume_Courant <- Volume_de_départ(Rayon)
|
|     Faire
|     |     Pour chaque objet du Volume_Courant
|     |     |     Détermination de l'intersection rayon-objet
|     |     FinPour
|     |     Tri des intersections
|     |     Si Non Intersection_Trouvée alors
|     |     |     Volume_Courant <- Volume_Suivant(Rayon)
|     |     FinSi
|     Jusqu'à Intersection_Trouvée OU En_Dehors_Scène
Fin
```

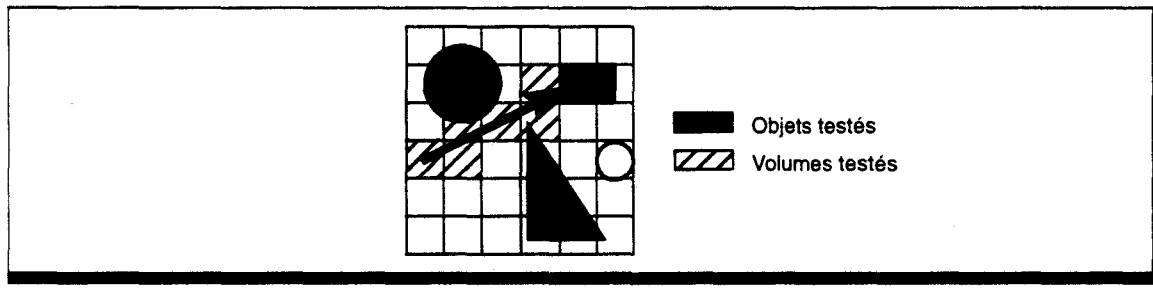


Figure 1.20 ***Recherche d'intersection (exemple en 2 dimensions).*** *On effectue un suivi du rayon de volume en volume.*

Plusieurs études ont été menées dans le cadre de cette méthode. On peut citer :

- Subdivision non uniforme : La hiérarchie peut être effectuée par partition binaire des volumes, comme l'ont utilisée Arnaldi, Priol et Bouatouch [Arna87]. Glassner utilise un arbre d'arité 8 (*octree*), correspondant à un découpage binaire des volumes dans chaque direction de l'espace [Glas84]. Le suivi de rayon nécessite un parcours de hiérarchie, qui est de complexité logarithmique.
- Subdivision uniforme : Les différences entre les propositions utilisant ce type de subdivision se situent au niveau du suivi du rayon. Fujimoto et al. [Fuji86] utilise une extension 3-D de l'algorithme DDA (Digital Differential Analyser). Amanatides et Woo [Aman87] ont défini une nouvelle méthode de suivi basée également sur le DDA mais qui, contrairement à la précédente, ne comporte pas de direction privilégiée de déplacement. Pour la machine Voxar, un algorithme à double paramètre a été développé, permettant d'accélérer la traversée de zones vides [Pito89].

  La subdivision non uniforme, ou la hiérarchie de volumes englobants, est plus adaptée pour les scènes possédant peu d'objets dispersés dans l'espace. La subdivision uniforme convient mieux à de grands ensembles d'objets régulièrement espacés. Une scène réelle comportant ces deux types d'ensembles, Snyder et Barr [Snyd87] proposent une structure de données permettant d'intégrer un mélange de hiérarchies et de grilles volumiques.

Analyse de complexité

Cette étude *rapide* de complexité est effectuée à partir de la subdivision uniforme, afin de simplifier l'analyse. Nous utiliserons pour cela les travaux de Devillers [Devi89].

Nous prenons, comme hypothèse, un rayon traversant la structure sans intersection avec les objets, afin d'obtenir un majorant du temps moyen nécessaire à la recherche d'une intersection.

Soit une grille de dimension $n \times n \times n$. Le nombre moyen de volumes traversés par un rayon est :

$$Nb_{vt} \approx n .$$

En posant T_{DS} le temps nécessaire à la détermination de l'élément de grille suivant sur la trajectoire, on en déduit le coût de la traversée :

$$T_{\text{traversée}} = Nb_{vt} \cdot T_{DS} = n \cdot T_{DS}$$

Une approximation du nombre moyen d'éléments de grille intersectant un objet X est donnée par :

$$Nb_{IT}(X) \approx 1 + V(X) + \frac{3}{4} \cdot A(X) + \frac{9}{4} \cdot D(X)$$

avec $V(X)$: Volume de l'objet,
$A(X)$: Aire de la surface de l'objet,
$D(X)$: Diamètre de l'objet (i.e diamètre de la sphère englobante de l'objet).

Cette expression nous permet de calculer τ, le nombre moyen d'objets intersectant un élément de la grille :

$$\tau = \sum_{X \in S} \frac{Nb_{IT}(X)}{n^3} = \frac{1}{n^3} \cdot \left(N + \mathcal{V} + \frac{3}{4} \cdot \mathcal{A} + \frac{9}{4} \cdot \mathcal{D} \right)$$

avec S : Ensemble des objets de la scène,
N : Nombre d'objets ($N = card(S)$),
et $\mathcal{V} = \sum_{X \in S} V(X)$, $\mathcal{A} = \sum_{X \in S} A(X)$, $\mathcal{D} = \sum_{X \in S} D(X)$.

Nous posons $\mathcal{N} = N + \mathcal{V} + 3/4 \cdot \mathcal{A} + 9/4 \cdot \mathcal{D}$ pour simplifier l'écriture, et T_{CO}, le temps moyen de calcul d'une intersection rayon-objet.

Le coût total moyen est alors de :

$$T_{IT} = T_{\text{Subdivision\_Spatiale}} = n \cdot T_{DS} + n \cdot \tau \cdot T_{CO} = n \cdot T_{DS} + \frac{\mathcal{N}}{n^2} \cdot T_{CO} \qquad \textbf{(Eq. 1.13)}$$

Rappelons qu'il s'agit du coût d'un rayon traversant la scène sans intersection. Or, l'utilisation d'un suivi du rayon implique que la recherche est arrêtée dès qu'une intersection est trouvée. Le coût moyen réel est donc difficile à définir. On peut cependant effectuer les remarques suivantes :

- $T_{\text{Subdivision\_Spatiale}}$ est de complexité linéaire par rapport à $\mathcal{N}$, et donc par rapport au nombre d'objets. Cependant si N augmente, la probabilité d'intersection augmente également, ce qui diminue le trajet moyen parcouru.
- La relation entre τ et n (taille de la grille de subdivision) est plus complexe à établir, car $V(X), A(X), D(X)$ sont exprimés par rapport à n.

Le gain obtenu par rapport à la méthode de base est : $\frac{1}{G} = \frac{n}{N} \cdot \frac{T_{DS}}{T_{CO}} + \frac{1}{n^2} \cdot \frac{\mathcal{N}}{N}$

Cette relation montre qu'il existe un optimum de la valeur de n en fonction du nombre d'objets de la scène.

Exemple : Prenons $T_{CO} = 10.T_{DS}$, $n = 100$, $N = 10^5$. Avec ces données, on obtient : $G = \frac{10^4}{1 + \frac{\mathcal{N}}{N}}$

Il est difficile de donner une valeur d'exemple pour $\mathcal{N}$, mais nous savons, par définition, que $\mathcal{N} > N$. Le gain est donc de l'ordre de 10^3. Cette valeur très élevée est confirmée par Fujimoto et al. [Fuji86] qui obtiennent un gain supérieur à 500 pour $N = 7000$ et $n = 40$.

Subdivision uniforme maximale : notion de voxel

Les volumes obtenus par subdivision uniforme de l'espace sont appelés *voxels*, en tant qu'*élément de volume*, dans la plupart des articles. Nous préférons garder cette dénomination pour le cas particulier de la subdivision en *volumes élémentaires* de l'espace et des objets.

Dans ce mode de représentation, les objets sont découpés en élément cubique de taille élémentaire (Figure 1.21). Le voxel est l'élément d'information volumique le plus fin. Il peut être vide, ou contenir un objet (avec ses caractéristiques : normale, couleur...).

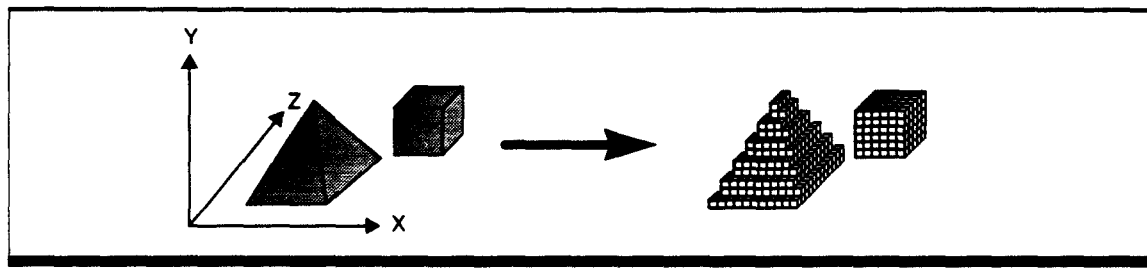


Figure 1.21 ***Discrètisation des objets en voxels.***

Le calcul d'intersection se réduit alors à un simple test de présence d'un objet au sein du voxel courant sur le trajet du rayon. On pose T_{TP} le temps nécessaire à cette détermination, et on a alors :

$$\tau \cdot T_{CO} = T_{TP}$$

L'expression du coût total (Eq. 1.13) devient :

$$T_{IT} = T_{\text{Subdivision\_Voxel}} = n \cdot (T_{DS} + T_{TP}) \qquad \textbf{(Eq. 1.14)}$$

On obtient un coût en temps constant par rapport au nombre d'objets, mais linéaire par rapport à la taille de la grille (ou à la finesse de discrètisation).

Le gain prend pour valeur :

$$\frac{1}{G} = \frac{n}{N} \cdot \left(\frac{T_{DS} + T_{TP}}{T_{CO}} \right)$$

Un gain important est obtenu si

- ➔ $n \ll N$, mais il faut cependant n assez grand pour obtenir une discrètisation correcte des objets. On prendra la valeur de la taille de l'écran en pixels comme largeur de grille, ce qui permet une adéquation entre la finesse de l'affichage et la finesse de la discrètisation. Cette méthode est donc d'autant plus efficace que le nombre d'objets est important.
- ➔ $(T_{DS} + T_{TP}) \ll T_{CO}$, ce qui impose une méthode de suivi du rayon efficace et un accès rapide à l'information de présence d'un objet au sein d'un voxel.

Exemple : Prenons $T_{CO} = 10 \cdot (T_{DS} + T_{TP})$, $n = 10^3$ et $N = 10^5$. On obtient alors $G = 10^3$.

Le gain est également important, et il faut retenir, de plus, que le temps de calcul est indépendant du nombre d'objets, ainsi que de leur type puisqu'il n'y a pas de calcul réel d'intersection.

Notons que le gain atteint est équivalent à celui d'une subdivision régulière, malgré la suppression du calcul d'intersection. Ceci s'explique par l'augmentation importante de nombre de subdivisions que doit traverser un rayon, et donc par un allongement du temps de parcours des rayons au sein de la structure voxel.

En résumé

Les méthodes de subdivision permettent d'accélérer considérablement le Lancer de Rayon. Cependant, elles nécessitent un pré-calcul de découpage dont il n'a pas été tenu compte. Le temps de préparation est, bien sûr, linéaire par rapport au nombre d'objets, mais il équivaut dans la plupart des cas à un maximum de 10% du temps de calcul global. Un autre inconvénient de ces techniques est qu'elles nécessitent un volume mémoire important (surtout pour la subdivision voxel), ce qui limite le nombre d'objets visualisables pour les subdivisions spatiales, ou la taille de la grille voxel et donc la précision de la discrètisation.

Malgré cela, c'est grâce à ces techniques que le Lancer de Rayon a pu obtenir sa place parmi les outils de synthèse d'image les plus répandus.

1.2.3 Diminution du nombre de rayons lancés

Le troisième facteur intervenant dans le coût temporel du Lancer de Rayon est le nombre de rayons lancés (rayons primaires et secondaires). Ce facteur peut être diminué, soit en limitant le nombre de rayons générés, soit en regroupant le plus grand nombre possible de rayons au sein d'entités de plus haut niveau (cônes, faisceaux...) dont le nombre est, par conséquent, moins élevé.

Diminution du nombre de rayons.

Dans le Lancer de Rayon classique, la propagation d'un rayon est terminée, si on atteint une profondeur d'arbre prédéfinie, si le rayon sort de la scène (pas d'intersection), ou si il atteint un objet opaque non réfléchissant. Hall et Greenberg [Hall83] ajoutent un autre critère d'arrêt, en prenant en compte la contribution qu'un rayon peut apporter au pixel dont il dépend. Si cette contribution est inférieure à un certain seuil, on peut alors terminer la récursion, sans préjudice notable sur l'image finale.

Regroupement de rayons

Le Lancer de Rayon classique n'utilise pas le phénomène de cohérence spatiale, qui se traduit par le fait que certains rayons suivent des itinéraires identiques (Figure 1.22).

Plusieurs auteurs ont étudié la possibilité de regrouper ces rayons au sein d'entités tridimensionnelles afin de diminuer le coût du Lancer de Rayon :

- Heckbert et Hanrahan [Heck84] utilisent une structure de *faisceaux* à base polygonale quelconque. Un premier faisceau est créé, partant de l'observateur et couvrant la surface correspondant à l'écran. Un algorithme de tri est utilisé pour déterminer les objets vus par ce faisceau. Les surfaces intersectées permettent de créer des faisceaux secondaires réfléchis et/ou réfractés par transformation matricielle du faisceau incident. On définit ainsi une arborescence de faisceaux, identique à l'arborescence de rayons du Lancer de Rayon classique.

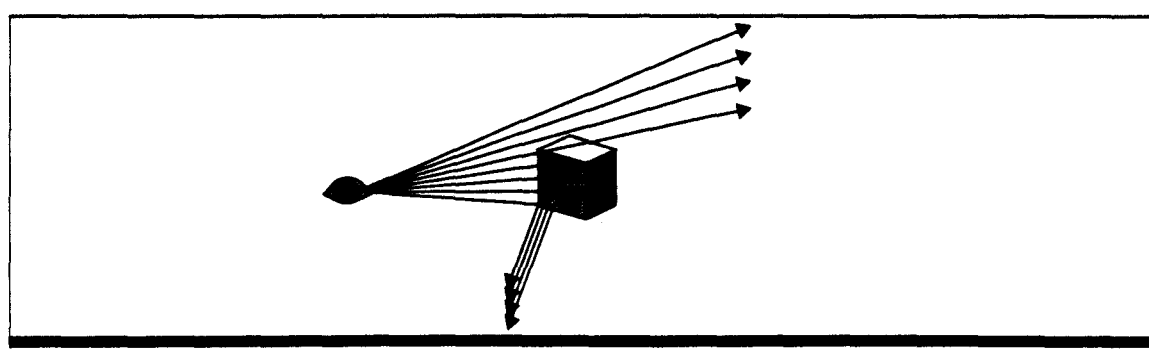


Figure 1.22 ***Cohérence spatiale.*** *Certains rayons suivent le même trajet.*

Du fait de l'utilisation de matrice de transformation géométrique pour définir les faisceaux, la réflexion et la réfraction doivent être des opérations linéaires. Cela ne permet d'utiliser

que des objets polygonaux. La transformation associée à une réfraction n'est pas linéaire, et pour résoudre ce problème, Heckbert et Hanrahan font l'approximation que le faisceau incident est parallèle (angle d'ouverture nul) afin d'obtenir une transformation linéaire.

- Dans la méthode précédente, la base des faisceaux est un polygone quelconque, pouvant même être concave. La détermination de l'intersection entre un tel faisceau et les objets de la scène est donc complexe. Ghazanfarpour [Ghaz92] propose de découper les faisceaux incidents par subdivision adaptative, afin de n'utiliser que des *pyramides* à base rectangulaire. Dans le cas de la réfraction, il utilise une pyramide englobante du faisceau réfracté réel, ce qui permet d'améliorer l'erreur commise.
- Amanatides [Aman84] regroupe les rayons de suréchantillonage au sein de *cônes* à base circulaire, partant de l'observateur et couvrant un pixel. Lors d'une intersection avec un objet, un nouveau cône est lancé, dans la direction de réflexion ou de transmission. Amanatides ne fournit pas d'expression générale de l'angle d'ouverture de ces cônes secondaires.

  Pour tenir compte du fait que l'intersection entre un objet et un cône peut être partielle, Amanatides gère une liste des 8 objets les plus proches de la source du cône, avec, pour chacun d'eux, le pourcentage de recouvrement de l'intersection. Il peut ainsi calculer une approximation de la valeur réelle d'éclairement.
- Shinya et al. [Shin87] proposent une généralisation des méthodes précédentes à l'aide de *pinceaux*. Un pinceau est composé d'un rayon axial (rayon central) et d'un ensemble de rayons *paraxiaux* (rayons ayant une direction proche du rayon axial). La déformation d'un pinceau par une surface réfléchissante ou transparente est approximée par une opération linéaire, basée sur la théorie de l'approximation paraxiale utilisée en optique et en électromagnétisme. Sous certaines conditions fixant l'angle d'ouverture des pinceaux utilisés, cette théorie fournie des résultats exacts. Cette méthode permet d'utiliser des objets de types quelconques, et donne une solution correcte au problème de la réfraction.

Ces approches permettent de réduire fortement le nombre de *rayons* lancés, mais les calculs nécessaires sont complexes. Le gain obtenu n'est de ce fait, que d'un facteur 10. De plus, ces techniques sont très difficilement compatibles avec les techniques de subdivision vues précédemment.

Leur principal avantage est d'améliorer la qualité du rendu en y adjoignant des effets de pénombre, ou de réflexion douce, et surtout de permettre un anti-aliassage efficace sans suréchantillonnage, puisqu'elles n'utilisent plus d'échantillonnage de point.

1.2.4 Conclusion

Nous proposons ici un tableau récapitulatif des gains potentiels des méthodes d'accélération du Lancer de Rayon (Table 1.1).

Cette comparaison fait ressortir les méthodes à subdivision spatiale et à subdivision voxel. Nous devons cependant noter que les valeurs de gain réellement obtenues sur ces techniques sont souvent en deçà de celles indiquées. Malgré cela, elles ont permis d'améliorer considérablement les temps de calcul des algorithmes de Lancer de Rayon, ce qui a permis de le rendre utilisable sur les machines courantes.

La méthode de subdivision voxel présente l'avantage de l'indépendance au nombre d'objets, mais avec un surcoût important de conversion des objets en voxels.

Les autres méthodes (volumes englobants, regroupement de rayons) ne permettent pas un gain sensible de performance, mais peuvent être utilisées en accélération secondaire à la subdivision. Certaines d'entre elles apportent en plus une amélioration de la qualité de l'image.

Méthode	1 / Gain	Ordre de grandeur
'de base'	-	1
Volumes englobants	$\frac{1}{\alpha} \cdot \frac{T_{CV}}{T_{CO}} + \eta$	1/10
Subdivision spatiale	$\frac{n}{N} \cdot \frac{T_{DS}}{T_{CO}} + \frac{1}{n^2} \cdot \frac{\mathcal{N}}{N}$	1/1000
Subdivision Voxel	$\frac{n}{N} \cdot \left(\frac{T_{DS} + T_{TP}}{T_{CO}} \right)$	1/1000
Regroupement rayons	?	1/10

N : Nombre d'objets,
η : Nombre d'objets à tester avec utilisation de volumes englobants,
n : Taille de la grille de subdivision,
$\mathcal{N}$: Nombre d'éléments de grille occupés par les objets,
T_{CO} : Temps de calcul d'une intersection rayon / objet,
T_{CV} : Temps de calcul d'une intersection rayon / volume englobant,
T_{DS} : Temps de calcul de l'élément de grille suivant pour un suivi de rayon,
T_{TP} : Temps du test de présence d'un objet au sein d'un voxel.

Table 1.1 ***Accélération potentielle des méthodes d'optimisation du Lancer de Rayon.***

1.3 Parallélisation du Lancer de Rayon

Malgré les améliorations apportées à l'algorithme de base, la génération d'images complexes par Lancer de Rayon nécessite encore plusieurs dizaines de minutes. En attendant que les processeurs aient la puissance nécessaire pour que ce temps devienne *raisonnable*, l'utilisation de machines parallèles a été envisagée afin d'obtenir un facteur d'accélération supplémentaire (la notion d'accélération et d'efficacité des architectures parallèles est décrite en Annexe C).

En effet, l'algorithme du Lancer de Rayon exhibe différentes formes de parallélisme potentiel :

- **Au niveau pixel** : Les valeurs d'intensité des pixels sont mathématiquement indépendantes. Il est donc possible de les calculer en parallèle. Le degré de parallélisme potentiel est théoriquement égal au nombre de pixels, soit 10^6 pour un écran 1000x1000.
- **Au niveau rayon** : Lors d'une intersection entre un rayon et un objet, plusieurs rayons secondaires sont créés, dont l'évaluation est indépendante. On peut envisager de gérer séparément ces rayons, et obtenir, ainsi, un parallélisme de degré 2, 3, ou plus, en fonction du nombre de sources lumineuses.
- **Au niveau calcul** : Le Lancer de Rayon manipule des entités à trois dimensions (que ce soient les objets ou les rayons). La plupart des calculs sont, de ce fait, des calculs vectoriels ou matriciels, qui peuvent être effectués en parallèle. Le degré de parallélisme est alors égal à la dimension des vecteurs (3 en général, 4 si l'on utilise des coordonnées homogènes, 5 dans la méthode proposée par Arvo et Kirk [Arvo87]).
- **Au niveau objet** : Les objets de la scène étant indépendants entre eux, l'évaluation de l'intersection entre un rayon et ces objets peut être réalisée en parallèle, avec un degré de parallélisme correspondant au nombre d'objets, soit de l'ordre de 10^3 à 10^6.
- **Au niveau spatial** : Dans le cas de la subdivision spatiale de la scène, les régions sont indépendantes, ce qui permet de les distribuer sur une machine multiprocesseur. Le gain est alors, théoriquement, de l'ordre du nombre de régions.

Pour une implémentation réelle, il est assez difficile de séparer aussi exactement les différents schémas énoncés. L'utilisation du parallélisme spatial, par exemple, implique bien souvent un parallélisme pixel, rayon et objet :

> On dispose d'un processeur par région, ce qui permet de lancer simultanément autant de rayons primaires qu'il y a de processeurs (parallélisme pixel).
>
> D'autre part les rayons secondaires vont suivre des chemins différents, ce qui va les amener dans des régions différentes et donc sur des processeurs différents, ce qui permet leur évaluation simultanée (parallélisme rayon).
>
> Enfin, un rayon passant de région en région, on obtient un calcul d'intersection en mode pipeline (ce qui induit un parallélisme objet).

Une autre classification des schémas parallèles est, par conséquent, nécessaire. Elle est basée sur le type du flot entre les processeurs. Dans les études existantes, on distinguera : les machines sans flot de données, avec flot d'objets ou avec flot de rayons [Prio89].

1.3.1 Architectures sans flot de données

Le calcul des arbres de rayons sous-tendus aux pixels étant indépendants, le parallélisme le plus trivial du Lancer de Rayon consiste en la distribution de ces calculs sur un multiprocesseur : chaque processeur met en oeuvre un algorithme purement séquentiel; il fonctionne indépendamment des autres.

Pour effectuer le calcul d'un arbre de rayons, il est nécessaire d'avoir connaissance de toute la scène, ce qui nécessite une recopie de la base de données sur chacun des processeurs. Cela implique soit un très gros volume mémoire, soit un nombre limité d'objets.

Pour assurer une utilisation optimale de la machine, le multiprocesseur est programmé en mode *demand-driven* : lorsqu'un processeur a terminé le calcul d'un arbre, il émet une requête

au processeur maître afin d'obtenir un nouveau pixel à calculer. De cette manière, on obtient un équilibrage automatique de la charge.

Plusieurs implémentations de cette méthode ont été réalisées. On peut citer les machines LINKS [Nish83], CRISTAL-TPX [Brus86] et SIGHT [Naru87]. Cette dernière intègre, en plus, un parallélisme de calcul, puisque chaque noeud dispose de trois unités de calcul flottant et une unité de calcul entier indépendante pour, entre autres, les calculs d'adresses des objets en mémoire.

Performances

Pour analyser les performances de ce type d'architecture, nous utiliserons l'expression utilisée pour mesurer la durée d'exécution d'une méthode *divide-conquer-marry* :

$$T(N, n) = g(N, n) + n \cdot T(N/n) + h(N, n)$$

avec
- N : Nombre de données,
- n : Nombre de sous-ensembles utilisés,
- $g(N, n)$: Temps nécessaire pour diviser le problème en n sous-problèmes (*divide*),
- $T(N/n)$: Temps nécessaire pour exécuter un sous-problème (*conquer*),
- $h(N, n)$: Temps nécessaire pour rassembler les données (*marry*).

Dans notre cas, les n sous-problèmes sont exécutés en parallèle. L'expression devient donc :

$$T(N, n) = g(N, n) + T(N/n) + h(N, n)$$

La valeur de $g(N, n)$ inclus le temps nécessaire pour distribuer les calculs des pixels. La valeur de $h(N, n)$ comprend, elle, le temps passé à récolter les résultats de calculs. Ces entités sont une traduction de la fraction séquentielle du problème.

L'algorithme exécuté sur le multiprocesseur étant le même que celui utilisé sur un monoprocesseur, on peut mesurer l'efficacité par :

$$Acc(n) = \frac{T_{\text{séquentiel}}}{n \cdot T_{\text{parallèle}}} = \frac{n \cdot T(N/n)}{n \cdot T(N, n)} = \frac{1}{1 + \frac{g(N, n) + h(N, n)}{T(N/n)}} \qquad \textbf{(Eq. 1.15)}$$

Il faut donc rechercher à minimiser la surcharge de parallélisation pour obtenir la meilleure efficacité possible. Ils sont, dans le cas des architectures sans flot de données, faible devant le temps de calcul du Lancer de Rayon, ce qui donne une efficacité proche de 1.

Le principal inconvénient de ce schéma de parallélisation est de nécessiter une recopie complète de la base de données sur chacun des noeuds de calculs.

1.3.2 Architectures à flot d'objets

Si l'on désire visualiser des scènes complexes dont le volume de stockage est supérieur à la capacité mémoire d'un noeud de calcul, il est nécessaire d'utiliser un système de mémoire partagée. Pour accélérer les accès à cette mémoire, et minimiser le goulot d'étranglement qu'ils induisent, un mécanisme de *cache* est utilisé : lorsqu'une donnée n'est pas présente en mémoire locale, elle est recherchée dans la mémoire partagée.

Dans le cadre du Lancer de Rayon, on peut observer deux phénomènes permettant d'assurer de l'efficacité de l'utilisation d'un cache :

1. Le parcours des arbres correspondant à deux pixels voisins sera pratiquement identique, du fait de la cohérence spatiale de la scène. D'un pixel à son voisin, on référencera les mêmes objets.
2. Une étude de Green et Paddon [Gree89] montre qu'en général, un petit nombre d'objets de la scène est utilisé le plus fréquemment (20% des objets sont utilisés pendant 80% du temps total sur 3 des 4 scènes tests).

Trois études utilisant ce type d'architecture peuvent être citées :

- La machine de Green et Paddon [Gree89]. Elle est composée de Transputers indépendants, reliés par un switch à un processeur de contrôle centralisant les données (objets et pixels). Chaque noeud dispose d'une mémoire de 512Ko à 1Mo.

  La mémoire locale est divisée en deux parties : la première contient un ensemble d'objets *résidents* définis par la $2^{ème}$ remarque précédente; l'autre portion sert de mémoire cache. Un premier lancer de rayon sous-échantillonné (sur un nombre restreint de pixels), permet de définir l'ensemble des objets les plus utilisés. Leur stockage en mémoire locale permet de diminuer grandement les accès au processeur central.

  Pour une configuration à 512Ko, en moyenne 10% des objets (sur les scènes tests) peuvent être stockés en mémoire locale, et 40% pour une configuration de 1Mo. Sur une machine à 22 noeuds, ils obtiennent une efficacité de 76% pour 512Ko et 98% pour 1Mo, l'accélération étant linéaire avec le nombres de processeurs.

- La Pixel Machine de l'AT&T Bell Laboratories [Potm89]. Cette machine a été définie pour permettre l'implémentation de l'ensemble des algorithmes de synthèse d'images. Elle est composée d'un pipeline suivi d'un réseau bidimensionnel, basés sur les processeurs DSP32. Chaque noeud est relié à un processeur central par l'intermédiaire d'un bus VME. Le réseau peut être constitué de 16 à 64 processeurs, chacun disposant de 512Ko.

  Pour le Lancer de Rayon, le pipeline effectue les diverses opérations de facettisations, de calculs de volumes englobants... Le réseau est utilisé pour le calcul des arbres de rayons, deux pixels voisins étant traités par deux processeurs voisins, avec répartition cyclique. On obtient ainsi un équilibrage de charge statique, il n'y a pas de requête de calcul émis vers le processeur central.

  La mémoire partagée située sur le processeur maître est découpée en pages. Chaque noeud peut mémoriser localement un certain nombre de pages. Du fait du mode de distribution des pixels, la cohérence spatiale des objets se traduit par un besoin quasi-simultané des mêmes données par tous les processeurs. Par conséquent, lorsqu'un noeud effectue une requête de page au processeur central, celle-ci est envoyée sur tous les noeuds du réseau. Une analyse des requêtes montre que le nombre de pages effectivement servies représentent 10% du nombre de pages demandées. La politique d'utilisation d'objets résidents n'est pas implémentée dans cette architecture.

  L'efficacité de cette machine est également proche de 1, mais Potmesil et Hoffert ont mesuré une perte de performance d'un facteur 5 entre une scène pouvant être intégralement stockée en mémoire locale, et une scène nécessitant le mécanisme de pagination. Cette dégradation peut s'expliquer par la gestion centralisée de la mémoire partagée qui produit un goulot d'étranglement.

- La machine de l'IRISA définie par Badouel et Priol [Bado91a]. La machine utilisée est un hypercube iPSC/2 d'Intel disposant de 64 noeuds basés sur le processeur 80386 avec 4Mo de mémoire locale.

  Chaque processeur a en charge le calcul d'une partie de l'image. Lorsqu'un noeud a terminé, il émet un requête vers les autres processeurs. Un noeud qui est encore chargé lui envoie alors la position d'une zone à calculer. La taille optimale de cette zone est de 3x3 pixels. Les requêtes circulent sur un anneau. Par conséquent, si un noeud reçoit la demande qu'il vient d'émettre, cela lui indique que l'image a été entièrement calculée.

  Pour éviter le problème de la centralisation de la base de données rencontré dans la proposition de Green et Paddon, celle-ci est répartie uniformément sur l'ensemble des noeuds. La mémoire locale est divisée en deux parties, l'une pour le stockage réparti des données, l'autre servant de cache. Cette approche est dénommée *mémoire virtuelle distribuée*.

  Chaque objet possède un numéro qui permet de déterminer le noeud où il est mémorisé. Lorsqu'un noeud désire obtenir un objet, il émet une requête vers le processeur concerné. En début d'exécution du Lancer de Rayon, de nombreuses requêtes sont émises, mais, en vertu des deux remarques sur l'efficacité des caches, le taux de défaut d'objets diminue rapidement.

L'efficacité mesurée est de l'ordre de 90% pour la configuration à 64 noeuds, avec un comportement linéaire en fonction du nombre de processeurs. Badouel a mesuré que le service des objets et des calculs représente, respectivement, 20% et 0,5% du temps total de calcul [Bado90].

Performance

Les performances de ce type d'organisation suivent les mêmes règles que celles du type précédent. La fonction $g(N, n)$ inclus en plus ici le temps de gestion des caches (et de la mémoire distribuée pour la dernière machine).

Dans le cas de la machine de Badouel et Priol, plusieurs demandes de pages peuvent être honorées en parallèle, ce qui permet de diminuer la surcharge. La complexité de la scène visualisable est directement proportionnelle à la capacité de mémoire de la machine, et donc du nombre de processeurs et de la quantité de mémoire locale.

Ces machines permettent ainsi de visualiser des scènes importantes, au détriment de l'accélération obtenue (20% de perte d'après l'étude de Badouel).

On peut cependant se poser la question de l'efficacité de ce type d'organisation lorsque la cohérence spatiale est faible, c'est-à-dire lorsque les rayons sont fortement divergents. Dans ce cas en effet, les *défauts de caches* peuvent devenir très important, et donc la valeur de $g(N, n)$ être très élevée. Ce phénomène peut amener à un *écroulement* de la machine (d'autant plus sensible dans la proposition de Badouel et Priol). Il ne semble pas que l'étude de ces phénomènes ait été abordée.

1.3.3 Architectures à flot de rayons

Les deux schémas de parallélisation précédents utilisent l'indépendance du domaine de calcul. Nous avons indiqué que le Lancer de Rayon présente également une indépendance du domaine spatial, les objets pouvant être gérés en parallèle. C'est la base des architectures de ce dernier type : un processeur a en charge un sous-ensemble de l'espace objet; la circulation des rayons au sein de la scène se traduit par un échange de ces rayons entre les noeuds pour passer de sous-volume en sous-volume, ce qui nécessite un réseau de processeurs avec liens de communications. L'algorithme du Lancer de Rayon implémenté correspond à celui utilisant les optimisations par subdivision spatiale (§ 1.2.2 - p. 26).

Le problème majeur de ce type de parallélisation provient de la répartition de charge. Lorsqu'un volume contient un grand nombre d'objets, ou un point de convergence des rayons, le noeud associé est plus chargé que les autres, ce qui a pour effet de diminuer l'efficacité de la machine. Plusieurs stratégies ont été adoptées :

- Modification dynamique des sous-volumes. Dippé et Swensen [Dipp84] subdivisent l'espace en tétraèdre, répartis sur un réseau 3D de processeurs. Nemoto et Omachi [Nemo86] utilisent des parallélépipèdes rectangles distribués également sur un réseau 3D.

  Chaque noeud a la connaissance de sa propre charge et de celle de ses voisins, déterminée par le nombre de rayons stockés en file d'attente et restants à calculer. Lorsque la différence est trop importante, un transfert de charge est effectué par modification des frontières des sous-volumes. Cet échange est effectué par émission d'un message vers le noeud voisin le moins actif.

  Cette méthode permet de résoudre le problème d'équilibrage de charge, mais au prix d'une surcharge de calcul et surtout de communication qui affecte de façon non négligeable l'efficacité.

- Equilibrage statique par précalcul. Pour éviter le surcoût de l'équilibrage dynamique, Priol et Bouatouch [Prio88] proposent d'utiliser une prédétermination de la charge. Pour cela, un Lancer de Rayon sous-échantillonné est effectué sur une structure 3D régulière de l'espace, afin de connaître la charge de chacun des sous-domaines. Ces derniers sont ensuite regroupés en partitions de charge égale. Cette nouvelle subdivision est distribuée sur un hypercube iPSC/2 pour l'application réelle du Lancer de Rayon.

Les études de Priol [Prio89] indiquent que pour un découpage régulier sans équilibrage, le taux d'occupation moyen des processeurs est de 17%, contre 82% en utilisant la méthode proposée. Mais l'efficacité obtenue est de 50% pour une machine à 32 processeurs, d'où une surcharge de près de 30% due aux échanges de rayons. L'efficacité a une croissance logarithmique en fonction du nombre de processeurs.

- L'équilibrage de la charge de calcul est donc le problème le plus important de ce type de parallélisation. Dans son implémentation W.Lefer a choisi une autre stratégie [Lefe92]. La recherche de l'intersection entre un rayon et les objets peut être séparée en deux calculs : premièrement le suivi du rayon dans la structure de subdivision (80% du temps de calcul), puis la recherche de l'intersection au sein d'une subdivision. Seul le deuxième calcul nécessite la connaissance de la géométrie des objets, et ne peut donc être exécuté que sur le processeur ayant en charge cette zone de l'espace. Par contre le suivi de rayon dans la structure d'accélération peut être effectué par la totalité des processeurs, à condition que chacun d'entre eux possède l'ensemble des informations relatives à la subdivision. Un processus *central* prend en charge la génération des rayons primaires, la distribution des calculs de suivi de rayon dans la structure d'accélération, l'envoi des calculs d'intersections sur les processeurs adéquats, et la collecte des rayons retours. Pour obtenir un équilibrage de charge efficace, les demandes de calculs sont regroupées au sein de *paquets*, avant d'être émises. Les études de Lefer ont montré une taille optimum de 40 rayons par paquet. L'implémentation a été effectuée sur une *surface* Meiko MK201 composée d'une station Sun et transputers T800 disposant de 2Moctets de mémoire. L'efficacité obtenue est en moyenne de 75%, avec une accélération qui semble linéaire en fonction du nombre de processeurs. Aucun essai n'a pu être effectué au delà de 8 processeurs. Le principal inconvénient de cette technique est l'utilisation d'un processus centralisé, qui pourrait constituer un goulot d'étranglement sur une machine dotée d'un plus grand nombre de processeurs. Cependant cette solution nous semble la plus appropriée pour résoudre le problème de l'équilibrage de charge.
- La machine Voxar [Mois91]. Cette machine utilisant un schéma d'accélération semi-voxel, nous la détaillerons plus précisément dans un chapitre ultérieur.

Performance

De par son mode de fonctionnement, il est difficile d'estimer les performances de ce schéma de parallélisation. Les rayons passant de processeur en processeur, on peut assimiler le fonctionnement global à un pipeline, ce qui permet d'en déterminer le comportement.

Si le pipeline est court, alors le temps d'exécution est conditionné par l'unité la plus lente, ce qui correspond au processeur le plus chargé. Si, par contre, le pipeline est très long, alors c'est le temps de traversée du rayon qui limite la machine. Pour déterminer ces comportements limites, nous considérerons un réseau 3D de P processeurs, avec une subdivision régulière [Clea86].

La charge d'un processeur est fixée par le nombre de rayons entrant dans les sous-domaines qu'il possède, et par le nombre de subdivisions que traversent ces rayons. On a donc :

$$T_P = O(R_S \cdot D_S)$$

avec R_S : Nombre de rayons gérés par le processeur,
D_S : Nombre moyen de sous-domaines intersectés par un rayon.

Le nombre de rayons R_S est fonction de la surface de l'enveloppe des sous-domaines, d'où $R_S = O\left(P^{-2/3}\right)$. D_S est fonction de la diagonale de cette enveloppe, d'où $D_S = O\left(P^{-1/3}\right)$.

On en déduit donc que $T_P = O\left(P^{-1}\right)$

Le rayon le plus long correspondant à la traversée d'une diagonale du réseau, la deuxième limite est $T_R = O\left(P^{1/3}\right)$

Le comportement de l'architecture est situé entre ces limites, et dans tous les cas le gain obtenu est inférieur à un gain linéaire (Figure 1.23)

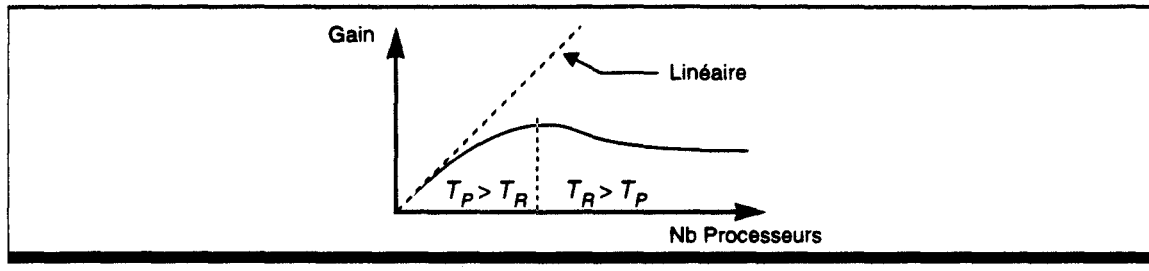


Figure 1.23 ***Comportement d'une architecture à flot de rayons.***

1.3.4 Conclusion

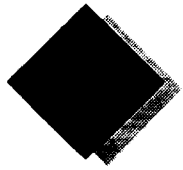
Les architectures précédentes utilisent un parallélisme limité, puisque les configurations maximales rencontrées sont de l'ordre de la centaine de processeurs. Cependant, elles ne possèdent aucune contrainte quand à l'augmentation du nombre de noeuds, si ce n'est l'accroissement du surcoût de parallélisation qui en découle. Celui-ci peut ne pas être négligeable et même influer fortement sur le gain potentiel. Nous pensons que ces approches peuvent souffrir d'un trop grand parallélisme.

Une autre caractéristique commune est l'utilisation d'objets sous leur forme mathématique ce qui implique un coût élevé de calcul d'intersection, et une possibilité restreinte de subdivision de l'espace.

Si les travaux dans ce cadre ont permis de dégager des parallélisations efficaces (schéma de Badouel ou Lefer), il nous faut encore attendre un gain en puissance des machines d'un facteur 1000 avant de pouvoir atteindre le temps réel.

La parallélisation du Lancer de Rayon associée à un espace voxel a été très peu étudiée. La subdivision voxel permet intrinsèquement le même gain qu'une subdivision régulière. Cependant elle présente la possibilité d'une parallélisation massive aisée, ce qui pourrait impliquer des performances supérieures aux autres techniques. C'est ce que nous nous proposons d'étudier dans le reste de ce texte.

CHAPITRE



Machines parallèles et machines massivement parallèles

La plupart des algorithmes utilisés pour la création d'images de grande qualité visuelle (Lancer de Rayon, Radiosité...) sont d'une complexité telle que leur mise en oeuvre sur une machine classique nécessite encore de nos jours plusieurs dizaines de minutes de calculs.

Pour diminuer les temps de réponse des machines, on dispose de deux voies :

1. Améliorer les performances pures de la machine, et pour cela, augmenter leur fréquence de fonctionnement ou définir des unités de calculs plus complexes. Ces deux possibilités sont tributaires de l'avancement de la technologie (degré d'intégration et surface maximale utilisable sans trop de déchet en fabrication).
2. Paralléliser les traitements, c'est-à-dire utiliser plusieurs processeurs pour réaliser le traitement complet.

Il existe plusieurs classification des machines parallèles, suivant le point de vue utilisé. Nous distinguerons dans un premier temps les architectures en fonction du type de parallélisme. Nous donnerons un deuxième classement suivant l'organisation mémoire utilisée.

Désirant utiliser une implémentation massivement parallèle du Lancer de Rayon, nous nous efforcerons de spécifier les particularités de ce type d'architecture.

2.1 Les différents modes de parallélisme

2.1.1 Flot de données

Prenons un ensemble de données $\{D_0, D_1, ..., D_d\}$ et de processus $\{P_0, P_1, ..., P_p\}$. On peut distinguer deux façons d'enchaîner ces deux ensembles. Prenons l'exemple d'une boucle de calcul pour montrer les deux types de parallélisation possible :

