Lancer de Faisceaux en Synthèse d'Images

Jean-Marc Hasenfratz

N° d'ordre : 398

Université de Limoges

THESE

pour obtenir le grade de

Docteur de l'Université de Limoges

Discipline Informatique

présentée et soutenue publiquement

par

Jean-Marc HASENFRATZ

le 05 janvier 1998

Lancer de Faisceaux en Synthèse d'Images

Jury composé de :

Professeur Dimitri PLEMENOS, Université de Limoges, Président

Professeur René CAUBET, Université Paul Sabatier de Toulouse, Rapporteur

Monsieur Alain CHESNAIS, Directeur des Opérations Alias/Wavefront, Examinateur

Professeur Djamchid GHAZANFARPOUR, Université de Limoges - ENSIL, Directeur de Thèse

Professeur Pascal LIENHARDT, Université de Poitiers, Rapporteur

SOMMAIRE

INTRODUCTION

On peut affirmer, sans grand risque de se tromper, que le XXI^{ème} siècle sera, entre autres, celui des images virtuelles. Pour s'en convaincre, il suffit de regarder à quel point notre quotidien a changé ces dernières années. La part consacrée aux images de synthèse ne fait que croître. Prenons par exemple les bulletins météorologiques télévisés qui sont présentés à l'aide d'animations 3D, les films utilisant massivement des effets spéciaux informatiques, les publicités, … Bien sûr, toutes les industries sont aussi très friandes d'images de synthèse. Et demain, peut-être nous déplacerons nous dans des mondes virtuels pour faire nos achats, aller au travail ou au cinéma …

Il est clair que les images de synthèse sont promises à un bel avenir mais on en "demandera toujours davantage", on cherchera à être plus rapide, plus précis et plus réaliste. Ce mémoire s'inscrit dans cette évolution et présente nos recherches dans ce domaine. Avant de présenter la démarche suivie, il est nécessaire d'aborder deux problèmes, relatifs à la synthèse d'images. Ces problèmes seront le fil directeur de nos travaux.

Le premier problème est celui de l'*aliassage*. On regroupe dans ce terme trois phénomènes visuels qui nuisent au réalisme des images. Le premier, et le plus connu, est le phénomène des *marches d'escalier* sur les contours des objets. On peut le traiter efficacement avec un lancer de rayons et un suréchantillonnage adaptatif des pixels. Le deuxième est le phénomène des *moirés* sur les textures, il peut être traité en utilisant des textures filtrées a priori en fonction de différents taux de compression. Enfin, le troisième phénomène est la *disparition des petits objets et des petites ombres*. C'est un problème complexe, qu'un lancer de rayons conventionnel ne peut résoudre efficacement. En effet, il est impossible d'assurer qu'aucun objet ne se glisse entre les rayons d'épaisseur infinitésimale lancés. C'est, de plus, un problème important puisqu'il produit des effets involontaires de scintillement de petits objets dans les animations. Notons que les difficultés rencontrées avec de petits objets se retrouvent dans d'autres domaines comme l'acoustique ou les télécommunications. On cherche à calculer dans ces disciplines, avec précision, la propagation des ondes dans des environnements particuliers, par exemple un théâtre dans le cas des ondes sonores, ou une ville lors de la détermination de la visibilité entre deux antennes dédiée à une liaison micro-ondes point à point.

Le deuxième problème est le *temps de calculs* nécessaire pour obtenir une image de synthèse réaliste. Un algorithme tel que le tampon de profondeur ("z-buffer"), en général câblé sur les stations graphiques, permet un calcul très rapide des images et des animations. Malheureusement, des effets tels que la réfraction, la pénombre et l'éclairage diffus sont difficiles à réaliser, voire impossible. Ces manques laissent aux images produites un aspect synthétique et irréel. Pour produire des images plus réalistes, on peut utiliser des algorithmes de lancer de rayons et de calcul de radiosité. Le premier propose un très grand nombre d'effets spéciaux comme la réflexion, la réfraction, la pénombre, le flou de bougé, la profondeur de champs, …, le deuxième crée des éclairages diffus très réalistes. Malheureusement, ces effets, augmentant le réalisme sont coûteux en temps de calculs. On

constate en fait que réalisme et rapidité de calcul sont des critères qui ne vont pas souvent ensemble et qu'il faut la plupart du temps choisir entre les deux.

Notre travail de recherche a donc consisté à trouver et développer des solutions à ces deux problèmes. Une première solution qui résout efficacement les problèmes d'aliassage et en particulier la disparition des petits objets et des petites ombres, est le remplacement des rayons d'épaisseur infinitésimale du lancer de rayons par des rayons volumiques. On parle alors de *lancer de faisceaux*. Cette approche n'est pas nouvelle puisque [AMAN 84] remplace les rayons par des cônes, [HECK 84] les remplace par un seul faisceau volumique et [GHAZ 92] les remplace par des pyramides. Ces propositions résolvent en principe les problèmes d'aliassage mais ne permettent pas toujours des effets de réflexion ou de réfraction réalistes et sont limitées à certains types de scènes. Une approche fondée sur [GHAZ 92] résout tous les problèmes d'aliassage, permet des effets de réfraction et de pénombre au moins aussi réaliste qu'un lancer de rayons et peut être utilisée pour des scènes non polygonales comme les scènes "CSG". Nous proposons, entre autres, dans ce mémoire l'étude détaillée de l'algorithme de [GHAZ 92] et des optimisations et extensions apportées [HASE 96][GHAZ 98].

Une autre solution, pour résoudre les problèmes d'aliassage, est de calculer exactement tous les contours visibles des objets à partir d'un point de vue. On parle alors d'algorithme de *calcul de visibilité exacte*. Pour cela, de manière très schématique, on projette tous les objets sur le plan de l'écran et on les découpe en fonction de leurs parties visibles depuis l'oeil. Cette approche est intéressante parce qu'elle permet de faire abstraction de toutes définitions d'un écran. On peut ainsi modifier la taille de l'image sans la recalculer entièrement et sans perte d'informations. Nous proposons dans ce mémoire un algorithme de calcul de visibilité exacte fondé sur le modèle des cartes généralisées et sur un lancer de faisceaux. Le modèle des cartes simplifie les différents découpages entre polygones et autorise une représentation des contours intégrant les propriétés d'adjacence entre les arêtes. Les faisceaux permettent de positionner très précisément les objets les uns par rapport aux autres et optimisent le nombre d'objets à traiter. Notre solution tient de plus compte des effets d'ombres portées, de réflexion et approche les effets de réfraction.

Une solution pour diminuer les temps de calcul est de diminuer le nombre d'objets à traiter. Pour cela, il existe plusieurs approches comme les volumes englobants, les subdivisions spatiales régulières ou adaptatives, les "BSP", … Mais toutes ces solutions ont leurs limites, en particulier lorsque l'on travaille avec des scènes décrites avec plusieurs dizaines de milliers de polygones ou lors d'animation. Nous proposons une approche qui, plutôt que de chercher les objets visibles, cherche ceux qui sont cachés afin de les exclure des traitements. Ces objets sont en général situés derrière d'autres objets très imposants qui rendent une partie de la scène non visible. Nous proposons donc des solutions pour trouver ces objets imposants et pour déterminer rapidement les objets qu'ils cachent.

Ce mémoire se décompose en cinq chapitres. Dans le 1er chapitre, nous étudions les algorithmes fondés sur des variantes du lancer de rayons et les algorithmes qui utilisent, d'une manière ou d'une autre, des faisceaux. Pour synthétiser la présentation, nous avons classé les différentes approches en quatre catégories :

- les algorithmes fondés sur des ensembles de rayons lancés en même temps ;
- les algorithmes utilisant exclusivement des faisceaux ;
- les algorithmes hybrides utilisant à la fois des rayons et des faisceaux ;
- les algorithmes utilisant ponctuellement les faisceaux pour résoudre un problème.

Le 2ième chapitre présente les résultats de l'analyse et de l'implémentation de l'algorithme de "*lancer de faisceaux pyramidaux*" proposé par [GHAZ 92] pour des scènes polygonales. Nous verrons que les problèmes d'aliassage sont résolus mais que les temps de

calculs restent importants. Une extension de cet algorithme aux scènes non polygonales ("CSG") est présentée.

Le chapitre 3 reprend l'algorithme de "*lancer de faisceaux pyramidaux*" pour améliorer ses performances. Plusieurs points de cet algorithme sont optimisés pour finalement améliorer très sensiblement les temps de calculs. Une nouvelle extension permettant des effets de pénombre avec des sources lumineuses volumiques est décrite.

Le chapitre 4 propose un nouvel algorithme de visibilité. Il est fondé sur les "*cartes généralisées*" [LIEN 89c] et sur un lancer de faisceaux. Cet algorithme permet de produire une description de tous les contours des objets, des ombres, des réflexions et des réfractions sur le plan image. La construction de ces contours est facilitée par le modèle topologique utilisé et en particulier par les propriétés d'adjacence dont on dispose.

Le dernier chapitre présente une méthode permettant de diminuer le nombre d'objets à traiter dans un algorithme de visibilité. Pour cela, des faisceaux sont lancés à partir du point de vue sur des polygones imposants afin de déterminer les objets de la scène manifestement cachés. Cette approche est exploitée dans un algorithme de calcul de la radiosité.

1. LES VARIANTES DU LANCER DE RAYONS

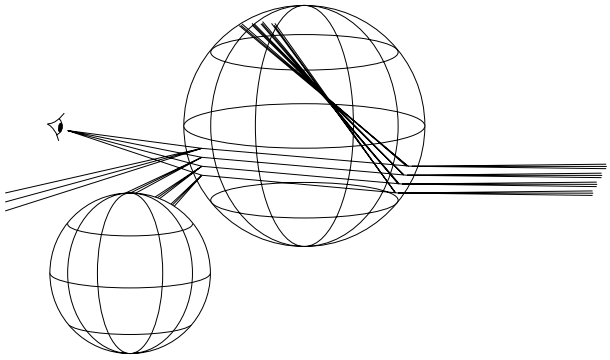


Figure 1.1 : Groupes de rayons rencontrant une sphère réfléchissante et réfractante.

*Le lancer de rayons est l'un des algorithmes considéré comme le plus intéressant pour créer des images de synthèse réalistes. On produit une image en lançant des rayons lumineux de l'oeil en direction de la scène. On lance au minimum un rayon au centre de chaque pixel de l'écran. Les surfaces visibles sont déterminées à l'aide de tests d'intersection entre les rayons et les objets de la scène. Récursivement, on peut tenir compte des effets de réflexion et de réfraction. Si l'on étudie plus attentivement le chemin emprunté par les différents rayons lancés, on constate que nombre d'entre eux suivent approximativement le même parcours (**Figure 1.1**). En fait, dans la plupart des scènes, il est possible de regrouper les rayons empruntant approximativement le même chemin. Cette constatation est résumée, par la suite, par les termes "cohérence de la scène".*

Le lancer de rayons, tel qu'il est proposé dans [WHIT 80], ne tient que très peu compte de cette cohérence. Pourtant, ce n'est qu'à travers elle que l'on peut espérer résoudre les problèmes d'aliassage et probablement aussi celui des temps de calcul. Plusieurs études sont fondées sur cette constatation et essayent d'exploiter au maximum cette cohérence des scènes. Nous nous proposons dans ce chapitre de faire un état de l'art des méthodes qui en ont découlé. Nous verrons que plusieurs approches très différentes existent. Certaines sont des échecs, d'autres semblent très prometteuses. Nous avons classé les solutions proposées en quatre grandes catégories.

- *Les algorithmes exploitant la cohérence de la scène pour traiter des groupes de rayons plutôt que de les considérer les uns indépendamment des autres. Les rayons sont ici d'épaisseur infinitésimale comme ceux utilisés par [WHIT 80]. Cette catégorie comporte seulement trois solutions [SPEE 85] [MÜLL 86] [SHIN 87].*
- *Les algorithmes fondés exclusivement sur des rayons volumiques appelés faisceaux. Les formes utilisées pour ces faisceaux sont diverses ; on trouve, en particulier, des cônes [AMAN 84] et des pyramides de section polygonale quelconque [HECK 84].*
- *Les algorithmes hybrides utilisant à la fois des rayons d'épaisseur infinitésimale et des faisceaux. Deux méthodes relativement proches sont décrites [MARK 90] [GHAZ 92].*

- *Les algorithmes utilisant ponctuellement des faisceaux pour améliorer un traitement précis. C'est le cas de la méthode proposée par [CHOI 92] pour optimiser le calcul des ombres portées.*

Le dernier paragraphe de ce chapitre propose une synthèse de ces méthodes. Nous y reprenons les objectifs visés par chaque méthode et les résultats obtenus. Nous y rappelons aussi les intérêts des diverses méthodes.

1.1 RAYONS D'EPAISSEUR INFINITESIMALE

1.1.1 Prédiction des chemins empruntés par les rayons lancés

Dans son article, [SPEE 85] cherche à limiter le nombre de calcul d'intersection rayon-objet. Pour cela, il construit des *tunnels* englobant un maximum de rayons très proches les uns des autres en profitant de la cohérence de la scène. Les rayons contenus dans ce tunnel rencontrent tous les mêmes objets (***Figure 1.1***), on évite ainsi des calculs d'intersections.

Pour construire ces tunnels, on retient le chemin parcouru par le dernier rayon lancé (comprenant les rayons réfléchis, réfractés et d'ombre) et on construit autour de chaque rayon un cylindre. Ce cylindre est tel qu'il ne touche aucun nouvel objet (***Figure 1.3***). Les cylindres, mis bout à bout, forment le tunnel.

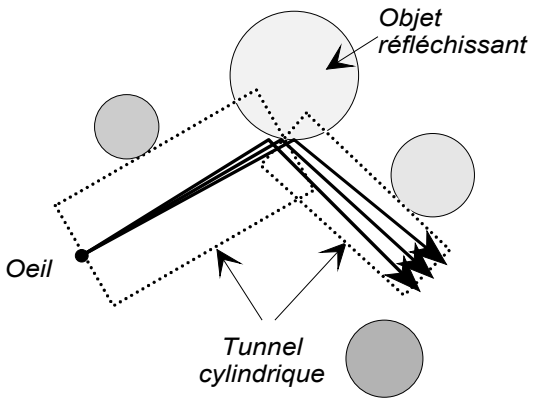


Figure 1.2 : Chemin emprunté par des rayons réfléchis englobés dans un tunnel.

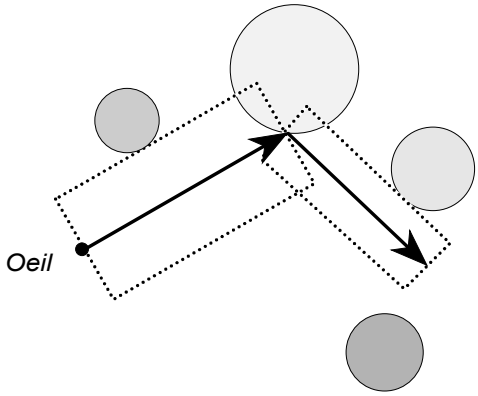


Figure 1.3 : Tunnel cylindrique autour du chemin parcouru par un rayon.

Lors du lancement du prochain rayon, on regarde si le chemin emprunté reste dans le tunnel, c'est-à-dire si les rayons n'ont pas d'intersection avec les cylindres. Si c'est le cas, les objets rencontrés sont les mêmes que ceux rencontrés par le dernier rayon. Il n'est donc pas nécessaire de refaire les calculs d'intersection. Sinon, on calcul les intersections rayons-objets et on construit un nouveau tunnel

Une méthode simple pour trouver les cylindres les plus gros autour de chaque rayon est proposée par [SPEE 85]. On construit deux plans perpendiculaires au rayon donné, P passant par l'origine et P' passant par le premier point d'intersection rayon-objet (***Figure 1.4***). On considère uniquement les objets compris entre P et P'. Le rayon R du cylindre correspond alors à la plus petite distance entre le rayon et les objets considérés.

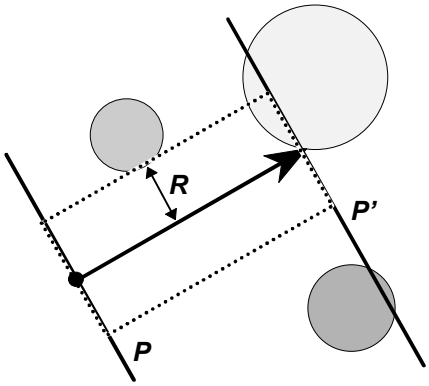


Figure 1.4 : Construction d'un cylindre.

[SPEE 85] ne propose aucune information pour déterminer si un rayon lancé dans un tunnel ne heurte pas ses parois. On peut supposer néanmoins que le passage à un problème en deux dimensions fournit une bonne solution. Malgré des calculs rapides et une logique simple, cette approche ne fournit pas les résultats escomptés. En effet, les tests effectués dans [SPEE 85] sur des scènes contenant un nombre croissant de sphères montrent qu'à partir de 9 sphères le lancer de rayons conventionnel est plus rapide que la solution proposée. Ceci est principalement dû à un déséquilibre entre le temps nécessaire à la construction des différents tunnels et le nombre de rayons qui peuvent profiter de cette structure. L'auteur conclut ainsi : "*Nous avons montré qu'en dépit du degré élevé de cohérence dans une image, l'obligation de maintenir la validité des structures d'intersection fait obstacle à l'obtention de gains importants. Ces résultats donnent à penser que des méthodes algorithmiques plus fondamentales sont nécessaires pour réduire de façon substantielle les coûts de calcul du traçage de rayons lumineux.*".

Cette démarche est globalement un échec, cependant elle met clairement en évidence la dualité entre les coûts de manipulation d'une structure de données et le gain de temps qu'elle peut produire. De plus, elle apporte une solution géométrique permettant de positionner un objet par rapport à un cylindre. Nous verrons, par la suite (***Paragraphe 2.1.1.1***), l'utilité d'une telle solution.

1.1.2 Lancement simultané de tous les rayons à chaque niveau de la récursion

[MÜLL 86] propose un algorithme de lancer de rayons regroupant les rayons en fonction de la profondeur de la récursion. Ainsi, tous les rayons primaires sont traités en même temps puis tous les rayons réfléchis et réfractés du premier niveau de récursion et ainsi de suite. En fait, cela revient à préférer un parcours en largeur plutôt qu'en profondeur, de l'arbre de récursion de [WHIT 80].

1.1.2.1 Hypothèses

La scène est modélisée avec des polygones. A chaque objet est associé un parallélépipède rectangle minimum dont les côtés sont parallèles aux trois plans de référence du repère. Ce parallélépipède correspond à la boîte englobante à l'objet. Toute la scène est délimitée par un parallélépipède rectangle dont les côtés sont aussi parallèles aux trois plans de référence du repère. Ce parallélépipède formera la *boîte englobante principale*. Tous les rayons lancés au cours de l'algorithme sont bornés par cette dernière (***Figure 1.5***).

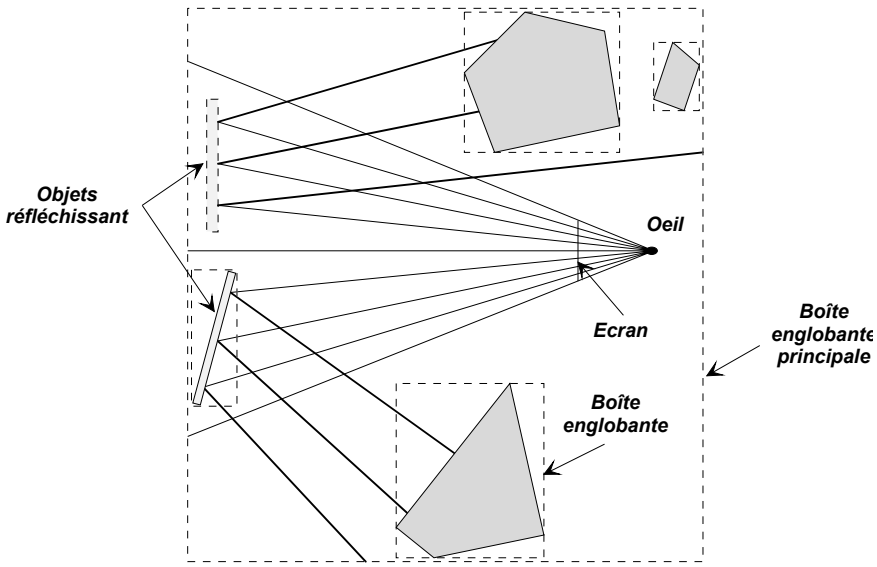


Figure 1.5 : Propagation des rayons primaires et des rayons de réflexion dans la scène.

On dispose d'une structure de données *R* permettant de déterminer rapidement si un rayon traverse une face d'une boite englobante [BENT 75]. Cette structure *R* est fondée sur un arbre bien équilibré [SEDG 83].

1.1.2.2 Principe des plans de balayage

L'idée proposée par [MÜLL 86] est de considérer en même temps tous les rayons primaires, réfléchis et réfractés, à un niveau de la récursion. Pour cela, pour chaque niveau de la récursion, l'espace de la scène est balayé à l'aide de chacun des six plans formant la boîte englobante principale. Supposons que l'on réalise cela pour le plan perpendiculaire à l'axe z et de plus petite coordonnée selon z. Cette face est déplacée dans la direction des z positifs. Durant ce balayage, seuls les rayons primaires, de réflexion et de réfraction ayant une composante positive suivant z sont considérés.

Le déplacement d'un plan est en fait simulé à l'aide de deux listes *RE ("ray event")* et *OE ("object event")* d'événements discrets triés selon z.

RE contient les événements suivants :

a) l'origine des rayons *r*, notée O_r ;

b) les points d'intersection rayon-objet obtenus lors du balayage de l'espace, noté I_r pour la première intersection du rayon *r*.

OE contient les événements suivants :

c) la face avant des boîtes englobantes des objets, c'est-à-dire les premières faces rencontrées lors du balayage, noté *f*.

Les événements de ces listes sont considérés dans l'ordre (les événements de même coordonnés en z sont traités dans n'importe quel ordre). Les traitements correspondant aux différents type d'événements sont les suivants :

a) Le rayon correspondant (en fait son point d'intersection avec le plan de balayage à cette coordonnée z) est inséré dans *R*. Cet événement est supprimé de *RE*.

b) Le rayon correspondant est supprimé de *R*.

c) On utilise la structure *R* pour déterminer les rayons qui traversent la face de la boîte englobante rencontrée lors de cet événement. Pour tous les rayons traversant cette face, un test d'intersection rayon-objet est effectué avec l'objet contenu dans la boîte considérée. Le point d'intersection avec l'objet, s'il existe, est ajouté à la place courante dans *R* avec *b)* pour type d'événement. S'il existe déjà un

point d'intersection pour ce rayon dans *RE*, celui ayant la composante z la plus grande est supprimé.

L'algorithme prend fin, pour un niveau de récursion, lorsque les six plans de balayage ont traversé, dans les trois directions positives et les trois directions négatives du repère, la boîte englobante principale. On traite ainsi tous les rayons quelle que soit leur direction de propagation. Toutes les intersections avec les faces des boîtes englobantes sont déterminées et par conséquent toutes les intersections possibles avec les objets. A chaque niveau de récursion, on dispose de l'ensemble des points d'intersections ainsi que des rayons incidents en ces points. On peut alors répéter l'algorithme de balayage des six plans pour le niveau de récursion suivant. Finalement, on aura parcouru l'arbre de récursion de [WHIT 80] en largeur.

Le ***Tableau 1*** montre l'évolution de l'algorithme en balayant l'espace suivant les coordonnées z croissantes avec le plan XY pour les rayons réfléchis du niveau de récursion proposé dans la ***Figure 1.6***.

Evénements	*Tests effectués*	*Actions*	*RE*	*OE*	*R*
		Initialisation.	O_{r1}, O_{r2}, O_{r3}, O_{r4}, O_{r5}, O_{r6}	*f1, f2, f3, f4, f5*	∅
f1	Pas d'intersection, R est vide.	Suppression de f1 dans OE.	O_{r1}, O_{r2}, O_{r3}, O_{r4}, O_{r5}, O_{r6}	*f2, f3, f4, f5*	∅
O_{r1}		Ajout de r1 dans R et suppression de O_{r1} dans RE.	O_{r2}, O_{r3}, O_{r4}, O_{r5}, O_{r6}	*f2, f3, f4, f5*	*r1*
f2	Pas d'intersection entre r1 et f2.	Suppression de f2 dans OE.	O_{r2}, O_{r3}, O_{r4}, O_{r5}, O_{r6}	*f3, f4, f5*	*r1*
O_{r2} **à** O_{r6}		Ajout de r2 puis r4, r5, r6 et r3 dans R et suppression de O_{r2} puis O_{r4}, O_{r5}, O_{r6} et O_{r3} dans RE.	∅	*f3, f4, f5*	*r1, r2, r3, r4, r5, r6*
f3	Intersection entre r1, r2, r3, r4, r5 et r6 avec f3 ? Seuls r2 et r3 traversent f3.	On calcule les intersections entre r2 et r3 avec l'objet C. Ces deux points d'intersection sont ajoutés dans RE. Suppression de f3 dans OE.	I_{r2}, I_{r3}	*f4, f5*	*r1, r2, r3, r4, r5, r6*
I_{r2} **puis** I_{r3}		Suppression de r2 puis r3 de R.	∅	*f4, f5*	*r1, r4, r5, r6*
f4	Intersection entre r1, r4, r5 et r6 avec f6 ? Seuls r5 et r6 traversent f4.	On calcule les intersections entre r5 et r6 avec l'objet D. Ces deux points d'intersection sont ajoutés dans RE. Suppression de f4 dans OE.	I_{r5}, I_{r6}	*f5*	*r1, r4, r5, r6*
I_{r5} **puis** I_{r6}		Suppression de r5 puis r6 de R.	∅	*f5*	*r1, r4*
f5	Intersection entre r1 et r4 avec f6 ? Aucune intersection.	Suppression de f5 dans OE.	∅	∅	*r1, r4*

Tableau 1 : Evolution de l'algorithme de balayage par plans pour l'exemple de la ***Figure 1.6*** .

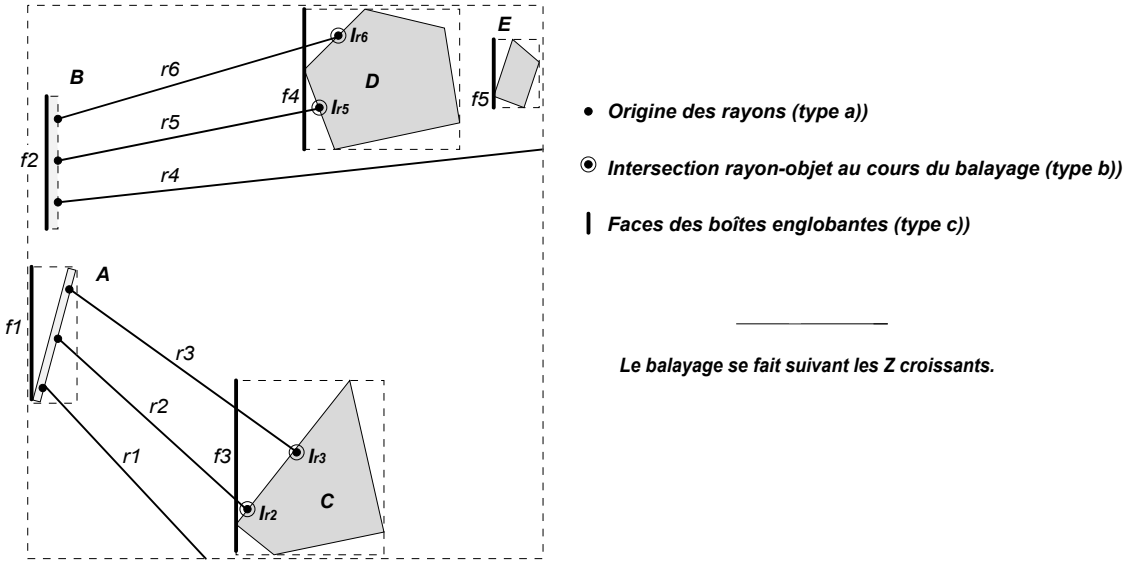


Figure 1.6 : Evénements de type a), b) et c) pour un niveau de récursion.

1.1.2.3 Calcul du rendu

Le calcul du rendu de l'image finale est proche de celui d'un lancer de rayons conventionnel. La principale différence est qu'au lieu de lancer un rayon après l'autre et de calculer la couleur pixel après pixel, on lance ici, tous les rayons en même temps et la couleur des pixels est affinée globalement. En effet, à chaque niveau de la récursion, les points d'intersection sont déterminés et mémorisés. Pour chacun de ces points, il ne reste plus qu'à lancer des rayons vers les sources lumineuses (comme dans un lancer de rayons conventionnel) pour déterminer s'il est à l'ombre ou non.

Le problème de l'aliassage est traité de manière très conventionnelle par un suréchantillonnage local déclenché par un contraste trop important entre des pixels voisins.

1.1.2.4 Discussion

[MÜLL 86] ne conclut pas réellement que son algorithme a des performances meilleures, en temps de calculs, qu'un lancer de rayons conventionnel. Ceci était pourtant l'objectif principal. Par contre, il précise bien que le coût mémoire est très important. Cette approche de parcours en largeur semble relativement limitée puisque le gain de temps n'est pas établi mais que l'augmentation de l'espace mémoire nécessaire est importante.

1.1.3 Faisceau générique et quantification des approximations réalisées

Le *lancer de crayons* proposé dans [SHIN 87] a pour objectif de diminuer les temps de calculs pour produire une image de synthèse et d'augmenter son réalisme en restant fidèle aux lois de l'optique. L'idée est de développer des outils mathématiques permettant de quantifier les erreurs d'approximation commises lors de la propagation des rayons. L'évaluation de cette erreur permet, quand cela est nécessaire, d'appliquer des traitements adéquats pour améliorer la qualité du rendu. Ces traitements font l'objet du ***Paragraphe 1.1.3.3***.

Pour approcher l'erreur commise, [SHIN 87] remplace les rayons conventionnels par des ensembles de rayons ayant la même direction appelés *crayon*. Un crayon est composé d'un *rayon axial* et de *rayons paraxiaux* voisins du rayon axial. Le lancer du rayon axial permet de connaître les caractéristiques optiques (transmission, réflexion et réfraction) du milieu traversé par l'ensemble du crayon. Des calculs complexes, fonctions de ces caractéristiques, permettent alors de définir les contraintes à appliquer aux rayons paraxiaux pour que l'on

puisse approcher les changements de direction du crayon par une transformation linéaire et obtenir un bon réalisme.

La théorie de l'"*approximation paraxiale*" [BORN 59] est utilisée pour représenter, sous la forme d'une matrice 4x4, la transformation d'un rayon traversant un système optique. Notons que cette théorie n'est valable que si les surfaces du système optique sont C1[1] continues.

L'utilisation d'une représentation matricielle du système optique permet de caractériser simplement les contraintes que doivent respecter les rayons paraxiaux pour un bon rendu.

Deux utilisation possibles du lancer de crayons sont présentées : le lancer de rayons fondé sur une représentation matricielle du système optique et une modification du "*lancer de faisceaux sur des objets polygonaux"* [HECK 84].

1.1.3.1 Théorie de "l'approximation paraxiale"

La théorie de "l'approximation paraxiale" [BORN 59] fournit une approximation linéaire des changements de direction d'un rayon dus à la transmission (propagation dans un milieu homogène), à la réflexion ou à la réfraction.

1.1.3.1.1 Définitions

◆ Rayon paraxial

Un *rayon paraxial* est un rayon situé dans le voisinage d'un rayon axial. Il est défini par rapport à celui-ci sous la forme d'un vecteur à quatre dimensions dans un repère lié au rayon axial

Système de coordonnées du rayon paraxial

Dans la ***Figure 1.7***, un système de coordonnées orthogonales x_1, x_2, z, appelé *système de coordonnées du rayon paraxial*, est utilisé pour représenter un rayon paraxial. L'origine *O* est un point du rayon axial et l'axe *z* correspond à la direction du rayon axial. Les axes supportés par x_1 et x_2 sont choisis arbitrairement dans un plan perpendiculaire à l'axe *z*.

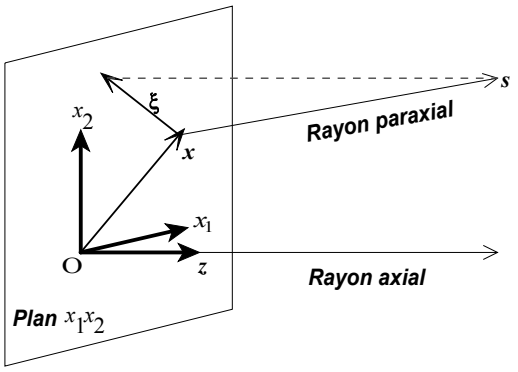


Figure 1.7 : Rayon axial et rayon paraxial.

Vecteur d'un rayon paraxial

Un rayon est généralement représenté par un point par lequel passe un vecteur directeur. Un rayon paraxial est représenté par deux sortes de vecteurs. Un vecteur

[1] On dit qu'une fonction est C1 continue ssi elle et sa dérivé sont continues. Dans notre cas, cela signifie qu'on ne considère pas les bords de polygones ni les surfaces contenant une arête adjacente à des polygones qui ne sont pas dans un même plan.

position $\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ qui correspond au point d'intersection entre le rayon paraxial et le plan $\{x_1, x_2\}$ relativement à *O*, et un "*vecteur directeur*" $\boldsymbol{\xi} = \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix}$ qui correspond à la projection du vecteur directeur unitaire ***s*** du rayon paraxial sur le plan $\{x_1, x_2\}$. En réunissant ces deux vecteurs, un rayon paraxial peut être caractérisé par la donnée du vecteur ψ au point *O* suivant :

$$\Psi = \begin{pmatrix} x \\ \xi \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \xi_1 \\ \xi_2 \end{pmatrix}$$

Ce vecteur ψ est appelé *vecteur du rayon paraxial*.

◆ ***Crayon***

Un crayon est construit autour d'un rayon axial à l'aide d'un ensemble de rayons paraxiaux représentés à l'aide de leur vecteur ψ. En général, le rayon axial et le rayon paraxial sont issus d'une même origine : le point de vue. Un crayon peut donc être représenté facilement par une origine et un angle d'ouverture, c'est-à-dire le degré de déviation maximale des rayons paraxiaux par rapport au rayon axial.

◆ ***Matrice système***

Lorsqu'un rayon traverse différents systèmes optiques, sa direction peut changer en fonction de la réflexion et de la réfraction de ce système. La déviation des rayons paraxiaux par rapport au rayon axial peut être choisie suffisamment petite pour que la transformation subie par le vecteur du rayon paraxial soit considérée comme linéaire. Cette transformation peut alors s'écrire:

$$\psi' = T\psi$$

où ψ est un vecteur d'un rayon paraxial entrant et ψ' un vecteur d'un rayon paraxial sortant d'un système optique. *T* est une matrice 4x4 appelée *matrice système*. Notons qu'un système optique composé de plusieurs sous-systèmes peut être représenté par le produit des matrices systèmes des différents sous-systèmes.

1.1.3.1.2 Matrices systèmes

Les systèmes optiques utilisés sont constitués de régions homogènes séparées par des surfaces lisses. Cette caractéristique des surfaces impose en particulier que l'on ne se place pas sur des arêtes au voisinage desquelles la continuité n'est pas C1. Les phénomènes optiques considérés sont la transmission (aucune déviation), la réflexion et la réfraction. Pour obtenir la matrice d'un système optique, il suffit de connaître les matrices des systèmes correspondant à chaque phénomène et de les multiplier entre elles. En utilisant la loi de Snell, ces différentes matrices prennent l'une des trois formes suivantes :

■ ***Matrice de transmission***

Dans une région homogène, les rayons ne sont pas déviés. La propagation d'un rayon paraxial de $z=0$ à z_0 (***Figure 1.8***) prend la forme :

Equation 1-1 :

$$T = \begin{pmatrix} 1 & 0 & z_0 & 0 \\ 0 & 1 & 0 & z_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

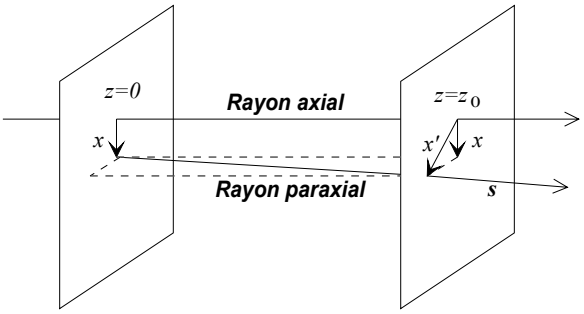


Figure 1.8 : Effet de transmission.

■ Matrice de réfraction

Considérons la surface réfractante Σ de la ***Figure 1.9*** et les indices de réfraction n et n'. Soit O l'intersection entre le rayon incident et la surface Σ. Soient x_1, x_2 et x_1', x_2' les systèmes de coordonnées orthogonales respectivement du rayon incident et du rayon réfracté. Les deux plans $\{x_1, x_2\}$ et $\{x_1', x_2'\}$ sont respectivement perpendiculaires au rayon incident et au rayon réfracté. Soit le système de coordonnées orthogonales (u_1, u_2) tangent à Σ en O. Pour simplifier, nous supposons que $x_2 = u_2 = x_2'$. L'expression de la matrice de réfraction à travers la surface Σ est donnée en approchant la surface Σ par une paraboloïde. En s'inspirant de [BARR 86], [SHIN 87] propose la matrice de réfraction suivante :

Equation 1-2 :

$$T = \begin{pmatrix} \Theta' \, \Theta^{-1} & 1 \\ \left(\Theta'^{\,t}\right)^{-1} hQ\Theta^{-1} & (n/n')\left(\Theta'^{\,t}\right)^{-1}\Theta^{t} \end{pmatrix}$$

où

$$\Theta = \begin{pmatrix} x_1.u_1 & x_2.u_1 \\ x_1.u_2 & x_2.u_2 \end{pmatrix}, \quad \Theta' = \begin{pmatrix} x_1'.u_1 & x_2'.u_1 \\ x_1'.u_2 & x_2'.u_2 \end{pmatrix}, \quad h = \cos\Theta' - (n/n')\cos\Theta$$

Avec un système de coordonnées respectant la ***Figure 1.9***, les matrices Θ et Θ' peuvent être mises sous la forme diagonale suivante :

$$\Theta = \begin{pmatrix} \cos\theta & 0 \\ 0 & 1 \end{pmatrix}, \quad \Theta' = \begin{pmatrix} \cos\theta & 0 \\ 0 & 1 \end{pmatrix}$$

La matrice Q correspond à la courbure de la surface Σ dans le repère u_1, u_2. Par exemple, si Σ est une sphère de rayon r,

$$Q = \begin{pmatrix} 1/r & 0 \\ 0 & 1/r \end{pmatrix}.$$

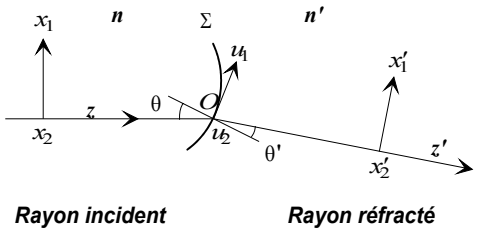


Figure 1.9 : Réfraction d'un crayon.

■ Matrice de réflexion

La matrice système dans le cas de la réflexion est un cas particulier de réfraction. La matrice s'obtient en remplaçant Θ' par (π-Θ) et n' par n dans l'***Equation 1-2***.

1.1.3.2 Erreur d'approximation

[SHIN 87] propose des fonctions pour évaluer les approximations faites dans la théorie présentée ci-dessus. Pour cela, il définit un *facteur de tolérance* τ. Ce facteur est fonction de la taille des pixels, de la dimension de la grille d'échantillonnage et du système optique traversé. D'autre part, il approche, à l'aide d'une série de puissance, le trajet effectué par un rayon paraxial traversant un système optique. Il obtient ainsi les fonctions d'erreur e_x, e_ξ liées respectivement au vecteur position x et au vecteur direction ξ du rayon paraxial.

La condition pour que l'image soit d'une qualité suffisamment bonne est alors un système d'inéquations mettant en jeu les fonctions e_x et e_ξ et le facteur de tolérance τ. Dans l'absolu, cette approche est parfaite, malheureusement, la notion d'*"image suffisamment bonne"* est très subjective et on ne peut donc pas quantifier cette notion. Néanmoins, des applications intéressantes de cette approche peuvent être trouvées.

1.1.3.3 Applications

L'étude faite par [SHIN 87] des déviations subies par un crayon est suffisamment générale pour s'appliquer à plusieurs algorithmes de lancer de rayons ou de lancer de faisceaux. Ainsi, [SHIN 87] réalise, entre autres, un lancer de rayons fondé sur les matrices systèmes et propose une généralisation de l'algorithme de lancer de faisceaux sur des objets polygonaux de [HECK 84] (voir ***Paragraphe 1.2.2***). Nous proposons uniquement une présentation théorique de ces deux méthodes, en effet, aucune implémentation n'a été effectuée.

1.1.3.3.1 Lancer de rayons utilisant les *matrices systèmes*

Dans cette méthode, les calculs de réflexion et de réfraction habituellement utilisés dans un lancer de rayons sont remplacés par des opérations sur les matrices et les vecteurs. L'auteur annonce un gain de temps de calculs sensible. L'algorithme utilisé est le suivant :

1. *Diviser l'écran en régions de n×m pixels (de l'ordre 5×5).*
2. *Lancer un rayon axial au centre de chaque région avec un lancer de rayons conventionnel. Calculer la matrice système.*
3. *Vérifier les conditions de continuité (voir plus bas). S'il existe une arête dans cette région, utiliser un lancer de rayons conventionnel pour tous les pixels de la région.*
4. *S'il n'y a pas d'arête, calculer le facteur de tolérance et l'angle d'ouverture maximum possible du crayon pour cette région ([SHIN 87] propose une équation pour calculer cet angle maximum). En se fondant sur cet angle :*
   a. *Si l'angle d'ouverture maximum possible pour la région permet de couvrir toute la région, lancer tous les rayons paraxiaux dans la région en utilisant la matrice système.*
   b. *Si l'angle d'ouverture maximum ne permet pas de couvrir au moins un pixel, utiliser un lancer de rayons conventionnel pour tous les rayons paraxiaux de la région.*
   c. *Sinon, diviser la région en sous-régions de sorte que l'angle d'ouverture maximum calculé puisse couvrir ces sous-régions individuellement. Répéter 2. à 4. pour chaque sous-région.*

Algorithme 1-1 : Lancer de rayons utilisant les matrices systèmes.

La condition de continuité proposée par l'auteur est très simple, elle se base sur la ou les surfaces traversées par le rayon axial de chaque région. Deux cas interdisant l'utilisation du lancer de crayons sont possibles :

1. Plusieurs surfaces sont traversées par des régions adjacentes : ces régions contiennent forcément une arête et la condition de continuité C1 n'est donc pas vérifiée ;
2. Des régions adjacentes traversent une même surface mais celle-ci contient une arête "brisant" la continuité C1.

Dans les autres cas, la condition de continuité est respectée et l'on peut utiliser la matrice système pour tous les rayons paraxiaux d'une même région.

Notons qu'avec une telle condition, les petits objets peuvent facilement passer à travers les rayons. Ces cas se produisent plus fréquemment qu'avec un lancer de rayons classique où l'échantillonnage de l'écran est d'au plus un pixel alors qu'ici il correspond à des régions de 5×5 pixels.

1.1.3.3.2 Modification du lancer de faisceaux sur des objets polygonaux

Le lancer de faisceaux sur des objets polygonaux développé dans [HECK 84] (voir ***Paragraphe 1.2.2***) est correct pour la réflexion. Par contre, pour la réfraction, il suppose que les rayons incidents sont perpendiculaires à la surface ou que tous les rayons sont parallèles. Cette approximation entraîne des effets de réfraction peu réalistes. [SHIN 87] propose d'utiliser ses matrices systèmes pour améliorer les résultats mais peu d'informations sont données à ce sujet (aucune implémentation n'a été réalisée par l'auteur).

On peut supposer que le lancer de crayons intervient dans la deuxième étape proposée dans [HECK 84], c'est-à-dire dans le calcul du rendu fondé sur la représentation arborescente de la scène. A ce niveau, l'ensemble des surfaces visibles est connu ainsi que le système optique pour y aboutir (il correspond aux sous-systèmes contenus dans la branche considérée). On peut donc lancer un crayon de section polygonale vers chaque surface à travers le système optique correspondant pour calculer la couleur de la surface. Si l'erreur d'approximation pour chacun de ces crayons est trop grande, les crayons sont subdivisés afin de vérifier les conditions de tolérance.

Notons que la condition de continuité est toujours vérifiée sur les surfaces visibles. De plus, le problème des petits objets est résolu au moment de la construction de l'arbre représentant les surfaces visibles de la scène indépendamment de l'utilisation des crayons. Par contre, le problème des marches d'escalier n'est pas pris en compte.

1.1.3.4 Discussion

L'étude d'un faisceau générique permet de considérer les problèmes communs aux lancers de faisceaux sans tenir compte de la forme de ceux-ci. Ainsi, le problème de la réfraction dans le lancer de faisceaux sur des objets polygonaux [HECK 84] peut être atténué par une meilleure approximation fondée sur les matrices systèmes.

Néanmoins, le lancer de crayons est uniquement possible sur des régions continues, en particulier sur des régions ne contenant pas d'arêtes. Il est donc nécessaire de faire appel à d'autres techniques le long de ces arêtes. De plus, le problème des petits objets et des petites ombres n'est pas pris directement en compte.

1.2 RAYONS VOLUMIQUES

1.2.1 Lancer de rayons avec des cônes

Une des premières approches volumiques d'un lancer de rayons est proposée dans [AMAN 84] en remplaçant les rayons conventionnel par les rayons volumiques en forme de cône. Le but annoncé de cet algorithme est de résoudre les problèmes d'aliassage, de produire des effets de pénombre et d'améliorer les temps de calculs. L'idée est de lancer vers la scène un cône à travers chaque pixel de l'écran de sorte à englober ce pixel. On peut ainsi, en théorie, calculer la proportion des différentes surfaces visibles à travers chaque pixel et effectuer un traitement de l'aliassage efficace. Nous verrons dans ce paragraphe que les différents buts ne sont pas atteints et que de nombreux problèmes subsistent.

1.2.1.1 Faisceaux coniques primaires

Pour chaque pixel, le rayon primaire d'épaisseur infinitésimale conventionnel est remplacé par un rayon volumique en forme de cône appelé *faisceau conique primaire*. Le sommet de ce faisceau conique est l'oeil. Son *angle au sommet*, c'est-à-dire l'angle formé par l'axe de symétrie du cône et son bord, est tel que le rayon *R* du cône au niveau de l'écran soit au moins égal à la diagonale *D* d'un pixel (***Figure 1.10***). Le faisceau englobe donc la totalité du pixel.

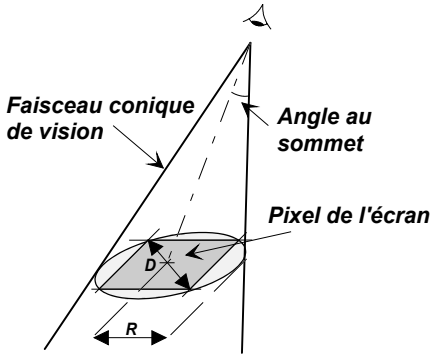


Figure 1.10 : Dimension d'un faisceau conique primaire.

Pour chaque faisceau conique primaire lancé, on étudie sa position par rapport à l'ensemble des objets. On en déduit l'existence d'une intersection faisceau conique-objet et la proportion de surface du cône retenue par l'objet. Les détails de ces calculs sont donnés dans le ***Paragraphe 1.2.1.2***.

Une liste des objets rencontrés triée en profondeur est associée à chaque faisceau. Ces listes permettent l'antialiassage des pixels. Si l'objet le plus proche (le premier objet de la liste) ne couvre pas la totalité du faisceau, l'objet suivant dans la liste participe à la couleur du point. Puisqu'on ne considère que la proportion de surface retenue, le calcul de la couleur est exact pour des objets qui ne se superposent pas. Pour tenir compte de ces derniers (***Figure 1.11***), des informations supplémentaires peuvent être stockées dans les listes triées des objets rencontrés. [AMAN 84] ne donne pas davantage de précisions sur le calcul de la couleur d'un pixel en particulier, la nature des informations supplémentaires. [KIRK 87], qui a aussi travaillé sur cet algorithme, propose une méthode pour calculer la couleur des pixels. Elle est simple mais très approximative.

Une première étape de la méthode de [KIRK 87] consiste à élaguer la liste triée des objets rencontrés. Cette liste est parcourue en commençant par l'objet le plus proche de l'oeil jusqu'à trouver un objet opaque couvrant entièrement le faisceau conique. Si cet objet

existe, tous les objets qui le suivent dans la liste sont supprimés (ils ne peuvent être directement visibles de l'oeil). Sinon, si aucun objet ne couvre entièrement le faisceau, la liste reste inchangée et l'on passe à l'étape suivante. Notons que l'on ne tient pas compte qu'un ensemble d'objets peut couvrir entièrement le faisceau (***Figure 1.11***).

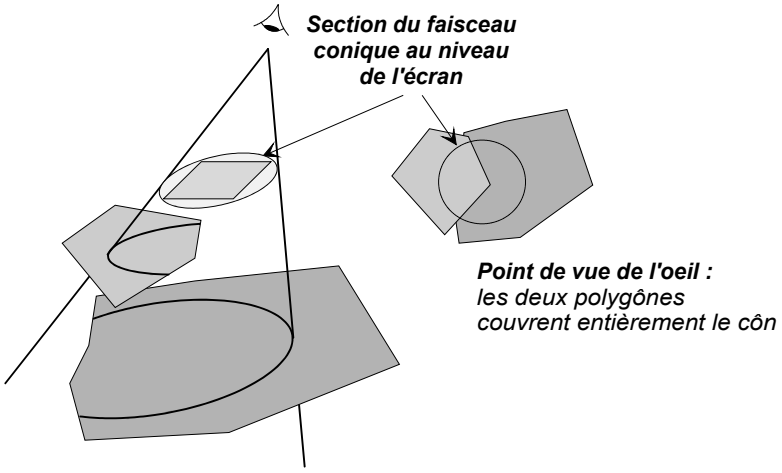


Figure 1.11 : On ne tient pas compte des ensembles d'objets (ici deux polygones) couvrant entièrement le faisceau conique.

La deuxième étape consiste à calculer la couleur d'un pixel. La liste construite précédemment est alors utilisée dans l'algorithme suivant :

---

si *la liste ne contient pas d'objet*
alors *la couleur du pixel est celle du fond initial*
sinon si *la liste contient exactement un objet*
alors

> ***si*** *celui-ci couvre entièrement le faisceau conique*
> ***alors*** *la couleur du pixel est celle de l'objet*
> ***sinon s'il*** *ne couvre qu'une proportion* p *du faisceau conique*
> ***alors*** *la couleur* I *est calculée de la façon suivante :*
>
> ---
>
> I = couleur de l'objet $*$ p + couleur du fond $*$ $(1-p)$;

---

sinon si *la liste contient* n *objets,* $n \geq 2$,
alors *la couleur* I *est calculée grâce à la récursion suivante :*

I_0 = couleur du $n^{ème}$ objet $*$ p_n + couleur du fond $*$ $(1-p_n)$

I_i = couleur du $(n-i)^{ème}$ objet $*$ p_{n-i} + I_{i-1} $*$ $(1-p_{n-i})$ pour $i \in [1; n[$

$I = I_{n-1}$

où le $n^{ème}$ objet est l'objet le plus éloigné et p_i est la proportion de surface retenue par le $i^{ème}$ objet.

Algorithme 1-2 : Calcul de la couleur d'un pixel.

Cette méthode fournit des résultats exacts pour des faisceaux ne traversant aucun, ou seulement un objet. Par contre, les résultats dans les autres cas sont faux puisque le calcul de la couleur ne tient absolument pas compte des recouvrements possibles entre les objets. Ainsi, dans l'exemple de la ***Figure 1.11***, la couleur du pixel dépend de la couleur du fond initial (départ de la récursion) alors que celui-ci n'est pas visible.

1.2.1.2 Calcul des intersections

Le calcul des intersections entre un rayon volumique et un objet peut être assez complexe. Mais il permet de connaître non seulement s'il y a une intersection mais aussi la

proportion du cône retenue par l'objet. Trois types d'objets sont considérés : les sphères, les plans et les polygones. Pour chacun d'eux, le calcul de l'intersection est divisé en deux étapes.

- tester rapidement s'il y a intersection entre le rayon conique et l'objet ;
- calculer une approximation de la proportion de la section du cône retenue par l'objet.

Nous allons étudier chacune de ces étapes pour les trois types d'objets.

1.2.1.2.1 Intersection faisceau conique-sphère

◆ Test rapide d'intersection

Soient *C* et *R*, respectivement le centre et le rayon de la sphère, *P* la projection orthogonale de *C* sur l'axe de symétrie du cône, *T* la distance de *P* à l'origine du cône, *A* l'angle au sommet du cône et *D* la distance de *P* à *C*. Ce test rapide est fondé sur la distance minimale entre le centre de la sphère et l'axe de symétrie du cône pour qu'il n'y ait pas d'intersection faisceau conique-sphère. Cette distance minimale *S* est obtenue pour une sphère tangente au cône (***Figure 1.12***) et s'exprime de la manière suivante :

Équation 1-3 :
$$S = T.\tan(A) + \frac{R}{\cos(A)}$$

L'évaluation de *S* nous permet alors de conclure de la manière suivante :

- si la valeur de *S* est inférieure à celle de *D*, alors il ne peut y avoir d'intersection entre le rayon et la sphère ;
- si la valeur de *T* est négative, c'est-à-dire si *P* se trouve derrière l'origine du cône, l'équation précédente (***Équation 1-3***) peut, dans certains cas, être substituée par des tests simples :
  - si la valeur absolue de *T* est supérieure à *R*, alors il ne peut y avoir intersection (***Figure 1.13.a***) ;
  - si la valeur absolue de *T* est inférieure à *R*, il faut tester si l'origine du cône appartient à la sphère, si c'est le cas, il coupe toute la sphère (***Figure 1.13.b***). Sinon, l'***Équation 1-3*** peut être utilisée pour conclure. Le signe de *T* ne fausse pas cette conclusion (***Figure 1.13.c*** et ***Figure 1.14***).

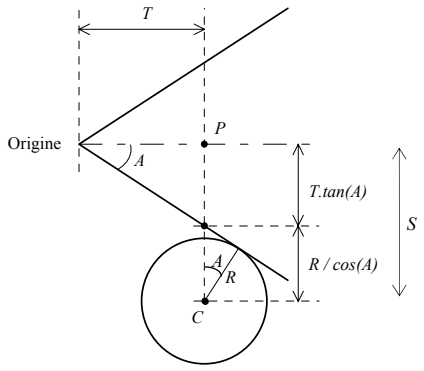
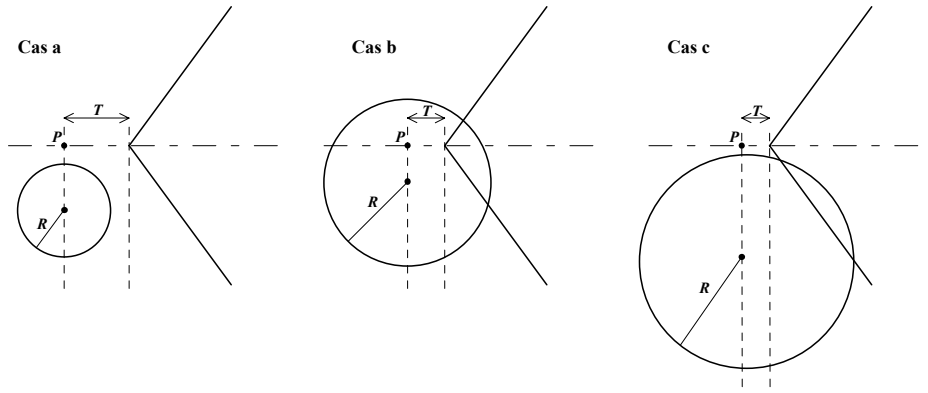


Figure 1.12 : Intersection cône-sphère.



Intersection impossible *Intersection systématique* *On se fonde sur l'Equation 1-3*

Figure 1.13 : Cas particulier où T est négatif.

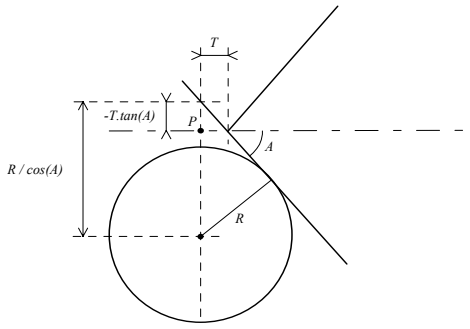


Figure 1.14 : Calcul de S pour un T négatif.

◆ ***Evaluation de la proportion de surface rencontrée***

L'évaluation de la proportion de surface du cône retenue par une sphère revient à trouver la surface de l'intersection de deux cercles, le contour de la sphère et le contour du cône le plus proche de la sphère. D'après [AMAN 84], un calcul rapide est obtenu à partir d'une évaluation polynomiale simple mais il ne donne pas de détails. [KIRK 87] propose une approximation de cette surface, fondée sur le découpage régulier de la région rencontrée. Cette région est découpée en bandes de largeur identique (***Figure 1.15***). La surface de chacune de ces bandes est approchée. La somme de ces surfaces fournit alors une approximation de la région rencontrée. Pour éviter de faire plusieurs fois ces calculs, [KIRK 87] construit, en prétraitement, une table permettant d'obtenir directement la surface en fonction du rayon de la sphère, du rayon du cône au niveau de l'intersection et de la distance du centre de la sphère à l'axe de symétrie du cône.

