



Doctorat ParisTech

THÈSE

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Signal et Images »

présentée et soutenue publiquement par

Gilles LAURENT

le 12 novembre 2018

Modèles d'approximation pour l'éclairage global temps réel de scènes virtuelles haute résolution

Directeur de thèse : **Tamy Boubekeur**
Encadrant industriel : **Sébastien Bloche**

Jury

M. Nicolas Holzschuch, DR, LJK, INRIA Grenoble-Rhône-Alpes

Rapporteur

M. Mathias Paulin, PR, IRIT, Université Paul Sabatier

Rapporteur

M. Jean-Philippe Farrugia, MCF, LIRIS, Univ. Claude Bernard Lyon I

Examinateur

Mme Florence Tupin, PR., LTCI, Telecom ParisTech

Examinaterice

M. Sébastien Bloche, R&D Visualization Director, Dassault Systèmes

Encadrant

M. Tamy Boubekeur, PR, LTCI, Télécom ParisTech

Directeur

THÈSE

TELECOM ParisTech
école de l'Institut Mines-Télécom - membre de ParisTech

Modèles d'approximation pour l'éclairage global temps réel de scènes virtuelles haute résolution

Gilles LAURENT

RÉSUMÉ

Dans cette thèse nous nous intéressons à une problématique importante de la synthèse d'image, la simulation en temps réel de l'éclairage global. Ainsi, à partir de scènes haute résolution, typiquement visualisées en conception assistée par ordinateur, nous proposons plusieurs modèles d'approximation pour atteindre ce but. Nous nous plaçons ici à la croisée des modèles mathématiques et de la programmation sur processeur graphique. Le premier point est alors nécessaire pour simplifier à fois la géométrie ainsi que le phénomène physique tout en conservant une plausibilité visuelle. Le second pour tenir la charge des traitements de données massives en temps réel.

Pour simuler le transport lumineux en s'abstrayant du modèle sous-jacent de la représentation de la géométrie, nous travaillons sur un pipeline de rendu fondé sur une représentation par nuage de points. Ces points étant alors interprétés comme des sources lumineuses distribuées à la surface de la scène, ils permettent de simuler un rebond de la lumière. À partir là, nous identifions et traitons trois problèmes majeurs lié à cette approche.

Dans un premier temps nous modélisons le signal lumineux porté par ces imposteurs ponctuels pour approcher au mieux la réflexion de la lumière. En s'inspirant des approches hiérarchiques fondées sur des arbres de partitionnement, nous proposons une formulation stochastique du transfert lumineux hiérarchique. Cela nous permet, d'une part, d'optimiser la quantité de calcul à réaliser en concentrant ceux-ci sur les régions nécessitant beaucoup de précision; d'autre part, en traitant les sources lumineuses de manière stochastique indépendamment les unes des autres, on se permet d'exploiter le calcul hautement parallèle à grain fin offert par les processeurs graphiques tout en conservant globalement l'avantage des approches d'éclairage à plusieurs échelles.

Dans un second temps, nous construisons efficacement la distribution de points sur la géométrie en utilisant le pipeline de rendu du processeur graphique. Pour cela, nous exploitons les unités de tessellation matériellement accélérées pour, à la volée, à la fois raffiner ou décimer la géométrie. En construisant ainsi ce nuage de points nous gérons une grande variété de scénarios dynamiques et nous nous adaptons à toute géométrie mélangeant des parties denses en triangles comme des régions composées de quelques triangles seulement. En combinant alors cet ensemble de points à notre modèle de lumières, nous approchons en temps réel, *via* le pipeline graphique, un rebond diffus de l'éclairage indirect sans prendre en compte la visibilité.

Nous proposons finalement de calculer de manière grossière mais très efficacement cette visibilité à l'aide d'une nouvelle structure de données. Ainsi, en réduisant les requêtes de visibilité à un processus évaluant la présence ou non d'obstacle le long d'une direction donnée, nous généralisons la notion de grille de voxels tridimensionnelle en y ajoutant deux dimensions supplémentaires correspondant à l'orientation. Cela nous permet alors d'exploiter la quantité de mémoire, mise à disposition par les générations récentes de processeurs graphiques, pour agencer les données de sorte à ce que les accès mémoire, principale limitation des performances, soient rendus optimaux par notre structure d'accélération.

TABLE DES MATIÈRES

Table des matières	ii
1 Introduction	1
1.1 Enjeux pour la CAO	2
1.2 Processeur Graphique et Rendu temps réel	3
1.3 Objectifs et Contraintes	4
1.4 Contributions	4
1.5 Organisation de la thèse	5
1.6 Publications et Productions	5
2 Contexte Technique	7
2.1 Synthèse d'image	7
2.1.1 Scène 3D	7
2.1.2 Grandeurs du rendu	8
2.1.3 Equations du rendu	12
2.1.4 Opérateur de transport	14
2.1.5 Intégration de Monte-Carlo	15
2.2 Processeur Graphique	16
2.2.1 Rastérisation	16
2.2.2 Pipeline Graphique	17
2.2.3 Contraintes du calcul parallèle	19
2.2.4 Moteur de rendu temps réel	19
3 Tour d'horizon des approches d'éclairage global et pipelines temps réel	21
3.1 Éclairage global par lancer de chemins	21
3.2 Éclairage global par <i>caches</i>	24
3.2.1 Radiosité ou Éléments Finis	25
3.2.2 Interpolation de champ lumineux	26
3.2.3 Radiosité Instantanée	29
3.2.4 Éclairage en espace image	31
3.2.5 Lumières Multiples et Approches Hiérarchiques	33
3.3 Synthèse	35
4 Coupes antérogrades de lumières	37
4.1 Introduction	37
4.2 Coupes antérogrades discrètes	38
4.2.1 Coupe Antérograde	39
4.2.2 Estimateur par coupes antérogrades	40
4.2.3 Reconstruction lumineuse diffuse par fonction de support	41
4.2.4 Intégration à fréquences multiples	46
4.2.5 Répartition des sources	49
4.2.6 Évaluation	49

4.3	Coupes antérogrades continues	54
4.3.1	Formalisme	54
4.3.2	Solution Discrète	54
4.3.3	Interprétation et Représentation	56
4.3.4	Prolongement Continu	57
4.3.5	Résultats	61
4.4	Discussion	63
5	Rééchantillonnage dynamique par points pour l'éclairage global	65
5.1	Introduction	65
5.2	Travaux Antérieurs	66
5.3	Vue d'ensemble	67
5.4	Construction à grain fin d'échantillonnages uniformes pour le calcul d'intégrales	69
5.4.1	Définition du problème	69
5.4.2	Contraintes	70
5.4.3	Solution Naïve	70
5.4.4	Angle d'attaque	70
5.4.5	Échantillonnage par triangle	72
5.4.6	Validité de l'échantillonnage	73
5.4.7	Répartition stochastique uniforme	73
5.4.8	Solution du problème	74
5.4.9	Échantillonnage multi-échelles	75
5.4.10	Lois non-uniformes	76
5.5	Implémentation de l'échantillonneur	76
5.5.1	Variable aléatoire par triangle	77
5.5.2	Algorithme d'échantillonnage global	80
5.5.3	Pipeline de subdivision	81
5.6	Vers l'éclairage global diffus temps réel	84
5.6.1	Pipeline de rendu complet	84
5.6.2	Lien avec les coupes antérogrades	85
5.6.3	Aplatissement en espace image	87
5.6.4	Rendu Entrelacé et Filtrage	88
5.6.5	Rendu Progressif	89
5.7	Résultats et Évaluation	89
5.8	Discussion et Ouverture	92
6	Cartes d'HyperVoxels Sphériques	95
6.1	Introduction	95
6.2	Vue d'ensemble	97
6.3	Marche de rayons	97
6.4	Marche Idéale	100
6.4.1	Cas Unidimensionnel	101
6.4.2	Représentation Mémoire	102
6.4.3	Marche naïve en dimension deux et trois	103
6.5	Cartes d'HyperVoxels Sphériques	105
6.5.1	Description Générale	105
6.5.2	Paramétrisation de l'espace des droites	106
6.5.3	Repère tangent à la sphère	109

6.5.4	HyperVoxel Sphérique	110
6.6	Algorithme	112
6.6.1	Construction des cartes	112
6.6.2	Requêtes optimales arbitraire	114
6.6.3	Auto-occlusion	115
6.6.4	Filtrage par pourcentage de visibilité	117
6.7	Résultats	117
6.8	Discussion	120
7	Conclusion	121
7.1	Résumé des contributions	121
7.2	Perspectives	123
Bibliographie		125
Table des figures		131

INTRODUCTION

Depuis plusieurs décennies, l'informatique graphique prend de plus en plus de place dans le paysage quotidien. Qu'il s'agisse des effets spéciaux dans les films, des jeux vidéos ou encore de la conception assistée par ordinateur (CAO), toutes ces technologies ont pu émerger grâce à la vertigineuse ascension de la puissance de calculs des ordinateurs. Ceux-ci permettent alors de modéliser et visualiser des mondes virtuels toujours plus complexes et réalistes. Nous nous intéressons ici au problème de synthèse d'images photo-réalistes à partir de modèles de scènes numériques. Pour produire des images donnant au mieux l'illusion du réalisme, celles-ci sont calculées à l'aide d'équations décrivant les observations réelles sur le comportement de la lumière. En cela, l'équation du rendu [Kajiya, 1986] offre un fondement mathématique pour modéliser et simuler le transport de la lumière à travers une scène. En outre, elle décrit l'énergie transportée par les chemins lumineux depuis les sources primaires jusqu'aux capteurs, après avoir rebondi et interagi avec la matière. La simulation des phénomènes qui en découlent s'appelle l'éclairage global.

Bien que les lois physiques sous-jacentes soient relativement bien comprises, la résolution numérique de cette équation demeure encore très difficile et coûteuse en temps de calcul. Cela devient extrêmement contraignant pour des scénarios nécessitant des performances temps réel sur des scènes complètement dynamiques et composées d'objets très détaillés. C'est pourquoi on distingue aujourd'hui deux axes de recherche ayant des objectifs différents. On a d'une part le rendu hors-ligne, qui vise à produire un résultat aussi réaliste que possible. Cependant, bien que d'importants progrès aient été réalisés, notamment grâce au développement des techniques de Monte-Carlo, les images peuvent nécessiter plusieurs heures, voire plusieurs jours de calculs à produire pour les scènes les plus complexes. On a d'autre part les rendus dits interactifs ou temps-réel qui, quant à eux, cherchent à produire une image aussi rapidement que possible pourvu que le résultat final soit visuellement plausible. Pour atteindre ce but, il est nécessaire de faire des approximations vis-à-vis de l'équation et de la solution exacte.

Dans cette thèse, nous nous focalisons sur le rendu interactif en cherchant à reformuler les entités en présence (éclairage, visibilité, *etc.*) de sorte à ce que les calculs puissent être grandement réduits tout en restant proches de la solution mathématique. Nous nous intéressons également au traitement de données haute résolution typiques des scénarios de conception assistée par ordinateur et de leur traitement sur les processeurs graphiques. Bien que ces derniers aient une puissance de calcul bien supérieure aux processeurs classiques, leur architecture est différente et les algorithmes permettant d'en tirer profit doivent s'adapter au calcul parallèle à grain fin.



FIGURE 1.1 – Synthèse d’images dans l’industrie. En haut, images tirées du film *Moana* (à gauche) et du jeu vidéo *Ori and the Blind Forest* (à droite). En bas, images extraites du logiciel *CATIA* sur un modèle de paquebot (à gauche) et d’usine (à droite), chacun constitué de centaines de millions de triangles.

1.1 Enjeux pour la CAO

Les travaux présentés dans ce document s’inscrivent dans le cadre d’une thèse CIFRE en collaboration avec *Dassault Systèmes*, principal acteur de la conception assistée par ordinateur. L’objectif est de produire un algorithme de rendu adapté aux scénarios de la CAO pour simuler l’éclairage global en temps réel.

Il est avant tout nécessaire de définir ce qui est entendu par le terme « adapté », et ce qui rend cette problématique singulière. En effet, dans de nombreux domaines, comme l’industrie cinématographique ou vidéoludique, le réalisme est nécessaire, cf. figure 1.1. En revanche, là où, pour un film par exemple, l’image représente le produit et donc ce que l’on cherche à obtenir, l’image ne représente qu’un moyen dans l’industrie de la CAO. En effet, ce que l’on cherche à produire, c’est le modèle qui est en cours de construction par les outils numériques. Il n’en demeure pas moins que le besoin de réalisme reste présent dans l’objectif d’avoir une meilleure visualisation du produit ou même d’en faire une promotion publicitaire. Ainsi, ce qui constitue la spécificité du modèle de CAO vient de ce qu’il doit représenter : à la fois le modèle physique de haute résolution nécessaire à sa construction, et l’objet virtuel 3D dont on cherche à produire une image. Ces modèles sont alors très souvent massifs (constitués de plusieurs dizaines de millions de triangles), dynamiques (ils peuvent être édités en temps réel), et la géométrie qui les compose ne peut pas être modifiée pour s’adapter à l’algorithme de rendu (le produit étant le modèle, pas l’image). Nous détaillons à présent les raisons de ces spécificités.

Scénarios Avant la fabrication d'un produit conçu sur ordinateur, un nombre important de simulations est réalisé afin de valider la viabilité de celui-ci. Ces simulations testent plusieurs critères physiques du futur produit tels que la résistance des matériaux aux efforts, à la chaleur ou encore au temps. L'aspect visuel est également un critère capital, notamment pour des produits visant la grande consommation qui se doivent d'être esthétiques, par exemple pour l'industrie automobile. Pour cela, les rendus sont presque toujours calculés à l'aide de techniques hors ligne de Monte-Carlo qui sont capables de produire un résultat très proche de la réalité physique mais qui peuvent cependant prendre plusieurs heures de calculs. Bien que cette étape soit nécessaire, elle ne permet pas d'interactivité car, à chaque modification de la scène (géométrie, éclairage, point de vue, etc.), le rendu doit être entièrement recalculé. Ainsi, afin d'accélérer le processus d'édition, il est nécessaire d'avoir un retour visuel calculé en temps-réel même si ce dernier n'est qu'approximatif.

Spécificités des modèles de CAO Pour en obtenir une précision infinie, la plupart des modèles sont décrits à haut niveau par des surfaces mathématiques lisses, (*Spline, NURBS, etc.*). Cependant, utilisées telles quelles, ces représentations sont peu pratiques pour les simulations et le rendu. C'est pourquoi les surfaces sont au préalable discrétisées à la résolution souhaitée sous forme de maillage à partir de leurs équations mathématiques. Or, puisque la finalité de cette représentation est de garantir la précision de la simulation physique avec une marge d'erreur très faible, les maillages, utilisés également pour le rendu, se doivent d'être très proches de la surface et sont dès lors très denses. En outre, puisque le modèle numérique est le patron du produit manufacturé, là où pour une scène de film ou de jeux vidéo on peut se permettre une certaine marge d'erreur afin d'adapter les modèles à l'algorithme de rendu, il est impossible d'en faire autant pour un modèle de CAO. Il nous est ainsi obligatoire de devoir travailler en temps réel sur le modèle conçu par l'utilisateur et non pas sur une version optimisée de celui-ci.

1.2 Processeur Graphique et Rendu temps réel

Les processeurs graphiques ont été historiquement conçus pour accélérer le tracer de triangles, permettant d'en traiter quelques milliers par seconde il y a trente ans, à plusieurs dizaines de millions aujourd'hui, voire plusieurs milliards. Cependant, malgré leur objectif initial extrêmement contraint, ces derniers sont devenus de plus en plus génériques et programmables. Cela leur permet ainsi de supporter de toutes nouvelles gammes d'algorithmes reléguant le tracer de triangles à une utilisation particulière des processeurs graphiques. Contrairement aux unités centrales de traitement, ou simplement processeurs, le poids historique des processeurs graphiques fait que leur architecture favorise le paradigme de programmation dite massivement parallèle puisque ces derniers sont composés de centaines, voire de milliers de coeurs. En revanche ceux-ci n'offrent pas les mêmes possibilités de calcul. En effet, chaque cœur d'une unité centrale travaille indépendamment sur des instructions ainsi que des données différentes, alors que les coeurs graphiques, du fait de leur nombre, n'offrent pas autant de finesse. Ainsi, pour utiliser leur puissance de calcul, il est nécessaire de les faire exécuter le même programme sur les différentes données à traiter. On parle alors du paradigme *SIMD*, de l'anglais *Single Instruction Multiple Data*. En conséquence, bien que la puissance théorique d'un processeur graphique soit grandement supérieure à celle d'une unité centrale, il est nécessaire d'adapter les algorithmes pour tirer profit du calcul parallèle.

Bien que nous revenons dessus dans le chapitre suivant, cf. §2.2.3, il est tout de même important de noter dès à présent que ce paradigme est particulièrement bien adapté au traitement de données

massives. En effet, dans ce genre de cas, il est souvent nécessaire de réaliser la même opération sur une grande quantité de données. Par exemple, dans le cadre du rendu, lorsque l'on doit calculer les coordonnées à l'écran de chaque sommet qui compose un maillage, la même opération de transformation est effectuée sur des centaines de milliers de sommets, ce qui rend cette architecture extrêmement bien adaptée à la gestion de données massives.

1.3 Objectifs et Contraintes

L'objectif de cette thèse est de proposer un algorithme de rendu temps réel sur processeur graphique. Celui-ci aura pour but de simuler les interactions de la lumière sur des scènes complexes, typiques en conception assistée par ordinateur.

Comme nous venons de le voir, les modèles 3D sur lesquels nous travaillons ont la spécificité d'être particulièrement massifs et peu contrôlables. Cela vient du fait qu'ils sont construits dans le but d'être un modèle pour la fabrication d'objets réels. Sur ces mêmes modèles, nous souhaitons être en mesure de calculer une approximation de l'éclairage global en scénario dynamique. Afin d'utiliser cette approximation comme une prévisualisation d'un rendu final, il est par ailleurs nécessaire qu'à chaque instant l'image reste cohérente avec le rendu hors ligne précis.

1.4 Contributions

Pour répondre à la problématique qui nous est posée, nous proposons, dans cette thèse, les trois contributions suivantes :

- Tout d'abord, *cf. chapitre 4*, puisque la géométrie massive sous la forme de liste de triangles est peu manipulable, nous nous intéressons à une représentation alternative de la géométrie, à savoir la représentation sous forme de nuage de points. En s'inspirant des approches calculant le transfert lumineux par points, nous proposons les « *Coupes Antérogrades de Lumières* ». Celles-ci permettent d'étendre la définition d'imposteur lumineux, propre aux approches par points, en adaptant de manière stochastique leur distance d'influence de sorte à optimiser la quantité de calculs aux régions de l'espace où ils ont le plus d'importance. Cette approche étant non biaisée, elle a pour but de produire une solution proche de la vérité terrain même en réduisant le nombre de calculs.
- Ensuite, *cf. chapitre 5*, nous proposons un modèle de pipeline entièrement exécuté sur le processeur graphique. D'une part, en travaillant, sur les unités de traitement géométrique, notre approche permet de construire, à la volée, l'échantillonnage dynamique de n'importe quelle géométrie 3D en homogénéisant la charge de travail sur tous les coeurs de calcul. D'autre part, en utilisant les unités de calcul des pixels, laissées vacantes pendant l'étape d'échantillonnage, nous proposons un pipeline complètement dynamique de calcul d'un rebond lumineux de l'éclairage diffus en temps réel se fondant sur les *coupes antérograde de lumières*.
- Enfin, *cf. chapitre 5*, comme notre premier pipeline ne permet pas de calculer l'occlusion générée par l'éclairage indirect, nous proposons une approche visant à calculer la fonction de visibilité générale. En cela, nous introduisons les « *Cartes d'HyperVoxels Sphériques* »

comme un modèle de structure de données permettant d'accélérer n'importe quelle requête de visibilité. Le résultat est alors imprécis, ce qui n'est cependant pas pénalisant pour de la simulation d'éclairage global diffus.

1.5 Organisation de la thèse

Dans la suite nous organisons la thèse de la façon suivante. Dans les chapitres 2 et 3, nous présentons respectivement les bases techniques nécessaires à la bonne compréhension de ce manuscrit ainsi que l'état de l'art des techniques de rendu temps réel de l'éclairage global. Nos contributions techniques correspondent aux chapitres 4, 5 et 6. Tout d'abord, cf. chapitre 4, nous présentons les *coupes antérogrades de lumières*, c'est-à-dire notre reformulation de l'équation du rendu adaptée à la représentation par points. Puis, cf. chapitre 5, nous présentons notre pipeline de rendu temps réel implémentant le transport diffus de la lumière par les *coupes antérogrades*. Enfin, cf. chapitre 6, nous présentons les *cartes HyperVoxels Sphériques*, notre solution temps réel permettant de calculer la visibilité générale, avant de conclure cette thèse au chapitre 7.

1.6 Publications et Productions

Cette thèse a donné lieu aux publications scientifiques suivantes :

- « *A Scalable Approach to Real-Time Global Illumination* »,
Gilles Laurent, Cyril Delalandre, Grégoire de La Rivière et Tamy Boubekeur,
I3D 2016 (poster).
- « *Forward Light Cuts : A Scalable Approach to Real-Time Global Illumination* »,
Gilles Laurent, Cyril Delalandre, Grégoire de La Rivière et Tamy Boubekeur,
Computer Graphics Forum (Proc. EGSR 2016) 2016. [Laurent et coll., 2016a]
- « *Spherical Voxel Maps* »,
Gilles Laurent, Cyril Delalandre et Tamy Boubekeur,
en soumission.

Par ailleurs, en étroite collaboration industrielle avec Dassault Systèmes, deux brevets internationaux ont également été déposés :

- « *Rendering the global illumination of a 3D scene* »,
Gilles Laurent, Cyril Delalandre, Grégoire de La Rivière et Tamy Boubekeur,
[Laurent et coll., 2016b].
- « *Visibility Function Of A Three-Dimensional Scene* »,
Gilles Laurent, Cyril Delalandre et Tamy Boubekeur,
[Laurent et coll., 2017].

CONTEXTE TECHNIQUE

Dans ce chapitre, nous introduisons les bases techniques nécessaires à la bonne lecture de la suite de ce document. Nous nous focalisons ainsi sur les deux grands axes de cette thèse. Tout d'abord, nous nous intéressons aux concepts liés à la synthèse d'image photo réaliste fondée sur l'équation du rendu. Puis dans un second temps, nous nous intéressons aux concepts liés au calcul haute performance sur processeur graphique dans le cadre du rendu.

2.1 Synthèse d'image

2.1.1 Scène 3D

Dans cette thèse, on s'intéresse au rendu réaliste, c'est-à-dire le rendu inspiré des équations de la physique. En outre, nous cherchons à décrire la synthèse d'image comme une approximation du transport lumineux. Ainsi, dans l'objectif de synthétiser une image, il est nécessaire de construire un modèle de ce que l'on souhaite rendre. C'est pourquoi on définit une scène 3D comme l'ensemble des paramètres décrivant le rendu. Parmi l'ensemble de ces paramètres, nous avons besoin de définir la notion de géométrie, de matériau, d'environnement lumineux et de capteur.

Géométrie On peut voir la géométrie comme le support de la matière qui compose la scène. On la représente très souvent par des listes de triangles qui constituent alors des maillages. La représentation sous forme de triangles n'est pas la seule possible, on peut tout à fait imaginer des représentations plus complexes, comme surfaces de Béziers ou encore des voxels ou des surfels. Cependant, parmi toutes les représentations possibles, le traitement de la géométrie pour le rendu sur processeur graphique est presque systématiquement réalisé sur des triangles.

Matériau Le matériau permet de décrire le comportement de la lumière à la surface de la géométrie. C'est ce qui définit l'aspect d'un objet, en décrivant par exemple sa couleur, sa texture, ou encore sa brillance.

Lumières Le dernier constituant intrinsèque d'une scène est l'environnement lumineux, c'est-à-dire l'ensemble des sources lumineuses primaires. Ce sont ces sources qui permettent aux objets de la scène d'être visibles en envoyant des rayons lumineux qui rebondissent sur les surfaces et atteignent les capteurs. On peut définir plusieurs formes de sources. La plus importante étant

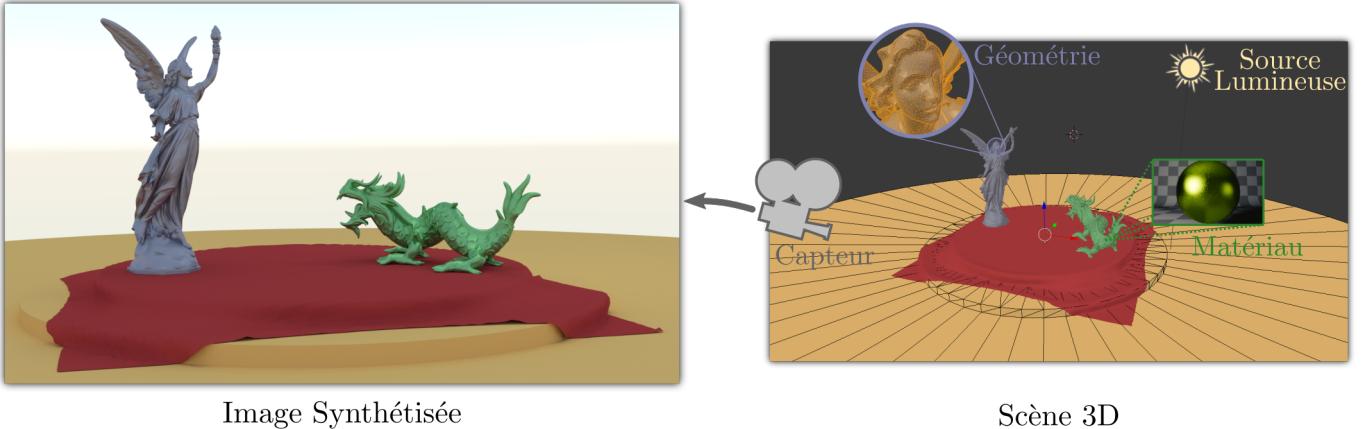


FIGURE 2.1 – Modélisation d’une scène 3D pour la synthèse d’image. À droite, l’ensemble des paramètres décrivant la scène 3D. À gauche, le résultat de la synthèse d’image à partir de la scène 3D et du capteur.

souvent le Soleil, classiquement modélisé par une source extrêmement éloignée dont les rayons arrivent parallèlement depuis le ciel. En extérieur, on peut également avoir besoin de définir une lumière ambiante provenant de toutes les directions et modélisant les multiples rebonds des rayons du Soleil traversant l’atmosphère. Dans des environnements plus confinés, comme des scènes d’intérieur par exemple, on modélise souvent la lumière par des sources ponctuelles pour décrire des lampes.

Capteur Enfin, il est nécessaire de modéliser un capteur, c’est-à-dire un mécanisme capable de mesurer et interpréter les rayons lumineux pour construire une image. Ces rayons ayant rebondi sur la scène depuis les sources primaires, ils nous permettent ainsi de visualiser la géométrie et les matériaux qui la constituent. En cela, l’œil constitue le capteur le plus évident pour nous, cependant la modélisation de son fonctionnement demeure extrêmement complexe. D’autres capteurs beaucoup plus simples, comme les caméras, sont traditionnellement utilisés pour capturer les images du monde réel. En synthèse d’image, et particulièrement en rendu temps réel, on se contente souvent d’un modèle simplifié de ces capteurs. Un certain nombre d’effets d’optiques, comme la profondeur de champ, sont directement liés au modèle de caméra utilisé pour réaliser le rendu.

2.1.2 Grandeur du rendu

Établie par Kajiya [Kajiya, 1986], l'*équation du rendu* formalise la notion de transport de la lumière le long de chemins lumineux. Ainsi, à l’aide d’une équation intégrale récursive, le comportement de la lumière, dépendant des matériaux et des sources primaires présents dans la scène, peut être décrit en chaque point de l’espace. Pour cela, il est nécessaire d’introduire un certain nombre de grandeurs, empruntées au monde de la physique, que nous décrivons ci-dessous, cf. figure 2.3. On note, cependant, que l’idée de décrire sous forme intégrale les interactions électromagnétiques, formalisées par les équations de Maxwell, est nettement plus ancienne [Yamauti, 1926, Buckley, 1927].

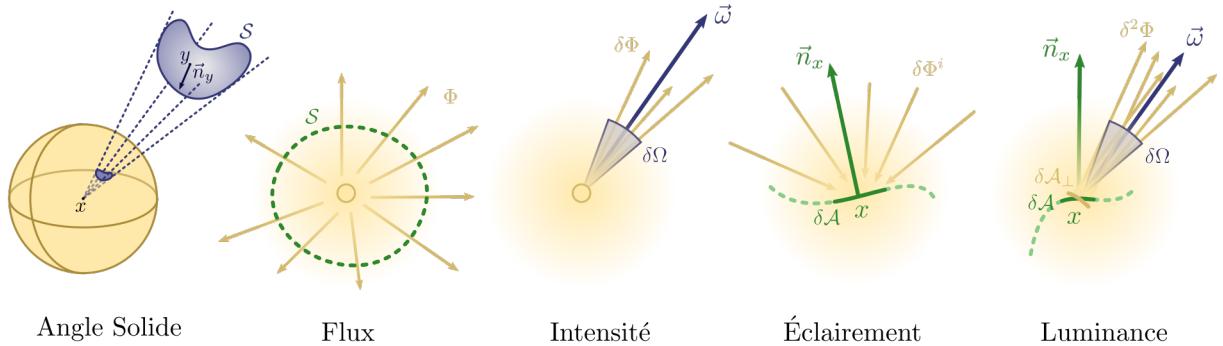


FIGURE 2.2 – Grandes physiques de la synthèse d'image.