```
Pour i = 0 jusqu'à n Faire
|    Tableau1[i] = Tableau1[i] + 1
|    Tableau2[i] = Tableau1[i] * 100
FinPour
```

Mode Pipeline

Dans ce mode, le flot de données est unique. Les données ne sont donc pas indépendantes, chacune étant fonction du résultat amont de la chaîne de calcul (Figure 2.1).

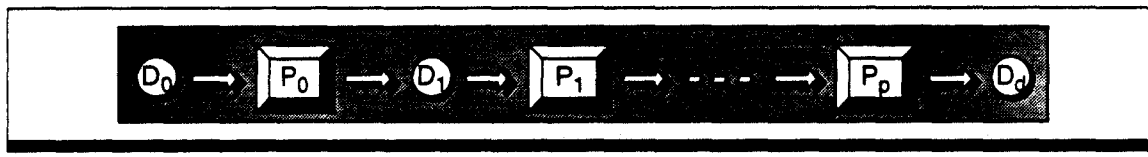


Figure 2.1 ***Mécanisme pipeline.***

Dans le cas de l'algorithme pris en exemple, le processeur P_0 exécute la première instruction de la boucle, le processeur P_1 la deuxième instruction. Les données sont envoyées les unes après les autres (`Tableau[0]`, `Tableau[1]`...).

Le temps de cycle du pipeline, c'est-à-dire l'intervalle de temps entre la sortie de deux valeurs, est conditionné par le temps de cycle de l'unité la plus lente. Il y a donc tout intérêt à équilibrer la charge des différents processeurs. Cela peut se faire en découpant les tâches les plus importantes, ce qui allonge le pipeline. L'inconvénient dans ce cas est que le temps nécessaire pour sortir la première valeur (temps de *set-up*) est lui aussi allongé. Il y a donc, suivant le type d'application à implémenter, à trouver un bon compromis.

Prenons Ti le temps de cycle d'une unité, correspondant au temps d'exécution d'une instruction, ou d'un bloc non décomposable d'instructions. Soit l, le nombre d'instructions dans la boucle; le pipeline sera constitué de l unités, dans le cas optimal. En négligeant le temps de *set-up*, les n valeurs sont calculées en un temps $n \cdot Ti$.

Mode Parallèle

Dans ce mode, les données sont indépendantes les unes des autres. On dispose d'un flot d'entrée parallèle de données, et d'un flot parallèle de sortie (Figure 2.2).

Chaque processeur exécute donc toute la boucle de notre exemple, chacun sur une valeur d'entrée différente (P_0 : `Tableau[0]`, P_1 : `Tableau[1]`...).

Chaque processeur exécute l instructions en un temps $l \cdot Ti$. Si l'on dispose de n unités, les n valeurs sont donc calculées en un temps $l \cdot Ti$.

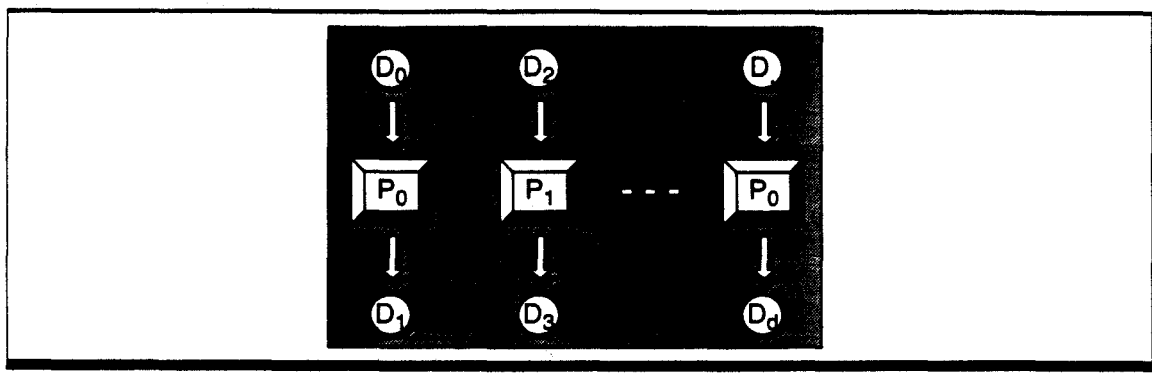


Figure 2.2 ***Mécanisme parallèle.***

Utilisation

Ces deux modes de fonctionnement ne sont pas complémentaires; ils sont même opposés : le pipeline n'est utilisable que lorsqu'il existe un flot de données modifiées au fur et à mesure des traitements; le mode parallèle nécessite des traitements sur des données indépendantes.

L'opposition entre les deux mécanismes se retrouve dans le fait que le temps d'exécution d'un algorithme est fonction du nombre de valeurs dans le cas du pipeline, et du nombre d'instructions dans le cas parallèle. La même dualité existe pour le nombre d'unités.

2.1.2 Flot de contrôle

Dans le cas des architectures parallèles, telles que définies précédemment, on distinguera deux modes de fonctionnement, suivant la façon dont sont contrôlées les unités de calcul.

Mode SIMD[1]

Dans ce mode, les unités de calcul parallèles sont commandées par une unité de contrôle unique (Figure 2.3). Elles effectuent donc la même instruction en même temps. Le fonctionnement est synchrone.

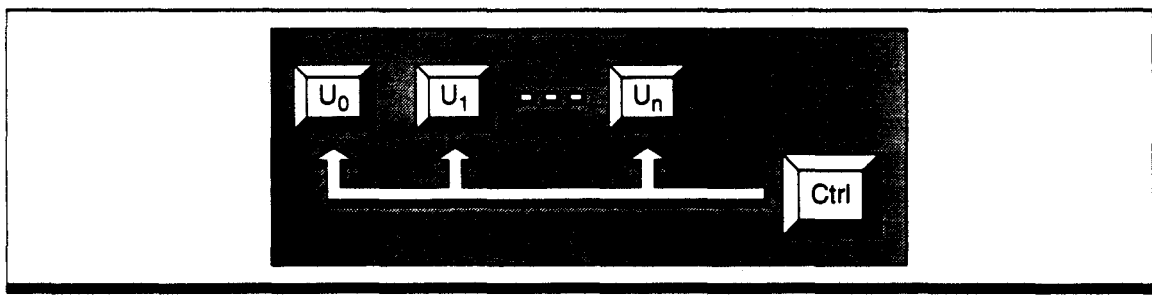


Figure 2.3 ***Mode SIMD.***

Pour permettre un traitement conditionnel, les unités de calculs disposent, généralement, d'un drapeau *enable*, pouvant être positionné par un résultat de calcul. Lorsque le drapeau est levé, l'instruction courante présentée par l'unité de contrôle est exécutée. Dans le cas contraire, le résultat de cette instruction n'est pas mémorisé.

1. Single Instruction stream - Multiple Data stream.

Mode MIMD[1]

Chaque unité de calcul dispose de sa propre unité de contrôle, ce qui permet d'exécuter des tâches différentes sur chaque processeur (Figure 2.4). Le fonctionnement est par conséquent asynchrone.

Il est difficile de concevoir une application disposant d'un grand nombre de processus purement MIMD (effectuant chacun une tâche différente). En général, les machines massivement parallèle de ce type sont utilisées en mode *SPMD*[2], où chaque processeur exécute le même algorithme sur des données différentes. Ce mode se différencie du fonctionnement *SIMD*, en ce sens qu'il existe de vraies instructions de saut conditionnel, par exemple. Cela permet d'accroître l'efficacité des traitements fortement conditionnels, seules les instructions utiles étant exécutées.

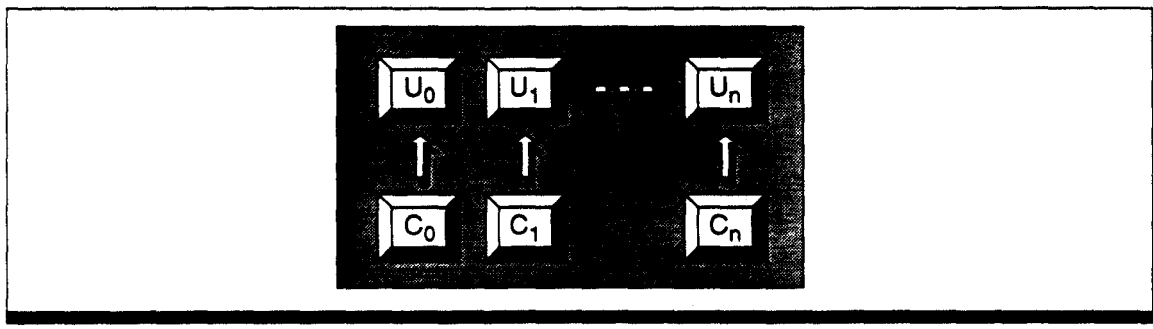


Figure 2.4 ***Mode MIMD.***

2.1.3 Parallélisme massif

Les premières machines massivement parallèles ont été, par souci de densité d'intégration, basées sur un contrôle SIMD. En effet l'utilisation d'un contrôleur centralisé permet de réduire la taille de chaque processeur à une simple unité opérative dotée d'un peu de mémoire de travail. On peut citer les machines classiques :

- le MPP (Massively Parallel Processor) construit autour d'un tableau de 128*128 processeurs élémentaires 1 bit, associés à une mémoire de 1Kbits,
- la Connection Machine disposant jusqu'à 65.536 cellules. Chaque unité contient une ALU 1 bit et une mémoire de 4Kbits,
- plus récente, la MassPar I, utilisant jusqu'à 16.384 noeuds. Les cellules ont une largeur de 4 bits, et contiennent 64Kbits.

Ce type de contrôle est plutôt adapté au parallélisme à grain fin[3]. Cependant un certain nombre d'applications nécessitent un parallélisme à gros grain, ou une puissance de calcul plus élevée au sein de chaque processeur (calculs météorologiques, simulations de particules...). Des machines massivement parallèles utilisant des processeurs généraux ont vu le jour pour répondre à cette demande (CM-5, projet TouchStone...). Utilisant des noeuds de calculs classiques, ces architectures proposent un contrôle MIMD.

Lors de la conception d'une architecture massivement parallèle, le choix du type de contrôle est très important. En effet, le schéma MIMD implique l'implémentation d'un circuit de contrôle et d'une mémoire de programme sur chacun des processeurs. Autrement dit, la taille des noeuds sera bien plus importante que dans le cas SIMD. D'autre part, il est nécessaire de mettre en oeuvre des dispositifs de synchronisation entre les processeurs, puisqu'il n'existe plus de contrôle centralisé entre eux.

Par conséquent, le choix MIMD ne se justifiera que lorsqu'il permet une efficacité plus importante de l'algorithme parallèle que l'on désire implémenter.

1. Multiple Instruction stream - Multiple Data stream.
2. Single Procedure stream - Multiple Data stream.
3. Le grain de parallélisme identifie la taille moyenne des blocs d'instructions.

2.2 Les différentes organisations d'accès mémoire

Dans le cas des machines parallèles se pose le problème de l'accès à la mémoire de travail. Il existe deux types de schémas de connexion entre les processeurs et la mémoire.

Mémoire partagée

Soient un ensemble $\{M_0, M_1, ..., M_m\}$ de bancs mémoires, et un ensemble $\{P_0, P_1, ..., P_p\}$ de processeurs. Le premier schéma permet à tout processeur P_i d'accéder à tout banc M_j, via un réseau d'interconnexion (Figure 2.5). Ce type de machine est parfois dénommé *SMPC*[1] ou *UMA*[2].

Réseau d'interconnexion

Il existe un certain nombre de réseaux d'interconnexion, les plus extrêmes étant :

- Bus unique. Il permet à un seul processeur d'accéder à un banc mémoire, à un instant donné. Il est nécessaire d'établir un protocole de réservation du bus, afin que les processeurs puissent coopérer et obtenir l'accès bus lorsqu'ils en ont besoin.
- Crossbar complet. Ce réseau, disposant de p entrées et m sorties, permet de réaliser une connexion entre chaque processeur et un banc mémoire, avec la restriction que deux processeurs ne peuvent avoir accès au même banc en même temps. Dans ce cas également, un protocole doit pouvoir permettre de modifier les connexions en fonction des demandes des processeurs.

Lorsque les processeurs accèdent souvent à la mémoire, le bus unique n'étant pas partageable, il constitue un goulot d'étranglement pouvant faire chuter les performances de la machine. Le réseau crossbar est, lui, très coûteux en terme de surface silicium et est assez complexe à gérer de manière dynamique. Des solutions intermédiaires ont été proposées, mais elles possèdent toutes l'inconvénient de l'un ou de l'autre des deux schémas précités.

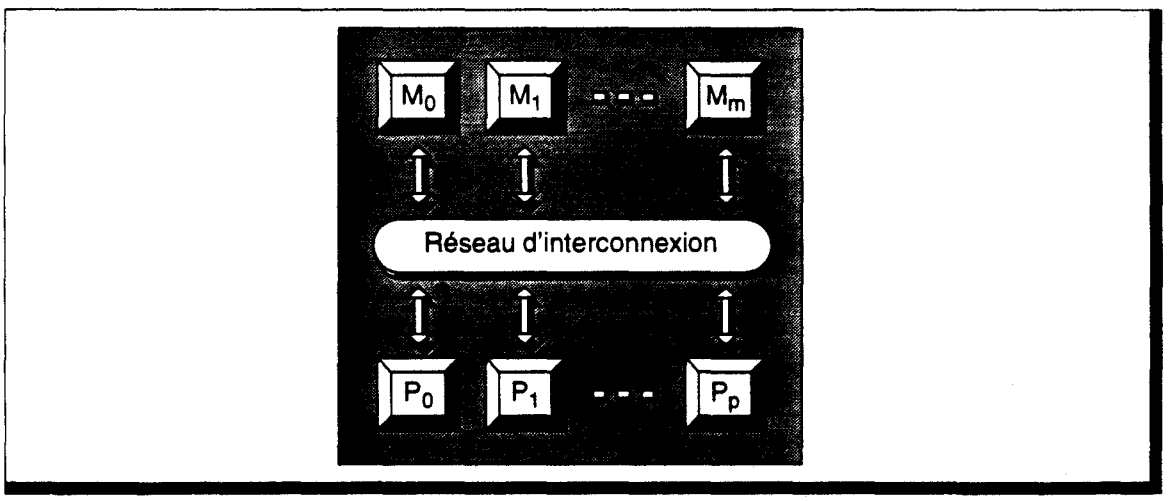


Figure 2.5 ***Schéma d'accès partagé à la mémoire.***

Echange d'informations

La communication d'informations entre deux processeurs se fait par simple écriture de l'un puis lecture de l'autre dans un emplacement mémoire connu. La diffusion d'informations à l'ensemble des processeurs se fait également très simplement : l'émetteur du message l'écrit en mémoire, les récepteurs vont ensuite le lire tour à tour.

1. Shared Memory Parallel Computer.
2. Uniform Memory Access.

Mémoire distribuée

Dans ce schéma, un processeur P_i est connecté de manière statique à un banc mémoire M_i. La mémoire est donc distribuée sur chaque processeur. Ces machines sont appelées *DMPC*[1] ou *NORMA*[2]. Il n'y a plus dans ce cas de conflit d'accès à la mémoire.

Lorsque deux processeurs désirent échanger des informations, il n'est pas possible, dans ce schéma, d'utiliser la mémoire. On ajoute donc un réseau d'interconnexion entre les processeurs, afin qu'ils puissent dialoguer.

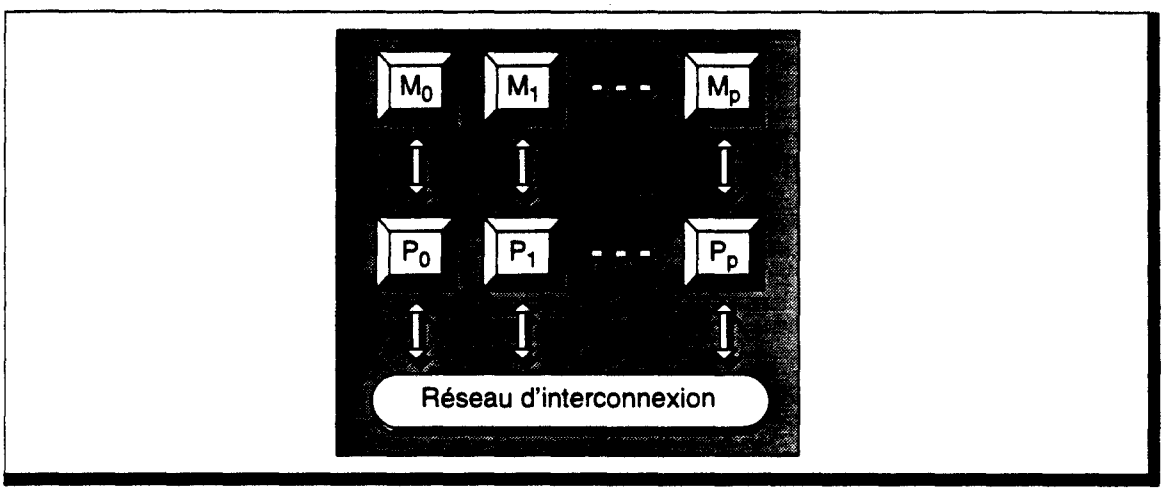


Figure 2.6 ***Schéma d'accès distribué à la mémoire.***

Réseau d'interconnexion

Le réseau d'interconnexion est souvent statique, et est caractérisé par sa topologie, c'est à dire par le graphe de connexion qu'il permet (Figure 2.7). Certaines machines disposent d'un réseau programmable, qui permet une configuration dynamique (définie au moment du chargement de la machine, et non modifiable en cours d'exécution) des connexions.

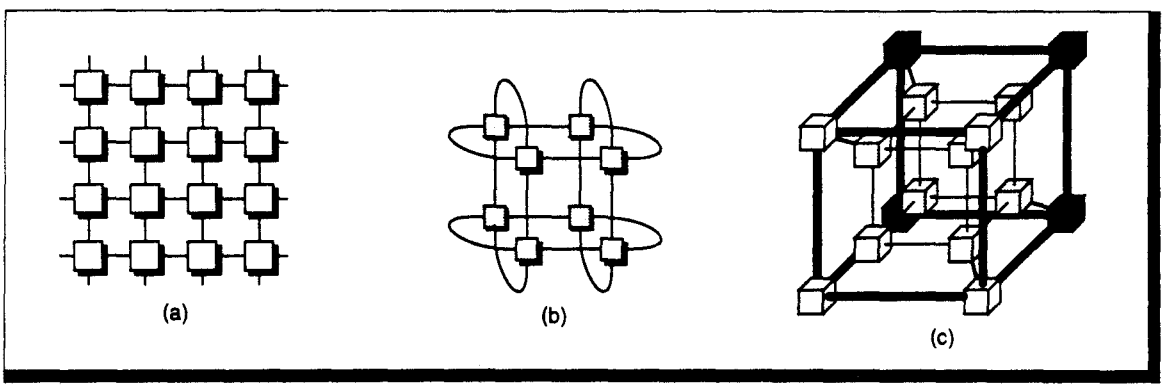


Figure 2.7 ***Exemple de topologie.*** *(a)- Grille 2D. (b)- Tore 2D. (c)- Hypercube 4D.*

Schéma de transfert d'informations

La plupart des machines *DMPC* ne possèdent pas de topologie complètement connectée, par souci de diminution de la complexité matérielle et du coût du réseau. Un processeur ne peut communiquer qu'avec un nombre restreint de ses voisins. Il est nécessaire d'établir un mécanisme permettant le routage de messages entre des noeuds *distants*[3] de la machine.

1. Distributed Memory Parallel Computer.
2. NO Remote Memory Acess.
3. Noeuds non directement connectés.

Un noeud de calcul est décomposable en trois unités (Figure 2.8) : l'unité de calcul (*P*), la mémoire (*M*) et une unité de communication (*C*). Cette dernière peut-être réalisée par logiciel ou par matériel. Elle prend en charge les transferts d'informations entre les noeuds.

On peut distinguer deux types de mécanismes de communication :

- *Store and Forward* (stocker et faire suivre). Les messages sont intégralement recopiés de noeud en noeud (Figure 2.8). L'unité de communication interrompt l'unité de calcul, lorsqu'elle reçoit un message, afin de réaliser le transfert en mémoire. Une fois le message complet, il est envoyé au noeud suivant sur le trajet. Ce dernier est en général déterminé par l'unité de calcul.

  Ce mécanisme est, par exemple, celui utilisé sur les architectures parallèles utilisant des Transputers de la famille T800 (ou précédents).

  Dans le cas où l'application implémentée sur une architecture de ce type est fortement communicante, les unités de calculs sont souvent interrompues pour réaliser le routage des messages. Il s'en suit donc une dégradation importante des performances globales du système.

  Pour éviter ce phénomène, l'unité de communication peut-être rendue autonome, et réaliser de manière transparente la transmission des messages jusqu'au noeud destination. On retrouve ce mécanisme, par exemple, sur les réseaux de communications entre stations de travail (réseau Ethernet), ou sur les Transputers T9000.

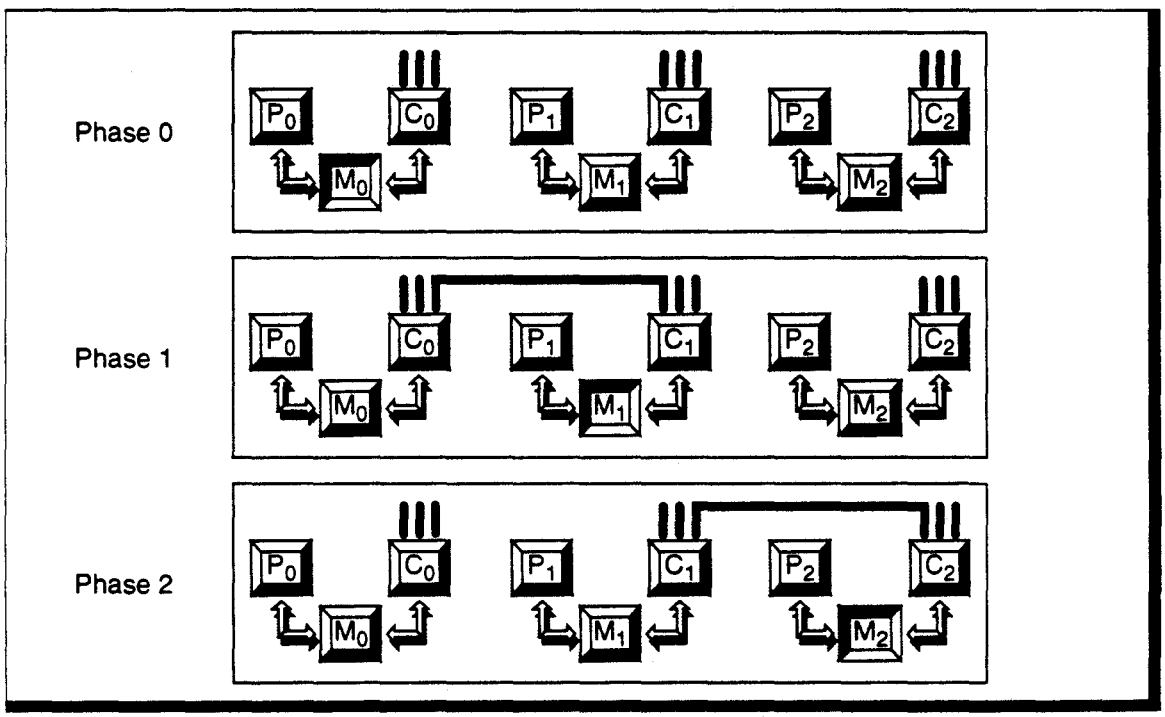


Figure 2.8 ***Routage par mécanisme Store and Forward.*** *Déroulement d'un transfert de message, noeud par noeud.*

- *Wormhole* (trou de ver). La technique précédente nécessite le stockage du message complet sur chacun des noeuds, ce qui peut demander beaucoup de mémoire locale. Une autre technique consiste à créer, temporairement, une liaison directe entre les deux processeurs extrêmes du trajet, par une pseudo commutation de circuit (Figure 2.9) : l'entête du message ouvre les voies de communications entre les noeuds du réseau; la suite du message est envoyé morceau par morceau (*flit*[1] *par flit*) en utilisant le chemin ainsi défini; la queue du message referme le circuit, et libère les unités de communications.

  Les unités de communications sont dans ce cas forcément autonomes.

1. Flow Control Digit : Unité minimale gérée par le dispositif de communication.

Un des inconvénients de cette technique est que durant tout le temps nécessaire à la transmission, les voies de communications sont réservées ce qui peut bloquer l'unité de calcul si elle désire émettre un message par l'une des ces voies. Il faut donc, autant que possible, limiter la taille des messages.

Ce type de transfert est utilisé, par exemple, sur les machines iPSC d'Intel.

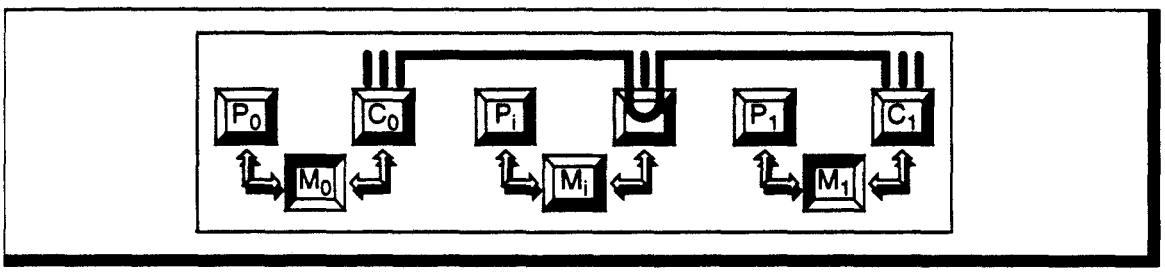


Figure 2.9 ***Routage par mécanisme Wormhole.***

Comparaison d'efficacité des deux modes de transfert

La technique du *Wormhole* permet de réduire d'un facteur supérieur à 10 le temps nécessaire à la transmission d'un message. Pour montrer cela, nous allons calculer le temps de latence[1] dans les deux cas [Dall90].

Soit à transmettre un paquet de L bits. Supposons que le canal de communication permette le transfert de W bits/cycle. Nous notons T_C la durée d'un cycle, et D la distance séparant les noeuds émetteur et récepteur. Nous nous plaçons dans le cas le plus favorable, où le réseau est non chargé; autrement dit, il n'y a aucun temps d'attente sur les noeuds entre la réception et la transmission du message.

- *Store and Forward* : Le paquet complet doit être reçu par un noeud, avant d'être envoyé au suivant (Figure 2.10.a). Le temps de latence est dans ce cas :

$$T_{\text{Store\_And\_Forward}} = \frac{L}{W} \cdot D \cdot T_C$$

- *Wormhole* : Dès qu'un flit est reçu, il est émis (Figure 2.10.b). Nous supposerons qu'un flit contient W bits. La latence est alors de :

$$T_{\text{Wormhole}} = \frac{L}{W} \cdot T_C + D \cdot T_C$$

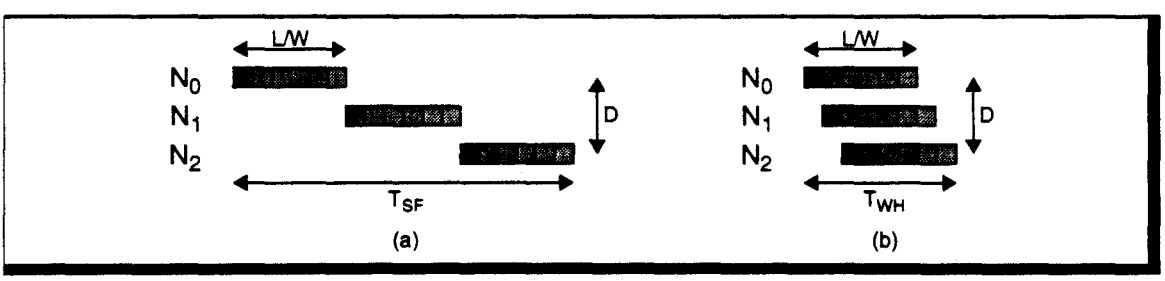


Figure 2.10 ***Latence de transmission.*** *(a)- Technique du Store and Forward. (b)- Technique du Wormhole.*

Le gain est donc évident, d'autant plus important que les quantités L/W et D sont grandes.

Dans le cas d'applications fortement communicantes[2], il peut malgré tout être plus intéressant d'utiliser la technique *store and forward* qui permet de libérer immédiatement les unités de communications. La latence des messages sera plus importante, mais le nombre global de messages acheminés à un instant donné peut être plus élevé ce qui peut améliorer le fonctionnement global de l'application.

1. Durée comprise entre le début de la transmission et la fin de la réception d'un message.
2. Applications où le ratio communication/calcul est grand devant 1.

Protocole d'échange d'informations

Les noeuds des architectures DMPC ont, en général, un fonctionnement asynchrone. Il est donc nécessaire d'établir un mécanisme permettant d'assurer que deux processus arriveront l'un à émettre et l'autre à recevoir le message à transférer. On distingue deux protocoles :

- *Echange par Rendez-Vous.* Ce mécanisme permet de synchroniser les processeurs au moment de la communication : le processus émetteur ou récepteur reste bloqué, jusqu'à ce que son *partenaire* déclenche le transfert du message (Figure 2.11). C'est le mode de fonctionnement adopté pour les processeurs de la famille des Transputers. Il est très simple à mettre en oeuvre, mais il est pénalisant dans les algorithmes du type *producteur-consommateur.* En effet, il ne permet pas de régulation des flux de transferts, étant donné le synchronisme de la communication : à *une* émission correspond, au même rythme, *une* réception.

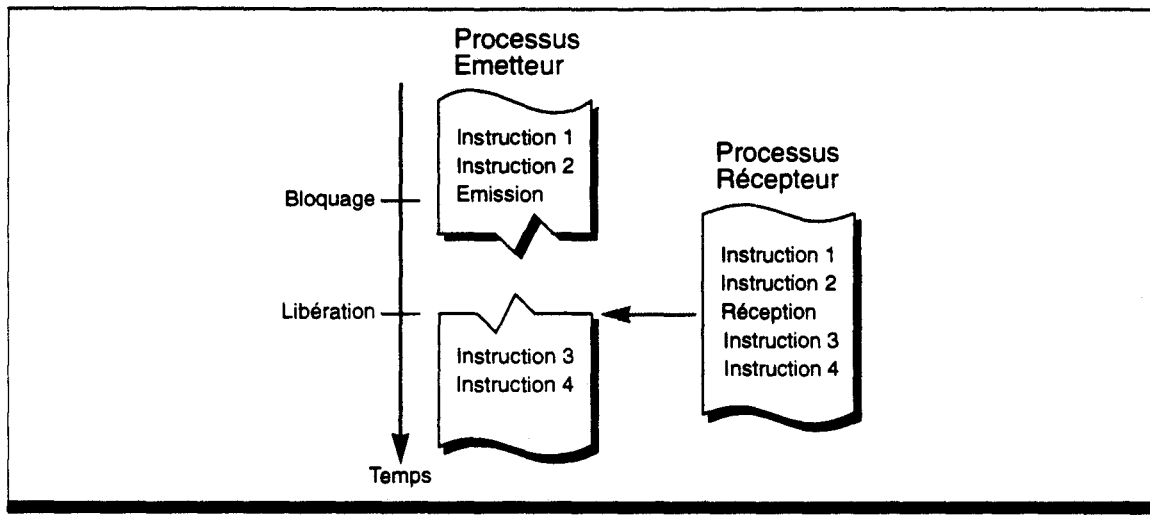


Figure 2.11 ***Echange synchrone par Rendez-Vous.***

- *Utilisation de files d'attente.* Les unités de communications disposent de files d'attentes dans lesquels sont déposés les messages à émettre et les messages à recevoir. Une instruction d'émission dépose le message dans la première file, sans blocage (hormis en cas de remplissage de la file). L'unité de communication transfère les messages de la file *de sortie* dans la file *d'entrée* du noeud destination. L'unité de communication de ce noeud effectuera la transmission vers l'unité de calcul lors de l'instruction de réception (Figure 2.12).

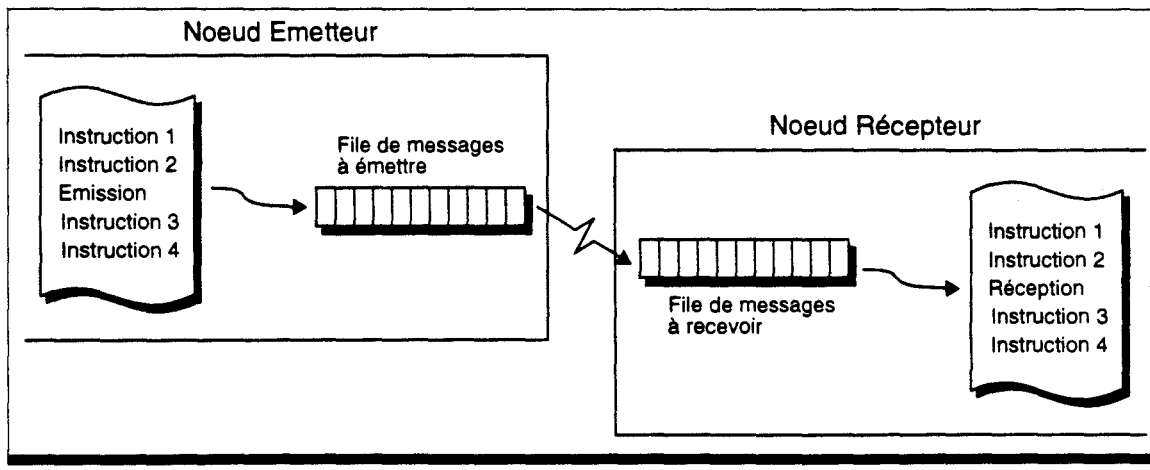


Figure 2.12 ***Echange asynchrone par files d'attente.***

Ce mécanisme permet donc un échange asynchrone de message, sans blocage des unités de calcul, avec cependant le problème de la gestion des saturations des files d'attentes. Il

nécessite une unité de communication indépendante de l'unité de calcul, disposant de sa propre mémoire de stockage. Il permet, de plus, une régulation automatique des flux de transfert.

Organisation mémoire et parallélisme massif

Le réseau d'interconnexion des architectures à mémoire partagée est réalisé par commutation de circuits. Il semble donc impensable d'utiliser ce type d'organisation pour un nombre très important de processeurs.

De fait, toutes les machines massivement parallèles existantes utilisent un schéma de mémoire distribuée. On trouve plusieurs types de topologie : les premières machines utilisaient une connexion en hypercube qui semblait, alors, plus intéressante, car il s'agit de la topologie présentant le diamètre[1] le plus faible. Cependant pour simplifier les organes de routage, mais aussi les algorithmes, on utilise de plus en plus la grille 2D (ou plus souvent le tore 2D).

Le choix du type de transfert d'informations est plus délicat. La technique *wormhole* permet de réduire considérablement le temps de transfert d'un message entre deux processeurs distants. Cependant, elle nécessite la mise en place d'un dispositif matériel spécifique plus coûteux que dans le cas de la méthode *store and forward*. D'autre part, si l'on désire effectuer un traitement sur le message durant le transfert, avec, par exemple, modification du noeud destinataire, il peut être nécessaire de posséder la totalité du message. Dans ce cas le modèle *store and forward* est plus adapté.

Là aussi, le choix de la technique de transfert, pour une machine spécialisée, sera fortement dépendant de l'algorithme à implémenter.

2.3 Mesure d'efficacité des machines massivement parallèles

Les outils classiques d'analyse et de quantification des performances des architectures parallèles (cf. Annexe C) ne prennent pas en compte un certain nombre de caractéristiques du parallélisme massif. En effet, pour un grand nombre d'applications (et en particulier en synthèse d'images) seul un nombre restreint de processeurs est utilisé dans la résolution du problème.

Prenons l'exemple d'une matrice de 100x100 processeurs, chacun prenant en charge un pixel d'un écran de résolution 100x100. Si cette machine est destinée à l'affichage de segments de droite, il est évident que seuls les processeurs dont le pixel appartient à la droite sont utiles pour son tracé. Nous disposons donc ici de 10.000 processeurs, là où 100 suffiraient (si la longueur des droites est de 100 pixels).

En posant t, le temps nécessaire au calcul d'un pixel, nous obtenons de façon théorique

$$T(1) = 100 \cdot t \text{ et } T(10.000) = t,$$

avec $T(n)$: le temps de calcul sur une machine à n processeurs

soit une efficacité bien faible :

$$Eff = \frac{T(1)}{n \cdot T(n)} = \frac{100 \cdot t}{10.000 \cdot t} = 0,01$$

L'application que nous décrivons est à temps fixe (et non à taille fixe), nous pouvons donc utiliser le modèle de Gustafson pour calculer la fraction séquentielle du problème, c'est-à-dire, la partie du code parallèle nécessaire pour prendre en compte l'aspect séquentiel inhérent du problème. Nous avons alors $f = 99\%$! Il va de soit que cette valeur est aberrante. Nous proposons une nouvelle mesure de l'efficacité.

1. Le diamètre d'un réseau est la distance la plus longue existant entre deux processeurs du réseau.

Modification du modèle de Gustafson [Gust88]

Nous allons introduire dans les mesures la valeur R de rendement, c'est-à-dire le pourcentage de processeurs utiles par rapport au nombre total de processeurs disponibles.

Dans le modèle de Gustafson, on considère le temps mis par un processeur unique pour réaliser le même calcul que les n processeurs. Or, dans le cas présent, seuls $R \cdot n$ processeurs sont utilisés. Par conséquent, nous redéfinissons $T(1) = s + R \cdot n \cdot p$, avec s représentant la partie séquentielle, et p représentant la partie parallélisable *pure*.

Nous pouvons alors exprimer la valeur d'une efficacité *relative* :

$$Eff_R(n) = \frac{T(1)}{R \cdot n \cdot T(n)}, \text{ soit } Eff(n) = R \cdot Eff_R(n)$$

Nous posons de plus :

$$Eff_R(n) = \frac{Acc(n)}{R \cdot n} \text{ et } f_R = \frac{R \cdot n - Acc(n)}{R \cdot n - 1}$$

Ces nouvelles mesures permettent une analyse correcte du comportement de l'application, mais il n'en reste pas moins que l'efficacité réelle ($Eff(n)$) est peu importante.

Critère de coût

Les machines massivement parallèles peuvent donc permettre une accélération importante, mais au prix d'une perte d'efficacité. Un autre critère de comparaison des machines parallèles est avancé par Dally [Dall90] : le rapport performance/coût.

Le coût d'une machine est lié à la surface de silicium qu'elle nécessite. On peut donc définir une nouvelle mesure de performance, comme étant le rapport entre l'accélération obtenue et l'accroissement de surface :

$$Perf(n) = \frac{A_1 \cdot T_1}{A_n \cdot T_n}$$

avec A_i : Surface de i processeurs.

Le processeur utilisé dans les architectures massivement parallèles, que l'on appelle classiquement *processeur élémentaire*, est de petite taille par rapport à un système monoprocesseur. On a donc $A_n/n < A_1$, ce qui implique que $Eff(n) < Perf(n)$.

Processeur 4 bits	$5M\lambda^2$	Processeur 32 bits	$50M\lambda^2$
Mémoire 4k bits	$1M\lambda^2$	Mémoire 4M bits	$1000M\lambda^2$

Table 2.1 ***Taille silicium approximative de différents composants.*** *λ représente la moitié de la largeur minimale de ligne dans la technologie utilisée [Dall90][Mead80].*

Prenons l'exemple d'une machine possédant 1024 processeurs 4 bits, chacun disposant d'une mémoire de 4k bits, et d'un système monoprocesseur 32 bits avec une mémoire équivalente au multiprocesseur, soit 4M bits. Les surfaces associées sont indiquées dans la Table 2.1. Avec ces données, on obtient : $A_1 = 1050M\lambda^2$, et $A_n = 1024 \cdot 6M\lambda^2$, soit $Perf(n) \approx 175 \cdot Eff(n)$.

Cette mesure de performance montre que les architectures massivement parallèles possèdent un intérêt certain, même si le rendement obtenu est faible, sous la seule condition que les processeurs élémentaires soient de petite taille. En effet, en liant la mesure de performance à celle d'efficacité relative, nous obtenons l'expression :

$$Perf(n) = \frac{A_1 \cdot n}{A_n} \cdot R \cdot Eff_R(n)$$

Par conséquent, plus le rendement est faible, plus il faut que A_n/n (taille d'un processeur élémentaire) soit petit devant A_1.

Cependant, il existe une relation directe entre la taille d'un processeur et sa *puissance*, ce qui influe sur la valeur d'efficacité relative. Pour les applications requérant de la puissance de la part des processeurs élémentaires, le rapport de coût est donc faible, ce qui implique qu'il faut rechercher un rendement correct.

Il faut, bien entendu, relativiser les remarques précédentes, en fonction de l'importance du système monoprocesseur pris en référence. Il va de soi que la taille d'une station de travail est bien différente de celle d'un super-calculateur. Cela permet des processeurs élémentaires d'importance variable, suivant le domaine d'application où l'on se place.

2.4 En conclusion

Lors du développement d'une architecture parallèle, plusieurs caractéristiques sont à prendre en compte, afin d'assurer de l'efficacité de la machine.

Il faut tout d'abord veiller à minimiser la fraction séquentielle du problème, en limitant les communications par exemple.

Il faut ensuite tendre à un rendement maximum, en équilibrant au mieux les charges des processeurs.

Il faut enfin tenir compte du coût de l'architecture. Le facteur d'accélération doit être plus important que le surcoût induit, pour que la machine présente un intérêt réel. On veillera pour cela à choisir le schéma de contrôle et de transfert d'informations le plus adapté aux applications envisagées.

2ème Partie

Vers une Machine Voxel dédiée au Lancer de Rayon Discret

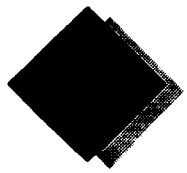
L'utilisation du *monde discret* dans le cadre de la synthèse d'images est assez récente, et de nombreux problèmes de définition des entités *objets discrets*, et de géométrie discrète sont encore à résoudre. Les méthodes de *rendu* de ces objets posent également des difficultés si l'on désire obtenir une qualité visuelle correcte; cette qualité est d'autre part liée à la méthode de discrètisation choisie. Enfin les techniques permettant leur visualisation doivent prendre en compte la nature discrète de l'espace de représentation utilisé. Il nous semble indispensable de ne pas dissocier l'étude de ces processus, car ils contribuent ensemble et de manière dépendante à la qualité de l'image synthétisée.

Le Lancer de Rayon Discret met en oeuvre ces trois traitements (voxelisation, rendu, visualisation). Il permet d'aborder un grand nombre des problèmes algorithmiques spécifiques au monde voxel, et constitue donc une excellente base d'étude. Il permet également d'atteindre une accélération importante de la technique du Lancer de Rayon, ce qui justifie en soi son utilisation.

Enfin, le Lancer de Rayon Discret permet d'exploiter d'un parallélisme massif, mais nécessitant l'utilisation d'une architecture dédiée. Nous pensons qu'une telle machine peut permettre la visualisation par Lancer de Rayon Discret en un temps proche du temps réel.

Le but des chapitres suivants est la définition générale de cette machine. Nous commencerons par proposer une définition d'une Machine Voxel et nous étudierons son adéquation avec les différents domaines d'applications de la synthèse d'images. Nous aborderons ensuite chacun des processus du Lancer de Rayon Discret, et nous étudierons une implémentation matérielle de ceux-ci.

CHAPITRE



Organisation d'une Machine Voxel

Nous proposons ici l'organisation générale d'une Machine Voxel, afin de pouvoir introduire les notions architecturales dont nous aurons besoin dans l'étude des algorithmes de visualisation volumique.

Nous présenterons également quelques machines utilisant l'espace discret $3D$ comme espace de représentation et stockage de données.

3.1 Définition d'une Machine Voxel parallèle

3.1.1 Notion de Machine Voxel

Une Machine Voxel dispose de quatre composants (Figure 3.1) : une mémoire de stockage des données volumiques, une unité de conversion des objets en voxels, une unité de manipulation et une unité d'affichage. Le processus de manipulation est plus spécifique aux applications de visualisation médicale ou scientifique, et ne sera pas utilisé dans le cadre du Lancer de Rayon Discret.

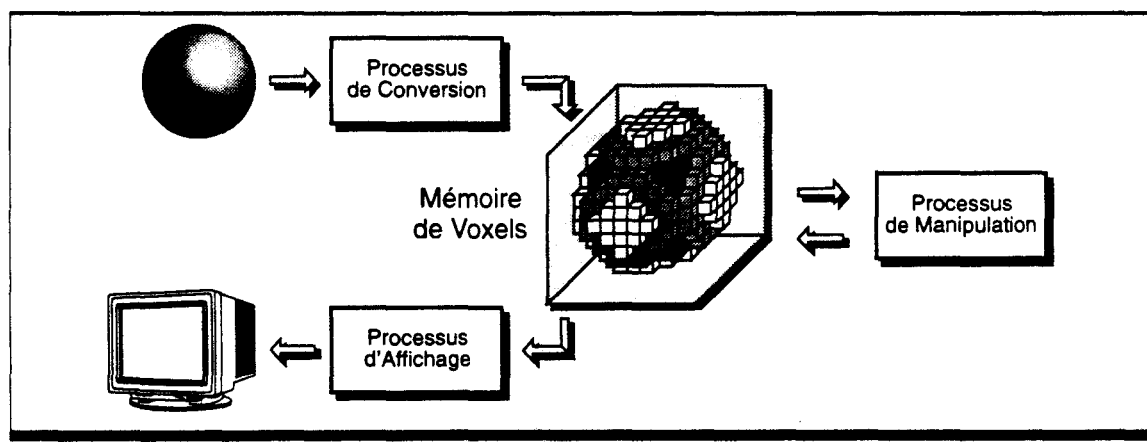


Figure 3.1 ***Organisation générale d'une Machine Voxel.***

Nous utiliserons comme base d'étude, pour le Lancer de Rayon Discret, un cube de dimension $N_V = 512$, soit un espace discret de $512 \times 512 \times 512$ voxels. Ce choix est suffisant pour obtenir une image de taille raisonnable, sans être trop important en terme d'espace mémoire à mettre en oeuvre.

3.1.2 Définition de l'entité voxel

Le voxel tel que nous le définissons est une entité volumique élémentaire. Cela implique qu'il n'est porteur que de la connaissance d'une seule caractéristique de matière. Il peut être un bon équivalent de la structure atomique des objets. Cependant, on se heurte à un facteur d'échelle totalement différent. En effet si les atomes ne sont pas visibles par l'oeil, il en est différemment pour le voxel.

Nous avons indiqué qu'en général la taille du cube voxel était voisine de la taille de l'image générée (ou inversement). La couleur de la matière (i.e. de l'objet) contenue en un voxel est donc directement visible en un pixel de l'écran (ou tout du moins apporte une grande contribution à ce dernier).

Par conséquent, si deux objets géométriques existent en un même voxel, seul un des deux sera réellement conservé, et le voxel prendra la couleur de ce dernier. On aboutit alors à une erreur visuelle (Figure 3.2) : suivant la direction selon laquelle est vu le cube voxel, un objet *normalement* non visible peut apparaître (cas de la vue suivant A). Cela provient du fait que le voxel doit être considéré comme possédant un volume non nul, alors qu'il est utilisé pour contenir un objet d'épaisseur virtuellement nulle.

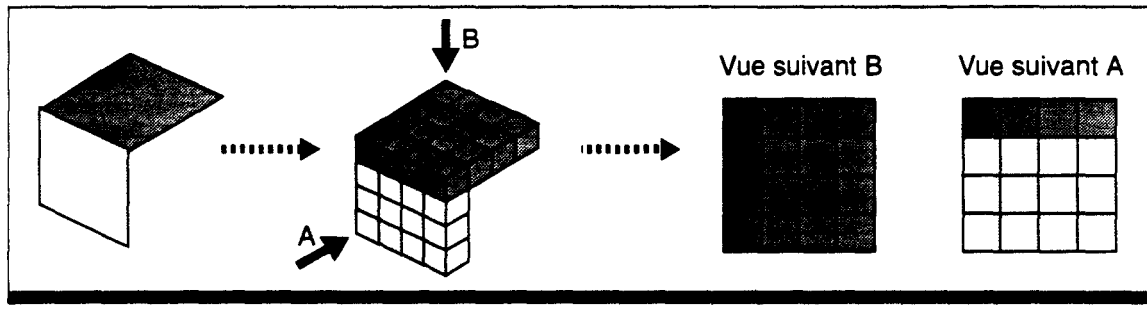


Figure 3.2 ***Problème de l'atomicité du voxel.*** *La voxelisation des deux polygones de gauche peut amener une erreur visuelle.*

Ce phénomène n'existe pas en rendu classique, où la discrètisation est fonction du point de vue. Ceci n'est pas réalisable dans le cas voxel, car la direction de vision est inconnue *a priori*. Pour le Lancer de Rayon, elle n'est même pas déterminable, un objet pouvant être vu de plusieurs points de l'espace par réflexion sur d'autres objets.

Pour résoudre ce problème, nous pouvons tenter d'apporter plusieurs solutions, que nous listons ci-dessous.

Voxel hétérogène

Contrairement au voxel que nous appellerons *homogène*, nous pouvons définir un voxel *hétérogène* permettant le stockage de plusieurs informations de matière (i.e. plusieurs objets).

Pour résoudre le problème énoncé ci-dessus, nous pouvons conserver une information par *face* du voxel. Mais cela entraîne de nouveaux problèmes :

- pour certaines directions de vue, plusieurs faces sont visibles. Si l'information désirée est la couleur visible, il est possible d'effectuer un moyennage sur les couleurs des différentes faces. Mais si l'on désire récupérer une autre caractéristique de la matière, telle que son type par exemple, comment peut-on décider de celle à prendre en compte?
- il est nécessaire de mettre en oeuvre un algorithme de choix de l'objet à conserver pour chacune des faces qui peut ne pas être simple si beaucoup d'objets existent en ce voxel; il peut être nécessaire d'effectuer un tri en profondeur des surfaces vues depuis chaque face.

Caubet propose pour la machine Voxar (cf § 3.3 - p. 69) de conserver une liste des objets existants en chaque voxel. Si cela est utilisable dans le cas du Lancer de Rayon, les inconvénients énoncés ci-dessus existent toujours pour des mécanismes de projection plus simples, tels que ceux utilisés par les machines de visualisation médicale (cf § 3.2 - p. 62). De

plus, cette méthode nécessite la mise en place d'un espace mémoire de taille variable pour chaque voxel, ou une limitation du nombre maximum d'objets mémorisables en un point de l'espace.

En tout état de cause, l'hétérogénéité entraîne une augmentation de la taille mémoire nécessaire au stockage d'un voxel.

Sur-échantillonnage voxel

Une autre méthode pour conserver un plus grand nombre d'objets consiste à échantillonner l'espace voxel plus finement que l'espace image. On dispose ainsi de plusieurs voxels par pixel-écran. La projection du cube nécessite également un moyennage de valeurs, ce qui n'a pas de sens si l'information désirée est d'un type autre qu'une couleur.

Par rapport à la solution précédente, aucun algorithme particulier de détermination d'objet à conserver n'est à mettre en place. La mémoire d'un voxel est ici de taille fixe car elle ne contient la description que d'un seul objet.

Cependant, le sur-échantillonnage, s'il permet d'améliorer la qualité globale de l'image finale, ne fait que diminuer les effets indésirables, qui restent présents, mais à plus faible contribution sur l'image générée.

Multi-cube voxel

Enfin, il peut exister une troisième méthode. Si un voxel ne peut conserver plusieurs objets, alors on peut utiliser plusieurs voxels, un par objet. Cela revient en fait à avoir un cube voxel par objet de la scène.

Il est certes inconcevable d'utiliser un cube complet par objet, on limitera donc la taille de chaque espace voxel à la taille du cube englobant de chaque objet.

Cette solution implique également une gestion dynamique de la mémoire, ce qui semble difficilement compatible avec une machine câblée spécifique.

De plus, il faut mettre en oeuvre un algorithme de projection très particulier.

En résumé

Il est possible d'éviter, ou de limiter, l'erreur de discrètisation évoquée plus haut, mais au prix d'une augmentation mémoire importante. Cependant, il n'est possible de déterminer facilement que la couleur visible en un voxel, qui peut être constituée par la moyenne des couleurs des objets visibles en un pixel. Pour tout autre attribut de matière, il ne peut être récupéré que l'information d'un seul objet, ce qui nécessite un algorithme de projection plus précis que celui habituellement utilisé sur les Machines Voxels. Le Lancer de Rayon est un excellent candidat pour résoudre ce problème.

Pour les études qui suivent, nous considérons une notion de voxel simple, i.e. avec une seule information de matière, sachant que toute extension est possible.

3.1.3 Machine Voxel et synthèse d'images

Le choix de l'utilisation d'un cube mémoire contenant la scène voxelisée implique un certain nombre de limitations et de contraintes, qui ne sont pas compatibles avec toutes les applications classiques de la synthèse d'images. Les principales contraintes, ou caractéristiques, sont :

- la scène visualisée est complètement contenue dans le cube voxel,
- tous les objets de la scène sont voxelisés à la même fréquence d'échantillonnage, i.e. tous les voxels ont la même *taille*,
- il n'y a plus de notion globale d'objet, i.e. le cube contient un ensemble de voxels possédant des attributs, et non des objets géométriques.

On peut distinguer deux types de scène : celles ayant une profondeur de champ importante (i.e. dont l'arrière plan est très éloigné, typiquement les scènes d'*extérieur*), et inversement celles dont la profondeur de champ est limitée (scènes d'*intérieur*, ou images du type CAO).

Dans le premier cas, seule une partie restreinte de la base de données est visible, il n'est donc pas envisageable de voxelisée directement l'ensemble de la scène si l'on désire conserver un échantillonnage correct. Il faut alors recharger le cube voxel à chaque modification de la position de l'observateur. De plus, l'importance de la profondeur de champ nécessite d'effectuer une pré-transformation perspective, afin que l'échantillonnage des objets situés en avant plan soit suffisamment précis. Cela implique une distorsion de l'espace, dont il faudra tenir compte lors du Lancer de Rayon, les angles et les distances n'étant pas conservés. Ce problème se rencontre dans toutes les implémentations utilisant une subdivision finie de l'espace $3D$.

Dans le second cas (scène d'intérieur, ou image de CAO), on peut envisager la voxelisation de la scène complète, en utilisant une post-projection perspective. Cependant, l'utilisateur peut désirer observer une partie restreinte de la scène, en se rapprochant d'un objet. Il est dans ce cas nécessaire d'effectuer un *zoom* sur les objets proches de l'observateur. On se heurte alors à un problème d'échantillonnage, la projection d'un voxel pouvant représenter une zone importante de l'écran. Afin que l'image générée puisse être de qualité, il est indispensable que la projection d'un voxel soit de dimension inférieure ou égale à la dimension d'un pixel. Il est par conséquent également nécessaire de procéder à un rééchantillonage de la scène en fonction de la position de l'observateur, et donc à un rechargement du cube voxel.

Un autre problème auquel on peut se heurter, en visualisation volumique, apparaît en animation, lorsque l'on désire déplacer un objet. En effet, il faut effectuer le *vidage* des voxels dans lesquels l'objet était présent, puis au rechargement de celui-ci à sa nouvelle position. Cette première opération peut se faire simplement, si l'on mémorise un numéro d'objet au sein de chaque voxel occupé. Cependant, si un voxel est situé à l'intersection de deux objets, la suppression d'un de ceux-ci impose de *reremplir* le voxel avec les attributs du second objet présent à cet endroit. Cette gestion pouvant être assez complexe, on préférera procéder à une voxelisation complète de la nouvelle scène.

Utilisation des Machines Voxels

A partir des remarques précédentes, on peut donc distinguer trois domaines d'applications.

Scènes 'statiques'

Les caractéristiques de ce type d'application sont les suivantes :
- La scène est contenue intégralement dans le cube,
- Aucun objet n'est en mouvement,
- L'observateur est situé en dehors de la scène,
- L'observateur est à distance constante du cube voxel.

Dans ce cas, il n'est pas nécessaire de procéder à un rechargement de la scène. Tout le potentiel du rendu volumique est utilisé.

Scènes 'dynamiques'

Les caractéristiques de ce type d'application sont :
- La scène peut ne pas être totalement contenue dans le cube,
- Un objet peut être animé,
- L'observateur peut être situé à l'intérieur de la scène,
- La profondeur de champ est faible.

Dans ce cas, il est nécessaire de recharger la scène à chaque changement du point de vue, ou à chaque mouvement d'un objet. Il faut donc veiller à ce que le processus de voxelisation soit suffisamment efficace si l'on désire un affichage rapide de l'animation.

L'apport du rendu volumique peut être négligeable en comparaison des méthodes classiques de rendu, sauf si l'on considère que le milieu est participatif, i.e. que la trajectoire d'un rayon est influencée par le milieu qu'il traverse. Ce traitement peut être implémenté de manière

naturelle sur une Machine Voxel, puisque les rayons traversent *réellement* l'espace, alors qu'il est plus difficile à prendre en compte dans les algorithmes classiques. Nous pensons qu'il s'agit ici d'un des domaines pour lequel une Machine Voxel est la plus adaptée.

Scènes d'extérieur

La principale caractéristique de ce type d'application est que la scène possède une profondeur de champ importante, ce qui implique une pré-transformation perspective. La visualisation volumique n'est pas adaptée à ce domaine de la synthèse d'image.

3.1.4 Machine Voxel parallèle

Les trois processus de la définition que nous avons proposée pour une Machine Voxel, pourraient être réalisés par logiciel sur une machine classique, les voxels étant stockés en mémoire centrale. Effectuons une estimation rapide de la puissance nécessaire : on dispose de 512^3 voxels, que l'on désire afficher à une fréquence de 25 images par seconde; il nous faut donc pouvoir manipuler près de 3,5 milliards de voxels par seconde, ce qui implique une puissance de plusieurs dizaines de Gips[1]. Les bandes passantes pour le transfert des informations sont du même ordre de grandeur. La visualisation volumique est un domaine où les monoprocesseurs actuels ne disposent pas encore de suffisamment de puissance, d'où la nécessité d'étudier des architectures matérielles spécifiques. L'utilisation des diverses machines disposant d'accélérateurs graphiques *standards* a été proposée, mais elle ne permet pas la manipulation en temps réel de voxels, car elles sont plus particulièrement dédiées à l'affichage d'objets polygonaux.

Pour permettre d'effectuer les divers traitements en un temps proche du temps réel, il peut être nécessaire de paralléliser les processus, mais il faut surtout utiliser une mémoire voxel à accès multiples, afin de minimiser les débits nécessaires aux transferts. En effet, le stockage de 512^3 voxels nécessite une mémoire de 128Mvoxels. Une telle capacité implique l'utilisation de boîtiers mémoires dynamiques dont le temps d'accès est actuellement de ~80ns en mode page, ce qui correspond à un débit de 12,5Mvoxels/secondes, soit un débit 268 fois inférieur à celui nécessaire.

On retrouve alors les deux schémas classiques de connexion entre plusieurs bancs mémoires et plusieurs unités de traitements (Figure 3.3). Cependant, dans le cas présent les réseaux de connexions pourront être remplacés par des unités disposant de fonctionnalités plus évoluées, afin de prendre en compte une partie des traitements. D'autre part, les bancs mémoires représentent, ici, uniquement la mémoire voxel. Si les unités de traitement ont besoin de mémoire de travail, elle est considérée comme étant localement attachée aux processeurs, bien que d'autres organisations soient possibles (mémoire partagée...).

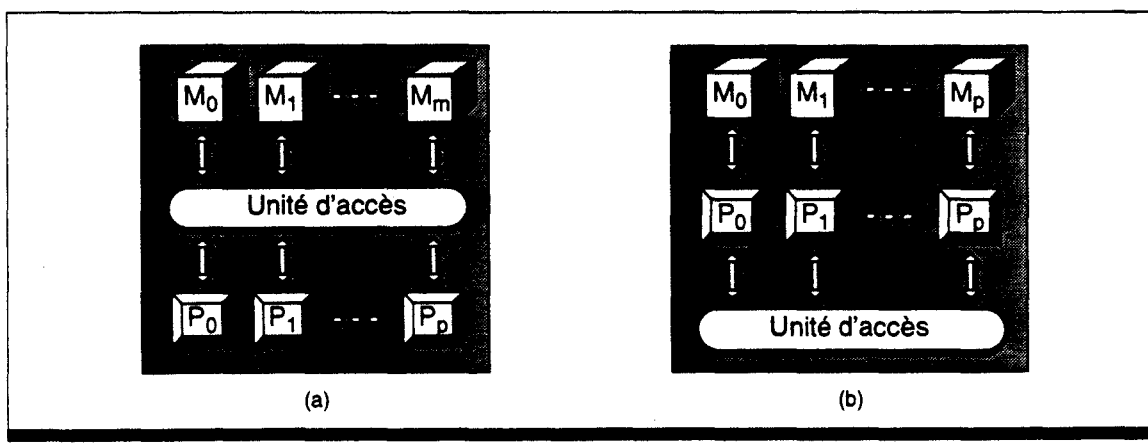


Figure 3.3 ***Schémas de connexion entre unités de calcul et de mémorisation.***

1. Giga (milliard) Instructions Par Seconde.

Machine Voxel à mémoire partagée

Cette organisation (Figure 3.3.a) présente le problème habituel des architectures à mémoire partagée : elle ne permet pas l'utilisation d'un grand nombre de processeurs partageant un grand nombre de bancs mémoires, de par la complexité de l'unité d'accès que cela nécessiterait. De plus, le partage de la mémoire implique des conflits d'accès qui peuvent anéantir le gain espéré par le parallélisme.

On utilisera plutôt ce type d'organisation avec une unité d'accès évoluée, et un nombre faible de processeurs. On peut, par exemple, utiliser l'Unité d'Accès pour effectuer une sélection des voxels répondant à un certain critère, ou pour mémoriser en un cycle un ensemble de voxels calculés par le ou les processeurs.

Les unités de calculs ne sont pas, dans ce type de machine, associées directement aux différents bancs mémoires, mais ont au contraire une vue d'ensemble de tout le cube voxel. Cette approche est de ce fait plus appropriée à :

- un parallélisme par découpage de tâches. Chaque processeur exécute une tâche différente nécessitant l'accès à toute la mémoire,
- un parallélisme objets. Chaque processeur prend en charge un objet différent, ou un ensemble de pixels disposant d'un attribut donné....

Ce type d'organisation est celle utilisée, par exemple, par la machine Cube, que nous présentons plus loin (§ 3.2.1 - p. 62).

Machine Voxel à mémoire distribuée

Cette organisation (Figure 3.3.b) permet, par contre, l'utilisation d'un très grand nombre d'ensemble processeur-mémoires. On peut imaginer, à l'extrême, l'association d'un processeur à un voxel unique.

Dans ce cas, l'Unité d'Accès peut consister en un *simple* réseau d'interconnexion spécialisé, ou intégrer un dispositif de traitement et de sélection d'une donnée parmi toutes celles fournies par les différents processeurs.

Contrairement à l'approche précédente, les processeurs sont ici directement liés à un banc mémoire, et n'ont pas connaissance de l'espace global. Ce type de machine est donc plus approprié à un parallélisme voxel, où chaque processeur prend en charge la partie des traitements concernant les voxels auquel il est associé.

Ce type d'organisation est celle utilisée, par exemple, par la machine Voxel Processor, que nous étudierons par ailleurs (§ 3.2.2 - p. 65).

3.1.5 Choix d'une approche

Nous avons voulu, dans cette étude, pousser la notion de représentation voxel jusqu'à son extrême, afin de dégager l'ensemble des problèmes algorithmiques et architecturaux soulevés par une telle approche.

Nous avons de ce fait choisi comme base de travail une mémoire voxel distribuée, associée à des unités de traitements permettant l'implémentation des algorithmes de visualisation discrète. L'architecture ultime consisterait à utiliser un processeur par voxel. Il semble évident qu'une machine comprenant près d'un milliard d'unités, aussi simple soient-elles, reste un souhait techniquement irréalisable, même à long terme.

Cependant, il existe de nos jours des architectures disposant de 2^{14}, voire 2^{16}, processeurs élémentaires connectés en grille, ou en tore. Nous nous tournerons donc vers des solutions permettant ce taux de parallélisme.

Nous voulons insister sur le fait qu'il ne s'agit pas pour nous de concevoir une architecture respectant les contraintes technologiques actuelles, mais plutôt de présenter les contraintes d'une machine voxel massivement parallèle et d'apporter un certain nombre de solutions. Le

taux d'intégration des circuits numériques évoluant souvent plus vite que les travaux de recherche, nous avons voulu, en quelque sorte, *prendre de l'avance*.

La répartition de la mémoire voxel, i.e. la connectivité entre les unités de traitement peut être réalisée de 3 manières différentes : répartition 3D, répartition 2D et répartition 1D. Dans les paragraphes suivants décrivant ces 3 modes, nous noterons N_V la dimension du cube voxel (qui contient donc N_V^3 voxels).

Dans chacun de ces modes, un noeud est constitué d'un *morceau* de mémoire voxel, d'une unité de traitement et d'un dispositif de connexion entre les noeuds. Ce dernier peut être un simple organe de communication, ou peut prendre en compte une partie des différents traitements. On pourra, par exemple, y intégrer un protocole de routage évolué permettant une communication entre deux processeurs quelconques, ou encore on pourra l'utiliser pour atteindre directement la cellule contenant un voxel donné.

Organisation distribuée 3D

Divers *découpages* 3D de la mémoire voxel sont envisageables. Une répartition donnée est notée $3D_{l,h,p}$, en fonction de la taille d'un sous-ensemble voxel, avec : l sa largeur, h sa hauteur et p sa profondeur (Figure 3.4). Ces 3 valeurs sont choisies comme étant des diviseurs de N_V, afin d'assurer une répartition 3D homogène.

Par exemple, pour la répartition $3D_{1,1,1}$ on associe un voxel à chaque processeur. Inversement la répartition $3D_{N_V,N_V,N_V}$ correspond à une machine mono-processeur sans découpage de la mémoire.

Le nombre de noeuds d'une répartition $3D_{l,h,p}$ est donné par : $N_n = \frac{N_V}{l} \cdot \frac{N_V}{h} \cdot \frac{N_V}{p} = \frac{N_V^3}{l \cdot h \cdot p}$

La réalisation d'un réseau 3D de processeurs est techniquement complexe, l'implantation matérielle de composants sur cartes permettant beaucoup plus facilement une organisation planaire.

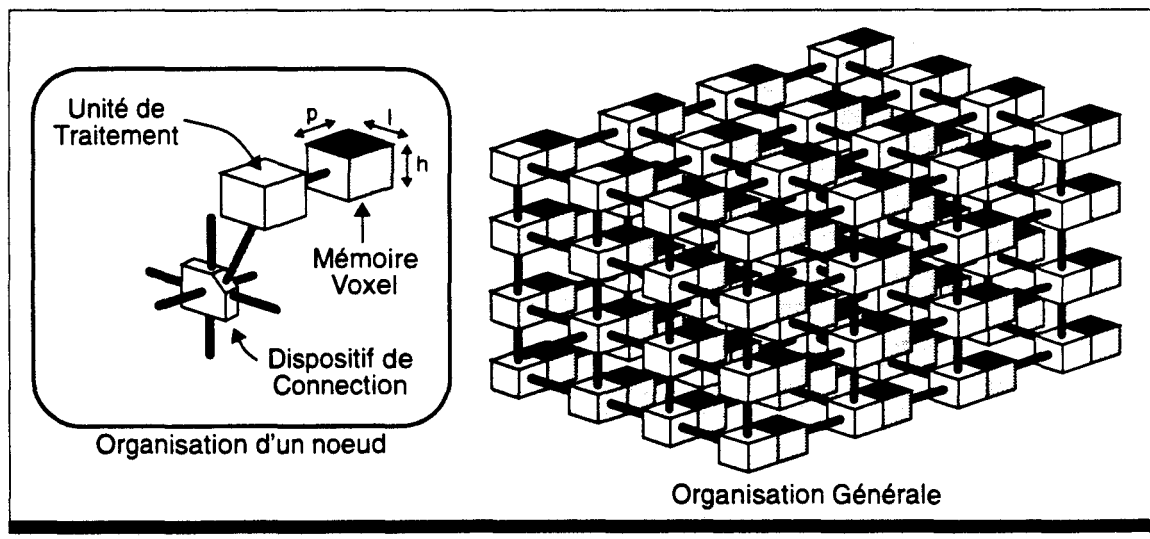


Figure 3.4 ***Organisation d'une Machine Voxel à répartition 3D.***

Organisation distribuée 2D

De par la remarque précédente concernant l'implantation matérielle d'un réseau 3D, il semble naturel d'étudier les possibilités offertes par une organisation planaire. Une répartition 2D donnée sera notée $2D_{l,h}$, et correspond à une répartition $3D_{l,h,N_V}$ (Figure 3.5).

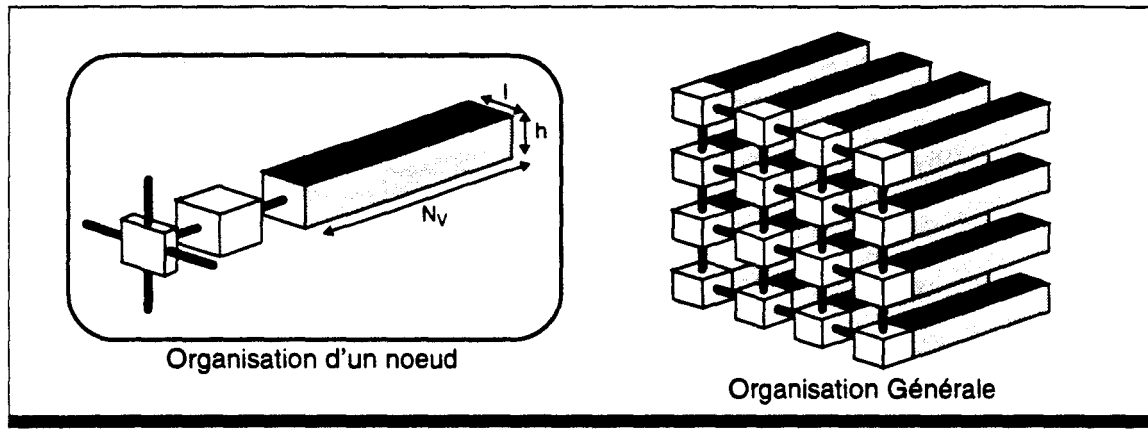


Figure 3.5 ***Organisation d'une Machine Voxel à répartition 2D.***

Le nombre de noeuds d'une répartition $2D_{l,h}$ est donné par : $N_n = \frac{N_V}{l} \cdot \frac{N_V}{h} = \frac{N_V^2}{l \cdot h}$

L'avantage de cette organisation est de permettre une réalisation plus aisée, tout en restant assez proche du modèle voxel *global*.

Cependant, un processeur a, maintenant, en charge un plus grand nombre de voxels, dont la répartition n'est plus homogène. On peut donc *a priori* penser que le traitement séquentiel au sein des processeurs sera plus élevé, ce qui diminuera l'efficacité de la parallélisation.

Organisation distribuée 1D

Afin de simplifier encore plus l'implantation matérielle, il peut être intéressant d'utiliser une répartition monodimensionnelle de la mémoire voxel. Une répartition donnée sera notée $1D_l$ et correspond à une répartition $2D_{l,N_V}$ ou $3D_{l,N_V,N_V}$ (Figure 3.6).

Le nombre de noeuds d'une répartition $1D_l$ est donné par : $N_n = \frac{N_V}{l}$

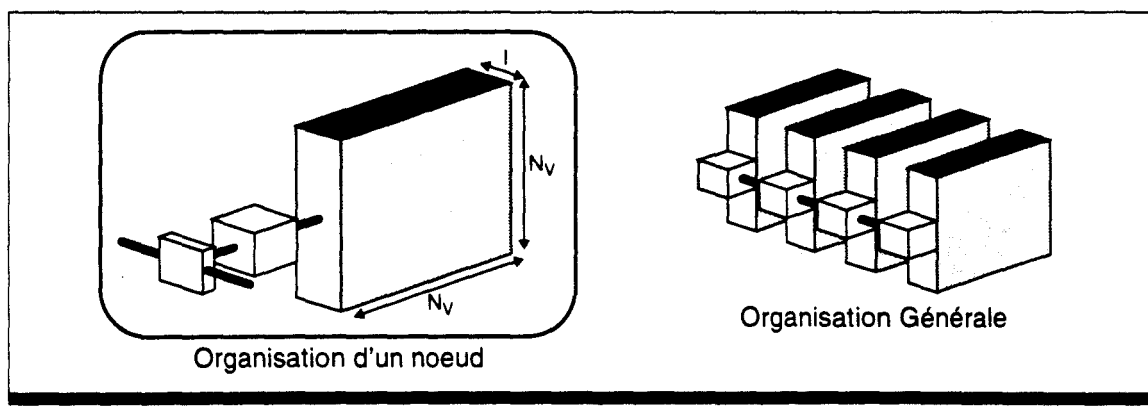


Figure 3.6 ***Organisation d'une Machine Voxel à répartition 1D.***

Cette organisation permet de faciliter grandement la réalisation, mais on peut s'attendre à une diminution importante de l'efficacité, pour les raisons évoquées ci-dessus. De plus, nous ne pouvons plus parler au sens strict d'architecture voxel massivement parallèle, ce qui nous

éloigne du but fixé. Nous l'intégrons cependant dans les études d'implémentation, afin d'établir les possibilités qu'elle offre.

3.2 Exemples de Machine Voxel pour la visualisation médicale

Les premières études dans ce domaine datent du milieu des années 70, en réponse aux besoins de l'imagerie médicale. Des algorithmes furent développés, permettant le traitement et la visualisation directe de données volumiques produites par scanner, résonance magnétique, ultra-sons, ou autre. Les méthodes traditionnelles effectuaient une transformation de ces données en un ensemble de surfaces polygonales sur lesquelles étaient utilisées les algorithmes classiques de rendu. Depuis la fin des années 80, plusieurs machines spécialisées ont été proposées pour résoudre le problème de l'*énormité* du besoin en mémoire, en débit et en traitement.

Les deux stratégies d'accélération du rendu volumique ont été utilisées dans les études proposées :

- Organisation mémoire partagée.
  Il s'agit en fait de permettre l'accès en parallèle à plusieurs voxels.
  On trouve trois machines basées sur ce principe : Cube [Kauf88], INSIGHT [Meag85] et PARCUM II [Jack88].
- Organisation mémoire distribuée.
  Ce schéma utilise une machine multi-processeurs permettant de paralléliser les traitements.
  Cette approche est utilisée par les machines : $3DP^4$ [Ohas85], PICAP II [Lenz86], RT-VRE [Smit92], Voxel Flinger [VoxF90], Voxel Processor [Gold87].

Nous décrirons plus particulièrement les machines Cube et Voxel Processor, qui permettent toutes deux d'obtenir un rendu volumique en temps réel (~30 images/seconde).

3.2.1 La machine Cube

L'organe central de cette machine est l'unité mémoire voxel (CFB - Cubic Frame Buffer), dont l'organisation permet l'accès simultané à plusieurs voxels (Figure 3.7). Trois unités spécialisées utilisent le CFB. Le processeur GP3 (3D Geometry Processor) effectue la conversion d'objets géométriques en voxels [Kauf87a], [Kauf87b]. Le processeur FBP3 (3D Frame Buffer Processor) assure le chargement des échantillons expérimentaux, ainsi que la manipulation du CFB : translation, rotation, composition... de sous-ensembles cubiques de voxels [Kauf91a]. Le processeur VP3 (3D Viewing Processor) exécute la projection 2D et le rendu, en fonction du point de vue choisi; il autorise les projections parallèles ou perspectives de la scène [Kauf91b]. Le processeur VP (Video Processor) est une unité conventionnelle de rafraîchissement écran. Le format d'un voxel est de 8 bits, représentant une valeur de couleur, de transparence ou autre.

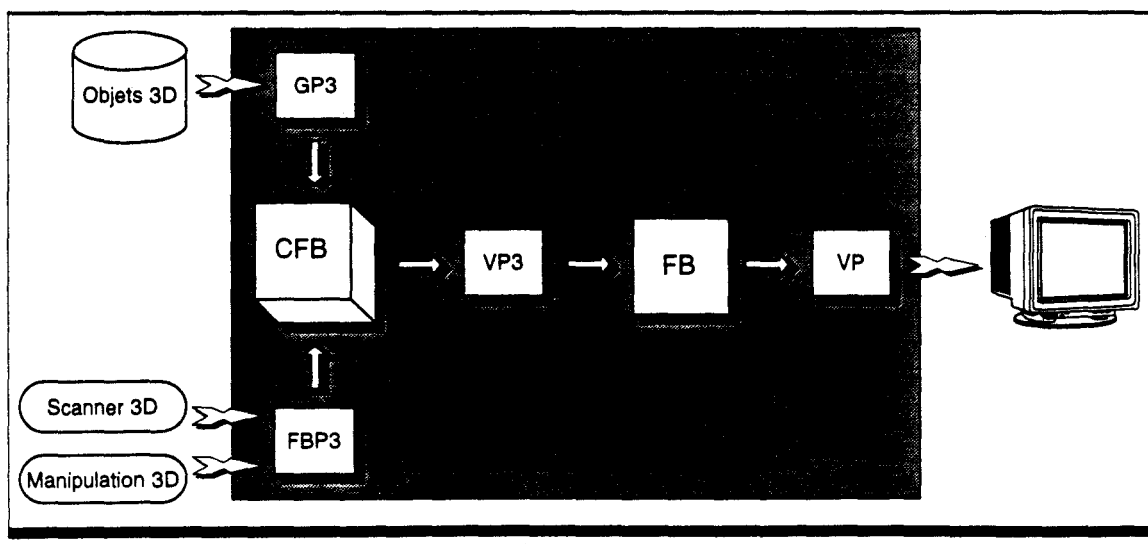


Figure 3.7 ***Schéma général de la machine Cube.*** *D'après [Kauf88].*

Organisation du CFB

Une des originalités de la machine Cube est l'organisation adoptée pour l'unité CFB [Kauf88].

L'affichage d'une scène voxel nécessite la détermination du voxel visible en chacun des pixels. Pour ce faire, un mécanisme d'adressage particulier est utilisé, afin de récupérer, en un cycle horloge, tous les voxels se projetant en un pixel donné (que nous appellerons rayon-voxel), et ceci dans les six directions de projections orthographiques (+x, -x, y, -y, +z, -z) :

Soit un cube de dimension n^3. La mémoire est découpée en n blocs de n^2 voxels, accessibles indépendamment.

A un voxel de coordonnées (x, y, z) est associé un numéro de bloc k, et un emplacement (i, j) dans ce bloc. La transformation d'adresse s'effectue à l'aide des relations suivantes : $k = (x+y+z) \bmod n$ et $i = x, j = y$.

Un rayon-voxel, pour une des projections précitées, est parallèle à un axe. Les coordonnées des voxels qu'il contient varient suivant une seule des directions principales (x, y, ou z), c'est-à-dire qu'une seule de ces trois variables est incrémentée le long du rayon-voxel. De par la transformation d'adresse effectuée, on obtient alors un numéro de bloc différent pour chacun des voxels (Figure 3.8).

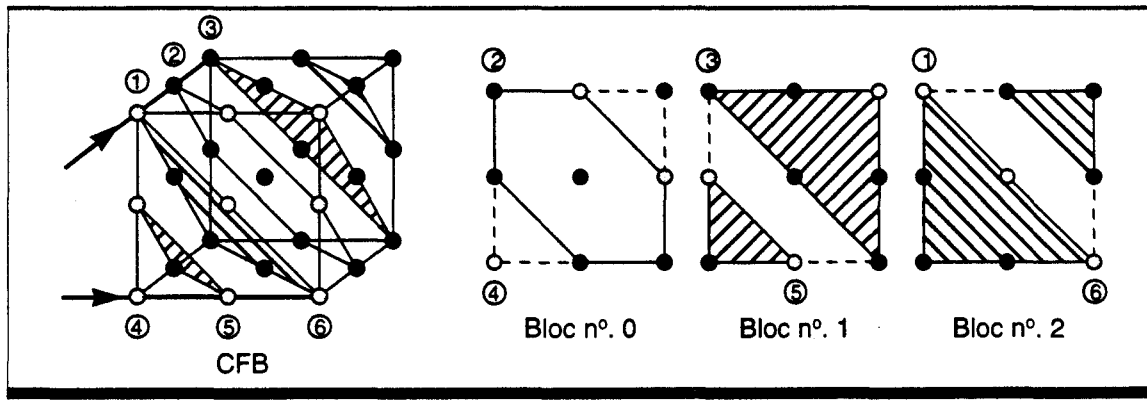


Figure 3.8 ***Principe d'organisation d'un CFB 3x3x3.*** *Distribution des blocs dans le CFB, et distribution des voxels dans les blocs. Egalement représentées, les positions des voxels de deux rayons.*

Lecture du CFB

Chaque bloc mémoire dispose de sa propre unité de contrôle et d'adressage. Les bancs mémoires fournissent donc en un cycle tous les voxels appartenant à un rayon-voxel (Figure 3.9). Une unité spécialisée, le *Voxel Multiple Write Bus* (VMWB), effectue la détermination du voxel visible, en fonction de la direction de visée. Cette unité constitue le premier étage du VP3.

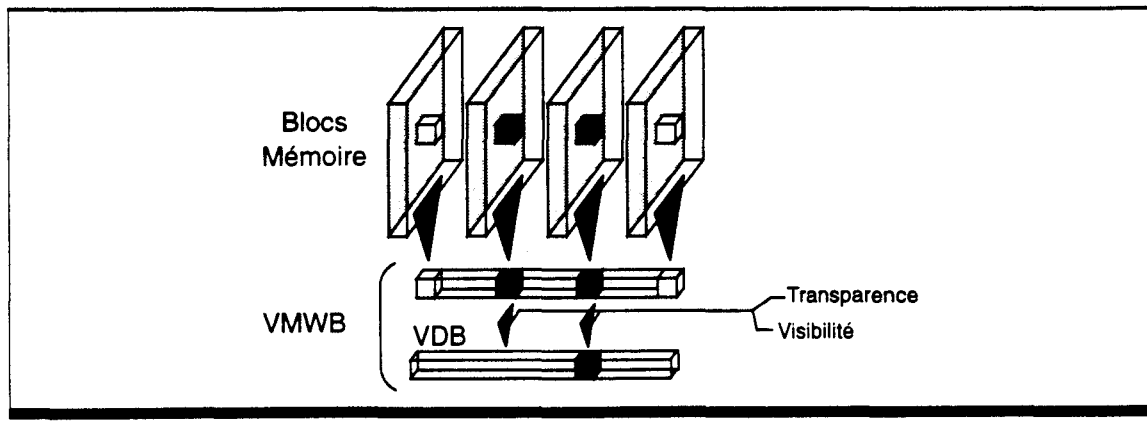


Figure 3.9 ***Principe de fonctionnement du VMWB.***

Un premier étage du VMWB stocke le rayon-voxel. Les voxels dont la valeur est différente de la valeur de transparence entrent ensuite en compétition sur le *Voxel Depth Bus* (VDB), qui ne garde que le voxel le plus proche de l'observateur. Sa valeur et sa profondeur sont envoyées au *Frame Buffer*. Le voxel visible est déterminée en un temps $\log(n)$.

Le VMWB permet également le clipping, en ne transférant au VDB que les voxels compris entre deux profondeurs limites.

Chargement du CFB

Inversement, le registre du VMWB peut être chargé en série, par les deux processeurs amonts (FBP3 et GP3). Le registre est ensuite transféré d'un bloc dans le CFB. Il n'y a donc pas d'accélération dans ce cas, et la durée de chargement est proportionnelle au nombre de voxels à écrire.

Performances

Une machine disposant d'une mémoire voxel d'organisation classique sans système d'adressage particulier, demande n^3 accès à la mémoire par projection. En prenant un temps de cycle de 100ns, cela équivaut à 13,4s pour une configuration de 512^3 voxels.

Dans le cas de Cube, la visualisation d'une image ne nécessite que n^2 accès au CFB, grâce à l'utilisation du VMWB. En tenant compte du temps nécessaire à la détermination du voxel visible au sein d'un rayon, une projection demande 62ms, soit un gain de 216, pour la même configuration. Ce temps est estimé à partir de résultats obtenus sur un prototype de 16^3 voxels.

Le VMWB permet également de réduire le débit du processeur de visualisation. Dans un système classique, il lui faut éliminer les voxels invisibles par une opération du type Z-Buffer. Il calculera donc, globalement, une valeur d'éclairement sur la moitié des voxels. Pour une résolution de $512x512x512x8$ bits, on obtient un débit de 0,5 Gbits par images. Pour la machine Cube, le VP3 ne traite que n^2 voxels, soit un débit de 2 Mbits par images. Le gain est, là aussi, appréciable. Le temps obtenu pour un calcul d'éclairement par la méthode du gradient est de l'ordre de 16ms, temps inférieur à celui nécessaire pour projeter un rayon; il n'influe donc pas sur la vitesse d'affichage.

Utilisation en Lancer de Rayon

Nous avons indiqué que l'architecture de base permet de déterminer le premier voxel visible le long d'un rayon perpendiculaire à un des axes de l'espace. Une extension du système d'accès au CFB permet de projeter un rayon de direction quelconque [Baka92], ou un ensemble de rayons parallèles situés sur un même plan support. Ce mécanisme peut être utilisé pour détecter le voxel intersectant un rayon donné, ceci en temps $\log(n)$.

Dans le cas du Lancer de Rayon, il n'est guère probable de pouvoir profiter de la possibilité de projection commune de rayons parallèles. Un accès au CFB est par conséquent nécessaire pour chaque détermination d'intersection. Le seul parallélisme obtenu se situe dans l'accès aux voxels appartenants à un rayon, ce qui permet uniquement de diminuer le temps de calcul d'une intersection. La complexité d'un Lancer de Rayon est alors : $T_{Lancer\_de\_Rayons} = Nb_{Rayons} \cdot T_{Projections}$. Le temps est donc indépendant du nombre d'objets.

Il faut cependant ajouter le temps nécessaire au chargement de la scène dans le CFB, qui, lui, est linéaire en fonction du nombre d'objets. En utilisant les algorithmes de voxelisation proposés pour la machine Cube, ce temps est même linéaire en fonction du nombre de voxels générés. Cette étape est nécessaire à chaque modification de la scène, mais la mémorisation complète de la scène permet de générer des vues différentes sans rechargement.

3.2.2 La machine Voxel Processor

Contrairement à la proposition de Cube, où l'accès mémoire est accéléré par une organisation particulière, la machine Voxel Processor utilise un parallélisme d'accès. La mémoire voxel est découpée en sous-cubes de dimensions **64x64x64** gérés indépendamment par des unités spécialisées (PEi). Une mémoire totale de 256^3 voxels nécessite 64 PE's.

Il est possible de ne mémoriser qu'un nombre réduit de plans en profondeur, un dispositif d'interpolation calculant les voxels situés sur les plans manquants. Ce mécanisme permet de décharger le hôte, qui en est habituellement chargé.

Le format d'un voxel est de 16 bits, constitués de 8 à 16 bits de densité (ou couleur), les bits restant servant d'identification (numéro d'objet par exemple).

Plusieurs unités spécialisées sont connectées à la mémoire voxel (Figure 3.10). L'unité HI (Host Interface) assure l'envoi des voxels vers la mémoire. L'unité CTU (Coordinate Transform Unit) effectue la transformation d'adresse entre l'espace objet, espace de représentation des voxels, et l'espace image, espace de représentation des pixels. L'unité DOTS (Dynamic Object Thresholding and Slicing) est chargée de l'interpolation des voxels sur les profondeurs non mémorisées, et permet la sélection à l'aide des identificateurs. L'unité SVO (Shading and Video Output) effectue le rendu volumique de la scène.

Projection de l'espace objet

La méthode utilisée pour générer l'image visible est la technique appelée *Back-to-Front Projection* [Frie85] : il existe un ordre de lecture de la mémoire 3D, dépendant de la direction de projection, assurant que si deux voxels se projettent en un même pixel, le dernier lu est plus proche de l'observateur; c'est par conséquent le voxel visible.

Chaque PE effectue la projection du sous-cube dont il a la charge, et génère une *mini*-image, de taille maxi $64\sqrt{3}x64\sqrt{3}$, en mémorisant pour chaque pixel la valeur de densité et la profondeur du voxel visible. Une mémoire d'image de taille $128x128$ est utilisée par commodité. Ces *mini*-images doivent ensuite être composées pour définir l'image finale.

La composition nécessite $64x128^2$ accès aux mémoires de *mini*-images. La machine Voxel Processor utilise un arbre pour pipeliner la composition, ce qui permet de diminuer le temps de cycle de projection (Figure 3.10). Un temps de latence correspondant à 3 images est, en contrepartie, ajouté au système.

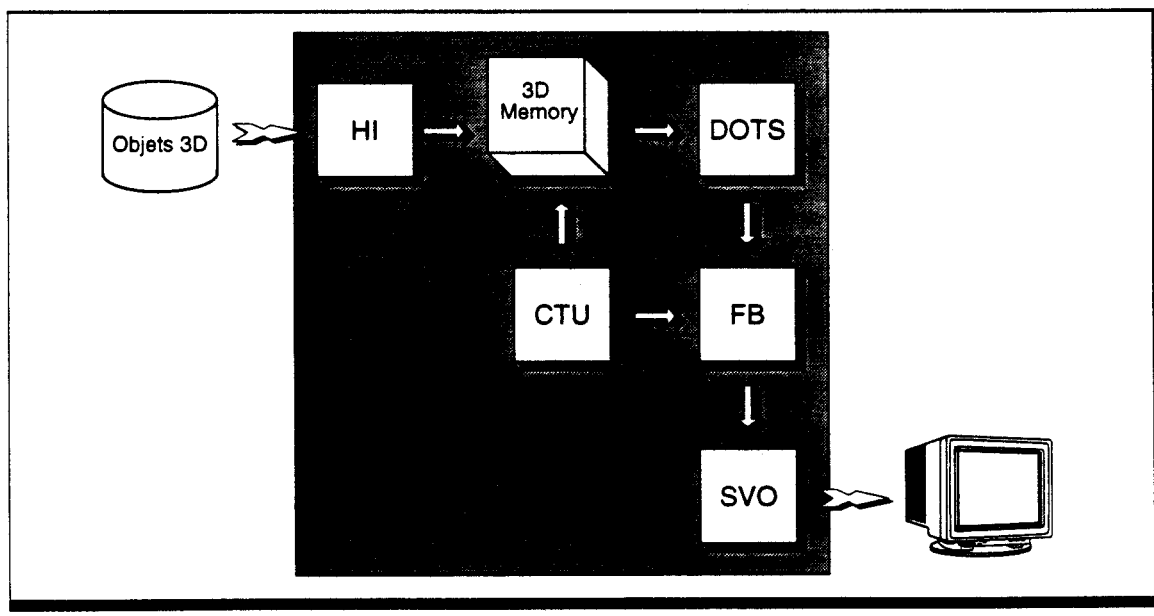


Figure 3.10 ***Schéma de principe de la machine Voxel Processor.*** *D'après [Gold89]. Les unités CTU, DOTS et FB sont réparties au sein des PE's.*

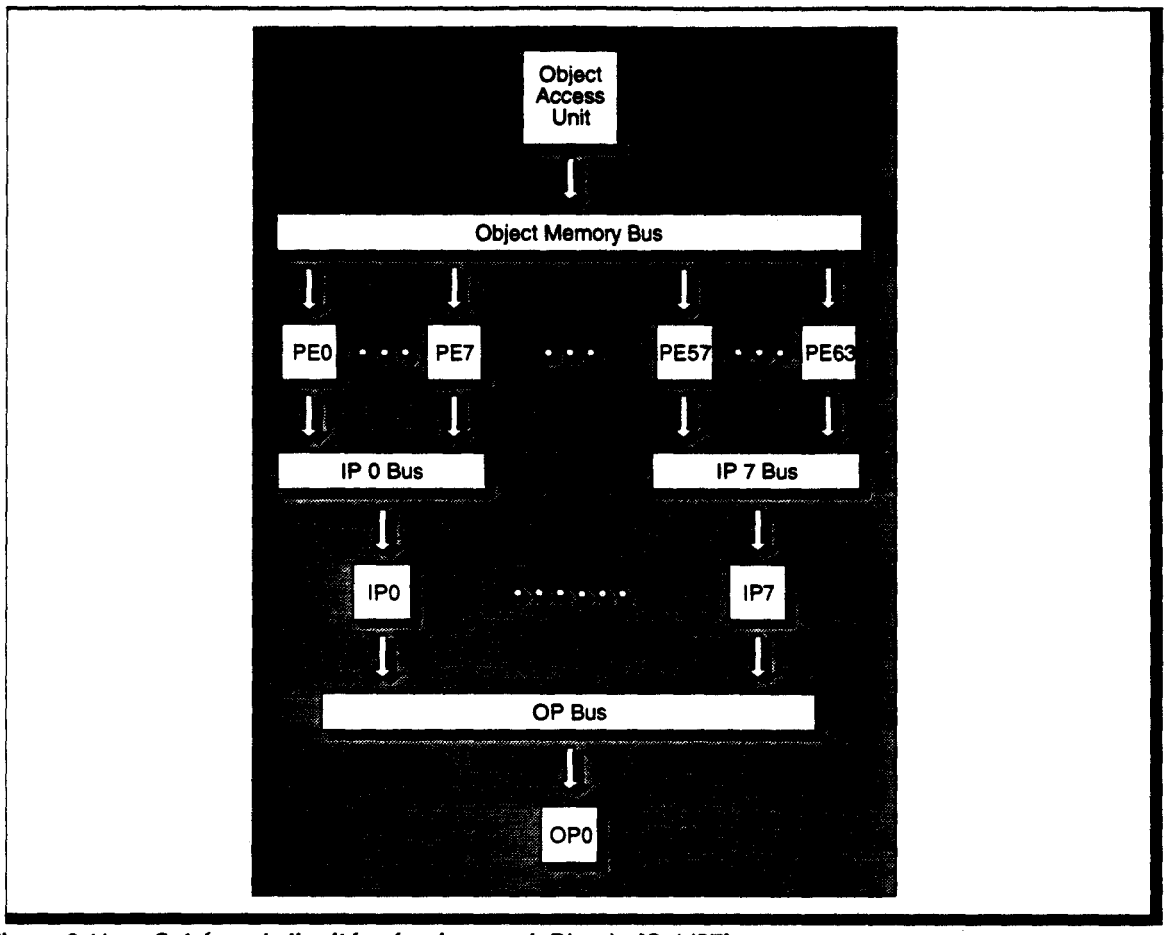


Figure 3.11 ***Schéma de l'unité mémoire voxel.*** *D'après [Gold87].*

Les unités IP (Intermediate Processor) génèrent 8 images 256x256, qui sont combinées par l'unité OP (Output Processor) pour définir l'image 512x512 finale. Les différentes compositions utilisent le principe de la projection BTF; les PEs sont de ce fait regroupés suivant une décomposition en octree de la mémoire voxel (Figure 3.12).

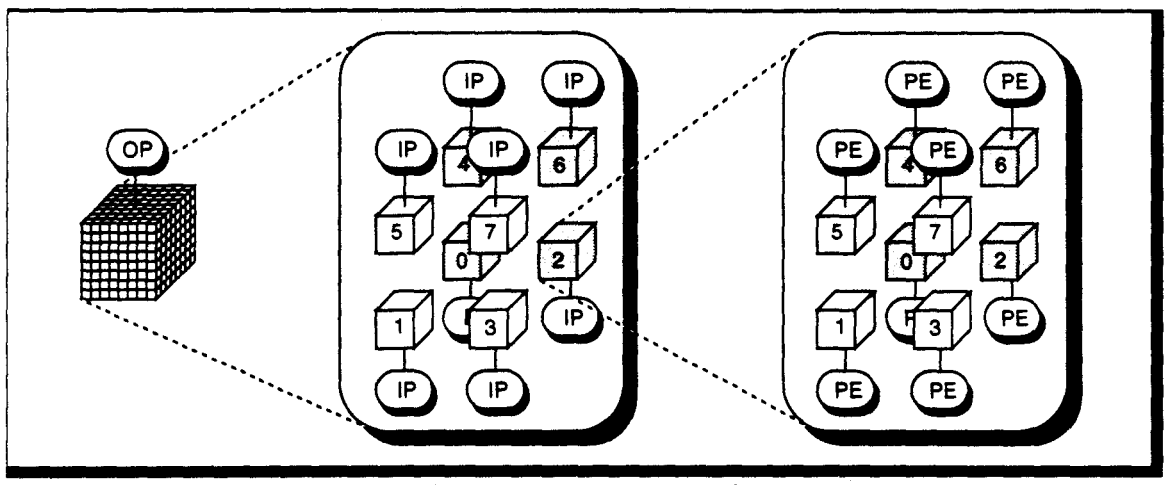


Figure 3.12 ***Décomposition hiérarchique de la gestion de la mémoire voxel.***

Conversion 3D vers 2D

La projection d'un sous-cube nécessite, dans une première phase, la détermination de la séquence de lecture des voxels, pour respecter les règles de la projection BTF. Cette séquence est déterminée par le processeur hôte, en fonction du point de vue, et est chargée dans la table de contrôle SCT-1 (Figure 3.13).

D'autre part, il peut être nécessaire d'effectuer une rotation du sous-cube, pour le placer dans le repère de l'observateur. Les coordonnées (X, Y, Z), dans l'espace image, des sommets du cube sont mémorisées dans la table SCT-2. A partir de la position des sommets, des opérations simples de décalages et d'additions permettent le calcul de la position de chaque voxel.

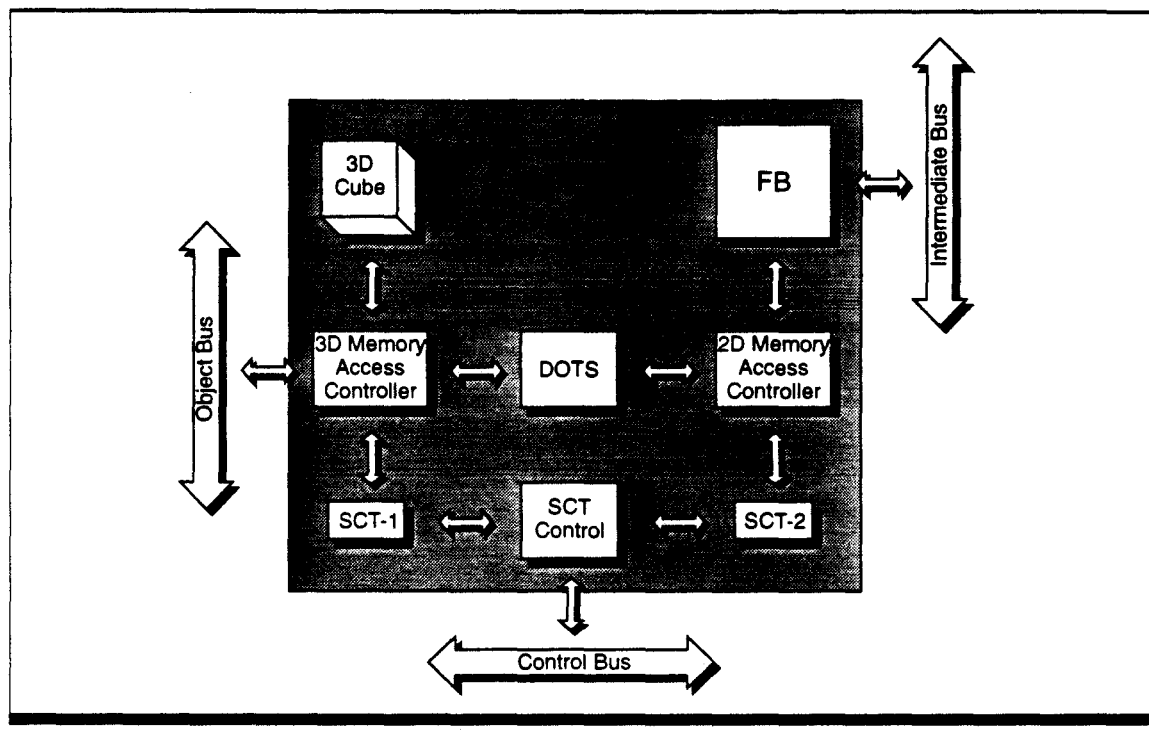


Figure 3.13 ***Organisation d'un processeur élémentaire.*** *D'après [Gold87].*

Dans le cas où les PE's mémorisent des portions d'une image globale, les tables SCT ont le même contenu sur tous les PE's, il serait donc possible de centraliser le contrôle de l'adressage. Cependant, la répartition des tables permet de stocker des scènes différentes sur les PE's (pour mettre en place un *multi-cube voxel*), ce qui implique des contenus différents.

Performances

En supposant un temps de cycle de 100ns, un PE génère la projection d'un sous-cube en $64^3 \times 100ns$, soit $26.2ms$. Les processeurs intermédiaires effectuent la composition de 8 mini-images en $8 \times (64\sqrt{3})^2 \times 100ns$, soit $9.8ms$. La génération de l'image finale nécessite, quand à elle, $8 \times (128\sqrt{3})^2 \times 100ns$, soit $39.3ms$.

Cette dernière opération est la plus longue du processus, c'est donc elle qui limite la fréquence d'affichage de la machine. On aboutit, ainsi, à la possibilité de générer ~25 images/s.

En cas de changement de point de vue, le seul effet est une latence de 3 images, avant que la nouvelle projection aboutisse au post-processeur de rendu.

Aucune estimation du temps nécessaire au rendu par *gradient shading* n'est fournit.

Un prototype de la machine, appelé *Voxel Processor Prototype* (VPP), a été réalisé à l'aide de composants MSI TTL. Il implémente un unique module PE, et a été inclus dans un environnement de travail complet [Gold85].

Utilisation en Lancer de Rayon

Dans le cadre du Lancer de Rayon., la machine Voxel Processor n'est utile qu'à la détermination du voxel intersectant un rayon, à l'aide d'une projection parallèle à la direction de ce rayon.

Or, une projection complète demande $\sim 164ms$, ce qui rend inadéquat cette machine pour la méthode du Lancer de Rayon.

3.2.3 En conclusion

Les deux machines que nous venons de présenter ont été pensées dans le cadre de la visualisation de données tri-dimensionnelles médicales, et présentent donc des caractéristiques remarquables dans ce domaine d'application.

Cependant elles ne semblent pas adaptées, ou tout du moins elles sont mal adaptées, aux besoins de la synthèse d'images, et plus particulièrement du Lancer de Rayon.

En effet, elles implémentent essentiellement des mécanismes de projection. Cube dispose de mécanisme de composition des projections, mais aucune n'offre de réelles possibilités de calcul au niveau voxel. La machine Voxel Processor dispose d'unités opératives parallèles, mais elles sont essentiellement câblées, et ne permettent pas la manipulation des voxels.

Il nous faut donc étudier une machine plus spécifique aux problèmes de la synthèse d'images afin de répondre à nos besoins.

3.3 Une Machine semi-voxel : Voxar

Le projet VOXAR (VOXel ARchitecture), de l'équipe de R.Caubet, consistait originellement à définir une Machine Voxel de type $3D$ telle que nous l'avons précédemment définie. Cependant, la mise en oeuvre d'une architecture composée de 1024^3 processeurs était très peu réaliste à l'époque, et la solution retenue a été l'utilisation d'un nombre restreint de noeuds (machine Volvox constituée d'un hypertore de 4x3x3 Transputers T800 d'Inmos) auxquels sont associés des sous-cubes voxels[1] de taille importante [Caub88].

Répartition des Métavoxels

Sur une machine moyennement parallèle se pose le problème de la répartition de charge. Le parallélisme est ici du type *flot de rayons* avec subdivision régulière de l'espace (§ 1.3 - p. 33), l'équilibrage ne peut alors se faire que par répartition judicieuse des sous-cubes voxels sur chacun des noeuds.

Dans le cas de la machine Voxar, il a été choisi d'utiliser un nombre important de sous-cubes, distribués de manière cyclique sur l'hypertore. Un processeur gère donc plusieurs métavoxels appartenant à des positions différentes de l'espace. Cette répartition cyclique permet à deux sous-cubes voisins d'être situés sur deux processeurs voisins. Les échanges de rayons sont donc réduits à des communications locales [Jess89].

La taille d'un sous-cube est également une donnée importante dans l'équilibrage de la charge. Plus elle est importante, moins un processeur possède de zones à traiter, et donc plus il risque de ne posséder que des zones vides ou, au contraire, comportant beaucoup d'objets. Inversement, plus les sous-cubes sont petits, plus le temps de suivi des rayons est important, comparé au temps dédié au calcul d'éclairement. Pour accélérer la traversée des zones vides de l'espace, un algorithme particulier de suivi incrémental des rayons a été défini. Il est basé sur le tracé de droite d'Amanatides et Woo, et permet le suivi d'un rayon au niveau voxel ou au niveau métavoxel. Un sous-cube vide est, de ce fait, traversé en une seule étape [Pito89]. Avec l'utilisation de cet algorithme, la taille optimale d'un sous-cube est comprise entre 8 et 32 [Mois91].

Mémorisation des Métavoxels

Seuls les métavoxels non vides sont réellement mémorisés au sein des processeurs, à l'aide d'une représentation de type *matrice creuse* [Maur91].

Le premier avantage est de limiter la mémoire nécessaire au stockage du cube voxel complet. La contre-partie à ceci est que pour une taille de mémoire donnée un cube de plus grande dimension est représentable.

Mais il existe un autre avantage plus intéressant: il est possible avec ce type de représentation de mémoriser un cube voxel de très grande dimension, mais également très *creux*, ce qui permet de stocker, par exemple, plusieurs objets fortement distants. Il existe cependant certainement une limitation de la taille de l'espace représentable, ce qui ne doit pas permettre d'utiliser des scènes à profondeur de champ importante.

Discrètisation en *atomes*

Nous avons vu, et nous verrons dans le chapitre suivant, quelques inconvénients à l'utilisation de l'entité *voxel*. Un autre problème qui est associé à l'espace discret, qu'il soit $2D$ et $3D$, est que certaines opérations de transformation géométrique (telle la rotation, en particulier) ne permettent pas de conserver la connectivité entre les voxels. L'application répétée de ces opérations peut même modifier complètement la forme initiale de l'objet. Il n'est donc pas possible d'utiliser directement le voxel en animation [Jess89].

1. Appelés Métavoxels dans le projet VOXAR.

Dans le cas de Voxar, la discrètisation des objets n'est pas effectuée au sein de la machine parallèle. Pour permettre une animation rapide, le voxel a été délaissé au profit d'une autre représentation discrète des objets : les mailles d'atomes [Mois91].

L'atome est une notion équivalente à celle de voxel (entité de matière), disposant d'un centre et d'un rayon. Cependant à la différence du voxel, il n'est pas attaché à la subdivision de l'espace $3D$: les atomes peuvent se recouvrir, la position du centre d'un atome n'est pas forcément de coordonnées entières; ils sont donc uniquement associés à l'objet et, de ce fait, un voxel peut contenir plusieurs atomes du même objet. L'algorithme du Lancer de Rayon doit prendre en compte cette multiplicité, afin de déterminer l'atome où se produit l'intersection.

Ces atomes sont liés par groupe de 3 au sein de *mailles*. Le découpage en mailles d'un objet peut être considéré comme équivalent à une *micro*-facettisation de cet objet. De fait, l'algorithme du Lancer de Rayon prend en compte cette représentation, pour effectuer un calcul analytique du point exact d'intersection, les attributs géométriques (normale...) étant recalculés par interpolation barycentrique à partir des données stockées dans les atomes composant la maille.

Utilisation des mailles d'atomes

Il existe trois états de représentation de la scène en mémoire [Maur91] :

- Représentation naturelle

  Lors du chargement d'un objet, les mailles sont stockées dans le processeur prenant en charge le voxel où est positionné le *premier* atome de la maille. Une maille est définie par la donnée des trois atomes qu'elle contient.

  On dispose ainsi d'une représentation géométrique, sur laquelle toutes les opérations de transformations peuvent être appliquées.

- Représentation naturelle sans respect du placement

  Après transformation géométrique, l'atome origine d'une maille peut avoir changé de position, il doit donc être *replacé* sur le processeur adéquat, pour revenir dans la représentation naturelle.

- Représentation voxelisée

  Avant de déclencher le Lancer de Rayon, il est nécessaire de plonger les mailles dans les voxels, c'est-à-dire de disposer les atomes dans les voxels où ils apparaissent. Pour cela, une maille est recopiée dans tous les voxels qui la concerne. Un voxel peut donc contenir plusieurs atomes de plusieurs objets.

Cette double représentation permet d'éviter d'avoir à recharger complètement la scène à chaque étape d'animation, et permet d'utiliser la machine parallèle pour effectuer les transformations géométriques. Elle implique cependant un quasi doublement de la mémoire nécessaire au stockage d'une scène.

Performances

L'implémentation sur le réseau de Transputers utilise un langage acteur développé spécifiquement pour ce projet [Pito90].

Les mesures de performances montrent une accélération quasi linéaire en fonction du nombre de processeurs, mais avec une efficacité moyenne proche de 50% [Mois91]. Il s'agit d'une valeur *normale* pour un parallélisme à flot de rayons (cf § 1.3 - p. 33).

Conclusion

Nous avons qualifié la machine Voxar d'approche semi-voxel. En effet, comme nous avons pu le voir, l'utilisation de mailles d'atomes nécessite un calcul analytique classique d'intersection, la subdivision voxel n'étant utilisée que pour le suivi des rayons. Il ne s'agit plus à proprement parler d'une Machine Voxel.

Cependant, les choix effectués permettent la prise en compte naturelle de l'animation directement sur la machine parallèle, ce qui est un avantage indéniable par rapport aux autres implémentations parallèles du Lancer de Rayon, où la scène est rechargée à chaque étape de l'animation.

3.4 Conclusion

Notre but premier est l'étude d'une Machine Voxel la plus pure *possible, c'est-à-dire utilisant le minimum d'informations géométriques.*

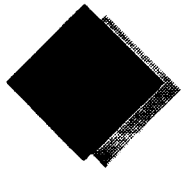
Ce choix pose certes quelques problèmes si l'on désire obtenir une image dont la qualité soit équivalente à celle d'un Lancer de Rayon classique. Les diverses solutions permettant de minimiser ces problèmes sortent du cadre de cette thèse. Nous tenterons cependant d'en apporter quelques-unes.

En tout état de cause, nous nous restreignons à la définition de base que nous avons donné pour une Machine Voxel, sachant que toutes améliorations pourront être apportées par la suite.

Notons que les domaines d'applications pour lesquels une Machine Voxel parallèle sera la plus efficace sont ceux utilisant un milieu participatif.

Parmi les implémentations matérielles de visualisation volumique, seules les machines spécifiques à la visualisation médicale utilisent une notion réelle de voxel. Cependant, elles sont trop spécialisées, et sont peu efficaces pour la mise en oeuvre d'un Lancer de Rayon Discret.

CHAPITRE



Processus de Voxelisation

Dans ce chapitre, nous allons aborder l'étude du premier processus à intégrer dans une Machine Voxel. Cette première étape consiste à transformer les objets géométriques de la base de données en un ensemble de voxels. Nous appellerons cette phase *voxelisation*, ou *discrètisation*. Nous donnons en Annexe D un certain nombre de définitions de topologie en géométrie discrète, telles que la *connexité*, la notion de *trou*, ou d'équation *diophantienne*.

Les premières études concernant la *voxelisation* d'objets ont été menées, à partir de 1986, par l'équipe de A.Kaufman dans le cadre du projet Cube. Il s'agit principalement de méthodes algorithmiques, parfois dérivées de méthodes de discrètisation 2D. Elles permettent d'obtenir des objets voxels satisfaisants, mais sans proposer de définition formelle de l'objet discret, ce qui ne permet pas toujours, par exemple, de *prouver* le type de connexité obtenu. Ces algorithmes ont en commun un *balayage aléatoire* de l'espace 3D. Nous entendons par là qu'il n'existe pas, lors de la génération des voxels, de direction de balayage utilisée de façon systématique.

B.Vidal a proposé une méthode de génération de polygones 3D utilisant un *balayage unique* (la génération des voxels est effectuée par balayage du plan de projection XY).

L'équipe de J.Françon a débuté en 1990 des études sur la géométrie discrète, en particulier sur le problème difficile de la rotation, ce qui leur a permis de définir une méthode formelle de description de certains objets discrets : la droite 2D, le plan 3D et la sphère 3D. Cette méthode, appelée *balayage diophantien*, a été reprise au LIFL, afin de définir d'autres objets.

Notre propos n'est pas ici de détailler les différents processus permettant la voxelisation d'objets. Après la définition des deux critères que nous proposons pour classer ces méthodes, nous présenterons les différentes techniques existantes. Nous ferons également l'étude de leur implémentation sur les différentes organisations massivement parallèles, afin de déterminer les potentialités de chacune des méthodes de voxelisation. Nous dégagerons ainsi le schéma global de fonctionnement de la machine RC^2 durant la phase de conversion. Enfin nous proposerons l'implémentation de l'une d'entre elles afin d'affiner la description architecturale de la machine.

4.1 Critères de classification des algorithmes de voxelisation

Nous classerons les différents algorithmes à l'aide de deux critères :

- le premier prend en compte le type de balayage effectué, c'est-à-dire l'ordre spatial de parcours des voxels au sein du cube mémoire.
- le second définit la méthode générique de voxelisation utilisée.

On appellera *plan principal*, un plan formé par 2 des trois vecteurs directeurs du repère de l'espace. Il existe ainsi 3 plans principaux, que nous nommerons : plan XY, plan XZ, et plan YZ.

On appellera *balayage d'un plan*, l'utilisation d'un plan principal dans lequel on fait varier les 2 coordonnées associées au plan. Par exemple, le balayage du plan XY correspond à une variation de coordonnées en Y puis en X, avec pour chaque position (x, y), le calcul du ou des voxels à mémoriser.

4.1.1 Types de balayage

On peut énumérer 4 possibilités de balayage du cube voxel, que nous nommons :

1. *Balayage plan unique*
   Ces méthodes effectuent la voxelisation par balayage du plan *XY*, quel que soit l'objet en cours de voxelisation. Ces méthodes sont donc bâties autour de l'algorithme suivant :