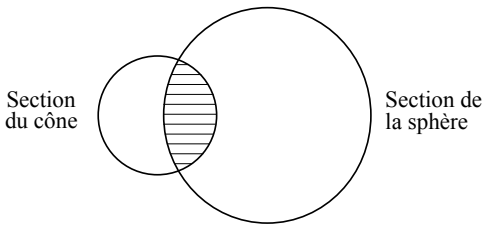


Figure 1.15 : La région rencontrée est divisée en bandes régulières.

1.2.1.2.2 Intersection faisceau conique-plan

◆ ***Test rapide d'intersection***

Le test rapide d'intersection entre un plan et un cône est fonction de la position du point *I*, intersection entre l'axe du cône et le plan, par rapport à l'origine du cône. Notons que les procédures de décision fournies par [AMAN 84] et par [KIRK 87] ne sont pas tout à fait exacte. En effet les deux approches considèrent que si le point *I* est derrière l'oeil, le plan ne peut être visible, or, comme le montre la ***Figure 1.16.c*** ceci n'est pas toujours le cas. Voici donc une méthode corrigée.

Si le point *I* est situé devant l'origine, il y a obligatoirement intersection (***Figure 1.16.a***). Sinon, l'angle α entre la normale *n* au plan et l'axe du cône est calculé. Une comparaison des angles $\Pi/2 - \alpha$ et *A*, permet alors de conclure s'il y a intersection ou non (***Figure 1.16.b*** et ***Figure 1.16.c***).

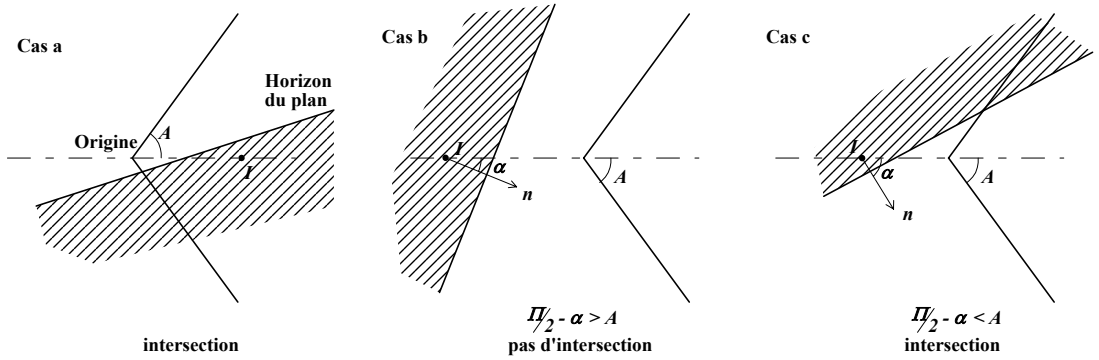


Figure 1.16 : Intersection cône-plan.

◆ ***Evaluation de la proportion de surface rencontrée***

Deux types d'intersection faisceau conique-plan sont possibles. Soit le plan coupe entièrement le cône, soit la coupe est partielle. Dans le premier cas, la proportion de surface rencontrée est de 100%. Dans le deuxième cas, un observateur placé au sommet du cône voit un demi-plan (***Figure 1.17***). Le bord de ce demi-plan est appelé *horizon*. On peut donc se ramener au cas bidimensionel d'intersection entre un cercle (la section du cône) et un demi-plan délimité par l'horizon. A l'aide des angles α et *A* et du rayon de la section du cône, nous pouvons calculer la distance *d* entre le centre du cercle et le bord du demi-plan. Cette distance *d* permet alors le calcul de la proportion de surface retenue par le demi-plan, à l'aide d'une approximation polynomiale. Là encore, [AMAN 84] ne fournit pas le polynôme. [KIRK 87] utilise la même méthode d'approximation de la surface que dans le cas d'une intersection avec une sphère en identifiant le demi-plan à une sphère de rayon infini. La table proposée contient 256 valeurs, [KIRK 87] précise que pour une image de 512x512 pixels, l'augmentation du nombre d'entrées dans la table n'est pas justifiée car la différence de rendu est minime. L'occupation mémoire de cette table n'est donc pas un inconvénient.

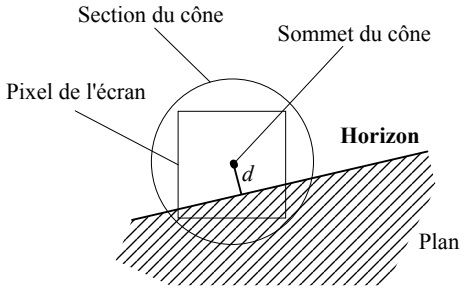


Figure 1.17 : Point de vue d'un observateur situé au sommet du cône.

1.2.1.2.3 Intersection faisceau conique-polygone

◆ Test rapide d'intersection

Une première solution consiste à calculer l'intersection entre le cône et le plan du polygone, puis à tester si la coupe obtenue traverse le polygone. Cette solution est très coûteuse car la coupe peut avoir des formes très diverses telles que celle d'un cercle, d'une ellipse, d'une parabole ou d'une hyperbole.

La deuxième solution projette le polygone dans un plan perpendiculaire à l'axe du cône afin de se ramener à l'intersection d'un cercle et d'un polygone. Ceci peut être réalisé en calculant la distance entre le centre du cercle et chaque arête du polygone.

◆ Evaluation de la proportion de surface rencontrée

Aucune information concernant le calcul de la proportion de surface du cône rencontrée par un polygone n'est donnée par [AMAN 84].

1.2.1.3 Faisceaux coniques d'ombre

Le calcul des ombres portées est très brièvement abordé par l'auteur ; seules quelques idées pour introduire des ombres douces sont présentées. En particulier, il ne précise ni comment déterminer la direction exacte des cônes d'ombre ni comment calculer la taille de leur base. Nous avons donc imaginé une solution pour les ombres de polygones, dues à des sources lumineuses ponctuelles. Puis nous considérons les effets de pénombre dus à des sources lumineuses volumiques sphériques et proposés par [AMAN 84].

1.2.1.3.1 Effet d'ombre

Nous considérons les hypothèses suivantes :

- la scène ne comporte que des objets polygonaux ;
- un faisceau conique primaire est lancé sur chaque pixel ;
- la liste des polygones rencontrés par chacun de ces faisceaux primaires est associée à ces faisceaux ;
- pour chaque polygone de ces listes, la proportion de surface du cône qu'il retient est connue.

Pour savoir si la région d'un polygone correspondant à un pixel est à l'ombre, nous lançons un *faisceau conique d'ombre* à partir de chaque source lumineuse ponctuelle vers chaque polygone de la liste du faisceau primaire. La direction du faisceau d'ombre est telle que son axe de symétrie passe par le point d'intersection de l'axe de symétrie du cône primaire et du plan du polygone (***Figure 1.18***). Son angle au sommet doit permettre de couvrir entièrement la région du polygone rencontrée par le faisceau primaire. (***Figure 1.18***).

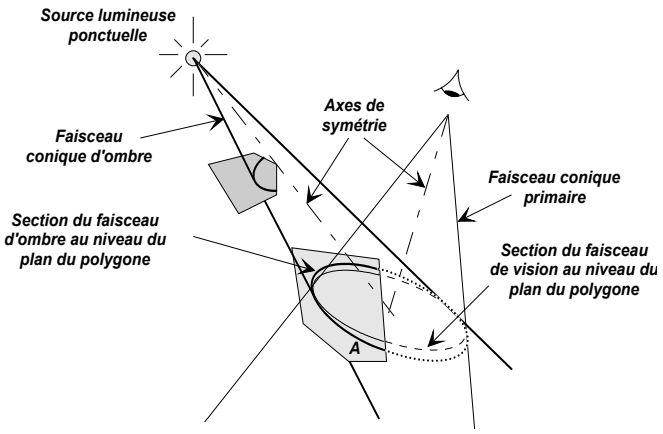


Figure 1.18 Faisceau conique d'ombre.

Pour chaque faisceau d'ombre, la somme des proportions des régions retenues par les polygones de la liste associée est calculée. Cette somme permet de pondérer la contribution de l'ombre dans la valeur finale de la couleur de la région rencontrée. La couleur finale du pixel peut alors être calculée comme dans le ***Paragraphe 1.2.1.1***.

1.2.1.3.2 Effet de pénombre

[AMAN 84] propose une solution simple pour les effets de pénombre dus à des sources lumineuses volumiques sphériques (des explications sur les causes des effets de pénombre sont proposé au ***Paragraphe 3.2.1***).

Les faisceaux d'ombre pour des sources lumineuses sphériques sont obtenus en déplaçant l'origine du cône de façon à entièrement la source lumineuse sphérique et à avoir une section au niveau du polygone rencontré identique à celle des faisceaux d'ombre pour les sources ponctuelles (***Figure 1.18***). Le calcul de la proportion de surface retenue par les différents objets du nouveau faisceau d'ombre est le même que pour les sources ponctuelles. Ces proportions permettent de pondérer les intensités finales pour obtenir les effets de pénombre.

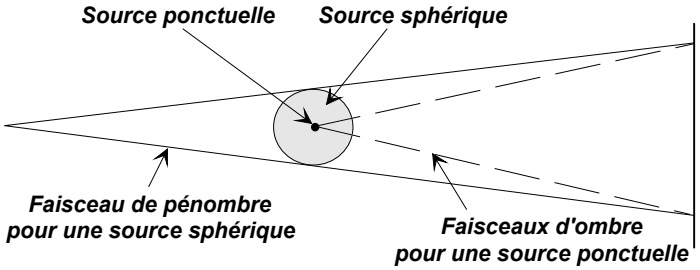


Figure 1.19 : Faisceau conique de pénombre.

Cette solution n'est pas exacte mais l'effet visuel est réaliste. Notons que l'exemple proposé dans [AMAN 84] est une scène composée de six sphères sans ombre ni effets de réflexion ou de réfraction ce qui laisse un doute quant au bon fonctionnement de l'algorithme.

1.2.1.4 Réflexion et réfraction

Les effets de réflexion et de réfraction sont complexes dans le cas du lancer de cônes. L'auteur précise uniquement l'utilisation de la courbure de la surface rencontrée par le faisceau primaire pour construire et lancer des faisceaux coniques réfléchis et réfractés. Aucune précision supplémentaire n'est donnée concernant le calcul du sommet des faisceaux ni leurs angles au sommet. En particulier, on ne sait pas toujours quelle doit être

la section du faisceau conique réfléchi ou réfracté. En effet, supposons un polygone réfléchissant qui coupe partiellement le faisceau conique primaire (***Figure 1.20***), on peut concevoir que l'on construit le cône réfléchissant à l'aide d'un oeil virtuel symétrique de l'oeil réel par rapport au polygone (***Figure 1.20***). Dans ce cas, la section du faisceau primaire et celle du faisceau réfléchi coïncident mais le faisceau réfléchi est trop grand et puisqu'on n'effectue aucun découpage faisceau-objet, on risque de faire intervenir des objets non visibles pour cette réflexion. Un autre cas de figure qui pose problème est le cas d'une sphère réfléchissante ou réfractante qui coupe partiellement le faisceau primaire (***Figure 1.21*** et ***Figure 1.22***). Dans ces cas, il est difficile d'imaginer la forme du faisceau conique réfléchi ou réfracté.

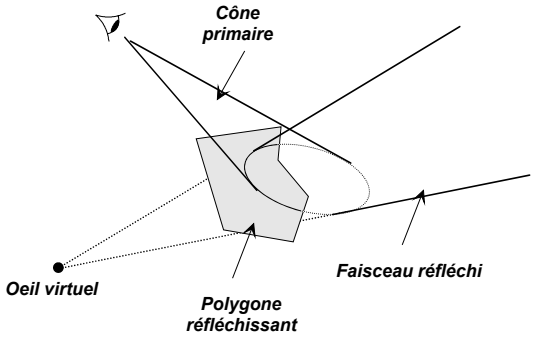


Figure 1.20 : Réflexion du faisceau primaire sur une partie d'un polygone.

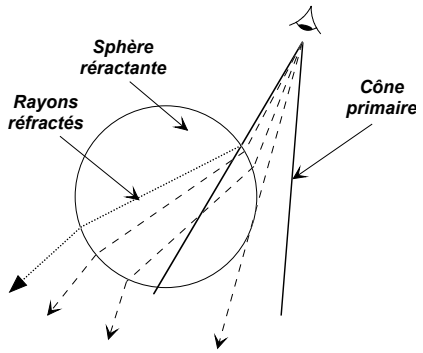


Figure 1.21 : Réfraction du faisceau primaire sur une partie d'une sphère.

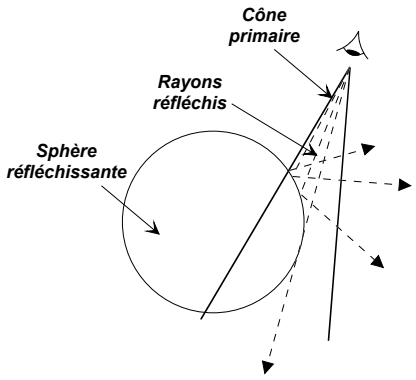


Figure 1.22 : Réflexion du faisceau primaire sur une partie d'une sphère.

Les effets de réflexion et de réfraction semblent donc pour le moins limités à quelques cas particuliers où les bords des objets réfléchissants ou réfractants ne sont pas à l'intérieur des cônes.

1.2.1.5 Autres effets

[KIRK 87] propose deux effets spéciaux fondés sur le lancer de cônes. Le premier est semblable à un algorithme de perturbation de la normale ("bump mapping") [BLIN 78]. Pour se faire, [KIRK 87] fait varier l'angle au sommet des cônes réfléchis voisins suivant une loi sinusoïdale. Le deuxième permet de produire des nuages. Il s'agit en fait de faire varier le coefficient de transparence sur la surface d'un polygone à l'aide d'une loi imitant la variation de densité d'un nuage. Des cônes de réfraction sont lancés à travers de tels polygones pour simuler un ensemble de nuages.

1.2.1.6 Discussion

Remplacer les rayons conventionnels d'épaisseur infinitésimale par des rayons volumiques en forme de cône permet de tenir naturellement compte de la cohérence de la

scène. Malheureusement, cette forme en cône entraîne de nombreux problèmes. Si connaître la position d'un objet par rapport à un cône est relativement aisé, calculer la proportion de surface retenue par l'objet est beaucoup plus complexe, même pour des objets simples comme le plan, le polygone ou la sphère. Ceci est principalement dû à la nécessité de découper les objets par rapport aux cônes. Le calcul de l'intensité lumineuse telle que [KIRK 87] le présente est très approximatif et semble pouvoir donner de bons résultats uniquement pour des scènes très (trop) simples. L'explication de la construction des faisceaux coniques réfléchis et réfractés n'a été trouvée dans aucun article. En particulier, on ne sait pas toujours quelle section doit avoir le faisceau au niveau de l'objet réfléchi ou réfracté. De manière générale, aucune équation caractérisant la variation de l'angle au sommet d'un cône traversant un système optique n'est proposée.

Le fait de chercher à calculer les proportions retenues à un faisceau lancé sur la totalité d'un pixel devrait permettre un bon traitement de l'aliassage. En particulier, les problèmes de marches d'escalier, de petits objets et de petites ombres devraient être résolus. Malheureusement, les informations fournies [AMAN 84] [KIRK 87] sont trop succinctes pour imaginer une solution qui donnerait de bons résultats.

L'utilisation d'une forme conique permet d'introduire naturellement des sources lumineuses sphériques. Les effets de pénombre sont alors possibles. Les effets de perturbation de la normale ("bump mapping") et de nuages semblent être possibles, mais n'impliquent pas forcément l'utilisation d'un lancer de cônes.

L'étude de cet algorithme permet donc de constater que l'utilisation d'un faisceau volumique en forme de cône est plutôt mal adaptée aux scènes complexes contenant des objets réfléchissants et réfractants. D'autres formes de faisceaux, en particulier des formes pyramidales, ont été utilisées. Le paragraphe suivant présente l'un de ces algorithmes.

1.2.2 Lancer de faisceaux sur des objets polygonaux

1.2.2.1 Introduction

[HECK 84] propose un *lancer de faisceaux sur des objets polygonaux*. Le but annoncé de cet algorithme est de diminuer les temps de calculs. Comme nous le verrons, des réserves à ce sujet peuvent être émises. Néanmoins, pour atteindre cet objectif, [HECK 84] se limite aux scènes polygonales pour lesquelles les algorithmes de découpage ("clipping") sont bien connus. L'algorithme développé comporte deux parties distinctes :

1. la recherche des surfaces directement visibles, par réflexion ou par réfraction ;
2. le calcul de la couleur des pixels de l'image.

La première étape est réalisée à l'aide d'un algorithme récursif de lancer de faisceaux. La deuxième utilise un algorithme séquentiel de remplissage de polygones de type balayage par lignes.

Dans son algorithme, [HECK 84] construit une structure arborescente qui décrit le parcours, dans la scène, d'un faisceau pyramidal issu de l'oeil et traversant exactement l'écran (***Figure 1.23***). Chaque objet qui se trouve sur le chemin de ce faisceau en retient une partie. Le faisceau est alors découpé selon la forme de l'objet (***Figure 1.23***) et la surface du polygone à l'intérieur de la pyramide est stockée dans l'arbre. Si cet objet est réfléchissant ou réfractant, la partie de la surface rencontrée sert de base à une pyramide réfléchie ou réfractée. Sa direction est obtenue à l'aide des lois de l'optique (ou d'approximations dans le cas de la réfraction).

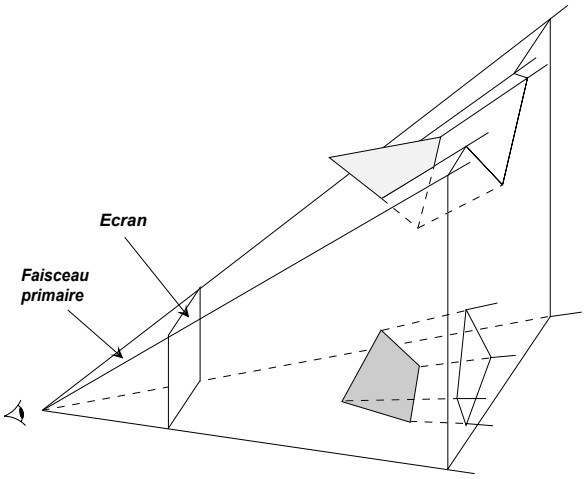


Figure 1.23 : Propagation du faisceau primaire dans la scène.
(La pyramide possède une base uniquement pour une meilleure lisibilité de la figure)

Pour calculer la couleurs des pixels, on parcourt l'arbre décrivant la propagation du faisceau dans la scène. Au niveau de chaque noeud, on considère les polygones qui lui sont associés et on leur applique un algorithme de remplissage. On peut ainsi, avec une récursion terminale, calculer la couleurs de tous les pixels de l'écran.

1.2.2.2 Construction de l'arbre des polygones rencontrés

Une structure arborescente décrivant les intersections faisceaux-objets est construite au fur et à mesure que le faisceau primaire se propage dans la scène. Cet arbre est semblable à celui du lancer de rayons proposé par [WHIT 80]. Les branches correspondent aux faisceaux lancés et les noeuds aux surfaces des objets rencontrés. Notons qu'à la différence du lancer de rayons, il peut y avoir plusieurs objets rencontrés par un même faisceau et par conséquent, un noeud peut contenir plusieurs morceaux de polygones (***Figure 1.24***). La racine de l'arbre fournit la liste des surfaces polygonales visibles par le faisceau primaire. Pour chacune de ces surfaces, si elle est réfléchissante et/ou réfractante, on lance respectivement un faisceau réfléchi et/ou un faisceau réfracté à partir de cette surface (voir ***Paragraphes 1.2.2.3*** et ***1.2.2.4***). Si des polygones appartiennent à ces nouveaux faisceaux, on crée de nouveaux noeuds contenant les surfaces de ces polygones à l'intérieur des faisceaux. Ces lancers sont effectués tant qu'une profondeur maximale n'est pas atteinte (en général 5 niveaux de récursion), que la taille des polygones n'est pas trop petite (de l'ordre du pixel) et que la contribution à l'intensité du pixel de la branche courante n'est pas trop faible.

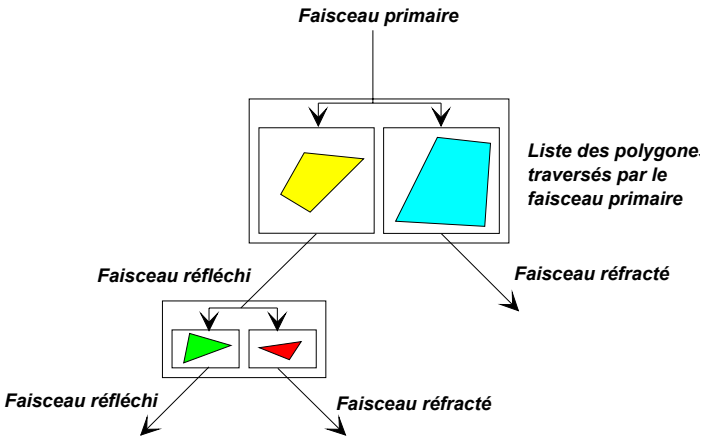


Figure 1.24 : Arbre issu du faisceau primaire.

L'algorithme de construction de l'arbre des polygones rencontrés est le suivant :

```
PolygonList BeamTrace(Beam beam, Matrix ctm)
    Transformer la scène dans le repère du faisceau en utilisant la matrice de
      transformation courante 'ctm'
    Trier par ordre de priorité l'ensemble des polygones de la scène contenus dans la
      liste 'scenePolygonList'
    pour chaque polygone poly de scenePolygonList
    faire
        B = intersection(beam, poly)
        si B≠NULL
        alors
            si RecurseFurther(depth, B, ...)
            alors
                si reflective(poly)
                alors newCtm = ctm * ReflectionMatrix(poly)
                    B.reflectiveTree = BeamTrace(B, newCtm)
                si refractive(poly)
                alors newCtm = ctm * RefractionMatrix(poly)
                    B.refractiveTree = BeamTrace(B, newCtm)
            Ajouter B à la liste des intersections faisceaux-polygones 'fragmentList'
            beam = Difference(beam, poly)
    retourner fragmentList
```

Algorithme 1-3 : Construction de l'arbre des polygones.

Les intersections faisceaux-surfaces sont déterminées à l'aide de la liste triée en profondeur des polygones de la scène [NEWE 72]. Dans le tri proposé par [NEWE 72], les polygones intersectants et les dépendances cycliques ne sont pas possibles, mais ceci n'est pas une limitation théorique de l'algorithme. Cette liste est mise à jour pour chaque nouveau faisceau. Pour chaque polygone *P* de la liste, si le faisceau traverse *P*, il est directement visible et est ajouté à la liste *fragmentList*. Pour être sûr qu'aucun autre polygone situé exactement derrière *P* ne soit, par la suite, considéré comme visible, la forme du faisceau est calculée à nouveau en tenant compte de l'intersection avec *P*. Si le faisceau ne traverse pas *P*, *P* ne peut être visible et il est écarté.

Pour simuler la réflexion et la réfraction, la fonction "*BeamTrace()*" est appelée récursivement avec un faisceau dont la section correspond au polygone rencontré. Les transformations nécessaires sont présentées dans le paragraphe suivant.

1.2.2.3 Réflexion

Lorsque le faisceau heurte un polygone réfléchissant, il est découpé suivant la forme de ce polygone. Le faisceau réfléchi est alors lancé. Il a pour base la partie du polygone qui a retenu le faisceau (***Figure 1.25***). La réflexion sur un plan étant une transformation linéaire, on peut facilement calculer la forme d'un faisceau réfléchi. Pour cela, il suffit de considérer l'oeil virtuel symétrique de l'oeil par rapport au plan du polygone réfléchissant (***Figure 1.25***).

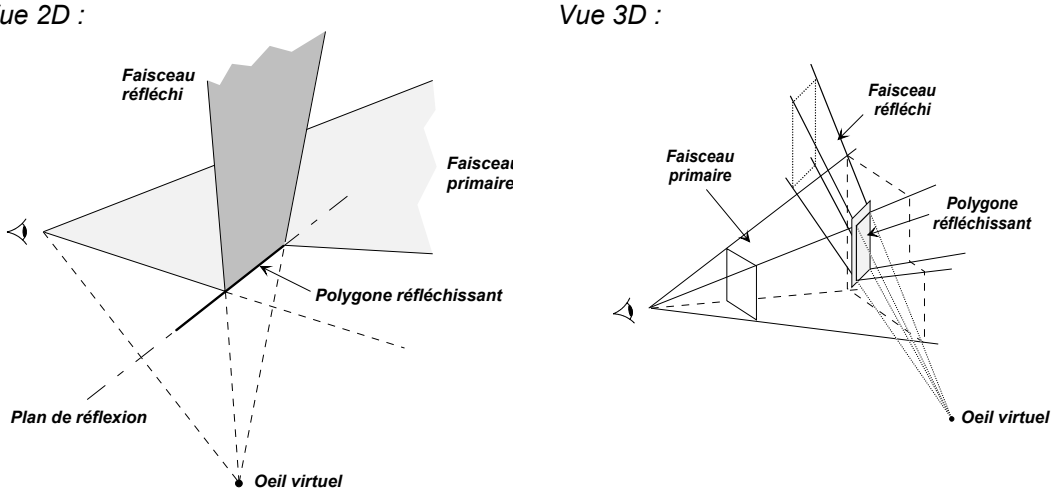


Figure 1.25 : Construction d'un faisceau de réflexion.

1.2.2.4 Réfraction

Contrairement à la réflexion, la réfraction d'un rayon sur une surface plane n'est généralement pas une transformation linéaire. La ***Figure 1.26*** et la ***Figure 1.27*** présentent les distorsions d'un damier infini, respectivement dans l'eau vu de l'extérieur et hors de l'eau vu de l'intérieur. La deuxième figure montre l'effet de l'"*angle critique*" pour des rayons passant d'un milieu plus dense vers un milieu moins dense, comme de l'eau (1.33[2]) vers l'air (1.0003[2]). Les rayons incidents d'angle critique sont réfractés parallèlement à la surface. Quand l'angle du rayon incident est supérieur à l'angle critique, il n'y a pas de réfraction, le rayon ne quitte pas le matériau (***Figure 1.28***).

[2] valeur pour des conditions normales de température et de pression.

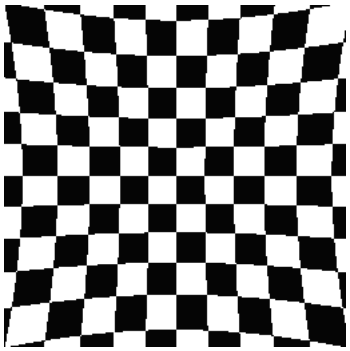


Figure 1.26 : Damier dans l'eau vu de l'extérieur.

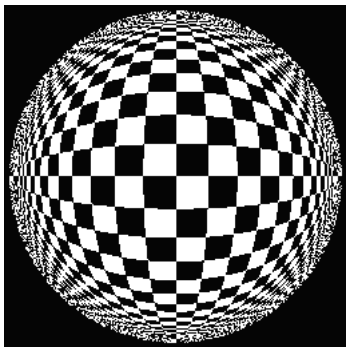


Figure 1.27 : Damier hors de l'eau vu de l'intérieur.

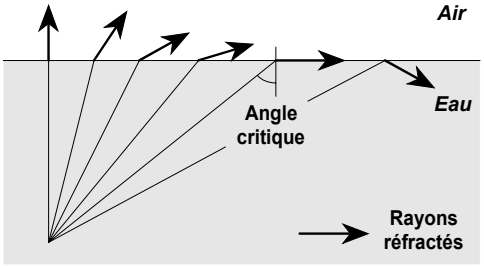


Figure 1.28 : Réfraction d'un rayon d'un milieu plus dense vers un milieu moins dense.

Puisque la réfraction dévie les lignes, elle ne peut être exprimée par une transformation linéaire. Il faut donc définir une approximation linéaire pour pouvoir construire un faisceau réfracté. Pour cela, [HECK 84] généralise l'utilisation de l'oeil virtuel et propose de placer le sommet du faisceau sur la perpendiculaire au plan réfractant passant par l'oeil (***Figure 1.29***). La position exacte est proposée en annexe de [HECK 84], elle dépend principalement des angles de réfractions des arêtes du faisceau.

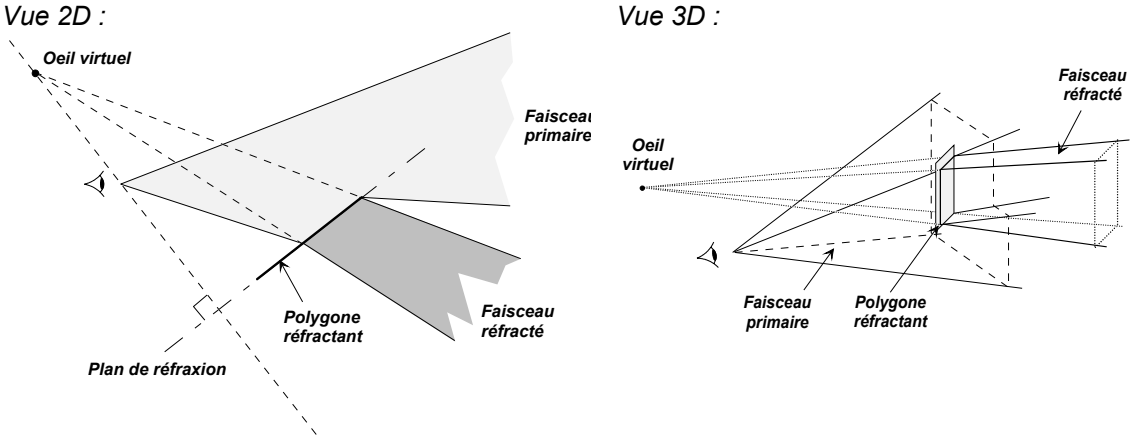


Figure 1.29 : Construction d'un faisceau réfracté.

Cette simplification supprime entre autres la notion d'angle critique et les effets obtenus ne sont plus optiquement corrects. Dans le cas de la ***Figure 1.26*** et de la ***Figure 1.27***, le damier ressemblera à un damier "normal", sans déformation. L'effet le plus visible apparaît lors du passage d'un milieu dense vers un milieu moins dense. Mais ceci est rare puisque l'observateur est généralement situé dans le milieu le moins dense (i.e. dans l'air et regardant dans l'eau).

1.2.2.5 Calcul du rendu

Chaque polygone directement visible de l'oeil est référencé dans la racine de l'arbre des polygones rencontrés. Chacun de ces polygones est associé au sous-arbre des fragments de surfaces projetés sur ce polygone par réflexion ou réfraction. L'intensité finale peut être calculée simultanément pour l'ensemble des pixels en appliquant l'équation récursive suivante :

$$I = c_d I_d + c_s I_s + c_r I_r + c_t I_t$$

où c_d, c_s, c_r et c_t sont respectivement les coefficients de diffusion, spéculaire, de réflexion et de transmission.

Pour utiliser le modèle de Gouraud [GOUR 71], il est nécessaire d'utiliser une mémoire tampon. Les polygones y sont successivement dessinés, dans l'ordre de parcours de l'arbre. Pour chaque surface de polygone, l'intensité diffuse I_d et l'intensité spéculaire I_s, sont calculées à l'aide du modèle d'éclairage de Phong [PHON 75]. Pour des valeurs constantes de c_d, c_s, c_r, c_t pour chaque polygone, le rendu peut être obtenu en additionnant les intensités dans la mémoire tampon pour chaque polygone séparément. L'addition étant commutative, le parcours de l'arbre peut s'effectuer dans n'importe quel ordre.

1.2.2.6 Calcul de l'ombrage

Dans un lancer de rayons classique, le calcul de l'ombre en un point nécessite de lancer un rayon en direction de chaque source lumineuse. Si ce rayon rencontre un objet avant d'atteindre la source alors il est à l'ombre de cet objet ; sinon il est éclairé. Puisque les rayons ne vont que vers les sources lumineuses et non dans toutes les directions, cette solution ne peut représenter des éclairages indirects.

Dans le cas du lancer de faisceaux présenté par [HECK 84], le calcul des ombres fait partie d'un prétraitement. Les auteurs proposent une extension récursive de l'algorithme de Atherton-Weiler [ATHE 78]. Elle consiste à lancer un faisceau, que nous appellerons *faisceau d'ombre*, à partir de chaque source lumineuse de façon à envelopper toute la scène. Si un faisceau par source est insuffisant et qui ne couvre pas l'ensemble de la scène, plusieurs faisceaux, ne se superposant pas, seront utilisés. Ces faisceaux peuvent être réfléchis et réfractés, simulant ainsi les éclairages indirects. Chaque faisceau est associé à la liste des polygones qui y sont entièrement visibles. On utilise pour cela l'algorithme de calcul des surfaces visibles de Weiler-Atherton [WEIL 77]. Ces polygones éclairés sont transformés dans le repère de la scène pour être plaqués sur leurs parents respectifs. Le calcul des surfaces visibles est alors effectué grâce, de nouveau, à l'algorithme de Weiler-Atherton. Le calcul des ombres est réalisé en utilisant les polygones "d'ombre". Les surfaces sur lesquelles sont plaqués les polygones "d'ombre" sont éclairées et les autres sont à l'ombre.

Outre des réflexions multiples de la lumière, cet algorithme offre d'autres avantages. En effet, il est possible de définir des sources lumineuses directionnelles et à section polygonale.

1.2.2.7 Antialiassage

L'antialiassage fait partie des extensions proposées. La solution consiste à construire un graphe d'adjacence des faces à partir de la structure arborescente obtenue (l'auteur ne précise pas l'algorithme de construction). Ce graphe permet alors de trouver les contours des objets, ombres, réflexions et réfractions. Ces contours correspondent aux pixels devant être antialiassés. Aucune précision n'est donnée quant à la manière de construire ce graphe. On peut supposer que les surfaces de polygones contenues dans l'arbre sont projetées dans

le plan de l'écran en suivant le chemin inverse parcouru par le faisceau dans la branche de la surface. Un algorithme de "découpage" de polygones permet alors de construire le graphe.

1.2.2.8 Discussion

La représentation de la scène sous la forme arborescente correspond à une description continue des contours de l'image[3]. En effet, c'est seulement au moment du calcul de la couleur des pixels qu'une définition de l'image est choisie. Cette représentation continue des contours de l'image permet alors un traitement efficace et simple des problèmes d'aliassage. De plus, un certain confort de travail est apporté lors de la création d'une image. En effet, la modification de l'intensité ou de la couleur des sources lumineuses ainsi que le changement de la couleur et des caractéristiques des matériaux associées aux polygones ne nécessitent pas de tout recalculer. Seule l'étape de rendu est nécessaire puisque l'observateur, les objets et les sources lumineuses n'ont pas bougé.

Le fait de lancer des faisceaux d'ombre vers la scène et de tenir compte, à ce niveau, des réflexions et réfractions permet de produire des éclairages indirects, très difficiles à obtenir avec un lancer de rayons classique. Des éclairages de type spot sont aussi possibles en limitant la section des faisceaux d'ombre.

Curieusement, aucun temps de calculs n'est donné alors que le but annoncé est un gain de temps. On peut supposer qu'il n'était pas optimal puisque [DADO 85] propose une version optimisée du même algorithme. Pour profiter au maximum de la cohérence de la scène, [DADO 85] utilise une structure très proche des subdivisions "BSP"[4] de [FUCH 80] et décompose l'algorithme en trois sous-problèmes : la détection des polygones visibles, le tri de ces polygones et leur découpage. Là encore, aucune comparaison en temps n'est proposée. On peut logiquement penser que les temps de calculs sont relativement importants. En effet, de nombreux découpages entre polygones quelconques (concaves, troués, ...) sont effectués, or un tel algorithme est très coûteux.

[3] On parle souvent d'une description des contours dans l'espace objet (voir ***Chapitre 4***).

[4] Dans les subdivisions spatiales binaires "BSP" (Binary SpatialPartisining Tree), l'espace est divisé récursivement par un seul plan en deux sous-espaces de dimension quelconque. Le plan est choisi judicieusement pour diviser les sous-espaces de manière équitable sans traverser aucun objet.

1.3 ALGORITHMES HYBRIDES

Dans le ***Paragraphe 1.1***, nous avons étudié des algorithmes regroupant les rayons d'épaisseur infinitésimale, dans le ***Paragraphe 1.2***, nous avons abordé les algorithmes fondés sur des rayons volumiques. Il est donc logique d'aborder dans ce paragraphe les algorithmes mettant en même temps à profit ces deux approches. Les deux solutions proposées ont de nombreux points en commun. Elles ont pourtant été élaborées de façon totalement indépendante. Notons aussi que la deuxième solution que nous proposons, [GHAZ 92], est à la base des recherches effectuées dans le cadre de cette thèse.

1.3.1 Choix dynamique entre les rayons et les faisceaux

1.3.1.1 Principe général

[MARK 88] et [MARK 90] proposent un algorithme hybride alliant un lancer de rayons et un lancer de faisceaux. Le but avoué est d'améliorer les temps de calculs du rendu d'une scène composée de polygones convexes. Pour cela, les auteurs utilisent une technique similaire à celle de [WARN 69] pour subdiviser récursivement et régulièrement l'écran. On lance un premier faisceau pyramidal issu de l'oeil à travers l'écran. La ***Figure 1.30*** illustre le cas le plus favorable. Les quatre arêtes rencontrent une même face F et aucun objet ne se trouve dans le faisceau et entre l'écran et le plan de F. La contribution de cette face à la couleur des pixels de l'écran peut alors être calculée pour l'ensemble de l'écran. Si la face F est réfléchissante, un faisceau réfléchi est lancé. Ce faisceau est porté par les quatre rayons réfléchis correspondant aux arêtes du faisceau primaire (***Figure 1.31***). La contribution des faces réfléchissantes à la couleur des pixels est calculée récursivement en lançant différents faisceaux réfléchis. Les cas des objets réfractants et les traitements des ombres sont abordés respectivement dans le ***Paragraphe 1.3.1.3*** et dans le ***Paragraphe 1.3.1.4***.

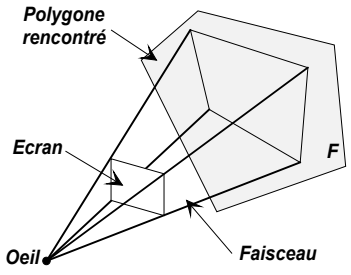


Figure 1.30 : Faisceau primaire traversant la totalité d'un polygone.

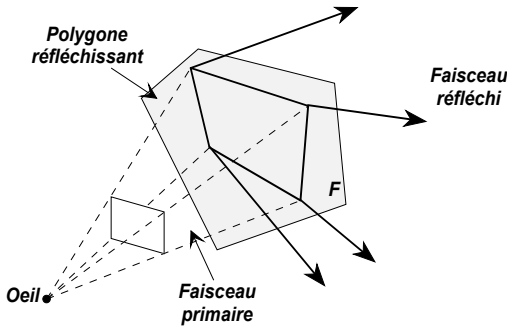


Figure 1.31 : Faisceau réfléchi.

Ce cas très favorable est bien sûr exceptionnel. La ***Figure 1.32*** montre le cas d'une intersection partielle avec un objet : toutes les arêtes du faisceau n'ont pas pour plus proche intersection le même objet. Le faisceau ne traverse donc aucun objet entièrement. Dans la ***Figure 1.33***, le faisceau est entièrement bloqué par un objet mais un autre objet s'est glissé dans le faisceau. Dans ces deux cas, il n'est donc pas possible de calculer globalement la couleur des pixels de l'écran.

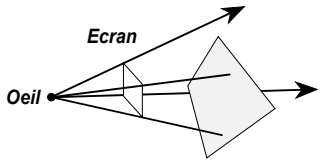


Figure 1.32 : Polygone bloquant une partie seulement du faisceau primaire.

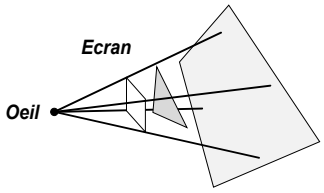


Figure 1.33 : Polygone "glissé" entre les arêtes du faisceau primaire.

[MARK 88] et [MARK 90] proposent deux solutions : soit on utilise un lancer de rayons conventionnel, soit on subdivise l'écran en quatre [WARN 69]. Seule la deuxième solution nous intéresse ici. Dans le cas de la subdivision de l'écran, on lance des rayons aux coins des sous-écrans obtenu pour construire quatre sous-faisceaux (***Figure 1.34***). Tant que l'on décide d'utiliser des faisceaux au lieu d'un lancer de rayons, ce mécanisme est appliqué sur l'ensemble du chemin parcouru par le faisceau (***Figure 1.34***).

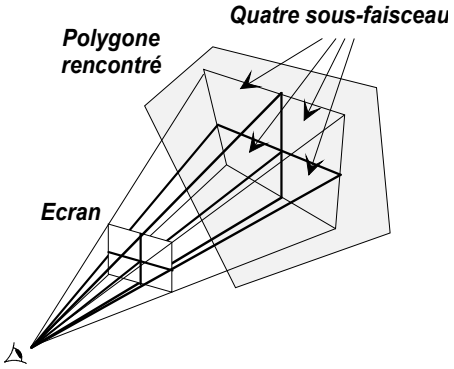


Figure 1.34 : Subdivision de l'écran et du faisceau primaire.

Le critère de choix C entre le lancer de rayons et les faisceaux est traduit par l'équation suivante :

Équation 1-4 :

$$C = F\frac{16 * F_w - D}{E_W^2 D}$$

où F est le nombre total de polygones dans la scène, D est la *profondeur* de la scène, F_W le nombre de polygones dans le faisceau et E_W la largeur de l'écran en pixels. La profondeur de la scène est une valeur (un peu vague) introduite dans [MARK 90] qui peut être interprétée comme le nombre de coordonnées différentes des objets suivant l'axe z. Si l'expression C est plus grande qu'un seuil fixé empiriquement, on conclut que le niveau de cohérence de cette partie de la scène est faible et on utilise alors un lancer de rayons conventionnel. Sinon, on subdivise l'écran et les faisceaux correspondants.

1.3.1.2 Position d'un objet par rapport à un faisceau

Etre capable de positionner un polygone par rapport à un faisceau est un point important de l'algorithme. Malheureusement, aucune information sur le sujet n'est fournie dans [MARK 88] ou dans [MARK 90]. On peut supposer néanmoins que les auteurs se ramènent à un problème bidimensionnel. On peut par exemple considérer le plan du polygone à positionner et les points d'intersection entre les arêtes du faisceau et ce plan. On utilise alors des algorithmes 2D d'intersection entre polygones pour trouver la position de l'objet par rapport au faisceau. Cela reste cependant une solution coûteuse.

En fait, les auteurs proposent uniquement une optimisation pour résoudre ce problème. En effet, ils utilisent une subdivision spatiale régulière (voir ***Paragraphe 3.1.1***)

pour améliorer les temps de calculs du lancer de rayons lors des calculs d'intersection rayon-objet. Cette subdivision sert aussi dans le lancer de faisceaux pour savoir si un objet se situe à l'intérieur d'un faisceau.

L'idée proposée par les auteurs pour positionner un objet par rapport à un faisceau est fondée sur cette subdivision spatiale régulière. Il s'agit, dans un premier temps de trouver les voxels de la subdivision spatiale situés partiellement ou entièrement dans le faisceau, puis de positionner les objets associés à ces voxels par rapport au faisceau. Trouver exactement les voxels étant au moins en partie dans le faisceau est une tâche relativement complexe et certainement coûteuse. Deux solutions sont donc proposées. La première consiste à considérer tous les voxels du parallélépipède englobant le faisceau (***Figure 1.35***). Le problème de cette solution est que beaucoup de voxels inutiles sont pris en compte. La deuxième solution utilise la subdivision régulière pour construire des "tranches" englobant le faisceau (***Figure 1.36***). La construction de ce volume englobant n'est pas détaillée par les auteurs. On pourra toutefois se référer à la solution que nous proposons ***Paragraphe 3.1.2***.

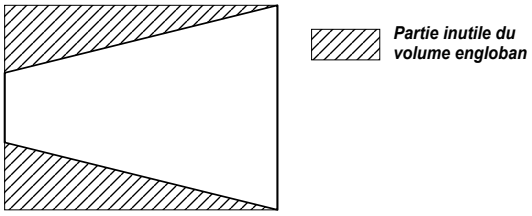


Figure 1.35 : Boîte englobante grossière d'un faisceau.

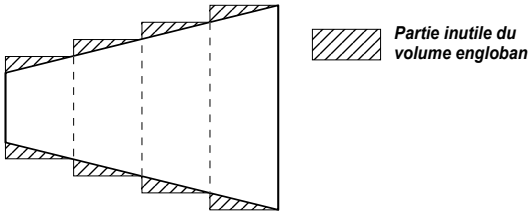


Figure 1.36 : Boîte englobante composée de quatre "tranches".

1.3.1.3 Réfraction

Nous avons vu au ***Paragraphe 1.2.2.4*** que la réfraction sur un plan n'est pas une transformation linéaire. Un faisceau traversant un polygone réfractant est donc déformé et ne ressemble plus à une pyramide. Considérons la ***Figure 1.37***, les quatre arêtes du faisceau primaire traversent un polygone réfractant et sont bloquées par le polygone *P*. Les intersections des rayons réfractés forment un quadrilatère appelé *Q*. La solution proposée par [MARK 90] pour obtenir des effets de réfraction se base sur le concept de *périmètre intérieur*. Ce périmètre est constitué d'un ensemble de pixels de l'écran. Il correspond aux pixels rencontrés par des rayons qui heurtent le polygone *P*, à l'intérieur du quadrilatère *Q*. Les rayons n'ayant pas pour origine un pixel situé à l'intérieur du périmètre ne traversent pas le polygone *P* dans *Q* : pour ces rayons, on utilise un lancer de rayons conventionnel pour évaluer leurs contributions à l'image finale. Les rayons traversant le périmètre sont assurés d'aboutir sur *P* dans *Q*, le calcul du rendu est donc simplifié.

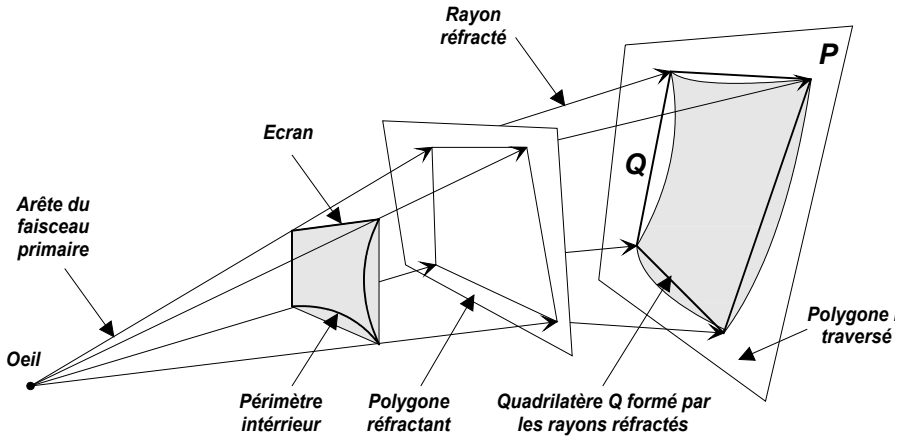


Figure 1.37 : Région intérieure.

La détermination du périmètre intérieur se fait par réduction successive de l'écran. On commence du côté gauche de la région, on lance un rayon à travers chaque ligne de la gauche vers la droite jusqu'à trouver un rayon aboutissant à l'intérieur de *Q*. On procède de même pour les trois autres côtés (***Figure 1.38***). Tous les rayons lancés pour déterminer ce périmètre sont aussi utilisés pour le calcul du rendu. Les rayons restant à lancer traversent donc l'intérieur du périmètre, leurs contributions à l'image finale peuvent donc être évaluées à l'aide d'un simple algorithme de remplissage.

Ce raisonnement est valable à condition de savoir si le faisceau réfracté contient des polygones. Notons que ce test doit prendre en compte le fait que les côtés du faisceau réfracté ne sont pas coplanaires. La solution proposée est de construire un faisceau réfracté englobant tous les rayons réfractés possibles. La construction du faisceau n'est pas clairement proposée dans [MARK 88], on pourra néanmoins trouver une solution simple proposée dans l'article de [GHAZ 92] (***Paragraphe 1.3.2.5***).

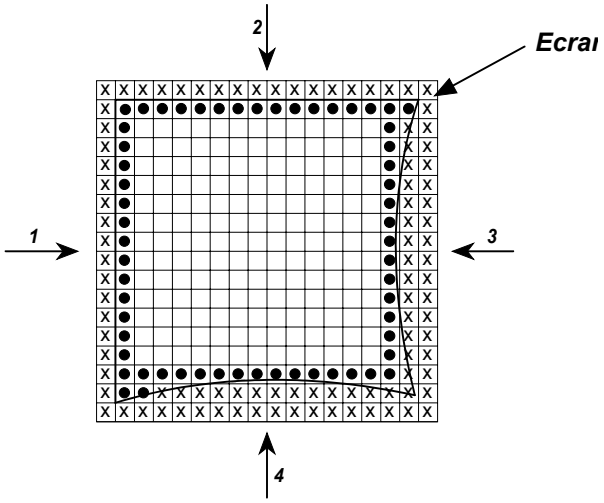


Figure 1.38 : Construction du périmètre intérieur.
Les points et les croix indiquent les pixels d'où on a lancé des rayons pour déterminer le périmètre intérieur. Les points représentent les premiers rayons à l'intérieur du périmètre. L'ordre dans lequel sont lancés les rayons est indiqué par les flèches.

Les auteurs font remarquer, à juste titre, que la détermination des périmètres intérieurs doit être réalisée pour chaque niveau de réfraction. Ceci nécessite de lancer de nombreux rayons réfractés. On peut alors penser que plus le nombre de niveaux de réfraction est important et moins on a intérêt à utiliser le concept de périmètre intérieur.

1.3.1.4 Ombre

Les effets d'ombre sont traités par un lancer de rayons conventionnel. Néanmoins, les auteurs proposent une extension fondée sur des *faisceaux d'ombre*. Supposons qu'un faisceau primaire soit bloqué par un polygone et soit vide. On construit un faisceau pyramidal d'ombre ayant pour sommet une source lumineuse ponctuelle et pour base le polygone bloquant. Si ce faisceau est vide (***Figure 1.39.a***), la région du polygone considérée est entièrement éclairée. Si les quatre arêtes du faisceau d'ombre traversent un même polygone (convexe) avant de heurter la base du faisceau, la région est totalement à l'ombre (***Figure 1.39.b***). Dans ces deux cas, les conditions d'éclairement sont connues. Dans tous les autres cas (***Figure 1.39.c***), ces conditions sont évaluées, soit en subdivisant le faisceau primaire en quatre sous-faisceaux et en construisant quatre sous-faisceaux d'ombre, soit en utilisant un lancer de rayons conventionnel.

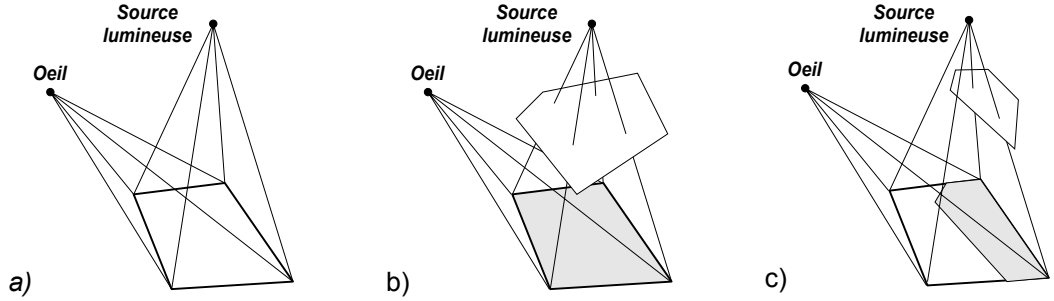


Figure 1.39 : Faisceaux d'ombre.

1.3.1.5 Extension à la radiosité

L'utilisation de faisceaux dans le calcul de la radiosité est proposée ici au niveau de l'évaluation du facteur de forme. Les auteurs fondent leur approche sur [COHE 86] qui considère deux carreaux et détermine la visibilité de l'un par rapport à l'autre. Pour cela, considérons deux carreaux uniquement en forme de quadrilatère et le décaèdre correspondant au volume englobant les huit sommets (***Figure 1.40***).

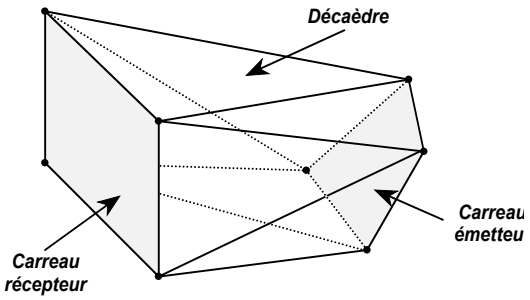


Figure 1.40 : Décaèdre englobant les deux carreaux.

Ce décaèdre est construit à l'aide des deux carreaux et de huit triangles rejoignant les sommets. Le calcul du facteur de forme entre deux carreaux est alors donné par l'algorithme suivant :

si *le décaèdre n'est pas vide*
alors
  si *le critère C de l'**Équation 1-4** indique que la cohérence de la scène est suffisante pour utiliser des faisceaux*
  alors
    Subdiviser le plus petit des deux carreaux en quatre
    Appliquer ce même algorithme sur chacun des sous-faiseaux
  sinon
    Utiliser un lancer de rayons, comme dans [COHE 86], pour évaluer la proportion d'énergie atteignant le carreau récepteur et déterminer le facteur de forme
sinon
  Les deux carreaux sont entièrement visible l'un de l'autre, on peut déterminer le facteur de forme comme dans [COHE 86]

Figure 1.41 : Evaluation du facteur de forme entre deux carreaux.

Ce mécanisme a le mérite de déterminer précisément la visibilité entre deux carreaux. Cependant, il risque d'être coûteux si l'on doit l'appliquer à toutes les paires de carreaux. Cette approche est néanmoins séduisante et une variante est utilisée au ***Paragraphe 5.2***.

1.3.1.6 Discussion

Ce lancer de faisceaux hybride est intéressant par le choix dynamique entre un lancer de rayons et un lancer de faisceaux. Ce choix permet de diminuer les temps de calculs en évitant de construire un grand nombre de faisceaux très fins. Un autre aspect intéressant est une réfraction sans approximation. Cette réfraction est néanmoins coûteuse et une solution plus élégante est proposée dans [GHAZ 92]. Enfin, l'extension à la radiosité est séduisante. Mais si le critère du choix de l'algorithme est en faveur du lancer de rayons, des petits objets peuvent se glisser entre les rayons et introduire des erreurs dans le calcul du facteur de forme.

Une autre utilisation des faisceaux dans le cadre de la radiosité est développée au ***Paragraphe 5.2***.

1.3.2 Lancer de faisceaux pyramidaux et subdivision adaptative

1.3.2.1 Description de la méthode

[GHAZ 92] propose un algorithme dont le but principal est le traitement efficace de tous les problèmes d'aliassage qui, contrairement aux autres méthodes, évite tout calcul d'intersection faisceau-objet. Dans un premier temps, une fenêtre correspondant à la totalité de l'écran est considérée. Cette fenêtre est récursivement subdivisée en quatre, jusqu'à ce qu'une *région uniforme* ou, à défaut, un certain niveau de précision en sous-pixels soit atteint. Dans une région uniforme, considérée non-ambiguë, il n'y a aucun problème d'aliassage à résoudre (marches d'escalier sur les contours, petits objets ou petites ombres) et aucune subdivision n'est nécessaire. Les problèmes d'aliassage pour les textures sont traités séparément. L'algorithme présenté dans [GHAZ 92] traite principalement des scènes constituées de polygones quelconques (convexes, concaves, troué, ...). Cette méthode est d'abord étudiée pour les polygones opaques sans réflexion, puis pour les cas de réflexions et de réfractions.

1.3.2.2 Faisceaux primaires

Dans chaque région, les rayons conventionnels partant de l'oeil et traversant le centre des pixels sont remplacés par des rayons lancés de l'oeil aux quatre coins de la région. Quatre rayons passant par les quatre coins de la région représentent une pyramide, avec l'oeil à son sommet et la région comme intersection (***Figure 1.42***). Une telle pyramide est

appelée *faisceau pyramidal primaire*. A chacun de ces faisceaux est associée une liste de polygones susceptibles d'être au moins partiellement à l'intérieur de la pyramide. Une subdivision d'une région conduit donc à quatre nouvelles sous-pyramides et quatre sous-listes mises à jour comme nous le verrons par la suite.

Notons que le coût en nombre de rayons lancés est, dans le pire des cas, presque identique à celui du lancer de rayons classique, soit un rayon par pixel si tous les faisceaux sont subdivisés.

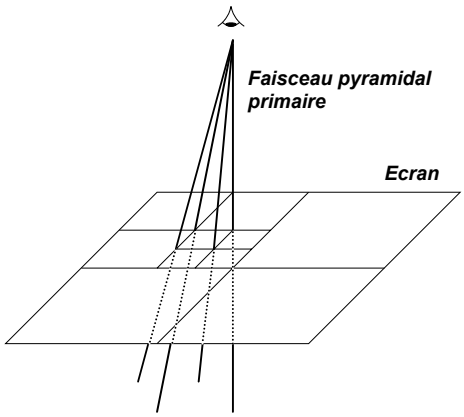


Figure 1.42 : Subdivision récursive de l'image.

Pour déterminer le type (uniforme ou ambigu) d'une région, on considère son faisceau primaire et la liste correspondante de polygones. Une série de tests de difficulté croissante est alors utilisée. On effectue dans un premier temps un test très simple utilisant les coordonnées des sommets des polygones et l'équation des quatre plans de la pyramide. Ce test trivial permet de détecter rapidement les polygones qui n'ont visiblement aucune intersection possible avec la pyramide et ceux qui sont entièrement à l'intérieur de celle-ci (***Figure 1.43***). Les premiers sont écartés de la liste et les seconds sont marqués. La liste courante est alors mise à jour, la position exacte des polygones non marqués par rapport à la pyramide n'est pas définie (***Figure 1.44***) et sera déterminée par d'autres tests. Ceux-ci ne sont pas aussi rapides que le premier mais ne concernent généralement qu'un nombre restreint de polygones.

