Simulation d'éclairage dans des environnements architecturaux complexes : approches séquentielle et parallèle

Daniel Meneveaux

► **To cite this version:**

Daniel Meneveaux. Simulation d'éclairage dans des environnements architecturaux complexes : approches séquentielle et parallèle. Interface homme-machine [cs.HC]. Université Rennes 1, 1998. Français. NNT : . tel-00007237

HAL Id: tel-00007237

https://theses.hal.science/tel-00007237v1

Submitted on 28 Oct 2004

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

présentée

DEVANT l'UNIVERSITÉ DE RENNES I

pour obtenir

le grade de : *DOCTEUR DE L'UNIVERSITÉ DE RENNES I*

Mention : Informatique

PAR

Daniel MENEVEAUX

Équipe d'accueil : IRISA, équipe SIAMES

École Doctorale : Informatique, traitement du signal et Télécommunications

Composante Universitaire : IRISA

Simulation d'éclairage dans des environnements architecturaux complexes : approches séquentielle et parallèle.

SOUTENUE LE 22 Juin 1998 devant la commission d'examen

M.	Jean-Pierre	BANÂTRE	Président
M.	Kadi	BOUATOUCH	Directeur
MM.	Erik	JANSEN	Rapporteurs
	Jean-Claude	PAUL	
MM.	Michel	LUCAS	Examinateurs
	Thierry	PRIOL	

à ma famille,

Au cours de ces trois années de thèse, j'ai beaucoup appris et cela grâce à de nombreuses personnes que je tiens à remercier pour leur soutien :

Tout d'abord l'IRISA, un institut offrant un cadre de travail idéal, et plus particulièrement tous les membres de l'équipe SIAMES qui m'ont accueilli et avec qui j'ai eu plaisir à travailler.

M. Jean-Pierre Banâtre, directeur de l'IRISA pour avoir accepté de présider mon jury de thèse.

MM. Jean-Claude Paul et Erik Jansen qui ont eu l'obligeance et surtout la grande patience de lire mon rapport de thèse.

MM. Michel Lucas et Thierry Priol qui ont accepté de faire partie de mon jury de thèse.

Pascal Guitton pour nos dicussions très enrichissantes mais également pour son soutien.

Eric Maisel, pour ses nombreuses idées.

Jean-Marc Deniel, Romuald Delmont et Franck Galpin pour leur aide en tant que stagiaires au sein de notre équipe.

Enfin, plus particulièrement le professeur Kadi Bouatouch, en qui je vois également un ami qui m'a encadré, guidé et soutenu au cours de mes travaux.

Table des matières

Introduction **12**

1 Modèle d'illumination globale et méthode de radiosité **16**
1.1 Problématique . . . 16
1.2 Grandeurs photométriques . . . 16
1.3 Reflectivité et réflectance . . . 18
1.4 L'équation de luminance . . . 18
1.5 De l'équation de luminance à l'équation de radiosité . . . 19
1.6 Radiosité . . . 19
1.7 Résolution du système d'équations . . . 21
1.8 Radiosité hiérarchique . . . 22
1.9 Structures de données . . . 23
1.10 Raffinement . . . 23
1.11 Résolution du système hiérarchique . . . 25
1.12 L'Oracle . . . 27
1.13 Algorithme de radiosité hiérarchique . . . 28
1.14 Discussion . . . 28

2 Structuration de scènes **30**
2.1 Problématique . . . 30
2.2 Travaux existants . . . 31
2.2.1 Environnements architecturaux . . . 31
2.2.2 Discussion . . . 36
2.3 Principes de notre méthode . . . 36
2.3.1 Hypothèses et objectifs . . . 36
2.3.2 Principes . . . 37
2.3.3 Méthode semi-automatique . . . 37
2.4 Notre méthode de découpage . . . 38
2.4.1 Définition de l'espace dual . . . 38
2.4.2 Classification . . . 41
2.4.3 Création des plans de découpage . . . 43
2.4.4 Règles de construction des cellules. . . . 44
2.4.5 Appartenance d'une surface à une cellule . . . 55
2.5 Détection des ouvertures . . . 55
2.5.1 Plans de découpage et ouvertures . . . 56
2.5.2 Calcul des ouvertures . . . 56
2.6 Création du graphe de visibilité . . . 58
2.6.1 Graphe d'adjacence . . . 58
2.6.2 Graphe de visibilité . . . 60
2.7 Mise en œuvre et utilisation du programme . . . 61
2.7.1 La fenêtre de travail . . . 61
2.7.2 La fenêtre de visualisation . . . 62
2.8 Résultats . . . 63
2.8.1 Le Soda-Hall du MIT . . . 63
2.8.2 Environnement non axial . . . 64
2.9 Discussion . . . 65

3 Regroupement de polygones et calcul de visibilité 67
3.1 Problématique . . . 67
3.2 Travaux antérieurs . . . 68
3.2.1 Regroupement de polygones . . . 68
3.2.2 Calculs de visibilité entre groupes de polygones . . . 73
3.2.3 Utilisation de groupes de polygones en simulation d'éclairage . . . 74
3.3 Notre approche de regroupement de polygones . . . 76
3.3.1 Hypothèses . . . 77
3.3.2 Les nuées dynamiques . . . 77
3.3.3 Application à la classification de polygones . . . 78
3.4 Notre approche de calcul de visibilité . . . 82
3.4.1 Les volumes de visibilité . . . 83
3.4.2 Déterminer les groupes de polygones contenus dans un volume de visibilité . . . 88
3.4.3 Propagation de la recherche . . . 89
3.5 Mise en œuvre . . . 92
3.5.1 Description technique . . . 92
3.5.2 Optimisations . . . 94
3.6 Résultats . . . 95
3.7 Discussion . . . 95

4 Modélisation d'environnements architecturaux complexes 99
4.1 Problématique . . . 99
4.2 Notre approche . . . 100
4.2.1 Modélisation et interface graphique . . . 100
4.2.2 Compilation (ou décompression) des fichiers . . . 102
4.3 Mise en œuvre . . . 104
4.4 Résultats . . . 104
4.5 Conclusion du chapitre . . . 107

5 Simulation d'éclairage séquentielle 108
5.1 Problématique . . . 108
5.2 Travaux antérieurs . . . 109
5.3 Motivations . . . 110
5.4 Principe de notre algorithme . . . 110
5.5 Ordonnancement . . . 111
5.5.1 Principe . . . 111
5.5.2 Stratégies . . . 114
5.5.3 Remarque . . . 118
5.6 Implémentation et Résultats . . . 118
5.6.1 Moyens matériels . . . 118
5.6.2 Implémentation . . . 118
5.6.3 Résultats . . . 119
5.7 Discussion . . . 121

6 Simulation d'éclairage parallèle 126
6.1 Introduction . . . 126
6.2 Travaux précédents . . . 126
6.2.1 T. Funkhouser . . . 126
6.2.2 C. C. Feng et S. N. Yang . . . 128
6.2.3 B. Arnaldi et. al. . . . 129
6.2.4 Discussion . . . 129
6.3 Nos objectifs . . . 130
6.4 Principes de notre méthode . . . 130
6.5 Description détaillée de notre méthode . . . 132
6.5.1 Découpage du graphe de visibilité . . . 132
6.5.2 Organisation de l'espace disque et cohérence des données . . . 136
6.5.3 Equilibrage de charge dynamique . . . 137
6.5.4 Prévision à long terme vs. équilibrage de charge dynamique . . . 139
6.5.5 Terminaison des processus et synchronisation . . . 139

6.5.6 Interblocages et congestion 140
6.5.7 Preuve formelle de la convergence de notre algorithme 142
6.6 MPI et mise en œuvre 145
6.6.1 MPI : Message Passing Interface 145
6.6.2 Initialisations 146
6.6.3 Envoi de message 146
6.6.4 Compilation et exécution 149
6.6.5 Un exemple : notre équilibrage de charge 150
6.7 Résultats 150
6.8 Discussion 152

Conclusion **152**

Table des figures

1.1 Géométrie du transfert énergétique. . . . 17
1.2 Géométrie du transfert énergétique. . . . 18
1.3 Facteur de forme. . . . 20
1.4 Pseudo code pour la procédure de récolte (gathering). . . . 21
1.5 Pseudo code pour pour la procédure d'émission (shooting). . . . 22
1.6 Hiérarchie et interactions. . . . 23
1.7 Structures de données. . . . 24
1.8 Pseudo code du raffinement. . . . 24
1.9 Pseudo code de RésoudreSystème. . . . 25
1.10 Pseudo code de **RécolteRad**. . . . 26
1.11 Pseudo code de **DécompositionReconstruction**. . . . 26
1.12 Pseudo code de **Oracle1**. . . . 27
1.13 Pseudo code de **HiérarchiqueRad**. . . . 28
1.14 Pseudo code de **RaffinerLien**. . . . 29
1.15 Pseudo code de **Oracle2**. . . . 29

2.1 Zone de visibilité exacte d'une cellule (image de gauche), cellules visibles de la cellule $\mathcal{A}$ (image de droite). . . . 31
2.2 Découpage binaire d'une scène. . . . 32
2.3 Tri des polygones de la scène selon leur orientation. . . . 32
2.4 Section d'un volume englobant. . . . 33
2.5 Algorithme de subdivision binaire. . . . 34
2.6 Droite passant par les ouvertures de plusieurs cellules. . . . 35
2.7 Représentation d'une droite dans un espace dual. . . . 38
2.8 Nuage de points. . . . 39
2.9 Transformée d'un polygone. . . . 40
2.10 Structure de données correspondant à un polygone. . . . 40
2.11 Structure données correspondant à la scène. . . . 40
2.12 Création de l'arbre binaire selon θ. . . . 42
2.13 Classification des points dans le repère dual selon ρ. . . . 42
2.14 Structure de données permettant de représenter l'arbre binaire. . . . 43
2.15 Classification des points dans le repère dual selon θ. . . . 44
2.16 Propagation de la création de l'arbre binaire. . . . 45
2.17 Plan de découpage. . . . 46
2.18 Structure de données correspondant à un nœud de l'arbre binaire. . . . 46
2.19 Repère local à un plan de découpage. . . . 46
2.20 Repère local à un plan de découpage (ou à la droite de découpage associée). . . . 47
2.21 Segment (Min, Max). . . . 47
2.22 Algorithme de création d'une pièce rectangulaire. . . . 49
2.23 Origines face-à-face. . . . 49
2.24 Pièces rectangulaires quelconques. . . . 50
2.25 Algorithme d'extraction de pièces ayant 2 murs parallèles. . . . 50
2.26 Pièces à deux murs parallèles. . . . 51
2.27 Pièces quelconques. . . . 51
2.28 Suivi de contours. . . . 52
2.29 Algorithme décrivant le suivi de contours. . . . 53
2.30 Pièces adjacentes. . . . 54
2.31 Pièces entre deux murs. . . . 54

2.32 Propagation de la recherche des cellules. 54
2.33 Fusion de deux plans de découpage. 55
2.34 Murs et plans de découpage. 56
2.35 Ouvertures et plans de découpage. 56
2.36 Algorithme de découpage d'un polygone intersectant un plan. 57
2.37 Orientation des segments d'ouverture. 58
2.38 Algorithme de création d'une ouverture. 59
2.39 Graphe d'adjacence. 59
2.40 Création du graphe d'adjacence. 60
2.41 Graphe de visibilité. 60
2.42 Utilisation du programme. 61
2.43 Fenêtre de travail. 62
2.44 Fenêtre de visualisation. 63
2.45 Le premier étage du Soda Hall découpé par notre algorithme. 64
2.46 Espace dual de *Castle*: un pic représente plusieurs polygones et la hauteur du pic correspond à la somme des aires des polygones associés. 65
2.47 La scène Castle découpée à l'aide de l'interface graphique. L'image du haut illustre la scène avec les ouvertures de chaque cellule. L'image du bas correspond à la visualisation du fichier obtenu. Une couleur différente est assignée à chaque cellule. 66

3.1 Relations de visibilité : (a) scène de départ, (b) zone de visibilité de la cellule $\mathcal{A}$, (c) cellules visibles depuis $\mathcal{A}$, (d) zone de visibilité d'un objet de la cellule $\mathcal{A}$. 67
3.2 Calcul de volumes englobants à l'aide de PPPs. Les images (a), (b) et (c) représentent trois PPPs. L'image (d) illustre le groupe et tous les PPPs et l'image (e) représente le groupe dans son volume englobant. 69
3.3 Création de volumes englobants à l'aide de PPPs. 70
3.4 Création d'une hiérarchie avec un algorithme de type découpage au milieu. 70
3.5 Création d'une hiérarchie de groupes. 71
3.6 Insertion d'un groupe dans un arbre binaire. 72
3.7 Création d'un volume de visibilité. 73
3.8 Droite passant par les ouvertures de plusieurs cellules. 74
3.9 Eclairement d'un groupe de polygones. 74
3.10 (a) Maillage des bords d'une boîte englobante, (b) Calcul d'un cœfficient de réflectance et de transmission pour une maille. 75
3.11 Liens α et β. 75
3.12 Transmission d'énergie à travers un groupe. 76
3.13 Simulation d'éclairage de Teller dans des environnements complexes. 76
3.14 Les nuées dynamiques en 2D. 78
3.15 Algorithme des nuées dynamiques. 79
3.16 Calcul des groupes : avec la distance au barycentre du polygone. 81
3.17 Calcul des groupes : avec une autre distance. 81
3.18 Image de gauche : $dist(B_i, P) = \min_{i \in [1,n]}(dist(P, S_i))$; Image de droite : $dist(B_i, P) = dist(B_i, G_s)$; B_i étant le barycentre du groupe i, S_i étant le i^{eme} sommet du polygone P et G_s étant le barycentre de P. 82
3.19 Calcul des relations de visibilité pour un groupe de polygones. 83
3.20 Calcul des relations de visibilité pour un groupe de polygones. 84
3.21 Volume de visibilité : plans verticaux (Z est l'axe vertical du repère). 85
3.22 Volume de visibilité : plans passant par les arêtes horizontales de l'ouverture. 86
3.23 Déterminer un sommet pour construire un demi-espace. 86
3.24 Les demi-espaces verticaux. 87
3.25 Calcul d'un demi-espace à bord non vertical. 87
3.26 Objets visibles de G. 88
3.27 Test d'appartenance d'un groupe à un volume de visibilité. 89
3.28 Détection d'une ouverture dans un volume de visibilité. 90
3.29 Modification du volume de visibilité : cas simple. 90
3.30 Modification du volume de visibilité : cas particulier. 91
3.31 Détection du cas particulier. 91
3.32 Correction du cas particulier. 92
3.33 Menu fichier : chargement d'une scène, création des cellules et sortie ou sortie du programme. 93

3.34 Menu groupe : initialisation du nombre moyen de polygones par groupes, modification de ce nombre, création des groupes et des relations de visibilité. 94
3.35 Boîte de dialogue permettant de modifier le nombre moyen de polygones par groupes. . . 94
3.36 Groupes de polygones créés avec un nombre moyen de polygones par groupe égal à : 5 (haut et gauche), 20 (haut et droit), 60 (bas et gauche), 150 (bas et droit). 97
3.37 Calcul des groupes : image de droite avec la distance au plus proche sommet, image de gauche avec la distance au barycentre du polygone. 98
3.38 Objets visibles (en vert) d'un groupe de polygones (en rouge). 98

4.1 Hiérarchie de pièces et de cellules. 101
4.2 Commande de création de murs, inclusion d'objet, homothétie et insertion de pièces dans un bâtiment. 102
4.3 Fichier de description d'une cellule. 103
4.4 Interface graphique de notre modeleur : module bâtiment. 105
4.5 Interface graphique de notre modeleur : module pièce. 105
4.6 La boîte de dialogue permettant de créer une ouverture. 106
4.7 Visualisation d'un bâtiment en fil de fer. 106

5.1 Simulation d'éclairage de Teller dans des environnements complexes. 109
5.2 Architecture du système. 111
5.3 Algorithme général. 112
5.4 Algorithme d'ordonancement. 113
5.5 Algorithme glouton. 114
5.6 Evaluation des coûts à moyen terme. 115
5.7 Algorithme du retour-arrière. 116
5.8 Evaluation des coûts à long terme (pour des raisons de lisibilité, tous les arcs ne sont pas dessinés). 117
5.9 Algorithme du voyageur de commerce. 118
5.10 Temps de calcul pour la i^{eme} cellule choisie, nombre de liens, nombre de mailles en mémoire pour la stratégie de retour-arrière C. 121
5.11 Coûts d'Entrées/Sorties. 123
5.12 Vue de haut, avant simulation d'éclairage. 124
5.13 Vue de haut, après simulation d'éclairage. 124
5.14 Vue intérieure. 125

6.1 Principe de notre algorithme parallèle. 131
6.2 Simulation d'éclairage sur un processeur. 133
6.3 Collecte d'énergie parallèle pour un groupe de groupes. 133
6.4 Exemple de découpage d'un graphe de visibilité. 134
6.5 Découpage binaire d'un graphe. 134
6.6 Regroupement de groupes. 135
6.7 Fichier contenant la desription des sous-graphes. 136
6.8 Organisation de l'espace disque. 137
6.9 Equilibrage de charge dynamique. 138
6.10 Problème de la prévision à long terme et de l'équilibrage de charge dynamique. 139
6.11 Algorithme de terminaison. 141
6.12 Problème de congestion. 141
6.13 Initialisation et terminaison de l'environnement MPI. 147
6.14 Envoi et reception de messages bloquants avec MPI. 148
6.15 Demande de groupe par un processeur. 151

Liste des tableaux

2.1 Paramètres par défaut. 62
2.2 Statistiques de la structuration par la méthode de J. Airey. 63
2.3 Statistiques obtenues en utilisant notre méthode. 63
2.4 Statistiques obtenues sur l'étage modélisé à l'IRISA. 64

3.1 Tableau récapitulatif des résultats obtenus. 95

4.1 Utilisation de l'espace disque selon les formats de fichiers. 105

5.1 Temps de calcul pour nos huit stratégies d'ordonnancement. 120

6.1 Caractéristiques des machines utilisées. 150
6.2 Résultats. 150

Introduction

Les techniques de simulation d'éclairage telles que la méthode de radiosité ont pour objectif de reproduire le plus fidèlement possible la propagation des ondes lumineuses à l'intérieur d'un environnement. Elles sont basées sur la théorie de la physique ondulatoire et prennent en compte les propriétés de la lumière et des matériaux. Pour cela, un modèle mathématique appelé *modèle d'illumination globale* décrit les transferts radiatifs entre les surfaces et permet d'exprimer très précisément les différentes grandeurs radiométriques entrant en jeu dans le processus d'éclairage telles que la luminance, la radiosité ou encore l'éclairement. Cela constitue une grande différence avec les techniques classiques de synthèse d'images dont l'objectif est simplement de créer des images donnant une impression de réalisme à partir d'un environnement tridimensionnel.

Le travail que nous avons réalisé au cours de cette thèse concerne l'application de la méthode de radiosité à des environnements architecturaux complexes tels que des bâtiments composés de plusieurs centaines de pièces (pouvant comporter plusieurs millions de surfaces). Notre étude a donné lieu à deux algorithmes de simulation d'éclairage (séquentiel et parallèle), nécessitant de découper l'environnement en plusieurs régions de l'espace appelées *cellules*, ceci afin de faire face aux contraintes liées au stockage de l'environnement en mémoire. Or les seules techniques de découpage existantes sont basées sur une subdivision binaire de l'espace (appelée BSP) et ne sont exploitables que pour des environnements ayant des propriétés particulières, comme nous le verrons au cours de ce rapport. C'est pourquoi nous avons introduit une nouvelle méthode de découpage, utilisant un espace dual à partir duquel nous regroupons les polygones verticaux quasiment alignés dans l'environnement. A chaque groupe de polygones verticaux nous associons un *plan de découpage* utilisé dans l'étape d'extraction de cellules 3D. Cette étape est basée sur des modèles génériques de cellules mis en correspondance (à l'aide d'une base de règles) avec des éléments géométriques de la scène. Lorsque toutes les cellules sont créées, nous effectuons des calculs de visibilité afin de construire un graphe (appelé *graphe de visibilité*) dans lequel un nœud correspond à une cellule et un arc entre deux nœuds indique que les deux cellules correspondantes sont mutuellement visibles. L'avantage majeur de notre approche est qu'elle peut être utilisée pour tout type de bâtiment, contrairement à la méthode BSP.

Cependant, la zone visible à partir d'une cellule est vaste et peut contenir plusieurs autres cellules. Pour affiner ces relations de visibilité, nous proposons d'utiliser la notion de groupes de surfaces ou d'objets (ou encore *cluster* en Anglais). Plusieurs techniques de regroupement de surfaces existent dans la littérature mais sont toutes basées sur la construction d'une hiérarchie de volumes englobants. Pour réaliser un regroupement plus précis, nous avons conçu un nouvel algorithme, utilisant une méthode de classification de type nuées dynamiques (ou *k-means*) issue des techniques d'analyse de données. Dans le cas d'environnements complexes, nous utilisons la structuration de l'environnement en cellules afin d'accélérer les calculs. Puis pour chaque groupe de surfaces obtenu, nous déterminons l'ensemble de ses groupes visibles. Il en résulte un graphe dans lequel un nœud est associé à un groupe et un arc entre deux nœuds indique que les groupes correspondants sont mutuellement visibles. Ce graphe est également appelé *graphe de visibilité*.

La seule technique de simulation d'éclairage pouvant être appliquée à des environnements com-

plexes est due à S. Teller et. al. [1]. Leur approche consiste à effectuer les calculs seulement sur une petite région de l'environnement. Cette dernière est stockée en mémoire alors que la plupart des informations concernant la description géométrique et photométrique de la scène sont stockées sur le disque de la machine. Lorsque le programme a besoin de traiter une nouvelle région, il peut la lire sur le disque après avoir rapatrié un certain nombre de surfaces pour lesquels les calculs ont déjà été effectués. Cependant, cette technique implique de nombreux échanges entre le disque et la machine. C'est pourquoi Teller et. al. proposent quatre stratégies d'ordonnancement des calculs pour les différentes parties visant à réutiliser au maximum les données en mémoire. Or aucune de ces stratégies ne prend en compte explicitement les coûts des accès au disque. C'est pourquoi nous avons effectué une étude complémentaire à celle de Teller dans le but de proposer de nouvelles stratégies d'ordonnancement, prédisant ces coûts à court, moyen ou long terme. Ces stratégies sont issues de la théorie des graphes parmi lesquelles nous avons utilisé par exemple la fermeture transitive, des algorithmes gloutons, retour-arrière, ou encore voyageur de commerce. Parmi les sept stratégies que nous proposons, l'une d'entre elle semble être plus performante que toutes les autres en termes de temps de calcul, c'est la stratégie du voyageur de commerce dont l'objectif est de prévoir les coûts d'accès au disque à long terme.

Grâce aux stratégies d'ordonnancement mises en œuvre, il est possible d'accélérer notablement le processus de simulation d'éclairage sur un seul processeur. Malheureusement, pour des environnements complexes, les temps de calcul restent tout de même prohibitifs. Une solution à ce problème est d'effectuer les calculs en parallèle sur plusieurs processeurs. Or les seuls algorithmes de radiosité hiérarchique parallèles traitant les environnements complexes sont de type maître-esclave et impliquent de nombreux messages ayant une taille importante. Afin de réduire la taille des messages et leur nombre, nous proposons une nouvelle approche de simulation d'éclairage parallèle de type SPMD (Single Program Multiple Data). Notre objectif est d'exécuter notre programme sur un réseau hétérogène de machines, sur une machine parallèle ou les deux ensemble. Pour cela, nous avons utilisé les fonctionalités de la librairie MPI (Message Passing Interface). D'autre part, nous supposons également que la base de données, même répartie ne peut être entièrement stockée en mémoire. Par conséquent, chaque processeur effectue les calculs de radiosité hiérarchique selon notre stratégie d'ordonnancement du voyageur de commerce, car c'est la plus efficace. Les données sont stockées sur un disque commun à tous les processeurs et pour résoudre les problèmes de cohérence, chaque processeur dispose d'un répertoire local dans lequel il peut écrire les fichiers résultant de ses calculs. D'autre part, nous avons également mis en œuvre une technique d'équilibrage de charge dynamique de type "vol de tâche" (ou *task stealing* en Anglais). A la fin de chaque itération de calcul, les processeurs se synchronisent à l'aide d'une reconfiguration dynamique des processeurs en anneau dans lequel circule un jeton. Cette synchronisation permet (i) de mettre à jour la base de données sur le disque, (ii) de gérer efficacement la terminaison du programme et (iii) de détecter la convergence de l'algorithme de radiosité. Enfin, nous avons apporté un preuve théorique de la convergence de notre algorithme de radiosité parallèle.

Afin de disposer de plusieurs environnements complexes pour effectuer nos tests, nous avons également mis en œuvre un outil permettant de modéliser un environnement complexe de manière interactive. Cet outil de modélisation a été mis en œuvre à l'aide des bibliothèques Motif (pour l'interface homme-machine) et OpenGL (pour l'affichage de la scène en 3D). L'utilisateur est guidé dans la modélisation de manière à structurer l'environnement dès sa création.

Enfin, nous avons conçu un programme permettant de visualiser de manière interactive des environnements complexes avant, pendant, ou après les calculs de simulation d'éclairage. Ce programme utilise le graphe de visibilité construit au cours de l'étape de structuration afin de réduire le nombre de surfaces transmises au pipeline graphique à l'affichage.

Le plan de cette thèse est le suivant. Le premier chapitre rappelle les principes de la radiosité hiérarchique et introduit les notions radiométriques liées au processus de simulation d'éclairage. Dans le second chapitre, nous présentons notre méthode de structuration d'environnements complexes avant de décrire notre regroupement de surfaces dans le chapitre 3. Notre outil de modélisation est ensuite présenté dans le chapitre 4 et nous décrivons nos algorithmes de simulation d'éclairage séquentiel et parallèle dans les chapitres 5 et 6 avant de conclure.

Principales contributions

Voici les principales contributions que nous avons apportées au cours de cette thèse :

- Une nouvelle méthode de découpage d'environnements architecturaux, basée sur la notion de modèle et respectant la topologie des environnements ;
- Une technique de regroupement de surfaces originale utilisant une méthode de classification issue de l'analyse de données ;
- L'enrichissement des calculs de radiosité avec E. Zeghers : meilleure prise en compte des ombres portées, optimisation de l'espace mémoire utilisé et implémentation des sources spectrales et directionnelles ;
- Une étude approfondie sur plusieurs stratégies d'ordonnancement des calculs de radiosité afin de réduire les accès au disque de la machine ;
- L'application de nos algorithmes en parallèle de manière efficace : réduction de la taille et du nombre de messages, algorithmes d'équilibrage de charge dynamique, reconfiguration dynamique des processeurs en anneau et algorithme de terminaison efficace.

Chapitre 1

Modèle d'illumination globale et méthode de radiosité

1.1 Problématique

La méthode de radiosité [2], [3] permet de simuler la propagation de rayonnements lumineux à l'intérieur d'un environnement. Grâce à cette technique, il est possible de quantifier précisément les différentes grandeurs radiométriques telles que la luminance, l'éclairement ou la radiosité. A l'aide de ces grandeurs, les éclairagistes peuvent évaluer le confort visuel (éblouissements, fatigue visuelle, etc.). Ceci n'est possible que grâce à l'élaboration d'un modèle d'illumination globale, à une approche physique, à l'utilisation d'outils mathématiques rigoureux et enfin à la conception et la mise en œuvre d'algorithmes efficaces en termes de temps de calcul et de mémoire. Le champ d'application de ces techniques de simulation d'éclairage est vaste : éclairage public et d'édifices historiques, confort visuel (bureaux ou ateliers), dimensionnement de l'éclairage des tunnels, de grandes surfaces, studio de cinéma, etc. Elles peuvent également jouer un rôle important dans la vérification de normes d'éclairage artificiel ou dans la simulation de luminaires avant fabrication. De plus, l'extension de ces techniques à des rayonnements non visibles tels que l'infrarouge est immédiate.

Cependant, de nombreuses structures de données doivent être mises en place afin de représenter précisément les multiples interréflexions lumineuses dans un environnement. Par exemple, les surfaces de la scène doivent être maillées en *éléments de surfaces*. Dans le cas de la radiosité hiérarchique [4], [5], [6], [7] ce maillage est stocké sous forme d'arbre quaternaire. D'autre part, l'emploi d'une grille régulière 3D peut accélérer les calculs permettant de déterminer les relations de visibilité entre les mailles des surfaces. D'autre part, pour éviter d'avoir à effectuer ces calculs plusieurs fois, des liens sont stockés.

Dans ce chapitre, nous verrons comment rendre compte précisément des échanges lumineux à l'intérieur d'un environnement. Ceux-ci sont décrits par une équation intégrale pour laquelle il n'existe pas de solution pratique qui soit exacte. Dans la littérature, les différentes techniques proposées conduisent à un système linéaire de la forme $Ax = b$. Ce système est alors résolu en utilisant soit une méthode numérique de type Jacobi, Gauss-Seidel (*gathering*) ou Southwell (*shooting*).

1.2 Grandeurs photométriques

Nous rappelons dans cette section les grandeurs photométriques apparaissant dans un modèle d'illumination globale. Afin d'introduire les différentes grandeurs photométriques entrant en jeu dans le processus de simulation d'éclairage, nous utilisons la figure 1.1 où la surface S_1 émet de

l'énergie vers la surface S_2.

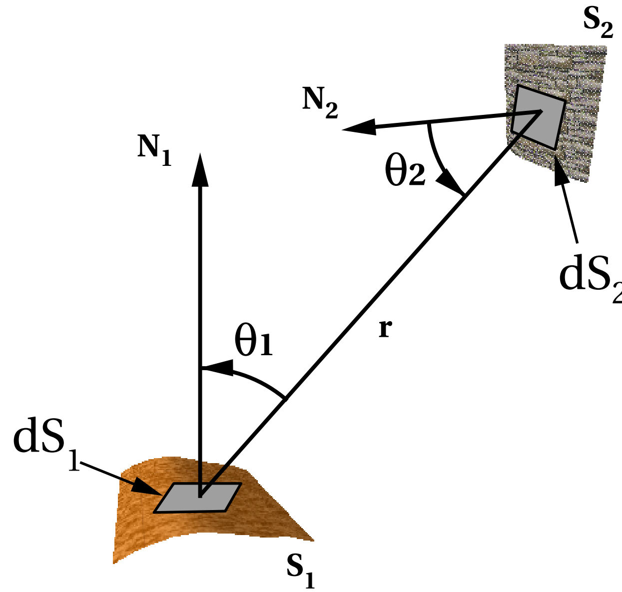


FIG. 1.1 – *Géométrie du transfert énergétique.*

- **flux** : énergie émise par une surface (ou atteignant une surface) par unité de temps.
- **luminance** : flux émis par unité de surface et par unité d'angle solide projeté. Le flux transmis par dS_1 à dS_2 est :

$$d^2\Phi = L_1 \cos\theta_1 dS_1 d\Omega_1$$

or

$$d\Omega_1 = \frac{\cos\theta_2 dS_2}{r^2}$$

et par conséquent

$$d^2\Phi = \frac{L_1 \cos\theta_1 \cos\theta_2 dS_1 dS_2}{r^2}$$

L_1 est la luminance de dS_1 (*l'oeil n'est sensible qu'à la luminance*).

- **intensité** : flux émis par unité d'angle solide. Elle est donnée par :

$$I = \frac{d\Phi}{d\Omega_1}$$

- **émittance ou radiosité** : c'est flux émis par unité de surface donné par :

$$B = \frac{d^2\Phi}{dS_1} = L_1 \cos\theta_1 d\Omega_1$$

- **éclairement** : flux reçu par unité de surface. Il est donné par :

$$A = \frac{d^2\Phi}{dS_2} = L_1 \cos\theta_2 d\Omega_2$$

avec

$$d\Omega_2 = \frac{\cos\theta_1 dS_1}{r^2}$$

1.3 Reflectivité et réflectance

L'énergie reçue par une surface est en partie réémise vers l'environnement et en partie absorbée. Deux coefficients expriment le rapport entre le flux réfléchi et le flux incident. Ces deux coefficients sont :

- **réflectivité** : c'est le rapport entre le flux réfléchi et le flux incident pour un petit élément de surface. C'est le rapport entre la radiosité et l'éclairement :

$$\rho = \frac{d^2\Phi_r}{d^2\Phi_i} = \frac{dB_r}{dA_i}$$

- **réflectance bidirectionnelle** : c'est le rapport entre le flux réfléchi dans la direction D_r et le flux incident provenant de la direction D_r pour un petit élément de surface. Elle s'exprime comme le rapport entre la luminance et l'éclairement :

$$R(D_i, D_r) = \frac{dL_r}{dA}$$

La relation entre ρ et R est :

$$\rho(D_i) = \frac{d^2\Phi_r}{d^2\Phi_i} = \int_{2\pi} R(D_i, D_r) \cos \Phi_r d\Omega_r$$

1.4 L'équation de luminance

La figure 1.2 illustre la géométrie du transfert énergétique.

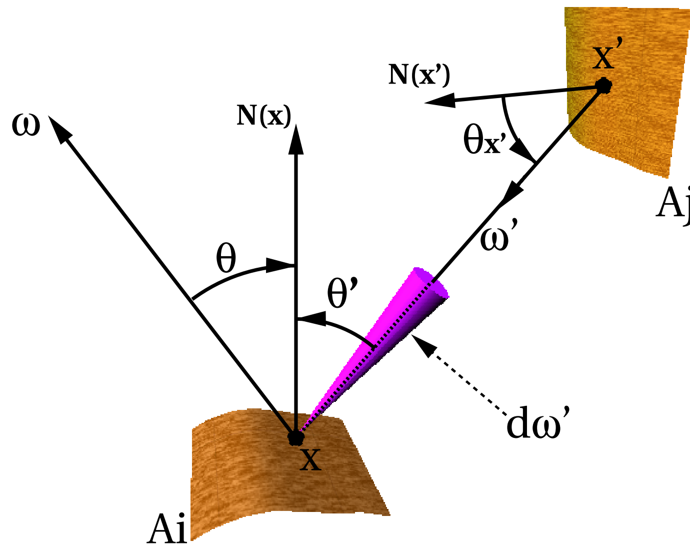


FIG. 1.2 – *Géométrie du transfert énergétique.*

Soit $\mathcal{M}$ l'espace des points sur les surfaces de l'environnement
$\mathcal{S}$ l'espace des directions angulaires sur la sphère unité.
χ l'espace des fonctions à valeurs réelles définies sur $\mathcal{M} \times \mathcal{S}$

L'équation de luminance dans un modèle d'illumination globale est donnée par :

$$L(x,\omega) = Le(x,\omega) + \int_{\Omega_i} k(x,\omega' \rightarrow \omega) L(x',\omega') cos\theta' d\omega'$$

$L, Le \in \chi$ sont les fonctions de luminance et d'auto-émission respectivement.

$$\begin{cases} \Omega_i & \text{hémisphère des directions entrantes} \\ k & \text{fonction de réflectance bi-directionnelle} \\ \theta' & \text{angle du rayon incident contenu dans l'angle solide } d\omega' \\ x' & \text{point sur une surface distante déterminé par } \omega' \text{ et } x \\ \omega \in \Omega_o & \Omega_o \text{ hémisphère des directions sortantes} \end{cases}$$

La luminance en un point x de l'environnement et dans une direction ω est égale à la luminance auto-émise en ce point et dans cette même direction ω à laquelle on ajoute la contribution lumineuse parvenant en x de chaque point x' de l'environnement $(L(x',\omega'))$ pondérée par la fonction de réflectance bi-directionnelle. Cette fonction traduit la proportion d'énergie provenant de cette direction ω' et réfléchie en x selon la direction ω.

Cette équation intégrale représente un système d'équations de dimension infinie. Il n'existe malheureusement pas de solution analytique à cette équation (ou ce système) dans un environnement non trivial. C'est pourquoi, il convient de développer des méthodes de résolution qui approcheront le plus correctement possible la solution théorique du système.

1.5 De l'équation de luminance à l'équation de radiosité

Puisque les surfaces sont supposées lambertiennes (réflecteurs et émetteurs), nous avons :

$$L(x,\omega) = \frac{B(x)}{\pi}, \text{ et } Le(x,\omega) = \frac{E(x)}{\pi}$$

$$k(x,\omega' \rightarrow \omega) = \frac{\rho(x)}{\pi}$$

avec $B(x)$ la radiosité au point x, $E(x)$ la radiosité auto-émise par le point x, et $\rho(x)$ la réflectivité en x. Il est important de remarquer que $\rho(x)$ ne dépend pas de x' et peut donc être sorti de l'intégrale.

Nous avons de plus la relation classique reliant l'angle soustendu par la source à son aire projeté donnée par (voir figure 1.2) :

$$d\omega' = \frac{cos\theta_{x'} dA_j}{r^2_{xx'}}$$

avec $r_{xx'}$ la distance de x à x' : $(r_{xx'} = ||x - x'||)$

Nous pouvons alors écrire l'équation de radiosité:

$$B(x) = E(x) + \rho(x) \int_{A_j} B(x') \frac{cos\theta' cos\theta_{x'} h(x,x')}{\pi r^2_{xx'}} dAj \tag{1.1}$$

avec $h(x,x') = 1$ si les points x et x' sont mutuellement visibles, $h(x,x') = 0$ sinon.

1.6 Radiosité

Lorsque les surfaces de l'environnement sont subdivisées en carreaux suffisamment fins pour que la radiosité $B(x)$ puisse être assurée d'être constante sur la totalité du carreau, nous pouvons déduire *l'équation de radiosité constante discrète* suivante, valide pour chaque échantillon de

longueur d'onde. Cette équation représente la forme très classique du modèle de radiosité et nous allons maintenant étudier celle-ci en détail en vue d'introduire une approche hiérarchique.

Nous avons :

$$B_i = E_i + \rho_i \sum_{j=1}^{N} F_{ij} B_j,$$

où

- B_i : Radiosité du carreau i;
- E_i : Radiosité auto-émise par le carreau i;
- ρ_i : réflectivité du carreau i;
- F_{ij} : facteur de forme donnant la fraction d'énergie quittant le carreau i et arrivant sur le carreau j;
- N : nombre de carreaux.

Ce système d'équations représente les échanges énergétiques entre les surfaces dans un environnement. La solution de ce système donne une représentation discrète de la fonction de radiosité dans la scène. Cette solution est indépendante du point de vue. Une fois le système d'équations résolu, les radiosités de chaque sommet d'un carreau sont évaluées par la moyenne des radiosités des carreaux adjacents. L'image de la scène est alors calculée en appliquant un lissage de Gouraud [8, 2].

L'expression du facteur de forme est donné par :

$$F_{ij} = \frac{1}{\pi A_i} \int_{A_i} \int_{A_j} \frac{h(\bar{x}_i, \bar{x}_j) \cos\theta_i \cos\theta_j}{r^2} dA_i dA_j,$$

où $\bar{x}_i$ est un point du carreau i, et A_i l'aire de ce carreau. Le terme $h(\bar{x}_i, \bar{x}_j)$ exprime la visibilité entre un point du carreau i et un point du carreau j.

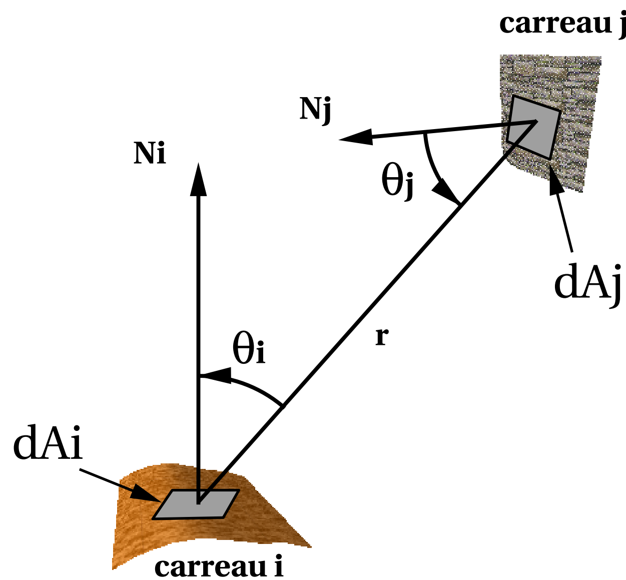


FIG. 1.3 – *Facteur de forme.*

Le facteur de forme entre un élément différentiel du carreau i (autour du point $\bar{x}_i$) et du carreau j (figure 1.3) est donné par :

$$F_{dA_iA_j} = \frac{1}{\pi}\int_{A_j} \frac{h(\bar{x}_i, \bar{x}_j)\cos\theta_i \cos\theta_j}{r^2} dA_j$$

Notons que si deux carreaux sont assez éloignés, ce facteur de forme est une bonne estimation pour F_{ij}. Pour calculer F_{ij}, le carreau i est subdivisé en R petits éléments dA_i^q (similaires à des éléments différentiels) et tous les facteurs de forme $F_{dA_i^qA_j}$ sont évalués. F_{ij} est alors égal à :

$$F_{ij} = \frac{1}{A_i}\sum_{q=1}^{R} F_{dA_i^qA_j} dA_i^q$$

1.7 Résolution du système d'équations

Les facteurs de forme doivent être calculés pour chaque paire de carreaux ce qui résulte en une complexité mémoire et temps en $O(n^2)$. La très grande place mémoire requise pour le stockage de ces facteurs de forme limite en pratique l'algorithme de radiosité. Cette difficulté peut être traitée par une approche de type radiosité progressive [3].

Dans l'approche conventionnelle de la radiosité, le système d'équations est résolu en utilisant une méthode itérative Gauss-Seidel. À chaque pas, la radiosité d'un unique carreau est remis à jour à partir des radiosités courantes de tous les autres carreaux. Ainsi à chaque étape, la radiosité provenant de tous les autres carreaux est récoltée dans un unique carreau récepteur (voir figure 1.4). La convergence est atteinte lorsque $\| B^{k+1} - B^k \|_\infty$ devient plus petit qu'un certain seuil, k étant le nombre d'itérations. B^k est le vecteur radiosité calculé à l'itération k.

```
pour tout i faire {
        pour chaque longueur d'onde faire
                B_i = E_i ;
}
Tant que non convergence {
        pour tout i faire {
                pour toute longueur d'onde faire
                        B_i = E_i + ρ_i Σ_{j=1,j≠i}^{n} F_ij B_j ;
        }
}
```

FIG. 1.4 – *Pseudo code pour la procédure de récolte (gathering).*

Dans l'approche radiosité progressive, la solution est obtenue par l'algorithme donné par la figure 1.5.

A chaque pas, la radiosité émise par un unique carreau est distribuée à tous les autres carreaux de la scène. Durant les premiers pas, ce sont les carreaux des sources lumineuses qui sont choisis pour émettre leur énergie puisque les autres carreaux ont très peu d'énergie. Aux pas suivants on choisit les sources secondaires, en commançant par les surfaces qui ont reçu le plus d'énergie directement des sources lumineuses, et ainsi de suite. Des images peuvent alors être produites très tôt dans un calcul de radiosité progressive. Notons qu'à chaque étape, seule une colonne du

```
pour tout i {
    pour chaque longueur d'onde
        ΔB_i = E_i ;
}
Tant que non convergence {
    j = carreau-de-max-delta-flux() ;
    pour tout i faire {
        pour toute longueur d'onde faire {
            ΔRad = ρ_i ΔB_j F_ji A_j/A_i ;
            ΔB_i = ΔB_i + ΔRad ;
            B_i = B_i + ΔRad;
        }
    }
    ΔB_j = 0 ;
}
```

FIG. 1.5 – *Pseudo code pour pour la procédure d'émission (shooting).*

système matriciel est calculée, évitant ainsi les problèmes de stockage mémoire.

Le critère de convergence est atteint si $\| \Delta B \cdot A \|_\infty$ est inférieur à un certain seuil. Ce seuil peut être un certain pourcentage du flux total émis par les sources. $(\Delta B \cdot A)$ est le vecteur représentant le flux non émis.

Remarquons que, suite à la convergence ou après réalisation d'un certain nombre d'itérations, des flux résiduels restent non émis. Les effets de ces résidus peuvent être approchés par un terme ambiant B_{amb} [3]:

$$B_{amb} = R \sum_{j=1}^{N} \Delta F_{*j} B_j,$$

où F_{*j} correspond à la contribution du carreau j à un autre carreau et R caractérise les multiples interréflexions :

$$F_{*j} = \frac{A_j}{\sum_{k=1}^{N} A_k}$$

$$R = 1 + \rho_{moy} + \rho_{moy}^2 + \rho_{moy}^3 + ... = \frac{1}{1 - \rho_{moy}},$$

où ρ_{moy} est la moyenne des réflectivités des objets et est donnée par :

$$\rho_{moy} = \frac{\sum_{k=1}^{N} \rho_k A_k}{\sum_{k=1}^{N} A_k}.$$

Afin de tenir compte des flux résiduels, la radiosité calculée est remise à jour comme :

$$B_i = B_i + \rho_i B_{amb}.$$

1.8 Radiosité hiérarchique

La radiosité hiérarchique a été introduite dans [4]. L'objectif de cette méthode est d'éviter un maillage trop fin des surfaces où il n'est pas nécessaire et de réduire le nombre de calculs de

facteurs de forme en subdivisant ces surfaces dans une hiérarchie (arbre quaternaire) d'éléments de surfaces (voir figure 1.6). Une feuille de la hiérarchie est appelée élément de surface (ou encore maille) et un nœud représente alors un groupe de mailles. Il existe une interaction entre un nœud A et un nœud B lorsque ces deux nœuds peuvent échanger de l'énergie. Un facteur de forme est calculé pour chaque interaction. Ceci signifie que l'on n'a pas à calculer un facteur de forme pour chaque paire de nœuds (ce qui est le cas dans une méthode de radiosité classique) mais seulement pour chaque paire de nœuds en interaction. En conséquence, le nombre de calculs de facteurs de forme est réduit de manière drastique ainsi que la place mémoire utilisée. Quand deux nœuds de différentes surfaces interagissent, un lien est créé entre eux.

Toutes les structures de données et algorithmes suivants, décrivant la radiosité hiérarchique, sont extraits de [9].

1.9 Structures de données

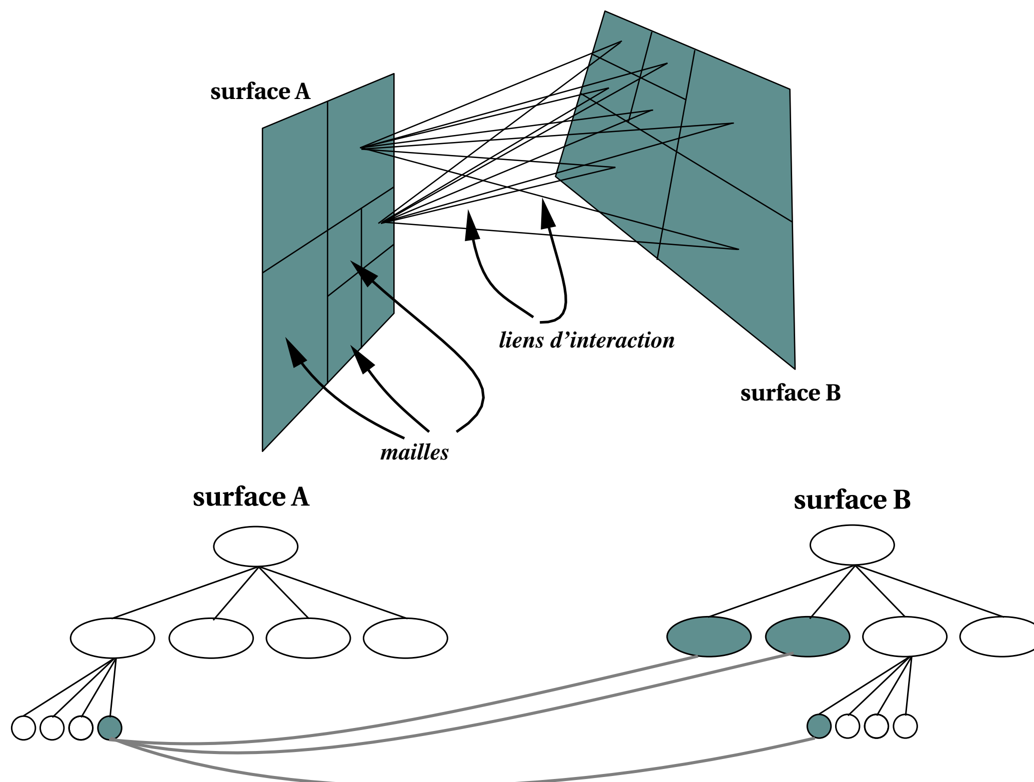


FIG. 1.6 – *Hiérarchie et interactions.*

La structure de donnée des arbres quaternaires représentant la hiérarchie et celle des liens sont données dans la figure 1.7.

1.10 Raffinement

Le raffinement consiste à subdiviser des surfaces en mailles, dans une hiérarchie d'éléments. Lorsque deux nœuds sont succeptibles d'interagir, un lien est créé entre eux. Le calcul de raffinement est réalisé par la procédure *Raffiner* décrite par la figure 1.8.

Le rôle de la fonction *oracle* est très important. Elle décide si deux nœuds peuvent être liés ou non. Rappelons qu'un nœud correspond soit à un élément soit à un groupe d'éléments. Nous

```
struct Noeudarbre {
        float B_g[ ] ;     /* récolte des radiosités aux échantillons λ */
        float B_s[ ] ;     /* émission des radiosités aux échantillons λ */
        float E[ ] ;     /* auto émittance aux échantillons λ */
        float aire;
        float ρ[ ] ;     /* réflectivité aux échantillons λ */
        struct Noeudarbre** fils;     /* pointeur vers la liste des 4 fils */
        struct Liennoeud* L;     /* premier lien de récolte du noeud */
}

struct Liennoeud {
        Noeudarbre* q;     /* noeud de récolte */
        Noeudarbre* p;     /* noeud d'émission */
        float F_qp;     /* facteur de forme de q à p */
        struct Liennoeud* next;     /* prochain lien de récolte de q */
}
```

FIG. 1.7 – *Structures de données.*

```
Raffiner(Noeudarbre *p, Noeudarbre *q, float F_ε) {
        Noeudarbre lequel, r ;

        si (oracle(p, q, F_ε)) alors Lien(p, q);
        sinon {
                lequel = Subdiv(p, q);
                si (lequel == q )
                        pour chaque noeud r de q Raffiner(p, r, F_ε);
                sinon si (lequel == p )
                        pour chaque noeud r de p Raffiner(r, q, F_ε);
                sinon
                        Lien(p,q);
        }
}
```

FIG. 1.8 – *Pseudo code du raffinement.*

verrons que si un lien est établi, le facteur de forme entre les deux nœuds associés peut être approché par un facteur de forme *aire différentielle-aire*. Pour faire cela, *oracle1* (voir figure 1.12) calcule un majorant des facteurs de forme *aire différentielle-aire* F_{pq} entre les carreaux p et q et F_{qp} entre les carreaux q et p. Il y a deux versions possibles de *oracle*: *oracle1* et *oracle2* (voir figure 1.15). La première utilise un critère de subdivision uniquement géométrique alors que la seconde utilise également un critère radiométrique basé sur le flux.

Subdiv(p,q) retourne vrai si les nœuds sous p et q (fils) doivent être utilisés. Elle retourne p si il apparaît que les nœuds de plus bas niveau de p vont satisfaire plus rapidement le critère de subdivision que les fils de q, sinon la procédure retourne q.