Flux Si l'on se donne une source lumineuse, c'est-à-dire une source générant un rayonnement électromagnétique ou des photons, on peut définir et mesurer la quantité d'énergie δQ qui traverse n'importe quelle surface \mathcal{S} , extérieure à la source, au cours d'un certain intervalle de temps δt . On définit alors le flux lumineux, Φ , qui traverse cette surface comme la limite du ratio $\frac{\delta Q}{\delta t}$ lorsque l'intervalle de temps tend vers 0.

$$\frac{\delta Q}{\delta t} \underset{\delta t \rightarrow 0}{\longrightarrow} \Phi \quad (2.1)$$

Cette grandeur étant une quantité d'énergie par intervalle de temps, elle a pour unité le Watt (W). Le flux lumineux permet ainsi de transposer les phénomènes physiques au rendu. En effet, c'est bien l'énergie, portée par la lumière, qui vient exciter les capteurs et qui permet de construire une image du monde.

Angle Solide Le flux lumineux n'est cependant pas suffisamment fin pour décrire le comportement des rayons à notre échelle. En effet, cela ne permet de prendre en compte, ni la direction, ni la localité de la lumière. Afin de parler de direction de la lumière, il nous faut d'abord définir la notion d'angle solide. Une direction peut être définie de manière unique par un vecteur unitaire, c'est-à-dire un point appartenant à la sphère unité. Ainsi, parler d'un ensemble de directions équivaut à parler d'un ensemble de points sur cette sphère. L'angle solide de cet ensemble de directions est alors défini comme l'aire de la section de sphère correspondante ; cette définition généralise ainsi la notion d'angle en dimension 2.

De plus, le contexte ne laissant pas d'ambiguïté, nous nommons par abus de langage, indifféremment, Ω pour désigner un ensemble de directions et l'angle solide associé ; l'angle solide représentant alors une mesure dans l'espace des directions. De cette façon, si la valeur d'angle solide $\delta\Omega$, correspondant à une boule centrée en $\vec{\omega}$ dans l'espace des directions, tend vers 0, la suite d'ensembles qui correspond tend vers le singleton $\{\vec{\omega}\}$.

On peut alors définir en un point donné x , pour n'importe quelle surface \mathcal{A} , l'angle solide de cette surface vu depuis x , $\Omega_x(\mathcal{A})$. Avant cela, on définit pour tout point $y \in \mathcal{A}$ de normal \vec{n}_y , l'angle solide élémentaire :

$$d\omega_y = \frac{d\mathcal{A}_y^\perp}{||x - y||^2} = \frac{\langle \vec{n}_y, \bar{y}x \rangle d\mathcal{A}_y}{||x - y||^2}, \quad (2.2)$$

où $\langle \vec{u}, \vec{v} \rangle$ est le produit scalaire entre les vecteurs \vec{u} et \vec{v} , et $\bar{u} = \frac{\vec{u}}{||\vec{u}||}$ est le vecteur normalisé de \vec{u} .

On définit alors :

$$\Omega_x(\mathcal{A}) = \int_{y \in \mathcal{A}} d\omega_y. \quad (2.3)$$

L'unité d'angle solide, le stéradian (*sr*), est un rapport de surfaces et n'a donc pas de dimension. On note, par ailleurs, que l'ensemble de toutes les directions a pour angle solide 4π sr, et, que l'ensemble des directions appartenant à un hémisphère pour angle solide 2π sr.

Intensité Étant donnée une source ponctuelle de lumière et une petite section de sphère d'angle solide $\delta\Omega$ centrée en ce point, on mesure $\delta\Phi$, le flux lumineux qui traverse cette section. On définit alors $I(\vec{\omega})$, l'intensité de cette source pour la direction $\vec{\omega}$ comme la limite du ratio $\frac{\delta\Phi}{\delta\Omega}$ lorsque $\delta\Omega$ tend vers $\vec{\omega}$.

$$\frac{\delta\Phi}{\delta\Omega} \xrightarrow[\delta\Omega \rightarrow \vec{\omega}]{} I(\vec{\omega}). \quad (2.4)$$

L'intensité lumineuse, dont l'unité est le Watt par stéradian ($W \cdot sr^{-1}$), permet ainsi de donner une description directionnelle de la lumière par unité d'angle solide. Il s'agit d'un bon descripteur du comportement d'une source ponctuelle. Cela traduit en effet que, dans le vide, un rayon lumineux se propage en ligne droite sans perdre d'énergie ; la densité de rayons lumineux étant alors uniquement proportionnelle à l'angle solide.

Luminance Étant donnés un point de l'espace x , une direction $\vec{\omega}$, un petit élément de surface $\delta\mathcal{A}_\perp$ orthogonal à cette direction, et un petit angle solide $\delta\Omega$ autour de $\vec{\omega}$, on peut mesurer le flux lumineux $\delta^2\Phi$ des rayons qui traversent $\delta\mathcal{A}_\perp$ avec une direction comprise dans $\delta\Omega$. En faisant tendre l'élément de surface vers 0 et $\delta\Omega$ vers la direction $\vec{\omega}$, le ratio $\frac{\delta^2\Phi}{\delta\Omega \delta\mathcal{A}_\perp}$ converge vers une grandeur $L(x, \vec{\omega})$ que l'on appelle luminance en x pour la direction $\vec{\omega}$.

$$\frac{\delta^2\Phi}{\delta\Omega \delta\mathcal{A}_\perp} \xrightarrow[\delta\mathcal{A}_\perp \rightarrow 0]{\delta\Omega \rightarrow 0} L(x, \vec{\omega}). \quad (2.5)$$

Son unité est alors le Watt par stéradian par mètre carré ($W \cdot sr^{-1} \cdot m^{-2}$). C'est à partir de cette grandeur que l'on est désormais en mesure de décrire formellement la lumière. On dit alors que définir la luminance en tout point de l'espace et pour chaque direction revient à définir le champ de luminance ; c'est ce que l'on cherche à réaliser pour synthétiser une image, du moins pour l'ensemble des pixels du capteur. On peut montrer par ailleurs que, dans le vide, cette grandeur est préservée le long d'un rayon lumineux. C'est ainsi ce qui traduit l'idée que la lumière se propage en ligne droite le long de rayon lumineux.

Par ailleurs, lorsque nous avons parlé du flux traversant la surface $\delta\mathcal{A}_\perp$, il était implicitement fait mention du flux incident Φ_i , c'est-à-dire celui issu de la lumière arrivant sur la surface. Cela nous permet donc de définir la luminance incidente que l'on note L_i . Or par la suite, il nous sera nécessaire de faire la distinction avec le flux sortant de la surface Φ_o , nous permettant de définir la luminance sortante L_o . Dans le vide, nous avons toujours $L_i(x, \vec{\omega}) = L_o(x, -\vec{\omega})$, respectant le principe de réciprocité de Helmholtz.

Éclairement et Exitance À la surface d'un objet, il est également intéressant de définir deux nouvelles grandeurs, que sont l'éclairement et l'exitance. Pour un petit élément de surface, $\delta\mathcal{A}$, autour de x , on peut définir l'éclairement $E(x)$ comme la limite du ratio $\frac{\delta\Phi_i}{\delta\mathcal{A}}$ lorsque $\delta\mathcal{A}$ tend

vers 0, et où $\delta\Phi_i$ correspond au flux incident à la surface. L'exitance $M(x)$ est définie de manière analogue mais en prenant en compte Φ_o , c'est-à-dire le flux sortant.

$$\frac{\delta\Phi_i}{\delta\mathcal{A}} \xrightarrow[\delta\mathcal{A} \rightarrow 0]{} E(x), \quad (2.6)$$

$$\frac{\delta\Phi_o}{\delta\mathcal{A}} \xrightarrow[\delta\mathcal{A} \rightarrow 0]{} M(x). \quad (2.7)$$

Ces deux grandeurs ont ainsi pour unité le Watt par mètre carré ($W.m^{-2}$).

Par ailleurs, en un point donné x d'une surface dont la normale est \vec{n}_x , l'éclairement issu d'une source angle solide Ω est lié à la luminance par :

$$E(x) = \int_{\omega_i \in \Omega} L_i(x, \vec{\omega}_i) \langle \vec{\omega}_i, \vec{n}_x \rangle d\omega_i. \quad (2.8)$$

Le terme $\langle \vec{\omega}_i, \vec{n}_x \rangle$ permet de prendre en compte le fait que, lorsqu'un rayon heurte une surface sous un angle rasant, sa contribution est complètement atténuée, tandis que, lorsque cela se produit de manière normale, sa contribution est entièrement prise en compte.

De la même façon, l'exitance en un point x d'une surface, vers toutes les directions de l'hémisphère $\mathcal{H}(\vec{n}_x)$ déterminé par la direction \vec{n}_x , est liée à la luminance par :

$$M(x) = \int_{\omega_o \in \mathcal{H}(\vec{n}_x)} L_o(x, \vec{\omega}_o) \langle \vec{\omega}_o, \vec{n}_x \rangle d\omega_o. \quad (2.9)$$

Matériaux En l'absence de tout matériau, la trajectoire de la lumière n'est pas altérée et se propage en ligne droite, ou plus précisément la luminance reste constante. Cependant, en présence d'obstacle, de support, etc. son comportement est modifié et sa trajectoire n'est dès lors plus rectiligne. On appelle alors « matériau » du milieu ou de la surface, la fonction qui décrit de quelle manière la lumière se comporte en chaque point et pour chaque direction. Sur une surface, nous appelons la fonction qui décrit de quelle manière la lumière se comporte, la fonction de distribution de diffusion bidirectionnelle ou *BSDF* (de l'anglais *Bidirectional Scattering Distribution Function*). Cette fonction se décompose en deux termes : la fonction de réflexion ou *BRDF* (de l'anglais *Bidirectional Reflectance Distribution Function*) et la fonction de transmission ou *BTDF* (de l'anglais *Bidirectional Transmittance Distribution Function*). Le premier terme traduisant les effets de rebonds de la lumière, le second traduisant les effets de transparence mais que nous n'étudions pas ici.

Dans une forme simplifiée, qui correspond à celle qui nous intéresse par la suite, la *BRDF*, notée traditionnellement f , correspond à une fonction à 7 degrés de liberté que sont : $\lambda \in \mathbb{R}$ pour la longueur d'onde, $x \in \mathcal{S}$ pour la position sur la surface, $\vec{\omega}_i$ pour la direction incidente et $\vec{\omega}_o$ pour la direction sortante. Cette fonction décrit, pour un petit élément de surface, la quantité de lumière $\delta L_o(\vec{\omega}_o)$ qui est réfléchie dans une la direction $\vec{\omega}_o$, étant donné un éclairement $\delta E(\vec{\omega}_i)$ selon un petit cone de directions incidentes. f est alors la limite du ratio $\frac{\delta L_o(\vec{\omega}_o)}{\delta E(\vec{\omega}_i)}$ lorsque l'élément de surface tend vers 0, et que le cône de directions incidentes tend vers $\vec{\omega}_i$. Pour un élément de surface x , dont la normale est \vec{n}_x , on a alors :

$$f(\lambda, x, \vec{\omega}_i, \vec{\omega}_o) = \frac{dL_o(\lambda, x, \vec{\omega}_o)}{dE(\lambda, x, \vec{\omega}_i)} = \frac{dL_o(\lambda, x, \vec{\omega}_o)}{dL_i(\lambda, x, \vec{\omega}_i) \langle \vec{n}_x, \vec{\omega}_i \rangle}. \quad (2.10)$$

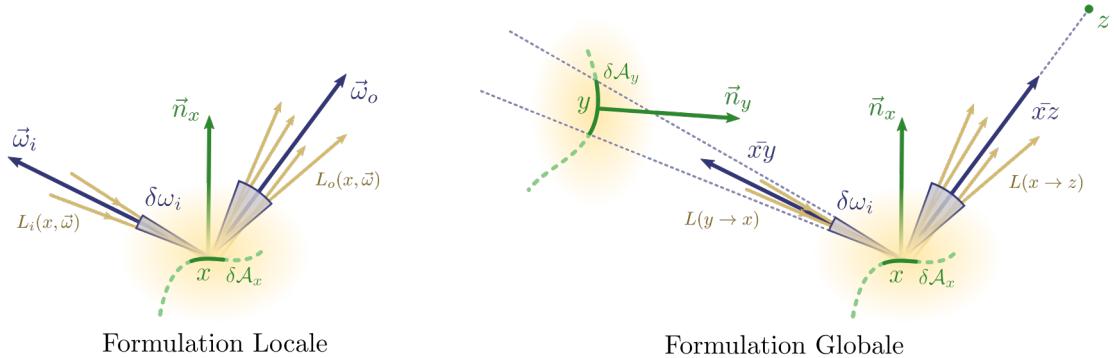


FIGURE 2.3 – Réflexion surfacique de la lumière. À gauche, la version locale nécessite de définir une fonction de luminance incidente. À droite, la version globale lie la luminance incidente en x à la luminance sortante en y grâce à l’angle solide de $\delta\mathcal{A}_y$ vu depuis x .

Dans la suite, on fait l’hypothèse que les matériaux reflètent les longueurs d’onde indépendamment les unes des autres. Dans nos calculs, on ne précise donc plus le terme λ , et on suppose que les formules faisant intervenir $L(x, \vec{\omega})$ sont valables pour n’importe quelle longueur d’onde. De plus, par le principe de réciprocité de Helmholtz, la *BRDF* est symétrique par rapport aux variables $\vec{\omega}_i$ et $\vec{\omega}_o$. Ainsi :

$$f(x, \vec{\omega}_i, \vec{\omega}_o) = f(x, \vec{\omega}_o, \vec{\omega}_i). \quad (2.11)$$

Dans la suite, nous nous intéressons principalement au transport diffus de la lumière. Cela se traduit sur la *BRDF*, que l’on qualifie alors de lambertienne, par le fait que la luminance sortante en x est constante quelque soient les directions incidentes et sortantes. On la note alors $f_d(x)$ ou simplement $f(x)$.

2.1.3 Equations du rendu

Formulation Locale L’équation de transport lumineux décrit, à partir de la luminance, le comportement de la lumière lorsque celle-ci se trouve en présence de matériaux. Dans sa version locale, pour un point x sur une surface, elle se définit de la façon suivante :

$$L_o(x, \vec{\omega}_o) = L^e(x, \vec{\omega}_o) + \int_{\omega_i \in \Omega} L_i(x, \vec{\omega}_i) f(x, \vec{\omega}_i, \vec{\omega}_o) \langle \vec{n}_x, \vec{\omega}_i \rangle \, d\vec{\omega}_i, \quad (2.12)$$

où $L^e(x, \vec{\omega}_o)$ représente le champ de luminance des sources primaires de lumière. Il s’agit bien d’une équation fonctionnelle sur L car, pour tout segment $[x, y]$ n’ayant pas d’intersection avec la géométrie, nous avons la relation de réciprocité :

$$L_o(x, \bar{x}\bar{y}) = L_i(y, \bar{y}\bar{x}), \quad (2.13)$$

signifiant que la luminance sortante de la surface en x dans la direction $\bar{x}\bar{y}$ est égale à la luminance incidente en y depuis la direction $\bar{y}\bar{x}$. Cette hypothèse est vérifiée, car nous supposons qu’il y a du vide entre les surfaces.

Par commodité, on note également :

$$F(x, \vec{\omega}_i, \vec{\omega}_o) = f(x, \vec{\omega}_i, \vec{\omega}_o) \langle \vec{n}_x, \vec{\omega}_i \rangle. \quad (2.14)$$

Il existe d'autres paramètres que nous n'avons pas évoqués mais dont devrait dépendre la luminance. Entre autres, nous supposons ici qu'entre deux éléments de surface de la scène, il n'y a que du vide, ce qui n'est pas le cas en général, mais la prise en compte des milieux participants dépasse le cadre de cette thèse et ne sera pas abordé ici.

Formulation Globale Comme nous nous plaçons dans l'approximation du transport lumineux entre des surfaces séparées par le vide, on se permet ici d'introduire deux nouvelles notations, simplifiant les formulations qui suivent. Ainsi, pour tout couple de points de l'espace $x, y \in \mathbb{R}^3$, on définit l'expression de la luminance :

$$L(x \rightarrow y) = L_o(x, \bar{y}) = L_i(y, \bar{x}). \quad (2.15)$$

De même, pour tout point y de la surface de la scène \mathcal{S} , et tout couple de points de l'espace $x, z \in \mathbb{R}^3$, on définit l'expression de la *BRDF* en x par :

$$f(y \leftrightarrow x \leftrightarrow z) = f(x, \bar{y}, \bar{z}). \quad (2.16)$$

Par ailleurs, dans la version locale de l'équation du rendu, cf. formule 2.12, en chaque point x de la surface, on intègre la luminance incidente sur tous les angles solides incidents. Cependant, il est également possible de voir cette équation comme la réflexion de la luminance émise par tous les éléments de surface $d\mathcal{A}_y$ de la scène \mathcal{S} , visible depuis x . L'équation de transport peut ainsi être reformulée par :

$$L(x \rightarrow z) = L^e(x \rightarrow z) + \int_{y \in \mathcal{S}} L(y \rightarrow x) V(x, y) f(y \leftrightarrow x \leftrightarrow z) \langle \vec{n}_x, \bar{y} \rangle^+ \frac{\langle \vec{n}_y, \bar{y} \rangle^+ d\mathcal{A}_y}{||x - y||^2},$$

où $\langle \vec{u}, \vec{v} \rangle^+ = \max(0, \langle \vec{u}, \vec{v} \rangle)$, et $V(x, y)$ est la fonction de visibilité, retournant 1 si x et y sont mutuellement visibles, et 0 autrement. Ici, le terme $\frac{\langle \vec{n}_y, \bar{y} \rangle^+ d\mathcal{A}_y}{||x - y||^2}$ correspond à l'angle solide du petit élément de surface autour du point y , et vu depuis x . De plus, les $\langle \cdot, \cdot \rangle^+$ permettent de capturer l'information comme quoi, la luminance est émise (resp. reçue) uniquement selon l'hémisphère orienté par la direction \vec{n}_y (resp. \vec{n}_x).

En introduisant le facteur de forme entre les points x et y :

$$G(x, y) = \frac{\langle \bar{y}x, \vec{n}_y \rangle^+ \langle \bar{y}x, \vec{n}_x \rangle^+}{||x - y||^2}, \quad (2.17)$$

l'équation surfacique ci-dessus est souvent réécrite de la manière suivante :

$$L(x \rightarrow z) = L^e(x \rightarrow z) + \int_{y \in \mathcal{S}} L(y \rightarrow x) f(y \leftrightarrow x \leftrightarrow z) G(x, y) V(x, y) d\mathcal{A}_y. \quad (2.18)$$

Dans la suite de cette thèse, nous utilisons cette formulation de l'équation du rendu pour décrire la lumière réfléchie sur une surface. Cependant, d'un point de vue calculatoire, nous devons évaluer le facteur de forme, $G(x, y)$, dont la valeur n'est pas bornée lorsque x et y se rapprochent ; cela est dû à la présence du terme $\frac{1}{||x - y||^2}$. Comme cela introduit des problèmes de précisions numériques sur ordinateur, il est fréquent de borner ce terme à l'aide d'un seuil arbitraire, ε_b et d'évaluer une version approchée du facteur de forme :

$$G_b(x, y) = \frac{\langle \bar{y}x, \vec{n}_y \rangle^+ \langle \bar{y}x, \vec{n}_x \rangle^+}{\max(\varepsilon_b, ||x - y||^2)}. \quad (2.19)$$

Cette approximation a pour cause d'introduire un assombrissement exagéré dans les zones concaves par rapport à l'équation originale. Certains travaux [Dachsbacher et coll., 2014] s'intéressent à ce problème, mais cela dépasse le cadre d'étude de cette thèse.

2.1.4 Opérateur de transport

Dans le but de fournir un cadre d'étude formel de l'analyse d'erreur dans les algorithmes de simulation de l'éclairage global, Arvo et coll. [Arvo et coll., 1994] ont introduit le concept d'opérateur de transport, transformant un champ de luminance en un autre champ de luminance. Dans notre cas d'utilisation, on le définit comme un endomorphisme linéaire de l'espace vectoriel des fonctions de $\mathcal{S} \times \Omega$ dans \mathbb{R} tel que :

$$\mathcal{T}: L \mapsto \begin{cases} \mathcal{S} \times \Omega & \rightarrow \mathbb{R} \\ (x, \vec{\omega}) & \mapsto \int_{y \in \mathcal{S}} L(y \rightarrow x) h(y \rightarrow x \rightarrow \vec{\omega}) d\mathcal{S}_y \end{cases}, \quad (2.20)$$

où

$$h(y \rightarrow x \rightarrow \vec{\omega}) = f(y \leftrightarrow x \leftrightarrow \vec{\omega}) G(x, y) V(x, y). \quad (2.21)$$

L'équation de transport peut alors s'écrire de la manière compacte suivante :

$$L = L^e + \mathcal{T}L. \quad (2.22)$$

Cette formulation permet de mettre en évidence l'équation fonction sur L en l'écrivant :

$$(Id - \mathcal{T}) L = L^e,$$

où Id est l'opérateur identité. On peut alors montrer que, sous certaines conditions sur les valeurs propres de \mathcal{T} , [Arvo et coll., 1994], l'endomorphisme $(Id - \mathcal{T})$ est inversible. Cela nous permet ainsi d'écrire :

$$L = (Id - \mathcal{T})^{-1} L^e = \sum_k \mathcal{T}^k L^e. \quad (2.23)$$

Cette formulation met en avant le caractère récursif de l'équation ainsi que le fait que la luminance est égale à la somme de la source primaire L^e , de l'éclairage direct $\mathcal{T}L^e$, du premier rebond de la lumière $\mathcal{T}^2 L^e$, du second $\mathcal{T}^3 L^e$ et ainsi de suite. On note ainsi, $L^k = \mathcal{T}^k L^e$, la luminance incidente produite par la lumière directe après exactement k rebonds ; $L^{1 \rightarrow \infty}$ pour la luminance produite par au moins un rebond ; ou plus généralement :

$$\forall i, j, 0 \leq i \leq j, \quad L^{i \rightarrow j} = \sum_{k=i}^j L^k = \sum_{k=i}^j \mathcal{T}^k L^e,$$

pour représenter la luminance produite par au moins i rebonds, et au plus j rebonds. Dans la suite, on parle d'éclairage direct de la scène pour qualifier $L^{0 \rightarrow 1}$; et d'éclairage indirect de la scène pour qualifier $L^{2 \rightarrow \infty}$.

Bien que de cette façon, l'équation du rendu semble avoir une solution concise, l'expression de L fait intervenir des successions d'intégrales ne présentant en général aucune forme analytique connue. Pour calculer la luminance il faut donc recourir à des méthodes numériques.

2.1.5 Intégration de Monte-Carlo

L'intégration de Monte-Carlo est une méthode numérique non déterministe permettant de donner une approximation aussi précise que l'on souhaite au calcul d'une intégrale. Ainsi, étant donnés Ω un espace mesurable et une fonction $f : \Omega \rightarrow \mathbb{R}$, on cherche à calculer :

$$I = \int_{x \in \Omega} f(x) dx. \quad (2.24)$$

Espérance Soit Y une variable aléatoire à valeur dans \mathbb{R} . On note μ_Y sa densité de probabilité, c'est-à-dire pour tout $A \subset \mathbb{R}$, la probabilité $\mathbb{P}(Y \in A) = \int_{y \in A} \mu_Y(y) dy$. On définit alors son espérance comme :

$$\mathbb{E}[Y] = \int_{y \in \mathbb{R}} y \mu_Y(y) dy. \quad (2.25)$$

Soit X une variable aléatoire à valeur dans Ω , possiblement distinct de \mathbb{R} . On note comme c-dessus μ_X sa densité de probabilité. Soit ϕ une application de Ω dans \mathbb{R} . On définit l'espérance de la variable aléatoire $\varphi(X)$ comme :

$$\mathbb{E}[\varphi(X)] = \int_{x \in \Omega} \varphi(x) \mu_X(x) dx \quad (2.26)$$

Loi des grands nombres Soit (Y_1, \dots, Y_N) , N variables aléatoires à valeur dans \mathbb{R} , indépendantes, et toutes de même densité de probabilité μ_Y d'espérance $\mathbb{E}[Y]$. On dit alors que ces variables sont indépendantes identiquement distribuée (ou i.i.d.). Notons alors :

$$\tilde{Y}_N = \frac{1}{N} \sum_{i \in [\![1, N]\!]} Y_i. \quad (2.27)$$

La loi des grands nombres nous dit alors que la suite des Y_N converge en probabilité vers $\mathbb{E}[Y]$, c'est-à-dire :

$$\forall \varepsilon > 0, \quad \mathbb{P}(\tilde{Y}_N - \mathbb{E}[Y] > \varepsilon) \xrightarrow[N \rightarrow \infty]{} 0. \quad (2.28)$$

Autrement dit, quelque soit la marge d'erreur ε que l'on se donne, on peut toujours trouver un N assez grand tel que la moyenne empirique \tilde{Y}_N soit un estimateur de $\mathbb{E}[Y]$ avec une probabilité « forte ». On dira par la suite que la moyenne empirique approche « bien » l'espérance mathématique, pourvu que l'on moyenne suffisamment de réalisations Y_i . Par abus de langage, on note cela par :

$$\tilde{Y}_N \xrightarrow[N \rightarrow \infty]{} \mathbb{E}[Y]. \quad (2.29)$$

De même que nous avions généralisé l'espérance dans le paragraphe précédent, la loi des grands nombres se généralise trivialement pour N variables aléatoires indépendantes (X_1, \dots, X_N) à valeur dans Ω et pour toute fonction φ à valeur dans \mathbb{R} , en posant $Y_i = \varphi(X_i)$. On a alors :

$$\tilde{Y}_N = \frac{1}{N} \sum_{i \in [\![1, N]\!]} Y_i = \frac{1}{N} \sum_{i \in [\![1, N]\!]} \varphi(X_i) \xrightarrow[N \rightarrow \infty]{} \mathbb{E}[\varphi(X)]. \quad (2.30)$$

Calcul d'intégrales Étant donnée une variable aléatoire X de densité de probabilité μ_X , et la fonction φ telle que $f(x) = \varphi(x)\mu_X(x)$. D'après ce que nous venons de décrire, on a :

$$I = \int_{x \in \Omega} f(x) dx = \int_{x \in \Omega} \varphi(x)\mu_X(x) dx = \mathbb{E}[\varphi(X)]. \quad (2.31)$$

Cela signifie que, calculer l'intégrale I équivaut à calculer l'espérance de la variable aléatoire $Y = \varphi(X) = \frac{f(X)}{\mu_X(X)}$. Ainsi, pour (X_1, \dots, X_N) , N variables aléatoires indépendantes à valeur dans Ω de même loi μ_X :

$$\tilde{Y}_N = \frac{1}{N} \sum_{i \in [\![1, N]\!]} Y_i = \frac{1}{N} \sum_{i \in [\![1, N]\!]} \frac{f(X_i)}{\mu_X(X_i)} \xrightarrow{N \rightarrow \infty} I. \quad (2.32)$$

Notons, qu'il est par ailleurs possible de donner une estimation sur la vitesse de convergence de la suite \tilde{Y}_N en fonction de la variance et des moments d'ordre supérieur de la loi de Y . Cependant, nous ne nous intéressons pas à cela dans cette thèse et nous nous contentons de la convergence simple de la loi.

En Pratique Afin d'approcher numériquement une intégrale I , nous nous donnons N réalisations indépendantes d'une variable aléatoire X de densité de probabilité μ_X , que nous notons x_1, \dots, x_n . Nous calculons alors la moyenne suivante :

$$\tilde{I}_N = \frac{1}{N} \sum_{i \in [\![1, N]\!]} \frac{f(x_i)}{\mu_X(x_i)}. \quad (2.33)$$

Enfin, pour un nombre d'échantillons N « assez grand », on estime que cette évaluation est « assez proche » de I .

2.2 Processeur Graphique

Nous nous intéressons ici au processeur graphique et plus précisément aux mécanismes permettant de synthétiser une image en temps réel.

2.2.1 Rastérisation

L'objectif ici est de calculer la projection d'une scène 3D sur un écran. L'écran est ainsi subdivisé en petits éléments, les pixels, pour lesquels on cherche la luminance issue du plus proche élément de la scène. Il est donc nécessaire de pouvoir calculer quel objet est le plus proche ; on dit que ce problème consiste à résoudre la visibilité depuis la caméra.

Là-dessus, le mécanisme de rastérisation offre une solution efficace sur processeur graphique pour obtenir, pour chaque pixel, la géométrie et le matériau du plus proche élément. Néanmoins, l'algorithme requiert deux conditions pour fonctionner. Tout d'abord, la géométrie doit être représentée par des triangles. Ensuite, seule une sous-classe de fonctions peut être utilisée pour projeter la scène sur l'écran. Dans notre cas, nous sommes restreint à la projection perspective et orthoscopique.