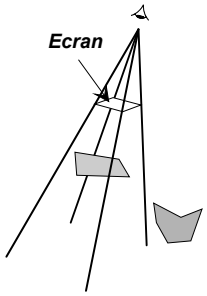


Figure 1.43 : Polygone entièrement à l'intérieur et polygone entièrement à l'extérieur.

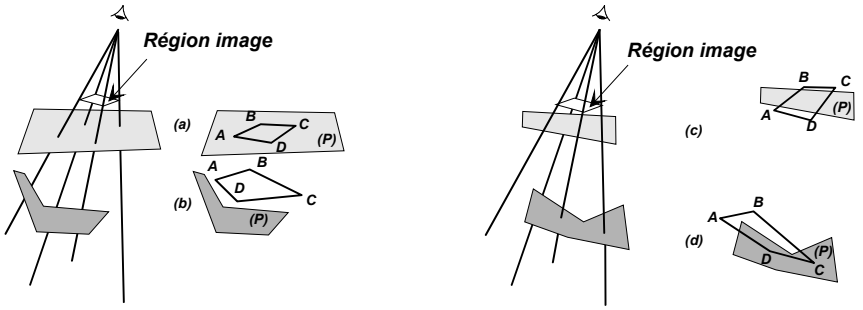


Figure 1.44 : Polygones avec une position non-définie par rapport à la pyramide.

Pour le deuxième test, tous les polygones de la liste courante, sont triés suivant leur distance à l'oeil. On considère alors le premier polygone, *P*, de cette liste triée. Si *P* est déjà marqué comme un polygone intérieur, il ne peut donc être le seul polygone visible dans la pyramide (au moins un autre polygone ou le fond initial est visible) et la région correspondante est subdivisée en quatre. Le premier test est alors appliqué aux quatre sous-pyramides afin de mettre à jour leur liste de polygones. La subdivision récursive prend fin lorsqu'une région uniforme est trouvée ou qu'un certain niveau de précision en sous-pixels est atteint. Si *P* n'est pas marqué intérieur, sa position par rapport à la pyramide est considérée comme indéfinie (***Figure 1.44***).

Dans ce cas, pour déterminer le type de *P*, il faut d'abord trouver les points d'intersection *A*, *B*, *C* et *D* de son plan avec les quatre arêtes de la pyramide. Ces points constituent les quatre sommets d'un quadrilatère *ABCD* (***Figure 1.44***).

On distingue alors deux groupes de polygones avec une position indéfinie :

- les polygones ayant au moins un sommet à l'intérieur de la pyramide et un à l'extérieur (***Figure 1.44.d***),
- les polygones n'ayant aucun sommet à l'intérieur de la pyramide (***Figure 1.44.a***, ***b*** et ***c***).

Pour déterminer si *P* fait partie du premier groupe, il suffit de vérifier qu'au moins un de ses sommets est à l'intérieur de *ABCD* et un à l'extérieur. Si c'est le cas, *P* ne peut être le seul polygone visible dans la pyramide (au moins un polygone ou le fond initial est visible). Par conséquent, la région correspondante est subdivisée et quatre nouvelles sous-pyramides seront obtenues.

Si *P* est un polygone du deuxième groupe, alors des tests simples peuvent déterminer l'un des cas suivants :

- *P* couvre entièrement la pyramide (***Figure 1.44.a***) ;
- *P* a une intersection partielle avec la pyramide (***Figure 1.44.c***) ;
- *P* est entièrement à l'extérieur de la pyramide (***Figure 1.44.b***).

Si *P* est du premier type, tous les sommets de *ABCD* sont à l'intérieur de *P* et ceux de *P* sont à l'extérieur de *ABCD*. Dans les autres cas, tous les sommets de *ABCD* sont à l'extérieur de *P* et ceux de *P* sont à l'extérieur de *ABCD*. Le deuxième cas peut être distingué du dernier en testant la position des bords de *P* par rapport à ceux de *ABCD*.

Si *P* couvre entièrement la pyramide, s'il n'existe pas d'intersection entre *P* et d'autres polygones à l'intérieur de la pyramide et s'il n'y a pas d'autres polygones entre *P* et le sommet de la pyramide, alors aucune subdivision n'est nécessaire à cette étape. Nous devons alors tester, comme nous le verrons par la suite, les faisceaux d'ombre formés par *ABCD* et les sources lumineuses.

Si *P* a une intersection partielle avec la pyramide, alors la région correspondante est subdivisée. Dans le cas où *P* est situé complètement en dehors de la pyramide (***Figure***

1.44.b), il est écarté et le polygone suivant de la liste est considéré. Si la liste est vide, la région est considérée uniforme et prend la couleur du fond initial.

A la fin de ces subdivisions récursives, on obtient un ensemble de régions considérées uniformes pour les faisceaux primaires. L'***Image 1*** montre ce résultat sur une scène très simple composée d'un cube vert posé sur un plan réfléchissant.

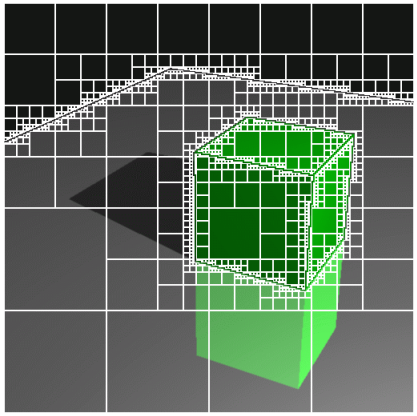


Image 1 : Subdivisions pour les faisceaux ***primaires*** *(carrés rouges). Chaque carré est donc considéré comme une région uniforme.*

1.3.2.3 Faisceaux d'ombre

Une région est détectée uniforme si après les tests effectués sur sa pyramide primaire, un seul polygone est visible dans la pyramide. L'intersection *ABCD* de ce polygone avec la pyramide primaire est utilisée pour former les pyramides d'ombre. Une pyramide de ce type a *ABCD* comme intersection et une source lumineuse *S* comme sommet (***Figure 1.45***). La visibilité de *ABCD* à partir de *S* est déterminée avec des tests similaires à ceux utilisés pour le faisceau primaire mais, outre les plans des côtés de la pyramide, le plan de *ABCD* est également considéré. Ces tests sont cependant plus simples que les précédents, car il ne faut distinguer par rapport à *S* que l'un des trois cas suivants :

- *ABCD* est entièrement visible de *S*, il est donc directement éclairé ;
- *ABCD* est entièrement caché de *S* par un polygone bloquant, il est donc dans l'ombre ;
- *ABCD* est partiellement visible de *S*.

Dans les deux premiers cas, il n'y a pas d'ambiguïté et aucune subdivision n'est nécessaire. Une nouvelle source lumineuse avec la pyramide correspondante sera examinée. Dans le troisième cas, la région correspondante à *ABCD* sur l'écran est subdivisée. Notons que cette subdivision n'entraîne pas d'autre test sur les quatre nouvelles sous-pyramides primaires. Nous devons uniquement tester les sous-pyramides d'ombre afin de déterminer la visibilité de leurs intersections à partir de chaque source lumineuse. Finalement, une région est considérée comme uniforme si la visibilité dans son faisceau primaire et dans toutes ses pyramides d'ombre est sans ambiguïté.

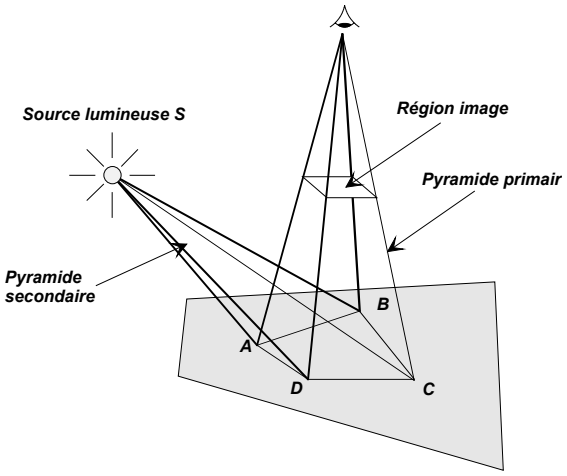


Figure 1.45 : Pyramide primaire et pyramide d'ombre.

Notons que les pyramides primaires et d'ombre ne sont là que pour détecter des régions ambiguës (arêtes aliassées, petits objets et petites ombres) et permettent ainsi, après subdivision, la localisation des régions uniformes (***Image 2***). La couleur de ces régions uniformes est alors calculée avec un lancer de rayons classique (***Paragraphe 1.3.2.6***).

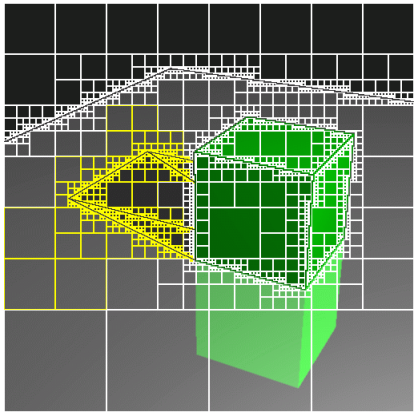


*Image 2 : Subdivisions des faisceaux **primaires** (en rouge) et subdivisions des faisceaux d'**ombre** (en bleu).*

1.3.2.4 Réflexions

Le concept de l'oeil virtuel tel qu'il est suggéré entre autres dans [HECK 84] est utilisé pour les réflexions. L'oeil virtuel est le point V symétrique de l'oeil par rapport au plan R du polygone réfléchissant (***Figure 1.46***). Si $ABCD$ est l'intersection entre le faisceau primaire et le polygone réfléchissant couvrant entièrement la pyramide, alors une pyramide de réflexion est produite avec V comme sommet et $ABCD$ comme intersection. Le faisceau réfléchi correspond alors à la partie supérieure de la pyramide réfléchie, nous parlerons de *pyramide tronquée* (***Figure 1.46***). Cette pyramide tronquée subit des tests similaires à ceux utilisés pour le faisceau primaire mais, outre les plans des quatre côtés de la pyramide, R est également considéré et seuls les polygones situés du côté de R opposé à V sont retenus. Si plus d'un polygone est visible dans cette pyramide tronquée réfléchie, on subdivise cette pyramide d'une façon similaire aux autres pyramides. Les résultats des subdivisions successives sont montrés ***Image 3***.

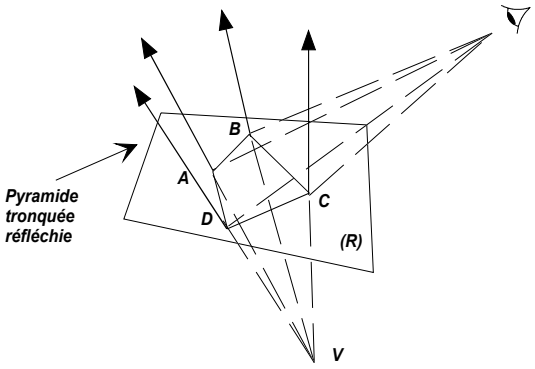


Figure 1.46 : Pyramide tronquée réfléchie.

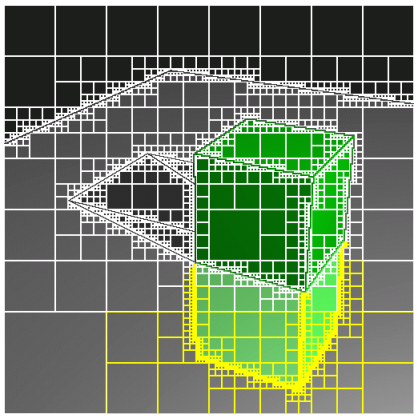


Image 3 : Subdivisions des faisceaux **primaires** et d'**ombre** (en rouge) et subdivisions des faisceaux de **réflexion** (en bleu).

1.3.2.5 Réfraction

La réfraction n'est pas une transformation linéaire (voir ***Paragraphe 1.2.2.4***). Elle pose par conséquent un problème difficilement soluble avec des algorithmes de lancer de faisceaux qui calculent explicitement les intersections objet-faisceau telles que [HECK 84] (***Paragraphe 1.2.2***). Il existe des solutions assez précises mais qui nécessitent des calculs complexes [SHIN 87] (***Paragraphe 1.2.1***). L'approche de [GHAZ 92] consiste à construire une *pseudo-pyramide* englobant tous les rayons possibles réfractés par une région d'un polygone réfractant.

Soit *ABCD* l'intersection entre la pyramide primaire et le plan du polygone réfractant et r_1, r_2, r_3 et r_4 les rayons réfractés en *A*, *B*, *C* et *D* (***Figure 1.47***). Un bord de *ABCD* et un rayon réfracté forment un plan.

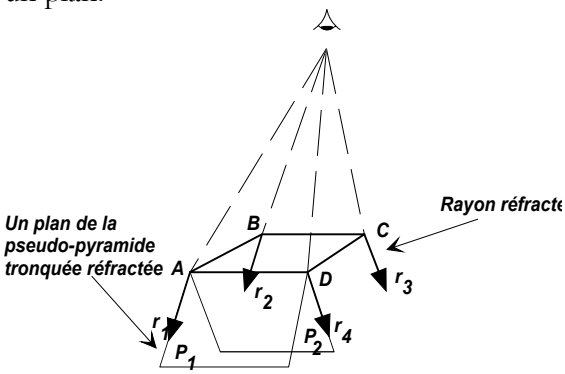


Figure 1.47 : Sélection des plans pour construire la pseudo-pyramide tronquée réfractée.

Considérons, par exemple, P_1 et P_2, les plans définis respectivement par le bord *AD* et le rayon r_1, le bord *AD* et le rayon r_4. Un de ces deux plans (P_2 dans notre exemple) se situe "entre" *ABCD* et l'autre plan. Utilisons le plan extérieur (P_1 dans notre exemple) ainsi que les trois autres définis de la même façon à partir des côtés *AB*, *BC*, et *CD* et les rayons réfractés pour construire un volume réfracté. Ce volume forme la *pseudo-pyramide tronquée réfractée* d'origine ABCD. Cette pseudo-pyramide contient tous les rayons réfractés par *ABCD*. Elle subit les mêmes tests de subdivision que ceux effectués dans le cas des autres pyramides secondaires. Ces pseudo-pyramides tronquées réfractées ne servent qu'à détecter

les régions ambiguës. Le calcul de la couleur d'une telle région dépend uniquement des rayons réfractés et de la précision fournie par le lancer de rayons. Les résultats des subdivisions successives sont présentés ***Image 4***.

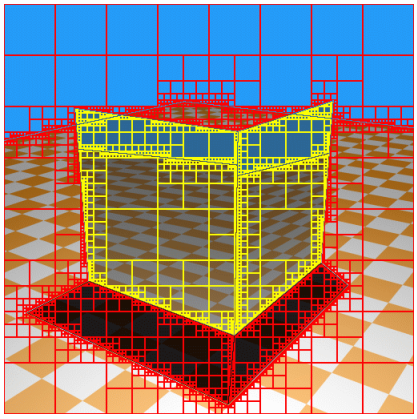


*Image 4 : Subdivisions des faisceaux **primaires** et d'**ombre** (en rouge) et subdivisions des faisceaux de réfraction (en jaune).*

Remarque : Pour chaque subdivision d'un faisceau de réflexion ou de réfraction, il est nécessaire de lancer un rayon au centre de la région image et de calculer le chemin qu'il parcourt pour atteindre le niveau de récursion courant (***Figure 1.48***). En effet, c'est le seul moyen pour définir l'arête commune aux quatre sous-faisceaux à construire.

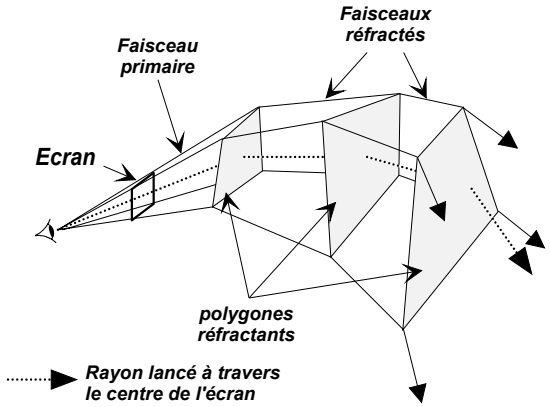


Figure 1.48 : Subdivision du chemin parcouru par un faisceau

1.3.2.6 Calcul de la couleur des pixels

Les parties précédentes avaient pour but de détecter les arêtes, les petits objets et les petites ombres correspondant à des régions ambiguës, puis d'effectuer une subdivision adaptative récursive de cette région jusqu'à obtenir une région uniforme ou jusqu'à atteindre une précision en sous-pixels suffisante. Une région est détectée uniforme si après le test de toutes les pyramides correspondantes il n'existe aucune ambiguïté sur la visibilité des polygones à l'intérieur de celle-ci. Deux sortes de régions uniformes sont à distinguer : celles qui sont égales ou plus grandes qu'un pixel et celles qui ne le sont pas.

Dans le premier cas, une région uniforme correspond à un ou plusieurs pixels et aucune subdivision de pixels n'est nécessaire. Quatre rayons passant par les quatre coins de chaque pixel sont lancés. La couleur de chaque pixel est la moyenne simple des couleurs obtenues à l'aide des quatre rayons.

Dans le deuxième cas, une région uniforme correspond à un ou plusieurs sous-pixels. Quatre rayons passant par les quatre coins de chaque sous-pixel sont lancés. La couleur de

chaque sous-pixel est la moyenne simple ou pondérée des couleurs obtenues à l'aide des quatre rayons. Enfin, la couleur finale de chaque pixel est la moyenne simple ou pondérée de la couleur des sous-pixels.

Pour les régions ambiguës mais dont la précision est suffisante, un seul rayon est lancé au centre du sous-pixel. On dispose ainsi de cinq valeurs, quatre sur les coins de la région correspondant aux arêtes du faisceau et une au centre, pour calculer l'intensité du pixel.

1.3.2.7 Textures

Il existe deux démarches différentes pour traiter l'aliassage des textures. La première consiste en un suréchantillonnage dans lequel la subdivision de pixels est nécessaire non seulement pour les pixels situés sur les contours, mais aussi pour ceux à l'intérieur des régions avec textures. Les inconvénients d'une telle démarche sont une augmentation considérable du temps de calculs et dans certains cas la persistance du phénomène de moirés [GHAZ 91].

La démarche proposée par l'auteur est une méthode de filtrage a priori comme [GANG 84]. Ce filtrage est bien adapté au lancer de faisceaux car il nécessite pour chaque pixel d'écran le calcul des deux taux de compression de la texture afin de choisir les versions appropriées de la texture préfiltrée. Ces taux de compression sont obtenus par comparaison entre la taille d'un pixel d'écran et celle de l'intersection entre sa pyramide correspondante et le polygone portant la texture.

1.3.2.8 Scènes non-polygonales

[GHAZ 92] propose brièvement une extension de son algorithme de lancer de faisceaux pyramidaux pour des scènes non polygonales contenant des objets tels que des sphères ou des cylindres. Pour ces deux types d'objets, il est possible de connaître leur position par rapport à une pyramide et donc de détecter les régions uniformes. Par contre, aucune information n'est donnée concernant des objets tels que le cône, le tore ou plus généralement des volumes de révolution. Nous proposons au ***Paragraphe 2.2*** une extension de l'algorithme de [GHAZ 92] à certaines de ces scènes non polygonales.

1.3.2.9 Discussion

La technique du *lancer de faisceaux pyramidaux avec subdivision adaptative* proposée par [GHAZ 92] est une approche relativement simple et efficace pour la visualisation de scènes polygonales. De plus, elle fournie un antialiassage robuste. Les faisceaux pyramidaux sont utilisés comme volumes englobants pour détecter les régions uniformes ou ambiguës. Contrairement aux méthodes similaires, aucun calcul d'intersection explicite faisceau-polygone n'est nécessaire ce qui autorise le traitement de scènes non polygonales comme les scènes "CSG". Le nombre de calculs d'intersection rayon-polygone est limité et les petits objets et les petites ombres sont détectés et traités. Un avantage important de cette méthode est l'absence des approximations linéaires ou calculs complexes utilisés dans les méthodes analogues [HECK 84] [SHIN 87] pour le cas de la réfraction.

Sous cette forme, cet algorithme est relativement coûteux en temps. Plusieurs optimisations sont possibles, en particulier la modélisation des scènes à partir de polyèdres et l'utilisation d'une subdivision spatiale régulière [GHAZ 98]. Les détails des optimisations et améliorations sont proposés au ***Chapitre 3***.

1.4 LES FAISCEAUX COMME OUTIL PONCTUEL DU LANCER DE RAYONS

Nous présentons dans ce paragraphe une utilisation des faisceaux pour résoudre un problème ponctuel. En effet, il s'agit ici de diminuer les temps de calculs nécessaires pour produire les ombres portées en utilisant des faisceaux pyramidaux.

1.4.1 Accélération du calcul des ombres

1.4.1.1 Principe général

[CHOI 92] présente une structure appelée *pyramide d'ombre* permettant de réduire le temps passé dans le calcul des ombres. Pour cela, une pyramide d'ombre est construite pour chaque objet (polygone convexe) rencontré pour la première fois par un rayon et pour chaque source lumineuse. Une liste des objets contenus dans chacune des pyramides est construite et associée aux différentes pyramides. Les pyramides sont elles-mêmes associées aux différentes faces des objets.

Pour savoir si un point d'une face est à l'ombre, un rayon issu de ce point est lancé vers chacune des sources, comme pour un lancer de rayons conventionnel. Par contre, les tests d'intersection ne sont effectués qu'avec les objets contenus dans les listes associées aux pyramides concernant la face rencontrée de l'objet.

Cette structure étant assez lourde, l'auteur propose la méthode PYSHA. C'est une méthode hybride pour tester si un point d'un polygone ou d'une sphère est à l'ombre. Elle combine des subdivisions spatiales avec sa structure de pyramide d'ombre.

1.4.1.2 Construction des pyramides et des listes

Les pyramides d'ombre ont pour base la face de l'objet rencontré et pour sommet une source lumineuse. Pour profiter de la cohérence de la scène, l'auteur propose une structure à mi-chemin entre la subdivision spatiale régulière et la subdivision adaptative. En effet, il commence par construire une subdivision régulière de voxels, puis regroupe ces voxels par blocs pour construire une structure ressemblant à un "*octree*" (***Figure 1.49***). A chaque noeud de l'arbre correspond une liste de *sous-espaces uniformes* et à chaque feuille la liste des objets traversant le voxel pointé. L'auteur propose un arbre de profondeur trois.

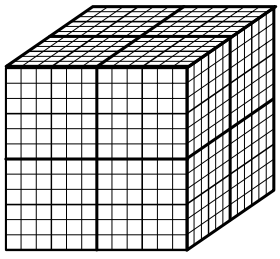


Figure 1.49 : Subdivision spatiale à trois niveaux de "précision".

Pour construire la liste des objets susceptibles d'être dans une pyramide d'ombre, l'auteur cherche à trouver les objets clairement à l'extérieur de la pyramide. Par négation, il peut alors construire la liste. Pour cela, deux étapes successives sont proposées :

1. trouver les sous-espaces uniformes clairement à l'extérieur de la pyramide ;
2. trouver les objets contenus dans les sous-espaces n'ayant pu être écartés à l'étape 1 et clairement situés en dehors de la pyramide.

Les objets n'ayant pu être positionnés clairement à l'extérieur du faisceau sont ajoutés dans la liste des objets associés à la pyramide.

Pour affirmer qu'un objet ou qu'un sous-espace est en dehors d'une pyramide, on cherche un plan séparant l'espace en deux demi-espaces, l'un contenant entièrement la pyramide et l'autre entièrement l'objet. Si un tel plan existe, on peut affirmer que l'objet est à l'extérieur de la pyramide. [CHOI 92] propose différents plans :

a) le plan formant la base de la pyramide (la face de l'objet rencontré) (***Figure 1.50.a***) ;
b) les plans formant les côtés de la pyramide (***Figure 1.50.b***) ;
c) des plans tangents aux arêtes de la pyramide (***Figure 1.50.c***).

Les plans tangents aux arêtes sont orientés de sorte que leur normale corresponde à la moyenne des normales des deux plans adjacents à l'arête considérée. Ces plans permettent d'écarter certains objets pour lesquels les plans de type *b)* n'ont pas suffit. C'est par exemple le cas de l'objet *C* dans la ***Figure 1.50.d***.

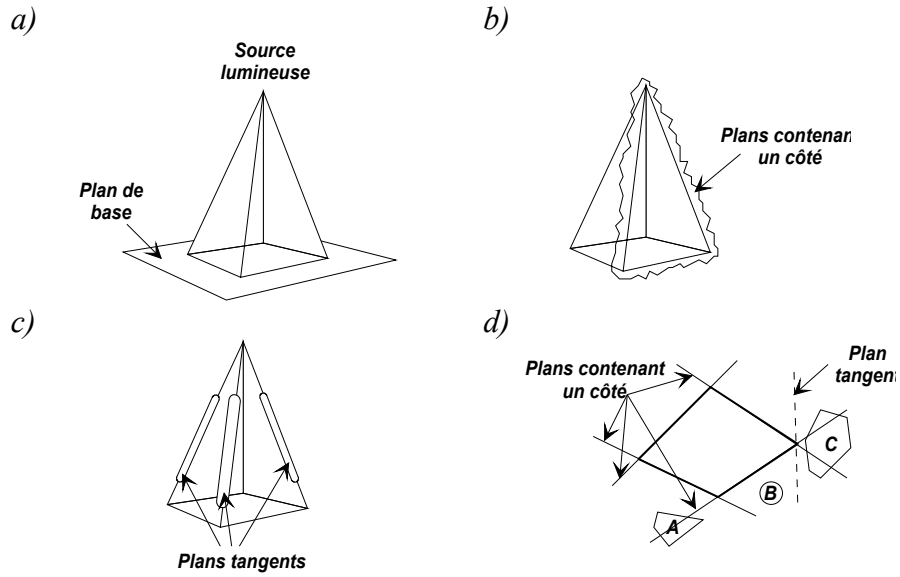


Figure 1.50 : Plans permettant d'écarter des objets visiblement hors de la pyramide. (dans la figure d, les objets A et B sont écartés grâce aux plans de type b) et l'objet C est écarté grâce à un plan de type c))

Pour améliorer les performances de l'algorithme, la liste est triée en se basant sur trois règles intuitives :

- plus un objet est grand par rapport aux autres et plus la probabilité qu'il a de couper entièrement un faisceau d'ombre est grande ;
- plus un objet est proche d'une source lumineuse et plus son ombre portée est grande ;
- plus un objet est orienté perpendiculaire à la direction de la pyramide et plus son ombre portée est importante.

Ces trois règles sont rassemblées dans l'équation du *facteur de forme des ombres f* suivante :

$$f = (A/d)\ |N \bullet P|$$

où A est la surface du polygone, d la distance de la source au barycentre du polygone, N la normale du polygone et P la direction de la pyramide. La direction de la pyramide correspond à un axe passant par le barycentre du polygone et par la source lumineuse (***Figure 1.51***).

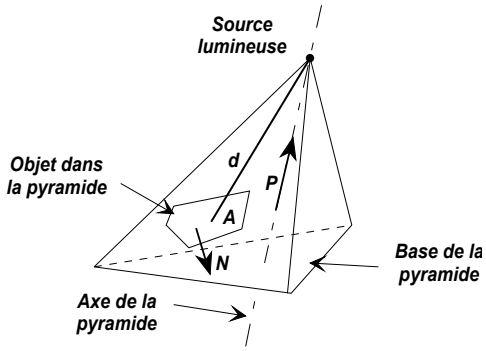


Figure 1.51 : Facteur de forme des ombres.

1.4.1.3 Raffinement de la liste

Une autre stratégie proposée par [CHOI 92] est fondée sur l'utilisation d'un algorithme de lancer de rayons travaillant par lignes de balayage et des objets constitués de polygones convexes. Il s'agit de construire des sous-listes des listes d'objets associées aux pyramides. Un objet est mis dans une sous-liste seulement si un rayon d'ombre l'a déjà rencontré pour la ligne courante et que le dernier rayon d'ombre ne l'a plus rencontré. Ceci correspond en fait à un objet qui a fait de l'ombre sur la ligne de balayage courante et qui n'en fait plus. Cette liste permet de ne plus tester d'intersection avec les objets ayant déjà produit une ombre portée.

1.4.1.4 PYSHA

[CHOI 92] étudie en détails l'évolution de son algorithme en fonction de différents paramètres tels que la taille des pyramides, la surface des objets dans la pyramide, leur nombre, … Il conclut que l'utilisation des pyramides d'ombre est optimale pour des pyramides ayant une large base et contenant un petit nombre d'objets. Dans le cas contraire, le surcoût de travail engendré par les différentes listes peut pénaliser l'algorithme. Après une comparaison de son algorithme avec un lancer de rayons optimisé avec une subdivision spatiale régulière, il propose PYSHA. PYSHA est une version hybride mélangeant un lancer de rayons avec subdivision spatiale régulière et la structure de pyramides d'ombre. En fait, pour chaque polygone rencontré, on va choisir d'utiliser l'une ou l'autre des deux méthodes. Le critère de choix est empirique et fondé sur la surface d'une éventuelle pyramide d'ombre et sur le nombre d'objets qu'elle contient. Ainsi modifié, l'algorithme proposé semble, sur la base de six scènes tests, plus rapide. Le gain varie entre 1.2 et 2.

1.4.1.5 Discussion

La solution proposée par [CHOI 92] met en oeuvre une optimisation classique du lancer de rayons : la subdivision régulière de l'espace. De plus, cette subdivision bénéficie aussi aux pyramides d'ombre lors du positionnement des objets. On peut regretter que ce positionnement ne soit pas optimum et n'utilise pas au maximum la subdivision spatiale régulière. Par la suite, au ***Paragraphe 3.1***, nous proposons une optimisation de l'algorithme de [GHAZ 92] qui utilise le même esprit que celui de [CHOI 92]. Notons aussi que l'utilisation seule d'un lancer de rayons pour calculer l'ombrage induit des problèmes d'aliassage, en particulier la disparition des petites ombres.

1.5 CONCLUSION

Ce chapitre a été consacré aux différentes variantes du lancer de rayons cherchant à profiter au maximum de la cohérence de la scène en regroupant les rayons. Il est maintenant temps de faire la synthèse de ces méthodes et de noter les apports possibles de chacune d'entre elles dans le cadre de nos recherches.

Le ***Tableau 2*** résume les méthodes et leurs intérêts respectifs.

La première partie de ce chapitre a été consacrée à l'utilisation de rayons d'épaisseur infinitésimale. Dans ce cadre, [SPEE 85] cherche à prédire le chemin que va parcourir un rayon en construisant un tunnel autour du dernier rayon lancé. Cette méthode est annoncée comme une voie à ne pas suivre. Elle apporte cependant une solution pour positionner un objet par rapport à un cylindre. La deuxième solution est celle de [MÜLL 86] qui lance en même temps tous les rayons d'un même niveau de récursion. Ce parcours en largeur de l'arbre de [WHIT 80] permet une gestion efficace des rayons lancés. La troisième solution est parmi les plus connues est celle de [SHIN 87] qui regroupe les rayons en crayons. Les équations proposées permettent de maîtriser la qualité de la réfraction, cependant ce regroupement n'est possible que sur des surfaces continues. En particulier, la solution n'est pas applicable sur les bords des polygones et ne traite donc pas les marches d'escalier.

La deuxième partie a été consacrée aux algorithmes utilisant uniquement des faisceaux. [AMAN 84] propose de lancer des cônes pour traiter les divers problèmes d'aliassage. Malheureusement, cette forme conique est peu appropriée à des scènes polygonales (entre autres) et de nombreux problèmes en découlent. La solution de [HECK 84] semble nettement plus intéressante. L'utilisation d'un faisceau de section polygonale permet de profiter de tous les algorithmes de découpage en 2D. De plus, la représentation de l'image de la scène par ses contours autorise un traitement efficace de l'aliassage même si cela n'était pas le but premier de cette solution. Les seuls faits à reprocher à cette solution sont l'approximation de la réfraction et la complexité des calculs d'intersection faisceau-objet.

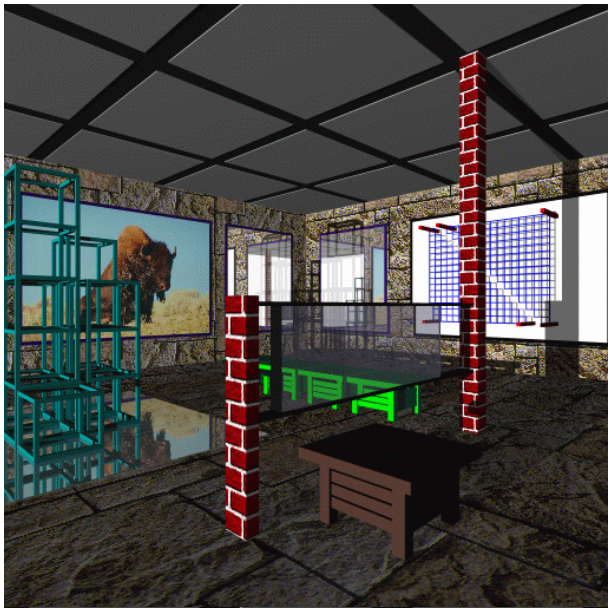
La troisième partie du chapitre s'est employée à présenter les solutions hybrides. Les deux solutions proposées sont relativement semblables alors qu'il n'y a eu aucune concertation entre les auteurs. Les buts sont toutefois différents. Dans le cas de [MARK 90], le but est un gain de temps, il alterne donc l'utilisation d'un lancer de rayons faisceaux avec l'utilisation d'un lancer de rayons. Dans le cas de [GHAZ 92], c'est le traitement de l'aliassage qui prime et l'on utilise le lancer de faisceaux pour détecter les régions à problèmes. Ce n'est que dans un deuxième temps que l'on lance des rayons pour le calcul du rendu proprement dit. En fait, chacun arrive à ses fins, [MARK 90] obtient de meilleurs temps de calculs par contre il ne traite pas systématiquement les problèmes d'aliassage et [GHAZ 92] résout ces problèmes mais propose une solution plus lente (voir ***Chapitre 2***).

Enfin, la quatrième partie était consacrée à l'utilisation d'un faisceau comme outil pour améliorer les temps de calculs des ombres. Cette approche apporte différentes solutions quant aux optimisations possibles lors de l'utilisation de faisceaux.

Méthode	Objectifs visés	Moyens employés	Résultats	Intérêts
[SPEE 85] *Prédiction des chemins empruntés par les rayons lancés*	Gain de temps	Créer un tunnel cylindrique autour du chemin parcouru par le dernier rayon lancé.	Décevant, l'algorithme est lent.	Détermination de la position d'un objet par rapport à un cylindre.
[MÜLL 86] *Lancement simultané de tous les rayons à chaque niveau de la récursion*	Gain de temps.	Utilisation d'un plan de balayage pour faire un parcours en largeur de l'arbre.	Mitigé, l'algorithme n'est pas plus rapide.	Une solution pour traiter l'aliassage est possible, en particulier dans le cas des petits objets.
[SHIN 87] *Faisceau générique et quantification des approximations réalisées*	Gain de temps.	Regroupement de rayons voisins sous forme de crayons.	Bon résultat , en particulier pour la réfraction. Pas de traitement de l'aliassage sur les contours des objets.	Une évaluation de la "qualité" de la réfraction est proposée de manière analytique.
[AMAN 84] *Lancer de rayons avec des cônes*	Traitement de l'aliassage, effets de pénombre.	Remplacer les rayons par des cônes.	Mitigé, effets de pénombre possibles, la réfraction est largement approchée, des problèmes théoriques et pratiques subsistent.	Détermination de la position d'une sphère ou du polygone par rapport à un cône.
[HECK 84] *Lancer de faisceaux sur des objets polygonaux*	Gain de temps.	Lancer un faisceau à travers l'écran puis découper le faisceau au fur et à mesure de sa propagation dans la scène.	Bon, on obtient tous les contours des objets, malheureusement la réfraction est approchée.	La construction d'un graphe d'adjacence permet la détection exacte des contours des objets, des ombres et des réflexions.
[MARK 90] *Choix dynamique entre les rayons et les faisceaux*	Gain de temps.	Un lancer de rayons conventionnel et un lancer de faisceaux avec une subdivision adaptative.	Bon, l'algorithme est certainement plus rapide, mais il ne tient pas compte des petits objets. De plus les réfractions multiples sont coûteuses à traiter.	La subdivision adaptative des faisceaux, l'optimiser pour positionner un objet par rapport à un faisceau. L'extension au calcul du facteur de forme.
[GHAZ 92] *Lancer de faisceaux pyramidaux et subdivision adaptative*	Traitement précis de l'aliassage, utilisation de scènes non polygonales.	Un lancer de faisceaux avec une subdivision adaptative pour détecter les régions aliassées. Un lancer de rayons pour calculer le rendu.	Bon, tous les problèmes d'aliassage sont résolus, le calcul de la réfraction est aussi précis qu'un lancer de rayons conventionnel. L'utilisation de scènes non-polygonales est possible. Le temps de calculs sont importants.	La subdivision adaptative des faisceaux, la détermination de la position d'un objet par rapport à un faisceau sans calcul explicite.
[CHOI 92] *Accélération du calcul des ombres*	Gain de temps.	Un faisceau pyramidal d'ombre pour chaque polygone visible et la liste des objets qu'ils contiennent.	Bon, le temps de calculs des ombres est diminué.	L'utilisation d'une subdivision régulière de l'espace pour optimiser le positionnement d'un objet par rapport à un faisceau. Le critère de tri des listes.

Tableau 2 : Synthèse des différentes méthodes proposées.

2. LANCER DE FAISCEAUX PYRAMIDAUX



Le chapitre précédent a été consacré à l'étude des différentes variantes du lancer de rayons et aux algorithmes utilisant des faisceaux. On a vu que le lancer de rayons n'est pas toujours très précis dans son rendu et que des petits objets et des petites ombres disparaissent parfois. Les algorithmes de lancer de faisceaux, par contre, peuvent résoudre ces problèmes d'aliassage. Malheureusement, la plupart d'entre eux cherchent à calculer explicitement les intersections entre les faisceaux et les objets. Cette approche devient alors complexe et coûteuse. D'autre part, les algorithmes de lancer de faisceaux ne résolvent pas toujours de manière très réaliste les effets de réfraction ou les résolvent de manière très complexe. Cependant, parmi les différentes propositions présentées au chapitre précédent, le "lancer de faisceaux pyramidaux" [GHAZ 92] résout efficacement le problème difficile de la disparition des petits objets et petites ombres. De plus, il produit des réfractions aussi réalistes que le lancer de rayons.

Ce lancer de faisceaux pyramidaux a constitué le point de départ de nos travaux. Nous l'avons implémenté et comparé à un lancer de rayons conventionnel [HASE 95]. Les résultats qualitatifs et quantitatifs sont présentés dans la première partie de ce chapitre.

Dans la deuxième partie, nous proposons une extension de l'algorithme. En effet, [GHAZ 92] annonce la possibilité d'appliquer sa méthode à des scènes non polygonales. Les scènes les plus communes dans ce domaine sont les scènes construites à partir de volumes élémentaires simples et d'opérations ensemblistes, on parle de scènes "Constructive Solid Geometry" ou "CSG". Nous nous sommes donc penchés sur cette possibilité [HASE 96].

2.1 IMPLEMENTATION DU LANCER DE FAISCEAUX PYRAMIDAUX

Rappelons que l'algorithme de lancer de faisceaux pyramidaux proposé par [GHAZ 92] comporte deux étapes (***Paragraphe 1.3.2***). La première a pour objet la détection les régions uniformes et des régions ambiguës. Pour cela, on lance récursivement des faisceaux à travers l'écran et les objets sont positionnés par rapport à ces faisceaux. Si le contour d'un objet, d'une ombre, d'une réflexion ou d'une réfraction est visible dans le faisceau, celui-ci est subdivisé en quatre sous-faisceaux. On considère alors, récursivement, chacun de ces sous-faisceaux. Si aucun contour n'est visible, nous sommes en présence d'une région uniforme et aucune subdivision supplémentaire n'est nécessaire. La récursion prend fin lorsque l'on a atteint une région uniforme ou une limite de subdivision en sous-pixels fixée a priori (en général 8x8 sous-pixels). La deuxième étape est consacrée au rendu proprement dit. On utilise un lancer de rayons conventionnel optimisé par les informations collectées dans la première étape. On lance un rayon par pixel contenu dans une région uniforme et plusieurs rayons par pixels situés dans une région ambiguë (voir ***Paragraphe 1.3.2.6***).

[GHAZ 92] spécifie dans son article que les optimisations des lancers de rayons conventionnels peuvent s'appliquer au lancer de faisceaux. Ces optimisations peuvent être utilisées lors du positionnement des objets par rapport aux faisceaux et lors du calcul effectif du rendu avec le lancer de rayons. Nous avons donc utilisé, dans un premier temps [HASE 95], une hiérarchie de sphères englobantes [RUBI 80] [KAY 86]. Pour étudier les résultats qualitatifs et quantitatifs de cette méthode, nous avons utilisé deux scènes tests. La première représente un "*rubic cube*"[5] (***Image 5***) favorisant au maximum l'utilisation d'une hiérarchie de sphères englobantes. En effet, dans cette scène, chaque cube est contenu dans une sphère englobante, chaque regroupement de huit sphères est lui-même contenu dans une sphère englobante. La deuxième scène est plus générale, elle représente un "*musée virtuel*"[6] (***Image 6***) et comporte en particulier des objets allongés (pilier, "échelles", ...). Ces exemples sont comparés à un lancer de rayons optimisé dans les mêmes conditions, c'est-à-dire avec une hiérarchie de volumes englobants.

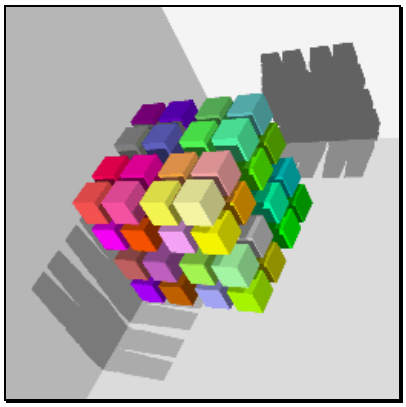


Image 5 : "rubic cube"

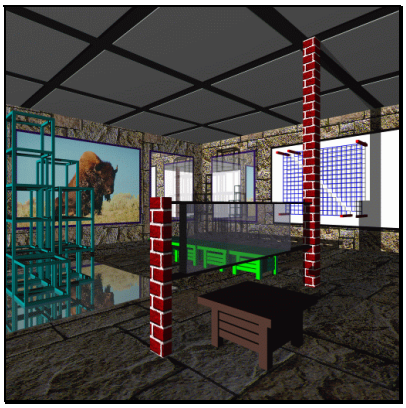


Image 6 : "musée virtuel"

[5] Rubic Cube : deux sources lumineuses ponctuelles, 268 polygones.

[6] Musée virtuel : deux sources lumineuses ponctuelles, 880 polygones.

Comme annoncé dans [GHAZ 92], les résultats qualitatifs sont très bons et tous les problèmes d'aliassage sont résolus. L'***Image 7*** montre ces résultats sur la scène du musée virtuel. On note que dans le cas d'un lancer de rayons avec subdivisions adaptatives d'au maximum 8x8 sous-pixels, la grille bleue est incomplète. Dans le cas du lancer de faisceaux, avec une même limite de subdivision de 8x8 sous-pixels, la grille est totalement formée.

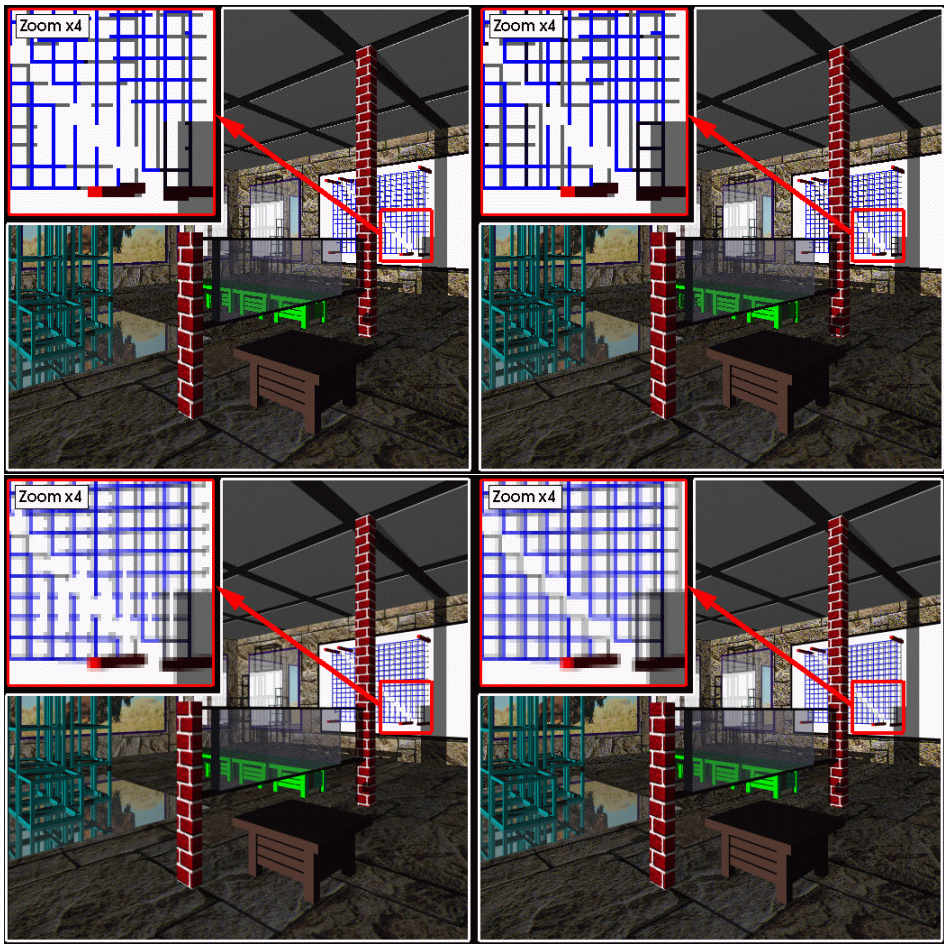


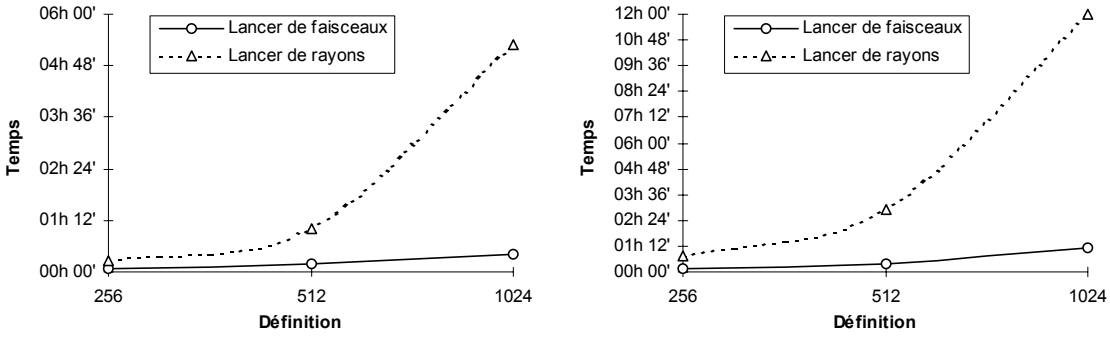
Image 7 : Comparaison qualitative : lancer de rayons sans subdivision (en haut à gauche), lancer de faisceaux sans subdivision (en haut à droite), lancer de rayons avec subdivision adaptative d'au maximum 8x8 sous-pixels (en bas à gauche), lancer de faisceaux avec au maximum 8x8 sous-pixels (en bas à droite).

Concernant l'aspect quantitatif, les résultats sont plus mitigés. Le gain de temps par rapport au lancer de rayons conventionnel optimisé dans les mêmes conditions, c'est-à-dire avec des volumes englobants, est important (***Figure 2.1***). Malheureusement ces temps[7] restent trop élevés : pour des définitions de 1024x1024, le "rubic cube" nécessite 23 minutes de calcul et le musée plus d'une heure.

Le lancer de faisceaux est donc plus rapide que le lancer de rayons mais le choix de l'optimisation ne semble pas être le bon puisque les temps de calculs sont relativement

[7] Les temps ont été mesurés sur une SGI Indy (µp R4010, 100Mhz, 32Mo) avec la fonction *clock()* de la librairie C. Le compilateur natif Silicon a été utilisé avec l'optimisation -O2.

importants. D'autre part, la construction de la hiérarchie de sphères englobantes est difficile et celle-ci n'est pas toujours appropriée à la scène. Prenons l'exemple du musée virtuel qui comporte plusieurs objets très allongés, (***Image 6***), les sphères englobantes construites autour de ces objets vont occuper beaucoup de place inutilement et probablement ralentir l'algorithme.



Rubic Cube. *Musée virtuel.*

Figure 2.1 : Comparaison en temps entre le lancer de faisceaux et le lancer de rayons conventionnel optimisé avec une hiérarchie de volumes englobants (sans subdivision de pixels).

Bien sûr, d'autres formes d'englobants sont possibles, par exemple les polyèdres de [GOLD 87]. Néanmoins le problème du choix de la hiérarchie reste posé. C'est pourquoi, nous verrons au chapitre suivant une amélioration fondée sur une subdivision spatiale régulière de l'espace.

2.2 UTILISATION DE SCENES "CSG"

Nous avons présenté au ***Paragraphe 1.3.2*** le lancer de faisceaux pyramidaux sur des scènes polygonales quelconques proposé par [GHAZ 92]. Cette approche, contrairement aux autres lancers de faisceaux, ne nécessite aucun calcul effectif d'intersection entre les objets et les faisceaux, il n'y a donc pas de problème de découpage 3D. Ceci permet l'utilisation de scènes non polygonales et en particulier des scènes modélisées par "CSG" [HASE 96]. Rappelons qu'une modélisation "CSG" utilise des *objets primitifs* et des opérations ensemblistes telles que l'*union*, la *différence* ou l'*intersection*. Généralement, les objets primitifs utilisés sont les polyèdres, les sphères, les cylindres et les cônes. L'application de ces opérations ensemblistes aux objets primitifs permet de créer des volumes complexes appelés *objets combinés*.

Nous présentons dans cette partie une extension du lancer de faisceaux pyramidaux sur des scènes "CSG". Cette extension est fondée sur les mêmes principes que ceux développés dans [GHAZ 92]. Pour bien comprendre les articulations de l'algorithme nous en proposons une vision modulaire à la ***Figure 2.2***. Pour inclure les scènes "CSG" à l'approche de [GHAZ 92] certaines modifications sont nécessaires, elles portent sur les trois modules suivants :

- le positionnement des objets par rapport aux faisceaux ;
- la construction des faisceaux et sous-faisceaux ;
- le test d'uniformité des régions.

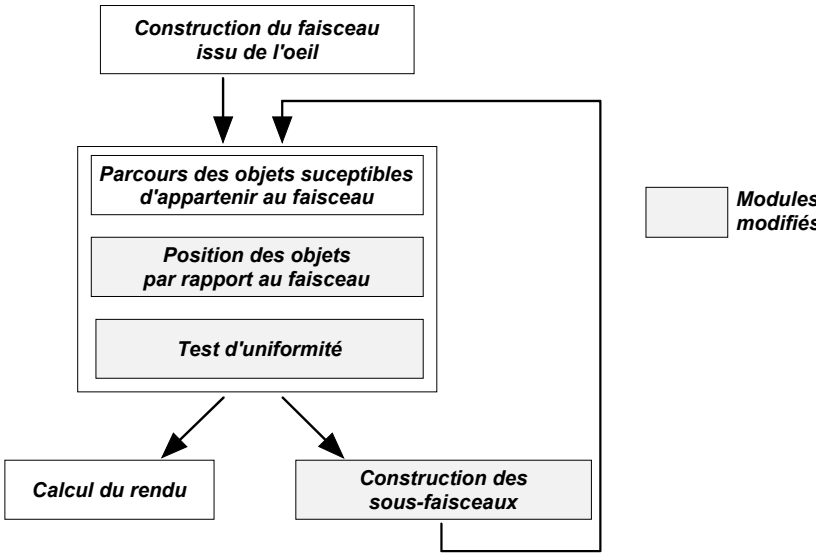


Figure 2.2 : Modules à modifier pour intégrer "CSG".

Le problème le plus important dans cette extension est le positionnement des objets par rapport aux faisceaux. En effet, tout l'algorithme s'appuie sur la possibilité de conclure rapidement qu'un objet est hors d'un faisceau ou bloque entièrement celui-ci. Nous devons donc disposer d'algorithmes simples permettant de situer tous les objets primitifs (polyèdres, sphères, cylindres et cônes) par rapport à un faisceau. Nous allons donc étudier ce problème en détail.

2.2.1 Position des objets primitifs par rapport aux faisceaux

La position d'un objet primitif par rapport à un faisceau peut être déterminée de manière simple. Pour cela, nous utilisons la règle, déjà utilisée dans [CHOI 92] (***Paragraphe 1.4.1.2***), suivante :

> ***Règle 1*** : *S'il existe un plan qui sépare l'espace en deux demi-espaces de sorte que l'un contienne la totalité du faisceau et l'autre la totalité de l'objet, alors l'objet est entièrement à l'extérieur faisceau.*

Pour tous les objets primitifs, il faut donc être capable d'effectuer les deux opérations suivantes :

1. positionner rapidement l'objet par rapport à un plan ;
2. déterminer judicieusement les plans à utiliser.

Le positionnement d'un polyèdre par rapport à un plan et la détermination des plans qu'il convient d'utiliser avec cet objet sont traités dans le chapitre suivant consacré, en particulier aux optimisations possibles. Il ne reste donc qu'à considérer les cas des sphères, cylindres et cônes. Notons que dans tous les cas, les premiers plans testés sont ceux formant le faisceau. Les plans supplémentaires sont spécifiques à chaque objet primitif. Les choix des plans est empirique, mais ils sont simple à construire et donne de bon résultats en pratique.

2.2.1.1 Objets sphériques

■ **Position par rapport à un plan**

Pour positionner une sphère de centre C et de rayon R par rapport à un plan, il suffit d'évaluer la distance entre C et le plan. Si cette distance est strictement supérieure à R, la sphère est entièrement d'un côté du plan. Dans tous les autres cas, on dira que le plan coupe la sphère.

■ **Choix des plans**

Des plans simples à définir et qui, dans la pratique, donnent de bons résultats, sont des plans tangents aux arêtes du faisceau et dont la normale pointe vers le centre C de la sphère (***Figure 2.3***). Pour construire un de ces plans, considérons une arête E et deux côtés consécutifs du faisceau dont les normales *n1* et *n2* pointent vers l'extérieur. Considérons A comme la projection orthogonale de C sur E. Le plan P est alors construit de sorte à être tangent à E et à avoir le vecteur AC pour normale. Il reste à voir si P vérifie la ***Règle 1***. Pour cela, il faut positionner la sphère mais aussi le faisceau par rapport à P. Pour positionner le faisceau, il suffit de vérifier que le plan P ne coupe pas le faisceau. Ceci peut être fait très simplement en comparant les signes des produits scalaires $n \bullet n1$ et $n \bullet n2$. Si les signes sont identiques, P ne coupe pas le faisceau sinon il coupe le faisceau et il faut considérer une nouvelle arête.

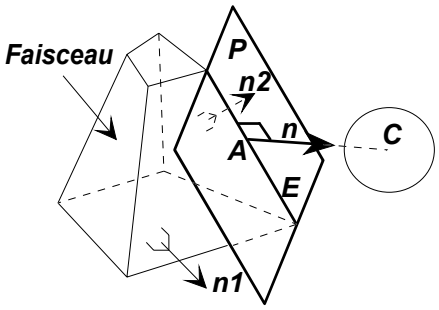


Figure 2.3 : Le plan P, tangent à l'arête E, divise l'espace en deux avec d'un côté le faisceau et de l'autre la sphère.

2.2.1.2 Objets cylindriques

■ **Position par rapport à un plan**

Positionner un cylindre par rapport à un plan *P* est légèrement plus compliqué. Il faut distinguer deux cas :

- Si l'axe de symétrie du cylindre est perpendiculaire au plan *P*, il faut considérer les centres des disques formant le cylindre. Si ces deux points sont du même côté de *P*, le cylindre est entièrement d'un côté de *P*, sinon *P* coupe le cylindre.
- Si l'axe de symétrie du cylindre n'est pas perpendiculaire au plan *P*, il faut considérer quatre points *qi, i∈[1,4]*, deux sur chaque contour des disques (***Figure 2.4***). Ces points sont les intersections entre les disques du cylindre et le plan perpendiculaire à *P* passant par les centres *C1* et *C2* des disques (***Figure 2.4***). Parmi ces quartes points se trouve le point le plus proche et le point le plus éloigné de *P*. Considérons *n1* comme la normale à une des bases *B* du cylindre et *n2* une normale à *P*. Le produit vectoriel *n3=n1×n2* correspond à la direction de l'intersection entre *P* et *B*. Le produit vectoriel *n4=n1×n3* est parallèle au plan contenant *B* et perpendiculaire à *P*. Pour déterminer les quatre points *qi*, il suffit de déplacer les centres des disques de *±R*n4* où *R* est le rayon du cylindre (***Figure 2.5***). La position du cylindre dépend maintenant de la position des points *qi* par rapport à *P*. Si les points *qi, i∈[1,4]*, sont du même côté de *P* alors le cylindre est entièrement d'un côté de *P*. Dans tous les autres cas, on dira que *P* coupe le cylindre.