Lien(p,q) établit un lien entre p et q, calcule le facteur de forme entre p et q et l'enregistre dans une structure de données lien. Ce facteur de forme est en fait estimé comme un majorant du facteur de forme comme nous le montrerons un peu plus tard.

Après le traitement de chaque paire de surfaces par *Raffiner*, le résultat est un ensemble de liens reliants deux nœuds des différents arbres quaternaires.

1.11 Résolution du système hiérarchique

La procédure *RésoudreSystème* calcule la solution du système hiérarchique d'équations (figure 1.9).

RécolteRad récolte l'énergie transportée par chaque lien au nœud récepteur. L'énergie récoltée est stockée dans le champ B_g (figure 1.10). *RécolteRad* correspond à une méthode de résolution du système linéaire de type Jacobi.

DécompositionReconstruction (ou PushPull): le processus de décomposition transmet la radiosité récoltée aux descendants de chaque nœud récepteur, et la reconstruction remonte aux nœuds pères la moyenne des radiosités des quatre fils (voir figure 1.11), préparant ainsi les radiosités B_s pour la prochaine itération dans *RésoudreSystème*.

Le critère de convergence utilisé dans *RésoudreSystème* est satisfait lorsque la différence des valeurs de radiosité entre deux itérations devient plus petit qu'un seuil spécifié a priori par l'utilisateur.

```
RésoudreSystème() {

    Tant que non Convergence {
        pour toutes les surfaces p faire : RécolteRad(p);
        pour toutes surfaces p faire : DécompositionReconstruction(p, 0.0);
    }
}
```

FIG. 1.9 – *Pseudo code de RésoudreSystème.*

```
RécolteRad(Noeudarbre *p) {
        Noeudarbre *q;
        Liennoeud *L;

        p → B_g = 0;
        pour chaque lien de récolte L de p faire {
                /* récolte d'énergie par les liens */
                p → B_g += p → ρ * L → F_pq * L → q → B_s;
        }

        Pour chaque noeud fils r de p faire:
                RécolteRad(r);
}
```

FIG. 1.10 – *Pseudo code de* **RécolteRad**.

```
DécompositionReconstruction(Noeudarbre *p, float B_bas) {
        float B_haut, B_tmp;

        /* si p est une feuille */
        si (p → fils == NULL) alors
                B_haut = p → E + p → B_g + B_bas;
        sinon faire {
                B_haut = 0;
                pour chaque noeud fils r de p faire {
                        B_tmp = DécompositionReconstruction(r, p → B_g + B_bas)
                        B_haut += B_tmp * (r→aire)/(p→aire);
                }

                p → B_s = B_haut;
                retourne (B_haut);
        }
}
```

FIG. 1.11 – *Pseudo code de* **DécompositionReconstruction**.

1.12 L'Oracle

La fonction oracle prend la décision de lier ou non deux nœuds p et q. En fait, un lien est construit si le facteur de forme F_{pq} est assez petit pour considérer la contribution énergétique de p à q comme petite, ce qui revient à dire que la radiosité de q due à p peut alors être considérée comme constante sur q. Dans *Oracle1* (figure 1.12), première version de l'oracle, F_{pq} est estimé par la relation :

$$F_{pq} \approx \frac{\cos\theta}{\pi}\Omega_q,$$

où Ω_q est l'angle solide dont l'origine est le centre de p et qui est soutendu par le disque entourant q.

Cette estimation est en fait un majorant du facteur de forme entre une aire différentielle de p et le carreau q. En présence d'occlusions, cet estimé peut être pondéré par un coefficient donnant le pourcentage de visibilité entre p et q comme dans [4].

Remarquons que l'on peut obtenir une meilleure estimation en calculant le facteur de forme aire-aire avec une méthode de type Monte Carlo. Dans notre cas, ce calcul est effectué à l'aide de la méthode de lancé de rayons. Afin d'accélérer les calculs, nous utilisons une grille régulière. L'espace est alors subdivisé en voxels ne contenant que quelques polygones à la fois.

Oracle1 estime à la fois F_{pq} et F_{qp}. Si ces deux facteurs de forme sont plus grands qu'un seuil donné F_ϵ, alors le nœud (ou l'élément) correspondant au plus grand facteur de forme est subdivisé. Si seulement un des facteurs de forme est plus grand que F_ϵ, par exemple s'il s'agit de F_{pq} alors p est subdivisé. Lorsque les deux sont inférieurs à F_ϵ, un lien bidirectionnel est établi entre p et q.

```
float Oracle1(Noeudarbre *p, Noeudarbre *q, float Fε) {

      si (p → aire < Aε et q → aire < Aε) faire
            retourne (FAUX);
      si (EstimeFacteurForme(p, q) < Fε ) alors
            retourne (FAUX);
      sinon faire
            retourne (VRAI);
}
```

FIG. 1.12 – *Pseudo code de* **Oracle1**.

Remarquons que *Oracle1* utilise un critère de subdivision uniquement géométrique basé sur les facteurs de forme. Ceci peut conduire en définitive à un nombre important d'éléments fins dans la scène. Il est plus subtil d'utiliser un critère basé sur la quantité d'énergie transférée entre les deux nœuds. Si cette énergie est plus petite qu'un certain seuil, un lien est établi. Plus précisément, si $F_{pq} \cdot B_q \cdot A_q \leq BF_\epsilon$ un lien est établi. Puisque les radiosités ne sont pas connues a priori, l'algorithme de raffinement procède en utilisant un autre oracle *Oracle2* (figure 1.15).

1.13 Algorithme de radiosité hiérarchique

L'algorithme de radiosité hiérarchique est donné par la figure 1.13. En première passe de cet algorithme, *Raffiner* utilise *Oracle2* et établit des liens au plus haut niveau à moins que la surface émettrice soit une source de lumière. La plupart de ces liens sont construits bien que les radiosités émises par la plupart des surfaces soient nulles. Ces liens sont raffinés dans une seconde passe à travers *RaffinerLien* (figure 1.14).

```
HiérarchiqueRad(float BFε) {
        Noeudarbre *p, *q;
        Liennoeud *L;
        int Fait = FAUX;

        pour toutes surfaces p p → Bs = p → E;
        pour chaque paire de surfaces p, q
                Raffiner(p, q, BFε);
        tant que non Fait {
                Fait = VRAI;
                RésoudreSystème();
                pour tous les liens L {
                        /* RaffinerLien retourne FAUX si une subdivision se produit */
                        si (RaffinerLien(L, BFε) == FAUX) alors Fait = FAUX;
                }
        }
}
```

FIG. 1.13 – *Pseudo code de* **HiérarchiqueRad**.

1.14 Discussion

Dans ce chapitre, nous avons présenté les principes de la simulation d'éclairage par méthode de radiosité telle qu'elle est décrite dans la littérature par un certain nombre d'auteurs. Ces différents modèles et algorithmes ont été mis en œuvre et utilisés dans le cadre de la simulation d'éclairage dans des environnements complexes.

Nous l'avons vu, cette mise en œuvre implique de nombreuses structures de données : maillage des surfaces de la scène, stoké dans un arbre quaternaire, liens, grille régulière. Or le stockage de toutes ces structures de données nécessite un espace mémoire important et il n'est pas possible d'effectuer les calculs de radiosité pour des environnements complexes sans prétraitement.

Dans ce rapport de thèse, notre objectif est de fournir des solutions permettant néanmoins d'effectuer ces calculs de radiosité avec des machines modérément performantes et surtout pour des environnements composés de plusieurs millions de polygones, voire plus.

```
int RaffinerLien(Liennoeud *L, float BFε) {
        int pas_subdivision = VRAI;
        Noeudarbre lequel;
        Noeudarbre *p = L → p;    /* émetteur */
        Noeudarbre *q = L → q;    /* récepteur */

        si (Oracle2(L, BFε)) {
                pas_subdivision = FAUX;
                lequel = Subdiv(p, q);
                DétruireLien(L);
                si (lequel == q)
                pour chaque noeud fils r de q Lien(p,r);
                sinon
                        pour (chaque noeud fils r de p) Lien(r,q);
        }

        retourne (pas_subdivision);
}
```

FIG. 1.14 – *Pseudo code de* **RaffinerLien**.

```
float Oracle2(Liennoeud *L, float BFε) {
        Noeudarbre *p = L → p;    /* émetteur */
        Noeudarbre *q = L → q;    /* récepteur */
        si (p → aire < Aε and q → aire < Aε)
                retourne (FAUX);
        si (p → Bs == 0.0)
                retourne (FAUX);
        si ((p → Bs * p → Aire * L → Fpq) < BFε)
                retourne(FAUX);
        sinon
                retourne (VRAI);
}
```

FIG. 1.15 – *Pseudo code de* **Oracle2**.

Chapitre 2

Structuration de scènes

2.1 Problématique

Comme nous l'avons vu dans le chapitre 1, de nombreuses structures de données sont nécessaires pour mettre en oeuvre la méthode de radiosité hiérarchique (quadtree, grille régulière, liens, etc.). Le stockage de ces structures de données requiert des ressources mémoire considérables, même pour des environnements peu complexes (quelques centaines de polygones). Or cela devient un réel obstacle dans le cas d'environnements contenant plusieurs millions de polygones tels que des bâtiments meublés de plusieurs étages. Pour surmonter ce problème, une solution est de découper la base de données en plusieurs parties et d'effectuer les calculs successivement pour chacune de ces parties. Dans ce chapitre, notre objectif est de proposer une technique de découpage efficace. Nous verrons par la suite comment mettre en œuvre la méthode de radiosité pour des environnements complexes.

Dans le cas d'environnements architecturaux, seulement quelques objets contribuent à l'éclairement d'une surface car les murs et les sols provoquent de nombreuses occlusions. J. M. Airey et al. [10], [11] et Teller et. al. [12], [1] proposent donc de découper les bâtiments en cellules selon les murs, de manière binaire (méthode BSP ou Binary Space Partitioning). Cependant, à chaque étape du calcul une grande partie du bâtiment est découpée par un seul plan et les pièces sont morcelées en plusieurs petites cellules n'ayant aucun sens topologique. D'autre part, lorsque plusieurs murs sont quasiment alignés (ou quasi-alignés), la subdivision est effectuée successivement à l'aide de chacun d'eux (engendrant de trop nombreuses cellules), alors qu'il serait plus judicieux de leur associer un seul plan de découpage.

Afin d'apporter une solution à ces problèmes, nous avons conçu un nouvel algorithme permettant d'extraire les pièces et les couloirs d'un bâtiment. La première étape de cet algorithme consiste à regrouper des murs quasi-alignés à l'aide d'un espace dual [13] [14]. Dans un second traitement, des cellules sont extraites suivant plusieurs règles (correspondant à des modèles de pièces), basées sur une connaissance globale de l'environnement. Pour chaque cellule, nous déterminons une liste d'ouvertures correspondant à des portes ou des fenêtres. A l'aide de ces ouvertures, nous construisons un *graphe de visibilité* pour lequel un noeud correspond à une cellule et un arc relie deux noeuds lorsque leurs cellules associées sont mutuellement visibles. Ce graphe sera employé d'une part pour accélérer les calculs de radiosité et d'autre part pour fluidifer la visualisation interactive des environnements car il permet de connaître instantanément la liste de tous les objets potentiellement visibles (susceptibles d'éclairer une surface) d'une cellule donnée. Par exemple un objet de la cellule $\mathcal{A}$ sur la figure 2.1 (a) peut potentiellement être éclairé par les objets de la scène contenus dans la partie dessinée en blanc. Sur la figure 2.1 (b), la zone blanche représente les cellules visibles de $\mathcal{A}$.

Dans la section suivante, nous présentons brièvement les travaux déjà effectués dans le domaine

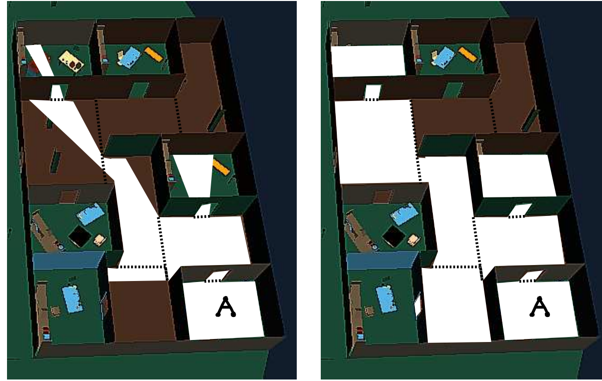


FIG. 2.1 – *Zone de visibilité exacte d'une cellule (image de gauche), cellules visibles de la cellule $\mathcal{A}$ (image de droite).*

du découpage d'environnements architecturaux. Puis nous décrivons le principe général de notre algorithme avant d'en détailler chaque point : le découpage de l'environnement, la recherche des ouvertures dans chaque cellule et les calculs de visibilité. Nous analysons et commentons ensuite les résultats obtenus à l'aide de notre méthode avant de conclure ce chapitre.

2.2 Travaux existants

Si peu d'auteurs se sont penchés sur le problème de la simulation d'éclairage dans des environnements architecturaux complexes, en revanche de nombreux algorithmes ont été proposés dans le but de réduire les calculs de visibilité, notamment dans le cas de la méthode de lancé de rayon [15], [16], [17], [18], [19].

2.2.1 Environnements architecturaux

Dans des environnements tels que les intérieurs de bâtiments, un certain nombre de connaissances peuvent être exploitées pour réduire le coût des calculs de visibilité [20] [21]. Dès 1970, C. B. Jones [22] décrit un algorithme d'affichage basé sur une structuration de la scène en cellules pour éliminer les surfaces cachées lors d'un affichage en fil de fer. L'utilisateur subdivise manuellement l'environnement en utilisant les facettes des objets. Le résultat de cette subdivision est un ensemble de régions dont les relations de visibilité sont représentées par un graphe. Dans ce graphe, un nœud correspond à une région et un arc représente une ouverture. Partant d'une région donnée (un nœud du graphe) et d'une direction de visée, le graphe est parcouru afin de déterminer toutes les régions visibles de l'observateur.

Subdivision binaire

La subdivision binaire (Binary Space Partitioning ou BSP) a été jusqu'à maintenant la seule technique permettant de structurer des bâtiments de manière automatique. Les premiers travaux exploitant cette technique dans le cadre d'environnements architecturaux complexes sont dus à J.M. Airey et. al. [10] [11]. Afin de faciliter les calculs, les auteurs supposent que les polygones occlusifs (murs, plafonds etc.) sont axiaux, cela signifie que leurs normales sont alignées avec l'un des axes du repère.

La subdivision BSP consiste à découper la scène en deux sous-scènes selon un *plan de découpage* (figure 2.2). Chaque sous-scène obtenue est à son tour subdivisée selon un nouveau plan et ce processus est répété de manière récursive tant qu'il reste des polygones suffisamment occlusifs. Le résultat obtenu est un arbre binaire dont les feuilles représentent de petites parties de la

scène : les *cellules*. La figure 2.2 illustre le découpage d'un environnement architectural à l'aide

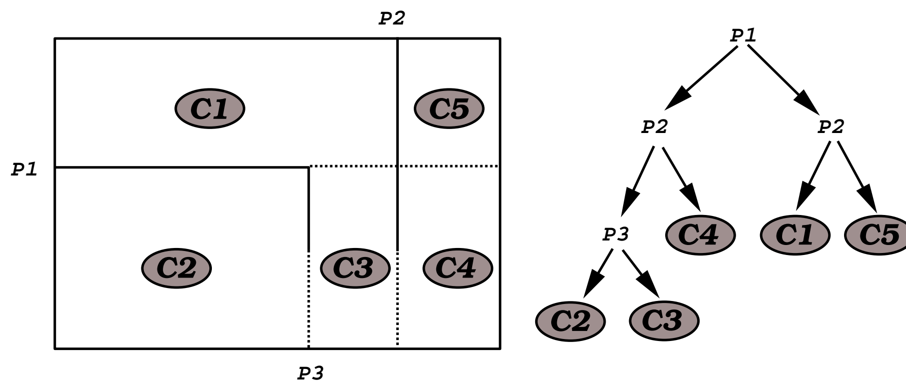


FIG. 2.2 – *Découpage binaire d'une scène.*

de la méthode BSP. Sur cette figure, le bâtiment est découpé en deux parties par le plan $P1$, puis chaque partie est à son tour découpée par un nouveau plan, etc. L'arbre binaire obtenu est représenté à droite de la figure.

Pour effectuer ce découpage, Airey propose de trier les polygones verticaux selon les 3 axes du repère de la scène (figure 2.3). Une liste de polygones est construite pour chaque axe. Les polygones dont la normale est colinéaire à O_x (resp. O_y ou O_z) sont placés dans la liste correspondante L_x (resp. L_y ou L_z). Ceux dont la normale n'est alignée avec aucun axe sont placés dans une liste annexe.

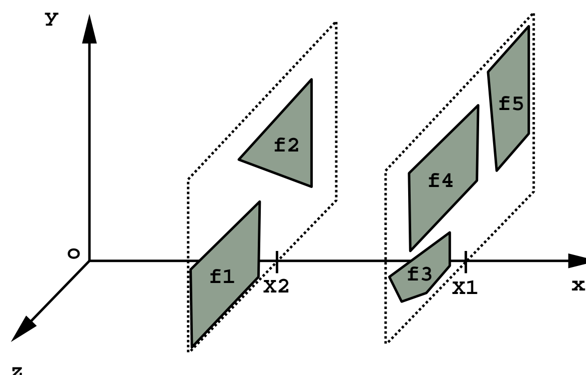


FIG. 2.3 – *Tri des polygones de la scène selon leur orientation.*

Pour découper une sous-scène, un plan de découpage est choisi de manière à atteindre les objectifs suivants :

- Suivre les murs afin que les cellules correspondent à des pièces. En effet, dans une même pièce tous les objets sont mutuellement visibles car il y a peu de polygones très occlusifs. Airey propose donc de suivre les murs en découpant les sous-scènes selon les plans les plus occlusifs.

- Equilibrer l'arbre binaire pour qu'il soit parcouru plus efficacement. Pour cela, le plan de découpage doit partager la sous-scène en deux volumes contenant le même nombre de plans occlusifs.

- Réduire le nombre de polygones découpés car le plan de découpage coupe souvent un grand nombre d'objets. Ces derniers sont découpés et cela entraîne un accroissement indésirable

de la base de données. Par conséquent, il est important de choisir un plan découpant peu de polygones.

Pour chacune de ces contraintes, il existe un plan de découpage dans la sous-scène courante. Cependant, Airey propose de déterminer un plan respectant au mieux toutes ces contraintes à la fois en attribuant une note à chaque plan occlusif de la sous-scène. Cette note est déterminée par une fonction linéaire dont les coefficients sont proportionnels à l'importance attribuée à chaque critère. Le plan ayant obtenu la meilleure note est choisi pour découper la sous-scène. Airey propose les coefficients de pondération suivants :

$$0.5*\textit{occlusif}+0.3*\textit{équilibrage}+0.2*\textit{découpage}$$

Les valeurs de *occlusif*, *équilibrage* et *découpage* sont comprises entre 0 et 1.

- $occlusif = A_{polygones}/A_{section}$ est appelé *degré d'opacité*, où $A_{polygones}$ est égal à la somme des aires de tous les polygones appartenant au plan de découpage et $A_{section}$ correspond à l'aire de la section du volume englobant de la sous-scène à découper (figure 2.4) ;
- $\textit{équilibrage} = N_{min}/N_{max}$ où $N_{min} = \min(N_{droite}, N_{gauche})$ et $N_{max} = \max(N_{droite}, N_{gauche})$ avec N_{droite} et N_{gauche} étant le nombre de polygones à droite et à gauche du plan de découpage ;
- $\textit{découpage} = 1 - N_d/N$ où N_d est le nombre de polygones découpés (i.e. par le plan de découpage) et N est le nombre total de polygones de l'environnement.

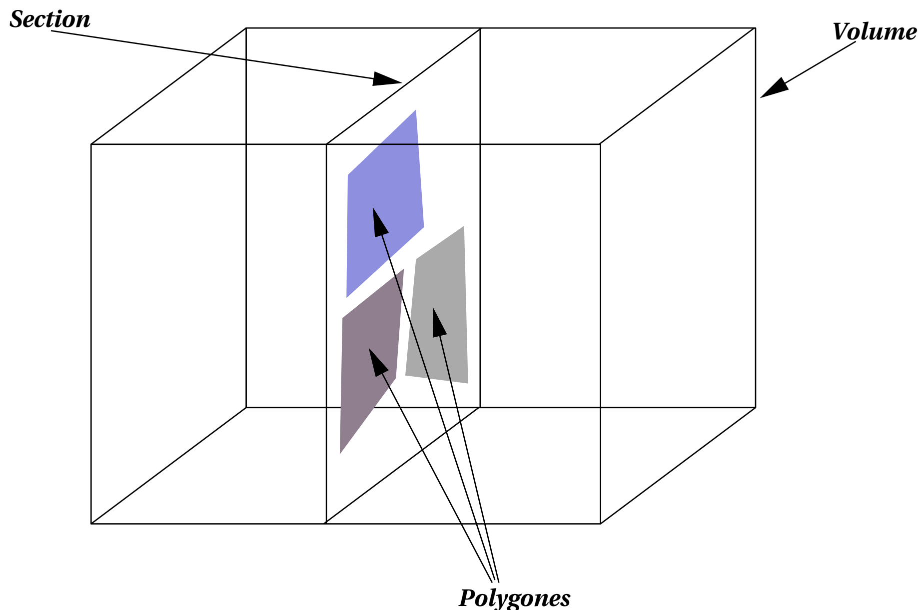


FIG. 2.4 – *Section d'un volume englobant.*

La figure 2.5 décrit l'algorithme de découpage binaire d'un environnement axial. Dans cet algorithme, la fonction *préfiltrage* est chargée de déterminer tous les plans dont le *degré d'opacité* est supérieur à un seuil donné par l'utilisateur. Suite à ce filtrage, seuls les plans les plus occlusifs sont candidats au découpage de la sous-scène courante. La fonction *Choix_Plan_Découpage* détermine finalement le plan de découpage selon la formule :

$$0.5*\textit{occlusif}+0.3*\textit{équilibrage}+0.2*\textit{découpage}$$

S. J. Teller propose d'étendre cette méthode de subdivision binaire à des environnements quelconques [12] [1] [23] (i.e. non axiaux). Le plan de découpage choisi est systématiquement le plan le plus occlusif. Et s'il ne donne pas d'algorithme détaillé pour cette technique, il admet en revanche que les calculs effectués sont complexes et difficiles à mettre en œuvre.

```
Découpage_BSP( ARBRE arbre, REEL seuil) {
        PLAN plan_découpage;
        LAXIAL ListeX, ListeY, ListeZ;

        /* Construction des 3 listes axiales en X, Y et Z */
        ListeX = Calcule_Liste_Axiale_X( arbre );
        ListeY = Calcule_Liste_Axiale_Y( arbre );
        ListeZ = Calcule_Liste_Axiale_Z( arbre );

        /* Seuls les plans dont l'aire est supérieure à */
        /* seuil sont candidats au découpage */
        Préfiltrage( ListeX, seuil );
        Préfiltrage( ListeY, seuil );
        Préfiltrage( ListeZ, seuil );

        /* Un plan est choisi en fonction du critère de découpage */
        /* parmi les plans retenus apres le préfiltrage */
        plan_découpage = Choix_Plan_Découpage( ListeX, ListeY, ListeZ );

        /* Si un plan de découpage existe, alors */
        /* les fils sont créés réecursivement */
        si ( plan_découpage ¡¿NULL ) alors {
                Créer_Fils_Droit( arbre, plan_découpage );
                Créer_Fils_Gauche( arbre, plan_découpage );

                Découpage_BSP( arbre.fils_droit, seuil );
                Découpage_BSP( arbre.fils_gauche, seuil );
        }
}
```

FIG. 2.5 – *Algorithme de subdivision binaire.*

Visibilité

Nous venons de décrire une technique de découpage d'environnements architecturaux. A partir des cellules générées par ce découpage, il est possible de précalculer un certain nombre d'informations concernant la visibilité des objets d'une cellule. En effet, une surface S peut être éclairée par :

- les surfaces de sa cellule ;

- des surfaces d'autres cellules, visibles à travers une séquence d'ouvertures.

Pour déterminer les relations de visibilité entre les cellules de l'environnement, Airey préconise d'utiliser les ouvertures [10]. Ces dernières sont situées sur les plans de découpage et obtenues par des opérations ensemblistes sur les polygones. A partir de ces ouvertures, les calculs de visibilité peuvent être effectués de plusieurs manières par des techniques basées sur la méthode du lancé de rayon. La première solution consiste à plaquer un hémicube sur l'ouverture. Cet hémicube est discrétisé et un rayon est lancé à travers chaque pixel de l'hémicube perpendiculairement à l'ouverture. Dès qu'un rayon intersecte une surface, cette dernière est visible de l'hémicube, de l'ouverture et par conséquent de la cellule correspondante. Une seconde technique consiste à plaquer non pas un hémicube, mais un hémisphère sur l'ouverture. Les rayons sont lancés à partir du centre de l'hémisphère à travers chacun de ses pixels.

Cependant, ces méthodes restent peu fiables car le lancé de rayon ne permet pas de déterminer tous les objets visibles à partir d'une ouverture. Pour remédier à ces problèmes, Airey propose une méthode analytique plus précise permettant de savoir précisément si deux polygones P_1 et P_2 sont mutuellement visibles. Il construit un volume V_e englobant les deux surfaces et effectue une série de tests visant à déterminer les surfaces contenues dans V_e provoquant une occlusion entre P_1 et P_2.

Dans [12] [1] [23] [24] S. Teller et al. utilisent également les ouvertures pour déterminer les relations de visibilité entre les cellules. Pour savoir si deux cellules $\mathcal{A}$ et $\mathcal{B}$ sont mutuellement visibles à travers une séquence d'ouvertures, Teller recherche une droite passant par toutes les ouvertures de cette séquence (figure 2.6). Si une telle droite existe alors $\mathcal{A}$ et $\mathcal{B}$ sont mutuellement visibles.

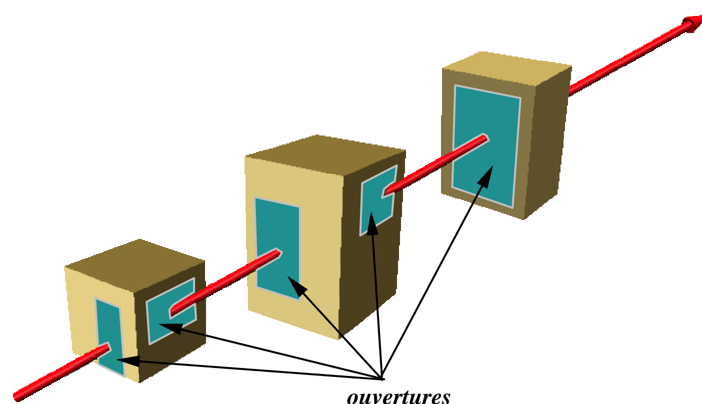


FIG. 2.6 – *Droite passant par les ouvertures de plusieurs cellules.*

2.2.2 Discussion

La seule technique permettant de structurer automatiquement un environnement architectural complexe est basée sur un découpage binaire de l'espace (BSP). Cette méthode développée en premier lieu par J. M. Airey et étendue par S. Teller constitue une étape de précalcul donnant lieu à plusieurs structures de données utilisées pour accélerer les calculs de radiosité [1] ou fluidifier la visualisation interactive (*walkthrough*) d'environnements complexes [25], [26].

Cependant, cette technique de découpage est assez mal adaptée aux environnements architecturaux. En effet, comme la topologie de l'environnement n'est pas prise en compte par les algorithmes, les cellules résultant du découpage sont trop nombreuses et ne correspondent pas aux pièces ou aux couloirs. Par conséquent les ouvertures sont également trop nombreuses et ceci entraîne des surcoûts de calcul.

D'autre part, si la méthode BSP est relativement facile à mettre en œuvre dans le cas d'environnements axiaux, en revanche, le traitement de bâtiments quelconques est beaucoup plus coûteux et plus complexe à mettre en œuvre. Par ailleurs tous les critères de découpage utilisés pour des environnements axiaux ne sont plus valides pour des environnements quelconques. Par exemple, le critère d'occlusion ne peut plus être appliqué car les polygones occlusifs peuvent avoir des orientations quelconques et l'utilisation d'une liste triée selon les trois axes du repère de la scène n'a plus aucun sens.

Dans les sections suivantes, nous décrivons une nouvelle méthode de découpage d'environnements architecturaux. Grâce à cette méthode, les cellules générées correspondent à des éléments topologiques de l'environnement. Comme Airey et Teller, nous déterminons les relations de visibilité entre les cellules à l'aide de leurs ouvertures. Le résultat de ces calculs consiste en deux types fichiers :

- Le premier contient la description de chaque cellule.
- Le second comporte la description d'un graphe exprimant les relations de visibilité entre les cellules.

2.3 Principes de notre méthode

2.3.1 Hypothèses et objectifs

Les scènes que nous traitons sont des bâtiments de plusieurs étages, non nécessairement axiaux et modélisés à l'aide de polygones plans convexes. Par convention, le plan horizontal contient les axes O_x et O_y du repère de la scène et l'axe O_z représente la hauteur.

Notre objectif est d'extraire toutes les pièces d'un bâtiment de manière automatique ou semi-automatique, même si aucune information ne permet de distinguer les murs des autres objets. Pour cela, la scène est découpée en plusieurs sous-scènes appelées *cellules* constituant une mosaïque 3D de l'espace de la scène. Notre cahier des charges est le suivant :

1. *Les cellules correspondent aux pièces ou aux couloirs de la scène.* Une pièce est une cellule idéale puisqu'elle contient un ensemble de polygones visibles les uns des autres. D'autre part, des objets situés dans deux pièces différentes sont séparés par des éléments occlusifs tels que des cloisons ou des murs.

2. *Le nombre total de polygones ne doit pas augmenter.* En effet, certains polygones sont découpés (par les plans de découpage). Et ce découpage entraîne un accroissement indésirable de la base de données. C'est pourquoi les plans de découpage doivent être déterminés de manière judicieuse.

3. *Les relations de visibilité entre les cellules sont déterminées automatiquement* dans le but de faciliter les calculs de visualisation ou de simulation d'éclairage.

2.3.2 Principes

Notre méthode de structuration est constituée de 4 étapes :

1. *Extraction des étages du bâtiment*
   Dans toute la suite de ce document, nous supposons que les étages de la scène sont connus. Dans le cas contraire, cette étape peut être réalisée à l'aide d'un découpage binaire de type BSP.

2. *Création de plans de découpage*
   Pour chaque étage du bâtiment, deux listes L_v et L_a sont construites : L_v contient les polygones verticaux (comme les murs par exemple) et L_a contient les autres polygones. Les polygones de L_v sont représentés par des points (θ, ρ) dans un espace dual où θ représente l'angle entre la normale du polygone et l'axe O_x et ρ correspond à la distance orthogonale du polygone à l'origine du repère. En regroupant les points (θ, ρ) voisins, nous déduisons les plans les plus occlusifs de la scène. Ces plans sont appelés *Plans de découpage.*

3. *Extraction des cellules*
   Une fois les plans de découpage déterminés, des cellules sont extraites grâce à un ensemble de règles de construction prenant en compte certaines connaissances générales sur l'environnement. Ces règles sont basées sur certains modèles de pièces. Par exemple :

   La plupart des pièces sont rectangulaires.

   La largeur d'une pièce n'est jamais inférieure à 50 cm.

   etc.

4. *Etablir des relations de visibilité entre les cellules*
   Ce précalcul donne lieu à un graphe indiquant précisément les relations de visibilité entre les cellules. Ce graphe appelé *graphe de visibilité* est construit en plusieurs étapes :

   (a) *Recherche des ouvertures* sur les plans de découpage de chaque cellule.

   (b) *Construction du graphe d'adjacence.* Ce graphe exprime les relations de voisinage entre les cellules. Un nœud du graphe représente une cellule et un arc relie deux cellules partageant une ouverture.

   (c) *Construction du graphe de visibilité.* Ce graphe représente les relations de visibilité entre les cellules de la scène. Ce graphe est construit en ajoutant des arcs au graphe d'adjacence. Un arc relie deux nœuds lorsque les cellules correspondantes sont visibles à travers une séquence d'ouvertures.

2.3.3 Méthode semi-automatique

Afin de donner à l'utilisateur la possibilité d'intervenir dans le processus de découpage, notre programme dispose d'une interface graphique utilisant les bibliothèques Motif et OpenGl. Cette interface comporte deux fenêtres. La première affiche une image en fil de fer de la scène vue de dessus. Dans cette fenêtre, il est possible de sélectionner une partie de la scène afin de la soumettre au découpage. D'autre part, l'utilisateur peut visualiser de manière interactive les cellules résultant de ce découpage dans la seconde fenêtre à l'aide d'OpenGl. Le mode de fonctionnement de ce programme facilite l'extraction des cellules et permet à l'utilisateur d'intervenir dans le processus de structuration. Ainsi, l'ensemble des calculs (découpage et visibilité) sont automatiques mais peuvent être guidés par l'utilisateur.

2.4 Notre méthode de découpage

Pour chaque étage du bâtiment, nous extrayons un certain nombre de cellules convexes, déterminées par l'intersection de plusieurs demi-espaces. Ces demi-espaces sont définis par des *plans de découpage* correspondant à plusieurs polygones quasi-alignés. Ensuite, l'extraction des cellules est guidée par des modèles de pièces auxquels nous faisons correspondre des *règles d'extraction.*

2.4.1 Définition de l'espace dual

A chaque mur d'un bâtiment, nous souhaitons faire correspondre un plan de découpage. Or les murs peuvent être composés de plusieurs polygones qui ne sont pas nécessairement situés sur le même plan. Dans cette section nous proposons une technique permettant de regrouper ces polygones grâce à un espace dual (θ, ρ). Un point de (θ, ρ) est associé à un plan (dans le repère du bâtiment) et plusieurs points de proches les uns des autres sont associés à plusieurs plans quasi-alignés.

Afin d'expliquer plus clairement notre méthode, nous la décrivons tout d'abord en dimension 2. Esuite, nous définissons formellement la notion de *plans quasi-alignés* avant de décrire notre algorithme en dimension 3.

Espace dual pour des droites 2D [14]

En dimension 2, une droite $\mathcal{D}$ est usuellement repérée par deux points ou une équation

$$N \bullet P + \rho = 0$$

Dans cette équation, N représente la normale de la droite et ρ est la distance orthogonale de l'origine du repère à $\mathcal{D}$.

Soit θ l'angle entre la droite et l'axe Ox. θ est aussi appelé *orientation* de la droite. Nous appelons ρ le *décalage* de la droite. Dans le repère (θ, ρ), une droite correspond donc à un point (figure 2.7).

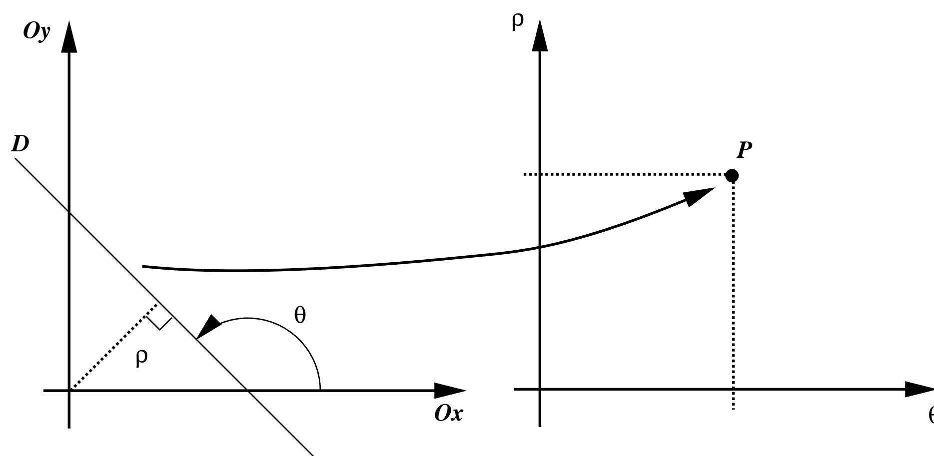


FIG. 2.7 – *Représentation d'une droite dans un espace dual.*

Pour un ensemble de segments de droite ayant une *orientation* et un *décalage* quasi-identiques, les points de l'espace dual correspondants sont proches les uns des autres (figure 2.8).

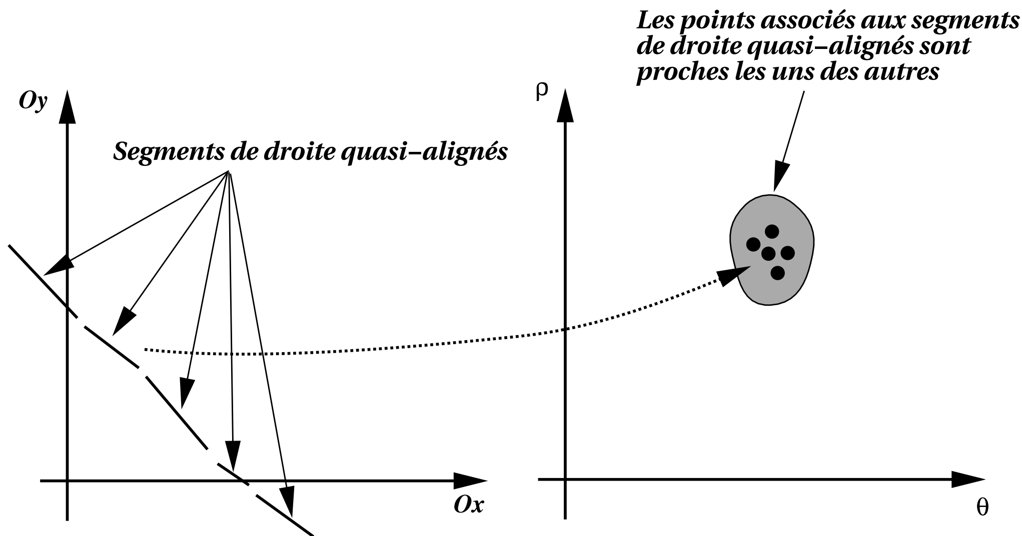


FIG. 2.8 – *Nuage de points.*

Espace dual en 3D

En dimension 3, le principe est équivalent : un polygone vertical P_v de normale $\vec{N}$ est représenté par un point P_o $(\theta, \rho, Aire)$ dans un espace dual (figure 2.9) où :

- θ est l'angle entre l'axe Ox et $\vec{N}$.
- ρ est la distance orthogonale de l'origine au plan du polygone.
- $Aire$ est l'aire du polygone.

A partir de ces notions, nous pouvons définir formellement le terme *quasi-alignés*. Deux polygones P_1 et P_2 représentés par les points $(\theta_1, \rho_1, Aire_1)$ et $(\theta_2, \rho_2, Aire_2)$ dans l'espace dual sont *quasi-alignés* si :

- $||\theta_1 - \theta_2|| < \epsilon_\theta$ et
- $||\rho_1 - \rho_2|| < \epsilon_\rho$

avec ϵ_θ étant une petite valeur d'angle et ϵ_ρ une petite distance.

L'algorithme de la figure 2.10 décrit la structure de données associée à un polygone dans notre programme.

La scène est décrite dans un fichier (au format NFF [27]) contenant toutes les informations concernant la géométrie de la scène et les matériaux des polygones. Lors de la lecture de ce fichier, les polygones de la scène sont insérés dans deux listes :

- la première liste ne contient que les polygones verticaux. C'est à dire ceux dont la normale a une composante nulle en z.
- la seconde liste contient tous les autres polygones.

Les structures de données correspondant au stockage de la scène en mémoire sont décrites par la figure 2.11.

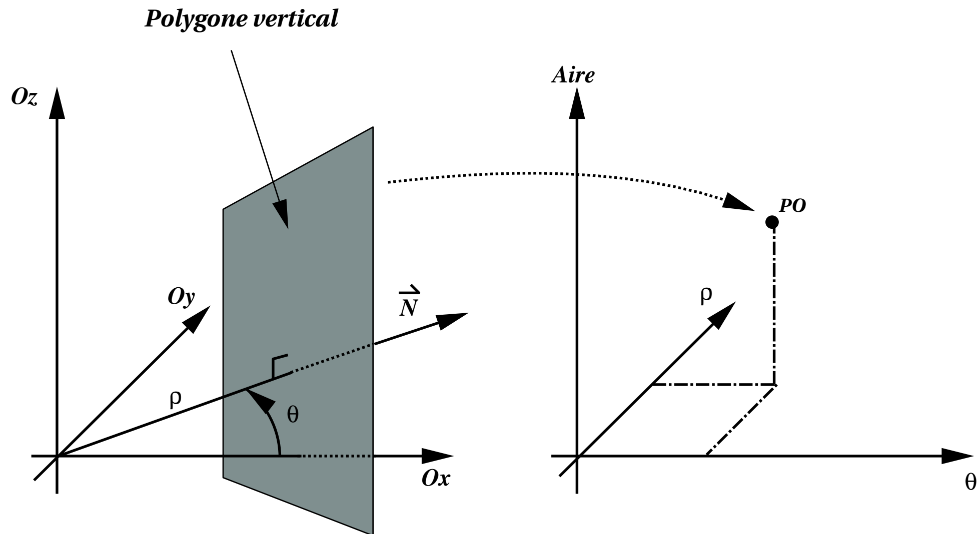


FIG. 2.9 – *Transformée d'un polygone.*

```
/* Definition de la structure d'un polygone */
struct POLYGONE {

        char couleur[50];       /* couleur du polygone */
        double prop[3];         /* propriétés de la couleur */
        int nb_sommets;         /* nombre de sommets */
        Point points[MAXPOINTS];        /* Liste des sommets */
        Vecteur normal;         /* vecteur normal */
        double aire;            /* aire du polygone */
        double theta;           /* valeur de l'angle de la normale */
        double rho;             /* distance a l'origine */

} *Polygone;
```

FIG. 2.10 – *Structure de données correspondant à un polygone.*

```
/* Definition de la structure "scène" */
typedef struct SCENE {

        Liste_Polygones murs;           /* Polygones verticaux */
        Liste_Polygones autres;         /* Autres surfaces */

} *Scene;
```

FIG. 2.11 – *Structure données correspondant à la scène.*

Les valeurs θ et ρ sont calculées à partir de la normale du polygone $\vec{N} = (N_x, N_y, 0)$. θ est l'angle entre l'axe Ox et $\vec{N}$ et ρ représente la distance entre l'origine et sa projection orthogonale sur le plan du polygone.

- $\theta = arcos(Nx) * signe(Ny)$.
- $\rho = -Nx * Px - Ny * Py$, où $P = (P_x, P_y, P_z)$ est un point du polygone.

A la lecture de la scène, les polygones verticaux sont insérés dans une liste triée. Dans cette liste les polygones sont dans un ordre croissant de θ. Puis plusieurs polygones ayant une valeur θ égale sont triés selon leur ρ.

Notre objectif est maintenant de regrouper les points (correspondant aux polygones verticaux) proches les uns des autres afin de leur associer un seul plan de découpage. Ce calcul consiste en une *classification* des points dans l'espace dual [28].

2.4.2 Classification

Pour effectuer le regroupement des points dans l'espace dual, nous avons opté pour une technique de subdivision binaire donnant lieu à un arbre dont les feuilles contiennent les groupes de points. La subdivision binaire que nous proposons est constituée de deux étapes :

- Tout d'abord, l'espace dual est subdivisé de manière récursive binaire selon l'axe θ (figure 2.12).
- Puis pour chaque feuille (pour chaque bande), de ce premier arbre, une nouvelle partition est déterminée (également de manière binaire) selon l'axe ρ (figure 2.13).

Rappelons que les polygones verticaux sont triés selon leur valeur θ dans l'ordre croissant et rangés dans une liste L_θ. Nous cherchons dans cette liste triée la plus grande distance entre deux valeurs de θ consécutives :

$$\max_{i \in [1, N-1]} (\theta_{i+1} - \theta_i),$$

N étant le nombre d'élements de L_θ. Soit I, l'indice tel que $(\theta_{I+1} - \theta_I) = \max_{i \in [1,N-1]}(\theta_{i+1} - \theta_i)$ (figure 2.12). Nous choisissons une valeur θ_p au milieu de l'intervalle $[\theta_I, \theta_{I+1}]$, c'est à dire : $\theta_p = \theta_I + 0.5 * (\theta_{I+1} - \theta_I)$, pour découper la liste L_θ en deux listes L_θ^1 et L_θ^2. Les polygones de L_θ^1 ont une valeur de θ inférieure à θ_p alors que ceux de L_θ^2 ont une valeur de θ supérieure à θ_p. Ce processus de découpage est répété de manière récursive pour L_θ^1 et L_θ^2 et il s'achève lorsque la distance maximale $\max(\theta_{i+1} - \theta_i)$ est en-dessous d'un certain seuil. Cette classification donne lieu à un arbre binaire B_θ pour lequel chaque feuille est une liste de polygones verticaux dont les valeurs de θ sont proches les unes des autres. Chaque feuille de B_θ est ensuite triée selon les valeurs de ρ et découpée de manière récursive en deux parties selon la même méthode que pour L_θ (figure 2.13). A chaque étape du découpage, la valeur de découpage ρ_p est égale à $\rho_I + 0.5 * (\rho_{I+1} - \rho_I)$, I étant l'indice tel que $\rho_{I+1} - \rho_I = \max_i(\rho_{i+1} - \rho_i)$. Le processus se termine lorsque $\max_i(\rho_{i+1} - \rho_i)$ est en dessous d'un certain seuil. A ce stade, chaque feuille i de B_θ est la racine d'un arbre binaire B_ρ^i résultant du découpage selon les valeurs de ρ. Finalement, nous obtenons un arbre binaire étendu B dont les feuilles sont des groupes de points proches les uns des autres dans l'espace dual.

La structure employée pour effectuer cette subdivision est décrite par l'algorithme de la figure 2.14.

dt et dr représentent la taille minimale d'une feuille de l'arbre, suivant θ d'une part et ρ d'autre part. *btr* est un pointeur sur l'arbre binaire BIN_R_TREE. btr est nul dans tous les nœuds non

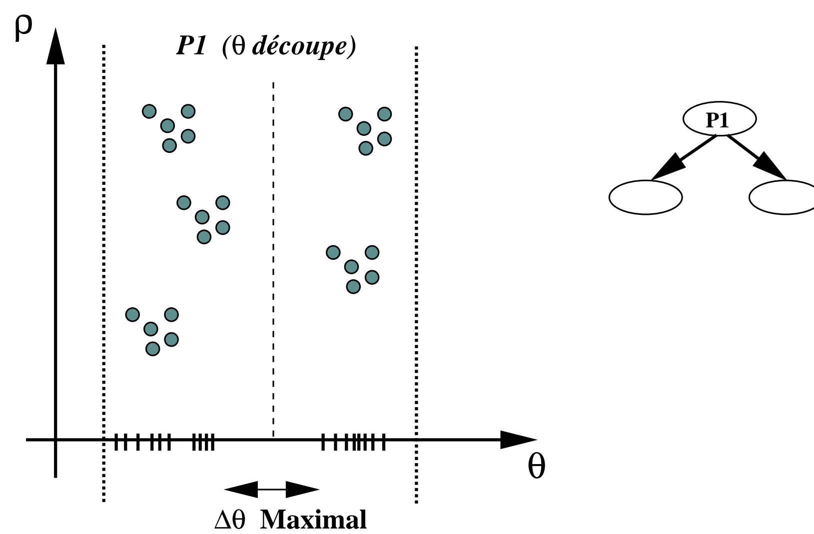


FIG. 2.12 – *Création de l'arbre binaire selon θ.*

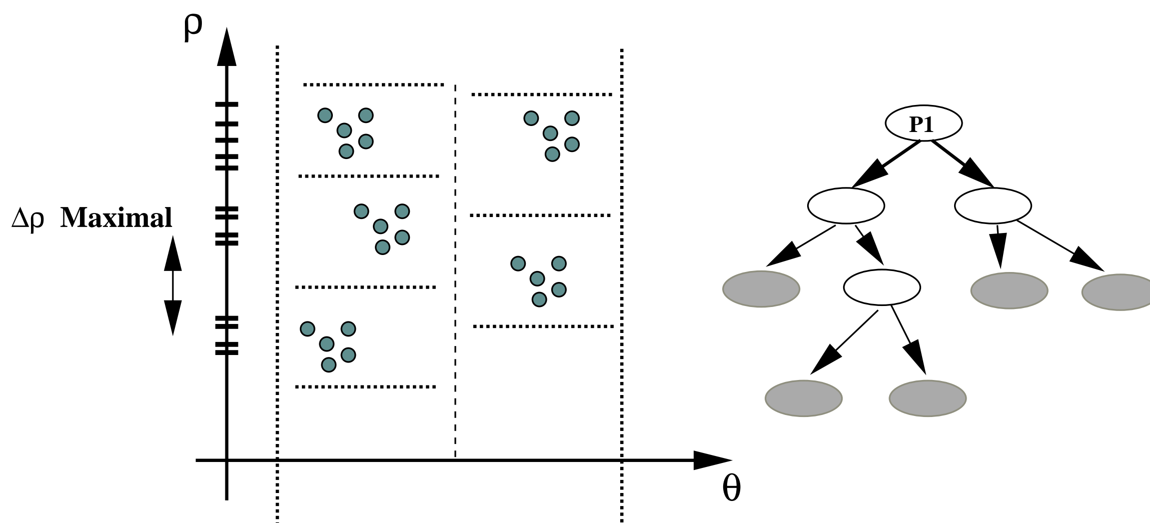


FIG. 2.13 – *Classification des points dans le repère dual selon ρ.*

```
typedef struct BIN_T_TREE {

        struct BIN_T_TREE *fils1, *fils2;     /* Fils 1 et 2 */
        double Tmin, Tmax, Rmin, Rmax;     /* θ et ρ min et max du noeud */
        double dt, dr;     /* Dθ et Dρ */
        Liste_Polygones p;     /* Liste des polygones associés */
        Bin_R_Tree btr;     /* Arbre binaire ρ */

} *Bin_T_Tree;


typedef struct BIN_R_TREE {

        struct BIN_R_TREE *fils1, *fils2;     /* Fils 1 et 2 */
        double Rmin, Rmax, Tmin, Tmax;     /* θ et ρ min et max du noeud */
        double dr;     /* Dρ */
        Liste_Polygones p;     /* Liste des polygones associés */
        Demi-Espace cut;     /* plan de découpage associé aux feuilles */

} *Bin_R_Tree;
```

FIG. 2.14 – *Structure de données permettant de représenter l'arbre binaire.*

terminaux de l'arbre. Il est alloué seulement aux feuilles. L'algorithme effectuant la création de cette première partie de l'arbre binaire est donné par la figure 2.15.

Dans cet algorithme, il s'agit d'allouer, d'initialiser la racine de l'arbre et de propager les informations de manière récursive. La subdivision s'arrête lorsque les feuilles ont atteint leur taille minimale. La propagation est effectuée grâce à la fonction *propage* (figure 2.16).

La fonction *cree_feuille* initialise le processus de propagation de l'arbre binaire de subdivision selon ρ. La fonction *trouve_position_theta* détermine selon quelle valeur de θ la partition binaire a lieu.