Une fois ces deux conditions remplies, l'algorithme de rastérisation fonctionne de la manière suivante, cf. figure 2.4. Pour chaque triangle, on détermine l'ensemble des pixels dont le centre

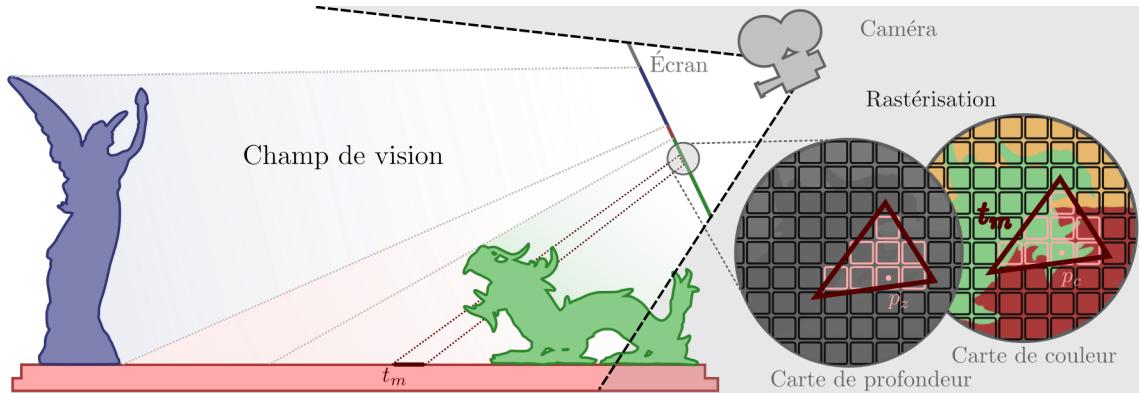


FIGURE 2.4 – Mécanisme de rastérisation avec test de visibilité. L’ensemble des triangles de la scène est projeté en 2D sur l’écran de la caméra. Pour chaque pixel de cet écran, on maintient une carte de profondeur enregistrant la profondeur du plus proche élément. Lorsqu’un triangle est projeté, on ne calcule la couleur de ses fragments que s’ils réussissent le test par rapport à la carte de profondeur. Auquel cas, celle-ci est mise à jour avec la valeur la plus petite.

se trouve à l’intérieur du triangle projeté sur l’écran. Le point correspondant est appelé fragment. En utilisant les coordonnées barycentriques du fragment dans le triangle, sa profondeur (c’est-à-dire sa distance au plan de l’écran) est linéairement interpolée à partir de celle des trois sommets. Cette profondeur est alors comparée avec la profondeur contenue dans le pixel correspondant de la carte de profondeur. Cette valeur correspond à celle du triangle le plus proche déjà traité. Si la profondeur du fragment est plus petite, c’est-à-dire que le fragment est plus proche, alors une couleur est calculée, stockée dans la carte de couleur et le pixel de la carte de profondeur est mis à jour. Si la profondeur est plus grande, c’est-à-dire que le fragment est derrière, alors le fragment est défaussé et rien ne se passe.

2.2.2 Pipeline Graphique

L’algorithme de rastérisation que nous venons de décrire existe depuis de longtemps, et de nombreuses optimisations matérielles ont été étudiées et développées pour permettre le plus d’efficacité et de généricité possible. De cette façon, les processeurs graphiques, qui ont été initialement conçus pour implémenter de manière rigide cet algorithme, sont devenus de plus en plus programmables.

Ainsi, le processus de construction d’image par rastérisation est désormais décrit par ce que l’on nomme le *pipeline graphique*, cf. figure 2.5. Il s’agit d’une succession d’étapes travaillant, dans un premier temps, sur la géométrie, permettant de transformer les triangles à la volée, puis, dans un second temps, sur chaque fragment généré. Une partie de ces étapes sont paramétrables par des programmes, que l’on nomme *Shaders*, et qui sont exécutés sur le processeur graphique.

Pipeline Minimal À partir d’une discréttisation de la scène sous forme de triangles, la première étape, le *Vertex Shader*, opère sur l’ensemble des sommets. Au cours de cette étape, il est fréquent de transformer la position des sommets pour les projeter dans l’espace écran à l’aide des paramètres de la caméra. Il est également possible d’y définir des attributs propres au sommet, comme la normale, ou les coordonnées de texture.

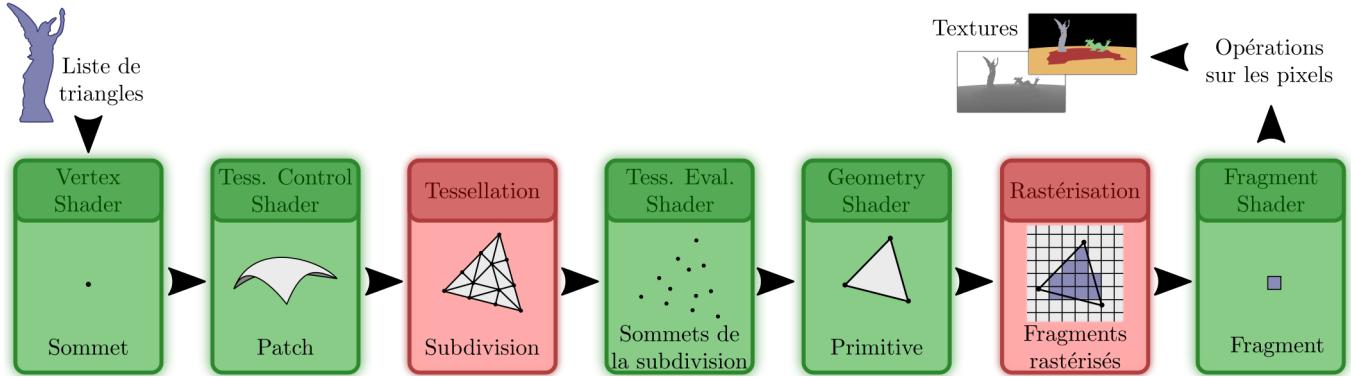


FIGURE 2.5 – Pipeline graphique programmable. Il permet de synthétiser en temps réel l'image d'une scène 3D constituée de triangles. L'ensemble des étapes programmables, en vert, travaille à différents niveaux : pour la préparation de la géométrie, du *Vertex Shader* au *Geometry Shader*, ainsi qu'au calcul de la couleur du pixel, le *Fragment Shader*. Les étapes non programmables, en rouge, optimisent les parties critiques du pipeline.

Dans le pipeline simple, dès que trois sommets, formant un triangle, ont été transformés, ils sont envoyés directement à l'unité de rastérisation. Celle-ci est en charge de déterminer l'ensemble des fragments de l'écran à l'intérieur du triangle projeté. Cette étape, cruciale pour l'algorithme de rastérisation, est implémentée matériellement dans le processeur graphique et il n'est pas possible de la reprogrammer. Il est néanmoins possible de moduler son comportement. En outre, au cours de cette étape, il est possible d'activer la construction de la carte de profondeur pour résoudre la visibilité avant de générer les fragments. Un certain nombre d'autres optimisations y sont réalisées pour minimiser la quantité de calculs nécessaire à la formation de l'image. Enfin, c'est également cette unité qui se charge d'interpoler linéairement la valeur des attributs des sommets pour chaque fragment produit.

Dans la dernière étape programmable de ce pipeline, le *Fragment Shader*, il est possible de définir la couleur des fragments du triangle. À l'aide des attributs interpolés, toutes les informations concernant la position, la normale, le matériau, etc., y sont précisément disponibles. Le *Fragment Shader* a ainsi pour but de calculer la fonction d'éclairage, ou toute autre fonction, à partir des attributs ainsi que des données globales, telles que des textures, la position des sources lumineuses etc. Le pipeline expose enfin une dernière paramétrisation une fois que le fragment a été calculé, permettant, en outre, de mélanger la couleur de sortie avec celle déjà contenue dans le pixel de la texture, par exemple pour simuler approximativement la transparence, ou accumuler la luminance d'un pixel.

Extensions du pipeline L'efficacité du pipeline graphique réside dans son architecture extrêmement parallèle. Ainsi, chaque sommet peut être traité en parallèle indépendamment des autres, et il en est de même pour les fragments. Cependant, pour pouvoir traiter la géométrie, il est parfois nécessaire de travailler à une granularité plus large que le simple sommet.

En cela, la première extension ajoutée au pipeline graphique est le *Geometry Shader* qui travaille à l'échelle de la primitive (point, ligne ou triangle). Cette étape est réalisée juste avant le processus de rastérisation et offre beaucoup de libertés. Il est bien entendu possible de retransformer les triangles, mais aussi d'en produire de nouveaux ou de les défausser. De même, un triangle peut être changé en n'importe quelle autre type de géométrie (ligne ou point). Cependant, cette étape

étant extrêmement expressive, il a été difficile pour les constructeurs de cartes graphiques d'optimiser toutes les fonctionnalités offertes par le *Geometry Shader*. En outre, l'amplification de la géométrie, c'est-à-dire la création dynamique de nouvelles primitives, se trouve être peu efficace lorsqu'elle est réalisée au cours de cette étape.

C'est pourquoi l'unité de tessellation, un peu plus contraignante, a été proposée pour permettre l'amplification de la géométrie. Cette unité est constituée de trois étapes dont seules deux sont programmables, le *Tessellation Control Shader* et le *Tessellation Evaluation Shader*. Ainsi, selon un schéma de subdivision codé matériellement, il est possible de subdiviser un patch abstrait (par exemple un triangle, ou un quadrilatère) en plusieurs triangles plus petits. La première étape, le contrôle, permet de définir dynamiquement le nombre de subdivisions à réaliser sur le patch. Enfin, une fois que la géométrie a été subdivisée, avant d'envoyer les primitives au *Geometry Shader*, l'étape d'évaluation permet de repositionner les sommets à l'aide de leurs coordonnées barycentriques.

2.2.3 Contraintes du calcul parallèle

L'architecture parallèle des processeurs graphiques est conçue pour effectuer une tâche identique sur des milliers, voire des millions, d'objets différents. Cette architecture est alors qualifiée de *SIMD* (de l'anglais *Single Instruction Multiple Data*) signifiant qu'un même jeu d'instructions est utilisé par les cœurs de calcul en même temps pour travailler sur des données différentes. Grâce à cette restriction, il est possible d'avoir, sur les architectures actuelles, plusieurs milliers de cœurs de calcul.

Bien que la programmation sur processeur graphique devienne de plus en plus générale, initialement, cette architecture a été pensée pour optimiser le pipeline graphique. Ainsi, pour chaque étape de *Shader*, le programme correspondant est exécuté sur des centaines de cœurs, traitant en parallèle les sommets, les fragments, ou la primitive en fonction de l'étape. Cependant, pour des raisons de performance, le fil d'exécution de ces programmes n'est pas réalisé de manière indépendante d'un cœur à l'autre. Ainsi, les cœurs sont regroupés par bloc (en général 32, ou 64) qui exécutent tous exactement la même instruction. De cette façon, si l'un d'entre eux (ou plusieurs) bloque le fil d'exécution, par exemple à cause d'un branchement conditionnel, ou de l'attente d'un accès mémoire, tous les autres cœurs doivent l'attendre, réduisant énormément la capacité de calcul globale. Dans le cas du pipeline graphique, cela implique qu'il est grandement préférable que tous les *Shaders* minimisent ces branchements en homogénéisant la charge de travail sur les unités de calcul.

2.2.4 Moteur de rendu temps réel

La plupart des moteurs de rendu temps réel modernes tirent profit du pipeline graphique pour pouvoir visualiser des scènes complexes. Cependant, pour gérer des géométries dynamiques constituées de plusieurs millions de triangles avec des milliers de matériaux différents, il est nécessaire de prendre certaines précautions pour communiquer avec le processeur graphique.

Il existe ainsi plusieurs interfaces de programmation permettant de commander le processeur graphique, notamment *OpenGL* sur laquelle nous nous focalisons exclusivement dans cette thèse, ou encore *DirectX* ou *Vulkan*. Pour un moteur de rendu, l'objectif consiste ainsi à préparer des commandes, depuis le processeur principal, en fonction de ce que l'on souhaite visualiser. La prépara-

tion de la scène est considérée, pour nous, comme une boite noire sur laquelle nous n'avons que peu de contrôle. Elle est censée traiter les interactions extérieures, et, pour exécuter un algorithme de rendu, nous ne demandons d'elle que deux fonctions de bases. Pour un point de vue donné, il faut être capable de produire l'ensemble des commandes définissant les sommets de la géométrie, les matériaux, et l'ensemble des *Shaders* à utiliser. La seconde fonction de base nécessaire consiste à pouvoir réaliser un filtre sur une image 2D en définissant n'importe quelle succession de calculs. Cette dernière fonctionnalité est souvent réalisée en détournant le fonctionnement classique du pipeline : la géométrie à tracer correspond à un rectangle couvrant tout l'écran, et le calcul du filtre est réalisé par pixel à l'aide d'un *Fragment Shader*.

Fort de ces deux fonctionnalités, il est possible de décrire un certain nombre d'algorithmes de rendu. Par la suite, on utilise notamment le rendu différé en construisant, par ce biais, le tampon géométrique (ou *G-Buffer* en anglais) [Saito et Takahashi, 1990]. En général, celui-ci est constitué de trois cartes correspondant à la profondeur, la normale et le matériau de la géométrie ayant traversée le pipeline de rastérisation. À l'aide du tampon géométrique, il est ensuite possible de calculer la réponse lumineuse des fragments sans avoir besoin de rendre toute la géométrie à chaque fois. Enfin, s'appuyer sur ces deux briques élémentaire constraint la façon dont il est possible de décrire ces algorithmes, mais demeure néanmoins nécessaire pour pouvoir traiter la charge de données typique des scénarios de conception assistée par ordinateur. Il s'agit dans cette thèse de notre point de départ pour inscrire nos approches dans un moteur temps réel. Nous nous concentrerons donc, par la suite, uniquement sur l'exécution en temps réel de nos algorithmes du point de vue du processeur graphique.

TOUR D'HORIZON DES APPROCHES D'ÉCLAIRAGE GLOBAL ET PIPELINES TEMPS RÉEL

Dans ce chapitre nous nous intéressons aux approches existantes visant à simuler le transport lumineux en temps réel. Pour cela nous faisons la distinction entre les approches par lancer de chemins et les approches fondées sur les *caches*. La première catégorie regroupe un ensemble de techniques produisant un résultat réaliste mais dont l'implémentation est aujourd'hui peu adaptée aux scénarios temps réel que nous visons. La seconde catégorie regroupe un ensemble de techniques utilisant un *cache* pour factoriser et réduire les calculs nécessaires au rebond lumineux. En classifiant ces algorithmes en fonction de la nature et de l'utilisation de leur *cache* de lumière, nous voyons que ceux-ci proposent des implémentations efficaces sur processeur graphique. Nous en déduisons alors trois problèmes liés aux approches fondées sur les *caches*, à savoir la nature de la grandeur qu'ils représentent, la manière de les construire et le calcul de leur visibilité. Enfin, nous introduisons nos travaux vis à vis de ces problèmes.

3.1 Éclairage global par lancer de chemins

Bien que nous ne nous appuyons pas sur cette approche dans la suite de cette thèse, le lancer de chemins (aussi appelé *Path Tracing* en anglais) est aujourd'hui la méthode la plus répandue pour calculer l'éclairage global [Christensen et Jarosz, 2016]. Elle est, par ailleurs, considérée comme la solution de référence en ce qui concerne la qualité de l'image produite. De même, dans nos comparaisons, nous considérons les images synthétisées avec cette technique comme la vérité terrain. C'est pourquoi nous donnons ici une brève explication de son fonctionnement. Nous fournissons, ensuite, une liste de limitations liées à son implémentation en temps-réel qui nous ont poussé à favoriser les approches fondées sur les *caches*. Pour d'avantage de précisions, nous référons le lecteur aux revues plus complètes sur le lancer de chemins [Davidović et coll., 2014].

Théorie L'approche par lancer de chemins consiste à construire des chemins lumineux entre les capteurs (c'est-à-dire les pixels) et les sources de lumière en lançant de manière aléatoire des rayons à travers la scène. Pour calculer l'énergie lumineuse transportée par ce chemin, l'équation récursive du rendu, *cf.* §2.1.2, est tout d'abord réduite à une somme d'intégrales sur l'espace des chemins de longueur k entre le pixels et l'ensemble des sources lumineuses. L'intégrande est

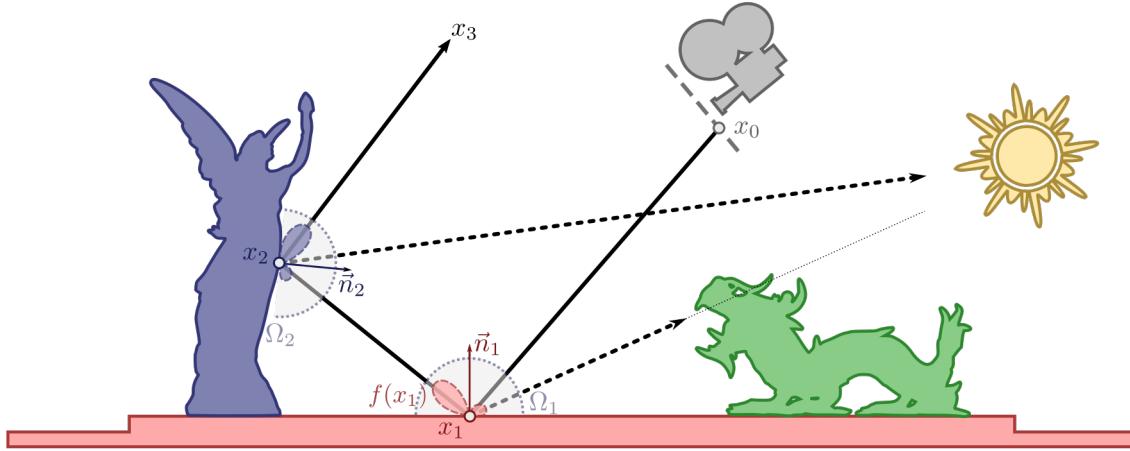


FIGURE 3.1 – Éclairage global par lancer de chemins. Pour chaque pixel de l'écran x_0 , on envoie un rayon dans la direction ω_0 . À son intersection avec la scène x_1 , on envoie alors deux nouveaux rayons. Le premier, le rayon d'ombre, part directement vers la source lumineuse et, si aucune géométrie ne l'intersecte, la contribution de cette source est ajoutée au chemin. Le second rayon prolonge le chemin d'un rebond en choisissant une direction préférentiellement par rapport à la *BRDF* $f(x_1)$. Ce processus est ensuite répété en x_2 jusqu'à ce que le rayon s'arrête ou qu'il sorte de la scène.

alors construite comme le produit des *BRDF* (ou *BSDF* dans le cas plus général) des surfaces intersectées par les rayons. En utilisant les notations définies précédemment, cf. §2.1.2, on définit alors :

$$L(x_1, -\omega_0) = \sum_k \int_{\omega_1 \dots \omega_k} \dots \int_{\Omega_1 \dots \Omega_k} L^e(x_{k+1}, -\omega_k) F(x_k, \omega_k, -\omega_{k-1}) \dots F(x_1, \omega_1, -\omega_0) d\omega_1 \dots d\omega_k,$$

où (x_0, ω_0) définit le rayon passant par un pixel, x_i est l'intersection entre le rayon d'origine x_{i-1} et de direction ω_{i-1} , et Ω_i est l'hémisphère orienté par la normale à la géométrie en x_i si ce point appartient à la scène, cf. figure 3.1.

Les méthodes de Monte-Carlo, cf. §2.1.5, sont alors utilisées pour estimer cette intégrale à grande dimension. De nombreuses techniques, que nous ne détaillerons pas, ont alors été développées pour choisir les chemins, de sorte à minimiser la variance de l'estimateur et ainsi accélérer la vitesse de convergence.

Algorithmme L'approche la plus simple à mettre en œuvre, notamment lorsque l'on cherche à implémenter cette technique en temps réel, consiste à partir d'un rayon envoyé depuis la caméra dans la direction du pixel. À chaque intersection avec la scène, on envoie alors deux rayons : le premier en direction d'une source lumineuse, que l'on appelle le rayon d'ombre (ou *Shadow Ray* en anglais), et le second dans n'importe quelle direction de l'hémisphère tangent à la surface préférentiellement par rapport à la *BRDF*. Le rayon d'ombre teste uniquement la visibilité entre l'intersection et la lumière. Si ces deux points sont mutuellement visibles, alors, la contribution lumineuse de cette source est ajoutée à celle du chemin, sinon, la luminance du chemin reste inchangée. Le second rayon sert, quant à lui, à allonger de un rebond la longueur du chemin construit. S'il génère une intersection avec la scène, alors le précédent calcul est réitéré, sinon cela veut dire que le rayon sort de la scène et le chemin s'arrête. Enfin, dans le but de réduire la longueur des chemins et d'éviter les boucles de rebonds infinis, le mécanisme de « roulette russe »

est mise en place : à chaque intersection, le rayon n'a qu'une probabilité de poursuivre son chemin proportionnelle à la réflectance de l'intersection, comprise entre 0 et 1.

Avantages Le principal atout de cette approche réside dans son caractère universel, dans le sens où, n'importe quelle scène avec n'importe quelle type de transferts lumineux peut être simulée. De plus, l'utilisation des approches par Monte-Carlo garantit que la convergence a lieu en un temps « raisonnable », quelque soit la dimension de l'intégrande, qui est ici très grande. On note également que sa simplicité d'implémentation et le fait que les rayons puissent être calculés indépendamment les uns des autres offrent une facilité de parallélisation à grain fin. On peut ainsi passer son implémentation à l'échelle sur tout type d'architecture allant du multiprocesseur aux fermes de calcul constituées de plusieurs millions de coeurs.

Difficultés pour le temps réel Avec cette approche, il vient, cependant, un certain nombres de défis techniques limitant aujourd'hui encore son utilisation dans les scénarios temps réel. La principale limitation est liée au fait que les chemins sont construits de manière aléatoire et décorrélées d'un pixel à l'autre, générant alors, un fort bruit sur l'image. Ainsi, au fur et à mesure que l'on agrège de nouveaux échantillons, l'image devient de plus en plus précise mais ce bruit diminue avec une convergence souvent très lente. Ce constat est d'autant plus dommageable que ce nombre de chemins est le facteur limitant des algorithmes temps réel. En effet, les meilleures implantations sont, aujourd'hui, en mesure d'envoyer un peu plus d'une dizaine de chemins par pixel et par trame en temps réel [Baldwin et Weber, 2016]. Pour résoudre le problème du bruit, il est presque toujours fait appel à une passe de filtrage sur l'image finale. Cependant, là où le débruitage en quelques minutes, voire quelques secondes [Zwicker et coll., 2015], et même une centaine de millisecondes pour l'inférence d'un réseau de neurones [Chaitanya et coll., 2017], constitue une durée négligeable pour des rendus hors lignes, ce surcout devient absolument prohibitif pour les scénarios temps réel. En effet, les premiers peuvent durer plusieurs heures, tandis que, les seconds ne peuvent pas prendre plus de quelques dizaines de millisecondes. Dans ce dernier cas, le bruit est retiré à l'aide de filtres plus rapides mais moins robustes vis à vis d'une forte variance, comme le filtre bilatéral [Tomasi et Manduchi, 1998].

La seconde difficulté réside en la brique élémentaire du lancer de chemins, c'est-à-dire le test d'intersection entre les rayons et les triangles, que l'on appelle le lancer de rayons (ou *Ray Tracing* en anglais) [Appel, 1968, Whitted, 1979, Cook et coll., 1984]. Bien que de la puissance de calcul des processeurs graphiques soit exploitée depuis des années pour le lancer de rayons [Wald et coll., 2001], il n'est cependant pas possible de tester naïvement tous les triangles de la scène un à un sans structure d'accélération sur la géométrie. Le problème découlant de cela est l'incompatibilité avec les scènes dynamiques, car, le cout de construction de cette structure peut s'avérer important. Nombre d'approches étudient ce problème pour lequel il ressort la nécessité de trouver un compromis entre la qualité de la structure, accélérant ainsi le nombre possible de tests rayon-triangle, et la vitesse de construction de celle-ci [Aila et coll., 2013]. On voit alors se dégager deux stratégies, l'une visant à reconstruire à la volée l'intégralité de la structure [Garanzha et coll., 2011, Karras et Aila, 2013] quitte à rendre les traversés de l'arbre moins efficace, et l'autre, optimisant la qualité de l'arbre mais ne pouvant pas être recalculé dans les scénarios dynamiques [Stich et coll., 2009].

Enfin, le dernier point concerne lui aussi le lancer de rayons, mais, est lié à l'inefficacité des accès mémoire incohérents qui en résulte. En effet, lorsque les rayons sont construits, ceux-ci le sont de manière aléatoire et décorrélée les uns des autres, c'est-à-dire avec des direc-

tions et des origines très différentes. Cela a pour conséquence que les calculs effectués pour résoudre leur intersection avec la scène varient fortement d'un rayon à l'autre et nécessitent l'accès à des zones de la mémoire possiblement très variées. En comparaison, le mécanisme de rastérisation, optimisé sur les processeurs graphiques, cf. §2.2.2, ne souffre pas de ce problème car, en contraignant les cas d'utilisation, les tests d'intersection sont réalisés à la granularité du triangle. Cela permet ainsi de partager en mémoire toutes les données communes, nécessaires à ce calcul, par exemple les positions, les normales, ou le matériau. Dans le cas du lancer de rayons, comme aucune contrainte n'est posée sur l'origine et la direction des rayons, il devient très compliqué de factoriser les accès mémoire et de les partager pour le calcul de plusieurs rayons. Pour palier ce manque de cohérence, un certain nombre de techniques [Novák et coll., 2010, Van Antwerpen, 2011, Laine et coll., 2013, Eisenacher et coll., 2013] proposent de réordonner les rayons en fonction d'heuristiques fondées sur leur position et leur orientation. De cette façon, en regroupant sur une même unité de calcul des groupes de rayons de même nature, la probabilité de pouvoir réutiliser la mémoire nécessaire pour la traversée de la structure d'accélération et pour la définition des triangles, s'en trouve accrue. Cependant, la quantité de calculs induite par ce réordonnement ne peut être complètement négligé. Cela limite, ainsi, le gain obtenu et le nombre de rayons qu'il est possible de traiter à chaque trame.

On observe néanmoins de récentes avancées sur les processeurs graphiques visant à accélérer matériellement la constructions de structure d'accélération et les test d'intersection entre rayons et triangles. Cela promet, peut-être, une vive amélioration du mécanisme de lancer de rayons dans les années à venir [Stich, 2018].

3.2 Éclairage global par *caches*

Dans cette section, nous nous intéressons à la seconde grande approche permettant de calculer l'éclairage global, à savoir les approches par *caches*. Contrairement aux approches par lancer de chemins, l'idée, cette fois-ci, consiste à positionner des *caches* dans la scène, sur les surfaces et éventuellement dans les zones vides, dans le but d'y capturer l'éclairage incident pour ensuite le propager dans la scène.

Il vient alors plusieurs problèmes nécessaires à la mise en œuvre de ce mécanisme. Tout d'abord, puisque les ordinateurs ont une mémoire limitée, il faut définir une famille de fonctions sur laquelle il est possible de projeter et reconstruire le signal lumineux. Cette famille doit ainsi permettre une reconstruction fidèle sans pour autant nécessiter trop de mémoire. Ensuite, la qualité du signal étant fortement liée à la position des *caches* et à leur quantité, il faut les placer en prenant soin d'en utiliser le minimum possible aux endroits où ils ont une véritable importance. Enfin, il est nécessaire de pouvoir calculer la visibilité entre les caches et les sources lumineuses ainsi que les capteurs (souvent les pixels). Pour ce calcul de visibilité, il est souvent suffisant de fournir une simple information binaire, visible ou non visible, contrairement aux approches par lancer de rayons, où, la position du point le plus proche ainsi que son matériau sont requis.

Dans la suite, nous proposons une classification des approches fondées sur les *caches* en parlant, tout d'abord, des techniques de *radiosité*, cf. §3.2.1, qui sont à l'origine de toute cette classe d'algorithmes. Puis, en ayant présenté les défis qu'elle engendre, nous parlons de la façon d'interpoler des champs lumineux, cf. §3.2.2, et d'optimiser le calcul pour atteindre le temps réel grâce à la *radiosité instantanée*, cf. §3.2.3, et les approches en espace image, cf. §3.2.4. Enfin, nous nous

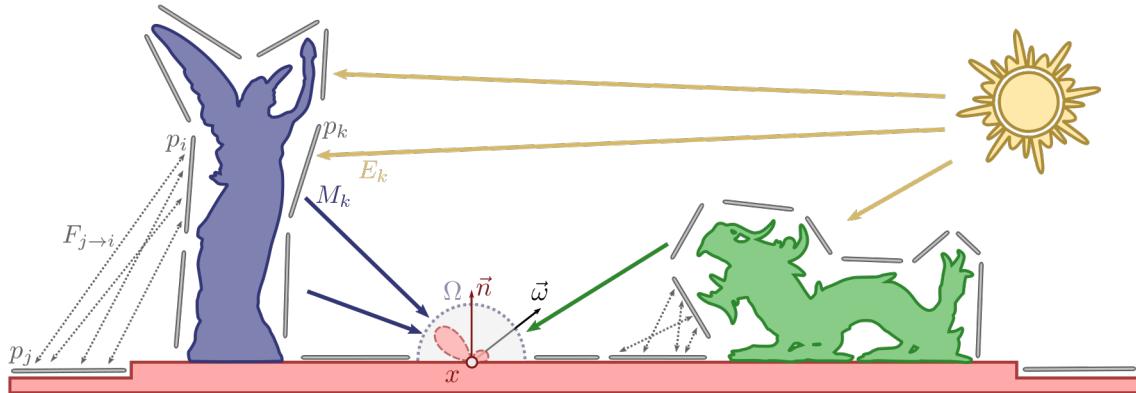


FIGURE 3.2 – Éclairage global par éléments finis. Tout d’abord, une fois que la scène a été discrétisée en éléments finis de surface (en gris), leurs facteurs de forme $F_{j \rightarrow i}$ sont calculés et enregistrés dans une matrice \mathbf{F} . Lors du calcul de l’éclairage, l’exitance directe des éléments finis M_k^e est déterminée analytiquement à partir de l’éclairage E_k issu des sources lumineuses. Puis le transport entre les éléments finis est simulé en appliquant N fois la matrice \mathbf{F} . Enfin la luminance au point x est calculée soit en évaluant la contribution des éléments les plus proches par collecte finale (comme ici), soit en interpolant celle de l’élément fini contenant x et de ses plus proches voisins.

intéressons aux approches hiérarchiques, *cf.* §3.2.5. Nous mettons en parallèle leur faculté à répondre aux problèmes que nous venons de décrire, au regard de leur implémentation en temps réel.