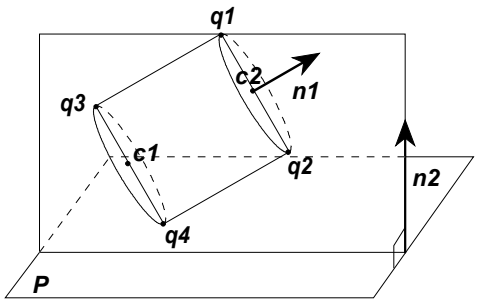


Figure 2.4 : Points les plus proche et les plus éloignés de P.

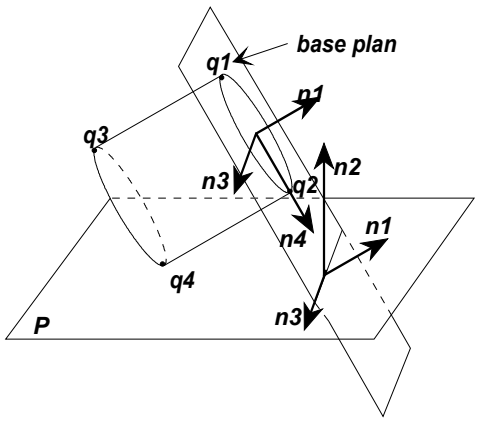


Figure 2.5 : Position d'un cylindre par rapport à un plan P.

■ **Choix des plans**

Là encore, des plans simples à définir et qui donnent de bons résultats dans la pratique sont tangents aux arêtes du faisceau (***Figure 2.6***). Soit *E* une arête du faisceau, *A* un point de *E*, *v* un vecteur directeur de *E*, *n1* et *n2* les normales des côtés adjacents à *E* et *d* la direction de l'axe de symétrie du cylindre. Deux cas sont à distinguer :

- si *d•n1* et *d•n2* sont de même signe, le plan à considérer contient *A*, *v* et le produit vectoriel *d×v* (***Figure 2.6.a***).
- si *d•n1* et *d•n2* ne sont pas de même signe, le plan à considérer contient *A*, *v* et *d* (***Figure 2.6.b***).

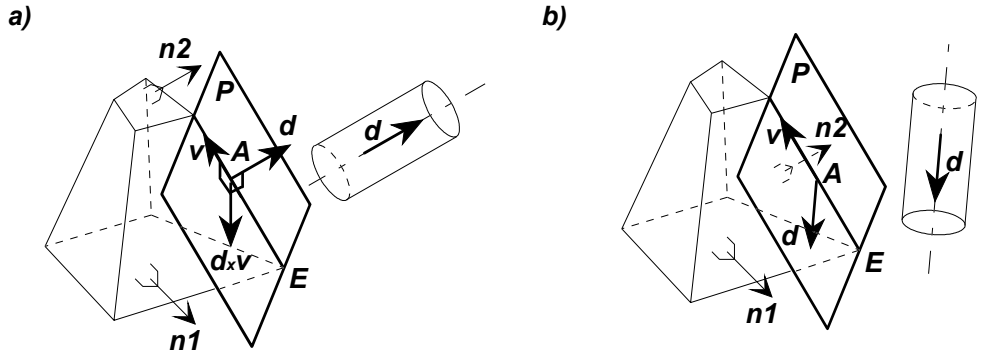


Figure 2.6 : Position d'un cylindre par rapport à un faisceau.

2.2.1.3 Objets coniques

■ **Position par rapport à un plan**

Déterminer la position d'un cône par rapport à un plan est traité de la même manière que pour un cylindre. On calcule le point *q1* et *q2* sur le contour de la base du cône exactement comme précédemment. On détermine ensuite la position des points *q1*, *q2* et du sommet *S* du cône par rapport au plan *P*. Si ces trois points sont du même côté de *P*, le cône est entièrement d'un côté de *P*, sinon on dira que *P* coupe le cône.

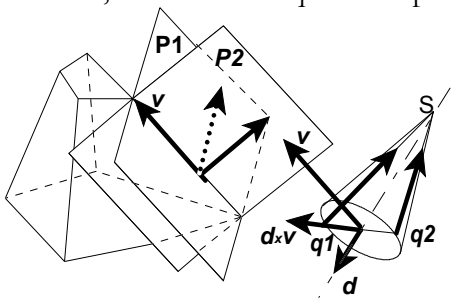


Figure 2.7 : Position d'un cône par rapport à un faisceau.

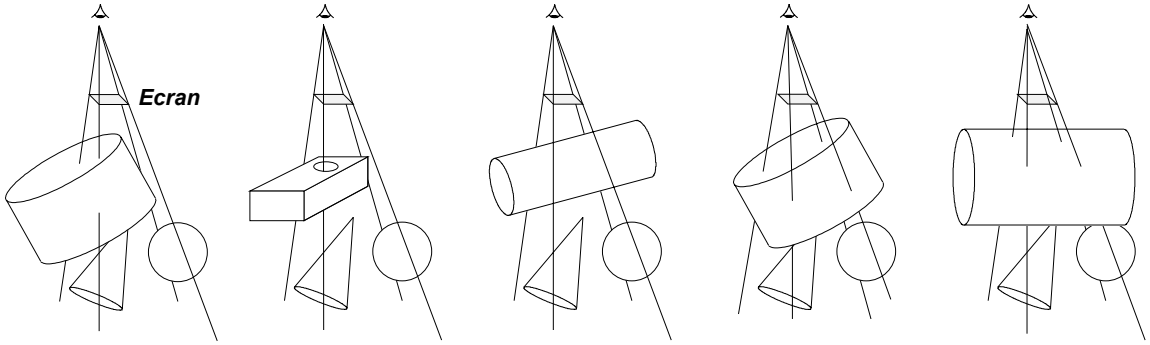
■ **Choix des plans**

Le choix des plans judicieux ressemble au cas du cylindre. On considère le point *q1* et *q2* sur la base du cône (***Figure 2.7***). Ces points sont obtenus en déplaçant le centre du disque de *±R*d×v*, où *R* est le rayon de la base du cône, *d* sa direction et *v* le vecteur directeur de l'arête considérée. Deux plans *P1* et *P2* sont susceptibles de valider la ***Règle 1***. *P1* contient *A*, *v* et le vecteur *q1S* et *P2* contient *A*, *v* et le vecteur *q2S*. Ces deux plans sont testés consécutivement.

2.2.2 Détection des régions uniformes

Le deuxième problème à résoudre concerne la détermination du type d'une région (uniforme ou ambiguë). Pour cela, nous construisons la liste des objets (sphères, cylindres, cônes et polyèdres) situés entièrement ou partiellement dans le faisceau. On regarde, pour chaque objet primitif susceptible d'être dans le faisceau, si un des plans judicieux, décrit au paragraphe précédent, vérifie la ***Règle 1***. Si c'est le cas, l'objet est clairement à l'extérieur du faisceau et l'objet n'est pas ajouté à la liste. Sinon, l'objet est supposé à l'intérieur du faisceau (entièrement ou partiellement) est ajouté à la liste. Notons que, comme dans l'approche de [GHAZ 92] on ne cherche pas la position exacte de tous les objets mais simplement à écarter rapidement ceux clairement hors des faisceaux.

Une fois cette liste construite pour un faisceau donné, différents cas sont considérés ***Figure 2.8*** :



a. *Seulement deux arêtes traversent un objet.*

b. *Aucune intersection arête-objet composite.*

c. *Aucune intersection arête-objet*

d. *Quatre intersections, mais avec des faces différentes.*

e. *Quatre intersections, avec la même face*

Figure 2.8 : Exemples de régions uniformes (e) et de régions non uniformes (a,b,c,d).

- Si la liste des objets associés au faisceau est vide, la région est uniforme.
- Si la liste n'est pas vide, quatre rayons correspondant aux quatre arêtes du faisceau sont lancés à travers la scène "CSG". On considère alors, pour chaque rayon, l'intersection la plus proche, quand elle existe.
  - Si ces quatre intersections ne sont pas sur la même face du même objet primitif (***Figure 2.8.a*** *et* ***d***), la région est non uniforme.
  - Dans les autres cas, les mêmes quatre rayons sont lancés dans la liste des objets primitifs associés au faisceau (nous ne considérons ici aucune opération "CSG"). Si aucun rayon ne traverse un objet (***Figure 2.8.c***), il y a un objet dans le faisceau (la liste est supposée non vide). Si les quatre rayons ne traversent pas tous les objets de la liste associée au faisceau et les mêmes faces des objets, toutes dans le même ordre, alors la région est considérée non uniforme (***Figure 2.8.b***). Dans tous les autres cas, la région est considérée uniforme (***Figure 2.8.e***).

Bien sûr, certaines subdivisions sont en surnombre avec cette méthode, mais elle a le mérite d'être simple. De plus, même avec des subdivisions inutiles, elle assure la détection de toutes les régions à problème. L'***Image 8*** montre les régions détectées uniformes pour trois objets primitifs.

Notons que la subdivision est la même quelles que soient les opérations ensembliste utilisées (union, différence ou intersection). Ce n'est que lors du calcul du rendu avec un lancer de rayons que l'on tient compte de ces opérations.

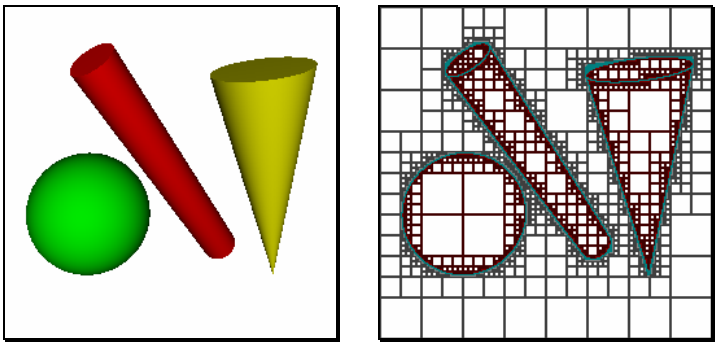


Image 8 : A droite : trois objets primitifs.
A gauche : les régions détectées uniformes.

2.2.3 Constructions des faisceaux et sous-faisceaux

Le dernier problème concerne la construction des faisceaux et sous-faisceaux. En effet, dans [GHAZ 92] les faisceaux étaient délimités par un ensemble de plans. En particulier, un faisceau primaire correspondant à une région uniforme avait une base plane (***Figure 2.9.a***). Dans le cas de scènes "CSG", la base des faisceaux peut être courbe (***Figure 2.9.b***), soit concave soit convexe.

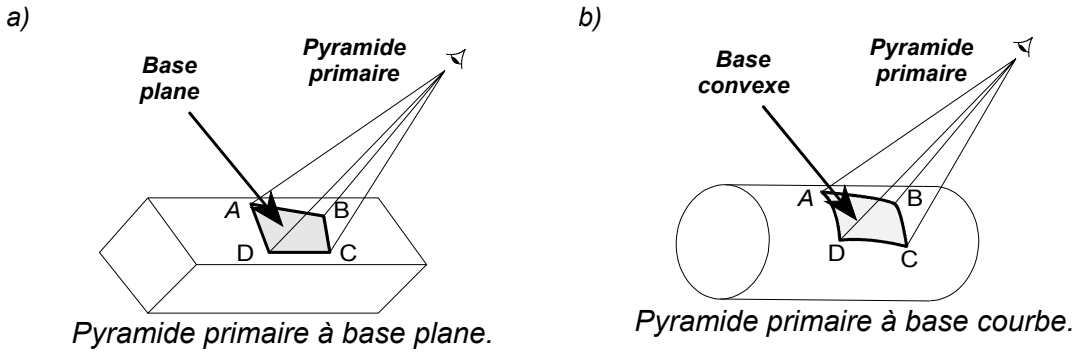


Figure 2.9 : Différences entre les pyramides primaires dans les cas de scènes polygonales et dans le cas de scènes "CSG".

2.2.3.1 Faisceaux primaires

Pour détecter si un objet se trouve dans le faisceau, nous utilisons la même approche que [GHAZ 92]. Mais pour cela, il faut se ramener à un faisceau délimité par des plans. On approche la base courbe du faisceau primaire par un plan *P*. La construction de ce plan dépend de l'opération ensembliste utilisée et donc de la concavité de la surface considérée.

- Pour une surface convexe, trois des quatre points d'intersection sont considérés (*A*, *C* et *D* dans la ***Figure 2.10***) pour définir *P*. Si le quatrième point (*B* dans la ***Figure 2.10***) et le sommet du faisceau ne sont pas du même côté de *P*, on considère trois autres points pour construire *P*. Un tel plan existe toujours et les quatre points d'intersection sont du même côté que le sommet.
- Pour une surface concave, le plan *P* est tangent au point *E* (***Figure 2.11***). Le point *E* correspond à l'intersection entre *r* et la surface concave. *r* est un rayon lancé au centre du faisceau avec pour direction la somme des vecteurs *OA*, *OB*, *OC* et *OD*.

Quelle que soit la concavité de la surface rencontrée, il est donc possible de construire un faisceau délimité par des plans. Ces nouveaux faisceaux nous permettent de positionner facilement les objets primitifs et de déterminer le type de la région considérée.

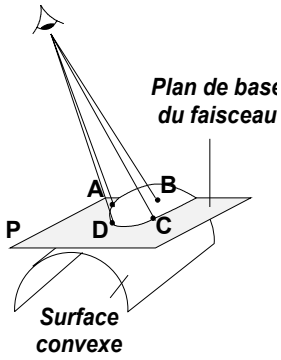


Figure 2.10 : Base d'un faisceau sur une surface convexe.

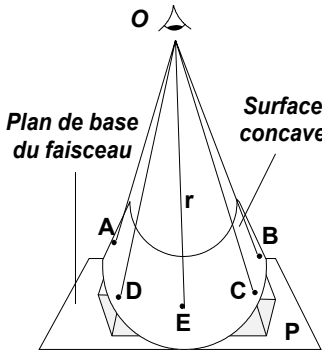


Figure 2.11 : Base d'un faisceau sur une surface concave.

Notons que ces faisceaux sont un peu plus grands que les faisceaux exacts mais que l'on n'oublie aucun objet. En fait, le volume en trop n'est jamais très important puisque plus le rayon de courbure de la surface est grand, plus la base réelle du faisceau se confond avec le plan approché et plus le volume inutile est petit. De même, plus la courbure est faible, plus la surface de la base est petite et plus le volume supplémentaire est petit.

2.2.3.2 Faisceaux d'ombre

Les faisceaux d'ombre sont semblables à ceux utilisés dans le cas de scènes polygonales (***Paragraphe 1.3.2.3***). Ils ont pour sommet la source lumineuse ponctuelle et la même base que le faisceau primaire (***Figure 2.12***) c'est-à-dire le plan ayant servi d'approximation.

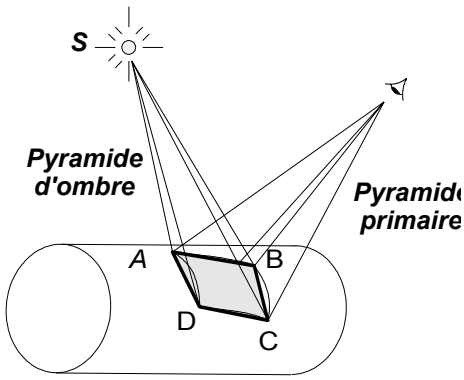


Figure 2.12 : Faisceau d'ombre.

2.2.3.3 Faisceaux de réflexion et de réfraction

La réalisation d'effets de réflexion et de réfraction avec des scènes "CSG" est plus complexe qu'avec des scènes polygonales. En effet, les faisceaux réfléchis sur des surfaces planes sont facilement constructibles, par contre, lorsqu'il s'agit de surfaces gauches, nous sommes obligés de les caractériser différemment. Pour construire des faisceaux de réflexion, nous cherchons un volume qui englobe tous les rayons réfléchis possibles par une surface gauche. C'est en fait une extension de la solution proposée pour les faisceaux réfractés dans le cas de scènes polygonales (***Paragraphe 1.3.2.5***).

Soient *A*, *B*, *C* et *D* les intersections des arêtes de la pyramide primaire avec l'objet réfléchissant et *r1*, *r2*, *r3* et *r4* les rayons réfléchis en *A*, *B*, *C* et *D* (***Figure 2.13***). La surface dépend des opérations "CSG" utilisées et donc de la concavité de la surface.

- Si la surface est convexe., chaque segment *AB*, *BC*, *CD* et *DA* peut être associé à un rayon réfléchi pour former un plan. Dans la ***Figure 2.13***, le segment *BC* et le

rayon *r2* forment un plan *P1* et le segment *BC* et le rayon *r3* forment un plan *P2*. Un seul de ces plans est "entre" *ABCD* et l'autre plan (*P1* dans notre exemple). Le plan le plus extérieur, ici *P2*, et les trois autres plans construits de la même manière sur les trois autres segments forment alors le faisceau réfléchi.

- Si la surface est concave (***Figure 2.14***), on peut considérer quatre plans pour chaque segment *AB*, *BC*, *CD* et *DE* (dans la ***Figure 2.14*** on considère les quatre plans construit sur l'arête *BC*). Chacun de ces plans est orienté par l'un des rayons réfléchis *r1*, *r2*, *r3* or *r4*. On obtient ainsi seize plans. Seuls les plans les plus extérieurs sont conservés pour former le faisceau réfléchi.

Dans les deux cas, les faisceaux forment des pseudo-pyramides car elles n'ont pas forcément un sommet en commun. D'autre part, ces pseudo-pyramides peuvent être dégénérer et ne plus définir un volume intérieur et un volume extérieur. Dans ce cas, il suffit de subdiviser le faisceau en quatre sous-faisceaux jusqu'à obtenir des faisceaux cohérents. La base des faisceaux de réflexion est déterminée de la même manière que dans le cas de faisceaux primaires (***Paragraphe 2.2.3.1***). Ces pseudo-pyramides réfléchies contiennent ainsi tous les rayons réfléchis possibles. On utilise les tests précédents pour déterminer les régions uniformes.

Notons qu'il est possible que trop d'objets soient considérés, mais cela n'entraîne pas d'erreur, au plus des subdivisions inutiles. On notera néanmoins qu'un faisceau primaire réfléchi peut atteindre une taille très grande au fur et à mesure des réflexions. Dans ce cas, presque tous les objets de la scène peuvent être considérés pour déterminer si la région est uniforme.

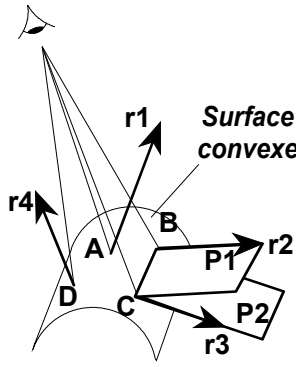


Figure 2.13 : Réflexion sur une surface convexe.

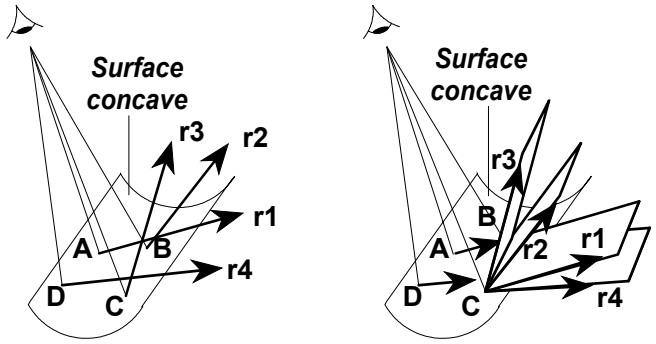


Figure 2.14 : Réflexion sur une surface concave

Le cas de la réfraction est tout à fait similaire à celui de la réflexion. Le but est toujours de construire un volume englobant tous les rayons réfractés possibles. Pour cela, une pseudo-pyramide tronquée de réfraction est construite comme précédemment.

2.2.4 Résultats

Nous avons proposé une extension du lancer de faisceaux de [GHAZ 92] pour l'utilisation de scènes "CSG". La construction des différents faisceaux tient compte de la concavité des surfaces produites par les opérations "CSG". En pratique, nous avons développé uniquement les faisceaux primaires et d'ombre. Les résultats sur une scène utilisant tous les types de concavités sont présentés ***Image 9***. La qualité obtenue est celle que l'on attendait, c'est-à-dire sans aucun problème d'aliassage, par contre, les temps de calculs sont relativement importants (plus d'une heure[8]) pour nos deux scènes tests.

[8] Les temps ont été mesurés sur une SGI Indy (µp R4010, 100Mhz, 32Mo) avec la fonction *clock()* de la librairie C. Le compilateur natif Silicon a été utilisé avec l'optimisation -O2.

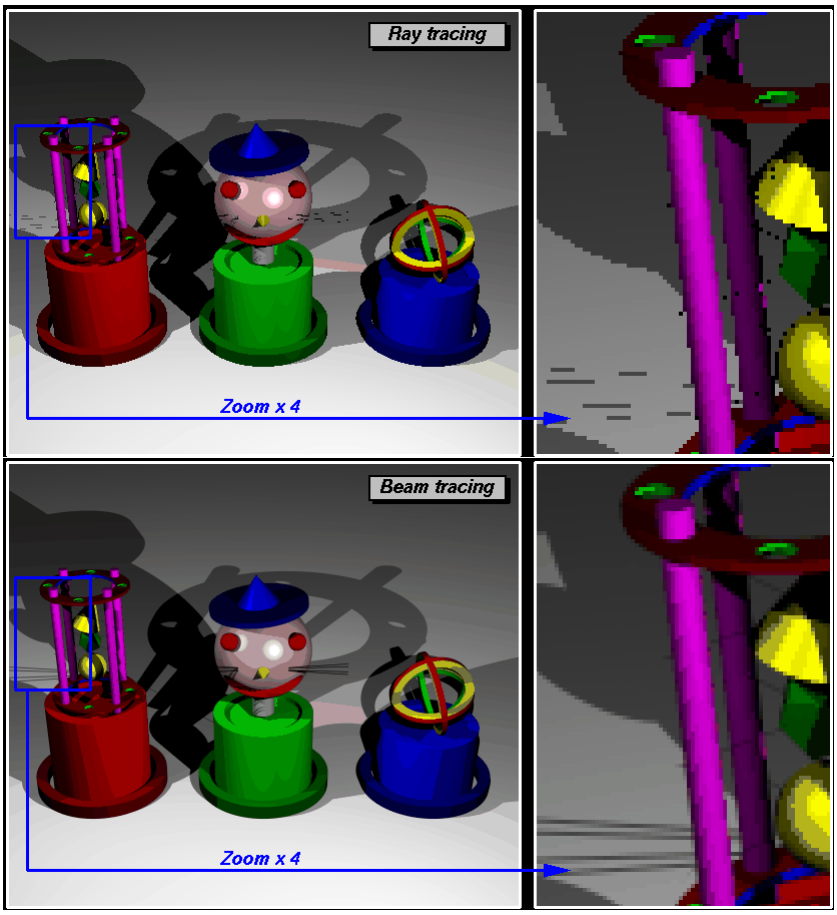


Image 9 : Haut : Lancer de rayons sans antialiassage sur une scène “CSG”.
Bas : Lancer de faisceaux sur une scène “CSG” avec au plus 8x8 sous-faisceaux par pixel.

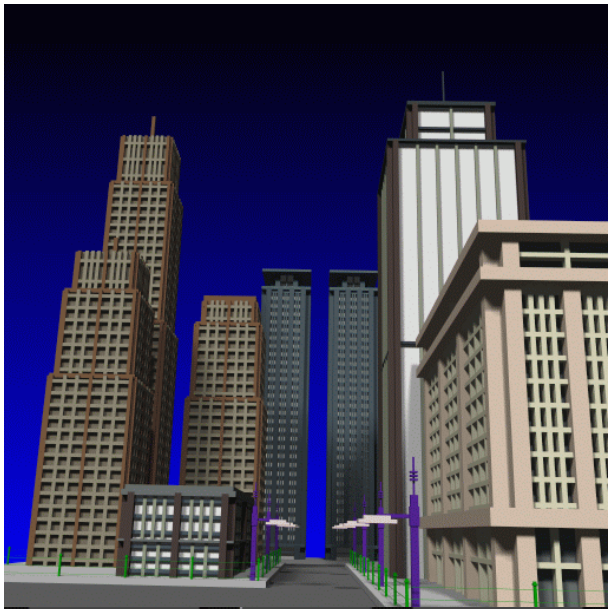
2.3 CONCLUSION

Nous avons présenté dans ce chapitre les résultats de l'implémentation de l'algorithme de lancer de faisceaux proposé par [GHAZ 92]. Les comparaisons en temps avec un lancer de rayons conventionnel optimisé avec une hiérarchie de sphères englobantes sont en faveur du lancer de faisceaux. Néanmoins, les temps de calculs sont importants malgré l'optimisation avec des sphères englobantes. Nous verrons dans le chapitre suivant qu'une autre approche permet de diminuer très largement ces temps.

En ce qui concerne les problèmes d'aliassage, nous avons pu vérifier qu'ils étaient tous résolus efficacement. En particulier, il n'y a plus de disparition de petits objets ou de petites ombres.

Comme aucune intersection explicite faisceau-objet n'est nécessaire dans l'algorithme de [GHAZ 92], nous avons pu traiter des scènes de type "CSG" sans modifier grandement l'algorithme de base [HASE 96]. Le plus gros problème était de positionner rapidement les objets primitifs par rapport aux différents faisceaux. Pour cela, des méthodes efficaces ont été exposées pour chaque type d'objet. Concernant les effets de réflexion et de réfraction, nous avons montré qu'il n'y a pas de problèmes théoriques et qu'une implémentation est possible.

3. OPTIMISATIONS ET EXTENSION DU LANCER DE FAISCEAUX PYRAMIDAUX



Comme nous avons pu le constater au chapitre précédent, le lancer de faisceaux pyramidaux résout tous les phénomènes d'aliassage. Malheureusement, les performances en temps de calculs de cet algorithme ne sont pas très bonnes. Nous avons donc cherché l'origine de cette lenteur et nous avons développé plusieurs optimisations [GHAZ 98]. La première partie de ce chapitre traite de ces améliorations.
*D'autre part, l'utilisation de rayons volumiques induit naturellement la notion de pénombre. [AMAN 84] utilise des cônes pour produire ces effets de pénombre, mais la description de son approche est très vague (**Paragraphe 1.2.1**). Nous proposons donc, dans la deuxième partie de ce chapitre, une extension qui permet de produire des effets de pénombre réalistes. Nous comparons les résultats obtenus, aussi bien qualitatifs que quantitatifs, avec ceux obtenus par un lancer de rayons stochastique.*

3.1 OPTIMISATIONS

Nous avons pu constater au chapitre précédent que le lancer de faisceaux pyramidaux fournit des images sans aliassage mais que le temps nécessaire pour les calculer était relativement important. Nous proposons donc, dans ce paragraphe, d'améliorer ces temps. Pour cela, nous fondons, une nouvelle fois, nos optimisations sur l'analyse modulaire de l'algorithme de [GHAZ 92]. Ces modules (***Figure 3.1***) permettent de mettre en évidence les différents problèmes à résoudre et surtout de préciser les points que nous allons reprendre afin de les accélérer. Ainsi, nous nous pencherons plus particulièrement sur cinq problèmes :

- le calcul du rendu ;
- le parcours des objets susceptibles d'appartenir à un faisceau ;
- la manière de trouver leur position par rapport à ce faisceau ;
- les moyens pour détecter les régions uniformes ;
- la construction des sous-faisceaux.

Nous terminerons cette étude par les résultats détaillés des performances.

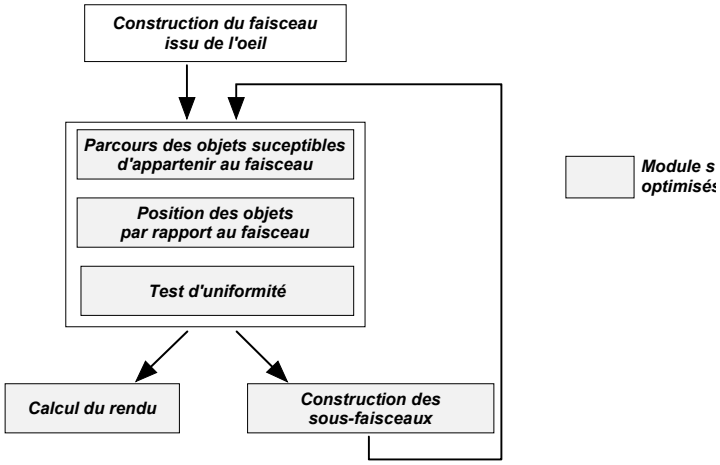


Figure 3.1 : Description modulaire de l'algorithme de [GHAZ 92].

3.1.1 Calcul du rendu

Suite aux essais peu concluants des volumes englobants (***Paragraphe 2.1***), nous nous sommes tournés vers les subdivisions spatiales de type grille régulière [FUJI 86] [AMAN 87] [CLEA 88] … et grille adaptative (“*octree*”) [GLAS 84] [SAME 89] [SPAC 91] … Dans le cas du lancer de rayons, aucune littérature ne donne clairement “l'avantage” à l'une ou à l'autre des subdivisions, par contre [ENDL 94] les compare et conclut : « *Dire de façon générale que l'utilisation de subdivisions spatiales adaptatives (“octree”) fournit de meilleurs résultats que l'utilisation de subdivisions spatiales régulières est impossible. Cela dépend toujours de la scène choisie.* ». La principale différence est située au niveau de l'occupation mémoire. En effet, dans la plupart des cas, la grille régulière est nettement plus gourmande. Dans le cas de notre lancer de faisceaux nous avons tout de même utilisé une subdivision régulière parce qu'elle est simple à mettre en oeuvre et parce qu'elle permet une bonne optimisation pour une complexité très acceptable. On peut tout de même signaler que [CLEA 88] écrivait : « *On admet de plus en plus que les subdivisions spatiales régulières sont plus rapides que les subdivisions spatiales adaptatives (“octree”)* », mais c'était en 1988 et depuis les

auteurs sont plus nuancés. D'autre part, l'occupation mémoire n'est pas exagérée et tourne autour de quelques Mega-octets (voir ***Paragraphe 3.1.5.2***, ***Figure 3.21***). En effet, si l'on considère qu'une station de travail actuelle dispose au minimum de 32 Mo et que la subdivision spatiale régulière occupe 8 Mo, alors ¼ de la mémoire est nécessaire à cette subdivision et il reste 24 Mo pour le programme et la description de la scène. Cette marge de manoeuvre reste donc tout à fait acceptable.

D'autres subdivisions spatiales, telles que les subdivisions spatiales binaires ("*BSP*") [FUCH 80] [KAPL 87] ou les macro-régions [DEVI 89] existent. Dans les cas des "BSP", l'espace est divisé récursivement par un seul plan en deux sous-espaces de dimension quelconque. Le plan est choisi judicieusement pour diviser les sous-espaces de manière équitable sans traverser aucun objet. Cette subdivision est relativement complexe et apporte des propriétés qui ne sont pas nécessaires pour nos travaux. Dans le cas des macro-régions, on construit des volumes vides, les plus grands possibles, en forme de parallélépipède. Là encore, ces régions n'apportent, pour nous, que peu d'intérêt.

Rappels sur la subdivision spatiale régulière

Pour comprendre nos travaux sur les faisceaux, rappelons brièvement quelques notions. La *subdivision spatiale régulière* [FUJI 86] [AMAN 87] [CLEA 88] de l'espace contenant la scène est une technique d'optimisation du lancer de rayons très répandue. L'espace est borné à la taille de la scène et est divisé en volumes élémentaires appelés *voxels*. Ces voxels sont tous de même taille, ne se recouvrent pas, ont la forme d'un parallélépipède rectangle et ont des arêtes parallèles aux trois axes principaux. A chaque voxel est associée la liste des objets qu'il contient. Si un objet s'étend sur plusieurs voxels, il appartient à plusieurs listes. Lorsqu'un rayon est lancé, le voxel contenant son origine est considéré. Si le rayon traverse au moins un des objets de la liste associée à ce voxel, ses intersections sont triées et l'intersection la plus proche de l'origine est retenue. Si cette intersection se trouve dans le voxel, on a trouvé l'objet traversé par le rayon le plus proche de l'origine du rayon. Si aucune intersection n'a été trouvée ou si la liste est vide, on suit le rayon et on effectue les mêmes tests sur le voxel suivant. Ce mécanisme est répété jusqu'à ce que l'on trouve une intersection et tant que l'on reste dans l'espace subdivisé. Le passage d'un voxel à l'autre est fondé sur un algorithme simple travaillant uniquement avec des entiers. Différentes solutions existent [AMAN 87] [CLEA 88], nous avons gardé [AMAN 87] pour sa rapidité et sa simplicité.

Un inconvénient bien connu de ce type de subdivision est le fait qu'un objet puisse appartenir à plusieurs voxels à la fois et que par conséquent on calcule pour différents voxels la même intersection rayon-objet. En effet, dans notre exemple (***Figure 3.2***), le polygone appartient aux voxels *a*, *b*, *c*, *d*, *e*, *f*, *g* et *h* et les calculs d'intersection rayon-objet sont effectués dans l'ordre pour les voxels *a*, *d*, *e* et enfin *g*. Il n'y a pas d'intersection calculée pour le voxel *h* puisqu'un point d'intersection a déjà été trouvé dans le voxel *g* précédent.

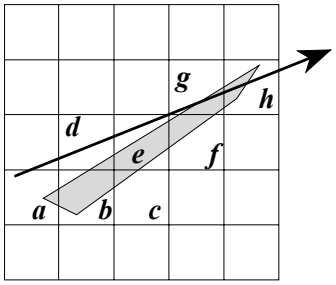


Figure 3.2 : Rayon traversant la subdivision spatiale régulière.

Pour éviter ces calculs inutiles, chaque rayon possède un identificateur unique et à chaque objet est associé l'identificateur du dernier rayon qui a servi à tester une intersection avec cet objet. On mémorise aussi le point d'intersection quand il existe. Avant chaque calcul d'intersection, il ne reste donc plus qu'à comparer l'identificateur du rayon courant avec celui associé à l'objet pour savoir si un calcul d'intersection a déjà été effectué. Notons, dès à présent, que cette astuce sera utilisée plusieurs fois avec nos faisceaux, comme nous le verrons dans les chapitres suivants.

3.1.2 Parcours des objets

[GHAZ 92] propose d'associer à chaque faisceau la liste des objets susceptibles d'être à l'intérieur de ce faisceau (***Paragraphe 1.3.2.2***). Pour chaque subdivision, quatre sous-faisceaux sont construits ainsi que quatre sous-listes. Pour le premier faisceau, on associe la liste contenant tous les objets de la scène. Cette approche nécessite une utilisation massive de listes (dynamiques) de plusieurs milliers d'objets. Cette manipulation intensive pénalise cet algorithme. C'est pourquoi nous proposons une solution dans laquelle cette liste est supprimée.

Le problème est donc de savoir quels sont les objets susceptibles de se trouver entièrement ou partiellement dans un faisceau. [CHOI 92] propose une solution dans le cadre de son optimisation du calcul des ombres pour un lancer de rayons (***Paragraphe 1.5***). Malheureusement, sa solution est relativement complexe. En effet, elle fait intervenir à la fois une subdivision spatiale régulière de l'espace et une subdivision adaptative de type octree. De plus, sa solution est très dépendante de la taille du faisceau et du nombre et de la taille des objets qu'il contient. Elle est relativement lente dans certains cas. Nous avons donc préféré utiliser uniquement une subdivision spatiale régulière dans laquelle on associe à chaque voxel la liste des polyèdres qu'il contient (entièrement ou partiellement).

L'idée de notre algorithme est de trouver l'ensemble des voxels situés dans le faisceau puis de positionner les objets contenus dans les listes associées aux voxels par rapport au faisceau. Pour résoudre ce problème, nous avons comparé plusieurs méthodes fondées sur des calculs en entiers ou/et sur une discrétisation de l'espace. Finalement, nous avons opté pour une méthode utilisant un balayage en "tranches" et des calculs fondés sur des entiers. Ainsi, le parcours des voxels du faisceau s'effectue par "tranches" perpendiculaires à l'une des trois directions principales *OX*, *OY* ou *OZ* (le choix du plan et de la direction de balayage est discuté par la suite). Chaque tranche est délimitée par un parallélépipède rectangle contenant tous les voxels traversés par les arêtes du faisceau (***Figure 3.3***). Ces arêtes, *aA*, *bB*, *cC*, *dD*, *ab*, *bc*, *cd*, *da*, *AB*, *BC*, *CD* et *DA*, sont définies par une extension en trois dimensions de l'algorithme de Bresenham décrit dans [AMAN 87].

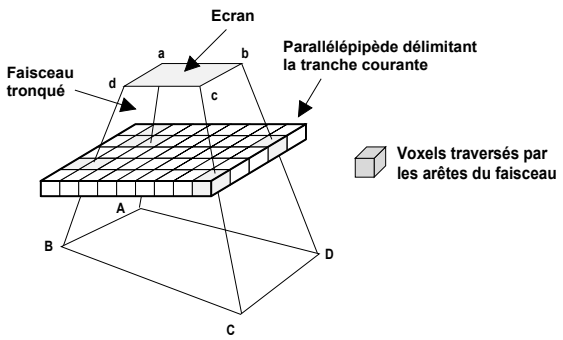


Figure 3.3 : Tranche délimitée par un parallélépipède rectangle

Notons que le faisceau doit être délimité de chaque côté par un plan. Or, il arrive qu'un faisceau se propage à l'infini en quittant la boîte englobante de la scène. Dans ce cas,

on "ferme" le faisceau avec un plan de base pour obtenir un volume fini. Pour cela, nous choisissons un plan virtuel qui représente le fond de la scène. Ce plan virtuel est parallèle à l'écran et passe par le coin de la subdivision spatiale régulière le plus éloigné du sommet du faisceau (***Figure 3.4***).

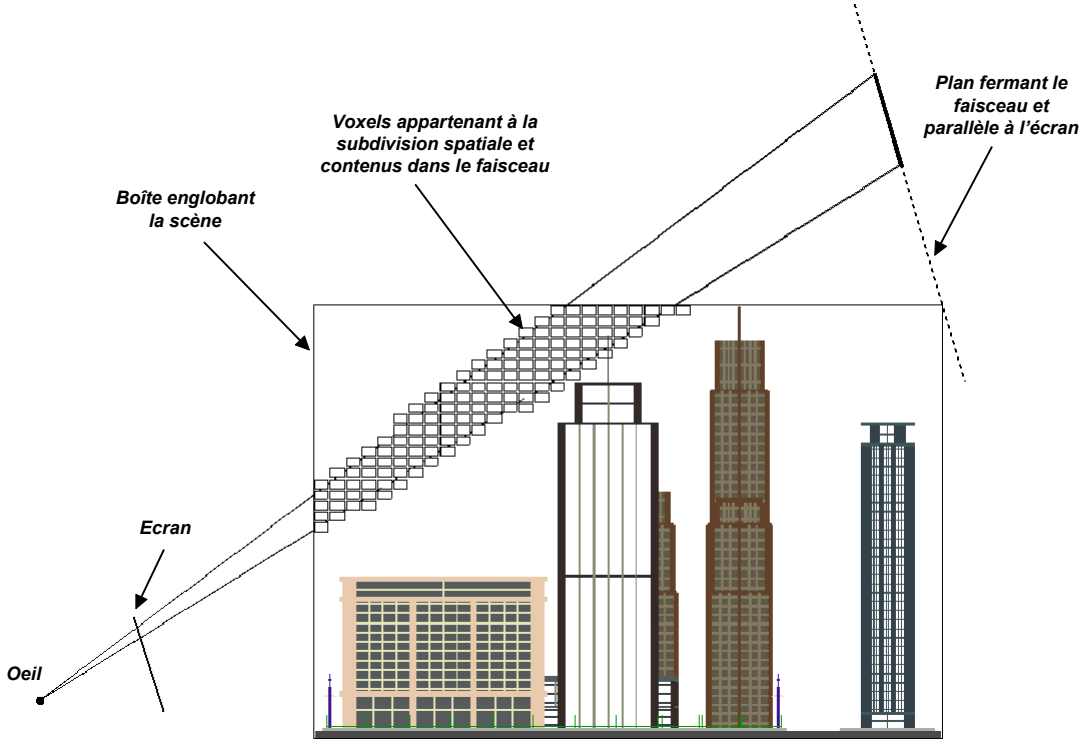


Figure 3.4 : Représentation d'un faisceau et de la subdivision spatiale régulière

Deux autres approches pour fermer le faisceau viennent à l'esprit. La première consiste à découper le faisceau selon les faces de la boîte englobante de la scène (***Figure 3.5***). Cette solution nécessite des calculs complexes pour prendre en considération tous les cas possibles de position de l'oeil par rapport à la scène. La seconde consiste à fermer le faisceau par un plan parallèle à l'une des faces de la boîte englobant la scène (***Figure 3.6.a***). Malheureusement cette solution n'est pas toujours possible comme le montre la ***Figure 3.6.b***.

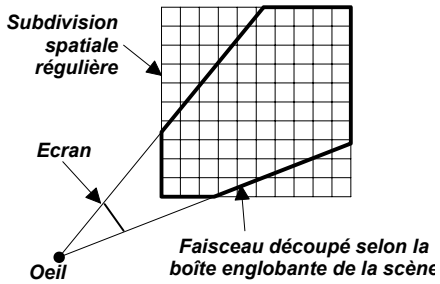
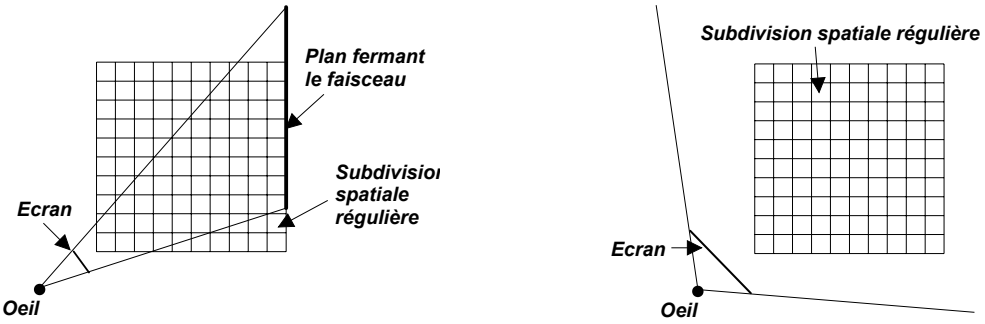


Figure 3.5 : Découpage du faisceau selon la boîte englobante de la scène.



a) Une des faces de la boîte englobante convient. *b) Aucune des faces ne convient.*

Figure 3.6 : Utilisation d'une face de la boîte englobante pour fermer le faisceau (cela n'est pas toujours possible).

3.1.2.1 Choix du plan de balayage

Rappelons que l'on cherche à savoir s'il y a des objets dans le faisceau. En fait, il suffit de trouver un objet à l'intérieur du faisceau pour pouvoir conclure que la région considérée n'est pas uniforme et qu'une subdivision est nécessaire. On ne cherche donc pas à parcourir les voxels dans un ordre précis. Une technique comme celle de [NEHL 95], qui permet de balayer les voxels (considérés comme un espace discret) selon un plan parallèle à l'écran et selon la direction de propagation du faisceau, n'est donc pas utile. C'est pourquoi, nous avons préféré une technique plus simple en considérant des plans parallèles aux axes du repère.

3.1.2.2 Choix de la direction de balayage

La ***Figure 3.7*** montre deux tranches d'un même faisceau selon deux directions différentes de balayage. Dans le cas *a),* la direction de balayage est proche de celle de propagation du faisceau et dans le cas *b)*, la direction est perpendiculaire à la direction de propagation du faisceau. On constate que le cas *a)* englobe moins de voxels se trouvant à l'extérieur du faisceau que le cas *b).*

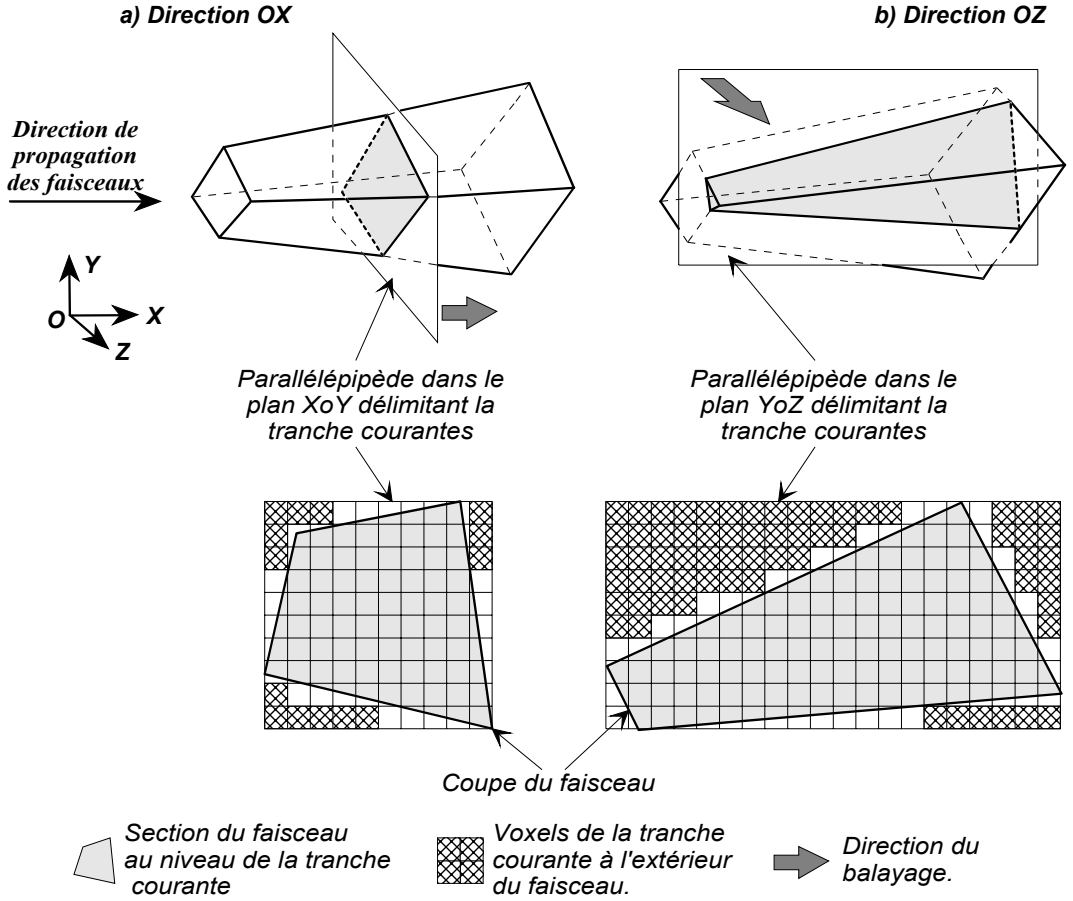
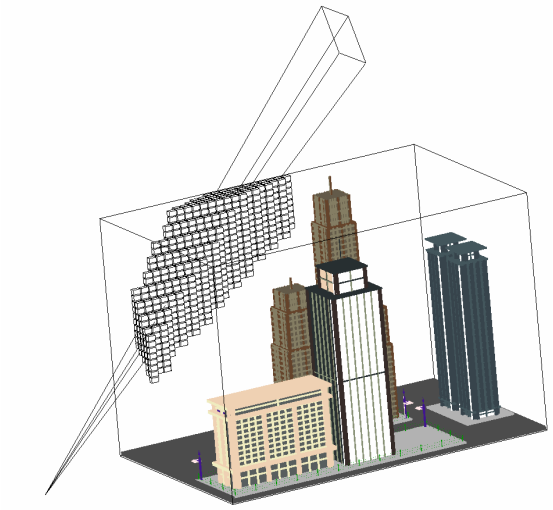


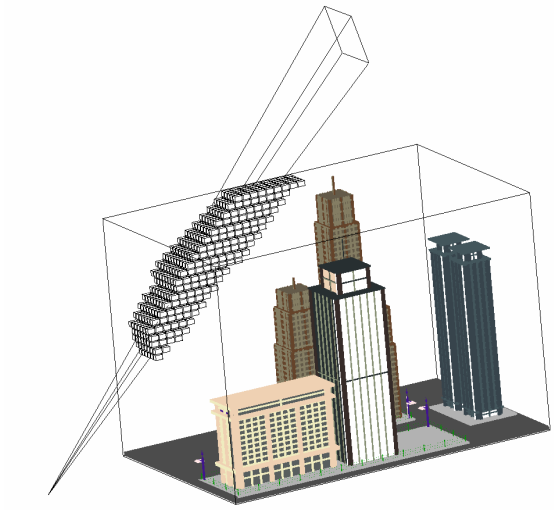
Figure 3.7 : Deux tranches de balayage correspondant à deux directions (OZ et OX) .

Le choix de la direction de balayage est donc très important et une mauvaise direction peut entraîner le parcours de nombreux voxels situés à l'extérieur du faisceau. A titre d'exemple, considérons un faisceau plongé dans une subdivision régulière de 40x40x40 (***Figure 3.8***). Un balayage selon la direction OX entraîne la visite de 1222 voxels, selon OY 987 voxels et selon OZ seulement 785 voxels, soit un gain de 64% entre OX et OZ.

a)



b)



c)

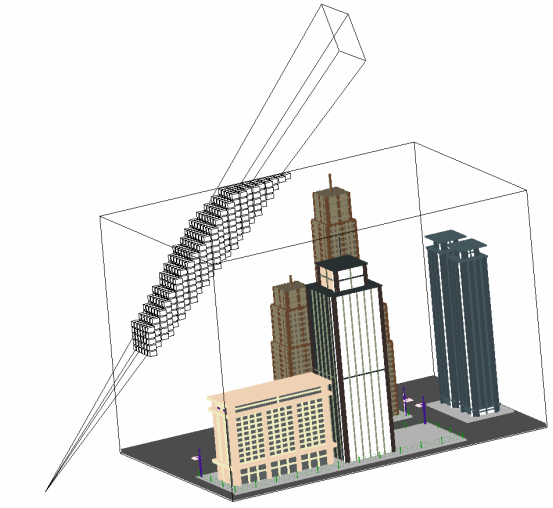


Figure 3.8 : Balayage des plans suivant les différents axes. Les voxels visités sont bornés à l'intérieur de la boîte englobante. La résolution de la grille régulière est de 40x40x40.

a) : Balayage suivant la direction OX : 1222 voxels
b) : Balayage suivant la direction OY : 987 voxels
c) : Balayage suivant la direction OZ : 785 voxels

Pour choisir la bonne direction, nous définissons le *vecteur directeur du faisceau* comme étant la somme des vecteurs directeurs des arêtes latérales du faisceau. La direction de balayage est alors la composante la plus grande du vecteur directeur du faisceau. Cette composante correspond à la plus petite surface de projection selon l'une des trois directions des tranches balayées.

3.1.3 Position des objets par rapport aux faisceaux

Le troisième module optimisé concerne la position des objets par rapport aux faisceaux. [GHAZ 92] propose un lancer de faisceaux sur des scènes décrites par des polygones de concavité quelconque. Cette généralité, souvent inutile soit par la construction même de la scène soit parce qu'un prétraitement simple peut nous ramener dans le cas de polygones convexes, alourdit et ralentit l'algorithme. Ceci est particulièrement le cas lors du positionnement d'un objet par rapport à un faisceau. On peut noter par ailleurs que la plupart des scènes polygonales est constituée de facettes adjacentes par leurs sommets ou leurs arêtes, or l'algorithme proposé dans [GHAZ 92] n'en profite absolument pas. Dans notre solution nous tenons compte de cette caractéristique des objets, en particulier lors de leur positionnement par rapport aux faisceaux. Nous proposons donc une version fondée sur des scènes formées de polyèdres convexes et tenant compte de la

notion d'adjacence des faces. Ceci permet d'améliorer notablement les performances de l'algorithme sans limiter de manière excessive son champ d'application. Notons que l'approche globale reste la même que dans [GHAZ 92] et reste compatible avec des scènes polygonales quelconques.

3.1.3.1 Position d'un polyèdre par rapport à un plan

Avant de pouvoir positionner un polyèdre par rapport à un plan, nous devons être capables de situer un point par rapport à un plan. Pour cela, un simple produit scalaire suffit en théorie. Pratiquement, nous sommes obligés d'introduire une épaisseur à notre plan pour contrecarrer les erreurs de calcul de l'arithmétique flottante. Nous introduisons donc arbitrairement une tolérance de ±1e-09.

L'***Algorithme 3-1*** présente la fonction permettant de positionner un point *pt* par rapport un plan défini par un point "*planPoint*" et sa normale "*planNormal*".

```
int Plane::SideOfPoint(const Point3D& pt) const
    const    SCALEPS = 1e-09;
    double   scal = planNormal * (pt - planPoint);

    si scal > SCALEPS
        retourner PT_NORMALSIDE;
    sinon si scal < -SCALEPS
        retourner PT_OPPOSITENORMALSIDE;
    sinon
        retourner PT_INPLANE;
```

Algorithme 3-1 : Position d'un point par rapport à un plan.

Positionner un polyèdre par rapport à un plan revient à positionner l'ensemble de ses sommets par rapport à ce même plan. Si tous les sommets sont du côté (resp. du côté opposé à) de la normale "*planNormal*" du plan, alors le polyèdre est du même côté (***Figure 3.9.a***). Si les sommets sont des deux côtés du plan, alors le polyèdre est coupé par le plan (***Figure 3.9.b***).

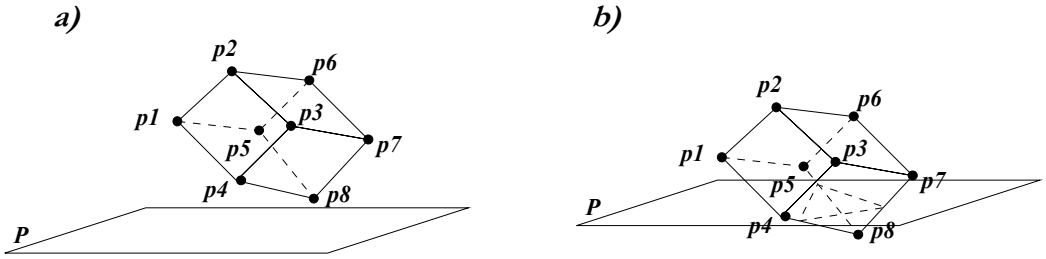


Figure 3.9 : : Position d'un polyèdre par rapport à un plan.

L'algorithme utilisé pour positionner un polyèdre par rapport à un plan *P* ne tient pas compte des sommets contenus dans le plan. Si le polyèdre (convexe) est tangent au plan, il est considéré comme entièrement d'un même côté du plan. Ce choix simplifie l'algorithme et ne porte pas à conséquence. Le principe général de l'algorithme est de déterminer la position d'un sommet par rapport à *P* et de regarder la position des autres sommets. Dès que l'on trouve un sommet qui n'est pas du même côté, on conclut que le plan P coupe le polyèdre. Si tous les sommets sont d'un même côté, il n'y a pas d'intersection plan-polyèdre.

L'***Algorithme 3-2*** présente la fonction qui positionne un polyèdre par rapport à un plan *P*. Un polyèdre est défini par "*nbVertex*" sommets stockés dans un tableau "*vertexList*" de "*Point3D*" et pour chaque face, par les indices de ces sommets.

```
// Valeur de retour :
//  SP_CUTPLANE : le plan coupe l'objet.
//  SP_NORMALSIDE        : l'objet est ENTIEREMENT du côté de la normale.
//  SP_OPPOSITENORMALSIDE : l'objet est ENTIEREMENT du côté opposé de la normale.
//  SP_BAD : cas impossible.

int Polyhedron::PlaneSide(const Plane& P)
    int       isTangent = 0;
    int       side, curSide;
    int       i;

    // On cherche le premier point d'un côté du plan.
    pour(i=0;i<nbVertex && (side=P.sideOfPoint(vertexList[i]))==PT_INPLANE;i++);

    si i==vertexN                          // On n'a pas trouvé un tel point.
        cerr << "Curieux, il existe un polyèdre dégénéré (il est plan)\n";
        retourner SP_BAD;
    sinon si i!=0                          // Il y a au moins un point dans le plan.
        isTangent = 1;
    i++;

    pour( ; i< nbVertex; i++)
        curSide = P.sideOfPoint(vertexList[i]);
        si curSide==PT_INPLANE             // On saute les points dans le plan.
            isTangent = 1;
        sinon si curSide!=side             // Le nouveau point n'est pas du bon côté.
            retourner SP_CUTPLANE;

    si side == PT_NORMALSIDE
        retourner SP_NORMALSIDE;
    sinon si side == PT_OPPOSITENORMALSIDE)
        retourner SP_OPPOSITENORMALSIDE;
```

Algorithme 3-2 : Position d'un polyèdre par rapport à un plan.

En regardant la fonction *Polyhedron::PlaneSide()* de l'***Algorithme 3-2*** il est clair que positionner un objet convexe, défini en tant que polyèdre, est moins coûteux que s'il était défini par un ensemble de polygones indépendants. Ceci est principalement dû au fait qu'une modélisation fondée sur des polyèdres évite la redondance des sommets communs à une même face. Un exemple très simple montre l'intérêt d'une telle approche. Considérons un cube et une modélisation sous forme de polygones indépendants c'est-à-dire six faces chacune constituée par quatre sommets. Pour dire que ce cube est d'un côté d'un plan, il faut donc positionner 4*6=24 sommets par rapport à ce même plan. Dans le cas d'une modélisation polyèdrale, seuls les huits sommets du cube sont positionnés. On diminue ainsi de 2/3 le nombre de points à positionner pour cet exemple très simple.