Chaque feuille de l'arbre binaire construit correspond donc à un groupe de points proches les uns des autres dans l'espace dual. Nous y associons un pointeur vers une structure de données représentant les paramètres d'un plan de découpage. Initialisée à NULL dans chaque nœud non terminal de l'arbre binaire, ce pointeur est instancié dans la phase de création de plans de découpage.

2.4.3 Création des plans de découpage

Chaque feuille de l'arbre binaire correspond à un groupe de points dans l'espace dual. Nous souhaitons déterminer un plan de découpage $N_d \bullet P + \rho_d = 0$ (correspondant à un couple (θ_d, ρ_d) dans l'espace dual) pour chacun de ces groupes de points. La valeur θ_d associée à N_d est choisie aussi proche que possible des cellules des polygones dans le groupe de points tout en favorisant les polygones les plus occlusifs. Plus précisément, θ_d est égal à la moyenne de tous les θ_i du

```
Bin_T_Tree créer_bin_T_tree(Polygon_List PL)
   /* PL est la liste des polygones verticaux */
{

      Bin_T_Tree ARBRE;

      /* La racine est initialisée : elle contient la liste de tous les polygones. */
      ARBRE = initialise(PL);

      /* Création de l'arbre binaire */
      ARBRE = propage(ARBRE);

}
```

FIG. 2.15 – *Classification des points dans le repère dual selon θ.*

groupe pondérée par l'aire $area_i$ des polygones de chaque groupe :

$$\theta_s = \frac{\sum area_i * \theta_i}{\sum area_i}$$

Voyons maintenant comment ρ_s est déterminé. Le plan de découpage est choisi de manière à ce que le demi-espace défini par $N_s \bullet \rho_s + d > 0$ contienne tous les polygones du groupe (figure 2.17). Cette condition peut être exprimée par $\rho_s > \max_i(-N_s \bullet P_i) + \epsilon$, ϵ étant une petite valeur choisie par l'utilisateur de manière à ce qu'aucun des polygones verticaux ne soit sur le plan de découpage (5cm par défaut dans notre implémentation).

La structure de données associée aux plans de découpage est décrite par la figure 2.18.

Chaque plan de découpage est projeté sur le plan O_{xy}. *origin[2]* est le point utilisé comme origine du repère local, propre au plan de découpage (figure 2.19). Cette origine correspond à la projection orthogonale de l'origine du repère de la scène sur le plan de découpage. Le vecteur unitaire du repère local est un vecteur horizontal avec : $\theta = \theta_N - 90$ où θ_N est l'angle entre la normale du polygone et l'axe Ox du repère de la scène (figure 2.20). *abscissa[2]* représente un segment dans lequel sont contenus les projections orthogonales de tous les polygones associés au plan de découpage. Nous appellerons ces repères locaux des *droites de découpage*.

Tous les polygones associés à un plan de découpage sont projetés sur la droite de découpage correspondante. Cette projection résulte en un ensemble de segments de droite contenus dans un segment (Min, Max) (figure 2.21). L'aire d'un plan de découpage correspond à la somme des aires de tous les polygones qui lui sont associés.

Nous disposons maintenant de plans de découpage correspondant aux éléments les plus occlusifs de l'environnement.

2.4.4 Règles de construction des cellules.

A partir des plans de découpage déterminés, des cellules sont extraites de la scène grâce à un ensemble de règles génériques, correspondant à certain modèle de pièces. A chaque modèle de

```
Bin_T_Tree propage(Bin_T_Tree BTT) {

        double position;     /* position de partitionnement de la liste de polygones */

        /* Si le noeud est de taille minimale, on en fait une feuille */
        if( (BTT->Tmax - BTT->Tmin) < BTT->dt) {
                BTT = crée_feuille(BTT);
                retourne BTT;
        }

        /* Recherche de la valeur de decoupe */
        position = trouve_position_theta(BTT->p, BTT->Tmax);

        /* Creation des deux fils */
        BTT->fils1 = initialise(BTT->fils1);
        BTT->fils2 = initialise(BTT->fils2);

        /* initialisation du premier fils */
        BTT->fils1->Tmin = BTT->Tmin;
        BTT->fils1->Tmax = position;
        BTT->fils1->Rmin = BTT->Rmin;
        BTT->fils1->Rmax = BTT->Rmax;

        /* initialisation du second fils */
        BTT->fils2->Tmin = position;
        BTT->fils2->Tmax = BTT->Tmax;
        BTT->fils2->Rmin = BTT->Rmin;
        BTT->fils2->Rmax = BTT->Rmax;

        /* BTT devient un noeud, sa liste de polygones est mise a nul */
        BTT->p = NULL;

        /* Propagation récursive du processus dans les deux fils */
        BTT->fils1 = propage(BTT->fils1);
        BTT->fils2 = propage(BTT->fils2);

        /* On retourne l'arbre resultant */
        retourne BTT;

}
```

FIG. 2.16 – *Propagation de la création de l'arbre binaire.*

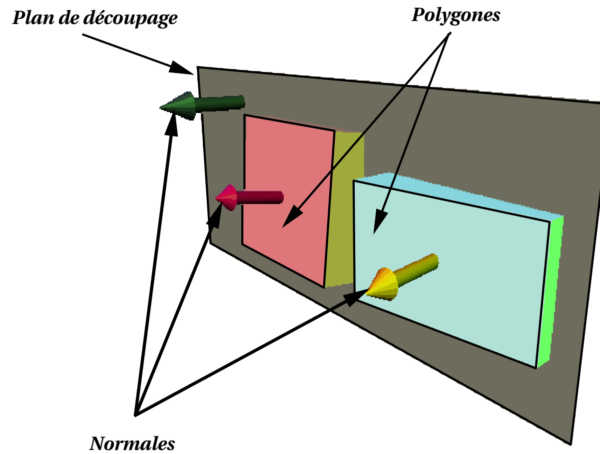


FIG. 2.17 – *Plan de découpage.*

```
struct DEMI-ESPACE {

    double theta;      /* Angle de la normale */
    double rho;        /* Distance a l'origine */
    double aire;       /* Somme des aires de la liste des poly associes */
    double A, B;       /* Ax + By + rho >= 0 ou equation du demi-espace */
    double origine[2]; /* Point Origine du repere propre */
    double abscisse[2]; /* dans le repere de la droite projetee abscisse du min et du max */

} *Demi-Espace;
```

FIG. 2.18 – *Structure de données correspondant à un nœud de l'arbre binaire.*

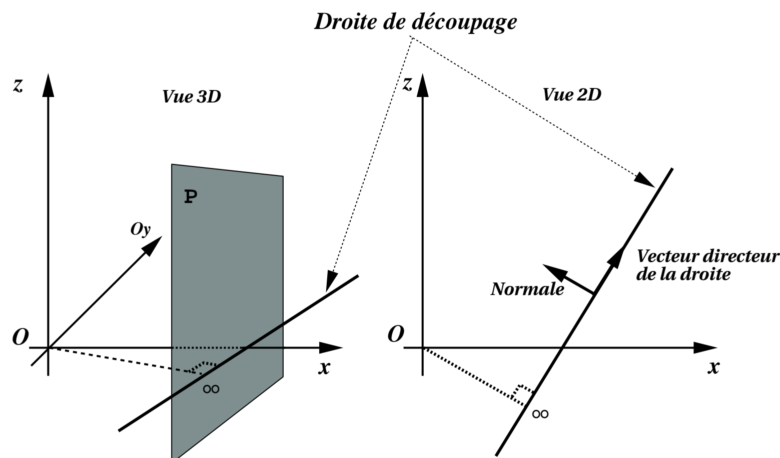


FIG. 2.19 – *Repère local à un plan de découpage.*

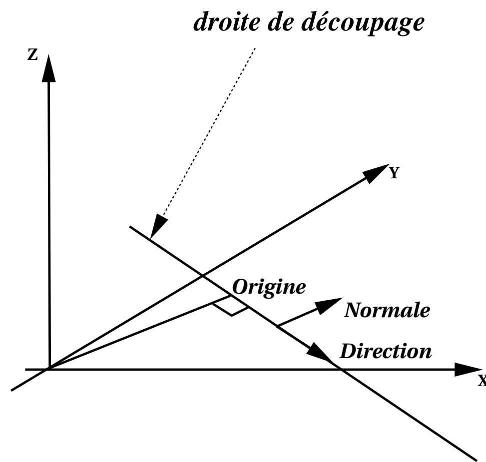


FIG. 2.20 – *Repère local à un plan de découpage (ou à la droite de découpage associée).*

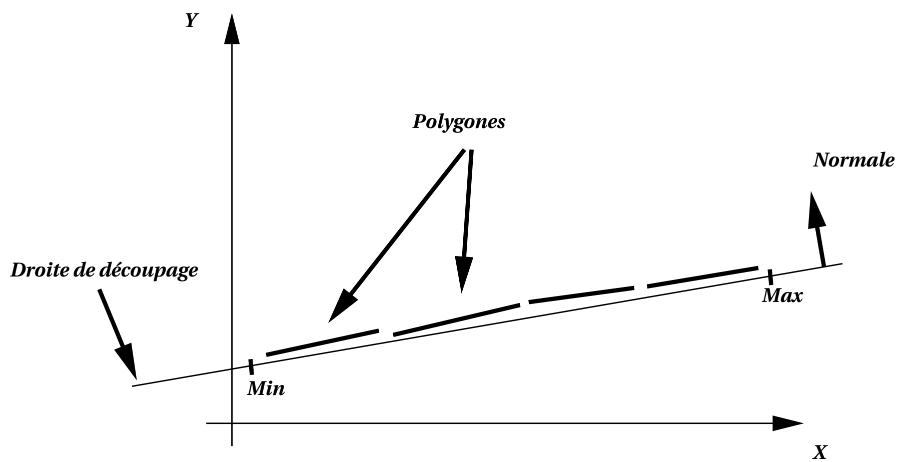


FIG. 2.21 – *Segment* (Min, Max).

pièce, nous avons associé une règle particulière. Par exemple, une règle correspond à un modèle de pièce rectangulaires, une autre aux couloirs, etc.

Définition d'une cellule

Dans la section précédente, nous avons vu comment déterminer les plans de découpage d'équation $N{\bullet}P{+}\rho = 0$. Chaque plan de découpage définit en fait un demi-espace d'inéquation : $N{\bullet}P{+}\rho \geq 0$ et l'intersection de plusieurs demi-espaces déterminent une région convexe que nous appelons *cellule*. Les règles de construction ont pour objectif de générer des cellules correspondant aux pièces ou aux couloirs d'un bâtiment. En choisissant judicieusement les plans de découpage, les pièces de la scène peuvent être déduites en totalité. Pour chaque cellule, le processus d'extraction est le suivant :

- Déterminer des plans de découpage d'une cellule.
- Déterminer la liste de tous les polygones se trouvant dans la cellule. Certains d'entre eux peuvent être découpés (par les plans de découpage).
- Rechercher les ouvertures de la cellule.

A chaque modèle de pièce, nous avons associé une règle particulière. Dans la pratique, les règles sont appliquées successivement, de la plus rigide à la plus souple. Nous avons mis en œuvre 4 règles permettant d'extraire :

- des cellules rectangulaires contraintes;
- des cellules rectangulaires quelconques;
- des cellules à deux murs parallèles;
- des cellules convexes quelconques;

Ces règles sont appliquées dans l'ordre où nous les avons données et lorsqu'une règle est choisie, toutes les cellules vérifiant la règle sont extraites. Voyons maintenant comment sont définies ces règles.

Pièces rectangulaires

Quel que soit le type d'architecture, les pièces rectangulaires sont les plus courantes. Elles correspondent par exemple à des bureaux ou des chambres à coucher et sont composées de 4 murs perpendiculaires. La règle correspondant à l'extraction de ce modèle de cellule est décrite par l'algorithme de la figure 2.22. Les cellules résultant de cet algorithme sont représentées par l'intersection de 4 demi-espaces.

Lorsque deux plans $P1$ et $P2$ sont face à face, il en est de même pour les origines de leurs repères (figure 2.23). Il est donc facile de déterminer deux plans de découpage dont les polygones associés sont face à face. Il suffit de vérifier que les segments (Min, Max) se chevauchent. La projection orthogonale du segment (Min, Max) de $P1$ sur le plan $P2$ a pour coodonnées $(-Min, -Max)$ dans le repère local à $P2$ (figure 2.23).

A partir de cet algorithme, nous avons introduit deux types de règles différentes.

1. *Pour extraire des cellules rectangulaires contraintes*
   Ce premier type de règle a pour objectif d'extraire uniquement les pièces parfaitement rectangulaires correspondant par exemple à des bureaux. Pour éviter qu'un couloir en L (étant aussi constitué de deux cellules rectangulaires) soit extrait grâce à cette règle il suffit de vérifier qu'aucun polygone occlusif (appartenant à l'un des plans de découpage) ne coupe l'un des 4 plans de découpage.

1 - Choisir le plan de découpage le plus occlusif P_0 dont le champ area défini dans la structure de données est maximum.

2 - Parcourir l'arbre binaire B_θ pour chercher l'ensemble E des plans de découpage en face de P_0. E est la ième feuille de B_θ et correspond à la racine d'un arbre binaire B_ρ^i.

3 - Soit P_1, l'élément de E le plus proche de P_0 tel que: $Sizemin \leq \rho_0 + \rho_1 \leq Sizemax$, et les segments (Min, Max) de P_0 et P_1 se recouvrent.

4 - Choisir un plan de decoupage P_2 perpendiculaire à P_0 tel que: $\theta_2 = \theta_0 + \pi/2$ en le recherchant dans B_θ.

5 - Rechercher dans l'arbre binaire B_θ (la partie supérieure de B_θ) pour trouver l'ensemble E' des plans de découpage faces à P_2. E' est une feuille j de l'arbre B_θ et correspond à la racine d'un arbre binaire B_ρ^j.

6 - Soit P_3 l'élément de E' le plus proche de $P2$ tel que: $Sizemin \leq ro + r1 \leq Sizemax$, et les segments (Min, Max) de P_2 et P_3 se recouvrent. P_3 est déterminé en parcourant dans B_r^j.

FIG. 2.22 – *Algorithme de création d'une pièce rectangulaire.*

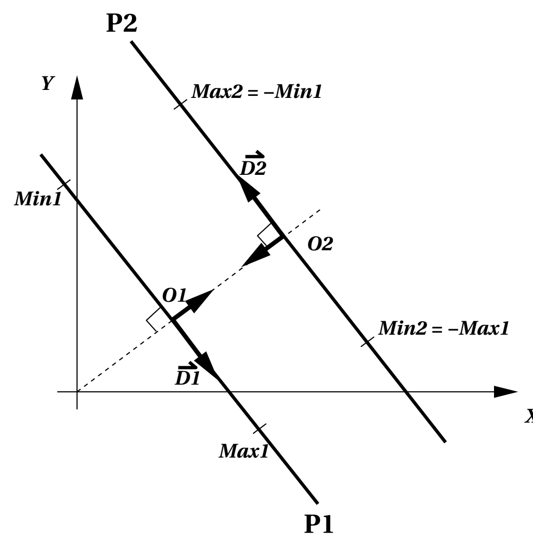


FIG. 2.23 – *Origines face-à-face.*

2. *Pour extraire des cellules rectangulaires quelconques*
   La règle de construction précédente est extrêmement contrainte. Notre objectif est maintenant d'extraire des pièces correspondant à des couloirs en L par exemple (figure 2.24). Cette fois, le même type de règle est appliqué mais de manière plus souple (les polygones très occlusifs peuvent être découpés). Dans cet exemple, le mur A doit être découpé au moment de la création de la cellule. Le choix des plans de découpage s'opère comme pour les pièces précédentes jusqu'à la fin de l'algorithme 2.22 et sans ajouter de contrainte.

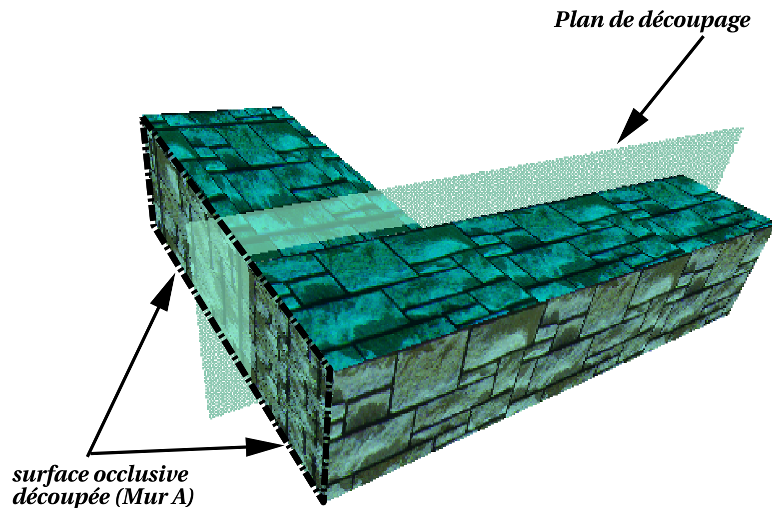


FIG. 2.24 – *Pièces rectangulaires quelconques.*

Pièces déterminées par deux murs parallèles

Dans cette section, nous décrivons un algorithme d'extraction de cellules correspondant à des couloirs (délimités par deux murs parallèles). Cet algorithme est décrit par les figure 2.25 et 2.26. Notons que l'intersection de deux demi-espaces parallèles correspond à une zone de l'espace infinie. Nous devons donc rechercher deux plans de découpage pour définir les extrémités des couloirs. Si aucun plan de découpage n'est trouvé parmi les plans de découpage de l'environnement, alors de nouveaux plans virtuels sont créés avec les points correspondants aux extrémités des segments (Min, Max), de la manière décrite ci-après.

1- Choisir 2 plans de découpage P_0 et P_1 face à face et tels que leurs segments (Min, Max) M_0 et M_1 se recouvrent.

2- Chercher 2 autres plans de découpage P_2 et P_3 dont les plans de découpage intersectent M_0 et M_1.

3- Si ces plans de decoupage existent, alors construire la cellule (figure 2.26, gauche).

4- Sinon, les créer à l'aide des points Min et Max de M_0 et de M_1 (figure 2.26, droite).

FIG. 2.25 – *Algorithme d'extraction de pièces ayant 2 murs parallèles.*

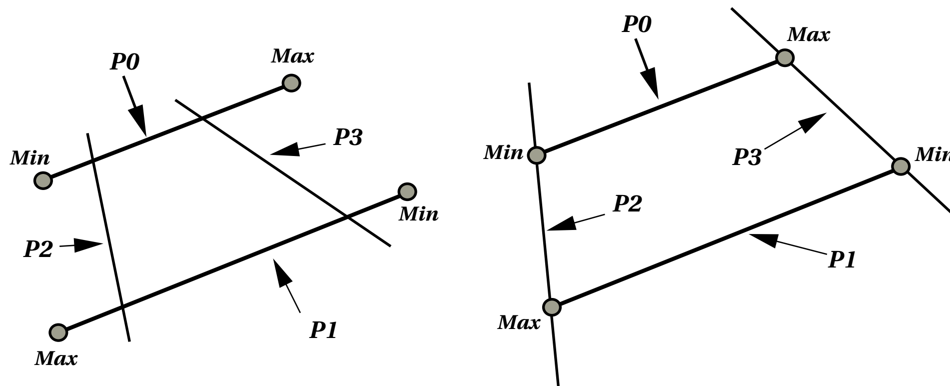


FIG. 2.26 – *Pièces à deux murs parallèles.*

Pièces convexes quelconques

Les plans de découpage restants sont utilisés pour former des cellules non conventionnelles correspondant soit à des pièces non rectangulaires soit à des zones de l'espace sans signification topologique (figure 2.27).

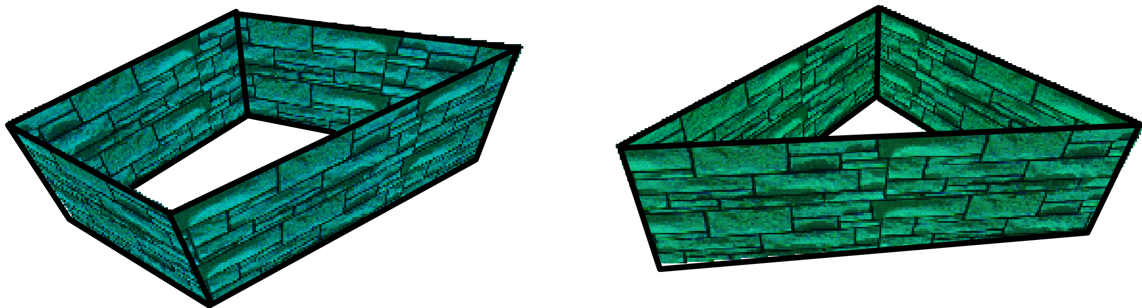


FIG. 2.27 – *Pièces quelconques.*

Ces cellules sont extraites selon une règle de suivi de contours dont le principe est de rechercher des plans contigus de manière à créer des cellules convexes (figure 2.29).

Tout d'abord, nous choisissons un plan de départ P_0 et suivons le contour de la cellule a partir de ce plan $P0$. Par convention, P_0 est le plan le plus occlusif. Le plan P_1 est celui dont l'intersection avec P_0 est la plus proche de l'extrémité Min du segment (Min, Max) de P_0. Puis le plan P_2 est choisi de la même manière : l'intersection de P_2 et P_1 est la plus proche possible de l'extrémité Min du segment (Min, Max) de P_1, etc. Nous effectuerons ainsi le suivi de contour de manière à ce que le point Min de $P0$ soit toujours contenu dans la cellule (figure 2.28).

Pour un plan de découpage P_i, le plan suivant P_{i+1} est choisi de la manière suivante :

- P_{i+1} contient le point Min de $P0$.
- le point $P_{i+1} \cap P_i$ est le plus proche du point $P_{i-1} \cap P_i$.

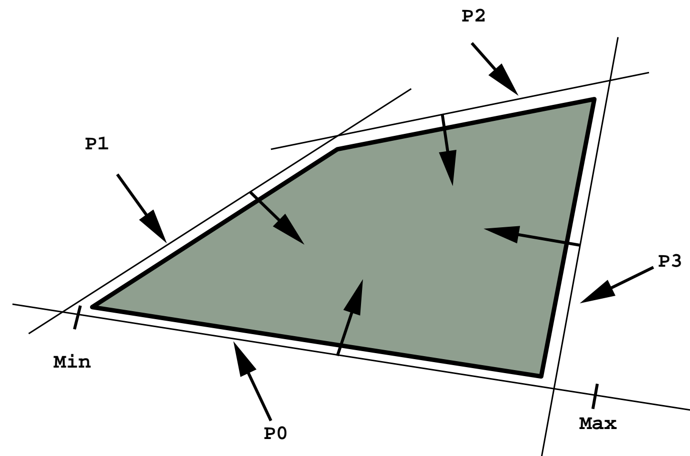


FIG. 2.28 – *Suivi de contours.*

Propagation

Pour accélerer le processus d'extraction de cellules, nous proposons de guider la recherche de celles-ci. En effet, dans un bâtiment, les pièces sont souvent adjacentes et séparées par un mur (figure 2.30). Dans notre algorithme, ceci est effectué en propageant la recherche à partir d'une première cellule extraite.

Soit une cellule rectangulaire C_1, définie par 4 plans de découpage P_1, P_2, P_3 et P_4. Soient P_3 et P_4 les deux plans associés aux plus longs murs de la cellule. Pour extraire une nouvelle cellule C_2, nous conservons P_1 et P_2 et considérons P_4 comme le plan P_3' de la cellule adjacente C_2. Il ne reste maintenant plus qu'à déterminer un nouveau plan P_4' faisant face à P_3' pour la cellule C_2. Puis ce processus est répété autant de fois que possible (figures 2.31 et 2.32) et pour tous les plans de découpage associés à C_1.

Note : pour déterminer un plan P opposé à un plan P_0, nous utilisons les nuages de points. Soit (θ_0, ρ_0) le point représentant le plan P_0, il suffit de déterminer le point (θ, ρ) représentant P dans l'espace dual tel que $\theta = \theta_0 + \Pi$ et $\rho = -\rho$.

Fusion de plans de découpage

Nous avons vu comment déterminer des cellules à partir de plans de découpage (situés à l'intérieur des murs). Pour deux cellules situées de part et d'autre d'un même mur, les plans de découpage des deux cellules ne sont pas parfaitement accolés (figure 2.33). D'autre part, l'espace situé entre ces deux plans contient de petits polygones ne correspondant à aucune pièce et ne satisfont aucune règle d'extraction de cellules car ils ne sont pas suffisament occlusifs. Aucune cellule ne peut donc en résulter. C'est pourquoi nous proposons de coller les plans de découpages de deux cellules adjacentes dans le but :

1. d'éviter d'avoir de petits polygones inutilisables entre les cellules ;
2. de créer une mosaïque de cellules couvrant tout l'espace contenant la scène.

La figure 2.33 représente deux cellules adjacentes C_1 et C_2. Ces deux cellules partagent un mur ayant une certaine épaisseur. Le plan de découpage P_1 (de normale N_1) passe à l'intérieur de ce mur ainsi que P_2 (de normale N_2). Ces deux plans de découpage ont une normale opposée ($N_1 = -N_2$), mais les valeurs ρ associées sont telles que ($\rho_1 + \rho_2 \neq 0$).

```
Cellule suivi_contours(Bin_T_Tree btt) {

        Bin_R_Tree P0 = NULL, P1 = NULL;     /* plans p0 et p1 */
        Cellule cell;    /* cellule retournée */
        bool trouve = FAUX;     /* booléen indiquant si une cellule est trouvée */
        Bin_R_Tree Pi, Pj;     /* avec j = i-1 */

        /* Allocation memoire */
        cell = init_cell();

        /* On recherche le plus occlusif */
        P0 = trouver_plan_occlusif(btt);
        cell = ajouter_plan_cell(cell, P0);

        /* On recherche un plan dont le demi-espace associé contient le Min de P0 */
        /* et dont l'intersection avec P0 est la plus proche du Min de P0 */
        P1 = chercher_plan_près_min_p0(btt, P0);
        cell = ajouter_plan_cell(cell,P1);

        Pj = P1;

        /* tant qu'on a pas trouve la cellule */
        tant que (trouve == FAUX) faire {

                /* On cherche le plan Pi suivant Pj */
                Pi = trouver_Pi(btt,P0,Pj);
                cell = ajouter_plan_cell(cell,Pi);

                if(tour_termine(Pi, P0)) trouve = VRAI;

                Pj = Pi;

        }

        return cell;
}
```

Fig. 2.29 – *Algorithme décrivant le suivi de contours.*

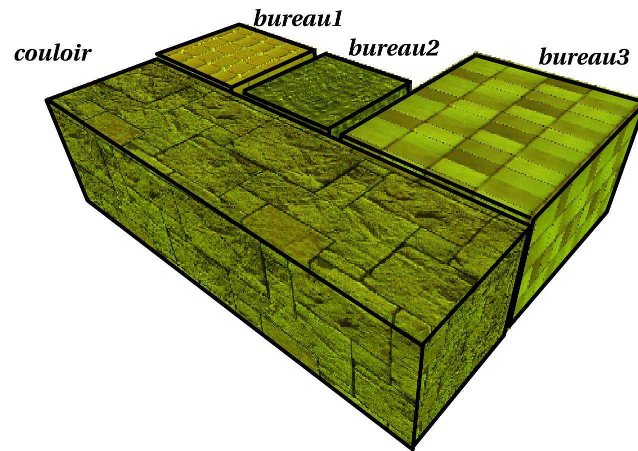


FIG. 2.30 – *Pièces adjacentes.*

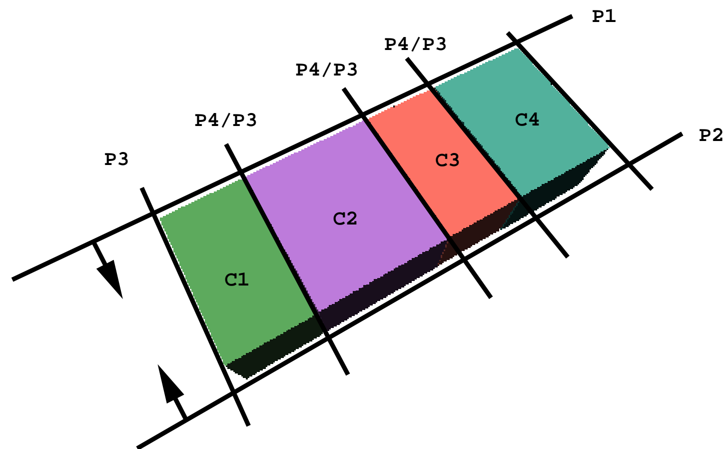


FIG. 2.31 – *Pièces entre deux murs.*

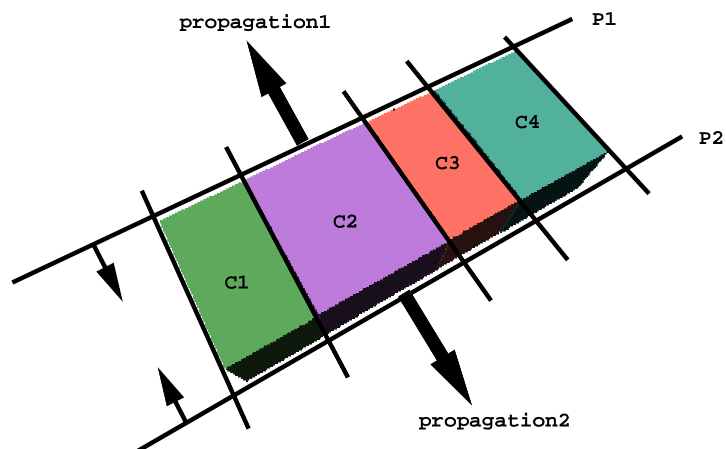


FIG. 2.32 – *Propagation de la recherche des cellules.*

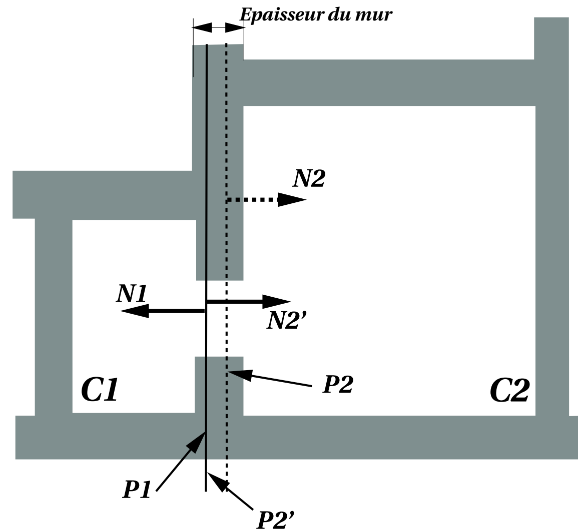


FIG. 2.33 – *Fusion de deux plans de découpage.*

Nous fusionons les plans de découpage P_1 et P_2 en remplaçant P_2 par un nouveau plan P_2' tel que $N_2' = -N_1$ et $\rho_2' = -\rho_1$. Ainsi, les cellules sont parfaitement accolées et ont une intersection vide. Notons que pour chaque cellule C déterminée, le programme essaie de fusionner tous les plans de découpage avec ceux définissant les cellules extraites avant C.

2.4.5 Appartenance d'une surface à une cellule

A un plan de découpage P d'équation $N \bullet P + \rho = 0$, nous faisons correspondre le demi-espace $E : N \bullet P + \rho \geq 0$. Une cellule est constituée de l'intersection de plusieurs demi-espaces. Nous allons maintenant voir comment déterminer les polygones appartenant à une cellule C définie par n plans de découpage ou plus précisément par n demi-espaces $E_1, E_2, ..., E_n$:

$$\begin{array}{c} E_1 : N_1 \bullet P_1 + \rho_1 \geq 0 \\ E_2 : N_2 \bullet P_2 + \rho_2 \geq 0 \\ . \\ . \\ E_n : N_n \bullet P_n + \rho_n \geq 0 \end{array}$$

Un point de l'espace appartient à C si il vérifie toutes les inéquations des demi-espace $E_1, E_2, ..., E_n$. Par conséquent, si tous les sommets d'un polygone appartiennent à C, alors le polygone tout entier appartient à C. En revanche, si certains sommets d'un polygone se trouvent à l'intérieur de C et d'autres se trouvent hors de C, ce polygone doit être découpé.

Remarque : Après le découpage de la scène certains polygones (correspondant par exemple aux murs extérieurs du bâtiment) peuvent n'appartenir à aucune cellule. Ces polygones sont insérés dans deux cellules supplémentaires contenant respectivement :

- les polygones verticaux.
- les polygones horizontaux.

2.5 Détection des ouvertures

Notre objectif est maintenant de déterminer les ouvertures de chaque cellule de la scène. Ces ouvertures nous permettront par la suite de construire un graphe exprimant les relations de visibilité entre les cellules.

2.5.1 Plans de découpage et ouvertures

Chaque cellule étant déterminée par un ensemble de plans de découpage, nous recherchons sur ces plans une liste d'ouvertures correspondant par exemple aux portes ou aux fenêtres de la cellule. Les plans de découpage étant situés à l'intérieur des murs, ils ne contiennent par conséquent aucun polygone de la scène (figure 2.34). En revanche, nous pouvons noter que certains polygones de la cellule ont une intersection non vide avec ces plans de découpage. En fait, ces polygones définissent les chambranles des portes, ou de manière plus générale les bords des ouvertures. L'intersection entre l'un de ces polygones et un plan de découpage est un segment de droite correspondant à l'arête d'une ouverture (figure 2.35).

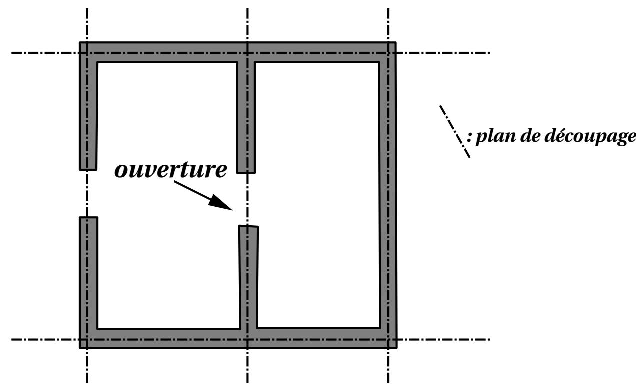


FIG. 2.34 – *Murs et plans de découpage.*

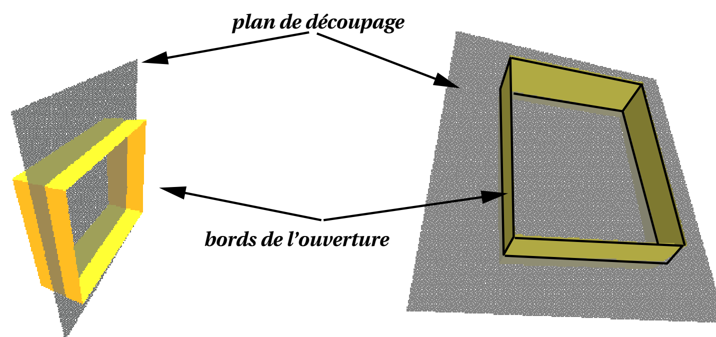


FIG. 2.35 – *Ouvertures et plans de découpage.*

2.5.2 Calcul des ouvertures

Comme nous l'avons dit précédemment, l'intersection entre un polygone et un plan est un segment de droite. Nous utilisons l'orientation des polygones pour orienter les segments de droite suivant l'algorithme de la figure 2.36.

Un polygone P est découpé par un plan de découpage en deux polygones $P1$ et $P2$. $P1$ est situé à l'intérieur de la cellule et $P2$ à l'exterieur (figure 2.37). L'orientation des segments est fixe et dépend de l'orientation des polygones. Pour chaque plan de découpage, nous avons donc une liste de segments orientés. Parcourir ces segments revient à suivre les bords des ouvertures sur un plan de découpage. Pour découvrir une ouverture, nous procédons de la manière suivante. Choisissons un segment d'ouverture AB, puis un segment BC, puis CD, etc, jusqu'à retomber

```
/* *************************************************** */
/* Découpage du polygone poly par le plan Ax + By + Cz = 0 */
/* Ce découpage donne lieu à deux polygones IN et OUT. */
/* *************************************************** */
decoup_poly_along_plane(Polygone poly, Float A, Float B, Float C, Polygone *IN, Polygone *OUT)
{
    /* On suppose que le polygone est dans la cellule */
    Bool Sommet_Precedent_Dedans = VRAI;
    /* intersection entre une arête et le plan de découpage */
    Point SD;

    pour chaque sommet S[i] de poly, faire {

        /* Si le sommet courant est hors de la cellule */
        si ( A * S[i].x + B * S[i].y + C < 0 ) alors {
            si ( i == 0 ou non sommet_precedent_dedans ) alors {
                Sommet_Precedent_Dedans = FAUX;
                Ajouter_Sommet_Polygone(S, OUT);
            }
            sinon {    /* le sommet precedent est dans la cellule */
                /* Pour la fois d'apres */
                Sommet_Precedent_Dedans = FAUX;
                SD = Decoupe_Segment(S[i], S[i-1], A, B, R);

                /* on ajoute le sommet commun à OUT */
                Ajouter_Sommet_Polygone(SD, OUT);
                /* ainsi que le sommet qui est dehors */
                Ajouter_Sommet_Polygone(S[i], OUT);
                /* Et on ajoute egalement le sommet commun à IN */
                Ajouter_Sommet_Polygone(SD, IN);
            }
        }

        sinon {    /* Le sommet courant est dans la cellule */
            si ( i == 0 ou sommet_precedent_dedans ) alors {
                Sommet_Precedent_Dedans = VRAI;
                Ajouter_Sommet_Polygone(S, IN);
                sinon    /* le sommet precedent est hors de la cellule */
                /* Pour la fois d'apres */
                Sommet_Precedent_Dedans = VRAI;
                SD = Decoupe_Segment(S[i], S[i-1], A, B, R);
                /* on ajoute le sommet commun à IN */
                Ajouter_Sommet_Polygone(SD, IN);
                /* ainsi que le sommet qui est dedans */
                Ajouter_Sommet_Polygone(S[i], IN);
                /* Et on ajoute egalement le sommet commun à OUT */
                Ajouter_Sommet_Polygone(SD, OUT);
            }
        }
    }
}
```

FIG. 2.36 – *Algorithme de découpage d'un polygone intersectant un plan.*

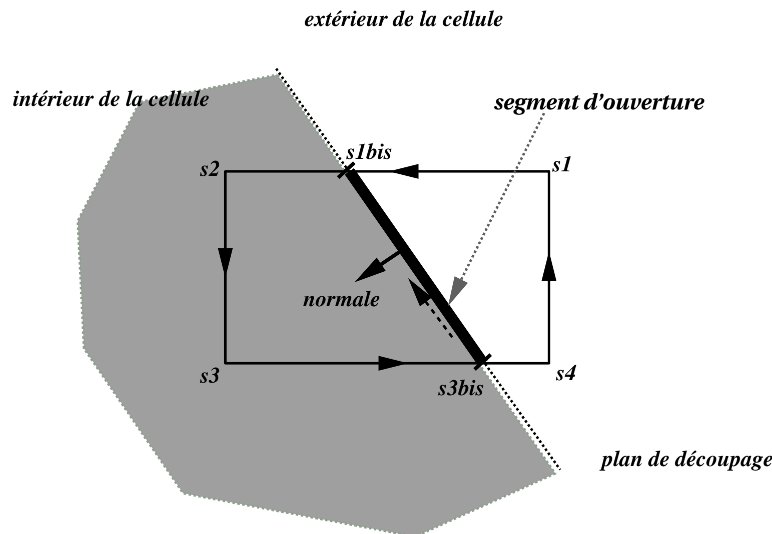


FIG. 2.37 – *Orientation des segments d'ouverture.*

sur A. L'algorithme global de détection des ouvertures est donné par la figure 2.38.

La fonction *chercher_segment* calcule les intersections entre les plans de découpage et les polygones de la cellule. La fonction *chercher_ouvertures* détermine les ouvertures à partir des segments trouvés. Ainsi, pour chaque cellule extraite de la scène, les ouvertures sont calculées. La construction du graphe de visibilité est effectuée grâce à ces ouvertures.

2.6 Création du graphe de visibilité

Le graphe de visibilité permet de connaître toutes les cellules visibles depuis une cellule donnée. Il permet d'accélérer les calculs de simulation d'éclairage et est également utilisé lors de la visualisation de la scène. Dans ce graphe, un sommet représente une cellule de l'environnement et un arc relie deux nœuds si les cellules associées sont mutuellement visibles. Dans notre mise en œuvre nous distinguons deux étapes :

- Création d'un *graphe d'adjacence* pour lequel un nœud est associé à une cellule et un arc relie deux noeuds si les cellules correspondantes ont une ouverture commune ;
- Création du *graphe de visibilité* en ajoutant des liens au graphe d'adjacence. Dans ce graphe de visibilité, un arc entre deux nœuds indique que les cellules correspondantes sont mutuellement visibles.

2.6.1 Graphe d'adjacence

Le graphe d'adjacence représente les relations de voisinage entre les cellules ayant des ouvertures communes (figure 2.39). Il est construit en même temps que les cellules. Pour chaque nouvelle cellule C, un nouveau nœud est créé. Puis, si le graphe contient des cellules C_i ayant une ou plusieurs ouvertures communes avec C, alors les arcs (C, C_i) sont ajoutés. L'algorithme de création de ce graphe est donné par la figure 2.40.

La fonction *ouverture_egale* retourne VRAI si et seulement si les deux ouvertures O et O_i sont identiques.

```
SEGMENT est une liste de segments.
Polygon_List OUV;

i = 0;

pour chaque cellule, faire {
        pour chaque plan de découpage, faire {
                pour chaque polygone P, faire {

                        si P coupe le plan de decoupage, alors {
                                SEGMENT[i] = chercher_segment(P);
                                i++;

                        }

                        /* OUV est une liste d'ouvertures, i est le nombre de segments trouves. */
                        OUV = chercher_ouvertures(SEGMENT, i);

                }
        }
}
```

FIG. 2.38 – *Algorithme de création d'une ouverture.*

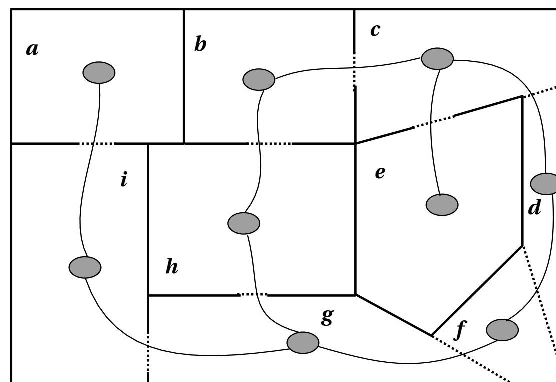
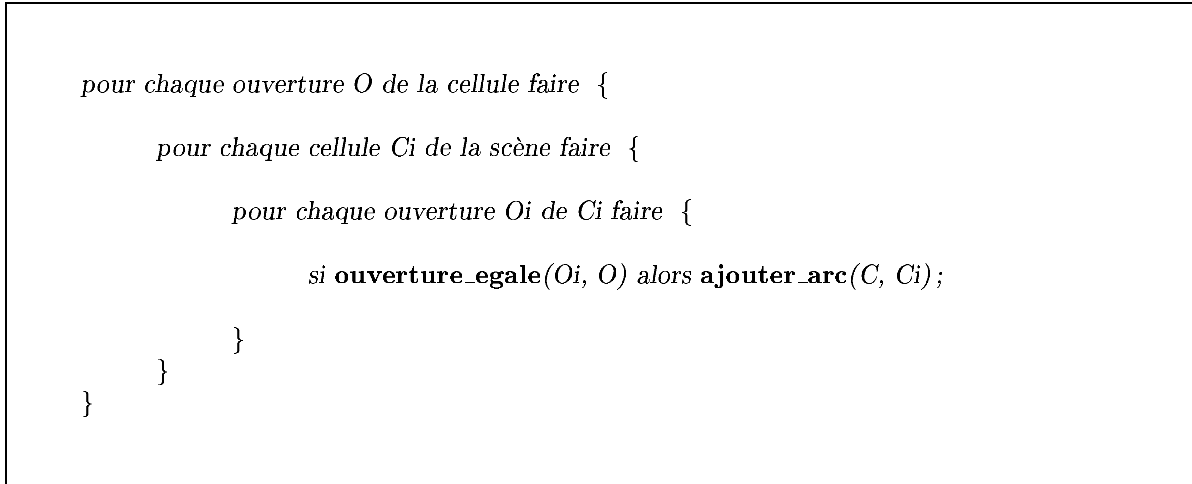


FIG. 2.39 – *Graphe d'adjacence.*



```
pour chaque ouverture O de la cellule faire {
    pour chaque cellule Ci de la scène faire {
        pour chaque ouverture Oi de Ci faire {
            si ouverture_egale(Oi, O) alors ajouter_arc(C, Ci);
        }
    }
}
```

FIG. 2.40 – *Création du graphe d'adjacence.*

2.6.2 Graphe de visibilité

La construction du graphe de visibilité (G_v) est effectuée à partir du graphe d'adjacence, après la création de toutes les cellules de la scène. Un arc est ajouté entre deux nœuds lorsque les cellules correspondantes sont visibles à travers une séquence d'ouvertures (figure 2.41).

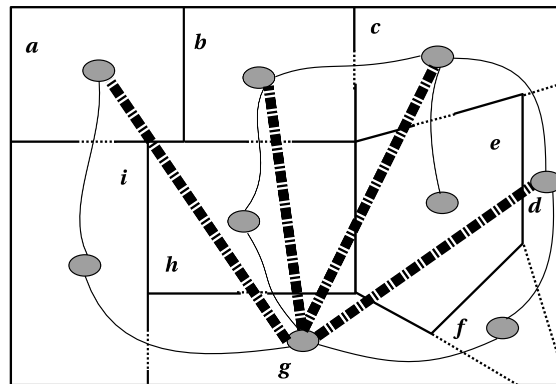


FIG. 2.41 – *Graphe de visibilité.*

Plus formellement, un arc (de G_v) relie deux cellules C_i et C_j s'il existe au moins un point I sur une ouverture de C_i et un point J sur une ouverture de C_j tel que I voie J à travers une séquence d'ouvertures $O_1, O_2, ..., O_n$ située entre C_i et C_j. Par exemple, soient quatre cellules C_1, C_2, C_3, C_4 telles que (C_1, C_2), (C_2, C_3), (C_3, C_4) soient des arcs du graphe d'adjacence. Ceci signifie que C_1 et C_2 partagent une ouverture (O_{12}). Il en est de même pour C_2 et C_3 (O_{23}) et pour C_3 et C_4 (O_{34}). Afin de déterminer si C_1 est visible à partir de C_4, N points sont aléatoirement choisis sur O_{12} et sur O_{34}. Des droites sont tracées entre les paires de points échantillonnés sur ces ouvertures. Dès que l'un des rayons coupe toutes les ouvertures intermédiaires (ici O_{23}), C_1 et C_4 sont visibles l'une de l'autre. Si par contre aucun des N rayons (N est fixé par l'utilisateur) ne coupe simultanément toutes les ouvertures, alors C_4 n'est pas visible de C_1. Ce traitement est effectué pour toutes les cellules de la scène en parcourant le graphe de manière récursive.

2.7 Mise en œuvre et utilisation du programme

L'automaticité de la structuration implique une certaine rigidité quant à l'extraction des cellules. En effet, une pièce peut satisfaire plusieurs règles de construction à la fois mais elle est extraite dès que posible par une seule règle : la plus contraignante. Or un utilisateur peut vouloir extraire des cellules particulières correspondant à des critères plus intuitifs. Dans le cas particulier de notre méthode, l'utilisateur suffisamment chevronné peut lui-même créer ses propres règles afin d'extraire des cellules spécifiques. Mais pour un utilisateur moins averti, il est impossible de suivre cette voie. C'est pourquoi avons mis en œuvre une interface graphique à l'aide de laquelle il peut sélectionner un ensemble de polygones de la scène, choisir la règle qu'il veut appliquer, ou encore fusionner plusieurs cellules.

Pour exécuter le programme en mode automatique, il suffit d'utiliser les options décrites par la figure 2.42. Si le programme est exécuté sans options, alors l'interface graphique est affichée.

```
main -i file_in -o file_out

        [-w width_room_min][-W width_room_max][-a area_min]
        [-T delta_theta][-R delta_rho]
        [-d 1] (demo_mode)
        [-r maxrays]
```

FIG. 2.42 – *Utilisation du programme.*

Les options *-i* et *-o* permettent de spécifier respectivement les fichiers source et destination du programme. Le fichier *file_in* correspond au nom du fichier contenant la scène avant la structuration (au format NFF) et le fichier *file_out* est un le nom générique de deux fichiers de sortie contenant :

1. des ensembles de polygones décrivant la géométrie de chaque cellule, séparés par des accolades.
2. pour chaque cellule : la liste des cellules voisines, la liste des cellules visibles, la liste des ouvertures et les demi-espaces définissant la cellule.

L'option *-w* (resp. *-W*) permet à l'utilisateur de modifier la largeur minimale (resp. maximale) d'une pièce et *area_min* exprime le seuil d'aire minimale pour lequel une surface est considérée comme étant occlusive. *delta_theta* et *delta_rho* sont les paramètres utilisés au cours de la classification (section 2.4.2). Enfin, *maxrays* correspond au nombre maximal de rayons lancés pour effectuer les tests de visibilité entre deux cellules. Les valeurs utilisées par défaut dans notre programme sont répertoriées dans le tableau 2.1. Pour exécuter le programme de manière à créer automatiquement toutes les cellules de l'environnement, les fichiers d'entrée et de sortie (options *-i* et *-o*) doivent être spécifiés. Si ces paramètres ne sont pas mentionnés, l'exécution est lancée en mode non automatique et deux fenêtres sont affichées à l'écran : la fenêtre de travail et la fenêtre de visualisation.

2.7.1 La fenêtre de travail

La fenêtre de travail (figure 2.43) comporte un menu et une partie graphique dédiée à OpenGl dans laquelle la scène vue de dessus est affichée en fil de fer. Dans cette fenêtre OpenGl, il est

min_width	$1m50$
max_width	$10m$
area_min	$2m^2$
delta_theta	$1deg$
delta_rho	$5cm$
maxrays	10

TAB. 2.1 – *Paramètres par défaut.*

possible de zoomer, sélectionner un polygone ou un groupe de polygones. Par ailleurs, plusieurs

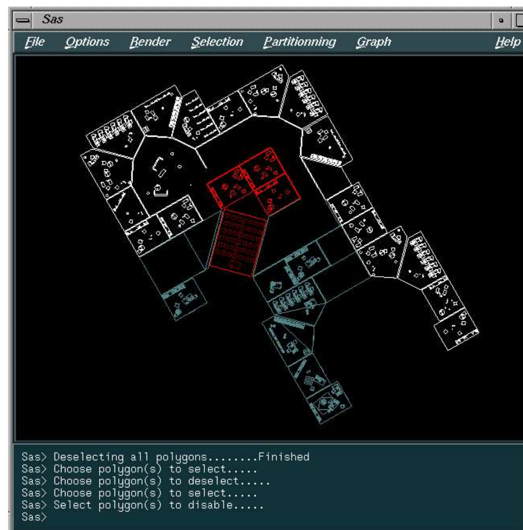


FIG. 2.43 – *Fenêtre de travail.*

modes de sélection sont proposés :

- La sélection d'un ensemble de polygones dans le but de les soumettre au découpage ;
- L'inactivation de polygones correspondant à des objets ne faisant pas partie des éléments occlusifs. Ce mode de sélection permet d'ignorer certains objets tels que des chaises dont les dossiers alignés peuvent former un plan de découpage ;
- La déselection de certains polygones pour éviter qu'ils soient pris en compte lors du découpage.