```
Pourchaque (Y)
|  Pourchaque (X)
|  |  Calcul de Z
|  |  Remplir_Voxel(X,Y,Z)
|  FPour
FPour
```

2. *Balayage diophantien*
   Cette technique utilise la définition d'objets par équation diophantienne. Elle consiste à balayer complètement l'ensemble du cube mémoire, et à calculer pour chaque voxel la valeur de l'équation qui permet de déterminer l'appartenance ou la non-appartenance du voxel à l'objet considéré. On effectue, comme dans le cas précédent, un balayage du plan *XY*, mais on exécute également un balayage en *Z*, et non plus un calcul direct des voxels à remplir, ce qui nous donne l'algorithme général :

```
Pourchaque (Y)
|  Pourchaque (X)
|  |  Pourchaque (Z)
|  |  |  Si Voxel(X,Y,Z) ∈ Objet
|  |  |  |  Remplir_Voxel(X,Y,Z)
|  |  |  FSi
|  |  FPour
|  FPour
FPour
```

3. *Balayage plan quelconque*
   Pour ces méthodes, contrairement au cas du plan unique, le plan principal support du balayage est choisi en fonction de l'orientation de l'objet à convertir. Le plan de balayage est donc différent d'un objet à un autre. Le *balayage plan quelconque* peut être défini par l'algorithme suivant (U,V représentent les coordonnées du plan de balayage choisi, W est la troisième coordonnée de l'espace 3D) :

```
Choix Balayage -> U, V
Pourchaque (V)
|  Pourchaque (U)
|  |  Calcul de W
|  |  (X,Y,Z)<-f(U,V,W)
|  |  Remplir_Voxel(X,Y,Z)
|  FPour
FPour
```

4. *Balayage aléatoire*
   Il n'existe plus ici de plan de balayage. La génération des voxels suit un ordre spatial aléatoire. L'algorithme général peut être défini sous la forme suivante :

```
Tantque Non Fin Trace
|  Voxel_Suivant = f(Voxel_Courant)
|  Remplir_Voxel()
FTantque
```

4.1.2 Méthodes de voxelisation

On peut définir 4 types génériques de méthodes de voxelisation, que nous nommerons :

1. *Extension 3D*
   Ces méthodes ont à la base un algorithme de discrètisation 2D classique, qui a été étendu à l'espace voxel, afin de prendre en compte une troisième dimension (Figure 4.1.a).
2. *Construction*
   Ces algorithmes ont été développés spécifiquement au LIFL pour l'utilisation d'une machine voxel à mémoire répartie 2D. Il s'agit de calculer un ensemble de valeurs qui permettront *ensuite* la construction réelle de l'objet à voxeliser. Le terme *construction* est repris en analogie de la CSG[1]. En effet, typiquement, on détermine pour un objet une valeur de profondeur avant et arrière, l'objet étant construit par remplissage des voxels compris entre ces deux valeurs (Figure 4.1.b).
3. *Extrusion*
   Le terme anglais utilisé pour spécifier ces méthodes est *weaving*, que nous traduisons librement par *extrusion*. Il s'agit d'utiliser une forme de base, qui est ensuite *recopiée*, avec déformation éventuelle, le long d'une génératrice, éventuellement courbe. Il s'agit donc, ici, d'une généralisation de l'extrusion classique (Figure 4.1.c).
4. *Equation diophantienne*
   Cette méthode uniquement utilisée en cas de balayage diophantien, consiste à construire un objet à partir de l'équation diophantienne de sa surface (Figure 4.1.d).

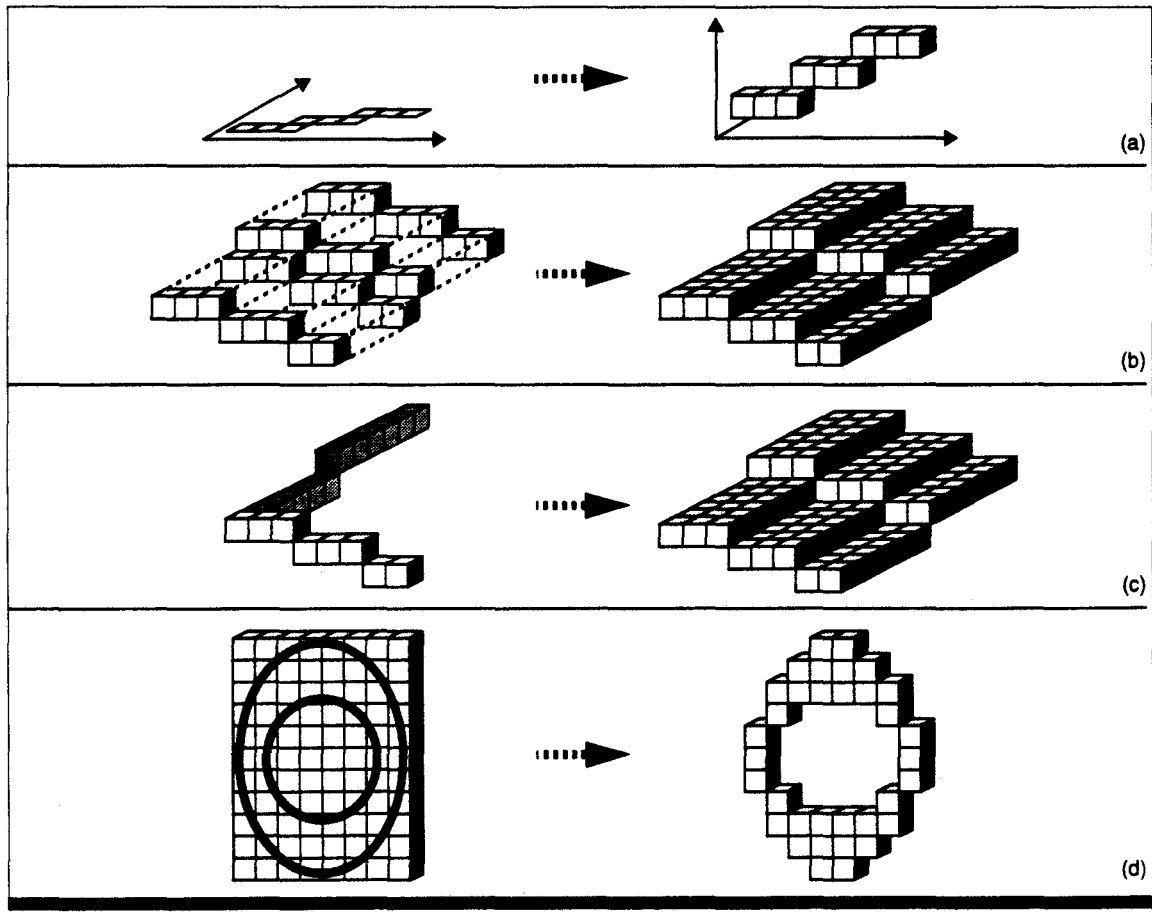


Figure 4.1 ***Les différentes méthodes de voxelisation.*** *(a) - Extension 3D. (b) - Construction. (c) - Extrusion. (d) - Equation diophantienne (vue d'une seule tranche, z=constante).*

1. Abréviation de Constructive Solid Geometry.

4.2 Organisation architecturale générale

Le but du processus de voxelisation est, bien entendu, le remplissage de la mémoire voxel à l'aide des informations pertinentes pour les processus de rendu et de visualisation (couleur, normale...).

Pour l'étude des algorithmes de conversion nous prendrons comme exemple une scène composée de facettes 3D. Nous supposerons un volume englobant de $16 \times 16 \times 16$ voxels pour chaque facette, ce qui donne une taille moyenne de 500 voxels pour celles-ci. Nous utiliserons également une fréquence horloge de 20 MHz pour le réseau, fréquence faible mais correspondant à des temps d'accès mémoire raisonnables.

Effectuons un rapide calcul des puissances impliquées par la phase de voxelisation. Supposons une scène composée de 10.000 facettes. Avec les données précédentes, nous obtenons un total de 5 Mvoxels, ce qui représente $1/25^e$ d'un cube de dimension 512. Supposons également une fréquence d'animation de 30 Hz.

- L'accès mémoire nécessite un chargement de 150 Mvoxels/s. Ce débit est incompatible avec les performances des boîtiers mémoires actuels. Un faible parallélisme d'accès au cube mémoire permettrait cependant de résoudre ce problème (organisation mémoire à 16 accès parallèles). Cependant, l'ordre de génération des voxels pouvant être quelconque, il n'est pas évident de déterminer le meilleur schéma d'organisation de ces 16 bancs mémoires.
- Les algorithmes de voxelisation de facettes triangulaires, par balayage aléatoire ou par construction, nécessitent le calcul d'une vingtaine d'opérations par voxel, soit, avec les données précédentes, l'exécution de 3 Gopérations/s. Cela représente à peu près 10 fois la puissance des processeurs généraux les plus performants d'aujourd'hui. Un faible parallélisme pourraient également être envisageable.
- Dans le cas des algorithmes par balayage diophantien, des calculs sont effectués pour tous les voxels appartenant au cube englobant des objets. Prenons un nombre moyen de 10 opérations par voxel (vide ou plein). Avec les données supposées, nous obtenons alors 9 Gopérations/s. Cette puissance pourrait être atteinte par un parallélisme moyen (~30 processeurs).

En tout état de cause, il est nécessaire de mettre en oeuvre une machine de calcul parallèle pour effectuer la voxelisation, le point *faible* de ce processus provenant de la partie traitement plutôt que de l'accès mémoire. La Machine Voxel disposant d'une unité de traitement associée à chaque sous-cube voxel, il semble intéressant de les utiliser pour effectuer la conversion.

L'organisation ainsi obtenue (discrètisation sur un réseau de processeurs) ressemble fortement aux organisations utilisées en rendu classique, et en particulier aux architectures à parallélisme pixel. Nous pouvons profiter des travaux menés sur ce type de machine afin d'en affiner le fonctionnement.

4.2.1 Connexion hôte - réseau

Les opérations de transformation géométriques, effectuées sur la machine hôte, nécessitent un parallélisme objet (répartition des objets sur les différents processeurs). Il existe alors un *conflit de parallélisme* entre la machine hôte, utilisant ce parallélisme objet, et le réseau de conversion, utilisant un parallélisme voxel. La solution communément adoptée est l'utilisation d'un bus, avec un point d'entrée unique sur le réseau (Figure 4.2).

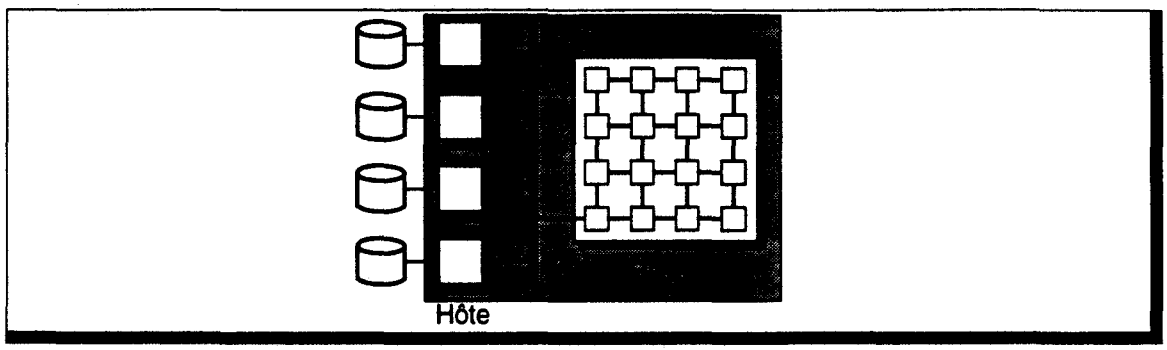


Figure 4.2 ***Organisation classique hôte/réseau de conversion.***

Les performances de conversion sont déterminées par le débit autorisé en entrée du réseau.

Caractérisons ce débit :

- Soit c_T, le nombre de cycles nécessaires au transfert des données permettant la conversion,
- Soit c_C, le nombre de cycles nécessaires à un processeur du réseau pour convertir les voxels dont il a la charge. Nous définirons plus précisément cette valeur pour chaque type d'algorithme de voxelisation.

Si la transmission de données et la conversion sont effectuées en séquence (Figure 4.3.a), le débit en entrée du réseau est (T_c est la période horloge) :

$$D_s = \frac{1}{(c_T + c_C) \cdot T_c}$$

Si la transmission et la conversion sont effectuées en parallèle (Figure 4.3.b), on obtient alors un débit de :

$$D_p = \frac{1}{max(c_T, c_C) \cdot T_c}$$

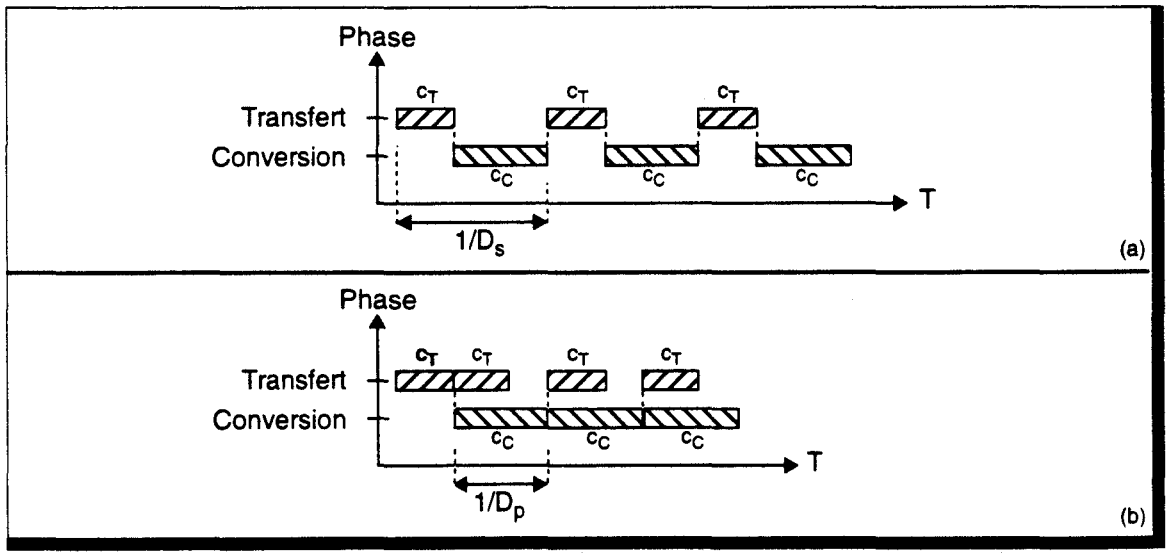


Figure 4.3 ***Déroulement des phases Transfert et Conversion.*** *(a) - En séquence. (b) - En parallèle.*

Nous choisirons la deuxième approche, qui permet un débit plus élevé. Notons également qu'il est inutile de réduire le temps d'exécution de la phase la plus lente. Il est au contraire intéressant d'équilibrer les deux phases de transfert et de conversion afin d'éviter les pertes d'efficacité. Cette implémentation est certes plus complexe que la première, mais comme nous le verrons, nous disposons de communications asynchrones par envoi de messages, ce qui nécessite la mise en place d'un système de boîte aux lettres. De ce fait la prise en charge du transfert en parallèle avec le traitement se fait de manière naturelle.

Un processeur général actuel est capable d'effectuer les calculs de préparation de 100.000 facettes par seconde. Si l'on suppose que l'horloge du réseau (T_c) est à 20MHz, cela nécessite que $max(c_T, c_C) < 200$ cycles. De ce fait une cellule du réseau ne peut traiter qu'un faible nombre de voxels. Les sous-cubes mémoires sont par conséquent de taille réduite, et dans les paragraphes suivants, nous supposerons toujours qu'ils sont de taille inférieure à la taille des boîtes englobantes des objets (i.e $l, h, p < 16$).

4.2.2 Mode de distribution des données dans le réseau

La majorité des calculs nécessaires à la voxelisation peut se résumer à l'évaluation d'expressions linéaires, tout au moins pour les objets les plus courants rencontrés en synthèse d'images. Ces expressions peuvent être transformées en un calcul de suites numériques, car les variables des polynômes ne prennent que des valeurs entières (il s'agit de coordonnées de l'espace discret).

Prenons l'exemple d'une droite 2D. Elle est représentable par l'équation linéaire $y = a \cdot x + b$, qui peut être réexprimée sous la forme : $y_i = y_{i-1} + a$, avec $y_0 = b$.

Cette transformation est extrêmement intéressante lorsque l'expression doit être calculée pour toutes les valeurs possibles de la variable (ici x). En effet on peut utiliser une version incrémentale de la boucle de calcul de l'expression :