3.1.3.2 Position d'un polyèdre par rapport à un faisceau

Rappelons que pour déterminer le type, uniforme ou ambiguë, d'une région (***Paragraphe 1.3.2.2***), il est nécessaire de pouvoir situer un objet par rapport à un faisceau. Pour cela, nous utilisons la même règle que celle utilisée dans le cas des scènes "CSG" :

> ***Règle 1*** *: S'il existe un plan qui sépare l'espace en deux demi-espaces de sorte que l'un contienne la totalité du faisceau et l'autre la totalité de l'objet, alors l'objet est entièrement à l'extérieur du faisceau.*

Si un tel plan peut être trouvé pour un faisceau et un objet donnés, nous concluons que l'objet est à l'extérieur du faisceau. Si un tel plan n'est pas trouvé (il peut néanmoins exister), nous concluons que l'objet peut être partiellement dans le faisceau. Notons que dire qu'un objet n'appartenant pas au faisceau est dans le faisceau ne fausse pas l'algorithme de lancer de faisceaux, au pire on effectue quelques subdivisions de trop. Par contre, chercher à tout prix la position réelle d'un objet par rapport à un faisceau peut être plus coûteux que de supposer qu'il est situé dans le faisceau alors qu'il ne l'est pas.

Pour appliquer la ***Règle 1.*** il existe plusieurs plans judicieux :

- le plan de l'écran (***Figure 3.10.a***),
- le plan contenant la base $ABCD$ du faisceau, (***Figure 3.10.b***),
- les quatre pans du faisceau (***Figure 3.10.c***),
- tous les plans tangents à une arête du faisceau et à $\vec{v} = \vec{E} \times \vec{e}$ où $\vec{E}$ est un vecteur parallèle à l'une des directions des arêtes du polyèdre et où $\vec{e}$ un vecteur directeur de l'arête (***Figure 3.10.d***).
- tous les plans tangents à une arête du "sommet" du faisceau et à $\vec{v} = \vec{E} \times \vec{e}$ où $\vec{E}$ est un vecteur parallèle à l'une des directions des arêtes du polyèdre et où $\vec{e}$ un vecteur directeur de l'arête (***Figure 3.10.e***).
- tous les plans tangents à une arête de la "base" du faisceau et à $\vec{v} = \vec{E} \times \vec{e}$ où $\vec{E}$ est un vecteur parallèle à l'une des directions des arêtes du polyèdre et où $\vec{e}$ un vecteur directeur de l'arête (***Figure 3.10.f***).
- tous les plans parallèles aux faces du polyèdre (***Figure 3.10.g***).

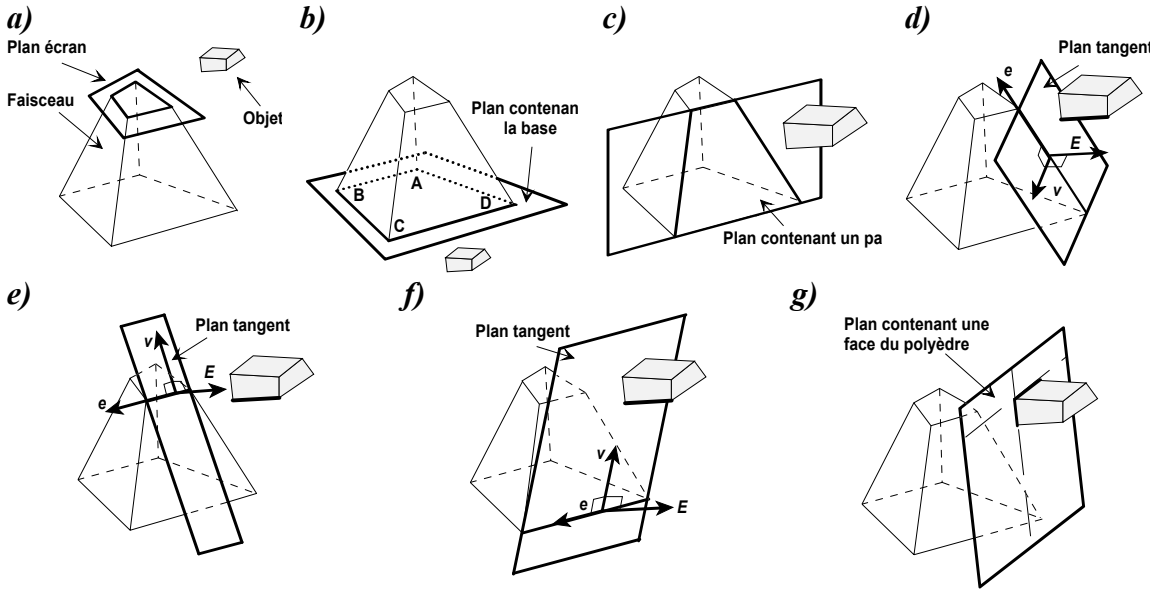


Figure 3.10 : Plans séparateurs judicieux.

Si l'un de ces plans vérifie la ***Règle 1***, l'objet est à l'extérieur du faisceau et on considère l'objet suivant sinon on essaye un autre plan. Tout le problème est de savoir s'il faut tester tous ces plans, ou si un sous-ensemble suffit et dans quel ordre les traiter. Pour

résoudre ce problème, nous avons compté le nombre de fois où ces plans vérifient la ***Règle 1*** lors de l'application de l'algorithme sur deux scènes tests : une ville virtuelle[9] (***Image 10***) et un ensemble de cubes[10] (***Image 11***). Il ressort du ***Tableau 3*** que pour ces deux scènes (sans réflexion ni réfraction) tous les plans ne sont pas nécessaires et que leurs efficacités sont très variées. Nous proposons de ne conserver que les quatre plans les plus efficaces à savoir "*Plan_Arête*", "*Plan_Polyèdre*", "*Plan_Pan*" et "*Plan_Base*" du ***Tableau 3***. Notons que les plans peuvent être redondants et qu'un objet peut être écarté par différents plans. Ceci ne fait qu'ajouter à la complexité du choix !

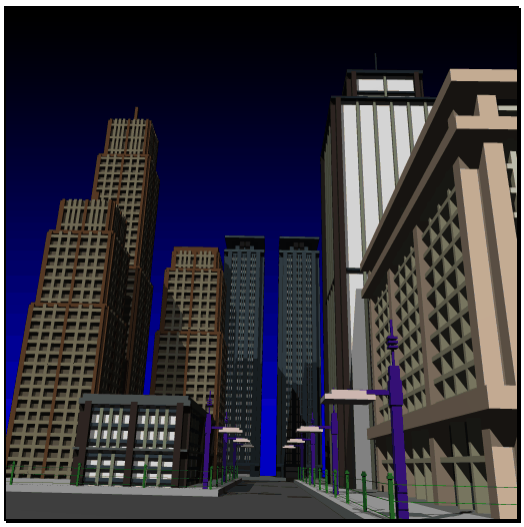


Image 10 : Ville virtuelle.

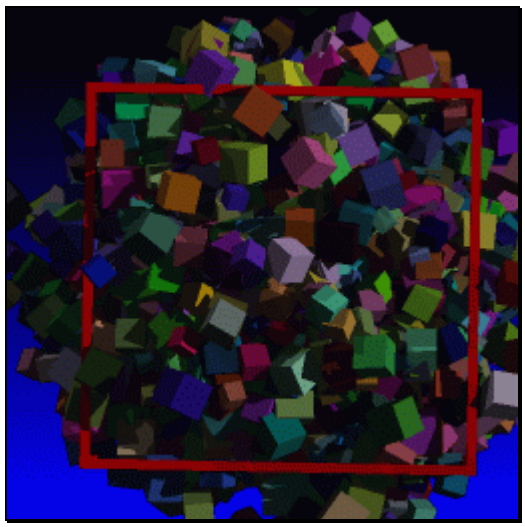


Image 11 : Ensemble de cubes aléatoires.

Plans testés	***Nom donné à l'ensemble***	***Ville 1 105 894 évaluations de la Règle 1***	***Cubes 1 834 642 évaluations de la Règle 1***
Plans tangents à une arête du faisceau	*Plan_Arête*	444 231	786 428
Plans parallèles aux faces du polyèdre	*Plan_Polyèdre*	405 102	549 042
Plans contenants un pan du faisceau	*Plan_Pan*	300 004	561 100
Plan contenant la "base" du faisceau	*Plan_Base*	228 419	155 450
Plans tangents à une arête de la "base" du faisceau	*Plan_TgBase*	130 644	517 898
Plans tangents à une arête du "sommet" du faisceau	*Plan_TgSommet*	4740	505
Plan contenant l'écran	*Film*	0	0

*Tableau 3 : Nombre de fois où un plan valide la **Règle 1**.*

Pour déterminer si l'ordre dans lequel on teste les plans a une importance, nous proposons là encore une étude statistique sur les deux scènes tests. Pour cela, on a mesuré le temps[11] d'exécution du lancer de faisceaux pour toutes les combinaisons possibles d'ordre de traitement des quatre plans retenus. Le ***Tableau 4*** des résultats montre que l'ordre de traitement n'a pas une grande importance. En effet, la différence de temps est

[9] Scène de la ville virtuelle : 1519 polyèdres (9114 polygones).

[10] Scène composée des cubes de tailles et de positions aléatoires : 2060 polyèdres (12360 polygones).

[11] Les temps ont été mesurés en secondes sur un Alpha Server 2000 4/275 (1 µp, 275Mhz, 128Mo) avec le "*profiler*" d'UNIX *gprof*. Le compilateur utilisé est g++ 2.7.1 avec les options -O4 -gp. La définition est de 1024x1024 et la subdivision spatiale est de 70x70x70 pour la ville et de 50x50x50 pour les cubes.

inférieure à 5% pour la ville virtuelle et inférieure à 7% pour la répartition aléatoire des cubes.

1er plan testé	2ème plan testé	3ème plan testé	4ème plan testé	Temps [sec]	
				Ville	Cubes
Plan_Polyèdre	*Pan*	*Base*	*Arête*	361	279
Plan_Polyèdre	*Pan*	*Arête*	*Base*	360	268
Plan_Polyèdre	*Base*	*Pan*	*Arête*	368	**266**
Plan_Polyèdre	*Base*	*Arête*	*Pan*	362	268
Plan_Polyèdre	*Arête*	*Base*	*Pan*	355	269
Plan_Polyèdre	*Arête*	*Pan*	*Base*	**369**	269
Plan_Pan	*Polyèdre*	*Base*	*Arête*	357	270
Plan_Pan	*Polyèdre*	*Arête*	*Base*	357	271
Plan_Pan	*Base*	*Polyèdre*	*Arête*	356	275
Plan_Pan	*Base*	*Arête*	*Polyèdre*	354	276
Plan_Pan	*Arête*	*Base*	*Polyèdre*	367	274
Plan_Pan	*Arête*	*Polyèdre*	*Base*	359	279
Plan_Arête	*Polyèdre*	*Pan*	*Base*	365	282
Plan_Arête	*Polyèdre*	*Base*	*Pan*	357	279
Plan_Arête	*Pan*	*Polyèdre*	*Base*	357	286
Plan_Arête	*Pan*	*Base*	*Polyèdre*	357	**288**
Plan_Arête	*Base*	*Pan*	*Polyèdre*	357	283
Plan_Arête	*Base*	*Polyèdre*	*Pan*	354	281
Plan_Base	*Polyèdre*	*Pan*	*Arête*	361	274
Plan_Base	*Polyèdre*	*Arête*	*Pan*	354	275
Plan_Base	*Pan*	*Polyèdre*	*Arête*	**351**	270
Plan_Base	*Pan*	*Arête*	*Polyèdre*	352	273
Plan_Base	*Arête*	*Pan*	*Polyèdre*	353	281
Plan_Base	*Arête*	*Polyèdre*	*Pan*	353	275

Tableau 4 : Temps nécessaire au lancer de faisceaux pour différents ordres dans lequel on teste les plans (en grisé les temps extrêmes).

3.1.4 Détection des régions uniformes

La quatrième optimisation concerne la détection des régions uniformes. Rappelons que l'algorithme de lancer de faisceaux cherche les régions uniformes de la scène, c'est-à-dire, les régions dans lesquelles un seul objet est visible. Pour cela, on lance des faisceaux et on cherche la position des objets qu'ils sont susceptible de contenir (***Paragraphe 1.3.2.2***). Le fait d'utiliser des polyèdres à la place de polygones, nous permet une amélioration des coûts de calcul en simplifiant la détermination des types des régions (uniformes ou non).

Pour définir le type uniforme ou non d'une région, nous associons à chaque arête du faisceau le premier objet qu'il traverse (quand il existe). Ce calcul d'intersection est bien sûr optimisé en utilisant la subdivision spatiale régulière (***Paragraphe 3.1.1***). Nous pouvons alors considérer quatre cas de figures :

1. Il y a au moins une intersection mais toutes les arêtes ne traversent pas le même objet (***Figure 3.11.a***) ;
2. Toutes les arêtes traversent le même objet mais pas la même face de l'objet (***Figure 3.11.b***) ;

3. Toutes les arêtes traversent le même objet et la même face de l'objet (***Figure 3.11.c*** ou ***d***) ;
4. Il n'y a pas d'intersection arête-objet (***Figure 3.11.e*** ou ***f***).

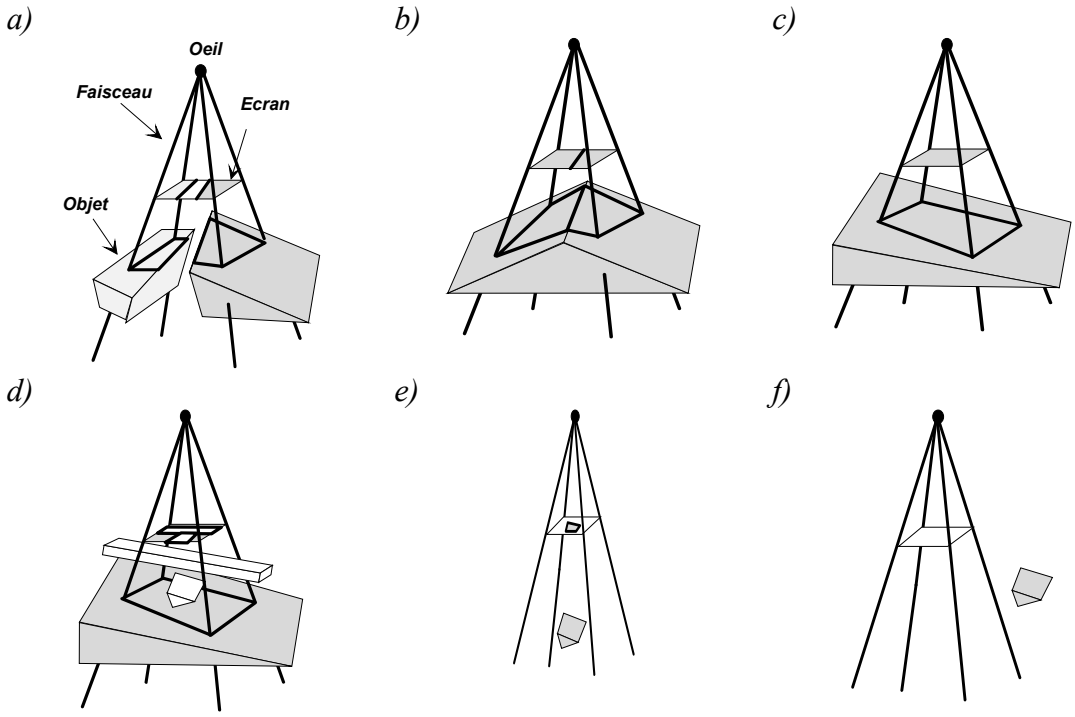


Figure 3.11 : Exemples d'intersections arêtes-objets

Cas 1 et 2 :

Dans les deux premiers cas, il y a au moins une arête de polyèdre visible dans le faisceau (***Figure 3.11.a*** ou ***b***). On conclut qu'il y a certainement un problème d'aliassage et que la région est clairement non uniforme. Des subdivisions sont nécessaires.

Cas 3 :

Ce cas peut être divisé en deux sous-cas :

- il n'y a pas d'arête de polyèdre visible à l'intérieur du faisceau (***Figure 3.11.c***), la région est uniforme et aucune subdivision n'est nécessaire ;
- il y a au moins une arête de polygone visible dans le faisceau (***Figure 3.11.d***), la région est non uniforme et des subdivisions supplémentaires sont nécessaires.

Pour différencier ces deux cas il faut trouver la position des objets de la scène par rapport à un faisceau. Pour cela, on utilise l'algorithme de balayage des voxels proposé au ***Paragraphe 3.1.2***. On peut ainsi trouver s'il existe un objet à l'intérieur du faisceau. Si tous les objets considérés sont à l'extérieur du faisceau, il n'y a pas de problème d'aliassage (***Figure 3.11.c***), la région est uniforme et aucune subdivision n'est nécessaire. Dans tous les autres cas, le faisceau peut contenir un objet (entièrement ou partiellement), la région est considérée non uniforme et des subdivisions supplémentaires sont nécessaires (***Figure 3.11.d***).

Cas 4 :

Ce cas (***Figure 3.11.e*** ou ***f***) est très similaire au cas précédent (***Figure 3.11.c*** ou ***d***). En effet, il suffit de "fermer" le faisceau (voir ***Paragraphe 3.1.2***, ***Figure 3.4***) pour obtenir un volume fini et se ramener au ***cas 3***.

3.1.4.1 Construction des sous-faisceaux

Rappelons que pour déterminer si une région est uniforme, on lance quatre rayons le long des quatre arêtes du faisceau. Si ceux-ci rencontrent une même face d'un objet (***Figure 3.11.c***) ou aucun objet (***Figure 3.11.e*** ou ***f***), on cherche à savoir s'il existe des polyèdres à l'intérieur du faisceau. Pour cela, on ferme le faisceau si cela est nécessaire et on balaye, par tranches, les voxels dans le faisceau. Si on rencontre un objet au cours de ce parcours, la région ne peut être uniforme et on subdivise le faisceau en quatre sous-faisceaux. Lors de la construction de ces sous-faisceaux, deux optimisations sont mises en place.

La première concerne le calcul des objets rencontrés par les arêtes. Notons que quatre sous-faisceaux partagent cinq arêtes. Pour gagner du temps, on calcule en même temps les sous-faisceaux. Considérons la région *ABCD* de la ***Figure 3.12.a*** et supposons qu'il faille la subdiviser. Pour cela, on lance les cinq rayons passant par *a*, *b*, *c*, *d* et *e* et on construit les quatre sous-faisceaux passant par les régions *Aaed*, *Bbea*, *Cceb* et *Ddec* en même temps. On a ainsi évité de lancer plusieurs fois les rayons communs. Malheureusement, ce mécanisme est difficilement généralisable et l'on est souvent obligé de répéter des calculs d'intersection. Considérons la ***Figure 3.12.b***, supposons maintenant que les quatre sous-régions sont ambiguës et qu'il faille donc les subdiviser. On procède comme précédemment en lançant pour chaque sous-région, cinq rayons. On constate alors que plusieurs rayons ont été lancés deux fois ; c'est le cas des rayons passant par *b1* et *d2*, *c2* et *a3*, *d3* et *b4*, *a4* et *c1*. Considérons à présent une image d'une définition de 1024x1024 pixels et supposons que les subdivisions récursives des faisceaux nous amènent systématiquement au niveau d'un pixel. On lance alors 1 747 629 rayons[12] aux coins des faisceaux alors que seulement 1024x1024=1 048 576 sont nécessaires avec un lancer de rayons. On lance donc au moins 2/3 de rayons en plus avec un lancer de faisceaux. Pour éviter cela, il est nécessaire de stocker tous les calculs intermédiaires lors de la récursion. Cette solution est complexe et coûteuse ; elle n'a donc pas été réalisée.

Ces calculs supplémentaires peuvent être gênants pour certaines scènes. C'est par exemple le cas de scènes contenant beaucoup de détails et nécessitant donc de très nombreuses subdivisions. Mais dans ce cas, un lancer de rayons conventionnel a du mal à produire une image sans problème d'aliassage.

[12] $4 + 1^2*5 + 2^2*5 + 4^2*5 + 8^2*5 + 16^2*5 + 32^2*5 + 64^2*5 + 128^2*5 + 256^2*5 + 512^2*5 = 1\ 747\ 629$

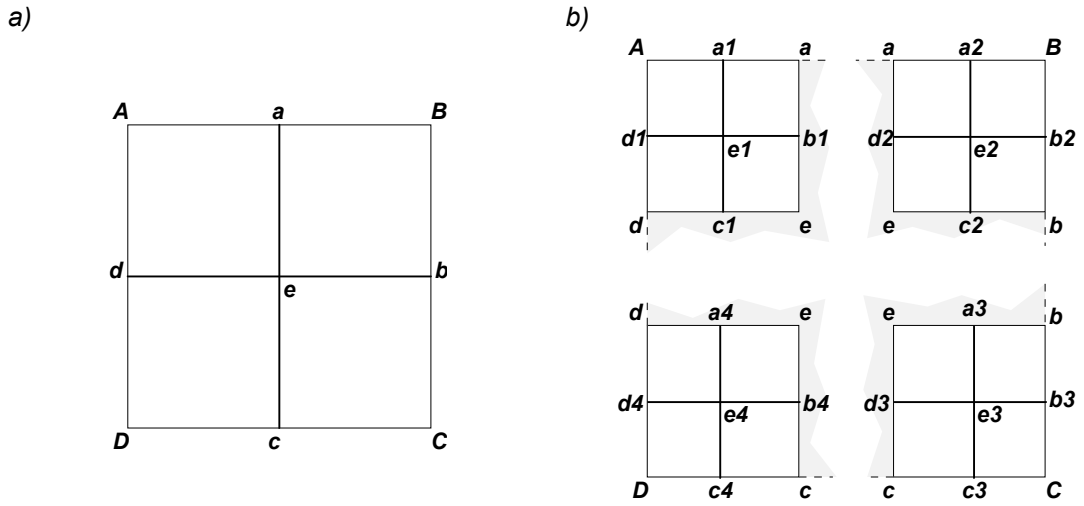


Figure 3.12 : Subdivision récursive des faisceaux.

La deuxième optimisation concerne le balayage par tranches des faisceaux et des sous-faisceaux. Considérons le faisceau de la ***Figure 3.13.a***, la direction X de balayage des tranches et la direction Y, une des directions de parcours dans une tranche. Cette figure montre le parcours des voxels jusqu'à celui contenant un objet. La présence, à la 9ième tranche, de cet objet entraîne la subdivision du faisceau (***Figure 3.13.b***). L'optimisation consiste alors à stocker avec les sous-faisceaux, la position du dernier voxel vide visité. De cette manière, si le parcours d'un des sous-faisceaux est nécessaire, il pourra débuter à cette position. En effet, avec un tel mécanisme, nous sommes sûrs que tous les voxels non visités sont vides.

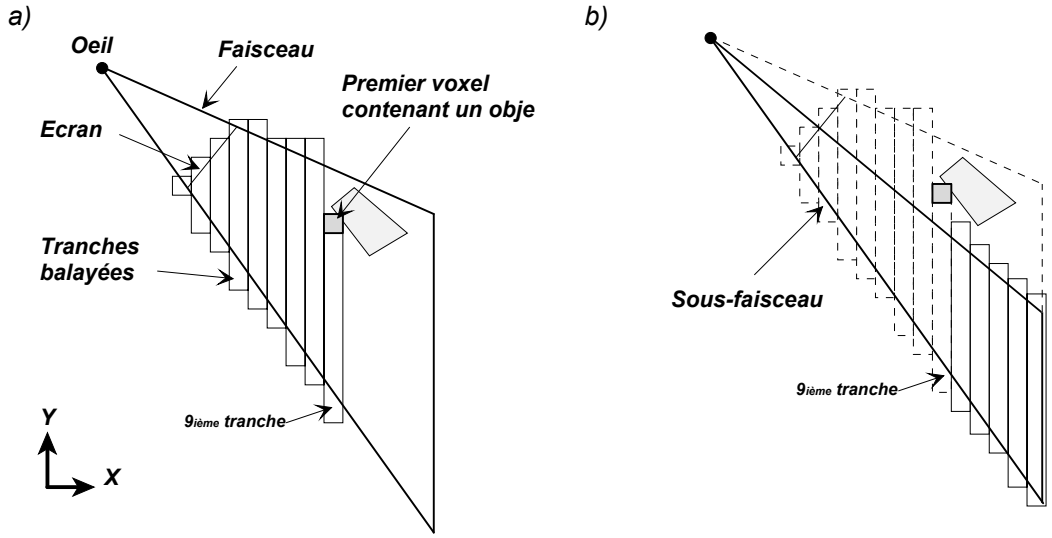


Figure 3.13 : Balayage d'un faisceau et d'un sous-faisceau.

3.1.5 Etude des performances

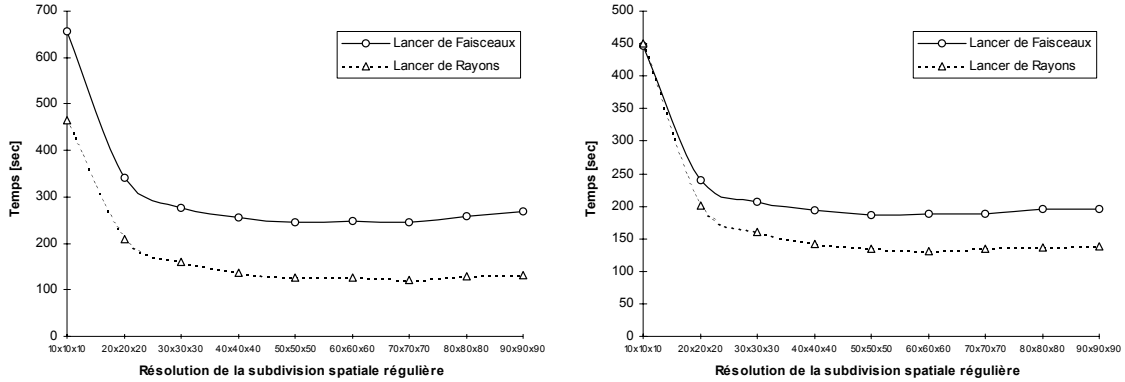
Les deux scènes tests utilisées pour évaluer les performances de notre algorithme représentent deux cas de figures très différents : une ville[13] (***Image 10***) dont l'occupation de la subdivision spatiale régulière est faible, en effet, 85% des voxels sont vides (voir

[13] Ville virtuelle : deux sources lumineuses ponctuelles, 1519 polyèdres (9114 polygones).

Paragraphe 3.1.5.2, ***Figure 3.22***) et un ensemble de cubes aléatoires[14] (***Image 11***) occupant l'espace de manière plus homogène avec 60% de voxels vides (voir ***Paragraphe 3.1.5.2***, ***Figure 3.22***). Sauf avis contraire, toutes les comparaisons sont faites avec des images d'une définition de 1024x1024 pixels.

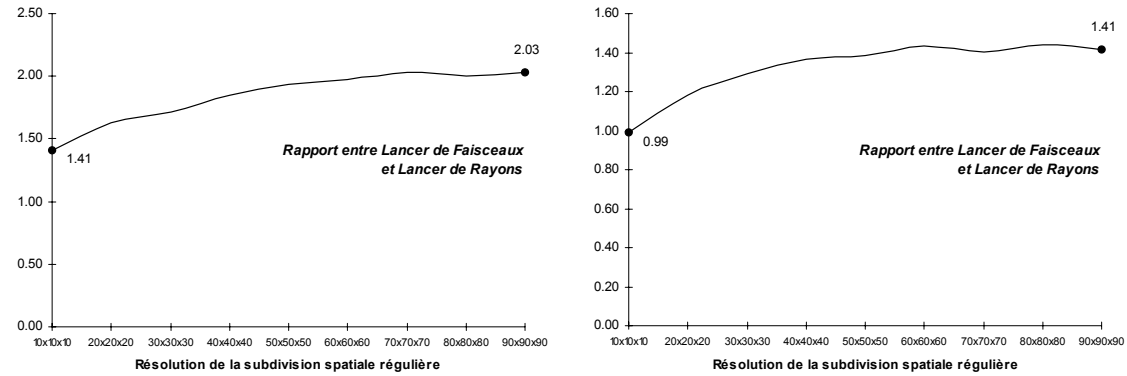
3.1.5.1 Comparaison en temps

Nous avons comparé notre lancer de faisceaux optimisé avec un lancer de rayons optimisé avec une subdivision spatiale régulière [AMAN 87]. Comme la résolution de la subdivision spatiale est un facteur très important, nous l'avons fait varier de 10x10x10 à 90x90x90. Ces bornes représentent les résolutions habituellement utilisées. Dans ces conditions, le lancer de faisceaux est moins rapide que le lancer de rayons (***Figure 3.14***). En fonction de la résolution de la subdivision et de la scène, le rapport de temps entre les deux approches varie entre 1 et 2 (***Figure 3.15***). Rappelons tout de même que le lancer de faisceaux a pour objet principal de résoudre de manière précise tous les problèmes d'aliassage et que cette lenteur relative peut alors passer au second plan. Notons aussi que, comparé à la version utilisant des volumes englobants (***Paragraphe 2.1***), ce lancer de faisceaux optimisé permet de calculer des images de scènes comportant beaucoup plus d'objets en beaucoup moins de temps.



Scène de la ville virtuelle. *Scène des cubes aléatoires.*

Figure 3.14 : Comparaison en temps[15] entre le lancer de faisceaux et le lancer de rayons .



Scène de la ville virtuelle. *Scène des cubes aléatoires.*

Figure 3.15 : Rapport des temps[15] entre le lancer de faisceaux et le lancer de rayons

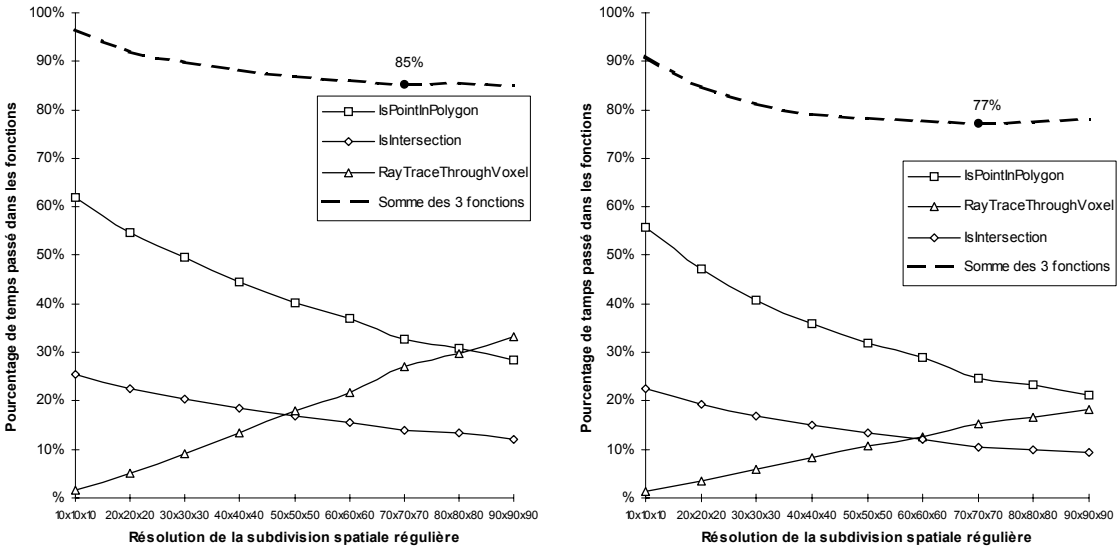
On peut se demander pourquoi le lancer de faisceaux est plus lent que le lancer de rayons. Pour comprendre cela, nous avons mesuré le temps passé dans chacune des fonctions à l'aide de la commande *gprof* d'UNIX. Les proportions de temps sont très

[14] Ensemble de cubes de tailles et de positions aléatoires : deux sources lumineuses ponctuelles, 2060 cubes (12360 polygones).

[15] Temps mesurés avec la fonction *clock()* de la librairie C sur une station HP Série 9000-J282 (2 µp PA8000, 175Mhz, 512Mo). Le compilateur natif HP a été utilisé avec les options -O4.

proches pour nos deux scènes tests, nous ne présentons donc que les résultats pour la scène de la ville virtuelle. Différents enseignements peuvent être extraits de ces mesures.

La ***Figure 3.16*** montre les fonctions qui sont les plus sensibles aux variations de la résolution de la subdivision spatiale et qui consomment le plus de temps machine. Ce sont les mêmes fonctions aussi bien pour le lancer de rayons que pour le lancer de faisceaux. Elles sont au nombre de trois : tester si un point est dans un polygone ("*IsPointInPolygon*"), tester s'il y a une intersection entre un rayon et un polyèdre ("*IsIntersection*") et lancer un rayon à travers les voxels ("*RayTraceThroughVoxel*"). On constate que plus on gagne du temps sur le calcul des intersections et plus on en perd en traversant les voxels. Ceci explique que les courbes de la ***Figure 3.14*** remontent autour d'une résolution de 70x70x70 pour le lancer de rayons et autour de 50x50x50 pour le lancer de faisceaux. De plus, on constate que pour des résolutions supérieures à 30x30x30, le lancer de rayons passe près de 85% du temps dans ces trois fonctions et le lancer de faisceaux près de 77%.

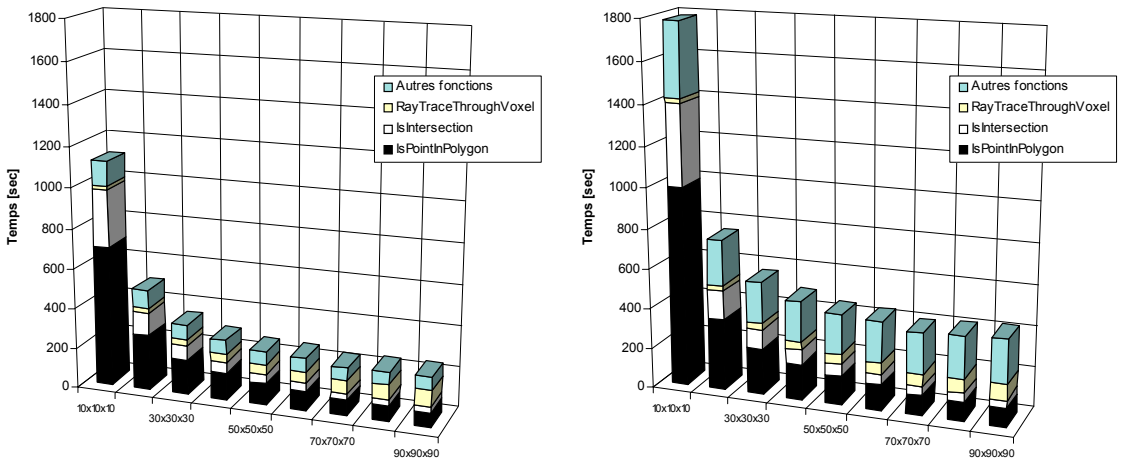


Lancer de rayons conventionnel. *Lancer de faisceaux.*

Figure 3.16 : Pourcentage de temps[16] passé dans les fonctions variant le plus en fonction de la résolution de la subdivision spatiale dans la scène de la ville virtuelle.

La ***Figure 3.17*** montre que le lancer de faisceaux passe plus de temps dans les trois fonctions principales que le lancer de rayons. Notons que ceci est tout à fait normal. En effet, comme nous l'avons précisé au ***Paragraphe 3.1.4.1***, les scènes avec beaucoup de détails nécessitent la construction de nombreux faisceaux et de nombreux calculs d'intersections.

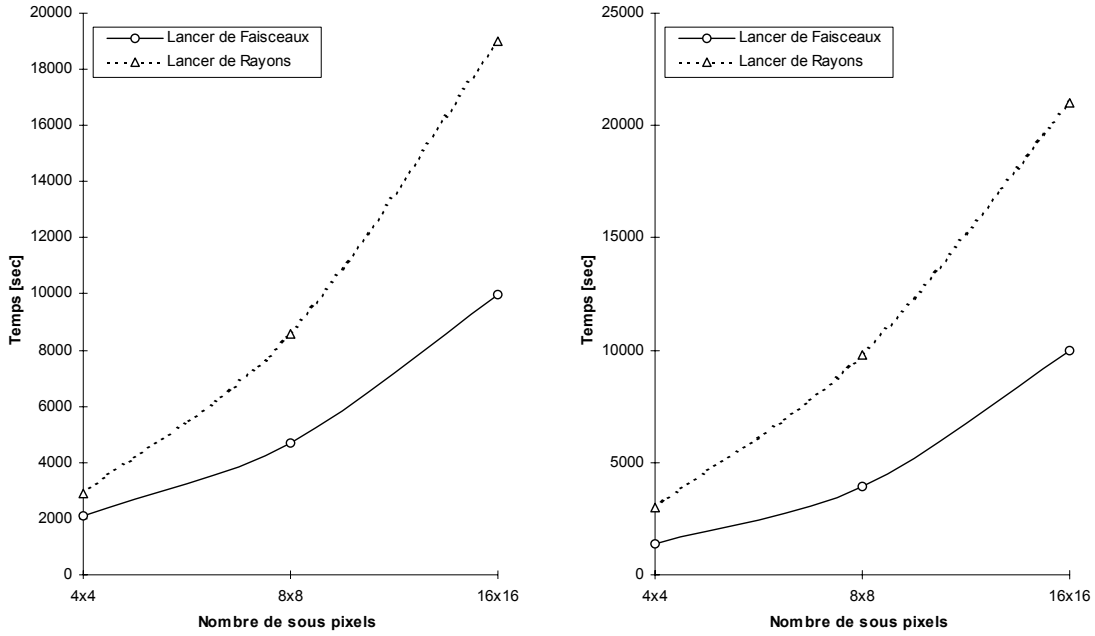
[16] Les temps ont été mesurés en secondes sur un Alpha Server 2000 4/275 (1 µp, 275Mhz, 128Mo) avec le "*profiler*" d'UNIX *gprof*. Le compilateur utilisé est g++ 2.7.1 avec les options -O4 -gp. La définition est de 1024x1024 et la subdivision spatiale est de 70x70x70 pour la ville et de 50x50x50 pour les cubes.



Lancer de rayons conventionnel. *Lancer de faisceaux.*

Figure 3.17 : Temps[16] passé dans les fonctions consommant le plus de ressources machine dans la scène de la ville virtuelle.

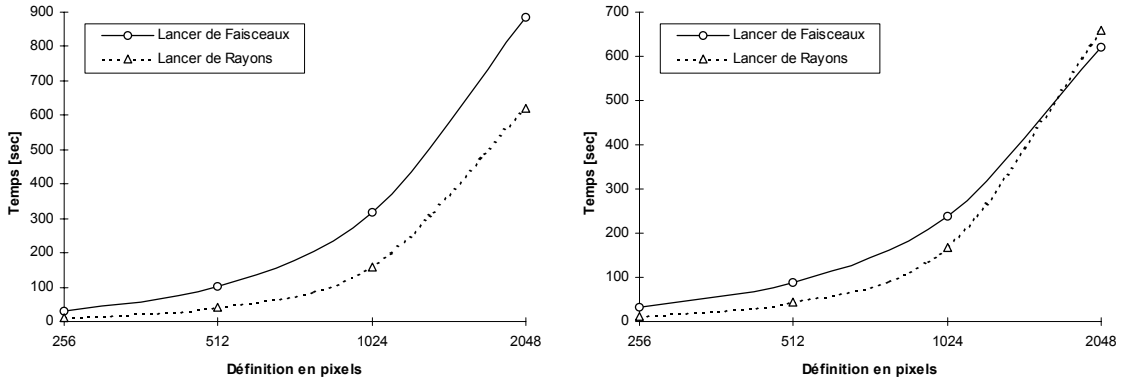
En général, on cherche à produire des images sans aliassage. Pour cela, on utilise un lancer de rayons avec une subdivision adaptative des pixels [WHIT 80]. Cependant, cette technique ne permet pas de détecter, à coup sûr, tous les petits objets et petites ombres. Il est donc difficile de comparer notre lancer de faisceaux avec un tel algorithme puisqu'il ne produit pas les mêmes images. Pour faire une comparaison plus réaliste, nous comparons notre algorithme à un lancer de rayons avec une subdivision systématique de tous les pixels. Précisons qu'une subdivision de 4x4 sous-pixels est acceptable pour traiter les problèmes de marches d'escalier, qu'une subdivision de 8x8 est un minimum pour essayer de traité la disparition des petits objets et petites ombres et que l'on se limite à 16x16 sous-pixels. En effet, une subdivision de 16x16 permet d'obtenir 256 valeurs pour chaque composante RVB, ce qui est proche de la limite de la sensibilité de l'oeil. Ainsi, on peut espérer comparer deux algorithmes produisant les mêmes images. La ***Figure 3.18*** montre le temps nécessaire à chacun des algorithmes pour différents niveaux de subdivision. Notons que dans le cas du lancer de faisceaux, ce niveau n'est pas systématiquement atteint. En effet, les subdivisions étant adaptatives, si une région est détectée uniforme mais qu'elle est plus grande que cette limite, elle n'est pas subdivisée. Ces courbes montrent clairement un gain de temps de l'ordre d'un facteur 2 lors de l'utilisation du lancer de faisceaux pour des images antialiassées à partir de 8x8 subdivisions au maximum.



Scène de la ville virtuelle. *Scène des cubes aléatoires.*

Figure 3.18 : Comparaison entre le lancer de rayons et le lancer de faisceaux pour le traitement de l'aliassage. Le lancer de rayons utilise une subdivision systématique.

Une dernière comparaison en temps intéressante dépend de la définition de l'image. Dans les exemples précédents, nous avons toujours considéré des images de 1024x1024 pixels. Ceci ne correspond pas toujours aux besoins. Ainsi, de très hautes définitions sont nécessaires pour réaliser des trucages vidéo[17]. C'est pourquoi, nous avons comparé notre lancer de faisceaux avec un lancer de rayons sans traitement de l'aliassage et pour des définitions allant de 256x256 à 2048x2048. Les courbes de la ***Figure 3.19*** montrent les temps de calculs pour ces différentes définitions. On constate que plus l'image est grande et plus on a intérêt à utiliser l'algorithme de lancer de faisceaux. En effet, le rapport de temps entre les deux algorithmes évolue en faveur des faisceaux en fonction de l'augmentation du nombre de pixels traités (***Figure 3.20***).

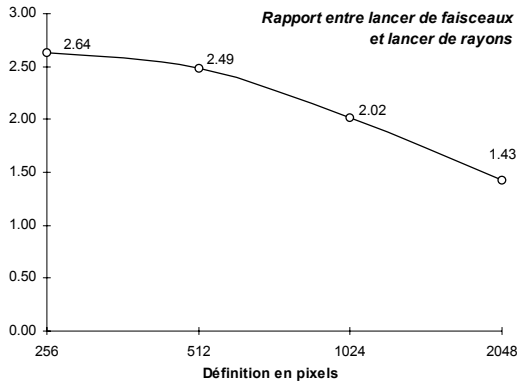


Scène de la ville virtuelle. *Scène des cubes aléatoires.*

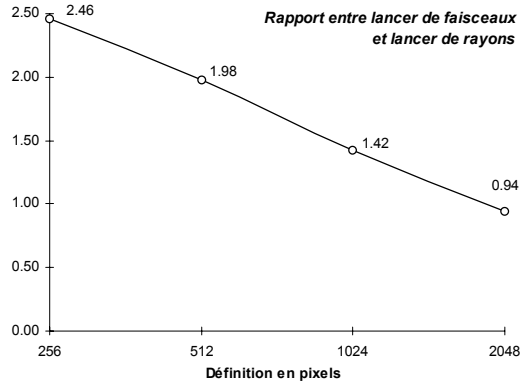
Figure 3.19 : Comparaison en temps[18] en fonction de la définition de l'image.

[17] Le support considéré lors de trucages vidéo est le film.

[18] Temps mesurés avec la fonction *clock()* de la librairie C sur une station HP Série 9000-J282 (2 µp PA8000, 175Mhz, 512Mo). Le compilateur natif HP a été utilisé avec les options -O4.



Scène de la ville virtuelle.



Scène des cubes aléatoires.

Figure 3.20 : Rapport entre le lancer de faisceaux et le lancer de rayons.

3.1.5.2 Occupation mémoire

La principale occupation de la mémoire est due à la subdivision spatiale régulière. Nous présentons cette occupation pour les scènes tests représentant la ville virtuelle et les cubes de dimensions et de positions aléatoires. La ***Figure 3.21*** montre l'évolution de la place mémoire utilisée par la subdivision spatiale régulière en fonction de différentes résolutions. On peut constater que cette occupation n'est pas excessive et toujours inférieure à 10 Mo. De plus, il est à noter qu'aucune optimisation visant à réduire cette occupation n'a été faite. En particulier, nous n'avons pas utilisé de techniques de stockage dérivées des matrices creuses. Cette technique peut faire gagner beaucoup de place. En effet, la ***Figure 3.22*** montre que, suivant la scène et pour une résolution de plus de 30x30x30, près de 60% à 80% de voxels sont vides. Pour avoir une idée de la taille de la liste des objets associés à chaque voxel, la ***Figure 3.23*** montre l'évolution du nombre maximum d'objets associés à un voxel en fonction de la taille de la subdivision spatiale. On peut noter que les courbes des temps ***Figure 3.11*** ont la même forment que celles du nombre maximum d'objets par pixel. On peut donc conclure que l'augmentation de la résolution de la subdivision spatiale ne diminue pas le nombre maximum d'objets par voxel et par conséquent ne diminue pas les temps de calculs.

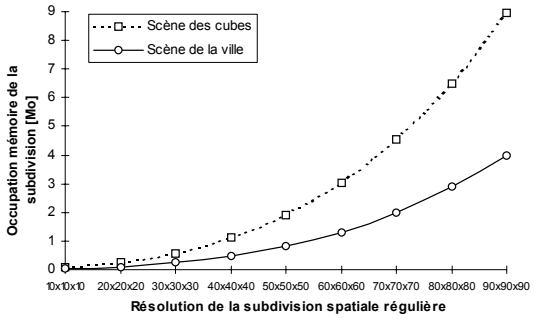


Figure 3.21 : Mémoire nécessaire pour stocker la subdivision spatiale régulière.

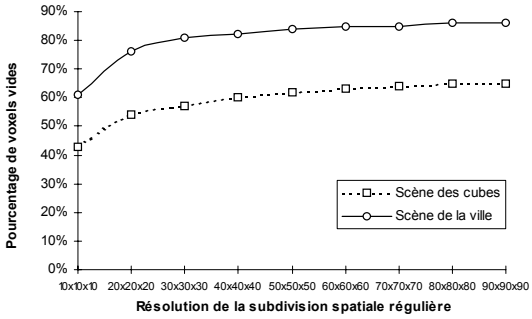


Figure 3.22 : Pourcentage de voxel vides.

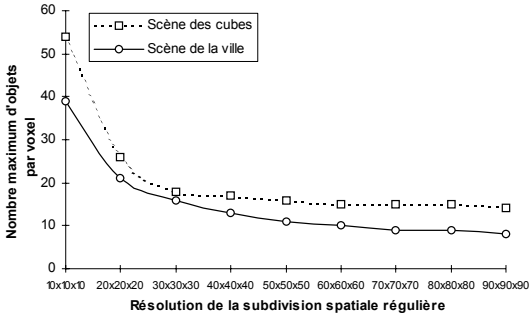


Figure 3.23 : Nombre maximum d'objets par voxel.

3.2 EXTENSION

La première partie a été consacrée aux optimisations apportées à l'algorithme proposé par [GHAZ 92] ainsi qu'à une étude détaillée des performances obtenues. Cette deuxième partie est à présent consacrée à une extension permettant de produire directement des effets de pénombre sans aucun échantillonnage stochastique. Nous présentons la méthode puis les résultats obtenus.

3.2.1 Effets de pénombre

Les effets de pénombre sur une région proviennent de sources lumineuses non ponctuelles. En effet, dans ce cas, un point peut être soit totalement éclairé, soit totalement à l'ombre ou dans une région appelée *pénombre*, éclairé seulement par une partie de la source lumineuse (***Figure 3.24***).

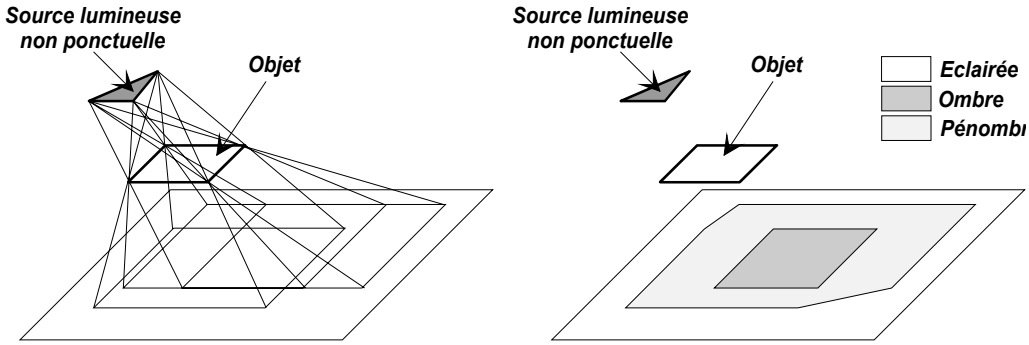


Figure 3.24 : Régions entièrement éclairées, dans la pénombre et totalement à l'ombre.

[WOO 90] présente plusieurs techniques pour produire de la pénombre :

- les techniques classiques fondées sur un lancer de rayons stochastique comme celui de [COOK 84] qui est relativement coûteuse pour de larges sources lumineuses ;
- le lancer de cônes développé par [AMAN 84] qui produit des pénombres à partir de sources lumineuses sphériques mais qui nécessite des calculs complexes d'intersection cône-objet.

Dans ce paragraphe, nous présentons une méthode utilisant notre lancer de faisceaux et des sources lumineuses sphériques. Nous allons montrer comment on peut produire des pénombres réalistes. Notons bien que l'on ne cherche pas des effets physiquement corrects mais plutôt une représentation "agréable" à l'oeil pour un coût raisonnable. En particulier, nous ne cherchons pas à appliquer très précisément les lois de l'optique. Les pénombres proposées sont semblables à celle produites par un lancer de rayons stochastique pour des petites sources lumineuses. Par contre, notre méthode offre un plus lors de l'utilisation de larges sources.

Pour obtenir des effets de pénombre, on peut diviser le problème en deux sous-problèmes :

- trouver les régions situées dans la pénombre (***Paragraphe 3.2.1.1***) ;
- évaluer la quantité d'énergie produite par la source lumineuse atteignant chaque point d'une région de pénombre (***Paragraphe 3.2.1.2***).

3.2.1.1 Détection des régions de pénombre

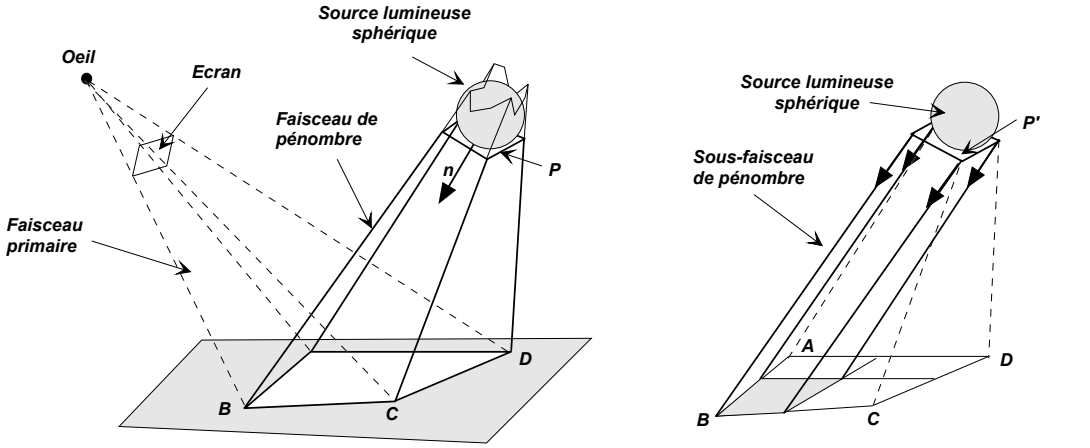
L'idée est semblable à celle utilisée dans le cas des sources ponctuelles : on utilise un faisceau pour déterminer les régions à problèmes (celles dans la pénombre). Pour repérer ces régions de pénombre, on construit, pour chaque région *ABCD* uniforme pour les faisceaux primaires de réflexion et de réfraction, un *faisceau de pénombre* (***Figure 3.25.a***). La base de ce faisceau est *ABCD*, ses quatre pans sont tangents à la source lumineuse sphérique considérée et le sommet est délimité par le plan *P* (***Figure 3.25.a***). *P* est un plan tangent à la source sphérique et dont la normale *n* pointe approximativement au "centre" de *ABCD* (***Figure 3.26***).

Un tel faisceau englobe les chemins parcourus par tous les rayons lumineux atteignant la surface *ABCD*. Ce faisceau est en fait un peu plus large qu'en réalité, mais les résultats présentés au ***Paragraphe 3.2.1.3*** montrent que l'effet réaliste est très convenable.

Le faisceau de pénombre est utilisé avec des tests similaires à ceux des faisceaux d'ombre (***Paragraphe 1.3.2.3.***) pour déterminer le type de la région *ABCD* : entièrement éclairé, totalement à l'ombre ou dans la pénombre.

Dans les deux premiers cas, la région est considérée uniforme et aucune subdivision n'est nécessaire. Pour les régions entièrement éclairées (***Figure 3.25.a***), le calcul de la couleur est le même que dans le cas d'une source ponctuelle (***Paragraphe 1.3.2.3.***). En fait, dans ce cas, la source sphérique est approchée à une source ponctuelle de même énergie, située au centre de la sphère. Pour les régions entièrement à l'ombre, le calcul de la couleur des points est identique au cas des sources ponctuelles (***Paragraphe 1.3.2.3.***).

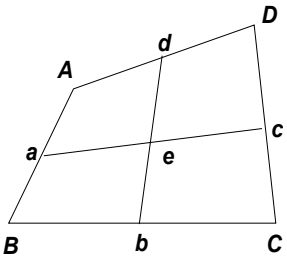
Dans le troisième cas, la région image est subdivisée. La subdivision récursive de la région et des faisceaux de pénombre correspondants ressemble au cas des faisceaux d'ombre. La base de chaque sous-faisceau est l'une des quatre sous-régions de *ABCD*, leurs pans sont tangents à la source lumineuse sphérique et leur sommet est délimité par le plan *P'* (***Figure 3.25.b***). *P'* est obtenu de la même manière que *P*. La récursion se termine lorsque l'on trouve une région uniforme (entièrement éclairée ou totalement à l'ombre) ou lorsque l'on a atteint la taille d'un pixel.



***a**: faisceau de pénombre.*

***b**: Subdivision du faisceau de pénombre correspondant à une région strictement plus grande qu'un pixel.*

Figure 3.25 : Faisceau de pénombre



Soit a *(resp.* b, c *et* d*) le milieu de* $\overline{AB}$ *(resp.* $\overline{BC}$, $\overline{CD}$ et $\overline{DA}$*). Le centre* e *est défini comme l'intersection de* $\overline{ac}$ *et de* $\overline{bd}$.

Figure 3.26 : Approximation du centre du polygone ABCD.

3.2.1.2 Proportion de lumière reçue par un pixel

Nous sommes à présent dans le cas d'une région correspondant à un pixel et pour laquelle il faut évaluer la proportion d'énergie reçue. Pour cela, nous subdivisons récursivement le faisceau de pénombre en quatre sous-faisceaux (***Figure 3.27***) jusqu'à obtenir une région uniforme ou à atteindre une limite arbitraire de subdivision. La base d'un sous-faisceau est construite comme précédemment, ses pans sont définis de façon à diviser P' en quatre polygones d'approximativement la même surface (***Figure 3.27***). Cette division utilise l'approximation du centre d'un polygone de la ***Figure 3.26***. Considérons par exemple le cas d'un pixel partiellement éclairé (***Figure 3.28.b*** ou ***c***) et son faisceau correspondant. Ce pixel et P' sont récursivement subdivisés. Finalement, l'éclairement de ce pixel est déterminé par la somme des surfaces des sous-faisceaux. Cette somme est une approximation de la proportion d'énergie reçue par le pixel. La ***Figure 3.28*** montre la transition d'un pixel entièrement éclairé à un pixel totalement à l'ombre.

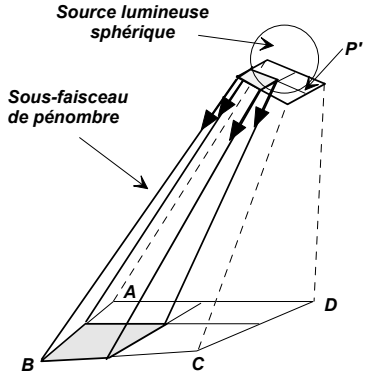


Figure 3.27 : Subdivision d'un faisceau de pénombre correspondant à une région d'un pixel.

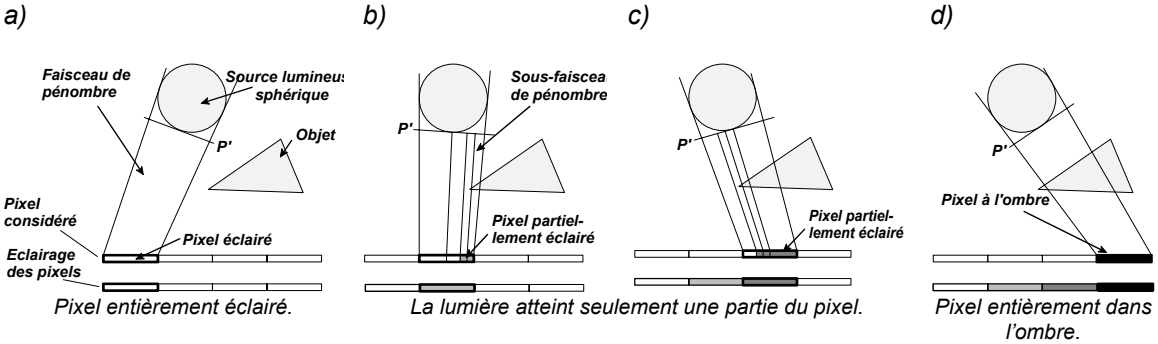


Figure 3.28 : Exemple, en 2D, de la transition de la pénombre sur quatre pixels

Il est à noter que pour les sous-pixels les plus petits (à la limite de la subdivision), le calcul de la couleur est semblable à celui des sources ponctuelles (***Paragraphe 1.3.2.3.***). En fait, dans ce cas, la source lumineuse sphérique est approchée par une source ponctuelle de

même énergie et placée au centre du polygone formant le sommet du sous-faisceau de pénombre.