3.2.1 Radiosité ou Éléments Finis

Parmi les premières approches visant à simuler le transport lumineux, on s’intéresse ici aux méthodes dites de radiosité (autre nom de l’exitance que nous avons définie précédemment, *cf.* §2.1.2) aussi appelées méthodes par éléments finis [Goral et coll., 1984]. Ces approches se focalisent principalement sur le transport diffus et consistent à discrétiser la surface par des éléments simples, en général des polygones, de sorte à ce que l’on puisse calculer le facteur de forme pour chaque paire d’entre eux, *cf.* figure 3.2. Celui-ci, que l’on note $F_{j \rightarrow i}$, correspond à la fraction d’exitance émise par l’élément p_j et reçue par l’élément p_i . Ainsi, en posant M_i comme l’exitance de l’élément p_i , et M_i^e comme l’exitance de l’élément p_i causée par l’éclairage direct, on peut décrire le transfert radiatif indirect par la formulation recursive suivante :

$$M_i = M_i^e + \sum_j F_{j \rightarrow i} M_j.$$

Pour simplifier la formule ci-dessus, nous avons implicitement incorporé le terme diffus de la BRDF dans l’expression de $F_{j \rightarrow i}$. Par ailleurs, en posant $\mathbf{M} = (M_k)_k$, $\mathbf{M}^e = (M_k^e)_k$, et $\mathbf{F} = (F_{j \rightarrow i})_{ij}$, on obtient la forme vectorielle suivante :

$$\mathbf{M} = \mathbf{M}^e + \mathbf{F} \cdot \mathbf{M} = \sum_{k=0}^{\infty} \mathbf{F}^k \cdot \mathbf{M}^e.$$

Avec cette dernière formule, pourvu que l'on connaisse les M^e qui sont en général calculables analytiquement, il est possible de trouver l'exitance à l'équilibre de la scène, grâce à une succession de multiplications matrice vecteur. Pour la résolution, il est souvent préférable de conserver la matrice \mathbf{F} , quitte à devoir réaliser cette succession de multiplications, plutôt que de calculer la matrice résolue ($\sum_k \mathbf{F}^k$) et n'avoir qu'un seul produit à faire. En effet, puisque \mathbf{F} a pour coefficients les facteurs de forme, ceux-ci sont, pour la plupart, nuls ou négligeables. Cela permet ainsi de stocker la matrice de manière parcimonieuse et d'accélérer les calculs [Gortler et coll., 1993]. De plus, en généralisant cette approche, grâce à une discréétisation de l'ensemble des directions, il est possible de prendre en compte le transport non diffus [Immel et coll., 1986]. Cela a néanmoins pour conséquence d'accroître significativement le nombre de coefficients à calculer.

Par ailleurs, on note que, bien qu'il en existe une formulation analytique [Schröder et Hanrahan, 1993], le calcul du facteur de forme reste compliqué en pratique, d'autant qu'aucune formule ne prend naturellement en compte la visibilité entre les polygones. Pour palier cela, Ward et coll. utilisent le lancer de rayons [Ward et coll., 1988] tandis que Cohen et Greenberg [Cohen et Greenberg, 1985] proposent d'utiliser un hemi-cube pour calculer l'angle solide des éléments les plus proches à l'aide de la rastérisation. Hanrahan et coll. [Hanrahan et coll., 1991] et Smits et coll. [Smits et coll., 1994] développent une formulation hiérarchique du transport. Cela leur permet de remplacer plusieurs facteurs de formes distant par quelques uns plus grossier, tout en conservant une approximation valide. Sur processeurs graphique, Dachsbacher et coll. [Dachsbacher et coll., 2007] introduisent la notion de luminance négative pour permettre de propager l'occlusion des éléments, sans avoir besoin de calculer explicitement la visibilité entre eux. Ils démontrent alors qu'une implémentation en temps réel de la *radiosité* est possible, mais ils génèrent également des artéfacts très perturbants liés aux termes négatifs.

Enfin, pour synthétiser la couleur finale du pixel x , l'exitance en ce point n'est pas utilisée directement. En effet, afin de conserver un système calculable, le nombre d'éléments finis est borné, ce qui fait que leur taille apparente sur l'image est bien supérieure à celle des pixels. Pour résoudre ce problème, plusieurs approches sont possibles. Soit la luminance des éléments est interpolée, comme proposé initialement par Goral et coll. [Goral et coll., 1984], cf. §3.2.2. Soit on réalise une collecte finale (aussi appelée *final gathering* en anglais) qui consiste à intégrer le produit des exitances des éléments voisins avec la *BRDF* du pixel. Cette dernière approche a donné lieu, plus tard, à la *radiosité instantanée*, cf. §3.2.3.

3.2.2 Interpolation de champ lumineux

Pour calculer la luminance des pixels, Ward et coll. [Ward et coll., 1988] ainsi que Goral et coll. [Goral et coll., 1984] proposent d'interpoler linéairement l'exitance calculée dans les *caches* les plus proches. Cette idée se base sur l'observation intuitive que la lumière varie peu dans le domaine spatial, c'est-à-dire que deux points « proches » dans l'espace ont une valeur de luminance « proche ». Cependant Ramamoorthi et coll. [Ramamoorthi et Hanrahan, 2001] et Basri et coll. [Basri et Jacobs, 2001] font remarquer que cette affirmation n'est valide que dans un certain cas d'application, à savoir lorsque l'on cherche à simuler des réflexions diffuses sur des surfaces lisses dont l'éclairage est distant. De même Durand et coll. [Durand et coll., 2005] étudient l'influence de la variation spatiale et directionnelle mais, bien qu'ils montrent que l'opérateur de transport puissent s'apparenter à filtre passe bas, il est nécessaire de faire cette interpolation avec soin. C'est pourquoi nous nous intéressons ici aux méthodes visant à interpoler le signal lumineux.

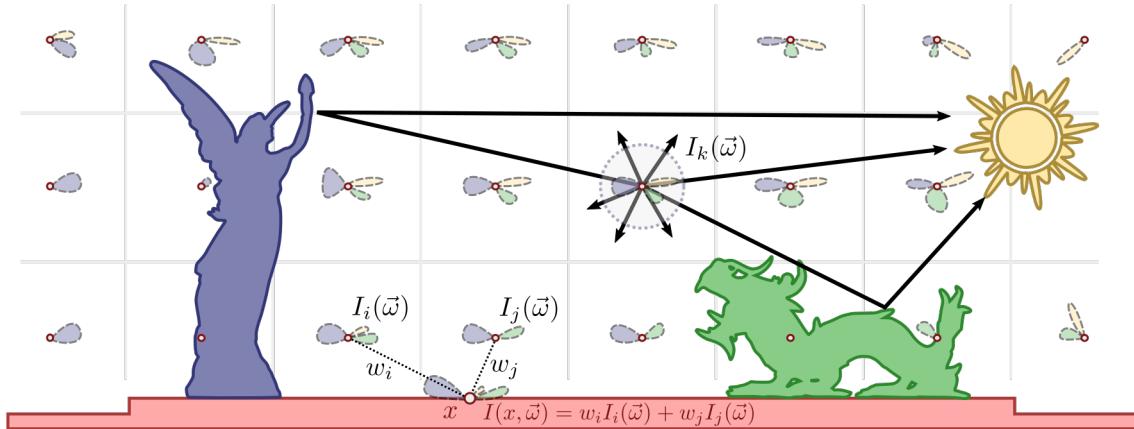


FIGURE 3.3 – Reconstruction par interpolation sur une grille régulière. Dans un premier temps, le centre de chaque cellule est interprété comme un *cache* dont l'intensité lumineuse est évaluée par une simulation du transport lumineux. Pour reconstruire le champ lumineux en x , il suffit alors d'interpoler l'intensité des *caches* les plus proches dont les poids, w_i , sont définis par la stratégie d'interpolation. Dans ce cas, ils peuvent être choisis comme ceux issus d'une interpolation bilinéaire en fonction de la distance au centre des cellules.

Pour améliorer la qualité de l'interpolation, Ward et Heckert [Ward et Heckbert, 1992] proposent de calculer le gradient de l'exitance par rapport au domaine spatial et angulaire. De cette façon, ils proposent un développement de Taylor d'un ordre supérieur leur permettant d'étendre le domaine de validité de leur calcul à des surfaces qui ne sont plus nécessairement planes et lisses. En s'intéressant ainsi à la variation angulaire de l'exitance par rapport à la normale de la surface, cela revient à considérer la luminance incidente. Par ailleurs, comme le montrent Durand et coll. [Durand et coll., 2005], plus on s'éloigne des sources lumineuses, plus le signal peut être vu comme le mélange entre beaucoup de sources, réduisant généralement sa fréquence spatiale. Cela pousse ainsi à considérer l'interpolation du champ lumineux juste avant le rebond, c'est-à-dire la luminance incidente, car sa fréquence est alors beaucoup plus faible que celle de la luminance sortante, cf. §2.1.2. Dans le cas de la réflexion diffuse, par exemple, bien qu'il soit plus coûteux de stocker une fonction plutôt qu'une simple couleur, on peut alors se contenter de la luminance incidente en un nombre très réduit de points avant de l'évaluer à la surface qui, quant à elle, peut avoir de forte variation au niveau de son albédo ou de sa normale cf. figure 3.3. Il vient alors le besoin de définir une base sur laquelle décomposer la luminance de sorte à ce que les coefficients aient de faible variations spatiales. De cette façon, il devient possible de les interpoler directement, plutôt que d'évaluer le signal en plusieurs points. Il existe ainsi un certain nombre de bases permettant de décomposer ce signal de manière efficace. Il est ainsi possible de décomposer les directions de la luminance sur les harmoniques sphériques [Kautz et coll., 2002, Sloan et coll., 2002], ou de calculer le gradient spatial de celles-ci dans le but de mieux interpoler les coefficients [Annen et coll., 2004]. Enfin, il est également possible d'utiliser les ondelettes pour capturer de manière adaptative les signaux hautes fréquences présents dans les rebonds spéculaires.

Enfin, pour évaluer la luminance finale des pixels, il faut également décomposer la *BRDF* locale dans cette base. De cette façon, l'opérateur de transport peut être vu comme une application bilinéaire pour laquelle il suffit d'avoir pré-calculé la réponse sur la base correspondante. Le calcul de la luminance revient alors à calculer un produit scalaire de vecteurs constitués des coefficients de la luminance et de la *BRDF*. Par ailleurs, grâce à cette représentation, il est égale-

ment possible de prendre en compte la fonction de visibilité et de la décomposer sur cette base. L'opérateur de transport devient alors une application trilinéaire pour laquelle il est nécessaire de calculer le produit triple sur les vecteurs de la base [Kautz et coll., 2002, Tsai et Shih, 2006, Sun et Ramamoorthi, 2009]. La quantité de coefficients à stocker pour représenter cette application peut cependant s'avérer extrêmement volumineuse, et l'évaluation peut devenir lente.

La seconde difficulté liée à l'interpolation de champs lumineux réside dans le fait de trouver les *caches* les plus proches avec lesquels faire ce calcul. On peut alors différencier deux approches opposées, la collecte (en anglais *gathering*) et l'aplatissement (en anglais *splatting*). Dans la première, l'objectif est de trouver pour chaque receveur, quelles sont les *caches* à utiliser, lire leurs données, et réaliser le calcul d'interpolation. L'avantage de cette approche est la compatibilité avec le calcul parallèle. En effet, si N processus accèdent à une même adresse alors la mémoire associée peut être mise en *cache*, factorisant ainsi les long délais d'accès. De plus, comme chaque pixel est à la responsabilité d'un unique processus, l'écriture de sa couleur finale peut être réalisée en une seule opération, sans collision avec d'autres processus. Cette approche n'est cependant possible que si l'on est en mesure de pouvoir trouver dynamiquement les *caches* les plus proches ainsi que la donnée qu'ils contiennent. Pour cela Ward et Heckbert [Ward et Heckbert, 1992] proposent de calculer l'exitance sur une grille régulière, et de reconstruire la valeur de l'exitance pour chaque pixel grâce à une interpolation trilinéaire des huit coefficients des cellules voisines. Dans la même idée, Kaplanyan et Dachsbacher [Kaplanyan et Dachsbaucher, 2010] proposent de représenter la luminance par des harmoniques sphériques dans une grille. Ils proposent ainsi un schéma de propagation lumineux leur permettant de capturer également une approximation grossière des occlusions lumineuses en temps réel. Plus récemment, McGuire et coll. [McGuire et coll., 2017] ont utilisé une structure à base de cartes cubiques pour accélérer une approche par lancer de rayons en espace image. Cela leur permet ainsi de calculer la visibilité entre les pixels et les *caches*. Sur des scènes moins dynamiques, Olsson et coll. [Olsson et coll., 2012] proposent d'utiliser des structures d'accélérations, telles que des arbres de partitionnement de l'espace, pour trouver les sources influentes. L'approche par collecte tend ainsi à hériter des problèmes similaires à ceux du lancer de chemins concernant la nécessité de maintenir une structure d'accélération sur les *caches*.

Alternativement à la collecte, la stratégie par aplatissement consiste à réaliser le processus inverse. Pour chaque *cache*, on cherche l'ensemble des récepteurs qu'il affecte, pour sommer sa contribution lumineuse. Bien que cette approche nécessite de réaliser un grand nombre d'écritures concurrentielles, car cette fois-ci, ce sont les *caches* qui sont traités en parallèle, celles-ci s'adaptent particulièrement bien au pipeline de rendu par rastérisation. En effet, de même que pour le rendu différé, cf. §2.2.4, le processeur graphique construit automatiquement sa propre structure d'accélération sur l'espace écran. De plus, il accélère matériellement l'opération de somme concurrente au niveau des fragments via les fonctionnalités de mélange (ou *blending* en anglais, cf. §2.2.2). Ainsi, tant qu'il est possible d'exprimer ce calcul sous la forme d'une simple somme, cette technique permet de calculer efficacement la contribution lumineuse au niveau des pixels sans pour autant recourir à des structures complexes d'accélérations. Gautron et coll. [Gautron et coll., 2005] proposent alors de décomposer l'éclairement d'un pixel comme une somme pondérée de la luminance contenue dans les *caches*. De cette façon, en stockant la somme de ces poids (qui dépendent de la distance et de l'orientation relative des pixels et des *caches*) dans la composante α des pixels, ils sont en mesure de normaliser leur calcul dans une passe de filtrage finale. Dans la même idée, Lehtinen et coll. [Lehtinen et coll., 2007] proposent de calculer le transport sur une décomposition de *caches* sur plusieurs niveaux de résolution. Le signal lumineux est alors reconstruit grâce à une somme pondérée mais dont les poids dépendent également d'une partition de l'unité en fonction

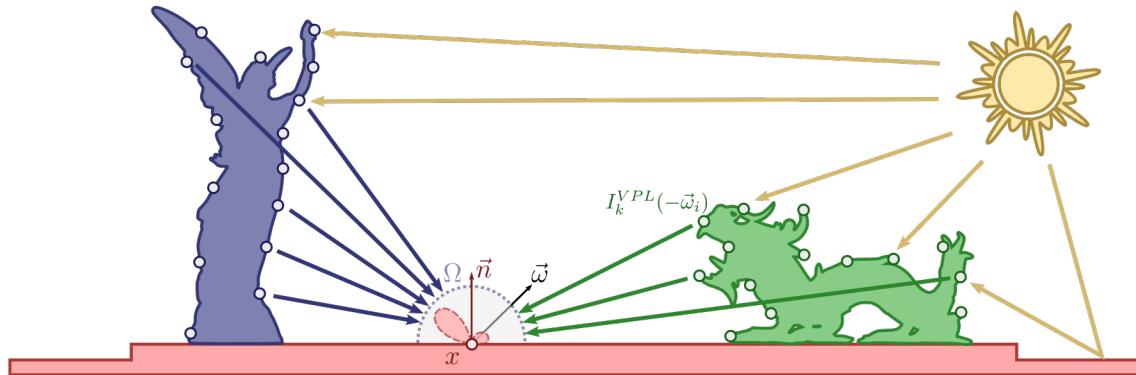


FIGURE 3.4 – Éclairage global par radiosité instantanée. L'intensité lumineuse des *VPL* est tout d'abord calculée à partir du transport lumineux depuis les sources et après un nombre quelconque de rebonds. Dans une dernière étape, la luminance du pixel en x est calculée en sommant la contribution de tous les *VPL* de la scène multipliée par la *BRDF* locale.

de la distance aux *caches* et de leur niveau de résolution tirant ainsi profit d'une approche hiérarchique. Nous développons plus loin cette idée dans nos travaux, *cf.* chapitre 4, en proposant notamment une famille de fonctions de poids lissant d'avantage la contribution lumineuse des *caches*.

Bien qu'une représentation d'ordre supérieur permette de réduire le nombre de *caches* nécessaires à la reconstruction du signal, il faut néanmoins faire un compromis avec le surcout engendré par leur calculs. De plus, certaines hypothèses de faibles fréquences ne peuvent pas toujours être respectées et une représentation uniforme sur toute la scène ne permet pas toujours une bonne reconstruction du signal. Arikán et coll. [Arikán et coll., 2005] proposent, en ce sens, de séparer la contribution lumineuse en fonction de la distance, où l'éclairage distant, de faible fréquence, est démontré comme étant bien capturé par une grille régulière grossière d'exitance. L'éclairage local, en revanche, nécessite un échantillonnage plus fin, tel que celui produit par la radiosité instantanée.

3.2.3 Radiosité Instantanée

Introduite par Keller [Keller, 1997], la technique de radiosité instantanée vise à simuler le transport lumineux à l'aide d'une discrétisation des surfaces de la scène par un ensemble de points. Ces points, que l'on nomme *VPL* (terme venant de l'anglais *Virtual Point Lights* et que nous conservons dans la suite de cette thèse), forment alors des *caches* capturant la lumière incidente directe et indirecte. L'énergie stockée dans ces sources virtuelles est estimée à partir de l'éclairage directe et des multiples rebonds lumineux. En dernière étapes, les *VPL* sont utilisés pour calculer très rapidement le rebond final des surfaces réfléchissantes vers les pixels.

Ainsi, pour définir les *VPL*, il est commun de séparer la luminance directe et indirecte. Puis, à partir de la formulation surfacique de l'équation de transfert, *cf.* formule 2.18, on peut alors écrire :

$$\begin{aligned} L(x \rightarrow \vec{\omega}) &= L_{\text{direct}}(x \rightarrow \vec{\omega}) + L_{\text{indirect}}(x \rightarrow \vec{\omega}) \\ &= L_{\text{direct}}(x \rightarrow \vec{\omega}) + \int_{y \in \mathcal{S}} L^{1 \rightarrow \infty}(y \rightarrow x) f(y \leftrightarrow x \leftrightarrow \vec{\omega}) G(x, y) V(x, y) d\mathcal{A}_y, \end{aligned}$$

en utilisant les notations que nous avons définies précédemment, *cf.* §2.1.3, et où $L^{1 \rightarrow \infty}(y \rightarrow x)$ correspond à la luminance sortante en y en direction de x issue des sources primaires après au moins un rebond.

En échantillonnant alors la scène par un nuage de points aléatoires $(y_k)_k$, indépendants et identiquement distribués selon une densité de probabilité μ^S , il est possible d'appliquer une méthode de Monte-Carlo, *cf.* §2.1.5, pour construire un estimateur de la luminance indirecte :

$$\hat{L}_{\text{indirect}}(x \rightarrow \bar{\omega}) = \frac{1}{N} \sum_{k=1}^N \frac{L^{1 \rightarrow \infty}(y_k \rightarrow x)}{\mu^S(y_k)} f(y_k \leftrightarrow x \leftrightarrow \bar{\omega}) G(y_k, x) V(y_k, x). \quad (3.1)$$

On définit alors un *VPL* comme une source de lumière ponctuelle positionnée en y_k dont l'intensité lumineuse est donnée par :

$$I_k^{\text{VPL}} = \frac{L^{1 \rightarrow \infty}(y_k \rightarrow x)}{N \mu^S(y_k)}. \quad (3.2)$$

La position des *VPL* reste importante et deux améliorations sont classiquement reconnues, la radiosité instantanée bidirectionnel [Segovia et coll., 2006] et la radiosité instantanée avec échantillonnage de Metropolis [Segovia et coll., 2007]. Dans ces deux extensions, la position des *caches* est itérativement modifiée de sorte à ne conserver que les *VPL* ayant une réelle influence sur la scène et ainsi réduire leur nombre.

Par ailleurs, bien que cette approche se fonde également sur un estimateur de Monte-Carlo pour calculer la luminance, celle-ci se distingue du *lancer de chemins* en ce sens qu'elle ne produit pas de bruit haute fréquence sur l'image finale. En effet, comme les *VPL* sont utilisés pour éclairer de nombreux pixels voisins, le signal construit comme la somme de fonctions lisses reste lisse et non bruité. Cependant, le facteur de forme $G(y_k, x)$, fait intervenir un terme inversement proportionnel au carré de la distance ayant pour conséquence de provoquer des taches lumineuses à proximité des *VPL*. Pour réduire cet effet indésirable, la valeur de $G(y_k, x)$ est, en général, tronquée ou remplacée par une approximation du premier ordre ne présentant pas de singularité, *cf.* formule 2.19. Cela a tout de même pour conséquence d'introduire un biais dans l'image finale dont les solutions permettant de le corriger s'adaptent peu au rendu temps réel [Dachsbarcher et coll., 2014].

De part son efficace simplicité, l'approche par *VPL* a donné lieu à un certain nombre d'implémentations temps réel sur processeur graphique [Ritschel et coll., 2012]. En effet, en se concentrant uniquement sur un seul rebond lumineux, cette approche ne requiert que, d'une part, pouvoir échantillonner un maillage, et d'autre part, calculer l'éclairage par des sources ponctuelles. Dachsbarcher et Stamminger exploitent ainsi le pipeline de rasterisation pour construire des cartes d'ombre reflectives [Dachsbarcher et Stamminger, 2005, Dachsbarcher et Stamminger, 2006]. L'idée consiste à projeter la scène depuis le point de vue des sources lumineuses de manière analogue au calcul des cartes d'ombre [Williams, 1978]. Au lieu de n'enregistrer que la profondeur, cette fois-ci ils proposent de stocker également la normale et la couleur du matériau sous chaque pixel. La carte ainsi produite par pure rastérisation peut alors être vue comme une liste de *VPL* qui sont utilisés pour éclairer la scène. Afin de limiter le nombre de *VPL*, Prutkin et coll. [Prutkin et coll., 2012] proposent de d'agglomérer ensemble les pixels similaires grâce à une itération de la technique *mean shift* [Comaniciu et Meer, 2002]. Le nombre réduit de sources lumineuses permet d'obtenir des capacités temps réel. Par ailleurs, pour passer à l'échelle sur des scènes complexes de plusieurs millions de sources, Olsson et coll. [Olsson et coll., 2012] proposent de regrouper ensemble les pixels similaires du tampon géométrique (ou *G-Buffer* en anglais, *cf.* §2.2.4). Cela permet de

factoriser le fait de déterminer les sous-ensembles de *VPL* affectant ces groupes de pixels, qui nécessite, autrement, beaucoup de calculs redondants lorsqu'ils sont réalisés directement à l'échelle du pixel. Cependant, cette approche, ayant recours à des structures d'arbres, ne peut être utilisée dans un scénario purement dynamique. Enfin, l'unité de rastérisation n'est pas la seule permettant d'échantillonner le maillage. En ce sens, on note que l'unité de tessellation peut également remplir ce rôle. Nalbach et coll. [Nalbach et coll., 2014] proposent ainsi un pipeline dynamique pour échantillonner le maillage en espace objet s'abstrayant ainsi de la contrainte sur la nature de la géométrie.

Enfin, un dernier problème à résoudre dans le cadre de la radiosité instantanée est le calcul de visibilité entre les *VPL* et les pixels, c'est-à-dire le terme $V(y_k, x)$ dans la formule 3.2. Nombre de techniques ont été développées dans le but de calculer ce terme en temps réel. En cela, Ritschel et coll. [Ritschel et coll., 2008b] proposent de construire une multitude des petites cartes d'ombre imprécises, à la manière d'une carte d'ombre classique, mais en utilisant un nuage de points pour représenter la scène. En utilisant par ailleurs le pipeline de rastérisation, ils proposent de construire l'ensemble de ces cartes en parallèle, se servant du *Vertex Shader* pour distribuer et projeter les points sur des *caches* positionnés autour des *VPL*. Par ailleurs, en généralisant l'idée de *VPL* à des petites surfaces, Dong et coll. [Dong et coll., 2009] proposent de réduire le nombre de requêtes en calculant plutôt des ombres douces pour ces petites surfaces. De plus, afin d'amortir le coût de calcul des ombres pour les *VPL*, Laine et coll. [Laine et coll., 2007] proposent, quant à eux, de choisir un sous ensemble de *VPL* à chaque trame pour lesquels les ombres sont effectivement mise à jour. Olsson et coll. [Olsson et coll., 2014] proposent un pipeline complet de calcul optimisé de l'ombrage indirect. Pour cela, ils exploitent les fonctionnalités matérielles de *cache* virtuelle de tableaux de texture cubique n'allouant que la mémoire effectivement nécessaire à la résolution de la visibilité. Barák et coll. [Barák et coll., 2013] optent pour une approche par analyse d'image de la carte d'ombre réfléctrice. Cela leur permet ainsi de positionner préférentiellement les *VPL* dans les régions de plus forte énergie grâce à une approche de Métropolis en espace image. Cette idée d'échantillonnage préférentiel est également partagée par Eismann et coll. [Ritschel et coll., 2011] où l'extraction des *VPL* se fait à l'aide d'une mesure de probabilité dépendant de la distance et de leur influence sur la scène. Enfin, plus récemment, Hedman et coll. [Hedman et coll., 2016] proposent une technique permettant de conserver une cohérence temporelle sur l'ensemble de la scène grâce à une liste de liens entre les *VPL* et un sous ensemble de pixels représentatifs, visibles à leur création. Un *VPL* n'est alors défaussé que lorsque tous les pixels auquel il est attaché sont sortis du champs de vision, réduisant ainsi l'effet de clignotement propre aux approches non déterministes.

3.2.4 Éclairage en espace image

L'unité de rastérisation constitue un outil extrêmement efficace pour échantillonner la surface, cf. §2.2. On distingue ainsi une importante classe d'algorithmes, capable de simuler très rapidement les interactions lumineuses, fondée sur l'espace image, c'est-à-dire l'ensemble des pixels vus depuis la caméra.

Dans cette optique, Mittring [Mittring, 2007] propose initialement de calculer l'occlusion ambiante en utilisant les pixels de la carte de profondeur comme une approximation de la géométrie. Cette technique tire doublement profit de son implémentation sur processeur graphique car, d'une part, l'accès aux pixels d'une texture est réalisé en temps constant et, d'autre part, aucun surcoût n'est engendré par la construction de la carte de profondeur qui doit être construite, quoi qu'il

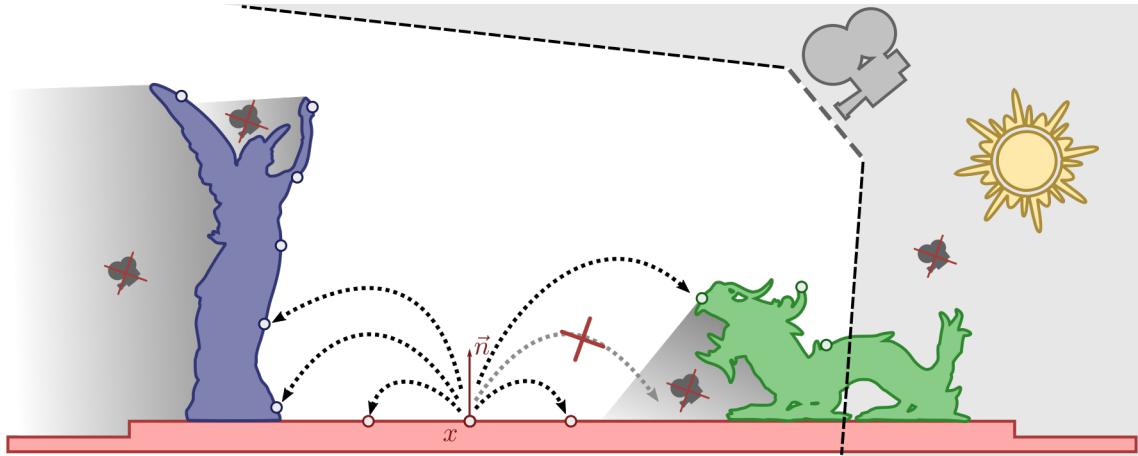


FIGURE 3.5 – Éclairage global en espace image. Pour synthétiser la couleur du pixel en x on regarde la luminance contenue dans les pixels voisins en espace image (points centrés de blanc). La contribution lumineuse est calculée de manière analogue à la radiosité instantanée. Certaines zones ne peuvent être échantillonnées car elles ne sont pas visibles depuis la caméra mais peuvent cependant influencer la scène (sous le dragon vert par exemple).

arrive, pour la rastérisation. Dans le même objectif, Bavoil et coll. [Bavoil et coll., 2008] utilisent la carte de profondeur pour évaluer une élévation moyenne autour des pixels démontrant alors une estimation plus stable de l’occlusion ambiante. Plus tard, Ritschel et coll. [Ritschel et coll., 2009b] ainsi que Soler et coll. [Soler et coll., 2010] proposent d’étendre ce mécanisme en échantillonnant cette fois-ci le tampon géométrique pour simuler un rebond diffus lumineux. En enregistrant ainsi la normale et la luminance réfléchie, les pixels voisins sont vu comme des VPL, ou des éléments finis, transmettant la lumière [Nichols et coll., 2009].