Ces modes de sélection peuvent être choisis à l'aide du menu *Selection*, mais la barre de menus fournit également d'autres facilités parmi lesquelles nous retrouvons les options du programme : largeur maximale ou minimale d'une pièce, nombre de rayons maximal pouvant être lancés au cours des calculs de visibilité, etc. L'extraction des cellules est déclenchée par le choix de l'une des règles proposées dans le menu *Partitioning* et le graphe de visibilité est calculé après la sélection de l'option *Create visibility graph* du menu *Graph*. L'option permettant de fusionner deux cellules est fournie par le menu *Partitioning*.

2.7.2 La fenêtre de visualisation

Pour que cette interface graphique soit un outil convivial et complet, nous avons ajouté une fenêtre OpenGL dans laquelle il est possible de visualiser de manière interactive (et en 3D) les cellules extraites à l'aide de la souris. La figure 2.44 illustre une salle de classe visualisée à l'aide de cette fenêtre.

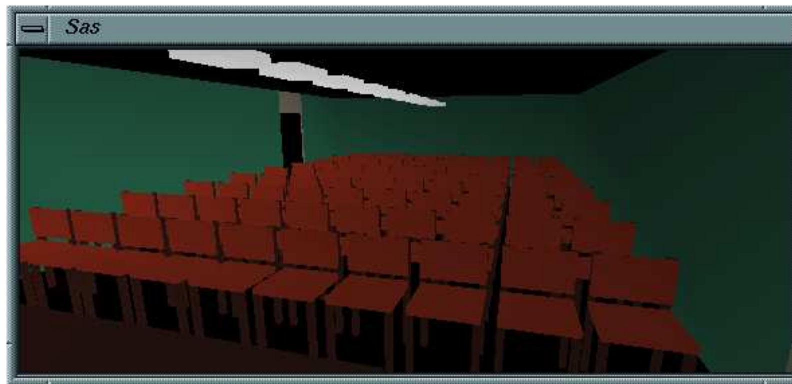


FIG. 2.44 – *Fenêtre de visualisation.*

2.8 Résultats

2.8.1 Le Soda-Hall du MIT

Dans cette section, les résultats obtenus avec notre méthode sont comparés à ceux obtenus à l'aide de la technique BSP (méthode de J. M. Airey pour des environnements axiaux). Le modèle sur lequel portent les tests est le premier étage du *Soda Hall* de Seth Teller. Cette scène, composée de 2003 facettes n'est pas meublée et comporte environ 100 pièces. Les principaux objets occlusifs sont perpendiculaires à l'un des axes du repère de la scène.

Résultats obtenus par la méthode d'Airey

Nb de cellules	1528.
Temps de calcul BSP	5 s.
Temps calcul des ouvertures	12 s.
Temps graphe de visibilité	310 s.

TAB. 2.2 – *Statistiques de la structuration par la méthode de J. Airey.*

Nous avons mis en œuvre la méthode BSP décrite par Airey. Les résultats que nous avons obtenus avec cette méthode sont décrits par le tableau 2.2. Notons que les temps de calcul dus au découpage du bâtiment et aux calculs de visibilité sont relativement faibles. En revanche, la subdivision binaire de l'espace ne permet pas d'extraire les pièces de la scène. En effet, cette méthode donne lieu à un certain nombre de cellules *dégénérées* correspondant par exemple à l'espace vide de l'intérieur d'un mur. D'autres problèmes sont traités dans [29] [30].

Résultats obtenus par notre méthode

Nb de cellules	155.
Temps de calcul des cellules	141s.
Temps calcul des ouvertures	< 1s.
Temps graphe de visibilité	<1s.

TAB. 2.3 – *Statistiques obtenues en utilisant notre méthode.*

Les résultats décrits dans le tableau 2.3 correspondent à l'exécution de notre programme en mode automatique pour l'étage du Soda Hall. Le nombre de cellules extraites du bâtiment est

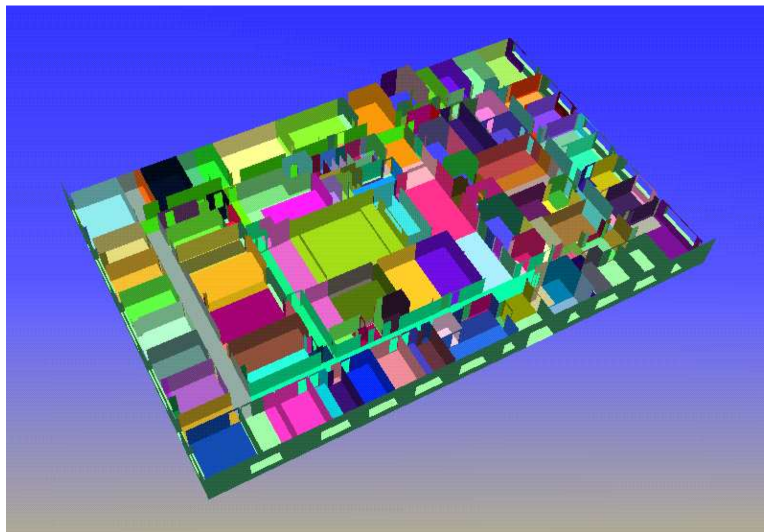


Fig. 2.45 – *Le premier étage du Soda Hall découpé par notre algorithme.*

effectivement beaucoup plus faible et chaque cellule correspond assez fidèlement à une pièce de l'étage (figure 2.45). Par ailleurs, si les temps de calcul dus à la création des cellules sont plus importants, les temps de calcul de visibilité sont beaucoup plus faibles et ceci pour plusieurs raisons :

- les cellules sont peu nombreuses et par conséquent il y a moins de nœuds à parcourir dans le graphe d'adjacence pendant les calculs de visibilité ;
- les cellules correspondent aux pièces de la scène, donc les ouvertures correspondent à des portes ou des fenêtres.

Nous avons pu comparer notre méthode de découpage à celle d'Airey pour le Soda Hall qui est un environnement axial. En revanche, pour des environnements quelconques, la mise en oeuvre d'un algorithme de type BSP impliquerait des temps de calcul beaucoup plus importants alors que grâce à notre technique, l'algorithme fonctionne de la même manière.

2.8.2 Environnement non axial

Nous avons appliqué notre programme à une scène contenant 6744 polygones. Chacune des pièces est meublée. Il y en a environ 30 (tableau 2.4). Pour cette scène comportant des pièces orientées de manière quelconque, la méthode BSP est inapplicable.

Nb de cellules	51.
Temps de calcul des cellules	128s.
Temps calcul des ouvertures	$<$ 1s.
Temps graphe de visibilité	$<$ 1s.

Tab. 2.4 – *Statistiques obtenues sur l'étage modélisé à l'IRISA.*

Le temps total d'execution du programme est de 2 mn et 28 sec. 20 secondes de ce temps ont été utilisées pour lire la scène sur le disque. Le nombre moyen d'ouvertures par cellules est 1.8 et le nombre de polygones de la scène est multiplié par 1.5 (à cause des découpage dus à la

structuration).

La figure 2.46 représente l'espace dual associé à la scène *Castle* découpée à l'aide de notre programme (figure 2.47).

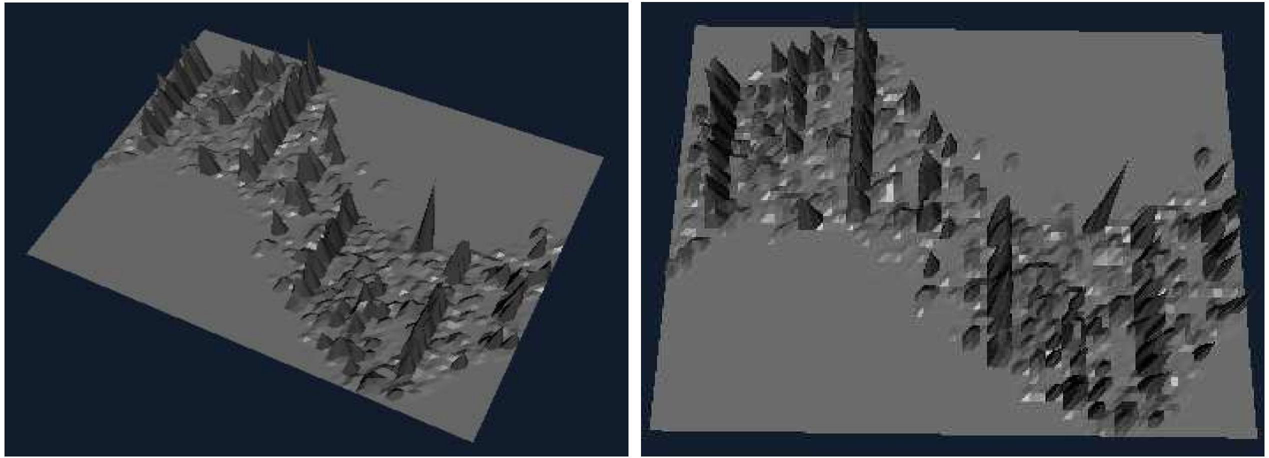


FIG. 2.46 – *Espace dual de Castle : un pic représente plusieurs polygones et la hauteur du pic correspond à la somme des aires des polygones associés.*

2.9 Discussion

Dans ce chapitre, nous avons décrit une nouvelle méthode de structuration d'environnements architecturaux complexes. Cette méthode permet d'extraire de manière automatique ou semi-automatique toutes les pièces d'un bâtiment. Elle utilise un espace dual permettant de regrouper plusieurs polygones quasi-alignés. A chacun de ces groupes de polygones, nous associons un seul plan de découpage. Puis des règles de construction permettent de mettre en correspondance ces plans de découpage et des modèles de pièce génériques. Cette mise en correspondance donne lieu à un ensemble de régions de l'espace appelées cellules. Nous déterminons leurs ouvertures au fur et à mesure de leur construction afin de construire un *graphe de visibilité*. Dans ce graphe, un noeud est associé à chaque cellule et un arc joint deux nœuds si les cellules correspondantes sont visibles à travers une séquence d'ouvertures. Ce travail a été l'objet d'une publication dans une revue internationale avec comité de lecture [31] et de deux rapport de recherche [29, 32].

A partir de cette étape de structuration, nous pouvons maintenant effectuer les calculs de simulation d'éclairage pour des environnements complexes. En effet, grâce au graphe de visibilité, nous pouvons instantanément connaître l'ensemble des cellules visibles d'une cellule donnée. Par conséquent, nous avons maintenant les moyens de calculer la radiosité de tous les polygones d'une cellule donnée sans avoir à stocker en mémoire la totalité de la base de données.

Dans le chapitre suivant, nous détaillons la méthode que nous avons conçue et qui permet d'affiner les calculs de visibilité afin de réduire encore la quantité de données présentes en mémoire au cours des calculs. Nous verrons par la suite comment prendre en compte les données fournies par ces programmes pour effectuer des calculs de simulation d'éclairage sur un ou plusieurs processeurs.

FIG. 2.47 – *La scène Castle découpée à l'aide de l'interface graphique. L'image du haut illustre la scène avec les ouvertures de chaque cellule. L'image du bas correspond à la visualisation du fichier obtenu. Une couleur différente est assignée à chaque cellule.*

Chapitre 3

Regroupement de polygones et calcul de visibilité

3.1 Problématique

Dans le chapitre 2, nous avons décrit plusieurs techniques permettant de structurer les environnements architecturaux complexes. Le résultat de cette structuration est d'une part un ensemble de *cellules* correspondant aux pièces de la scène et d'autre part un graphe exprimant les relations de visibilité entre ces cellules. Mais nous souhaitons aller au-delà de cette structuration en affinant davantage les relations de visibilité. En effet, la structuration de la scène en cellules ne permet pas de représenter fidèlement la zone de visibilité exacte de chaque pièce. Sur la figure 3.1 par exemple, différents types de relations de visibilité sont illustrées. L'image (a) correspond à un environnement intérieur architectural contenant plusieurs cellules. Un observateur situé dans la cellule $\mathcal{A}$ de cet environnement peut potentiellement voir les polygones contenus dans la zone dessinée en blanc sur l'image (b). Or d'après le graphe de visibilité, la zone potentiellement visible de la cellule $\mathcal{A}$ (en blanc) contient beaucoup plus de polygones (image c). Par ailleurs, la région visible à partir d'un objet situé dans $\mathcal{A}$ (image d) ne contient que peu de polygones.

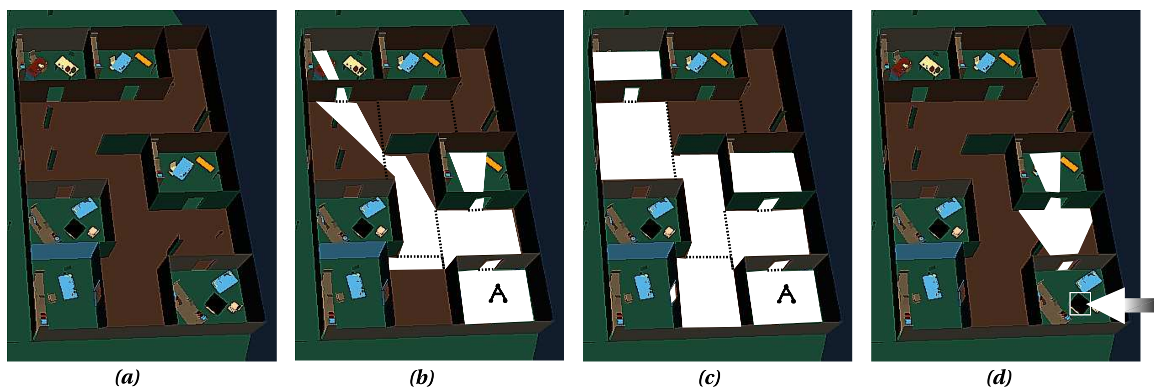


FIG. 3.1 – *Relations de visibilité : (a) scène de départ, (b) zone de visibilité de la cellule $\mathcal{A}$, (c) cellules visibles depuis $\mathcal{A}$, (d) zone de visibilité d'un objet de la cellule $\mathcal{A}$.*

Pour affiner les relations de visibilité à partir de la structuration d'un environnement, plusieurs alternatives sont envisageables. La première d'entre elles consiste à exprimer de manière plus précise les zones de visibilité pour chaque cellule. Ceci revient à redécouper toutes les cellules en plus petites parties. Toutefois ce problème est relativement complexe à résoudre et chaque cellule risque d'être morcelée en plusieurs dizaines de sous-cellules ne collant plus à la topologie de l'environnement. Une seconde alternative, cas extrême, consiste à précalculer toutes les rela-

tions de visibilité de polygone à polygone. L'inconvénient majeur de cette approche est le temps prohibitif nécessaire pour effectuer ces calculs. Enfin, une troisième alternative plus raisonnable consiste à déterminer des groupes de polygones et évaluer leurs relations de visibilité.

La notion de groupe de polygones est déjà utilisée par de nombreux auteurs, notamment dans le cadre de la méthode de radiosité [33], [34], [35], [36], [37], [38], [1]. Ces groupes de polygones (appelés *clusters* en Anglais) peuvent être obtenus de plusieurs manières :

- soit en utilisant un modeleur approprié ;
- soit à l'aide d'une technique de subdivision de l'espace (binaire, quaternaire, etc.), donnant lieu à une hiérarchie [19], [39], [40], [17] ;
- soit en construisant une hiérarchie d'objets [41], [42].

Il n'existe aucune technique automatique permettant d'obtenir directement des groupes de polygones sans créer de hiérarchie de volumes englobants. C'est pourquoi en alternative à ces solutions, nous proposons une nouvelle méthode de regroupement automatique de polygones, basée sur un algorithme de classification de type *nuées dynamiques* (ou $k-means$ en Anglais) [28, 43]. Nous proposons également un procédé permettant d'évaluer les relations de visibilité entre les groupes de polygones.

Ce chapitre est composé de 6 parties. La première est consacrée aux travaux déjà existants concernant la création et l'utilisation de groupes de polygones. Dans la seconde et la troisième partie, nous décrivons notre approche de regroupement de polygones et notre procédé de calcul de visibilité. Enfin, nous présentons notre mise en œuvre et quelques résultats avant de conclure.

3.2 Travaux antérieurs

Remarque : le terme anglais *cluster* est associé à un groupe de primitives géométriques (polygones, sphères, cônes, etc.) ou un objet composé de plusieurs polygones. Dans la suite de ce document, nous utiliserons plutôt les termes *groupe* ou *groupe de polygones* pour des raisons de commodité de langage.

Peu de travaux ont traité le problème du regroupement automatique de polygones. En revanche, la notion de groupe est fréquemment employée en simulation d'éclairage (Sillion [33], Smits et Arvo [34], Kok [35, 36], Xu [37], Rushmeier [38]).

Dans les sous-sections suivantes, nous décrivons les techniques déjà existantes :

- de regroupement de polygones dans un environnement ;
- de calcul de visibilité entre les groupes d'un environnement ;
- de simulation d'éclairage utilisant ces groupes.

3.2.1 Regroupement de polygones

La plupart des techniques d'accélération en synthèse d'images sont basées sur le regroupement de polygones. Dans la littérature, nous pouvons distinguer deux types d'approche :

- les approches de *subdivision spatiale*, [19], [39], [40], [17] dont nous ne détaillerons pas les procédés ;
- les approches dont le but est de construire une hiérarchie de volumes englobants que nous décrivons dans cette section.

Afin d'accélérer les calculs d'intersection entre un rayon et un groupe, T.M. Kay et J.T. Kajiya ont proposé en 1986 [42] une technique permettant de déterminer des volumes englobants convexes précis. Ces volumes sont constitués de plusieurs paires de plans parallèles (ou PPP) (figure 3.2 a,b ou c). Un PPP définit une zone de l'espace contenant un groupe. L'union de toutes ces zones forme un volume englobant précisément le groupe (figure 3.2 d,e). Afin d'accélerer les calculs, la normale de chaque PPP est fixée par l'utilisateur.

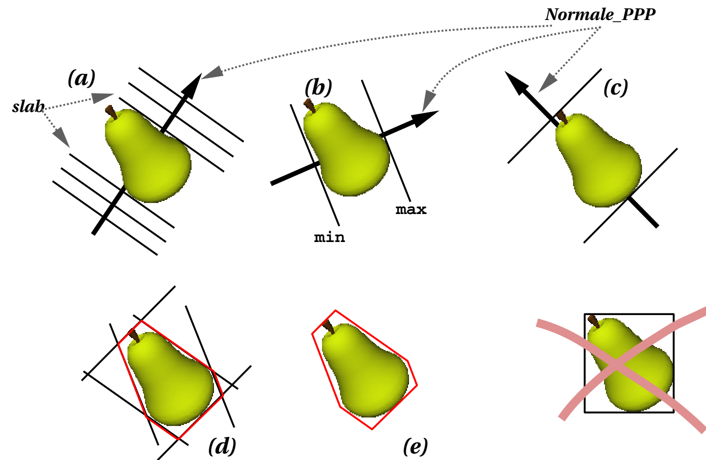


FIG. 3.2 – *Calcul de volumes englobants à l'aide de PPPs. Les images (a), (b) et (c) représentent trois PPPs. L'image (d) illustre le groupe et tous les PPPs et l'image (e) représente le groupe dans son volume englobant.*

Plus formellement, chaque PPP est défini par :

$$\begin{cases} N \bullet P + D1 = 0 \\ N \bullet P + D2 = 0 \end{cases}$$

N étant la normale du PPP, fixée par l'utilisateur. Deux valeurs Min et Max sont associées à chaque PPP :

$$\begin{cases} Min = min(D1, D2) \\ Max = max(D1, D2) \end{cases}$$

Un volume englobant est déterminé pour chaque groupe de la scène selon l'algorithme 3.3.

Chaque ppp i englobant un groupe est caractérisé par un vecteur normal $Normales_ppp[i]$ et les deux valeurs Min et max représentées dans l'algorithme par $Min_ppp[i]$ et $Max_ppp[i]$.

Dès lors que tous les volumes englobants sont créés, une hiérarchie est construite à l'aide d'une technique de "découpage au milieu" (*median cut*) décrite par l'algorithme de la figure 3.4.

Dans cet algorithme, la fonction *Median_Cut* a pour paramètre une liste de polygones triées selon un axe du repère ou selon la normale d'un PPP. Cette liste est subdivisée en deux sous-listes contenant le même nombre de groupes par la fonction *Coupe_Liste_Au_Milieu*. Chaque sous-liste est à son tour subdivisée en deux sous-listes, et ainsi de suite, de manière à créer un arbre binaire. C'est cet arbre binaire que T. Kay et J. Kajiya utilisent pour accélerer les tests d'intersections entre un rayon et les groupes de la scène. En effet, si un rayon ne rencontre pas le volume englobant d'un nœud de l'arbre, alors il ne rencontre pas non plus les fils de ce nœud. Par conséquent, il est inutile de continuer à parcourir le sous-arbre correspondant.

```
#define MAX_PPPS 100

struct Volume {
        int nb_ppps; /* nombre de ppps du volume englobant */
        float Min_ppp[MAX_PPPS]; /* Descritption des ppps */
        float Max_ppp[MAX_PPPS];
}

Volume Créer_Volume_Englobant( Objet Obj,
        Vecteur Normales_ppp[ ],
        nb_ppps) {
        struct Volume VOL;     /* Volume englobant de l'objet */
        int i;     /* Compteur de ppps */

        Pour i de 0 à nb_ppps, faire: {
                /* Création du ieme ppp */
                VOL.Min_ppp[i] = Trouver_Min(Obj, Normales[i]);
                VOL.Max_ppp[i] = Trouver_Max(Obj, Normales[i]);
        }

        return VOL;
}
```

FIG. 3.3 – *Création de volumes englobants à l'aide de PPPs.*

```
Struct ArbreBinaire {
        Liste_Polygones Plist;
        ArbreBinaire Fils_Droit;
        ArbreBinaire Fils_Gauche;
}

Median_Cut( Liste_Polygones Lp, Arbrebinaire *Arbre ) {
        Liste_Polygones Lp1, Lp2;

        Coupe_Liste_Au_Milieu( Lp, &Lp1, &Lp2 );
        Arbre->Fils_Droit.Plist = Lp1;
        Arbre->Fils_Gauche.Plist = Lp2;
        Median_Cut( Lp1, &Arbre->Fils_Droit );
        Median_Cut( Lp2, &Arbre->Fils_Gauche );
        Retourner Arbre;
}
```

FIG. 3.4 – *Création d'une hiérarchie avec un algorithme de type découpage au milieu.*

Pour J. Goldsmith et J. Salmon, l'objectif est de construire une hiérarchie de volumes englobants qui soit la plus efficace possible pour le lancé de rayons [41]. Cette hiérarchie est un arbre binaire créé de manière ascendante en ajoutant les groupes dans la hiérarchie les uns après les autres. Dans cet arbre, les feuilles représentent les volumes englobants des groupes.

```
struct OBJET {
        Objet Obj;     /* Description de l'objet */
        Volume Vol_engl;     /* Volume englobant de l'objet */
}
struct NOEUD {
        OBJET *Elem_hierarchie;     /* Objet et son volume englobant */
        NOEUD *fils_gauche;     /* Fils gauche du noeud */
        NOEUD *fils_droit;     /* Fils gauche du noeud */
} *Noeud;

Noeud Creer_Hierarchie_Objets( OBJET Scene[ ], int nb_objets )
 /* Scene est un tableau contenant nb_objets */
 {
        Noeud ARBRE = NULL;
        pour i de 0 à nb_objets faire {
                /* On ajoute les objets un à un dans l'ARBRE */
                Ajouter_Objet_Arbre( Scene[ i ], ARBRE );
        }
        retourner ARBRE;
}
```

FIG. 3.5 – *Création d'une hiérarchie de groupes.*

Leur algorithme est décrit par la figure 3.5. Dans cet algorithme, un groupe est défini par sa géométrie et un volume englobant. Notons que pour les feuilles de l'arbre, le champ *Obj* du groupe contient effectivement une description géométrique alors que pour les nœuds intermédiaires, seul le champ *Vol_engl* est utilisé. La fonction *Creer_Hierarchie_Objets* se charge de créer la hiérarchie en ajoutant les groupes de la scène les uns après les autres dans l'arbre binaire *ARBRE* selon un certain critère défini ci-après.

Au départ de l'algorithme, chaque groupe est contenu dans un volume englobant. Puis les groupes sont ajoutés successivement à la hiérarchie. Pour chaque ajout, l'arbre binaire est parcouru selon l'algorithme donné par la figure 3.6.

Dans cet algorithme, l'insertion d'un groupe dans l'arborescence ne nécessite pas le parcours de tous les nœuds de l'arbre. En effet, à chaque nœud, une heuristique permet de savoir quel sous-arbre doit être parcouru : soit le fils droit, soit le fils gauche. Cette heuristique est déterminée de la manière suivante. Lorsqu'un groupe est inséré dans un sous-arbre, le volume englobant associé à ce sous-arbre augmente. Il s'agit alors d'insérer le groupe dans le sous-arbre dont cette augmentation est la plus faible. Pour évaluer cette augmentation, Goldsmith et Salmon calculent l'aire de la frontière du volume englobant (fonction *Calculer_Augm_Volume*). Lorsque la position d'insertion du groupe est déterminée, la fonction *fusionner_nœuds* se charge d'ajouter le groupe en créant un nouveau sous-arbre dont la racine est un nouveau nœud ayant deux fils : *Obj* et $ARBRE->Vol_engl$.

```
Noeud Ajouter_Objet_Arbre( struct OBJET Obj, Noeud *ARBRE ) {
    Noeud N = Creer_Noeud( Obj );     /* Noeud créé à partir de l'objet */
    Noeud racine;     /* Racine du sous-arbre à insérer */
    float S1,S2;

    si ARBRE == NULL alors {
        retourner N;

        sinon si (ARBRE->fils_gauche == NULL et
                      ARBRE->fils_droit == NULL) alors {
            racine = Fusionner_Noeuds( Obj, ARBRE->Vol_engl );
            retourner racine;
        }

        sinon {     /* Il faut parcourir l'arbre */
            S1 = Calculer_Augm_Volume( Obj, ARBRE->fils_droit );
            S2 = Calculer_Augm_Volume( Obj, ARBRE->fils_gauche );
            si S1 < S2 alors {
                ARBRE->fils_droit =
                Ajouter_Objet_Arbre( Obj, ARBRE->fils_droit );
                retourner ARBRE;
            }
            sinon {
                ARBRE->fils_gauche =
                Ajouter_Objet_Arbre( Obj, ARBRE->fils_gauche );
                retourner ARBRE;
            }
        }
    }
```

FIG. 3.6 – *Insertion d'un groupe dans un arbre binaire.*

3.2.2 Calculs de visibilité entre groupes de polygones

Afin de réduire la part importante des calculs nécessaires pour établir les relations de visibilité entre les groupes d'un environnement, E. Haines et J. R. Wallace construisent des volumes appelés *tubes*. Ils permettent de déterminer les polygones provoquant une occlusion partielle ou totale entre deux groupes. Ces volumes sont construits en 3 étapes à partir des boîtes englobantes des groupes (figure 3.7) :

(a) Créer une boîte englobante autour de chacun des deux groupes ;

(b) Créer une boîte englobant les deux groupes à la fois ;

(c) Affiner cette boîte englobante de manière à créer un volume convexe contenant les deux groupes.

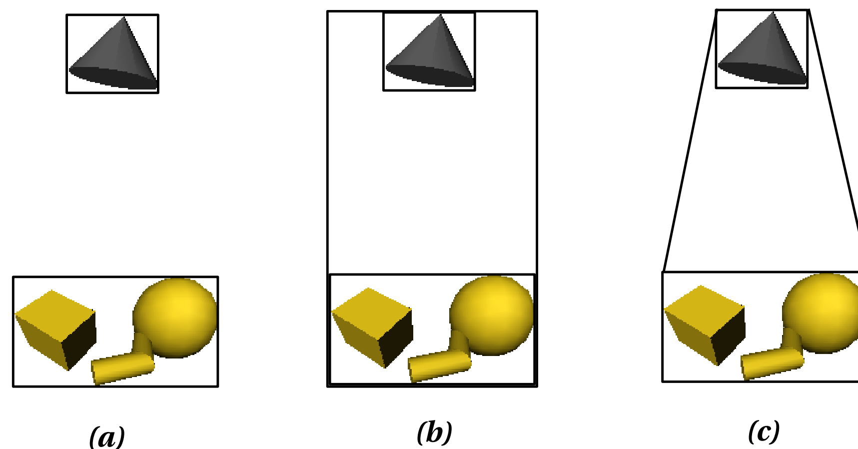


FIG. 3.7 – *Création d'un volume de visibilité.*

Si les relations de visibilité entre deux groupes sont déterminées par lancé de rayon, seuls les polygones contenus dans les *tubes* sont pris en compte, de manière à réduire le nombre de tests d'intersection à effectuer.

Dans [1, 23, 12, 24], Teller et al. supposent que les groupes de polygones sont déjà constitués, par exemple lors de la modélisation de la scène. Leur objectif est d'effectuer des calculs de radiosité dans des environnements complexes en s'appuyant sur une subdivision binaire de type BSP [12]. Cette subdivision binaire donne lieu à des régions appelées *cellules* pouvant comporter une ou plusieurs ouvertures. Les auteurs utilisent ces ouvertures pour déterminer les relations de visibilité entre les cellules. En effet, deux cellules C_1 et C_2 peuvent être mutuellement visibles à travers une séquence d'ouvertures. Ainsi, pour savoir si C_1 est visible de C_2, il suffit de trouver une droite passant par toutes les ouvertures intermédiaires (figure 3.8).

Une fois ces relations de visibilité établies, Teller et. al. ajoutent une nouvelle structure de données indiquant pour chaque paire de groupes la liste des polygones situées entre eux (les *blockers*) [23].

De nombreux ouvrages traitent également le problème du calcul de visibilité mais de manière plus générale [12], [44], [45], [46], [23], [24], [47], [48], [49], [50], [51], [52], [53], [54], [55]. Bien que ces techniques ne soient pas décrites dans ce document, le lecteur pourra y trouver de précieuses informations.

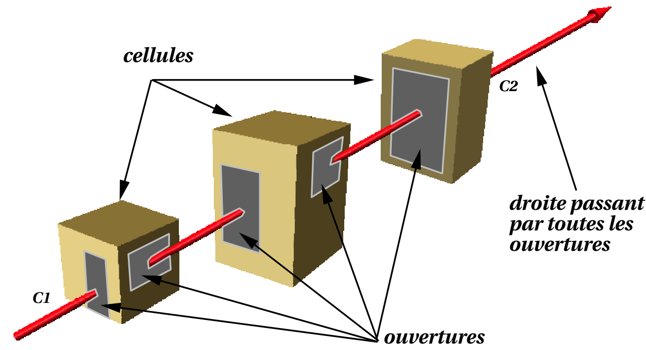


FIG. 3.8 – *Droite passant par les ouvertures de plusieurs cellules.*

3.2.3 Utilisation de groupes de polygones en simulation d'éclairage

Afin d'accélerer le processus de simulation d'éclairage, plusieurs méthodes introduisent la notion de groupe afin de réduire les coûts de calcul. On peut discerner deux familles d'algorithmes. Pour la première, l'espace est subdivisé en plusieurs régions sur lesquelles les calculs de radiosité sont effectués. Pour la seconde, c'est une hiérarchie de groupes qui est utilisée.

Xu et. al. subdivisent l'espace en plusieurs régions dont les frontières sont transparentes [37]. Ils gérent une radiosité à deux niveaux: inter-régions et intra-régions. En effet, à l'intérieur de chaque région les calculs de radiosité sont effectués de manière précise et les frontières de chaque région sont considérées comme des polygones transparents.

Kok utilise des groupes de polygones [35] [36], mais aucun volume englobant n'est créé. Chaque polygone contient simplement un marqueur indiquant à quel groupe elle appartient. L'éclairement calculé pour un groupe est alors directement réparti sur les polygones qu'il contient (figure 3.9).

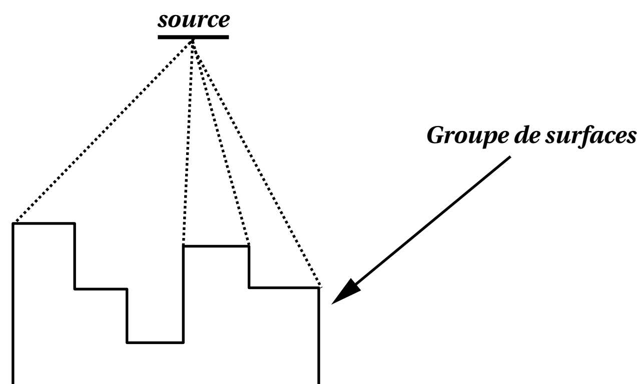


FIG. 3.9 – *Eclairement d'un groupe de polygones.*

Rushmeier et al. utilisent eux aussi des groupes de polygones [38] contenus dans des volumes englobants. Dans leur approche, les frontières des volumes englobants sont subdivisés en mailles (figure 3.10 a) et pour chaque maille une transmittance et une réflectance sont déterminées. Pour effectuer ce calcul, des rayons sont lancés à l'intérieur du volume englobant (figure 3.10 b).

La réflectance correspond au rapport :

$$\frac{R_{intersecte}}{R_{total}}$$

et la transmittance correspond au rapport :

$$\frac{R_{ok}}{R_{total}}$$

où R_{total} est le nombre total de rayons lancés, $R_{intersecte}$ le nombre de rayons intersectant au moins un polygone et R_{ok} le nombre de rayons ayant réussi à traverser tout le volume englobant sans intersecter de polygone. Enfin, les calculs de simulation d'éclairage sont effectués directement pour des groupes de polygones.

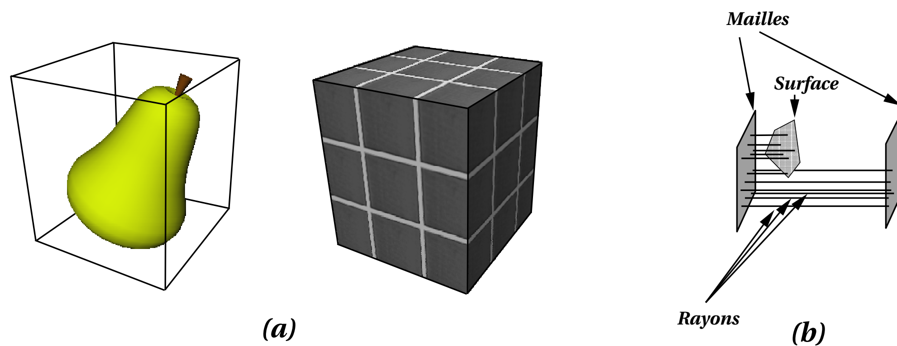


FIG. 3.10 – *(a) Maillage des bords d'une boîte englobante, (b) Calcul d'un cœfficient de réflectance et de transmission pour une maille.*

Smits et. al. utilisent la technique de regroupements de polygones [34] proposée par Goldsmith et Salmon [41]. Pour effectuer les calculs de simulation d'éclairage, ils créent deux types de liens α et β entre deux groupes (figure 3.11). Les liens α permettent d'effectuer les calculs de radiosité très précisément, alors que le second type de lien est une approximation très grossière des facteurs de forme entre les différents groupes. Toute la technique consiste à savoir quel type de lien employer en fonction de la taille des groupes, leur distance, etc.

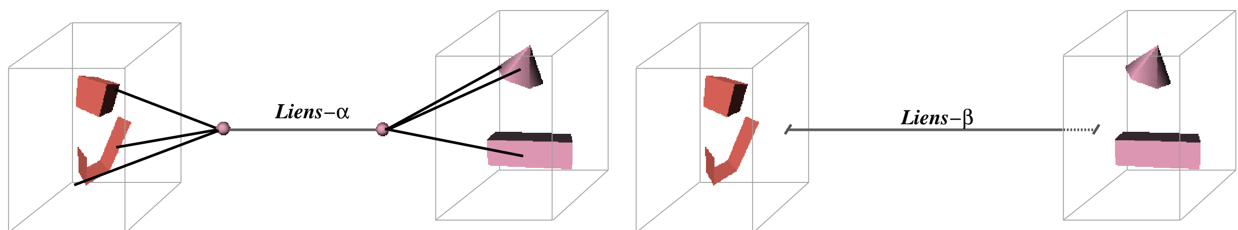


FIG. 3.11 – *Liens α et β.*

F. Sillion et. al. considèrent un groupe comme un milieu semi-transparent caractérisé par des cœfficients de transmission et de diffusion. En effet, une partie du flux incident traverse le groupe et une autre partie est réfléchie de manière diffuse [48, 33, 56].

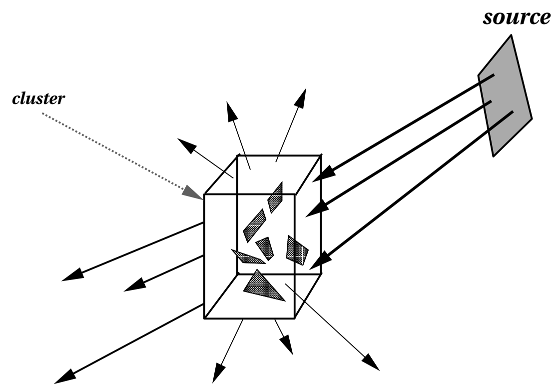


Fig. 3.12 – *Transmission d'énergie à travers un groupe.*

Teller quant à lui suppose que les groupes de polygones sont fournis par le modeleur [1]. Il calcule très précisément la visibilité entre tous les groupes de polygones de la scène. Puis à partir de ces relations de visibilité, les calculs de simulation d'éclairage sont effectués à l'aide d'une technique classique de collecte (d'énergie) basée sur une méthode itérative de type Jacobi.

```
Lire_Groupe(R);
Pour chaque groupe émetteur S Visible de R faire {
        Lire_Groupe(S);
        Lire_Blockers(S,R);
        Installer_Liens(R,S);
        Collecte_Jacobi(R);
        Supprimer(S);
        Supprimer(Blockers);
}

Invoquer_PushPull(R);
Supprimer(R);
```

Fig. 3.13 – *Simulation d'éclairage de Teller dans des environnements complexes.*

La figure 3.13 décrit l'algorithme utilisé successivement pour chaque groupe R (récepteur) de la scène. La fonction *Collecte_Jacobi* de cet algorithme collecte l'énergie lumineuse pour chaque maille d'un groupe selon la méthode itérative de Jacobi. Une fois la collecte terminée pour toutes les sources, la fonction *Invoquer_PushPull* se charge d'effectuer la mise à jour des mailles du groupe. Lorsque les calculs sont effectués pour un groupe R, la fonction *supprimer* libère l'espace mémoire correspondant aux liens, aux blockers et aux groupes.

3.3 Notre approche de regroupement de polygones

Dans les méthodes décrites précédemment, la notion de groupe est souvent employée. Cependant, à partir d'une base de données non structurée, aucune technique automatique ne permet

d'extraire des groupes de polygones sans créer de hiérarchie.

C'est pourquoi nous proposons une alternative à ce problème. Notre méthode, basée sur un algorithme de classification appelé *les nuées dynamiques*, permet de déterminer rapidement des groupes de polygones. Pour accélérer les calculs dans le cas d'environnements complexes, nous utilisons la structuration décrite dans le chapitre 2. Rappelons que cette structuration donne lieu à :

- un ensemble de cellules correspondant à des pièces et pouvant comporter une ou plusieurs ouvertures (portes, fenêtres, etc.) ;
- un graphe exprimant les relations de visibilité entre ces cellules.

Des groupes de polygones sont créés dans chaque cellule et leurs relations de visibilité sont déterminées à l'aide des ouvertures des cellules.

Notre méthode est totalement automatique et ne nécessite aucune connaissance a priori sur le modèle. Tout type de primitive géométrique (polygones, sphères, cônes, etc.) peut être utilisé dans cet algorithme moyennant un calcul de barycentre. Par ailleurs seulement quelques minutes sont nécessaires pour créer les groupes de polygones et évaluer leurs relations de visibilité pour des environnements contenant plusieurs dizaines de milliers de polygones. De plus l'utilisateur a la possibilité de contrôler la taille des groupes en agissant sur le nombre de polygones moyen contenus dans chaque groupe.

3.3.1 Hypothèses

Comme nous l'avons précisé en introduction, cette technique de regroupement de groupes peut être utilisée pour tout type de polygone ou de primitive géométrique. Nous utilisons le barycentre de ces primitives pour déterminer les groupes. Pour des primitives géométriques de base telles que des polygones 3D convexes à 3 ou 4 sommets, des sphères, des cônes ou des cubes, le barycentre est facile à déterminer. En revanche, pour des surfaces plus complexes que des surfaces (surfaces Nurbs ou Splines par exemple) dont le barycentre ne peut pas être calculé simplement, nous proposerons d'autres solutions.

3.3.2 Les nuées dynamiques

Notre algorithme de regroupement de polygones est basé sur une méthode de classification utilisée en analyse d'images. Usuellement utilisée dans un espace de dimension 2, cette méthode permet de retrouver n groupes de points parmi un ensemble de N points, n étant fixé par l'utilisateur. Ce procédé regroupe les points par proximité en utilisant des barycentres mobiles. Afin de mieux comprendre le mécanisme de ce processus, nous présentons d'abord la méthode en 2D. Ensuite, nous décrivons notre extension de cet algorithme pour regrouper les polygones d'un environnement 3D.

Principe

Au départ de l'algorithme, n barycentres sont choisis de manière aléatoire. Ils se déplacent à chaque étape du calcul jusqu'à atteindre une position fixe. Lorsque tous les barycentres sont stabilisés, les groupes sont constitués.

La figure 3.14 (a) montre n barycentres matérialisés par des croix et choisis de manière aléatoire. Dans cet exemple, $n = 3$. Dans la figure 3.14 (b), un groupe de points est associé à chaque barycentre. Pour effectuer ce calcul, un point I est inséré dans un groupe j si :

$$dist(B_j, I) = \min_{i \in [1,n]} (dist(B_i, I)),$$

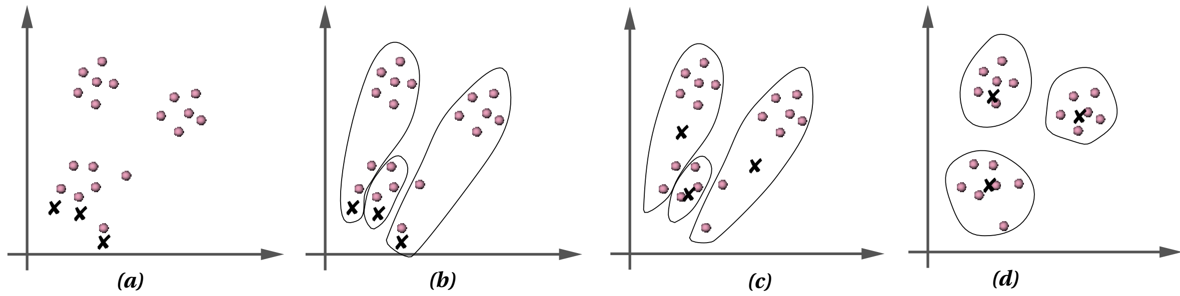


FIG. 3.14 – *Les nuées dynamiques en 2D.*

B_j étant le barycentre correspondant au groupe j.

Pour chaque groupe formé, on recalcule son barycentre (figure 3.14 c). Nous avons donc n nouveaux barycentres à partir desquels n nouveaux groupes sont constitués. Pour chaque groupe, le barycentre est recalculé, etc. Ce processus est répété tant que les groupes ne sont pas stabilisés (i.e. tant que les barycentres changent de position).

Théoriquement, il peut y avoir des états instables. C'est le cas si un barycentre oscille indéfiniment entre deux groupes de points légèrement différents. Dans la pratique, ce cas est très rare et les groupes sont constitués au bout de 3 ou 4 étapes. Aussi, pour prévenir ces états instables, il suffit par exemple de conserver les groupes obtenus au bout de 10 étapes.

Algorithme

L'algorithme général des nuées dynamiques est donné par la figure 3.15. Dans cet algorithme, la première étape consiste à choisir aléatoirement N_g barycentres. N_g étant le nombre de groupes que l'utilisateur souhaite obtenir. Le calcul permettant de déterminer la position du barycentre B_i du groupe i est donné par :

$$\overrightarrow{OB_i} = \frac{\sum_{j \in [1,n_i]} \overrightarrow{OP_j}}{n_i},$$

n_i étant le nombre de points du groupe i.

Cette technique de classification donne lieu à N_g goupes de points proches les uns des autres. Notre objectif est maintenant d'adapter cet algorithme pour créer des groupes de polygones dans un espace à trois dimensions.

3.3.3 Application à la classification de polygones

Par rapport à l'algorithme en deux dimensions :

- les objets à regrouper ne sont plus des points mais des polygones ;
- les barycentres mobiles sont calculés en trois dimensions ;

C'est pourquoi pour adapter l'algorithme à notre problème, il est nécessaire de déterminer :

- une notion de distance entre un barycentre 3D et un polygone (ou d'un objet géométrique) de manière à pouvoir trouver le barycentre le plus proche d'un polygone ;
- une notion de barycentre pour un groupe de polygones (ou d'objets géométriques).

```
Polygone **NuéesDynamiques(Ng, Np, Polygone polygones[ ]) {
        entier Ng; /* nombre de groupes recherchés */
        entier Np; /* nombre de polygones à classer */
        Polygones Rk[Ng],Rk-1[Ng]; /* Barycentres des groupes à déterminer */
        Polygones G[Ng][Np]; /* Ng = Nombre de groupes */
        entier J, i, j, k;
        Réel D, d[Np] ; /* d[i] est la distance entre le polygone i et un barycentre */

        /* Choisir aléatoirement 1 barycentre pour chaque groupe */
        Rk = ChoixAléatoire(N, polygones);
        Rk-1 = O;

        Tant que Rk ≠ Rk-1 faire {

                /* Pour chaque polygone */
                Pour i de 1 à Np faire {
                        D = + INFINITY;

                        /* Pour chaque barycentre */
                        Pour j de 1 à N faire {
                                d[i] = Distance(Rk[j],polygones[i]);
                                /* Si la nouvelle distance est plus petite, on mémorise */
                                /* la nouvelle distance et le goupe correspondant */
                                si (d[i] < D) alors D = d[i] ; J = j;
                        }

                        /* J est l'indice du groupe dont le barycentre est le plus */
                        /* proche de polygones[i]. */
                        AjouterPolygoneGroupe( polygones[i], G[J] );
                }

                /* Les nouveaux barycentres sont calculés */
                Rk-1 = Rk;
                Pour j de 1 à N faire Rk[j] = barycentre(G[j]);

        }
        retourner G;
}
```

FIG. 3.15 – *Algorithme des nuées dynamiques.*

Avec ces deux notions, l'algorithme des nuées dynamiques peut immédiatement être adapté à notre problème. D'ailleurs, le nouvel algorithme est identique à celui présenté en deux dimensions. La fonction *Choix_Aléatoire* détermine N_g barycentres de manière aléatoire. Dans notre implémentation, nous choisissons le premier sommet des N_g premiers polygones de la scène ($N_g << N$, N étant le nombre total de polygones dans la scène). La fonction *Distance* calcule la distance entre un barycentre $R^k[j]$ et le polygone $poly[i]$. Enfin, la fonction *Barycentre* détermine le barycentre du groupe de polygones $G[j]$. Voyons maintenant comment calculer la distance entre un polygone et un barycentre.

Notion de distance Barycentre-Polygone

Lorsque tous les barycentres sont déterminés, chaque polygone est inséré dans un groupe. Pour savoir à quel groupe appartient un polygone P, nous calculons la distance entre P et chaque barycentre. P est inséré dans un groupe j si :

$$dist(B_j, P) = \min_{i \in [1,n]}(dist(B_i, P)),$$

B_j étant l'un des n barycentres.

Pour évaluer la distance entre un polygone et un barycentre, plusieurs solutions sont envisageables. Il est possible de prendre par exemple :

- la distance entre le barycentre et le sommet le plus proche :

  $$Distance = \min_{i \in [1,n]}(dist(P, S_i))$$

  S_i étant le i^{eme} sommet d'un polygone à n côtés.
- la distance orthogonale du point au plan du polygone.
- etc.

Intuitivement, plusieurs polygones sont proches les uns des autres si leurs barycentres sont proches les uns des autres. C'est pourquoi nous avons choisi la distance suivante entre un point P et un polygone P_o :

$$dist(P, P_o) = dist(P, G_s)$$

G_s étant le barycentre du polygone P_o contenant n sommets $S_1, S_2, ..., S_n$. Avec cette distance, on voit bien que le grand polygone de la figure 3.16 est bien dans le même groupe que la poire.

Notons que le choix de cette distance influe sur le regroupement. Par exemple, si la distance choisie entre un point P et un polygone P_o est :

$$dist(P, P_o) = \min_{i \in [1,n]}(dist(P, S_i))$$

alors les polygones seront regroupés de la manière illustrée sur la figure 3.17. Sur cette figure, le grand polygone n'est pas dans le même groupe que la poire.

Si les primitives géométriques sont trop complexes comme par exemple les surfaces implicites, le calcul de leur barycentre est très coûteux. Il suffit alors de modifier le calcul de distance et utiliser par exemple la somme des distances aux sommets ou bien la distance par rapport au plan du polygone, etc. Notons qu'en modifiant ce calcul de distance, l'aspect des groupes peut être différent, comme le montre la figure 3.18 (ou 3.37 en couleurs). Notre algorithme peut donc être utilisé pour n'importe quel type de primitive géométrique.

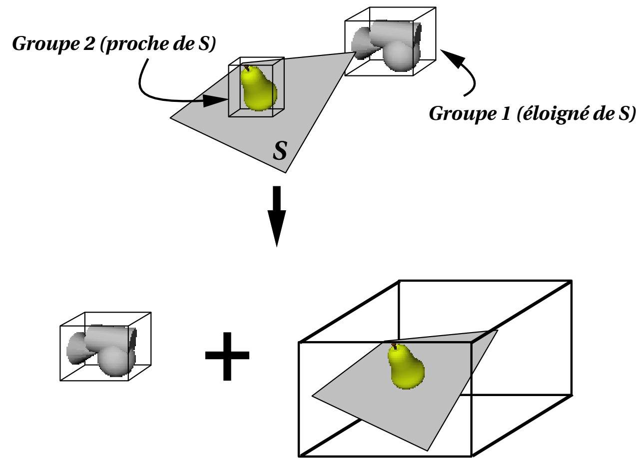


FIG. 3.16 – *Calcul des groupes : avec la distance au barycentre du polygone.*

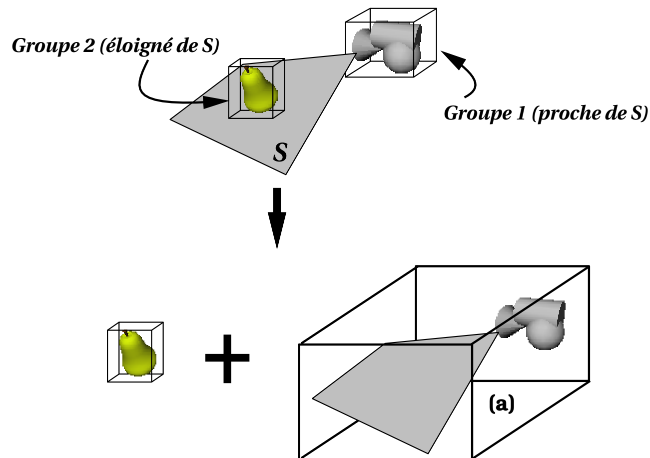


FIG. 3.17 – *Calcul des groupes : avec une autre distance.*

Calcul du barycentre d'un groupe de polygones

Dans notre approche, une scène 3D est décrite à l'aide de polygones plans et convexes comportant 3 ou 4 sommets. Pour déterminer le barycentre d'un ensemble de polygones, nous déterminons le barycentre des sommets des polygones. Ce calcul est effectué de la manière suivante :

$$\overrightarrow{OB_j} = \frac{\sum_{i=1}^{N_p}(\sum_{j=1}^{N_{si}} \overrightarrow{OS_j^i})}{N_{sommets}}$$

S_j^i étant le j^{eme} sommet du polygone i et $N_{sommets} = \sum_{i=1}^{N_p} N_{si}$. Cela revient à calculer le barycentre des barycentres des polygones.