```
Pour x=X0 jusqu'à x=X1
|  y = a.x + b
FinPour
```

⟹

```
y = a.X0 + b
Pour i=1 jusqu'à (X1-X0)
|  y = y + a
FinPour
```

On remplace ainsi les multiplications par des additions successives. Le calcul incrémental est possible pour toutes les expressions linéaires, quelque soit leur degré et le nombre de variables. Cette particularité est largement exploitée en synthèse d'images.

Il existe une autre différence primordiale dans le fonctionnement de l'algorithme incrémental. Celui-ci étant équivalent au calcul d'une suite, l'évaluation d'une valeur de l'expression devient dépendante de l'évaluation de la valeur précédente. Si le calcul est réparti sur un ensemble de processeurs (i.e. y_i est calculé sur le processeur i), il existe alors une dépendance entre les processeurs, le processeur i ayant besoin de connaître la valeur calculée par le processeur $i-1$.

Deux implémentations sont envisageables : multipipeline ou arbre (Figure 4.4 a et b).

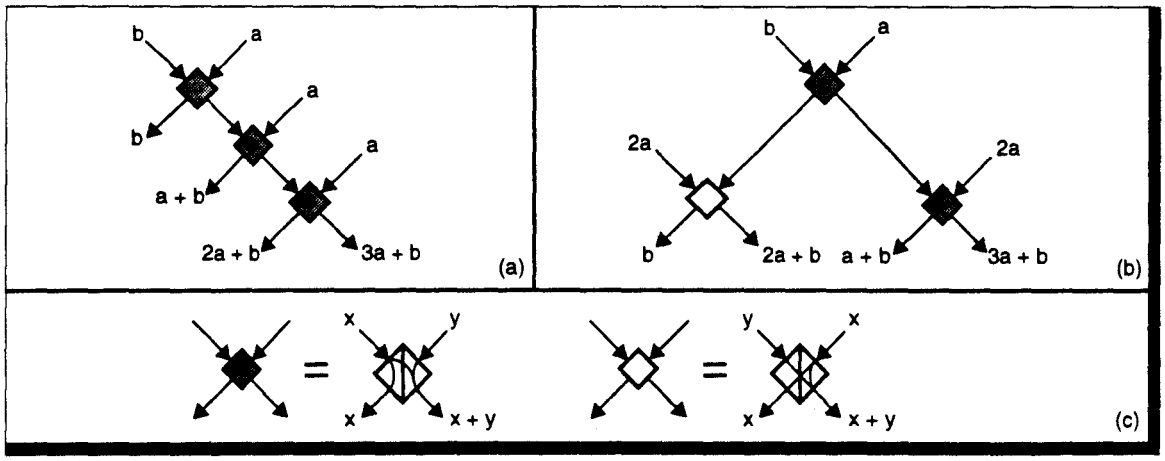


Figure 4.4 ***Calcul incrémental d'une expression linéaire.*** *(a) - Par multipipeline. (b) - Par arbre. (c) - Fonction des deux types de noeud.*

L'évaluation d'expressions à deux variables (exemple, $z(x, y) = a \cdot x + b \cdot y + c$) est effectuée par composition soit de plusieurs pipelines, soit de plusieurs arbres (Figure 4.5) : on peut poser $f(y) = b \cdot y + c$, d'où $z(x, y) = a \cdot x + f(y)$. On obtient ainsi deux expressions monovariables. $f(y)$ est évalué par un premier élément (pipeline ou arbre), et pour chaque y, un autre élément

détermine les valeurs $z(x, y)$. La même extension est possible pour des expressions à trois variables.

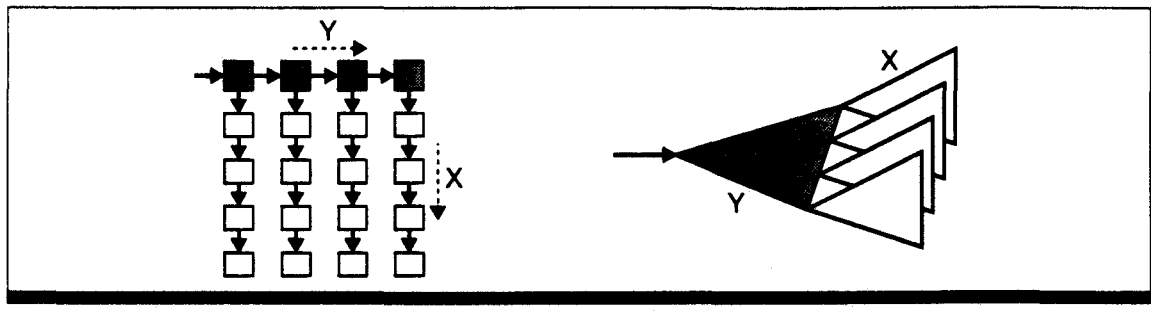


Figure 4.5 ***Calcul incrémental d'expression linéaires à deux variables.***

Arbre d'évaluation

Cette méthode correspond au choix effectué par H.Fuchs pour la réalisation de la machine Pixel-Planes [Fuch81]. Cette machine dispose d'une grille de processeurs pixels SIMD, permettant d'effectuer un certain nombre de calculs sur les pixels (éclairement par exemple). La conversion des objets en pixels utilise un remplissage par équations. Un arbre d'additionneurs, externe à la grille, permet l'évaluation parallèle et synchrone des expressions linéaires. Ce fonctionnement synchrone est l'atout essentiel de l'arbre d'évaluation. Cependant sa réalisation physique peut être assez problématique, surtout pour un arbre 3D, de par la disposition non régulière des signaux de données.

Multipipeline d'évaluation

Le projet RC étudié à Lille par E.Lepretre et A.Atamenia est le pendant de la machine Pixel-Planes. Il s'agit d'un réseau pixel 2D permettant l'affichage de facettes avec éclairement de Gouraud, et utilisant le mode de transmission multipipeline [Lepr89] [Atam89].

Ce mode permet d'intégrer l'évaluation des expressions linéaires au sein des cellules du réseau. On dispose alors des caractéristiques opposées au modèle précédent :

- intégration plus aisée de par la régularité de la distribution des signaux,
- fonctionnement non synchrone, ou plus exactement, fonctionnement du type pipeline (les processeurs sont synchrones, mais les données ne sont pas disponibles au même instant).

Choix effectué

Nous avons préféré retenir la solution multipipeline par souci de faisabilité. Nous pensons, en effet, que la réalisation d'un arbre d'évaluation 3D est trop complexe techniquement. De plus le multipipeline permet d'utiliser les cellules du réseau pour l'évaluation finale (i.e. en X) des expressions, d'où une économie substantielle de matériel.

4.2.3 Implémentation d'un réseau multipipeline

Suivant le type de réseau utilisé, on adoptera un schéma pipeline pour un réseau 1D, un multipipeline double pour le réseau 2D et triple pour le réseau 3D (Figure 4.6).

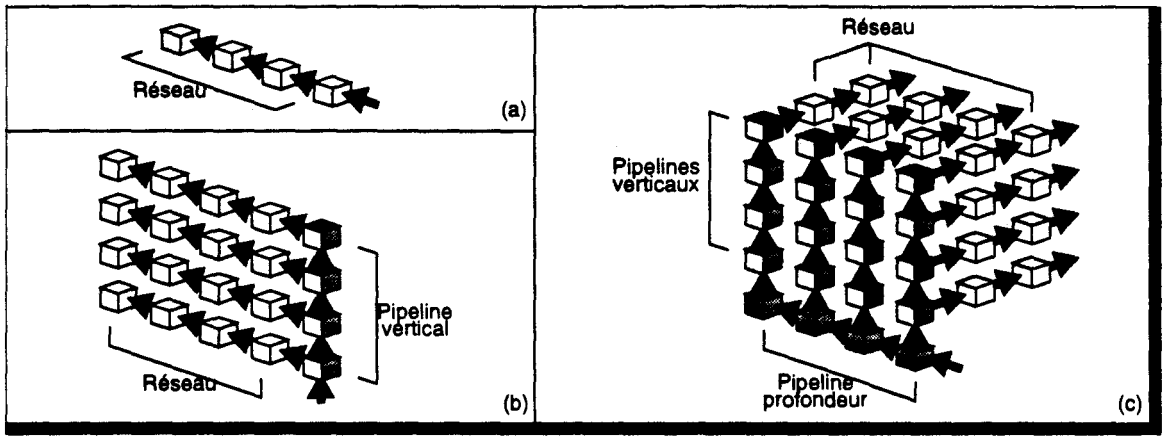


Figure 4.6 ***Distribution multipipeline.*** *(a) - Cas 1D. (b) Cas 2D. (c) Cas 3D (vue partielle).*

Ici, le (ou les) pipeline(s) d'accès au réseau sont câblés. Cette solution offre plusieurs avantages :

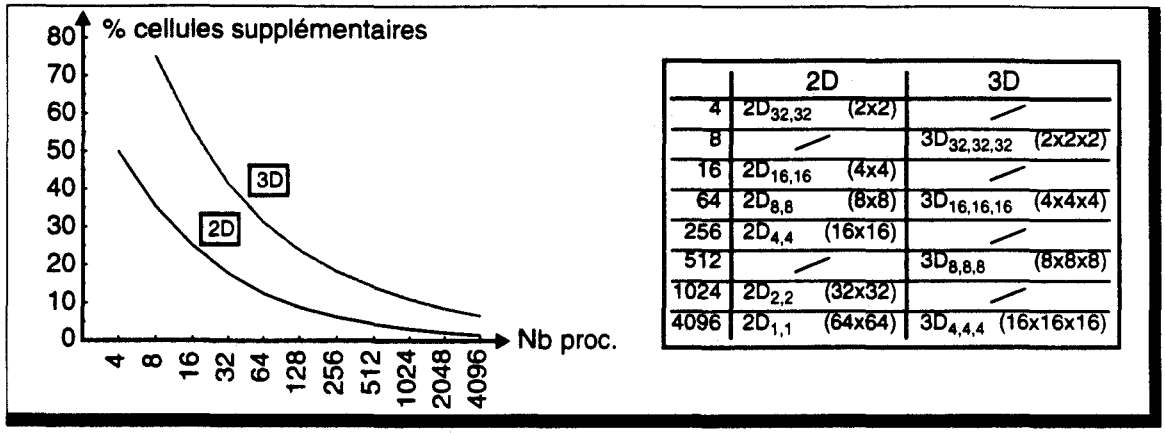
1. Les processeurs du bord du réseau n'ont à prendre en compte la transmission des informations que dans une seule direction, ce qui diminue c_C, et augmente donc le débit d'entrée.
2. Si le traitement des transmission dans les autres directions perpendiculaires était exécuté par les processeurs de bord, il y aurait diminution sensible de l'efficacité globale du réseau, les autres cellules internes du réseau n'ayant pas à effectuer ce traitement.
3. Les cellules supplémentaires pourront être utilisées à d'autres fonctions, comme par exemple le routage des pixels en fin de calcul d'image.

Le multipipeline câblé présente cependant un inconvénient majeur, qui est l'augmentation du coût silicium (sauf pour l'organisation 1D). Le rapport performance/coût serait donc à analyser afin de déterminer la validité d'un tel choix. Il est difficile sans chiffres de calculer la modification de ce rapport. Cependant, nous pouvons considérer que :

- Si le réseau est entièrement câblé, alors de toute façon le matériel permettant le transfert dans les directions perpendiculaires devra être ajouté dans les cellules de bords.
- Si les cellules ont en charge un grand nombre de voxels, alors le réseau est principalement constitué de mémoire (comparativement aux unités logiques). Les pipelines d'accès ne comportent pas de mémoire, et dans ce cas, l'accroissement de matériel peut être supposé négligeable.
- Si le réseau est fortement découpé (grand nombre de cellules ayant peu de voxels en charge), alors le temps de conversion est faible, et le temps de traitement des transferts deviendrait non négligeable.

Ces remarques montrent que le rapport performance/coût est en faveur de l'utilisation de pipelines d'accès extérieurs au réseau. Cependant, si ces derniers n'ont aucune utilité durant les autres phases du traitement total, alors leur intérêt pourra être remis en cause.

Le surcoût (en nombre de cellules) des pipelines externes est représenté Figure 4.7, en fonction du nombre de processeurs constituant le réseau (on suppose un réseau régulier, i.e dont le nombre de cellules est identique dans toutes les directions, i.e $l = h = p$).



	2D	3D
4	$2D_{32,32}$ (2x2)	
8		$3D_{32,32,32}$ (2x2x2)
16	$2D_{16,16}$ (4x4)	
64	$2D_{8,8}$ (8x8)	$3D_{16,16,16}$ (4x4x4)
256	$2D_{4,4}$ (16x16)	
512		$3D_{8,8,8}$ (8x8x8)
1024	$2D_{2,2}$ (32x32)	
4096	$2D_{1,1}$ (64x64)	$3D_{4,4,4}$ (16x16x16)

Figure 4.7 ***Surcoût des pipelines d'accès en fonction du nombre de processeurs du réseau.*** *Le tableau donne un exemple d'organisation en fonction du nombre de processeurs, pour N_V=64. Entre parenthèses, est donné la répartition des processeurs.*

4.2.4 Utilisation d'un réseau partiel

Jusqu'à présent, nous avons considéré que le découpage de l'espace voxel était total, c'est-à-dire qu'un processeur du réseau de conversion à en charge un sous-cube unique, composé de voxels adjacents. Pour diminuer le nombre de processeurs du réseau, il est nécessaire d'augmenter la taille des sous-cubes gérés par ceux-ci. Une autre organisation permet d'effectuer le réduction du réseau sans modifier les tailles mémoires.

Des travaux effectués sur les architectures de rendu classique utilisant un parallélisme massif pixel ont montré l'avantage de l'utilisation d'un réseau *partiel* [Moln91] [Karp93]. Dans ce cas, l'écran est découpé en zones rectangulaires. Le réseau effectue la conversion des pixels appartenant à une zone virtuelle, et il est appliqué successivement à chaque portion de l'image réelle (Figure 4.8).

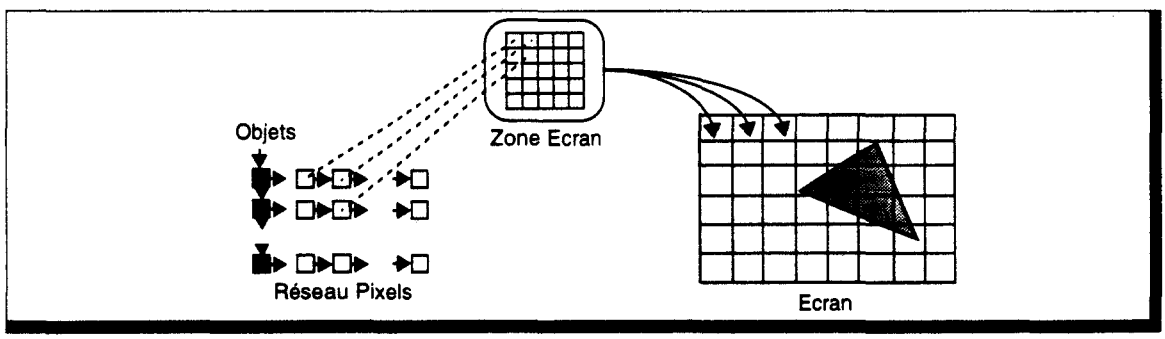


Figure 4.8 ***Parallélisme massif pixel partiel, pour le rendu classique.***

Un objet appartenant à plusieurs zones-écran devra être converti au sein de chacune des zones où il apparaît, ce qui implique que globalement un plus grand nombre d'objets seront transmis au réseau par rapport au cas du parallélisme total. Le nombre moyen de zones auxquelles appartient un objet est appelé *facteur de duplication*. Il est défini en fonction de la taille des zones, et de la taille moyenne du rectangle englobant des facettes :

$$d = \left(\frac{l_Z + l_E}{l_Z}\right) \cdot \left(\frac{h_Z + h_E}{h_Z}\right) \qquad \textbf{(Eq. 4.1)}$$

avec l_Z, h_Z : largeur, hauteur des zones-écran,

l_E, h_E : largeur, hauteur moyenne des rectangles englobants.

Ce facteur est important, car il agit sur les performances réelles de conversion. En effet si la *puissance* intrinsèque du réseau est de F facettes/s, la duplication des objets implique une puissance réelle de F/d facettes/s. Il est donc important de diminuer le taux de duplication, mais cela implique une augmentation de la taille du réseau, d'où la nécessité d'un compromis.

Cette approche peut également être utilisée pour réduire le coût d'une machine voxel, du moins en ce qui concerne la phase de voxelisation. Il faudra vérifier sa validité pour les autres traitements.

Il nous faut, ici, contrairement au rendu classique, conserver les résultats de voxelisation de l'ensemble des zones, puisque le cube voxel complet est nécessaire pour les phases suivantes. On obtient alors un *pliage* des sous-cubes sur le réseau, i.e. un processeur a en charge plusieurs portions disjointes de l'espace (Figure 4.9).

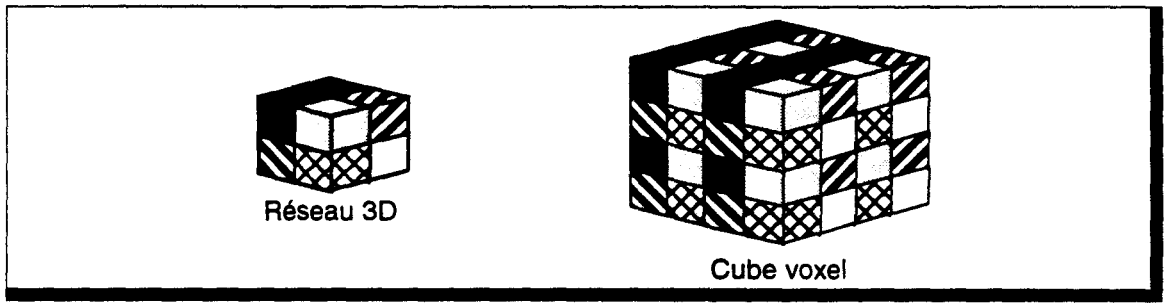


Figure 4.9 ***Enroulement du cube voxel sur un réseau partiel.*** *Exemple sur un réseau 3D de 2x2x2 cellules. Un processeur a en charge les sous-cubes voxel de même teinte que lui.*

Il serait possible de ne pas considérer cette organisation comme un réseau partiel, mais plutôt comme une distribution circulaire des sous-cubes de l'espace sur un réseau *complet*. Il existe cependant une différence quant à l'efficacité de ces deux approches :

- si l'on considère un traitement par zone, les processeurs n'effectueront, à une étape donnée, que la conversion des objets appartenant à la zone en cours. Plus la zone est petite, plus l'efficacité du parallélisme est importante.
- si l'on considère un traitement par pliage, alors un processeur peut très bien avoir en charge des voxels où n'apparaissent aucun objet. L'efficacité est alors bien plus faible.

Le parallélisme partiel permet donc d'améliorer le temps d'exécution de certains algorithmes de voxelisation. Le fait que l'espace voxel est replié sur les différents processeurs devra cependant être pris en compte lorsque l'accès à l'ensemble du cube voxel est nécessaire, ce qui est le cas pour le Lancer de Rayon.

Il nous faut reformuler la valeur du taux de duplication pour prendre en compte la troisième dimension de l'espace voxel. Nous noterons L_Z, H_Z, P_Z la largeur, hauteur et profondeur d'une zone-espace, exprimées en nombre de voxels. Nous définirons de la même manière L_E, H_E, P_E, la largeur, hauteur et profondeur moyennes des *cubes* englobants.

On pose $l_Z = \left\lceil \frac{L_Z}{l} \right\rceil$, la largeur d'une zone-espace exprimée en nombre de sous-cubes voxels.

On définit également h_Z, p_Z, l_E, h_E, p_E en prenant le sous-cube voxel comme unité de mesure.

On peut alors étendre le calcul de taux de duplication [Moln91] aux 3 cas d'organisation parallèles :

$$d_{3d} = \left(\frac{l_Z + l_E}{l_Z}\right) \cdot \left(\frac{h_Z + h_E}{h_Z}\right) \cdot \left(\frac{p_Z + p_E}{p_Z}\right), \quad \text{taux de duplication 3D}$$

$$d_{2d} = \left(\frac{l_Z + l_E}{l_Z}\right) \cdot \left(\frac{h_Z + h_E}{h_Z}\right), \quad \text{taux de duplication 2D}$$

$$d_{1d} = \left(\frac{l_Z + l_E}{l_Z}\right), \quad \text{taux de duplication 1D}$$

La Figure 4.10 donne un exemple du taux de duplication 3D pour différentes tailles de cubes englobants et de zone-espaces. Cette figure nous montre l'importance du choix de la taille du réseau en fonction de la taille moyenne des englobants. Cette dernière est fonction du type d'application de la base de données d'objets.

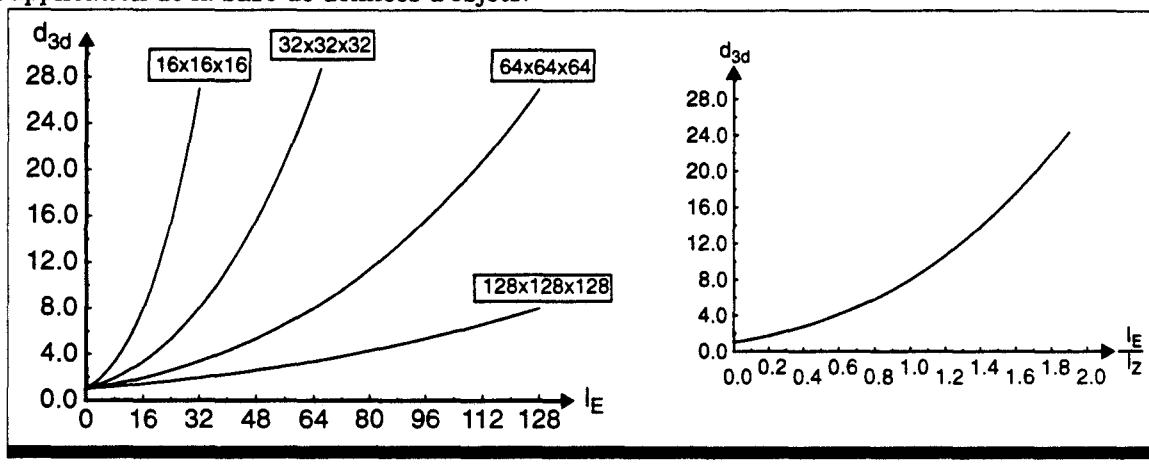


Figure 4.10 ***Taux de duplication 3d.*** *Exprimé en fonction de la taille des cubes englobants, pour 4 tailles de zone-espace possible. On considère les trois dimensions égales, i.e.* $l = h = p$, $l_Z = h_Z = p_Z$ *et* $l_E = h_E = p_E$. *La figure de droite exprime le taux de duplication en fonction du rapport de taille zone/cube englobant.*

Les performances des machines de rendu classique sont définies pour des englobants 2D de taille 20x20 pixels, ce qui justifie l'utilisation de réseau 32x32 ($d_{2d} = 2,6$) [Karp93].

Si nous considérons des englobants 3D de taille 20x20x20 voxels, alors il serait nécessaire de mettre en oeuvre un réseau de 64x64x64 processeurs, pour obtenir un taux de duplication équivalent.

4.2.5 En résumé

- Des deux schémas parallèles possibles, nous avons choisi de n'étudier que les machines à mémoire voxel distribuée.
- Trois organisations de distribution sont envisageables : réseau de processeurs $3d_{l,h,p}$, $2d_{l,h}$, ou $1d_l$. La taille des sous-cubes permettra de déterminer, par exemple, la puissance intrinsèque de voxelisation P_l.
- L'alimentation du réseau est effectuée par l'intermédiaire d'un multipipeline d'accès, qui peut soit être extérieur au réseau, soit utiliser les processeurs de bord du réseau.
- Une parallélisation partielle peut être utilisée pour diminuer le réseau. La taille de celui-ci permet, en fonction de la taille moyenne de l'englobant des objets, de déterminer le taux de duplication d qui fixe la puissance réelle de conversion $P_R = P_l / d$.

Nous obtenons donc un grand nombre de choix d'implémentations, que nous allons préciser par l'étude des différents algorithmes de voxelisation.

4.3 Etude des algorithmes de voxelisation

4.3.1 Voxelisation par balayage aléatoire

On trouve principalement dans cette catégorie 2 méthodes de conversion : l'extension 3D et l'extrusion. Elles ont en commun la voxelisation d'un objet par suivi de son contour.

Algorithmes de voxelisation par extension 3D

Il existe 4 algorithmes de ce type. Leur complexité est linéaire en fonction du nombre de voxels générés.

Tracé de ligne 3D (algorithme de Kaufman [Cohe91]).

Cet algorithme est une extension du tracé de droite de Bresenham [Bres65]. En partant d'une des extrémités de la droite, un calcul de distance minimale permet de déterminer le meilleur voxel suivant le long de la droite. De proche en proche, la droite est ainsi construite.

Kaufman propose un algorithme pour les trois connexités possibles. Il faut cependant indiquer qu'il n'existe aucune définition formelle de la droite 3D (une droite réelle est définie par un système de deux équations, ce qui n'a pas d'équivalent en arithmétique discrète). Il n'est donc pas possible de juger de la qualité des droites générées.

Tracé de cercle 3D (algorithme de Kaufman [Cohe91]).

Il n'existe pas de définition formelle de cet objet. Par conséquent, on ne peut pas déterminer un algorithme de tracé exact du cercle 3D. La méthode retenue ici consiste à :

- déterminer le plan principal correspondant au plan de plus grande pente du cercle 3D.
- suivre le contour de l'ellipse 2D correspondant à la projection du cercle sur ce plan.
- pour chaque point de l'ellipse, calculer la troisième coordonnée de l'espace.

L'algorithme ne permet pas de spécifier la connexité du cercle obtenu. Il assure juste qu'il n'est pas 26-connexe.

Tracé de polygone 3D (algorithme de Kaufman [Kauf87a]).

Là aussi, la méthode consiste à utiliser une projection 2D :

- détermination du plan principal correspondant au plan de plus grande pente du polygone.
- remplissage par suivi de contour du polygone dans ce plan (algorithme de *remplissage à godets*). Le suivi des arêtes utilise une extension 3D de l'algorithme du DDA.

L'auteur indique que le polygone est sans trou 6-connexe, sans pouvoir spécifier ou agir sur la connexité du plan.

Courbe, Surface, Volume paramétrique 3D (algorithme de Kaufman [Kauf87b]).

La méthode utilisée permet le tracé de courbes, surfaces et volumes paramétriques cubiques[1].

Un motif de points initiaux est calculé sur une des extrémités de l'objet. Puis à l'aide des différences finies de troisième ordre, les voxels suivants sont définis de proche en proche. Cette algorithme est par conséquent équivalent à un suivi de contour, ce qui nous l'a fait classer avec les autres méthodes du même type.

Les surfaces et volumes sont générés par voxelisation successive de courbes paramétriques.

La conversion des courbes est possible dans chacune des 3 connectivités, les surfaces sont sans trou 6-connexe, les volumes sont uniquement sans trou.

1. Polynômes de degré 3 en paramètres.

Implémentation parallèle

Les méthodes précédentes étant toutes basées sur le suivi de contour, elles ont donc toutes en commun une caractéristique primordiale : la voxelisation débute en une des extrémités de l'objet, le voxel suivant peut être situé dans n'importe quelle direction. En particulier, le suivi du contour peut réimpliquer des processeurs du réseau ayant déjà servi dans la voxelisation de l'objet courant (comme dans le cas du cercle, par exemple). De même, il n'existe pas de direction privilégiée de *propagation* dans l'ordre spatial de création des voxels, surtout dans le cas de surfaces concaves.

Ces remarques impliquent que :

- il n'est pas possible d'utiliser l'effet multipipeline pour injecter un objet par cycle horloge dans le réseau, sous peine d'aboutir à des conflits d'accès, une cellule pouvant recevoir des données de conversion depuis plusieurs de ces voisins.
- les processeurs doivent être complètement connectés entre voisins (topologie de grille 3D).

Par conséquent, ces algorithmes semblent mal adaptés à la parallélisation sur réseau multipipeline.

Algorithmes de voxelisation par extrusion

Cohen, dans le cadre du projet Cube, propose une technique de génération de cylindres généralisés 3D, ayant comme base le cercle 3D [Cohe91]. Un cylindre généralisé est défini comme le produit tensoriel d'un objet N_B-connexe, appelé *base*, et d'un chemin N_G-connexe, appelé *génératrice*. Si $N_B = 6$ et $N_G = 26$, l'objet obtenu par extrusion est sans trou 6-connexe. Si $N_B = N_G = 6$, l'objet extrudé est sans trou 26-connexe. Il n'est pas possible de préciser plus finement la connexité de l'objet créé.

Un certain nombre d'objets peuvent être construits par extrusion :

- Cylindre 3D
  L'objet de base est un cercle 3D précalculé (ou tout autre objet correspondant à la base du cylindre), et recopié le long d'une ligne 3D.
- Cône 3D
  Dans le cas du cône, l'objet de base ne peut plus être simplement recopié, il doit subir une déformation lors de son application le long de la génératrice. Le cône peut être généré de deux manières possibles : voxelisation de droites partant du cercle de base du cône jusqu'au sommet, ou voxelisation de cercles se rétrécissant depuis la base jusqu'au sommet du cône. Dans le premier cas on peut garantir un cône sans trou 6-connexe, alors que dans le deuxième cas, on obtient un cône sans trou 26-connexe.
- Tore 3D
  Un cercle correspondant à la section du tore est utilisé comme génératrice sur laquelle sont appliqués des cercles 3D pour constituer l'enveloppe du tore. Il est sans trou 6-connexe.
- Sphère 3D
  Une méthode identique à la précédente permet la création de sphères sans trou 6 ou 26-connexe.

Le gros inconvénient de ces méthodes est la nécessité de précalculer un motif de base, stocké sur la machine hôte, ou la nécessité de recalculer ce motif le long de la génératrice. Le motif de base utilise un balayage aléatoire et l'on se heurte donc aux mêmes problèmes que ceux évoqués précédemment en ce qui concerne la parallélisation sur le réseau de conversion.

De plus aucune assurance de connexité ne peut être donnée, ce qui peut rendre leur utilisation problématique. J.Delfosse a utilisé une extrusion le long d'un des axes principaux, auquel cas la connexité obtenue peut être spécifiée [Delf93]. Cependant, la classe d'objets ainsi définie est trop restreinte.

En résumé

Les méthodes de cette classe (balayage aléatoire) sont peu adaptées au type de machine que nous avons choisi pour cette étude, du moins si l'on cherche à atteindre des performances de conversion élevées. Elles semblent bien plus adaptées à une utilisation sur une architecture à mémoire partagée.

Si toutefois elles sont utilisées sur le processeur hôte, avec chargement des voxels en mémoire après transformation, la durée globale de ce transfert sera directement proportionnel au nombre de voxels à stocker.

4.3.2 Voxelisation par balayage plan unique

Etant donné qu'un objet 3D présente en général plusieurs voxels en une position (x, y) donnée, la voxelisation par balayage plan unique nécessite la mise en oeuvre de la méthode de construction. Celle ci a été spécialement développée dans le cadre d'un réseau 2D, mais nous étudierons ses caractéristiques pour les trois types de réseau.

On ne trouve dans cette catégorie que deux algorithmes : voxelisation de polygones et de droites 3D.

Polygone 3D par construction

A partir des coefficients de l'équation du plan support du polygone, Vidal calcule une information de seuil permettant de définir une longueur de palier en Z, ce qui permet d'obtenir deux valeurs de profondeur avant et arrière [Vida92]. Il faut remarquer que la notion de seuil utilisée par Vidal est identique à la notion de distance arithmétique permettant la balayage diophantien. Cette valeur est cependant calculée empiriquement pour assurer une 18-connexité.

Le plan XY est complètement balayé pour créer les dexels[1] correspondant au plan support. Pour créer le polygone, ce plan est découpé à l'aide des plans perpendiculaires au polygone et passant par les arêtes : pour chaque position (x, y) , les profondeurs de ces plans sont calculées afin de limiter les dexels.

Droite 3D par construction

La méthode de construction proposée par Vidal a été reprise pour créer un algorithme de voxelisation de droite (cf. Annexe E). Il est bien connu qu'une droite 2D est composée d'une succession de paliers de deux longueurs possibles, dont il est possible de déterminer la taille et la position au sein de la droite. A partir de cette constatation, une méthode de tracé de droite 3D par palier a pu être proposée :

- l'algorithme de tracé de droite 2D d'Amanatides et Woo [Aman87] est utilisé sur la projection de la droite sur le plan XY,
- à chaque déplacement en X (ou en Y, suivant la pente de la droite), un nouveau palier de droite est construit par définition de la profondeur avant et arrière de cette droite.

Implémentation parallèle

L'utilisation d'un balayage selon le plan XY permet de profiter complètement de l'effet multipipeline pour injecter un objet dans le réseau à chaque cycle de conversion.

Le nombre de cycles horloge (c_C) correspondant à un cycle de conversion est dépendant de l'implémentation exacte de la méthode de voxelisation par construction :

1. Dexel = Depth Element, par analogie au pixel. Il s'agit d'un ensemble de voxels en profondeurs situés en une position (x,y) donnée.

- Si seules les valeurs de profondeur avant et arrière sont nécessaires, alors c_C est uniquement fonction du nombre de dexels gérés par un processeur :

$$c_C = l \cdot h \cdot c_V \qquad \text{pour un réseau 3D ou 2D}$$

$$c_C = l \cdot h_E \cdot c_V \qquad \text{pour un réseau 1D}$$

avec c_V : Nombre de cycles pour la conversion d'un dexel

L'utilisation d'algorithmes incrémentaux permet d'assurer un nombre de cycles identique sur chacun des processeurs du réseau, puisque chaque cellule exécute exactement le même code et communique à sa voisine les données nécessaires à cette exécution.

- S'il est nécessaire de *remplir* les paliers, c'est-à-dire s'il faut spécifier complètement l'ensemble des voxels constituant l'objet, il faut ajouter à la valeur c_C précédente le nombre de cycles de remplissage, qui est fonction de la profondeur des paliers, et du nombre de dexels appartenant effectivement à l'objet. Cette donnée est dépendante du type et de l'orientation de l'objet. Nous pouvons cependant prendre la profondeur du sous-cube mémoire comme un majorant de la taille d'un palier :

$$C_C \leq (l \cdot h \cdot c_V) + (l \cdot h \cdot p \cdot c_R) \qquad \text{pour un réseau 3D}$$

$$C_C \leq (l \cdot h \cdot c_V) + (l \cdot h \cdot p_E \cdot c_R) \qquad \text{pour un réseau 2D}$$

$$C_C \leq (l \cdot h_E \cdot c_V) + (l \cdot h_E \cdot p_E \cdot c_R) \qquad \text{pour un réseau 1D}$$

avec c_R : Nombre de cycles nécessaire au remplissage d'un voxel ($c_R < c_V$)

Cette fois-ci, c_C n'est plus constant, ce qui ne permet plus le transfert synchrone de données entre processeurs. Deux possibilités sont envisageables :

- communication asynchrone avec tampons de messages. Il faut dans ce cas définir une taille de tampon évitant tout remplissage complet, ou interrompre les transferts jusqu'à vidage partiel du tampon plein. Cette solution est complexe en temps et en matériel, même si elle permet théoriquement une adaptation automatique du flux et un débit plus élevé.
- communication synchrone par rendez-vous. Un processeur attend que son voisin ait terminé son exécution avant de lui transmettre les données. Cette solution ne nécessite aucun matériel supplémentaire, mais ne permet aucune régulation de flux. Le débit sera fortement influencé par le processeur le plus lent.

Désirant utiliser une machine massivement parallèle, il faut réduire le coût matériel des cellules du réseau et nous préférerons donc la deuxième solution, même si cela risque de diminuer quelque peu les performances globales de conversion.

Afin de caractériser plus précisément les longueurs de paliers, nous avons pris en exemple une sphère facettisée (Figure 4.11), car elle contient des triangles dans toutes les orientations possibles. A partir de mesures sur des sphères de tailles différentes, on obtient une profondeur moyenne de palier de 5 voxels, pour une profondeur maximale de 41 voxels (ceci pour des triangles dont les arêtes mesurent jusqu'à 155 voxels).

La faible valeur de profondeur moyenne s'explique aisément par observation de la répartition des différentes longueurs de palier sur les triangles composant la sphère (Figure 4.12) : la majeure partie des triangles sont composés de paliers de profondeur inférieure à 5 voxels.

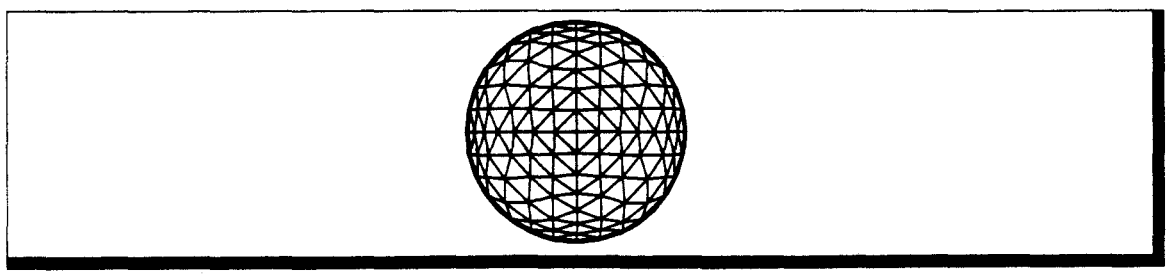


Figure 4.11 ***Sphère facettisé utilisée pour l'étude des profondeurs de paliers.*** *La sphère réellement utilisée est subdivisée deux fois plus finement.*

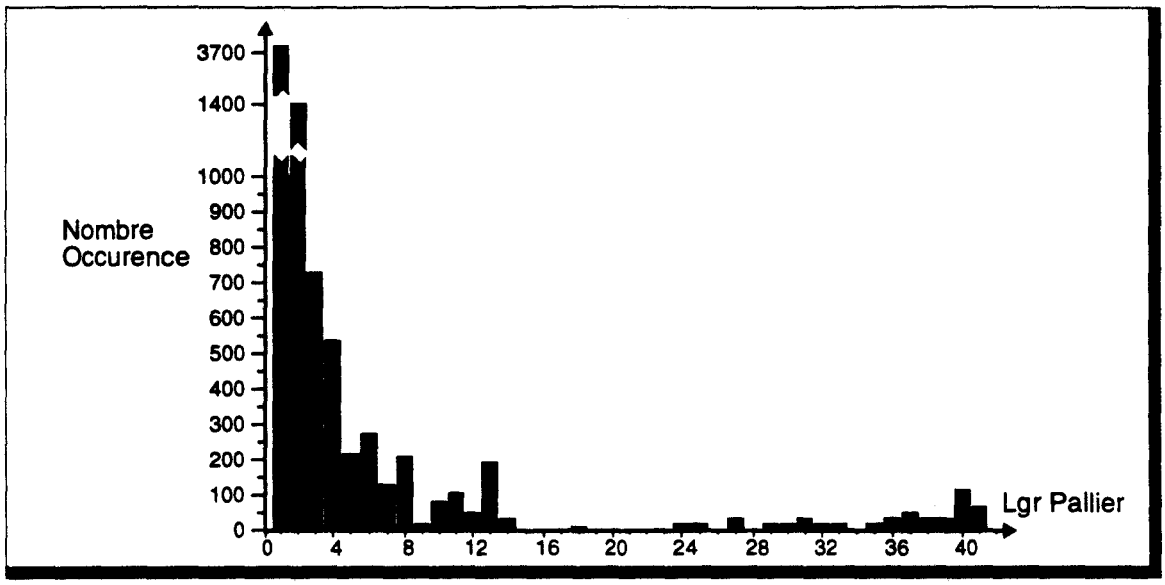


Figure 4.12 ***Nombre d'occurrence des différentes profondeurs de palier. Mesure effectuée sur une sphère de rayon 2048. L'arête la plus longue est de 155 voxels.***

Nous allons maintenant préciser l'implémentation de cette catégorie d'algorithmes sur les trois organisations de réseau.

Réseau 3D

La technique de *construction* calcule en chaque position (x, y) les profondeurs avant et arrière des paliers constituant l'objet. Les calculs incrémentaux en Y seront naturellement implémentés dans le pipeline vertical, les incréments en X étant réalisés au sein des pipelines du réseau. Dans le cas du réseau 3D, on dispose de plusieurs processeurs en profondeur, chacun d'eux pouvant *posséder* un *morceau* du palier. Deux implémentations sont envisageables :

1. Chaque processeur calcule les deux valeurs Z_{AV} et Z_{AR}. Ils exécutent donc le même code sur les mêmes données, ce qui implique une redondance de calcul et par conséquent une diminution d'efficacité.
2. Seuls les processeurs en bord de réseau (i.e. situé sur le plan avant) exécutent le calcul. Les deux valeurs de profondeurs sont, ensuite, transmises aux autres cellules de même position. Dans ce cas, il est nécessaire d'utiliser un dispositif de communication entre voisins selon la direction Z, ce qui n'est pas inclus dans l'organisation multipipeline. D'autre part, seuls les processeurs de bords étant impliqués dans le calcul, les autres cellules sont en attente, ce qui diminue l'efficacité globale. Il est enfin nécessaire d'ajouter, au temps de conversion, le temps de transfert des profondeurs, celui-ci étant d'autant plus important qu'un processeur traite plusieurs dexels, un transfert étant effectué pour chaque dexel du sous-cube.

La première solution est préférable.

Le remplissage des paliers, par contre, est exécuté en parallèle sur tous les processeurs ayant en charge les voxels appartenant au palier. Mais au vu de la taille moyenne des paliers, cela implique, en moyenne, au plus 2 cellules, sauf si les sous-cubes mémoires sont de profondeur inférieure à 5.

En tout état de cause, la topologie 3D sera largement sous-utilisée. Cela provient, bien entendu, du fait que les algorithmes par construction ont été créés pour les organisations 2D.

Réseau 2D

Sur un réseau de ce type, il faut veiller à limiter au maximum les boucles de calcul en Z, ce qui a amené l'étude de la voxelisation par construction. Le seul calcul en Z inévitable est le remplissage des paliers, mais comme nous l'avons vu celui-ci est minime.

Le cycle de conversion est : $c_C \approx (l \cdot h \cdot c_V) + (5 \cdot l \cdot h \cdot c_R)$. Pour la voxelisation de facettes, on a $c_V + 5 \cdot c_R \approx 40$, ce qui implique que pour obtenir un cycle de conversion de 200 cycles horloges permettant un débit de 100.000 facettes/s, il faut $l \cdot h < 5$. Dans ce cas un cube voxel de dimension $N_V = 512$ nécessite un réseau de 256×256 cellules.

Inversement un réseau, plus raisonnable, de taille 64×64 processeurs permet la conversion de ~8000 facettes/secondes, et, dans ce cas, l'utilisation des cellules pour l'exécution de la voxelisation ne se justifie plus.

Nous donnons en Figure 4.13 une estimation de la puissance nécessaire à un processeur classique en fonction des performances de conversion attendues, comparée à la taille de réseau permettant les mêmes performances (pour $N_V = 512$, et un réseau complet). Il va de soi que le rapport performance/coût silicium est en faveur de la solution processeur classique, bien que pour atteindre des performances élevées, un seul processeur n'ait plus la puissance nécessaire (du moins avec la technologie actuelle). De plus l'efficacité du réseau est faible, car peu de cellules sont réellement impliquées dans la conversion.

Nous avons déjà indiqué que la puissance de conversion intrinsèque du réseau n'est pas fonction de sa taille, mais de la taille du sous-cube que chaque cellule à en charge. L'utilisation d'un réseau partiel présente ici un intérêt certain puisqu'il permet de diminuer la taille réseau, tout en conservant les performances de conversion (au taux de duplication près). On trouvera en Table 4.1 quelques exemples de performances pour divers réseaux partiels.

En conséquence, une solution composée d'un réseau partiel (32×32) permet d'obtenir des puissances intéressantes, pour un rapport performance/coût silicium raisonnable en comparaison à un processeur classique (ou plutôt un multiprocesseur, étant données les puissances nécessaires).

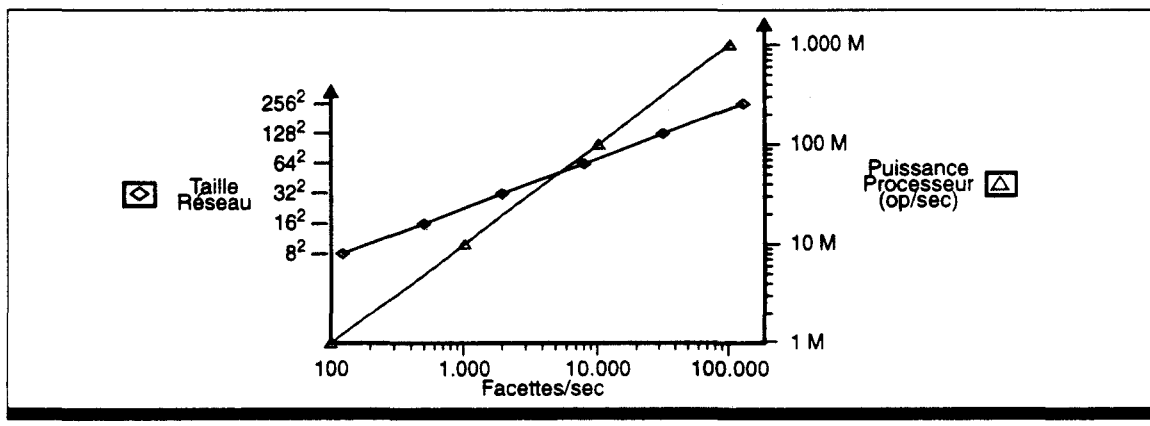


Figure 4.13 ***Comparaison processeur/réseau en fonction des performances de conversion.*** *On considère un réseau complet avec* $N_V = 512$.

Taille Réseau	16x16	32x32	32x32
l.h	4	4	1
N_{VP}	32	64	32
d	2,25	1,56	2,25
P_I	125.000	125.000	500.000
P_R	55.555	80.128	222.222
Op/se	555 M	801 M	2,22 G

l.h : Taille sous-cube voxel	P_I : Puissance intrinsèque
N_{VP} : Taille cube voxel partiel	P_R : Puissance réelle
d : Taux de duplication	Op/sec : Puissance processeur classique équivalente

Table 4.1 ***Performances de conversion pour divers réseaux partiels.***

Réseau 1D

La différence essentielle par rapport au réseau 2D provient de l'absence du pipeline vertical. De ce fait, les cellules doivent exécuter les interpolations verticales. On limitera les calculs sur la hauteur de l'englobant des facettes, d'où :

$$c_C \approx (h_E \cdot c_I) + (I \cdot h_E \cdot c_V) + (5 \cdot I \cdot h_E \cdot c_R)$$

avec h_E : hauteur de l'englobant

c_I : nombre de cycles pour l'interpolation verticale

Les calculs d'interpolation verticale sont pratiquement identiques à ceux de l'interpolation horizontale, d'où $c_I = c_V$, ce qui donne :

$$c_C \approx (h_E \cdot (1 + I) \cdot c_V) + (5 \cdot I \cdot h_E \cdot c_R)$$

Pour la solution 2D, nous avons abouti à un réseau partiel gérant des sous-cubes de faible taille ($I \cdot h = 4$, ou même $I \cdot h = 1$). Or en général, $h_E > 2$; la valeur de c_C dans le cas 1D est donc bien plus importante que pour l'organisation précédente.

Par contre, le nombre de cellules du réseau est largement plus faible, d'autant plus qu'il est possible d'utiliser un réseau partiel en largeur.

Le choix entre le réseau 2D et le réseau 1D porte donc uniquement sur un compromis entre les performances de conversion et le coût silicium.

4.3.3 Voxelisation par balayage plan quelconque

Il n'a pas été proposé d'algorithmes dans cette catégorie. Cependant une modification des méthodes de suivi permet la voxelisation par balayage plan quelconque. Il s'agit alors de voxelisation par extension 3D.

Prenons par exemple le tracé de polygone 3D de Kaufman. La méthode commence par déterminer un plan principal au sein duquel le suivi 2D de contour est effectué. Nous pouvons remplacer le suivi par un remplissage par équation : on teste alors si un point de position (u, v) appartenant au plan principal est a l'intérieur de la projection de la facette. Ce test peut être effectué à l'aide d'un algorithme incrémental, par balayage du plan principal. De par le principe de ces algorithmes, il n'existe qu'un voxel par position (u, v), le plan principal étant choisi dans ce but, sauf lorsque l'on désire un connexité 6, auquel cas il peut exister 2 voxels contigus dans la direction perpendiculaire au plan.

Implémentation parallèle

On restreint le balayage à la boîte englobante de l'objet, afin de minimiser les calculs inutiles. Il semble évident que ce type de balayage n'est efficace que sur un réseau 3D. En effet sur les autres organisations l'efficacité sera faible et le cycle de conversion important lorsque le balayage est effectué dans le plan YZ.

Effectuons une estimation des performances pour le réseau 3D. Celles-ci sont tributaires du plan utilisé pour le balayage. En effet, suivant le cas, les cellules auront ou non un calcul d'interpolation à effectuer. Nous noterons c_I le nombre de cycle nécessaire à une interpolation, et c_R le nombre de cycles nécessaires au remplissage du ou des voxels. Suivant la connexité désirée, il pourra être plus ou moins élevé, mais de toute manière, il reste faible devant c_I. On obtient donc :

- Balayage dans le plan XY: $c_C = (I \cdot h \cdot c_I) + (I \cdot h \cdot c_R)$
- Balayage dans le plan XZ: $c_C = (I \cdot p \cdot c_I) + (I \cdot p \cdot c_R)$
- Balayage dans le plan YZ: $c_C = h \cdot p \cdot c_R$

La valeur de c_I est légèrement inférieure à c_V, puisqu'une seule valeur de profondeur est à calculer. La différence par rapport aux algorithmes précédents se situe donc essentiellement au niveau du remplissage. On obtient en moyenne un gain de 50%.

Une parallélisation partielle est applicable, avec les mêmes retombées que précédemment quant aux performances réelles. Indiquons toutefois que le taux de duplication 3D est plus élevé, ce qui nécessite un réseau de taille $32 \times 32 \times 32$ pour être au niveau de puissance du réseau 32×32 avec balayage plan unique.

4.3.4 Voxelisation par balayage diophantien

Cette méthode est conjuguée à une utilisation de la représentation des objets par équation diophantienne. Le problème majeur de cette technique est la détermination de l'épaisseur arithmétique à utiliser pour obtenir une connexité donnée. Une démarche générale a été établie pour calculer cette valeur [Andr92], mais elle n'est formellement applicable que dans un nombre restreint de cas :

- le plan 3D [Andr92], dans tous les cas de connexité voulu,
- la sphère 3D, 18-connexe [Andr92], ou 6-connexe [Delf93].

D'autres objets ont cependant pu être construit, mais par application intuitive de la démarche, qui permet d'obtenir une valeur d'épaisseur arithmétique assurant une connexité 6; cette épaisseur pouvant parfois ne pas être la valeur minimale :

- l'ellipsoïde 3D [Delf93],
- le tore 3D [Delf93].

Cette technique est applicable pour d'autres objets. Elle doit en particulier permettre la voxelisation des objets quadriques.

Implémentation parallèle

La technique de voxelisation associée consiste, comme indiqué précédemment, à calculer la valeur de l'équation diophantienne en chaque point de l'espace 3D. Le cycle de conversion est par conséquent directement proportionnel au volume du sous-cube géré par chaque cellule. Il faut donc veiller pour l'organisation 2D et 1D à restreindre les calculs au volume englobant des objets, d'où :

- pour un réseau 3D : $c_C = l \cdot h \cdot p \cdot c_V$,
- pour un réseau 2D : $c_C = l \cdot h \cdot p_E \cdot c_V$,
- pour un réseau 1D : $c_C = l \cdot h_E \cdot p_E \cdot c_V$

Les courbes de performance intrinsèque associées sont données en Figure 4.14. On y trouve également la puissance nécessaire pour assurer le balayage diophantien sur un processeur classique. L'utilisation d'un parallélisme partiel étant également envisageable, le rapport performance/coût silicium est cette fois-ci bien plus en faveur du réseau.

Le débit de conversion que nous désirons atteindre est de 100.000 facettes/seconde. Seul le réseau 3D permet de disposer de telles performances, bien qu'un réseau $2D_{1,1}$ soit également intéressant.

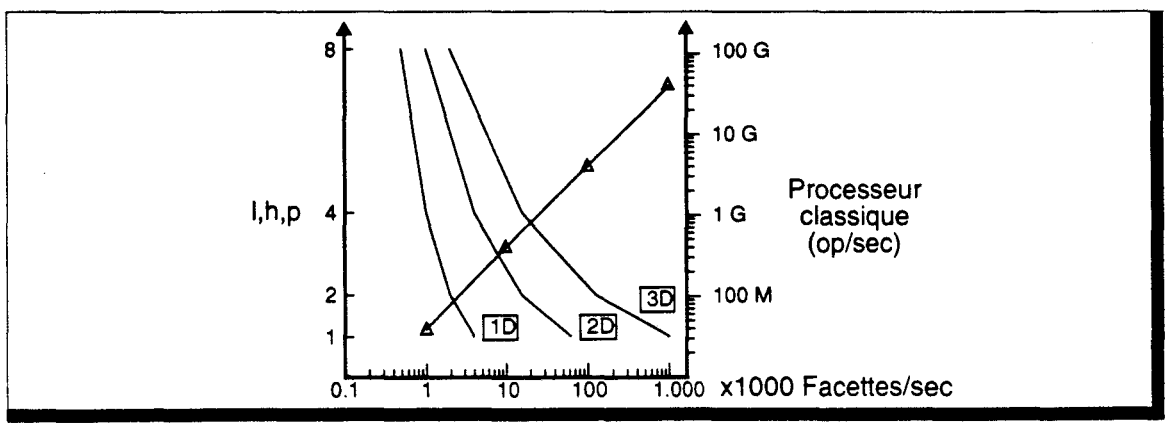


Figure 4.14 ***Performances du balayage diophantien.*** *l,h,p représente la taille des dimensions libres du sous-cube voxel.*

4.3.5 Parallélisation interne de la conversion

Dans les évaluations précédentes, le nombre de cycles nécessaires aux différents calculs suppose que les traitements sont effectués de manière séquentielle par les cellules du réseau. En spécialisant ces dernières il est possible d'envisager un traitement parallèle de la conversion au sein de chaque processeur.

On peut, par exemple, utiliser une unité de calcul du contour, une unité de calcul de la profondeur... On peut ainsi diminuer fortement le temps de cycle de conversion et donc augmenter les performances du réseau.

Cependant ces unités n'ont de raison d'être que pour la phase de conversion, qui ne constitue qu'une partie du traitement. Dans le cadre d'une architecture pour le Lancer de Rayon Discret, nous n'utiliserons pas ce parallélisme interne afin de minimiser le coût matériel.

4.3.6 Voxelisation d'une scène complète

En général, une scène est composée d'un ensemble de morceaux d'objets de base, qui juxtaposés forment des objets à la forme plus complexe (sur les machines classiques, tous les objets sont facettisés).

Pour l'algorithme du Lancer de Rayon, il est nécessaire de garantir la connexité des objets complexes. Si l'on prend l'exemple des polygones, ceux-ci sont obtenus par voxelisation de leur plan support, avec limitation par les arêtes. Cette méthode ne permet pas d'assurer une connexité donnée sur les arêtes. Cela implique qu'il peut exister des trous entre deux facettes ayant une arête géométrique commune. On retrouve bien entendu le même problème pour n'importe quel objet de base que l'on découpe par des plans (patches quadriques, par exemple).

Seuls les algorithmes effectuant un remplissage par suivi de contour peuvent garantir une connexité entre surfaces, car dans ce cas le même objet de contour est utilisé pour définir une arête commune à deux surfaces. Cependant, on ne peut garantir que des surfaces globales sans trou 6-connexe, et de plus, comme nous l'avons indiqué précédemment, ces algorithmes ne sont pas compatibles avec l'utilisation d'un réseau voxel.

Pour les algorithmes par construction ou par balayage diophantien, nous avons utilisé un artifice permettant d'obtenir des surfaces sans trou 6-connexe, bien qu'aucune preuve formelle n'ait été établie : pour effectuer le découpage des surfaces supports nous utilisons des plans situés *légèrement* à l'extérieur des arêtes réelles (on considère en fait des plans décalés d'un *demi-voxel*). De cette façon, nous obtenons un recouvrement des surfaces, qui permet de combler les trous entre surfaces (c'est du moins ce que nous avons observé sur les essais effectués). Il serait intéressant de tenter de formaliser cette méthode afin de justifier son utilisation.

4.3.7 En résumé

Nous résumons dans la Table 4.2 les résultats des études que nous venons d'effectuer. On y trouve pour chaque type d'algorithme le type d'organisation matérielle la plus adaptée, la taille optimale des réseaux, et la taille optimale des sous-cubes voxels, ceci dans le but d'atteindre des performances de conversion de l'ordre ou supérieur à 100.000 facettes/seconde.

De cette analyse, nous pouvons conclure que pour atteindre les performances (relativement faibles) que nous nous sommes fixées, il est nécessaire d'utiliser un parallélisme fin (sous-cube de taille 2 voire 1). Pour limiter le matériel, un parallélisme partiel est utilisable.

Balayage	Voxelisation	Organisation	Taille	Sous-cube
Aléatoire	Extension 3D	(non adapté)	—	—
	Extrusion			
Plan unique	Construction	Réseau partiel 2D	32x32	2x2 ou 1x1
Plan quelconque	Extension 3D	Réseau partiel 3D	32x32x32	2x2x2 ou 1x1x1
Diophantien	Equa. dioph.	Réseau partiel 2D	32x32	1x1
		Réseau partiel 3D	32x32x32	2x2x2 ou 1x1x1

Table 4.2 ***Résumé des organisations optimales en fonction du type d'algorithme de voxelisation.***

L'organisation 1D ne présente pas suffisamment d'avantages par rapport à ses concurrentes *pour être conservée. Le rapport performances/coût est en faveur du réseau 2D, comparé au réseau 3D. Nous conservons donc a priori l'organisation 2D, ce qui élimine les algorithmes à balayage plan quelconque. Cependant les objets voxelisables par cette méthode le sont également par balayage diophantien. Nous n'imposons de ce fait aucune limitation quant au type d'objets utilisables.*

Notons que les tailles de réseau ou de sous-cube sont optimales pour des facettes de boîte englobante 16x16x16. Pour des facettes plus grandes, ou d'autres types d'objets (tels que les quadriques), l'englobant est plus volumineux et nécessite donc des réseaux de taille plus importante pour éviter un taux de duplication trop élevé.

4.4 Implémentation détaillée de la voxelisation de facettes

Si les objets graphiques de haut niveau (cylindres, cônes, surfaces courbes...) présentent un intérêt indéniable en modélisation, l'étude de leur voxelisation n'est pas encore très avancée. Les plans de coupes, par exemple, ne sont pas encore incorporés dans les algorithmes. Nous verrons de plus que quelques difficultés de définition du vecteur normal existent sur ces objets (§ 6.1 - p. 137).

En attendant l'avancement des travaux concernant les objets de haut niveau, nous conservons la facette qui permet de toute manière la modélisation de tout type d'objet complexe.

De ce fait, l'organisation et l'algorithme les plus adaptés sont le réseau 2D et la voxelisation par construction et balayage plan unique. Afin d'obtenir des performances suffisantes, l'algorithme est en partie câblé. Son étude nous permettra de dégager les dispositifs matériels à intégrer.

4.4.1 Définition des éléments constituant la machine RC^2

Le réseau 2D fonctionne en mode multipipeline durant la phase de conversion (Figure 4.15) :

- les cellules RC sont connectées en lignes *horizontales*,
- le pipeline *vertical* DSH (Distribution des Segments Horizontaux) alimente le réseau,
- l'unité PDP (Préparation et Découpage des Polygones) est connectée au point d'entrée de DSH, et effectue les derniers calculs de préparations (par exemple, la transformation des polygones en triangles à base horizontale).

Il faut bien entendu veiller à un équilibrage correct des processus exécutés sur chacune de ces parties de la machine. En particulier, le débit de sortie de PDP doit être équivalent au débit d'entrée de DSH. Cela peut impliquer une parallélisation de cette unité. Nous supposerons pour l'instant que PDP est constitué d'un processeur général unique de puissance suffisante.

Les cellules DSH sont une version minimisée des cellules RC, en effet elles implémentent des calculs identiques, mais ne possède pas de mémoire de voxel.

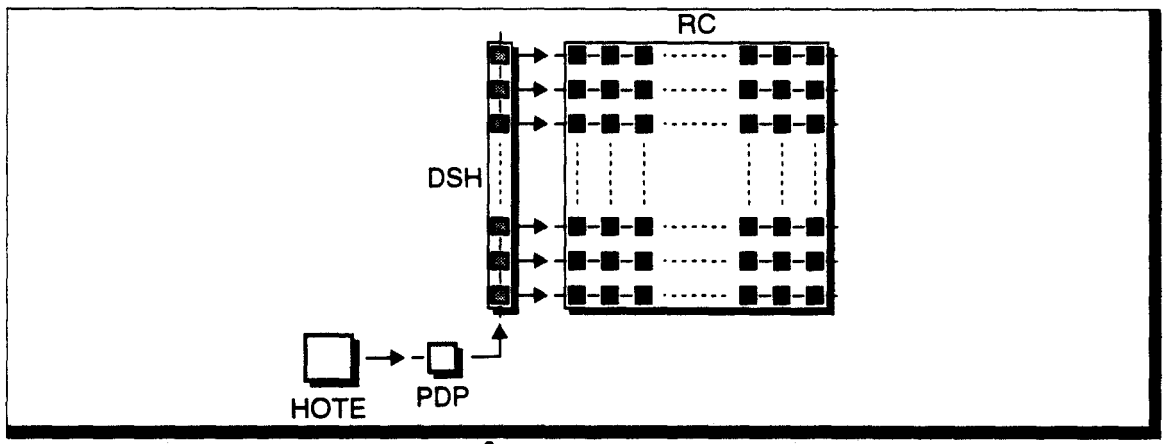


Figure 4.15 ***Organisation générale de RC^2, durant la phase de voxelisation.***

Cette architecture et l'algorithme de voxelisation de facettes que nous présentons ont fait l'objet d'une simulation fonctionnelle sur une machine Parsytec dotée de 32 transputers T800, disposant chacun de 1Mo de mémoire. Cette faible capacité mémoire n'a permis que la simulation d'un réseau 128x128. Par manque de mémoire, nous n'avons pu implémenter qu'une méthode d'éclairement simple (Gouraud).

4.4.2 Principe de fonctionnement

Pour simplifier la cellule de base, l'objet géométrique que nous avons choisi est la facette triangulaire à base horizontale, c'est-à-dire dont l'un des côtés est parallèle à l'axe *Y*. Ce choix permet de définir une facette à l'aide de deux segments de droites, au lieu de trois. Cette

restriction à pour unique but de minimiser le coût matériel de conversion, et les algorithmes que nous allons mettre en place pourraient sans aucun problème être étendus.

Il existe deux types de facettes dans cette catégorie, suivant la position de l'arête horizontale, que nous appellerons FHH (*facette horizontale haute*, Figure 4.16.a) et FHB (*facette horizontale basse*, Figure 4.16.b).

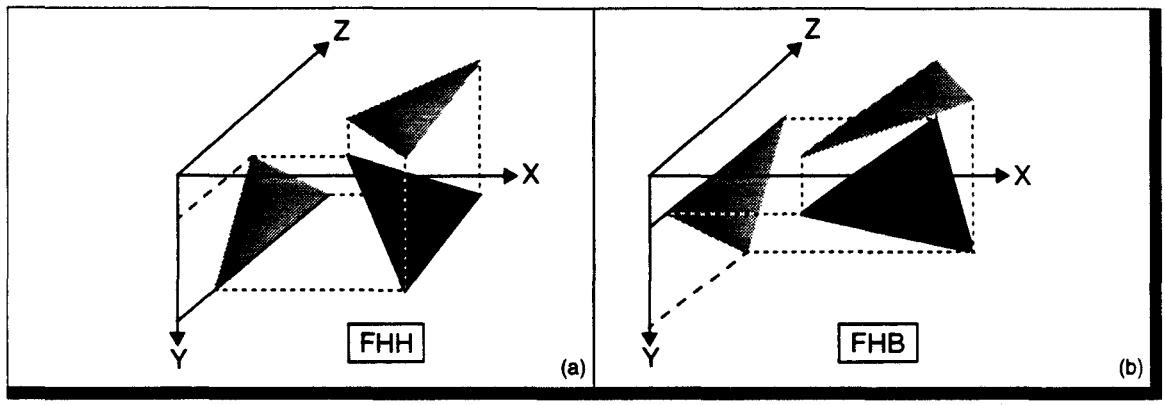


Figure 4.16 ***Deux types de facettes à base horizontale.***

Comme précédemment évoqué le balayage plan unique consiste à évaluer les attributs de chaque voxel par calcul incrémental dans le plan *XY*. L'utilisation de la méthode de voxelisation par construction permet le calcul direct des profondeurs avant et arrière de chaque palier composant la facette. Ces paliers sont ensuite remplis afin d'affecter à chaque voxel les valeurs de couleurs (pour une interpolation de Gouraud), ou de normales (pour une interpolation de Phong ou un Lancer de Rayon Discret).

La méthode proposée par Vidal [Vida92] consiste à voxeliser le plan support puis à découper celui-ci à l'aide de plans perpendiculaires à la facette et passant par les arêtes (Figure 4.17.a). Ces plans permettent de calculer les profondeurs limitant les paliers du plan support. Il existe cependant certaines orientations de plan pour lesquelles la profondeur n'est pas calculable (plans perpendiculaires au plan *XY*). Dans ce cas particulier, le plan de découpe supprime l'existence de la facette au sein de certaines cellules, plutôt que de limiter une profondeur.

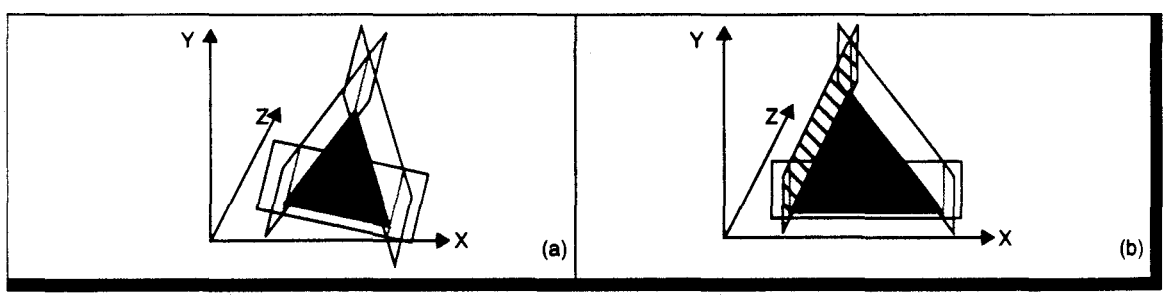


Figure 4.17 ***Définition d'une facette par découpage.*** *(a) - Chaque plan de découpage limite la profondeur (b) - Cas particulier, le plan hachuré ne permet pas de calcul de profondeur.*

Toujours par souci de simplification des algorithmes, nous avons préféré décomposer la définition des bords de facettes en 2 étapes : on définit tout d'abord le contour de la projection de la facette dans le plan *XY* (ce qui définit les cellules où la facette est présente), ensuite on effectue la limitation de profondeur.

La méthode permettant de déterminer l'intérieur de la facette 2D correspondant à cette projection est celle classiquement utilisée en parallélisme pixel massif :

- chaque arête du contour est définie par une expression linéaire (l'équation de la droite contenant l'arête); cette expression permet de séparer l'espace 2D en deux sous-espaces : l'ensemble des points pour lesquels l'expression est négative, et l'ensemble des points pour lesquels elle est positive.

- les arêtes sont orientées de telle sorte que pour un point à l'intérieur de la facette, toutes les expressions du contour prennent une valeur négative.

Il suffit alors pour une cellule du réseau de calculer les expressions de contour pour les coordonnées (x, y) qu'elle occupe afin de déterminer si la facette est présente ou non à cette position (Figure 4.18.a). Pour les facettes à base horizontale, on peut n'utiliser que les expressions de deux arêtes, si les cellules effectuant le test sont limitées à celles comprises dans le cadre englobant de la facette.

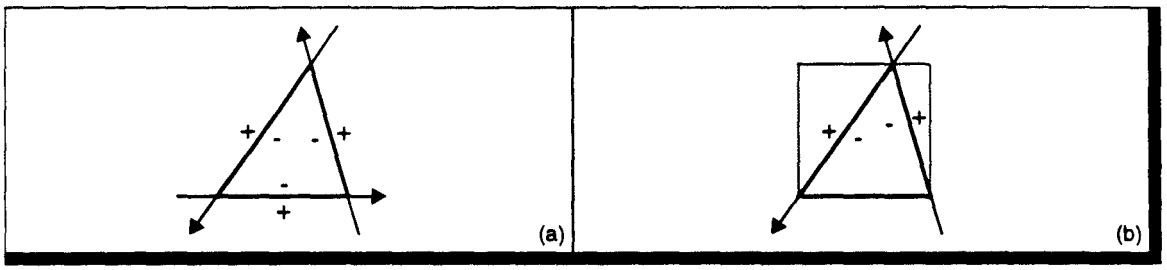


Figure 4.18 ***Définition du contour par équation.*** *(a) - Cas général, une équation par arête. (b) - En utilisant le cadre englobant, deux arêtes suffisent pour les facettes à base horizontale.*

L'algorithme de voxelisation de facettes se divise donc en 3 phases :

- Détermination du contour,
- Si intérieur à la facette, calcul des profondeurs avant et arrière, avec limitation
- Pour chaque voxel des paliers, calcul des attributs géométriques et/ou photométriques.

Chacune de ces phases utilise massivement le calcul incrémental d'expressions linéaires, ce qui permet une parallélisation efficace.

4.4.3 Parallélisation de l'algorithme

Nous présentons les différentes phases permettant la conversion de facettes, avec pour chacune d'elles la parallélisation sur la machine RC^2. Nous utilisons deux fonctions de transferts de données entre les différentes unités :

- Envoi_DSH permet le transfert dans une cellule DSH,
- Envoi_RC effectue un transfert dans une cellule RC.

Orientation et cadre englobant

L'orientation des facettes consiste à ordonner le parcours des sommets afin que les expressions définissant les arêtes soient négatives pour les points intérieurs. Pour les triangles à base horizontale, on obtient deux orientations possibles (Figure 4.19).

Le cadre est défini par : $xm = min(x1, x2)$, $xM = max(x1, x2, x3)$
$ym = y1$, $yM = y2$

A partir de ces coordonnées, une première exclusion de cellules du réseau est effectuée. Seules les cellules situées à l'intérieur du cadre exécuteront réellement les phases suivantes de l'algorithme. Cela n'a aucun effet sur le débit possible, puisque comme évoqué précédemment les performances globales de conversion du réseau sont tributaires de la cellule la plus lente. Le cadre englobant permet uniquement de n'utiliser que deux arêtes pour définir le contour.

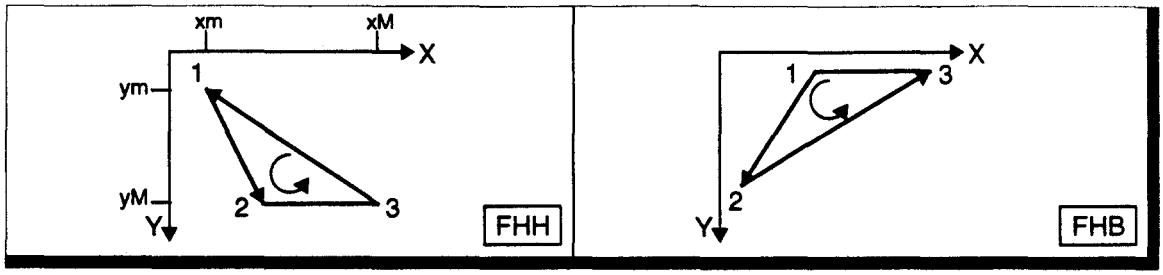


Figure 4.19 ***Orientation des deux types de triangles à base horizontale.***

La répartition sur la machine est la suivante :

PDP

```
Orientation facette
Calcul xm, xM, ym, yM
Envoi_DSH( xm, xM, ym, yM )
```

DSH(j)

```
Si ym <= j <= yM
| Dans_Cadre <- .vrai.
| Envoi_RC( xm, xM )
Sinon
| Dans_Cadre <- .faux.
FinSi
Envoi_DSH( xm, xM, ym, yM )
```

RC(i)

```
Si xm <= i <= xM
| Dans_Cadre <- .vrai.
Sinon
| Dans_Cadre <- .faux.
FinSi
Envoi_RC( xm, xM )
```

Calcul du contour

Les deux arêtes non horizontales sont utilisées pour restreindre les cellules actives à celles comprises à l'intérieur du contour 2D de la facette. Le traitement effectué est identique pour ces deux arêtes. Prenons en exemple l'arête 1-2 de la Figure 4.19.

La droite Δ_{12} est définie par l'équation : $\frac{y-y_1}{y_2-y_1}-\frac{x-x_1}{x_2-x_1}=0$, qui peut être exprimée sous la forme : $\Delta x_{12}\times(y-y_1)-\Delta y_{12}\times(x-x_1)=0$.

On appelle erreur[1], la valeur $E(i,j)=\Delta x_{12}\times(j-y_1)-\Delta y_{12}\times(i-x_1)$, c'est-à-dire la valeur de l'expression définissant le segment en tout point (i,j) . Cette valeur représente l'écart entre un point de l'espace et la droite Δ_{12}.

Cette erreur peut-être calculée de manière incrémentale :

on pose $E(i,j)=\Delta E_x\times(i-x_1)+\Delta E_y\times(j-y_1)$.

on a alors $E(i+1,j)=E(i,j)+\Delta E_x$, et $E(i,j+1)=E(i,j)+\Delta E_y$.

1. Au sens de l'erreur dans l'algorithme de tracé de droite de Bresenham.

L'évaluation de l'erreur ne sera effectuée que par les cellules appartenant au cadre englobant de la facette. L'erreur initiale au coin du cadre est $E_{init} = E(x_m, y_m) = \Delta E_x \times (x_m - x_1)$.

La droite Δ_{12} définit donc deux demi-plans signés, un point (i,j) appartenant à la facette s'il se trouve dans le demi-plan négatif. Cependant, afin d'obtenir sur le contour une droite équivalente à une droite de Bresenham, le point (i,j) est considéré à l'intérieur s'il est situé à moins d'un demi-pixel de la droite réelle. Cela se traduit par l'utilisation d'un seuil sur la valeur de l'erreur :

un point (i,j) appartient à la facette si $E(i,j) \leq \frac{1}{2} max(|\Delta E_x|, |\Delta E_y|) = S$

Nous obtenons l'implémentation parallèle suivante :

PDP

```
Pour les deux arêtes de contour
|  Calcul ΔEx, ΔEy, E = Einit, S
|  Envoi_DSH( ΔEx, ΔEy, E, S )
FinPour
```

DSH(j)

```
Si Dans_Cadre
|  Envoi_RC( ΔEx, E, S )
|  E <- E+ΔEy
FinSi
Envoi_DSH( ΔEx, ΔEy, E, S )
```

RC(i)

```
(Init: Dans_Segment <- .vrai.)

Si Dans_Cadre
|  Si E>S
|  |  Dans_Segment <- .faux.
|  FinSi
|  E <- E+ΔEx
FinSi
Envoi_RC( ΔEx, E, S )
```

Calcul des profondeurs

La méthode utilisée ici définit un palier (i.e une profondeur avant et une profondeur arrière) en chaque point (i,j) situé à l'intérieur du contour 2D.

On utilise tout d'abord l'équation du plan support de la facette pour calculer la profondeur vraie en chaque point (i,j) . On appelle $T(i,j)$ cette valeur. Puis en fonction des coefficients du plan, Vidal définit la profondeur K des paliers.

On obtient alors $Z_{AV}(i,j) = T(i,j) - K/2$ et $Z_{AR}(i,j) = T(i,j) + K/2$.

Prenons l'exemple de la facette FHH de la Figure 4.19.

L'équation du plan est : $a \times (x - x_1) + b \times (y - y_1) + c \times (z - z_1) = 0$, avec a, b, c les composantes du vecteur normal à la facette, soit après simplification :

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \vec{V}_{12} \wedge \vec{V}_{13} = \begin{bmatrix} \Delta y_{12} \times \Delta z_{23} \\ (\Delta z_{12} \times \Delta x_{13}) - (\Delta z_{13} \times \Delta x_{12}) \\ -(\Delta x_{23} \times \Delta y_{12}) \end{bmatrix}$$

Deux cas se présentent suivant l'orientation de la facette :

- Facette quelconque

  L'équation du plan peut se réécrire : $z = \Delta z_x \times (x - x_1) + \Delta z_y \times (y - y_1) + z_1$, avec $\Delta z_x = -(a/c)$ et $\Delta z_y = -(b/c)$.

  On pose alors : $T(i,j) = \Delta z_x \times (i - x_1) + \Delta z_y \times (j - y_1) + z_1$, la valeur de profondeur en (i,j). Cette expression est calculable en incrémental, de la même manière que nous l'avons explicité dans le cas de la droite de contour.

  La longueur de palier est définie par : $K = max(|\Delta z_x|, |\Delta z_y|, 1)$ [Vida92].

- Facette *vue de face* (i.e $c = 0$, la facette est *parallèle* à l'axe Z)

  Dans ce cas, on ne peut définir une valeur de profondeur unique en un point (i,j). Pour ne pas avoir à traiter ce cas particulier, on prend $Z_{AV}(i,j) = 0$, $Z_{AR}(i,j) = N_V$, $\Delta z_x = 0$, $\Delta z_y = 0$.

L'implémentation parallèle de cette évaluation est la suivante :

PDP
```
Calcul Δz_x, Δz_y, Z_AV = T_init - K/2 , Z_AR = T_init + K/2
Envoi_DSH( Δz_x, Δz_y, Z_AV, Z_AR )
```

DSH(j)
```
Si Dans_Cadre
| Envoi_RC( Δz_x, Z_AV, Z_AR )
| Z_AV <- Z_AV+Δz_y
| Z_AR <- Z_AR+Δz_y
FinSi
Envoi_DSH( Δz_x, Δz_y, Z_AV, Z_AR )
```

RC(i)
```
Si Dans_Cadre
| Mémoriser Z_AV Z_Deb = Z_AV et Z_AR
| Z_AV <- Z_AV+Δz_x
| Z_AR <- Z_AR+Δz_x
FinSi
Envoi_RC( Δz_x, Z_AV, Z_AR )
```

Limitation de profondeur

Du fait de la technique de construction, les facettes présentent pour l'instant un remplissage en profondeur trop important, qu'il faut limiter sur les bords du triangle (Figure 4.20.a).

La limitation par un bord, dont nous avons explicité le principe ci-dessus, utilise un plan perpendiculaire à la facette et incluant l'arête. La comparaison en un point (i,j) entre la profondeur actuellement calculée et celle de ce plan permet la limitation (Figure 4.20.b).

Il faut cependant remarquer que cette méthode utilisée telle quelle peut être trop *restrictive*, comme dans le cas des facettes minces où il y a suppression de voxels devant appartenir à la facette (Figure 4.20.c). La solution bien connue[1] est de *décaler* les plans de limitation d'un demi-pixel vers l'extérieur de la facette (Figure 4.20.d)

1. Equivalente au choix du seuil de 1/2 pour les arêtes de contour, et donc équivalente à l'erreur initiale de 1/2 utilisée dans l'algorithme de tracé de droite de Bresenham.

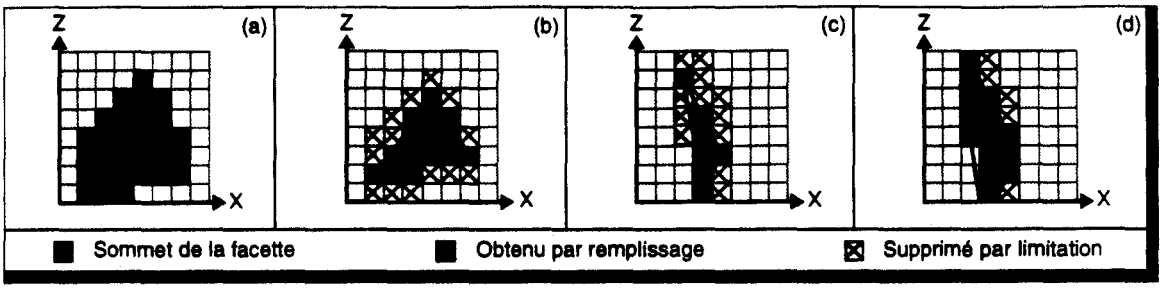


Figure 4.20 ***Processus de limitation en profondeur.*** *(a) - Sans limitation. (b) - Limitation par les bords. (c) - Problème apparaissant sur les facettes minces. (d) - Limitation autour des bords.*

Pour simplifier les évaluations, on utilisera plutôt l'équation d'une droite, au lieu d'un plan, pour obtenir la profondeur limitante. Il est alors nécessaire de travailler sur un des plans principaux pour définir cette droite (de la même façon que pour les droites de contour). Il nous faut avoir la plus grande précision possible pour le calcul de profondeur limitante, ce qui nous fait choisir le plan de plus grande pente pour définir les droites de limitation.

Le plan de plus grande pente est choisi à partir de la valeur des composantes de la normale à la facette :

$$\vec{n} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

si $max(a, b, c) = a$ -> projection dans le plan YZ

si $max(a, b, c) = b$ -> projection dans le plan XZ

si $max(a, b, c) = c$ -> projection dans le plan XY, (effectué par calcul du contour)

Il ne reste donc que les deux premiers cas à traiter. Prenons l'exemple d'une projection dans le plan XZ, et de la limitation par la projection de l'arête 1-2. Il existe 4 possibilités d'orientation d'une arête, suivant lesquelles on effectue une limitation de la profondeur avant ou arrière des paliers (Figure 4.21).

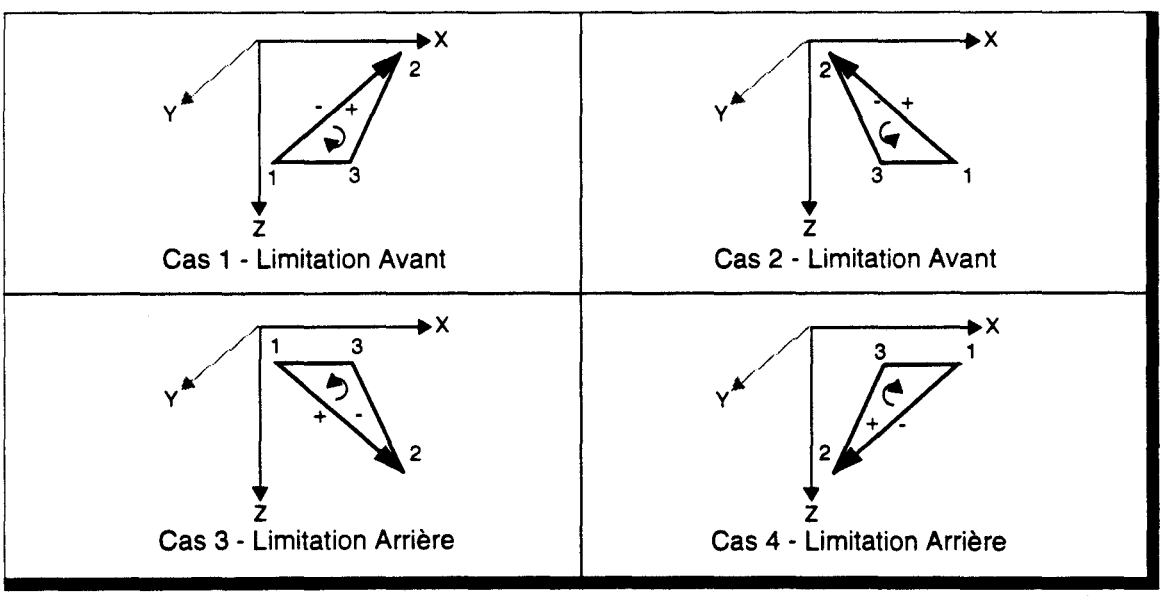


Figure 4.21 ***Quatre orientations possibles pour une arête de limitation.***

L'équation de la droite Δ_2 est : $-\Delta z_{12} \times (x - x_1) + \Delta x_{12} \times (z - z_1) = 0$.

L'écart entre un point (i, z) et la droite est : $E(i, z) = -\Delta z_{12} \times (i - x_1) + \Delta x_{12} \times (z - z_1)$.

Le seuil d'appartenance à l'intérieur de la droite est $S = 1/2 \times max(|\Delta z_{12}|, |\Delta x_{12}|)$.

Pour une position i donnée, la valeur de profondeur limitante z_i est donnée par $E(i, z_i) = \pm S$. Il faut tenir compte de l'orientation de Δ_{12} pour définir si la limitation est effectuée du côté positif ou négatif de la droite. La différence entre les deux cas peut se faire par connaissance de la composante n_y de la normale :

si $n_y < 0$, on prend $E(i, z_i) = -S$ (cas 1 et 4 de la Figure 4.21),

si $n_y > 0$, on prend $E(i, z_i) = +S$ (cas 2 et 3 de la Figure 4.21)

On obtient donc : $z_i = \frac{\pm S}{\Delta x_{12}} + \frac{\Delta z_{12}}{\Delta x_{12}} \times (i - x_1) + z_1 = \Delta z_x \times (i - x_1) + z_0$, qui peut être mis sous forme incrémentale.

Pour avoir une méthode homogène quelque soit le plan de projection utilisé, on définira : $z = \Delta z_x \times (i - x_1) + \Delta z_y \times (j - y_1) + z_0$, l'un des deux coefficients d'incrémentation étant nul.

Il reste à définir si la limitation est avant ou arrière. En observant la Figure 4.21, on peut définir le critère suivant : Limitation_Avant = $(n_y > 0)\ XOR\ (\Delta x_{12} > 0)$.

L'exécution du calcul de limitation est implémentée comme suit :

PDP
```
Calcul des composantes n_x et n_y de la normale
Choix du plan de projection
Pour chaque arête
|  Calcul Δz_x, Δz_y, z = z_init, Type_Limitation
|  Envoi_DSH( Δz_x, Δz_y, z, Type_Limitation )
FinPour
```

DSH(j)
```
Si Dans_Cadre
|  Envoi_RC( Δz_x, z, Type_Limitation )
|  z <- z+Δz_y
FinSi
Envoi_DSH( Δz_x, Δz_y, z, Type_Limitation )
```

RC(i)
```
Si Dans_Cadre
|  Si ( Type_Limitation = Limitation_Avant )
|  |  z_AV = max(z, z_AV)
|  Sinon
|  |  z_AR = min(z, z_AR)
|  FinSi
|  z <- z+Δz_x
FinSi
Envoi_RC( Δz_x, Δz_y, z, Limitation_Avant )
```

Interpolation de valeur (couleur ou normale)

Qu'il s'agisse d'interpoler une valeur de couleur, ou une valeur de normale, l'algorithme est le même. Cependant dans le deuxième cas, il est nécessaire de normaliser la valeur calculée, ce qui comme nous le verros ultérieurement est effectué en dehors de la phase de voxelisation (cf. § 6.3 - p. 147). Nous ne l'aborderons donc pas ici.

La valeur à interpoler est définie par équation dans le plan de plus grande pente, sous la forme :
$a \cdot u + b \cdot v + c \cdot L + d = 0$, avec u, v les coordonnées du plan de projection et L la valeur à interpoler.

Les coefficients de cette équation sont utilisés pour réécrire l'expression de L sous forme incrémentale : $L = \Delta L_x \times (x - x_1) + \Delta L_y \times (y - y_1) + \Delta L_z \times (z - z_1) + L_1$, où un des coefficients d'incréments est à priori nul. Suivant le plan de projection, on a :

- plan XY -> $\Delta L_x = -a/c$, $\Delta L_y = -b/c$ et $\Delta L_z = 0$
- plan XZ -> $\Delta L_x = -a/c$, $\Delta L_y = 0$ et $\Delta L_z = -b/c$
- plan YZ -> $\Delta L_x = 0$, $\Delta L_y = -a/c$ et $\Delta L_z = -b/c$

Si cette expression est utilisée telle quelle, alors pour chaque incrément en x ou y, il est nécessaire de reprendre le calcul de L en $z = 0$, ou en toute autre valeur constante. Un balayage complet en profondeur est donc nécessaire, alors que seul un petit nombre de voxels peuvent être occupés au sein d'un dexel. Pour minimiser le coût de l'interpolation en z, nous débutons donc le calcul sur chaque début de palier, ce qui implique de prendre en compte l'incrément de profondeur pour définir l'incrément d'interpolation de L. On obtient dans ce cas :

- plan XY -> $\Delta L_x = -a/c$, $\Delta L_y = -b/c$ et $\Delta L_z = 0$
- plan XZ -> $\Delta L_x = -(a/c) + \Delta z_x \times \Delta L_z$, $\Delta L_y = \Delta z_y \times \Delta L_z$ et $\Delta L_z = -b/c$
- plan YZ -> $\Delta L_x = \Delta z_x \times \Delta L_z$, $\Delta L_y = -(a/c) + \Delta z_y \times \Delta L_z$ et $\Delta L_z = -b/c$

D'autre part, un palier s'étend de part et d'autre de la profondeur réelle en un point donné. La valeur initiale de L pour le calcul incrémental doit donc également être calculée sur un début de palier, d'où : $L_{init} = \Delta L_x \times (x_m - x_1) + \Delta L_y \times (y_m - y_1) + L_1 - K \times \Delta L_z$.

L'implémentation parallèle que nous obtenons est la suivante :

PDP

```
Choix du plan de projection
Pour chaque valeur à interpoler
| Calcul ΔLx, ΔLy, ΔLz, L = Linit
| Envoi_DSH( ΔLx, ΔLy, ΔLz, L )
FinPour
```

DSH(j)

```
Si Dans_Cadre
| Envoi_RC( ΔLx, ΔLy, ΔLz, L )
| L <- L+ΔLy
FinSi
Envoi_DSH( ΔLx, ΔLy, ΔLz, L )
```

RC(i)

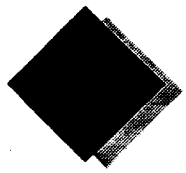
```
Si Dans_Cadre
| Mémorisation L, ΔLz
| L <- L+ΔLx
FinSi
Envoi_RC( ΔLx, ΔLy, ΔLz, L )
```

Remplissage des paliers

Cette dernière étape permet d'attribuer à chaque voxel occupé les différentes valeurs (couleurs, normale) qui définissent l'objet en ce point.

Le calcul à effectuer est l'interpolation en z, le long des paliers, des valeurs que l'étape précédente a préchargées. Seules les cellules du réseau sont actives durant cette étape :

CHAPITRE



Processus de Rendu

Rappelons que nous entendons par *rendu*, l'ensemble des algorithmes permettant de donner un aspect réaliste à une image de synthèse, l'effet le plus simple, par exemple, étant le *coloriage*, ou *ombrage*, des objets visualisés. La méthode de rendu la plus complexe que nous désirons mettre en oeuvre dans le cadre de ce travail est le Lancer de Rayon.

Nous présenterons dans ce chapitre différentes méthodes de calcul d'ombrage dans l'espace voxel, utilisées classiquement en imagerie médicale, mais insuffisante pour la synthèse d'image.

Nous définirons ensuite le fonctionnement général du réseau durant la phase de rendu.

Nous tenons à indiquer que cette phase de traitement, et en particulier le Lancer de Rayon Discret en lui-même, n'a pas encore été approfondie. Avant de pouvoir définir une implémentation parallèle complète, il nous faut disposer d'un algorithme séquentiel du Lancer de Rayon Discret, ce qui sort des travaux de cette thèse.

Avant propos

Si le processus de voxelisation a pu être simulé sur la machine parallèle à base de Transputers dont nous disposons, il n'en est pas de même pour le processus de rendu. Cette impossibilité est essentiellement dûe au manque de mémoire au sein de chaque noeud Transputer.

Or, étant donné la complexité de l'algorithme du Lancer de Rayon, dont le fonctionnement est en grande partie dépendant de la scène à visualiser, seule une réelle simulation peut permettre de valider les choix d'architecture, particulièrement dans le cas du Lancer de Rayon Discret, où les communications jouent un rôle important dans l'efficacité de l'implémentation parallèle.

Parmi les logiciels de simulation actuels, nous n'avons pu en trouver permettant la modélisation d'un très grand nombre de circuits, le fonctionnement interne de ces logiciels restreignant le nombre de variables utilisables. Devant cet état de fait, nous avons débuté l'écriture de notre propre simulateur fonctionnel (SITOR II [Gbad93]), qui répond à nos exigences de taille de circuit et de rapidité de simulation. Celui-ci venant d'être opérationnel, nous avons pas encore pu l'utiliser.

Nous ne donnerons par conséquent ici que des évaluations générales d'implémentation.

5.1 Rendu *purement* volumique

Nous entendons par rendu *purement* volumique l'application de modèles d'éclairement sur des objets voxelisés ne contenant aucune information géométrique, ce qui est le cas par exemple des cubes voxels fournis par les appareils médicaux. Or les modèles d'éclairement utilisés en synthèse d'image (cf § 1.1 - p. 7) nécessitent toujours la connaissance de la normale à la surface éclairée.

Un certain nombre de méthodes ont été utilisées, soit pour définir de nouveaux modèles n'utilisant aucune information géométrique, soit pour pallier ce manque en recréant les informations manquantes.

5.1.1 Eclairement sans informations géométriques

Ces méthodes utilisent uniquement les seules données existantes en tout voxel, i.e. leur position.

On trouve dans cette catégorie, par exemple, la méthode d'*ombrage par profondeur* (depth only shading) [Gold84] qui affecte à chaque pixel une intensité fonction de la profondeur du voxel affiché en ce point. L'éclairement n'est donc possible que par projection du cube. Il produit un lissage important puisque les voxels de profondeur proche prendront une intensité voisine.

Une autre méthode éclaire directement les voxels en les considérant comme des cubes de taille unitaire [Herm79]. On obtient cette fois des effets indésirables d'arêtes artificielles. Par contre cette méthode est indépendante de la projection.

Il est évident qu'il n'est pas possible d'obtenir un éclairement correct sans connaissance aucune d'informations géométriques plus précises, entre autre le vecteur normal à la surface.

5.1.2 Eclairement par reconstruction de la géométrie

Il existe un grand nombre de techniques de ce type. Elles ont presque toutes en commun une méthode d'approximation de la normale par considération de la position des voxels voisins au voxel considéré. On peut en effet approximer une surface en considérant l'enveloppe des voxels qui la constitue.

Nous détaillerons les méthodes utilisées sur les deux Machines Voxels que nous avons déjà présentées : Cube et Voxel Processor.

Méthode de rendu de la machine Cube

Elle est basée sur le calcul d'un gradient de profondeur, d'où son appellation : *congradient shading* [Cohe90].

Calcul de la normale

La seule information accessible est la profondeur du voxel visible en un pixel. La normale en ce point peut être approximée par un gradient de profondeur :

$$\vec{N} = \left[\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}, -1\right]$$

Le calcul exact des dérivées est impossible sans la connaissance de l'équation de la surface au point considéré. Dans l'espace discrètisé, il faut donc se contenter d'estimations calculées par différence centrale, en utilisant la profondeur des voxels du voisinage.

Notons $z_{i,j}$ la profondeur du voxel visible au pixel (i,j). On prendra $\frac{\partial z}{\partial x} \approx \frac{\Delta z}{\Delta x} = \frac{z_{i+L,j} - z_{i-L,j}}{2 \cdot L}$, de même pour la composante en y.

L représente la demi-largeur du voisinage utilisé pour le calcul de moyenne. Les études effectuées par Cohen montrent qu'une valeur de 1 est suffisante.

Calcul de l'éclairement

Le souci d'affichage en temps réel (soit la production de ~6Mpixels/s pour 25 images/s), et de minimalisation du matériel mis en oeuvre, interdit le calcul vrai d'une équation d'éclairement, telle que celle de Phong. La solution retenue pour la machine Cube est de ne conserver qu'un éventail restreint de valeurs de normale, pour lesquelles l'éclairement est précalculé.

Une fonction de seuillage f_1 est appliquée, générant une normale clippée :

$$\overline{N}_i = f_1(N_i) = \begin{cases} N_i & \text{si} \quad -H/2 \le N_i \le H/2 \\ H/2 & \text{si} \quad N_i > H/2 \\ -H/2 & \text{si} \quad N_i < -H/2 \end{cases}$$

Cette fonction est utilisée sur les composantes en x et y. Des études sur la largeur H du seuil indiquent qu'une valeur de 3 permet un échantillonnage correct. Les normales clippées, concaténées avec les valeurs de couleur des voxels sont mémorisées dans la mémoire de trame 2D. Lors de l'affichage, elles servent d'index sur une table contenant les valeurs d'éclairement.

Les pertes de qualités dues au clipping sont flagrantes sur un objet synthétisé (Figure 5.1), mais le rendu est correct en imagerie médicale, car il permet d'observer les détails de relief de la structure.

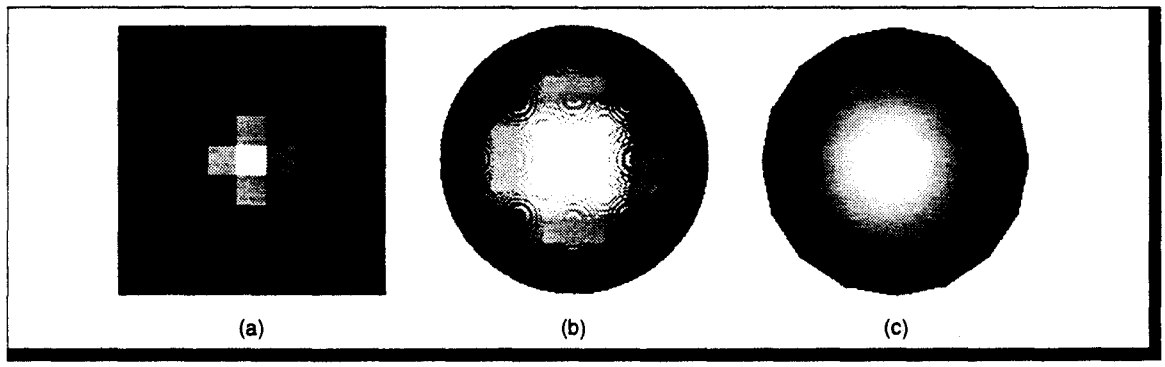


Figure 5.1 ***Congradient Shading.*** *(a)- Table d'éclairement utilisée. (b)- Eclairement obtenue par cette méthode sur une sphère. (c)- Eclairement de Phong sur une sphère facettisée.*

Méthode de rendu de la machine Voxel Processor

La méthode retenue pour la machine Voxel Processor est l'utilisation du gradient de profondeur [Gord85]. Mais, contrairement à la machine Cube, la normale n'est pas seuillée. Lorsque d'un pixel au suivant il y a changement de l'objet visible, il peut se produire une forte discontinuité de valeur de profondeur. Pour éviter ce phénomène, Gordon et Reynolds proposent d'utiliser une fonction pondérée des différences gauche et droite (haute et basse) de profondeur :

Soit $z_{i,j}$, la profondeur du voxel visible au pixel (i,j).

On pose $\delta_b = z_{i,j} - z_{i-1,j}$ et $\delta_f = z_{i+1,j} - z_{i,j}$. La valeur du gradient en x est définie par la moyenne pondérée de ces deux différences :

$$\frac{\partial z}{\partial x} \approx \frac{W_b \cdot \delta_b + W_f \cdot \delta_f}{W_b + W_f} \qquad \textbf{(Eq. 5.1)}$$

La fonction de pondération choisie est :

$$W(\delta) = \begin{cases} 1 & \text{si} \quad \delta \le a \\ \varepsilon & \text{si} \quad \delta \ge b \\ \frac{1+\varepsilon}{2} + \frac{1-\varepsilon}{2} \cdot \cos\left(\frac{\delta - a}{b - a} \cdot \pi\right) & \text{sinon} \end{cases}$$

Cette fonction accorde un poids important à une faible différence ($\delta \leq a$). Si la différence est élevée ($\delta \geq b$), le poids est choisi faible ($W = \varepsilon$) mais non nul, pour que la division utilisée dans le calcul de la normale (Eq. 5.1) puisse être exécutée. Entre ces deux limites, la fonction cosinus effectue un lissage des valeurs (Figure 5.2.a).

Cette fonction est utilisée pour calculer les deux pondérations $W_b = W(|\delta_b|)$ et $W_f = W(|\delta_f|)$.

Les études de Gordon et Reynolds montrent que des valeurs $a = 2$, $b = 5$ et $\varepsilon = 10^{-5}$ produisent des images de bonnes qualités (Figure 5.2.b). Le nombre des valeurs δ_b et δ_f possibles est limité, ce qui permet d'implémenter la fonction de pondération dans une table.

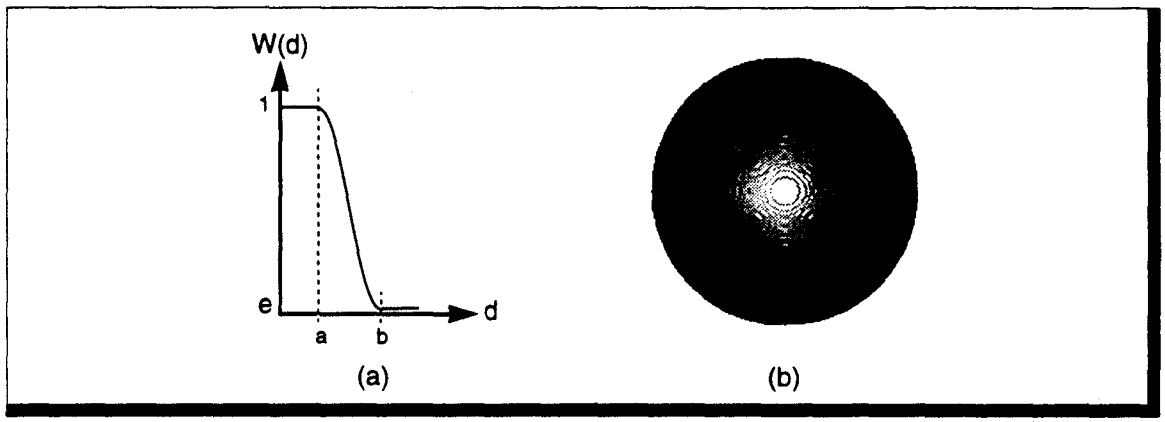


Figure 5.2 ***Gradient Shading.*** *(a) - Fonction de pondération. (b) - Rendu sur une sphère.*

La complexité du calcul de la moyenne pondérée permet difficilement un rendu temps réel. Cependant, des versions *dégradées* de cette méthode peuvent être implémentées pour effectuer un post-éclairement *au vol*.

5.1.3 En résumé

Les algorithmes d'éclairement utilisés en visualisation volumique médicale ne sont pas satisfaisant en synthèse d'images de par le manque de réalisme qu'ils obtiennent. Ceci est en grande partie dû à l'absence d'information géométrique concernant la surface stockée en chaque voxel. En visualisation médicale il est important de pouvoir observer les détails de surface et les méthodes les renforçant légèrement sont donc mieux adaptées. Nous situant dans le domaine de la synthèse d'image, il nous faut au contraire avoir un lissage correct des intensités lumineuses. Les informations géométriques sont dans notre cas calculables, et nous pouvons doter chaque voxel des données nécessaires à un rendu réaliste. Ils nous faut donc revenir sur les modèles d'éclairement classiquement utilisés en synthèse, et les adapter à l'espace discret 3D.

5.2 Organisation architecturale générale

Une des phases du processus de rendu est la projection du cube voxel sur le plan écran. Dans le cas des algorithmes classiques manipulant des objets géométriques, cette projection est précalculée : on effectue les transformations géométriques adéquates afin de placer les objets dans le *repère écran* à 2 dimensions, les objets étant transformés en pixels après cette opération. Dans le cadre du rendu volumique, ce traitement n'est pas possible, et il est nécessaire de procéder à la recherche du (ou des) voxel(s) visible(s) à *travers* chaque pixel-écran. Si l'on utilise une projection perspective, alors cette détermination passe nécessairement par un Lancer de Rayon à un seul niveau (*Ray Casting*), i.e. sans traitement des réflexions et réfractions.

Nous allons donc directement étudier le Lancer de Rayon Discret. Les implémentations pour des modèles d'éclairement plus simple (diffus, spéculaire) pouvant en être facilement déduites.

5.2.1 Définition et répartition des traitements

Rappelons le principe du Lancer de Rayon Discret :