3.2.1.3 Résultats

Les tests ont été effectués sur la scène de la ville virtuelle (***Image 12***) et sur une scène pédagogique représentant sept piliers (***Image 13***). Les différences de temps d'exécution entre le lancer de rayons stochastique et le lancer de faisceaux sont sensibles. La ***Figure 3.29*** présente les temps pour les deux scènes en fonction du nombre de sous-faisceaux de pénombre pour le lancer de faisceaux et du nombre de rayons stochastiques pour le lancer de rayons (la définition de l'image est de 1024x1024). On peut constater que pour une image comportant de nombreux détails, comme celle de la ville, le lancer de rayons est plus rapide. Par contre, pour une image plus simple, comme celle des piliers, le lancer de faisceaux est plus rapide. Les temps supérieurs pour le lancer de faisceaux sont là encore certainement dû à la redondance des rayons lancer lors des subdivisions (voir ***Paragraphe 3.1.4.1***).

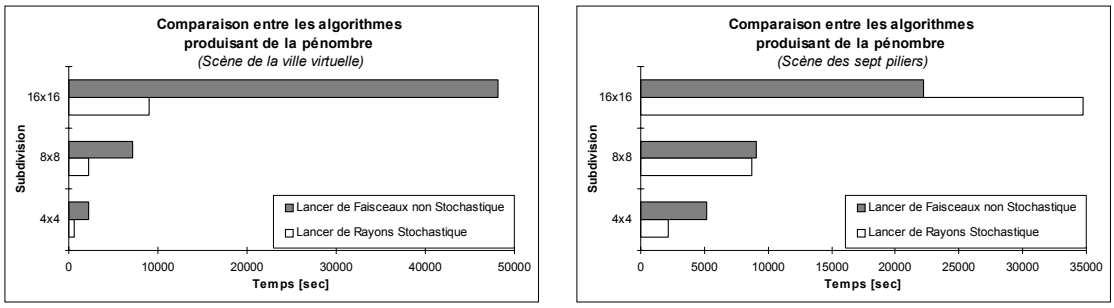


Figure 3.29 : Comparaison en temps[19] entre le lancer de rayons stochastique et le lancer de faisceaux non stochastique.

Concernant la différence visuelle entre le lancer de rayons stochastique et le lancer de faisceaux, elle est à peine visible sur la scène de la ville. En effet, nous sommes obligés d'agrandir huit fois les régions de pénombre pour voir une différence (***Image 12***). Ceci est dû à la petite taille des sources par rapport aux ombres portées. Par contre, dans le cas de scènes avec de larges sources lumineuses, comme dans le cas des sept piliers, la différence est visible (***Image 13***) et le lancer de faisceaux produit une pénombre de meilleure qualité.

[19] Les temps ont été mesurés en secondes sur une SGI Indy (µp R4010, 100Mhz, 32Mo) avec la fonction *clock()* de la librairie C. Le compilateur utilisé est g++ avec l'optimisation -O4.

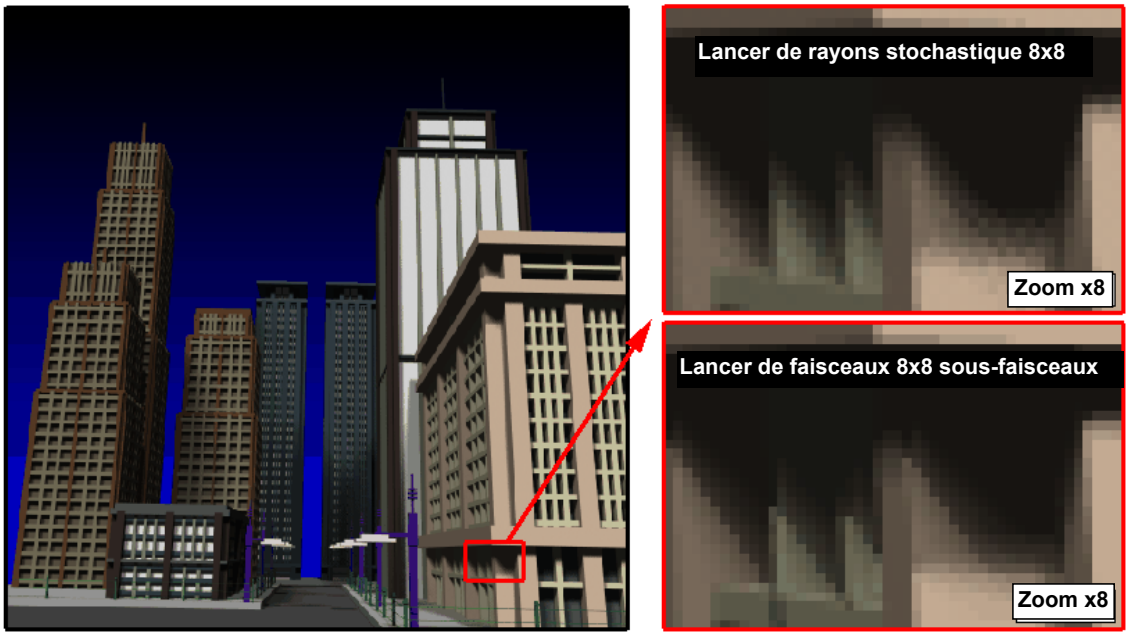


Image 12 : Zoom du haut : Lancer de rayons stochastique global avec 8x8 rayons,
Zoom du bas : lancer de faisceaux avec une subdivision maximale de 8x8 sous-faisceaux.

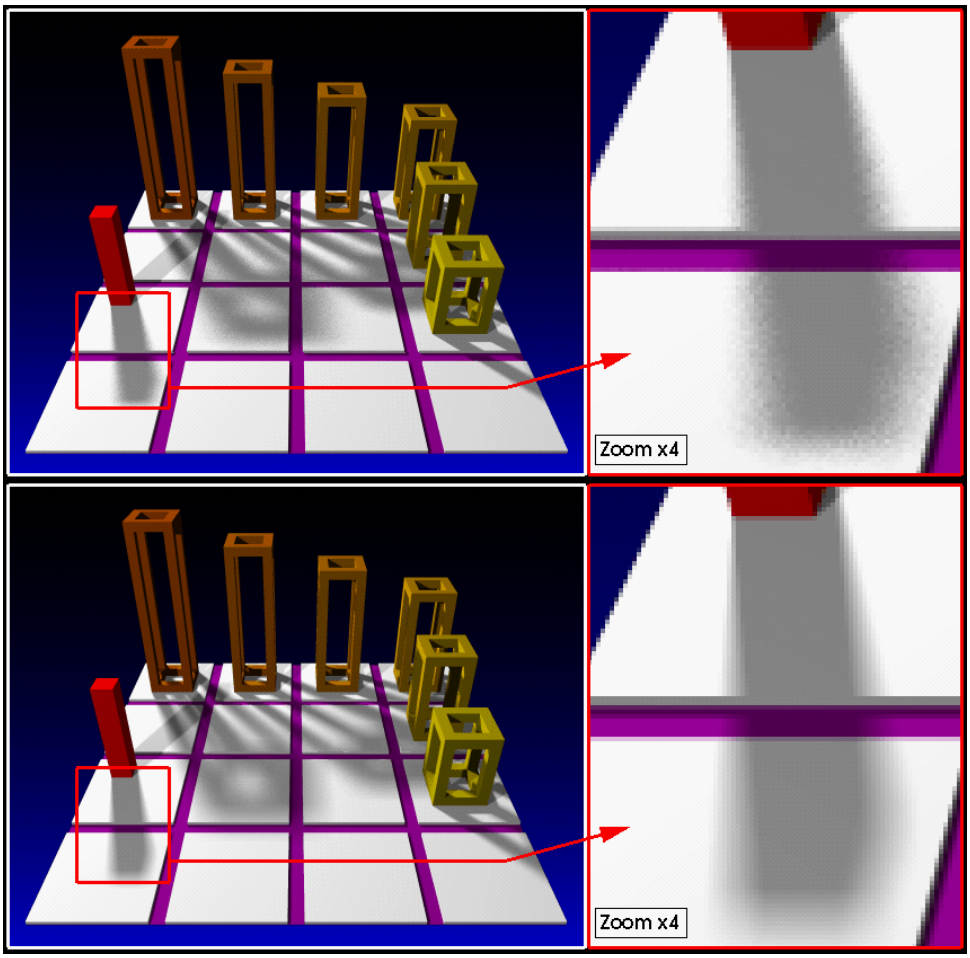


Image 13 : Haut : lancer de rayons stochastique global avec 8x8 rayons,
Bas : lancer de faisceaux avec une subdivision maximale de 8x8 sous-faisceaux.

En conclusion, les temps de calculs sont meilleurs avec un lancer de rayons stochastique mais la qualité atteinte avec le lancer de faisceaux est identique voire meilleure à celle produite par un lancer de rayons stochastique.

3.3 CONCLUSION

Nous avons présenté dans ce chapitre les optimisations [GHAZ 98] apportées à l'algorithme de lancer de faisceaux pyramidaux proposé par [GHAZ 92]. Celles-ci ont été appliquées à différentes phases de l'algorithme pour finalement améliorer les temps de calculs par rapport à la version initiale (***Paragraphe 2.1***). Les différentes comparaisons avec un lancer de rayons conventionnel optimisé ont permis de constater que le lancer de faisceaux est intéressant lorsque l'on a besoin de subdiviser les pixels, c'est-à-dire chaque fois que l'on cherche à résoudre les problèmes d'aliassage.

Dans la deuxième partie de ce chapitre, nous avons considéré les effets de pénombre. Ceux-ci ont été développés et produisent des effets tout à fait réalistes. En particulier, les sources lumineuses très grandes peuvent être utilisées. Les temps de calculs sont dépendants de la quantité de détails dans l'image. Moins il y a de détails et plus le lancer de faisceaux est rapide et intéressant par rapport à un lancer de rayons stochastique. Il reste néanmoins souvent plus lent mais produit une qualité de pénombre identique ou supérieure à un lancer de rayons stochastique.

Le lancer de faisceaux pyramidaux permet de produire des images sans problème d'aliassage, avec des effets de réflexion et de réfraction réalistes, des effets de pénombre de bonne qualité (***Image 14***) et des temps de calculs souvent avantageux. Les chapitres suivants vont montrer comment utiliser les faisceaux dans d'autres domaines de la synthèse d'images.

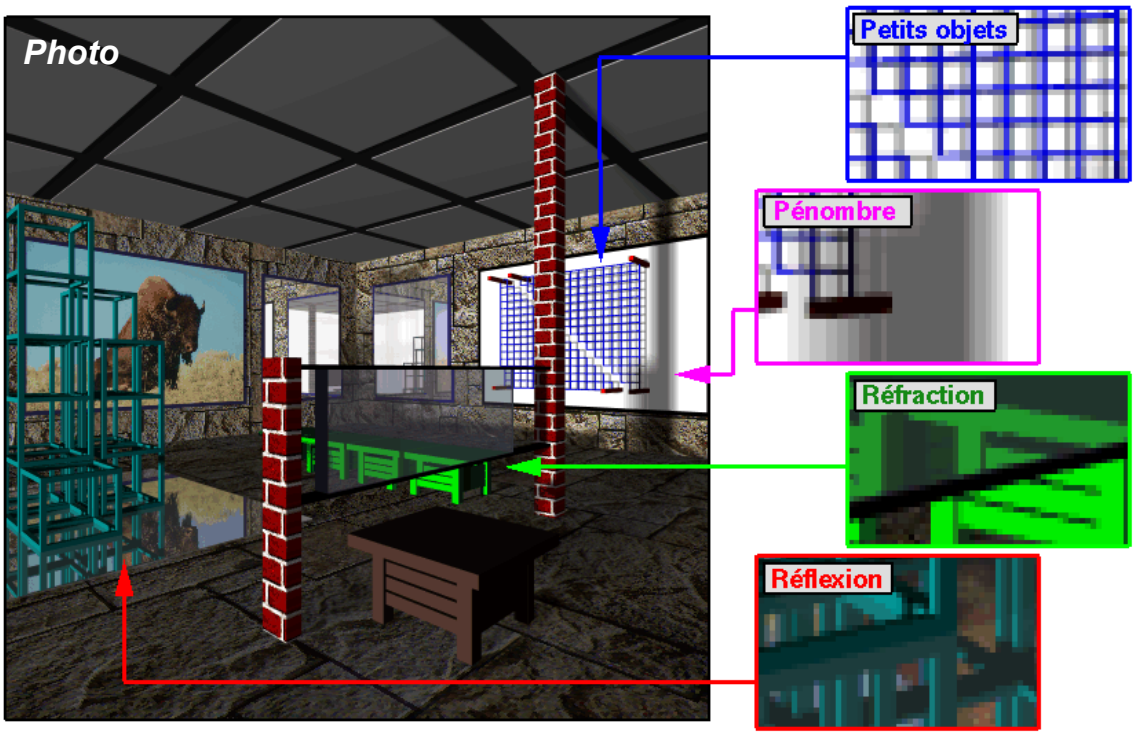
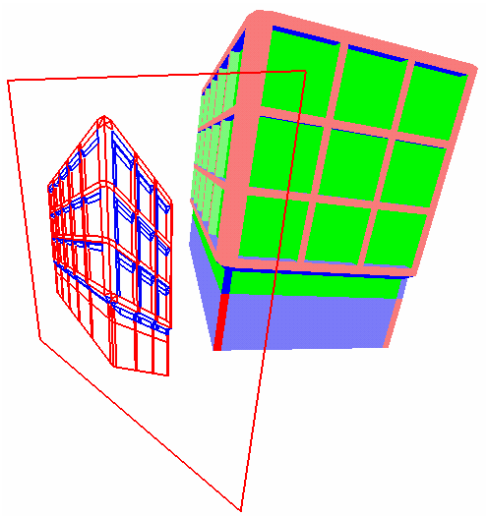


Image 14 : Synthèse des effets obtenus avec notre lancer de faisceaux.

4. CARTES GENERALISEES ET FAISCEAUX POUR UNE VISIBILITE EXACTE



Lorsque l'on parle d'algorithme de visibilité, on s'accorde à différencier deux approches : celle travaillant dans l'espace image et celle travaillant dans l'espace objet. La première est dépendante de la définition de l'écran (l'image). Le but de ces algorithmes est simplement de calculer la couleur de chaque pixel de l'écran. Les plus connus d'entres eux sont certainement le tampon de profondeur ("z-buffer") [CATM 74] et le lancer de rayons [APPE 68]. La deuxième approche est indépendante de toute définition de l'image, sa précision est en général limitée à celle des calculs flottants. La représentation de l'image est exacte et peut être agrandie sans perte d'information. Une description complète de ces algorithmes est proposée dans [DORW 94]. Le lancer de faisceaux pyramidaux [GHAZ 92] proposé et optimisé aux chapitres précédents travaille dans l'espace image. Dans ce chapitre, nous proposons une solution dans l'espace objet.

Le modèle topologique des cartes généralisées [LIEN 89c] permet entre autre de disposer directement d'informations concernant l'adjacence des arêtes et des faces. Ceci nous permet de réaliser un algorithme de calcul exact de la visibilité et de traiter efficacement les problèmes d'aliassage. Nous proposons donc d'étudier un tel algorithme, fondé sur les cartes généralisées. Cet algorithme est, de plus, optimisé par des faisceaux pour limiter le nombre d'objets à traiter.

Disposer d'un algorithme travaillant dans l'espace objet, c'est-à-dire, produisant une image dans $\mathcal{R}^2$ indépendante de la définition de l'écran, a plusieurs intérêts. Citons entre autres la possibilité d'effectuer des agrandissements sans aucune perte de qualité ou la possibilité d'imprimer, de manière optimale, l'image sur n'importe quel support (papier, film, …). De plus, les problèmes d'aliassage (marches d'escalier et disparition des petits objets et petites ombres) peuvent être traités efficacement.

Ce travail a été réalisé en collaboration avec l'équipe modélisation. C'est une première approche pour réaliser un algorithme de visibilité alliant les cartes généralisées et les faisceaux, c'est pourquoi, nous nous sommes concentrés principalement sur tous les problèmes théoriques. Une partie seulement de notre solution a été développée pour vérifier et valider nos choix. En particulier, nous n'avons pas implémenté les effets de réflexion et

de réfraction. La partie rendu a été réalisée très simplement en utilisant le tampon de profondeur câblé sur nos stations de travail.
Ainsi, ce chapitre se divise en trois parties : des notions générales sur les cartes généralisées, les deux phases de notre algorithme et les travaux à venir.

4.1 NOTIONS DE CARTES GENERALISEES

La présentation de notre solution étant focalisée sur l'aspect algorithmique et sur l'utilisation des faisceaux, nous ne présentons pas en détails le modèle des cartes généralisées. Nous les considérons comme une boîte noire aux propriétés avantageuses, comme la connaissance directe des adjacences entre arêtes et faces. Pour des détails sur ce modèle et pour une description complète des avantages, nous renvoyons le lecteur aux différentes publications [LIEN 89a] [LIEN 89b] [LIEN 89c]. Il est cependant nécessaire de connaître un minimum de vocabulaire et de propriétés.

Dans le domaine de la représentation par les bords ("*B-rep*"), un objet géométrique est défini par un *modèle topologique* et un *modèle de plongement*. Les modèles topologiques qui sont utilisés ici sont les *2* et *3-G-cartes*. Ces modèles permettent respectivement de décrire la topologie des *subdivisions* de dimension 2 et 3. Une subdivision d'un espace de dimension 2 (resp. 3) peut être informellement définie comme une partition de cet espace en *sommets*, *arêtes* et *faces* (***Figure 4.1***) (resp. sommets, arêtes, faces et volumes). Outre la description des *cellules* (sommets, arêtes, faces et volumes), les modèles topologiques incluent la description des *relations de bord* entre cellules (par exemple relations de bord entre les faces le long des arêtes de leur bord ou relations de bord entre volumes le long des faces de leur bord). De ces relations de bord on déduit les relations d'*incidence* et les relations d'*adjacence* entre les cellules. Deux cellules en relation par leur bord sont dites adjacentes si elles sont de même dimension et incidentes si elles sont de dimensions différentes. On parle ainsi de sommets incidents à une face et de faces adjacentes. Nous verrons au ***Paragraphe 4.2.2*** comment se servir de la notion d'adjacence pour calculer le rendu final de l'image. Enfin, toujours sans rentrer dans les détails des deux modèles topologiques utilisés, il est important de noter que ceux-ci nous permettent d'obtenir immédiatement les différents *bords* des cartes (pour une 2-G-carte, c'est l'ensemble des arêtes incidentes à une seule face (***Figure 4.1***).

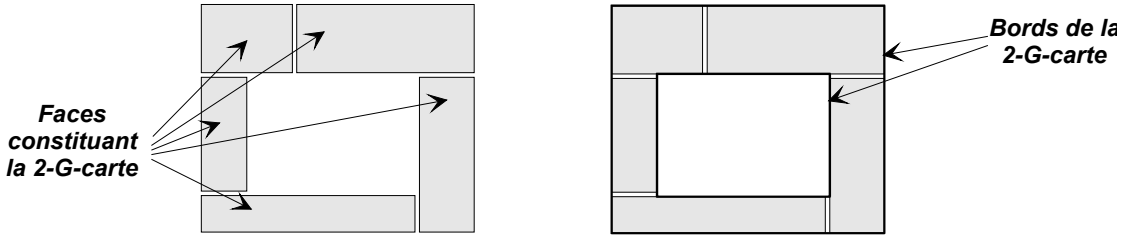


Figure 4.1 : Faces et bords d'une 2-G-carte.

Le modèle de plongement choisi ici est linéaire. On associe un point à un sommet, un segment de droite à une arête, un polygone à une face et un polyèdre à un volume.

Outre les notions de modèle topologique et de plongement, deux opérations importantes sont utilisées dans notre approche : le *découpage* et le *coraffinement*. Le découpage d'une face A par une face B produit deux faces. La face découpée A' et la face B qui n'a pas été modifiée (***Figure 4.2***). Le coraffinement d'une face A et d'une face B produit trois faces : A' qui correspond au découpage de A par B, B' qui correspond au découpage de B par A et AB qui correspond à la partie commune entre A et B (***Figure 4.2***).

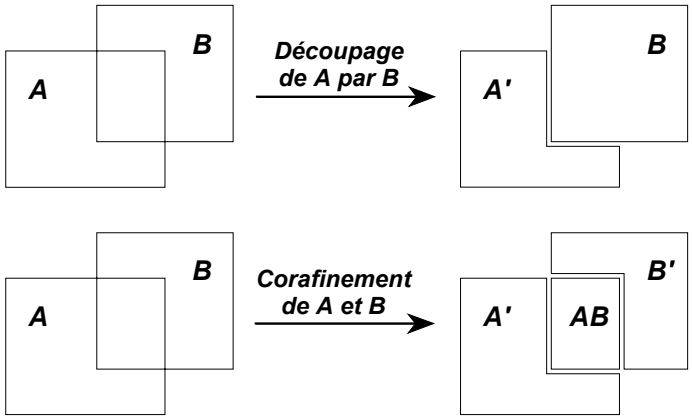


Figure 4.2 : Différences entre le découpage et le coraffinement.

Le coraffinement est un mécanisme très important dans un algorithme de visibilité travaillant dans l'espace objet. En effet, c'est une des opérations les plus réalisées. L'utilisation des cartes généralisées facilite grandement cette opération.

4.2 UNE APPROCHE FONDEE SUR LES CARTES ET LES FAISCEAUX

Nous proposons ici un algorithme de calcul de visibilité se plaçant dans l'espace objet. Il inclut des effets d'ombre et de réflexion et approche les effets de réfraction. Le but de cet algorithme est ainsi de décrire, avec précision, les contours des objets visibles, des ombres, des réflexions et des réfractions. Comme nous le verrons par la suite (***Paragraphe 4.2.2.2***), une représentation sous forme de G-cartes permet de résoudre de manière efficace tous les problèmes d'aliassage, en particulier la disparition de petits objets et petites ombres. Le traitement des marches d'escalier est aussi résolu de façon optimale. D'autre part, cette représentation autorise la manipulation de l'image à l'écran sans perte d'information. Des effets de zoom très précis peuvent, par exemple, être effectués.

Pour faciliter le calcul et la manipulation de ces contours, nous utilisons des cartes généralisées (voir paragraphe précédent). Pour diminuer le nombre d'objets à traiter, nous optimisons l'algorithme à l'aide de faisceaux et d'une subdivision spatiale régulière (voir ***Paragraphe 3.1***). Notre approche est ainsi fondée sur une "symbiose" entre les G-cartes et les faisceaux. Cet algorithme comporte deux étapes distinctes :

- Le calcul, optimisé par les faisceaux, d'une 2-G-carte dans le plan de l'écran représentant tous les contours des objets, de leurs ombres portées, des réflexions et réfractions ;
- Le calcul du rendu proprement dit à partir de cette 2-G-carte.

Pour réaliser cet algorithme nous émettons plusieurs hypothèses. Tout d'abord, la scène est décrite à l'aide d'une 3-G-carte appelée $3G_{objets}$. On dispose ainsi directement des propriétés d'adjacence des sommets, des arêtes, des faces et des volumes. Pour des raisons d'efficacité, nous nous limitons à des scènes formées de polygones convexes. Ceci n'est pas une limitation très importante, puisqu'il est toujours possible de découper un polygone concave en un ensemble de polygones convexes. Les polygones de la scène ne doivent pas avoir d'intersection ni se superposer de manière cyclique (***Figure 4.3***). Là encore, ces limitations peuvent être levées en découpant les polygones afin de supprimer ces cas particuliers. Notons néanmoins que les superpositions cycliques de polygones dépendent du point de vue, il est donc impossible d'effectuer ce découpage en prétraitement. De plus, ces découpages introduisent de nouvelles arêtes ; celles-ci sont marquées afin d'être supprimées lors de la phase de rendu.

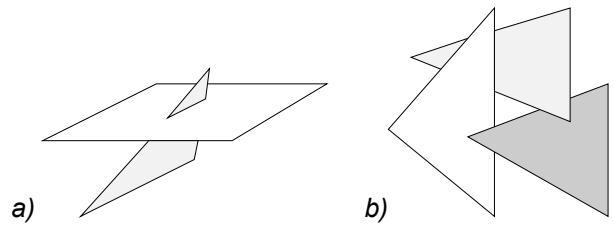


Figure 4.3 : Cas particuliers : a) intersection de polygones.
b) polygones superposés de manière cyclique.

4.2.1 2-G-carte de tous les contours

La première étape de notre algorithme consiste à décrire tous les contours visibles des objets, des ombres, des réflexions et des réfractions, sous la forme d'une 2-G-carte dans le plan de l'écran. La construction de chacun de ces contours est à présent étudiée. Nous commencerons par traiter le cas d'objets opaques et non réfléchissants.

4.2.1.1 Contour des objets

La construction des contours des objets est dans un premier temps décrite succinctement pour avoir une idée globale de la démarche, puis les différents points sont justifiés et décrits en détail. Cette première étape est en fait un algorithme de calcul de la visibilité directe, fondé sur un algorithme de découpage optimisé avec des faisceaux. La démarche générale est la suivante :

- Initialiser la 2-G-carte de tous les contours des objets visibles sur l'écran E à l'ensemble vide. Cette carte est appelée $2G^{E}_{objets}$.
- Construire un *faisceau primaire* F semblable à celui utilisé pour le lancer de faisceaux (***Paragraphe 1.3.2.2***). Il a pour sommet l'oeil et traverse la totalité de l'écran E (***Figure 4.4***).
- Construire, la liste L des faces de la 3-G-carte $3G_{objets}$ susceptibles d'être partiellement ou entièrement dans le faisceau F.
- Trier L en profondeur (en fonction de la distance minimale à l'oeil).
- Considérer la première face P_1 de L et construire un faisceau F_{P1} issu de l'oeil et traversant exactement P_1 (***Figure 4.5***). Notons que ce faisceau a autant de pans que la face a de côtés.
- Marquer la position exacte de toutes les faces de L par rapport à F_{P1}. Si une face se trouve derrière P_1 (c-à-d du côté opposé à E) et entièrement dans F_{P1}, la supprimer de la liste L.
- Si une face P_i de L se trouve devant P_1 (c-à-d du même côté que E), inverser l'ordre de P_i et P_1 dans L et retourner au point 5.
- Projeter la face P_1 dans le plan de E (***Figure 4.5***). Cette projection est une 2-G-carte appelée $2G^{E}_{P1}$. Associer à cette 2-G-carte la face P_1 de la 3-G-carte $3G_{objets}$.
- Découper cette projection suivant les contours intérieurs et extérieurs (les bords) de la 2-G-carte $2G^{E}_{objets}$ et suivant le contour de l'écran.
- Supprimer la face P_1 de L.
- Si L n'est pas vide, retourner en 5.

Une fois toutes les faces de L visitées, la 2-G-carte $2G^{E}_{objets}$ décrit le contour de toutes les faces visibles (***Figure 4.6***) depuis l'oeil.

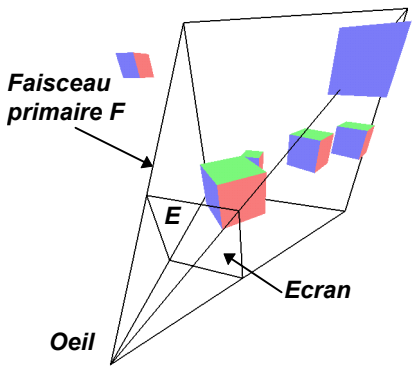


Figure 4.4 : Construction du faisceau primaire.

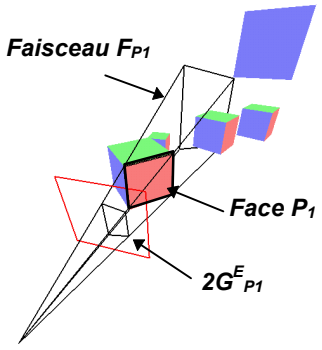


Figure 4.5 : Construction du premier faisceau F_{P1} et d'une partie de la 2-G-carte $2G^{E}$.

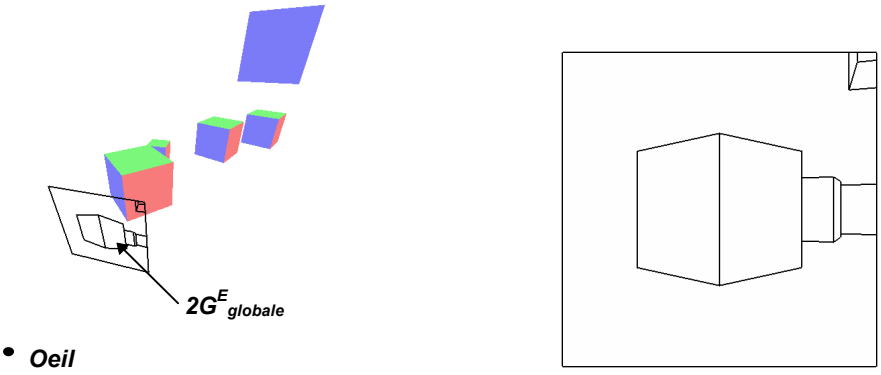


Figure 4.6 : La 2-G-carte $2G^E_{objets}$ décrit les contours des objets visibles (A droite : la carte dans l'espace ainsi que les objets, à gauche : la carte dans le plan de l'écran).

Reprenons à présent cet algorithme en justifiant et en détaillant les différentes étapes.

Etape 1 :

La première étape initialise uniquement la 2-G-carte $2G^E_{objets}$ à la carte vide. Cette carte est complétée au fur et à mesure du déroulement de l'algorithme par les parties visibles des faces. Une fois l'algorithme terminé, elle décrit tous les contours des faces visibles depuis l'oeil dans l'écran E.

Etape 2 :

Un faisceau primaire est construit de la même manière que dans le lancer de faisceaux pyramidaux (***Paragraphe 1.3.2.2***) à la seule différence qu'il est toujours fermé par un plan virtuel. Ce plan représente le fond de la scène, sa construction est décrite au ***Paragraphe 3.1.2, Figure 3.4***.

Etape 3 :

On construit la liste des faces susceptibles d'être partiellement ou entièrement dans le faisceau primaire. Pour cela, on utilise une subdivision spatiale régulière et la méthode décrite au ***Paragraphe 3.1.2*** parcourant les voxels d'un faisceau. Les objets associés à ces voxels sont alors positionnés par rapport au faisceau. Pour cela, on cherche un plan vérifiant la ***Règle 1***[20] annoncé au ***Paragraphe 2.1.1.1***. Les plans testés sont les mêmes que ceux utilisés lors de l'optimisation du lancer de faisceaux (***Paragraphe 3.1.2***). On exclut de cette liste toutes les faces dont la normale ne pointe pas du côté de l'oeil car ces faces ne peuvent pas être visibles.

Etape 4 :

On effectue un tri rapide ("*quick sort*") des faces en profondeur. Pour cela, on considère la face et son sommet le plus proche de l'oeil. La première face de la liste est celle qui a le sommet le plus proche de l'oeil.

Etape 5 :

On construit un faisceau F_{P1} qui a pour sommet l'oeil, qui traverse exactement la face P_1 et qui est fermé par un plan virtuel simulant le fond de la scène (***Figure 4.4***). La construction de ce plan est décrite au ***Paragraphe 3.1.2***. Ce faisceau permettra de déterminer si la face P_1 est visible et de marquer les faces cachées par P_1.

---

[20] Rappel : **Règle 1**: s'il existe un plan qui sépare l'espace en deux demi-espaces de sorte que l'un contienne la totalité du faisceau et l'autre la totalité de l'objet, alors l'objet est entièrement à l'extérieur du faisceau.

Etape 6 :

On détermine précisément la position de tous les objets de la liste L par rapport au faisceau F_{Pl}. Pour cela, on utilise les mêmes techniques que celles développées dans le chapitre précédent. On se base sur la subdivision spatiale régulière pour parcourir les voxels du faisceau. On parcourt les listes des faces associées à ces voxels pour déterminer leur position respective. A la différence des faisceaux utilisés au chapitre précédent, ceux utilisés ici sont divisés en deux : la partie avant comprise entre l'écran et la face P_1 et la partie arrière comprise entre la face P_1 et le plan virtuel (***Figure 4.7***). On détermine en fait la position de chaque face par rapport au faisceau F_{P1} puis par rapport à ces deux parties avant et arrière. Pour cela, on construit des plans tangents aux différentes arêtes du faisceau et des faces pour trouver un plan permettant de valider la ***Règle 1*** vue au ***Paragraphe 3.1.3.2***. Davantage de détails sur le choix de ces plans sont proposés au ***Paragraphe 4.2.1.4***. On peut ainsi différencier, entre autres, les faces entièrement contenues dans la partie avant du faisceau (***Figure 4.7***, faces de l'objet 1), celles entièrement contenues dans la partie arrière (faces de l'objet 2), celles partiellement dans la partie avant du faisceau (faces de l'objet 3) et celles partiellement contenues dans la partie arrière du faisceau (faces de l'objet 4). Les faces entièrement contenues dans la partie arrière du faisceau sont supprimées de la liste. En effet, elles ne peuvent être visibles car cachées par la face P_1.

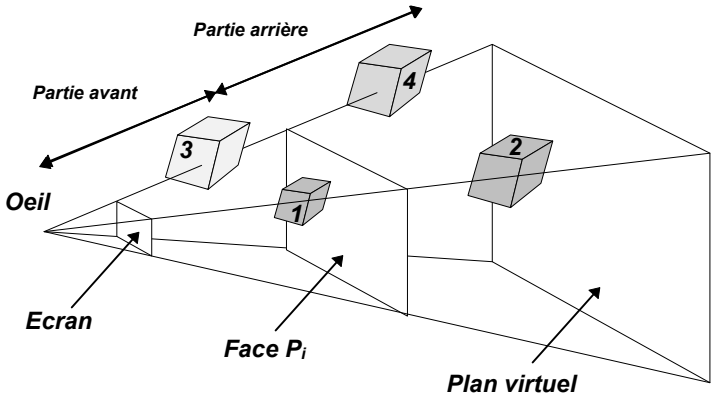
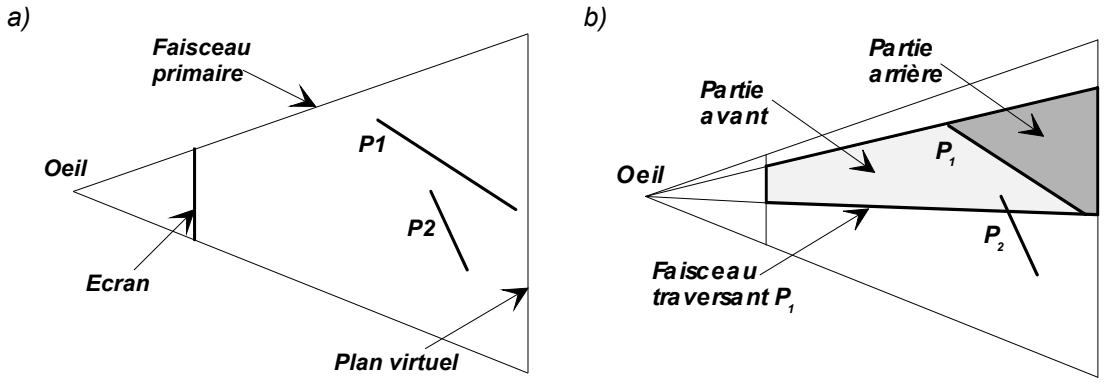


Figure 4.7 : Position des objets par rapport à un faisceau traversant exactement une face P_i.

Etape 7 :

Dans cette étape, on vérifie que la face P_1 choisie est bien la première face visible dans L. Le tri en profondeur augmente les chances pour que ceci soit le cas, mais rien ne le prouve. Prenons l'exemple la ***Figure 4.8.a***, le tri en profondeur ordonne les faces en mettant en premier P_1 puis P_2, or P_2 cache P_1. Pour reconnaître ce cas de figure, il suffit de regarder si des faces ont été marquées comme étant partiellement ou entièrement dans la partie avant du faisceau (***Figure 4.8.b***). Notons que pour que notre solution soit exacte, il est indispensable de pouvoir déterminer avec précision la position des faces par rapport à la partie avant du faisceau (voir **Paragraphe 4.2.1.4**). Si des faces ont été marquées, une face P_2 cache P_1, on inverse l'ordre des deux faces. On effectue ainsi des permutations jusqu'à trouver une face qui n'est cachée par aucune autre. Une telle face existe puisque, par hypothèse, on ne considère ni les faces auto-intersectantes ni les faces se superposant cycliquement.



Faisceau primaire et faces triées en profondeur.

Faisceau traversant la face P_1 et la face P_2 située dans la partie avant du faisceau.

Figure 4.8 : Le tri en profondeur ne respecte pas l'ordre dans lequel les faces sont visibles.

Etape 8 :

Dans cette étape, on traite la face P_1 en la projetant sur le plan de l'écran. On utilise pour cela une simple projection perspective centrée en l'oeil. Cette projection permet de construire, sur le plan de l'écran, une 2-G-carte de la totalité du contour de la face visible. Notons néanmoins que cette projection peut poser problème. Considérons la ***Figure 4.9.a***, la projection de la face P_1 sur l'écran se détermine simplement en calculant les intersections des droites passant par l'oeil et les sommets de P_1 avec l'écran. Si on utilise cette méthode de calcul dans la ***Figure 4.9.b***, le résultat obtenu n'est pas celui escompté et l'on ne projette pas la partie visible de P_1 sur l'écran. Ce problème se pose pour toutes les faces dont une partie au moins est derrière un plan parallèle à l'écran passant par l'oeil (***Figure 4.9.c***). Pour résoudre ce problème, il suffit de découper toutes ces faces selon le plan de l'écran et de ne conserver que la partie devant l'écran (***Figure 4.9.d***).

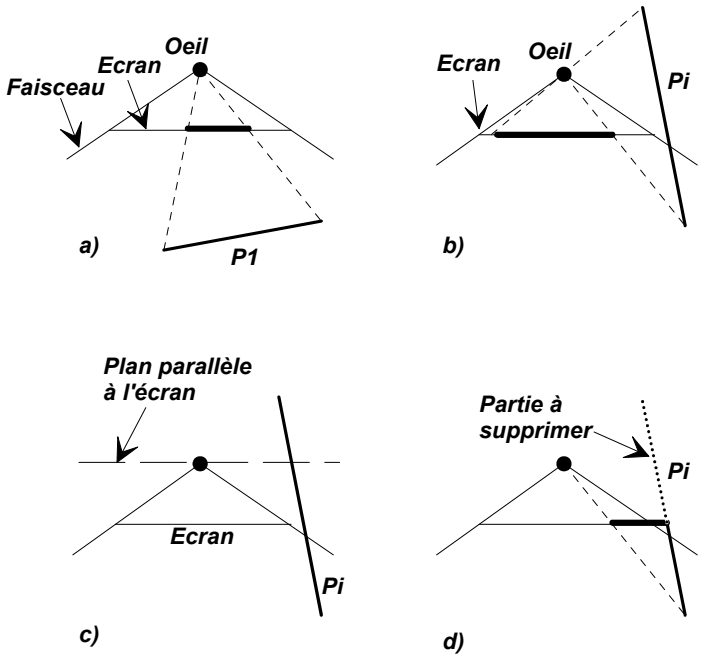


Figure 4.9 : Face devant être découpée avant la projection.

Etape 9 :

Après plusieurs itérations de l'algorithme, la 2-G-carte $2G^E_{objets}$ décrit les contours des objets visibles déjà traités. Il s'agit ici de découper la 2-G-carte $2G^E_{Pi}$ des contours de la face courante suivant les bords intérieurs et extérieurs de la 2-G-carte $2G^E_{objets}$ puis suivant les bords de l'écran. Notons que la 2-G-carte $2G^E_{objets}$ peut avoir une forme quelconque, concave et surtout trouée et qu'il faut donc considérer tous les bords. Pour cela, on utilise un algorithme de découpage d'un polygone quelconque par un autre polygone quelconque. Cet algorithme est fondé sur les 2G-cartes et est inspiré de l'algorithme de Weiler-Atherton [WEIL 77]. De plus, il exploite avantageusement les propriétés des 2-G-cartes en profitant de la connaissance des bords visibles des faces déjà traitées. Cet algorithme de découpage de polygones, fondé sur les 2-G-cartes a été développé au sein de l'équipe modélisation. Il ne sera donc pas présenté ici mais fera, on l'espère, l'objet d'une publication ultérieure.

A chaque face ajoutée à la 2-G-carte $2G^E_{objets}$ on associe la face P_1 de la 3-G-carte $3G_{objets}$ correspondante. Ce lien servira par la suite pour les calculs des ombres portées, des réflexions, des réfractions et pour le calcul du rendu.

Etape 11 et 12 :

On supprime la première face de la liste L. Ceci permet de relancer, tant que L n'est pas vide, la procédure pour une autre face visible.

La ***Figure 4.10*** montre le résultat de cette première étape sur une scène représentant un immeuble. La partie droite de la figure correspond à la $2G^E_{objets}$ obtenue.

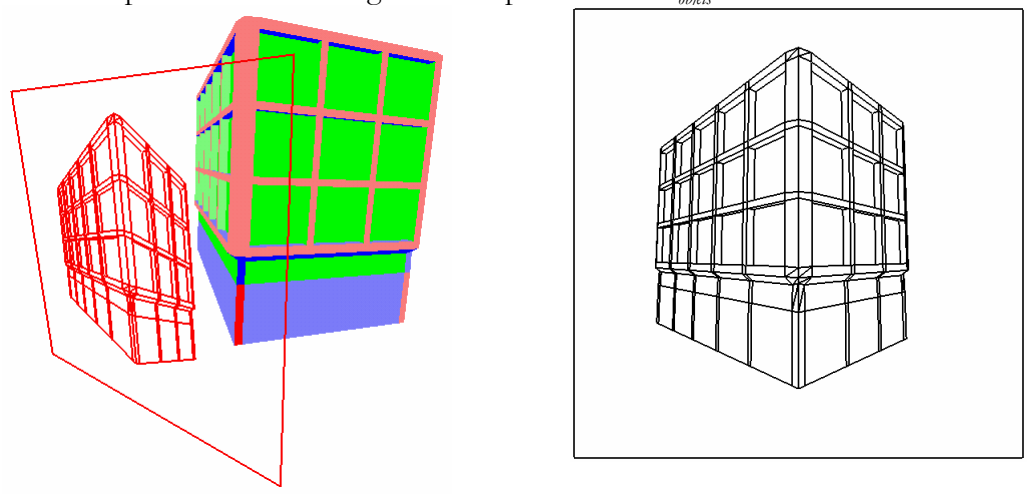


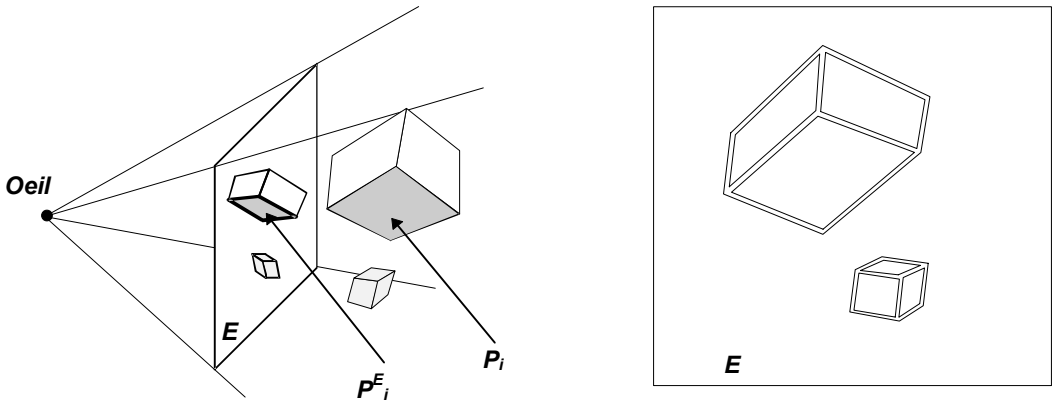
Figure 4.10 : La 2-G-carte $2G^E_{objets}$ décrit les contours visibles de l'immeuble (A gauche : la carte dans l'espace ainsi que l'immeuble, à droite : la carte dans le plan de l'écran).

4.2.1.2 Contour des ombres

Dans le paragraphe précédent, nous avons calculé la 2-G-carte des contours visibles des objets. Dans ce paragraphe, nous allons ajouter à cette description les contours des ombres portées. Pour cela, on construit sur chaque face visible une 2-G-carte représentant les ombres projetées sur cette face. Ces cartes sont ensuite ajoutées à la carte globale $2G^E_{objets}$. Les différentes étapes de cette méthode sont détaillées ci-dessous. Pour une meilleure clarté des figures, nous considérons une nouvelle scène (***Figure 4.11***).

Détails de la démarche utilisée :

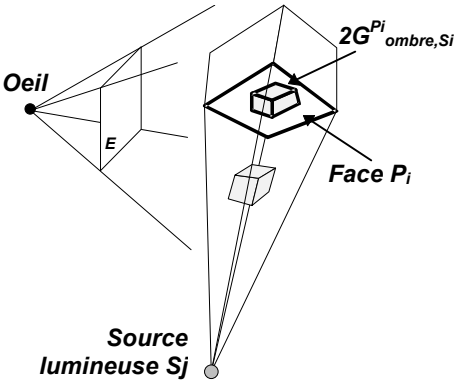
- Considérer, l'une après l'autre, toutes les faces P^E_i de la 2-G-carte $2G^E_{objets}$ (***Figure 4.11.b***).
- Pour chaque face P^E_i, considérer la face P_i de la 3-G-carte $3G_{objets}$ qui lui est associée à l'étape 9 (***Figure 4.11.a***) et une source lumineuse *Sj*. Construire un faisceau d'ombre semblable à ceux construits dans l'algorithme de lancer de faisceaux. Le sommet du faisceau correspond à la source lumineuse *Sj* et la base à la face P_i (***Figure 4.12.a***).
- Appliquer exactement le même algorithme de construction des contours que dans le paragraphe précédent en remplaçant l'oeil par la source lumineuse *Sj* et l'écran par la face P_i. On obtient ainsi la projection des contours des ombres des objets sur la face P_i (***Figure 4.12.a***). Cette projection est une 2-G-carte appelé $2G^{Pi}_{ombres,Sj}$.
- Construire une 2-G-carte $2G^{Pi}_{ombres,Sj}$ (***Figure 4.12.b***) pour chaque source lumineuse *j*.
- Corafiner l'ensemble de ces cartes pour constituer la 2-G-carte des contours de toutes les ombres portées sur la face P_i. Cette dernière carte est appelée $2G^{Pi}_{ombres}$ et est associée à la face P_i. Cette association permet de ne calculer cette carte qu'une seule fois, la première fois que l'on fait référence à la face P_i. En effet, dans les cas de réflexion et de réfraction, on peut avoir besoin plusieurs fois des mêmes ombres portées sur une face.
- Projeter la carte $2G^{Pi}_{ombres}$ dans le plan de l'écran
- Découper cette projection suivant le contour de la face P^E_i. On utilise, pour cela, le même algorithme de découpage que dans le cas des contours des objets.
- Corafiner cette découpe avec la 2-G-carte $2G^E_{objets}$. On construit ainsi la carte $2G^E_{objets,ombres}$ des contours visibles des objets et des ombres portées sur ces objets.



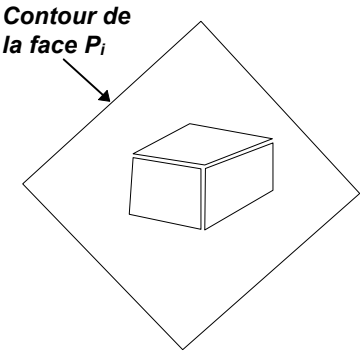
a) *Face P_i associée à une face P^E_i. de la 2-G-carte $2G^E_{Pi}$.*

b) *Face P^E_i formant la 2-G-carte $2G^E_{objets}$.*

Figure 4.11 : Représentation des cartes face P^E_i et P_i.

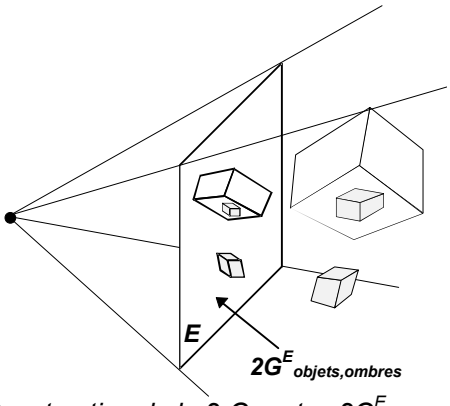


a) *Faisceau d'ombre.*

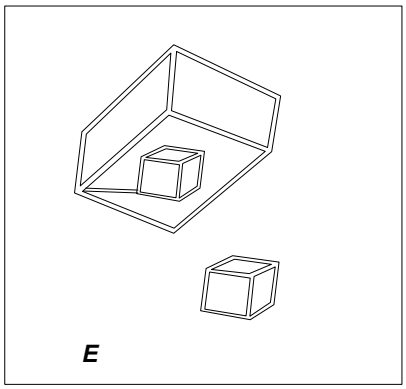


b) *Contours des ombres projetées sur la face P_i.*

Figure 4.12 : Faisceau d'ombre et 3-G-cartes des contours des ombres projetées sur P_i.



a) *Construction de la 2-G-cartes* $2G^E_{objets,ombres}$.



b) *2-G-cartes* $2G^E_{objets,ombres}$ *dans le plan de l'écran.*

Figure 4.13 : 2-G-cartes $2G^E_{objets,ombres}$ *des contours visibles des objets et des ombres portées.*

A titre d'exemple, on peut considérer à nouveau la scène du ***Paragraphe 4.2.1.1*** et visualiser sur la ***Figure 4.14*** le premier faisceau d'ombre lancé ainsi que les 2-G-cartes $2G^{Pi}_{ombres,Si}$ et $2G^E$. La ***Figure 4.15*** montre la 2-G-carte $2G^E_{objets}$ décrivant les contours et les ombres des objets.

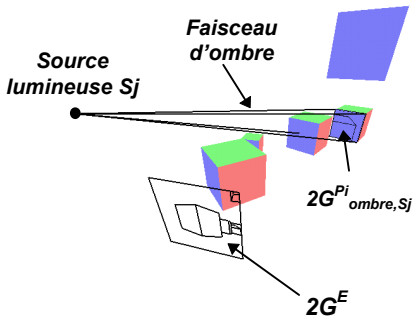


Figure 4.14 : Faisceau d'ombre issu de la source lumineuse Sj et 2-G-carte $2G^{Pi}_{ombres,si}$ *des ombres portées sur la face* P_i.

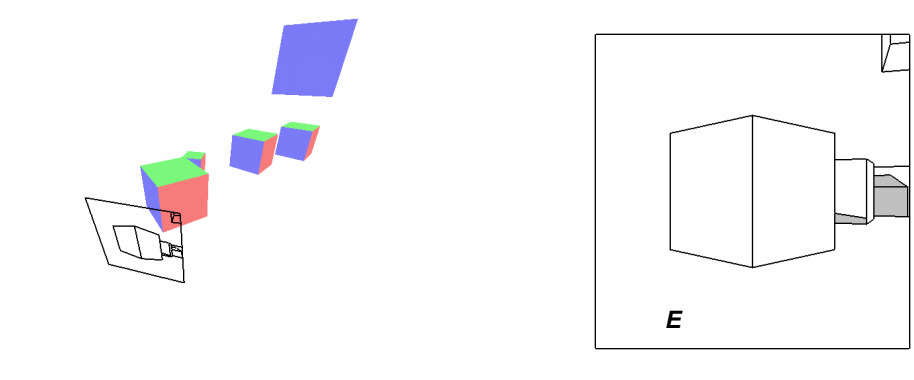


Figure 4.15 : La 2-G-carte $2G^{e}_{objets,ombres}$ décrit les contours des objets visibles et des ombres portées (A gauche : la carte dans l'espace ainsi que les objets, à droite : la carte dans le plan de l'écran).

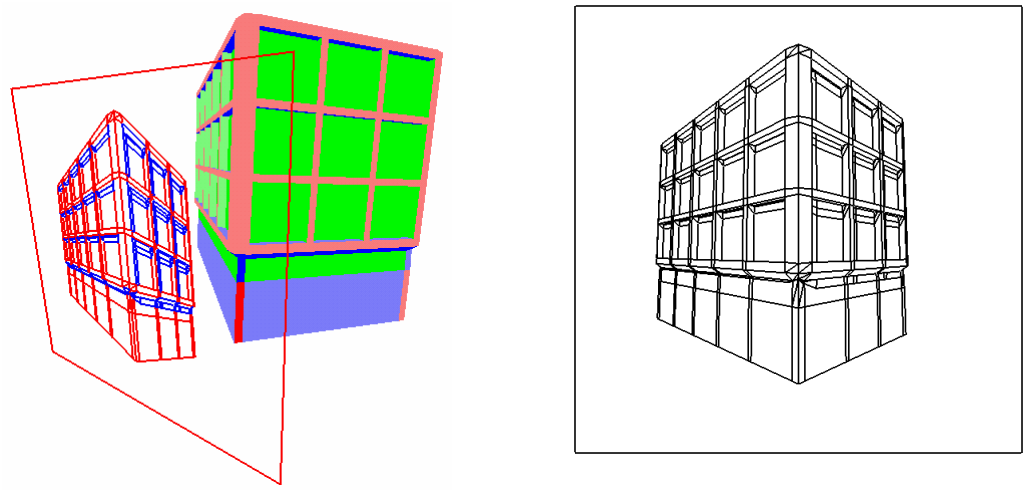


Figure 4.16 : La 2-G-carte $2G^{E}_{objets,ombres}$ décrit les contours visibles de l'immeuble ainsi que les ombres portées visibles. (A gauche : la carte dans l'espace ainsi que l'immeuble, à droite: la carte dans le plan de l'écran).

4.2.1.3 Effets de réflexion et de réfraction

Jusqu'à présent nous avons traité des objets opaques et non réfléchissants. Considérons à présent les objets réfléchissants et réfractants. Dans un premier temps, nous décrivons l'algorithme général permettant de traiter des réflexions et réfractions multiples. Puis, nous présentons la construction des faisceaux de réflexion et de réfraction.

4.2.1.3.1 Traitement des réflexions et réfractions multiples

L'approche utilisée pour traiter les réflexions et réfractions multiples est proche de celle du lancer de rayons. En effet, la structure de l'algorithme utilise le même type de récursion (***Algorithme 4-1***).

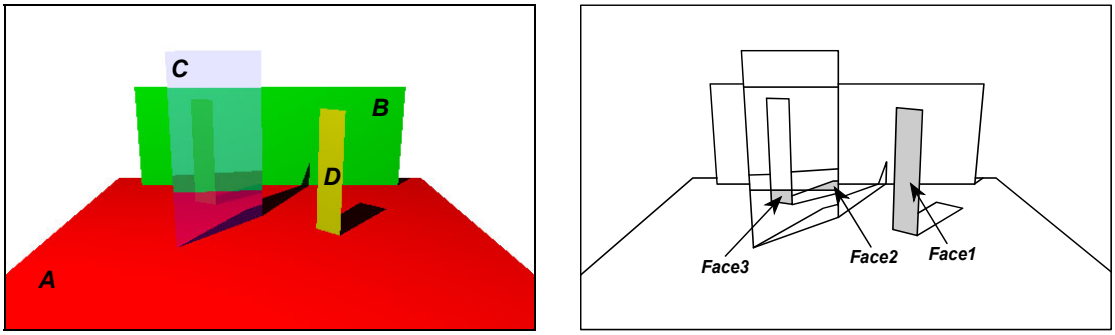
Un premier faisceau traversant exactement l'écran permet de détecter les faces visibles et de les trier en profondeur (***Paragraphe 4.2.1.1***). Sur chacune de ces faces visibles, on construit les cartes des ombres portées (voir ***Paragraphe 4.2.1.2***). Si la face est réfléchissante (resp. réfractante) on construit un faisceau réfléchi (resp. réfracté) pour déterminer, d'une manière semblable à celle du ***Paragraphe 4.2.1.1***., les contours des objets réfléchis (resp. réfractés). On dispose ainsi, sur chaque face des 2-G-cartes des contours des ombres, des réflexions et des réfractions. La construction de ces cartes suit une logique

récursive et est répétée pour les différents niveaux de réflexion et réfraction. Lorsque l'on a atteint une surface non réfléchissante et opaque, la carte de la face et celle des ombres qu'elle supporte sont projetées et corafinées au niveau inférieur. On construit ainsi, de proche en proche, la 2-G-carte finale de tous les contours. L'***Algorithme 4-1*** synthétise ce mécanisme.

```
// Cet algorithme est appelé avec pour paramètre F le faisceau issu de l'oeil et traversant
// exactement l'écran et pour paramètre P, l'écran. La carte retournée décrit tous les
// contours (objets, ombres, réflexions et réfractions).
2GCarte Visibilité(Faisceau F, Face P)
    // Initialisation de la carte finale de tous les contours
    Initialiser la 2-G-carte 2G^P située dans le plan P à la carte vide
    Construire la liste L des faces dans le faisceau F
    Trier L en profondeur
    Tant qu'il existe des faces dans L, considérer Pi la première d'entre elles
        Construire un faisceau F_Pi traversant exactement Pi
        Marquer la position exacte de chaque face de L par rapport à F_Pi
        Supprimer de L toutes les faces cachées par Pi (derrière P)
        si une face Pi' se trouve devant Pi inverser Pi' et Pi
        sinon        // ---------- Calcul des ombres portées ----------
            pour chaque source lumineuse Sj située du côté de la normale de P
            faire
                Construire un faisceau d'ombre F^P_ombres,Sj
                Appliquer exactement le même algorithme de construction des contours
                    que celui utilisé au Paragraphe 4.2.1.1 en remplaçant l'oeil par la source
                    lumineuse Sj et l'écran par la face P. On obtient une 2-G-carte décrivant
                    les ombres projetées sur cette face pour la source Sj. Cette projection est
                    appelée 2G^P_ombres,Sj.
            Corafiner l'ensemble des cartes 2G^P_ombres,Sj pour constituer la 2-G-carte des
                contours de toutes les ombres portées sur la face P. Cette dernière carte est
                appelée 2G^Pi_ombres.
            Associer la carte 2G^Pi_ombres à la face P.
            // ---------- Calcul des réflexions ----------
            Initialiser la carte 2G^Pi_réflexion à la carte vide
            si Pi est une face réfléchissante et que l'on a pas atteint la limite de profondeur
                de la récursion
                Construire un faisceau réfléchissant F^Pi_réflexion
                Construire la cartes de réflexion 2G^Pi_réflexion=Visibilité(F^Pi_réflexion,Pi)
            // ---------- Calcul des réfractions ----------
            Initialiser la carte 2G^Pi_réfraction à la carte vide
            si Pi est une face réfractante réfléchissante et que l'on a pas atteint la limite de
                profondeur de la récursion
                Construire un faisceau réfractant F^Pi_réfraction
                Construire la cartes de réfraction 2G^Pi_réfraction=Visibilité(F^Pi_réfraction,Pi)
            Corafiner les cartes suivantes : la 2-G-carte correspondant à la face Pi,
                2G^Pi_ombres, 2G^Pi_réflexion, 2G^Pi_réfraction, pour constituer la 2-G-carte
                2G^Pi_objet,ombres,réflexion,réfraction
            Projeter cette carte sur la face P
            Découper cette projection selon les bords de la carte 2G^P
            Corafiner cette découpe avec la carte 2G^P
            Supprimer Pi de L
    retourner la 2-G-carte 2G^P
```

Algorithme 4-1 : Calcul de la carte de visibilité décrivant tous les contours (objets, ombres, réflexions, réfractions).