Choix du nombre de groupes dans une cellule

Dans l'algorithme des nuées dynamiques, le nombre de groupes à créer doit nécessairement être connu à l'avance. Or il est difficile de demander à l'utilisateur de fixer ce nombre pour chaque cellule ou pour tout l'environnement. Il est plus simple de fixer le nombre moyen de polygones N_m que l'on souhaite obtenir par groupes. Dans notre mise en œuvre N_m vaut 50 par défaut, mais l'utilisateur peut modifier cette valeur dans le programme. Le nombre de groupes N_g vaut :

$$N_g = \frac{N}{N_m}$$

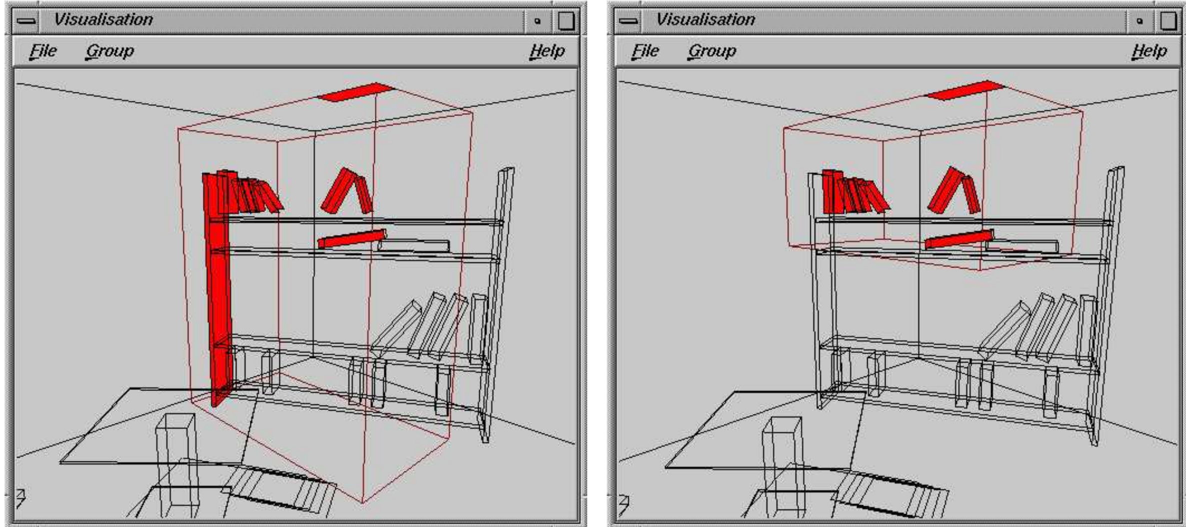


FIG. 3.18 – *Image de gauche :* $dist(B_i, P) = \min_{i \in [1,n]}(dist(P, S_i))$ *; Image de droite :* $dist(B_i, P) = dist(B_i, G_s)$ *;* B_i *étant le barycentre du groupe* i*,* S_i *étant le* i^{eme} *sommet du polygone* P *et* G_s *étant le barycentre de* P*.*

où N est le nombre de polygones de la cellule (ou de la scène).

Regrouper des polygones dans des environnements architecturaux

Remarquons que l'intersection de deux boîtes englobant deux groupes différents n'est pas nécessairement nulle. D'après notre définition, un groupe est un groupe de polygones proches les uns des autres, chaque polygone étant contenu dans un groupe et un seul.

Pour accélérer le regroupement de polygones, nous utilisons la structuration de l'environnement en cellules (chapitre 2). Au lieu d'effectuer les calculs sur tous les polygones de la scène à la fois, nous appliquons l'algorithme des nuées dynamiques successivement pour chaque cellule.

Notons que si la base de données est très volumineuse, il est possible d'effectuer les calculs en ne chargeant en mémoire qu'une seule cellule à la fois.

Enfin, lorsque tous les groupes sont créés, nous déterminons leurs relations de visibilité selon l'algorithme décrit dans la section suivante.

3.4 Notre approche de calcul de visibilité

Pour déterminer les groupes visibles à partir d'un groupe donné, plusieurs techniques déjà existantes peuvent être employées. En effet, la méthode de Teller [1] est particulièrement bien adaptée à notre problème. Cependant, si cette technique est très précise, elle reste néanmoins coûteuse et compliquée à mettre en œuvre. D'autre part, Teller [12] ou E. Haines [57] proposent :

1. de construire un volume contenant les deux groupes ;

2. de rechercher dans ce volume les polygones susceptibles de créer une occlusion entre les deux groupes.

Nous préférons aborder le problème sous un angle différent en déterminant les groupes contenus dans la zone visible d'un groupe donné à travers une ou plusieurs ouvertures. Cette approche est peu coûteuse car elle utilise les connaissances de l'étape de structuration de l'environnement. En effet, les cellules correspondent à des sous-parties de l'espace ne contenant pas d'élément très occlusif. Nous pouvons par conséquent supposer que deux groupes de la même cellule sont mutuellement visibles. D'autre part deux groupes situées dans deux cellules différentes sont visibles à travers une séquence d'ouvertures (figure 3.19).

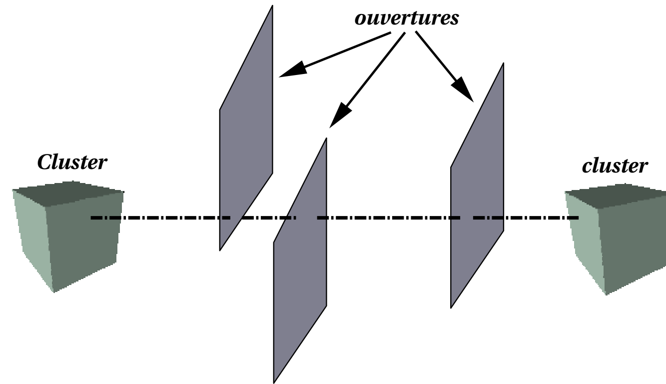


FIG. 3.19 – *Calcul des relations de visibilité pour un groupe de polygones.*

Pour exprimer les relations de visibilité entre les groupes de l'environnement nous utilisons un graphe dans lequel un nœud représente un groupe. Un arc est créé entre deux nœuds lorsque les deux groupes correspondants sont visibles l'un de l'autre. La construction de ce graphe est effectuée en deux étapes :

1. Déterminer les relations de visibilité intra-cellules. Pour un groupe donné C appartenant à une cellule Cel, cette opération consiste à ajouter un arc entre C et tous les autres groupes de Cel ;

2. Déterminer les relations de visibilité inter-cellules en recherchant les groupes visibles d'un groupe donné C à travers une séquence d'ouvertures. Pour effectuer ce calcul, nous construisons un volume appelé *volume de visibilité* correspondant à la zone de l'espace visible à partir de C à travers une ouverture. Tous les objets contenus dans ce volume de visibilité sont potentiellement visibles de C. Si des ouvertures se trouvent également dans ce volume de visibilité, la recherche est propagée dans d'autres cellules afin de déterminer tous les groupes visibles de C.

Les sous-sections suivantes décrivent les calculs effectués pour créer ces volumes de visibilité.

3.4.1 Les volumes de visibilité

Soit C, un groupe contenu dans une cellule X. Rappelons que :

- tous les groupes de X sont considérées comme étant visibles de C ;
- des groupes n'appartenant pas à X peuvent être visibles à travers les ouvertures de X.

Pour simplifier les calculs, nous supposons que les ouvertures des cellules sont des polygones rectangulaires. Cette hypothèse est justifiée dans le cas d'environnements intérieurs car les cellules correspondent aux pièces d'un bâtiment et par conséquent les ouvertures sont très souvent des portes ou des fenêtres. Pour des ouvertures non rectangulaires, nous utilisons un rectangle

englobant l'ouverture. Notre objectif est maintenant de déterminer un volume correspondant à la zone visible de C à travers une ouverture rectangulaire O: le volume de visibilité $V(C,O)$. Pour effectuer ce calcul, nous utilisons une boîte englobante, alignée avec les axes du repère et contenant tous les polygones de notre groupe C. Or le volume de visibilité défini par une boîte englobante cubique et une ouverture rectangulaire correspond à une pyramide s'appuyant O et contenant tous les groupes susceptibles d'être visibles de C.

Algorithme

```
Rechercher_Visibles( cellule X ) {

        Pour chaque groupe Ci de X faire {
                Creer_Tous_Visibles( Ci, X );
                Pour chaque ouverture O de X faire {
                        V = Creer_Volume( Ci, O );
                        Rechercher_Visible_Volume( V, Ci, O, X );
                }
        }
}


Rechercher_Visible_Volume( Volume V,
    Groupe C,
    Ouverture O,
    cellule X )
{

        Xa = Trouver_Cellule_Adjacente( X, O );
        Chercher_Groupes_Visibles( Ci, V, Xa );

        Pour chaque ouverture Oi de Xa faire {
                si Oi est contenu dans V alors {
                        Vn = Modifier_Volume( O, Oi, V, C );
                        Rechercher_Visible_Volume( Vn, C, Oi, Xa );
                }
        }
}
```

FIG. 3.20 – *Calcul des relations de visibilité pour un groupe de polygones.*

La fonction *Rechercher_Visibles* décrite dans l'algorithme 3.20 est appelée pour chaque cellule de l'environnement. Cette fonction a pour objectif de déterminer les relations de visibilité pour tous les groupes de polygones d'une cellule X donnée. La première étape de cette recherche consiste à ajouter des arcs dans le graphe entre C et tous les autres groupes de X (fonction *Creer_Tous_Visibles*). Une fois cette opération effectuée, il reste à déterminer les groupes visibles de C à travers les ouvertures de X à l'aide des volumes de visibilité. Ces volumes sont construits par la fonction *Créer_Volume* et les groupes contenus dans le volume de visibilité sont déterminés par *Rechercher_Visible_Volume*. Ensuite, la fonction *Trouver_Cellule_Adjacente* recherche la cellule adjacente C_a ayant la même ouverture O que X et la fonction *Chercher_Groupes_Visibles* parcourt les groupes de la cellule X_a, détermine ceux qui sont contenus dans le volume de visibi-

lité V et ajoute un arc dans le graphe entre C_i et ses groupes visibles. Lorsque tous les groupes de la cellule X_a sont parcourus, les ouvertures contenues dans le volume V sont déterminées. En effet, par ces ouvertures C voit d'autres groupes. Un nouveau volume de visibilité est créé en fonction du groupe C et des ouvertures O et O_i (fonction *Modifier_Volume*). Puis une nouvelle recherche est effectuée dans ce volume (par *Rechercher_Visibles_Dans_Volume*).

Création d'un volume de visibilité

La pyramide correspondant au volume de visibilité est déterminée par 4 demi-espaces E_1, E_2, E_3 et E_4. Les plans définissant ces demi-espaces contiennent une arête de l'ouverture et une arête de la boîte englobante de C. Les équations de ces 4 demi-espaces sont les suivantes :

$$E_1 : N_1 \bullet OP + d_1 \geq 0$$
$$E_2 : N_2 \bullet OP + d_2 \geq 0$$
$$E_3 : N_3 \bullet OP + d_3 \geq 0$$
$$E_4 : N_4 \bullet OP + d_4 \geq 0$$

N_1, N_2, N_3 et N_4 correspondent aux normales des plans. Le volume de visibilité étant défini par la boîte englobante de C et par les arêtes de l'ouverture, deux plans sont verticaux (figure 3.21). Nous les associons à E_1 et E_2. D'autre part, les plans définissant E_3 et E_4 contiennent les arêtes horizontales de l'ouverture (figure 3.22).

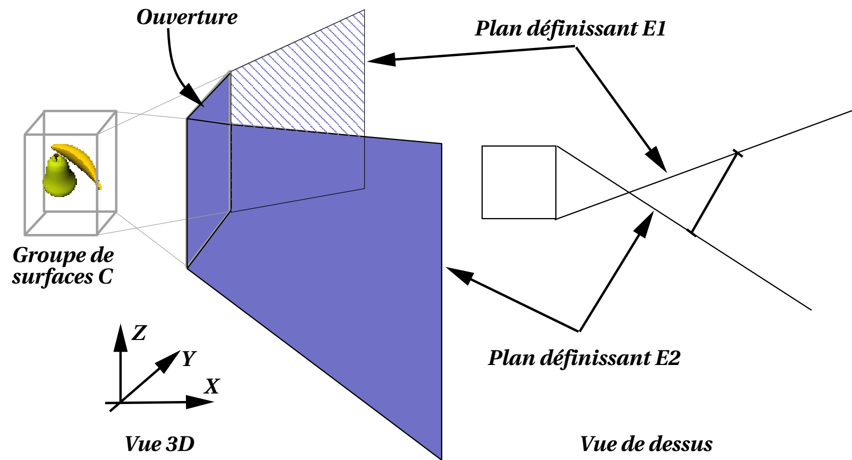


FIG. 3.21 – *Volume de visibilité : plans verticaux (Z est l'axe vertical du repère).*

Par ailleurs, la pyramide doit représenter la zone visible à partir de tout point du groupe. En d'autres termes, l'angle $(\pi - \theta_i)$ de la figure 3.23 doit être maximal pour les 4 demi-espaces définissant la pyramide. Pour chaque demi-espace, cette contrainte est vérifiée en choisissant le sommet S_I de la boîte englobante vérifiant :

$$\vec{V_{S_I}} \bullet \vec{V_o} = \max_{i \in [1..8]} (\vec{V_{S_i}} \bullet \vec{V_o})$$

Notons qu'il n'est pas nécessaire d'effectuer ce test pour tous les sommets de la boîte englobante. En effet pour les demi-espaces ayant des bords verticaux, deux sommets de la boîte englobante situés l'un au-dessus de l'autre génèrent le même demi-espace. La même remarque peut être faite concernant les deux autres demi-espaces. Le plan contenant l'arête horizontale supérieure de l'ouverture doit être défini à l'aide de l'un des 4 sommets inférieurs du volume englobant et

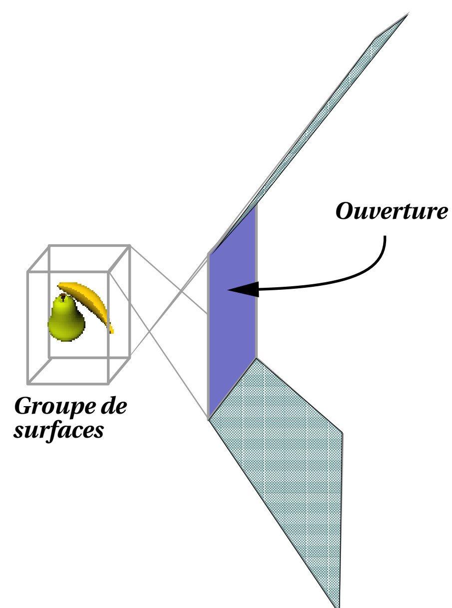


FIG. 3.22 – *Volume de visibilité : plans passant par les arêtes horizontales de l'ouverture.*

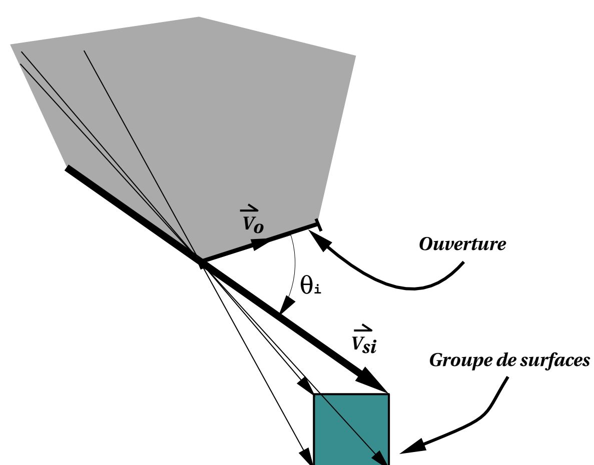


FIG. 3.23 – *Déterminer un sommet pour construire un demi-espace.*

le plan contenant l'arête horizontale inférieure de l'ouverture doit être défini à l'aide de l'un des 4 sommets supérieurs du volume englobant.

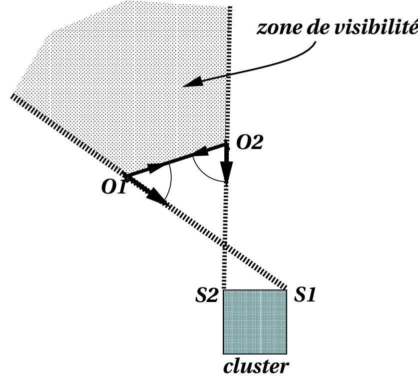


FIG. 3.24 – *Les demi-espaces verticaux.*

Les valeurs N_1 et N_2 des demi-espaces E_1 et E_2 sont obtenues de la manière suivante (figure 3.24) :

Posons $\overrightarrow{V_1} = \frac{\overrightarrow{O_1S_1}}{||\overrightarrow{O_1S_1}||}$ et $\overrightarrow{V_2} = \frac{\overrightarrow{O_2S_2}}{||\overrightarrow{O_2S_2}||}$. Nous avons alors

$$\overrightarrow{N_1} = \pm(V_1.y, V_1.x)$$
$$\overrightarrow{N_2} = \pm(V_2.y, V_2.x)$$

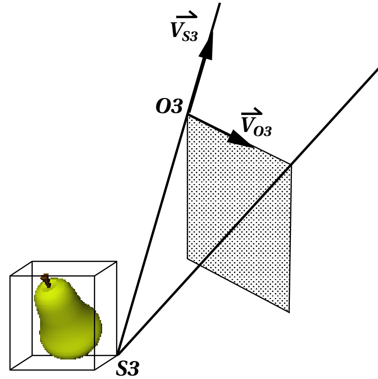


FIG. 3.25 – *Calcul d'un demi-espace à bord non vertical.*

Pour les deux autres demi-espaces, N_3 et N_4 sont calculés différemment puisqu'ils ne sont pas alignés avec les axes du repère (figure 3.25). Soit $\overrightarrow{V_{O3}}$ (resp. $\overrightarrow{V_{O4}}$) le vecteur correspondant à l'arête de l'ouverture pour le demi-plan E_3 (resp. E_4) et $\overrightarrow{V_{S3}}$ (resp. $\overrightarrow{V_{S4}}$) le vecteur correspondant à $\overrightarrow{S_3O_3}$ (resp. $\overrightarrow{S_4O_4}$).

$$\overrightarrow{N_3} = \overrightarrow{V_{O3}} \wedge \overrightarrow{V_{S3}}$$
$$\overrightarrow{N_4} = \overrightarrow{V_{O4}} \wedge \overrightarrow{V_{S4}}$$

Les coefficients d_i de l'équation des plans associés à chacun des demi-espaces sont déterminés à l'aide de S_i de la manière suivante :

$$d_i = -\overrightarrow{N_i} \bullet \overrightarrow{OS_i}$$

Enfin, pour déterminer le sens de chacun des vecteurs N_i, $(i \in [1,4])$, il suffit de vérifier que l'ouverture est contenue dans le demi-espace E_i, $(i \in [1,4])$ en prenant simplement un sommet S de cette ouverture :

$$\overrightarrow{N_i} \bullet \overrightarrow{OS} + d_i \geq 0$$

Si l'inégalité est vérifiée, l'orientation de $\overrightarrow{N_i}$ est correcte, sinon il suffit de poser $\overrightarrow{N_i} = -\overrightarrow{N_i}$.

Pour chaque groupe de chaque cellule, nous construisons un volume de visibilité. Ensuite, l'objectif est de déterminer les groupes contenus dans ces volumes de visibilité construit.

3.4.2 Déterminer les groupes de polygones contenus dans un volume de visibilité

Dans la section précédente, nous avons vu comment calculer le volume de visibilité $V(C,O)$ d'un groupe C à travers une ouverture O. Tous les groupes contenus dans ce volume $V(C,O)$ sont potentiellement visibles de C. Dans cette section, nous décrivons une technique permettant de savoir quels sont les groupes contenus dans un volume défini par 4 inéquations. Un groupe est visible de C si sa boîte englobante est entièrement ou en partie dans $V(C,O)$.

Pour savoir si un point de l'espace se trouve dans le volume de visibilité, il suffit de s'assurer qu'il vérifie le système d'inéquations définissant les 4 demi-espaces de la pyramide. De même, le sommet d'une boîte englobante se trouve à l'intérieur d'un volume de visibilité si et seulement si il vérifie les équations des 4 demi-espaces.

Mais si aucun sommet du volume englobant ne vérifie le système d'inéquations, il est tout de même possible que certains polygones se trouvent à l'intérieur de la pyramide (par exemple l'objet $\mathcal{O}_1$ de la figure 3.26). En revanche, si tous les sommets d'une boîte englobante se trouvent hors d'un demi-espace, le groupe correspondant se trouve entièrement à l'extérieur du volume englobant. Cette affirmation est vérifiée dans notre cas car : (i) les arêtes des boites englobantes sont alignées avec les axes du repère de la scène et (ii) le volume de visibilité est une pyramide. Ainsi, au lieu de tester si la boîte englobante se trouve effectivement à l'intérieur du volume de visibilité, il suffit de vérifier qu'elle ne se trouve pas complètement à l'extérieur de l'un des 4 demi-espaces.

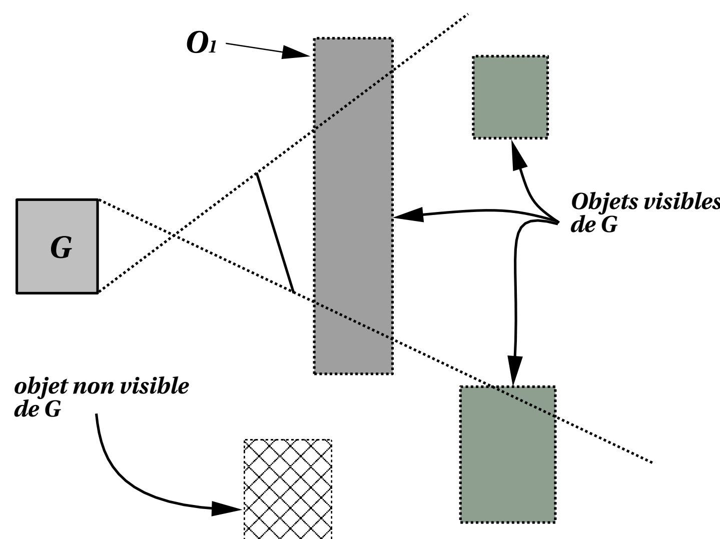


FIG. 3.26 – *Objets visibles de G.*

Dans l'algorithme donné par la figure 3.27, la fonction $GroupeDansPyramide$ retourne toujours $VRAI$ (i.e. le groupe est contenu dans le volume de visibilité) sauf lorsque tous les sommets du groupe se trouvent hors de l'un des 4 demi-espaces.

```
typedef struct {
        /* coefficients des 4 inéquations correspondants */
        /* aux demi-espaces */
        Reel A[4], B[4], C[4], D[4];
} Pyramide;

Booleen Groupe_Dans_Pyramide( Groupe C, Pyramide Vis ) {
        Booleen Reponse = VRAI;
        Booleen reponse_intermediaire;
        Boite_Englobante B;

        B = Boite_Englobante( C );

        /* Pour chaque demi-espace */
        pour i de 1 à 4 faire {

                /* on regarde si les 8 sommets sont hors du demi-espace */
                reponse_intermediaire = FAUX;
                pour Chaque sommet (X, Y, Z) de B, faire {

                        si (A[i] * X + B[i] * Y + C[i] * Z + D[i] ≥ 0) alors {
                                reponse_intermediaire = VRAI;
                        }
                }

                si reponse_intermediaire = FAUX alors
                        reponse = FAUX;
        }

        retourner reponse;
}
```

FIG. 3.27 – *Test d'appartenance d'un groupe à un volume de visibilité.*

3.4.3 Propagation de la recherche

Pour déterminer les groupes visibles à partir d'un groupe C à travers une ouverture, nous construisons un volume de visibilité et lorsqu'une ouverture se trouve dans ce volume de visibilité, les groupes situés de l'autre côté de cette ouverture sont également visibles de C. Pour prendre en compte ces groupes, nous créons un nouveau volume de visibilité. Cette étape de propagation des volumes de visibilité nous permet de déterminer avec précision tous les groupes visibles à partir de C (figure 3.28).

Lorsqu'une ouverture O' appartient à un volume de visibilité V, un nouveau volume de visibilité V' est construit à partir de O' et du groupe. Cependant, selon la position de O' dans le volume de visibilité V, plusieurs cas peuvent se présenter. Si O' est complètement visible du groupe G

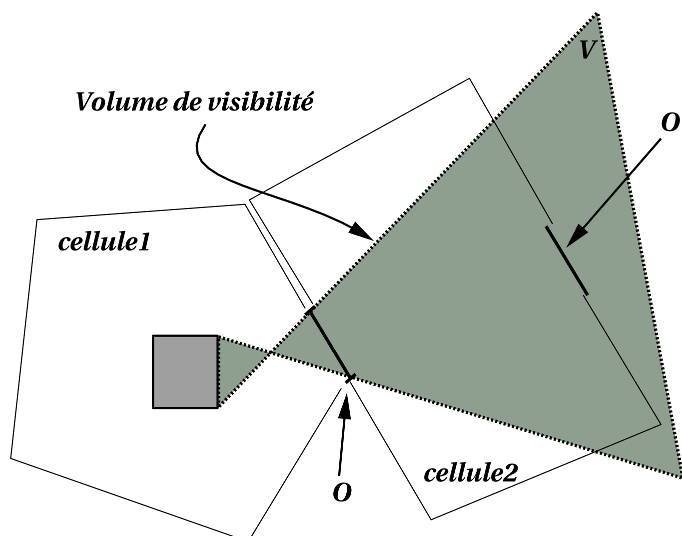


FIG. 3.28 – *Détection d'une ouverture dans un volume de visibilité.*

(figure 3.29), alors V' est correct.

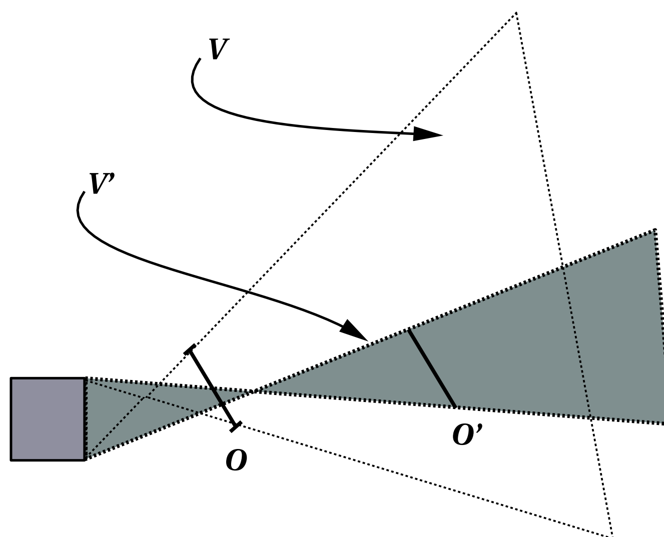


FIG. 3.29 – *Modification du volume de visibilité : cas simple.*

En revanche, si l'ouverture n'est pas complètement visible du groupe C (figure 3.30), le nouveau volume de visibilité est incorrect et doit donc être rectifié (figures 3.31 et 3.32)

Détection du cas particulier

Le nouveau volume de visibilié étant construit, nous vérifions qu'il est correct en effectuant des tests à l'aide des ouvertures intermédiaires. Si les ouvertures intermédiaires sont situées hors du volume V' (figure 3.31), le volume de visibilité V' est erroné.

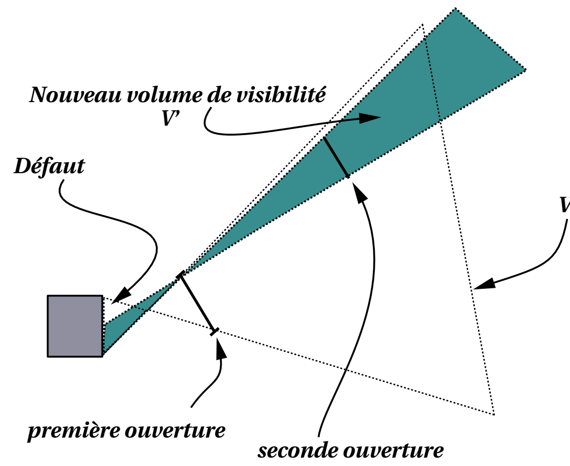


FIG. 3.30 – *Modification du volume de visibilité : cas particulier.*

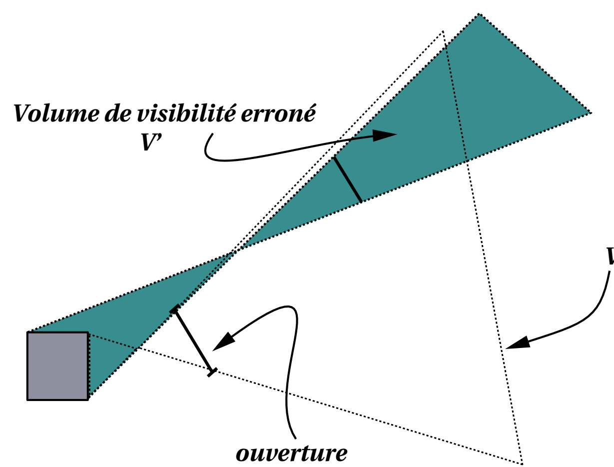


FIG. 3.31 – *Détection du cas particulier.*

Modification du volume de visibilité

Si le nouveau volume de visibilité est incorrect, certains demi-espaces doivent êtres réajustés en fonction des ouvertures intermédiaires. Ainsi, les ouvertures sont reparcourues de la première à la dernière et le volume de visibilité est réajusté à chaque ouverture.

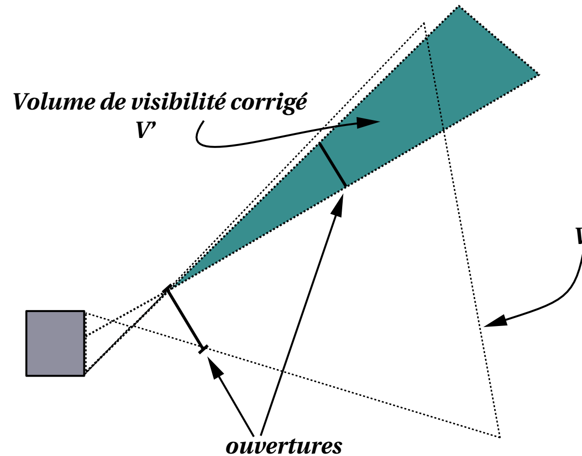


FIG. 3.32 – *Correction du cas particulier.*

Dans la figure 3.32 le volume est réajusté en fonction de la première ouverture.

Ces calculs permettent de déterminer des volumes correspondant aux zones de l'espace visible d'un groupe donné. Tous les objets contenus dans ces volumes sont visibles du groupe. Puis la recherche est propagée. Les ouvertures contenues dans ce nouveau volume sont recherchées. Si il y en a une, de nouveaux volumes de visibilité sont créés, etc.

3.5 Mise en œuvre

L'algorithme décrit dans ce chapitre permet de regrouper les polygones d'un environnement complexe et de déterminer les relations de visibilité entre les groupes obtenus. Le programme que nous avons mis en oeuvre permet à un utilisateur de visualiser les résultats du regroupement et de calculs de vsibilité grâce à une interface graphique basée sur les bibliothèques Motif et OpenGl. Dans cette section, nous décrivons notre mise en œuvre et quelques optimisations apportées dans le but d'accélérer le regroupement de polygones.

3.5.1 Description technique

Notre méthode de regroupement de polygones a été implémentée en langage C++, avec OpenGl pour la visualisation et Motif pour l'interface homme-machine.

Notre programme peut être compilé et exécuté sur diverses machines : Sun, Silicon Graphics ou PC (avec Linux). En effet, les bibliothèques Motif et OpenGl sont des standards présents sur tout type de plateforme.

Une barre de menu permet de :

- charger un environnement architectural complexe ;

– créer des groupes de polygones ;

– déterminer les relations de visibilité entre les groupes de polygones ;

– sauvegarder des groupes de polygones et leurs relations de visibilité.

D'autre part, une fenêtre OpenGl permet de visualiser en temps réel les groupes et les relations de visibilité obtenues.

Cette application utilise les deux fichiers générés par notre programme de structuration d'environnements complexes. Le premier contient un graphe de visibilité comportant pour chaque cellule :

– la liste des cellules adjacentes ;

– la liste des cellules visibles ;

– la liste des ouvertures ;

– le nombre de polygones de la cellule ;

– la position de la cellule dans le fichier contenant la description géométrique de la scène.

Le second fichier contient une liste de cellules séparées par des accolades. Chaque cellule est une liste de polygones convexes à 3 ou 4 sommets.

Pour un graphe de visibilité donné, le programme charge en mémoire les polygones de toutes les cellules de l'environnement et la liste de leurs ouvertures.

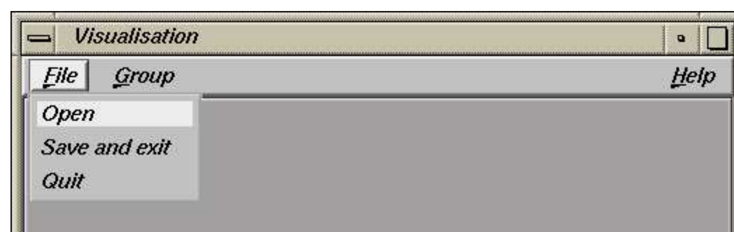


FIG. 3.33 – *Menu fichier : chargement d'une scène, création des cellules et sortie ou sortie du programme.*

La figure 3.33 illustre le menu de chargement et de sauvegarde. Dans ce menu, l'option de sauvegarde déclenche automatiquement la création des groupes de polygones et des calculs de visibilité.

A l'aide de ce menu, il est également possible de modifier le nombre moyen de polygones contenus dans un groupe (figure 3.33). Ce nombre est initialisé à 50 par l'option *init* (figure 3.35).

D'autre part, en cliquant avec la souris dans la fenêtre OpenGl, il est possible de choisir une cellule. Celle-ci est affichée seule, les groupes sont créés uniquement pour elle et aucun calcul de visibilité n'est effectué. En revanche, si toute la scène est visualisée (sélectionnée), tous les groupes sont créés cellule après cellule et les relations de visibilité sont déterminées.

Ensuite, les groupes de polygones peuvent être visualisés un à un ainsi que leurs relations de visibilité.

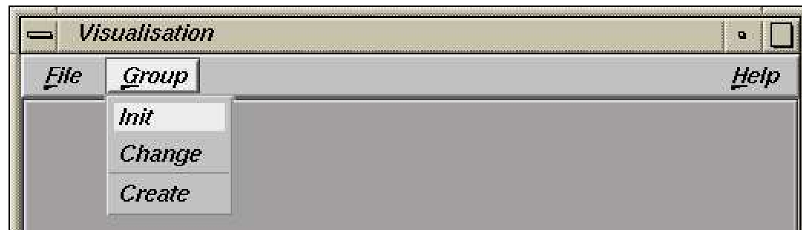


FIG. 3.34 – *Menu groupe : initialisation du nombre moyen de polygones par groupes, modification de ce nombre, création des groupes et des relations de visibilité.*

FIG. 3.35 – *Boîte de dialogue permettant de modifier le nombre moyen de polygones par groupes.*

Cette interface graphique nous permet de contrôler tous les paramètres entrant en jeu dans le calcul du regroupement de polygones. Ainsi, en visualisant les groupes obtenus et en agissant sur le nombre moyen de polygones par groupes, il est possible de modifier la taille des groupes. Cette application permet également de représenter sous forme graphique les relations de visibilité déterminées pour chaque groupe de la scène. En effet, il est possible de visualiser les groupes les uns après les autres à l'aide du clavier (leur boîte englobante est rouge) et leurs groupes visibles (boîtes englobantes vertes).

Ce programme génère deux fichiers. Le premier contient un graphe exprimant les relations de visibilité entre tous les groupes de l'environnement. Le second fichier comporte la description des polygones contenus dans chaque groupe. Ces deux fichiers ont le même format que ceux issus du programme de structuration d'environnements complexes. Ainsi, les calculs de simulation d'éclairage peuvent être effectués indifféremment avec le même programme pour les cellules de l'environnement (avec une machine ayant une mémoire importante) ou pour les groupes de polygones (et pour n'importe quel type de machine).

3.5.2 Optimisations

Nous avons utilisé ces algorithmes pour déterminer les groupes de plusieurs environnements architecturaux complexes. Nous avons apporté quelques modifications permettant d'affiner encore davantage les calculs de visibilité entre les groupes :

– lorsqu'un groupe contient un polygone allongé, la zone de visibilité est très importante. Pour avoir des volumes englobants plus fins, nous avons choisi d'isoler les polygones dont la longueur est supérieure à 1m50 (par exemple les murs) ;

– d'autre part, lorsqu'un groupe ne contient qu'un seul polygone, il est possible de réduire les calculs de visibilité en tenant compte de sa normale. En effet, un polygone dont la

normale n'est pas orientée vers une ouverture ne peut pas voir de groupe à travers cette ouverture.

3.6 Résultats

La figure 3.36 représente les groupes d'une cellule obtenus avec notre programme. Sur cette figure, nous avons fait varier le nombre moyen de polygones par groupe.

Notons cependant que dans notre mise en œuvre, ce nombre moyen de polygones par groupe est le même pour toutes les cellules de l'environnement. Il est cependant possible de changer ce paramètre pour chaque cellule et laisser intervenir l'utilisateur de manière plus importante.

Les temps de calcul sont relativement faibles puisque pour environ 10 000 polygones, la création des 2 158 groupes de polygones est achevée après 27 secondes de calcul sur une machine onyx de Silicon Graphics (tableau 3.1). D'autre part, seulement 1 minute et 15 secondes sont nécessaires pour effectuer tous les calculs de visibilité.

Nombre de polygones de la scène	environ 10.000
Nombre de groupes obtenus	2158
Nombre moyen de groupes visibles d'un groupe	70
Nombre maximal de groupes visibles	171
Nombre minimal de groupes visibles	23
Temps de création des groupes	27secs
Temps de calcul de visibilité	1mn15secs

TAB. 3.1 – *Tableau récapitulatif des résultats obtenus.*

la figure 3.38 illustre les relations de visibilité pour un groupe de polygones. Sur cette figure, tous les objets visibles d'un groupe de polygones G donné sont représentés. Le groupe G est représenté par sa boîte englobante (en rouge) et les groupes visibles sont en vert.

3.7 Discussion

Dans ce chapitre nous avons présenté une nouvelle technique de regroupement de polygones basée sur un algorithme de classification appelé *les nuées dynamiques*. Son intérêt majeur est qu'elle ne nécessite pas de construire une hiérarchie de volumes englobants, contrairement à toutes les autres méthodes de regroupement de polygones.

Concernant les calculs de visibilité, notre approche relève de la géométrie algorithmique mais nous sommes parvenus à éviter l'utilisation de tout un arsenal d'outils mathématiques comme le fait Teller par exemple avec les coordonnées de Plücker. Ceci est dû à notre programme de structuration qui fournit des éléments topologiques tels que des pièces, couloirs, portes, murs, etc. Il en résulte des temps de calcul de visibilité faibles, même pour des environnements très complexes.

Ce programme très rapide et facile à mettre en œuvre constitue donc une alternative à la création de groupes de polygones avec hiérarchie de volumes englobants.

Les calculs donnent lieu à deux fichiers :

- un graphe exprimant les relations de visibilité entre les groupes de l'environnement ;
- un fichier contenant la liste des polygones de chaque groupe.

Le format de ces deux fichiers est strictement identique à celui des fichiers générés par le programme de structuration de la scène en cellules. Grâce à ce format de fichier unique, nous pourrons indifféremment appliquer les calculs de simulation d'éclairage pour des scènes complexes structurées en cellules ou en groupes.

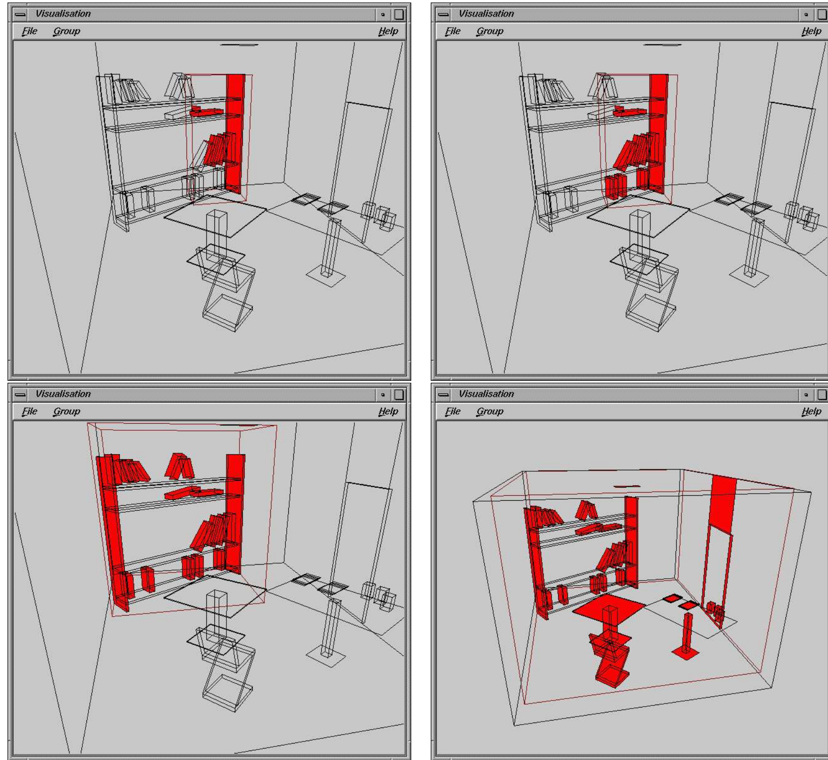


FIG. 3.36 – *Groupes de polygones créés avec un nombre moyen de polygones par groupe égal à : 5 (haut et gauche), 20 (haut et droit), 60 (bas et gauche), 150 (bas et droit).*

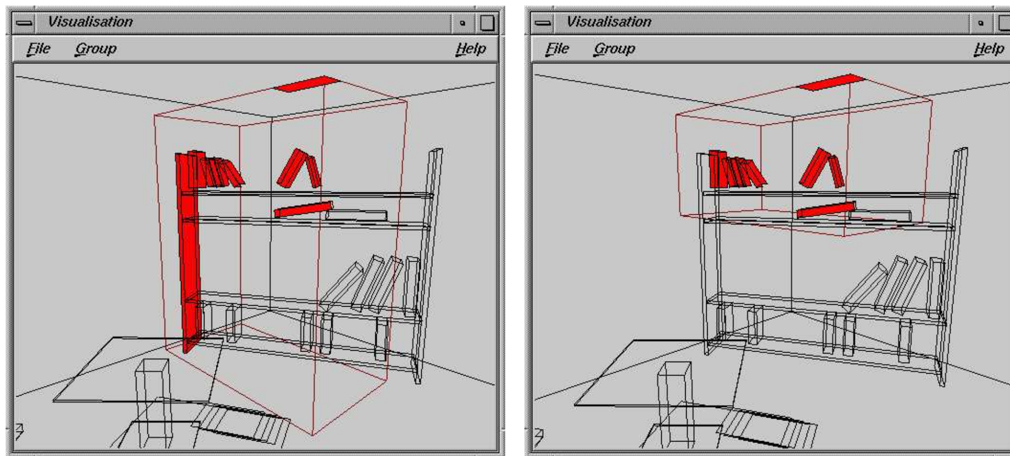


FIG. 3.37 – *Calcul des groupes : image de droite avec la distance au plus proche sommet, image de gauche avec la distance au barycentre du polygone.*

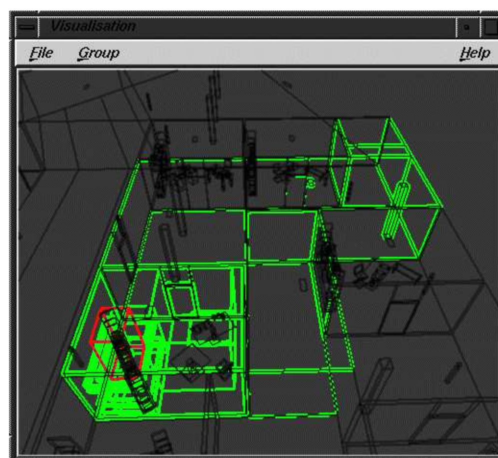


FIG. 3.38 – *Objets visibles (en vert) d'un groupe de polygones (en rouge).*

Chapitre 4

Modélisation d'environnements architecturaux complexes

4.1 Problématique

Pour tester nos différents programmes de structuration, de regroupement de surfaces et de simulation d'éclairage, un jeu de plusieurs scènes complexes était nécessaire. Cependant, les modèles géométriques correspondant à des bâtiments de plusieurs étages sont rarement disponibles lorsqu'ils existent. Nous avons donc dû les modéliser. Or les modeleurs classiques (Power Animator d'Alias Wavefront ou AutoCad par exemple) bien que très performants ne sont pas conçus pour générer des environnements très complexes et sont par conséquent délicats à utiliser pour notre application. En effet, ces modeleurs permettent de créer des environnements de manière très précise, mais toutes les surfaces sont stockées en mémoire et affichées au cours de la modélisation. Cela devient très rapidement problématique dès que l'on dépasse une dizaine de pièces. L'affichage devient très lent et chaque opération demande une grande patience. D'autre part, les fichiers générés contiennent toutes les surfaces de la scène ainsi que leurs propriétés photométriques. Ils nécessitent par conséquent plusieurs dizaines de Mega octets d'espace disque.

Pour résoudre ces problèmes, nous avons conçu un modeleur interactif utilisant la bibliothèque graphique OpenGl et l'interface homme-machine Motif. Ce modeleur, spécialisé dans la construction d'environnements architecturaux complexes utilise une bibliothèque d'objets que nous avons constituée à l'aide du modeleur d'Alias Wavefront. Notre objectif est de réduire le nombre de polygones à afficher à chaque instant. Pour cela, notre programme est composé de deux modules. Le premier permet à l'utilisateur de construire une pièce et de l'ajouter à une bibliothèque. Les pièces sont affichées selon trois points de vue différents afin de pouvoir placer précisément les objets. Dans le second module, l'assemblage des pièces est réalisé par étage, la visualisation étant effectuée uniquement selon une vue de dessus. Au lieu de modéliser un bâtiment complet puis de le découper entièrement ensuite, le logiciel guide l'utilisateur afin que la structuration soit automatiquement effectuée. Enfin, les fichiers produits par ce modeleur ne contiennent que très peu d'informations :

- la description des murs de chaque pièce : les murs verticaux sont représentés par des segments dans le plan horizontal (Ox, Oy) ;
- la liste des objets contenus dans chaque pièce : les objets sont définis par des fichiers dans un répertoire particulier (la bibliothèque d'objets). Le fichier généré par notre modeleur contient pour chaque objet le nom de son fichier ainsi qu'un vecteur de translation et un angle de rotation autour de l'axe vertical de la scène (Oz), correspondant à sa position dans la cellule ;
- la liste des sols et des plafonds ainsi que leurs propriétés géométriques pour chaque cellule ;

- la liste des ouvertures (correspondant à des polygones transparents) : ces ouvertures sont utilisées pour créer automatiquement le graphe de visibilité ;

De cette manière, le stockage sur disque d'un bâtiment de plus de 600 pièces ne nécessite que quelques dizaines de Kilo octets. Pour regénérer les fichiers complets, un second programme interprète les fichiers générés par le modeleur. Ce programme est en fait un compilateur qui en plus détermine le graphe d'adjacence et calcule les relations de visibilité entre les cellules de l'environnement.

Le plan de ce chapitre est le suivant. La première partie donne les principes de notre modeleur. Dans la seconde, nous décrivons nos algorithmes avant de présenter quelques résultats. Enfin, nous concluons par une brève discussion.

4.2 Notre approche

Notre modeleur est composé de deux programmes. Le premier est interactif et permet à l'utilisateur de modéliser des environnements architecturaux complexes. Le second programme décompresse les fichiers générés par le premier dans une phase de compilation. Cette compilation crée un fichier contenant la description géométrique et photométrique précise de la scène et un fichier correspondant au graphe de visibilité.

4.2.1 Modélisation et interface graphique

Les fonctionnalités de notre modeleur sont les suivantes :

- Spécification des propriétés photométriques du sol, des murs et du plafond de chaque pièce ;

- Ajout et positionnement interactif des objets dans les pièces ;

- Mécanisme de placement automatique des pièces afin de faire correspondre précisément les ouvertures des pièces.

L'interface homme machine utilisée est Motif car elle est portable et offre à l'utilisateur un contrôle total des paramètres de l'interface. Par ailleurs, la bibliothèque graphique utilisée pour ce modeleur est OpenGl car elle est un standard reconnu et d'autre part elle est également portable sur de très nombreuses machines telles que SUN, Silicon Graphics, ou encore PC.

Dans ce modeleur, une notion de hiérarchie est introduite de manière automatique, afin de guider l'utilisateur vers une modélisation plus modulaire. Cette hiérarchie est constituée des éléments suivants :

- Les objets ;

- Les cellules ;

- Les pièces ;

- Les étages.

Dans cette hiérarchie, une pièce non convexe peut être constituée de plusieurs cellules (convexes) et les bâtiments sont crées en juxtaposant les pièces (figure 4.1).