```
Pour chaque pixel écran
|    Couleur_Pixel <- TRACER (rayon primaire)
FPour

Fonction TRACER (rayon)
Debut
|    Suivi du rayon jusqu'à voxel occupé
|    Si (non_intersection OU profondeur_arbre_maxi) alors
|    |    Couleur <- Couleur_Fond
|    Sinon
|    |    Couleur <- Couleur_locale
|    |
|    |    Si (réflexion) alors
|    |    |    Couleur <- Couleur + TRACER (rayon réfléchi)
|    |    Fsi
|    |
|    |    Si (transmission) alors
|    |    |    Couleur <- Couleur + TRACER (rayon transmis)
|    |    Fsi
|    Fsi
|    Retourner (Couleur)
Fin
```

L'objectif de l'étude présente est de juger de la mise en oeuvre d'un Lancer de Rayon *purement* discret, c'est-à-dire que chaque voxel contient toutes les informations nécessaires à l'exécution de l'algorithme (en particulier, les informations géométriques tel le vecteur normal, et les informations photométriques, telle la couleur de base de l'objet). Nous nous placerons donc dans cette hypothèse. Les différentes méthodes à définir et à mettre en oeuvre pour effectuer un Lancer de Rayon Discret sont :

1. Calcul de l'éclairement local,
2. Suivi du rayon dans l'espace voxel et intersection,
3. Détermination des rayons primaires à lancer

L'équation d'éclairement de Whitted est classiquement utilisée par les algorithmes de Lancer de Rayon pour calculer l'intensité lumineuse en chaque point d'intersection entre un rayon et un objet (cf § 1.1.4 - p. 16) :

$$I_{tot} = I_{amb} + \sum_{i=sources} S_i \cdot \left(I_{diff_i} + I_{spec_i} \right) + I_{refl} + I_{trans}$$

Le flag S_i représente l'un des avantages du Lancer de Rayon par rapport aux méthodes de rendu par projection : la prise en compte des ombres portées est intégrée dans l'algorithme. En effet, lors du calcul de la couleur visible en un point d'intersection entre un rayon et un objet, on vérifie tout d'abord si ce point est éclairée (ce qui positionne S_i), c'est-à-dire s'il est atteint par les rayons provenant des sources lumineuses.

Dans le Lancer de Rayon classique, on procède par émission d'un rayon depuis le point d'intersection vers chacune des sources. S'il n'existe aucune intersection entre ce rayon et les objets de la scène, alors l'éclairement dû à la source lumineuse est ajouté à la valeur de l'intensité locale. Ces rayons appelés rayons d'ombrage sont donc émis vers les sources à chaque intersection entre un rayon primaire ou secondaire (cf § 1.1.5 - p. 19) et un objet. Cette méthode est bien entendu également applicable au Lancer de Rayon Discret. Cependant, il faut noter qu'un même voxel occupé peut être atteint par deux rayons primaires ou secondaires différents. Si on applique la technique de base du Lancer de Rayon, on relancera les rayons d'ombrage (identiques) lors des deux intersections. Or ces rayons sont indépendants de la direction du rayon incident, et nous avons ici la possibilité de mémoriser un indicateur en chacun des voxels afin de n'effectuer l'émission des rayons d'ombrage que lors de la première intersection.

De la même manière, une partie de l'intensité locale en chaque point d'un objet est indépendante de la direction des rayons incidents (intensité ambiante et diffuse). Là aussi, nous pourrions utiliser un indicateur pour éviter toute redondance de calcul.

Nous avons préféré mettre en place une autre méthode qui consiste à scinder la phase de rendu en deux traitements :

1. Un précalcul de l'éclairement local, qui permet d'utiliser une technique plus optimale de détermination des ombres portées [Vida92].
   L'étude de ce traitement fait l'objet du chapitre suivant.
2. Le Lancer de Rayon en lui-même, c'est-à-dire l'évaluation de la couleur à positionner en chaque pixel écran, que nous détaillons dans ce chapitre.

5.2.2 Parallélisme envisageable

Dans le mode de fonctionnement que nous avons choisi comme base du réseau voxel, la scène est distribuée sur l'ensemble des processeurs. Les rayons doivent donc être transmis de cellule en cellule. Le parallélisme utilisé est à flot de rayons.

Nous avons indiqué que dans ce mode, le temps d'exécution est fonction avant tout de la performance de transmission des rayons. Pour le parallélisme massif voxel, il est même complètement régi par la transmission puisque le temps de calcul d'intersection est pratiquement nul. Il nous faut donc avant tout veiller à obtenir une grande efficacité de suivi de rayon, plutôt qu'une performance élevée de calcul d'éclairement.

Le gain potentiel du Lancer de Rayon Discret sur notre réseau voxel, par rapport à une version séquentielle est fonction du nombre de rayons traités en parallèle. Nous ne nous avancerons pas sur une estimation théorique du gain, le fonctionnement du réseau pendant le Lancer de Rayon Discret étant trop complexe pour être défini facilement. Seule une simulation pourra nous permettre d'en évaluer les performances. Là encore, le logiciel de simulation que nous avons développé sera utilisé à cette fin, une fois l'algorithme de Lancer de Rayon Discret complètement défini.

Cependant, nous pouvons donner quelques indications qui permettront d'affiner le choix d'architecture.

Connexions entre cellules

Les rayons ont des directions de propagation quelconques, et avancent de cellule en cellule. Il est nécessaire de connecter entre elles les cellules voisines. Plusieurs topologies sont envisageables. Par exemple pour le réseau $2D$, on peut utiliser une topologie d'interconnexion en grille à 4 ou 6 voisins. Le choix à retenir est *a priori* fonction de l'algorithme utilisé pour effectuer le suivi de rayon. Cependant, plus la connectivité est importante, plus les circuits nécessitent de broches de connexion et plus la réalisation matérielle des cartes est difficile.

Comme il est possible d'émuler une connectivité donnée sur une connectivité plus basse, nous utiliserons une connectivité 4 pour un réseau $2D$, et 6 pour un réseau $3D$. Ce choix est bien sûr au détriment du temps de parcours d'un rayon, mais il est technologiquement bien plus raisonnable.

Contrôle SIMD / MIMD

Le mode de contrôle le plus simple pour une architecture massivement parallèle est le contrôle SIMD. Cela n'implique pas forcément également une simplicité de programmation. La majorité des algorithmes comportent des séquences d'instructions dont l'exécution est conditionnelle. La même instruction étant transmise à tous les processeurs, un dispositif matériel permet de désactiver ceux pour lesquels la condition n'est pas satisfaite.

Prenons l'exemple d'une structure *'SI condition ALORS séquence_1 SINON séquence_2'*, en supposant que la condition est vraie pour la moitié des processeurs, et que les deux séquences ont même durée d'exécution. Dans ce cas, 50% de la machine est activée pendant l'exécution de *séquence_1*, puis l'autre moitié durant l'exécution de *séquence_2*. Globalement, l'efficacité de la machine est donc de 0,5, ce qui se traduit par une diminution des performances attendues. Ceci est inhérent à l'approche SIMD, et il faut veiller à minimiser les séquences conditionnelles de ce type.

Or l'algorithme du Lancer de Rayon Discret est fortement conditionnel :

- une cellule peut recevoir plusieurs ou aucun rayon de ses voisines,
- ces rayons peuvent être de différents types (rayons primaires/secondaires, rayons de retour contenant une intensité),
- le niveau maximal de profondeur de l'arbre des rayons peut être ou non atteint,
- il peut exister intersection ou non,
- suivant les propriétés photométriques de la matière, on générera ou non un rayon réfléchi ou transmis...

Ce grand nombre d'états possibles nous amène à penser que l'efficacité interne de l'algorithme risque d'être très faible sur une machine SIMD. Très peu d'implémentations SIMD ont d'ailleurs été proposées, et les essais que nous avons pu mener sur une machine MassPar MPP1, dotée de 1024 processeurs, sont peu concluants [Berg93].

Le contrôle MIMD semble beaucoup plus naturel, afin que chaque cellule ne puisse exécuter que les traitements correspondants à son état courant (avec une machine MIMD et l'exemple de structure *SI ALORS SINON* que nous avons cité ci-dessus, l'efficacité est proche de 1, si les deux séquences ont même temps d'exécution). La programmation est plus aisée, mais le dispositif de contrôle est réparti ce qui induit un surcoût matériel pouvant être non négligeable.

Le noyau de l'algorithme de Lancer de Rayon Discret étant assez restreint, car ne nécessitant pas de calcul d'intersection, nous avons opté pour la solution MIMD.

Dispositif de communication

Le fonctionnement MIMD implique que l'émission d'un rayon et sa réception par le processeur voisin peuvent se produire à des instants différents.

Deux possibilités existent alors :

1. Synchronisation par rendez-vous
   Un processeur désirant émettre un rayon attend que celui avec lequel il désire communiquer soit disponible. Ce fonctionnement nécessite très peu de matériel. Mais la plus grande partie du Lancer de Rayon Discret consiste à effectuer le suivi des rayons; l'attente va donc influer fortement sur les performances globale du réseau voxel. De plus le blocage d'un processeur peut entraîner un risque non négligeable de *dead-lock* (A désire communiquer avec B qui désire communiquer avec A...).
2. Communication asynchrone
   Dans ce cas, un dispositif matériel doit être ajouté à chaque cellule, permettant de stocker les rayons en attendant de pouvoir les traiter.

Pour atteindre une efficacité importante, et donc un gain important, il faut libérer les processeurs le plus rapidement possible, et nous avons choisi la deuxième solution.

5.2.3 Définition matérielle d'une cellule

De l'algorithme du Lancer de Rayon Discret nous pouvons extraire deux opérations différentes : le suivi d'un rayon jusqu'à intersection, et le calcul d'éclairement avec création éventuelle de nouveaux rayons.

Si le coût global du premier traitement est le plus important, ce n'est que par le nombre élevé de rayons, le calcul d'éclairement d'un voxel étant bien plus complexe. De plus la probabilité d'intersection entre un rayon et un objet au sein d'un sous-cube de petite taille (telle que l'étude de la phase de conversion nous a amené à le choisir) est relativement faible. Là aussi nous désirons éviter de bloquer l'avancée des rayons à cause du traitement de l'éclairement.

Nous avons donc séparé ces deux opérations au sein de deux unités matérielles différentes, fonctionnant en parallèle : une Unité de Calcul et une Unité de Routage *avancée* des rayons. Nous entendons par *avancée* le fait que l'Unité de Routage gère complètement et de manière autonome le suivi des rayons : un rayon n'est envoyé à l'Unité de Calcul que si une intersection a été détectée par le dispositif de routage. Nous utilisons également des communications asynchrones entre ces deux unités.

Nous obtenons une cellule pour laquelle nous pouvons définir des algorithmes MIMD sans pertes d'efficacité dûes au type de contrôle de ses différentes unités, leur fonctionnement étant complètement asynchrone. Le parallélisme interne permet à un rayon sans intersection au sein du sous-cube de poursuivre son *chemin* vers la cellule suivante sans attente dûe à la charge de l'Unité de Calcul.

Ce fonctionnement correspond à celui du processeur cellulaire défini par l'équipe de Mazare à Grenoble [Paya91] (Figure 5.3). Nous baserons nos études sur ce type de circuit. Les communications utilisent des systèmes de *boites aux lettres* (BAL) composées d'une unité de stockage d'informations (rayon ou autres) et d'un drapeau indiquant qu'un message a été déposé. Une BAL est utilisée dans chacune des directions de connexions entre cellules. Elles sont gérées par l'Unité de Routage. On dispose également de BAL entre les deux unités internes de la cellule.

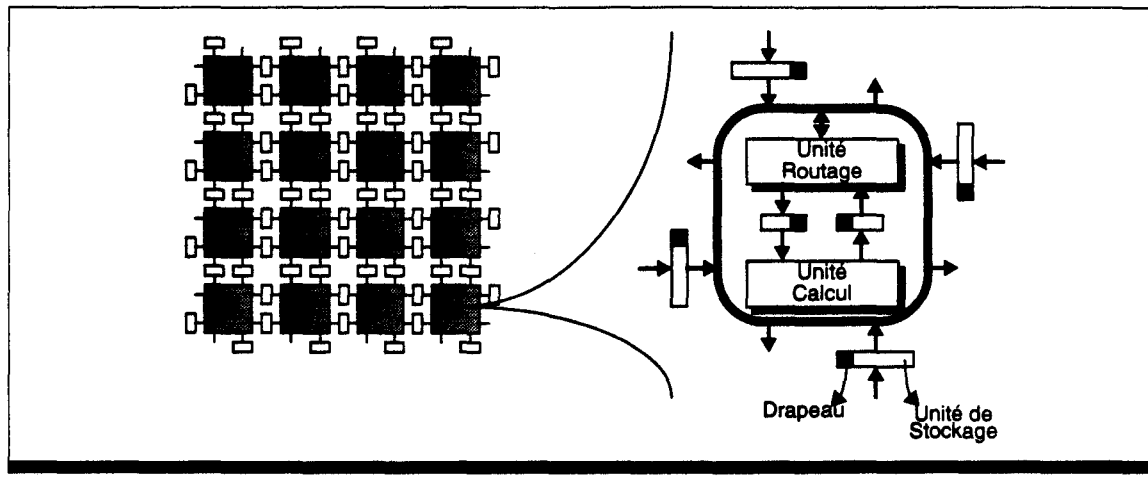


Figure 5.3 ***Réseau cellulaire, tel que défini dans [Latt88].*** *Représentation d'un réseau 2D. Une cellule identique peut être définie pour un réseau 3D, par adjonction d'unités de stockage.*

5.3 Lancer de Rayon Discret : Description du fonctionnement

Nous allons ici étudier plus précisément la phase de Lancer de Rayon Discret proprement dite. Celle-ci débute par le choix des rayons primaires à transmettre, puis par le suivi de ceux-ci jusqu'à intersection, la génération de rayons secondaires le cas échéant, et le retour des informations d'intensités lumineuses.

5.3.1 Génération des rayons primaires

Un rayon primaire est un rayon dont l'origine est situé à la position de l'observateur et passant par le centre d'un pixel écran. Or il n'existe pas de relation simple entre les pixels et les voxels, le centre d'un pixel pouvant avoir des coordonnées non entières (Figure 5.4).

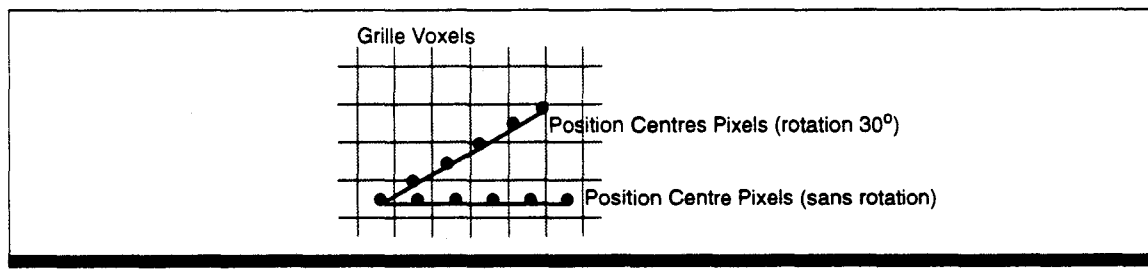


Figure 5.4 ***Position 'hors grille' des centres de pixels.*** *Exemple pour une grille 2D.*

Il existe plusieurs choix des voxels qui serviront d'origine aux rayons primaires. Un utilitaire permettant la projection d'un cube voxel a été écrit afin d'observer la qualité visuelle obtenue dans chacun des cas. Les résultats que nous présentons proviennent de la projection d'un simple cube dont les surfaces sont parallèles aux plans principaux, le plan de projection étant orienté à 45 degrés suivant les deux rotations d'un repère sphérique.

Conversion entière

Pour chaque centre de pixel, on prend comme origine du rayon le voxel dont le centre est le plus proche. L'inconvénient de cette méthode est que plusieurs voxels seront utilisés pour des pixels différents, ce qui produit des régions de pixels de même intensités (Figure 5.5.a).

Plan voxel-écran

Plutôt que d'utiliser les pixels pour le choix des voxels origines, nous utilisons une voxelisation du plan support de la grille correspondant à l'écran. Depuis chaque voxel appartenant à ce plan, un rayon primaire est émis. Il s'agit ensuite d'affecter à chaque pixel-écran la valeur d'intensité lui correspondant. Pour cela, on calcule pour chaque pixel le pourcentage de sa surface de projection qui appartient au sein des différents voxels qu'il recouvre. Les valeurs des intensités calculées pour chaque rayon primaire sont pondérées par ce pourcentage.

Si le plan est 26 connexe, alors il existe moins de rayons primaires que de pixels, on est donc placé dans un cas de sous-échantillonnage. Même si l'image finale apparaît filtrée, on obtient toujours des zones d'intensités identiques (Figure 5.5.b).

Si le plan est 6 connexe, alors il existe plus de rayons primaires que de pixels, on est cette fois placé dans un cas de sur-échantillonnage, ce qui améliore la qualité de l'image (Figure 5.5.c), mais l'échantillonnage reste visible. En effet, deux rayons discrets différents peuvent coïncider sur le même voxel d'intersection, ce qui donnera la même valeur d'intensité.

Suréchantillonage

Pour limiter le risque que deux rayons différents intersectent un objet dans le même voxel, nous pouvons utiliser un suréchantillonage réel des rayons : nous conservons identique le cube voxel contenant les objets, mais les rayons utilisent un *pas de progression* plus fin que cette grille voxel. On utilise un suréchantillonage identique de la grille pixel. Pour chaque *sous-pixel*, une conversion entière permet de déterminer le *sous-voxel* origine du rayon primaire. On obtient alors une image de qualité correcte (Figure 5.5.d - échantillonnage 2x2).

Notons que cette méthode nécessite un algorithme de suivi de rayon sur-échantillonné. Le parcours des rayons est alors plus long (identique à un parcours dans un cube de dimension plus élevé). De plus, un plus grand nombre de rayons primaires est généré.

Pour l'exemple que nous avons pris, cette méthode est 6,7 fois plus lente que le plan voxel-écran de connexité 26, et 2,8 fois plus lente que le plan 6 connexe.

En fonction des performances et de la qualité voulue, on choisira donc une méthode plutôt qu'une autre.

5.3.2 Suivi des rayons et intersection

Le suivi d'un rayon en lui-même est constitué par l'utilisation d'un algorithme de tracé de droite 3D. Suivant l'organisation du réseau voxel, on choisira donc l'algorithme de Kaufman pour un réseau $3D$, ou notre algorithme par construction pour un réseau $2D$. Cet algorithme sera câblé au sein de l'unité de routage des cellules du réseau.

Lorsqu'un rayon atteint un nouveau voxel, un test doit être effectué pour déterminer la présence ou non d'un objet. Rappelons que les performances du réseau durant la phase de Lancer de Rayon Discret sont en grande partie fixées par la capacité de traitement des rayons. Par conséquent, nous voulons éviter au maximum d'utiliser l'Unité de Calcul de la cellule. En effet si un calcul d'éclairement est en cours, il est nécessaire d'atteindre la fin de son évaluation pour effectuer le test d'intersection, qui peut se révéler négatif!

Il est donc plus avantageux de dédier à l'Unité de Routage le test d'intersection. Pour cela, après la phase de voxelisation, un masque binaire est chargé par l'Unité de Calcul dans l'Unité de Routage. Ce masque indique pour chaque voxel du sous-cube la présence d'un objet. Si il y a réellement intersection, alors le rayon est redirigé vers l'Unité de Calcul, sinon il est envoyé vers le voxel suivant. Nous pouvons ainsi acheminer au plus vite les rayons sans intersection dans le sous-cube.

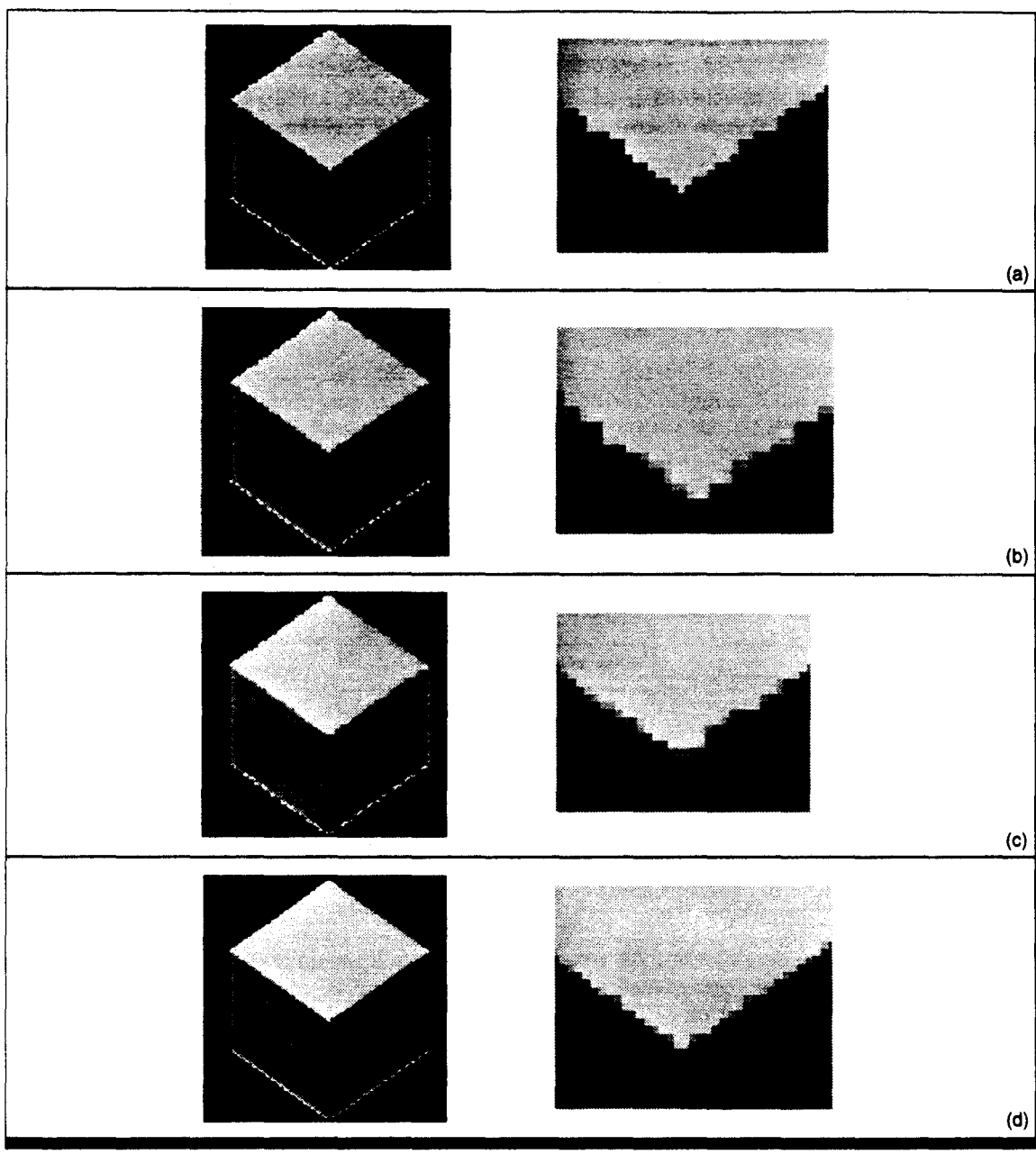


Figure 5.5 ***Projection d'un cube parallèle.*** *(les vues droites sont des agrandissements du coin avant) (a) - Conversion entière. (b) - Plan 26-connexe. (c) - Plan 6 connexe. (d) - Suréchantillonage.*

En cas d'intersection, le rayon doit être chargée dans la BAL d'entrée de l'Unité de Calcul, il est donc nécessaire que cette première soit vide. Dans le cas contraire, l'Unité de Routage sera bloquée, ce qui peut provoquer un phénomène de dead-lock. La solution habituelle est d'ajouter une file d'attente en entrée de l'Unité de Calcul. La définition de la longueur de cette file est très délicate; il faut éviter de la sur-estimer pour réduire le coût matériel, mais il faut également ne pas la sous-estimer pour éviter les blocages. En fonction des performances comparées de débit de l'Unité de Routage et de calcul d'éclairement de l'Unité de Calcul, la longueur optimum sera également différente. Il nous faut donc attendre les résultats de simulation pour en juger. Par contre un système permettant d'éviter tout blocage permettrait d'éviter ce surcoût matériel. Nous étudierons ce dispositif plus loin (cf § 5.4 - p. 123).

Intersection rayon discret / objet discret

Afin d'obtenir une image visuellement correcte, il est bien sûr important d'afficher en un pixel l'intensité lumineuse correspondant au point d'intersection réel entre un rayon et un objet réel. Toute dérive importante peut entraîner des effets indésirables. Nous concevons bien que l'utilisation du monde discret va ici poser quelques problèmes, que nous allons énumérer.

Connexité des rayons

Un phénomène important dû à l'utilisation de la géométrie est l'existence d'effets passes murailles dépendant de la connexité relative des objets et des rayons (cf. Annexe D). Cet effet peut être éliminé en choisissant une 6-connexité pour l'un des deux groupes. Mais dans ce cas, l'objet ou le rayon ayant une *épaisseur* trop importante, on peut aboutir à une erreur d'intersection, i.e. détecter une intersection dans un voxel n'appartenant pas réellement à l'objet ou au rayon (Figure 5.6). Les études menées par l'équipe de Kaufman sur un Lancer de Rayon Discret séquentiel indiquent qu'il n'existe que de l'ordre de 7% d'erreurs de ce type [Yage92]. Ces chiffres correspondent à des essais sur des scènes ne comportant pratiquement que des sphères. Sur des objets formés de grands polygones, on peut douter de la validité de ces chiffres (Figure 5.7.a).

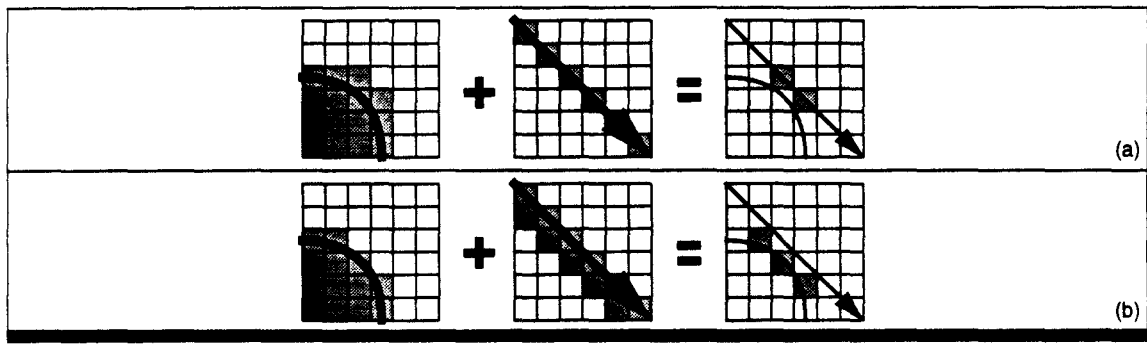


Figure 5.6 ***Erreur d'intersection.*** *Suivant la connexité de l'objet et du rayon. L'image de droite représente les voxels où une intersection erronée est trouvée. (a) - Objet 6-connexe et rayon 26-connexe. (b) - Objet 26-connexe et rayon 6-connexe.*

De plus si un choix erroné du voxel d'intersection crée peu d'erreur visuelle sur des objets ayant une courbure importante, il n'en est pas de même pour des objets plans, tel un miroir par exemple (Figure 5.7.b).

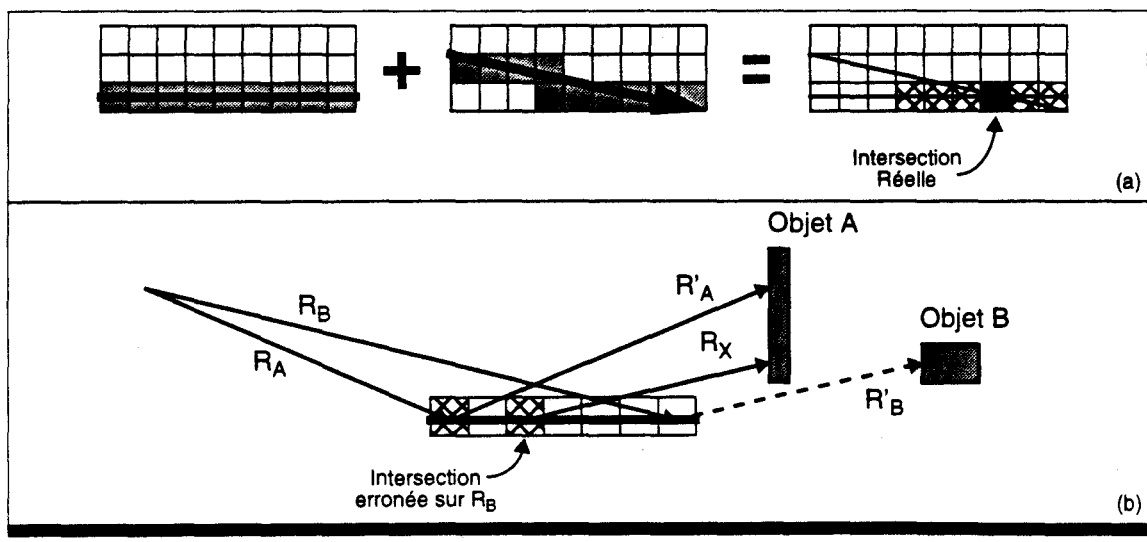


Figure 5.7 ***Erreur d'intersection sur un plan.*** *(a) - Un rayon rasant générera un grand nombre d'intersections erronées. (b) - Le rayon R_B doit générer le rayon réfléchi R'_B, mais par erreur d'intersection, le rayon R_X est lancé -> seul l'objet A sera visible par réflexion.*

En tout état de cause, il semble nécessaire d'utiliser une technique de vérification d'intersection. Le pourcentage d'erreur étant plus faible que le pourcentage d'intersection juste, cette vérification devrait être assurée par l'Unité de Calcul. La seule possibilité pour réaliser ce traitement est de posséder la description géométrique de l'objet et d'effectuer un calcul réel d'intersection. Pour son Lancer de Rayon Discret séquentiel Kaufman n'utilise le cube voxel stocké en mémoire centrale que pour accélérer la recherche d'intersection avec un rayon, tous les autres traitements étant effectués à partir des définitions géométriques sur un processeur standard.

Dans notre cas, nous voulons implémenter tous les calculs sur le réseau, mais il est inenvisageable de conserver la description de l'objet en chaque voxel. Nous proposons d'approximer la surface au sein d'un voxel par un morceau de plan. En effet, nous possédons déjà la valeur de la normale à l'objet en ce point. La définition d'un plan ne nécessite qu'une donnée de *positionnement* supplémentaire. Si $\vec{N} = [N_X\ N_Y\ N_Z]$ est la normale à un plan, son équation est : $N_X \cdot x + N_Y \cdot y + N_Z \cdot z + d = 0$. Nous appelons d la donnée de *positionnement* du plan.

Nous pouvons alors appliquer un test d'intersection entre ce plan et le rayon incident. L'évaluation du coût et de la qualité de ce traitement n'a pas encore été effectuée.

5.3.3 Génération des rayons secondaires

Après détection de l'intersection, et suivant les caractéristiques photométriques de l'objet, un rayon réfléchi et un rayon réfracté sont transmis. Ces rayons sont émis à partir du voxel où s'est produit l'intersection. Or, suivant la topologie de l'objet, ces rayons pourraient intersecter immédiatement le même objet (Figure 5.8). D'autres phénomènes plus complexes du même type peuvent apparaître.

Il est important de supprimer cette erreur, pour assurer la synthèse correcte de l'image. Nous ne disposons pas encore de solution correcte à ce problème.

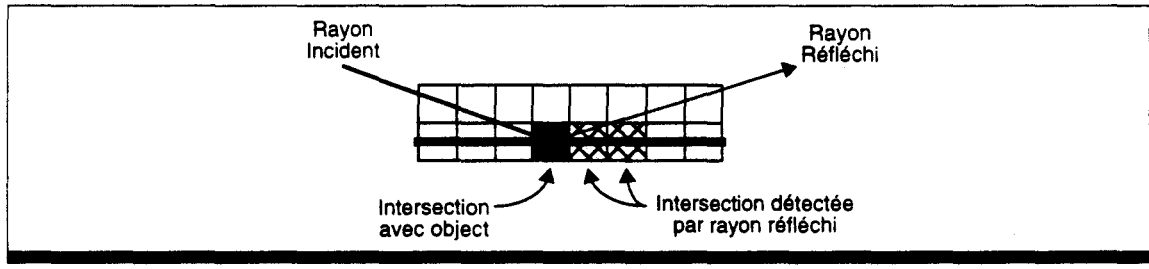


Figure 5.8 ***Erreur d'intersection sur rayon réfléchi.***

5.3.4 Rayons retour

Dans l'algorithme séquentiel classique, ainsi que dans certaines implémentations parallèles, l'arbre des rayons secondaires est parcouru en profondeur d'abord. L'algorithme est appelé récursivement : lors d'une intersection, un rayon secondaire est créé et lancé; l'information d'intensité lumineuse correspondant au point d'intersection est *remontée* dans la récursion vers le point d'intersection ayant généré le rayon secondaire (Figure 5.9).

Cette méthode implique qu'il est nécessaire de garder pour chaque point d'intersection les informations permettant le calcul de l'intensité, jusqu'au retour des rayons secondaires générés. Cela peut nécessiter un volume mémoire supplémentaire, ou un blocage de l'unité de calcul jusqu'à terminaison du calcul d'intensité.

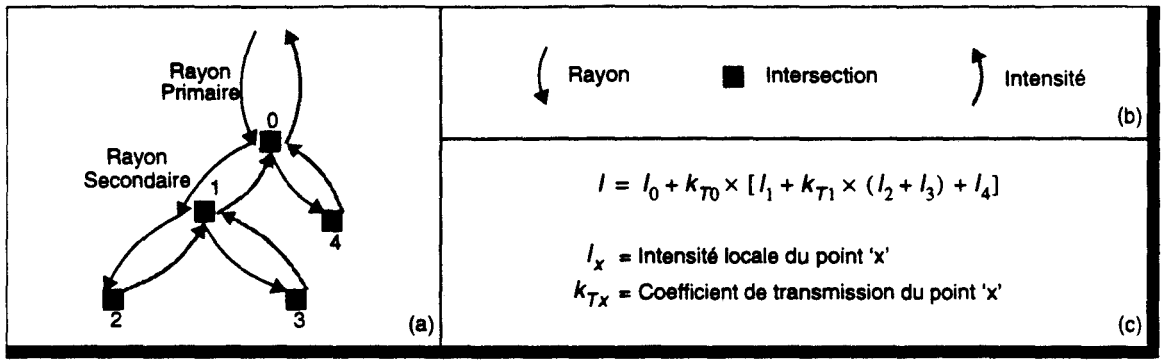


Figure 5.9 ***Exécution récursive du Lancer de Rayon.*** *(a) - Ordre de création et de retour de la récursion. (b) - Légende. (c) - Expression de l'intensité totale par cette méthode.*

L'exécution récursive de l'algorithme n'est cependant pas nécessaire, l'expression donnée en Figure 5.9.c pouvant se réécrire : $I = I_0 + k_{T0} \cdot I_1 + k_{T0} \cdot I_4 + k_{T0} \cdot k_{T1} \cdot I_2 + k_{T0} \cdot k_{T1} \cdot I_3$. Si l'on transmet l'accumulation des coefficients de transmission (et/ou de réfraction) aux rayons secondaires, les intensités locales peuvent alors être remontées directement à la cellule ayant en charge le rayon primaire. Une simple accumulation d'intensités permet le calcul de la couleur au pixel considéré (Figure 5.10).

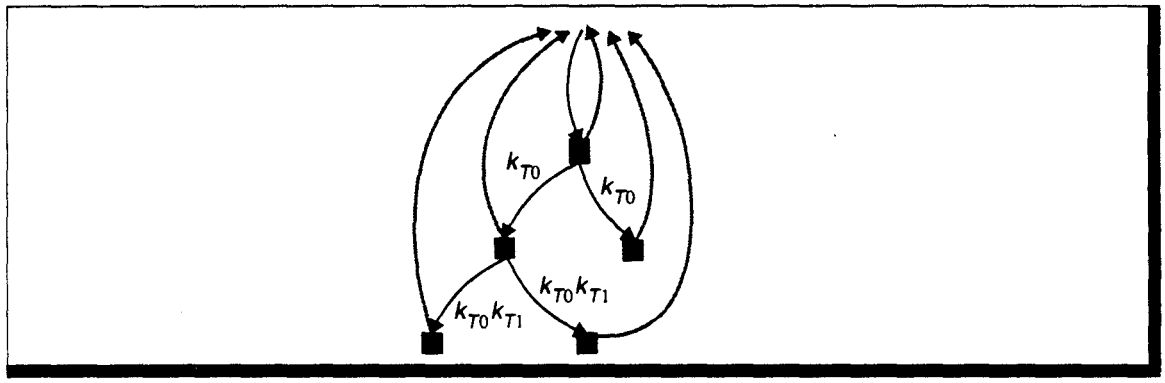


Figure 5.10 ***Exécution non récursive du Lancer de Rayon.*** *Les intensités sont envoyées directement au noeud père.*

Remarquons que les *rayons retours* porteurs de l'information d'intensité lumineuse n'ont pas obligation à suivre la ligne droite. Ceci peut permettre l'utilisation d'un routage beaucoup plus simple (routage en X puis en Y par exemple). Par contre contrairement au mode d'exécution précédent, la cellule ayant en charge le pixel dont est issu l'arbre de rayons ne peut déterminer l'instant où elle a reçu toutes les informations d'intensités lumineuses, car elle ne peut connaître le nombre de rayons secondaires qui seront générés. Il est donc nécessaire de mettre en place un mécanisme de détection de fin d'exécution.

Nous proposons d'utiliser un mécanisme défini par Cleary [Clea86] :

- un compteur *Cpt* est adjoint à chaque rayon. Il est initialisé à 1 pour le rayon primaire.
- si un rayon ne possède pas d'intersection avec un objet de la scène ou si aucun rayon secondaire n'est émis, le compteur est renvoyé tel quel.
- si un rayon est en intersection, le rayon retour associé est émis avec une valeur *Cpt*/2.
- si deux rayons secondaires sont émis, chacun prendra une valeur *Cpt*/4
- si un seul rayon secondaire est émis, il prend une valeur *Cpt*/2

Tous les rayons ont été reçus par la cellule gérant le pixel si $\sum Cpt = 1$.

5.3.5 Efficacité suivant organisation du réseau

Le fonctionnement des deux unités internes de la cellule influe sur l'efficacité globale du Lancer de Rayon Discret. En dehors de toutes simulations, nous fournissons un ensemble d'informations permettant de juger de cette efficacité suivant l'organisation du réseau (2D ou 3D).

Fonctionnement comparé de l'Unité de Routage

La performance du réseau en terme de *débit de rayons* est influencé par le temps nécessaire à un rayon pour traverser une cellule, ce qui dans le cas d'un réseau 2D est directement fonction de la longueur des paliers de rayons. Pour spécifier plus précisément cette longueur, nous avons simulé un suivi incrémental 26-connexe sur un réseau 2D, et nous avons lancé un rayon depuis le voxel situé au centre du cube vers chacun des voxels du bord du cube (i.e. dans chaque direction de l'espace 3D). Pour $N_V = 512$, nous obtenons une longueur moyenne de palier de 1 voxel (Table 5.1).

De ce fait, les temps de parcours des rayons seront à peu près identiques pour une organisation $3D$ ou $2D$. Cependant, si la simulation effectuée est représentative des rayons secondaires, il en est autrement des rayons primaires. En effet, si nous effectuons, par exemple, une projection parallèle, ou quasi parallèle, suivant Z, chaque rayon primaire aura un parcours entièrement interne à une cellule du réseau $2D$.

Afin de réduire le temps de traversée d'un rayon pour une organisation $2D$, nous proposons un dispositif matériel permettant la détermination de l'intersection le long d'un palier de rayon en un nombre de cycles indépendant de la longueur de celui-ci (cf § 5.4 - p. 123). Ceci implique la mise en oeuvre d'un suivi de rayon utilisant le tracé de droite par construction. A l'aide de cette unité, le temps de gestion d'un rayon au sein d'une cellule est pratiquement identique sur un réseau $2D$ et sur un réseau $3D$.

Lgr	Nb Occur.	Lgr	Nb Occur.	Lgr	Nb Occur.	Lgr	Nb Occur.
1	1.403.910	7	1.208	13	200	32	80
2	122.368	8	1.024	15	184	43	64
3	22.304	9	304	16	160	64	40
4	8.160	10	544	19	144	128	24
5	3.432	11	240	22	120	255	2
6	1.888	12	224	26	104		

Table 5.1 ***Occurrence des paliers de différentes longueurs.***

La probabilité qu'un rayon atteigne un sous-cube est fonction de la surface de ce dernier. A nombre de processeurs constant, elle est donc identique pour les deux organisations. Le nombre de rayons devant être géré par une Unité de Calcul est le même dans les deux cas.

Le débit global en terme de rayons/seconde est fonction du nombre de rayons pouvant être transmis d'une cellule vers une autre en un cycle horloge. Il est donc fonction du nombre d'Unités de Calcul. Si nous travaillons à nombre de processeurs constant, nous obtenons donc des performances équivalentes.

Fonctionnement comparé de l'Unité de Calcul

Le débit de rayons est également fonction de la charge des Unités de Calcul. Plus une Unité de Calcul est chargée moins elle pourra libérer rapidement l'Unité de Routage de sa cellule. La charge d'une Unité de Calcul est globalement fixée par le nombre de voxels occupés qu'elle gère.

De par la cohérence spatiale, si un voxel est occupé la probabilité qu'un voxel voisin soit occupé est élevée. Cette localité a permis l'utilisation efficace de réseau partiel pour la discrètisation. Par contre, elle va ici créer une influence défavorable. En effet, même si en moyenne les cellules

disposent du même nombre de voxels occupés dans le cas $3D$ et $2D$, la répartition est beaucoup plus homogène pour l'organisation$3D$. On trouvera donc plus de cellules fortement chargées et d'autres faiblement chargées dans ce type de réseau.

Nous pensons donc que la charge est plus équilibrée sur un réseau $2D$, ce qui devrait permettre une efficacité plus élevée. Cette remarque nous a fait retenir cette organisation comme offrant le meilleur compromis performances/coût matériel.

5.4 Dispositif de détection d'intersection

Pour augmenter le débit de traitement des rayons au sein du réseau, nous avons choisi de séparer le traitement du suivi de rayons du reste des calculs afin de permettre à un rayon de traverser un sous-cube voxel, où il n'a aucune intersection, sans utilisation de l'Unité de Calcul.

Il nous faut pour cela disposer au sein de l'Unité de Routage d'un dispositif permettant de déterminer l'éventuelle intersection entre un rayon et les voxels occupés appartenant à la cellule. Une mémoire, contenant pour chaque voxel un drapeau indiquant la présence ou l'absence de matière, est remplie par l'Unité de Calcul lors de la phase de voxelisation. Cette mémoire permet à l'Unité de Routage de déterminer l'intersection potentielle entre un rayon et un voxel donné.

Rappelons que le suivi de rayons est effectué par l'algorithme de tracé de droite 3D par construction, c'est-à-dire qu'au sein d'une cellule un rayon est caractérisé par sa profondeur avant et arrière. Ainsi, il est nécessaire de tester chaque voxel appartenant au palier du rayon pour déterminer la position d'une éventuelle intersection. Plus le palier est profond, plus le temps nécessaire à cette détermination sera élevé. Or, les performances du Lancer de Rayon Discret sont avant tout fonction du débit des rayons au sein du réseau. Il nous faut donc minimiser la durée nécessaire à une cellule pour faire avancer le rayon.

Nous proposons d'utiliser un dispositif matériel permettant la détermination de la position de l'intersection en un nombre de cycles minimum indépendant de la longueur du palier. Les chiffres que nous donnons ci-après, concernant la taille et le temps de transfert des circuits, correspondent à la technologie CMOS 1.2µm d'ES2.

Principe de la détection d'intersection

On dispose d'un masque spécifiant l'occupation de chaque voxel (que nous nommerons VP[n]), et des deux profondeurs avant (Zav) et arrière (Zar) du palier de rayon.

La détection d'intersection est effectuée en deux temps : recherche de la position du premier voxel occupé situé à une profondeur supérieure ou égale à Zav, puis vérification que cette position est inférieure ou égale à Zar.

Recherche premier voxel occupé

Ce traitement peut être réalisé à l'aide d'une chaîne de cellules qui propagent un signal de recherche depuis VP[Zav] jusqu'à VP[i] tel que VP[i]=1 et $i \geq$ Zav (Figure 5.11).

Nous avons synthétisé ce circuit pour une valeur $n = 16$. Il comporte ~600 transistors. Son temps de transfert est de 31ns.

Notons que chaque cellule comporte 4 portes logiques, et que la longueur de la chaîne de génération des signaux Si est de $2n$ portes logiques. Par conséquent le temps de transfert d'une chaîne composée de 512 cellules (correspondant à un cube voxel de dimension $N_V = 512$) sera bien trop élevé (de l'ordre de 1µs).

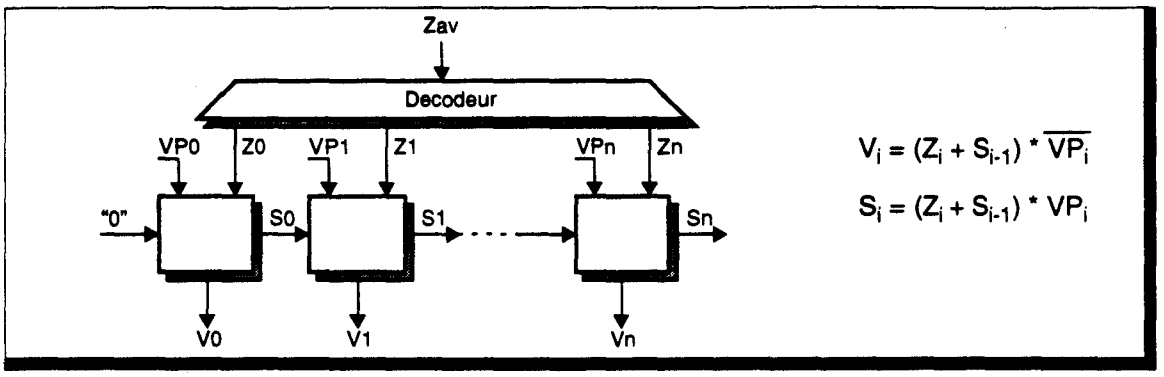


Figure 5.11 ***Unité de recherche du premier voxel occupé.*** *Version 1 : Utilisation d'une chaîne de recherche et d'un décodeur.*

Amélioration par synthèse complète

Pour tenter d'améliorer sensiblement le temps de traitement de l'unité de recherche du premier voxel occupé, nous avons effectué une synthèse directe, laissant au logiciel de synthèse de Solo1400 effectuer la minimisation du circuit dans son ensemble.

Nous obtenons dans ce cas un modèle comportant ~1500 transistors, pour un temps de traitement de 14ns (toujours pour 16 bits).

Même si les performances sont améliorées, elles sont encore trop faibles pour utiliser ce modèle sur une largeur de 512 bits.

Détection en 2 phases

Il ne semble donc pas possible de créer un circuit permettant la recherche sur une grande largeur, pour un coût silicium et un temps de traitement raisonnables. Nous proposons donc de découper le traitement en deux phases de recherche.

Pour cela, nous découpons le masque VP en bloc de 16 bits de large, avec pour chaque groupe un bit VPB indiquant la présence d'un voxel occupé au sein de ce bloc. Nous effectuons alors une première recherche pour déterminer le premier bloc occupé, puis une deuxième recherche est déclenchée sur le groupe ainsi déterminé. Le schéma de principe correspondant à ce fonctionnement est donné en Figure 5.12. Les modules UROC sont les Unités de Recherche d'OCcupation. Les signaux T_H et T_L indiquent la réussite de la recherche.

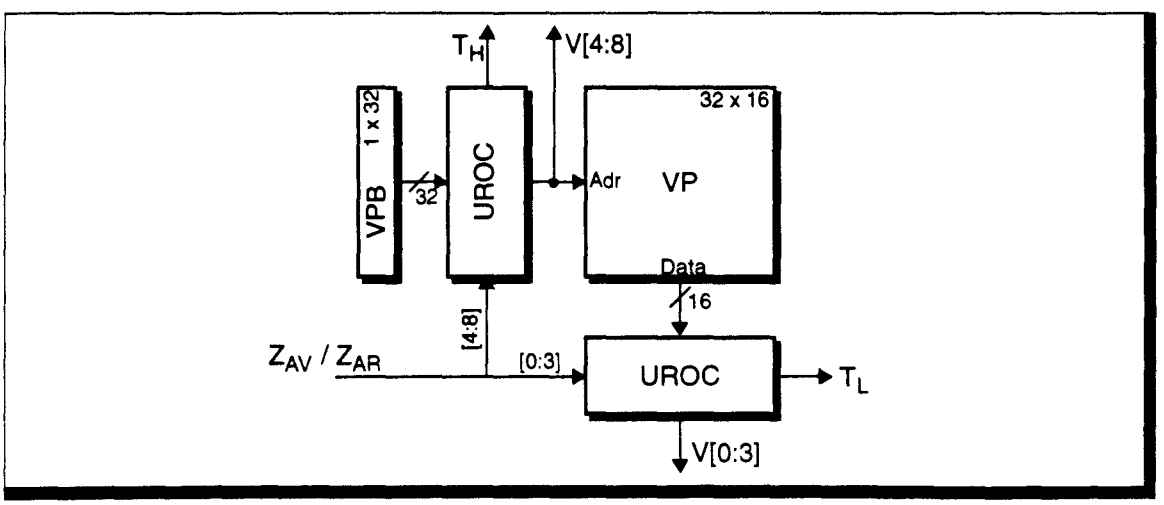


Figure 5.12 ***Unité de recherche par blocs.***

Implémentation de l'Unité de Recherche d'OCcupation

Nous avons voulu conserver le temps de transfert permis par la synthèse globale, tout en minimisant le nombre de portes nécessaires. Nous avons pour cela modifié le fonctionnement de l'ensemble 'Décodeur-Cellules de recherche' (Figure 5.13) :

- Le décodeur positionne à 1 tous les signaux Z_i tels que $i \geq Z_{AV}$,
- Les signaux générés par une cellule de recherche sont maintenant :

$$S_i = (Z_i * VP_i) + S_{i-1}$$

$$V_i = (Z_i * VP_i) * \overline{S_{i-1}}$$

On remarque que l'on peut extraire le calcul de $Z_i * VP_i$, et que la longueur de la chaîne de recherche est maintenant de n portes, ce qui permet de diminuer sensiblement le temps de réponse du circuit.

Pour limiter la recherche à la profondeur ZAR, nous utilisons un dispositif analogue au précédent : un décodeur positionne à 1 tous les signaux Z'_i tels que $i \leq Z_{AR}$.

Les cellules de recherche calculent alors : $V_i = (Z_i * VP_i) * \overline{S_{i-1}} * Z'_i$

La dernière partie de l'Unité de Recherche d'OCcupation consiste à réencoder les signaux V_i, pour générer la valeur correspondant à la profondeur obtenue. Le signal TAG indique la réussite de la recherche.

Les caractéristiques de cette unité pour les deux configurations nécessaires sont données en Table 5.2.

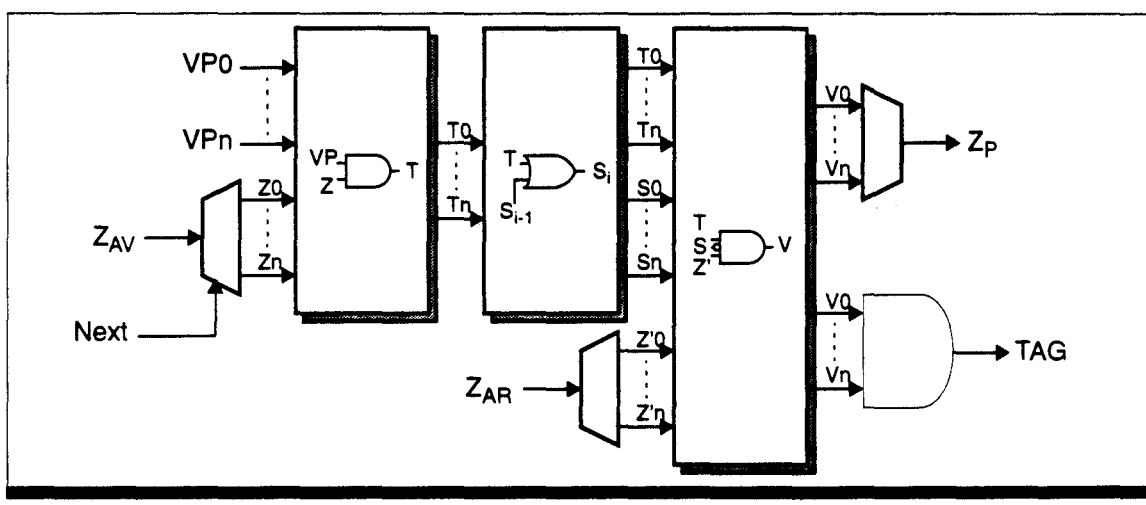


Figure 5.13 ***Unité de Recherche d'OCcupation.***

n	Nb Xtors	Tps Prop.	Surface
16	1600	13.5ns	0.7 x 1.0mm
32	4600	17ns	1.7 x 1.6mm

Table 5.2 ***Caractéristiques techniques de l'Unité de Recherche d'OCcupation.***

Utilisation de l'Unité de Recherche par Blocs

L'utilisation d'une recherche par blocs introduit en effet de bord dont il faut tenir compte dans l'algorithme de traitement.

Prenons l'exemple du masque d'occupation donné en Figure 5.14. Avec les positions Z_{AV} et Z_{AR} telles qu'indiquées, la première recherche sur VPB fournira en résultat le premier bloc. Or le premier voxel occupé est situé dans le second bloc.

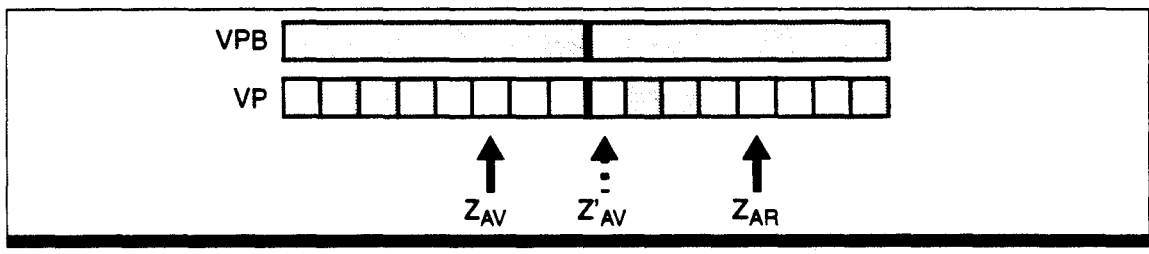


Figure 5.14 ***Exemple de l'effet induit par la recherche par bloc.***

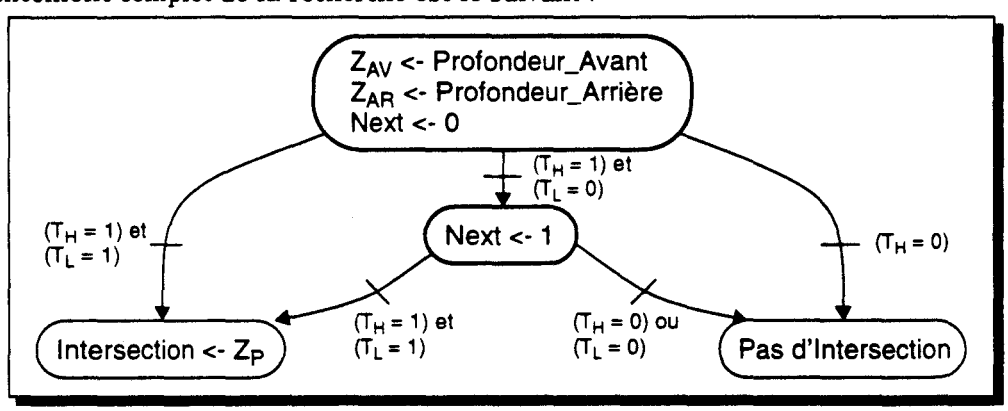
Cette erreur est cependant facilement détectable car en effet la recherche au sein du premier bloc sera négative. Dans ce cas, une recherche doit être redéclenchée sur le bloc suivant (Z'_{AV}).

Pour faciliter cette deuxième opération, le circuit de décodage de Z_{AV} utilise le signal Next pour modifier les signaux générés (Figure 5.13) :

- Si Next = 0 -> $Z_i = 1$ si $i \geq Z_{AV}$
- Si Next = 1 -> $Z_i = 1$ si $i > Z_{AV}$

Le surcoût induit est de 80 transistors, et un allongement du temps de propagation de ~2ns.

Le séquencement complet de la recherche est le suivant :



Si l'unité mémoire permettant le stockage de VP est suffisamment rapide, ce traitement pourra être effectué en deux cycles horloge.

En conclusion

La surface occupée par l'Unité de Recherche par Bloc est relativement important (~ $4mm^2$). Cependant il permet de trouver une intersection le long d'un palier de longueur quelconque en 2 cycles horloge, ce qui justifie amplement son utilisation.

5.5 Fonctionnement de l'Unité de Routage

Rappelons que nous avons choisi une communication asynchrone par transfert de messages. Un des problèmes importants de ce type de fonctionnement est la possibilité d'aboutir à un interblocage des dispositifs de routage. Nous allons étudier plus précisément ce phénomène et y apporter une solution, mais auparavant nous allons spécifier le fonctionnement algorithmique de l'Unité de Routage.

5.5.1 Routage des messages

L'algorithme général du routage des messages peut-être défini comme suit :

```
Scrutation des Boîtes aux lettres
Si (Message présent)
|  Suivant (Type Message)
|  |  Traitement Message
|  Fin
FSi
```

Chaque cellule dispose de 5 boîtes aux lettres en entrée (4 extérieures et 1 depuis l'Unité de Calcul). Il serait possible d'envisager un traitement parallèle et simultané des 5 entrées. Il est alors nécessaire de disposer de 5 unités complètes de détection d'intersection et de calcul du suivi de rayon. De plus, deux messages pouvant avoir à emprunter la même boîte aux lettres de sortie, il faut mettre en place un protocole de gestion de ces ressources partagées. Dans le cadre d'une architecture massivement parallèle, nous voulons limiter la taille des cellules élémentaires, et nous utiliserons un traitement séquentiel sur les 5 entrées avec une Unité de Routage unique.

Traitement suivant le type de message

Il existe 2 types de messages, pour lesquels le traitement et le routage à effectuer sont différents :

- Message *Rayon d'Intersection*

  Nous appelons ainsi les messages contenant les rayons primaires et secondaires. Ils doivent être routés au sein du réseau en suivant une ligne droite, jusqu'à intersection ou sortie de l'espace voxel.

  Le traitement associé est le suivant :

```
Test Intersection
Si (Intersection)
|  Envoi Message à l'Unité de Calcul
Sinon
|  Calcul prochain voxel
|  Si (Sortie Espace Voxel)
|  |  Envoi Rayon Retour avec Couleur Fond
|  Sinon
|  |  Envoi Message Cellule suivante
|  FSi
FSi
```

- Message *Rayon Retour*

  Ces messages contiennent l'information de couleur résultant d'une intersection. Ils doivent simplement être dirigés vers la cellule prenant en charge le pixel-source associé au rayon ayant généré l'intersection. Le chemin de routage peut-être quelconque, et nous choisirons un routage de type *e-cube* :

```
Si (A Destination)
|  Envoi Message à l'Unité de Calcul
Sinon
|  Envoi Message Cellule suivante
FSi
```

Gestion des Boîtes aux Lettres

La transmission de messages étant asynchrone, il est nécessaire de mettre en place un dispositif permettant de connaître la disponibilité d'une boîte aux lettres, avant de pouvoir y déposer un message (pour le processus émettant le message) ou de pouvoir en lire le contenu (pour le processus recevant le message). Chaque boîte dispose pour cela de deux drapeaux : Vide, et Plein (Figure 5.15).

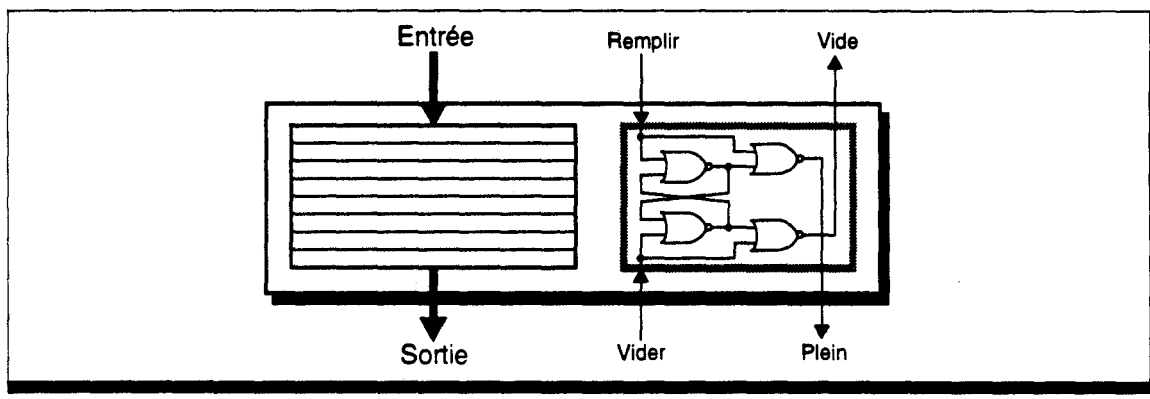


Figure 5.15 ***Schéma d'une boîte aux lettres (d'après [Latt88]).*** *Les deux drapeaux* Vide *et* Plein *permettent de connaître l'état de la boîte. Ils sont générés par un dispositif assurant un non recouvrement des signaux.*

Avant d'émettre un message, l'Unité de Routage doit vérifier que la boîte aux lettres destinataires est disponible (drapeau Vide positionné). Dans le cas contraire, le message ne peut être déposé, et :

- soit, le traitement en cours est bloqué jusqu'à libération de la boîte aux lettres. Dans ce cas, l'Unité de Routage complète est bloquée, ce qui peut entraîner un blocage en chaîne, les autres messages ne pouvant être acheminés.
- soit, le traitement du message est aborté afin de libérer l'Unité de Routage. Mais dans ce cas, tout le traitement du message est à recommencer lorsque l'Unité de Routage prendra de nouveau en charge sa gestion.
- soit, le traitement du message est suspendu, les informations résultantes du traitement effectué étant conservées. L'Unité de Routage est libérée, sans nécessiter de réeffectuer le traitement du message, lors de la prochaine prise en compte de celui-ci.

Les performances du Lancer de Rayon Discret étant fonctions du débit global atteint par le réseau de communication, la dernière proposition apparaît comme la plus adaptée à notre application. Elle nécessite cependant un espace de stockage pour les informations déjà calculées (Figure 5.16) :

- *Rayon d'Intersection* avec intersection

  Il faut conserver la profondeur à laquelle l'intersection a été détectée. Cette valeur remplacera la valeur de profondeur du voxel courant, contenue dans le message.

On positionne également un drapeau Flag_Int, indiquant que le test d'intersection a déjà été effectué avec succès.

- *Rayon d'Intersection* sans intersection

  La position du prochain voxel est mémorisé dans le message à la place de la position courante. On positionnera en plus un drapeau Flag_Suivi.

- *Rayon Retour*

  Un drapeau Flag_Dest est positionné si le message est arrivé à destination et doit être transmis à l'Unité de Calcul. Un drapeau Flag_Ret est utilisé pour un message devant encore être routé.

  Notons qu'un rayon retour, lorsqu'il est consommé par la cellule destination, ne produit aucun autre message. Il pourrait donc être intéressant que l'Unité de Calcul le prenne en compte dès son arrivée afin de libérer la boîte aux lettres qu'il occupe. Il faut dans ce cas mettre en oeuvre une procédure d'interruption du traitement en cours dans l'Unité de Calcul. Sans simulations globales de la machine, nous ne pouvons indiquer si cela permet d'augmenter sensiblement les performances de communications, et nous n'étudierons pas plus avant ce procédé.

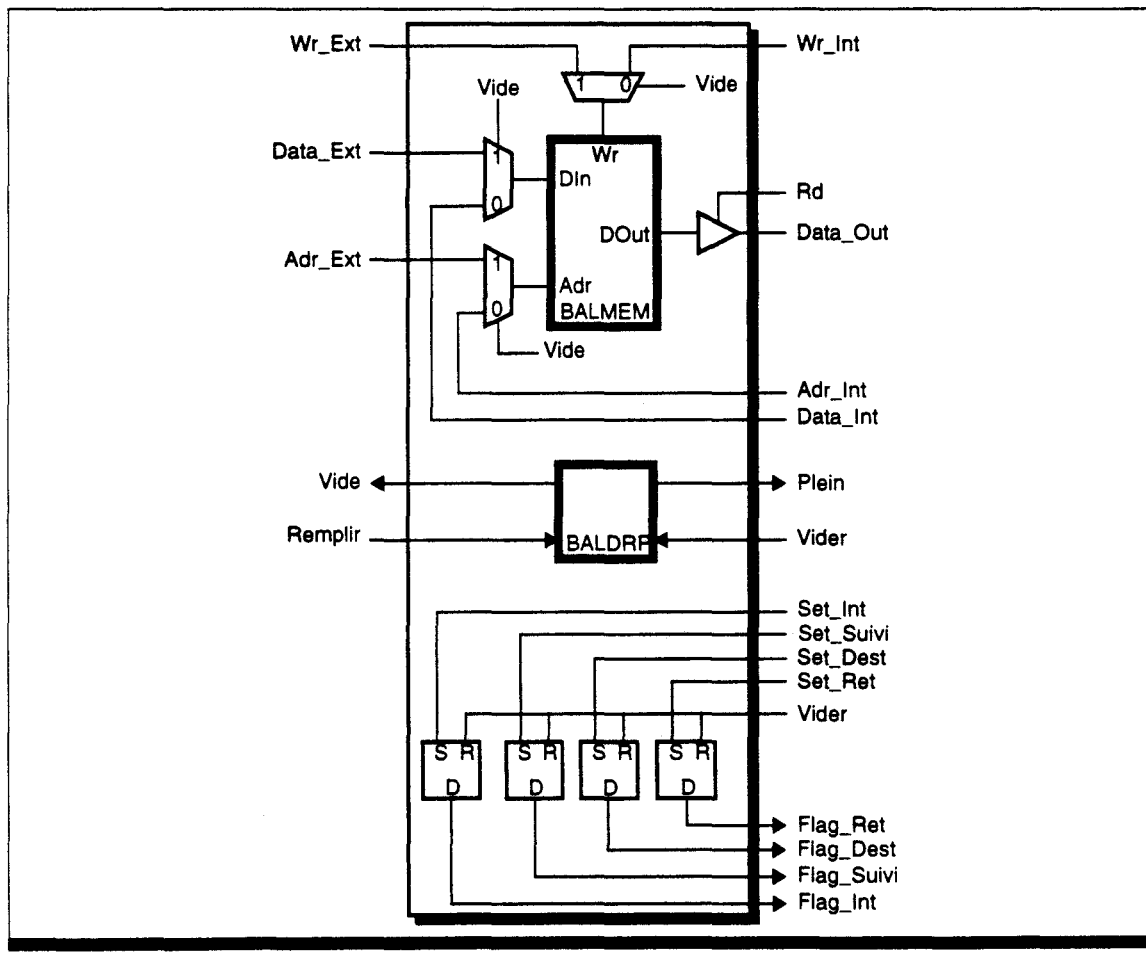


Figure 5.16 ***Schéma complet d'une boîte aux lettres.*** *Les* Flag_xxx *sont gérés par des bascules RS.*

On obtient maintenant l'algorithme général suivant :