Utiliser l'espace image pour échantillonner la scène présente un autre avantage, car, pour des raisons de perspective, la résolution est plus importante à proximité de la caméra tandis qu'elle est plus faible à grande distance. Les régions proches de la caméra étant, en général, les plus importantes, ce processus adapte naturellement le niveau de résolution. Malgré tous ses avantages, cette technique apporte avec elle deux défauts majeurs.

Tout d'abord, bien que les pixels puissent être lus en temps constant, il est parfois nécessaire d'accéder à des zones très éloignées en mémoire car les distances en espace monde sont décorrélées de celle en espace image. Ainsi, même en limitant la distance significative du transport de la lumière en espace monde, il peut être nécessaire de parcourir un grand nombre de pixels, réduisant ainsi les performances ou générant des artefacts visuels dépendant du point de vue. Cette situation est néanmoins souvent résolue en calculant plusieurs niveaux de résolution de l'image et en adaptant le niveau que l'on échantillonne à l'échelle du phénomène que l'on souhaite simuler [Nichols et Wyman, 2009, Nichols et coll., 2010].

Cependant, le principal problème se situe dans le fait qu'un échantillonnage de la scène dans l'espace image manque un grand nombre d'informations absentes de l'écran mais dont l'influence est visible, *cf.* figure 3.5. En cela, la géométrie en dehors du champ de vision, celle cachée par des éléments plus proche ou encore la géométrie vue sous un angle rasant ne peut être capturée par la rastérisation. Pour gérer le cas où la scène sort du champ de vision, il est communément accepté d'élargir virtuellement le champ de vision et d'atténuer l'importance de l'éclairage indirect

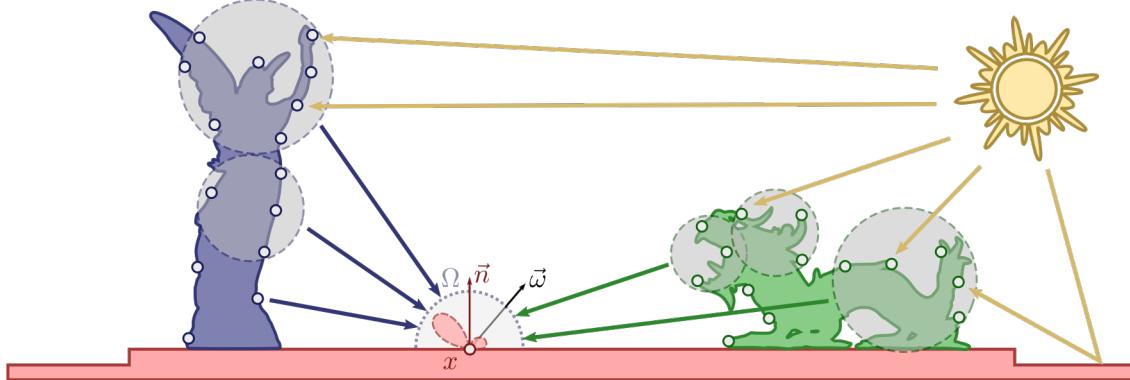


FIGURE 3.6 – Approche hiérarchique pour l'éclairage global. De manière analogue à la radiosité instantanée, la scène est discrétisée en *VPL* (points centrés de blanc) dont la luminance est calculée à partir de l'éclairage direct. Cependant, ces *VPL* sont structurés sous forme d'arbre. Pour le calcul de la couleur du pixel x , une représentation « optimale » de l'éclairage incident est construite. Celle-ci est réalisée en extraignant directement des *VPL* ou des groupes de *VPL* (cercles pointillés) à partir de l'arbre de sorte à ce que l'éclairage incident construit par cette approximation soit « proche » de l'éclairage issue de tous les *VPL*. La fonction de luminance $I_k(-\omega_i)$ de chaque nœud intermédiaire est calculée une seule fois et est telle qu'elle approche « convenablement » l'ensemble des *VPL* sous-jacents.

dépendant d'éléments s'approchant du bord. Pour gommer les problèmes liés à un sous échantillonnage directionnel, Vardis et coll. [Vardis et coll., 2013] proposent de construire des rendus sous plusieurs points de vue et de se servir d'une métrique liée à l'orientation pour déterminer les directions pour lesquelles un élément possède une influence significative. Le cas généralement traité est celui dit de la *désocclusion*, c'est-à-dire lorsqu'un objet dont l'influence est notable sur la scène est caché par un autre et devient soudainement visible. Pour éviter cette situation, il est nécessaire de calculer et de conserver plusieurs couches de profondeur de la scène vue depuis la caméra. Pour cela, on recourt alors souvent à la technique d'épluchage de la profondeur (ou *depth peeling* en anglais) [Bavoil et Sainz, 2009, Ritschel et coll., 2009b]. Cependant, ces dernières approches nécessitent plusieurs rendu de la scène car il n'existe pas d'approche triviale, telle que la construction d'une carte de profondeur, pour extraire les deux, voire les k , plus proches valeurs en une seule passe de rendu. Par ailleurs, selon leur étude, Mara et coll. [Mara et coll., 2016] montrent que près d'un tiers du temps nécessaire à la construction d'une image est passé dans des étapes de préparation du rendu, comme le test d'occlusion du graphe de scène, la résolution des animations, *etc.* et produire plusieurs rendus peut devenir très couteux. Pour palier ce problème, ils proposent d'utiliser la cohérence temporelle des trames pour prédire la position des fragments dans la trame suivante et ainsi réaliser en un seul rendu le calcul du second fragment le plus proche.

3.2.5 Lumières Multiples et Approches Hiérarchiques

Les approches de type lumières multiples (ou *Many-Light* en anglais) [Dachsbaecher et coll., 2014] forment un cadre formel de calcul généralisant la radiosité instantanée. Ainsi, elles visent à calculer l'éclairage d'une scène soumise à un grand nombre de sources lumineuses indépendamment de la façon dont leur intensité a été calculée. Initialement introduit par Walter [Walter et coll., 2005, Walter et coll., 2006], les coupes de lumière (ou *Light Cuts*) proposent une

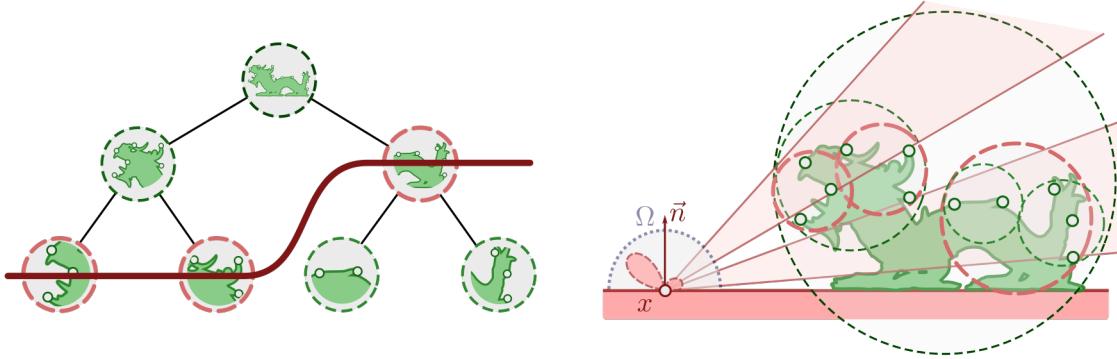


FIGURE 3.7 – Coupe de lumière. L’ensemble des *VPL* (points blancs cerclés de vert) sont hiérarchiquement regroupés dans un arbre (à gauche). Chaque nœud de l’arbre est un regroupement de *VPL* et définit une fonction de luminance approchant au mieux la luminance associée. Pour chaque pixel x , on choisit un ensemble de cônes de directions puis on parcourt l’arbre depuis la racine jusqu’à obtenir un ensemble de nœuds qui produisent un éclairage convenable pour ce pixel. Ici, le critère d’arrêt se fonde sur l’angle solide du nœud. Ainsi, ce dernier est subdivisé, s’il est vu depuis x sous un angle solide trop important. L’ensemble des nœuds choisis pour représenter l’éclairage est appelé coupe (courbe rouge à gauche).

approche hiérarchique d’éclairage dans le but de passer à l’échelle vis à vis des calculs nécessaires pour traiter une masse importante de sources lumineuses. L’idée consiste à construire un arbre sur l’ensemble des lumières, les regroupant hiérarchiquement en fonction de la nature des sources, c’est-à-dire leur position, orientation, couleur *etc.* Chaque nœud intérieur continent alors une distribution de luminance approchant au mieux l’ensemble des sources lumineuses sous-jacentes. Pour calculer l’éclairage en un point, il est alors nécessaire d’extraire une coupe de cet arbre, c’est-à-dire une collection de nœuds telle que chaque branche de l’arbre est coupée par exactement un nœud de la coupe, pour ensuite sommer la contribution lumineuse représentée par l’ensemble des nœuds de cette coupe, *cf.* figure 3.6. La construction de la coupe est réalisée pour chaque pixel que l’on souhaite éclairer de sorte à ce qu’elle représente le plus fidélement possible l’éclairage local avec un minimum de sources, *cf.* figure 3.7. Pour cela, on descend l’arbre depuis la racine, et pour chaque nœud, on parcourt ses enfants seulement si la différence de l’éclairage produit par ce nœud par rapport à celui produit par ses enfants est plus grande qu’un seuil. Bien que de cette façon, on introduise un biais sur l’image, cette approche se justifie par le fait que le nombre de calculs peut être considérablement réduit, alors qu’en choisissant convenablement le seuil, davantage de précision ne serait pas perçu visuellement. En effet, la loi de Weber-Fechner montre de manière empirique que le système sensoriel humain n’est pas en mesure de percevoir des variations d’intensité lumineuse inférieure à 1% [Blackwell, 1972]. Cela signifie qu’un signal lumineux ayant une erreur relative d’au plus 1% par rapport à une vérité terrain n’est en général pas perçue. Selon Walter et coll. [Walter et coll., 2005], ce seuil est atteint dans le pire cas tandis qu’en pratique, ils constatent qu’aucune différence n’est perçue si l’erreur relative demeure inférieure à 2%.

Les approches hiérarchiques par points, introduites par Christensen [Christensen, 2008] avec *PBGI* (pour *Point Based Global Illumination*) et Ritschel et coll. [Ritschel et coll., 2009a] avec le *micro-rendu*, exploitent également l’idée de représenter l’éclairage sous plusieurs niveaux de résolution. Cependant, à la différence des coupes de lumières, la métrique utilisée pour choisir le nœud se fonde uniquement sur l’angle solide du regroupement de points. La visibilité est alors résolue grâce à une étape de rasterisation dans une carte cubique de manière similaire aux hémicubes. Avec *ManyLods*, Hollander et coll. [Hollander et coll., 2011] proposent une implémentation temps

réel pour calculer efficacement les coupes à l'aide du pipeline graphique. Il se servent ainsi des extensions du *Geometry Shader* pour construire des listes de sommets en restant sur le processeur graphique. De plus, ils tirent profit de la cohérence temporelle pour éviter de reconstruire entièrement leurs coupes se contentant de mettre à jour leur calcul d'une trame à l'autre. Maletz et Wang [Maletz et Wang, 2011] proposent, quant à eux, une approche stochastique pour extraire leurs nœuds représentatifs grâce à une métrique sur l'ensemble de groupe de *VPL*. Ce choix étant réalisé en parcourant linéairement chaque groupe, l'algorithme se passe ainsi du besoin d'arbre. Plus récemment, Nalbach et coll. [Nalbach et coll., 2015] proposent une approche entièrement fondée sur le pipeline de rastérisation, utilisant à la fois l'unité de tessellation pour échantillonner la géométrie ainsi que le test de profondeur pour former un micro tampon stockant les plus proches distances sur plusieurs directions aléatoires par fragment.

Dans l'idée de filtrer la luminance selon plusieurs niveaux de résolution, Crassin et coll. [Crassin et coll., 2011] proposent, quant à eux, d'utiliser une structure d'*octree* démontrant une implémentation temps réel sur processeur graphique. Ils proposent ainsi de parcourir les nœuds de l'arbre (des *voxels*) qui appartiennent à un cône issu du pixel. En fonction de la distance et d'une représentation de l'opacité de chaque nœuds, ils calculent ainsi la contribution lumineuse du champ de luminance tout en capturant également la visibilité.

3.3 Synthèse

Dans ce chapitre nous avons passé en revue un ensemble d'approches visant à calculer l'éclairage global que nous avons classées en deux grandes catégories, les approches par lancer de chemins et les approches fondées sur les *caches*.

Lancer de chemins La première catégorie constitue une approche de référence, notamment en ce qui concerne le rendu hors ligne de précision. En cela, elle propose un cadre de calcul permettant de simuler presque tous les phénomènes, pourvu que l'on puisse les décrire et y consacrer le temps nécessaire. De plus, nous avons vu que le lancer de chemins se dérive également sous des implémentations en temps réel grâce la facilité de sa parallélisation. Cependant, nous avons également cités trois problèmes majeurs, intrinsèquement liés à cette approche, qui demeurent difficile à traiter en temps réel. Tout d'abord, dû au faible nombre de rayons qu'il est possible d'envoyer, les images sont rendues avec un fort bruit aléatoire qui est difficile à supprimer. Ensuite, le fait que le test de visibilité se fonde sur le lancer de rayons, il est nécessaire de construire et maintenir une structure d'accélération généralement peu compatible avec les scénarios dynamiques. Enfin, à cause du caractère aléatoire des rayons, nous avons vu que matériellement, les accès mémoire du lancer de rayons ont tendance à former des schémas très incohérents et peu compatibles naturellement avec les processeurs graphique. Cela rend leur implémentation très compliqués et peut engendrer des surcoûts prohibitifs de préparation des calculs.

Approches fondées sur les *caches* Ces dernières contraintes nous ont donc poussés à explorer l'autre grande classe d'algorithmes que nous avons qualifiée de « fondée sur les *caches* ». Tout comme la précédente catégorie, ces algorithmes visent également à construire des chemins entre les pixels et les sources primaires pour synthétiser le transport lumineux, mais cette fois-ci, un certain nombre de calculs sont factorisés grâce à ce que nous avons appelé *cache*. Nous avons ainsi vu qu'il est nécessaire de résoudre trois principaux problèmes liés à cette approche.

Tout d'abord, il est nécessaire de définir la nature de l'information qu'un *cache* est censé stocker. En cela, les méthodes par éléments finis substituent la géométrie par des éléments de surfaces simples, et les transferts lumineux sont capturés par la matrice des facteurs de forme. Les méthodes de radiosité instantanée, quant à elles, définissent un ensemble de points, les *VPL*, stockant la luminance incidente pour simuler le dernier rebond, souvent diffus pour le temps réel. Pour interpoler le champ lumineux, nous avons vu qu'il est également possible de définir d'autres bases de fonctions prenant en compte la variation spatiale et directionnelle pour améliorer la qualité des *caches* et réduire leur nombre. La représentation par différents niveaux de résolution du champ lumineux permet également aux approches de type lumières multiples de passer à l'échelle sur un nombre très important de sources.

Le second problème à résoudre, notamment pour les approches temps réel, réside dans l'échantillonnage même de la scène et le positionnement des *caches*. Cette problématique est en général résolue par un traitement préliminaire sur la géométrie supposant que celle-ci restera relativement statique au cours du rendu. Ainsi, les approches par éléments finis et hiérarchiques tirent profit de cette hypothèse pour accélérer, plus tard, les calculs d'éclairage. On note, cependant, l'exception notable des approches fondées sur l'espace image qui utilisent l'unité de rastérisation pour construire un échantillonnage de manière très efficace de la scène. Cette échantillonnage peut même être considéré comme gratuit s'il s'agit de réutiliser l'image produite pour le point de vu principal. Bien que cette approche permette de prendre en compte absolument toute géométrie qui peut être rendu, elle manque un certain nombre d'informations dû à la projection.

Enfin, le troisième problème que nous avons évoqué est le calcul de la visibilité entre les *caches* et les pixels. Nous nous sommes ici intéressés aux phénomènes de basse fréquence liés à la visibilité indirecte, car le champ complet de ce domaine dépasse le cadre de cette thèse et est généralement orthogonal à la stratégie utilisée pour calculer le *cache* de luminance. Pour obtenir ou évaluer la fonction de visibilité, il est souvent suffisant d'obtenir une représentation moins riche de la scène, ce que les approches en espace image parviennent très bien à faire. Cependant, il est aussi notable que la visibilité peut être implicitement prise en compte telle que dans les approches par éléments finis ou hiérarchiques qui, en sélectionnant les *caches* les plus proches, excluent de fait le besoin de visibilité.

Notre Approche Au vu de la problématique évoquée ci-dessus, nous tentons de répondre, par la suite, aux trois problèmes liés aux approches fondées sur les *caches*. En effet, de par les scénarios que nous cherchons à traiter, nous pensons qu'il n'est pas souhaitable de devoir construire et maintenir une structure d'accélération. Ainsi nous proposons dans un premier temps une nouvelle représentation du champ lumineux, inspirée des approches hiérarchiques de lumières multiples. En construisant les coupe de manière stochastique, notre approche s'affranchit cependant du besoin d'arbre. De plus, comme nous nous intéressons aux scénarios temps réel dynamiques, dont l'environnement lumineux, la géométrie et le point de vue peuvent changer à chaque trame, nous nous restreignons à la simulation d'un rebond diffus. Dans un second temps, nous proposons une approche entièrement fondée sur le pipeline graphique utilisant notre nouvelle représentation pour échantillonner les *VPL* à partir de la scène et calculer en temps réel, à la volée, l'éclairage indirect. Cette première approche ne prend pas en compte la visibilité. C'est pourquoi dans un dernier chapitre, nous proposons d'étendre notre pipeline par la construction d'une structure de donnée semi-dynamique, visant à calculer de manière très efficace, bien qu'imprécise, la visibilité entre nos *VPL* et nos pixels.

COUPES ANTÉROGRADES DE LUMIÈRES

Dans ce chapitre nous proposons une nouvelle approche permettant de calculer une approximation de l'opérateur de transport défini précédemment, *cf.* §2.1.4. Inspirée du formalisme des lumières multiples, notre technique se fonde sur un échantillonnage à plusieurs échelles de la scène pour optimiser l'efficacité du calcul de l'intégrale en fonction de la distance parcourue par un rebond de la lumière. Cependant, à l'inverse des précédentes techniques, nous proposons une approche probabiliste pour construire la hiérarchie de lumière. Cela nous permet de fournir une implémentation libre de toute structure d'accélération sur les données et donc d'être parfaitement adaptée au calcul parallèle à grain fin des processeurs graphiques.

4.1 Introduction

Telles que nous les avons introduites au chapitre précédent, les coupes d'arbre forment un outil très efficace pour répondre à deux problèmes majeurs liés à la simulation du transport lumineux : la minimisation de l'utilisation des ressources de calcul d'une part, et l'utilisation optimale de celles-ci d'autre part. En ce sens, on montre que, pour produire un résultat convenable, la précision de la source de lumière n'a pas besoin d'être identique partout dans la scène quel que soit le receveur. En outre, il ressort que l'éclairement en un point par une source surfacique relativement éloignée peut être fidèlement capturé à partir de seulement quelques échantillons grossiers, tandis que, lorsque la source et le receveur deviennent proches, il est nécessaire d'échantillonner fortement la source pour obtenir un résultat fiable. Comme le font remarquer Durand et coll. [Durand et coll., 2005], *cf.* §3.2.2, ce constat peut s'expliquer de manière fréquentielle. En effet, plus l'évaluation de la lumière se fait loin de la source, plus le signal est de basse fréquence spatiale, ce qui signifie qu'il peut être reconstruit à l'aide d'une densité d'échantillonnage plus faible.

Les approches hiérarchiques qui tirent profit de ce constat, *cf.* §3.2.5, nécessitent de fait, une certaine structuration des données dans le but de calculer les coupes de lumière. Cela se traduit algorithmiquement par la construction et la mise à jour d'arbres sur l'ensemble des *VPL*. Cependant, l'utilisation de telles structures de données a des conséquences fortes sur la manière dont doit être préparée la scène, ce qui est dommageable à la fois pour les scénarios dynamiques et pour la généralité d'utilisation de l'approche. Cela implique qu'un algorithme se fondant sur des structures complexes rend son utilisation plus ou moins difficile en fonction du contexte.

Dans ce chapitre, nous développons une solution exploitant les avantages relatifs aux approches multi-échelles de répartition optimale des calculs tout en s'affranchissant de la nécessité de

construire une structure de données sur la scène. Pour cela, nous introduisons la notions de *coupe antérograde* (ou encore *coupe vers l'avant*) par opposition aux coupes « traditionnelles » pouvant quant à elles être qualifiées de *coupe rétrograde* (ou encore *coupe vers l'arrière*). Cette distinction se fonde sur la façon d'explorer le chemin lumineux. En ce sens, on évoque l'idée de « vers l'avant » lors que l'on considère les chemins lumineux partant de la lumière vers les capteurs et l'idée de « vers l'arrière » pour les parcours dans l'autre sens. Tout comme les approches par lumières multiples, nous partons d'un nuage de *VPL* construit aléatoirement et réparti à la surface de la scène. Nous l'utilisons alors comme *cache* afin de capturer et retransmettre l'éclairage afin de simuler un rebond lumineux. Grâce à un modèle statistique de la scène, nous construisons au dessus du nuage de *VPL* une succession de sous-échantillonnages multi-échelles de la géométrie permettant de définir des familles de *VPL* dont les zones d'influence sont bornées dans l'espace tout en construisant un estimateur sans biais de l'éclairage, cf. §4.2. Nous généralisons ensuite notre approche en faisant varier continûment la densité de notre échantillonnage, cf. §4.3. Bien que la méthodologie présentée ici puisse être appliquée au transport général de la lumière, nous nous focalisons uniquement à l'étude du premier rebond diffus.

4.2 Coupes antérogrades discrètes

Étant donné un champ de luminance L défini à la surface de la géométrie \mathcal{S} , on cherche à calculer le résultat d'une application de l'opérateur de transport \mathcal{T} , cf. §2.1.4, sur ce champ. Autrement dit on cherche à calculer la luminance réfléchie par la scène. On note le champ résultant de cette opération \mathcal{T}_L :

$$\forall x \in \mathcal{S}, \vec{\omega} \in \Omega(x), \quad \mathcal{T}_L(x, \vec{\omega}) = \int_{y \in \mathcal{S}} L(y \rightarrow x) h(y, x, \vec{\omega}) \, dS_y. \quad (4.1)$$

On utilise les notations définies précédemment, cf. §2.1.4, pour la fonction de transfert $h(y, x, \vec{\omega})$ définissant la quantité de lumière transportée entre y et x et retransmise en direction de $\vec{\omega}$.

Si l'on se donne également un échantillonnage de la surface dont les échantillons (y_1, \dots, y_N) suivent la loi de densité $\mu^{\mathcal{S}}$, on peut alors définir un estimateur $\hat{\mathcal{T}}_{L,N}$ de \mathcal{T}_L par :

$$\forall x \in \mathcal{S}, \vec{\omega} \in \Omega(x), \quad \hat{\mathcal{T}}_{L,N}(x, \vec{\omega}) = \frac{1}{N} \sum_{n=1}^N \frac{L(y_n \rightarrow x) h(y_n, x, \vec{\omega})}{\mu^{\mathcal{S}}(y_n)}. \quad (4.2)$$

Il s'agit d'un estimateur non biaisé, cf. §2.1.5, dans le sens où, en espérance sur le choix de l'échantillonnage, ces deux valeurs sont égales. En outre, si l'on augmente le nombre d'échantillons N , on a bien $\hat{\mathcal{T}}_{L,N} \xrightarrow[N \rightarrow \infty]{} \mathcal{T}_L$ en tout point.

$\hat{\mathcal{T}}_{L,N}$ formalise ainsi l'idée de pouvoir discréteriser une surface émettant de la lumière par un nombre fini de points, les $(y_n)_n$ que l'on nomme *VPL*, afin de pouvoir approcher le calcul d'un rebond lumineux. Pour les mêmes raisons que pour les approches par lumières multiples, le champ réfléchi $\hat{\mathcal{T}}_{L,N}$ est la somme de la contributions N *VPL*. Cependant, pour obtenir un résultat convenable, N est en général très grand, de l'ordre de plusieurs centaines de milliers, voire millions. Or, pour un point donné de l'espace, tous les *VPL* ne contribuent pas avec la même importance dans le calcul de la valeur finale de luminance, ce qui rend le calcul de la somme ci-dessus inutilement compliqué. Pour répondre à ces deux problèmes, nous définissons un nouvel estimateur non biaisé de \mathcal{T}_L grâce à la notion de coupe antérograde qui, s'inspirant de l'éclairage par lumières multiples, propose une meilleure répartition des calculs.

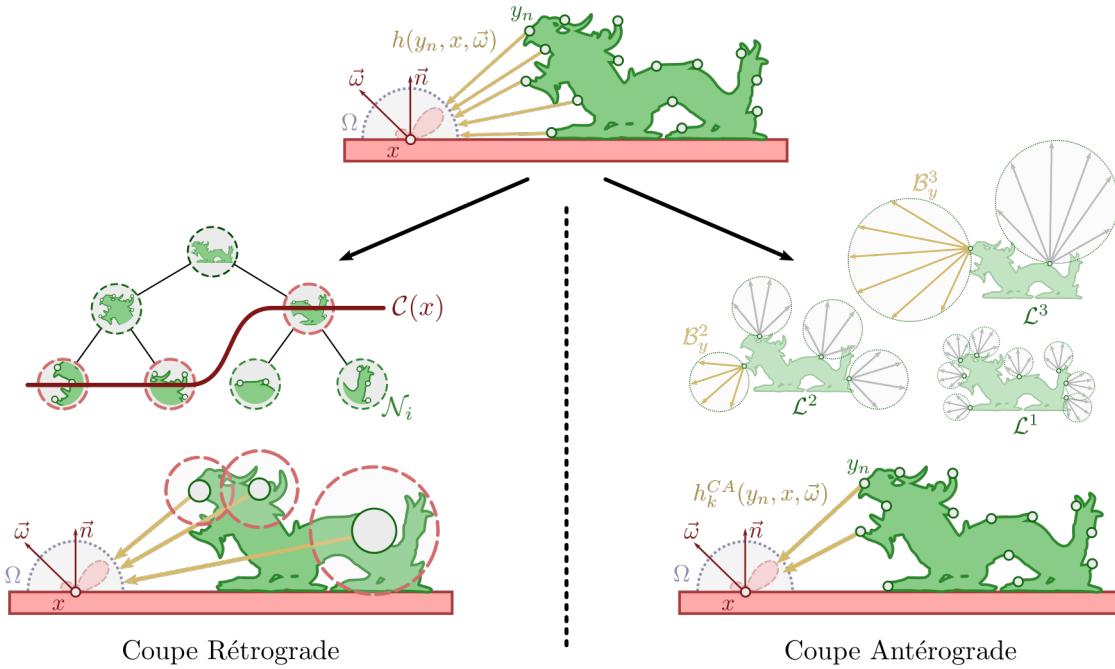


FIGURE 4.1 – Coupe Rétrograde et Coupe Antérograde. En haut, l'éclairage de référence au point x est calculé à partir de la contribution de tous les *VPL*, y_k .

À gauche, pour approcher ce calcul, un arbre est construit sur l'ensemble des *VPL* où chaque nœud \mathcal{N}_i forme une représentation grossière des *VPL* sous-jacents. La coupe $\mathcal{C}(x)$ est alors calculée pour chaque point x en extrayant des nœuds de cet arbre de sorte à ce qu'ils représentent au mieux l'éclairage.

À droite, le support spatial de chaque *VPL* est calculé au moment de son échantillonnage, en imitant le comportement d'un arbre. Un échantillonnage grossier, \mathcal{L}^3 , avec un grand support \mathcal{B}^3_y , permet de représenter l'éclairage distant tout comme les nœuds proches de la racine. De même un échantillonnage fin, \mathcal{L}^1 , avec un petit support \mathcal{B}^1_y , permet de représenter l'éclairage local tout comme les nœuds proches des feuilles. L'éclairage en x est issu de la contribution de tous les *VPL* dont le support contient x .