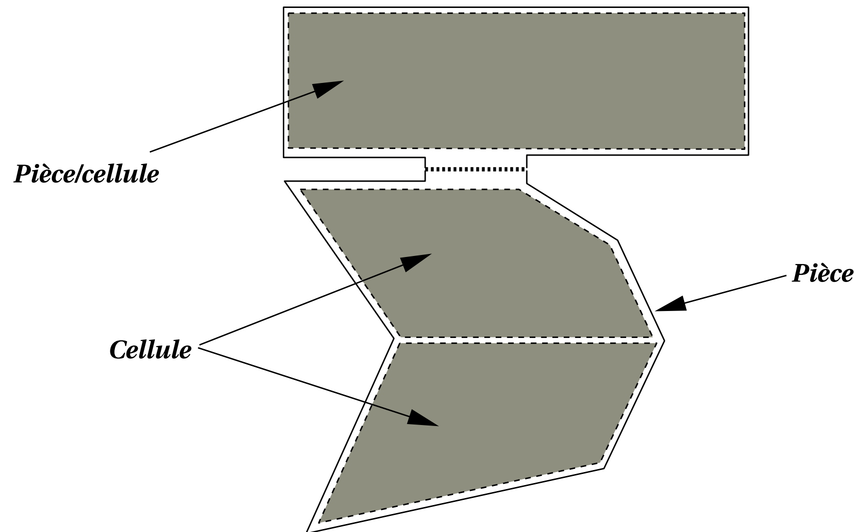


FIG. 4.1 – *Hiérarchie de pièces et de cellules.*

Manipulation d'objets ou de pièces

- *Objets*
  Le placement des objets à l'intérieur d'une pièce est effectué à l'aide de la souris dans le premier module. Dans le fichier créé par le programme, un objet est défini par un nom de fichier ainsi qu'un vecteur de translation et une rotation autour de l'axe vertical (Oz). Il est également possible d'effectuer un changement d'echelle.

- *Pièces et cellules*
  Les murs d'une pièce sont définis en 2D par des segments de droite dessinés par l'utilisateur à l'aide de la souris. L'extrusion est effectuée de manière automatique par le modeleur. Puis comme les objets, les pièces sont placées dans un bâtiment à l'aide de la souris en déterminant une translation et une rotation.

Dans le fichier généré par le modeleur, ces commandes correspondent au texte de la figure 4.2.

BibliothÈque d'objets ou de pièces

Afin de faciliter l'inclusion d'objets dans chacun des modules, une bibliothèque d'objets a été mise en œuvre sous forme de liste. Cette liste est gérée grâce aux fonctionnalités de Motif. Pour éviter de surcharger la mémoire, les objets ajoutés à la liste ne sont réellement chargés en mémoire qu'au moment de leur affichage. Ainsi, la bibliothèque peut contenir un grand nombre d'objets.

De la même manière, une bibliothèque de pièces a été mise en œuvre sous forme de liste. Comme dans le cas de la bibliothèque d'objets, une liste chainée est utilisée. Ici, la liste contient des pièces.

Le magnétisme

Afin de faciliter le placement des pièces les unes par rapport aux autres, un mode de placement magnétique a été mis en œuvre. Dans ce mode, les pièces se positionnent automatiquement de

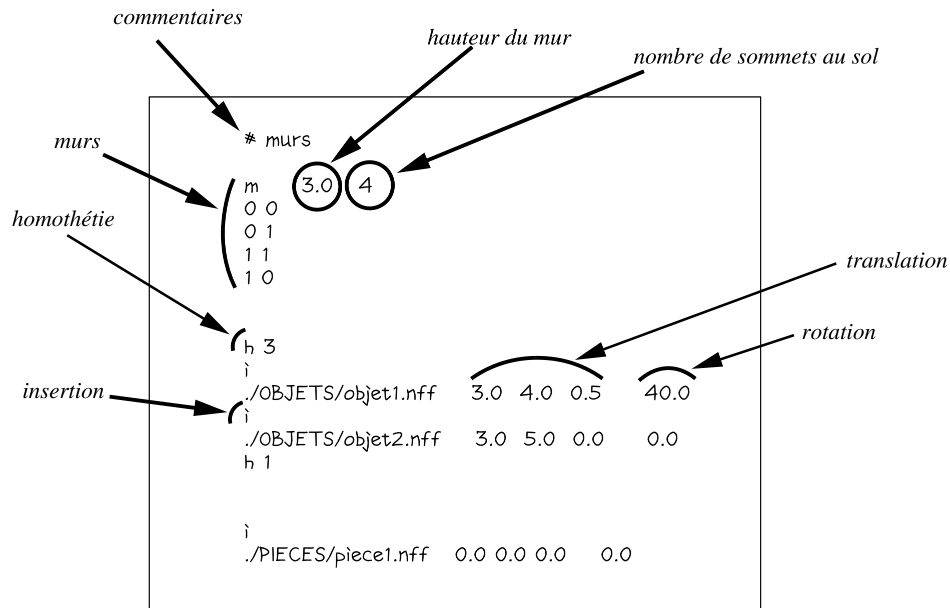


FIG. 4.2 – *Commande de création de murs, inclusion d'objet, homothétie et insertion de pièces dans un bâtiment.*

manière à faire correspondre leurs ouvertures de la manière suivante :

- lorsque l'utilsateur approche une pièce d'une autre et que celle-ci possède une ouverture identique, la pièce approchée subit une rotation puis une translation afin de coller les deux ouvertures ;
- dans le cas où plusieurs choix sont possibles, l'alignement se fait avec la pièce la plus proche ;
- le magnétisme doit être assez rapide pour placer les pièces de manière automatique sans gêner l'utilisateur ;

En conséquence de ces critères :

- la recherche n'est réalisée que sur les pièces de l'étage courant ;
- les recherches d'ouvertures quasi-parallèles est effectuée sur les pièces les plus proches et pour des ouvertures de même taille ;
- le critère significatif à minimiser est l'angle inter-ouvertures au cas où les ouvertures sont assez proches ;

4.2.2 Compilation (ou décompression) des fichiers

Les fichiers générés par le modeleur interactif ne contiennent qu'un minimum d'informations. Ceci permet de réduire leur taille. Or pour effectuer les calculs de visualisation ou de simulation d'éclairage, toutes les données doivent être explicitement connues. Pour obtenir un fichier contenant précisément toutes les informations géométriques et photométriques de l'environnement, nous effectuons une compilation du fichier issu du modeleur. Ce fichier contient des desciptions données par la figure 4.3.

```
##################################
# 5 x 4, ouverture au nord

f Citron 1.0 0.0 0.0

m
3 6

3 4
5 4
5 0.1
0.1 0.1
0.1 4
2 4

# chambranle
i
OBJETS.NFF/mich.nff 2 4 0 0
i
OBJETS.NFF/oh.nff 2 4 0 0

# sol
f Email_bleu 1.0 0.0 0.0
p 4
0.1 0.1 0
5 0.1 0
5 4 0
0.1 4 0

# OBJETS
h 15
i
OBJETS.NFF/Etagere3.nff 0.1 0.5 0 90
h 1
i
OBJETS.NFF/Lampe0.nff 2.5 2 2.8 0
i
OBJETS.NFF/Chaise2.nff 4.2 0.8 0 160
h 10
i
OBJETS.NFF/Table8.nff 3 1.5 0 20
h 1

# FIN de la pièce
#################################

w
```

FIG. 4.3 – *Fichier de description d'une cellule.*

Le compilateur lit simplement les informations et les transforme point par point. Chaque ligne correspond soit à une commande, soit à des données. Les commandes sont les suivantes :

- "i" : insertion de fichier ;
- "m" : description de murs ;
- "f" : description photométrique d'un groupe de surfaces ;
- "p" : description géométrique d'une surface ;
- "h" : facteur d'homothétie ;
- "#" : commentaire ;
- "w" : fin de la cellule.

Durant cette étape de compilation, la structuration de la scène est effectuée automatiquement car les cellules et les ouvertures sont déjà décrites explicitement dans le fichier de description de l'environnement. Par conséquent, les calculs de visibilité peuvent être effectués et le graphe de visibilité est construit automatiquement, sans aucune intervention de l'utilisateur.

Avec ce compilateur, il est également possible de construire un graphe de visibilité plus fin, correspondant aux *clusters* que nous avons décrits dans le chapitre 3. En effet, à l'aide de la commande "i", chaque objet est clairement identifié. Cette fonctionalité n'a pas encore été mise en œuvre, mais elle n'implique que quelques modifications mineures dans notre programme de compilation.

4.3 Mise en œuvre

Les différents modules de notre interface graphique sont décrites par les figures 4.4 et 4.6. Pour chaque module, l'utilisateur dispose de plusieurs opérations : ajout, suppression, translation, rotation, etc. Ces opérations sont accessibles via une boîte de dialogue contenant plusieurs icônes.

Comme nous l'avons déjà dit, toute cette interface a été développée à l'aide des bibliothèques OpenGL et Motif. OpenGL est utilisée pour l'affichage de la scène et Motif pour les barres de menus.

4.4 Résultats

Les fichiers générés après la compilation sont au format NFF [27]. Ils seront directement pris en entrée de nos algorithmes de simulation d'éclairage séquentielle ou parallèle.

Le tableau 4.1 représente les tailles des fichiers générés par notre modeleur.

Selon le type de bâtiment, le gain apporté par notre stockage de fichier peut être plus ou moins élevé. Cependant, plus le bâtiment est important, plus les redondances d'objets sont nombreuses et le gain de place sera important.

Par ailleurs, les bâtiments construits à l'aide de ce modeleur peuvent contenir plusieurs milliers de pièces sans que l'utilisateur ne soit gêné au moment de la modélisation.

La figure 4.7 correspond à la visualisation en fil de fer d'un bâtiment construit à l'aide de notre modeleur.

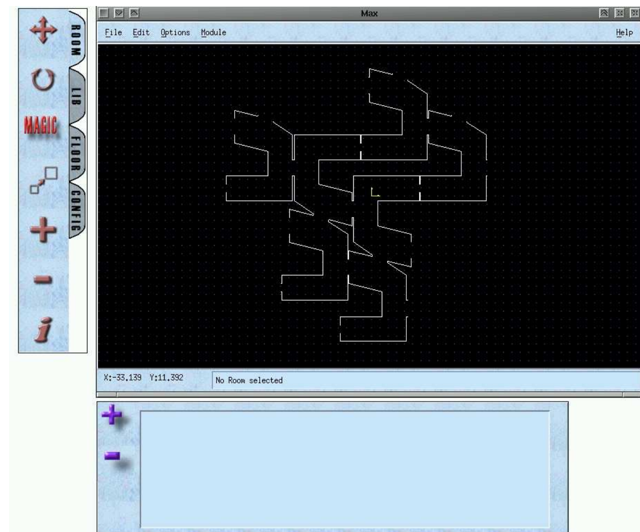


FIG. 4.4 – *Interface graphique de notre modeleur : module bâtiment.*

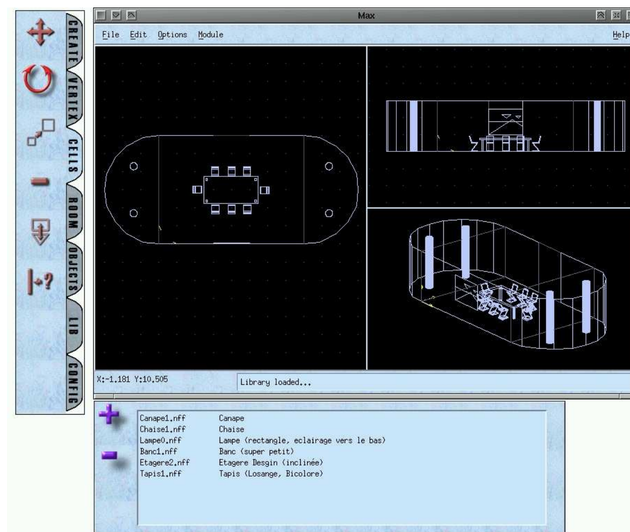


FIG. 4.5 – *Interface graphique de notre modeleur : module pièce.*

	mod	nff	mod + gzip	nff + gzip
Petit bâtiment non meublé (env. 10 pièces)	145 Ko	202 Ko	4.7 Ko	24 Ko
Grand bâtiment meublé (beaucoup de redondances)	62 Ko	15 Mo	8.7 Ko	545 Ko

TAB. 4.1 – *Utilisation de l'espace disque selon les formats de fichiers.*

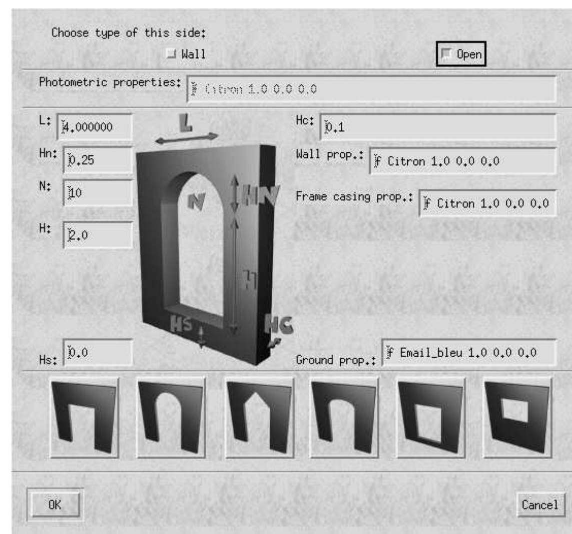


FIG. 4.6 – *La boîte de dialogue permettant de créer une ouverture.*

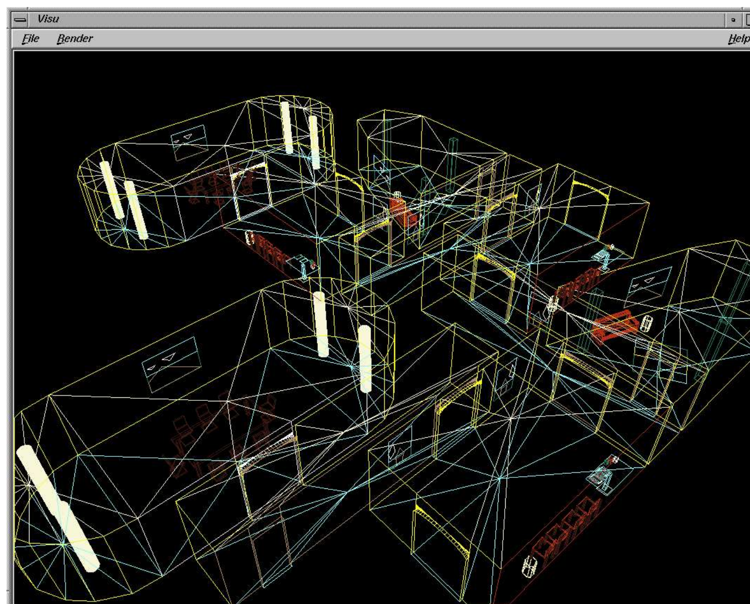


FIG. 4.7 – *Visualisation d'un bâtiment en fil de fer.*

4.5 Conclusion du chapitre

Dans ce chapitre, nous avons décrit un modeleur d'environnements architecturaux complexes permettant de construire des bâtiments meublés de plusieurs étages. Afin qu'il soit convivial, nous avons conçu une interface graphique, développée à l'aide des bibliothèques Motif et OpenGL. Par ailleurs, les fichiers issus de notre modeleur ne prennent que très peu de place en mémoire car ils contiennent seulement quelques informations concernant la géométrie de la scène. Pour obtenir une description géométrique et photométrique complète, nous avons mis en œuvre un compilateur dont l'objectif est de *décompresser* les données. Enfin, la contribution principale de ce travail concerne la structuration semi-automatique de l'environnement. En effet, celle-ci est automatiquement guidée par l'utilisateur qui définit lui-même les cellules et les ouvertures de manière naturelle au moment de la construction du bâtiment. Le compilateur se charge de déterminer automatiquement les relations de visibilité entre les cellules de l'environnement et de créer une base de données adéquate pour les calculs de radiosité ou de visualisation. Ce modeleur est donc un outil interactif permettant de générer des environnements complexes de manière plus conviviale qu'avec les modeleurs classiques car seule une petite partie de l'environnement réside en mémoire à chaque instant et l'affichage est par conséquent beaucoup plus rapide.

Chapitre 5

Simulation d'éclairage séquentielle

5.1 Problématique

Dans les chapitres précédents, nous avons vu différentes techniques de structuration d'environnements architecturaux complexes (i.e. bâtiments). Cette structuration peut être effectuée soit dès la modélisation du bâtiment, soit au cours d'une étape de précalcul. Elle consiste à :

1. découper le bâtiment en groupes de polygones correspondant soit à des régions de l'espace (les cellules), soit à des objets (les *groupes*) ;
2. construire un graphe (appelé *graphe de visibilité*) exprimant les relations de visibilité entre les groupes de polygones.

A partir de cela, les calculs de radiosité peuvent être effectués en ne chargeant en mémoire qu'une petite partie des données. En effet, l'énergie lumineuse émise par une surface à l'intérieur d'un groupe de polygones G_p peut atteindre les surfaces de G_p et d'autres surfaces appartenant à des sous-scènes visibles de G_p. Par conséquent, pour effectuer ce calcul, seul G_p et ses groupes de polygones visibles sont chargés en mémoire. Ainsi, la plupart des données restent sur le disque dur de la machine.

Cependant, cette méthode implique de nombreux accès au disque ralentissant notablement les calculs, d'autant plus que chaque sous-scène peut être lue (ou écrite) sur le disque plusieurs fois du fait du caractère itératif de la méthode de radiosité. Les calculs doivent donc être ordonnés efficacement afin de réduire au maximum les accès-disque (ou Entrées/Sorties). Teller et. al. [1] ont déjà proposé plusieurs stratégies basées sur la localité des données en mémoire. Cependant, ces stratégies nous semblent insuffisantes car elle ne prennent pas en compte les coûts d'accès disque. Or ceux-ci sont prépondérants pour ce type de calcul. C'est pourquoi nous proposons de nouvelles stratégies d'ordonnancement issues d'algorithmes de la théorie des graphes. Notre objectif est d'estimer le plus précisément possible les coûts d'Entrées/Sorties à court, moyen et long terme. Cette étude peut être considérée comme un travail complémentaire à celui de Teller et. al. mais outre les nouvelles stratégies que nous avons mises en oeuvre, notre méthode offre également la possibilité d'effectuer indifféremment des calculs de radiosité pour des environnements dont les sous-scènes correspondent à des cellules ou à des *groupes de polygones*. En effet, les fichiers produits par notre programme de structuration d'environnements ont le même format que ceux produits par notre programme de regroupement de surfaces. Par ailleurs, notre méthode est générique et fonctionne aussi bien avec la radiosité progressive qu'avec les techniques de Jacobi ou Gauss-Seidel.

Le plan de ce chapitre est le suivant. Un résumé des travaux existants est présenté, suivi d'une vue d'ensemble de notre algorithme. Puis nos stratégies d'ordonnancement sont décrites de manière détaillée. Enfin nous discutons les résultats de nos travaux et concluons ce chapitre.

5.2 Travaux antérieurs

Peu de travaux abordent le problème de la simulation d'éclairage séquentielle dans des environnements complexes. A notre connaissance, Teller et al. [1] sont les seuls auteurs ayant proposé un algorithme permettant de résoudre ce problème. Leur objectif est d'utiliser la méthode de radiosité pour effectuer les calculs d'illumination globale. A cette fin, la scène est découpée en régions $3D$ (appelées *cellules*) selon la méthode BSP. Chaque cellule contient un ensemble de *groupes* pour lesquels des informations de visibilité très précises sont précalculées. Pour chaque *groupe* C, les informations suivantes sont stockées sur le disque afin d'être réutilisées au cours des calculs de radiosité :

- la liste des *groupes* visibles $\{C_i\}$;
- la liste des liens de C ;
- la liste des polygones situés entre C et chaque *groupe* C_i ;
- etc.

L'éclairement des surfaces d'un *groupe* R (récepteur) est calculé à l'aide de la méthode itérative de Jacobi, selon l'algorithme décrit par la figure 5.1.

```
Lire_Groupe(R);
Pour chaque source S Visible de R faire {
        Lire_Groupe(S);
        Lire_Blockers(S,R);
        Installer_Liens(R,S);
        Collecte_Jacobi(R);
        Supprimer(S);
        Supprimer(Blockers);
}

Invoquer_PushPull(R);
Supprimer(R);
```

FIG. 5.1 – *Simulation d'éclairage de Teller dans des environnements complexes.*

Le choix du *groupe* R (*Lire_groupe*) est guidé par des heuristiques visant à réduire les accès-disque. Ces heuristiques sont les suivantes :

- ordre **aléatoire** : le *groupe* R est choisi de manière aléatoire ;
- ordre **de modélisation** : les *groupes* sont choisis dans l'ordre de leur modélisation pour collecter l'énergie ;
- ordre **source** : cette méthode choisit l'objet ayant le plus souvent servi de source ;
- ordre **cellule** : choisit les objets en effectuant un parcours de l'arbre BSP afin de conserver une localité géométrique des données.

D'après les résultats donnés par Teller et. al., la quatrième technique (utilisant l'arbre BSP) est la plus performante.

Plusieurs approches parallèles traitent également le problème de la simulation d'éclairage dans des environnements complexes. Nous les décrirons dans le chapitre suivant, qui est consacré à ce sujet.

5.3 Motivations

Les résultats obtenus par Teller et. al. [1] montrent bien que les temps de calcul sont très dépendant des accès disque (Entrées/Sorties) et de l'ordre dans lequel les *groupes* sont choisis pour collecter l'énergie. Cependant, les stratégies proposées ne sont pas directement liées aux coûts d'Entrées/Sorties. Nous souhaitons donc étudier plus précisément l'influence de ces accès disque sur les temps de calcul de la simulation d'éclairage afin de déterminer une stratégie d'ordonnancement réellement efficace. Pour atteindre ce but, nous avons mis en œuvre différentes stratégies d'ordonnancement, estimant les coûts des Entrées/Sorties à court moyen et long terme.

5.4 Principe de notre algorithme

Les environnements traités par notre algorithme sont des bâtiments meublés définis par des polygones plans et convexes à 3 ou 4 sommets. Dans une étape de précalcul, l'environnement est découpé en régions appelées cellules. Puis dans chaque cellule les polygones sont regroupées en groupes. Enfin, des calculs de visibilité génèrent un graphe dans lequel un nœud représente un *groupe* et un arc entre deux nœuds indique que les *groupes* correspondants sont mutuellement visibles.

Note : tout au long de ce chapitre, nous parlerons de *groupe* pour des raisons de simplicité de langage. Néanmoins, un *groupe* correspond à un ensemble de polygones ou une cellule. En effet, il est possible d'utiliser notre programme de simulation d'éclairage indifféremment pour des *groupes* ou des cellules car la structuration et le regroupement de surfaces produisent des fichiers ayant le même format. Par ailleurs, dans notre implémentation, nous avons choisi la méthode de radiosité progressive (*shootin*) car elle permet de visualiser la scène de manière interactive au cours des calculs. Cependant, il est également possible d'employer les méthodes de jacobi ou gauss-seidel (*gathering*).

Notre programme de simulation d'éclairage est constitué des différents modules représentés par la figure 5.2. Les calculs de radiosité sont effectués en choisissant tour à tour tous les *groupes* de la scène afin qu'ils émettent leur énergie lumineuse. Le *groupe* (appelé *groupe émetteur*) choisi à chaque étape par le module d'ordonnancement est chargé en mémoire ainsi que ses *groupes* visibles, ensuite les calculs de radiosité sont effectués. Afin de réduire les accès-disque, chaque *groupe* est conservé en mémoire le plus longtemps possible et réutilisé plusieurs fois au lieu d'être systématiquement relu sur le disque. Certains *groupes* peuvent donc être présents en mémoire sans intervenir dans les calculs de radiosité courants. Lorsqu'un nouveau *groupe* doit être chargé (car il participe aux calculs de radiosité), certains *groupes* (ne participant pas au calcul) doivent être supprimés de la mémoire pour faire de la place. Pour prévoir précisément les coûts de chargement/déchargement des groupes, nous utilisons le graphe de visibilité et un tableau *Groupes_En_Mémoire[ ]* indiquant pour chaque *groupe* s'il est présent en mémoire et s'il est utilisé pour le calcul de radiosité courant. Les stratégies que nous avons mises en oeuvre permettent de prévoir les coûts de chargement/déchargement des *groupes* à court, moyen ou long terme.

L'espace mémoire de la machine ne peut contenir qu'un nombre réduit de surfaces N_{max}, en

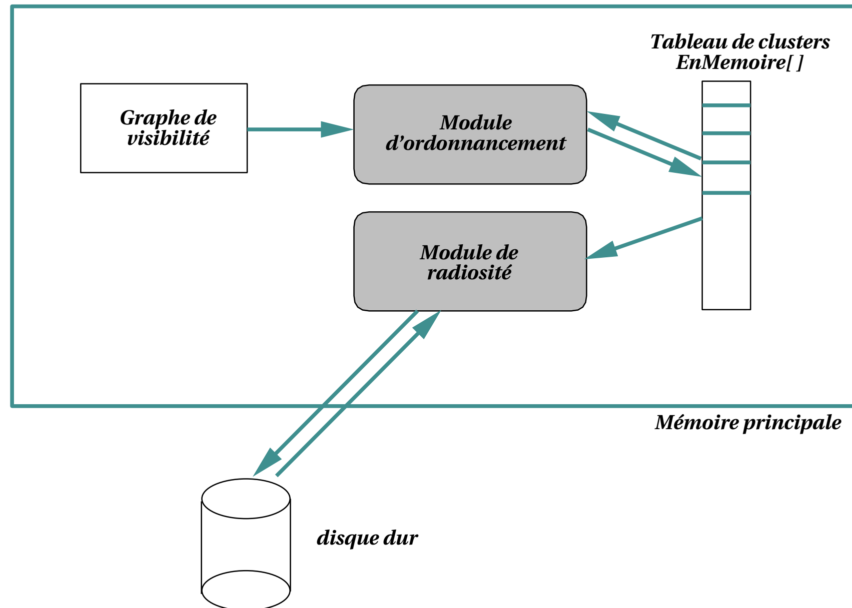


FIG. 5.2 – *Architecture du système.*

pratique fixé par l'utilisateur lui-meme. Notre algorithme de simulation d'éclairage s'adapte automatiquement à ce nombre. En d'autres termes, lorsque l'espace mémoire est plein, un ou plusieurs *groupes* sont rapatriés sur le disque afin de faire place à de nouveaux *groupes*. Par ailleurs, afin de réduire les accès au disque, les *groupes* sont conservés le plus longtemps possible en mémoire et réutilisés plusieurs fois.

Notre algorithme de simulation d'éclairage est donné par la figure 5.3. Au cours d'une itération (ou $ITERATION$ dans cet algorithme), tous les *groupes* sont choisis tour à tour afin d'émettre leur énergie. Au cours de la première itération, seules les surfaces ayant une auto-émittance propre émettent leur énergie, de manière à obtenir des résultats visuels très rapidement.

La fonction $Choisir_Groupe()$ correspond au module d'ordonnancement qui choisit le *groupe* devant émettre son énergie selon l'une de nos stratégies d'ordonnancement. La procédure $Shoot_Groupe()$ effectue le calcul de *shooting* pour toutes les surfaces du *groupe* émetteur C choisi.

Dans la section suivante, nous décrivons en détail nos 7 stratégies d'ordonnancement.

5.5 Ordonnancement

5.5.1 Principe

Tous les *groupes* de la scène sont successivement choisis pour émettre leur énergie. Dans ce contexte, le rôle du module d'ordonnancement est de déterminer, à chaque étape de calcul, le *groupe* impliquant le minimum d'Entrées/Sorties. Le chargement des *groupes* concernés par les calculs de radiosité entraîne le retrait de certains *groupes* (non utilisés) de la mémoire afin de faire place à tous les *groupes* prenant part au calcul. Ces opérations provoquent de nombreux accès-disque, très coûteux en termes de temps de calculs et doivent donc être ordonnées. Pour être réellement efficace, l'algorithme d'ordonnancement doit réduire les accès-disque tout en maintenant une convergence rapide de l'algorithme de radiosité.

Tous les *groupes* doivent donc être choisis tour à tour dans un certain ordre, de manière à réduire les coûts d'accès disque. Or ce problème est équivalent au problème du voyageur de commerce,

```
structure DRAPEAUX_GROUPES {
        booléen En_Memoire;
        booléen Utilisée;
}

global_illumination() {
        DRAPEAUX_GROUPES Groupes_En_Memoire[nombre_de_groupes];
        Integer C; /* identificateur de groupe */
        ITERATION = 0;

        /* GV est le graphe de visibilité */
        GV = Lire_Graphe_Visibilité_Disque();
        Initialise(Groupes_En_Memoire[ ]);

        Tant que non (convergence) ET (ITERATION < Max_Iterations) {
                Supprimer_Les_Marqueurs_De_Tous_Les_Groupes();

                /* Choix du premier groupe émetteur */
                C = Choisir_Groupe(GV,ITERATION,Groupes_En_Memoire);

                /* Choisir_Groupe() modifie Groupes_En_Memoire[ ] */
                Marque_Groupe(C);

                /* La cellule C est marquée pour l'itération courante */
                tant que non(tous les groupes sont marqués) {
                        /* La cellule C émet son énergie */
                        Shoot_Groupe(C,Groupe_En_Memoire);

                        /* Choix d'une cellule qui n'a pas encore émis son énergie */
                        /* pendant cette itération */
                        C = Choisir_Groupe(GV,ITERATION,Groupe_En_Memoire);
                        Marque_Groupe(C);
                }
                ITERATION = ITERATION + 1;
        }
}
```

FIG. 5.3 – *Algorithme général.*

mais plus complexe dans notre cas puisque la base de données varie au cours des calculs (création de nouveaux liens, nouveaux éléments de surface, etc.).

Nos méthodes d'ordonnancement fonctionnent selon l'algorithme décrit par la figure 5.4. Lorsqu'un *groupe* est choisi pour émettre son énergie, il doit se trouver en mémoire ainsi que ses *groupes* visibles. Si l'un ou l'autre de ces *groupes* n'est pas déjà en mémoire, il doit être lu sur le disque, impliquant des accès-disque dont le coût est évalué (par *evaluate_cost*) en termes de polygones ou en termes de *groupes* selon la stratégie d'ordonancement choisie.

```
int Choisir_Groupe(VG,ITERATION,C_En_Memoire) {
        int cout_min, cout;
        cost_min = +infini;

        pour chaque groupe Ci {
                /* Chercher l'ensemble VS des groupes visibles de Ci */
                VS = Chercher_Visibles(Ci,GV);
                cout = Evaluer_Cout(Ci,VS,C_En_Memoire);
                if ( cout < cout_min ) {
                        cout_min = cout;
                        resultat = Ci;
                }
        }
        Mise_A_Jour(GV,Ci,C_En_Memoire);
        retourner resultat;
}
```

FIG. 5.4 – *Algorithme d'ordonancement.*

Mise_a_jour initialise à $VRAI$ les champs *En_Mémoire* et *Utilisé* de la structure de données $CELL_FLAGS$ (voir figure 5.3) associée aux *groupes* de VS (voir figure 5.4). Cette procédure initialise également a $FAUX$ le champ *Utilisé* associé aux *groupes* non visibles du *groupe* émetteur choisi et résidant en mémoire.

Dans notre implémentation, l'utilisateur peut fixer une valeur M_{limit} correspondant au nombre maximal de polygones pouvant être stockés en mémoire. Nous avons préféré choisir un nombre de surfaces plutôt qu'une taille mémoire pour deux raisons. La première est que le nombre de mailles et de liens varie au cours du temps et il est difficile de prévoir la taille mémoire qui sera requise pour un *groupe*. La seconde provient du fait que le nombre de liens et de mailles a une complexité en $O(n \log n)$, n étant le nombre de surfaces de départ. Nous pouvons donc estimer à priori la taille M_{limit} de manière plus sûre. Pour les mêmes raisons, certaines stratégies d'ordonnancement évaluent les coûts correspondant aux accès-disque en termes de surfaces de départ. En revanche pour d'autres stratégies ces coûts sont évalués en termes de *groupes* à transférer du disque ou vers le disque. D'après les tests que nous avons effectués, 1Go est nécessaire pour stocker 20000 surfaces.

5.5.2 Stratégies

Méthode aléatoire

Cette méthode est utilisée comme étalon pour les autres stratégies. Le *groupe* émetteur est choisi de manière aléatoire parmi tous les *groupes* de la scène n'ayant pas déjà émis leur énergie au cours de l'itération courante. Notons que cette stratégie ne prend en compte ni les coûts des accès-disque, ni la convergence de l'algorithme de radiosité.

Algorithme glouton S

Dans cette stratégie, l'objectif est de prévoir à court terme les coûts dûs aux accès-disque. Chaque *groupe* de la scène est considéré comme le *groupe* émetteur potentiel suivant et son coût de chargement C_{choix} est évalué. Ce coût (exprimé en termes de surfaces) comprend le coût dû à : la lecture du *groupe* émetteur, la lecture de ses *groupes* visibles et le déchargement de certains *groupes* dans le but de laisser de l'espace mémoire aux *groupes* à charger.

$$C_{choix} = (C_{chargement} - C_{dem}) + C_{dt}$$

$C_{chargement}$ est le nombre de surfaces total à charger en mémoire, C_{dem} correspond au nombre de surfaces déjà en mémoire et C_{dt} est le nombre de surfaces à rapatrier sur le disque.

Ce calcul est effectué pour tous les *groupes* de la scène à chaque étape. Le *groupe* émetteur choisi est celui dont C_{choix} est minimal (figure 5.5).

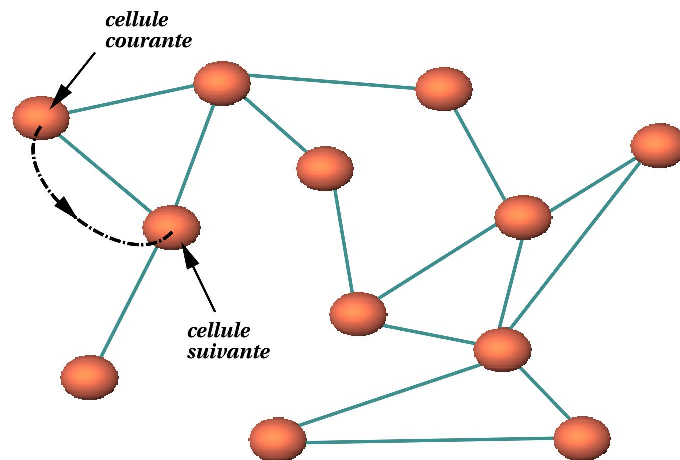


FIG. 5.5 – *Algorithme glouton.*

Algorithme glouton C

D'après la stratégie précédente, un *groupe* C comportant beaucoup de surfaces est coûteux à charger en mémoire, même si tous ses *groupes* visibles y sont déjà. Par conséquent, l'algorithme d'ordonnancement repousse alors le choix de C à la fin de chaque itération. Or ses *groupes* visibles ne seront sans doute plus en mémoire à ce moment précis et ils devront être à nouveau chargés en mémoire. Pour résoudre ce problème, nous proposons d'évaluer les coûts de chargement en termes de groupes. Ceci permet de favoriser la réutilisation des *groupes* déjà en mémoire.

$$C_{choix} = (C_{chargement} - C_{dem}) + C_{dt}$$

$C_{chargement}$ est le nombre de *groupes* total à charger en mémoire, C_{dem} correspond au nombre de *groupes* déjà en mémoire et C_{dt} est le nombre de *groupes* à écrire sur le disque pour faire de

la place aux *groupes* à charger.

Retour-arrière S

Les deux algorithmes glouton choisissent un *groupe* émetteur à chaque étape de calcul et ce choix ne pouvant être ni reconsidéré ni modifié peut mener à un ordonnancement global inefficace. Nous proposons par conséquent une stratégie permettant d'estimer précisément les coûts de chargement à moyen terme selon un algorithme de retour arrière (*backtracking*). Notre solution consiste à évaluer les coûts pour tous les parcours possibles d'un nombre N de *groupes* afin de choisir le moins coûteux. Ceci nous amène à choisir d'un seul coup une liste de N *groupes* émetteurs (figure 5.6).

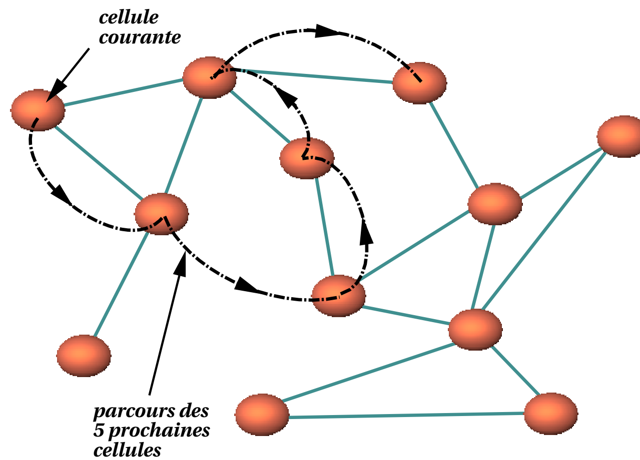


FIG. 5.6 – *Evaluation des coûts à moyen terme.*

Etant donné un ensemble de *groupes* $CELL$, nous choisissons un premier *groupe* C_0 comme *groupe* émetteur. Puis nous déterminons les *groupes* émetteurs suivants : $C_1 \in \{CELL - C_0\}$, $C_2 \in \{CELL - \{C_0, C_1\}\}$, ... , $C_{N-1} \in \{CELL - \{C_0, C_1, ..., C_{N-2}\}\}$. Toutes les possibilités pour choisir ces N *groupes* sont représentées par un arbre (de profondeur N) pour lequel la racine est C_0 et les noeuds sont les groupes. Dans cet arbre, la valeur associée à chaque arc (C_i, C_j) représente le coût dû aux accès-disques lorsque C_j est choisie en tant que *groupe* émetteur juste après C_i. La stratégie de retour-arrière consiste à parcourir cet arbre et extraire le chemin de longueur N de coût minimum. Le coût est évalué en termes de surfaces de la même manière que l'algorithme glouton S. Notons que pour accélérer la recherche, nous avons mis en œuvre une technique d'élagage permettant d'abandonner la recherche pour certains sous-arbres trop coûteux.

L'algorithme de retour-arrière donné figure 5.7 modifie le tableau C_choisis[ ] représentant la liste des N *groupes* choisis.

La fonction Choose_Cell() correspondant à cette stratégie de retour-arrière retourne successivement chaque élément de ce tableau. Lorsque le tableau a été entièrement parcouru, la fonction *Retour_Arrière* est appelée à nouveau avec $C_Courant = C_Choisis[prof_max]$. Ce processus est répété jusqu'à ce que tous les *groupes* de la scène aient été choisis dans une itération.

```
int Retour-Arrière(C_Courant, Cout, *Cout_min, prof, prof_max, C_Choisis[ ])
/* Cout= cout du chemin suivi, vaut 0 au premier groupe choisi */
/* cout_min : valeur minimale de tous les couts deja evalues */
{
    integer cout_min_courant = infini;
     /* valeur minimale des couts associés aux chemins */
     /* commencant au groupe courant et terminant au groupe le plus profond */

    integer cout_courant;
     /* cout du au chemin commencant au groupe courant et */
     /* finissant aux feuilles de l'arbre */

    si ( Cost > *Cout_Min ) retourne infini;
    si ( prof == prof_max) {
        si ( cout < *cout_min ) *cout_min = Cost;
        retourne 0;
    }
    pour chaque cellule C non marquée faire {
        cout_courant = Evalue_Cout(Current_Cell, C);
        marque(C);
        cout_courant + = Retour-Arrière(C,Cost,&Cout_Min, Prof + 1,
                Prof_Max, C_Choisis[ ]);
        si ( cout_courant < cout_min_courant ) alors {
            cout_min_courant = cout_courant;
            C_Choisis[prof] = C;
        }
    }
    supprime_marque(C);
    retourne cout_min_courant;
}
```

FIG. 5.7 – *Algorithme du retour-arrière.*

Retour-arrière C

Pour les mêmes raisons que pour les algorithmes gloutons, nous proposons une stratégie de retour-arrière dont les coûts sont évalués en termes de *groupes* afin de favoriser l'utilisation des *groupes* déjà en mémoire.

Algorithme du voyageur de commerce

Comme nous l'avons dit précédemment, notre problème d'ordonnancement est équivalent au problème du voyageur de commerce. Nous proposons donc un algorithme permettant de déterminer a priori un parcours de tous les *groupes* de la scène (figure 5.8). Cette stratégie consiste à évaluer les coûts d'Entrées/Sorties à long terme.

Pour cela, un graphe orienté et valué (OVG) est construit à partir du graphe de visibilité. Dans ce nouveau graphe OVG un noeud correspond à un *groupe* et un arc est associé à chaque paire de *groupes* $(C1, C2)$. La valeur d'un arc correspond au coût dû au chargement du *groupe* $C2$ (en termes de surfaces) dans le cas où $C2$ est choisie comme *groupe* émetteur juste après le *groupe* $C1$. Notons que ce coût ne prend en compte que les les coûts de lecture des groupes car la construction de OVG est effectué avant les calculs de radiosité. Par conséquent, nous ne connaissons pas les *groupes* déjà en mémoire et les estimations des coûts ne sont pas exactes (car nous ne pouvons estimer les écritures sur disque).

Ce graphe OVG est utilisé par l'algorithme du voyageur de commerce décrit par la figure 5.9.

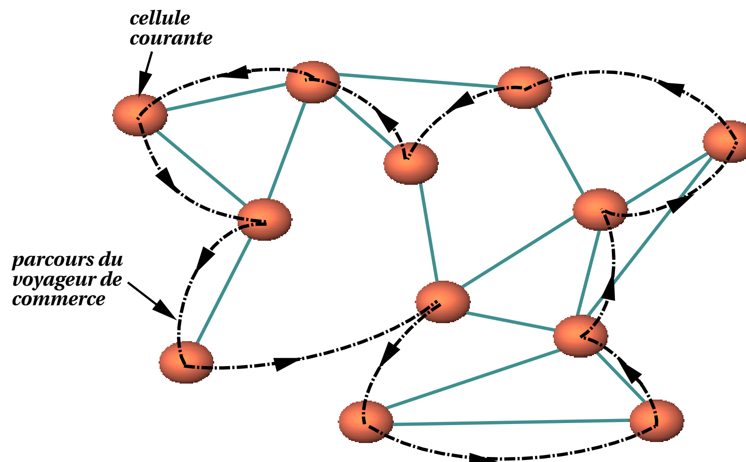


FIG. 5.8 – *Evaluation des coûts à long terme (pour des raisons de lisibilité, tous les arcs ne sont pas dessinés).*

La fonction *Choisir_Min_Arc* choisit l'arc non marqué de valeur minimum. *Non_Cycle* retourne $VRAI$ si les sommets successifs ne forment pas un cycle. Si les arcs de GA forment un cycle contenant tous les sommets de OVG alors *Complete_Cycle* retourne $VRAI$.

Maximum d'énergie

Afin d'accélérer la convergence de l'algorithme, les techniques de simulation d'éclairage utilisant la radiosité progressive choisissent de faire émettre la surface ayant la plus grande quantité d'énergie non encore émise.

Suivant ce principe, cette stratégie choisit le *groupe* ayant le maximum d'énergie non émise comme *groupe* émetteur. Notons que cette méthode ne prend pas en compte les coûts dus aux accès-disque.

```
Liste_Cellules Voyageur_Commerce(Graphe OVG)
 /* OVG est le graphe orienté-valué */
 {
        Liste_Cellules GA = NULL; /* Liste d'arcs représentant le meilleur chemin */
        booléen fini = FAUX;

        tant que non(fini) {
               pour chaque arc A de OVG {
                      Min = Choisir_Min_Arc(A);
                      Marque(Min);
                      si ( Non_Cycle(Min, GA, OVG) ) {
                             Ajoute_Arc_Liste(Min,GA);
                      }
                      sinon si ( Complete_Cycle(Min, GA, OVG) ) {
                             Ajoute_Arc_Liste(Min,GA);
                             fini= VRAI;
                      }
               }
        }
        return GA;
}
```

FIG. 5.9 – *Algorithme du voyageur de commerce.*

5.5.3 Remarque

Les stratégies proposées ci-dessus prennent en compte soit les coûts dûs aux accès-disque, soit la convergence de l'algorithme de radiosité. En effet, combiner ces deux critères s'avère être une tâche ardue car il est difficile de prévoir à long terme le comportement de l'algorithme de radiosité.

5.6 Implémentation et Résultats

5.6.1 Moyens matériels

Les calculs ont été effectués sur une machine de type SUN Ultraspark, 200 Mhz, 256 Mo de mémoire RAM et 523 Mo de mémoire virtuelle. La taille mémoire utilisée dans le cadre de ces calculs n'a pas depassé 150 Mo pour les cellules et 60 Mo pour les groupes de polygones.

5.6.2 Implémentation

Chaque *groupe* est stocké sur disque dans un fichier contenant une liste de surfaces, leurs réflectances, les spectres et distributions d'intensité des sources lumineuses. Le graphe de visibilité est également sauvegardé dans un fichier contenant la description de chaque groupe. Dans ce fichier, chaque *groupe* est décrit par les données suivantes :

- le nom du fichier correspondant au groupe ;
- le numéro d'identification du groupe ;
- la liste de ses *groupes* visibles (l'ensemble VS).

Au cours des calculs de radiosité, le contenu des *groupes* est modifié pour inclure les nouvelles mailles créées et les liens associés. Lorsqu'un *groupe* modifié est sauvegardée sur disque, le fichier correspondant contient toutes les nouvelles données : maillage, liens, radiosité, etc.

Afin d'accélérer les calculs, lorsqu'un *groupe* est choisi comme émetteur, nous vérifions qu'il a effecivement une quantité d'énergie suffisante à émettre. Si cela n'est pas le cas, alors ce groupe n'est pas chargé en mémoire, et l'algorithme passe au *groupe* émetteur suivant.

D'autre part, une option est offerte à l'utilisateur pour visualiser la scène de manière interactive alors que les calculs de radiosité sont en cours. Ceci est réalisé à l'aide de deux processus exécutés sur deux machines distantes : le premier effectue les calculs de radiosité alors que le second se charge de la visualisation interactive. Chaque surface mise à jour (ayant reçu de l'energie) par le programme de simulation d'éclairage est envoyée au processus d'affichage via un message TCP-IP, ceci est mise en œuvre à l'aide de la librairie *socket* sous UNIX.

Une autre possibilité de notre algorithme d'illumination globale (figure 5.3) est d'interrompre les calculs après un certain nombre d'itérations fixées par l'utilisateur, de sauvegarder la base de données en l'état et de relancer la simulation d'éclairage plus tard à partir de cet état pour d'autres itérations ou bien jusqu'à la convergence de l'algorithme de radiosité.

Les accès-disque impliquent de nombreuses allocations et désallocations mémoire, gérées par les routines *malloc* et *free* de la librairie C *libc.so*. Cependant, nous avons constaté de nombreuses opérations de swap alors que la taille des données manipulées n'était pas importante. Ceci est dû à l'inefficacité de la gestion de la fragmentation de la mémoire par ces routines. Suite à ces problèmes de *ramasse-miettes* nous avons décidé d'effectuer une gestion de la mémoire par nous-mêmes. Notre mécanisme consiste à réutiliser efficacement les emplacements mémoires laissés vides lorsque les données sont rapatriées sur le disque. Pour cela, à chaque type de donnée structurée T_s manipulé, nous avons associé deux listes chainées L_u et L_v. L_u contient les éléments utilisés (de type T_s) et L_v les éléments vides. Lorsqu'une donnée est rapatriée sur le disque, son élément correspondant de la liste L_u est placé dans la liste L_v. Réciproquement, lorsqu'une donnée est lue sur le disque, le premier élément de L_v est rempli par cette donnée et est placé dans la liste L_u. Si L_v est vide, alors un nouvel élément du type structuré correspondant est alloué en mémoire.

5.6.3 Résultats

Les sept stratégies d'ordonnancement décrites dans les sections précédentes ont été testées pour différentes scènes. Cette section présente quelques résultats obtenus avec une scène composée de 57786 surfaces, dont environ 2000 sont des sources lumineuses. Cette scène comporte 615 cellules et 12942 groupes.

Toutes nos stratégies d'ordonnancement ont été appliquées à la fois aux cellules et aux groupes de polygones. Dans le cas des cellules, la taille mémoire minimale requise était de $150Mo$. Dans le cas des groupes de polygones, cette taille minimale était de $60Mo$. C'est pourquoi pour notre simulation d'éclairage parallèle (cf. chapitre 6), nous avons utilisé les *groupes* plutôt que cellules.

D'autre part, la méthode de radiosité implique des temps de calcul très longs, notamment dans le cas d'environnements complexes. Afin d'accélérer les calculs lors de nos tests, nous avons utilisé un maillage grossier et un seuil de convergence élevé.

Les résultats obtenus avec nos stratégies d'ordonancement opérant sur les cellules sont donnés dans la figure 5.11 pour huit stratégies. Pour chaque stratégie trois courbes sont présentées. La

première courbe donne le nombre de surfaces résidentes en mémoire à la i^{eme} itération (courbe rouge). La seconde donne le nombre de surfaces appartenant à la cellule émettrice et ses cellules visibles (courbe verte). La troisième donne les coûts d'entrées/sorties en termes de surfaces pour la i^{eme} cellule émettrice (courbe bleue).

Cette figure montre que les stratégies *aléatoire* et *Maximum d'énergie* sont les plus coûteuses en termes d'accès-disque. Ce résultat est prévisible puisque ces derniers ne sont pas pris en compte par ces méthodes. Les temps de calculs correspondants aux stratégies sont donnés par la table 5.1.

Nous avons supposé dans ce chapitre que la base de données ne pouvait être stockée entièrement. Cependant, pour des bases de données moins complexes, toutes les cellules peuvent tenir en mémoire. En utilisant une machine disposant de $500Mo$ de mémoire RAM et $1Go$ de mémoire virtuelle, avec la stratégie du voyageur de commerce, les temps de calcul sont très nettement plus faibles que lorsque nous effectuons la gestion des accès-disque nous mêmes.

Enfin, losque nous appliquons nos stratégies d'ordonancement aux groupes de polygones, les temps de calculs obtenus sont environ $1,5$ fois inférieurs à ceux obtenus avec les cellules. Ceci s'explique par le fait que les relations de visibilité sont plus précises avec les groupes de polygones qu'avec les cellules. Par conséquent, les calculs de visibilité au cours de la simulation d'éclairage sont peu coûteux. Notons que la taille mémoire minimale requise pour effectuer les calculs de simulation d'éclairage pour les groupes de polygones est environ de 60 Mo.

Méthode Aléatoire	Voyageur de Commerce	Voyageur de Commerce + SWAP	Maximum d'Energie
229 mn	130 mn	27 mn	280 mn

Glouton S	Glouton C	Retour-Arrière S	Retour-Arrière C
161 mn	140 mn	1378 mn	975 mn

TAB. 5.1 – *Temps de calcul pour nos huit stratégies d'ordonnancement.*

Plusieurs conclusions peuvent être tirées de ces résultats. Premièrement, il est préférable d'utiliser les stratégies reposant sur les coûts d'accès-disque plutôt que sur l'énergie à émettre. De plus, les stratégies reposant sur le nombre de cellules à lire ou à écrire sur disque sont plus efficaces que les stratégies basées sur le nombre de surfaces. Ceci peut être expliqué de la manière suivante. Lorsque les coûts d'entrées/sorties sont évalués en termes de surfaces, les cellules comportant beaucoup de polygones ne sont choisies qu'à la fin de l'itération courante. Or leurs cellules visibles, ayant été choisies bien avant au cours de l'itération, ne sont plus en mémoire. La sélection de l'une de ces cellules importantes implique à nouveau le chargement d'un grand nombre de cellules. Par conséquent, les accès-disque augmentent ainsi que les temps de calcul. En revanche, si les coûts sont évalués en termes de cellules, une cellule comportant un grand nombre de surfaces peut être choisie n'importe quand au cours de l'itération et l'algorithme est plus efficace puisque la cohérence spatiale est mieux exploitée.