```
PourChaque (BAL entrée)
|  Si (Flag_xxx et BAL sortie disponible)
|  |  Remplir BAL sortie
|  |  Vider BAL entrée
|  Sinon Si (Plein)
|  |  Traitement Message suivant type
|  |  Si (BAL sortie disponible)
|  |  |  Remplir BAL sortie
|  |  |  Vider BAL entree
|  |  Sinon
|  |  |  Positionner Flag_xxx et mémoriser valeurs calculées
|  |  Fsi
|  Fsi
FPour
```

5.5.2 Interblocage des communications

Un des problèmes majeurs inhérent au Lancer de Rayon parallèle à flot de rayons est la possibilité d'aboutir à un interblocage (*deadlock*) des unités de communications.

Rappelons brièvement les deux circonstances nécessaires à la création d'un tel blocage:

1. L'entrée d'une unité de communication est bloquée tant que le message en cours de traitement n'a pu être envoyé.
2. Il existe un cycle de dépendance entre les unités de communication (Figure 5.17).

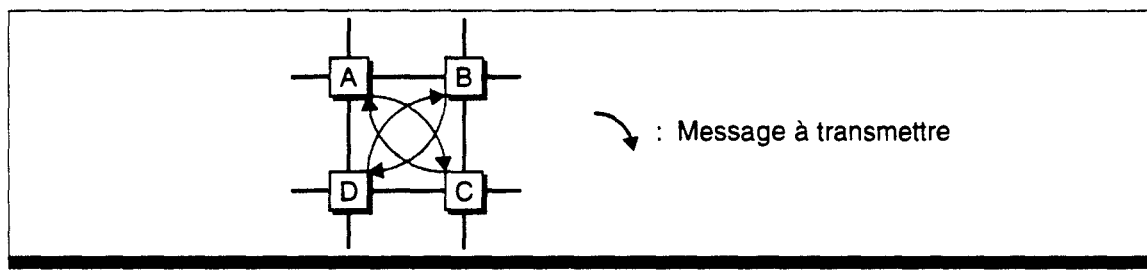


Figure 5.17 ***Cycle de dépendance sur les communications.*** *L'unité* A *est bloquée, car l'unité* B *est bloquée, car l'unité* C *est bloquée, car l'unité* D *est bloquée, car l'unité* A *est bloquée, car...*

L'élimination de l'un et l'autre de ces deux points permet de garantir des communications sans blocage (*deadlock free*). Détaillons plus précisément le phénomène de blocage dans le cas du Lancer de Rayon Discret afin de déterminer les solutions envisageables ou à rejeter.

Il existe tout d'abord un certain nombre de caractéristiques du suivi de rayon qui crée une différence notable entre cette application et les applications classiques de communications par messages:

1. Il y a obligation que les rayons suivent une ligne droite,
2. Le chemin d'un rayon est calculé au fur et à mesure de son évolution, autrement dît, un rayon n'est transmis que d'une cellule vers une cellule voisine, son chemin n'est pas prédéfini,
3. La cellule destinataire d'un rayon n'est pas connue à l'avance, puisqu'il s'agit de la cellule où se trouve la première intersection le long du rayon,
4. Nous ne voulons pas utiliser de buffers de messages, afin de limiter la taille des cellules,
5. Aucun contrôle global des communications n'est envisageable.

Le blocage d'un rayon provient du fait que l'Unité de Routage destinataire ne peut le consommer. A cela il peut y avoir deux raisons:

- l'Unité de Routage traite un autre message qu'elle doit transmettre à une cellule suivante qui est bloquée,
- l'Unité de Routage doit transmettre un message à son Unité de Calcul qui est bloquée.

Le blocage d'une Unité de Calcul, quant à lui, provient de l'impossibilité de transmettre un message à l'Unité de Routage associée.

La possibilité de blocage d'un dispositif est donc dûe à une contention sur les ressources de communications. Si un cycle existe entre les ressources, alors il s'ensuit un interblocage.

Réseau de communication acyclique

Nous nous sommes refusés l'utilisation de files d'attente au sein des Unités de Routage, nous ne pouvons donc garantir la consommation des messages, c'est-à-dire le non blocage d'une unité. Par conséquent, il nous faut nous efforcer de rompre les cycles de dépendance. Plusieurs solutions ont été proposées dans ce but, le principe commun étant de définir un ordre, au moins partiel, sur les ressources de communications, c'est-à-dire que certains chemins ne peuvent être empruntés que sous certaines conditions:

Algorithme de routage e-cube

Sullivan et Brashkow ont proposé un algorithme de routage permettant de garantir l'absence de cycle de dépendance [Sull77] : le déplacement est effectué suivant un ordre décroissant de dimension. Ainsi sur un réseau 2D, le message se déplacera tout d'abord en Y, puis en X. Cet algorithme a été défini à l'origine pour un n-cube binaire (hypercube de dimension n, avec 2 noeuds par dimension).

Cet algorithme contraint le chemin suivi par chaque message, chemin qui est loin de la ligne droite. Il n'est donc pas utilisable dans notre cas.

Structured Buffer Pool

On suppose ici que chaque Unité de Routage dispose d'un certain nombre de buffers de messages. Un message ne peut être transmis à la cellule suivante que si elle dispose d'un certain nombre de buffers libres. Toueg et Ullman [Toue81] proposent un ensemble de critères quant au nombre de buffers libres nécessaires pour assurer le non blocage. On peut citer par exemple le *backward-count controler*, qui n'autorise une transmission que si le nombre de buffers non libres est inférieur à la distance déjà parcourue par le message.

Ces techniques supposent, de plus, qu'un message arrivé à destination est obligatoirement consommé.

Les contraintes que nous nous sommes posées ne permettent pas l'utilisation de ce type d'algorithme.

Canaux virtuels (Virtual Channels)

La méthode proposée par Dally et Seitz consiste à rompre un cycle de dépendance par utilisation de plusieurs canaux virtuels sur un même lien de communication, de telle manière qu'il ne puisse exister de cycle entre canaux virtuels [Dall87].

L'exemple qu'ils donnent pour expliquer le principe de leur méthode est reproduit en Figure 5.18. Les noeuds, ainsi que les liens de communications, sont numérotés (Figure 5.18.a). Les dépendances entre les différents liens sont représentées en Figure 5.18.b.

Pour *casser* le cycle, on définit un ensemble de canaux virtuels *hauts* (c1x), et un autre ensemble de canaux virtuels *bas* (c0x). Un canal haut est emprunté lorsque le numéro de la cellule courante est inférieur au numéro de la cellule destinataire (inversement pour un canal bas). Un ordre est ainsi défini entre ces canaux: c13 > c12 > c11 > c10 > c03 > c02 > c01, ce qui garantit l'absence de cycle.

Cette méthode est applicable à divers types de réseau, et en particulier à un réseau 2D. Cependant, elle suppose qu'un message arrivant à destination est consommé, ce qui n'est pas le cas avec nos contraintes. Supposons que l'ensemble des canaux soit occupé par un message. Alors le message sur le canal c01 ne peut être consommé par le noeud n0, car le canal c10 est occupé, ce qui bloque le noeud n0. Tous les canaux sont donc bloqués.

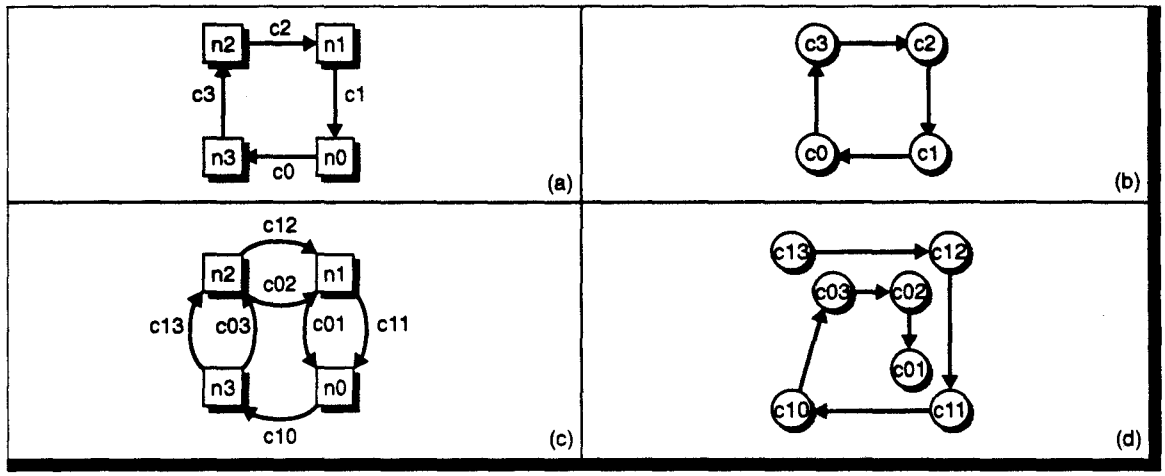


Figure 5.18 ***Principe des canaux virtuels.*** *(a) - Réseau classique. (b) - Cycle de dépendance associé. (c) - Réseau avec canaux virtuels. (d) - Dépendance sans cycle associée.*

Routage adaptatif

Pour remédier au problème de la non-disponibilité d'un organe de routage, il est également possible de mettre en oeuvre un mécanisme de routage adaptatif [Ngai89][Lind91]. Le message est alors détourné de son chemin original pour contourner la cellule bloquée ou défaillante.

Ce type de fonctionnement n'est pas possible dans notre cas; il est en effet indispensable de traverser chacune des cellules le long du chemin du rayon afin de tester la présence d'un voxel occupé.

En résumé

Aucun des dispositifs mis en place dans les cas classiques n'est ici utilisable, principalement de par le fait que nous désirons éviter la mise en place de buffers de messages (qui devraient être de taille très importante pour éviter dans tous les cas les phénomènes d'interblocages).

La transmission en ligne droite est également une contrainte forte. Elle a été mise en oeuvre sur un hypercube [Bado91b], mais en utilisant une communication par *Virtual Cut-Through Buffers* [Kerm79], qui est une technique du type *wormhole* avec bufferisation d'un message complet en cas d'indisponibilité d'une ressource, ce qui ne convient pas non plus à notre application.

Il ne semble donc pas possible de définir un schéma de communication permettant de garantir le non-blocage.

5.5.3 Destruction des rayons bloquants

S'il n'est pas possible d'envisager de conserver tous les messages en circulation tout en évitant les dead-locks, une des seules solutions restant à notre disposition est la suppression des messages susceptibles de créer les blocages. Nous définirons plus loin le choix de ces messages, et nous commencerons par détailler le processus de suppression.

Suppression d'un message

Un rayon ne peut être simplement détruit au risque d'obtenir une erreur d'évaluation de la couleur à afficher en un pixel. Si l'on retire un message du réseau, il est par conséquent nécessaire de le réémettre ultérieurement.

Nous avons indiqué que lorsqu'une cellule a émis un rayon secondaire ou un rayon retour, elle ne conserve aucune information de celui-ci (c.f § 5.3.4 - p. 119). Le message supprimé ne peut donc être relancé depuis la cellule source du rayon.

La solution retenue est de rediriger le message vers la machine hôte à l'aide d'un routage non-bloquant (du type *e-cube* : acheminement horizontal jusqu'au pipeline vertical, puis vertical jusqu'au processeur hôte (c.f § 4.4.1 - p. 94)). Il est possible d'assurer la consommation du message par le hôte à l'aide d'une file d'attente suffisante, d'où le non-blocage. Le rayon est, *après un certain temps*, renvoyé dans le réseau jusqu'à la cellule où il était bloqué[1].

Détection d'un blocage

Comme nous l'avons indiqué en début de ce paragraphe, on peut distinguer deux cas d'interblocage : un cycle entre les Unités de Routage, un cycle mettant en jeu le transfert de message vers une Unité de Calcul.

Nous ne disposons d'aucun contrôle global sur les unités de communications, et il n'est donc pas possible de détecter ces cycles. La seule information dont nous pouvons disposer localement au sein d'une cellule est la non possibilité de transférer un message dans une boîte aux lettres de sortie, mais cela peut provenir d'un engorgement temporaire des unités de communications autour de la cellule.

La seule possibilité que nous avons est alors de présumer un *dead-lock* si un message n'a pas pu être routé après un certain nombre de tentatives, et de détruire le message dans ce cas. Le nombre d'essais de transfert avant suppression doit être suffisamment élevé pour éviter les destructions en cas de blocage temporaire, et suffisamment faible pour rompre rapidement les cycles réels. Cette valeur de compromis ne pourra être obtenue que par simulation.

Choix des messages à supprimer

Ce choix doit répondre à deux contraintes : limiter le nombre de messages supprimés pour obtenir un débit global correct du réseau de communication, assurer que le cycle potentiel est bien rompu.

Dépendance entre les Unités de Routage

Nous supposerons dans un premier temps qu'il n'existe dans le réseau aucun cycle de dépendance mettant en jeu une Unité de Calcul. Il n'existe alors que des dépendances sur les boîtes aux lettres situées entre les cellules (Figure 5.19).

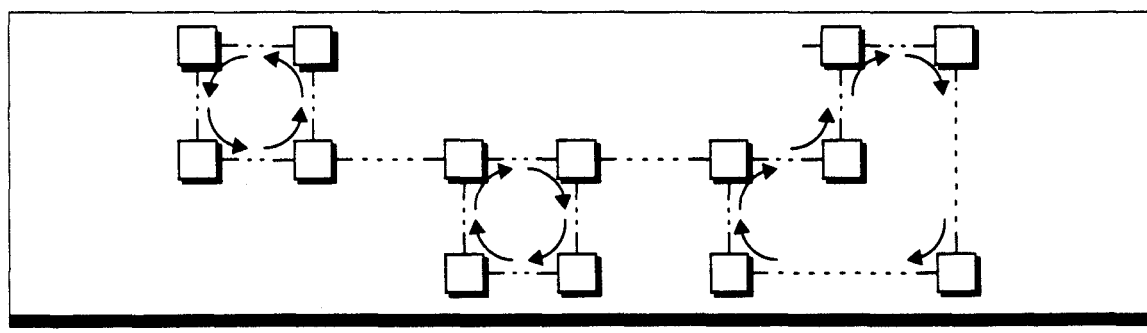


Figure 5.19 ***Cycles de dépendance entre Unités de Routage.***

1. Aucune étude n'a été faite pour définir la politique de replacement des messages dans le réseau.

Pour rompre un de ces cycles, il est suffisant de supprimer un des messages les composant, afin de créer un *trou* au sein de la dépendance. Le choix de ce message doit être fait localement par chacune des cellules, il ne peut donc être effectué qu'à partir des seules informations topographiques disponibles, à savoir la direction d'un message. Il existe, par exemple, toujours au moins un message provenant du *nord* ou du *sud* et en direction de l'*est*. Il peut certes y avoir un grand nombre de messages de ce type sur des cycles très complexes, mais il est difficile de mettre au point un choix simple permettant un meilleur filtrage, de par la nécessité de n'utiliser que les informations locales. Nous nous contenterons donc de celui que nous venons de proposer (Figure 5.20).

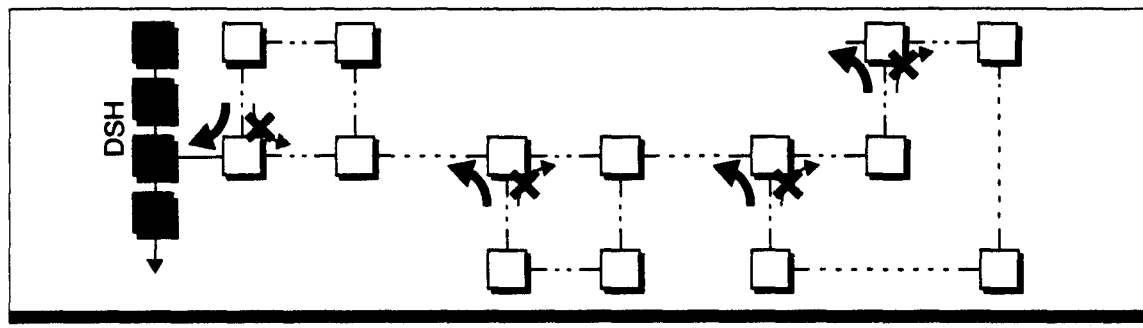


Figure 5.20 ***Ouverture des dépendances entre Unités de Routage.***

Si un message redirigé ne peut être acheminé vers le pipeline vertical, c'est qu'il existe un blocage sur le chemin qu'il doit emprunter. Si ce blocage provient de l'existence d'un cycle de dépendance, ce dernier finira par être rompu. Comme on assure l'acheminement des messages situés au bord du pipeline vertical (Figure 5.20), on peut également assurer, de proche en proche, le routage des messages supprimés.

Dépendance incluant une Unité de Calcul

Dans un cycle incluant une Unité de Calcul, il peut ne pas exister de messages *supprimables* tels que définis précédemment (Figure 5.21.a). Dans le cas contraire, la méthode énoncée ci-dessus permet de rompre le cycle de dépendance.

Nous allons définir un choix de message à supprimer équivalent au précédent, afin de garantir de la même manière les contraintes que nous avons imposées :

> Si un message provenant du *nord* ou du *sud* à destination de l'Unité de Calcul est bloqué, et si le message en sortie de l'Unité de Calcul doit être routé vers l'*est*, alors ce dernier est supprimé (i.e. redirigé vers le pipeline vertical puis vers le processeur hôte) (Figure 5.21.b).

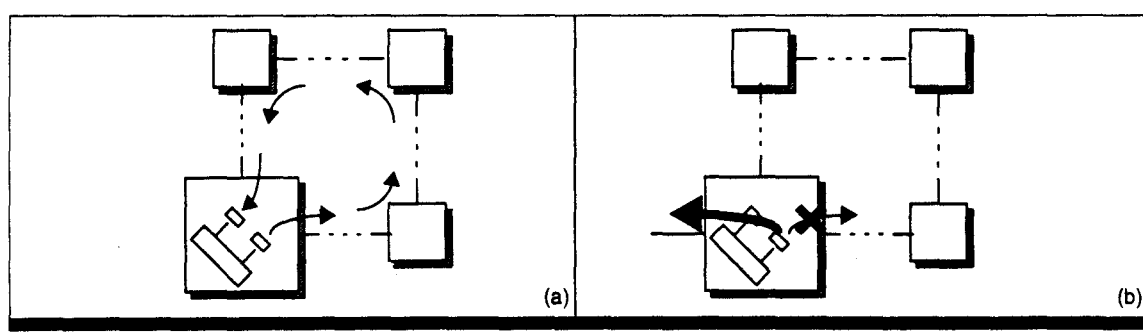


Figure 5.21 ***Dépendance incluant une Unité de Calcul.*** *(a) - Il n'existe pas de message supprimable en entrée de cellule. (b) - Suppression du message en sortie de l'Unité de Calcul.*

La libération de la boîte aux lettres en sortie de l'Unité de Calcul permet à celle-ci de pouvoir consommer le message qui est à son entrée, ce qui enclenche un transfert possible de tous les messages du cycle.

5.6 En résumé

En résumé, la phase de rendu est séparée en deux étapes :

- *Précalcul des éclairements locaux, avec détermination des ombres portées*
- *Lancer de Rayon Discret*

La cellule est composée de deux unités internes : l'Unité de Routage, et l'Unité de Calcul. Le contrôle est du type MIMD, les commandes de lancement des traitements étant transmises en mode multipipeline.

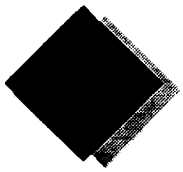
Les communications entre cellules sont asynchrones par message et utilisent un dispositif de Boîte aux Lettres gérées par l'Unité de Routage.

La répartition des traitements sur les deux unités internes est la suivante :

- *Unité de Calcul :*
  - *précalcul de l'éclairement local*
  - *génération des rayons primaires*
  - *vérification d'intersection*
  - *fin du calcul de l'éclairement local*
  - *émission des rayons retours*
  - *génération des rayons secondaires*
- *Unité de Routage :*
  - *transmission des commandes de lancement de traitement*
  - *suivi incrémental des rayons*
  - *détermination d'intersection*
  - *propagation des rayons retours*

Pour chacune de ces étapes l'organisation 3D ne présente, a priori, aucun avantage à nombre de processeurs constants. Sa réalisation étant plus complexe techniquement, nous restreignons nos études à un réseau 2D.

CHAPITRE



Processus de Précalcul de l'Eclairement Local

Un des avantages à l'utilisation d'un cube voxel est la possibilité d'y stocker certaines informations. Dans les approches classiques du Lancer de Rayon, l'éclairement local en un point d'un objet est recalculé à chaque fois que ce point est atteint par un rayon. Ces traitements inutiles peuvent être évités avec l'approche discrète. Nous proposons en effet d'effectuer un calcul de l'éclairement local sur l'ensemble des voxels, avant de déclencher la phase de Lancer de Rayon.

Le calcul de l'intensité lumineuse en un point d'un objet nécessite la connaissance du vecteur normal à la surface de cet objet. Dans un premier paragraphe, nous proposerons quelques méthodes permettant de calculer ce vecteur.

L'éclairement local est dû à la réception d'énergie depuis les sources lumineuses. Il faut donc pouvoir déterminer si un point est éclairé ou non par une source lumineuse. Ce traitement est effectué par un lancer de rayons d'ombrage, que nous spécifierons dans le deuxième paragraphe.

Nous terminerons par une évaluation du calcul de l'éclairement, en proposant l'utilisation de certains opérateurs câblés. Ces unités ont été implémentées à l'aide du logiciel de CAO VLSI Solo1400, en technologie CMOS 1,2µm.

6.1 Calcul du vecteur normal en chaque voxel

Il s'agit de stocker en chaque voxel occupé la valeur du vecteur normal à la surface. De la même manière que pour le cas des afficheurs classiques, 2 méthodes de calcul de cette valeur sont utilisables, suivant le type d'objet utilisé :

- pour les objets facettisés, on effectue une interpolation des normales calculées en chaque sommet (interpolation de Phong),
- pour les objets définis directement par leur équation, on effectue un calcul de la valeur de la normale en chaque point de discrètisation.

Il faut cependant étendre ces méthodes à l'espace 3D.

Interpolation sur objets voxelisés

Classiquement l'interpolation des composantes de la normale est effectuée sur la projection écran de chaque facette. Or dans le cas 3D, il peut exister plusieurs voxels en profondeur (paliers en Z). Il est bien entendu inconcevable d'appliquer la même valeur en chaque voxel d'un palier. Il faut par conséquent tenir compte de la profondeur lors de l'interpolation (Figure 6.1).

On cherche donc à exprimer chaque composante de la normale sous la forme d'une expression trilinéaire : $N_i = a \cdot x + b \cdot y + c \cdot z + d$. Or l'interpolation est effectuée sur un plan, ce qui ne permet de définir qu'une expression bilinéaire (z est lié à (x, y)). Ceci pose un problème que nous ne pouvons actuellement résoudre formellement.

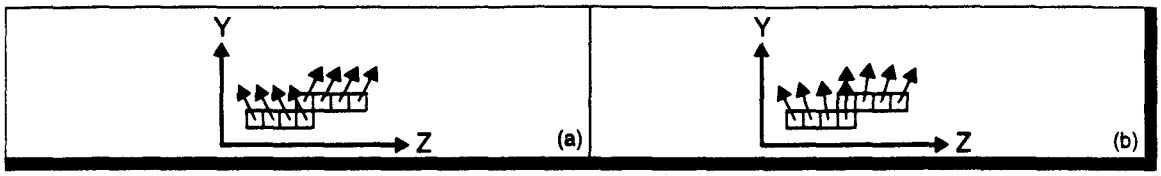


Figure 6.1 ***Interpolation sur objets voxelisés.*** *Vue dans le plan YZ. (a) - Sans interpolation en Z. (b) - Interpolation désirée.*

Nous proposons donc d'utiliser l'interpolation bilinéaire la meilleure possible. Pour cela, il suffit de calculer l'expression dans le plan de plus grande pente de la facette. En effet, dans ce plan il n'existe qu'un seul voxel en *profondeur*, sauf lors de l'application d'un algorithme de conversion 6-connexe, où il peut exister 2 voxels pour une position (u, v) donnée. Dans ce dernier cas, on effectue une recopie de la valeur de normale. Cela introduit certes une erreur de direction de normale, mais elle est minime tant que la variation de normale au sein de la facette est faible. Aucun critère quant à cette variation n'a été pour l'instant établi. L'erreur visuelle introduite n'a pas encore été observée, le logiciel de lancer de rayon discret étant encore en phase de développement. Par contre l'utilisation de cette méthode pour un ombrage de Gouraud (interpolation de couleur) ne montre aucun défaut visuel. Si cette erreur était trop importante, nous pouvons proposer une méthode d'interpolation différente, qui n'a pas encore été formalisée entièrement :

> Prenons l'exemple d'une interpolation de valeur le long d'une droite 2D (Figure 6.2). L'interpolation classique définit la valeur à calculer en fonction de la différence de coordonnées entre la position courante et la position de l'extrémité de la droite : $V(x) = a \cdot (x - x_0) + V(x_0)$. Dans ce cas, il y a identité de valeur pour deux points situés à la même coordonnée x. Une autre interpolation plus juste est l'interpolation par distance : $V(t) = a \cdot t + V(0)$, avec t la distance entre le point courant et l'extrémité de la droite. On peut encore améliorer cette définition en utilisant la distance entre la projection perpendiculaire du point courant sur la droite et l'extrémité de la droite. On obtient ainsi la valeur du point de la droite le plus proche du centre du voxel courant. Cette dernière possibilité offre de plus l'avantage d'être indépendante de la direction principale d'interpolation.
>
> Il reste à définir complètement ces méthodes pour les interpolations sur des facettes voxelisées.

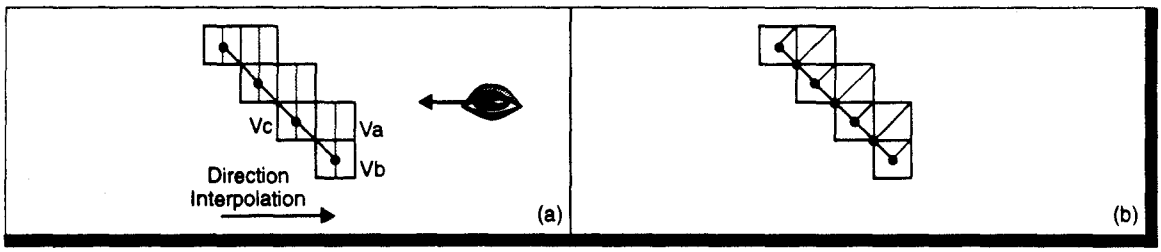


Figure 6.2 ***Echantillons utilisés par l'interpolation sur une droite 2D.*** *(a) - Interpolation classique, les voxels ayant même abscisse ont même valeur. (b) - Utilisation de la distance sur projection perpendiculaire, chaque voxel à une valeur propre, pouvant être considérée comme exacte.*

L'exemple de la droite 2D que nous venons de prendre nous permet de montrer plus précisément l'erreur commise par recopie de valeur sur les voxels de même position d'interpolation (i.e. les voxels de blindage de surface). Supposons que l'interpolation est effectuée dans la direction X comme sur la Figure 6.2.a. Dans ce cas, le voxel Va prend la même valeur de normale que le voxel Vb. Vue depuis la direction indiquée sur cette figure, le voxel Va est affichée devant le voxel Vc, et la valeur de normale récupérée est celle de Vb alors qu'elle devrait être celle de Vc. Par conséquent, suivant la direction de projection, les valeurs obtenues sont correctes, ou complètement erronée. La méthode utilisée dans la Figure 6.2.b nous semble de ce fait être le meilleur compromis possible, afin d'assurer une erreur minimale sur l'ensemble des directions de projection.

Les composantes de la normale étant définies par des expressions linéaires du premier degré, il est possible d'utiliser une version incrémentale des expressions, Mais dans ce cas, le calcul doit être effectué sur la totalité de la facette, c'est-à-dire lors de la voxelisation de celle-ci. La mise en oeuvre de l'interpolation diffère suivant le type de l'algorithme de conversion.

- Suivi de contour, ou balayage plan quelconque : la conversion est effectué dans le plan de plus grande pente. On calcule alors $N_i = a_u \cdot u + b_u \cdot v + c$.
- Balayage diophantien : On utilise une *pseudo*-expression trilinéaire $N_i = a \cdot x + b \cdot y + c \cdot z + d$, dans laquelle seuls les coefficients correspondants au plan de plus grande pente sont différents de 0. Ainsi, si l'interpolation est effectuée dans le plan YZ, on prend $a = 0$.
- Conversion par construction : On pourrait utiliser la technique précédente, l'éventuelle interpolation en z étant effectuée lors du *remplissage* des paliers. Mais il est alors nécessaire de recalculer la valeur N_i de chaque extrémité de palier. S'il est possible de déterminer l'incrément de valeur permettant ce calcul (Figure 6.3.a), il faut tenir compte de l'existence de paliers de demi-longueur aux extrémités des facettes (Figure 6.3.b). Ces paliers particuliers sont créés par le découpage du plan support de la facette. Il faudra interpoler les composantes de normales depuis l'extrémité réelle du palier. Cela induit un surcoût qui peut être non négligeable sur des facettes possédant des paliers de grandes longueurs.

L'interpolation s'intègre dans tous les cas parfaitement aux algorithmes de conversion. Le nombre de cycles de conversion est augmenté du calcul incrémental des trois composantes du vecteur normal (~10 cycles), ce qui amène une baisse de performances de conversion de l'ordre de 10%.

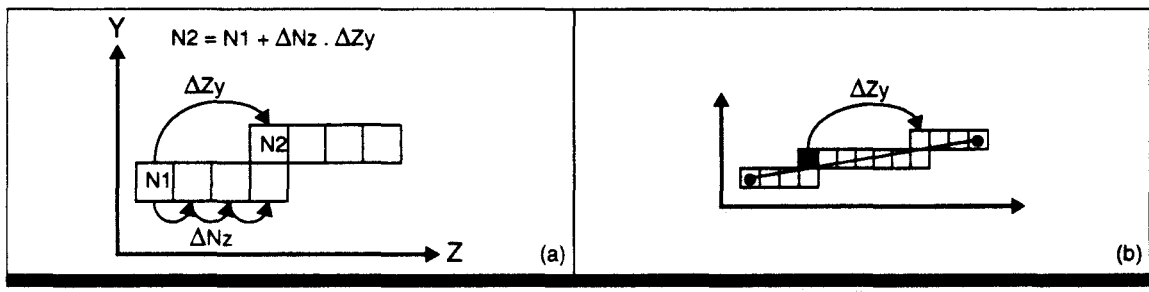


Figure 6.3 ***Incrément de valeur sur changement de palier.*** *(a) - Principe : l'incrément de valeur en z est combiné avec l'incrément de profondeur définissant les débuts de palier. (b) - Erreur de calcul de la valeur sur le voxel grisé, causée par la demi-longueur du premier palier.*

Calcul direct sur objets définis par équation

Un objet *explicite* est défini par l'équation de sa surface : $S(x, y, z) = 0$. La normale en tout point d'un tel objet est définie par les dérivées partielles de l'équation de surface : $N_i = \frac{\partial}{\partial i} S(x, y, z)$.

Pour des objets de complexité faible, comme les quadriques, l'expression de la normale est une fonction linéaire du premier degré et peut donc être calculée en incrémental, ce qui permettrait d'intégrer facilement l'évaluation des normales dans la phase de conversion, comme évoquée précédemment.

Il faut cependant remarquer que les normales sont calculées en des points discrets situés en dehors de la surface réelle de l'objet (en particulier lorsque la voxelisation est effectuée en connexité 6). On peut montrer que dans ce cas la normale obtenue correspond à la valeur de la normale du point de projection sur la surface, dans la direction du *centre* de la quadrique. Or ce point de projection n'est pas situé dans le même plan que le voxel considéré, ce qui induit une direction de normale différente de celle attendue (Figure 6.4).

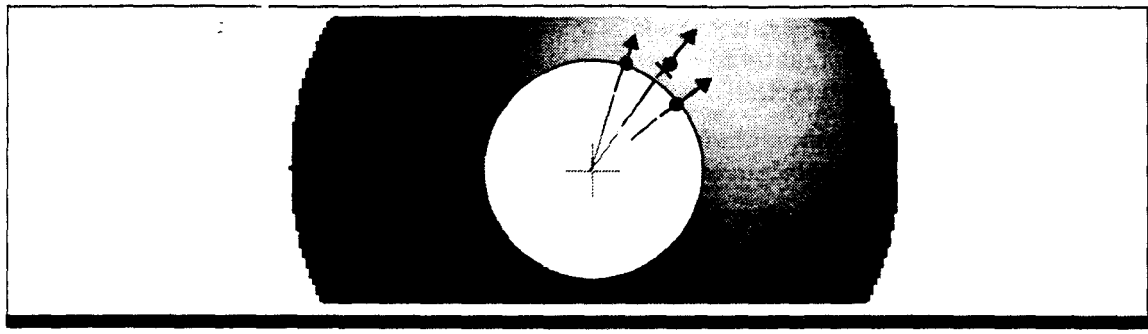


Figure 6.4 ***Erreur de calcul du vecteur normal.*** *Les points représentent le centre de voxel où est appliquée l'expression définissant les composantes de la normale.*

Cette erreur est en fait commise en chaque voxel, puisque peu d'entre eux appartiennent réellement à la surface géométrique. Plus l'écart entre le centre du voxel et l'objet réel est important, plus la direction de la normale sera faussée. Cela pourrait être particulièrement perceptible sur les voxels utilisés pour le blindage de surface (i.e en connexité 6). Là encore, il nous faut attendre les résultats du logiciel de Lancer de Rayon Discret pour décider de la nécessité ou non de l'amélioration du calcul de normale. Il pourrait être préférable de prendre en compte la normale au point le plus proche appartenant à la surface, comme nous l'avons présenté dans le petit exemple précédent de la droite 2D.

En conclusion

Que ce soit pour des objets facettisés ou pour des objets géométriques plus complexes, il existe une méthode permettant le calcul incrémental des composantes du vecteur normal. Le calcul incrémental est identique à celui mis en oeuvre pour le calcul de la profondeur, ce qui permet d'intégrer l'évaluation du vecteur normal à la phase de conversion. Il n'y a donc aucune modification de fonctionnement global de la conversion, et les conclusions concernant l'efficacité des méthodes suivant le type de réseau restent inchangées. Il en résulte uniquement une augmentation du temps de cycle de base et donc une baisse légère des performances de conversion.

Cependant aucune méthode ne peut permettre un calcul exact de la normale, du fait même de l'utilisation de l'espace discret. Nous avons donc proposé une technique de calcul permettant de minimiser cette erreur.

6.2 Détermination de l'éclairement d'un voxel

Nous avons proposé d'effectuer une première phase de calcul de l'éclairement local des voxels (cf § 5.2.1 - p. 111). De plus, pour éviter la convergence vers les sources, nous proposons d'inverser le procédé et de lancer les rayons d'ombrage depuis les sources lumineuses vers le reste de l'espace voxel, à la manière dont une source diffuse réellement son énergie lumineuse dans toutes les directions [Vida92]. Dans ce cas, il est nécessaire d'émettre un rayon dans chacune des directions possibles, soit N_V^3 rayons. Si un voxel occupé est atteint par un rayon (avec vérification éventuelle de l'intersection, cf § 5.3.2 - p. 116), alors il est considéré comme éclairé, et les autres voxels situés derrière lui sont marqués comme étant à l'ombre. Cependant, parmi les N_V^3 rayons, seuls ceux à destination de voxels occupés ont un intérêt, mais il n'est pas possible de connaître la position de ceux-ci.

Vidal a proposé deux méthodes permettant d'assurer un balayage complet de l'espace voxel avec un minimum de rayons :

1. Un rayon partant de la source et atteignant l'un des voxels de bord du cube passe par un certains nombre d'autres voxels vers lesquels il est donc inutile d'émettre des rayons d'ombrage. De plus Vidal a montré qu'en ne lançant des rayons que vers les voxels de bord, on couvre totalement tout le cube voxel (Figure 6.5.a). On réduit ainsi le nombre de rayons d'ombrage à $\sim 6 \cdot N_V^2$.
2. Si l'on lance ces rayons dans $6 \cdot N_V^2$ directions *logiques* différentes, il n'existe partant d'une cellule qu'un petit nombre de directions *physiques* possibles (autant que de connexions entre une cellule et ses voisines). Vidal a alors proposé de regrouper en *faisceaux* les rayons empruntant la même connexion. Au fur et à mesure de la propagation, ces faisceaux sont subdivisés en fonction des directions prises (Figure 6.5.b).

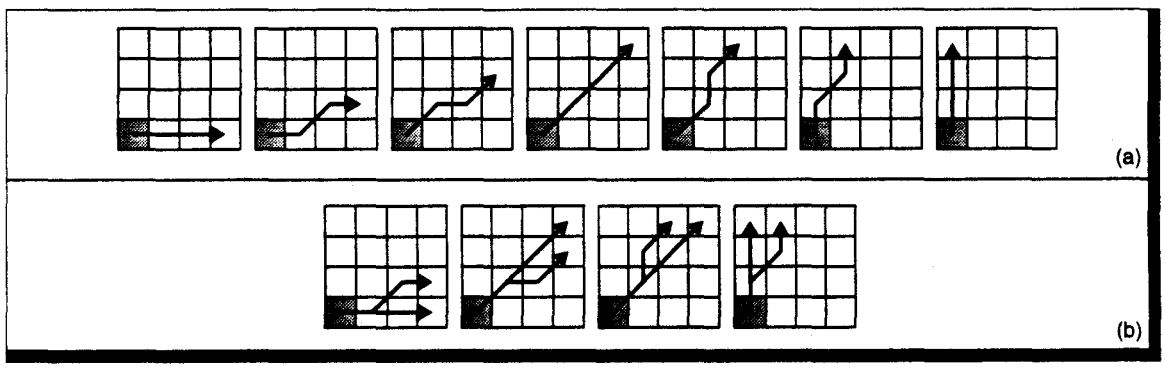


Figure 6.5 ***Détermination des ombres portées.*** *(a) - Par émission de rayons d'ombrage vers les voxels de bords (b) - Par émission de faisceaux d'ombrage.*

Dans les deux cas, on remarque qu'un même voxel peut être atteint par plusieurs rayons ou faisceaux différents. Comme l'indique Vidal, ce phénomène pourrait permettre de prendre en compte l'occultation partielle d'un voxel, par un autre objet se trouvant sur l'un seulement des rayons atteignant ce premier. La prise en compte de ce phénomène n'a pas encore été développée, et nous considérons pour l'instant qu'un voxel est éclairé s'il est atteint par au moins un rayon lumineux.

Dans les deux évaluations qui suivent, nous n'avons pas pris en compte le parallélisme entre l'Unité de Routage et l'Unité de Calcul. Il aurait en effet été plus difficile d'estimer les performances des deux méthodes dans ce cas. Nous supposerons donc qu'une cellule est uniquement constituée d'une seule unité assurant tous les traitements. Nous n'avons également pas considéré le dispositif de détection d'intersection. La comparaison entre les deux méthodes reste cependant valable, chacune d'entre elles profitant de la même manière du matériel qui est réellement employé.

6.2.1 Evaluation du lancer de rayons d'ombrage

Mise en oeuvre

La mise en oeuvre de cette méthode est relativement simple. Remarquons qu'il existe deux types de sources lumineuses : celles contenues dans le cube voxel, et les sources externes. Dans le premier cas, on émet un rayon vers chacun des voxels de bord, dans l'autre cas, on n'émet que vers les bords du cube situés à l'opposé de la source (Figure 6.6). La création des rayons doit donc être effectuée par la cellule disposant de la source lumineuse dans le premier cas, alors qu'ils sont créés par la machine hôte dans la seconde configuration.

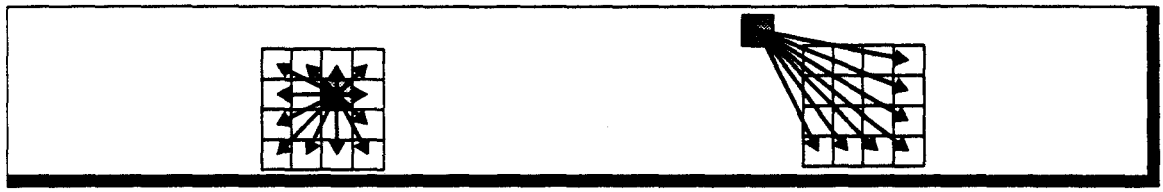


Figure 6.6 ***Emission des rayons d'ombrage.*** *Source interne, et Source externe.*

Nous avons indiqué que deux rayons différents peuvent à un instant donné passer par le même voxel. Il peut s'en suivre un conflit d'accès à ce voxel. Cependant ces deux rayons se propageant globalement dans la même direction (i.e du centre vers le bord), il ne peut exister d'interblocage des cellules, à condition de n'effectuer les émissions que depuis une source à la fois. Cela provoquera au plus un blocage temporaire d'un ensemble de cellules.

Afin d'éviter ce phénomène, il pourrait être possible d'ordonner les directions d'émission de telle manière qu'il ne puisse se produire de conflits (i.e, il faut éviter d'envoyer à la suite deux rayons à destination de voxels voisins). L'étude de telles méthodes et de leur efficacité est fortement tributaire du fonctionnement des organes de communications utilisés au sein du réseau. Là aussi, seule une simulation précise permettra de parvenir à une étude complète de ce traitement.

Evaluation du coût pour une source interne

Comme nous venons de l'évoquer, il est difficile d'évaluer correctement cette méthode sans tenir compte d'éventuels conflits de communications. Nous supposerons que ces conflits ont pu être résolus par émission ordonnée des rayons d'ombrage.

Le temps nécessaire à une émission complète est fonction principalement de la charge du processeur contenant la source, car c'est celui qui a à traiter le plus grand nombre de rayons. En plus de la génération des rayons d'ombrages, ce processeur doit effectuer le suivi de ces rayons au sein du sous-cube dont il a la charge. Le nombre de cycles nécessaire pour cette phase est donc fonction du type du réseau. Pour évaluer la valeur maximale du nombre de cycles, nous prendrons le rayon de longueur maximale (i.e un rayon traversant un sous-cube en diagonale, dont la longueur est $max(l, h, p)$ en connexité 6). D'où, en notant T_G le nombre de cycle pour générer un rayon et T_S le nombre de cycles pour effectuer le suivi du rayon :

$$T_{max} \approx 6 \cdot N_V^2 \times (T_G + max(l, h, p) \cdot T_S), \quad \text{soit } T = O\left(N_V^2\right) \quad \text{pour un réseau } 3D_{l,h,p}$$

$$T_{max} \approx 6 \cdot N_V^2 \times (T_G + N_V \cdot T_S), \quad \text{soit } T = O\left(N_V^3\right) \quad \text{pour un réseau } 2D_{l,h}$$

Le temps de suivi de rayon est important dans le cas des réseaux $2D$. De ce fait, si l'ordre de génération des rayons est quelconque, une cellule pourrait recevoir un rayon alors qu'elle est occupée à effectuer le suivi d'un autre rayon. Il faut donc veiller à émettre les rayons de telle manière que la longueur de traversée au sein des sous-cubes soit en ordre croissant. Dans ce cas un nouveau rayon atteignant une cellule à un temps de traversée égal ou plus long au précédent, et l'on évite ainsi tout blocage.

Si (i, j, k) est la position de la source, on générera les rayons de la manière suivante :

```
Lgr <- max(k, N_V - k)
z_1 <- k
z_2 <- k - 1
Pour n de (Lgr - 1) à 0
|  Si (z_1+n < N_V)
|  |  Générer rayons vers bords à prof. z_1+n
|  FinSi
|  Si (z_2-n ≥ 0)
|  |  Générer rayons vers bords à prof. z_2-n
|  FinSi
FinPour
```

Si les coûts que nous venons de définir sont réalistes pour le réseau $3D$, il n'en est pas de même pour l'autre organisation. En effet seuls les rayons qui se propageront uniquement dans la cellule contenant la source ont une longueur de N_V voxels. Cela représente un ou deux rayons pour un réseau $2D_{1,1}$. Pour spécifier plus précisément le coût réel du lancer des rayons d'ombrage, nous avons simulé un suivi incrémental 26-connexe sur ce réseau :

Pour $N_V = 512$, nous obtenons une longueur moyenne de palier de 1 voxel (Table 6.1).

Lgr	Nb Occur.	Lgr	Nb Occur.	Lgr	Nb Occur.	Lgr	Nb Occur.
1	1.403.910	7	1.208	13	200	32	80
2	122.368	8	1.024	15	184	43	64
3	22.304	9	304	16	160	64	40
4	8.160	10	544	19	144	128	24
5	3.432	11	240	22	120	255	2
6	1.888	12	224	26	104		

Table 6.1 ***Occurrence des paliers de différentes longueurs.*** *Pour un réseau $2D_{1,1}$ ($N_V = 512$) Le voxel source est situé au centre du cube.*

Nous donnons Figure 6.7 le nombre d'étapes de suivi de rayon nécessaire à une diffusion complète (i.e vers tous les voxels de bords). Le nombre de cycles que nous obtenons pour un réseau $2D_{1,1}$ est $\sim 7 \cdot N_V^2 \cdot T_S$ (le temps de génération n'est pas pris en compte).

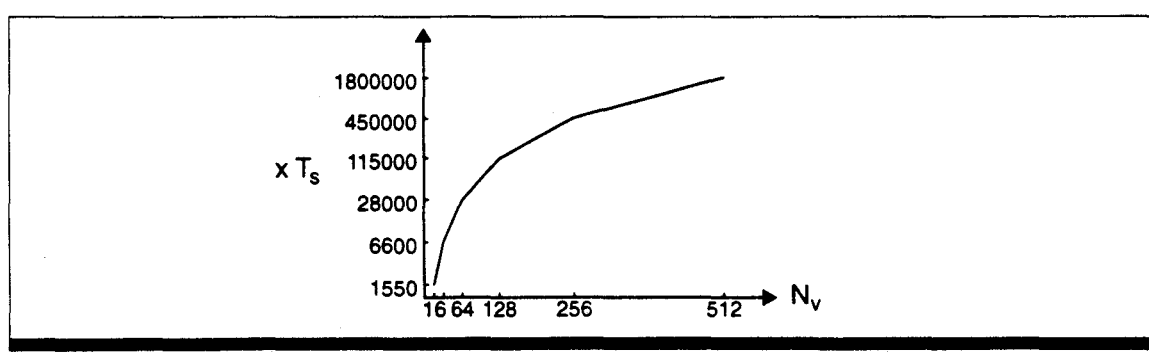


Figure 6.7 ***Etapes de suivi de rayons pour une diffusion complète.***

Fin de propagation des rayons d'ombrages

Au temps de génération des rayons d'ombrage, il est nécessaire d'ajouter le temps de propagation du dernier rayon envoyé dans le réseau :

- dans le cas $3D$, on obtient $T_{tot} \approx 6 \cdot N_V^2 \times (T_G + max(l, h, p) \cdot T_S) + N_V \cdot T_S = O\left(6 \cdot N_V^2 + N_V\right)$,
- dans le cas $2D$, les derniers rayons traités sont ceux qui se propagent au sein de la même cellule. On peut raisonnablement penser que pendant la propagation de ces deux rayons (durée : $\sim 2 \cdot N_V/2$) les rayons précédents auront terminés leur parcours (le plus long rayon dans le cube voxel étant lui-même de longueur N_V). D'où $T_{tot} \approx O\left(7 \cdot N_V^2\right)$

On remarque, à partir de ces estimations, que l'organisation $2D$ présente pratiquement les mêmes performances que l'organisation $3D$, car 90% des rayons d'ombrages ont des paliers de longueur 1 voxel. L'organisation $2D$ étant moins coûteuse matériellement, elle est donc la plus adaptée au lancer des rayons d'ombrages.

Source externe au cube

Le nombre de rayons passant par une cellule est fonction de la position de la source. Le nombre maximal est atteint lorsque la source est située juste sur un des bords du cube. Dans ce cas, la cellule voisine de la source recevra $\sim 5 \cdot N_V^2$ rayons d'ombrages. Les évaluations précédentes restent donc valables, à un facteur multiplicatif près.

6.2.2 Evaluation du lancer de faisceaux d'ombrages

Mise en oeuvre

Cette technique consiste, comme évoqué précédemment, à regrouper les rayons d'ombrages empruntant les mêmes cellules au sein de faisceaux [Vida92]. Lorsqu'un faisceau atteint une cellule, il est subdivisé et transmis aux cellules voisines (Figure 6.8).

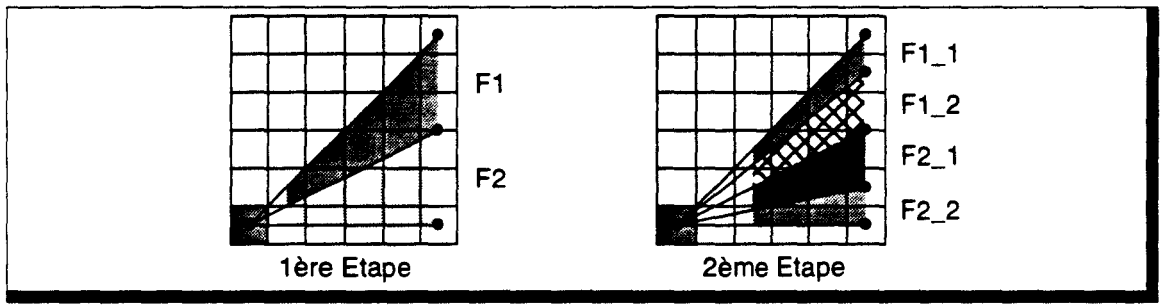


Figure 6.8 ***Subdivision des faisceaux d'ombrages.***

Un faisceau $2D$ est défini par les ordonnées (Y_{min}, Y_{max}) des deux voxels limites qu'il doit atteindre. La subdivision crée, éventuellement, deux nouveaux faisceaux (Y_{min}, Y_{med}) et (Y_{med}, Y_{max}). Cette valeur Y_{med} est calculée en considérant le rayon passant entre les deux cellules voisines à la cellule courante (Figure 6.9.a). Suivant la position de cette valeur par rapport au faisceau incident, on crée et émet vers les cellules voisines un ou plusieurs sous-faisceaux (Figure 6.9.b,c,d).

Si (x, y) est la position de la cellule courante, alors $Y_{med} = \dfrac{\left(y + \frac{1}{2}\right) \times (N_V - 1)}{x + 1}$.

Contrairement à la méthode précédente, l'utilisation de faisceaux d'ombrage peut créer des conflits d'accès à une cellule (Figure 6.10). Ces conflits étant inhérents à la méthode, nous ne pouvons définir un ordre d'émission pouvant les supprimer.

Vidal propose l'utilisation d'une file d'attente au sein de chaque cellule, dans laquelle sont stockés les faisceaux à subdiviser. Avant de traiter un faisceau, une cellule vérifie tout d'abord si ses voisines dans la direction de propagation sont prêtes à recevoir un faisceau, i.e. si leurs files d'attentes ne sont pas pleines. Dans le cas contraire, le faisceau est replacé dans la file d'attente.

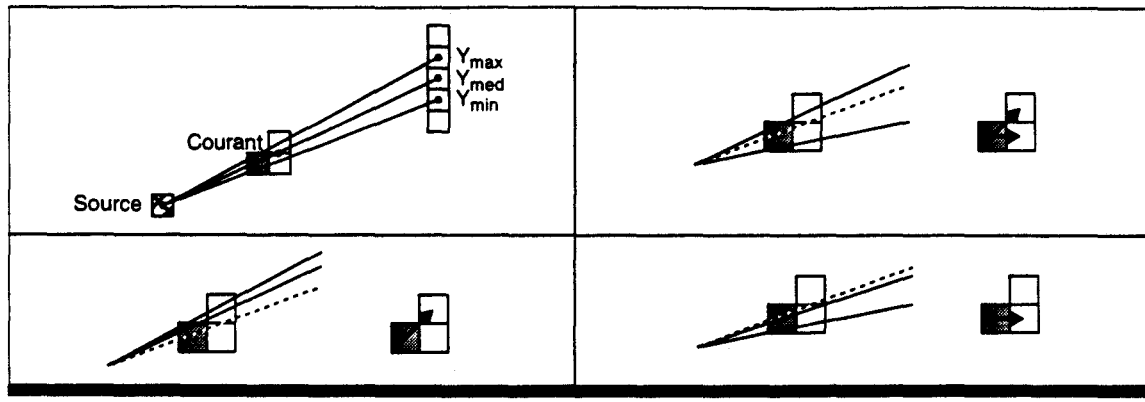


Figure 6.9 ***Principe de la subdivision.*** *(a) - Calcul de la valeur Y_{med}. (b),(c),(d) - Trois configurations possibles.*

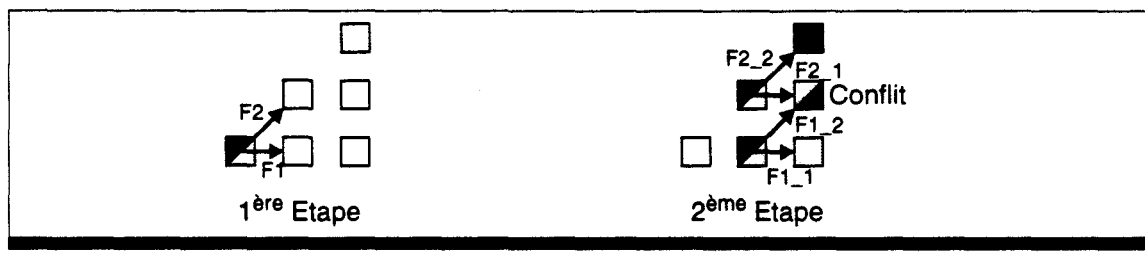


Figure 6.10 ***Conflit d'accès inhérent à la subdivision de faisceaux.***

Extension à l'espace $3D$

Pour une source de position quelconque, on lancera un faisceau vers chacune des faces du cube voxel, soit 6 faisceaux de direction de propagation différente. Chaque faisceau est représenté par 2 couples de coordonnées (coin haut-gauche et coin bas-droit) et est susceptible de se subdiviser en un maximum de 9 sous-faisceaux.

Cette méthode est par conséquent très coûteuse en temps de calcul (4 divisions, 2 tri de valeurs par faisceau), ainsi qu'en matériel (file d'attente et gestion associée).

Evaluation du coût

Depuis la cellule source, au plus 6 faisceaux sont émis. Le temps de génération est donc constant et faible, seul le temps de propagation d'un faisceau au sein du réseau est à prendre en compte.

Théoriquement, le temps du lancer de faisceaux est donc de : $T_{tot} = 6 \cdot (T_G + T_S) + N_V \cdot T_S$.

Cependant le nombre important de conflit d'accès vient augmenter la complexité réelle de cette méthode. Les simulations effectuées par B.Vidal, pour un réseau $2D_{1,1}$ indiquent que le temps réel de diffusion est de l'ordre de $N_V^2 \cdot T_S$ pour des files d'attente de longueur 1, et de l'ordre de $N_V \cdot \log N_V \cdot T_S$, pour des files d'attente de longueur $N_V/2$.

6.2.3 Comparaison des deux méthodes

En dehors d'une réelle simulation de ces deux méthodes, nous ne pouvons juger que des performances théoriques des deux méthodes de détermination des ombres portées.

En notant que le temps de cycle est bien plus important pour le traitement d'un faisceau que pour le suivi d'un rayon (d'un facteur > 10), la méthode du lancer de faisceaux ne peut permettre d'atteindre un gain de performances suffisant pour justifier l'augmentation de coût matériel qu'elle nécessite. Elle ne peut avoir un intérêt que si la subdivision est câblée et

parallélisée. Cependant les opérateurs à utiliser dans ce cas, ne seront plus utilisés par ailleurs.

La méthode de lancer de rayons d'ombrage semble donc bien plus adaptée à un réseau massivement parallèle, où il faut chercher à diminuer le coût matériel de chaque cellule.

6.2.4 Rayons ou faisceaux d'ombrage sur réseau partiel

Dans les études des paragraphes précédents, nous avons supposé que le réseau voxel était complet. Aucune étude précise n'a encore été menée sur l'utilisation d'un réseau partiel dans le cadre du lancer de rayon, qu'il s'agisse de rayons d'ombrages ou autres. Il nous faut tout d'abord terminer l'étude sur réseau complet avant de passer à l'implémentation sur réseau partiel.

Cependant nous pouvons déjà soulever un certain nombre de problèmes. En effet, lors d'une phase de lancer de rayon, la totalité de l'espace voxel est utilisée, traversée par un grand nombre de rayons ou faisceaux. L'espace voxel étant *plié* sur le réseau de cellules, un rayon arrivant sur le *bord* du réseau est réinjecté sur le *bord* opposé du réseau. Il est donc très difficile d'éviter les conflits d'accès aux cellules. Il nous semble qu'aucun ordre particulier d'émission des rayons d'ombrages ne permettra d'éviter ces conflits.

Une des solutions envisageables est l'utilisation de files d'attente, qui comme nous l'avons évoqué est assez coûteuse matériellement, d'autant plus que le nombre de conflits peut être très élevé dans le cas présent.

D'autre part, il existe un risque important de phénomène de *dead-lock*, inexistant sur le réseau complet grâce à la direction de propagation unique. On peut dire que la diffusion depuis la source ne crée pas de *croisement* de rayons, tant qu'une seule source à la fois est utilisée. Or le réseau partiel va créer de tels croisements.

Le réseau partiel présente donc un avantage indéniable pour la phase de conversion, mais augmente considérablement les problèmes sur les phases de lancer de rayons. Cette solution est donc *mise de côté*, jusqu'à ce que le réseau complet soit entièrement étudié.

6.3 Calcul de l'éclairement local

L'éclairement local d'un objet est composé de trois intensités :

$$I_{locale}(\lambda) = I_{amb}(\lambda) + \sum_{sources} (I_{diff}(\lambda) + I_{spec}(\lambda))$$

avec
$$I_{amb}(\lambda) = k_a \times C_\lambda ,$$
$$I_{diff}(\lambda) = k_d \times C_\lambda \times I_{src}(\lambda) \times (\vec{N} \bullet \vec{S})$$
$$I_{spec}(\lambda) = k_{s,\lambda} \times I_{src}(\lambda) \times (\vec{R} \bullet \vec{O})^{n_s} = K_{spec,\lambda} \times (\vec{R} \bullet \vec{O})^{n_s}$$

Les différents coefficients constants k_x, et C_λ définissent les propriétés photométriques de l'objet. Ils doivent être chargés en fin de phase de conversion, lors du remplissage des paliers.

Restent donc à calculer les différents vecteurs nécessaires pour élaborer la valeur de l'éclairement local. Parmi ceux-ci, le vecteur normal $\vec{N}$, le vecteur source $\vec{S}$ et le vecteur réfléchi $\vec{R}$ sont indépendants de la position de l'observateur, ou de la direction d'un rayon incident. En conséquence, l'éclairement ambiant et diffus peut être précalculé, une fois terminée la phase de lancer des rayons d'ombrage.

On différenciera donc 2 parties de calcul de l'intensité locale :

Phase de Précalcul

```
Pour chaque voxel occupé
|  Calcul I_amb
|  Calcul S
|  Normalisation N, S
|  Calcul R
|  Calcul I_diff + I_amb
|  Calcul K_spec,λ
FinPour
```

Phase de L.R.D