4.2.1 Coupe Antérograde

Étant donné un arbre, une *coupe* correspond à un sous-ensemble de nœuds de cet arbre qui respectent un certain prédictat, cf. §3.2.5. Pour une *coupe de lumière*, ce prédictat est fonction d'un certain nombre de paramètres et notamment la position et l'orientation du récepteur ainsi que de la topologie de l'arbre. En effet, en parcourant de la racine vers les feuilles, les fils d'un nœud ne sont explorés que si son prédictat n'est pas vérifié; autrement il est ajouté à la liste qui forme la coupe. En pratique, une coupe est calculée pour chaque récepteur, en général pour chaque pixel, cf. figure 4.1.

La raison d'être de notre *coupe antérograde* est de pouvoir être calculée à partir des *VPL* directement et indépendamment les uns des autres afin de s'adapter à un calcul parallèle à grain fin. Pour cela, nous ne pouvons plus recourir à la notion d'arbre car les nœuds sont alors très corrélés entre eux et le calcul parallélisé devient compliqué et potentiellement très lent. Nous proposons donc de remplacer la structure hiérarchique par plusieurs jeux d'échantillonnages à différentes fréquences de plus en plus grossières. Comme nous l'avons évoqué précédemment, en fonction de la distance à la source, le signal lumineux devient de plus en plus basse fréquence et peut donc être représenté par un échantillonnage de plus faible densité. Bien que discutable, cette hypothèse est relativement

valable dans le cadre d'un transport diffus, c'est pourquoi nous nous limitons à ce cas dans ce chapitre. Dans ce scénario, nos différents niveaux d'échantillonnages sont donc utilisés pour calculer l'éclairage efficacement aux zones de l'espace où ils contribuent le plus.

À partir de cette multi-résolution, nous construisons tout d'abord, *cf.* §4.2.2, un premier estimateur non biaisé du transport lumineux à partir d'un jeu d'échantillonnages à plusieurs échelles. Cet estimateur est régi par une équation sur une famille de fonctions, que nous nommons les fonctions de support. Nous analysons, dans un second temps, *cf.* §4.2.3, les variations de l'opérateur de transport pour proposer une première solution. Puis, à partir de celle-ci, nous formalisons un second estimateur, *cf.* §4.2.4, s'inspirant de la définition des fonctions de supports mais fondé sur un unique échantillonnage pour favoriser le calcul parallèle. Enfin, nous présentons et discutons nos résultats, *cf.* §4.2.6.

4.2.2 Estimateur par coupes antérogrades

On se donne ici K échantillonnages de \mathcal{S} , $\mathcal{L}^1, \dots, \mathcal{L}^K$, tels que $\mathcal{L}^k = \left\{ y_1^k, \dots, y_{N_k}^k \right\}$ où $(y_n^k)_{n,k}$ forme une famille d'échantillons indépendants identiquement distribués selon la loi de densité $\mu^{\mathcal{S}}$. En introduisant les fonctions inconnues w_k , on définit alors notre estimateur comme :

$$\forall x \in \mathcal{S}, \vec{\omega} \in \Omega(x), \quad \hat{\mathcal{T}}_L^{\text{CAL}}(x, \vec{\omega}) = \sum_{k=1}^K \frac{1}{N_k} \sum_{y_n^k \in \mathcal{L}^k} h_k^{\text{CA}}(y_n^k, x, \vec{\omega}) \frac{L(y_n^k \rightarrow x)}{\mu^{\mathcal{S}}(y_n^k)} \quad (4.3)$$

et

$$h_k^{\text{CA}}(y_n^k, x, \vec{\omega}) = w_k(y_n^k, x) h(y_n^k, x, \vec{\omega}).$$

Cette formulation nous permet de construire une nouvelle fonction de transfert $h_k^{\text{CA}}(y_n^k, x, \vec{\omega})$ faisant intervenir un nouveau terme w_k . Nous utilisons ces fonctions pour porter l'information de support spatial de chaque *VPL*. Cependant, pour produire un estimateur sans biais, les w_k doivent respecter l'égalité suivante :

$$\forall y \in \mathbb{R}^3, \forall x \in \mathcal{H}^*(y, \vec{n}_y), \quad \sum_{k=1}^K w_k(y, x) = 1, \quad (4.4)$$

où $\mathcal{H}^*(y, \vec{n}_y)$ représente le demi-espace $\{x \in \mathbb{R}^3, \langle \bar{y}x, \vec{n}_y \rangle > 0\}$. Dans la suite, lorsque y appartient à \mathcal{S} , nous notons sans ambiguïté cet ensemble $\mathcal{H}^*(y)$ en prenant comme normale celle définie sur la surface en ce point. De plus, comme le terme de transfert h ne contribue que sur ce demi-espace, il est suffisant de restreindre la contrainte sur les w_k pour les valeurs de x dans ce même ensemble. En dehors de $\mathcal{H}^*(y)$ nous les supposons nulles.

Enfin, la contrainte définie par la formule 4.4 est nécessaire et suffisante pour que $\hat{\mathcal{T}}_L^{\text{CAL}}$ soit un estimateur non biaisé de \mathcal{T}_L . En effet :

$$\begin{aligned}
\mathbb{E} \left[\hat{\mathcal{T}}_L^{\text{CAL}} \right] (x, \vec{\omega}) &= \sum_{k=1}^K \frac{1}{N_k} \mathbb{E} \left[\sum_{y_n^k \in \mathcal{L}^k} w_k(y_n^k, x) \frac{L(y_n^k \rightarrow x) h(y_n^k, x, \vec{\omega})}{\mu^S(y_n^k)} \right]_{y_k \sim \mu^S} \\
&= \sum_{k=1}^K \mathbb{E} \left[w_k(y, x) \frac{L(y \rightarrow x) h(y, x, \vec{\omega})}{\mu^S(y)} \right]_{y \sim \mu^S} \\
&= \mathbb{E} \left[\underbrace{\left(\sum_{k=1}^K w_k(y, x) \right)}_{= 1 \text{ si } h \neq 0} \frac{L(y \rightarrow x) h(y, x, \vec{\omega})}{\mu^S(y)} \right]_{y \sim \mu^S} \\
&= \mathbb{E} \left[\frac{L(y \rightarrow x) h(y, x, \vec{\omega})}{\mu^S(y)} \right]_{y \sim \mu^S} = \mathcal{T}_L(x, \vec{\omega}). \quad \square
\end{aligned}$$

Cependant, telle que nous l'avons défini, notre estimateur dépend d'une famille de fonctions inconnues $(w_k)_k$ n'étant contrainte que par une équation de partition de l'unité. Cette famille de fonctions est donc un paramètre libre de notre formulation pour lequel nous cherchons à présent une solution.

4.2.3 Reconstruction lumineuse diffuse par fonction de support

L'objectif de cette section est de donner une solution pour le calcul des $(w_k)_k$. Pour cela, nous nous restreignons au cadre de l'étude de l'éclairage diffus sans prendre en compte le terme de visibilité. On exprime alors l'estimateur de coupe antérograde diffus $\mathcal{T}_L^{\text{CAL}, d}$ par :

$$\forall x \in \mathcal{S}, \quad \mathcal{T}_L^{\text{CAL}, d} = \sum_{k=1}^K \frac{1}{N_k} \sum_{y_n^k \in \mathcal{L}^k} w_k(y_n^k, x) \frac{\langle \bar{x}y_n^k, -\vec{n}_y \rangle^+ \langle \bar{x}y_n^k, \vec{n}_x \rangle^+ L(y_n^k)}{\|x - y\|^2} \frac{L(y_n^k)}{\mu^S(y_n^k)}. \quad (4.5)$$

Pour interpréter les $(w_k)_k$ comme des fonctions définissant les régions d'influences de nos VPL, nous procédons comme suit. Plus k est grand, moins l'échantillonnage correspondant \mathcal{L}^k est dense, et plus les VPL qu'il contient sont utilisés pour éclairer des régions distantes. Nous progressons en deux temps pour formaliser cette idée.

Tout d'abord, en s'inspirant des coupes utilisées dans les approches hiérarchiques fondées sur les arbres, nous construisons un critère permettant de définir spatialement les régions d'influence des VPL. Il s'agit là de partitionner l'espace de définition de chaque VPL y_n^k , c'est-à-dire $\mathcal{H}^*(y_n^k) \subset \mathbb{R}^3$, en une famille d'ensembles $\mathcal{D}^k(y_n^k)$ recouvrant $\mathcal{H}^*(y_n^k)$.

Dans un second temps, pour chaque k , nous restreignons l'ensemble de définition de l'application $x \mapsto w_k(y_n^k, x)$ à l'une des parties définies plus tôt, en l'occurrence $\mathcal{D}^k(y_n^k)$. À l'intérieur de leur domaine de définition, nous faisons en sorte que les w_k respectent la condition de partition de l'unité. Pour cela, nous exploitons les symétries de $\mathcal{D}^k(y_n^k)$ pour réduire à 1 la dimension de l'espace de définition des w_k .

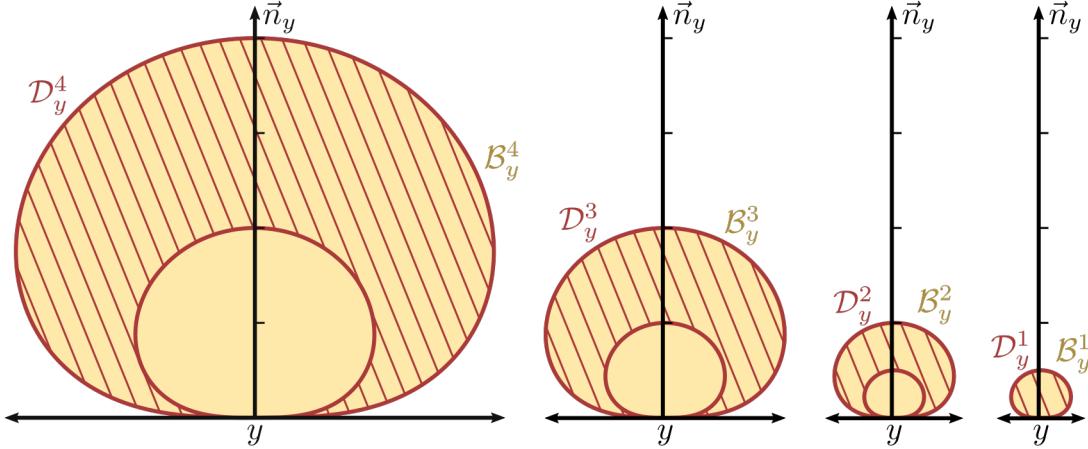


FIGURE 4.2 – Domaine de définition des fonctions de support. À chaque niveau de *VPL* \mathcal{L}^k est associé une fonction de support w_k dépendant de l'origine y du *VPL* et de son orientation \vec{n}_y . Le domaine de définition de cette fonction est alors inclus dans l'ensemble \mathcal{B}_y^k qui correspond à l'ensemble des points x de l'espace pouvant observer un disque de rayon $a < a_k$ autour de y sous un angle solide $\Omega_a(y \rightarrow x)$ égal à ε_Ω . Cependant, pour un niveau k , on considère que seuls des *VPL* ayant un disque de rayon $a \in [a_{k-1}, a_k]$ ont une importance. Cela permet ainsi de définir $\mathcal{D}_y^k(y)$ le véritable domaine de définition de la fonction de support. La borne inférieure permet ainsi de limiter la contribution d'un *VPL* issu d'un échantillonnage grossier de sorte à ce qu'il n'ait pas de contribution locale. L'éclairage des régions proches du *VPL* est alors assuré par les *VPL* des niveaux inférieurs et donc plus densément distribués.

Domaines de définition Comme nous nous plaçons dans le cadre d'une réflexion diffuse, il semble intéressant de définir les zones d'influences, $\mathcal{D}^k(y_n^k)$, à l'aide de la fonction d'angle solide. En effet, en voyant un *VPL* comme un petit disque lumineux diffus, d'aire a , celui-ci émet une quantité de lumière en x proportionnelle à son angle solide. Il est donc cohérent de faire intervenir cette grandeur pour définir nos partitions.

Pour cela, on peut considérer un disque a_n autour de y_n^k , segmenter l'ensemble des valeurs de $[0, 2\pi]$, et construire une partie de $\mathcal{H}^*(y_n^k)$, comme l'ensemble des points de l'espace voyant ce disque sous un angle solide compris dans l'un de ces segments. Dans notre approche en revanche, nous développons un critère légèrement différent pour construire notre partition. Pour cela, nous proposons de segmenter, plutôt, l'ensemble des valeurs d'aires des disques, c'est-à-dire $[0, \infty[$. Une partie $\mathcal{D}^k(y_n^k)$ de $\mathcal{H}^*(y_n^k)$ sera pour nous, les points de l'espace, voyant sous un angle solide constant, un disque autour du *VPL* dont l'aire est comprise dans l'un des segments, cf. figure 4.2. Grâce à cela, nous corrélons immédiatement l'idée de densité d'échantillonnage (alors inversement proportionnelle à l'aire des disques) avec la définition des supports. Nous formalisons à présent cette idée.

Dans ce contexte, nous notons Ω_a la fonction angle solide à l'ordre 1, que nous appelons simplement angle solide, d'un disque d'aire a , centré en y , de normale \vec{n}_y et observé depuis le point x , cf. §2.1.2,

$$\forall x, y \in \mathbb{R}^3, \quad \Omega_a(y \rightarrow x) = a \frac{\langle \bar{xy}, -\vec{n}_y \rangle^+}{\|y - x\|^2}. \quad (4.6)$$

De plus, en supposant données $K + 2$ aires $(a_k)_k$ telles que

$$0 = a_0 < a_1 < a_2 < \dots < a_K < a_{K+1} = +\infty, \quad (4.7)$$

on définit K fréquences d'échantillonnage de la surface de moins en moins denses et inversement proportionnelles aux $(a_k)_{k \in [1, K]}$.

Par analogie avec les approches par coupes, *cf.* figure 4.1, chacune de ces fréquences peut s'apparenter à un niveau \mathcal{L}^k d'un arbre dont les *VPL* correspondent aux nœuds de ce niveau. On suppose ici que le niveau \mathcal{L}^k est constitué de *VPL* représentés par un disque imaginaire dont l'aire est a_k . À partir de là, pour calculer la coupe d'un arbre, l'angle solide du disque peut être utilisé pour déterminer si un nœud doit être raffiné ou non. Ainsi, pour un receveur donné x , les nœuds de l'arbre sont parcourus tant que l'angle solide $\Omega_{a_k}(y_n^k \rightarrow x)$ du disque d'aire a_k associé au *VPL* y_n^k est supérieur à un certain seuil ε_Ω , lui-même défini par l'utilisateur. L'arbre est donc exploré tant que :

$$\Omega_{a_k}(y_n^k \rightarrow x) > \varepsilon_\Omega. \quad (4.8)$$

Dans notre approche, on cherche cependant à réaliser le calcul inverse, dans le sens où l'on doit définir l'ensemble $\mathcal{B}_{y_n^k}(\varepsilon_\Omega, a_k)$ des receveurs $x \in \mathbb{R}^3$ pour lesquels la condition ci-dessus est vérifiée. Pour calculer cet ensemble il nous faut donc déterminer l'image réciproque de la fonction « angle solide » sur l'intervalle $[0, \varepsilon_\Omega]$, c'est-à-dire :

$$\mathcal{B}_{y_n^k}(\varepsilon_\Omega, a_k) = \left\{ x \in \mathbb{R}^3 \text{ tels que } \Omega_{a_k}(y_n^k \rightarrow x) > \varepsilon_\Omega \right\}. \quad (4.9)$$

Afin d'alléger les notations, nous supposons dans la suite que ε_Ω est donné et n'apparaît plus en tant que variable dans les définitions. Ainsi, $\mathcal{B}_{y_n^k}(a_k)$ fait implicitement référence à $\mathcal{B}_{y_n^k}(\varepsilon_\Omega, a_k)$.

Pour un point et une normale donnés, l'ensemble $\mathcal{B}_{y_n^k}(a)$ correspond à un ensemble borné de l'espace, *cf.* figure 4.2. En effet, comme il est lié à l'angle solide d'un disque, plus on s'éloigne de y plus le disque apparaît petit ; ainsi, quelle que soit sa taille initiale, il existe toujours une distance à partir de laquelle il devient plus petit que le seuil ε_Ω . Cela constitue un avantage important lorsque l'on projettera l'image de ces ensembles dans l'espace écran au chapitre suivant, *cf.* chapitre 5. Par ailleurs, l'angle solide étant directement proportionnel à l'aire du disque, la famille des $(\mathcal{B}_{y_n^k}(a))_a$ est strictement croissante au sens de l'inclusion, c'est-à-dire $\forall a_1 \subsetneq a_2, \mathcal{B}_y(a_1) \subsetneq \mathcal{B}_y(a_2)$. Enfin, en plus de leur caractère croissant, la réunion de ces ensembles $\bigcup_{a \in \mathbb{R}^{++}} \mathcal{B}_y(a)$ s'identifie au demi-espace ouvert $\mathcal{H}^*(y)$. Pour toutes ces raisons, cette famille d'ensembles est un bon candidat pour être le support de nos fonctions inconnues w_k .

Cependant, en poursuivant l'analogie avec les coupes d'arbre, tel que défini ci-dessus, le domaine de définition $\mathcal{B}_{y_n^k}(a_k)$ ne reflète pas le fait que le *VPL* y_n^k appartient à la coupe uniquement si aucun de ses antécédents dans l'arbre n'a été sélectionné plus tôt. En effet, la coupe est censée être raffinée tant que le prédictat est vérifié, *cf.* formule 4.8. Dans une approche antérograde, il serait donc nécessaire de retirer le domaine de définition des nœuds fils du *VPL*, qui, est inclus dans celui du *VPL* courant. Or, notre approche s'abstirant de toute structure d'arbre, il nous est impossible de connaître, ni même de définir, la relation de parenté entre les *VPL*. Nous continuons tout de même notre analogie en supposant qu'un *VPL* dans \mathcal{L}^k possède ses fils dans \mathcal{L}^{k-1} et, bien que nous ne puissions pas connaître la position des *VPL* fils, nous supposons qu'ils sont statistiquement positionnés au même endroit que leur parent. Nous supposons en revanche que l'aire de leur disque correspond à celle de leur niveau. On définit donc le domaine de définition $\mathcal{D}_{y_n^k}(a_{k-1}, a_k)$ par, *cf.* figure 4.2 :

$$\mathcal{D}_{y_n^k}(a_{k-1}, a_k) = \mathcal{B}_{y_n^k}(a_k) \setminus \mathcal{B}_{y_n^k}(a_{k-1}). \quad (4.10)$$

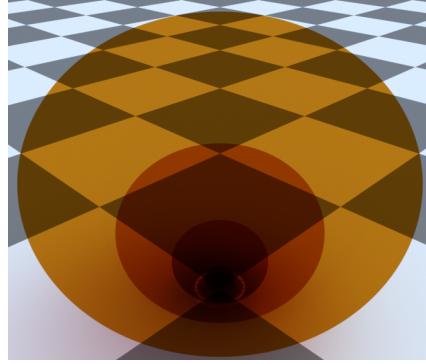


FIGURE 4.3 – Surfaces iso en dimension 3 de l’application α_y . Ces surfaces définissent les bords $\mathcal{B}_y(a)$ de nos fonctions de support.

Dans la construction des $\mathcal{D}_{y_n^k}$, le fait de retirer du domaine de définition le support des niveaux inférieurs (qui sont donc plus densément échantillonnés) justifie l’existence de nos coupes. En effet, comme nous l’avons dit, un échantillonnage grossier ne permet pas de représenter un signal haute fréquence tel que celui produit à proximité des surfaces. En revanche, il est en général suffisant pour représenter un éclairage distant comme le montrent les approches fondées sur les arbres. Ce choix de partition traduit exactement cette observation.

Partition de l’unité Grâce aux $\mathcal{D}_{y_n^k}$ que nous venons de définir, il est possible de fournir une solution triviale à la détermination de nos w_k . Pour cela, il suffit de les définir comme des fonctions indicatrices de leur domaine de définition, c’est-à-dire :

$$\forall k \in [1, K], \forall x, y \in \mathbb{R}^3, \quad w_k(x, y) = \begin{cases} 1 & \text{si } x \in \mathcal{D}_y(a_{k-1}, a_k) \\ 0 & \text{sinon} \end{cases}. \quad (4.11)$$

Puisque les $(\mathcal{D}_y(a_{k-1}, a_k))_k$ sont deux à deux disjoints, et que leur réunion recouvre entièrement $\mathcal{H}^*(y)$, la famille $(w_k^{(-1)})_k$ correspond bien à une partition de l’unité sur le demi-espace $\mathcal{H}^*(y)$. Cependant, la discontinuité des fonctions de bases entraîne dès lors une discontinuité sur le signal reconstruit. Nous cherchons à présent à résoudre cela en proposant une famille de fonctions \mathcal{C}^0 .

Réduction de dimension Afin de rendre nos fonctions continues, il est nécessaire que celles-ci tendent vers 0 sur le bord de leur domaine de définition. Dans ce but, on cherche une paramétrisation locale du bord de ces domaines. Pour cela, on définit l’application $\alpha_y : \mathbb{R}^3 \rightarrow \mathbb{R}$ qui pour un point donné $x \in \mathcal{H}^*(y)$ donne la plus petite valeur $a^* \in \mathcal{A}$ telle que $x \in \mathcal{B}_y(a^*)$. Comme les $\mathcal{B}_y(a)$ sont des ensembles strictement croissants par rapport à a , on peut définir cette fonction par :

$$\forall x \in \mathbb{R}^3, \quad \alpha_y(x) \quad \text{tel que} \quad \Omega_{\alpha_y(x)}(y \rightarrow x) = \varepsilon_\Omega. \quad (4.12)$$

Avec notre définition de la fonction angle solide cela donne :

$$\forall x \in \mathbb{R}^3, \quad \alpha_y(x) = \varepsilon_\Omega \frac{\|x - y\|^2}{\langle \vec{n}_y, \bar{y}x \rangle}. \quad (4.13)$$

De cette façon, le bord de $\mathcal{B}_y(a)$, que l’on note $\partial\mathcal{B}_y(a)$, correspond à la surface d’iso-valeur a de la fonction α_y , c’est-à-dire

$$\partial\mathcal{B}_y(a) = \{x \in \mathbb{R}^3 \text{ tels que } \alpha_y(x) = a\}. \quad (4.14)$$

Ainsi, le bord intérieur (resp. extérieur) du domaine $\mathcal{D}_y(a_{k-1}, a_k)$ correspond à $\partial\mathcal{B}_y(a_{k-1})$ (resp. $\partial\mathcal{B}_y(a_k)$), cf. figure 4.2 et 4.3. La fonction w_k que l'on cherche à construire doit être nulle sur ces deux bords.

Par ailleurs, puisque pour un couple (y, \vec{n}_y) donné, la réunion des iso-surfaces $\bigcup_{a \in \mathbb{R}^{+*}} \partial\mathcal{B}_y(a)$ coïncide avec le demi-espace $\mathcal{H}^*(y, \vec{n}_y)$, nous pouvons chercher des solutions de la formule 4.4 qui soient également constantes sur ces iso-surfaces, c'est-à-dire sous la forme :

$$\forall k \in [1, K], \quad w_k(y, x) = s_k(\alpha_y(x)), \quad (4.15)$$

où $(s_k : \mathbb{R} \rightarrow \mathbb{R})_k$ est une famille de fonctions inconnues à une dimension. Sous cette hypothèse, la condition de partition de l'unité sur les $w_k(y, x)$ définie par la formule 4.4 se traduit sur les s_k par :

$$\forall a \in \mathbb{R}^{+*}, \quad \sum_{k \in [1, K]} s_k(a) = 1. \quad (4.16)$$

En effet, en supposant la formule 4.16 vérifiée alors, pour tout $y \in \mathbb{R}^3, x \in \mathcal{H}^*(y)$:

$$\sum_k w_k(x, y) = \sum_k s_k(\alpha_y(x)) = 1.$$

Et, réciproquement, si la formule 4.4 est vérifiée, comme les $\partial\mathcal{B}_y(\cdot)$ recouvrent $\mathcal{H}^*(y)$, pour tout $a \in \mathbb{R}^+$ il existe un couple x_a, y_a tel que $a = \alpha_{y_a}(x_a)$ et ainsi :

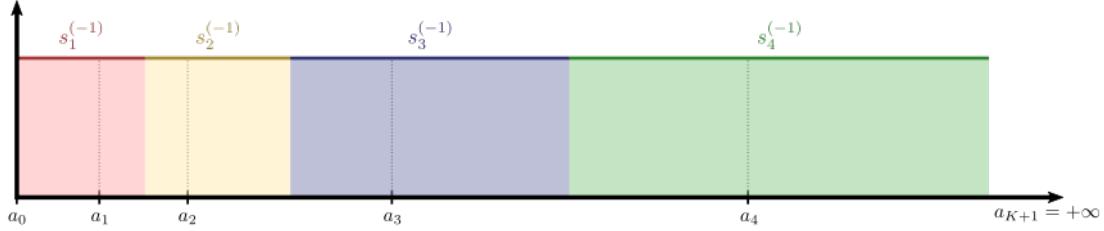
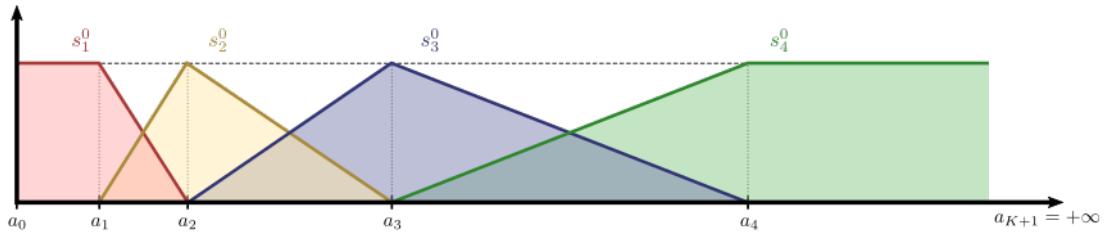
$$\sum_k s_k(a) = \sum_k s_k(\alpha_{y_a}(x_a)) = \sum_k w_k(y_a, x_a) = 1. \quad \square$$

Cette façon de définir nos w_k présente plusieurs avantages. Tout d'abord, cela réduit grandement la complexité du problème car dès lors il est suffisant de chercher une famille de fonctions à une variable réelle. Par ailleurs, dans l'estimateur diffus que nous avons défini dans ce paragraphe, cf. formule 4.3, on s'aperçoit que le terme de transfert est proportionnel à la fonction angle solide. Cela signifie que les surfaces $\partial\mathcal{B}_{y_n^k}(a)$ correspondent également à des surfaces d'iso-valeur pour le terme de contribution lumineuse. Cela permet d'avoir une fonction avec le minimum de variation spatiale ce qui autrement apporterait des artéfacts particulièrement gênants.

Solution \mathcal{C}^{-1} Notre paramétrisation nous permet dans un premier temps de retrouver la solution triviale que nous avions définie plus tôt, cf. formule 4.11. Nous proposons en revanche ici une solution dont les bornes sont plus cohérentes avec la solution du paragraphe suivant.

$$\forall k \in [1, K], \forall a \in \mathbb{R}, \quad s_k^{(-1)}(a) = \begin{cases} 1 & \text{si } k = 1 \text{ et } a \in \left[0, \frac{a_1+a_2}{2}\right[\\ 1 & \text{si } k \in]1, K[\text{ et } a \in \left[\frac{a_{k-1}+a_k}{2}, \frac{a_k+a_{k+1}}{2}\right[\\ 1 & \text{si } k = K \text{ et } a \in \left[\frac{a_{K-1}+a_K}{2}, +\infty\right[\\ 0 & \text{sinon} \end{cases}. \quad (4.17)$$

La figure 4.4 illustre la famille de fonction ainsi construite. Comme nous l'avons dit précédemment et comme nous le verrons dans les résultats, cette solution n'est pas satisfaisante du fait de la discontinuité du signal reconstruit, cf. §4.2.6.

FIGURE 4.4 – Solution \mathcal{C}^{-1} de s_k .FIGURE 4.5 – Solution \mathcal{C}^0 de s_k .

Solution \mathcal{C}^0 Afin de produire une solution continue de notre problème initial, cf. formule 4.4, il nous faut construire les s_k comme des fonctions continues. Nous proposons ici la solution la plus simple, qui correspond à un ensemble de fonctions linéaires par morceaux. Pour cela nous prolongeons les domaines de définition fournis pour la solution triviale de sorte à ce que deux niveaux successifs soit linéairement interpolés. Il vient ainsi la définition suivante dont une illustration est fournie par la figure 4.5 :

$$\forall k \in [1, K], \forall a \in \mathbb{R}, \quad s_k^{(0)}(a) = \begin{cases} 1 & \text{si } k = 1 \text{ et } a \in [a_0, a_1[\\ \frac{a - a_{k-1}}{a_k - a_{k-1}} & \text{si } k > 1 \text{ et } a \in [a_{k-1}, a_k[\\ \frac{a_{k+1} - a}{a_{k+1} - a_k} & \text{si } k < K \text{ et } a \in [a_k, a_{k+1}[\\ 1 & \text{si } k = K \text{ et } a \in [a_K, +\infty[\\ 0 & \text{sinon} \end{cases}. \quad (4.18)$$

Solutions d'ordres supérieurs Nous avons testé des solutions plus lisses en interpolant entre les niveaux avec des polynômes d'Hermite par exemple. Cependant le gain visuel obtenu ne justifiait pas le surcoût de calcul engendré alors que la solution \mathcal{C}^0 permettait déjà de résorber le principal artefact visuel. Pour cette raison, bien qu'il soit possible d'envisager une toute autre forme de solution, nous n'avons pas davantage cherché dans ce sens.