Nos stratégies d'ordonnancement n'ont pu être testées sur de nombreuses scènes à cause des temps de calcul importants dus à la méthode de radiosité. Cependant, pour toutes les scènes testées, l'algorithme du voyageur de commerce a donné les meilleurs résultats pour les cellules. En revanche, lorsque les *groupes* sont utilisés, le temps nécessaire pour déterminer a priori le parcours de tous les nœuds (du graphe de visibilité) est très important (plusieurs heures) car le graphe comporte un nombre très conséquent de nœuds (plusieurs dizaines de milliers).

Par ailleurs, les temps de calcul impliqués par les stratégies retour-arrière sont importants car le parcours de l'arborescence est très coûteux. De plus, l'ordre déterminé par ces stratégies n'est pas beaucoup plus avantageux que celui donné par notre algorithme du voyageur de commerce. Ceci explique les mauvaises performance de ces algorithmes.

La figure 5.10 montre trois courbes différentes donnant pour la i^{eme} *cellule* émettrice choisie : (i) le temps de calcul, (ii) le nombre de mailles en mémoire, (iii) le nombre de liens en mémoire. Notons que durant la première itération, seules les sources lumineuses émettent leur énergie. Les temps de calculs sont relativement faibles. En revanche, ces temps augmentent de façon importante durant les itérations suivantes. De plus, le nombre de mailles ainsi que le nombre de liens augmentent progressivement jusqu'à atteindre la valeur maximum associée à M_{limit}.

Les figures 5.12, 5.13 et 5.14 montrent deux images obtenues après un calcul de radiosité complet. Ce calcul e été effectué de manière plus précise en 5 jours de simulation. Le résultat de ce calcul est un environnement complexe comportant $360,000$ mailles et 3.7 millions de liens. Le stockage total de cette base de données aurait nécessité environ $3Go$ de mémoire pour effectuer les calculs de radiosité. Aussi, la visualisation d'une telle scène de manière interactive n'est pas possible sans optimisation. C'est pourquoi nous avons développé un module de visualisation interactif basé sur le graphe de visibilité. Lorsque l'observateur se trouve dans une cellule, seules ses cellules visibles sont transmises au pipeline graphique, de manière à alléger sa charge de travail. Ceci permet de fluidifer l'affichage et de rendre plus souple les déplacements interactifs de l'observateur dans la scène.

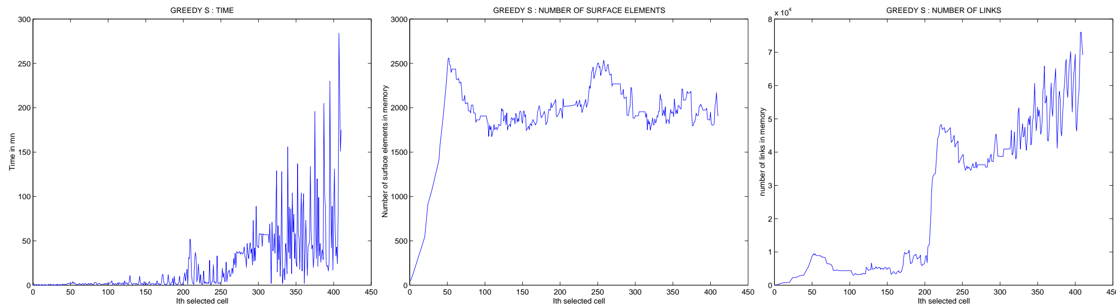


FIG. 5.10 – *Temps de calcul pour la i^{eme} cellule choisie, nombre de liens, nombre de mailles en mémoire pour la stratégie de retour-arrière C.*

5.7 Discussion

Dans ce chapitre nous avons voulu montrer qu'il était possible d'effectuer des calculs de radiosité pour des environnements complexes avec quasiment tout type de calculateur. Pour cela, seule une petite partie des données réside en mémoire au cours des calculs, mais les accès disque générés par cette technique ralentissent la simulation de manière importante. Pour les réduire, nous proposons 7 stratégies d'ordonnancement ayant pour objectif de prévoir à court, moyen ou long terme les coûts d'Entrees/Sorties. D'après nos résultats, les stratégies les plus simples se sont avérées les plus efficaces. En effet, le temps nécessaire à la recherche d'un chemin optimal est également du temps perdu pour l'algorithme de simulation d'éclairage. Il est donc nécessaire d'employer une stratégie peu coûteuse.

Notons que pour obtenir un maximum d'efficacité, notre méthode gère elle-même l'espace mémoire où sont stockés les *groupes* impliqués dans les calculs courants de radiosité.

Contrairement aux méthodes décrites dans [1], notre algorithme met en oeuvre une méthode d'emission d'énergie plutôt que de collecte afin de permettre à l'utilisateur de visualiser interactivement la simulation pendant le déroulement des calculs.

Afin de rendre notre algorithme utilisable sur plusieurs types de machines, plusieurs options sont disponibles. Par exemple, l'utilisateur peut fixer lui-même les ressources mémoire qu'il souhaite utiliser. De plus, il est possible d'interrompre les calculs après un certain nombre d'itérations puis de redémarrer le processus plus tard sans affecter les résultats des calculs de radiosité. Enfin, notre programme est configuré de manière à être exécuté soit sur machines SUN, soit SGI (Silicon Graphics Incorporation).

Ce travail a donné lieu a un rapport de recherche INRIA [58] et à un article dans une conférence internationale avec comité de lecture [59].

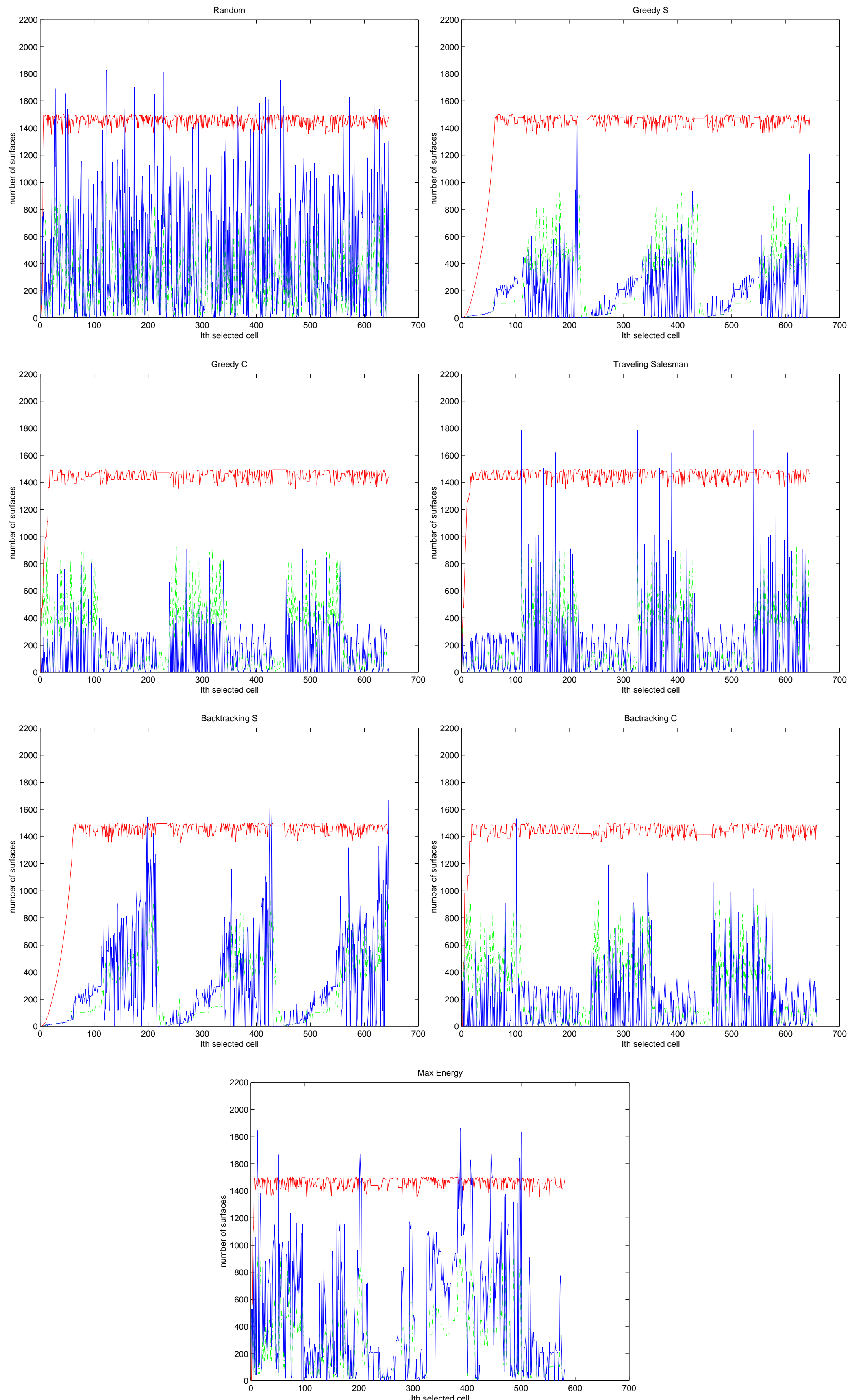


FIG. 5.11 – *Coûts d'Entrées/Sorties.*

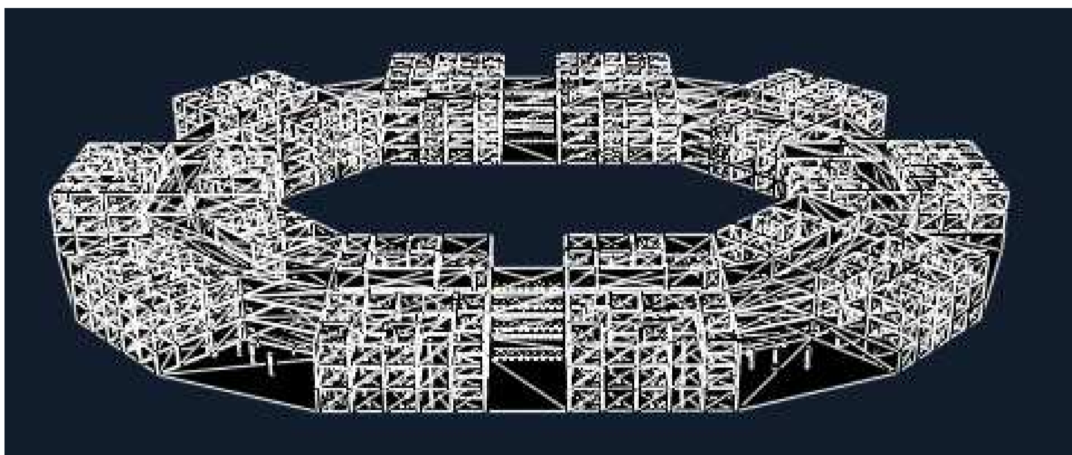


FIG. 5.12 – *Vue de haut, avant simulation d'éclairage.*

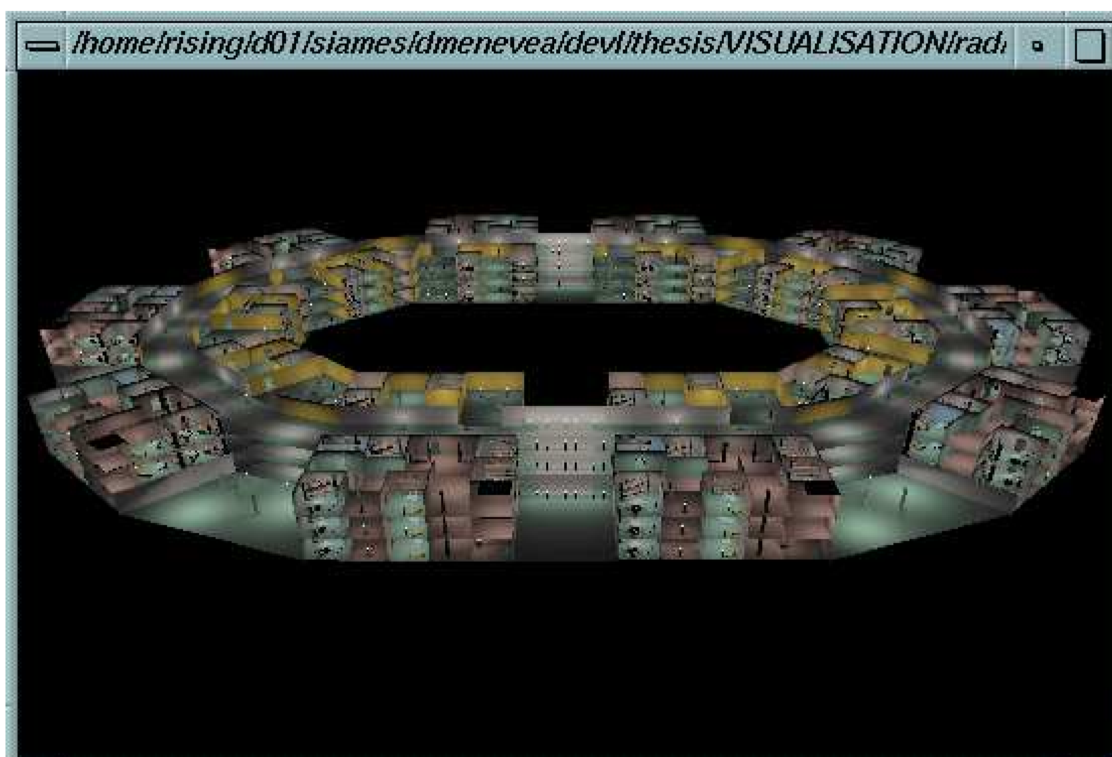


FIG. 5.13 – *Vue de haut, après simulation d'éclairage.*

FIG. 5.14 – *Vue intérieure.*

Chapitre 6

Simulation d'éclairage parallèle

6.1 Introduction

Nous avons montré qu'il était possible d'effectuer des calculs de radiosité de manière séquentielle dans des environnements architecturaux complexes, avec quasiment n'importe quel type de calculateur. Or, ces calculs restent malgré tout très coûteux en termes de temps de calcul. Pour accélérer le processus de simulation d'éclairage, la seule solution dont nous disposons est le parallélisme. Plusieurs approches hiérarchiques parallèles, basées sur un système maître-esclave ont déjà été proposées [60, 61] mais impliquent de nombreux messages volumineux et par conséquent ralentissent les calculs de manière importante. Une autre approche, de type SPMD a également été développée à l'IRISA mais pour la radiosité non hiérarchique (maillage constant) [62].

Dans ce chapitre, nous décrivons un nouvel algorithme de simulation d'éclairage hiérarchique parallèle mis en œuvre à l'aide de la bibliothèque MPI (Message Passing Interface). Cette bibliothèque permet d'exécuter notre programme sur une machine parallèle, ou un réseau hétérogène de machines (pouvant éventuellement comporter une ou plusieurs machines parallèles).

La mise en œuvre des algorithmes proposés dans ce chapitre est très étroitement liée à notre découpage et à notre programme de simulation d'éclairage séquentiel. Notre programme parallèle peut opérer sur les cellules ou les groupes de la base de données. Dans un premier temps, le graphe de visibilité est découpé en plusieurs sous-graphes, attribués aux processeurs. Un processeur effectue les calculs de simulation d'éclairage uniquement pour les données dont il a la charge, c'est à dire les cellules correspondant aux nœuds de son sous-graphe en appliquant la meilleure des 7 stratégies d'ordonnancement que nous avons proposées dans le chapitre 5. Au cours des calculs, les seuls messages envoyés sont dus à l'équilibrage de charge dynamique et à la terminaison de l'algorithme. La cohérence des données est assurée en utilisant un même disque accessible par tous les processeurs.

Ce chapitre est constitué des points suivants. Tout d'abord nous décrivons les techniques de simulation d'éclairage parallèles déjà existantes et s'appliquant à des environnements complexes. Puis nous donnons une vue globale de notre algorithme avant de le détailler point par point. Nous présentons ensuite les résultats que nous avons obtenus avant de conclure.

6.2 Travaux précédents

6.2.1 T. Funkhouser

L'objectif de T. Funkhouser [60] est d'effectuer une simulation d'éclairage parallèle dans des environnements architecturaux complexes à partir du découpage binaire de l'espace (BSP) décrit dans [12, 23, 1]. Rappelons que ce découpage binaire donne lieu à un ensemble de cellules

correspondant à des régions de l'environnement qui contiennent des groupes (note : un groupe est un groupe de polygones). A l'intérieur de chaque région, des calculs de visibilité très précis résultent en une base de données permettant de connaître pour chaque groupe :

- la liste de ses groupes visibles ;
- la liste des polygones provoquant une occlusion entre chaque paire de groupes.

Les cellules sont utilisées uniquement dans le but de déterminer les relations de visibilité entre les groupes. Une fois ces relations de visibilité établies, les étapes de calcul sont les suivantes :

- création de groupes de groupes de manière à avoir plus de groupes que de processeurs ;
- répartition par le maître des tâches aux processeurs esclaves, une tâche correspondant à un calcul de radiosité pour un groupe de groupes ;
- réception de groupes de groupes par chaque esclave qui calcule l'éclairement pour chaque surface de ce groupe, et renvoie le résultat du calcul au maître.

Création des groupes de groupes

Les groupes de groupes sont déterminés à l'aide d'un graphe dans lequel un nœud représente un groupe et le poids d'un arc entre deux nœuds est égal à un facteur de forme estimé. Des groupes de nœuds sont ensuite formés selon l'un des deux algorithmes suivants :

1. *Fusion de groupes de nœuds*
   Au départ de l'algorithme, il y a un nœud par groupe dans le graphe, puis deux groupes sont fusionnés en fonction du poids des arcs, selon un algorithme glouton. Le poids d'un arc entre deux groupes de nœuds A et B équivaut à la somme des facteurs de forme (estimés) entre les groupes de A et de B. Tant que les conditions suvantes sont vérifiées, des groupes sont fusionnés :
   - le nombre de groupes est supérieur à une valeur $MinGroupes$. $MinGroupes$ est spécifié par l'utilisateur, et $MinGroupes > N_{procs}$, N_{procs} étant le nombre de processeurs ;
   - le nombre de liens estimé pour chaque groupe est inférieur à $MaxLinks$. Ce nombre est estimé en fonction des facteurs de formes (représentés par des arcs valués dans le graphe).
2. *Découpage binaire du graphe*
   Pour cette stratégie, il ne s'agit pas de regrouper des nœuds, mais plutôt de subdiviser le graphe en plusieurs sous-graphes. Puisque les nœuds correspondent à des groupes, Funkhouser propose d'utiliser les plans de découpage obtenus au cours de la structuration. A chaque étape de la subdivision, un sous-graphe est subdivisé en deux groupes de nœuds (groupes de groupes) selon un plan de découpage. Note : les groupes de groupes ne sont autres que les cellules obtenues par le découpage BSP.

Simulation d'éclairage

1. *Le maître*
   Le processeur maître maintient une table indiquant les groupes présents dans chaque processeur esclave. Avant d'envoyer un groupe de groupes G à un esclave S, il vérifie que tous les groupes de G et tous les groupes visibles de G sont présents dans S. Si ce n'est pas le cas, alors il lit les groupes manquants sur le disque, crée un message contenant les groupes, leurs mailles, liens, etc. et l'envoie à S. Ceci permet d'éviter de dupliquer tous les groupes sur processeur.

Trois types d'ordonnancement sont mis en œuvre :

(a) FFT (First Fit Decreasing) : le principe est de distribuer d'abord les tâches les plus coûteuses à exécuter.

(b) WS (Working Set) : le processeur maître choisit une tâche à donner à un esclave de manière à envoyer le plus petit message possible. Ceci revient donc à calculer le pourcentage de groupes déjà en mémoire chez l'esclave.

(c) FFT-WS : cette dernière stratégie consiste à combiner les deux autres. Le maître trie les tâches à exécuter selon la stratégie FFT et choisit parmi les N premières celle qui correspond le mieux pour WS.

2. *Les esclaves*
   Tous les processus esclaves exécutent le même algorithme :
   - attendre une tâche du maître ;
   - récupérer un groupe de groupes G et ses groupes visibles ;
   - calculer l'éclairement pour toutes les polygones de tous les groupes de G selon la méthode itérative de Jacobi (jusqu'à convergence) ;
   - renvoyer le résultat du calcul au maître.

6.2.2 C. C. Feng et S. N. Yang

Dans un article plus récent [61], Feng et Yang décrivent un algorithme de radiosité hiérarchique parallèle utilisant la librairie PVM (Parallel Virtual Machine). Leur approche est également basé sur un système maître esclave, mais avec une granularité plus fine que pour l'algorithme de Funkhouser [60].

Dans une étape préliminaire, la scène est découpée en cellules de manière binaire à l'aide de la méthode BSP [12, 23, 1]. Puis tous les calculs sont gérés par un maître distribuant des tâches à des esclaves. Le principe de l'algorithme est le suivant :

1. des ensembles de visibilité E_vi sont déterminés (en parallèle) pour chaque cellule. L'ensemble E_vi associé à une cellule C_i contient les polygones de C_i et les polygones visibles de C_i ;

2. le maître détermine ensuite un parcours des ensembles de visibilité E_vi selon un algorithme du voyageur de commerce. L'objectif est de réduire les accès au disque ;

3. Puis les ensemble E_vi sont choisis un à un suivant ce parcours par le processeur maître. Ce dernier attribue les polygones de C_i (associé à E_vi) aux processeurs esclaves qui effectuent les calculs de radiosité.

Construction des ensembles de visibilité

La construction des ensembles de visibilité E_vi est effectuée de manière parallèle. Dans un premier temps, le processeur maître distribue les cellules aux esclaves dont le rôle est de déterminer les objets visibles à partir des ouvertures. Pour cela, des *tampons de visibilité* sont associés à chaque ouverture de la cellule, de manière à déterminer les objets provoquant une occlusion entre deux ouvertures.

Ensuite, le maître construit un graphe (appelé *graphe d'adjacence*) pour lequel un nœud représente une cellule et un arc entre deux nœuds correspond à une ouverture. Ce graphe est distribué aux processeurs esclaves qui déterminent les objets visibles de chaque cellule à l'aide des tampons de visibilité. Ce calcul est effectué par un parcours en profondeur du graphe d'adjacence

et en concaténant les tampons de visibilité. Le résultat de ces calculs est transmis au processeur maître.

Simulation d'éclairage parallèle

Les calculs de radiosité sont effectués en parallèle pour un seul $E_v i$ à la fois et l'ordre dans lequel les $E_v i$ sont parcourus est déterminé par un algorithme du voyageur de commerce (calculé par le maître). Chaque processeur esclave se charge de calculer l'éclairement pour un sous-ensemble de polygones de $E_v i$ selon un algorithme d'émission d'énergie (*shooting*). Pour cela, le processeur maître partage la cellule C_i (associée à $E_v i$) en plusieurs groupes de polygones (estimant le coût de calcul pour chaque groupe) et distribue ces groupes aux processeurs esclaves. Pour réduire le nombre de messages, $E_v i$ est entièrement dupliqué dans la mémoire de chaque esclave. Enfin, un équilibrage de charge dynamique entre les processeurs esclaves est mis en œuvre suivant une méthode de “vol de tâche” (task stealing).

Lorsque tous les $E_v i$ ont émis leur énergie une fois, la seconde étape de l'algorithme consiste à faire réémettre l'énergie des ensembles $E_v i$ jusqu'à convergence de l'algorithme. Cette fois l'ordre dans lequel les $E_v i$ sont choisis est différent : à chaque étape, le maître choisit celui qui a le plus d'énergie à émettre.

6.2.3 B. Arnaldi et. al.

Dans [62], la scène est subdivisée en autant de régions qu'il y a de processeurs. Les régions sont des voxels ayant tous la même taille et la simulation d'éclairage est effectuée selon un algorithme de radiosité progressive.

Pour cela, les polygones de l'environnement sont maillés de façon régulière et sur chaque maille est appliquée une hémisphère discrétisée en pixels à travers lesquels sont lancés des rayons. Un *masque de visibilité* est associé à cette hémisphère. Si le rayon lancé à travers un pixel i coupe un polygone P de la région, alors le i^{eme} pixel du *masque de visibilité* est marqué et l'énergie arrivant sur P est calculée. Si à l'intérieur d'un même voxel (i.e. processeur) tous les pixels du masque de visibilité n'ont pas été marqués, alors la maille et son masque de visibilité sont transmis aux processeurs traitant les voxels voisins. Ainsi, les mailles peuvent transiter de région en région jusqu'à ce que tout le masque de visibilité soit marqué.

6.2.4 Discussion

L'accélération des calculs de simulation d'éclairage par la parallélisation n'est pas récente. En effet, dès l'avènement de la méthode de radiosité plusieurs algorithmes parallèles ont été mis en œuvre. En revanche, le problème de la simulation d'éclairage dans des environnements complexes a été peu étudié et les solutions proposées sont relativement coûteuses en termes de messages envoyés entre les différents processeurs prenant part aux calculs.

Le travail de Funkhouser est original car il est basé sur une approche de type groupe itératif, dont il a démontré la convergence de manière mathématique. Cependant, sa méthode implique le transit de nombreux messages entre les esclaves et le maître, ce qui réduit considérablement les performances de l'algorithme.

Feng et. al. ont pour objectif de réduire la granularité de l'algorithme en distribuant les polygones de chaque ensemble $E_v i$ à tous les processeurs esclaves. Or, la duplication des ensembles de visibilité ($E_v i$) requiert une taille mémoire plus importante que l'algorithme de Funkhouser et

le nombre de messages envoyés entre les processeurs n'est pas réduit. Par ailleurs, la seconde étape de l'algorithme est basé sur l'énergie non émise par les ensembles E_vi. Or nous avons vu dans le chapitre précédent qu'une telle stratégie est inefficace et engendre des pertes de temps très conséquentes. Enfin, la convergence de leur algorithme parallèle n'est pas prouvée. En effet la démonstration est différente de celle de T. Funkhouser puisque l'algorithme de radiosité est cette fois basé sur la méthode itérative de Southwell (*shooting*).

6.3 Nos objectifs

Nos objectifs sont multiples :

- Accélérer les calculs de radiosité en utilisant une approche itérative inspirée de la méthode de résolution de Gauss-Seidel car elle converge plus rapidement que celle de Jacobi.

- Prouver formellement la convergence de cette approche.

- Dans le cas d'une machine parallèle, réduire le nombre de messages ainsi que leur taille de manière drastique en utilisant un disque dur commun à tous les processeurs.

- Dans le cas d'un réseau de machines hétérogènes, stocker la base de données sur un disque unique et y accéder via NFS (Network File System), de manière transparente à l'utilisateur et au programmeur. Ceci facilite notablement la mise en œuvre de l'algorithme, même si le nombre de messages envoyés de manière implicite (par le système) est relativement important.

- Proposer un algorithme d'équilibrage de charge efficace et un algorithme de terminaison élégant.

- Enfin, utiliser un langage de programmation parallèle qui soit portable tel que PVM (Parallel Virtual Machine) ou MPI (Message Passing Interface) afin de pouvoir exécuter le programme sur différents types de plateforme : machines parallèles, réseau de machines hétérogènes, etc.

6.4 Principes de notre méthode

Afin d'atteindre les objectifs sus-cités, nous proposons une approche parallèle de type SPMD (Single Program Multiple Data) mise en œuvre à l'aide de la librairie MPI. L'utilisation de MPI nous permet d'exécuter notre programme aussi bien sur un réseau de machines hétérogènes que sur une machine parallèle.

Pour les mêmes raisons que dans le chapitre 5, nous parlerons de groupe par simplicité de langage. Néanmoins, un groupe correspond à un ensemble de polygones ou à une cellule. Il est possible d'utiliser notre programme de simulation d'éclairage parallèle indifféremment pour des groupes ou des cellules car la structuration et le regroupement de surfaces produisent des fichiers ayant le même format.

Les grandes lignes de notre algorithme sont les suivantes. Dans un premier temps, le graphe de visibilité est découpé en plusieurs sous-graphes, chaque sous graphe étant attribué à un processeur. Chaque processeur est chargé de calculer l'éclairement de tous les groupes dont il a la charge, c'est à dire tous les nœuds du sous-graphe qui lui a été attribué (figure 6.1).
Rappelons que notre algorithme de simulation d'éclairage est basé sur des techniques de projection avec des ondelettes comme fonctions de base et permet deux types de résolution : *shooting* (i.e. les polygones émettent leur énergie) ou *gathering* (i.e. les polygones collectent de l'énergie).

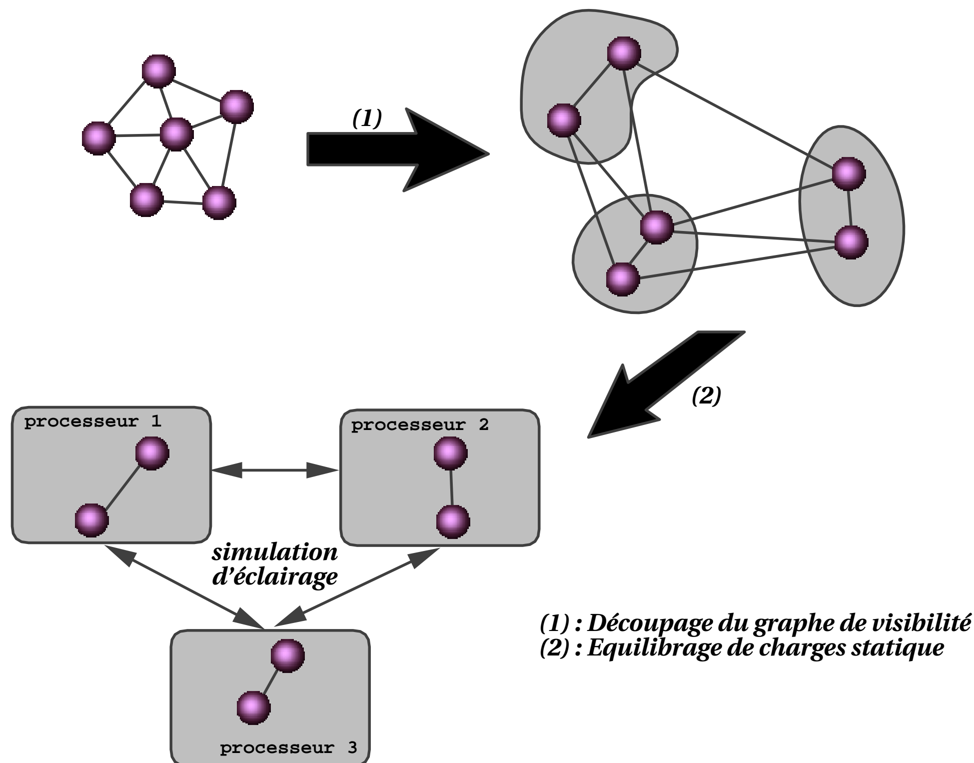


FIG. 6.1 – *Principe de notre algorithme parallèle.*

Pour plus de précisions, le lecteur pourra se référer au chapitre 1.

Contrairement à la simulation d'éclairage séquentielle, nous avons cette fois préféré utiliser l'algorithme de collecte (*gathering*) plutôt que d'émission d'énergie (*shooting*) pour les calculs parallèles, notamment pour résoudre le problème de la cohérence des données. En effet, lorsqu'un groupe est choisi pour collecter de l'énergie lumineuse, seulement l'énergie de ses polygones est mise à jour. Ainsi, une même surface ne peut pas être mise à jour par deux processeurs différents.

Chaque processeur effectue donc les calculs de simulation d'éclairage une fois pour chacun de ses groupes (figure 6.3). A la fin de cette étape, tous les polygones de la scène ont collecté de l'énergie lumineuse une fois et une seule. Cela correspond à une itération de la méthode itérative permettant de résoudre l'équation de radiosité.

Pour cela, chaque processeur choisit un groupe (appelé *groupe récepteur*), collecte l'énergie arrivant sur ce groupe, sélectionne un autre groupe et répète ce processus pour tous ses groupes.

La sélection d'un groupe implique le chargement en mémoire de plusieurs groupes non attribués au processeur mais visibles du groupe récepteur. Pour réduire les transferts d'informations entre le disque et la mémoire, nous utilisons la stratégie du voyageur de commerce, décrite dans le chapitre 5.

Rappelons que les structures de données manipulées par les processeurs sont placées sur un même disque accessible par tous les processeurs. Plus précisément, les données sont structurées de la manière suivante. Un répertoire commun contient toute la base de données (c'est à dire tous les groupes). D'autre part, chaque processeur dispose d'un répertoire local, non accessible par les autres processeurs. Chacun peut y écrire les fichiers contenant les nouvelles mises à jours qu'il a effectuées.

Le répertoire commun contient la totalité des groupes, mis à jour à l'itération $i-1$ et les répertoires locaux contiennent les mises à jour effectuées à l'itération courante i.

Ainsi, lorsqu'un processeur calcule l'éclairement de l'un de ses groupes C à l'itération i, il utilise les groupes visibles de C. Or certains d'entre eux appelés C_{autre} sont attribués à d'autres processeurs, mais ils sont disponibles sur le répertoire commun qui contient les groupes mis à jour à l'itération précédente $i-1$. C est alors modifié (nouveau maillage, nouveaux liens) ainsi que ses groupes visibles. Après avoir calculé l'éclairement pour C, le processeur sauvegarde uniquement les structures de données associées à C dans son répertoire local. Celui-ci contient donc les mises à jour de C à l'itération i. D'autre part, les groupes C_{autre} sont pris en compte et sauvegardés de la même manière par d'autres processeurs.

Lorsque tous les processeurs ont calculé l'éclairement des groupes qui leur ont été attribués, l'itération courante i est terminée et une étape de synchronisation a alors lieu, au cours de laquelle chaque processeur déplace les fichiers de son répertoire local vers le répertoire commun. Cette étape est extrêmement rapide si tous les répertoires locaux sont sur le disque commun car le système (UNIX) modifie simplement les noms de fichiers. Dans le cas contraire, il y a effectivement un déplacement du contenu des fichiers, et cette opération est bien plus lente.

Ensuite, une nouvelle itération de la méthode de résolution est lancée. Un algorithme complet est donné par la figure 6.2. Il décrit la collecte d'énergie (*gathering*) sur un processeur et jusqu'à convergence.

La seule restriction inhérente à cet algorithme est la nécessité de disposer d'un disque commun à tous les processeurs. Remarquons que dans le cas ou nous utilisons un réseau de machines, ce problème est géré par le sytème de gestion de fichiers NFS (Network File System) et de façon transparente pour l'utilisateur et le programmeur. En revanche ce mode d'utilisation augmente de façon considérable le nombre de messages envoyés entre processeurs.

Dans les sections suivantes, nous abordons les différents points liés à la parallélisation de l'algorithme de radiosité : les problèmes de convergence, de terminaison et d'équilibrage de charge.

6.5 Description détaillée de notre méthode

Dans un premier temps nous décrivons le découpage du graphe de visibilité. Ce découpage permet d'avoir une première répartition raisonnable des données sur l'ensemble des processeurs (i.e. équilibrage de charge statique). Nous décrivons ensuite notre organisation des fichiers sur le disque et nos algorithmes d'équilibrage de charge dynamique, de synchronisation, et de terminaison.

6.5.1 Découpage du graphe de visibilité

L'algorithme proposé est du type SPMD (Single Program, Multiple Data). Le même code est utilisé sur tous les processeurs, seules les données en mémoire varient d'un processeur à l'autre. Les calculs doivent donc être distribués le plus équitablement possible sur tous les processeurs.

Le principe est donc de distribuer les groupes de la scène sur les différents processeurs, de manière à :

1. avoir la meilleure localité des données possibles afin de réduire les accès-disque ;
2. réaliser le meilleur équilibrage de charge statique possible.

```
RadiositéParallèle(C) {
    C : Ensemble des groupes à traiter ;
    Booléen convergence = FAUX ;

    tant que ( non convergence ) {

        /* Collecte l'energie de tous les groupes du groupe C en utilisant */
        /* la stratégie du voyageur de commerce pour l'ensemble des groupes de C */
        CollecteParallèle(C) ;

        /* Lorsque la collecte est terminée, l'équilibrage de charge a lieu */
        EquilibreCharge() ;

        /* Enfin on teste la convergence de l'algorithme */
        convergence = Terminaison() ;
    }
}
```

FIG. 6.2 – *Simulation d'éclairage sur un processeur.*

```
CollecteParallèle(C) {
    C : Ensemble des groupes à traiter;;
    CH: Chemin à suivre;;
    Env: environement comprenant un groupe et ses groupes visibles;;

    CH = VoyageurCommerce(C);;

    Pour chaque groupe Ci de CH faire {

        /* Charger en mémoire Ci et ses groupes visibles */
        Env = Chargement_mémoire( Ci ) ;

        Pour chaque surface Si de Ci faire {
            Pour chaque surface Sj de Env faire {
                CollecteSurface(Si, SJ) ;
            }
        }
    }
}
```

FIG. 6.3 – *Collecte d'énergie parallèle pour un groupe de groupes.*

Pour atteindre cet objectif, nous proposons de découper le graphe de visibilité de manière à avoir des groupes de nœuds (groupes) proches les uns des autres (figure 6.4).

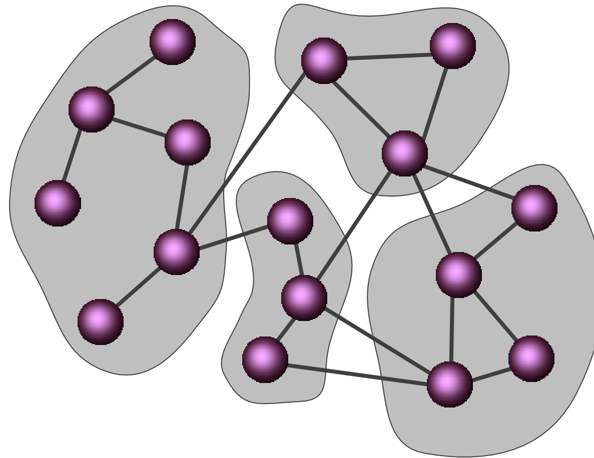


FIG. 6.4 – *Exemple de découpage d'un graphe de visibilité.*

Méthodes existantes

Les techniques les plus utilisées (dans [60] par exemple) à notre connaissance pour effectuer des découpages de graphes sont les suivantes :

1. *Découpage binaire récursif.* A l'aide de certains critères (poids des arcs, des nœuds, etc.), le graphe est découpé en deux sous- graphes. cette opération est répétée de manière récursive pour ces sous-graphes jusqu'à atteindre des groupes de taille minimale (figure 6.5).

2. *Regroupement binaire.* Cette méthode effectue l'opération inverse. Au départ de l'algorithme, chaque groupe constitue un groupe à lui seul. Puis les groupes sont rassemblés deux à deux selon certains critères jusqu'à atteindre le nombre désiré de groupes (figure 6.6).

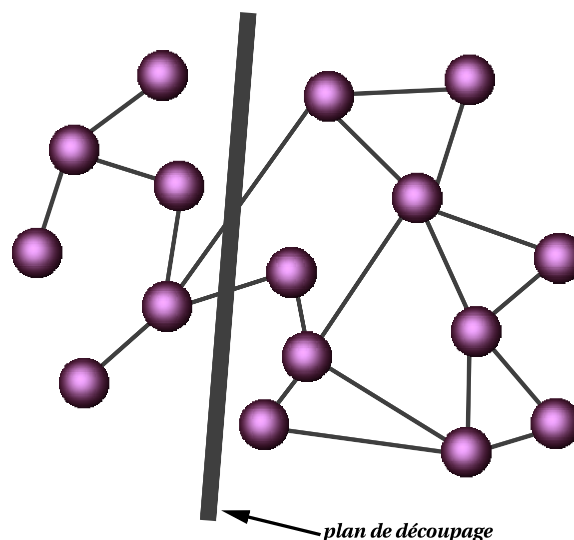


FIG. 6.5 – *Découpage binaire d'un graphe.*

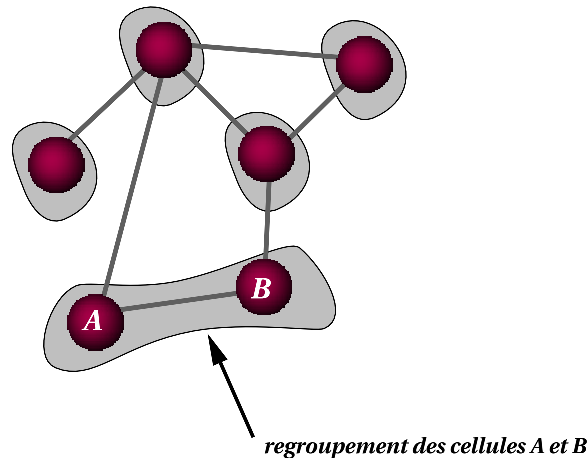


FIG. 6.6 – *Regroupement de groupes.*

Notre approche

L'objectif étant de regrouper plusieurs nœuds d'un graphe en utilisant la notion distance entre nœuds, nous proposons d'utiliser un algorithme de classification de type *nuées dynamiques* [43], employé en analyse de données. Ce choix est motivé par le fait que les algorithmes classiques de découpage de graphes nécessitent des critères particuliers (comment découper un graphe de manière binaire ? ou encore quels nœuds regrouper ?) alors que dans notre problème de découpage, les techniques de classification répondent parfaitement à notre attente. D'un point de vue général, l'objectif de la classification est de rassembler un certain nombre de points dispersés dans un espace (le plus souvent bidimensionnel), pour d'obtenir des ensembles de points localement proches.

- *k-means ou les nuées dynamiques*
  Parmi les différentes méthodes proposées dans la littérature, l'une d'entre elles a retenu notre attention. En effet, la technique des nuées dynamiques permet de trouver de façon élégante N groupes de points proches les uns des autres, N étant connu à l'avance. Nous avons déjà décrit le principe de cet algorithme dans le chapitre 3.

  Dans cet algorithme, les groupes sont constitués à l'aide de barycentres mobiles. Au départ de l'algorithme, N barycentres sont choisis de manière aléatoire. Ils se déplacent à chaque étape du calcul jusqu'à atteindre une position fixe. Lorsque tous les barycentres sont stabilisés, les groupes sont constitués.

- *Analogie avec les graphes.*
  Notre problème est relativement similaire à celui de la classification puisque nous souhaitons regrouper les nœuds d'un graphe. De plus nous souhaitons avoir dans un même groupe des nœuds proches les uns des autres.

- *Application au graphe d'adjacence ou graphe de visibilité.*
  Nous pouvons utiliser soit le graphe de visibilité (où les nœuds correspondent à des groupes ou des cellules), soit le graphe d'adjacence (seulement pour les cellules). Le graphe d'adjacence (obtenu lors du découpage de la scène en cellules) nous donne sans doute une meilleure notion de proximité puisqu'un arc entre deux nœuds indique que les cellules correspondantes sont voisines. L'algorithme de découpage est effectué comme suit. Tout d'abord, le graphe (d'adjacence ou de visibilité) G_{groupe} est recopié dans un nouveau graphe

G_{val}. A chaque arc de G_{val} nous affectons un poids égal à 1. Puis nous déterminons les distances entre toutes les paires de nœuds de G_{val} en calculant la fermeture transitive avec l'algorithme de Warshall. La distance entre deux nœuds est égale au nombre d'arcs constituant le plus court chemin joignant ces nœuds. Enfin, à l'aide de cette notion de distance entre deux nœuds du graphe, nous pouvons découper notre graphe en N groupes, selon l'algorithme des nuées dynamiques, N étant le nombre de processeurs utilisés au cours de l'éxécution parallèle. Nous avons ainsi une répartition des données permettant un équilibrage de charge statique.

- *Structures de données résultantes.*
  Lorsque tous les groupes sont déterminés, le programme crée un fichier contenant l'ensemble des informations. Ce fichier est toujours nommé *partition.dat* et contient :
  - Le nombre de groupes.
  - puis pour chaque groupe : le nombre de nœuds et la liste des indices des groupes correspondants.

  La figure 6.7 décrit un fichier généré par notre programme de découpage de graphe.

```
nb_groupes 3

Groupe 0
nb_cellules 3
1
2
3

Groupe 1
nb_cellules 2
4
5

Groupe 2
nb_cellules 3
6
7
8
```

FIG. 6.7 – *Fichier contenant la desription des sous-graphes.*

Ce fichier situé dans le répertoire commun est accessible en lecture pour tous les processeurs prenant part aux calculs. Ainsi chaque processeur connaît non seulement les groupes dont il a la charge mais également la répartition des autres groupes de la scène sur les autres processeurs.

6.5.2 Organisation de l'espace disque et cohérence des données

Tous les fichiers représentant les groupes sont situés sur un disque accessible par tous les processeurs. Un répertoire commun est utilisé, mais pour éviter les problèmes d'accès concurrents,

chaque processeur possède son propre répertoire sur le disque commun et peut ainsi y écrire ses propres résultats (figure 6.8).

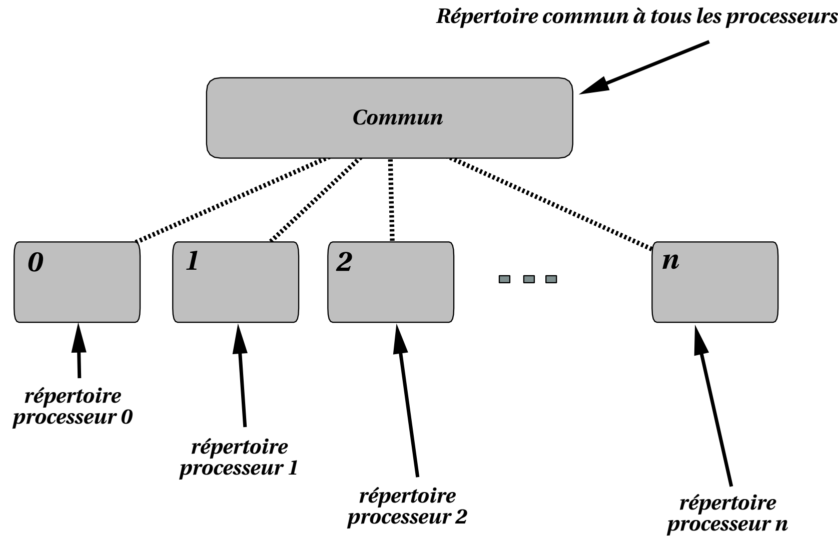


FIG. 6.8 – *Organisation de l'espace disque.*

Selon cette organisation du disque, un processeur P_i effectuant les calculs de radiosité pour un groupe C_j pendant une itération k peut disposer :

- de certains groupes visibles de C_j, calculés par P_i à l'itération k (dans le répertoire local à P_i) ;
- de tous les groupes visibles de C_j, calculés par $P_{i'}, i' \in [1, N]$, à l'itération $k-1$ (dans le répertoire commun), N étant le nombre total de processeurs ;

En pratique, nous utilisons toujours les groupes déjà calculés par P_i (c'est à dire à l'itération k) dans le but d'accélérer la convergence des calculs de radiosité.

Les données évoluent au cours du temps puisque les radiosités ainsi que le maillage des polygones de chaque groupe sont modifiés par les processeurs qui en ont la charge. Pour conserver la cohérence des données sur le disque, tous les groupes mis à jour sont déplacés vers le répertoire commun à la fin de chaque itération.

Note : les calculs de transfert énergétiques entre une surface A et une surface B impliquent une mise à jour des mailles de A et de B. Or si A et B, appartiennent à deux processeurs différents P_1 et P_2, les mises à jour de A sont uniquement sauvegardées par P_1 et celles de B par P_2. La cohérence des données n'est assurée que si le maillage de A (resp. B) est identique pour P_1 et pour P_2. Dans notre mise en œuvre, cela est vérifié puisque les maillages des surfaces A et B sont effectués uniquement selon des critères géométriques et non énergétiques.

Outre les problèmes dûs à la cohérence des donnée, plusieurs autres points sont prépondérants lors de la conception d'un algorithme parallèle. On peut citer l'équilibrage de charge, la terminaison, les problèmes de congestion et l'interblocage. Notre algorithme de radiosité parallèle prévoit la gestion de chacun de ces points particuliers.

6.5.3 Equilibrage de charge dynamique

Il est très difficile d'évaluer a priori les temps de calcul pour la méthode de radiosité car ils dépendent non seulement des propriétés des matériaux (spectres de réflectance), mais également

de la géométrie de l'environnement, du nombre de surfaces ayant une émittance propre, etc. Il est donc fort probable que notre stratégie d'équilibrage de charge statique soit insuffisante. C'est pourquoi nous avons mis en œuvre un algorithme d'équilibrage de charge dynamique de type "vol de tâche" (ou *task stealing*). Nous pouvons considérer une tâche comme le calcul de radiosité pour un groupe C et ses groupes visibles. L'attribution d'une tâche à un processeur P est effectuée en indiquant à P qu'il doit effectuer les calculs d'éclairement pour le groupe C. Nous utiliserons plus simplement le terme *donner un groupe* à P. Le principe est le suivant. Lorsqu'un processeur a terminé les calculs de simulation d'éclairage pour toutes les mailles dont il a la charge, il demande aux autres processeurs (tour à tour) de nouveaux groupes. Cet algorithme d'équilibrage de charge est décrit par la figure 6.9.

```
EquilibreCharge() {
    Env : environnement comprenant un groupe et ses groupes visibles ;
    C : Identificateur d'un groupe ;
    PR : Ensemble des Processeurs ;
    Pi : Identificateur d'un processeur ;
    Mon_id : Identificateur du processus courant ;

    Mon_id = get_id() ;

    Pour chaque Processeur Pi ≠ Mon_id ∈ PR faire {
        PrevenirProcesseur(Pi) ;
        C = AttendreTravail(Pi) ;
        Env = Charger en mémoire C et ses groupes visibles ;
        Pour chaque surface Si de Ci faire {
            Pour chaque surface Sj de Env faire {
                CollecteSurface(Si, SJ) ;
            }
        }
    }
}
```

FIG. 6.9 – *Equilibrage de charge dynamique.*

Nous avons vu que notre algorithme de simulation d'éclairage parallèle opérait en plusieurs itérations, à la manière de la méthode de radiosité. Au cours de chaque itération, lorsqu'un processeur P a effectué tous les calculs de radiosité pour ses groupes, il envoie des messages aux autres processeurs afin de récupérer de nouveaux groupes à traiter (vol de tâche). Ces groupes (dérobés aux autres processeurs) sont définitivement attribués à P, jusqu'à ce qu'un autre processeur ne les réclame à son tour. Grâce à cette méthode, la répartition des groupes est modifiée itération après itération et tend vers une configuration optimale en termes d'équilibrage de charge.

Grâce à la fonction *PrevenirProcesseur()* un processeur P envoie au processeur P_i un message contenant les identificateurs (entiers) de tous les groupes qu'il a en mémoire. P_i peut ainsi déterminer parmi ses groupes le meilleur à transmettre. Ce groupe est déterminé à l'aide de l'algorithme *glouton C* décrit dans le chapitre 5. Le principe de cet algorithme glouton est le suivant. Pour chaque groupe non encore traité, le nombre de groupes à charger et à décharger par le processeur demandeur P est évalué. Le processus P_i choisit le groupe le moins coûteux en termes d'Entrées/Sorties.

L'identificateur du groupe déterminé est alors envoyé par P_i et reçu par P par le biais de la fonction *AttendreTravail()*.

Lorsque tous les processeurs ont terminé l'iteration courante, chaque processeur déplace vers le répertoire commun les groupes qu'il a déjà traités et sauvegardés dans son propre répertoire. Ainsi l'ensemble des processeurs aura accès aux groupes mis à jour à l'itération $i - 1$.