```
Si intersection
|  Calcul O
|  Calcul I_spec
|  Calcul I_locale
FinSi
```

6.3.1 Calcul des vecteurs $\vec{N}$, $\vec{S}$ et $\vec{R}$

Lors de la voxelisation des objets, une valeur de normale a été chargée dans chaque voxel occupé. Cependant cette valeur n'est pas normalisée. La normalisation d'un vecteur nécessite le calcul de la norme de celui-ci ($\|M\| = \sqrt{n_x^2 + n_y^2 + n_z^2}$), puis la division de chaque composante du vecteur par cette norme. Nous étudierons plus précisément ce calcul dans le paragraphe suivant.

Le vecteur réfléchi est défini par $\vec{R} = 2\vec{N}(\vec{N} \bullet \vec{S}) - \vec{S}$, et on peut montrer facilement que si $\|\vec{N}\| = \|\vec{S}\| = 1$, alors $\|\vec{R}\| = 1$. Il n'est donc pas nécessaire de normaliser ce dernier.

Notons que la norme au carré de $\vec{N}$ (i.e. $n_x^2 + n_y^2 + n_z^2$) est calculable incrémentalement; cette valeur pourrait donc être interpolée lors de la phase de voxelisation. Cependant, le fonctionnement multipipeline de la voxelisation induit que toute augmentation du temps de cycle de conversion ralentit globalement le débit :

> Supposons que pour chaque objet à voxeliser, il existe un ensemble de p cellules au sein desquelles l'objet présente un palier de longueur $2 \cdot v$ voxels, alors qu'il est de longueur v dans les autres cellules. Supposons également que les cellules où se trouvent ces paliers longs sont différentes pour chaque objet.
>
> Le débit global est dans ce cas imposé par les cellules les plus lentes. Il est donc en $O(2 \cdot v)$. Si n objets sont chargés dans le réseau, le temps global est donc en $O(2 \cdot v \cdot n)$.

Par contre la phase de précalcul de l'éclairement est indépendante du nombre d'objets, puisque fonction uniquement du nombre de voxels occupés au sein de chaque cellule :

> Si les n objets précédents ne se recouvrent pas, alors chaque cellule contient $(n-1) \cdot v + 2 \cdot v$ voxels. Le temps de calcul des normes au carré est alors en $O((n+1) \cdot v)$.

Cette évaluation est certes peu réaliste, mais exprime bien l'avantage à ne pas interpoler la valeur de la norme au carré lors de la phase de conversion.

Si $[S_x\ S_y\ S_z]$ est la position de la source courante, et $[i\ j\ k]$ la position du voxel courant, alors le vecteur source (non normalisé) est $[(S_x - i)\ (S_y - j)\ (S_z - k)]$.

Il serait possible de calculer incrémentalement le vecteur $\vec{S}$ lors de la propagation du rayon d'ombrage, en décrémentant les coordonnées de la position de la source lors de chaque changement de voxel. Cependant, comme dans le cas précédent, on augmente le temps de cycle du suivi de rayon du temps correspondant à trois décrémentations, soit $3 \cdot n$ opérations supplémentaires à effectuer si n est la longueur du rayon. Il est donc également préférable de calculer complètement le vecteur source en phase de précalcul d'éclairement, où seules 3 opérations sont nécessaires.

6.3.2 Evaluation du calcul d'éclairement

Le calcul d'éclairement pourrait être exécuté durant le lancer des rayons d'ombrage, dès qu'un rayon lumineux atteint un objet. Cependant, comme nous le verrons, ce calcul nécessite un grand nombre d'opérations. Une cellule gérant plusieurs voxels, la propagation du rayon d'ombrage risque donc d'être bloquée durant tout le temps nécessaire au calcul de l'éclairement d'un voxel précédent. Le temps de cycle de la propagation d'un rayon sera par conséquent fortement allongé.

L'éclairement mettant en oeuvre des calculs complètement indépendants du reste des traitements et de l'environnement, il peut être implémenté en mode *SPMD*, sans synchronisation entre cellules, et nous le séparons de la phase de lancer des rayons d'ombrage.

De ce fait, le temps nécessaire au déroulement de cette phase est celui nécessaire à la cellule possédant le plus grand nombre de voxels pour réaliser l'ensemble des calculs. L'envoi de donnée au sein du réseau est effectué en mode multipipeline, il faut donc ajouter au temps de traitement, le temps nécessaire à la propagation de la commande de lancement de cette phase. Le majorant du temps de calcul est alors :

$$T_{3D} = (l_c + h_c + p_c)\, T_P + \alpha\,(l \cdot h \cdot p)\, T_{ppc} \qquad \text{pour un réseau } 3D_{l,h,p}$$

$$T_{2D} = (l_c + h_c)\, T_P + \alpha\,(l \cdot h \cdot N_V)\, T_{ppc} \qquad \text{pour un réseau } 2D_{l,h} \qquad \textbf{(Eq. 6.1)}$$

avec
- T_P : temps de propagation de la commande
- T_{ppc} : temps d'un calcul complet pour un voxel
- α : taux d'occupation du cube voxel
- l_c : le nombre de sous-cubes en largeur = $\lceil N_V / l \rceil$ (idem en hauteur et en profondeur)

A la vue des calculs nécessaires pour l'éclairement local, on peut considérer que le temps de propagation est faible, voire même négligeable. Tous les calculs sont indépendants et comme ils ne sont exécutés que sur les voxels occupés, on peut considérer que la fraction séquentielle du problème est pratiquement nulle et donc que l'efficacité est très proche de 1. Le gain est alors directement fonction du nombre de processeurs. Si on évalue les 3 organisations à nombre de cellules égal, on peut considérer que les trois types de réseau ont les mêmes performances.

6.3.3 Unités matérielles pour la normalisation

Calcul classique de la normalisation

Rappelons brièvement le calcul permettant la normalisation d'un vecteur quelconque.

Soit un vecteur $\vec{V}$, défini par ses trois composantes dans un repère normalisé : $\vec{V} = [V_x\ V_y\ V_z]$.

La norme de $\vec{V}$ est $\|\vec{V}\| = \sqrt{V_x^2 + V_y^2 + V_z^2}$.

Le vecteur normalisé $\vec{V}$ équivalent est alors défini par : $\vec{V} = \left[\frac{V_x}{\|\vec{V}\|}\ \frac{V_y}{\|\vec{V}\|}\ \frac{V_z}{\|\vec{V}\|}\right]$.

Trois opérations complexes sont donc nécessaires pour une normalisation : une mise au carré, une extraction de racine carrée et une division. Ce sont ces trois opérations dont nous voulons minimiser le coût.

Format des opérateurs

Nous avons précédemment indiqué que les données fournies en résultat de la voxelisation sont au format virgule fixe 20 bits (§ 4.4.4 - p. 103). Un opérateur de calcul de x^2 fournira donc une valeur sur 40 bits.

Il a été montré qu'une précision de 12 bits, au minimum, est nécessaire sur les vecteurs normaux, si l'on désire un calcul d'éclairement avec 8 bits de précision [Lefe94]. De ce fait, l'opérateur $\sqrt{x}$ doit fournir une valeur sur 12 bits, et donc accepter une entrée sur 24 bits.

Nous noterons P_{Pr} le produit *nombre_d'opérations* * *surface_silicium* d'une approche programmée, et P_{Ma} ce même produit dans le cas d'utilisation d'une unité matérielle. La comparaison entre ces différentes valeurs nous permettra de choisir entre les différentes possibilités (cf § 2.3 - p. 48).

Enfin, pour servir de point de comparaison, notons qu'un additionneur 10 bits à propagation de retenue est composé de ~350 transistors, pour une surface de 0,12mm^2.

Calcul de l'élévation au carré

Approche programmée

La multiplication série classique de 2 nombres sur n bits nécessite n additions et décalages successifs. Nous considérerons une approche complètement programmée, auquel cas le décalage et l'addition nécessitent deux opérations distinctes.

Nous désirons effectuer la multiplication de deux nombres de 20 bits avec résultat sur 40 bits, et une unité 10 bits. Dans ce cas, 4 opérations sont nécessaires pour chaque addition ou décalage, d'où un total de 160 opérations pour la multiplication, soit $P_{Pr} = 160 \times 0.12 = 19.2$.

PLA d'accélération du calcul

Il existe des réseaux permettant l'évaluation parallèle de la multiplication. Ces réseaux utilisent n^2 cellules de bases. Cependant l'élévation au carré est une multiplication particulière puisque les deux valeurs d'entrées sont égales. De ce fait, les expressions peuvent êtres simplifiées ce qui réduit le coût matériel d'une unité dédiée au calcul du carré.

Nous avons tenté la synthèse directe d'une PLA, mais il n'a pas été possible de créer un circuit pour une entrée sur plus de 8 bits. Un autre essai a consisté à scinder la PLA en deux circuits calculant chacun une partie du résultat. La surface totale est de ~0.6mm^2, pour un temps de cycle de 30ns.

Entre ces deux extrémités (évaluation série, réseau parallèle), nous proposons une unité de précalcul de l'élévation au carré. Prenons un exemple pour un mot de 4 bits (les chiffres représentent les différents bits de la donnée d'entrée; le point est l'opération *et logique)* :

						5	4	3	2	1	0
x						5	4	3	2	1	0
						5.0	4.0	3.0	2.0	1.0	0.0
+					5.1	4.1	3.1	2.1	1.1	0.1	.
+				5.2	4.2	3.2	2.2	1.2	0.2	.	.
+			5.3	4.3	3.3	2.3	1.3	0.3	.	.	.
+		5.4	4.4	3.4	2.4	1.4	0.4	.	.	.	.
+	5.5	4.5	3.5	2.5	1.5	0.5	.	.	.	.	.

Les 6 termes à additionner peuvent être réécrits différemment sous la forme :

	5.5	.	4.4	.	3.3	.	2.2	.	1.1	.	0.0
+	5.4	5.3	5.2	5.1	5.0	.	.	.	.	.	.
+	.	.	4.3	4.2	4.1	4.0	.	.	.	.	.
+	.	.	.	.	3.2	3.1	3.0	.	.	.	.
+	.	.	.	.	.	.	2.1	2.0	.	.	.
+	.	.	.	.	.	.	.	.	1.0	.	.

On peut alors préadditionner un certain nombre de termes, et les réordonner, pour aboutir à l'utilisation de 3 nombres à additionner (le point d'exclamation est l'opération *non logique*) :

	5.4	5.!4	4.3	4.!3	3.2	3.!2	2.1	2.!1	1.0	1.!0	.	0.0
+	.	.	5.3	5.2	5.1	5.0	4.0	3.0	2.0	1.0	.	.
+	.	.	.	.	4.2	4.1	3.1	2.1	.	.	.	.

Cette méthode est généralisable pour tout nombre sur $2n$ bits. Le nombre d'opérations à effectuer est réduit à n additions. Les termes à précalculer sont très simples et peuvent être fournis par une PLA. Pour une entrée sur 8 bits, la PLA générée à une surface de 0,18mm^2. Pour 12 bits, la surface est de 0,46mm^2.

Pour diminuer encore les calculs à effectuer, nous avons intégré l'addition 2 à 2 des expressions en interne dans la PLA. Nous fournissons, par exemple, la valeur $T_1 + T_4$ ou $T_2 + T_3$ pour une entrée sur 8 bits. Il reste alors une unique addition à effectuer. Cette PLA occupe une surface de 0,28mm^2. La même intégration des additions n'a pas pu être réalisée pour une donnée 12 bits, le nombre de termes produits dépassant dans ce cas les capacités du synthétiseur de PLA du logiciel Solo1400. Or, nous avons besoin d'une unité acceptant 20 bits en entrée. Au vue des résultats précédents, cet opérateur sera de taille trop important, pour être intéressant.

Par contre, nous pouvons scinder le calcul de x^2 en plusieurs étapes en décomposant la donnée d'entrée en deux mots de 10 bits. Nous utilisons alors la PLA comme opérateur d'accélération du calcul de la puissance. Dans ce cas, le calcul complet comprend : 4 mises au carré partielles, 4 additions 10 bits, et 3 additions 40 bits, soit un total de 20 opérations. La surface d'un opérateur 10 bits est de ~0,4mm^2. D'où $P_{Ma} = 20 \times (0.4 + 0.12) = 10.4$

Conclusion

Avec l'unité d'accélération de la multiplication, nous obtenons un rapport $\frac{P_{Pr}}{P_{Ma}} = \frac{19.2}{10.4} = 1.85$

Le gain n'est pas suffisant pour justifier complètement son utilisation. Cependant le gain en temps de calcul est de 8, et la mise au carré intervient fréquemment dans l'algorithme global. Si le taux d'intégration le permet il pourra donc être intéressant d'implémenter cette unité.

Calcul de l'inverse de la racine carrée

Calcul classique

L'approche programmée du calcul de $1/\sqrt{x}$ nécessite de calculer premièrement l'extraction de la racine carrée, puis la division, qui impliquent chacune ~100 opérations[1] 10 bits, pour une entrée 30 bits.

On obtient par conséquent $P_{Pr} = 100 \times 0.12 = 12$, par opération.

Synthèse directe de l'extraction de la racine carrée

Nous avons tout d'abord cherché à implémenter un opérateur de calcul de $\sqrt{x}$ sous forme d'une PLA. Solo1400 n'a pas permis la synthèse directe de l'ensemble de l'unité, et nous l'avons donc découpé en plusieurs sous-unités, chacune fournissant une partie du résultat. Nous donnons dans le Table 6.2 les résultats pour différentes configurations.

Nous n'avons pu effectuer la synthèse complète pour une donnée sur 20 bits. Cependant, si l'on suppose que la PLA nécessaire au calcul du bit b_i est 50% plus complexe que celle générant le bit b_{i-1}, on aboutit alors à une surface totale de plus de 3mm^2.

La taille pour une entrée sur 24 bits est estimée à plus de 14mm^2, ce qui est bien trop important, même si le gain de performances induit est intéressant.

	Out	In	NTP	Surf	Tp
x : 16 bits $\sqrt{x}$: 8 bits	0:6	11	119	0,391	28
	7:7	16	66	0,263	20
	0:5	10	83	0,267	22
	6:7	16	105	0,397	26
x : 20 bits $\sqrt{x}$: 10 bits	0:5	12	140	------	------
	0:4	10	78	0,252	22
	5:5	12	67	0,215	20
	6:6	14	100	0,336	25
	7:7	16	147	-------	------

Out : Portion calculée
In : Nb signaux d'entrée
NTP : Nb Termes Produits
Surf : Surface PLA (mm^2)
Tp : Temps propagation (ns)

Table 6.2 ***Synthèse de PLA pour le calcul de*** $\sqrt{x}$. *Plusieurs configurations de découpage ont été testées. Le synthétiseur n'a pas permis le traitement de PLA dont NPT > 140.*

Autres techniques

Plusieurs dispositifs ont été proposés pour l'extraction de racine carrée, ou le calcul de division. On peut citer :

- Les réseaux parallèles, tels celui de Majithia [Maji72]. Ces réseaux sont définis de manière à pouvoir calculer une extraction de carré et une division. Nous avons réalisé une implémentation de ce type d'opérateur, sous forme pipelinée afin de permettre en fonctionnement à 16MHz [Lapo91]. Il comporte 24.000 transistors, pour une surface de 8mm^2.
- Les techniques d'approximation du calcul (cf Annexe F). Ces méthodes nécessitent soit des calculs complexes (mise en oeuvre de multiplications), soit ne permettent pas d'obtenir la précision dont nous avons besoin (12 bits significatifs).

Nous ne retiendrons que le réseau parallèle, qui permet un rapport $\frac{P_{Pr}}{P_{Ma}} \approx \frac{12}{8} = 1.5$, avec un gain de 100 en nombre d'opérations.

1. Le nombre exact dépend des optimisations mises en oeuvre. Nous donnons ici une valeur approchée.

Conclusion

En ce qui concerne ce calcul, seul un opérateur réalisé sous forme de réseau parallèle pipeliné présente un intérêt. Il est cependant de taille importante. Son utilisation est donc conditionnée par l'intégration possible sur le processeur.

6.3.4 Performances

Nous nous placerons dans le cas d'une organisation $2D_{1,1}$. Le temps nécessaire au pré-calcul d'éclairement a été spécifié (Eq. 6.1) comme étant :

$$T_{Ecl} \approx \alpha \cdot N_V \cdot T_{ppc}$$

En considérant que le cube voxel est rempli à $1/25^e$ (cf § 4.2 - p. 76), on obtient alors :

$$T_{Ecl} = \frac{512}{25} \cdot T_{ppc}$$

La valeur de T_{ppc} est, bien entendu, fonction des unités matérielles utilisées. Les différentes valeurs possibles de T_{Ecl} sont données en Table 6.3. Notons qu'il s'agit uniquement d'un ordre de grandeur, cette phase de traitement n'ayant pas été complètement implémentée.

A partir de ces chiffres, nous pouvons en déduire que l'utilisation d'opérateurs câblés n'apporte qu'un gain relativement faible. Ceci s'explique par le fait que le plus grande partie des calculs est constituée de multiplications, pour lesquelles nous n'avons pas proposé d'unité matérielle (un réseau parallèle de multiplication a une taille comparable à celle de l'extracteur de racine carrée).

Approche	Nb Op	T_{Ecl}	Surf. Op.	P_{Pr}/P_{Ma}
1	4.000	8ms	0.12mm^2	/
2	3.200	6.5ms	0.52mm^2	0.3
3	2.400	4.9ms	8.52mm^2	0.023

Table 6.3 ***Performances du précalcul d'éclairement.*** *T_{Ecl} est calculé en considérant une fréquence horloge de 10MHz, et une source lumineuse unique.*
Approche 1 : Approche programmée. Approche 2 : Opérateur x^2. Approche 3 : Opérateur x^2 et réseau parallèle racine carrée et division.

Conclusion

Pour obtenir un gain de performances intéressant, il est nécessaire de mettre en oeuvre un ensemble d'opérateurs câblés tel que la surface d'une Unité de Calcul sera de l'ordre de 17mm^2.

Cette surface est peu compatible avec une parallélisation massive. Nous ne conserverons donc que l'opérateur x^2, qui est également utilisé pour le calcul de l'éclairement spéculaire, durant la phase de Lancer de Rayon Discret (cf Annexe F, § F.4).

6.4 En résumé

Le processus de précalcul de l'éclairement local est composé de trois phases :

- *Le calcul des vecteurs normaux*
  Ce traitement est inclus dans le processus de voxelisation.
- *Le lancer des rayons d'ombrage, qui permet de déterminer les voxels éclairés*
- *Le calcul de l'éclairement en chaque voxel*
  Pour ce traitement, on a défini un opérateur câblé permettant l'accélération du calcul de la fonction x^2.

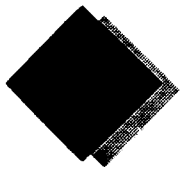
Dans les trois cas, l'organisation 3D ne présente aucun avantage par rapport à l'organisation 2D.

3ème Partie

Synthèse et Conclusion

Les études que nous avons menées dans la deuxième partie de ce document nous ont permis d'effectuer un certain nombre de choix architecturaux en fonction des algorithmes à mettre en œuvre pour effectuer la synthèse d'une image par Lancer de Rayon Discret.

Ces choix étant *dispersés* tout au long des 4 chapitres précédents, nous proposons dans cette dernière partie d'en faire la synthèse, avant de conclure ce travail.



Synthèse

Retour sur les domaines d'applications

Dans le cadre originel de ce travail, nous avions limité les types de scènes visualisables à celles permettant de convertir et stocker l'ensemble des objets au sein du cube voxel de la machine. Les séquences d'animation envisagées se restreignaient à des mouvements de l'observateur (déplacement *autour* du cube). Nous avons qualifié les applications de cette catégorie de *scènes statiques* (cf. § 3.1.3 - p. 55).

Dans ce domaine strict, la scène est chargée une fois et visualisée plusieurs fois sous différents angles. Le précalcul de l'éclairement prend donc ici tout son avantage, raison pour laquelle nous l'avons proposé et étudié.

Quant à l'utilisation du réseau pour la voxelisation, elle se justifie par la diminution sensible du temps de chargement (une facette est convertie en quelques cycles, alors que si la voxelisation est effectuée sur la machine hôte on ne peut charger qu'un voxel par cycle). La voxelisation interne est cependant restrictive, car elle est cablée pour un (ou plusieurs) type d'objet prédéfini. Il est donc important de conserver la possibilité d'une discrétisation externe.

Nous avons également désiré étudier d'autres domaines d'applications. Nous avons indiqué que l'utilisation d'une machine voxel, telle que nous l'avons définie, n'est pas adaptée à la visualisation de *scènes d'extérieur* (à grande profondeur de champ). Il est par contre envisageable de traiter des *scènes dynamiques* (vue d'une partie de la scène, ou déplacement d'objets au sein de l'image). Cependant, nous avons montré que dans ce cas il était pratiquement nécessaire de recharger le cube voxel à chaque image.

Il nous semble alors indispensable de mettre en oeuvre une voxelisation interne si l'on désire un affichage rapide, mais il faut se *contenter* des objets prévus par l'algorithme de discrétisation cablé.

Par contre, le processus de précalcul de l'éclairement peut sembler moins utile, puisqu'il doit être réexécuté à chaque chargement. Il apporte cependant un avantage qui nous semble indéniable de par le fait qu'il évite d'avoir à lancer les rayons d'ombrages durant la phase de rendu par Lancer de Rayon (cf. § 5.2.1 - p. 111). Ceci permet de décharger considérablement le réseau de communication et de limiter les engorgements temporaires, voire les *deads-locks*. L'utilisation du pré-éclairement doit donc permettre d'améliorer les performances globales. Seule une simulation *grandeur nature* nous apportera les informations nécessaires pour statuer sur son importance.

En résumé :

- la voxelisation interne est importante pour les scènes statiques, et *indispensable* pour les scènes dynamiques,
- le précalcul d'éclairement est *indispensable* pour les scènes statiques, et important pour les scènes dynamiques.

Fonctionnement général du réseau

Parmi les différentes organisations possibles (cf. § 3.1.5 - p. 58), nous avons retenu l'organisation distribuée $2D$ asynchrone (Figure 7.1) qui offre presque les mêmes performances que l'organisation $3D$, sans l'inconvénient de la complexité de réalisation matérielle.

L'organisation mémoire distribuée a été choisie afin d'être près de la notion de machine voxel idéale (un processeur prend en charge le traitement d'un seul voxel), machine dont nous voulons étudier le comportement et la faisabilité.

Le fonctionnement asynchrone a été choisi comme étant le meilleur pour l'exécution de l'algorithme du Lancer de Rayon (cf § 5.2.2 - p. 112).

Les études menées sur les phases de pré-éclairement et de rendu par Lancer de Rayon considèrent qu'une cellule est associée à un seul dexel[1]. Pour un cube voxel de dimension $N_V = 512$, cela représente un réseau de 2^{18} processeurs. Si l'on veut réduire ce nombre, il faut regrouper la prise en charge de plusieurs dexels sur une cellule. Si cela ne pose pas de problèmes dans la phase de voxelisation, il faudra néammoins adapter les deux autres processus.

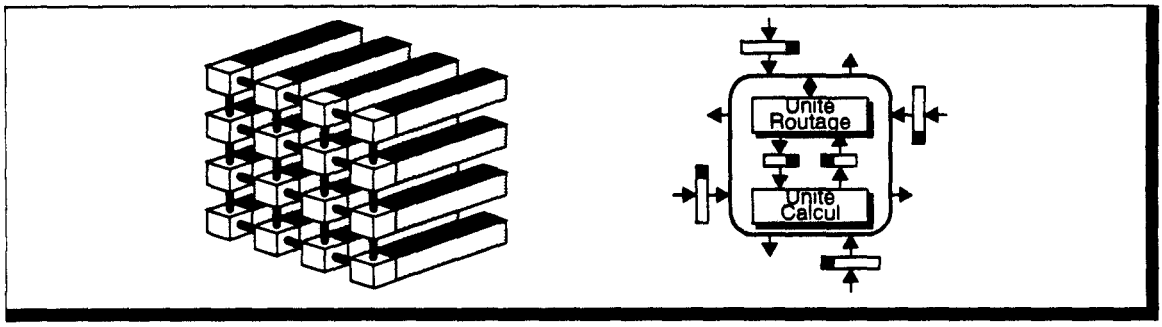


Figure 7.1 ***Organisation distribuée 2D retenue, et schéma synoptique d'une cellule.***

Fonctionnement durant la phase de voxelisation

Pour l'exécution de ce processus, le mode de fonctionnement qui nous semble le plus intéressant est du type multipipeline (Figure 7.2) (cf § 4.2 - p. 76). Cette architecture permet l'application d'un algorithme de discrétisation d'objets par balayage du plan XY (cf. § 4.3.2 - p. 86).

Nous avons discuté de la possibilité d'utiliser un réseau partiel durant ce traitement (cf § 4.2.4 - p. 81), mais ce choix a été repoussé pour l'instant car il augmente les risques d'interblocage durant les phases de précalcul de l'éclairement et de rendu.

Le multipipeline est implémenté avec un pipeline vertical externe au réseau. Ces cellules sont également utilisées en phase de Lancer de Rayon pour véhiculer les rayons supprimés vers la machine hôte (cf § 5.5.3 - p. 132). Il présente donc un avantage indéniable par rapport à une solution utilisant les cellules du bord du réseau comme pipeline vertical.

La seule direction de communication utilisée durant la voxelisation est de l'*ouest* vers l'*est*, et l'Unité de Routage est inutile. De fait, on donnera à l'Unité de Calcul un accés direct aux deux boîtes aux lettres de communications entre cellules voisines.

L'algorithme de voxelisation par balayage plan unique effectue le remplissage du cube voxel, avec, pour chaque voxel occupé, le vecteur normal à l'objet et les informations photométriques.

1. Ensemble des voxels dont la projection sur le plan XY est la même.

Une fois le chargement de la scène terminée, les mémoires des Unités de Recherche par Bloc sont mises à jour.

Les performances atteintes sont de l'ordre de 100.000 facettes par seconde.

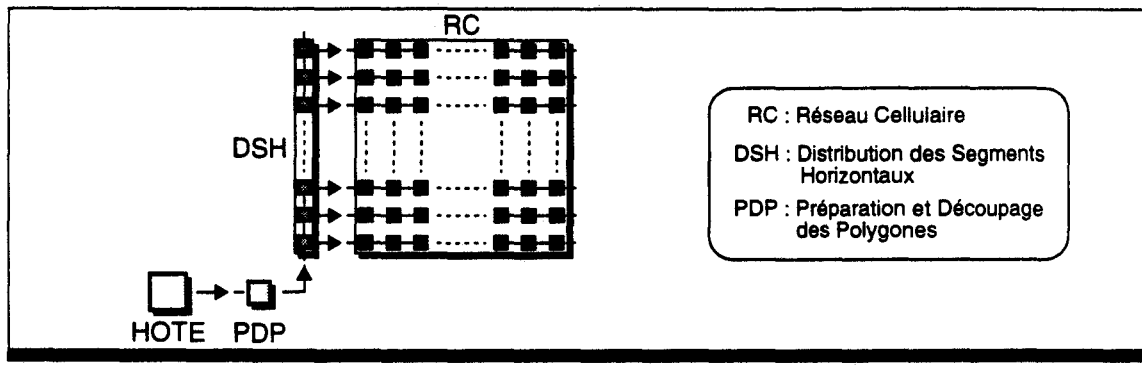


Figure 7.2 ***Fonctionnement multipipeline durant la voxelisation.***

Fonctionnement durant la phase de précalcul et de rendu

Durant ces deux phases, le réseau fonctionne complètement en mode MIMD.

Les Unité de Routage assurent la transmission des messages contenant les rayons, et les Unités de Calcul effectuent, en parallèle, les opérations d'éclairement pour le traitement de précalcul, les opérations liées à une intersection rayon/objet dans la phase de rendu. L'algorithme de suivi de rayon utilise la méthode de construction par palier (cf Annexe E), ce qui permet de traverser un dexel en une opération.

La synchronisation des traitements, entre cellules et entre Unités, est assurée par le mécanisme de communication par boîtes aux lettres (Figure 7.3).

Une Unité de Recherche par Bloc est utilisée pour permettre la détermination de l'intersection entre un palier de rayon et le dexel géré par une cellule, en deux cycles horloge.

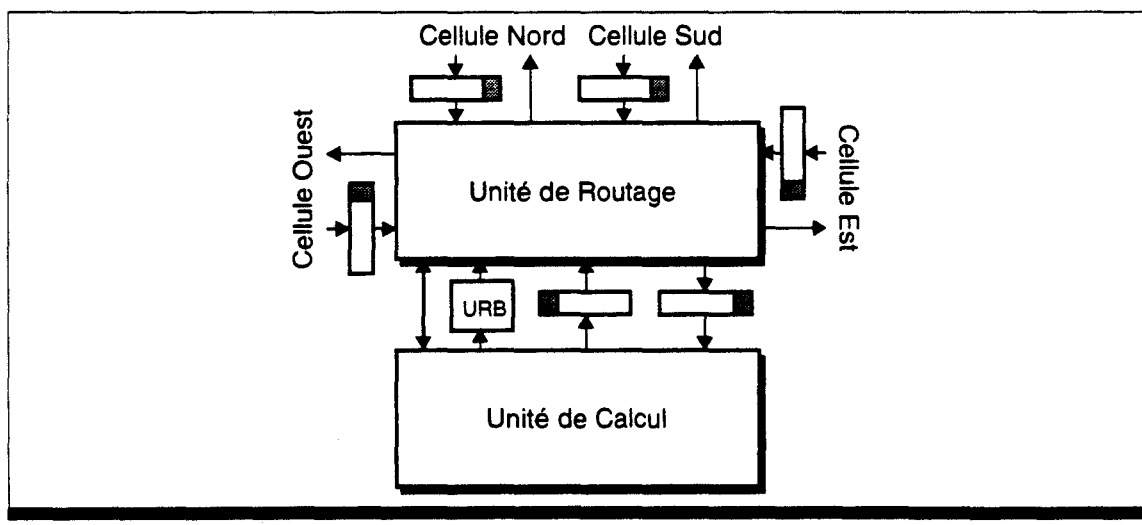


Figure 7.3 ***Synoptique d'une cellule du réseau.*** *On y trouve entre autre l'Unité de Recherche par Bloc.*

En phase de précalcul, on effectue le lancement des rayons d'ombrages source par source, ce qui permet de garantir le non-blocage des ressources de communications (cf. § 6.2.1 - p. 142). Une unité d'accélération du calcul de la fonction x^2 est intégrée dans l'Unité de Calcul. Elle est également utilisée pour le calcul de l'intensité spéculaire par approximation durant la phase de rendu, ce qui justifie son implémentation. L'unité proposée pour le calcul de la fonction $\sqrt{x}$ est trop coûteuse en terme de surface silicium pour que nous l'ayons conservée.

Nous avons montré que des interblocages des ressources de communication pouvaient apparaître durant la phase de rendu. Nous avons proposé, pour y remédier, un protocole de *retrait* temporaire de certains messages, afin de rompre les cycles de dépendances. Le pipeline vertical externe, qui est construit à partir de cellules du même type que celles du réseau mais sans Unité de Routage, est utilisé pour transmettre ces rayons vers la machine hôte.

Proposition d'implémentation de la cellule du réseau

Nous fournissons ci-après les schémas synoptiques qui permettront l'étude détaillée de l'implémentation du réseau de la machine RC^2. Le schéma des boîtes aux lettres a été donné en Figure 5.16, page 129.

La Figure 7.4 décrit la connexion de l'Unité de Calcul aux organes externes qui l'entourent, à savoir :

- les deux boîtes aux lettres d'interface avec l'Unité de Routage. Ces boîtes ne nécessitent pas l'utilisation des drapeaux Flag_xxx puisqu'il n'y a pas de calculs particuliers nécessitant d'être sauvegardés en cas d'impossibilité de routage.
- la mémoire de l'Unité de Recherche par Bloc.
- l'accès direct aux boîtes aux lettres Ouest et Est durant la phase de voxelisation.

La Figure 7.5 présente la connexion de l'Unité de Routage aux organes qui l'entourent :

- les boîtes aux lettres en entrée dans les quatre directions. Ces boîtes disposent des signaux Flags_xxx, afin de prendre en compte les calculs de suivi ou d'intersection déjà effectués, en cas de non possibilité de routage immédiat.
- l'accès aux boîtes aux lettres de sorties, situées en entrée des cellules voisines.
- l'Unité de Recherche par Bloc d'une intersection.
- les boîtes aux lettres d'interface avec l'Unité de Calcul.
- l'accès direct aux boîtes Ouest et Est depuis l'Unité de Calcul.

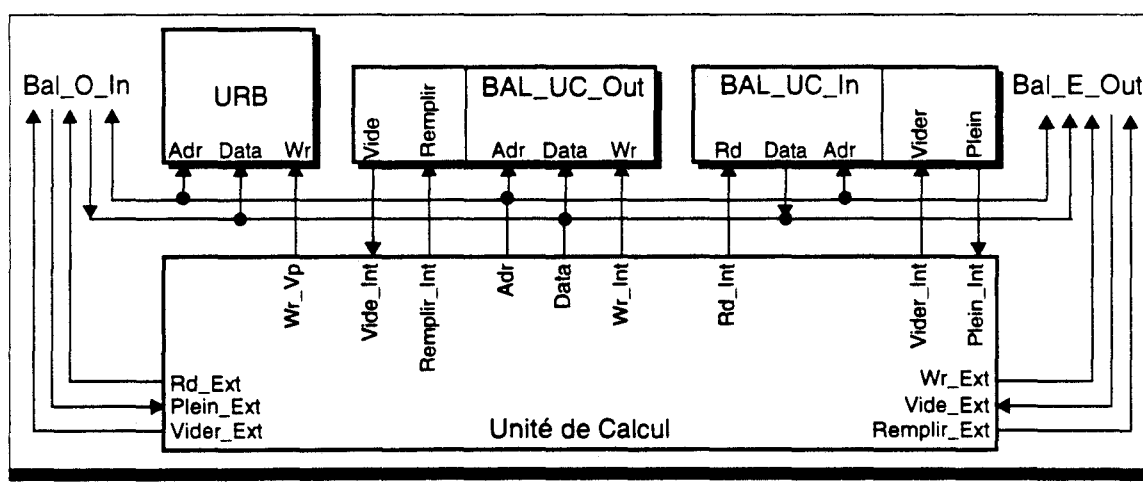


Figure 7.4 ***Connexion de l'Unité de Calcul aux organes externes.***

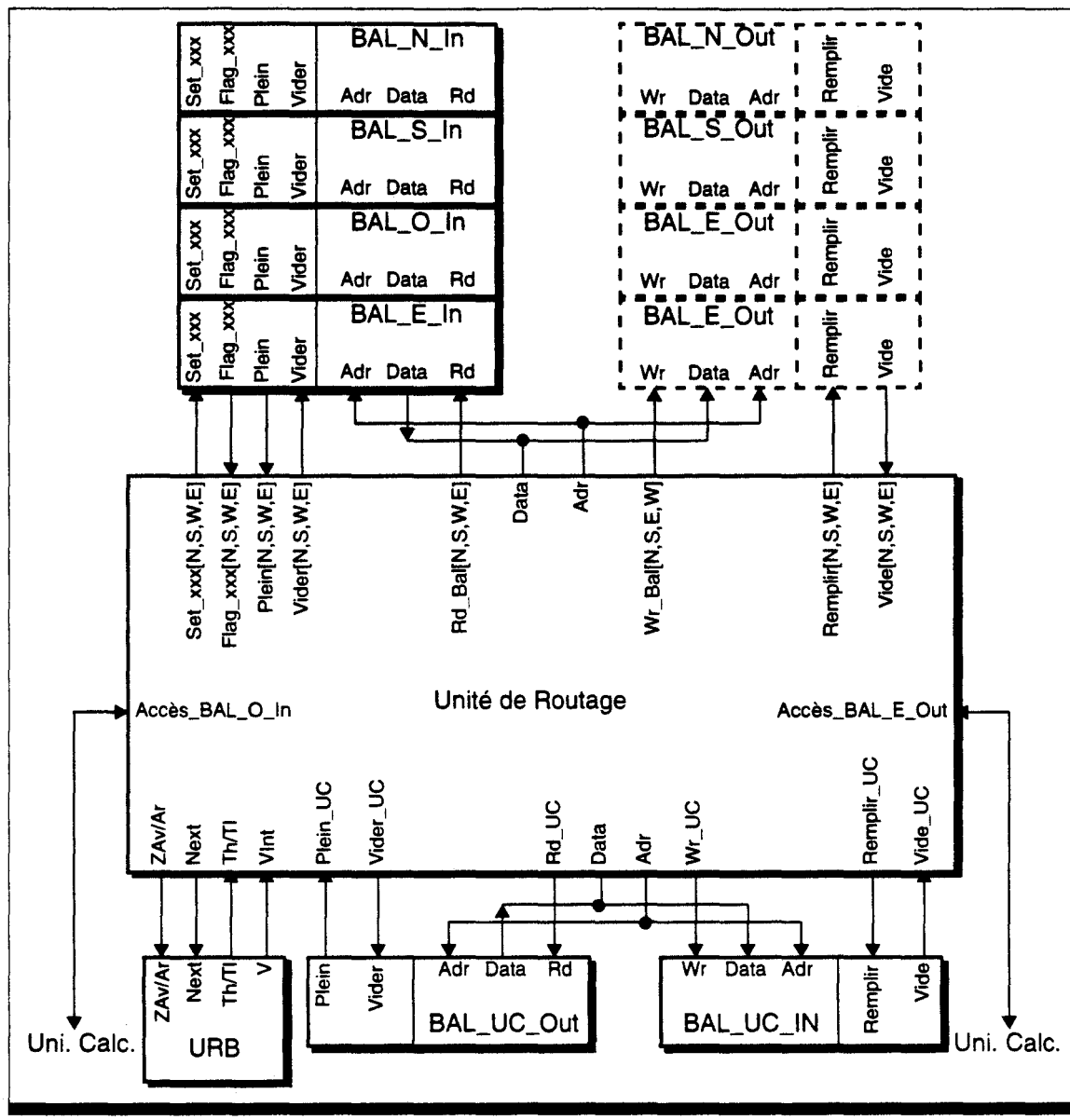


Figure 7.5 ***Connexion de l'Unité de Routage aux organes externes.***

Contrôleur des Unités de Calcul et de Routage

Le contrôleur de l'Unité de Calcul implémente les trois algorithmes correspondant aux trois phases de calcul. Celui de l'Unité de Routage effectue le suivi de rayon et la transmission des différents autres messages. Nous avons indiqué que de par le fonctionnement MIMD du réseau, chaque cellule doit disposer de sa propre unité de contrôle (cf. § 4.4.5 - p. 103).

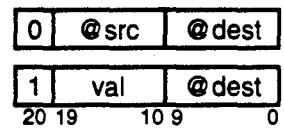
Le code correspondant au Lancer de Rayon Discret est relativement complexe, il est donc préférable d'utiliser un contrôle de type micro-programmé. Nous voulons pour permettre une certaine souplesse de modification du code mettre en place une mémoire de programme externe. Cependant un contrôle micro-programmé nécessite un grand nombre de signaux, ce qui impliquerait un circuit disposant d'un grand nombre de broches.

Nous avons pour ces raisons choisi d'utiliser la méthode présentée par D.W Jones [Jone88], qui consiste en un noyau RISC minimum, n'utilisant qu'une seule instruction : move @src, @dest. Il permet de minimiser le nombre de signaux externes à ceux nécessaires pour adresser tous les circuits d'une unité.

Il est souvent nécessaire de pouvoir utiliser une valeur constante (dans une opération d'incrémentation par exemple). On ajoutera donc un drapeau pour identifier une deuxième opération : move val, @dest.

Les unités opératives sont de largeur 10 bits (cf. § 4.4.4 - p. 103), nous prendrons donc un format de champ d'instruction de largeur 10 bits également, ce qui permet l'adressage de 1024 unités, nombre suffisant.

Le format d'instruction est donc :



Pour valider ce principe, nous avons réalisé une UAL utilisant ce contrôle à l'aide du logiciel SOLO1400 [Lacq92]. Nous prendrons l'Unité de Calcul en example dans le paragraphe suivant.

Partie Opérative

Nous donnons en Figure 7.6 le schéma de la partie opérative de l'Unité de Calcul, utilisant le contrôle explicité ci-dessus. Tous les signaux non marqués correspondent aux signaux de contrôle générés par les décodeurs d'adresses de l'instruction.

On retrouve par exemple l'entrée Direct_In permettant de placer une valeur constante sur le bus de données.

L'accès à la mémoire voxel se fait par concaténation de 2 adresses: l'adresse du voxel, et l'adresse de la donnée désirée dans ce voxel (cette valeur pouvant provenir du contrôleur).

Le multiplexeur en sortie de l'UAL permet de sélectionner (par le jeu des adresses) le résultat de l'opération désirée.

Nous ne détaillerons pas plus avant ce schéma, qui ne constitue qu'un synoptique de principe, à partir duquel on pourra établir le schéma détailé.

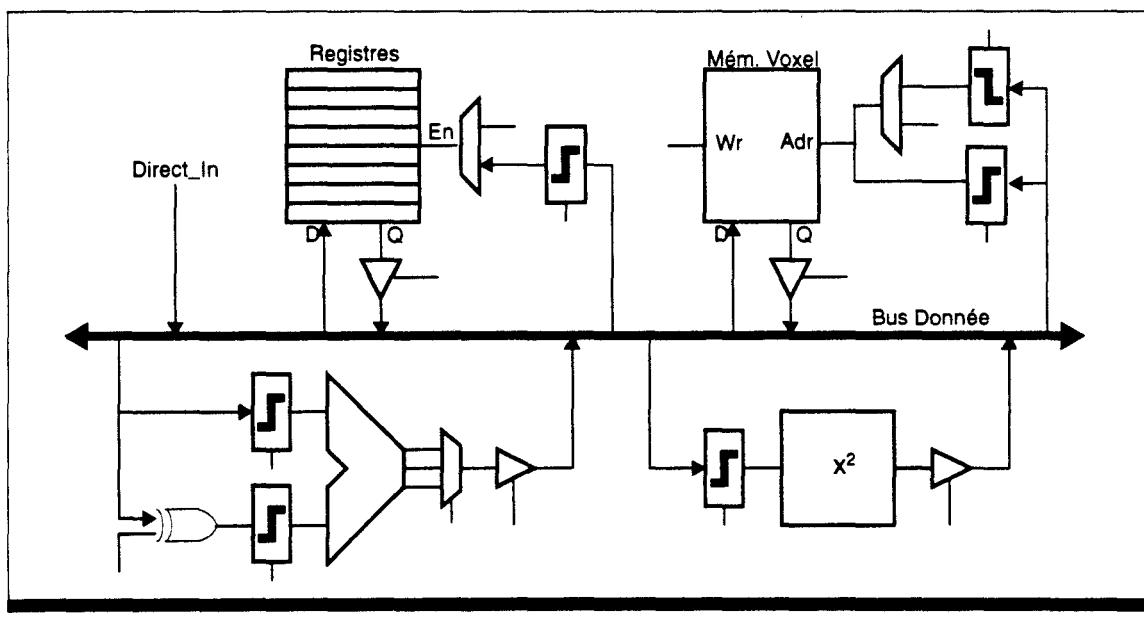
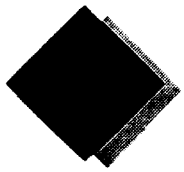


Figure 7.6 ***Schéma de principe de la partie opérative de l'Unité de Calcul.*** *Les signaux non marqués sont les signaux de contrôle produits par le décodage d'adresse.*



Conclusion

L'utilisation de l'espace discret 3D pour la synthèse d'images réalistes est un domaine où peu d'études ont été faites jusqu'à présent. Nous avons voulu étudier l'adéquation entre le monde voxel et l'algorithme du Lancer de Rayon, en prenant pour but la visualisation rapide, voire temps réel, de séquences animées.

La méthode proposée utilise un *cube* voxel, qui contient une description discrète de la scène à visualiser. Cette mémorisation complète de l'espace apporte un certain nombre d'avantages, tel que la possibilité de disposer d'une information géométrique ou photométrique en tout point de l'espace. Le calcul d'intersection entre un rayon lumineux et les objets de la scène, classiquement utilisé en Lancer de Rayon, se réduit ainsi en un simple test de présence de matière au sein d'un voxel. Le cube discret présente cependant l'inconvénient majeur d'être de dimension finie, ce qui limite les domaines d'applications, en permettant difficilement, par exemple, le traitement de scènes d'extérieur où la profondeur de champ est importante.

L'adaptation de l'algorithme de base, appelé Lancer de Rayon Discret, pose de nouveaux problèmes propres à la géométrie discrète, comme, par exemple, la définition de l'intersection exacte entre un rayon et une surface discrets, ou la connexité entre surfaces discrètes adjacentes. Nous avons apporté quelques solutions, mais l'étude complète des problèmes soulevés sort du cadre de ce travail. Il est cependant important de s'y consacrer, afin de pouvoir améliorer la qualité des images produites par Lancer de Rayon Discret.

Nous avons montré dans une première partie que le Lancer de Rayon Discret apporte une accélération de l'algorithme de base équivalente aux méthodes de subdivision spatiale. Ces dernières offrent des caractéristiques permettant une implémentation efficace sur des architectures moyennement parallèle, mais nous pensons qu'elles ne permettent pas une parallélisation massive, contrairement au modèle voxel. Le Lancer de Rayon Discret pourrait donc permettre une amélioration sensible des temps de calculs.

Partant de cette constatation, nous proposons une architecture de machine voxel spécialisée pour le Lancer de Rayon Discret, construite autour d'un réseau cellulaire massivement parallèle, où chaque processeur élémentaire prend en charge un sous-ensemble connexe de l'espace voxel. Il implémente un parallélisme MIMD à flot de rayons. Nous avons montré que dans ce cas les performances globales sont en majeure partie fonction du temps de parcours des rayons au sein du cube voxel. Nous proposons de découpler les organes de communications des organes de calcul, afin de permettre à un rayon de pouvoir *traverser* une cellule même lorsque celle-ci possède un calcul en cours. Les différentes unités ayant un fonctionnement asynchrone, nous utilisons des communications par messages au moyen de boîtes aux lettres.

L'affinement des spécifications de l'architecture a été effectuée par l'étude des trois phases de calcul du Lancer de Rayon :

- la Voxelisation, qui réalise la discrètisation des objets. Dans un certain nombre d'applications (objets animés, par exemple), ce traitement doit être exécuté à chaque image. Il est donc important qu'il s'effectue rapidement. Nous proposons d'utiliser la puissance du réseau pour cette tâche, et nous avons détaillé un algorithme de discrètisation de facettes par balayage plan unique qui permet la conversion de plus de 100.00 facettes par seconde.
- le Précalcul d'éclairement, qui effectue la détermination de l'éclairement local en tout point de l'espace discret par lancement de rayons d'ombrage. Durant cette phase tous les calculs photométriques indépendants de la position de l'observateur sont exécutés. Pour diminuer les temps de calculs de ce traitement, nous proposons une unité matérielle d'accélération de l'élévation au carré.
- le Rendu qui finalise tous les calculs photométriques par lancer des rayons primaires. Durant cette phase peuvent se produire des interblocages dûs à des cycles de dépendance sur les ressources de communications. Nous avons proposé un protocole de suppression de messages pour éviter un blocage définitif.

Seule la première de ces trois phases a pu être complètement simulée. Il est indispensable pour affiner les spécifications et le fonctionnement du réseau de poursuivre l'étude des deux autres tâches. Un simulateur fonctionnel spécifique a été écrit dans ce but.

La parallélisation du Lancer de Rayon classique a fait l'objet d'un grand nombre de travaux qui ont abouti à des schémas dont l'efficacité parallèle est très importante et qui ont pratiquement *clos* le domaine. Le Lancer de Rayon Discret, s'il n'atteint pas ce niveau d'efficacité, possède cependant un parallélisme massif intrinsèque permettant d'augmenter le niveau de performances. Sa mise en oeuvre requiert, avec la technologie actuelle, la définition d'une architecture spécialisée, et nous avons voulu dans ce travail cerner le maximum des problèmes et des avantages nouveaux qu'il apporte. Une des voies ouvertes est, par exemple, la possibilité de simuler un milieu participatif. Il s'interface également naturellement avec l'imagerie médicale.

Une fois résolus les problèmes de géométrie et de rendu, le Lancer de Rayon Discret devrait donc trouver sa place, que ce soit dans les applications nécessitant un affichage rapide d'images d'un réalisme élevé, ou dans l'affichage d'images plus complexes que celles obtenues à partir de base de données géométriques classiques.

Annexes

ANNEXE

Modèles physiques d'éclairement

Nous présentons dans cette annexe un modèle d'éclairement basé sur le transfert d'énergie lumineuse entre deux surfaces. Il permet d'améliorer les effets de reflets spéculaires sur les matériaux rugueux. Pour cela une surface est considérée comme étant constituée de micro-facettes réfléchissantes.

Définitions physiques

Nous donnons ici la liste des termes et définitions utilisés en physique optique [Fole90].

- Flux = $\frac{\text{Energie lumineuse}}{\text{Unite de temps}} \equiv \left[\frac{\text{J}}{\text{s}}\right] \equiv [\text{W}]$, qui correspond à la vitesse à laquelle l'énergie lumineuse est transmise, ou reçue.
- Intensite rayonnante = $\frac{\text{Flux}}{\text{Angle solide sous-tendu par la source}} \equiv \left[\frac{\text{W}}{\text{sr}}\right]$, qui correspond à ce que l'on appelle habituellement l'intensité de la source.
- Irradiance = $\frac{\text{Flux}}{\text{Unite d'aire}} \equiv \left[\frac{\text{W}}{\text{m}^2}\right]$, qui correspond à la densité de flux reçue ou émise par une surface.
- Radiance = $\frac{\text{Intensite rayonnante}}{\text{Unité d'aire projetée}} \equiv \left[\frac{\text{W}}{\text{sr} \cdot \text{m}^2}\right]$, qui correspond à ce que l'on appelle habituellement l'intensité de la surface.

Remarque : Ces valeurs ne sont pas des constantes. Ce sont des fonctions de distribution spectrale. Autrement dit, elles indiquent la valeur que prend un terme en fonction d'une longueur d'onde.

En utilisant les notations de la Figure A.1, on obtient les relations suivantes, en posant F_i, le flux émis par la source lumineuse incidente :

1. $E_i = \frac{F_i}{dA}$ (irradiance de la source incidente sur la surface dA).
2. $I_i = \frac{F_i}{d\omega \cdot dP} = \frac{F_i}{d\omega \cdot dA \cdot \cos\theta_i}$ (radiance de la source incidente).

A l'aide de ces deux relations, on obtient la relation entre la lumière incidente et la lumière que peut réfléchir ou transmettre la surface :

$$E_i = I_i \cdot \cos\theta_i \cdot d\omega = I_i \cdot (\vec{N} \bullet \vec{S}) \cdot d\omega \qquad \textbf{(Eq. A.1)}$$

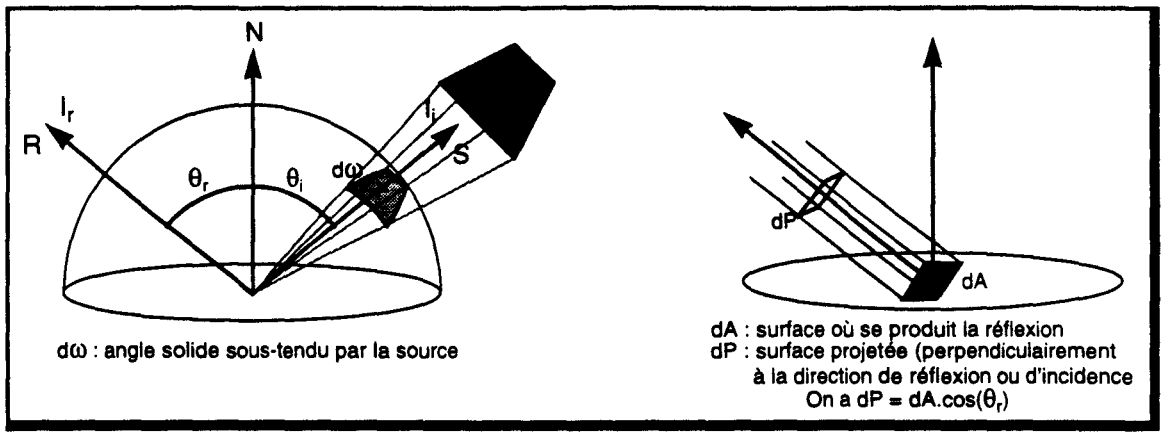


Figure A.1 ***Notations utilisées pour exprimer les quantités physiques liées à l'optique.***

L'intensité réfléchie (ou radiance réfléchie) n'est pas une fonction directe de l'intensité incidente (ou radiance incidente). En effet, deux sources ayant la même intensité (la même radiance), mais sous des angles solides différents provoqueront une luminosité (irradiance) différente sur la surface. Il faut donc plutôt calculer l'intensité réfléchie à partir de l'irradiance provoquée sur la surface. Le rapport entre les deux est appelé *réflectance bidirectionnelle*, notée ρ, avec :

$$I_r = \rho \cdot E_i$$

On obtient donc :

$$I_r = \rho \cdot I_i \cdot (\vec{N} \bullet \vec{S}) \cdot d\omega \qquad \textbf{(Eq. A.2)}$$

Application

Il est habituel de séparer l'éclairement en deux entités, l'éclairement diffus et l'éclairement spéculaire.

De la même façon, on prendra l'approximation suivante :

$$\rho = k_d \cdot \rho_d + k_s \cdot \rho_s$$

avec ρ_d : réflectance diffuse bidirectionnelle.
ρ_s : réflectance spéculaire bidirectionnelle.

On prendra également en compte une intensité ambiante uniformément répartie dans l'espace. On notera $\rho_a I_a$ ce terme ambiant, ρ_a étant la réflectance ambiante non directionnelle.

On obtient alors l'équation générale :

$$I_r = \rho_a \cdot I_a + I_i \cdot (\vec{N} \bullet \vec{S}) \cdot d\omega \cdot (k_d \cdot \rho_d + k_s \cdot \rho_s) \qquad \textbf{(Eq. A.3)}$$

Expression de ρ_s

L'effet spéculaire doit être fonction de la longueur d'onde de la lumière incidente, des caractéristiques du matériau (entre autre sa rugosité), et de l'orientation de la surface, comme de la position de la source.

En 1967, Torrance et Sparrow [Torr67] ont défini un modèle théorique en considérant la surface d'un objet comme constitué de micro-facettes réfléchissantes, orientées aléatoirement, et dont la taille est grande par rapport à la longueur d'onde de la lumière incidente[1].

Ce modèle a été repris par Blinn [Blin77], qui considère que les reflets spéculaires sont de la même couleur que la source de lumière incidente, puis par Cook et Torrance [Cook81], qui intègrent la variation de la réflectance spéculaire en fonction de la longueur d'onde de la lumière incidente.

1. Pour éviter les phénomènes tels que la diffraction, et pour pouvoir appliquer des lois statistiques.

Modèle de Blinn

Dans le modèle théorique de Torrance-Sparrow, l'intensité réfléchie provient des micro-facettes orientées dans le sens du vecteur $\vec{H}$ (Figure A.2). L'ensemble de ces facettes ne permettent pas une réflexion complète de l'intensité qu'elles reçoivent, comme le montre la Figure A.2. Et, de plus, une partie seulement de l'intensité reçue sera réfléchie, l'autre étant absorbée.

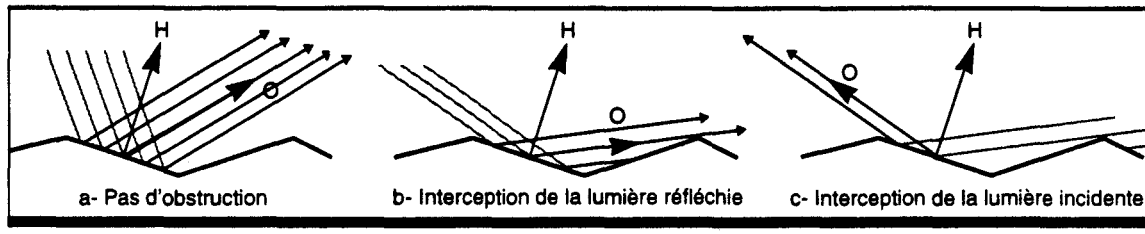


Figure A.2 ***Les trois cas d'obstruction du faisceau incident.***

On obtient alors [Blin77] :

$$\rho_s = \frac{D \cdot G \cdot F}{(\vec{N} \bullet \vec{O})} \qquad \textbf{(Eq. A.4)}$$

avec D : Fonction de distribution de l'orientation des micros-facettes.

G : Facteur d'atténuation géométrique, dû à l'orientation des facettes.

F : Loi de réflexion de Fresnel.

La luminosité perçue par l'observateur est fonction de la surface de la facette qu'il voit. Or cette surface est d'autant plus importante que la facette est penchée. Ce facteur est inversement proportionnel au cosinus de l'angle entre le vecteur normal moyen aux facettes et le vecteur observateur, d'où l'utilisation du terme $(\vec{N} \bullet \vec{O})$.

- D : Fonction de distribution

  Ce terme indique la distribution des facettes dans une direction donnée. Torrance-Sparrow ont utilisé une gaussienne pour le calculer :

  $$D(\alpha, m, c) = c \cdot exp\left(-\left(\frac{\alpha}{m}\right)^2\right)$$

  avec α : Orientation de la facette par rapport à la normale moyenne, $\alpha = cos^{-1}(\vec{N} \bullet \vec{H})$.

  m : Racine carrée de la valeur moyenne des carrés des écarts de pente des micro-facettes.

  c : Constante caractéristique du matériau.

  Lorsque c est important, la surface paraît terne. Au contraire, lorsque c est faible, la surface a un aspect brillant.

  Lorsque m est petit (< 0.5), l'inclinaison des facettes varie peu; la réflexion obtenue est fortement directionnelle. La surface paraît donc d'un aspect lisse. Par contre, lorsque m est grand (> 0.5), l'inclinaison des facettes varie fortement; la réflexion obtenue est donc large. La surface paraît rugueuse.

- G : Facteur d'atténuation géométrique

  Le terme G est fonction du type d'obstruction rencontrée (Figure A.2). Pour le calculer, il faut supposer que les micro-facettes forment deux par deux un V.

  Dans le cas a, il n'y a pas d'obstruction : $G_a = 1.0$.

  Dans le cas b, Blinn calcule $G_b = \frac{2(\vec{N} \bullet \vec{H})(\vec{N} \bullet \vec{O})}{(\vec{O} \bullet \vec{H})}$.

Dans le dernier cas, on obtient $G_c = \frac{2(\vec{N} \bullet \vec{H})(\vec{N} \bullet \vec{S})}{(\vec{O} \bullet \vec{H})}$.

Dans le cas général, on prendra $G = min(G_a, G_b, G_c)$.

- F : Loi de Fresnel

  La loi de réflexion de Fresnel indique le pourcentage de lumière incidente qui est réfléchie par la facette plutôt que d'être absorbée. Ce facteur est fonction de l'angle d'incidence φ_i et de l'angle de réfraction φ_t, défini par les lois de Descartes à l'aide de l'indice de réfraction η du matériau composant la surface de l'objet (on a $\sin(\varphi_t) = \eta \cdot \sin(\varphi_i)$) :

$$F(\varphi_i, \eta) = \frac{1}{2}\left(\frac{sin^2(\varphi_i - \varphi_t)}{sin\ (\varphi_i + \varphi_t)} + \frac{tan^2(\varphi_i - \varphi_t)}{tan^2(\varphi_i + \varphi_t)} \right)$$

  avec $\varphi_i = cos^{-1}(\vec{S} \bullet \vec{H})$.

  Cette équation peut être réécrite sous la forme :

$$F = \frac{1}{2} \cdot \frac{(g-c)^2}{(g+c)^2} \cdot \left[1 + \frac{(c(g+c)-1)^2}{(c(g-c)+1)^2} \right] \qquad \textbf{(Eq. A.5)}$$

  avec $c = \cos(\varphi_i) = \vec{S} \bullet \vec{H}$

$$g = \sqrt{n^2 + c^2 - 1}$$

Aucun des trois termes développés ci-dessus n'est défini en fonction de la longueur d'onde. La distribution spectrale de l'intensité spéculaire n'est donc définie que par la distribution spectrale de la source lumineuse (définie par I_i).

Modèle de Cook-Torrance

Cook et Torrance ont repris le modèle fourni par Blinn d'après les travaux de Torrance et Sparrow, afin de prendre compte plus correctement l'effet que produit la réflexion spéculaire sur la couleur observée [Cook81].

Ils définissent la réflectance bidirectionnelle par

$$\rho_s = \frac{F_\lambda}{\pi} \cdot \frac{D}{\vec{N} \bullet \vec{O}} \cdot \frac{G}{\vec{N} \bullet \vec{S}} \qquad \textbf{(Eq. A.6)}$$

Le terme $\vec{N} \bullet \vec{O}$ a été introduit pour tenir également compte de la surface apparente des facettes vue depuis la source.

- Ils ont, tout d'abord, proposé d'utiliser le facteur de distribution de Beckmann [Beck63], mieux adapté aux surfaces rugueuses, et ne nécessitant pas de constante arbitraire :

$$D(\alpha, m) = \frac{1}{4m^2 cos^4(\alpha)} \cdot exp\left(-\frac{tan^2(\alpha)}{m^2} \right)$$

  avec les notations vues précédemment dans le modèle de Blinn.

- D'autre part, l'indice de réfraction η est fonction de la longueur d'onde du faisceau incident.

  Le facteur de Fresnel est donc lui-aussi dépendant de cette longueur d'onde. Il y a alors combinaison des distributions spectrales de l'intensité de la source lumineuse et du terme de Fresnel (Eq. A.5), ce qui peut de plus amener un déplacement de couleur suivant l'angle d'incidence φ_i.

En effet, lorsque l'angle d'incidence est normal, on a : $\varphi_i = 0$, d'où $c = 1$ et $g = \eta$. On obtient alors le terme de Fresnel sous incidence normale, dont la distribution spectrale constitue la couleur de l'objet :

$$F_\lambda(0) = \left(\frac{\eta_\lambda - 1}{\eta_\lambda + 1}\right)^2 \qquad \textbf{(Eq. A.7)}$$

De plus, sous incidence rasante, on a : $\varphi_i = \pi/2$, d'où $c = 0$ et $g = \sqrt{\eta^2 - 1}$. On obtient alors un terme de Fresnel constant, ce qui indique que seul le spectre de la source lumineuse intervient dans la détermination de la couleur vue :

$$F_\lambda\left(\frac{\pi}{2}\right) = 1$$

Par conséquent, la couleur de la réflexion spéculaire varie entre la couleur de l'objet (sous incidence normale), et la couleur de la source (sous incidence rasante).

Dans la plupart des cas, la valeur de l'indice de réfraction, en fonction de la longueur d'onde, n'est pas connue. Par contre, on dispose des mesures de $F_\lambda(0)$ pour un grand nombre de matériaux, et un grand nombre de longueurs d'ondes. On peut donc en déduire η_λ, puisque l'on a la relation

$$\eta_\lambda = \frac{1 + \sqrt{F_\lambda(0)}}{1 - \sqrt{F_\lambda(0)}}$$

Pour les valeurs de longueur d'ondes dont on ne dispose pas de mesures, il est possible d'effecteur une interpolation.

La valeur de n_λ est ensuite réinjectée dans la formule du terme de Fresnel (Eq. A.5).

Ces calculs étant très coûteux, Cook et Torrance proposent plutôt d'effectuer une interpolation directe de la couleur résultante, à l'aide de la méthode suivante :

1. On calcule une valeur moyenne $F_{moy}(0)$ des termes de Fresnel sous incidence normale. On en déduit un indice de réfraction moyen :

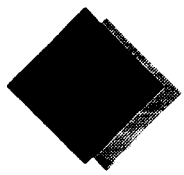
$$n_{moy} = \frac{1 + \sqrt{F_{moy}(0)}}{1 - \sqrt{F_{moy}(0)}}$$

2. Cette valeur réinjectée dans le terme de Fresnel (Eq. A.5), permet d'obtenir $F_{moy}(\varphi_i)$. On interpole ensuite la couleur résultante entre la couleur C_0 en incidence normale, soit la couleur de l'objet, et la couleur $C_{\pi/2}$ en incidence rasante, soit la couleur de la source :

$$C(\varphi_i) = C_0 + (C_{\pi/2} - C_0) \cdot \frac{max(0, F_{moy}(\varphi_i) - F_{moy}(0))}{F_{moy}(\pi/2) - F_{moy}(0)}$$

3. Cette valeur est ensuite utilisée à la place de $F_\lambda(\varphi_i)$ dans le calcul de ρ_S (Eq. A.6). La valeur de couleur prenant déjà en compte le spectre de la source incidente, il faut également remplacer I_i par un facteur d'intensité indépendant de la longueur d'onde dans l'équation de ρ_s (Eq. A.4).

ANNEXE



Calcul d'intersection entre un objet et un rayon

Nous proposons dans cette annexe d'établir le coût de calcul d'une intersection entre un objet et un rayon, dans le cas d'une facette puis d'une sphère.

Dans le calcul d'intersection, on utilise, en général, une définition paramétrique du rayon, ce qui permet de simplifier grandement les opérations nécessaires.

Un rayon est défini par :

$$\begin{aligned} X &= X_0 + X_d \cdot t \\ Y &= Y_0 + Y_d \cdot t \\ Z &= Z_0 + Z_d \cdot t \end{aligned} \qquad \textbf{(Eq. B.1)}$$

avec (X_0, Y_0, Z_0) : Coordonnées initiales du rayon,
(X_d, Y_d, Z_d) : Vecteur directeur du rayon.

Les coordonnées paramétriques du rayon sont ensuite substituées dans l'équation de l'objet, ce qui permet d'obtenir une équation en t. La solution de cette équation donne la position paramétrique du point d'intersection P_I le long du rayon.

Dans certains cas d'objets, comme la facette, il faut ensuite vérifier que l'intersection est bien située à l'intérieur de l'objet.

B.1 Intersection Facette-Rayon

Nous présentons ici la méthode utilisé par Badouel [Bado90].

Détermination de l'intersection

L'équation du plan support de la facette est donnée par : $a \cdot X + b \cdot Y + c \cdot Z + d = 0$. Les coefficients directeurs de ce plan sont supposés être stockés dans la description de la facette.

La substitution des coordonnées du rayon dans cette équation nous donne :

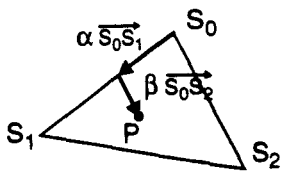
$$t_I = \frac{a \cdot X_0 + b \cdot Y_0 + c \cdot Z_0 + d}{a \cdot X_d + b \cdot Y_d + c \cdot Z_d}$$

Trois tests permettent de rejeter directement l'intersection :

1. Si $a \cdot X_d + b \cdot Y_d + c \cdot Z_d = 0$, le rayon et la facette sont parallèles.
2. Si $t \leq 0$, l'intersection se situe avant l'origine du rayon.
3. Si $t > min_t$ (avec min_t la plus proche des intersections déjà calculées), une intersection plus proche existe.

Vérification de l'intersection

Il est bien sûr possible d'effectuer un test de position de P_I par rapport à chacun des côtés de la facette, mais Badouel a défini une méthode plus rapide, fonctionnant uniquement sur des facettes triangulaires. Il utilise une définition paramétrique (proche de la représentation barycentrique) de la facette :



Le point P est définit par :

$$\overrightarrow{S_0P} = \alpha \cdot \overrightarrow{S_0S_1} + \beta \cdot \overrightarrow{S_0S_2}$$

Il est intérieur à la facette si :

$$\alpha \geq 0, \beta \geq 0, \alpha + \beta \leq 1 \qquad \textbf{(Eq. B.2)}$$

Pour résoudre le système d'équation en α et β, on effectue une projection suivant le plan de plus grande pente. Ce plan est déterminé par comparaison des coefficients a, b et c du plan support de la facette.

Supposons que la projection soit effectuée dans le plan XY. Le système à résoudre devient alors :

$$\begin{cases} X_P - X_0 = \alpha \cdot (X_1 - X_0) + \beta \cdot (X_2 - X_0) \\ Y_P - Y_0 = \alpha \cdot (Y_1 - Y_0) + \beta \cdot (Y_2 - Y_0) \end{cases}$$

On obtient donc :

$$\alpha = \frac{(X_P - X_0) \cdot (Y_2 - Y_0) - (Y_P - Y_0) \cdot (X_2 - X_0)}{(X_1 - X_0) \cdot (Y_2 - Y_0) - (Y_1 - Y_0) \cdot (X_2 - X_0)} \quad \text{et} \quad \beta = \frac{(Y_P - Y_0) - \alpha \cdot (Y_1 - Y_0)}{Y_2 - Y_0}$$

Il suffit ensuite de vérifier les contraintes sur α et β (Eq. B.2).

Coût du calcul

Le tableau ci-dessous (Table B.1) indique le nombre d'opérations en arithmétiques flottantes nécessaires pour chacune des phases du calcul d'intersection. Les différents tests ont été numérotés chronologiquement suivant leur ordre d'apparition dans le paragraphe courant.

Phase calcul	Nombre d'opérations
Dénominateur du calcul de t_I	5
Test 1	1
Calcul final t_I	7
Test 2	1
Test 3	1
Projection système équation	10
Calcul α	7
Test 4	1
Calcul β	3
Test 5	1
Test 6	2
Calcul coordonnées P_I	6

Table B.1 ***Coût des différentes phases du calcul d'intersection facette-rayon.***

Il est extrêmement difficile de connaître la probabilité de passer ou d'échouer à l'un des tests, en partie parce que cela est fonction de la répartition des facettes dans la scène et de leur orientation générale.

Pour permettre une estimation du nombre moyen de calculs nécessaires, nous prendrons comme hypothèse que les échecs aux tests sont équiprobables.

Dans ce cas on obtient un nombre moyen de 27 opérations flottantes.

B.2 Intersection Sphère-Rayon

La méthode utilisée est tirée de [Glas89]

Détermination de l'intersection

La sphère est définie par :

$$(X-X_c)^2 + (Y-Y_c)^2 + (Z-Z_c)^2 = R^2$$

avec (X_c, Y_c, Z_c) :Coordonnées du centre de la sphère
R : Rayon de la sphère

En substituant les coordonnées paramétriques du rayon dans l'équation de la sphère, on obtient :

$$(X_0 + X_d \cdot t - X_c)^2 + (Y_0 + Y_d \cdot t - Y_c)^2 + (Z_0 + Z_d \cdot t - Z_c)^2 = R^2$$

Ce qui peut s'exprimer sous la forme d'une équation du second degré en t :

$$A \cdot t^2 + B \cdot t + C = 0$$

La résolution de cette équation mène à trois possibilités de rejet de l'intersection :

1. Le discriminant est négatif. Il n'y a pas d'intersection.
2. Le discriminant est non négatif. On garde la plus petite valeur positive (notée t_l). S'il n'y a pas de valeur positive, l'intersection est située avant l'origine du rayon.
3. On a $t_l < min_t$. Il existe une intersection plus proche.

Vérification de l'intersection

L'équation de la sphère intégrant la définition de son contour, aucun test particulier n'est nécessaire.

Coût du calcul

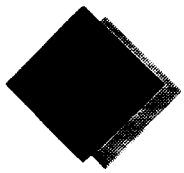
Le tableau ci-dessous (Table B.2) indique le nombre d'opérations en arithmétiques flottantes nécessaires pour chacune des phases du calcul d'intersection. Les différents tests ont été numérotés chronologiquement suivant leur ordre d'apparition dans le paragraphe courant.

Phase de calcul	Nombre d'opérations
Calcul coef. A, B, C	15
Calcul discriminant	4
Test 1	1
Calcul final de t_I	7
Test 2	1
Test 3	1
Calcul coordonnées P_I	6

Table B.2 ***Coût des différentes phases de calcul d'intersection sphère-rayon.***

Ici aussi nous ferons l'hypothèse d'équiprobabilité des échecs, ce qui donne un nombre moyen de 28 opérations flottantes.

ANNEXE



Outils de mesure des performances des architectures parallèles

Pour déterminer l'efficacité d'une machine ou d'un algorithme parallèle, il est nécessaire de définir des quantités attachées aux performances et des outils pour les mesurer. Nous indiquons dans cette annexe les principaux outils de mesure couramment utilisés.

C.1 Accélération et Efficacité

Une première façon d'évaluer une machine parallèle est de mesurer le temps d'exécution d'un algorithme par rapport à une machine monoprocesseur. On détermine ainsi le gain obtenu.

- *Accélération*. L'accélération est le rapport du temps de calcul $T(n)$ d'une machine disposant de n processeurs, par rapport à $T(1)$, temps mis sur un processeur unique :

$$Acc(n) = \frac{T(1)}{T(n)}$$

- *Efficacité*. L'accélération obtenue sur une machine n'est pas le seul critère intéressant et à retenir. Il faut également déterminer l'efficacité avec laquelle la parallélisation est utilisée pour exécuter un algorithme :

$$Eff(n) = \frac{T(1)}{n \cdot T(n)} \qquad \textbf{(Eq. 8.1)}$$

La meilleure parallélisation possible est bien sûr celle permettant d'avoir $Acc(n) = n$ et donc $Eff(n) = 1$. Cependant, que dire d'une valeur d'efficacité faible? Cela correspond-il à de mauvais choix de politique de parallélisation, ou est-ce inhérent à l'application, auquel cas il est nécessaire de se contenter de cette faible efficacité? Cette unique valeur n'est pas suffisante pour statuer sur la qualité obtenue. Il faut également tenir compte des possibilités intrinsèques de parallélisation de l'application.

C.1.1 Loi d'Amdahl

Une application est rarement totalement parallélisable. Il existe toujours une portion de code purement séquentielle, par exemple la lecture d'un fichier, l'affichage écran des résultats, la nécessité de posséder un certain nombre de résultats intermédiaires avant de débuter une phase de calcul.

A partir de cette constation, Amdahl définit s et p, le temps passé par un processeur unique pour exécuter le code séquentiel et le code *parallélisable* [Amda67]. On définit également la fraction séquentielle par $f = s/T(1) = s/(s+p)$.

On a donc $T(1) = s+p$ et $T(n) = s+p/n$, d'où

$$Acc(n) = \frac{n}{(n-1) \cdot f + 1} \text{ et } Eff(n) = \frac{1}{(n-1) \cdot f + 1}$$

On retire deux informations de ces valeurs :

1. Il est inutile de vouloir trop paralléliser un problème, car pour n grand, la portion séquentielle devient prépondérante devant la portion parallèle. L'accélération obtenue ne peut de toute façon permettre une exécution plus rapide que le temps s.
2. L'accélération ou l'efficacité décroissent rapidement en fonction de f. Cela impliquerait qu'il est pratiquement impossible d'obtenir de grande valeur d'accélération, sauf dans les applications où $s \approx 0$.

Cette deuxième constatation est contredite par l'expérience, car, pour de nombreux problèmes, de grandes accélérations sont obtenues.

C.1.2 Loi de Gustafson

Pour lever la contradiction précédente, Gustafson remarque que la valeur de p n'est pas indépendante de n [Gust88]. Il indique, qu'en général, un utilisateur profite, par exemple, de l'augmentation de la mémoire globalement disponible sur un multiprocesseur ou de la diminution de temps d'exécution, pour augmenter la taille du problème, en accroissant le nombre de données, la finesse de résolution...

La loi d'Amdahl est donc valable dans le cas de problème de *taille fixe*. Gustafson étudie, lui, les applications de *temps fixe*, c'est-à-dire ceux où la durée d'exécution est gardée constante par modification de la taille du problème.

Gustafson redéfinit donc : $T(n) = s + p$ et, par extrapolation, $T(1) = s + n \cdot p$.

On obtient alors :

$$Acc(n) = n + (1 - n) \cdot f \text{ et } Eff(n) = 1 + \left(\frac{1}{n} - 1\right) \cdot f$$

La décroissance de l'accélération est, cette fois, linéaire ce qui semble plus optimiste.

C.2 Mesure de la fraction séquentielle

La partie séquentielle d'une application parallèle est difficilement discernable, car elle peut se *cacher* sous la nécessité de synchronisation entre les processeurs, d'échanges d'informations, de répartition des données et des tâches..., ceux-ci étant fonction du nombre de processeurs.

La mesure expérimentale de f et l'analyse de sa variation, en fonction du nombre de processeurs, permet d'obtenir des indications sur le fonctionnement de l'application [Karp90].

C.2.1 Modèle d'Amdahl

La fraction séquentielle est théoriquement constante dans ce modèle. On peut la mesurer par :

$$f = \frac{\frac{n}{Acc(n)} - 1}{n - 1}$$

Elle définit la fraction séquentielle telle qu'elle serait obtenue sur un monoprocesseur.

Les différentes variations de cette valeur, traduisant le surcoût de la parallélisation, ont été analysées par Karp et Flatt. Par exemple :

- une mauvaise répartition de charges produit une variation irrégulière de f,
- un surcoût de synchronisation produit une augmentation régulière,
- une diminution régulière de la fraction séquentielle (fonctionnement sub-linéaire) peut-être liée à une diminution des coûts de gestion ou d'accès aux données lorsque n augmente (par augmentation du nombre de caches mémoire)...

C.2.2 Modèle de Gustafson

On obtient dans ce modèle : $f = \frac{n - Acc(n)}{n-1}$. Soit : $f_{Gustafson} = f_{Amdahl} \cdot Acc(n)$.

Dans le modèle de Gustafson, le rapport entre la portion séquentielle et la portion parallèle n'est plus constante, puisque $T(1) = s + n \cdot p$. La multiplication par $Acc(n)$ permet donc d'obtenir une valeur de fraction séquentielle théoriquement constante, mesurant cette valeur sur n processeurs. Les mêmes analyses de variation sont alors possibles.

ANNEXE

Topologie en géométrie discrète

Connectivité de 2 voxels

Un voxel étant un élément de l'espace discret, il est repéré par un triplet de coordonnées entières. La connectivité est une notion représentant le type d'adjacence entre deux voxels voisins.

Autour d'un voxel, on peut définir 3 ensembles de voisins, suivant la façon dont l'on passe de ce voxel aux autres [Kong89] :

- par les faces (Figure 8.1.a); le voxel central et un de ses voisins sont dits en connectivité 6.
- par les faces et les arêtes (Figure 8.1.b); le voxel central et un de ses voisins sont dits en connectivité 18.
- par les faces, les arêtes et les sommets (Figure 8.1.c); le voxel central et un de ses voisins sont dits en connectivité 26.

Une définition formelle peut être trouvée dans [Andr92] et [Vida92].

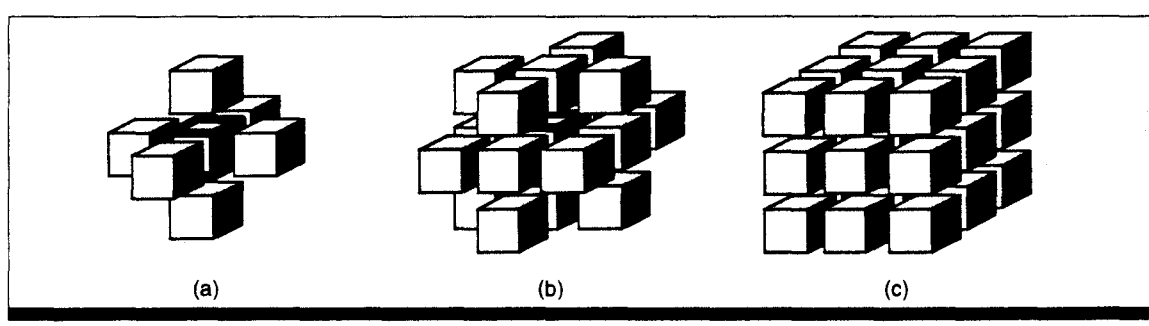


Figure 8.1 ***Voisinage dans l'espace discret.*** *(a) - 6 voisins (b) - 18 voisins (c) - 26 voisins.*

Chemin et ensemble k-connexe

La connectivité entre voxels d'une même surface ne représente qu'une propriété locale. Pour caractériser un ensemble, on définit la *k-connexité* qui prend en compte la connectivité de tous les voxels le composant.

- Un chemin 3D entre un voxel A et B est dit *k-connexe* s'il existe une suite de voxels permettant de passer de l'un à l'autre, tels que deux voxels adjacents du chemin sont en connectivité k (Figure 8.2).
- On définira de la même manière un ensemble k-connexe de voxels. Dans un tel ensemble, deux voxels adjacents quelconques sont en connectivité k (Figure 8.3).
- On dira d'un ensemble est *connecté* s'il est 6-connexe, 18-connexe ou 26-connexe. Dans le cas contraire, on dira qu'il est *déconnecté*.

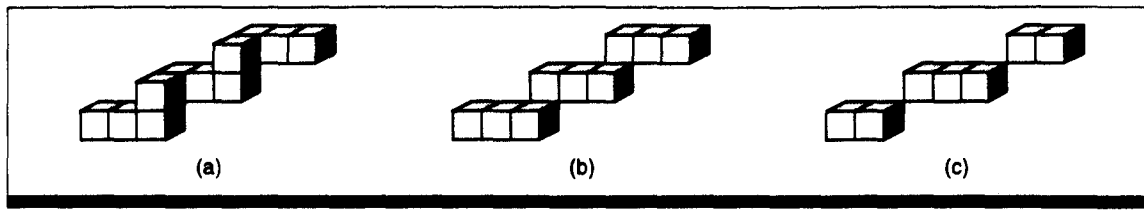


Figure 8.2 ***Chemin 3D k-connexe.*** *(a) - 6-connexe (b) - 18-connexe (c) - 26-connexe.*

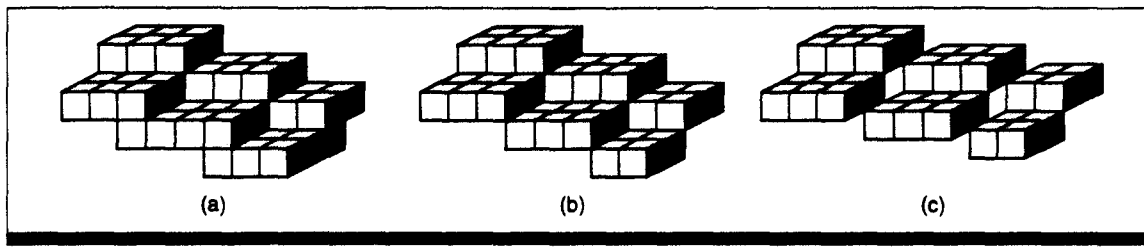


Figure 8.3 ***Ensemble k-connexe.*** *(a) - 6-connexe (b) - 18-connexe (c) - 26-connexe.*

Trou k-connexe

La connexité d'une surface n'est pas suffisante pour la définir complètement. En effet, on peut s'apercevoir sur la Figure 8.3.c qu'il peut exister des *trous* au sein des surfaces.

Lors de la mise en oeuvre d'un algorithme de Lancer de Rayon Discret, les rayons sont eux-mêmes discrètisés. Il faut donc assurer qu'un rayon ne puisse pas *accidentellement* traverser une surface trouée. Cet effet est appelé effet *passe-muraille* [Vida92].

- On dit qu'une surface possède un trou k-connexe s'il existe un rayon k-connexe pouvant la traverser sans intersection, i.e. sans voxels en commun (Figure 8.4).

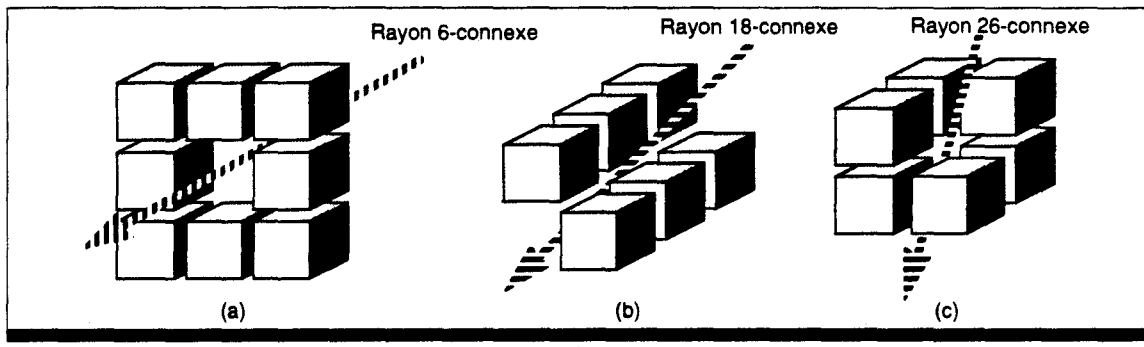


Figure 8.4 ***Trou k-connexe.*** *(a) - 6-connexe (b) - 18-connexe (c) - 26-connexe.*

- Connaissant la connexité du rayon et de la surface, il est possible de déterminer les configurations présentant cet effet passe-muraille (Table 8.1). En fonction de la connexité choisie pour effectuer le lancer de rayon, on peut donc déterminer la connexité nécessaire lors de la voxelisation des objets. Inversement, si l'on connaît la connexité fournie par un algorithme de voxelisation, on peut définir la connexité à utiliser pour le lancer de rayon.

  En tout état de cause, il est intéressant, voire même important, d'avoir des algorithmes de voxelisation pour lesquels on peut fixer la connexité.

Connexité rayon	Connexité surface 6	18	26
6	NON	NON	NON
18	NON	OUI	OUI
26	NON	OUI	OUI

Table 8.1 ***Effet passe-muraille suivant la configuration des connexités.***

Equation diophantienne

Nous introduisons ici la définition et l'utilisation des équations diophantiennes.

Dans l'espace continu $\mathcal{R}^3$, une surface 3D peut-être définie par une équation implicite $f(x, y, z) = 0$, prenant ses valeurs dans $\mathcal{R}^3$. Tout triplet $(x, y, z) \in \mathcal{R}^3$ vérifiant l'équation définit la position d'un point appartenant à la surface.

Dans l'espace discret, les points appartenant à une surface sont de coordonnées entières. Une équation à coefficients entiers pour laquelle on n'autorise que des solutions dans $\mathcal{Z}^n$ est appelée *équation diophantienne*.

Cependant, il est évident que si l'on ne considère que les voxels solutions de l'équation, alors la surface obtenue sera fortement trouée. Par conséquent, il nécessaire d'autoriser la prise en compte de voxels situés légèrement en dehors de la surface réelle.

Dans la pratique, on peut considérer que l'on définit une enveloppe épaisse de la surface, et que l'on accepte tous les voxels situés dans cette enveloppe. Une telle surface est représentée par deux inéquations diophantiennes : $0 \leq f(x, y, z) \leq \omega$. L'*épaisseur arithmétique* ω permet de fixer l'épaisseur de l'enveloppe (Figure 8.5), ce qui détermine la connexité de la surface.

L'équation implicite d'une surface discrète est généralement notée : $f(x, y, z) \in [o, \omega[$

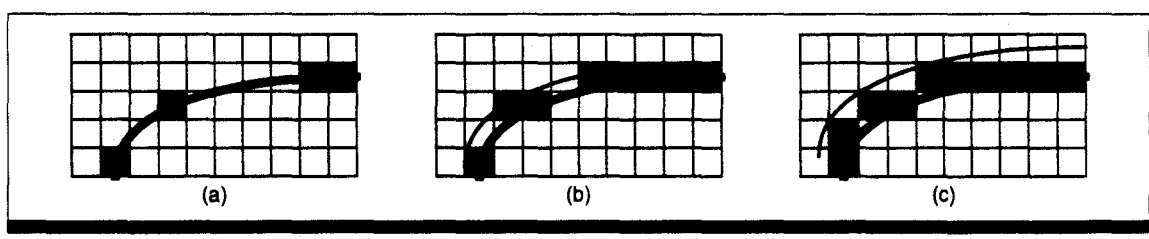
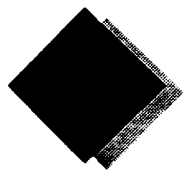


Figure 8.5 ***Effet de l'épaisseur arithmétique sur le choix des voxels composants une surface.*** *(a) - $\omega_a = 0$ (b) - $\omega_b > 0$ (c) - $\omega_c > \omega_b$*

ANNEXE



Tracé de droite 3D par construction

Dans cette annexe nous présentons l'algorithme de suivi de rayons par construction.

Sur une machine voxel à organisation $2D$, on ne dispose que d'une unité de calcul pour tous les voxels situés sur la même position (x, y). Il faut éviter autant que possible toute interpolation en Z. Pour accélérer le suivi des rayons sur ce type d'organisation, nous avons défini un algorithme de tracé de droite 3D par construction, i.e. qui calcule directement la position avant et arrière des paliers constituant la droite.

Une version de cette méthode a également été utilisée pour la voxelisation de droite 3D.

E.1 Principe de l'algorithme

Une droite 2D est formée de paliers de deux longueurs possibles disposés à intervalle régulier, ce qui n'est pas le cas des droites 3D tracées par les algorithmes classiques. B.Vidal a proposé de définir une droite 3D par utilisation de deux des projections sur les plans principaux [Vida92]. Nous allons utiliser ce principe : on construit la projection sur le plan XY par une version 2D de l'algorithme d'Amanatides et Woo [Aman87]; la projection sur le plan XZ ou YZ, suivant l'orientation de la droite, est tracée par paliers.

Ces deux algorithmes sont mixés afin de construire la droite 3D : à chaque changement de coordonnées X ou Y, un palier est tracé.

Technique de construction des paliers en 2D

On trace un palier en X pour chaque valeur de Y (Figure E.1). On peut montrer qu'il existe deux longueurs de paliers possibles, suivant la valeur de la pente de la droite.

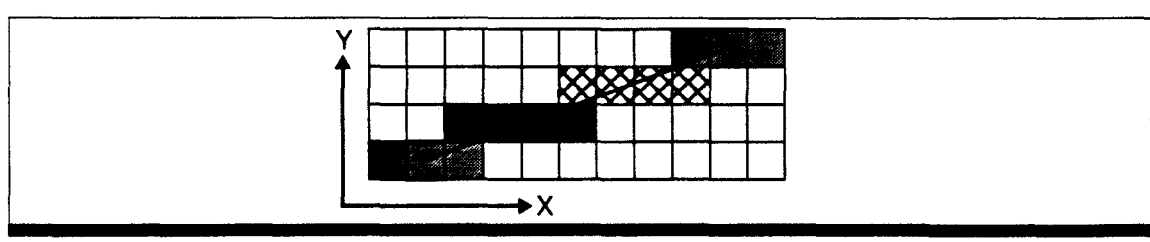


Figure E.1 ***Paliers générés par la méthode de construction.***

Posons $P = \lfloor \Delta_X / \Delta_Y \rfloor$, la partie entière de la pente et $E = \Delta_X \bmod \Delta_Y$, le reste de la division entière. Les deux longueurs de paliers sont respectivement $P+1$ et $P+2$ pour une droite en connexité 6, le palier initial (i.e. pour le point initial de la droite) étant de longueur $P+1$.

Le tracé incrémental de la droite consiste en une modification de l'algorithme de Bresenham. On calcule une erreur directement en fin de chaque palier, et en fonction de cette erreur, on choisit la longueur du palier suivant :

L'erreur commise au point initial est E. A chaque incrément en Y, cette erreur est augmentée de la valeur de $2E$. Lorsqu'elle devient supérieure à Δ_Y, alors un palier de longueur $P+2$ est tracé, et l'erreur est diminuée de la valeur $2\Delta_Y$.

Pour équilibrer le motif de la droite, on trace en fait un demi-palier au point initial.

Algorithme de construction 2D

L'algorithme associé à cette méthode est le suivant :