Lors du rendu, nous aurons besoin de connaître avec précision les différentes faces des objets de la 3-G-carte $3G_{objet}$ intervenant dans la 2-G-carte finale des contours. Pour cela, à chaque coraffinement des ombres, réflexions et réfractions, on associe aux faces issues du coraffinement les faces réfléchies et réfractées. On retient aussi la quantité de lumière atteignant la face, cette valeur est comprise entre 0 et 1. Prenons l'exemple de la ***Figure 4.17*** et la *Face1* de la 2-G-carte finale des contours. Aucune ombre n'est portée sur cette face et elle n'est ni réfléchissante, ni réfractante, il n'y a donc aucune association et l'on retient que toute la lumière atteint cette face (par convention on utilisera la valeur '1' pour traduire cela). Une notation traduisant cela est proposée ***Figure 4.18***. Dans le cas de la *Face2*, aucune ombre n'est portée directement sur cette face, elle est réfléchissante et permet de voir une partie à l'ombre de l'objet *A*, elle est aussi réfractante et permet de voir une partie éclairée de l'objet *B*. L'association est schématisée ***Figure 4.18***. Dans le cas de la *Face3*, aucune ombre n'est portée directement sur cette face, elle est réfléchissante et permet de voir une partie éclairée de l'objet *D*, elle est aussi réfractante et permet de voir une partie éclairée de l'objet *A*. L'association est schématisée ***Figure 4.18***.



Scène composée de 4 objets A, B, C et D. *2-G-carte finale des contours.*

Figure 4.17 : Effets d'ombres, de réflexion et de réfraction.

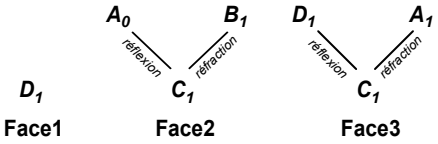


Figure 4.18 : Représentation des associations faites avec différentes faces.

4.2.1.3.2 Contours des réflexions

Pour construire la carte des objets réfléchis sur une surface *P* on utilise le même algorithme que pour les contours des objets (***Paragraphe 4.2.1.1***). Nous remplaçons ici le faisceau primaire par un *faisceau de réflexion* et l'écran par la face réfléchissante. Ce faisceau de réflexion est semblable à celui utilisé dans l'algorithme de lancer de faisceaux de [GHAZ 92]. Le sommet du faisceau correspond à l'oeil virtuel et la base correspond à la face réfléchissante *P* (***Figure 4.19***).

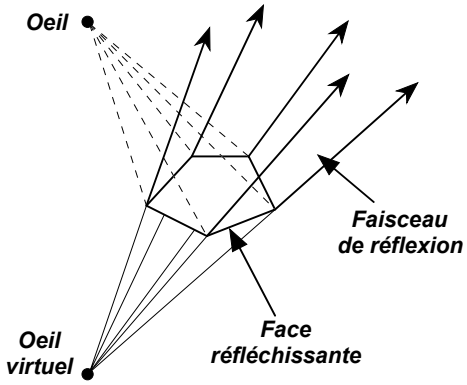


Figure 4.19 : Faisceau de réflexion.

4.2.1.3.3 Contours des réfractions

La prise en compte des effets de réfraction est plus problématique. On a vu au ***Paragraphe 1.2.2.4*** que la réfraction n'est pas une transformation linéaire. Or notre approche est fondée sur des contours décrivant, à l'aide de 2-G-carte, des segments de droite. La solution approximative que nous proposons est la même que celle de [HECK 84]. Elle consiste à utiliser un oeil virtuel pour se ramener à un traitement linéaire. Pour cela, on construit un oeil virtuel situé sur la perpendiculaire à la face réfractante passant par l'oeil (***Figure 4.20***). Sa distance au plan de la face est fonction de l'indice de réfraction de la face considérée (voir annexe de [HECK 84]). Cette solution a le mérite d'être simple et d'être semblable aux effets de réflexion. Malheureusement, on perd les effets de déformation des droites ce qui nuit au réalisme. En effet, une face polygonale vue à travers une surface réfractante reste une face polygonale. Ainsi, le damier de la ***Figure 1.25*** comportera des cases en forme de trapèze ce qui ne trompera personne ! Notons néanmoins qu'une solution intégrant les déformations dues aux réfractions semble possible. Pour cela, nous proposons de changer de modèle de plongement. En particulier, nous voudrions associer une courbe à une arête des 2-G-cartes de réfraction. Mais cette approche n'en est encore qu'à ses balbutiements.

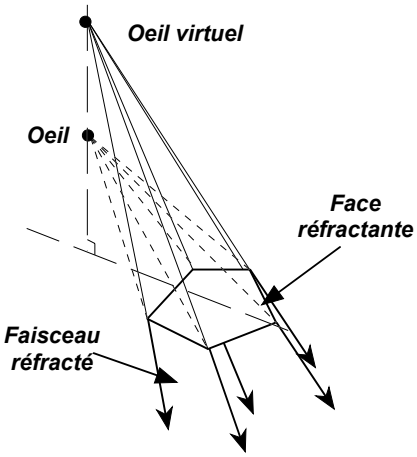


Figure 4.20 : Faisceau réfractant.

4.2.1.4 Position exacte d'une face par rapport à un faisceau

Dans le ***Paragraphe 3.1.3.2***, nous avons cherché des plans susceptibles de diviser l'espace en deux demi-espaces de sorte que le faisceau soit d'un côté et que le polyèdre convexe soit de l'autre. Si parmi les plans testés, l'un d'eux vérifiait cette règle, nous

pouvions affirmer que le faisceau et les objets étaient disjoints. Par contre, si aucun des plans testés ne vérifiait cette règle nous ne pouvions rien conclure.

Dans le cas de l'algorithme présenté dans ce chapitre, nous devons disposer d'un moyen simple pour déterminer, avec précision, la position d'une face (polygone) convexe par rapport à un faisceau (c'est-à-dire un polyèdre convexe). Pour cela, nous définissons la propriété $\mathcal{P}$ relative à un plan Π suivante:

Propriété $\mathcal{P}$ *: le plan* Π *partage l'espace en deux demi-espaces tels qu'un polyèdre est inclu entièrement dans l'un de ces demi-espaces et qu'un polygone est inclu entièrement dans l'autre demi-espace.*

L'idée pour réaliser cet algorithme est de trouver un ensemble fini de plans tel que si aucun d'eux ne vérifie la propriété $\mathcal{P}$, on puisse conclure qu'il y a intersection entre le polyèdre et le polygone. L'ensemble que nous proposons est constitué de tous les plans construits à partir de trois points pris parmi les sommets du faisceau et les sommets du polygone convexe. Nous allons montrer à présent que si aucun de ces plans ne vérifie la propriété alors il y a intersection entre le polyèdre convexe et le polygone convexe. Enonçons tout d'abord une règle simple [UNIV 90] :

Règle 2 *: Si un polyèdre convexe et un polygone convexe sont disjoints alors il existe un plan divisant l'espace en deux demi-espaces, l'un contenant le polyèdre et l'autre contenant le polygone.*

Démontrons le théorème suivant :

Théorème *: Soit* V *un polyèdre convexe, soit* P *un polygone convexe et soit* S *l'ensemble des sommets de* V *et* P*. Si* V *et* P *n'ont pas de points en commun alors il existe un plan* Π *contenant trois points distincts non alignés de* S *et partageant l'espace en deux demi-espaces* E1 *et* E2 *tel que* V⊂E1 *et* P⊂E2*.*

Démonstration

Soit V un polyèdre convexe et soit P un polygone convexe.

D'après la ***Règle 2***, il existe un plan Π tel que $V \cap \Pi$ et $P \cap \Pi$ soient vides et tel que Π vérifie $\mathcal{P}$.

En translatant Π vers V on obtient un plan limite Π' tel que Π' vérifie $\mathcal{P}$ et tel qu'il existe un point M appartenant à $V \cap \Pi'$.

Puis, en considérant une rotation $\mathcal{R}_1$ d'angle α_1 dont l'axe contient M et est contenu dans Π' et en considérant Π'' l'image de Π' par la rotation $\mathcal{R}_1$, il existe un angle α_1 limite tel que $\mathcal{P}$ soit encore vraie et Π'' contient M et un autre point de S appelé N.

Enfin, en considérant la rotation $\mathcal{R}_2$ d'angle α_2 d'axe MN, il existe un angle α_2 limite tel que $\Pi''' = \mathcal{R}_2(\Pi'')$ contienne un autre point de S non aligné avec M et N.

Donc, si P et V sont disjoints, il existe un plan contenant trois points non alignés de S tel que $\mathcal{P}$ soit vérifiée.

***Théorème contraposé** : Si tous les plans construits avec trois points non alignés de* S *ne vérifient pas* $\mathcal{P}$ *alors* P *et* V *ne sont pas disjoints, autrement dit, si aucun des plans ne vérifie* $\mathcal{P}$*,* P *et* V *ont au moins un point en commun.*

Trouver la position exacte d'une face P par rapport à un faisceau F est à présent simple. On considère, l'un après l'autre, tous les plans que l'on peut construire à partir de trois sommets appartenant au faisceau ou au polyèdre. Si un de ces plans vérifie la ***Règle 2*** alors la face P est entièrement à l'extérieur du faisceau F. Si aucun de ces plans ne vérifie la ***Règle 2***, P est entièrement ou partiellement à l'intérieur de F.

4.2.2 Rendu de la 2-G-carte finale

La première partie de l'algorithme a été consacrée à la construction de tous les contours des objets, des ombres portées, des réflexions et des réfractions. On dispose donc d'une 2-G-carte constituée d'un ensemble de faces, chacune représentant une région non ambiguë (***Paragraphe 4.2.1.3.1***). En effet, on est sûr qu'il n'y a aucun problème d'aliassage dû à des arêtes à l'intérieur de ces régions. Rappelons que pour toutes les 2-G-cartes construites sur l'écran ou sur une face de la 3-G-carte $3G_{objet}$ due aux ombres, réflexions ou réfractions on a associé à chacune de ces faces les objets qui l'ont engendré (***Paragraphe 4.2.1.3.1***). On peut ainsi, pour chaque région uniforme, retrouver dans l'ordre les objets qui interviennent dans le calcul de la couleur de cette face.

4.2.2.1 Principes généraux

Le calcul du rendu de chacune des faces de la 2-G-carte $2G^{E}_{objets}$ de tous les contours sur l'écran peut s'effectuer de différentes manières. La première consiste à appliquer un algorithme de remplissage de polygones 2D sur chacune des faces de la 2-G-carte $2G^{E}_{objets}$. Pour pouvoir lisser les surfaces, il est néanmoins nécessaire de retenir, pour chaque sommet de chaque face de la 2-G-carte $2G^{E}_{objets}$ les normales des faces correspondantes dans la $3G_{objet}$. Le rendu final utilise alors la couleur et les normales des objets associés aux différentes faces. Une autre solution néanmoins très proche est l'utilisation d'un tampon de profondeur ("*z-buffer*") qui a l'avantage d'être rapide.

La troisième solution consiste à utiliser un lancer de rayons conventionnel. Cette solution est rapide puisque l'on connaît par avance les objets que va rencontrer chaque rayon. En effet, pour chacune des faces de la 2-G-carte $2G^{E}_{objets}$ est associée l'arborescence des diverses réflexions et réfractions et ombres portées. Aucun test d'appartenance d'un point à un polygone n'est donc nécessaire. La ***Figure 4.21*** propose une arborescence des réflexions, réfractions et ombres portées pour une face de la 2-G-carte $2G^{E}_{objets}$.

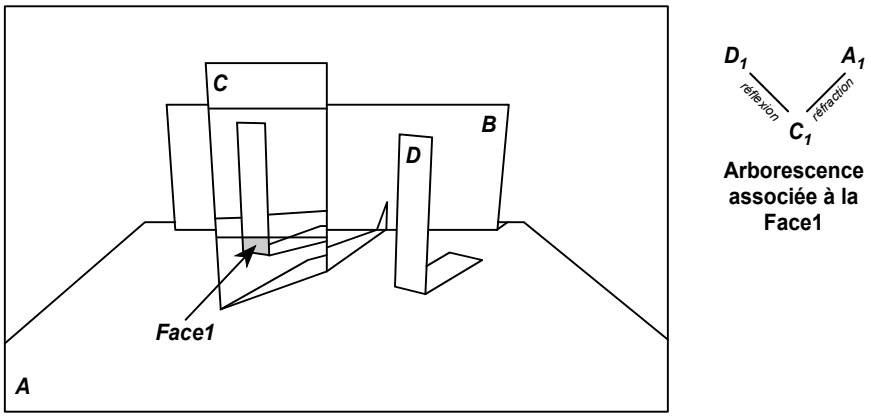


Figure 4.21 : A chacune des faces de la 2-G-carte $2G^{E}_{objets}$ est associée l'arborescence des diverses réflexions et réfractions et ombres portées

Cette approche fondée sur un lancer de rayons permet qui plus est de tenir compte de la position des sources lumineuses dans le calcul de la couleur. On peut donc réaliser un lissage des surfaces.

4.2.2.2 Traitement des problèmes d'aliassage

Quelle que soit l'approche générale utilisée, elle permet un traitement efficace des problèmes d'aliassage. D'une part, il n'y a plus de disparition de petits objets ou petites ombres puisque le problème de visibilité est résolu dans l'espace objet. D'autre part, l'antialiassage de contours peut être effectué de manière optimale. Pour cela, nous utilisons les propriétés des 2-G-cartes et en particulier les relations d'adjacence entre les faces. Considérons la 2-G-carte placée sur la grille de pixels de l'écran de la ***Figure 4.22***. Le rendu des pixels entièrement dans une face ne pose pas de problème, en effet, il ne risque pas d'y avoir de problème de marches d'escalier. Les seuls risques sont les moirés, ceux-ci seront traités avec une des techniques habituelles de filtrage a priori comme [GANG 84]. Le rendu des pixels communs à plusieurs faces (***Figure 4.22***) est plus délicat mais sans problème théorique. En effet, les relations d'adjacence entre les faces fournies par les cartes, permettent, pour chaque pixel, de connaître avec précision les surfaces des pixels occupées par les différentes faces. Une simple pondération en fonction de ces surfaces permet alors de traiter efficacement les problèmes de marches d'escalier.

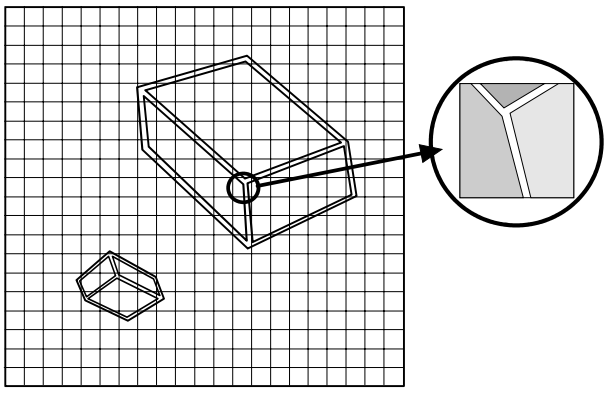


Figure 4.22 : Antialiassage au niveau d'un pixel.

4.3 CONCLUSION

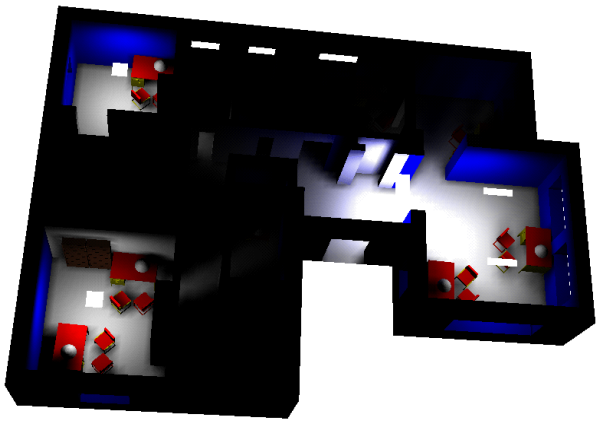
Ce chapitre a été consacré à un algorithme de visibilité travaillant dans l'espace objet. Il permet, à partir d'une scène polygonale, de construire une 2-G-carte sur l'écran représentant tous les contours des ombres portées, des réflexions et des réfractions. Pour réaliser cette carte, nous avons, comme dans le lancer de faisceaux de [GHAZ 92], utilisé des faisceaux pour éliminer rapidement les objets clairement non visibles. Nous avons aussi utilisé les faisceaux dans l'algorithme de découpage pour connaître la position exacte d'une face par rapport à une autre, afin de les traiter dans le bon ordre.

Travailler dans l'espace objet permet de détecter tous les petits objets et toutes les petites ombres, il n'y a donc plus de disparition de ces objets. De plus, la représentation des contours sous forme de 2-G-carte permet un traitement efficace des marches d'escalier. Pour cela, on a utilisé les relations d'adjacence des faces intrinsèques au modèle utilisé.

Cette représentation a d'autres avantages. Contrairement aux algorithmes travaillant dans l'espace image, cette représentation permet des agrandissements de l'image finale sans perte de qualité (dans la limite de la précision du calcul flottant). De plus, ces agrandissements peuvent être produits sans nouveau calcul de visibilité.

Diverses extensions sont envisageables et prévues dans la suite de nos travaux. Tout d'abord, on peut profiter de la modélisation de la scène sous forme de 3-G-carte et donc de la connaissance de la topologie de la scène, pour améliorer le positionnement des objets par rapport aux faisceaux. On peut ensuite envisager d'utiliser un plongement non linéaire pour produire des effets de réfraction plus réalistes. On peut aussi concevoir un algorithmes de déformation ou d'animation exploitant la représentation des contours sous la forme de 2-G-cartes. On pourrait ainsi calculer les images clefs d'une animation à l'aide de l'algorithme présenté ici, puis interpoler les images manquantes.

5. LANCER DE FAISCEAUX POUR OPTIMISER LE CALCUL DE LA VISIBILITE



On constate depuis les débuts de la synthèse d'images que les scènes développées sont de plus en plus complexes et contenant toujours plus d'objets. Parallèlement à cette évolution, les algorithmes de visualisation et de rendu se sont perfectionnés. Très rapidement, on a cherché à regrouper les objets suivant différents critères afin d'optimiser les traitements. Les approches les plus anciennes et les plus connues sont les volumes englobants [RUBI 80] [KAY 86] [GOLD 87] …, les subdivisions spatiales régulières [FUJI 86] [AMAN 87] [CLEA 88] … (voir ***Paragraphe 3.1.1****) et les subdivisions spatiales hiérarchiques ("octree") [GLAS 84] [SAME 89] [SPAC 91] … D'autres approches dédiées plus précisément à la radiosité ont été développées. On peut penser par exemple aux algorithmes de [MARK 90] et [HAIN 91] essayant de déterminer la visibilité entre un objet émetteur et un objet récepteur, au tampon de profondeur ("z-buffer") hiérarchique [GREE 93], aux notions de cellules et de portails introduites par [TELL 94] et [LUEB 95]. Enfin, les recherches les plus récentes concernent l'animation et plus particulièrement le déplacement au sein de mondes virtuels. Dans ce domaine, on peut citer quatre techniques :*

- *La première consiste à disposer de plusieurs descriptions du même objet avec différentes résolutions [FUNK 93]. Ceci permet de choisir dynamiquement la meilleure résolution en fonction de la taille apparente de l'objet.*
- *La deuxième utilise des imposteurs pour remplacer des ensembles d'objets éloignés du point de vue par une texture de même apparence [SHAD 96] [SILL 97]. On diminue ainsi le nombre de polygones à traiter en utilisant les algorithmes de placage de textures câblés sur la plupart des stations graphiques.*
- *La troisième solution consiste à "plonger" la scène dans une subdivision spatiale régulière (grille de voxels) et à construire, pour chaque voxel, la liste des voxels qui lui sont visibles [YAGE 95].*

Cette approche est lourde et consomme trop de mémoire pour être utilisable.

- *La dernière solution est celle qui a inspiré une partie de nos travaux dans le cadre du lancer de faisceaux présenté dans ce chapitre. Elle cherche à éliminer grossièrement les parties de la scène non visibles du point de vue parce que cachées par quelques objets imposants. On parle d'objets occultants ou d'occulteurs [COOR 96] [COOR 97] [HUDS 97].*

Le premier paragraphe de ce chapitre présente les trois approches les plus intéressantes pour nos travaux : déterminer la visibilité entre deux objets [MARK 90] [HAIN 91], la notion de cellule et de portail [TELL 94] [LUEB 95] et celle d'occulteur [COOR 96] [COOR 96] [HUDS 97]. Le paragraphe suivant est consacré à notre solution fondée sur les faisceaux.

5.1 DETAILS DE TROIS APPROCHES

5.1.1 Visibilité entre deux objets

Dans le calcul de la radiosité, la majorité du temps est passée dans l'évaluation du facteur de forme et en particulier dans celui de la visibilité entre deux carreaux. La première solution pour optimiser ce calcul est proposée par [MARK 90] (voir ***Paragraphe 1.3.1.5***). Cependant elle semble coûteuse et reste à l'état d'extension. La solution la plus connue dans ce domaine est certainement le volume défini par [HAIN 91] ("*shaft culling*"). Le principe est simple : on cherche à construire la liste des objets susceptibles de se trouver entre l'objet émetteur et l'objet récepteur. Les objets de cette liste sont les seuls utilisés pour évaluer la proportion de surface du récepteur visible depuis l'émetteur. Cette évaluation est effectuée à l'aide de plusieurs lancer de rayons [WALL 89]. Dans un premier temps, on construit autour de l'émetteur et du récepteur (sphérique ou polyédrique) une boîte englobante parallèle aux trois axes principaux. On construit ensuite quatre plans de sorte à délimiter un volume englobant les boîtes des deux objets (***Figure 5.1***). La façon de construire ces plans est délicate et tient compte des différents cas particuliers (voir [HAIN 91]).

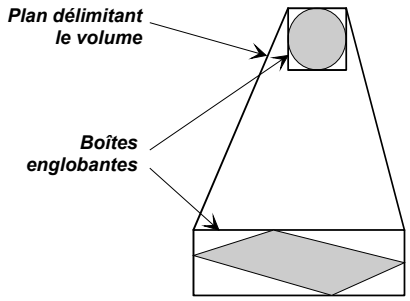


Figure 5.1 : Construction du volume entre un objet émetteur et un objet récepteur (d'après [HAIN 91]).

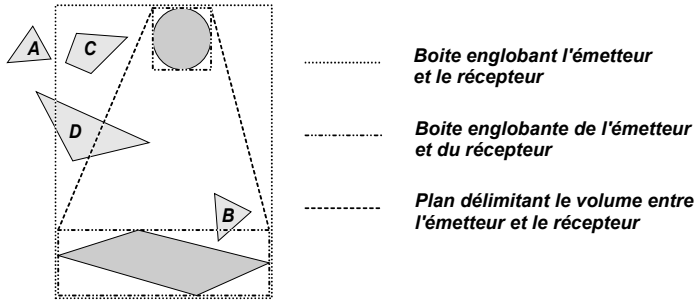


Figure 5.2 : Position d'un objet par rapport au volume (d'après [HAIN 91]).

Dans un deuxième temps, on détermine la liste des objets situés entre l'émetteur et le récepteur. Pour cela, on regarde si des objets sont situés à l'intérieur du volume constitué par les quatre plans et les faces extérieures des boîtes englobant récepteur et émetteur. Des tests simples et de plus en plus fins sont effectués. Chaque objet de la scène est tout d'abord positionné par rapport à la boîte englobant l'émetteur et le récepteur. Si l'objet est à l'extérieur de cette boîte, il ne peut être visible (***Figure 5.2, objet A***) et est écarté de la liste. Sinon, on positionne l'objet par rapport aux deux boîtes englobantes de l'émetteur et du récepteur, s'il touche entièrement ou partiellement l'une de ces boîtes, il peut être visible, il est donc ajouté à la liste (***Figure 5.2, objet B***). Sinon, on positionne l'objet par rapport aux quatre plans joignant les boîtes des deux objets (***Figure 5.2, objet C et D***) pour ajouter ou non l'objet dans la liste.

Si l'on utilise une hiérarchie de volumes englobants représentant la scène, il n'est pas nécessaire de positionner tous les objets par rapport aux différentes boîtes ou plans. En effet, on peut commencer par positionner le volume englobant au haut de la hiérarchie puis, si besoin est, passer à ces fils comme on le ferait dans un algorithme de lancer de rayons. Malgré cette optimisation, il n'est pas clair que le gain de temps soit important. En effet, [HAIN 91] propose d'utiliser cet algorithme pour tous les couples d'objets émetteur-récepteur, ceci risque donc de mettre en jeu un très grand nombre de couples et de

dégrader les performances. Pour limiter cet effet, [HAIN 91] propose d'utiliser une fonction de décision pour choisir dynamiquement d'utiliser ou non cet algorithme pour un couple donné. Malheureusement, l'auteur sous-entend que sa fonction choisit presque toujours d'utiliser l'algorithme de "*shaft culling*" ! D'autre part, remarquons que le volume construit autour des émetteur et récepteur est grossier, ceci principalement parce que l'on utilise des boîtes englobantes aux faces parallèles aux axes. Nous verrons au ***Paragraphe 5.2.3*** que ceci peut être amélioré.

5.1.2 La notion de cellule et de portail

[TELL 91] fait remarquer que dans le cas de scènes architecturales, de nombreux polygones ne sont pas visibles parce que cachés par des murs. Partant de ce constat et des travaux de [AIRE 90] dans le même domaine, [TELL 91] propose un prétraitement au rendu permettant de déterminer les objets mutuellement visibles dans un espace à deux dimensions. [TELL 94] présente l'extension à des scènes en trois dimensions. Il y représente la scène comme un ensemble de *cellules* correspondant à des polyèdres convexes (l'intérieur des pièces) adjacents par des polygones transparents (les portes) appelés *portails*. Les cellules sont construites sur la base d'arbres BSP[21] [FUCH 80]. Un graphe de visibilité entre cellules est construit de proche en proche en propageant la visibilité entre cellules. Deux cellules ne sont visibles qu'à travers un ou plusieurs portails. Considérons deux cellules mutuellement visibles à travers une série de portails. En général, seule une partie de chaque volume est visible depuis l'autre. En considérant deux par deux les cellules, il est possible de déterminer les volumes mutuellement visibles et donc les polygones mutuellement visibles. La ***Figure 5.3*** montre un exemple de propagation de la visibilité depuis un polygone *A* dans une cellule *S1* vers les polygones de la cellule *S2*.

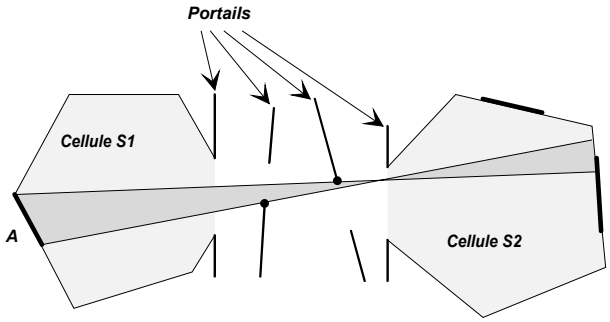


Figure 5.3 : Propagation de la visibilité.

Cette technique est complexe à mettre en oeuvre et fait intervenir des caractéristiques bien particulières de la scène. En effet, cette approche n'est possible que pour des scènes "bien composées" c'est-à-dire dans lesquelles les cellules et les portails sont clairement identifiables. Les principales scènes mises en jeu sont donc les scènes architecturales et particulièrement les intérieurs d'immeubles ou de maisons.

[LUEB 95] propose une variante de [TELL 94] qui n'utilise pas de prétraitement. Il projette les portails dans le plan de l'image pour déterminer les cellules visibles.

5.1.3 La notion d'occulteur

La notion d'*occulteur* ou d'objet cachant de nombreux autres apparaît principalement dans les algorithmes de réalité virtuelle lorsque l'on désire se déplacer dans des ensembles

[21] Dans le cas des subdivisions spatiales adaptatives (octrees), l'espace est récursivement divisé, si cela est nécessaire, par deux plans perpendiculaires, en huit sous-espaces parallélépipédiques de même dimension. Dans les cas des BSP (Binary Space-Partitioning Tree), l'espace est divisé par un seul plan en deux sous-espaces de dimension quelconque. Le plan est choisi judicieusement pour diviser les sous-espaces de manière équitable sans traverser aucun objet (voir [FUCH 80] [KAPL 87]).

architecturaux. Cette approche, bien que très intuitive, n'est étudiée que depuis quelques mois par [COOR 96], [COOR 97] et [HUDS 97]. Elle a pour but de diminuer le nombre d'objets à considérer lors du calcul du rendu avec un tampon de profondeur, ou un autre algorithme de rendu, en supprimant les objets qui sont clairement non visibles. Précisons bien que le but n'est pas de trouver exactement la liste des polygones visibles mais d'éliminer rapidement une partie des objets non visibles. Le principe utilisé est simple : pour un point de vue donné, trouver les polygones (les occulteurs) qui cachent entièrement un grand nombre d'objets (voir ***Paragraphe 5.1.3.1.3***), marquer ces objets comme "*non visibles*" et traiter uniquement les objets qui ne sont pas marqués (***Figure 5.4***). Notons que ces derniers ne sont pas forcément visibles. En effet, dans l'exemple de la ***Figure 5.4***, le polygone *b* est caché par le polygone *a* et n'est pas marqué "*non visible*". Par contre, tous ceux qui sont marqués sont cachés. Aucune erreur n'est donc possible.

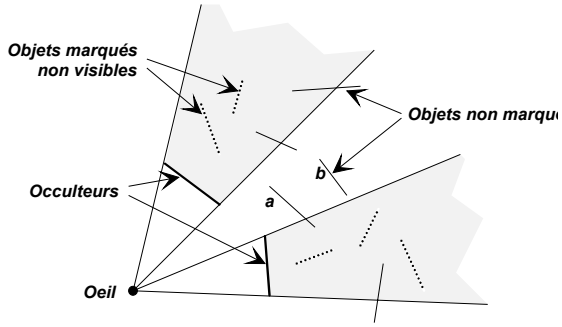


Figure 5.4 : Notion d'occulteur et d'objets non visibles.

5.1.3.1 L'approche de [COOR 97]

[COOR 97] se place dans le cadre d'une animation calculée en temps réel. Il propose d'utiliser les occulteurs pour diminuer le nombre de polygones à traiter. Il présente aussi une optimisation pour profiter de la cohérence de la scène d'une image sur l'autre. Cette dernière n'est pas présentée ici car elle sort du cadre de nos recherches. Le problème de [COOR 97] peut se décomposer en trois sous-problèmes :

1. Comment savoir qu'un objet est caché par un autre ;
2. Comment trouver un maximum d'objets cachés ;
3. Comment choisir de bons occulteurs.

5.1.3.1.1 Savoir si un objet est caché

Le premier problème peut s'énoncer de la manière suivante : *"étant donnés un point de vue, un ensemble d'occulteurs convexes et un objet convexe O, comment savoir si O est visible ?"* Pour répondre à cette question, on construit la boîte englobante *T* de l'objet *O*. La solution proposée met en oeuvre deux types de plans : les *plans séparateurs* et les *plans porteurs* (***Figure 5.5***). Un plan séparateur d'une boîte englobante et d'un occulteur polygonal convexe est tangent à une arête de la boîte et à un sommet de l'occulteur de sorte que chacun des objets soit du côté opposé du plan à l'autre. Un plan porteur est semblable sauf que les deux objets sont du même côté du plan.

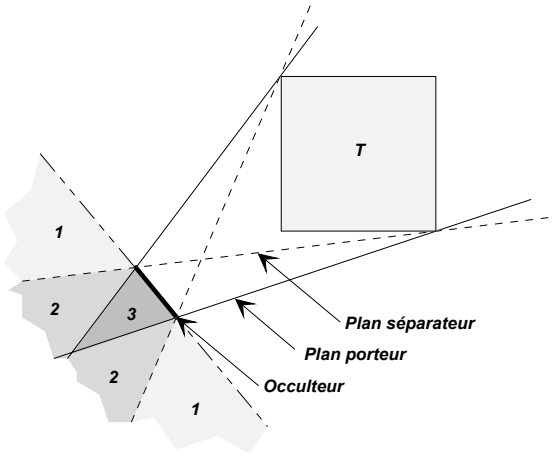


Figure 5.5 : Notion de plan séparateur et de plan porteur.

Si l'on considère un occulteur, la boîte englobante *T* d'un objet et les plans séparateur et porteur, on peut définir trois régions particulières (***Figure 5.5***).

1. la région *1* à partir de laquelle toute la boîte englobante *T* est visible ;
2. la région *2* à partir de laquelle seule une partie de *T* est visible ;
3. la région 3 à partir de laquelle on ne voit pas *T*.

La réponse partielle au problème posé peut donc être formulée de la manière suivante : *"si on se place dans les hypothèses de la* ***Figure 5.5****, on peut affirmer que si l'observateur se situe dans la zone 1 il verra entièrement l'objet, s'il se trouve dans la zone 3 il ne verra rien de l'objet et s'il se trouve dans la zone 2, il verra une partie de la boîte englobante de l'objet. Dans cette zone 2, il verra donc entièrement, partiellement ou pas du tout l'objet"*. Notons que les auteurs élargissent légèrement cette réponse à des ensembles d'occulteurs adjacents.

5.1.3.1.2 Trouver un maximum d'objets cachés

Une solution simple pour trouver les objets cachés est de déterminer pour chaque occulteur le statut (visible, partiellement visible ou non visible) de chaque objet à l'aide de la solution proposée au paragraphe précédent. Cette solution étant en $O(n^2)$ pour n polygones, elle n'est bien sûr pas envisageable. [COOR 97] propose d'utiliser une subdivision spatiale adaptative ("*octree*") [GLAS 84] [SAME 89] [SPAC 91] afin de limiter les investigations. En fait, il cherche uniquement le statut des noeuds de la subdivision adaptative. Puis, par association, il connaît les polygones non visibles. L'***Algorithme 5-1*** permet de construire la liste *OS* des objets visibles à partir d'un point de vue *P*, d'un ensemble d'occulteurs *S* et d'un noeud *T* de la subdivision spatiale. Au départ, *T* correspond à la racine de l'arbre. Les noeuds grisés de la ***Figure 5.6*** indiquent les noeuds rapportés visibles par l'algorithme.

```
Visible(OccluderSet S, OctreeNode T, Viewpoint P, ObjectSet OS)
    statut = déterminer le statut de T en fonction de S et de P;
    si(statut == VISIBLE)
    alors
        Ajouter à OS tous les polygones contenus dans le noeud T
    sinon si(statut == INVISIBLE)
    alors
        retourner;
    sinon si(statut == PARTIAL)
        si T est une feuille
        alors
            Ajouter à OS tous les polygones contenus dans le noeud T
        sinon
            // On construit la liste des occulteurs cachant partiellement le noeud
            // car seul ces occulteurs peuvent cacher entièrement des noeuds fils.
            S'=∅
            pour chaque occulteurs Ai de S
                si Ai cache partiellement T
                alors
                    S'=S'∪{Ai}
            // On applique récursivement le même algorithme sur les noeuds fils.
            pour chaque fils T' de T
                Visible(S', T', P, OS)
```

Algorithme 5-1 : Détermination des objets visibles.

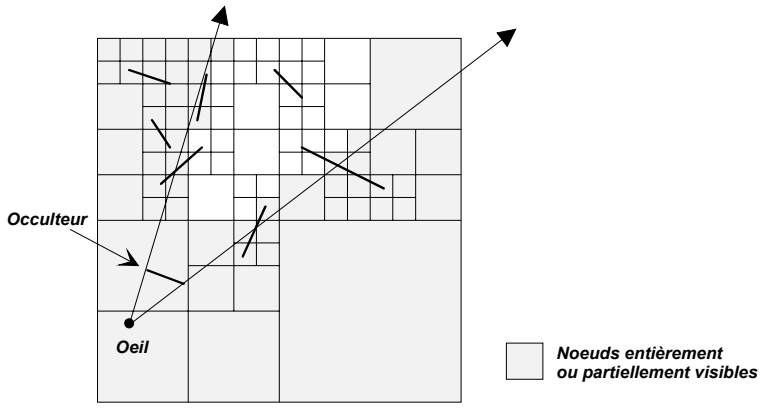


Figure 5.6 : Noeuds de la subdivision spatiale adaptative rapportés visibles par l'algorithme "Visible()" avec un seul occulteur.

5.1.3.1.3 Choisir de bons occulteurs

La définition d'un bon occulteur est intuitive, il s'agit d'un polygone convexe qui cache un grand nombre d'objets. Traduire cette définition en une équation mathématique n'est pas chose facile. [COOR 97] propose, toujours intuitivement, que le choix des "bons occulteurs" dépend principalement de la taille des polygones, de leurs distances au point de vue et de leurs orientations par rapport à la direction de vision (ceci fait penser au critère proposé par [CHOI 92] au ***Paragraphe 1.4.1.2***). Il traduit ces intuitions en quelque chose qui ressemble à l'expression d'un angle solide :

Equation 5-1 :

$$E = \frac{-A(\vec{N}.\vec{V})}{\left\|\vec{D}\right\|^2}$$

où A est la surface du polygone, $\vec{N}$ la normale, $\vec{V}$ la direction de vision et $\vec{D}$ le vecteur allant du point de vue vers le centre de l'occulteur. Cette expression est la base du choix des occulteurs.

[COOR 96] propose, en prétraitement, de discrétiser toutes les directions possibles du point de vue. Pour chaque noeud de la subdivision spatiale, il construit une liste d'occulteurs. Pour construire cette liste, il discrétise tous les points de vue possibles dans ce noeud. Pour chacun d'eux et pour chaque polygone éloigné au plus d'une distance donnée, la valeur de E est calculée. Seuls les k polygones ayant fourni les meilleures valeurs de E sont insérés dans la liste. La valeur de k est un paramètre de l'algorithme.

Ce prétraitement permet donc, à tout moment, de disposer d'une liste de "bons occulteurs". Notons cependant qu'il est nécessaire de discrétiser les directions et les positions des points de vue possibles. Il faut donc connaître à l'avance les déplacements possibles du point de vue ou les limiter. Quelle que soit la solution, celle-ci est coûteuse en place mémoire et limite l'interactivité.

5.1.3.2 Raffinement de l'approche proposée par [COOR 97]

[HUDS 97] se place aussi dans le cadre d'un déplacement dans un univers virtuel. Sa solution est très proche de celle de [COOR 97]. En effet, il utilise une subdivision spatiale régulière, à chacun de ces volumes, il associe une liste de "bons occulteurs", puis, lors de la phase de rendu, les objets cachés par ces occulteurs sont éliminés en utilisant une subdivision spatiale hiérarchique comme dans [COOR 97].

De légères nuances sont néanmoins à préciser. Les critères de choix des "bons occulteurs" sont affinés. Il utilise la même expression E (***Equation 5-1***) que [COOR 97] pour évaluer l'angle solide mais en plus, il tient compte du nombre d'objets cachés. Pour cela, lors du prétraitement, il choisit aléatoirement plusieurs points de vue dans chaque volume de la subdivision régulière et, pour chaque occulteur ayant une valeur E suffisante, il compte le nombre d'objets cachés. Ceci permet de faire un classement des occulteurs selon leur efficacité. Un autre raffinement, lié à l'animation, utilise le fait qu'un "bon occulteur" pour une image à toutes les chances de l'être aussi pour l'image suivante.

5.2 UNE SOLUTION FONDEE SUR LES FAISCEAUX

La partie précédente était consacrée à quelques méthodes permettant d'optimiser le calcul de la visibilité. Certaines d'entre elles étaient dédiées à la radiosité et d'autres aux déplacements dans des mondes virtuels. Le but de l'algorithme présenté ici est de diminuer le temps passé dans la détermination de la visibilité et ce de la manière la plus générale possible, sans être trop dépendant d'un algorithme donné. Pour valider ces travaux, il a néanmoins fallu appliquer notre solution à un algorithme de rendu. Nous avons choisi la radiosité de [WALL 89] pour des scènes polygonales convexes comme exemple d'application de notre approche.

Le choix de cet algorithme peut prêter à discussion. En effet, cette approche pour la radiosité a près de dix ans et depuis, de nombreuses variantes ont été développées. On peut penser en particulier aux techniques de "*clustering*" dont une publication récente est [GIBS 96]. Bien sûr, d'autres méthodes existent et nous renvoyons le lecteur à l'étude comparative empirique de [WILL 97] concernant ces algorithmes. Rappelons que l'utilisation de [WALL 89] sert seulement de support et que notre approche s'applique à presque toutes les techniques de radiosité, en particulier le "*clustering*". De plus, deux autres arguments plaident en la faveur de [WALL 89] : cet algorithme était déjà développé au sein de notre équipe au moment de nos travaux ; il est simple et permet donc de se focaliser sur l'optimisation que nous apportons.

Précisons bien que ce cadre de la radiosité n'est pas limitatif et que notre approche peut s'appliquer dans d'autres domaines de la synthèse d'images comme la réalité virtuelle.

Les principaux avantages de cette optimisation sont les suivants :

- des prétraitements simples et rapides ;
- des structures de données simples et très bien optimisées (subdivisions spatiales régulières) ;
- une facilité à s'insérer dans la chaîne des processus de rendu puisque entièrement autonome ;
- une application à différents domaines de la synthèse d'images (radiosité, réalité virtuelle, ...) ;
- une robustesse et des temps de calculs réduits.

Pour décrire notre algorithme, nous utilisons un étage d'une maison (***Figure 5.7***). Pour une meilleure lisibilité, cette scène est représentée sans plafond. Cette présentation est composée de trois sections : une description des principes généraux de notre algorithme, le détail des différents choix effectués et les résultats chiffrés.

5.2.1 Principes généraux

Avant d'étudier en détail notre contribution, rappelons brièvement les principes du calcul de la radiosité selon [WALL 89]. Cet algorithme servira de terrain d'expérimentation à notre solution. Tout d'abord, nous considérons une scène polygonale de laquelle on extrait un maillage en carreaux. Dans notre cas, les polygones doivent être convexes et le maillage est régulier (***Figure 5.7*** et ***Figure 5.8***), mais ce dernier pourrait être quelconque en particulier adaptatif ou discontinu ("*discontinuity meshing*" [LISC 92]).

Schématiquement, la boucle évaluant la radiosité de la scène peut s'écrire de la manière suivante :

tant que *l'on a pas atteint un certain équilibre ou un nombre d'itération donnée*
faire
Ce = carreau ayant le plus d'énergie à émettre
pour chaque *carreau Cr de la scène*
faire
$V_{Ce,Ci}$ *= visibilité entre Ce et Cr*
si $V_{Ce,Ci}$ *>0 (Ce voit au moins partiellement Cr)*
alors
$FF_{Ce,Cr}$ *= évaluer le facteur de forme entre Ce et Cr*
Mettre à jour l'énergie reçue de Cr

Algorithme 5-2 : Boucle d'évaluation de la radiosité selon [WALL 89].

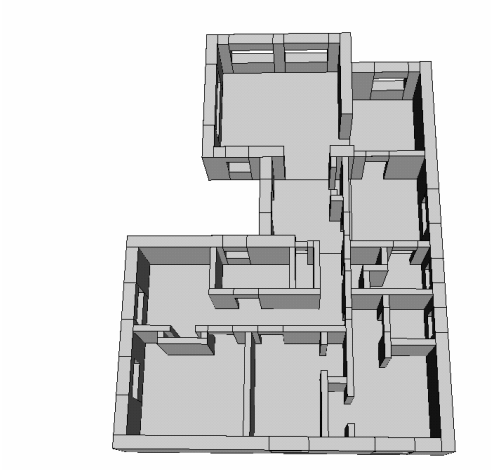


Figure 5.7 : Scène polygonale.

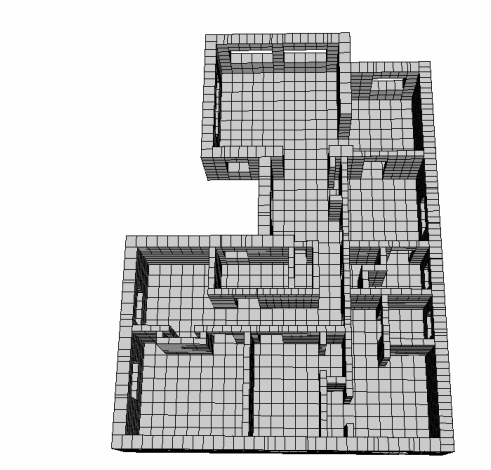


Figure 5.8 : Scène maillée régulièrement.

Le calcul de la visibilité entre deux carreaux est effectué en lançant un ou plusieurs rayons du centre du carreau émetteur *Ce* vers le centre ou les coins du carreau récepteur *Cr*. La moyenne du nombre de rayons atteignant directement le carreau récepteur correspond à la proportion de surface de *Cr* visible depuis *Ce*. Dans notre application, nous utilisons un seul rayon lancé de centre à centre et nous optimisons le lancer de rayons en utilisant une subdivision spatiale régulière (voir ***Paragraphe 3.1.1***).

Le but de notre algorithme est d'améliorer les temps de calcul lors de l'évaluation de la visibilité. Pour cela, on cherche à éliminer les carreaux clairement non visibles depuis l'émetteur. Notons que le but n'est pas de calculer une visibilité exacte mais bien de diminuer le nombre de carreaux fournis au processus évaluant précisément cette visibilité. Dans ce but, on utilise la subdivision spatiale régulière et la notion de faisceaux de la manière suivante :

1. Chercher des polygones susceptibles de cacher un grand nombre de carreaux c'est-à-dire de "bons occulteurs" (***Paragraphe 5.2.2***) ;
2. Construire, pour chaque occulteur, un faisceau issu du carreau émetteur et traversant exactement l'occulteur (***Figure 5.9***)(***Paragraphe 5.2.3***) ;
3. Parcourir les voxels de chacun des faisceaux, positionner les carreaux associés à ces voxels par rapport aux faisceaux et les marquer "*non visibles*" lorsqu'ils sont entièrement dans un faisceau (voir ***Paragraphe 5.2.4***) ;
4. Effectuer le test de visibilité uniquement entre le carreau émetteur et les carreaux qui ne sont pas marqués "*non visibles*".

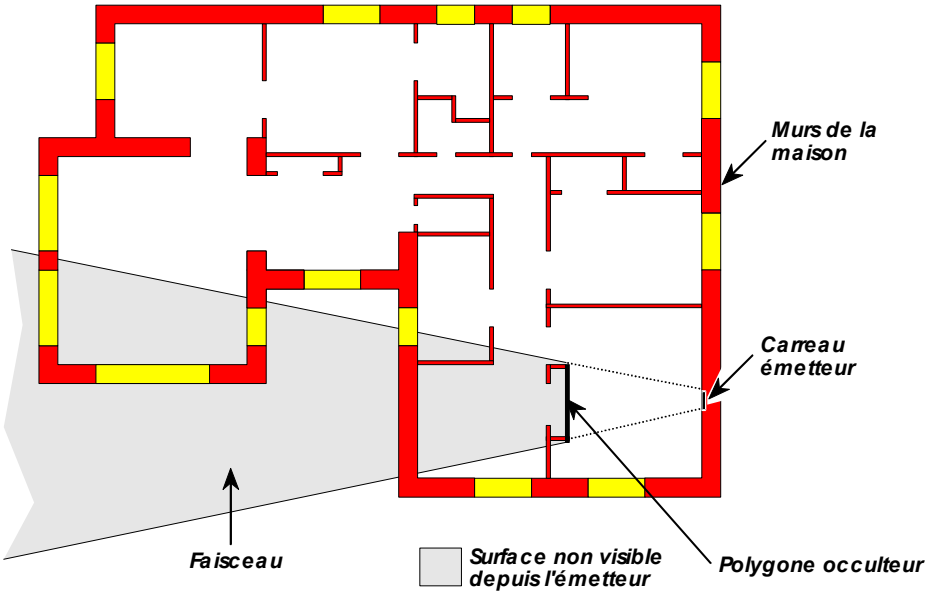


Figure 5.9 : Plan d'une maison, notion d'occulteurs et de faisceaux.

Cette approche propose un processus cohérent et autonome pouvant être inséré dans de nombreux algorithmes de synthèse d'images. L'exemple de son insertion dans l'algorithme de radiosité de [WALL 89] est schématisé ***Figure 5.10***. De plus, cette solution est modulaire et permet une approche en trois sous-problèmes :

1. Comment trouver rapidement de "bons occulteurs" ?
2. Comment construire les faisceaux issus de l'émetteur ?
3. Comment parcourir et marquer rapidement les carreaux situés entièrement à l'intérieur des faisceaux ?

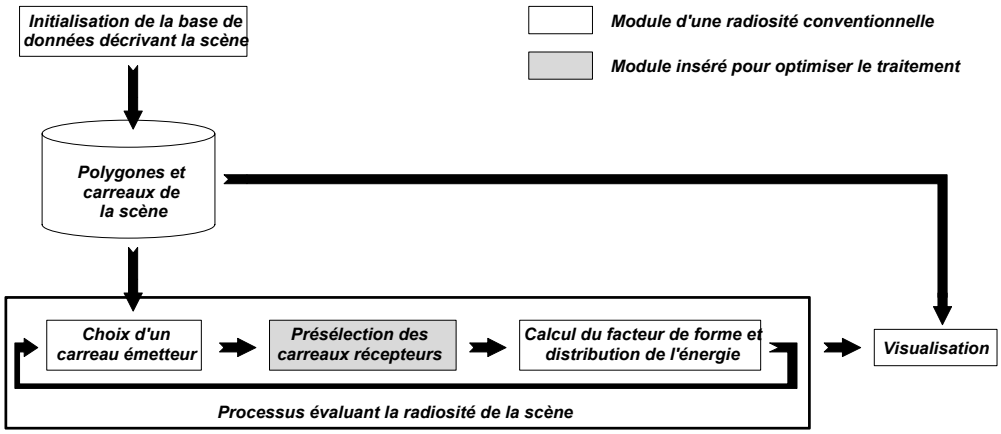


Figure 5.10 : Insertion de notre optimisation dans une radiosité conventionnelle.

5.2.2 Choix des occulteurs

[COOR 97] et [HUDS 97] quantifient la notion de "bon occulteur" et proposent l'***Equation 5-1***. Cette approche impose de discrétiser toutes les positions possibles du point de vue et d'évaluer l'***Equation 5-1*** pour tous les polygones. On peut bien sûr imposer une certaine surface et une distance minimale pour diminuer le nombre d'occulteurs potentiels, mais ces critères restent très subjectifs et difficiles à maîtriser.

Nous proposons une approche simple qui a montré son efficacité dans les différents tests effectués. Les critères de choix des bons occulteurs sont leur visibilité directe depuis le carreau émetteur, leur appartenance aux objets formant la "trame" de la scène (sols, murs,

plafonds, toit, ...) et leur convexité. Pour trouver ces polygones, nous lançons un nombre fini (paramètre de l'algorithme discuté au ***Paragraphe 5.2.5.1***) de rayons dans toutes les directions devant l'émetteur. Les polygones heurtés, s'ils font partie de la "trame" de la scène, sont choisit pour occulteur (***Figure 5.11***). Bien sûr, nous profitons de la subdivision spatiale régulière pour lancer ces rayons.

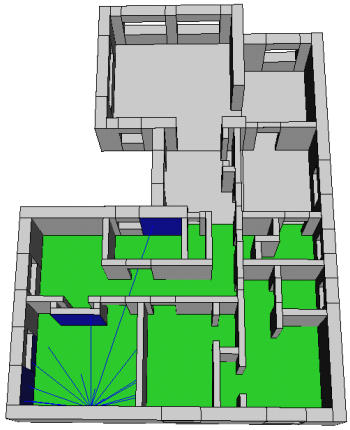


Figure 5.11 : Détection des "occulteurs". En bleu, les rayons lancés et les occulteurs détectés sur les murs et en vert les occulteurs détectés sur le sol.

Nous avons expérimenté deux distributions spatiales pour lancer ces rayons afin d'étudier leur importance dans la détection des occulteurs. La première distribution est régulièrement distribuée sur une demi-sphère posée au centre de l'émetteur. La deuxième est aléatoire sur la même demi-sphère. Pour étudier l'impact de ces deux distributions sur la détection des occulteurs, nous les avons toutes deux implémentées et testées sur un algorithme de radiosité [WALL 89]. La scène utilisée contient 14957 polygones, maillés par 42317 carreaux et on a réalisé 100 itérations pour distribuer l'énergie. La ***Figure 5.12*** montre que pour un même nombre de rayons, la distribution spatiale aléatoire fournit davantage d'occulteurs. De plus, on constate qu'à partir de 1000 rayons lancés, il y a de moins en moins de nouveaux occulteurs détectés et que le pourcentage de carreaux non visibles stagne autour de 40 à 45%.

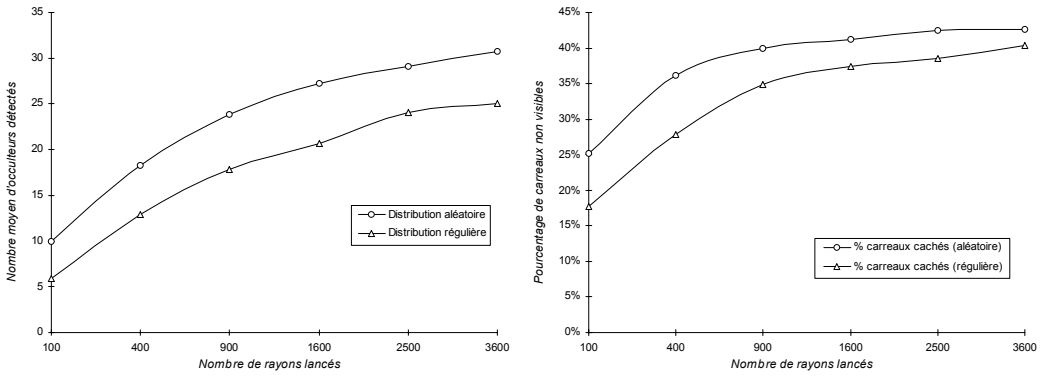


Figure 5.12 : Comparaison entre une distribution régulière et une distribution aléatoire des rayons. A gauche : nombre d'occulteurs détectés, à droite : pourcentage de carreaux non visibles.

5.2.3 Construction des faisceaux

Les faisceaux sont employés comme boîtes englobant les carreaux cachés par un occulteur donné. Cela signifie que quel que soit le rayon lancé depuis un point du carreau émetteur et traversant l'occulteur, ce rayon se situe à l'intérieur du faisceau. Dans la plupart des cas, un tel faisceau peut se construire simplement. Considérons l'occulteur de la ***Figure 5.13*** comportant quatre arêtes. Les quatre pans du faisceau sont construits de sorte à être chacun tangent à une arête de l'occulteur et passant par un sommet de l'émetteur tel que l'occulteur et l'émetteur soient entièrement du même côté. Ces quatre pans englobent ainsi exactement l'occulteur et plus largement l'émetteur (***Figure 5.13***). Le faisceau est borné d'un côté par le plan de l'occulteur et de l'autre par un plan virtuel tangent à la boîte englobante de la scène (voir ***Paragraphe 3.1.2, Figure 3.4***). Notons que l'occulteur peut avoir un nombre quelconque de côtés, le faisceau aura alors le même nombre de pans.

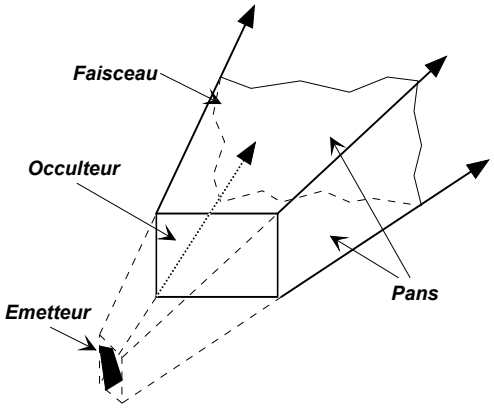


Figure 5.13 : Faisceau englobant l'émetteur et traversant un occulteur.