6.5.4 Prévision à long terme vs. équilibrage de charge dynamique

Rappelons que pour chaque processeur les groupes sont choisis dans un ordre spécifique, déterminé au début de chaque itération par un algorithme du voyageur de commerce. Or lors de l'équilibrage de charge, un processeur P peut céder un groupe à un autre processeur P_d qui en fait la demande. L'éclairement du groupe correspondant est alors calculé par P_d et l'ordre de parcours de P est modifié (figure 6.10).

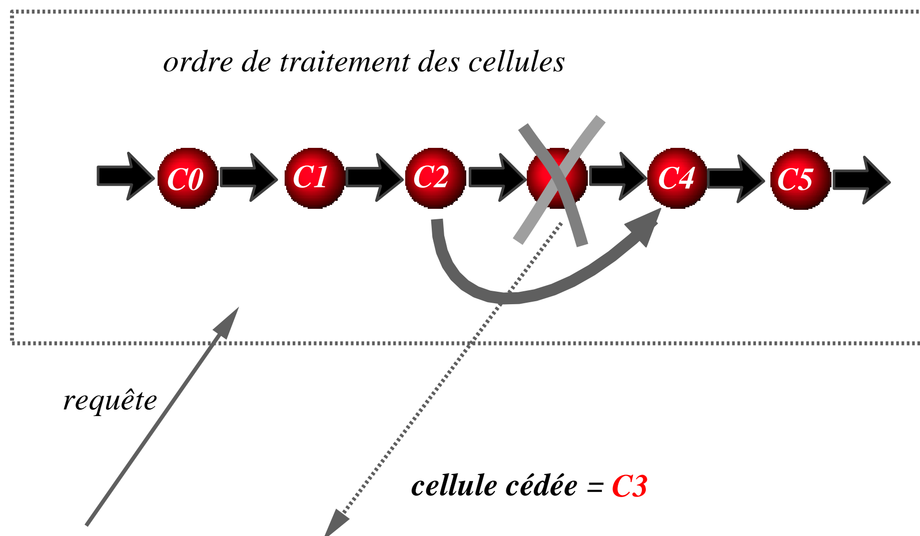


FIG. 6.10 – *Problème de la prévision à long terme et de l'équilibrage de charge dynamique.*

Pour prendre en compte cette modification, chaque processeur P_i maintient une table indiquant les groupes qui sont toujours à sa charge. Ainsi lorsqu'une cellule C est choisie pour collecter (selon l'ordre déterminé par l'algorithme du voyageur de commerce), P_i peut vérifier que C lui appartient toujours.

6.5.5 Terminaison des processus et synchronisation

Notre algorithme de simulation d'éclairage est itératif. Cela signifie que plusieurs étapes de calculs sont nécessaires avant d'atteindre la convergence. A la fin de chaque itération, les processus se synchronisent afin de :

1. déplacer les fichiers (correspondant aux groupes mis à jour) vers le répertoire commun ;
2. tester la convergence de l'algorithme ;
3. terminer le programme en cas de convergence ;

Pour cela, nous reconfigurons les processeurs en anneau de manière dynamique. Un jeton circule sur cet anneau et effectue deux tours. Le premier tour est destiné à synchoniser les processeurs

et le second permet de tester l'avancement de l'algorithme de radiosité et de terminer le programme le cas échéant.

- *Le premier tour du jeton*
  Chaque processeur i calcule une valeur $V_i = \max_{j \in [1,N_i]} |B_j^k - B_j^{k-1}|$, où B_j^k est la radiosité de la maille j à l'itération k et N_i est le nombre de mailles affectées au processeur i. Le jeton est initialisé par le processeur d'identificateur 0 et contient une valeur V_0. Lorsque le processeur 0 a terminé d'effectuer tous ses calculs (collecté l'énergie pour tous ses groupes), alors il envoie le jeton au processeur suivant (c'est à dire le processeur 1). Lorsqu'un processeur i reçoit un jeton, il le traite uniquement lorsqu'il a terminé tous ses calculs pendant l'itération courante. Ce traitement consiste à comparer la valeur du jeton J avec V_i. Si $J < V_i$ alors la nouvelle valeur du jeton sera $J = V_i$. Ainsi, le jeton revient au processeur 0 seulement une fois que tous les processeurs ont terminé leurs calculs et avec une valeur J utilisée pour tester la convergence de l'algorithme de radiosité.

- *Synchronisation*
  Après ce premier tour, les processeurs déplacent les fichiers de leurs répertoires locaux vers le répertoire commun.

- *Second tour du jeton*
  A la fin du premier tour, le processeur 0 compare la valeur de J au seuil de convergence S_c. Si ce seuil est atteint, le calcul est terminé. Le processeur renvoie alors le jeton avec une valeur -1 pour signifier la fin du calcul aux autres processeurs. Lorsqu'un processeur i reçoit le second jeton, il vérifie sa valeur et termine le programme si il vaut -1.

L'algorithme correspondant à cette synchronisation est donné par la figure 6.11.

Dans cet algorithme, la fonction *MisesAJour()* correspond au déplacement des fichiers du répertoire local au processus vers le répertoire commun à tous les processus.

6.5.6 Interblocages et congestion

La gestion des interblocages est primordiale pour tout algorithme parallèle. Un interblocage survient lorsqu'un processeur A attend une donnée provenant d'un autre processeur B. Si le processeur B ne répond pas à la requête et se me lui aussi en attente de données devant provenir du processeur A (ou d'un autre processeur X lui-même en attente), un blocage a lieu.

Notre solution à ce problème est d'utiliser les fonctions de réception non bloquantes de la librairie MPI. Ainsi, lorsqu'un processeur attend un message, il ne reste pas bloqué et continue à traiter les messages qu'il reçoit. Par conséquent, aucune requête ne reste sans réponse et aucun blocage ne peut survenir.

Nous ne sommes pas non plus confrontés aux problèmes de congestion. En effet, les seuls messages envoyés concernent l'équilibrage de charge et la terminaison. Lorsqu'un processeur a calculé l'éclairement de tous les groupes dont il a la charge, il envoie une seule requête à la fois et vers un seul processeur. En conséquence, un seul processeur peut lui répondre par requête envoyée.

Le problème pourrait provenir du nombre de requêtes reçues par un seul processeur (figure 6.12). Considérons le pire des cas, où un seul processeur P est encore en activité, alors que tous les autres ont déjà terminé leurs calculs de simulation d'éclairage. Dans ce cas, le processeur P reçoit un message de chaque processeur. Etant donné la faible taille des messages (quelques dizaines d'octets), les ressources machines ne sont pas source de congestion. Le processeur P traite toutes ces requêtes les unes après les autres dans leur ordre d'arrivée. Le traitement de

```
booléen Terminaison() {
        Booléen convergence = FAUX; ;

        /* Envoi du premier jeton */
        si (ProcessusCourant == 0) alors EnvoyerJeton(Energie) ;

        EnergieGlobale = AttendreJeton();
        si (EnergieGlobale < Energie) alors EnergieGlobale = Energie ;

        /* test de convergence par le processus 0 ou renvoi du jeton */
        si (ProcessusCourant ≠ 0) alors EnvoyerJeton(EnergieGloable) ;
        sinon si (EnergieGlobale < SeuilConvergence) EnergieGlobale = −1 ;

        /* déplacement des fichiers dans le répertoire commun */
        MisesAJour() ;

        /* Envoi du second jeton */
        si (ProcessusCourant == 0) alors EnvoyerJeton(EnergieGlobale) ;

        EnergieGlobale = AttendreJeton();
        si (EnergieGlobale == −1) convergence = VRAI ;
        si (ProcessusCourant ≠ 0) alors EnvoyerJeton(EnergieGloable) ;

        retourne convergence ;
}
```

FIG. 6.11 – *Algorithme de terminaison.*

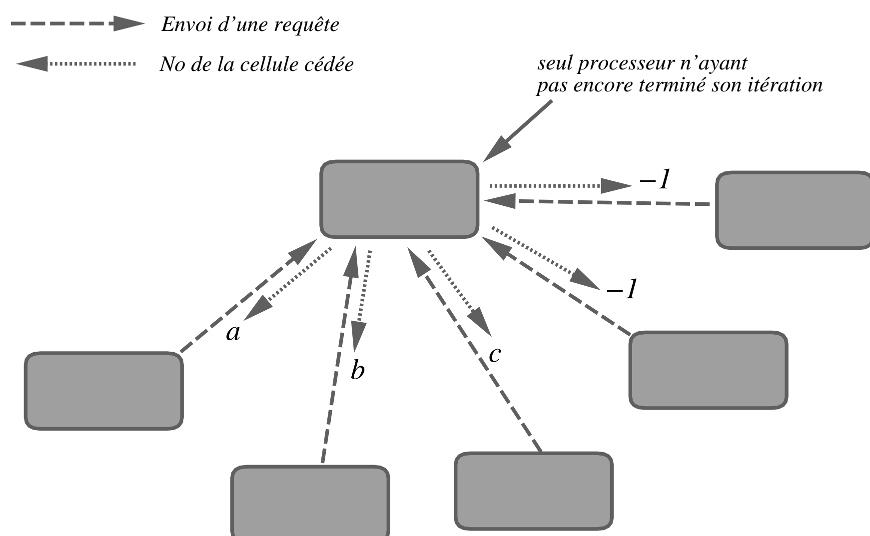


FIG. 6.12 – *Problème de congestion.*

chaque requête étant quasi-instantanné, la perte de temps due à ce traitement est très faible. P cède ses groupes selon la stratégie du “premier arrivé, premier servi”. En d'autres termes, tant qu'il lui reste des groupes, il les cède aux processeurs ayant fait la requête en premier. Les autres processeurs entrent alors dans la phase de synchronisation et se positionnent en attente du jeton (par la fonction $Terminaison$ figure 6.11).

6.5.7 Preuve formelle de la convergence de notre algorithme

Nous avons décomposé la simulation d'éclairage en plusieurs sous-problèmes résolus simultanément par plusieurs processeurs différents. Sur chaque processeur la résolution utilisée est la méthode itérative de Gauss-Seidel. Cependant, l'algorithme suivi ne correspond pas exactement à cette méthode itérative. Nous expliquons ici en quoi elle est différente et montrons que la parallélisation du processus de simulation d'éclairage n'affecte pas la convergence de l'algorithme de radiosité.

Afin d'étudier la convergence des systèmes itératifs induits par la projection de l'équation de luminance, réécrivons le système sous la forme :

$$Ax = b,$$

où x est le vecteur de radiosité à calculer, b est le vecteur des émittances propres et A la matrice du système.

Ce système peut être écrit sous la forme :

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{6.1}$$

Nous pouvons également écrire :

$$x = A^{-1}b.$$

Si A est inversible.

Définition 1
Une matrice A est monotone si elle est inversible et si $A^{-1} \geq 0$ (i.e. si $a_{ij} \geq 0 \; \forall \; i \neq j$).

Théorème 1
Les propriétés (i) et (ii) ou (i) et (iii) entraînent que A est monotone :

(i) $\forall \; i \neq j, a_{ij} \leq 0;$

(ii) $\forall \; i, \sum_{j=1}^{n} a_{ij} > 0;$

(iii) A est inversible et $\forall i, \sum_{j \; 1}^{n} a_{ij} \geq 0.$

Lorsque les fonctions de base sont constantes, la matrice A est de la forme :

$$A = I - N$$

I étant la matrice identité, et N une matrice à coefficients positifs.

La matrice A est inversible puisqu'elle vérifie les points (i) et (ii) du théorème 1.

Nous pouvons donc maintenant rechercher le vecteur x. Posons tout d'abord $x^{(k)}$ tel que :

$$\lim_{k \to \infty} x^{(k)} = x.$$

Nous pouvons ainsi réécrire l'équation (6.1) de la manière suivante :

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_4^{(k)} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{6.2}$$

Avec $a_{ii} = 1 \ \forall \ i \in [0, n]$ et $a_{ij} \leq 0 \ \forall \ i \neq j$.

La méthode de Gauss-Seidel exprime ce système sous la forme suivante où n est le nombre total de mailles et $(k+1)$ est le numéro de l'itération courante :

$$\begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2n} \\ 0 & 0 & 0 & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \\ \vdots \\ x_n^{(k)} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix} \tag{6.3}$$

L'algorithme parallèle mis en œuvre permet à chaque processeur de traiter un groupe de groupes ou bien encore un groupe de mailles à la manière de la méthode itérative de Gauss-Seidel. Chaque maille collecte son énergie de plusieurs autres mailles ayant elle-même collecté soit à l'itération courante $(k+1)$, soit à l'itération précédente (k). La méthode Gauss-Seidel itère selon une réécriture du produit Ax sous la forme :

$$\begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2n} \\ 0 & 0 & 0 & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \\ \vdots \\ x_n^{(k)} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix} \tag{6.4}$$

$$G_1 x^{(k+1)} + G_2 x^{(k)} = b.$$

Chaque processeur L calcule l'éclairement de p mailles d'indices allant de L_1 à L_p. Ainsi, pour un processeur, le système peut être écrit :

$$\begin{pmatrix} \tilde{0} & \tilde{0} & \tilde{0} \\ \vdots & \vdots & \vdots \\ \tilde{0} & M^L & \tilde{0} \\ \vdots & \vdots & \vdots \\ \tilde{0} & \tilde{0} & \tilde{0} \end{pmatrix} \begin{pmatrix} x_1^{k+1} \\ \vdots \\ \vdots \\ \vdots \\ x_n^{k+1} \end{pmatrix} + \begin{pmatrix} \tilde{0} & \tilde{0} & \tilde{0} \\ \vdots & \vdots & \vdots \\ \tilde{A}_1^L & \tilde{0} & \tilde{A}_2^L \\ \vdots & \vdots & \vdots \\ \tilde{0} & \tilde{0} & \tilde{0} \end{pmatrix} \begin{pmatrix} x_1^k \\ \vdots \\ \vdots \\ \vdots \\ x_n^k \end{pmatrix} = \begin{pmatrix} \vdots \\ b_{L_1} \\ \vdots \\ b_{L_p} \\ \vdots \end{pmatrix}$$

Le premier terme de la partie gauche de l'équation correspond aux contributions des mailles attribuées au processeur L et le second terme exprime les contributions calculées par les autres

processeurs à l'itération précédente k.

M^L est une matrice triangulaire inférieure dont les éléments sont $(M^L)_{i,j}$, où $i,j \in [L_1, L_p]$. $\tilde{A}_1^L$ et $\tilde{A}_2^L$ sont composés des éléments a_{ij} de la matrice A. $\tilde{A}_1^L$ est une matrice contenant les éléments $(\tilde{A}_1^L)_{ij} = (a_{ij})$, où $i \in [L_1, L_p]$ et $j \in [1, L_1]$. $\tilde{A}_2^L$ est une matrice contenant les éléments $(\tilde{A}_2^L)_{ij} = (a_{ij})$, où $i \in [L_1, L_p]$ et $j \in [L_p+1, n]$. $\tilde{0}$ sont des matrices carrées contenant des éléments nuls $(\tilde{0})_{ij} = 0$, $i \in [i_1, i_2]$, $j \in [j_1, j_2]$, où $i_1, i_2, j_1, j_2 \in [1, n]$.

Si nous considérons les N processeurs impliqués dans l'algorithme de radiosité parallèle, nous obtenons :

$$\begin{pmatrix} M^1 & \tilde{0} & \tilde{0} \\ \tilde{0} & \ddots & \tilde{0} \\ \tilde{0} & \tilde{0} & M^N \end{pmatrix} \begin{pmatrix} x_1^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix}$$

$$+ \begin{pmatrix} \tilde{0} & \tilde{0} & \tilde{0} \\ A_1 & \tilde{0} & \tilde{0} \\ A_2 & A_3 & \tilde{0} \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ \vdots \\ x_n^{(k)} \end{pmatrix} + \begin{pmatrix} \tilde{0} & A_4 & A_5 \\ \tilde{0} & \tilde{0} & A_6 \\ \tilde{0} & \tilde{0} & \tilde{0} \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ \vdots \\ x_n^{(k)} \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Chaque matrice M^L est associée à un processeur L. Les matrices $A_1, A_2, A_3, A_4, A_5, A_6$ sont composées des matrices A_i^L, où $L \in [1, N], i = 1, 2$.

Nous pouvons réécrire ce système sous une forme condensée si D est inversible :

$$\begin{aligned} & Dx^{(k+1)} - Lx^{(k)} - Ux^{(k)} = b \\ \Leftrightarrow \quad & Dx^{(k+1)} = b + (L+U)x^{(k)} \\ \Leftrightarrow \quad & x^{(k+1)} = D^{-1}b + D^{-1}(L+U)x^{(k)} \end{aligned}$$

où $U \geq 0$ et $L \geq 0$ car $\tilde{A}_i^L \leq 0$.

Notons que D est une matrice triangulaire inférieure et contient des valeurs "1" $(a_{ii} = 1, \forall i)$ sur sa diagonale. Ainsi $det(D) = 1 \neq 0$, ce qui signifie que D est inversible.

Maintenant que nous avons exprimé $x^{(k+1)}$ en fonction de $x^{(k)}$, montrons que notre méthode itérative converge.

Soit $\varepsilon^{(k)}$, l'erreur exprimée par :

$$\varepsilon^{(k)} = x^{(k)} - x$$

Nous avons alors :

$$\begin{cases} (a) & Dx = b + (U+L)x \\ (b) & Dx^{(k+1)} = b + (U+L)x^{(k)} \end{cases}$$

$$\begin{aligned} (b)-(a) \quad \Leftrightarrow \quad & D(x^{(k+1)} - x) = (U+L)(x^{(k)} - x) \\ \Leftrightarrow \quad & D\varepsilon^{(k+1)} = (U+L)\varepsilon^{(k)} \\ \Leftrightarrow \quad & \varepsilon^{(k+1)} = D^{-1}(U+L)\varepsilon^{(k)} \end{aligned}$$

Et nous avons également l'équivalence :

$$\forall \varepsilon^{(0)}, \lim_{k \to \infty} (D^{-1}(U+L))^k = 0 \Leftrightarrow \rho(D^{-1}(U+L)) < 1$$

Nous devons maintenant prouver que $\rho(D^{-1}(U+L)) < 1$, $\rho()$ étant le rayon spectral.
Montrons que $\rho(D^{-1}(U+L)) < 1$ en utilisant le corollaire 5.6 de la page 125 de Young [63] (ce corrolaire a également été utilisé par Funkhouser dans [60]).

Corollaire 1
Si une Matrice A est monotone et décomposable de deux manières différentes $A = Q_1 - R_1$ et $A = Q_2 - R_2$ et si $R_2 < R_1$ alors $\rho(Q_2^{-1}R_2) \leq \rho(Q_1^{-1}R_1)$.

Puisque D est une matrice triangulaire inférieure contenant des valeurs "1" sur sa diagonale et des valeurs négatives ou nulles ailleurs ($D_{ij} = -\rho_i F_{ij}, \rho_i$ étant les réflectances et F_{ij} les facteurs de formes entre les mailles i et j), nous avons alors :

$$A = I - N = D - (U + L) = (I - L') - (U + L)$$

D'où :

$$U + L = N - L' \leq N$$

avec $N \geq 0$ et $L' \geq 0$.
Ecrivons $Q_1 = I$, $R_1 = N$ and $Q_2 = D$, $R_2 = (U + L)$. En utilisant le corollaire 1 nous obtenons $\rho(D^{-1}(U+L)) \leq \rho(I^{-1}N) = \rho(N) < 1$ □.

Par conséquent, notre algorithme de type groupe itératif basé sur la méthode de Gauss-Seidel converge.

6.6 MPI et mise en œuvre

La mise en œuvre de notre algorithme parallèle a été réalisée à l'aide de la bibliothèque MPI (Message Passing Interface). Cette bibliothèque fournit de nombreuses primitives permettant d'envoyer ou de recevoir des messages de manière simplifiée tout en conservant des aspects plus bas-niveau des transmissions de données tels que les envois ou réceptions de messages bloquants ou non bloquants par exemple.

L'algorithme parallèle présenté dans ce document n'occasionne que très peu de messages entre les différents processus. En effet, seulement deux types de messages sont envoyés : les messages concernant l'équilibrage de charge et le jeton permettant de synchroniser les processus et tester la convergence et la terminaison de l'algorithme de radiosité.

Cette section présente brièvement les fonctionnalités de base MPI. Elle ne constitue en aucun cas une documentation exhaustive, mais donne un aperçu de la bibliothèque tout en exposant les primitives utilisées dans notre approche.

6.6.1 MPI : Message Passing Interface

Avec l'environnement MPI, chaque processus est reconnaissable par un identificateur. Cet identificateur est un entier i, $i \in [0..N]$, N étant le nombre de processus prenant part aux calculs.

Pour chaque processus, il est possible de connaître son identificateur et le nombre total de processus.

6.6.2 Initialisations

Pour initialiser les différents processus pris en considération dans un programme parallèle, MPI nécessite une initialisation. Cette initialisation est effectuée automatiquement à l'aide de la procédure :

$$MPI_Init(intac, char * av)$$

Cette procédure initialise le processus et l'ajoute à l'environnement MPI. Ainsi, les communications peuvent être établies entre tous les processus.

Pour détruire l'environnement MPI, la terminaison d'un processus doit être précédé d'un appel à la procédure :

$$MPI_Finalize()$$

Entre ces deux opérations, toutes les fonctionnalités de la bibliothèque MPI sont disponibles et peuvent être exécutées. Par exemple, pour connaître l'identificateur du processus courant :

$$MPI_Comm_rank(MPI_COMM_WORLD, int * mpi_rank)$$

Ou le nombre total de processus :

$$MPI_Comm_size(MPI_COMM_WORLD, int * mpi_size)$$

Le code de notre programme parallèle est écrit en conséquence (figure 6.13)

6.6.3 Envoi de message

L'approche parallèle proposée dans ce document nécessite uniquement des envois ou des receptions de messages classiques, non bloquants.

Le programme donné par la figure 6.14 réalise un envoi bloquant de message entre deux processus d'un environnement MPI. Le processus émetteur est bloqué tant que le récepteur n'a pas effectivement reçu le message (appel de la fonction *MPI_Recv*).

Ce programme, exécuté sur deux processus (ou processeurs) différents réalise l'envoi d'un message contenant le texte "Message". L'envoi et la reception sont synchrones. Ainsi, tant que le processus d'identificateur 1 n'a pas recu le message, il reste en attente. De la même manière, tant que le message n'a pas été correctement reçu, le processus d'identificateur 0 reste bloqué.

Pour envoyer un message ou recevoir un message sans rester en attente, MPI fournit les routines suivantes :

- Envoi d'un message en mode non bloquant :

```
int MPI_Irecv( buf, count, datatype, source, tag, comm, request );
    void            *buf;
    int             count;
    MPI_Datatype    datatype;
    int             source;
    int             tag;
    MPI_Comm        comm;
    MPI_Request     *request;
```

```
main(int argc, char **argv) {
    int MpiSize;  /* Nombre de processus de l'environnement MPI */
    int MpiRank;  /* Identificateur du processus courant */
    MpiStatus status;  /* Etat de la reception et erreurs */
    GroupeGroupes C;  /* Groupe de groupes dont le processus dispose */

    /* Initialisation de l'environnement MPI */
    MPI_Init(&argc, &argv) ;

    /* Initialisation de l'environnement MPI */
    MPI_Comm_size(MPI_COMM_WORLD, &MpiSize) ;

    /* Initialisation de l'environnement MPI */
    MPI_Comm_rank(MPI_COMM_WORLD, &MpiRank) ;

    /* Recuperer les groupes dont le processus a la charge */
    C = RecupereListeGroupes(MpiRank) ;

    /* Effectuer la simulation d'éclairage sur C */
    RadiositeParallele(C) ;

    /* Terminaison de l'environnement MPI */
    MPI_Finalize() ;
}
```

FIG. 6.13 – *Initialisation et terminaison de l'environnement MPI.*

```
main(int argc, char **argv) {
    int MpiSize; /* Nombre de processus de l'environnement */
    int MpiRank; /* Identificateur du processus */
    char Message[60]; /* Message recu */
    MpiStatus status; /* Etat de la reception et erreurs */

    /* Initialisation de l'environnement MPI */
    MPI_Init(&argc, &argv); ;

    /* Initialisation de l'environnement MPI */
    MPI_Comm_size(MPI_COMM_WORLD, &MpiSize); ;

    /* Initialisation de l'environnement MPI */
    MPI_Comm_rank(MPI_COMM_WORLD, &MpiRank); ;

    switch(mpi_rank) {
        case 0: {
            MPI_Send("Message", 50, MPI_CHAR, 1, 0,
                MPI_COMM_WORLD, &status) ;
            printf("Ok, message envoye...") ;
            break ;
        }
        case 1: {
            MPI_Recv(Message, 50, MPI_CHAR, 0, 0,
                MPI_COMM_WORLD, &status) ;
            printf("Ok, message recu %s...", Message) ;
            break ;
        }
    }

    /* Terminaison de l'environnement MPI */
    MPI_Finalize(); ;
}
```

FIG. 6.14 – *Envoi et reception de messages bloquants avec MPI.*

– Réception d'un message en mode non bloquant :

```
int MPI_Isend( buf, count, datatype, dest, tag, comm, request );
     void            *buf;
     int             count;
     MPI_Datatype    datatype;
     int             dest;
     int             tag;
     MPI_Comm        comm;
     MPI_Request     *request;
```

– Etat d'une requête (p. ex. si le message attendu est bien arrivé) :

```
int MPI_Test ( request, flag, status );
     MPI_Request  *request;
     int          *flag;
     MPI_Status   *status;
```

– Rendre une requête bloquante et attendre que le message soit bien arrive ou bien envoyé :

```
int MPI_Wait ( request, status );
     MPI_Request  *request;
     MPI_Status   *status;
```

Notre algorithme utilisant exclusivement des réceptions et envois non bloquants, aucune partie de notre code n'est donné dans ce paragraphe. En effet, les procédures décrites dans les sections suivantes illustreront très largement l'utilisation de ces routines.

6.6.4 Compilation et exécution

– *Compilation*
  Si le fichier correspondant à ce programme se nomme ping.c, la compilation est la suivante :

$$cc\ ping.c - lmpi$$

  Un shell-script permet de simplifier la compilation évitant de rechercher les librairies dans l'arborescence UNIX : mpicc. Avec ce script, la compilation est la suivante :

$$mpicc\ ping.c$$

– *Execution*
  Pour exécuter un programme mpi sur plusieurs machines à la fois, plusieurs possibilités sont offertes à l'utilisateur. La plus simple d'entre elles consiste à exécuter le programme à l'aide de la commande *mpirun* :

$$mpirun{-}np < nb_procs > -machinefile < nom_fichier >< programme > [options_programme]$$

  – *procs* représente le nombre de processus à exécuter.

  – *fichier* contient les noms des machines sur lesquelles exécuter les processus.

- *executable* est le nom du programme parallèle à exécuter.

- *options* correspond aux options du programme à exécuter.

Il est aussi possible de profiter d'une fenêtre comportant un certain nombre d'informations en utilisant la commande *xmpi*.

6.6.5 Un exemple : notre équilibrage de charge

Lorsqu'un processus a terminé une itération (i.e. effectué une collecte d'énergie pour tous les polygones de toutes les groupes dont il a la charge), il demande tour à tour à chaque processeur une nouvelle tâche (i.e. une nouvelle groupe).

Cette opération implique l'envoi et la réception d'un message vers chaque processeur jusqu'à ce qu'un numero de groupe lui soit donné. Cependant, pour éviter les interbloquages, la procédure ne peut rester en attente.

Le code correspondant à la demande d'un groupe à d'autres processeurs est décrit par la figure 6.15.

6.7 Résultats

Cette section donne quelques résultats obtenus à l'aide de notre programme parallèle et pour une scène composée de 3 étages et contenant 57,786 polygones dont environ 2000 ont une émittance propre. Elle contient 615 cellules et 12942 *groupes*. Le nombre de mailles est de 360000 et le nombre de liens est égal à 3.7 millions. Notre algorithme parallèle a été testé sur plusieurs machines différentes connectées à notre réseau local (de type ethernet). Les caractéristiques des machines sont données par le tableau 6.1.

Nom	Type de machine	Processeur	Memoire	Fréquence
SGI1	SGI	R10000	380 Mb	195 MHz
SGI2	SGI	R5000	128 Mb	180 MHz
SGI3	Machine parallèle SGI	4 x R10000	380 Mb	195 MHz
SUN1	SUN	UltraSpark	256 Mb	200 MHz
SUN2	SUN	SuperSpark	128 Mb	100 MHz

TAB. 6.1 – *Caractéristiques des machines utilisées.*

Les tests effectués ont donné lieu aux résultats présentés dans le tableau 6.2

Combinaison	Temps d'exécution	Accélération
1 SUN1	16920 mn	1
2 SUN1	6552 mn	2.5
3 SUN1	5394 mn	3.2
4 SUN1	4338 mn	3.9
1 SGI3	4594 mn	3.7
4 SUN1 + 1 SGI1 + 3 SGI2	2880 mn	5.9
4 SUN1 + 1 SUN2 + 1 SGI1 + 4 SGI2	2160 mn	7.8

TAB. 6.2 – *Résultats.*

```
int ask_task_message(int *requete, int TailleRequete) {
        extern int **USED_CELLS;
        MPI_Status sta;
        MPI_Request req, reqsend;
        extern int mpi_rank, mpi_size;
        int fla, recu, i, attente;

        /* J'essaie de recuperer une tâche sur l'un des autres processeurs */
        for(i = 0, recu = -1; i ¡mpi_size && recu == -1; i ++) {
                /* Sauf de moi-meme */
                if(i ≠ mpi_rank) {

                        /* Je lance la requete */
                        MPI_Isend(requete, TailleRequete, MPI_INT, i,
                            TASK_STEALING_FLAG, comm, &reqsend);
                        MPI_Wait(&reqsend, &sta);

                        /* J'attends le résultat de la requete : un numero de groupe */
                        MPI_Irecv(&recu, 1, MPI_INT, i, TASK_GIVING_FLAG,
                            comm, &req);

                        /* En evitant les inter-blocages... */
                        attente = 1;
                        while(attente) {

                                /* Je regarde si une requete est arrivee d'un autre processeur... */
                                give_task_message();

                                /* Je regarde si j'ai eu une reponse a ma requete */
                                MPI_Test(&(req), &fla, &sta);
                                if(sta.MPI_SOURCE == i &&
                                    sta.count > 0 && sta.MPI_ERROR == 0 &&
                                    sta.MPI_TAG == TASK_GIVING_FLAG) {
                                        MPI_Wait(&req, &sta);
                                        attente = 0;
                                }
                        }
                        /* Je m'approprie definitvement le groupe recu */
                        AjouterDansTableau(USED_CELLS[mpi_rank], recu);
                }
        }
        /* Je retourne le numero de groupe recu */
        return recu;
}
```

FIG. 6.15 – *Demande de groupe par un processeur.*

Les accélérations données dans cette table ont été calculées par rapport au temps de référence obtenu pour une machine SUN UltraSpark qui est l'une de nos machines les plus performantes. Nous pouvons remarquer que les accélérations obtenues avec deux et trois processeurs dépassent l'accélération idéale. Ceci peut être expliqué par le fait que notre algorithme implique de nombreux transferts de données entre le disque et la mémoire, sollicitant par conséquent de manière importante le *cache* et les mécanismes de *swap*. Or la répartition des données sur plusieurs processeurs permet :

- d'augmenter les zones de *cache* et de *swap* ;
- de réduire leur sollicitation pour chaque processeur qui ne traite alors qu'une petite quantité de données.

Pour la machine parallèle SGI3 les temps de calcul sont équivalents à ceux obtenus par quatre machines de type SUN1. Cela peut être expliqué par le fait que la machine parallèle SGI3 n'est pas équipée d'accès-disques parallèles et ceci crée un goulot d'étranglement au niveau de l'éxécution du programme.

6.8 Discussion

Comme nous l'avons vu, notre algorithme parallèle de radiosité hiérarchique peut être mis en œuvre facilement sur une machine parallèle ou un réseau de machines grâce à la librairie MPI. Comme les structures de données utilisées sont stockées sur un disque commun à tous les processeurs, les performances de l'algorithme dépendent très fortement du temps nécessaire aux nombreux accès au disque. Pour améliorer ces performances, il est nécessaire de réduire ces temps. Pour certaines machines parallèles, des systèmes matériels ont été mis en place afin de permettre un accès parallèle aux disques. Malheureusement, nous n'avons pu faire de tests avec un tel équipement.

Une solution logicielle à ce problème est de :

1. placer les répertoires locaux des processeurs sur des disques locaux ;
2. distribuer la base de données sur plusieurs disques différents et y accéder via NFS.

Par conséquent, plusieurs groupes pourraient être lus simultanément sur le répertoire commun et les fichiers pourraient être mis à jour en parallèle sur des disques différents.

Notons que l'utilisation de MPI n'est pas pénalisante pour notre algorithme car le nombre de messages transitant entre les processeurs est très réduit et la taille des messages est très faible. Afin d'éviter les problèmes de mémoire, seulement une petite partie de la scène est stockée en mémoire à au cours des calculs. Cette petite partie est déterminée à l'aide de nos techniques de découpage et de regroupement de surfaces. Dans le but de réduire les accès au disque, nous utilisons notre stratégie du voyageur de commerce décrite dans le chapitre 5. En effet, cette stratégie est la meilleure de celles que nous avons mises en œuvre (algorithmes gloutons, retour-arrière, etc.). La cohérence des données est assurée via l'utilisation de plusieurs répertoires situés sur un disque commun à tous les processeurs.

Pour les nombreux tests que nous avons effectués, nos algorithmes de synchronisation, d'équilibrage de charge statique et dynamique et de terminaison semblent être efficaces.
Enfin, nous avons donné une preuve mathématique de la convergence de notre algorithme de radiosité hiérarchique basé sur la méthode itérative de Gauss-Seidel.

Conclusion

Le travail réalisé au cours de cette thèse concerne la mise en œuvre d'algorithmes séquentiel et parallèle de simulation d'éclairage pour des environnements complexes. Ces algorithmes nécessitent une étape de prétraitement au cours de laquelle les environnements sont découpés en plusieurs régions appelées *cellules*. Cependant, les techniques de découpage déjà existantes sont toutes basées sur une subdivision binaire de l'espace (BSP) et nous paraissent mal adaptées à des environnements architecturaux quelconques. En effet, comme la topologie de l'environnement n'est pas prise en compte par les algorithmes, les cellules résultant du découpage sont trop nombreuses et ne correspondent pas aux pièces ou aux couloirs. Par conséquent les ouvertures sont également trop nombreuses et ceci entraîne des surcoûts de calcul. D'autre part, si la méthode BSP est relativement facile à mettre en œuvre dans le cas d'environnements axiaux, en revanche, le traitement de bâtiments quelconques est beaucoup plus coûteux et plus complexe à mettre en œuvre. Par ailleurs tous les critères de découpage utilisés pour des environnements axiaux ne sont plus valides pour des environnements quelconques. Par exemple, le critère d'occlusion ne peut plus être appliqué car les polygones occlusifs peuvent avoir des orientations quelconques et l'utilisation d'une liste triée selon les trois axes du repère de la scène n'a plus aucun sens. C'est pourquoi nous avons proposé une nouvelle méthode de découpage, basée sur des modèles génériques de pièces, permettant de traiter tout type d'environnement architectural. Le résultat de ce découpage est un graphe de visibilité ainsi qu'un ensemble de *cellules*.

Nous avons également présenté un nouvel algorithme de regroupement de polygones basé sur une méthode de classification de type *nuées dynamiques* employée habituellement en analyse de données. Notre objectif était de réduire les données stockées en mémoire au cours de la simulation en affinant les calculs de visibilité. Cet algorithme est plus précis car il n'utilise pas de volumes englobants. De plus, il est rapide puisqu'il utilise les cellules et le graphe de visibilité. Il fournit un autre graphe de visibilité ainsi qu'un ensemble de groupes de polygones appelés groupes, utilisés par nos algorithmes de simulation d'éclairage séquentiel et parallèle.

Notons que nos algorithmes de simulation d'éclairage peuvent opérer soit à l'aide de cellules (avec graphe de visibilité inter-cellules) ou de groupes (avec graphe de visibilité inter-groupes). Ceci est possible car les fichiers résultant de nos programmes de structuration et de regroupements de polygones contiennent le même type d'information.

Concernant notre algorithme séquentiel, les calculs sont effectués seulement pour un sous-ensemble de polygones à la fois. Les informations géométriques et photométriques nécessaires à la représentation des objets de la scène sont stockées sur disque. La simulation de la propagation de la lumière est alors considérée comme la composition de tâches élémentaires consistant à (i) charger en mémoire les informations nécessaires à la simulation, (ii) effectuer les calculs de radiosité, (iii) remettre à jour la base de données sur le disque. Néanmoins, afin de réduire les nombreux échanges d'information entre le disque et la mémoire, il est nécessaire d'ordonner les calculs de manière efficace. Pour cela, nous avons proposé de nouvelles stratégies d'ordonnancement des calculs reposant sur une prédiction des coûts de ces échanges d'informations à court, moyen ou long terme.

La version parallèle de notre algorithme de radiosité utilise l'environnement de programmation MPI (Message Passing Interface). Cela permet d'exécuter le programme sur un réseau hétérogène de machines ou une machine parallèle ou les deux à la fois. Toutes les données sont stockées sur un disque commun à tous les processeurs afin de réduire le nombre de messages et leur taille. Chaque processeur effectue les calculs pour un ensemble de régions suivant notre stratégie d'ordonnancement la plus efficace utilisant un algorithme du voyageur de commerce. Un mécanisme d'équilibrage de charge dynamique selon une technique de task stealing (vol de tâche) permet d'éviter que certains processeurs restent inactifs au cours des calculs. Enfin, la terminaison de l'algorithme est détectée à l'aide d'une reconfiguration dynamique des processeurs en anneau qui permet également la gestion de la convergence de l'algorithme.

Nous avons également conçu un outil permettant de modéliser des environnements complexes. Grâce à cet outil intéractif utilisant les librairies Motif et OpenGL, l'utilisateur peut aisément modéliser des environnements complexes et demander à cet outil de réaliser la structuration.

Enfin, un dernier outil permet de visualiser les environnements complexes avant, pendant et après les calculs de simulation d'éclairage à l'aide des cellules du graphe de visibilité.

Dans cette thèse, nous nous sommes attachés à résoudre les problèmes inhérents à la simulation d'éclairage pour des environnements architecturaux complexes. Nous souhaiterions cependant généraliser notre approche afin de pouvoir l'utiliser pour tout type d'environnement complexe (urbains ou ruraux par exemple). Des études complémentaires sont par conséquent nécessaires pour établir une correspondance entre les topologies des différents environnements. Nous pensons qu'il est possible d'adapter notre algorithme de découpage à des environnements urbains en modifiant nos règles d'extraction de cellules. Néanmoins, le problème reste encore largement ouvert notamment en ce qui concerne les calculs de visibilité.

Une autre perspective de notre travail est relative à notre algorithme de simulation d'éclairage parallèle. En effet, les structures de données utilisées sont stockées sur un disque commun à tous les processeurs. Or les performances de l'algorithme dépendent très fortement du temps nécessaire aux nombreux accès au disque. Pour améliorer ces performances, il est nécessaire de réduire ces temps. Pour certaines machines parallèles, un système matériel permet un accès parallèle au disque. Nous n'avons pu faire de tests avec un tel équipement car il n'est pas disponibles dans notre laboratoire. En outre, une solution logicielle à ce problème serait de :

1. placer les répertoires locaux des processeurs sur des disques locaux ;
2. répartir le répertoire commun sur plusieurs disques différents.

De cette manière, plusieurs groupes pourraient être lus simultanément sur le répertoire commun et les fichiers pourraient être mis à jour en parallèle sur des disques locaux différents.

Bibliographie

[1] Seth Teller & Celeste Fowler & Thomas Funkhouser & Pat Hanrahan. Partitioning and ordering large radiosity computations. In *Computer Graphics Proceedings, Annual Conference Series*, pages 443–450, 1994.

[2] Michael F. Cohen & Donald P. Greenberg. The hemi-cube: A radiosity solution for complex environments. In Computer Graphics, editor, *Proceedings of SIGGRAPH '85*, volume 19, pages 31–40, March 1985.

[3] Michael F. Cohen & Shenchang Eric Chen & John R. Wallace & Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics*, 22(4):75–84, August 1988.

[4] Pat Hanrahan & David Salzman & Larry Aupperle. A rapid hierarchical radiosity algorithm for unoccluded environments. *Computer Graphics*, 25(4):197–205, July 1991.

[5] E. Languénou & K. Bouatouch & P. Tellier. An adaptive discretization method for radiosity. In *Proceedings of Eurographics '92*, pages C205–C216, Cambridge, UK, September 1992.

[6] Dani Lischinski & Filippo Tampieri & Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. In *Computer Graphics Proceedings, Annual Conference Series*, 1993.

[7] K. Bouatouch and S. N. Pattanaik. Discontinuity meshing and hierarchical multiwavelet radiosity. *Graphics Interface'95*, 1995.

[8] C.M. Goral & K.E. Torrance & D.P. Greenberg & B. Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222, July 1984.

[9] John R. Wallace Michael F. Cohen. *Radiosity and realistic image synthesis.* Academic Press, 1993.

[10] J. M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculation.* PhD thesis, University of north Carolina at Chapel hill, 1990.

[11] Frederick P. Brooks John M. Airey, John H. Rohlf. Towards image realism with interactive update rates in complex virtual building environments. *ACM Siggraph*, pages 41–50, May 1990.

[12] Seth Jared Teller. *Visibility Computations in Density Occluded Polyhedral Environments.* PhD thesis, University of California at Berkeley, 1992.

[13] P. V. C. Hough. Methods and means for recognizing complex patterns. *United States Patent*, 1962.

[14] Henri Maitre. Un panorama de la transformation de hough. *Traitement du signal*, 2(4):305–317, 1985.

[15] Andries Van Dam. James D. Foley. *Fundamentals of Interactive computer graphics.* Addison-Wesley, 1982.

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

[17] K. Bouatouch & M. Madani & T. Priol & B. Arnaldi. A new algorithm of space tracing using a csg model. Eurographics'87, 1987.

[18] Jean-Philippe Thirion. Tries: Data structures based on binary representation for ray tracing. In C.E. Vandoni and D.A. Duce, editors, *Eurographics '90*, pages 531–541. Eurographics Association, Elsevier Science Publishers, 1990.

[19] A. Fournier and P. Poulain. A ray tracing accelerator based on a hierarchy of 1d sorted lists. *Graphics Interface'93*, 1993.

[20] Harry Plantinga & Charles R. Dyer. Visibility, occlusion, and the aspect graph. *International Journal of Computer Vision*, 5(2):137–160, 1990.

[21] Harry Plantinga. Conservative visibility preprocessing for efficient walkthroughs of 3d scenes. *Graphics Interface'93*, pages 166–173, 1993.

[22] C. B. Jones. A new approach to the 'hidden line' problem. *The Computer Journal*, 14(3):232–271, March 1971.

[23] Seth Teller & Pat Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics Proceedings, Annual Conference Series*, pages 239–246, 1993.

[24] Seth J. Teller & Carlo H. Sequin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics*, 25(4):61–69, July 1991.

[25] C. Séquin T. Funkhouser, S. Teller and D. Khorramabadi. The uc berkeley system for interactive visulalization of large architectural models. *Presence*, 5(1):13–44, 1996.

[26] Thomas A. Funkhouser & Carlo H. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Computer Graphics Proceedings, Annual Conference Series*, pages 247–254, August 1993.

[27] E. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 1987.

[28] G. Saporta. *Probabilités, analyse des données et statistiques.* ed. Technip 1990, 1990.

[29] D. Meneveaux E. Maisel F. Coudret K. Bouatouch. Structuration de scènes architecturales complexes en vue de simulation d'éclairage. Technical report, INRIA/IRISA, 1996.

[30] F. Coudret. Structuration de scènes complexes axiales pour l'optimisation des calculs de visibilité. Technical report, IRISA, Rapport de D.E.A., 1995.

[31] D. Meneveaux, E. Maisel, and K. Bouatouch. A new partitioning method for architectural environments. *To appear in Journal of Visualization and Computer Animation*, 1998.

[32] D. Meneveaux & E. Maisel & K. Bouatouch. A new partitioning method for architectural environments. Technical Report 3148, INRIA, April 1997.

[33] F. Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and objects clusters. *IEEE Transaction On Graphics*, 1(3), 1995.

[34] Brian Smits & James Arvo & Donald Greenberg. A clustering algorithm for radiosity in complex environments. In *Computer Graphics Proceedings, Annual Conference Series*, pages 435–442, 1994.

[35] A. J. F. Kok. Grouping patches in progressive radiosity. In *Fourth Eurographics Workshop on rendering*, pages 221–231, Paris, France., 1994.

[36] A. J. F. Kok. *Ray Tracing and Radiosity Algorithms for Photorealistic Image Synthesis.* Delft University Press, 1994.

[37] Hau Xu Qun-Sheng Peng and You-Dong Liang. Accelerated radiosity method dor complex environments. *Eurographics Workshop*, pages 51–61, 1989.

[38] Holly Rushmeier Charles Patterson and Aravindan Veerasamy. Geometric simplification for indirect illumination calculations. *Graphics Interface'93*, 1993.

[39] A. Glassner. Spacetime ray tracing for animation. *IEEE Computer Graphics and Application*, 8:60–70, 1988.

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

[41] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. In *IEEE Computer Graphics and Applications*, volume 7, pages 14–20, May 1987.

[42] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. *ACM SIGGRAPH'86.*, pages 269–278, 1986.

[43] D. J. Hall and G. H. Ball. Isodata a novel method of data analysis and pattern classification. Technical Report 5 RI project 5533, Stanford Research Institute, CA, USA, 1965.

[44] Satyan Coorg and Seth Teller. Temporally coherent conservative visibility. *ACM Symposium on Computational Geometry*, May 1996.

[45] Satyan Coorg and S. Teller. Real-time occlusion culling for models with large occluders. *ACM Symposium on Interactive 3D graphics.*, 1997.

[46] S. Teller. Computing the antipenumbra of an area light source. In *SIGGRAPH'92*, 1992.

[47] F. Durand G. Drettakis and C. Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility tool. *SIGGRAPH'97*, 1997.

[48] F. Sillion and G. Drettakis. Feature-based control of visibility error: A multi-resolution clustering algorithm for global illumination. *ACM SIGGRAPH'95*, 1995.

[49] Georges Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojection. *ACM SIGGRAPH'94*, 1994.

[50] Kevin Weiler & Peter Atherton. Hidden surface removal using polygon area sorting. In Computer Graphics, editor, *Proceedings of SIGGRAPH '77*, volume 11, pages 214–222, 1977.

[51] H. Fuchs. On visible surface generation by a priori tree structures. *ACM SIGGRAPH*, 1980.

[52] Norman Chin & Steven Feiner. Near real-time shadow generation using bsp trees. *Computer Graphics*, 23(3):99–106, July 1989.

[53] Bruce F. Naylor. Partitioning tree image representation and generation from 3d geometric models. In *Graphics Interface '92*, pages 201–212, 1992.

[54] Yiorgos Chrysanthou and Mel Slater. Shadow volume bsp trees for computation of shadows in dynamic scenes. *Symposium ACM on Interactive 3D Graphics*, pages 45–49, 1995.

[55] Chris Georges David Luebke. Portals and mirrors: Simple, fast evaluation of potentially visible sets. *Symposium on Interactive 3D Graphics*, pages 105–106, 1995.

[56] F. Sillion. Clustering and volume scattering for hierarchical radiosity calculations. *5th Eurographics Workshop on Rendering*, 1994.

[57] Eric A. Haines & John R. Wallace. Shaft culling for efficient ray-cast radiosity. In *Proceedings of 2nd Workshop on Rendering*, pages 122–138, May 1991.

[58] D. Meneveaux and K. Bouatouch. Memory management schemes for radiosity computation in complex environments. Technical Report 3149, INRIA, 1997.

[59] D. Meneveaux, K. Bouatouch, and E. Maisel. Memory management schemes for radiosity computation in complex enviroments. In *Computer Graphics International*, 1998.

[60] Thomas Funkhouser. Coarse-grained parallelism for hierarchical radiosity using group iterative methods. *ACM SIGGRAPH'96 proceedings*, pages 343–352, August 1996.

[61] C C Feng and S N Yang. A parallel hierarchical radiosity for complex scenes. *Parallel Rendering Symposium*, 1997.

[62] B. Arnaldi, T. Priol, L. Renambot, and X. Pueyo. Visibility masks for solving complex radiosity computations on multiprocessors. Technical Report 1055, IRISA, 1996.

[63] D. M. Young. *Iterative solution of large linear systems.* Computer science and applied mathematics. Academic Press, New York, 1971.

Resumé : *Effectuer des calculs d'illumination globale pour des environnements complexes et les visualiser de manière interactive demeure un problème difficile en synthèse d'images. En effet, la radiosité hiérarchique est un processus très coûteux en termes de temps de calcul et de ressources mémoire, même pour des scènes de complexité moyenne. Par conséquent, dans le cas d'environnements complexes, une étape de précalcul est nécessaire. Ce précalcul consiste à découper la scène en plusieurs régions (appelées cellules) et évaluer les relations de visibilité entre ces régions. La méthode de découpage (ou de structuration) que nous proposons est inspirée de l'analyse d'images et consiste à mettre en correspondance des modèles génériques de cellules avec des éléments géométriques déduits de la scène. Comparée à la technique de subdivision binaire de l'espace, cette méthode fournit des résultats beaucoup plus convaincants car le nombre de cellules obtenues est plus faible et par conséquent les calculs de visibilité et de simulation d'éclairage se trouvent simplifiés. A chaque étape de la simulation d'éclairage, seule une petite partie de la scène réside en mémoire. Les informations géométriques et photométriques nécessaires à la représentation des objets de la scène sont stockées sur disque. La simulation de la propagation de la lumière est alors considérée comme la composition de tâches élémentaires consistant à (i) charger en mémoire les informations nécessaires à la simulation, (ii) effectuer les calculs de radiosité, (iii) remettre à jour la base de données sur le disque. Néanmoins, afin de réduire les nombreux échanges d'information entre le disque et la mémoire, il est nécessaire d'ordonner les calculs de manière efficace. Pour cela, nous proposons plusieurs stratégies d'ordonnancement des calculs reposant sur une prédiction des coûts de ces échanges d'informations à court, moyen ou long terme. Ces stratégies utilisent les connaissances relatives à la structuration de la scène en cellules et les relations de visibilité qui existent entre elles. Nous avons mis en œuvre une version parallèle de cet algorithme à l'aide de l'environnement de programmation MPI (Message Passing Interface). Dans ce cas, toutes les données sont stockées sur un disque commun à tous les processeurs afin de réduire le nombre de messages et leur taille. Chaque processeur effectue les calculs pour un ensemble de régions selon les mêmes stratégies d'ordonnancement que l'algorithme séquentiel. Un mécanisme d'équilibrage de charge dynamique suivant une technique de vol de tâche (task stealing) permet d'éviter que certains processeurs restent inactifs au cours des calculs. Enfin, la terminaison de l'algorithme est réalisée à l'aide d'une reconfiguration dynamique des processeurs en anneau qui permet également la gestion de la convergence de l'algorithme.*

Mots-clés : *Simulation d'éclairage, Radiosité, géométrie algorithmique, environnements complexes, ordonnancement, algorithmique parallèle.*