```
/* Initialisation */
ΔX <- Xf-Xi ; ΔY <- Yf-Yi

Si (ΔY = 0) alors
|     P <- 2×ΔX ; E <- 0;
Sinon
|     P <- ⌊ΔX/ΔY⌋ ; E <- ΔX mod ΔY
FinSi

/* Détermination Palier et Erreur Initiale */
Si (P est impair) alors
|     PCour <- (P+1)/2 ; ECour <- E - 2·ΔY
Sinon
|     PCour <- P/2 ; ECour <- E - ΔY
FinSi

/* Tracé */
Pour y de Yi à Yf
|     Tracer Palier de Longueur PCour+1
|     ECour <- ECour+2E
|     Si (ECour < 0) alors
|     |     PCour <- P
|     Sinon
|     |     PCour <- P+1 ; ECour <- ECour-2ΔY
|     FinSi
FinPour
```

E.2 Mise en oeuvre

Comme indiqué précédemment, l'algorithme 2D d'Amanatides et Woo est utilisé pour le tracé de la projection sur le plan XY.

Le choix du plan XZ ou YZ, pour la définition des paliers, est fonction de la pente de la droite. Si $\Delta_X \geq \Delta_Y$, on choisit le plan XZ, sinon le plan YZ.

Si le plan utilisé pour la construction est le plan XZ, un nouveau palier doit être calculé sur tout changement de la coordonnée X. De même pour le plan YZ on calcule un palier à chaque incrément en Y.

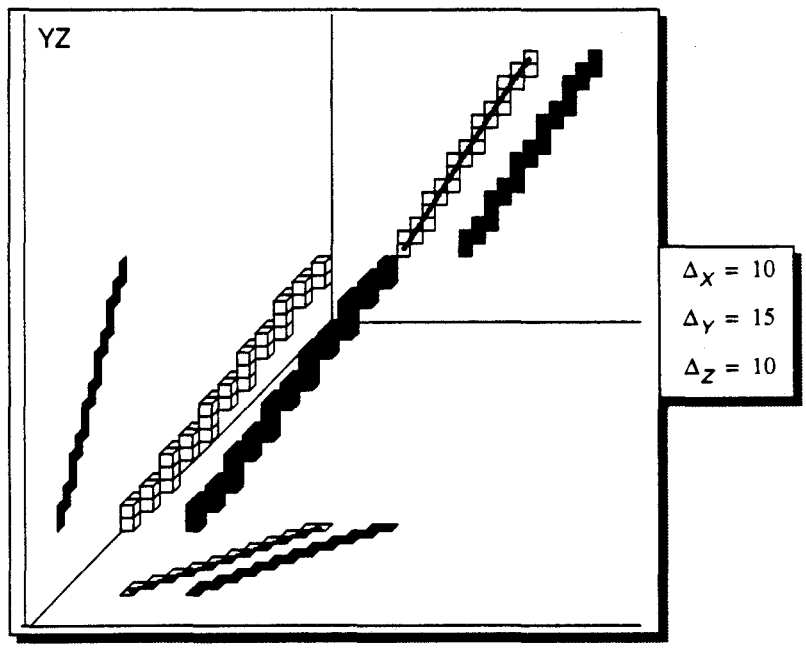
L'algorithme général possède la structure suivante :

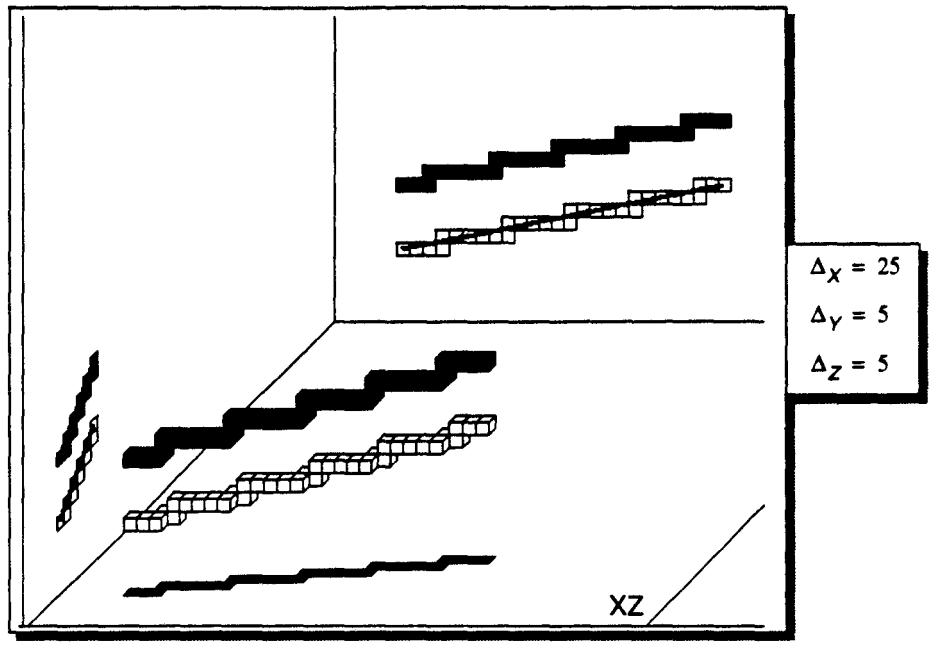
```
TantQue (Non Fin Tracé)
|     Tracé du Palier
|     Suivant (Erreur Amanatides)
|     |     Incrémentation en X
|     |     Si (Palier dans le plan XZ) alors
|     |     |     Calcul nouveau Palier
|     |     FinSi
|     Sinon
|     |     Incrementation en Y
|     |     Si (Palier dans le plan YZ) alors
|     |     |     Calcul nouveau Palier
|     |     FinSi
|     FinSi
FinTantQue
```

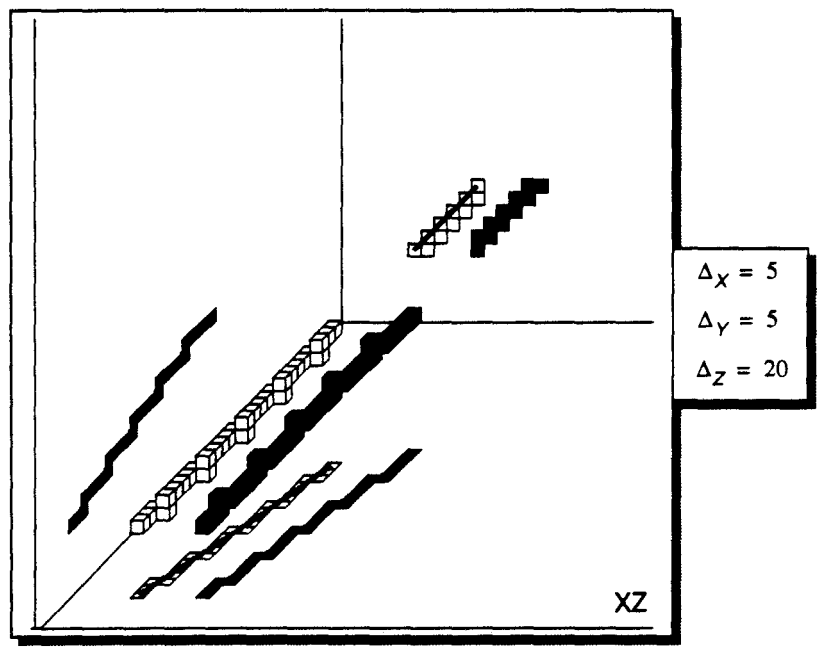
E.3 Résultats

Nous donnons ci-après le résultat de l'application de cet algorithme pour un certain nombre de droites 3D.

Les droites blanches sont produites par la version 3D de l'algorithme d'Amanatides et Woo. Les droites noires sont calculées par notre algorithme. Nous avons également représenté les projections sur les trois plans principaux. Le plan annoté correspond au plan utilisé pour la construction des paliers.



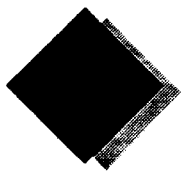




On remarque aisément quelques différences de choix de voxels lorsque l'on se situe en fin d'un palier. Cela provient du fait que l'on ne peut modifier les coordonnées (x, y) en cours de tracé d'un palier. Cependant ces positions étant indéterminées (i.e l'une ou l'autre est valable), ces différences n'ont pas d'influence sur la qualité de la droite tracée.

L'algorithme de construction des paliers est en moyenne 40% plus lent que l'algorithme d'Amanatides, si l'on effectue le remplissage des paliers, alors qu'il est 10% plus rapide si les paliers ne sont pas tracés. Ce faible gain s'explique aisément par le fait qu'en moyenne les paliers sont de très faible longueur (cf § 5.3.5 - p. 121).

ANNEXE



Approximations pour le calcul de l'éclairement local

L'éclairement local nécessite deux opérations très coûteuses : la normalisation de vecteur (qui nécessite une division par une racine carrée) et l'élévation à la puissance (pour le calcul de l'intensité spéculaire).

Pour diminuer le temps d'exécution de ces traitements, nous avons recherché des méthodes d'approximation permettant de réduire notablement le coût de ce traitement.

Nous jugerons également de la qualité de l'éclairement obtenu en utilisant ces méthodes afin de prendre la décision de leur mise en oeuvre ou non.

Critère de comparaison

La comparaison d'une approximation avec une fonction réelle est toujours délicate. Le critère que nous prenons consiste à calculer l'erreur absolue à pleine échelle :

Soit $f(x)$, la fonction réelle, $g(x)$ l'approximation de $f(x)$, et M la valeur maximale de $f(x)$ sur l'ensemble de définition considéré.

L'erreur absolue à pleine échelle est définie par : $E(x) = \frac{g(x) - f(x)}{M}$.

On peut ainsi connaître le nombre de bits significatifs fournis par la fonction $g(x)$.

F.1 Calcul de l'inverse de la racine carrée

Avant d'étudier l'approximation de la fonction $1/\sqrt{x}$, nous commencerons par l'étude de la fonction $\sqrt{x}$.

Calcul exact de l'extraction de la racine carrée

Les méthodes classiques de calcul de cette fonction effectuent la détermination récursive de chacun des bits du résultat, soit un temps de calcul fonction de la largeur des mots. Un cycle de calcul nécessite 3 additions sur la largeur des mots en entrée.

Nous ne détaillerons pas ces techniques bien connues. On pourra trouver dans [Cowg64] la méthode par divisions successives, et dans [John87] la méthode dichotomique.

Certaines améliorations du temps de calcul d'un digit du résultat ont été proposées par utilisation de la notation redondante en base 2 [Maje85] ou en base 4 [Erce90].

Un réseau cellulaire a été proposé, utilisant la méthode des divisions successives. La cellule de base est constituée d'un simple additionneur 1 bit et d'un multiplexeur. Ce réseau permet également le calcul de x^2. Pour un résultat sur n bits, il nécessite $n(n+1)$ cellules, le délai de réponse étant en $o(n(n+1))$. Le cycle de base est cependant beaucoup plus faible puisqu'il est de l'ordre du temps de transfert de l'additionneur 1 bit [Maji72].

Calcul de la racine carrée par affinement progressif

L'affinement progressif est basé sur la méthode de Newton-Raphson. Une valeur initiale proche du résultat attendu doit être fournie. A partir de celle-ci, un calcul itératif permet d'améliorer le résultat fourni à chaque étape (Figure F.1).

La méthode générique est la suivante :

Soit à calculer le zéro z de la fonction $f(x) = 0$. Si on pose z_i, une valeur donnée proche de z, alors une valeur plus proche du résultat sera, sous certaines conditions de convergence :

$$z_{i+1} = z_i - \frac{f(z_i)}{f'(z_i)}$$

Pour le calcul de $\sqrt{N}$, on prend $f(x) = N - x^2$, ce qui donne : $z_{i+1} = \frac{1}{2} \cdot \left(z_i + \frac{N}{z_i} \right)$

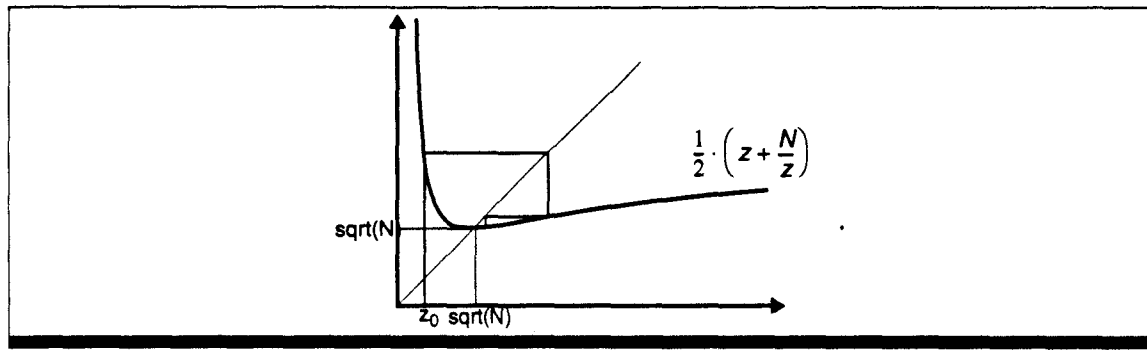


Figure F.1 ***Principe de l'affinement progressif par la méthode de Newton-Raphson.***

On peut également utiliser la fonction $f(x) = 1 - Nx^{-2}$, qui donne : $z_{i+1} = \frac{x_i}{2N} \cdot \left(3N - x_i^2 \right)$. Cette méthode permet d'éviter le calcul de la division, au prix d'une convergence moins rapide [Hwan73].

Le problème majeur de cette technique est la détermination de la valeur z_0 qui doit permettre la convergence. Un grand nombre d'études ont été effectuées afin de déterminer la meilleure valeur possible, aucune méthode simple n'ayant été déterminée. La plus courante est l'utilisation d'une table, qui fournit la valeur initiale en fonction des premiers bits de la variable d'entrée.

Chaque itération double le nombre de bits corrects du résultat (au bit le moins significatif près). Suivant le format des données, et la précision de la valeur initiale, on utilisera de 1 à 4 itérations de correction.

Calcul par méthode CORDIC

La méthode CORDIC permet le calcul de fonctions complexes à l'aide de méthodes simples. Il s'agit également d'un traitement itératif.

Le principe est le suivant :

Soit à calculer $z_0 = f(x_0)$.

On définit une fonction $F(x, y)$ telle que :

1. Il existe une valeur initiale connue y_0 telle que $F(x_0, y_0) = z_0$,
2. Il existe un mécanisme connu de co-transformation d'une paire (x_i, y_i) en (x_{i+1}, y_{i+1}) tel que $F(x_{i+1}, y_{i+1}) = F(x_i, y_i)$,
3. La séquence de transformation de x tend vers une valeur connue x_g, et donc y tend vers y_g.

$F(x, y)$ est également définie telle que $F(x_g, y_g) = y_g$.

Partant de (x_0, y_0), on applique la co-transformation jusqu'à obtenir $x = x_g$.

On a alors $y = y_g = F(x_g, y_g) = z_0$.

Cette méthode permet le calcul de fonctions telles que : $z = w \cdot e^x$, $z = w + ln(x)$, $z = w/x$, $z = w/(\sqrt{x})$ [Chen72].

Si la variable x est codée sur N bits, le traitement nécessite en moyenne $N/4$ itérations.

Approximation de la racine carrée

Toutes les méthodes classiques sont soit longues à évaluer, soit nécessitent des calculs complexes. Nous avons donc recherché une technique permettant la détermination rapide d'une valeur approchée de $\sqrt{x}$, tout en conservant des calculs simples.

Nous avons repris les études de Hashemian qui propose une approximation par morceaux de la fonction $\sqrt{x}$ à l'aide de droites [Hash90]. Nous avons modifié le choix des droites afin d'assurer une continuité d'ordre 0 de la fonction.

Approximation linéaire par morceaux

La fonction $\sqrt{x}$ est approximée par :

$$\begin{cases} f(x) = \dfrac{x}{2^{m+1}} + 2^{m-1} & \text{si } 2^{2m-1} \le x < 2^{2m} \\ f(x) = \dfrac{2x}{2^{m+1}} + 2^{m-1} - 2^{m-2} & \text{si } 2^{2m-2} \le x < 2^{2m-1} \end{cases}$$

Cette approximation donne une valeur exacte pour $x = 2^{2m}$, l'erreur la plus importante étant atteinte en $x = 2^{2m-1}$, où elle vaut $\sim 0.043 \times 2^m$, d'où une erreur maximale de ~4%, soit 4 bits significatifs pour un résultat sur 8 bits (Figure F.2).

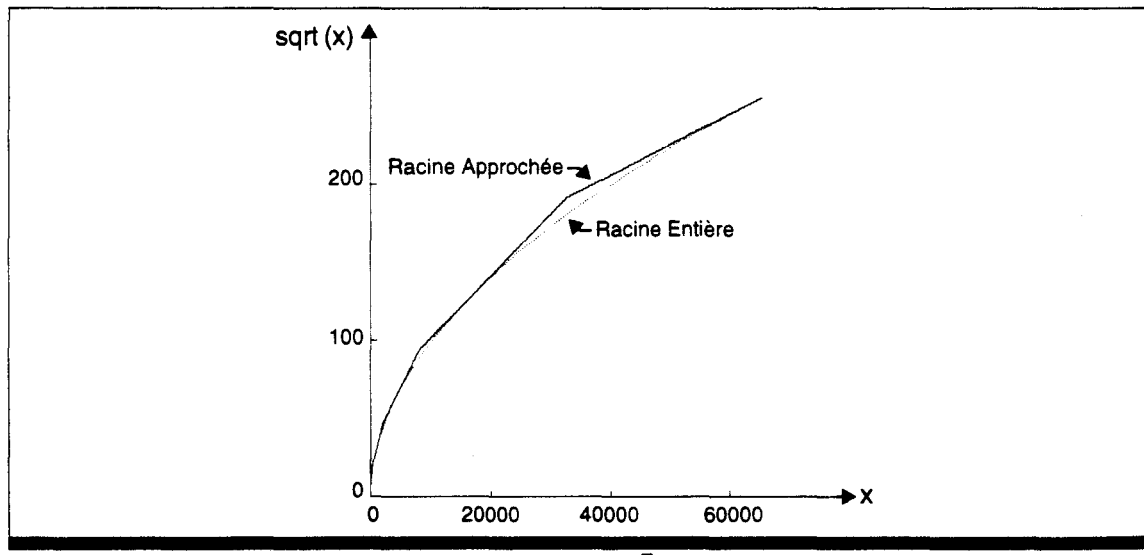


Figure F.2 ***Approximation linéaire par morceaux de $\sqrt{x}$.***

Correction de l'approximation

L'erreur d'approximation de la méthode est trop importante pour être utilisée telle quelle. En effet, si la fonction $\sqrt{x}$ est utilisée pour la normalisation du vecteur directeur d'un rayon, par exemple, une erreur de 4% implique un écart de ~20 voxels pour un rayon traversant complètement un cube de dimension $N_V = 512$. Une correction de valeur est nécessaire. La correction par Newton-Raphson étant trop coûteuse, Hashemian propose une autre technique, plus simple, mais bien entendu convergeant plus lentement.

Hashemian remarque que si $2^{2m-1} \leq x \leq 2^{2m+1}$, alors $\frac{x}{2^{m+1}}$ est une valeur approchée de $\sqrt{x}$ avec une erreur relative de ~40%.

Soit z = Racine_Approchée (x) , alors l'erreur commise par l'approximation est $\sqrt{z^2 - x}$.

La valeur approchée étant toujours supérieure à la valeur vraie, alors une correction possible est :

$$z_{cor} = z - \frac{z^2 - x}{2^{m+1}}$$

Après application de cette correction, l'erreur maximale tombe à 1,5% (6 bits significatifs).

Approximations de l'inverse de la racine carrée

La normalisation de vecteur nécessite une division par la norme du vecteur. Or le calcul d'une division à un coût équivalent à celui d'une racine carrée. Nous avons donc recherché une approximation permettant le calcul direct de $1/\sqrt{x}$.

Approximation linéaire

Nous avons défini une approximation par morceaux à l'aide de droites. Les droites choisies ne permettent pas la meilleur approximation possible, mais elles utilisent des coefficients qui sont des puissances de 2, ce qui permet de remplacer les divisions par des décalages. On obtient :

$$\begin{cases} f(x) = -\frac{x}{2^{3m}} + \frac{1}{2^{m-1}} & \text{si } 2^{2m-1} \leq x < 2^{2m} \\ f(x) = -\frac{2x}{2^{3m}} + \frac{1}{2^{m-1}} + \frac{1}{2^{m+1}} & \text{si } 2^{2m-2} \leq x < 2^{2m-1} \end{cases}$$

Cette approximation donne une erreur maximale de ~4,7% (4 bits significatifs).

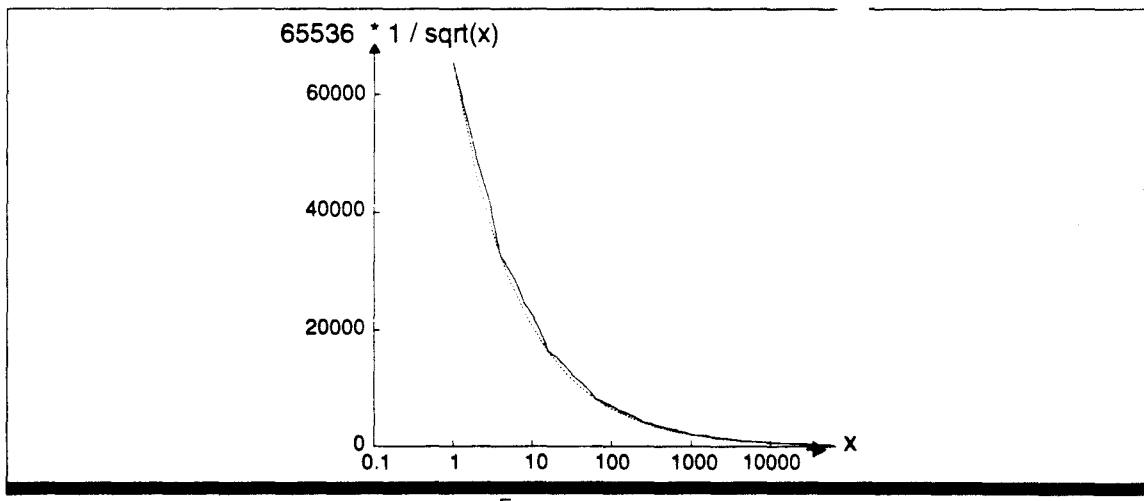


Figure F.3 ***Approximation linéaire de*** $1/\sqrt{x}$.

Correction de l'approximation linéaire

De la même façon, l'erreur de l'approximation est trop importante. Nous ne pouvons plus ici utiliser la correction proposée par Hashemian. Nous reprendrons donc la méthode de Newton-Raphson.

Nous proposons deux fonctions de correction :

- Correction 1

On utilise $f(x) = N - \frac{1}{x^2}$, d'où la fonction de correction : $z_{cor} = \frac{z}{2} \times \left(3 - Nz^2\right)$.

L'erreur maximale est ramenée à -0,61% (6 bits significatifs).

- Correction 2

On utilise $f(x) = 1 - Nx^2$, d'où la fonction de correction : $z_{cor} = \frac{1}{2} \times \left(z + \frac{1}{Nz}\right)$.

L'erreur maximale est ramenée à 0,18% (7 bits significatifs).

La deuxième correction est bien entendu la meilleure, mais la nécessité de calculer une division, alors que nous cherchons justement à l'éviter, nous a fait rejeter cette fonction.

Approximation quadratique

Pour éviter le calcul coûteux de la correction précédente, nous avons remplacé les droites par des formes paraboliques, dans le but d'améliorer l'approximation. Là aussi nous utilisons les formes quadratiques les plus facilement calculables, et non pas les meilleures possibles.

Nous obtenons deux formes possibles :

- Approximation quadratique 1

$$\begin{cases} f(x) = \dfrac{x^2}{2^{5m+1}} - \dfrac{3x}{2^{3m+1}} + \dfrac{1}{2^{m-1}} & \text{si } 2^{2m-1} \le x < 2^{2m} \\ f(x) = \dfrac{3x^2}{2^{5m-1}} - \dfrac{7x}{2^{3m}} + \dfrac{27}{2^{m+3}} & \text{si } 2^{2m-2} \le x < 2^{2m-1} \end{cases}$$

L'erreur maximale de cette approximation est de -1,96% (5 bits significatifs).

- Approximation quadratique 2

$$\begin{cases} f(x) = \dfrac{1}{3}\left[\dfrac{x^2}{2^{5m-1}} - \dfrac{11x}{2^{3m+1}} + \dfrac{13}{2^{m+1}}\right] & \text{si } 2^{2m-1} \le x < 2^{2m} \\ f(x) = \dfrac{1}{3}\left[\dfrac{5x^2}{2^{5m-2}} - \dfrac{11x}{2^{3m-1}} + \dfrac{41}{2^{m+2}}\right] & \text{si } 2^{2m-2} \le x < 2^{2m-1} \end{cases}$$

L'erreur maximale est ici de -0,72% (6 bits significatifs).

Choix de la méthode

Si l'approximation quadratique est meilleure, elle est toutefois équivalente à une approximation linéaire suivie d'une correction. Le calcul mis en oeuvre étant également d'un coût équivalent, nous ne conserverons que l'approximation linéaire, suivie d'une correction, voire de 2 corrections consécutives.

F.2 Calcul de l'élévation à la puissance

L'élévation à la puissance est un des traitements les plus coûteux s'il est exécuté de manière classique. Il peut être effectué par multiplications successives, le coût étant alors fonction de l'exposant utilisé (entre 2 et 100 pour l'intensité spéculaire), soit en utilisant un logarithme, une multiplication puis une exponentiation. Quelle que soit la méthode elle nécessite de lourds calculs que nous avons voulu minimiser en utilisant une approximation par une forme parabolique (Figure F.4).

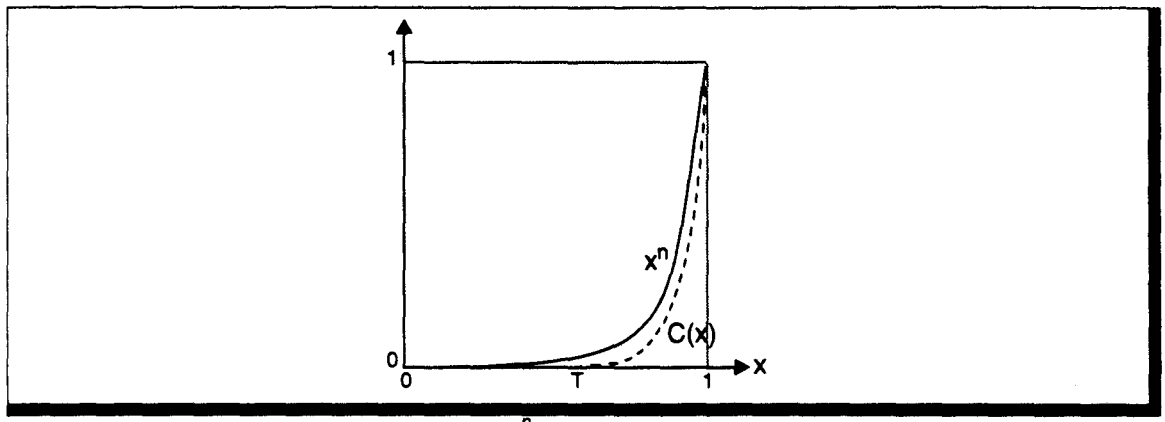


Figure F.4 ***Approximation parabolique de*** x^n.

Nous prenons donc $C(x) = Ax^2 + Bx + C$, avec les contraintes suivantes :

1. $C(T) = 0$
2. $C(1) = 1$
3. $C'(T) = 0$
4. T le plus grand possible tel que $T^{2n} \leq 0.001$, sauf pour $n = 2$, où l'on prend $T^n \leq 0.001$.

Cette dernière contrainte est fixée arbitrairement afin d'obtenir une qualité d'approximation suffisante.

La parabole ainsi définie est : $C(x) = \frac{(x-T)^2}{(1-T)^2}$, et on a $C'(1) = \frac{2}{1-T}$.

La dérivée en 1 est différente de la valeur vraie (qui vaut n), mais les essais que nous avons effectué en contraignant la dérivée en 1 plutôt qu'en T sont de moins bonne qualité.

L'erreur maximale obtenue est de ~10% pour les exposants élevés. Cette erreur peut être jugée importante, mais l'élévation à la puissance utilisée pour l'intensité spéculaire n'est qu'une formulation empirique du phénomène physique sous-jacent. Notre approximation peut être considérée comme *correcte*, puisqu'elle produit visuellement un effet comparable.

Calcul entier

Jusqu'à présent nous avons considéré que $0 \leq x \leq 1$. Cependant nous utilisons des variables entières. En ajoutant certains facteurs multiplicatifs dans l'expression $C(x)$, on permet le calcul entier.

Si, par exemple, $0 \leq x \leq 1024$, et que l'on désire obtenir un résultat sur 8 bits, on prendra (la nouvelle valeur de T est multipliée par 1024) :

$$C(x) = 256\frac{(x-T)^2}{(1024-T)^2}$$

Pour simplifier les calculs, on choisira T, tel que $1024 - T$ soit une puissance de 2. On remplace ainsi la division par un simple décalage. Les valeurs possibles de n de par cette contrainte sont données Table F.1. L'échantillonnage de valeurs nous paraît suffisant.

T	1024 - T	n	T	1024 - T	n
0	2^{10}	2	960	2^6	53
512	2^9	5	992	2^5	108
768	2^8	12	1008	2^4	219
896	2^7	25			

Table F.1 ***Valeurs de T choisies pour une entrée sur 10 bits.***

F.3 Calcul de l'éclairement diffus

On a utilisé les différentes approximations de $1/\sqrt{x}$ pour la calcul de l'intensité diffuse (i.e pour la normalisation des vecteurs $\vec{N}$ et $\vec{S}$).

Les différences visuelles sont difficiles à reproduire dans ce document à cause de leur faible écart. Afin de percevoir la qualité de rendu de chaque méthode d'approximation, nous fournissons en Figure F.5 des diagrammes représentant la valeur de l'intensité diffuse sur une sphère.

On peut observer que l'approximation quadrique est à peine meilleure que l'approximation linéaire avec une correction simple, pour un coût légèrement plus élevé de cette première.

La meilleure qualité est obtenue avec la correction 2, mais celle-ci nécessite une division.

L'aspect visuel de l'approximation linéaire avec la correction 1 est satisfaisante, et le rapport qualité/coût est en faveur de cette méthode.

F.4 Calcul de l'éclairement spéculaire

L'approximation de x^n a été utilisée pour le calcul de l'éclairement spéculaire, en conjonction avec l'intensité diffuse, pour obtenir les diagrammes de la Figure F.6.

Malgré l'erreur assez importante de l'approximation de l'élévation à la puissance, le résultat visuel est correct.

L'aspect est amélioré si l'on utilise un calcul exact de racine carrée, et non une des méthodes définies précédemment.

En tout état de cause, le gain de calcul est tellement important pour les puissances élevées, que nous accepterons les petites pertes de qualité qui en résultent.

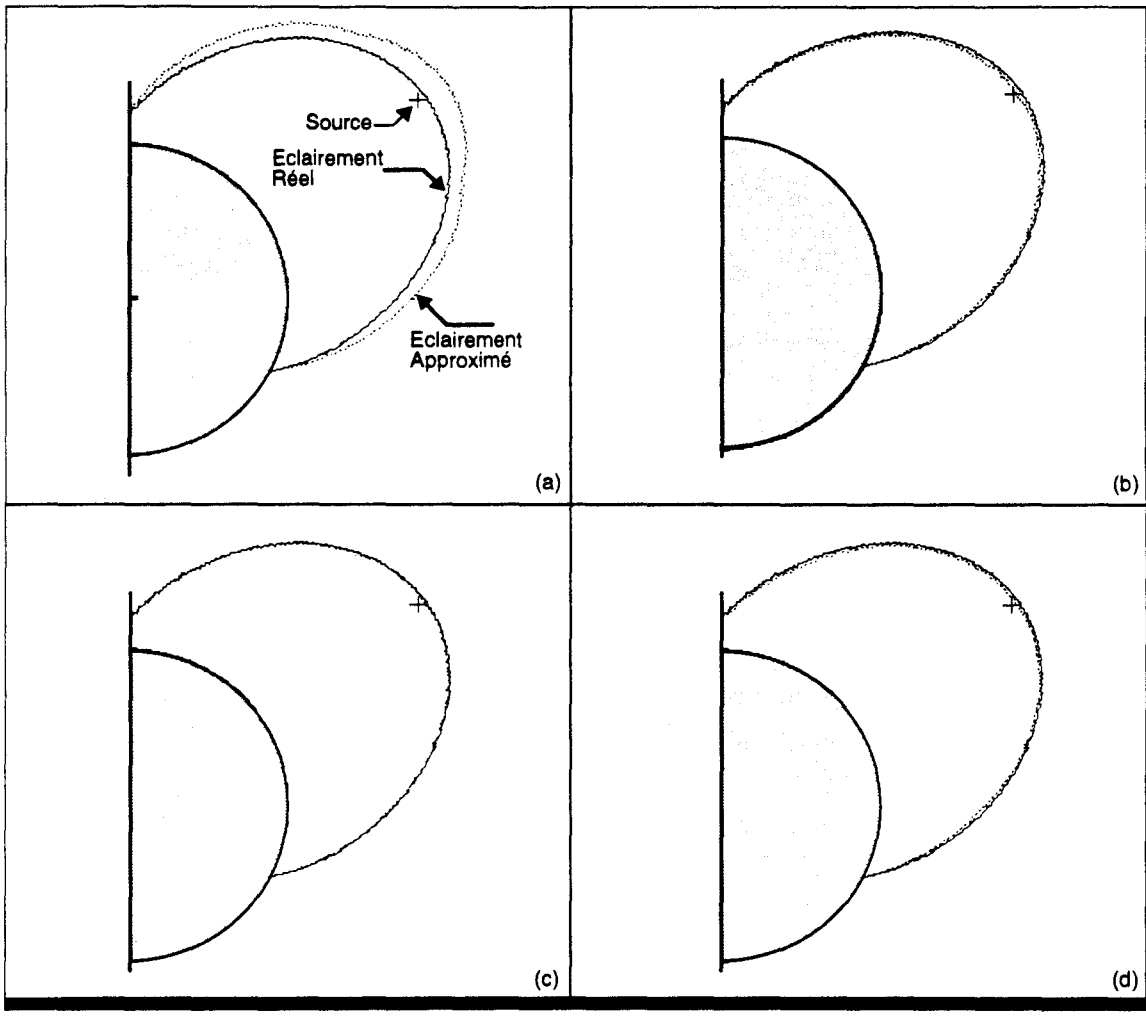


Figure F.5 ***Eclairement diffus sur une sphère.*** *(a) - Approximation Linéaire. (b) - Approximation Linéaire + Correction 1. (c) - Approximation Linéaire + Correction 2. (d) - Approximation Quadratique 1.*

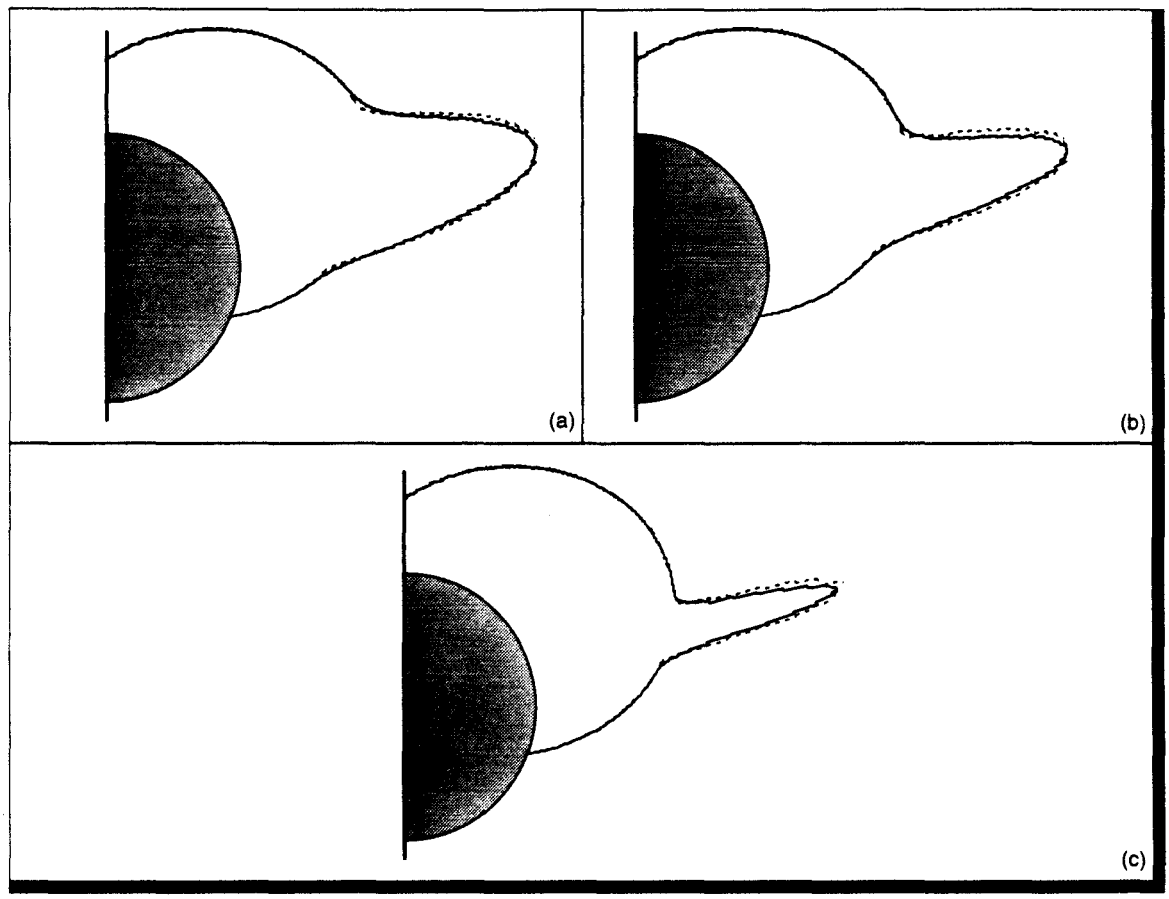


Figure F.6 ***Eclairement diffus + spéculaire sur une sphère.*** *Le trait noir correspond à un calcul direct, le trait grisé à un calcul approché (approximation linéaire de $1/\sqrt{x}$ + correction + approximation de x^n). Les pics correspondent à une insuffisance d'échantillonnage, et non à un défaut de calcul.*
(a) - exposant spéculaire = 5. (b) - exposant spéculaire = 12. (c) - exposant spéculaire = 53.

Bibliographie

Les références données ci-dessous sont classées par ordre alphabétique.

[Aman84] Amanatides.J
Ray Tracing with Cones
ACM Computer Graphics, vol. 18, no. 3, July 1984, pp. 129-135

[Aman87] Amanatides.J, Woo.A
A Fast Voxel Traversal Algorithm for Ray Tracing
Proceedings Eurographics'87, Marechal Editor, Amsterdam, 1987, pp. 3-9

[Amda67] Amdahl.G.M
Validity of the single-processor approach to achieving large-scale computer capabilities
AFIPS Conference Proceedings, vol. 30, AFIPS Press, 1967, pp. 483-485

[Andr92] Andres.E
Cercles discrets et Rotations discrètes
Thèse de doctorat, Université Louis Pasteur, Strasbourg, Novembre 1992

[Arna87] Arnaldi.B, Priol.T, Bouatouch.K
A New Space Subdivision for Ray Tracing CSG Modelled Scenes
IEEE Visual Computer, vol. 3, no. 2, August 1987, pp. 98-108

[Arvo87] Arvo.J, Kirk.D
Fast Ray Tracing by Ray Classification
ACM Computer Graphics, vol. 21, no. 4, July 1987, pp. 55-63

[Atam89] Atamenia.A
Architectures celluaires pour la synthèse d'images
Thèse de doctorat, Université de Lille I, Juin 1989

[Bado90] Badouel.D
Schémas d'exécution pour les machines parallèles à mémoire distribuée. Une Etude de Cas: Le Lancer de Rayon
Thèse de Doctorat. Institut de Formation Supérieure en Informatique et Communication, Rennes. 1990

[Bado91a] Badouel.D, Priol.T
An Efficient Parallel Ray Tracing Scheme for Highly Parallel Architectures
Advances in Computer Graphics Hardware V, Eurographics Seminars, Grimsdale and Kaufman Editors, Springer-Verlag, 1991, pp. 93-106

[Bado91b] Badouel.D, Wüthrich.C.A, Fiume.E.L
Routing Startegies And Message Contention on Low-dimensional Interconnection Networks
Tech. Rep. 258, Comp. Syst. Research Institute., Univ. of Toronto, December 1991

[Baka92] Bakalash.R, Kaufman.A, Pacheco.R, Pfister.H
An Extended Volume Visualization System for Arbitrary Parallel Projection
Eurographics'92, 7th Workshop on Graphics Hardware, Lister Editor, September 1992, pp. 64-69

[Beck63] Beckmann.P, Spizzichino.A
The Scattering of Electromagnetic Waves from Rough Surfaces
Macmillan, New York, 1963

[Berg93] Bergad.K
Lancer de Rayon sur Architecture SIMD
Mémoire de D.E.A., Université de Lille I, Juillet 1993

[Blin77] Blinn.J.F
Models of Light Reflection for Computer Synthesized Pictures
ACM Computer Graphics, vol. 11, no. 2, Summer 1977, pp. 192-198

[Bres65] Bresenham.J.E.
Algorithm for Computer Control of a Digital Plotter
IBM Systems Journal, vol. 4, no. 1, 1965, pp. 25-30

[Brus86] Brusq.R
Ray-Tracing Image Synthesis: the CRISTAL Machine, Results and Perspectives
Deuxième Colloque Image, Nice, Avril 1986, pp. 404-410

[Catm75] Catmull.E
Computer Display of Curved Surfaces
Proc. of IEEE Conf. Comput. Graphics Pattern Recognition Data Struct., May 1975, p. 11

[Caub88] Caubet.R, Duthen.Y, Gaildrat.V
VOXAR: A Tridimensional Architecture for Fast Realistic Image Synthesis
New Trends in Computer Graphics, Thalmann and Thalmann Editors Springer-Verlag, 1988, pp. 135-149

[Chen72] Tien Chi Chen
Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots
IBM Journal of Research and Development, July 1972, pp. 380-388

[Clea86] Cleary.J.G, Wyvill.B.M, Birtwistle.G.M, Vatti.R
Multiprocessor Ray Tracing
Computer Graphics Forum, vol. 5, no. 1, Mars 1986, pp-3-12

[Cohe90] Cohen.D, Kaufman.A, Bakalash.R, Bergman.S
Real Time Discrete Shading
IEEE Visual Computer, vol. 6, no. 1, February 1990, pp. 16-27

[Cohe91] Cohen.D, Kaufman.A
Scan-Conversion Algorithms for Linear and Quadratic Objects
Volume Visualization, A.Kaufman Editor, IEEE Computer Society Press, Los Alamitos, August 1990, pp. 280-301

[Cook81] Cook.R.L, Torrance.K.E
A Reflectance Model for Computer Graphics
ACM Transaction on Graphics, vol. 1, no. 1, January 1982, pp. 7-24

[Cowg64] Cowgill.D
Logic Equations for a Built-In Square Root Method
IEEE Transactions on Electronic Computers, April 1964, pp. 156-157

[Dall87] Dally.W.J, Seitz.C.L
Deadlock-Free Message Routing in Multiprocessor Interconnection Networks
IEEE Transactions on Computers, vol. C-36, no. 5, May 1987, pp. 547-553

[Dall90] Dally.W.J
Network and Processor Architecture for Message-Driven Computers
Chapter 3, in VLSI and Parallel Computation, Suaya and Birtwistle Editors, Morgan Kaufmann Publishers, 1990

[Delf93] Delfosse.J
Discrétisation d'Objets Graphiques
Mémoire de D.E.A, Université de Lille I, Juillet 1993

[Devi89] Devillers.O
Tools to Study the Efficiency of Space Subdivision Structures for Ray Tracing
Actes de PIXIM 89, Gagalowicz Editor, Paris, 1989, pp. 467-481

[Dipp84] Dippé.M, Swensen.J
An Adaptative Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis
ACM Computer Graphics, vol. 18, no. 3, July 1984, pp. 149-158

[Erce90] Ercegovac.M.D, Lang.T
Radix-4 Square Root Without Initial PLA
IEEE Transactions on Computers, vol. 39, no. 8, August 1990, pp. 1016-1024

[Fole90] Foley.J, Van Dam.A, Feiner.S, Hughes.J
Computer Graphics: Principles and Practices (second edition)
The Systems Programming Series, Addison Wesley 12110, 1990

[Frie85] Frieder.G, Gordon.D, Reynolds.R.A
Back-to Front Display of Voxel-Based Objects
IEEE Computer Graphics and Applications, vol. 5, no. 1, January 1985, pp. 52-60

[Fuch81] Fuchs.H, Poulton.J
Pixel-Planes: A VLSI-Oriented Design for a Raster Graphics Engine
VLSI Design, vol.2, no.3, 1981, pp. 20-28

[Fuji86] Fujimoto.A, Tanaka.T, Iwata.K
ARTS: Accelerated Ray-Tracing System
IEEE Computer Graphics and Applications, vol. 6, no. 4, April 1986, pp. 16-26

[Gbad93] Gbadamosi.B.M
Etude et Conception d'un Simulateur Fonctionnel avec Description Hiérarchique de Modèle
Mémoire de D.E.A., Université de Lille, Juin 1993

[Ghaz92] Ghazanfarpour.D
Visualisation réaliste par lancer de pyramides et subdivision adaptative
Actes de MICAD 92, Edition Hermès, Paris, 1992, pp. 167-180

[Glas84] Glassner.A.S
Space Subdivision for Fast Ray Tracing
IEEE Computer Graphics and Applications, vol. 4, no. 10, October 1984, pp. 15-22

[Glas89] Glassner.A.S, editor
An Introduction to Ray Tracing
Academic Press, London, 1989

[Gold84] Goldwasser.S.M.
A Generalized Object Display Processor Architecture
IEEE Computer Graphics and Applications, vol. 4, no. 10, October 1984, pp. 43-55

[Gold85] Goldwasser.S.M, Reynolds.R.A, Bapty.T, Baraff.D, Summers.J, Talton.D, Walsh.E
Physician's Workstation with Real-Time Performance
IEEE Computer Graphics and Applications, vol. 5, no. 12, December 1985, pp. 44-56

[Gold87] Goldwasser.S.M, Reynolds.R.A
Real-Time Display and Manipulation of 3-D Medical Objects: The Voxel Processor Architecture
Computer Vision, Graphics, and Image Processing, vol. 39, no. 1, July 1987 pp. 1-27

[Gold89] Goldwasser.S.M, Reynolds.R.A, Talton.D.A, Walsh.E.S
High Performance graphics processors for medical imaging applications
Parallel Processing for Computer Vision and Display, Dew, Earnshaw and Heywood Editors, Addison-Wesley, 1989, pp. 461-470

[Gora84] Goral.C.M, Torrance.K.E, Greenberg.D.P., Battaile.B
Modeling the Interaction of Light Between Diffuse Surfaces
ACM Computer Graphics, vol. 18, no. 3, July 1984,pp. 213-222

[Gord85] Gordon.D, Reynolds.R.A
Image Space Shading of 3-Dimensional Objects
Computer Vision, Graphics, and Image Processing, vol. 29, 1985, pp. 361-376

[Gour71] Gouraud.H
Computer Shading of Curved Surfaces
IEEE Transaction on Computers, vol. C-20, no. 6, June 1971, pp. 623-629

[Gree89] Green.S.A, Paddon.D.J
Exploiting Coherence for Multiprocessor Ray Tracing
IEEE Computer Graphics and Applications, vol. 9, no. 11, November 1989, pp. 12-26

[Gust88] Gustafson.J.L
Reevaluating Amdahl's Law
Communications of the ACM, vol. 31, no. 5, May 1988, pp. 532-533

[Hall83] Hall.R.A, Greenberg.D.P
A Testbed for Realistic Image Synthesis
IEEE Computer Graphics and Applications, vol. 3, no. 11, November 1983, pp. 10-20

[Hash90] Hashemian R.
Square Rooting Algorithms for Integer and Floating-Point Numbers
IEEE Transactions on Computers, vol. 39, no. 8, August 1990, pp. 1025-1029

[Heck84] Heckbert.P.S, Hanrahan.P
Beam Tracing Polygonal Objects
ACM Computer Graphics, vol. 18, no. 3, July 1984, pp. 119-127

[Herm79] Herman.G.T, Liu.H.K
Three Dimensional Display of Human Organs from Computed Tomograms
Computer Graphics and Image Processing, vol. 9, no. 1, January 1979, pp. 1-21

[Hwan73] Hwang.W.G
A Recurrence Relation for the Square Root
Journal of Approximation Theory, no. 9, 1973, pp. 299-306

[Jack88] Jackel.D, Strasser.W
Reconstructing Solids from Tomographic Scans - The PARCUM II System
Advances in Computer Graphics Hardware II, Eurographics Seminars, Kuijk and Strasser Editors, Springer-Verlag, 1988, pp. 209-227

[Jess89] Jessel.J.P, Cipres.P, Pitot.P, Caubet.R, Duthen.Y
Une modélisation unifiée intégrant des informations locales: la discrétisation en atomes
Actes de Pixim 89, Gagalowicz Editor, Hermes, 1989

[John87] Johnson.K.C
Algorithm 650: Efficient Square Root Implementation on the 68000
ACM Transactions on Mathematical Software, vol. 13, no. 2, June 1987, pp. 138-151

[Jone88] Jones.D.W
The Ultimate RISC
Computer Architecture News, vol. 16, no. 3, 1988, pp. 48-55

[Karp90] Karp.A.H, Flatt.H.P
Measuring Parallel Processor Performance
Communications of the ACM, vol. 33, no. 5, May 1990, pp. 539-543

[Karp93] Karpf.S
Architectures Massivement Parallèles pour la Synthèse d'Images Temps Réel
Thèse de Doctorat, Université de Lille I, Janvier 1993

[Kauf87a] Kaufman.A
An algorithm for 3D Scan-Conversion of Polygons
Proceedings Eurographics'87, Marechal Editor, Amsterdam, 1987, pp. 197-208

[Kauf87b] Kaufman.A
Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes
ACM Computer Graphics, vol. 21, no. 3, July 1987, pp. 171-179

[Kauf88] Kaufman.A, Bakalash.R
Memory and Processing Architecture for 3D Voxel-Based Imagery
IEEE Computer Graphics and Applications, vol. 8, no. 11, November 1988, pp. 10-23

[Kauf91a] Kaufman.A
The voxblt *Engine: A Voxel Frame Buffer Processor*
Advances in Computer Graphics Hardware III, Eurographics Seminars, Kuijk Editor, Springer-Verlag, 1991, pp. 85-102

[Kauf91b] Kaufman.A, Bakalash.R, Cohen.D
Viewing and Rendering Processor for a Volume Visualization System
Advances in Computer Graphics Hardware IV, Eurographics Seminars, Grimsdale and Strasser Editors, Springer-Verlag, 1991, pp. 171-178

[Kerm79] Kermani.P, Kleinrock.L
Virtual Cut-Through: A New Computer Communication Switching Technique
Computer Networks 3, 1979, pp. 267-286

[Kong89] Kong.T.Y, Rosenfeld.A
Digital Topology: Introduction and Survey
Computer Vision, Graphics, and Image Processing, no. 48, 1989, pp. 357-393

[Lacq92] Lacquement.O
Unité de routage de Messages
Mémoire de Projet VLSI, EUDIL, Université de Lille I, Janvier 1992

[Lapo91] Laporte.H
Etude et Conception d'un composant VLSI dans le cadre du Project IMOGENE: L'Extracteur de Racine Carrée
Mémoire de DEA, Université de Lille I, Juillet 1991

[Latt88] Lattard.D
Architecture Massivement Parallele: Un Réseau de Cellules Intégré pour la Reconstruction d'Images
Thèse de doctorat, Institut National Polytechnique de Grenoble, Novembre 1989

[Lefe92] Lefer.W
Etude de la parallélisation de l'algorithme du lancer de rayon en synthèse d'images
Thèse de doctorat, Université de Caen, Septembre 1992

[Lefe94] Lefevere.V
Architectures spécialisées pour l'éclairement de Phong en temps réel
Thèse de doctorat, Université de Lille, Février 1994

[Lenz86] Lenz.R, Gudmundsson.B, Lindskog.B, Danielson.P.E
Display of Density Volumes
IEEE Computer Graphics and Applications, vol. 6, no. 7, July 1986, pp. 20-29

[Lepr89] Leprêtre.E
Algorithmes Parallèles et Architectures Cellulaires pour la Synthèse d'Images
Thèse de doctorat, Université de Lille I, Juin 1989

[Lind91] Linder.D.H, Harden.J.C
An Adaptative and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes
IEEE Transactions on Computers, vol. 40, no. 1, January 1991, pp. 2-12

[Maje85] Majerski.S
Square-Rooting Algorithms for High-Speed Digital Circuits
IEEE Transactions on Computers, vol. 34, no. 8, August 1985, pp. 724-733

[Maji72] Majithia.J.C
Cellular Array for Extraction of Squares and Square Roots of Binary Numbers
IEEE Transactions on Computers, vol. 21, no. 9, September 1972, pp. 1023-1024

[Maur91] Maurel.H, Moisan.B, Jessel.J.P, Duthen.Y, Caubet.R
Génération de séquences animées rendues par Lanver de Rayons : Le Projet Voxar
Journées Graphiques GrosPlan, 1991, pp. 57-64

[Mead80] Mead.C, Conway.L
Introduction to VLSI Systems
Addison-Wesley, 1980

[Meag85] Meagher.D.J
Applying Solids Processing Methods to Medical Planning
Proceedings NCGA'85, April 1985, pp. 101-109.

[Mois91] Moisan.B, Maurel.H, Duthen.Y, Caubet.R
Un lancer de rayons sur une partition volumique de la scène pour un réseau de processeurs à mémoire locale : le projet Voxar
La lettre du Transputer et des calculateurs distribués, no. 12, Décembre 1991, pp. 9-27

[Moln91] Molnar.S
Image-composition Architecture for Real-Time Image Generation
PhD. Thesis, University of North Carolina, October 1991

[Naru87] Naruse.T, Yoshida.M, Takahashi.T, Naito.S
SIGHT - a Dedicated Computer Graphics Machine
Computer Graphics Forum, vol. 6, no. 4, December 1987, pp. 327-334

[Ngai89] Ngai.J.Y, Seitz.C.L
A Framework for Adaptive Routing in Multicomputer Networks
Computer Architecture News, vol. 19, no. 1, 1989, pp. 6-14

[Nemo86] Nemoto.K, Omachi.T
An Adaptative Subdivision by Sliding Boundary Surfaces for Fast Ray Tracing
Proc. of Graphics Interface'86, May 1986, pp. 43-48

[Nish83] Nishimura.H, Ohno.H, Kawata.T, Shirakawa.I, Omura.K
LINKS-1: a Parallel Pipelined Multimicrocomputer System for Image Creation
Proc. of the 10th Annual Intern. Symp. on Computer Architecture, 1983, pp. 387-394

[Nish85] Nishita.T, Okamura.I, Nakamae.E
Shading Models for Point and Linear Sources
ACM Transaction on Graphics, vol. 4, no. 2, April 1985, pp. 124-146

[Ohas85] Ohashi.T, Uchiki.T, Tokoro.M
A Three-Dimensional Shaded Display Method for Voxel-Based Representation
Proceedings Eurographics'85, Vandoni Editor, Amsterdam, 1985, pp. 221-232

[Paya91] Payan.E
Etude d'une architecture cellulaire programmable: Définition fonctionnelle et Méthodologie de programmation
Thèse de doctorat, Institut National Polytechnique de Grenoble, Juin 1991

[Phon75] Phong.B.T.
Illumination for Computer Generated Pictures
Communications of the ACM, vol. 18, no. 18, June 1975, pp. 311-317

[Pito89] Pitot.P, Caubet.R, Duthen.Y, Gaildrat.V
Le suivi analytique de rayons: Un algorithme incrémental rapide pour la machine Voxar
Actes de Micad 89, Edition Hermes, Paris, 1989, pp. 653-664.

[Pito90] Pitot.P, Moisan.B, Duthen.Y, Caubet.R
A Transputer Based Implementation of the Voxar Project
Proceedings Euromicro'90, 1990, pp. 347-354

[Potm89] Potmesil.M, Hoffert.E.M
The Pixel Machine: A Parallel Image Computer
ACM Computer Graphics, vol. 23, no. 3, July 1989, pp. 69-78

[Poul90] Poulin.P, Amanatides.J
Shading and Shadowing with Linear light Sources
Proceedings Eurographics'90, Vandoni and Duce Editors, Amsterdam, 1990, pp. 377-386

[Prio88] Priol.T, Bouatouch.K
Experimenting With a Parallel Ray-Tracing Algorithm on a Hypercube Machine
Proceedings Eurographics'88, Duce and Jancene Editors, Amsterdam, 1988, pp. 243-259

[Prio89] Priol.T
Lancer de Rayon sur des Architectures Parallèles: Etude et Mise en Oeuvre
Thèse de Doctorat. Institut de Formation Supérieure en Informatique et Communication, Rennes. 1989

[Rubi80] Rubin.S.M, Whitted.T
A 3-Dimensionnal Representation for Fast Rendering of Complex Scenes
ACM Computer Graphics, vol. 14, no. 3, July 1980, pp. 110-116

[Shin87] Shinya.M, Takahashi.T, Naito.S
Principles and Applications of Pencil Tracing
ACM Computer Graphics, vol. 21, no. 4, July 1987, pp. 45-54

[Smit92] Smit.J, Wessels.H.J, Van der Horst.A, Bentum.M.J
On the Design of a Real-Time Volume Rendering Engine
Eurographics'92, 7th Workshop on Graphics Hardware, Lister Editor, September 1992, pp. 70-76

[Snyd87] Snyder.J.M, Barr.A.H
Ray Tracing Complex Models Containing Surface Tessellations
ACM Computer Graphics, vol. 21, no. 4, July 1987, pp. 119-128

[Sull77] Sullivan.H, Brashkow.T.R
A Large Scale Homogeneous Machine
Proceedings 4th Symposium on Computer Architecture, 1977, pp. 105-124

[Torr67] Torrance.K, Sparrow.E
Theory for Off-Specular Reflection from Roughened Surfaces
Journal of the Optical Society of America, vol. 57, 1967, pp. 1105-1114

[Toue81] Toueg.S, Ullman.J.D
Deadlock-Free Packet Switching Networks
SIAM Journal Computer, vol. 10, n0. 3, August 1981, pp. 594-611

[Vida92] Vidal.B
Vers un lancer de rayon discret
Thèse de doctorat, Laboratoire d'Informatique Fondamentale de Lille, 1992

[Verb84] Verbeck.C.P, Greenberg.D.P
A Comprehensive Light Source Description for Computer Graphics
IEEE Computer Graphics and Application, vol. 4, no. 7, July 1984, pp. 66-75

[VoxF90] Reality Imaging Corporation
Voxel Flinger™ System Overview
Reality Imaging Corporation, Solon, 1990

[Warn69] Warnock.J.E
A Hidden Surface Algorithm for Computer Generated Halftone Pictures
Dep. Computer Sciences, University of Utah, Tech. Report 4-15, June 1969

[Warn83] Warn.D.R
Lighting Controls for Synthetic Images
ACM Computer Graphics, vol. 17, no.3, July 1983pp. 13-21

[Whit80] Whitted.T
An Improved Illumination Model for Shaded Display
Communications of the ACM, vol. 23, no. 6, June 1980, pp. 343-349

[Yage92] Yagel.R, Cohen.D, Kaufman.A
Discrete Ray Tracing
IEEE Computer Graphics and Applications, vol. 12, no. 5, September 1992, pp. 19-28

Index des Références bibliographiques

[Lenz86] 62
[Lepr89] 79
[Lind91] 132
[Maje85] F-1
[Maji72] 151, F-1
[Maur91] 69, 70
[Mead80] 49
[Meag85] 62
[Mois91] 37, 69, 70
[Moln91] 81, 83
[Naru87] 34
[Ngai89] 132
[Nemo86] 36
[Nish83] 34
[Nish85] 13
[Ohas85] 62
[Paya91] 114
[Phon75] 9, 15
[Pito89] 27, 69
[Pito90] 70
[Potm89] 35
[Poul90] 13
[Prio88] 36
[Prio89] 33, 37
[Rubi80] 26
[Shin87] 27
[Smit92] 62
[Snyd87] 31
[Sull77] 131
[Torr67] 14, A-2
[Toue81] 131
[Vida92] 86, 95, 99, 112, 141, 144, D-1, D-2, E-1
[Verb84] 13
[VoxF90] 62
[Warn69] 9
[Warn83] 12
[Whit80] 18, 19, 26
[Yage92] 118