4.2.4 Intégration à fréquences multiples

Il nous reste à présent à définir nos différents échantillonnages. Comme nous l'avons dit précédemment, les multiples fréquences des échantillonnages $\mathcal{L}^1, \dots, \mathcal{L}^K$ visent à reproduire statistiquement le comportement d'un arbre de VPL. Nous cherchons donc à répartir les VPL dans les \mathcal{L}^k selon une loi géométrique. En effet, si on en supprime les arêtes, cela correspond au comportement des noeuds internes d'un arbre, où, dans le cas extrême d'un arbre complet d'arité q et de profondeur p , s'il y a N feuilles (c'est-à-dire N VPL), il y aura $\frac{N}{q}$ noeuds de profondeur $p-1$, $\frac{N}{q^2}$

nœuds de profondeur $p - 2$ et ainsi de suite. Faire une coupe dans cet arbre revient alors à sélectionner des nœuds de différentes fréquences d'échantillonnage dépendant du prédictat. On choisit alors des nœuds de faible profondeur (c'est-à-dire à fréquence d'échantillonnage faible) lorsque ceux-ci sont éloignés du receveur et des nœuds de profondeur proche des feuilles (c'est-à-dire à fréquence d'échantillonnage forte) lorsque ceux-ci sont proches du receveur.

Nous proposons donc de définir les densités d'échantillonnage grâce au paramètre libre $q_N > 1$ par :

$$\forall k \in [1, K], \quad N_k = q_N^{-(k-1)} N_1. \quad (4.19)$$

En notant N le nombre total d'échantillons, on a :

$$N = \sum_{k \in [1, K]} N_k. \quad (4.20)$$

On souhaite à présent extraire nos K échantillonnages à partir d'un seul car il est algorithmiquement beaucoup plus simple d'en produire un plutôt que plusieurs. Pour cela, il nous faut reformuler notre premier estimateur défini par l'équation formule 4.3. Ainsi en posant $\mathcal{L}^* = \bigcup_k \mathcal{L}^k$ et $\mathcal{L}^* = \{y_1^*, \dots, y_N^*\}$, on peut écrire :

$$\hat{\mathcal{T}}_L^{\text{CAL}}(x, \vec{\omega}) = \frac{1}{N} \sum_{y_n^* \in \mathcal{L}^*} \left(\sum_{k \in [1, K]} \mathbb{1}_{\mathcal{L}^k}(y_n^*) w_k(y_n^*, x) \frac{N}{N_k} \right) \frac{L(y_n^* \rightarrow x) h(y_n^*, x, \vec{\omega})}{\mu^S(y_n^*)}, \quad (4.21)$$

où $\mathbb{1}_{\mathcal{L}^k}(y_n^*)$ est la fonction indicatrice valant 1 si $y_n^* \in \mathcal{L}^k$ et 0 sinon. C'est grâce à cette fonction que l'on est en mesure d'inverser les symboles \sum de la formulation initiale. On peut alors écrire :

$$\hat{\mathcal{T}}_L^{\text{CAL}}(x, \vec{\omega}) = \frac{1}{N} \sum_{y_n^* \in \mathcal{L}^*} W_n(x) \frac{L(y_n^* \rightarrow x) h(y_n^*, x, \vec{\omega})}{\mu^S(y_n^*)}, \quad (4.22)$$

et

$$W_n(x) = \sum_{k \in [1, K]} \frac{N}{N_k} \mathbb{1}_{\mathcal{L}^k}(y_n^*) w_k(y_n^*, x). \quad (4.23)$$

Cette formulation de l'estimateur de coupe antérograde permet d'interpréter notre échantillonnage multi-échelle, comme issu d'un échantillonnage simple \mathcal{L}^* d'une seule densité de probabilité μ^S mais dont la valeur échantillonnée est multipliée par une nouvelle fonction $W_n(x)$. Ici, $W_n(x)$ permet de distribuer de manière déterministe les VPL selon les différents niveaux d'échantillonnage. En outre, grâce aux fonctions de support $w_k(y_n^*, x)$ qui la composent ainsi qu'aux indicatrices $\mathbb{1}_{\mathcal{L}^k}(y_n^*)$, c'est également la fonction $W_n(x)$ qui permet de définir la zone d'influence du VPL y_n^* .

Comme nous l'avons dit, étant donné $\mathcal{L}^1, \dots, \mathcal{L}^K$, cette fonction distribue de manière déterministe les VPL de \mathcal{L}^* dans les différents niveaux. Cependant, cette approche nécessite de construire indépendamment les K échantillonnages. Nous proposons donc une approche permettant d'extraire ces échantillonnages de manière probabiliste en ayant comme seul pré-requis la donnée d'un unique échantillonnage \mathcal{L} composé de N échantillons $\{y_1, \dots, y_N\}$ suivant la loi μ^S .

On se donne également une famille de N variables aléatoires indépendantes identiquement distribuées $(\kappa_n)_n$ à valeurs discrètes dans $[1, K]$ de loi $\mu^{\mathcal{L}}$ telle que :

$$\forall n, \forall k \in [1, K], \quad \mathbb{P}(\kappa_n = k) = \mu_k^{\mathcal{L}} = \frac{N_k}{N}. \quad (4.24)$$

Il s'agit bien là d'une loi de probabilité car $\sum_k \frac{N_k}{N} = 1$. De plus, les variables aléatoires κ_n permettent de conserver, en espérance tout du moins, la répartition du nombre de *VPL* par niveau. Nous souhaitons, en effet, préserver le fait que les *VPL* ayant une faible région d'influence soient plus nombreux que ceux ayant une plus grande région d'influence.

En supposant par ailleurs les κ_n indépendantes des échantillons y_n , on construit une variable aléatoire \hat{W}_n par :

$$\hat{W}_n(x) = \frac{w_{\kappa_n}(y_n, x)}{\mu_{\kappa_n}^{\mathcal{L}}} = \frac{N}{N_{\kappa_n}} w_{\kappa_n}(y_n, x), \quad (4.25)$$

dont l'espérance est 1 pour toute valeur de y_n . En effet :

$$\begin{aligned} \mathbb{E} [\hat{W}_n(x)]_{\kappa_n \sim \mu^{\mathcal{L}}} &= \mathbb{E} \left[\frac{w_{\kappa_n}(y_n, x)}{\mu_{\kappa_n}^{\mathcal{L}}} \right]_{\kappa_n \sim \mu} \\ &= \sum_{k \in \llbracket 1, K \rrbracket} \frac{w_k(y_n, x)}{\mu_k^{\mathcal{L}}} \mu_k^{\mathcal{L}} \\ &= \sum_{k \in \llbracket 1, K \rrbracket} w_k(y_n, x) = 1. \quad \square \end{aligned}$$

Cela nous permet ainsi d'écrire un nouvel estimateur par coupe antérograde tel que :

$$\hat{\mathcal{T}}_L^{\text{CAL}}(x, \vec{\omega}) = \frac{1}{N} \sum_{y_n \in \mathcal{L}} \hat{W}_n(x) \frac{L(y_n \rightarrow x) h(y_n, x, \vec{\omega})}{\mu^{\mathcal{S}}(y_n)}. \quad (4.26)$$

Il s'agit bien d'un estimateur non biaisé de \mathcal{T}_L , car :

$$\mathbb{E} [\hat{\mathcal{T}}_L^{\text{CAL}}](x, \vec{\omega}) = \frac{1}{N} \sum_{y_n \in \mathcal{L}} \mathbb{E} \left[\hat{W}_n(x) \underbrace{\frac{L(y_n \rightarrow x) h(y_n, x, \vec{\omega})}{\mu^{\mathcal{S}}(y_n)}}_{= B_n} \right]$$

or, étant donné que κ_n est indépendante de y_n , \hat{W}_n est indépendante de B_n . D'où

$$\mathbb{E} [\hat{W}_n B_n] = \underbrace{\mathbb{E} [\hat{W}_n]}_{= 1} \mathbb{E} [B_n] = \mathbb{E} [B_n]$$

et donc

$$\begin{aligned} \mathbb{E} [\hat{\mathcal{T}}_L^{\text{CAL}}](x, \vec{\omega}) &= \frac{1}{N} \sum_{y_n \in \mathcal{L}} \mathbb{E} \left[\frac{L(y_n \rightarrow x) h(y_n, x, \vec{\omega})}{\mu^{\mathcal{S}}(y_n)} \right] \\ &= \mathcal{T}_L(x, \vec{\omega}). \quad \square \end{aligned}$$

Construction de la variable aléatoire Pour construire la variable κ_n telle que nous l'avons définie, cf. formule 4.24, on considère tout d'abord une variable aléatoire u sur $[0, 1]$ qui est supposée donnée ou facile à construire, et une famille $\tilde{m}_1, \dots, \tilde{m}_K$ définie par

$$\tilde{m}_k = \sum_{i \in \llbracket 1, k \rrbracket} \mu_i^{\mathcal{L}}. \quad (4.27)$$

On calcule alors le rang κ_n en itérant sur tous les k et en retournant le plus petit k ne vérifiant pas $u < \tilde{m}_k$. Un tel algorithme produit bien une variable de loi définie par la formule 4.24.

4.2.5 Répartition des sources

Le dernier estimateur que nous avons défini, *cf.* formule 4.26, permet de justifier, à la façon des lumières multiples, le calcul du transport de la lumière à l'aide d'un échantillonnage de la surface par des points. Cependant, tandis que les précédentes approches évaluent directement l'intégrande dans la résolution numérique, nous proposons de rajouter un terme, $\hat{W}_n(x)$, n'ayant pas d'influence en espérance, mais permettant au moment d'évaluer la luminance des *VPL* de répartir leur contribution de manière adaptée.

En pratique, afin de fournir une approximation numérique du transport lumineux tel que définie plus tôt, *cf.* formule 4.1, nous construisons la position de nos *VPL* en les distribuant aléatoirement sur la surface. De plus, nous évaluons pour chacun d'eux une seconde variable aléatoire, κ_n *cf.* formule 4.24, permettant de complètement définir la fonction de transfert définie par notre estimateur, *cf.* formule 4.26. Ainsi pour un receveur donné, nous déterminons l'ensemble des *VPL* qui affectent ce point puis nous évaluons la somme de toutes leurs contributions.

Il est par ailleurs intéressant de noter que dans nos définitions, nous avons laissé un certain nombre de paramètres libres, dont les distances d'influences dépendant des aires $(a_k)_k$, *cf.* formule 4.7, ainsi que la répartition des densités dépendant des $(N_k)_k$, *cf.* formule 4.19. De plus, l'estimateur $\hat{\mathcal{T}}_L^{\text{CAL}}(x, \vec{\omega})$, *cf.* formule 4.26, demeure sans biais même si l'on choisit les distances d'influence indépendamment des densités d'échantillonnage. Notre formulation permet ainsi de décorrélérer ces deux grandeurs. Cependant, d'après la définition du terme $\hat{W}_n(x)$, ce dernier est inversement proportionnel au nombre N_{κ_n} de *VPL* de son niveau. Cela signifie qu'un *VPL* appartenant à un niveau clairsemé possède une intensité forte tandis qu'un *VPL* appartenant à un niveau fortement peuplé, possède une intensité faible. Ce constat est en parfaite adéquation avec celui que nous faisions pour définir les fonctions de support, *cf.* §4.2.3. En effet, puisque qu'un *VPL* ayant une faible densité, se retrouve avec une forte intensité lumineuse, il ne faut pas que sa contribution affecte les régions proches de lui, car, à cause du facteur de forme $G(y_n^k, x)$, l'éclairement à proximité du *VPL* est déjà important. Pour respecter cela, il est donc intéressant de lui affecter une aire représentative a_k importante. Ainsi, tout comme pour la famille des $(N_k)_k$, nous choisissons notre famille d'aires $(a_k)_k$ selon une évolution géométrique en fonction d'un paramètre libre $q_a > 1$ tel que :

$$\forall k \in [1, K], \quad a_k = a_1 q_a^{k-1}. \quad (4.28)$$

De cette façon, nous lions naturellement l'aire représentative des fonctions de support qui évolue proportionnellement avec la densité d'échantillonnage. En pratique, nous choisissons également $q_a = q_N$.

4.2.6 Évaluation

Dans ce paragraphe nous décrivons le cas d'application de notre estimateur sur un jeu de modèles afin d'évaluer notre technique. Nous avons comme données d'entrée une scène ainsi qu'une fonction permettant d'en produire autant d'échantillonnages que l'on souhaite. Nous ne développerons qu'au chapitre suivant, *cf.* chapitre 5, un pipeline efficace pour produire cet échantillonnage. Nous supposons donc ici qu'il est simplement possible de réaliser ce calcul.

La scène comporte une source lumineuse primaire ponctuelle et nous appliquons un rebond diffus de la lumière sur des surfaces lambertiennes. Nous interprétons ainsi chaque échantillon comme

	Lancer de chemins		Radiosité inst.			
	RMSE	5.38	RMSE	0	#VPL	29336
Radiosité inst. (N=64K)	SSIM	0.9921	SSIM	100%	%pix./VPL	6.60%
	$\Delta_{p>2\%}$	3.86%	$\Delta_{p>2\%}$	0%	#VPL/pix.	1936.94
Coupe ant. (N=64K)	RMSE	7.09	RMSE	5.32	#VPL	29336
	SSIM	0.9840	SSIM	0.9875	%pix./VPL	1.51%
	$\Delta_{p>2\%}$	6.31%	$\Delta_{p>2\%}$	4.44%	#VPL/pix.	443.55
Coupe ant. (N=16K)	RMSE	9.36	RMSE	7.36	#VPL	7404
	SSIM	0.9686	SSIM	0.9740	%pix./VPL	1.54%
	$\Delta_{p>2\%}$	12.63%	$\Delta_{p>2\%}$	11.97%	#VPL/pix.	114.23
Coupe ant. (N=2K)	RMSE	13.47	RMSE	13	#VPL	954
	SSIM	0.8956	SSIM	0.9048	%pix./VPL	1.36%
	$\Delta_{p>2\%}$	38.53%	$\Delta_{p>2\%}$	37.33%	#VPL/pix.	13.02

TABLE 4.1 – Table de résultats pour les coupes antérogrades discrètes

une source lumineuse dont la fonction d'éclairage est donnée par l'opérateur $\hat{\mathcal{T}}_L^{\text{CAL}}$, où L est la luminance directe, *cf.* §2.1.4.

Nous évaluons notre approche en mettant en concurrence deux critères, à savoir la différence visuelle par rapport à la vérité terrain et le nombre de calculs réalisés. Ce dernier point étant crucial pour nous car il est directement corrélé à la vitesse d'exécution de notre algorithme. Nous étudions les différents paramètres de notre algorithme, à savoir la répartition des régions d'influences $(a_k)_k$, *cf.* formule 4.28, la densité globale N , *cf.* formule 4.20, ainsi que les différentes densités des sous-échelles $(N_k)_k$, *cf.* formule 4.19.

Les images produites sont comparées en basse dynamique sur lesquelles nous corrigons la courbe de gamma afin de visualiser le résultat. Ce filtre est identique à celui employé dans le moteur de rendu *PBRT* [Pharr et coll., 2016], que nous avons employé pour produire nos images de référence. Les différentes mesures calculées sont toutes réalisées sur des images produites de la sorte.

Comparaisons Sur la figure 4.6, nous évaluons notre approche sur une scène d'intérieur éclairée uniquement par le soleil. Le résultat visualisé correspond à l'éclairage direct et l'éclairage indirect après un rebond lumineux. Les surfaces en présence ont toutes un matériau diffus et les images sont produites à la résolution de 1280×720 pixels. La table 4.1 rapporte les différentes métriques d'erreur que nous évaluons.

La comparaison est réalisée par rapport à deux références. Tout d'abord, comme nous l'avons dit plus tôt, l'approche par lancer de chemins nous permet d'obtenir une vérité terrain de cette scène. Pour la produire, nous avons généré $32K^1$ chemins par pixel. Cependant, pour pouvoir calculer nos estimateurs, il nous a été nécessaire d'introduire un biais en remplaçant le facteur de forme G par le terme G_b , *cf.* formule 2.19. Se comparer directement au lancer de chemin ne serait donc

¹ici et dans la suite, le suffixe K correspond à la multiplication par $2^{10} = 1024$, de même le suffixe M correspond à $2^{20} \approx 10^6$

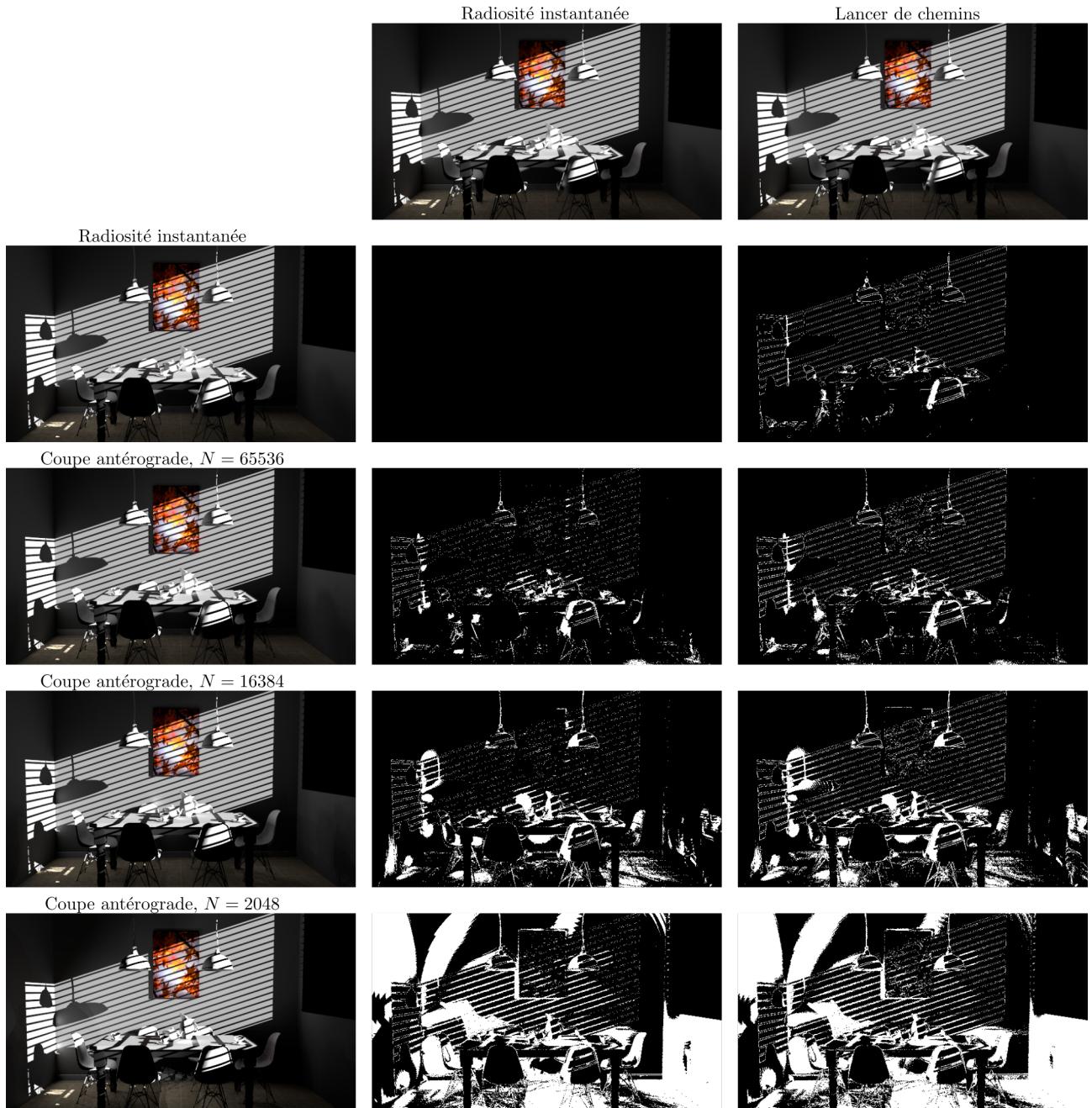


FIGURE 4.6 – Comparaisons des coupes antérogrades discrètes sur une scène d'intérieur. En haut, nos deux références, évaluées l'une par radiosité instantanée et l'autre par lancer de chemins. À gauche, les images sont produites par notre algorithme en faisant varier N , le nombre de VPL. Au centre, les pixels blancs correspondent aux pixels ayant une différence supérieure à 2% par rapport à la référence.

pas complètement correct. C'est pourquoi nous évaluons également notre approche vis-à-vis des résultats produits par l'algorithme de radiosité instantanée. Nous utilisons pour cela 64K *VPL*.

Par ailleurs, pour évaluer l'erreur commise avec notre approche, nous utilisons trois métriques différentes : la racine de l'erreur quadratique moyenne (RMSE), la *Structural SIMilarity* (SSIM) [Wang et coll., 2004], et le pourcentage de pixels dont l'erreur relative est supérieure à 2% ($\Delta_{p>2\%}$). Cette dernière métrique nous permet ainsi de visualiser la carte des différences notables entre les images, cf. figure 4.6.

Enfin, nous comparons ici l'influence du nombre de *VPL*, N , variant de 64K à 2K. Tout d'abord, comme il s'agit d'une scène d'intérieur, tous les *VPL* construits ne sont pas éclairés directement. Nous notons alors #*VPL* le nombre de *VPL* qui ont une luminance non nulle. Pour ces *VPL* uniquement, nous calculons ensuite deux métriques équivalentes permettant de visualiser les performances de notre algorithme : le pourcentage moyen de pixels de l'écran affectés par un *VPL* (%pix./*VPL*) et le nombre moyen de *VPL* qui influe un pixel (#*VPL*/pix.). On note alors que ces deux mesures sont indépendantes de la résolution de l'écran.

De plus, #*VPL*/pix. est proportionnelle au nombre de *VPL* et ne dépend, par ailleurs, que de la distribution des *VPL* (c'est-à-dire $\{N_1, N_2, \dots\}$) et des régions d'influence a_k , cf. §4.2.5. Elle constitue ainsi une métrique intéressante pour évaluer le gain de notre choix de distribution par rapport à la radiosité instantanée. En effet, dans notre évaluation, la référence et l'un des rendus ont été calculés avec 64K *VPL*. On note alors que, pour la référence, chaque pixel est touché, en moyenne, par 1936 *VPL*, tandis que dans notre approche, cette valeur ne vaut que 443. Cela signifie que, dans notre approche, nous avons réalisé près de 4 fois moins de calculs et que nous avons donc été environ 4 fois plus rapide. En parallèle, la SSIM entre les deux images est de 0.98, ce qui signifie qu'elles restent très proches. De même, lorsque l'on diminue N , le gain de performance par rapport à cette référence augmente car #*VPL*/pix. est proportionnelle à N . En revanche, la qualité visuelle est alors dégradée.

Par ailleurs, #*VPL*/pix. est peu pratique dans l'absolu, car elle dépend du nombre de *VPL*. C'est pourquoi, nous introduisons également %pix./*VPL*. On peut alors interpréter cette valeur comme un indicateur d'efficacité de chaque *VPL* pris séparément. Ainsi, pour cette scène, chaque *VPL* de la radiosité instantanée influence, en moyenne, 6.6% des pixels de l'écran. Pour notre choix de coupe antérograde, chaque *VPL* influence, en moyenne, 1.5% des pixels de l'écran. Cela signifie, qu'en moyenne, chacun de nos *VPL* réalise environ 4 fois moins de travail sur cette scène.

Grâce à ces différentes métriques, nous sommes ainsi en mesure de visualiser l'influence des paramètres sur les résultats. Pour la scène présentée ici², nous avons choisi empiriquement les paramètres tels que $K = 3$, $a_1 = 0.25$, $q_a = 2$ et $q_N = 1/3$ et nous avons observé un gain moyen en performance de 4. Nous pensons par ailleurs qu'un choix plus automatique des paramètres pourrait être réalisé en utilisant cette mesure, mais nous laissons cela pour de futures recherches.

Discussion Sur certains résultats, on peut observer un effet de seuil sur les bords anguleux des maillages, notamment dans les coins. Cet effet, davantage visible avec un échantillonnage faible, est causé par notre façon de distribuer les zones d'influences de manière discrète. En effet, bien que d'un niveau à l'autre la transition soit lisse grâce à nos fonctions de support, la quantité d'échantillons qui contribuent, quant à elle, change brusquement lors de cette transition. Cela a pour conséquence de faire ressortir une région par rapport à sa voisine. Nous allons à présent tenter

²Scène *The Breakfast Room* réalisée par Bitterli [Bitterli, 2016]

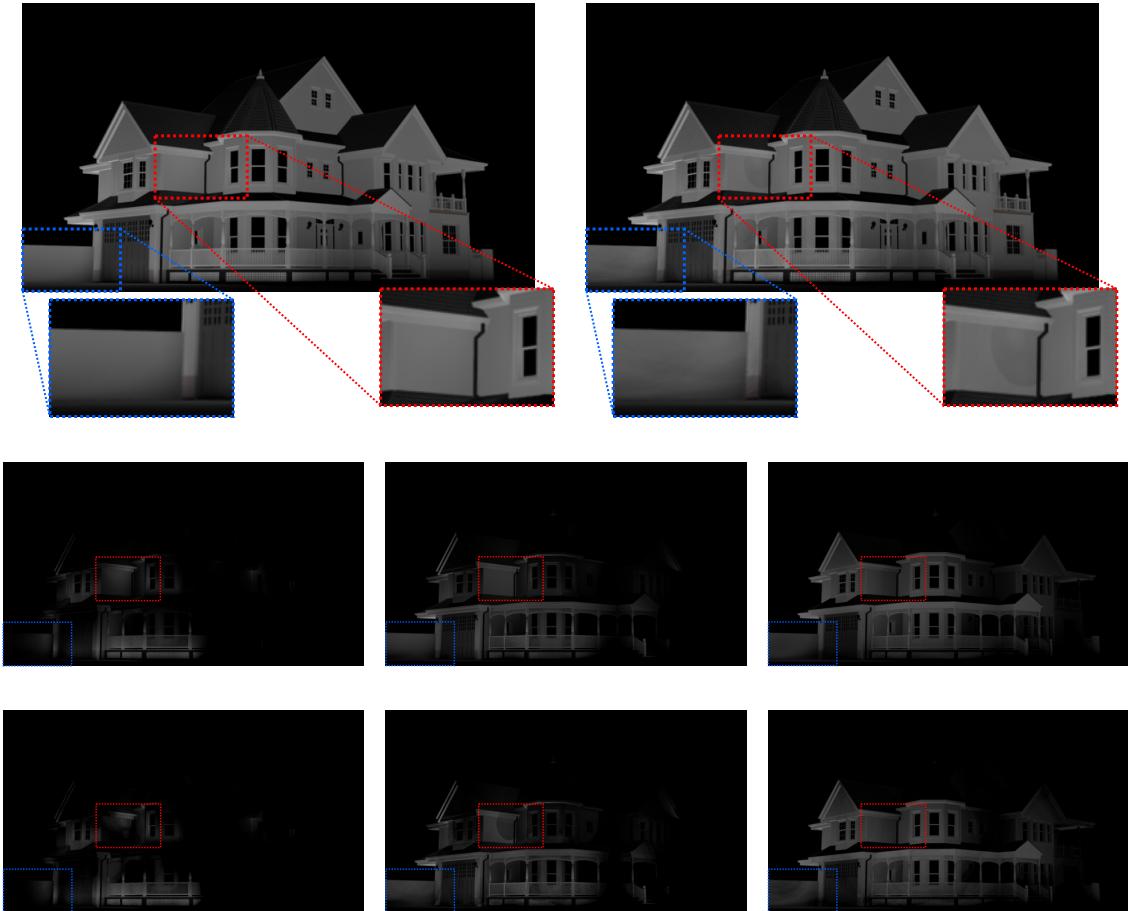


FIGURE 4.7 – Comparaison de fonctions de support \mathcal{C}^0 et \mathcal{C}^{-1} sur le rebond indirect sans ombres indirectes. En haut à gauche, rendu avec la solution \mathcal{C}^0 , à droite avec la solution \mathcal{C}^{-1} . Les deux lignes inférieures correspondent à la contribution respective des trois premières fonctions de support (sur les cinq) de notre algorithme pour la solution \mathcal{C}^0 au-dessus et \mathcal{C}^{-1} en dessous. Les deux images sont produites avec le même nombre de *VPL* (64K) à la même position et avec les mêmes valeurs pour μ_k et a_k . La solution continue permet ici de réduire visuellement les artefacts engendrés par la solution discontinue (encadrés rouge et bleu) pour un même coup de calcul. Ceux-ci étant particulièrement visibles lorsque, comme ici, le nombre de *VPL* est particulièrement faible.

d'atténuer cet artéfact en faisant évoluer de manière continue les zones d'influence en fonction de la variable aléatoire associée aux *VPL*.

4.3 Coupes antérogrades continues

Dans la section précédente, nous avons introduit la notion de coupe antérograde. Nous avons alors fini par interpréter l'estimateur qui correspond comme une application qui échantillonne la surface et lui attribue aléatoirement une région d'influence. Comme nous avons cherché, dans un premier temps, à imiter le comportement d'une coupe au sein d'un arbre de *VPL*, nous passions de manière discrète d'une région d'influence à l'autre. Or nous avons pu constater que cela pouvait se voir sur de la reconstruction du signal.