Malheureusement une telle construction est parfois inutile ou on obtient des faisceaux dégénérés. La ***Figure 5.14*** montre plusieurs de ces cas particuliers :

a) lorsque l'occulteur est plus petit que l'émetteur, le faisceau est croisé et donc inutilisable ;

b) l'émetteur est dans le plan de l'occulteur, le faisceau est dégénéré ;

c et d) le plan de l'émetteur coupe l'occulteur ou inversement, le faisceau est dégénéré.

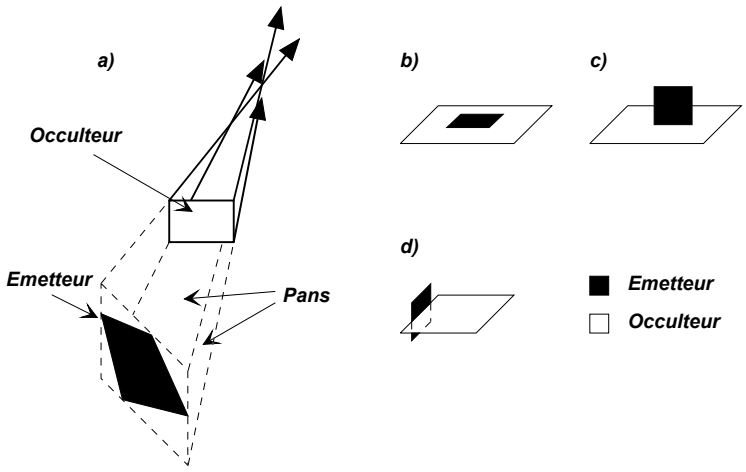


Figure 5.14 : Cas particuliers où le faisceau est croisé ou impossible à construire.

La construction des faisceaux présentée ci-dessus est très générale ; elle suppose que la visibilité est calculée en n'importe quel point de l'émetteur. Or cette hypothèse est trop forte dans le cas de la radiosité. En effet, [WALL 89] préconise de toujours lancer un rayon du barycentre du carreau émetteur. Cette nouvelle hypothèse simplifie la construction des faisceaux. Les pans sont à présent tangents aux arêtes de l'occulteur et au barycentre du carreau émetteur (***Figure 5.15***).

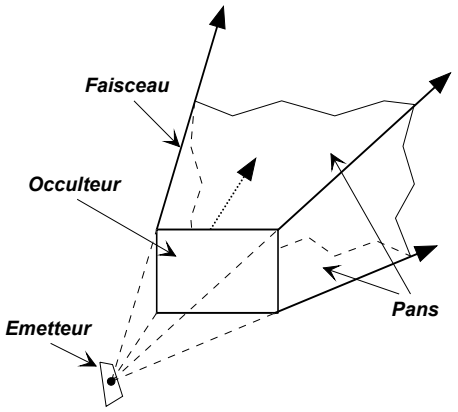


Figure 5.15 : Faisceau simplifié.

A titre indicatif, nous avons essayé les deux solutions. La ***Figure 5.16*** montre une comparaison en temps[22] entre l'utilisation de faisceaux généraux, englobant tout le carreau émetteur et celle de faisceaux simplifiés, issus du barycentre. La scène utilisée comporte 14957 polygones, 42317 carreaux et on a réalisé 100 itérations pour distribuer l'énergie avec différents nombres de rayons pour la détection des occulteurs. On constate que la différence de temps entre les deux approches est minime, la généralisation de ces faisceaux n'est donc pas plus coûteuse en temps de calcul.

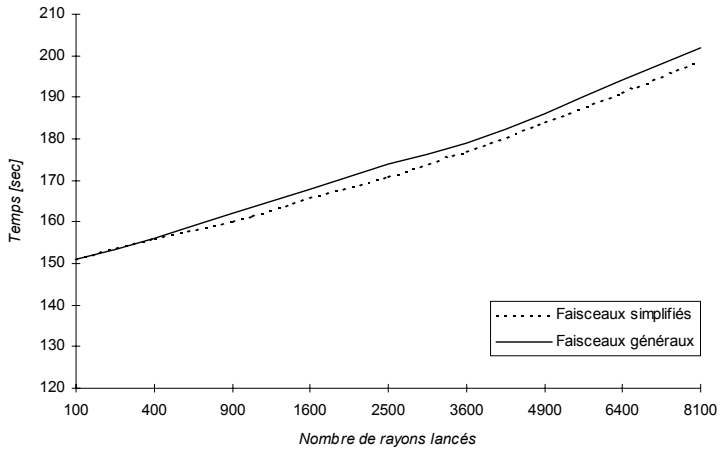


Figure 5.16 : Comparaison entre l'utilisation des faisceaux généraux et celle des faisceaux simplifiés en fonction du nombre de rayons lancés pour détecter les occulteurs.

5.2.4 Parcours et marquage des carreaux

Nous supposons ici que l'on dispose d'un carreau émetteur, d'un occulteur et d'un faisceau construit sur ces deux carreaux. Dans la phase d'initialisation de l'algorithme, on a construit une subdivision spatiale régulière dans laquelle est associée à chaque voxel la liste

[22] Temps mesurés en secondes sur une SGI O2 (µp R5000, 180Mhz, 128Mo) avec la fonction clock() de la librairie C. Le compilateur utilisé est g++ 2.7.1 avec l'option -O4.

des carreaux et des polygones qui les traversent (voir ***Paragraphe 3.1.1***). Lors du maillage, on crée des liens des polygones vers les carreaux et des carreaux vers les polygones. On suppose aussi que les carreaux sont convexes, ce qui est vérifié dans la très grande majorité des cas.

Pour déterminer les carreaux non visibles depuis l'émetteur, on détermine ceux situés entièrement dans le faisceau. Pour cela, on parcourt tous les voxels du faisceau à l'aide de l'algorithme proposé au ***Paragraphe 3.1.2*** (***Figure 5.17*** et ***Figure 5.19***). Pour chaque carreau de la liste associé à chaque voxel visité dans le faisceau, on positionne ce carreau par rapport aux plans formant le faisceau. Si tous les sommets sont à l'intérieur du faisceau, le carreau l'est aussi. Dans ce cas uniquement, on est assuré qu'il n'est pas visible depuis l'émetteur et il est marqué "*non visible*" (***Figure 5.18*** et ***Figure 5.20***).

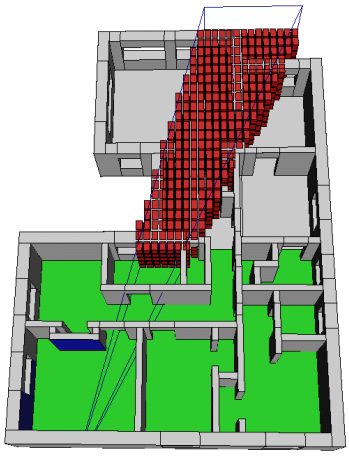


Figure 5.17 : Construction et remplissage du faisceau lancé sur un "occulteur". En rouge les voxels dans le faisceau.

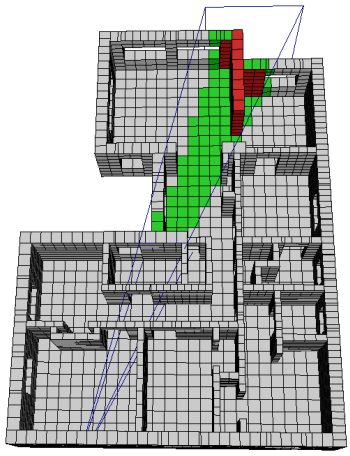


Figure 5.18 : Polygones (rouge) et carreaux (vert) détectés non visibles pour un "occulteur" donné.

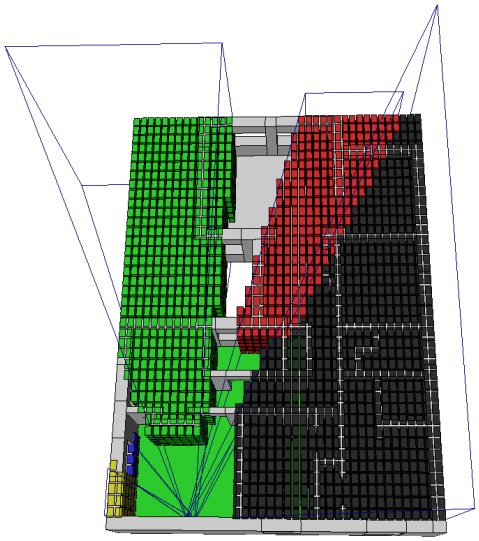


Figure 5.19 : Remplissage des faisceaux lancés sur tous les "occulteurs".

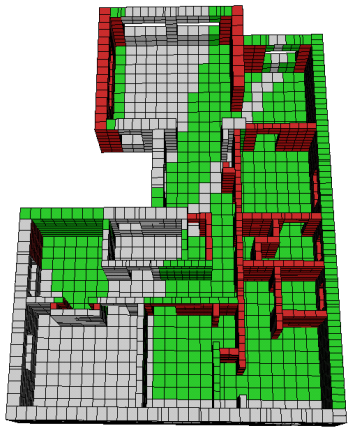


Figure 5.20 : Polygones (rouge) et carreaux (vert) détectés non visibles pour tous les "occulteurs".

La détection des carreaux cachés par les occulteurs doit être la plus rapide possible, en particulier lorsque l'on a à travailler avec un grand nombre de carreaux (plusieurs

dizaines de milliers). Pour accélérer ce traitement, nous avons développé plusieurs optimisations.

La première concerne les sommets des carreaux et des polygones. Ces sommets sont stockés dans un tableau alloué dynamiquement lors de la définition de la scène. Les sommets communs à plusieurs carreaux ou polygones ne sont pas dupliqués. Si cela est nécessaire, un prétraitement est effectué pour supprimer les doublons. A chaque sommet sont associés ces coordonnées et un identificateur unique de faisceau (entier long). On associe aussi à ce point un tableau des positions du point par rapport aux différents plans du faisceau dont on a conservé l'identificateur. Des notions d'occupation mémoire sont proposées dans l'encadré ci-contre. La première fois que l'on positionne un point par rapport à ces plans, on met à jour les différents identificateurs. Lors de prochaines évaluations de la position du point par rapport à ce même faisceau, aucun calcul n'est renouvelé. Grâce à cette structure, un point n'est donc positionné effectivement qu'une seule fois par rapport à un faisceau donné. En particulier, si un point correspond à un sommet de plusieurs polygones ou carreaux, aucun calcul n'est dupliqué.

Notions d'occupation mémoire

Considérons la scène test de la maison pour évaluer l'occupation mémoire nécessaire au stockage des sommets des polygones et des carreaux. La scène contient 14957 polygones, 42317 carreaux et 229096 sommets. Si on stocke les sommets sous forme de trois coordonnées doubles (8 octets), la place nécessaire est de :

*3*229096*8=**5.24Mo**.*

Si l'on supprime les sommets en commun, on a besoin uniquement de 38634 sommets, ce qui correspond à :

*3*38634*8=**0.88Mo***

Ceci correspond donc à un gain de presque 600%. Ajoutons à présent les informations supplémentaires nécessaires à chaque point pour effectuer notre optimisation. Ces informations sont : un identificateur unique de faisceau (un "long" de 4 octets) et un tableau des positions (des "unsigned char" de 1 octet) du point par rapport aux différents plans du faisceau (il y a en général 6 plans) dont on a conservé l'identificateur. L'occupation mémoire est alors de :

*(3*38634*8) + 38634*4 + 6*38634 = **1.25Mo**,*

ce qui correspond à une augmentation de 41%. Cette occupation étant très faible, il n'y a aucun problème de stockage.

Les optimisations suivantes concernent le positionnement des carreaux eux-mêmes. Là encore, on cherche à retenir un maximum de calculs afin d'éviter de les refaire. Ainsi, trois règles sont suivies pour gagner du temps :

1. Un carreau marqué "non visible" pour un faisceau et un émetteur donné n'est plus positionné par rapport à d'autres faisceaux issus du même émetteur. On utilise pour cela un identificateur (entier long) permettant de désigner de manière unique un carreau émetteur. Cet identificateur est associé au carreau lorsqu'il est détecté "non visible".
2. Un polygone appartenant à plusieurs voxels sera positionné une seule fois par rapport à un même faisceau. On utilise pour cela un identificateur (entier long) permettant de désigner de manière unique un faisceau. Cet identificateur est associé au carreau lorsqu'il est positionné pour la première fois par rapport à ce faisceau.
3. Si un polygone est entièrement dans un faisceau, tous les carreaux qui lui sont associés le sont aussi. Pour chaque carreau à positionner, on cherche donc la position du polygone associé. Pour limiter le traitement des polygones, on utilise le même système d'identificateur et de marquage que pour les carreaux. Un carreau entièrement dans un faisceau est marqué "non visible" et on lui associe l'identificateur de l'émetteur courant. Un polygone appartenant à plusieurs voxels n'est positionné qu'une seule fois par rapport à un faisceau donné.

L'***Algorithme 5-3*** fait la synthèse de l'algorithme de marquage.

Notons que de nombreux carreaux sont clairement non visibles depuis l'émetteur ; ce sont ceux situés derrière l'émetteur. Une solution dans l'esprit des faisceaux est de construire le volume derrière l'émetteur et borné par la boîte englobante de la scène. On peut parcourir ensuite tous les voxels de ce volume pour positionner les carreaux comme précédemment (***Figure 5.21***). Mais cette solution est coûteuse et une solution plus rapide

existe. En effet, il suffit de ne pas traiter ces carreaux dans le processus d'optimisation et de tester la position des carreaux par rapport à l'émetteur lors du test exact de visibilité. Ceci peut être réalisé rapidement avec un simple produit scalaire.

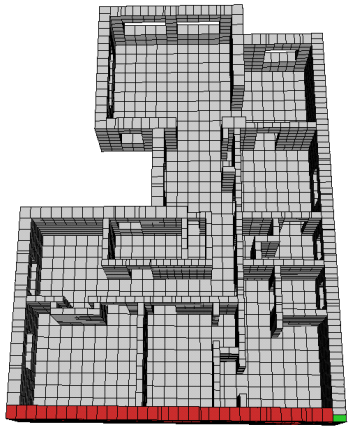


Figure 5.21 : Polygones (rouge) et carreaux (vert) non visibles puisque derrière le carreau émetteur.

```
void Beam::PositionOfPatchAndPolygonsInVoxel(Emetter emit, Voxel V)
    // Pour chaque carreau dans le voxel courant ...
    patch = le premier carreau
    tant que (pacth)
        // On vérifie que l'on n'a pas déjà traité ce carreau dans un autre faisceau issu du même
        // émetteur et que l'on n'a pas déjà traité ce carreau dans un autre voxel du même
        // faisceau
        si patch.GetEmitId() != emit.GetEmitId() && patch.GetBeamId() != id
            polygon = polygone associé à ce patch
            // On vérifie que l'on n'a pas déjà traité ce polygone dans un autre faisceau issu du
            // même émetteur ET que l'on n'a pas déjà traité ce polygone dans un autre voxel du
            // même faisceau.
            si polygon.GetEmitId()!=emit.GetEmitId() && polygon.GetBeamId()!=id
                si IsPolygonTotalyInsideBeam(polygon)
                    // Tout le polygone est dans le faisceau donc non visible
                    polygon.SetPosition(NON_VISIBLE)
                    // On retient que ce polygone est NON_VISIBLE pour cet émetteur.
                    polygon.SetEmitId(emit.GetEmitId())
                    // On marque tous les carreaux de ce polygone comme non visibles à la fois
                    // dans cet émetteur et pour ce faisceau.
                    polygon.MarkAllPatchs(NON_VISIBLE, ident, emit.GetEmitId())
                    polygon.SetBeamId(ident)
                alors
                    // Une partie du polygone est visible
                    polygon.SetPos(VISIBLE)
                    si IsPatchTotalyInsideBeam(patch)
                        // Le carreau est NON_VISIBLE
                        patch.SetPosition(NON_VISIBLE)
                        // On marque ce carreau comme non visible pour cet émetteur
                        patch.SetEmitId(emit.GetEmitId())
                    alors
                        // Le carreau est VISIBLE
                        patch.SetPosition(VISIBLE)
                    // On marque ce carreau comme déjà traité par ce faisceau.
                    patch.SetBeamId(ident)
            alors
                si IsPatchTotalyInsideBeam(patch)
                    // Le carreau est NON_VISIBLE
                    patch. SetPosition(NON_VISIBLE);
                    // On marque ce carreau comme non visible pour cet émetteur.
                    patch.SetEmitId(emit.GetEmitId());
                alors
                    // Le carreau est VISIBLE
                    patch. SetPosition(VISIBLE)
                // On marque ce carreau comme déjà traité par ce faisceau.
                patch.SetBeamId(ident)
        patch = carreau suivant
```

Algorithme 5-3 : Position des carreaux associés à un voxel.

5.2.5 Résultats

La scène test que nous utilisons comporte 14957 polygones et 42317 carreaux (***Image 15***). Elle représente un étage d'une maison (***Image 16***). Cette scène a été construite de sorte à être la moins particulière possible. Elle contient un nombre important de

carreaux et les objets (bureaux, chaises, lampes) ont été répartis de manière homogène dans toute la maison. Ainsi, on retrouve ce mobilier au moins une fois dans chacune des pièces, indépendamment de leur éclairage. De plus, certains objets, comme la lampe, sont denses : ils sont petits et composés de 1200 carreaux, d'autres sont plus grands et modélisés de manière plus grossière comme les bureaux (560 carreaux) et les chaises (400 carreaux). Le calcul de la radiosité a été effectué sur cette scène avec un plafond. Il a été supprimé dans l'***Image 15*** et l'***Image 16*** pour une meilleure lisibilité. Nous allons dans un premier temps étudier l'influence des paramètres sur notre algorithme, puis comparer le temps d'exécution avec celui de la radiosité de [WALL 89].

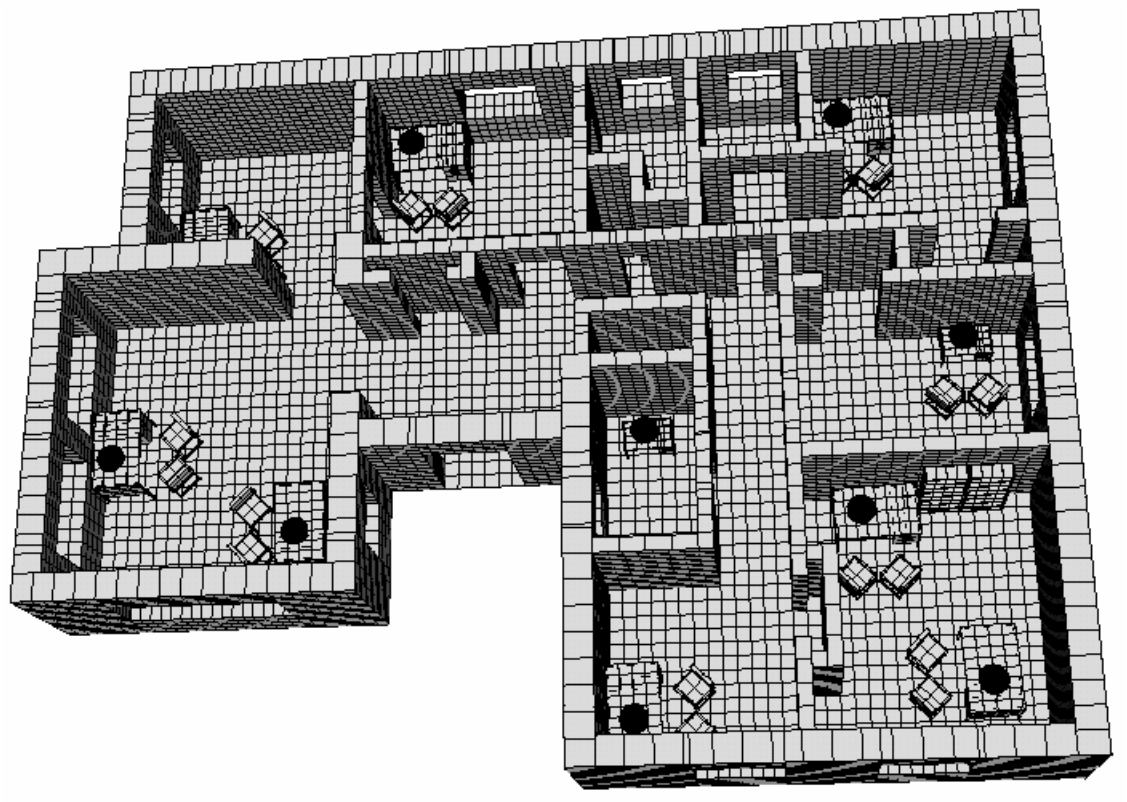


Image 15 : Maillage de la scène représentant un étage d'une maison.

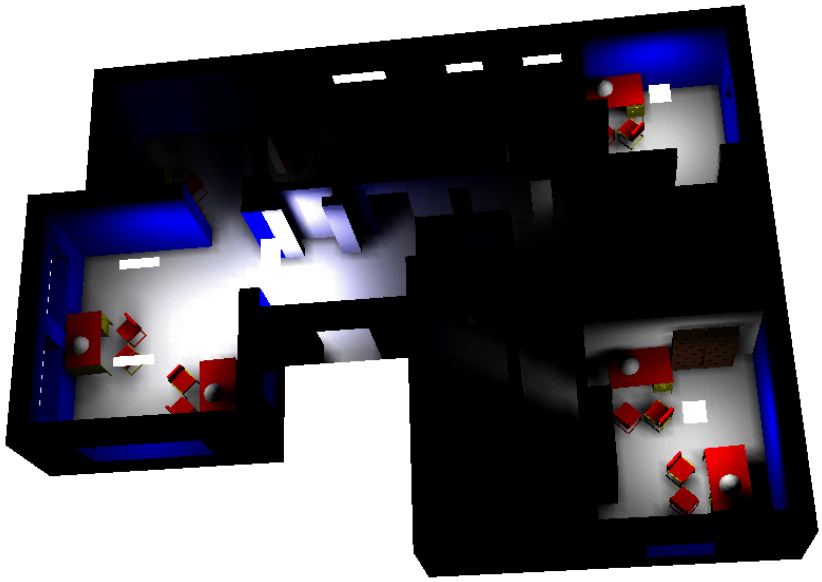


Image 16 : Rendu de la scène représentant un étage d'une maison.

5.2.5.1 Importance des paramètres

Le choix des paramètres (résolution de la subdivision spatiale, nombre de rayons lancés pour la détection des occulteurs) peut sembler complexe. En fait, l'algorithme est, dans une certaine mesure, peu sensible à ces paramètres. Nous allons montrer cela en nous basant sur un grand nombre d'essais.

Au vu de la scène (***Image 16***), on peut légitimement choisir la même résolution de la subdivision spatiale régulière en longueur et en largeur. Nous disposons donc de trois paramètres :

NxNz : la résolution de la subdivision en longueur et largeur

Ny : la résolution de la subdivision en hauteur

NbRay : le nombre de rayons lancés pour détecter les occulteurs

Pour montrer le peu d'influence de ces paramètres, nous avons exécuté 1080 fois notre algorithme de radiosité optimisé, pour 100 itérations de la radiosité, en faisant varier, par dizaine, les paramètres dans les bornes suivantes :

$30 \leq NxNz \leq 120$

$10 \leq Ny \leq 120$

$10^2 \leq NbRay \leq 90^2$

La ***Figure 5.22*** représente les 1080 triplets possibles pour ces paramètres. Pour une meilleure lisibilité du graphe, nous représentons $\sqrt{NbRay}$ au lieu de *NbRay* et *NxNy+100* au lieu de *NxNy*. Ces triplets sont ordonnés de manière croissante de gauche à droite suivant le temps d'exécution de l'algorithme. On constate que plus les triplets sont à gauche et plus chaque paramètre peut être borné finement. Ainsi, le paramètre *NbRay* se concentre entre 20^2et 60^2, *NxNy* entre 60 et 120 et *Ny* entre 20 et 60. La ***Figure 5.23*** présente les temps d'exécution pour les triplets dont tous les trois paramètres correspondent aux valeurs où se concentrent les différents paramètres. On constate que ces triplets sont

concentrés vers la gauche ce qui correspond aux temps d'exécution compris entre 253 et 327 secondes. Ces deux figures permettent donc d'affirmer (pour cette scène voulue très générale) que l'algorithme n'est que peu sensible aux quatre paramètres lorsqu'ils sont compris dans les bornes :

$$60 \leq NxNz \leq 120$$
$$20 \leq Ny \leq 60$$
$$20^2 \leq NbRay \leq 60^2$$

Ces bornes couvrant un éventail important de triplets (5*7*5=175 triplets), on peut choisir de manière presque intuitive les valeurs des paramètres sans risquer des temps d'exécution importants.

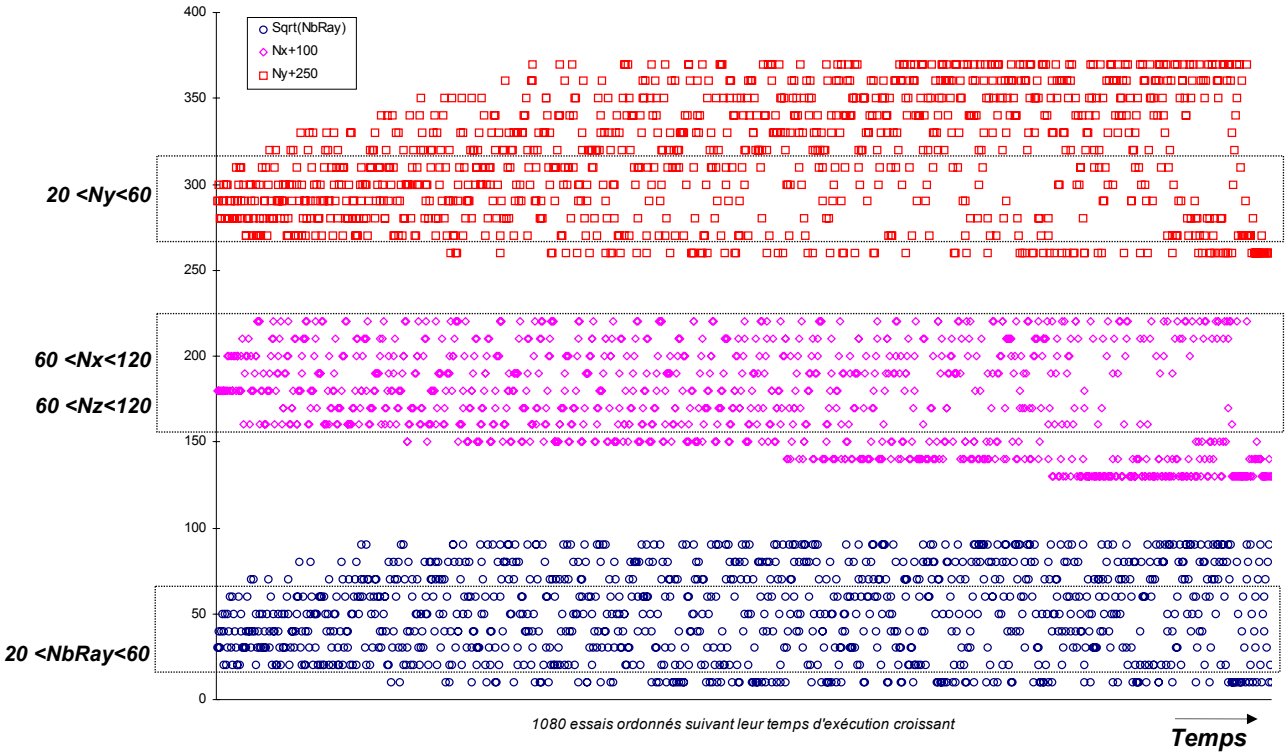


Figure 5.22 : Influence des paramètres sur les temps d'exécution.

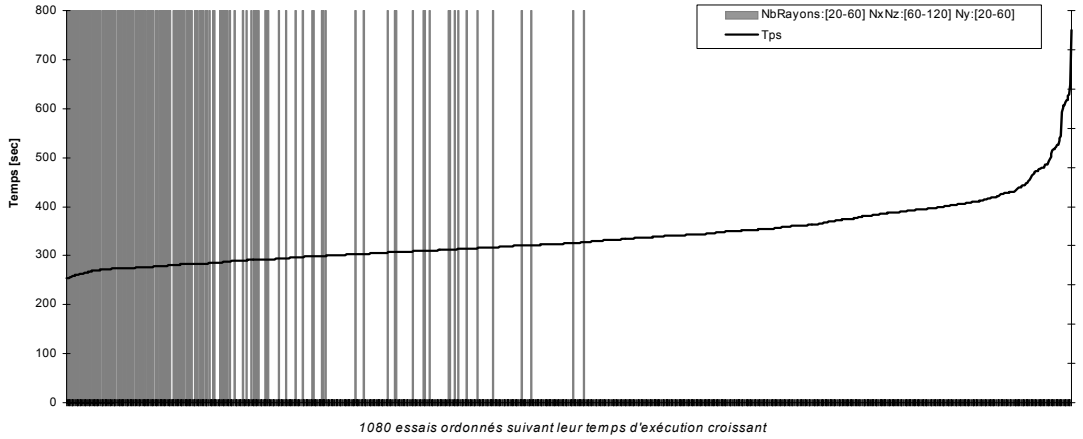


Figure 5.23 : Temps[23] d'exécution pour les triplets appartenant aux bornes choisies.

[23] Temps mesurés en secondes sur une SGI O2 (µp R5000, 180Mhz, 128Mo) avec la fonction clock() de la librairie C. Le compilateur utilisé est g++ 2.7.1 avec l'option -O4.

5.2.5.2 Comparaison avec une radiosité conventionnelle optimisée

On compare notre méthode à la radiosité de [WALL 89]. Dans cette comparaison, on veut montrer que choisir les occulteurs, construire les faisceaux et marquer les carreaux non visibles permet de gagner du temps dans une proportion non négligeable.

Avec [WALL 89], la visibilité est évaluée en lançant un seul rayon du centre du carreau émetteur vers le centre de chacun des carreaux de la scène. Dans notre méthode, on commence par supprimer les carreaux clairement non visibles, puis on utilise exactement l'algorithme de [WALL 89]. Bien sûr, on profite de la subdivision spatiale régulière pour optimiser le lancé de tous les rayons lors du calcul de la visibilité entre carreaux. Dans les deux cas, on considère 100 itérations pour évaluer la radiosité de la scène.

Nous avons montré au ***Paragraphe 5.2.5.1*** que la résolution de la subdivision spatiale n'avait pas une très grande importance lorsque l'on utilise les faisceaux pour évaluer la radiosité de la scène. Par contre, nous ne savons rien quant au comportement de la radiosité de [WALL 89] en fonction de cette résolution. Nous avons donc comparé les deux algorithmes pour des valeurs de subdivision comprises entre 30x10x30 et 120x120x120 en utilisant 900 rayons pour détecter les occulteurs. Ce dernier choix est fondé sur la ***Figure 5.12*** qui montre un plafonnement du nombre de carreaux cachés par les occulteurs à partir de 900 rayons lancés. La ***Figure 5.24*** montre que les temps d'exécution des deux algorithmes suivent une même variation. La radiosité optimisée présente cependant une amplitude de variation moindre. Ceci montre à nouveau la faible influence des paramètres de la résolution pour cette optimisation. On remarque de plus que les temps d'exécution sont globalement en faveur de la version optimisée et ce, quels que soient les paramètres. On notera que les meilleurs temps sont 346 secondes pour l'algorithme de [WALL 89] et 254 secondes pour notre solution, c'est-à-dire un gain de temps de plus de 25%. Ce rapport, relativement important, montre donc la pérennité de cette approche.

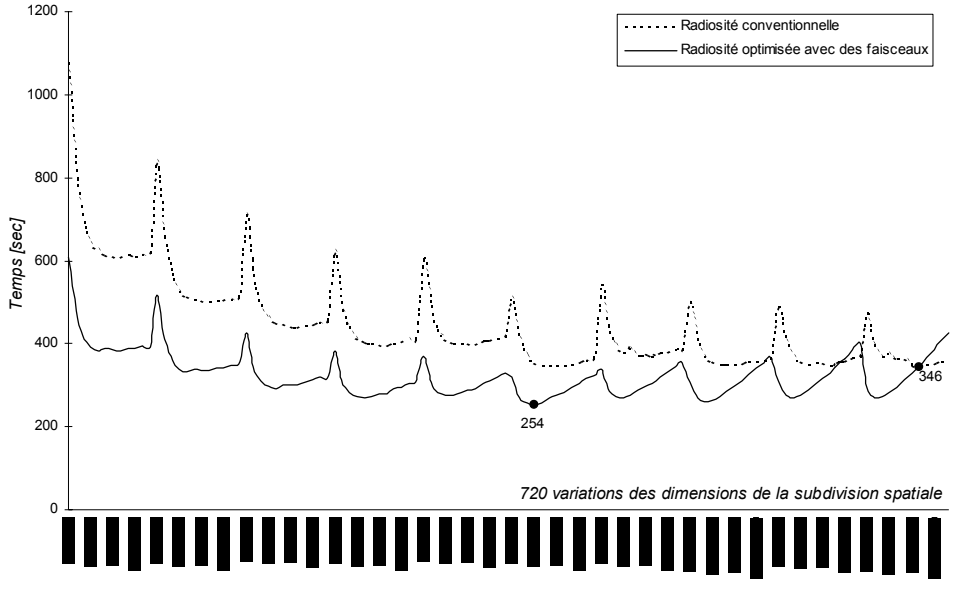


Figure 5.24 : Comparaison en temps[24] entre une radiosité conventionnelle et une radiosité optimisée avec des faisceaux et 900 rayons pour détecter les occulteurs.

[24] Temps mesurés en secondes sur une SGI O2 (µp R5000, 180Mhz, 128Mo) avec la fonction clock() de la librairie C. Le compilateur utilisé est g++ 2.7.1 avec l'option -O4.

5.3 CONCLUSION

Nous avons proposé dans ce chapitre une méthode exploitant les algorithmes développés au ***Chapitre 3*** autour des faisceaux. Cet outil permet de résoudre de nombreux problèmes en synthèse d'images et en particulier celui du calcul de la visibilité. Ce dernier problème a été optimisé dans le cadre de l'algorithme de radiosité de [WALL 89] et un gain de temps non négligeable a été réalisé. De plus, notre algorithme possède différentes propriétés intéressantes : il est autonome et il nécessite uniquement une description de la scène sous la forme d'un ensemble de polygones. Il peut ainsi être inséré dans n'importe quel processus de rendu. En particulier, dans une radiosité avec un découpage discontinu ou utilisant le *"clustering"*. On peut même utiliser cette approche dans le cadre de la réalité virtuelle et de déplacements en temps réel.

Contrairement aux autres approches, notre solution ne nécessite pas de prétraitement ou uniquement un prétraitement très simple consistant à supprimer les sommets stockés en double. D'autre part, l'occupation mémoire est acceptable. En effet, si l'on suppose que l'optimisation du lancer de rayons dans la solution de [WALL 89] utilise une subdivision spatiale régulière, la capacité mémoire est augmentée d'à peu près 40% (pour optimiser le positionnement des sommets), c'est-à-dire de quelques Mega-octets.

Néanmoins, si on se fonde sur les travaux de [COOR 97], une amélioration peut être apportée. En effet, il peut être bénéfique de détecter les occulteurs adjacents afin de construire un seul faisceau fondé sur l'union de ces occulteurs. Ceci permettrait de diminuer le nombre de faisceaux ainsi que le nombre de voxels communs à plusieurs faisceaux. Malheureusement, pour le moment, on ne sait pas trouver facilement et rapidement ces occulteurs adjacents dont l'union formerait un bon occulteur. De plus, même si ceci était réalisable, les occulteurs ne seraient plus plans ce qui risque de ralentir l'algorithme et de perdre donc tout intérêt.

CONCLUSION ET PERSPECTIVES

Les objectifs des travaux présentés dans ce mémoire résident dans l'étude et les applications des algorithmes de lancer de faisceaux pour améliorer le rendu des images de synthèse. En particulier, nous avons cherché à développer des algorithmes traitant de manière efficace les problèmes d'aliassage et/ou permettant un gain de temps.

Nous avons présenté les algorithmes fondés sur une variante du lancer de rayons qui essaient de profiter de la cohérence des scènes. La plupart d'entre eux ont pour but un gain de temps. Cependant, malgré des approches très simples comme essayer d'anticiper le chemin emprunté par un rayon ou parcourir en largeur l'arbre de récursion du lancer de rayons, les temps de calculs ne sont pas diminués. L'approche qui consiste à regrouper les rayons proches en crayons est générale et permet de quantifier les erreurs commises, mais les calculs nécessaires sont très sophistiqués et coûteux. En plus, les applications sont limitées en raison de la condition de continuité C1 nécessaire sur les surfaces. Deux solutions, fondées uniquement sur des faisceaux volumiques, ont été étudiées. Le lancer de cônes est une approche qui doit théoriquement résoudre tous les problèmes d'aliassage, cependant de nombreuses zones de flou persistent. En particulier, la construction des cônes d'ombre, de réflexion et de réfractions n'est pas clairement expliquée. Cette solution semble en fait vouée à l'échec. La deuxième solution proposée consiste à lancer un seul faisceau sur la scène et à le découper chaque fois qu'il rencontre un objet. Cette approche permet de résoudre les problèmes d'aliassage, en particulier le problème de la disparition des petits objets. Par contre, les effets de réflexion sont approchés à une transformation linéaire. De plus cette solution est complexe et coûteuse à cause du découpage des polygones de forme quelconque. La troisième catégorie d'algorithme regroupe deux approches hybrides. La première choisit dynamiquement d'utiliser un lancer de rayons ou un lancer de faisceaux. Cette approche ne traite pas précisément les problèmes d'aliassage mais propose une extension intéressante vers le calcul du facteur de forme dans la radiosité. La deuxième solution, plus intéressante, utilise les faisceaux comme volumes englobants uniquement pour détecter les régions à problèmes. Le rendu final est effectué par un lancer de rayons produisant des effets de réfraction de qualité. Le fait de ne pas faire de calculs explicites d'intersection entre objet et faisceau permet d'utiliser des scènes de type "CSG". Cette solution est sans doute la plus robuste et celle qui traite le mieux les problèmes d'aliassage. Par contre, il est nécessaire d'y apporter de nombreuses optimisations pour obtenir des temps de calcul raisonnables. Enfin, la quatrième catégorie étudiée concerne seulement une optimisation du calcul des ombres. Cette approche est très intéressante parce qu'elle fournie des outils pour positionner des objets par rapport à un faisceau.

Sur la base de l'algorithme de lancer de faisceaux pyramidaux nous avons développé différentes optimisations, en remplaçant les polygones quelconques de la scène par des polyèdres. Nous avons utilisé une subdivision spatiale régulière afin d'améliorer la recherche des objets susceptibles d'être dans un faisceau. Finalement, nous avons mis au point des tests simples et rapides pour positionner précisément les objets par rapport aux faisceaux. Ces différentes optimisations ont permis un gain de temps important et ramené

le rapport de temps entre le lancer de faisceaux pyramidaux et le lancer de rayons conventionnel à 2. La solution fondée sur les faisceaux résout tous les problèmes d'aliassage mais est plus lente ; une fois de plus il faut choisir entre réalisme et coût en temps de calculs.

La deuxième approche pour résoudre les problèmes d'aliassage est l'utilisation d'un algorithme de visibilité exacte. Nous avons proposé une première approche d'un tel algorithme fondée sur un modèle topologique de cartes généralisées et sur un lancer de faisceaux. Cette solution prend en compte de manière précise les ombres portées, les effets de réflexion et approche les réfractions. Cette approche est intéressante parce qu'elle fournit une carte de tous les contours visibles. Des manipulations telles que le changement d'échelle sont alors possibles sans perte d'informations. Pour compléter ce travail, il est nécessaire de faire une étude quantitative et comparative pour avoir une idée des temps de calcul. Le deuxième point est le traitement plus précis des effets de réfraction, en tenant compte des déformations qu'elle introduit. Pour cela, nous proposons de changer le plongement linéaire en un plongement non linéaire, en associant une arête à une courbe. Cette courbe sera calculée à partir de plusieurs points clefs obtenus avec un lancer de rayons conventionnel et sera, par exemple, définie par une courbe de Béziers. Enfin, à plus long terme, on cherche à utiliser la carte des contours pour optimiser le calcul des animations. En effet, les propriétés topologiques de la carte peuvent permettre de ne calculer la visibilité exacte que de certaines images clefs d'une animation et d'interpoler les images manquantes par des techniques de déformation ("*morphing*").

Après les problèmes d'aliassage nous avons cherché à utiliser les faisceaux pour diminuer les temps de calculs dans un processus de calcul de visibilité. Pour cela nous avons proposé une technique visant à réduire le nombre d'objets à traiter lors du calcul de la visibilité. Nous avons introduit la notion de "bon occulteur" correspondant à un objet imposant qui cache une grande partie de la scène. Des faisceaux issus du point de vue et lancés sur ces occulteurs nous ont permis de marquer les objets non-visibles. Différentes solutions pour détecter les "bons occulteurs" ont été présentées ainsi qu'une technique fondée sur une subdivision spatiale régulière pour déterminer les objets non-visibles. Nous avons montré que l'approche est relativement simple à mettre en oeuvre, que les structures de données utilisées sont bien connues et largement optimisées. De plus, cette approche s'insère facilement dans un processus de rendu. Comme exemple d'application, nous avons utilisé un algorithme de radiosité. L'insertion a été très facile et les gains de temps substantiels. Nous cherchons à présent à utiliser cette technique dans une approche "*réalité virtuelle*", en particulier pour le déplacement dans des scènes très importantes comportant plusieurs centaines de milliers de polygones. Une première idée pour réaliser cela est d'associer à un ensemble de polygones opaques, un polyèdre de taille inférieure ou égale. Ce polyèdre joue alors le rôle de "bon occulteur" dans le calcul de la visibilité. Dans le cas d'une ville virtuelle, l'ensemble de polygones est par exemple un immeuble ou une partie du immeuble à travers lequel on ne voit pas et le polyèdre un parallélépipède légèrement plus petit que l'immeuble. Le polyèdre remplace alors l'immeuble lors de la détermination des objets non-visibles. Une deuxième approche consiste à diviser l'espace en volumes disjoints en fonction des emplacements où peut se trouver la caméra. Une détermination de visibilité entre ces volumes, faite en prétraitement, peut faciliter le calcul de la visibilité en temps réel.

BIBLIOGRAPHIE

[AIRE 90] John M. AIREY, John H. ROHLF, Frederick P. BROOKS, "Towards Image Realism with Interactive Updates in Complex Virtual Building Environments", Computer Graphics: Proc. 1990 Symposium on Interactive 3D Graphics, March 1990, 24 (2), pp. 41-50.

[AMAN 84] John AMANATIDES, "Ray Tracing with Cones", Computer Graphics (Proceedings of SIGGRAPH'84), July 1984, 18 (3), pp. 129-135.

[AMAN 87] John AMANATIDES, Andrew WOO, "A Fast Voxel Traversal Algorithm for Ray-Tracing", Proceedings of Eurographics Conference, August 1987, pp. 27-38.

[APPE 68] Athur APPEL, "Some Techniques for Shading Machine Renderings of Solids", AFIPS 1968, Spring Joint Computer Conference, 32, 1968, pp. 37-45

[ATHE 78] Peter ATHERTON, Kevin WEILER, Donald GREENBERG, "Polygon Shadow Generation", Computer Graphics (SIGGRAPH'78), 12(3), 1978, pp. 275-281

[BENT 75] J. L. BENTLEY, J. H. FRIEDMAN, "Data Structures for Range Searching", Computing Surveys, 1975, 11, p 397.

[BLIN 78] James F BLINN, "Simulation of wrinkled surfaces", Computer Graohics (SIGGRAPH'78), August 1978, 12, pp. 286-292,.

[BORN 59] M. BORN, E. WOLF, "Principles of Optics", 1959, pp. 190-196, New-York: Pergamon.

[CATM 74] E. CATMUL, "A Subdivision Algorithm for Computer Display of Curved Surfaces", Ph. D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, December 1974.

[CHOI 92] H. K. CHOI, C. M. KYUNG "PYSHA - A Shadow-Testing Acceleration, Scheme for Ray-Tracing", Computer-Aided Design, February 1992 Vol. 24(2) pp. 93-104

[CLEA 88] John G. CLEARY, Geoff WYVILL, "Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Subdivision", Journal of the Visual Computer, 1988, 4(1), pp. 65-83.

[COHE 86] F. C; COHEN, D. P. GREENBERG, D. S; IMMEL, P. J. BROCK, "An Efficient Radiosity Approach for Realistic Image Synthesis", IEEE Computer Graphics and Applications, March 1986.

[COOK 84] Robert L. COOK, T. PORTER, L. CARPENTER, "Distributed Ray Tracing", Computer Graphics (Proceedings of SIGGRAPH'84), July 1984, 18(3), pp. 137-145.

[COOR 96] Satyan COORG, Seth TELLER, "Temporally Coherent Conservative Visibility", Proceedings of the Twelfth Annual ACM Symposium on Computational Geometry, Philadelphia, PA, May 1996.

[COOR 97] Satyan COORG, Seth TELLER, "Real-Time Occlusion Culling for Models with Large Occluders", Proceedings of the 1997 Symposium on Interactive Graphics, pp. 83-90 and 189.

[DADO 85] Norm DADOUN, David G. KIRKPATRICK "The Geometry of Beam, Tracing", ACM Symposium on Computational Geometry, 1985, pp. 55-61

[DEVI 89] Olivier DEVILLERS, "The Macro-regions: an Efficient Space Subdivision Structure for Ray Tracing", Proceedings of Eurographics'89, November 1989, pp. 27-38.

[DORW 94] Susan E. DORWARD, "A Survey of Object-space Hidden Surface Removal", International Journal of Computer Geometry & Applications, 1994, 4(3), pp. 325-362.

[ENDL 94] Robert ENDL, Manfred SOMMER, "Classification of Ray-Generators in Uniform Subdivisions and Octrees for Ray Tracing", Computer Graphics Forum, 1994, 13(1), pp. 3-19.

[FUCH 80] Henry FUCHS, Zvi M. KEDEM, Bruce F. NAYLOR, "On Visible Surface Generation by a Priori Tree Structures", Computer Graphics, June 1980,14(3), pp 124-133.

[FUJI 86] Akira FUJIMOTO, Takayuki TANAKA, Kansei IWATA, "ARTS: Accelearted Ray-Tracing System", IEEE Journal of Computer Graphics and Applications, April 1986, 6 (2), pp. 16-26.

[FUNK 93] Thomas A. FUNKHOUSER, Carlo H. SEQUIN, "Adaptive Display Algorithm for Interactive Frame Rates during Visualization of Complex Virtual Environments", SIGGRAPH'93. Conference proceedings on Computer graphics, 27, August 1993, pp. 247-254.

[GANG 84] M. GANGNET, D. GHAZANFARPOUR, "Techniques for Perspective Mapping of Plane Textures", International Electronic Image Week, CESTA, Biarritz, May 1984, pp. 29-35.

[GHAZ 91] Djamchid GHAZANFARPOUR, Bernard PEROCHE, "A High Quality Fitering Using Forward Texture Mapping", Computers & Graphics, 15(4), 1991, pp 569-577.

[GHAZ 92] Djamchid GHAZANFARPOUR, "Visualisation réaliste par lancer de pyramides et subdivision adaptative", proceedings of MICAD 92, PARIS, February 1992, pp.167-181.

[GHAZ 98] Djamchid GHAZANFARPOUR, Jean-Marc HASENFRATZ, "A Beam Tracing with Precise Antialiasing for Polyhedral Scenes", Computer & Graphics, January 1998 (à paraître), 22(1).

[GIBS 96] Simon GIBSON, Roger HUBBOLD, "Efficient Refinement and Clustering for Radiosity in Complex Environments", Computer Graphics Forum, 1996, 15(5).

[GLAS 84] Andrew GLASSNER, "Space Subdivision for Fast Ray Tracing", IEEE Journal of Computer Graphics and Applications, October 1984, 4(5), pp. 15-22.

[GOLD 87] Jeffrey GOLDSMITH, John SALMON, "Automatic Creation of Object Hierarchies for Ray Tracing", IEEE Journal of Computer Graphics and Applications, May 1987, pp. 14-20.

[GOUR 71] H. GOURAUD, "Continuous Shading of Curved Surfaces", IEEE Transaction on Computers, C-20(6), June 1971, pp 623-629.

[GREE 93] Ned GREENE, Michael KASS, Gavin MILLER, "Hierarchical Z-buffer visibility", SIGGRAPH'93. Conference proceedings on Computer graphics, 1993, pp. 231-238.

[HAIN 91] Eric A. HAINES, John R. WALLACE, "Shaft Culling for Efficient Ray-Cast Radiosity", Second Eurographics Workshop on Rendering, May 1991.

[HASE 95] Jean-Marc HASENFRATZ, Djamchid GHAZANFARPOUR, "Lancer de Faisceaux", 3ème Journées AFIG, Marseille 1995, pp 61-67.

[HASE 96] Jean-Marc HASENFRATZ, Djamchid GHAZANFARPOUR, "Rendering CSG Scenes with General Antialiasing", CSG 96 Set-theoretic Solid Modelling

Techniques and Applications, Information Geometers, Winchester, 1996, pp 275-289.

[HECK 84] Paul S. HECKBERT, Pat HANRAHAN "Beam Tracing Polygonal Objects", Computer Graphics (SIGGRAPH'84 Proceedings), July 1984, 18(3), pp. 119-127

[HUDS 97] Tom HUDSON, Dinesh MANOCHA, Jonathan D. COHEN, Ming C. LIN, Kenneth E. HOFF, Hansong ZHANG, "Accelerated Occlusion Culling Using Shadow Volumes", Proc. of ACM Symposium on Computational Geometry, June 1997.

[KAPL 87] Michael R. KAPLAN, "The Use of Spatial Coherence in Ray Tracing", Techniques for Computer Graphics, 1987, pp. 173-193.

[KAY 86] Timothy L. KAY, James T. KAJIYA, "Ray Tracing Complex Scenes", Computer Graphics, August 1986, 20(4), pp. 269-278.

[KIRK 87] David B. KIRK "The Simulation of Natural Features Using Cone Tracing", The Visual Computer, 1987, 3, pp. 63-71.

[LIEN 89a] Pascal LIENHARDT, "Subdivision of N-Dimensional Spaces and N-Dimensional Generalized Maps", 5° Annual A.C.M. Symposium on Computational Geometry, Saabrücken, June 1989, pp. 228-236.

[LIEN 89b] Pascal LIENHARDT, "2-G-Maps : A Model for the Manipulation of General 2-Dimensional Subdivisions", International Conference on C.A.D. & C.G., Beijing, Chine, August 1989.

[LIEN 89c] Pascal LIENHARDT, "Subdivision of Surfaces and Generalized Maps", Eurographics'89, September 1989, pp. 439-452.

[LISC 92] Dani LISCHINSKI, Filippo TAMPIERI, Donald P. GREENBERG, "Discontinuity MESHING for Accurate Radiosity", IEEE Journal of Computer Graphics and Applications, November 1992 12(6), pp. 25-39.

[LUEB 95] David P. LUEBKE, Chris GEORGES, "Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets", ACM Interactive 3D Graphics Conference, Monterey, CA, April 1995, pp. 105-106.

[MARK 88] Joseph MARKS, Robert J. WALSH, Mark FRIEDEL "Hybrid Beam-Ray Tracing", Harvard University - Center for Research in Computer Technology TR-03-88, 1988.

[MARK 90] Joseph MARKS, Robert J. WALSH, John CHRISTENSEN, Mark FRIEDEL "Image and Intervisibility Conherence in Rendering", Proceedings of Graphics Interface'90, May 1990, pp. 17-30.

[MÜLL 86] H. MULLER "Image Generation by Space Sweep", Computer Graphics Forum 1986, 5, pp. 189-196.

[NEHL 95] Philippe NEHLIG, Claudio MONTANI,"A Discrete Template Based Plane Casting Algorithm For Volume Viewing", DGCI 95, Clermont-Ferrand, September 1995.

[NEWE 72] B. F. NEWELL, R. G. NEWELL, T. L. SANCHA, "A Solution to the Hidden Surfaces Problem", Proceedings of ACM National Conference 1972, pp 443-450.

[NEWM 79] W. M. NEWMAN and R. F. SPROULL, "Principles of Interactive Computer Graphics", MacGrey-Hill Publ. Comp. 1979.

[PHON 75] B. T. PHONG, "Illumination for Computer Generated Pictures", ACM, Graphics and Image Processing, 1975, 18(6).

[RUBI 80] S. M RUBIN. and T. WHITTED, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes", Computer Graphics, July 1980, pp.110-116.

[SAME 89] Hanan SAMET, "Implementing Ray Tracing with Octrees and Neighbor Finding", Computers and Graphics, 1989, 4(13), pp. 445-460.

[SEDG 83] Robert SEDGEWICK, “Algorithms”, Addison-Wesley, 1983.

[SHAD 96] Jonathan SHADE, Dani LISCHINSKI, David H. SALESIN, Tony DEROSE, John SNYDER, “Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments”, SIGGRAPH’96. Proceedings of the SIGGRAPH 96 Conference on Computer graphics, 1996, pp. 75-82.

[SHIN87] Mikio SHINYA, Tokiichiro TAKAHASHI, Seiichiro NAITO “Principles and Applications of Pencil Tracing”, SIGGRAPH’87, July 1987, 21(4), pp. 45-54.

[SILL 97] François X. SILLION, George DRETTAKIS, Benoit BODELET, “Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery”, Proceedings of Eurographics'97, Budapest, Hungary, 1997, 16(3).

[SPAC 91] John SPACKMAN, Philip WILLIS, “The {SMART} Navigation of a Ray Through an Oct-Tree”, Computers and Graphics, 1991, 2(15), pp. 185-194.

[SPEE 85] L. Richard SPEER, Tony D. DEROSE, A. BARSKY “A Theoretical and Empirical Analysis of Coherent Ray-Tracing”, Computer Generated Images: The State of the Art, 1985.

[TELL 91] Seth J. TELLER, Carlo H. SEQUIN, “Visibility Preprocessing for Interactive Walkthroughs”, SIGGRAPH’91. Proceedings of the 18th international conference on Computer graphics, July 1991, 25 (4), pp. 61-70.

[TELL 94] Seth TELLER, Pat HANRAHAN, “Visibility Computations for Global Illumination Algorithms (ou Global Visibility Algorithms for Illumination Computations)”, Computer Graphics (SIGGRAPH’94 Proceedings), August 1994, 27, pp. 443-450.

[UNIV 90] Encyclopædia Universalis, Chapitre convexité - Ensembles convexes, Théorème de séparation, juin 1990, p. 503.

[WALL 89] John R. WALLACE, Kells A. ELMQUIST, Eric A. HAINES, “A Ray Tracing Algorithm for Progressive Radiosity”, SIGGRAPH’89, July 1989, 23(3), pp 315-324.

[WARN 69] J. WARNOCK, “A Hidden Surface Algorithm for Computer Generated Half-Tone Pictures”, TR 4-15, NTIS AD-753 671, Computer Science Departement, University of Utah, Salt Lake City, UT, June 1969.

[WEIL 77] Kevin WEILER, Peter ATHERTHON, “Hidden Surface Removal Using Polygon Area Sorting”, Computer Graphics, 1977, 11(2), pp. 214-222.

[WHIT 80] Turner WHITTED “An Improved Illumination Model for Shaded Display”, Communications of the ACM, June 1980, 23(6), pp. 343-349.

[WILL 97] Andrew J. WILLMOTT, Paul S. HECKBERT, “An Empirical Comparison of Radiosity Algoritms”, CMU-CS-97-115, CS Dept, Carnegie Mellon University, April 1997.

[WOO 90] Andrew WOO, Pierre POULIN, Alain FOURNIER, “A Survey of Shadow Algorithms”, IEEE Journal of Computer Graphics and Applications, November 1990, pp. 13-32.

[WYVI 86] Geoff WYVILL, Tosiyasu L. KUNII, Yasuto SHIRAI, “Space Division for Ray Tracing in CSG”, IEEE Journal of Computer Graphics and Applications, April 1986, 6(4), pp. 28-34.

[YAGE 95] Roni YAGEL, William RAY, “Visibility Computation for Efficient Walkthrough of Complex Environments”, Presence: Teleoperators and Virtual Environments, 1995, 5(1).

BIBLIOGRAPHIE CROISEES

A

AIRE 90 122
AMAN 84 ..... 6; 9; 22; 25; 26; 27; 28; 30; 53; 54; 69; 91
AMAN 87 ..... 70; 71; 72; 119
APPE 68 99
ATHE 78 35

B

BARR 86 18
BENT 75 13
BLIN 78 29
BORN 59 16

C

CATM 74 99
CHOI 92 ..... 10; 50; 51; 52; 54; 60; 72; 125
CLEA 88 ..... 70; 71; 119
COHE 86 ..... 41; 42
COOK 84 91
COOR 96 ..... 120; 123; 126
COOR 97 ..... 120; 123; 124; 125; 126; 129; 141

D

DADO 85 36
DEVI 89 71
DORW 94 99

E

ENDL 94 70

F

FUCH 80 ..... 36; 71; 122
FUJI 86 ..... 70; 71; 119
FUNK 93 119

G

GANG 84 ..... 49; 117
GHAZ 91 49
GHAZ 92 ..... 6; 10; 37; 40; 42; 47; 49; 52; 53; 54; 55; 56; 57; 59; 63; 64; 67; 68; 70; 72; 76; 77; 91; 97; 99; 113; 118
GHAZ 98 ..... 6; 49; 69; 97
GIBS 96 127
GLAS 84 ..... 70; 119; 124
GOLD 87 ..... 58; 119
GOUR 71 35
GREE 93 119

H

K

L

M

N

P

R

S

T

U

W

Y