Dans cette section, nous cherchons ainsi à étendre notre problème au cas continu en définissant un moyen de choisir continûment la région d'influence du *VPL*. Pour cela, nous reformulons dans un premier temps notre problème et nos contraintes dans le domaine continu. Puis nous exprimons les solutions développées à la section précédente dans ce nouveau formalisme pour proposer, enfin, une solution entièrement continue à partir de ces solutions.

4.3.1 Formalisme

Notre problème est ici le même que celui défini précédemment, cf. §4.2. En d'autres termes, étant donné un champ de luminance L défini à la surface de la géométrie \mathcal{S} , on cherche à calculer le résultat d'une application de l'opérateur de transport \mathcal{T} . Pour cela on introduit un nouveau paramètre \mathcal{W} tel que :

$$\forall x \in \mathcal{S}, \vec{\omega} \in \Omega(x), \quad \mathcal{T}_L(x, \vec{\omega}) = \int_{y \in \mathcal{S}} \int_{\nu \in \mathcal{V}} \mathcal{W}(\nu, y, x) L(y \rightarrow x) h(y, x, \vec{\omega}) d\mathcal{S}_y d\nu, \quad (4.29)$$

où \mathcal{V} est un espace mesurable quelconque pour lequel la fonction de poids $\mathcal{W}(\nu, y, x)$ doit vérifier l'égalité suivante :

$$\forall y \in \mathbb{R}^3, \forall x \in \mathcal{H}^*(y), \quad \int_{\nu \in \mathcal{V}} \mathcal{W}(\nu, y, x) d\nu = 1. \quad (4.30)$$

Comme pour les $(w_k)_k$ dans le cas discret, cf. formule 4.4, la fonction \mathcal{W} est une inconnue de notre problème pour laquelle nous devons fournir des solutions. Cependant, là où la famille des $(w_k)_k$ était indexée par un paramètre entier k , notre nouvelle formulation permet d'indexer continûment, via la variable ν , l'ensemble des fonctions de support. Par ailleurs, dans le but de fournir une définition propre, nous avons également dû introduire l'espace de définition \mathcal{V} de la variable ν . Il serait tout à fait possible de considérer que ν soit une variable multidimensionnelle à valeurs dans une partie de \mathbb{R}^n , mais cela complexifierait notre étude et nous laissons cela pour des travaux futurs. Ainsi, bien qu'arbitraire, mais dans le but d'étendre naturellement nos travaux établis dans le cadre discret, nous considérons dans la suite que \mathcal{V} correspond à un intervalle de \mathbb{R} .

4.3.2 Solution Discrète

Dans cette section, nous montrons comment retrouver les mêmes résultats que ceux obtenus dans la section précédente mais exprimés dans notre nouveau formalisme. Nous nous plaçons dans le

même contexte que celui de la section précédente en définissant de la même façon les K densités cibles $\mu_1^{\mathcal{L}}, \dots, \mu_K^{\mathcal{L}}$, cf. Eqneqn :flc-kappa, ainsi que les fonctions de support $w_k(y, x)$, cf. formule 4.15, 4.18.

On pose par ailleurs $\mathcal{V} = [0, K + 1]$ et

$$\forall \nu \in \mathcal{V}, \forall x, y \in \mathbb{R}^3, \quad \mathcal{W}_{(\text{CAL})}(\nu, y, x) = \frac{\tilde{w}_{(\text{CAL})}(\nu, y, x)}{\tilde{\mu}_{(\text{CAL})}^{\mathcal{L}}(\nu)} \sum_{k \in [\![1, K]\!]} \mu_k^{\mathcal{L}} \delta_k(\nu), \quad (4.31)$$

où $\delta_k(\nu)$ représente la distribution de Dirac centrée en k , et $\tilde{w}_{(\text{CAL})}(\nu, y, x)$ (resp. $\tilde{\mu}_{(\text{CAL})}^{\mathcal{L}}(\nu)$) est un prolongement sur \mathcal{V} par rapport à ν de la variable k de $w_k(y, x)$ (resp. $\mu_k^{\mathcal{L}}$) qui vérifie :

$$\forall \nu \in \mathcal{V}, \quad \tilde{w}_{(\text{CAL})}(\nu, y, x) = \begin{cases} w_k(y, x) & \text{si } \nu = k \in [\![1, K]\!] \\ 0 & \text{sinon} \end{cases}, \quad (4.32)$$

et

$$\tilde{\mu}_{(\text{CAL})}^{\mathcal{L}}(\nu) = \begin{cases} \mu_k^{\mathcal{L}} & \text{si } \nu = k \in [\![1, K]\!] \\ 1 & \text{sinon} \end{cases}. \quad (4.33)$$

On vérifie ainsi d'une part que $\mathcal{W}_{(\text{CAL})}(\nu, y, x)$ est une solution de formule 4.30. En effet :

$$\begin{aligned} \forall x, y \in \mathbb{R}^3, \int_{\nu \in \mathcal{V}} \mathcal{W}_{(\text{CAL})}(\nu, y, x) d\nu &= \int_{\nu \in \mathcal{V}} \frac{\tilde{w}_{(\text{CAL})}(\nu, y, x)}{\tilde{\mu}_{(\text{CAL})}^{\mathcal{L}}(\nu)} \sum_{k \in [\![1, K]\!]} \mu_k^{\mathcal{L}} \delta_k(\nu) d\nu \\ &= \sum_{k \in [\![1, K]\!]} \int_{\nu \in \mathcal{V}} \frac{\tilde{w}_{(\text{CAL})}(\nu, y, x)}{\tilde{\mu}_{(\text{CAL})}^{\mathcal{L}}(\nu)} \mu_k^{\mathcal{L}} \delta_k(\nu) d\nu \\ &= \sum_{k \in [\![1, K]\!]} \frac{\tilde{w}_{(\text{CAL})}(k, y, x)}{\tilde{\mu}_{(\text{CAL})}^{\mathcal{L}}(k)} \mu_k^{\mathcal{L}} \\ &= \sum_{k \in [\![1, K]\!]} w_k(y, x) = 1. \quad \square \end{aligned} \quad (4.34)$$

D'autre part, comme $\sum_k \mu_k^{\mathcal{L}} = 1$, on peut également interpréter le terme $\sum_k \mu_k^{\mathcal{L}} \delta_k(\nu)$ comme la densité de probabilité d'une variable aléatoire discrète κ à valeurs dans $[1, K]$, telle que :

$$\forall k \in [\![1, K]\!], \quad \mathbb{P}(\kappa = k) = \mu_k^{\mathcal{L}}, \quad (4.35)$$

ainsi que l'intégrale $\int_{\nu} \mathcal{W} d\nu$ comme l'espérance d'une variable aléatoire $\hat{\mathcal{W}}_{(\text{CAL})}(\kappa)$ définie par :

$$\hat{\mathcal{W}}_{(\text{CAL})}(\kappa) = \frac{\tilde{w}_{(\text{CAL})}(\kappa, y, x)}{\tilde{\mu}_{(\text{CAL})}^{\mathcal{L}}(\kappa)} = \frac{w_{\kappa}(y, x)}{\mu_{\kappa}^{\mathcal{L}}}. \quad (4.36)$$

Ce dernier estimateur est identique à celui que nous avons défini dans l'approche discrète, cf. formule 4.25. On vient donc de démontrer que, bien que nos contraintes aient été exprimées à l'aide d'une formulation continue dans ce chapitre, cf. formule 4.30, les solutions développées dans la section précédente permettent également de répondre à ce nouveau problème que nous venons de poser. Ainsi, nous partons de nos précédentes solutions et, en analysant les deux termes qui constituent cet estimateur nous prolongeons complètement la solution de manière continue sur la dimension offerte par la variable ν .

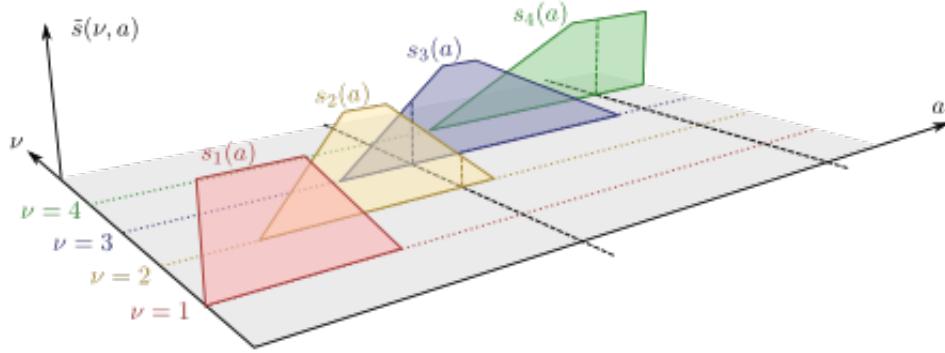


FIGURE 4.8 – Plongement trivial de s_k en dimension 2. La fonction résultante $a \mapsto \tilde{s}(\nu, a)$ est identique à $a \mapsto s_\nu(a)$ pour les valeurs de ν entières.

4.3.3 Interprétation et Représentation

La solution que nous venons de fournir se décompose en un quotient de deux termes. Nous avons au numérateur les fonctions de support \tilde{w} et au dénominateur nous avons la famille de termes de normalisation $\tilde{\mu}^L$ que nous avons interprétée comme une densité de probabilité.

\tilde{w} – fonctions de support Dans la solution proposée des $w_k(y, x)$ cf. formule 4.15, nous avions introduit la famille de fonctions s_k . Celle-ci avait pour but de réduire la dimension du problème. Ainsi, pour la version continue, prolonger $w_k(y, x)$ en $\tilde{w}_{(CAL)}(\nu, y, x)$ décrite par formule 4.32 revient à prolonger $s_k(a)$ en $\tilde{s}_{(CAL)}(\nu, a)$ telle que :

$$\forall \nu \in \mathcal{V}, \forall a \in \mathbb{R}, \quad \tilde{s}_{(CAL)}(\nu, a) = \begin{cases} s_k(a) & \text{si } \nu = k \in [1, K] \\ 0 & \text{sinon} \end{cases}, \quad (4.37)$$

en posant

$$\forall \nu \in \mathcal{V}, \forall x, y \in \mathbb{R}^3, \quad \tilde{w}_{(CAL)}(\nu, y, x) = \tilde{s}_{(CAL)}(\alpha_y(x)), \quad (4.38)$$

où $\alpha_y(x) : \mathbb{R}^3 \rightarrow \mathbb{R}$ est la fonction définie précédemment, cf. formule 4.12. La figure figure 4.8 illustre ce prolongement.

$\tilde{\mu}^L$ – densité d'échantillonnage La densité d'échantillonnage $\tilde{\mu}^L$ est le second terme que nous avons introduit dans notre solution. Son pendant discret, μ_k^L , cf. formule 4.24, permettait de capturer la notion de répartition multi-échelle des échantillons et de les répartir en différents niveaux. En outre, c'est grâce à cette fonction de répartition que l'on était en mesure de répartir les zones d'influence des VPL.

Par ailleurs, le terme de normalisation $\tilde{\mu}^L(\nu)$ traduit lui aussi le fait que plus une zone de l'espace est échantillonnée, plus le terme $\tilde{\mu}^L(\nu)$ est grand, moins les VPL correspondants ont d'intensité et moins ils contribuent à l'éclairage.

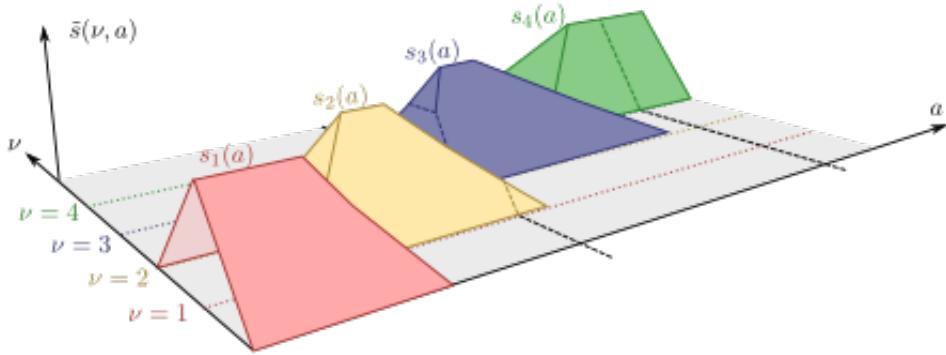


FIGURE 4.9 – Prolongement linéaire de s_k en dimension 2. La fonction résultante $\nu \mapsto \tilde{s}(\nu, \cdot)$ est identique à s_k pour les valeurs de ν entière. Pour toute valeur de ν non entière, la fonction est interpolée linéairement soit vers 0 là où il n'existe qu'une seule fonction s_k non nulle, soit entre les deux s_k et s_{k+1} là où leurs domaines de définition se recoupent.

4.3.4 Prolongement Continu

Dans ce paragraphe, nous cherchons à fournir une solution particulière à notre problème, *cf.* formule 4.30. Pour cela, nous prolongeons la solution discrète en reconstruisant une fonction lisse à partir de la solution $\hat{\mathcal{W}}_{(\text{CAL})}$ définie plus haut, *cf.* formule 4.36.

Nous procédons ainsi en deux temps. Tout d'abord, nous construisons la fonction de distribution continue $\tilde{\mu}(\nu)$, puis dans un second temps nous cherchons à construire la fonction $\tilde{w}(\nu, y, x)$, sous la forme :

$$\tilde{w}(\nu, y, x) = \tilde{s}(\nu, \alpha_y(x)), \quad (4.39)$$

où $\tilde{s}(\nu, a)$ est une fonction inconnue vérifiant :

$$\forall a \in \mathbb{R}, \quad \int_{\nu \in \mathcal{V}} \tilde{s}(\nu, a) d\nu = 1. \quad (4.40)$$

Ainsi, en se donnant v comme une variable aléatoire de densité de probabilité $\tilde{\mu}$, on définit une nouvelle variable aléatoire $\int_{\nu} \mathcal{W}(\nu, y, x) d\nu$, dont l'espérance est 1, par :

$$\hat{\mathcal{W}}^{(0)}(y, x) = \frac{\tilde{w}(v, y, x)}{\tilde{\mu}(v)}. \quad (4.41)$$

On note que cette dernière contrainte est la traduction dans le domaine continu de la contrainte exprimée sur les s_k , *cf.* formule 4.16. Pour les mêmes arguments qui avaient alors été évoqués dans la version discrète, si l'on cherche les solutions de $\hat{\mathcal{W}}^{(0)}$ sous la forme réduite ci-dessus, *cf.* formule 4.39, la contrainte définie par la formule 4.40 est bien équivalente à la formule 4.30.

Densité d'échantillonnage Nous conservons ici l'analogie simple avec la représentation d'arbre. Les $\mu_k^{\mathcal{L}}$ représentaient alors une loi géométrique, c'est pourquoi nous faisons le choix de distribuer notre nouvelle densité de VPL suivant une loi exponentielle de paramètre $\lambda^{\mathcal{L}}$ que nous laissons comme paramètre libre. Ainsi,

$$\tilde{\mu}^{\mathcal{L}}(\nu) = \lambda^{\mathcal{L}} e^{-\lambda^{\mathcal{L}} \nu}. \quad (4.42)$$

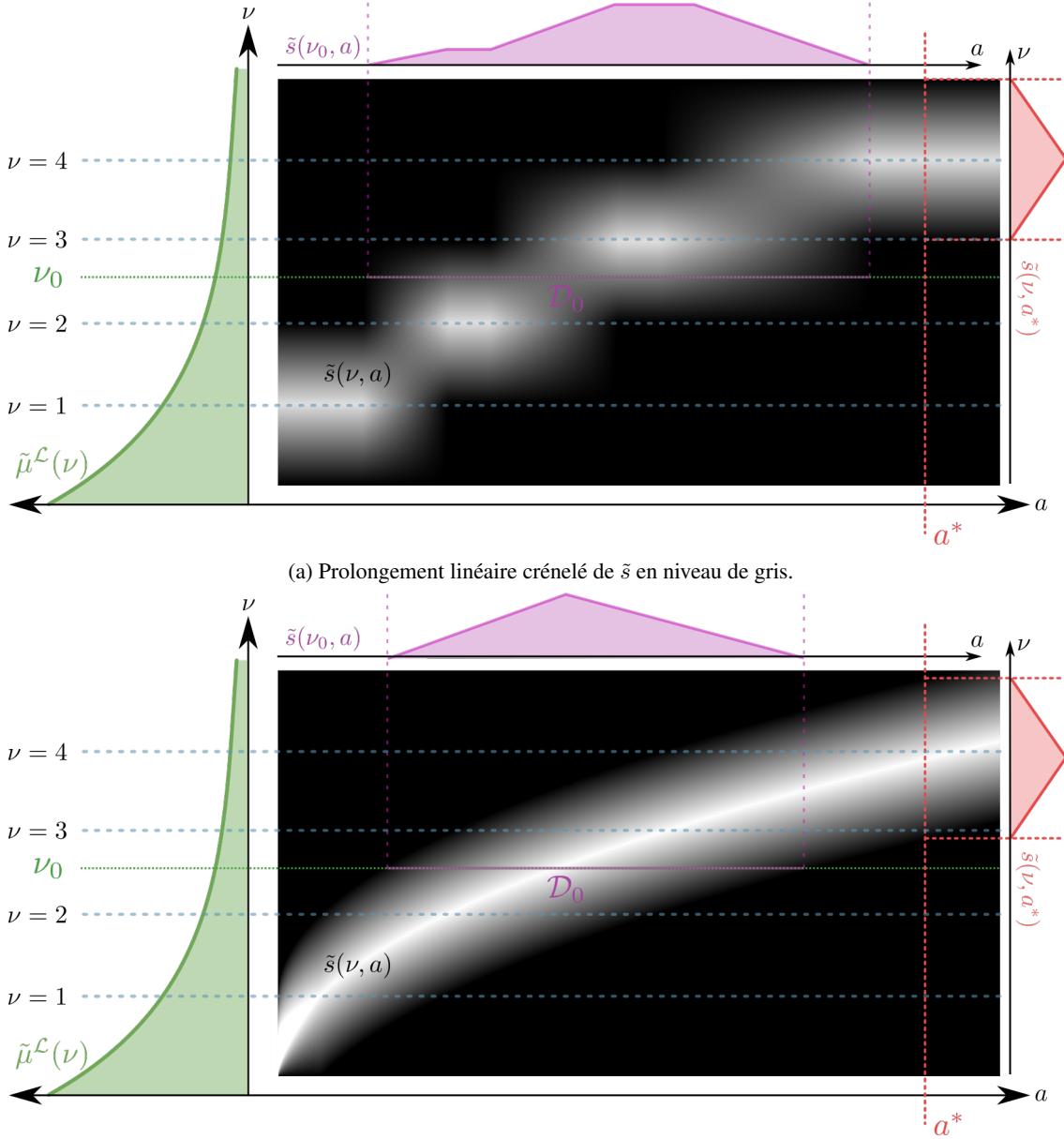


FIGURE 4.10 – Sur chaque figure, la partie centrale représente la fonction \tilde{s} en niveaux de gris, où le noir correspond à la valeur 0 et le blanc à 1. Les abscisses a représentent la région d'influence tandis que les ordonnées ν représentent le niveau choisi. Le choix de ν se fait à l'aide de la densité de probabilité continue représentée en vert à gauche par la fonction $\tilde{\mu}^L(\nu)$. Ainsi, pour un choix ν_0 donné, l'intensité du VPL associé sera multipliée par la fonction de support $a \mapsto \tilde{s}(\nu_0, a)$ dont le graphe est représenté en mauve au dessus et dont le domaine de définition borné est représenté par \mathcal{D}_0 . Enfin, pour chaque a^* fixé, l'application $\nu \mapsto \tilde{s}(\nu, a^*)$, dont le graphe est représenté à droite en rouge, correspond à l'axe sur lequel notre contrainte de partitionnement de l'unité s'applique. Ce sont donc uniquement ces fonctions qui doivent respecter le fait que leurs intégrales se soient égales à 1. Comme nous avons choisi ici de prolonger linéairement, leurs graphes sont naturellement de forme triangulaire.

Par ailleurs, entre les deux prolongements ci-dessus, on observe que bien que l'application \tilde{s} soit lisse dans les deux cas, le domaine de définition \mathcal{D}_0 , quant à lui, évolue de manière abrupte lorsque ν_0 varie, dans la solution 4.10a d'autant qu'entre deux valeurs entières de ν les variations de la fonction de support évoluent elles aussi fortement. La solution 4.10b, quant à elle, ne présente pas ces inconvénients.

La figure 4.10 illustre ce choix de densité. Les faibles valeurs de ν correspondant aux zones d'influence locales, cette répartition permet donc de convenablement prolonger l'idée des coupes antérogrades. On note cependant que désormais les ν peuvent prendre n'importe quelle valeur sur \mathbb{R}^+ alors que les fonctions de support ont tout à fait le droit d'être identiquement nulle à partir d'un certain ν . Il faut donc prendre soin de faire correspondre au mieux les domaines de définition de \tilde{s} de sorte à ce qu'un minimum de calculs soient perdus dans le choix d'un ν trop grand.

Construction de la variable Afin de construire une variable suivant une telle densité de probabilité, on se donne une variable aléatoire u uniforme sur $]0, 1[$ et la fonction de répartition

$$M(\nu) = \int_{t \in [0, \nu]} \mu^{\mathcal{L}}(t) dt = 1 - e^{-\lambda^{\mathcal{L}} \nu}. \quad (4.43)$$

On peut alors montrer que la variable ν définie par

$$\nu = M^{-1}(u) = -\frac{\ln(1-u)}{\lambda^{\mathcal{L}}} \quad (4.44)$$

suit une loi exponentielle de paramètre $\lambda^{\mathcal{L}}$.

Fonctions de support L'objectif de cette section est de partir de la solution très discontinue $\tilde{s}_{(\text{CAL})}$, cf. formule 4.37, et d'en construire une interpolation lisse \tilde{s} sur tout le domaine $\mathcal{V} \times \mathbb{R}^+$. Pour cela, on note que, en chaque point de $a \in \mathbb{R}^+$, il existe soit exactement une fonction s_k , soit exactement deux fonctions s_k et s_{k+1} qui ont une valeur non nulle, cf. figure 4.8. Notre première idée consiste donc à construire une interpolation linéaire à la manière des interpolations splines d'ordre 1. Plus formellement, on définit notre reconstruction linéaire par :

$$\tilde{s}(\nu, a) = \begin{cases} (\nu - (k-1))s_k(a) & \text{si } s_{k-1}(a) = 0 \text{ et } s_k(a) \neq 0 \text{ pour } \nu \in [k-1, k[\\ (k+1-\nu)s_k(a) & \text{si } s_k(a) \neq 0 \text{ et } s_{k+1}(a) = 0 \text{ pour } \nu \in [k, k+1[\\ (k+1-\nu)s_k(a) + (\nu - k)s_{k+1}(a) & \text{si } s_k(a) \neq 0 \text{ et } s_{k+1}(a) \neq 0 \text{ pour } \nu \in [k, k+1[\\ 0 & \text{sinon} \end{cases}. \quad (4.45)$$

La figure 4.9 montre le graphe de cette fonction. La figure 4.10a montre la même fonction en niveau de gris. On remarque alors plusieurs choses. Tout d'abord, cela répond bien à notre problème dans le sens où pour chaque valeur de ν que l'on choisit aléatoirement dans $]0, K[$, on obtient une fonction de support bornée en a . Par ailleurs on note que les petites valeurs de ν correspondent à l'échantillonnage dense de VPL . Ainsi le fait que la zone inférieure droite de la carte de hauteur soit nulle est de première importance afin de limiter la contribution de nombreux VPL . En revanche, cette solution n'est pas satisfaisante car l'effet de palier n'est pas résorbé et l'on passe de manière abrupte d'une distance d'influence à son double très rapidement lorsque ν croît. En s'inspirant de cette solution, nous en fournissons à présent une nouvelle qui suit globalement les formes de cette fonction mais dont la distance des supports évolue, quant à elle, continûment avec ν .

Nous paramétrons ainsi notre fonction selon 3 courbes cf. figure 4.11. Tout d'abord nous définissons la courbe des maxima locaux ν_{mil} qui, inspiré par notre première solution, suivra grossièrement un

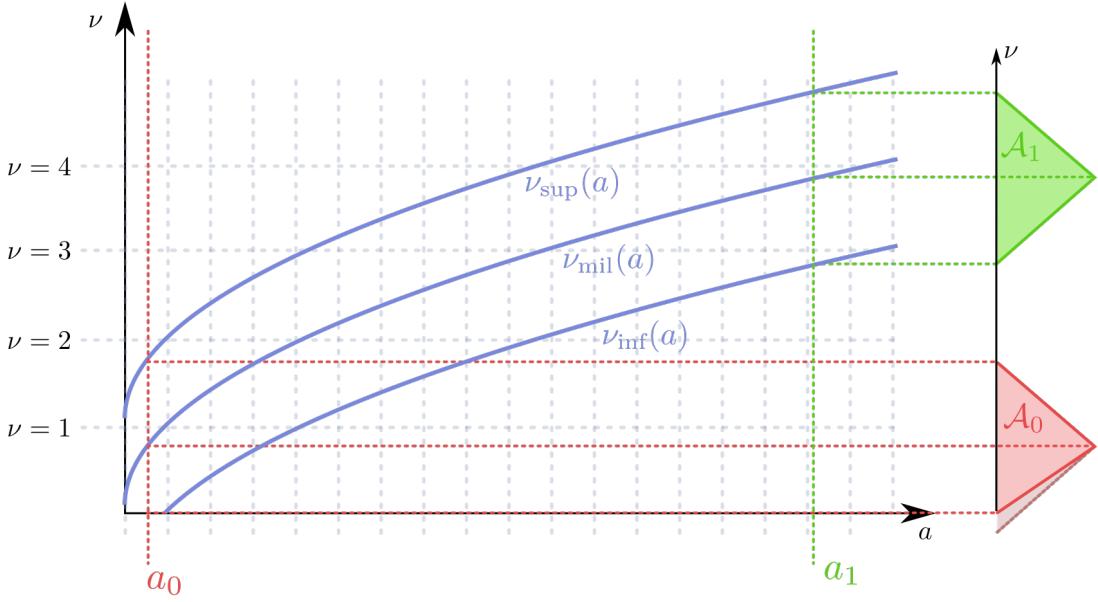


FIGURE 4.11 – Définition des bornes du prolongement linéaire lisse de \tilde{s} . La fonction \tilde{s} est encadrée par les deux courbes ν_{inf} et ν_{sup} et prend sa valeur maximale locale sur la courbe ν_{mid} . Ces trois courbes sont translatées entre elles selon l’axe vertical. Pour chaque valeur de a , on interpole linéairement de 0 à 1 pour ν variant de $\nu_{\text{inf}}(a)$ à $\nu_{\text{mid}}(a)$ et de 1 à 0 pour la partie supérieure. Pour toute valeur de a_1 telle que la courbe ν_{inf} est strictement au-dessus de l’axe des abscisses, le triangle généré par l’interpolation (en haut en vert) est d’aire \mathcal{A}_1 égale à 1. La condition de partition de l’unité cf. formule 4.40 est donc vérifiée. Pour a_0 ne vérifiant pas cette condition, le triangle généré (en bas en rouge) est d’aire \mathcal{A}_0 inférieure à 1. La fonction \tilde{s} sera donc normalisée par l’aire de ce triangle.

logarithme. Ensuite nous encadrerons en haut ν_{sup} et en bas ν_{inf} cette fonction par une translation de une unité de sorte que :

$$\begin{aligned}\nu_{\text{mil}}(a) &= \lambda^D \log(1 + a) \\ \nu_{\text{sup}}(a) &= \nu_{\text{mil}}(a) + 1 \\ \nu_{\text{inf}}(a) &= \max(0, \nu_{\text{mil}}(a) - 1)\end{aligned}, \quad (4.46)$$

où λ^D est un facteur d’échelle qui est laissé libre.

Enfin, pour tout $a \in \mathbb{R}^+$ et toute valeur de $\nu \in [\nu_{\text{inf}}(a), \nu_{\text{sup}}(a)]$, on construit \tilde{s} comme l’interpolation linéaire suivante :

$$\tilde{s}(\nu, a) = \begin{cases} \frac{1}{\mathcal{A}_{\text{ti}}(a)} \frac{\nu - \nu_{\text{inf}}(a)}{\nu_{\text{mil}}(a) - \nu_{\text{inf}}(a)} & \text{si } \nu \in [\nu_{\text{inf}}(a), \nu_{\text{mid}}(a)] \\ \frac{\nu_{\text{sup}}(a) - \nu}{\mathcal{A}_{\text{ti}}(a)} & \text{si } \nu \in [\nu_{\text{mid}}(a), \nu_{\text{sup}}(a)] \\ 0 & \text{sinon} \end{cases}, \quad (4.47)$$

où

$$\mathcal{A}_{\text{ti}}(a) = \min \left(1, \frac{\nu_{\text{mid}}(a) + 1}{2} \right) \quad (4.48)$$

est l’aire du triangle interpolant tel que défini sur la figure figure 4.11 de sorte à ce que la condition de partition de l’unité formule 4.40 soit vérifiée. Nous illustrons également cette dernière solution en comparaison avec la solution trivialement prolongée sur la figure figure 4.10b.

	Lancer de chemins		Radiosité inst.			
	RMSE	5.38	RMSE	0	#VPL	29336
Radiosité inst.	SSIM	0.9921	SSIM	100%	%pix./VPL	6.60%
	$\Delta_{p>2\%}$	3.86%	$\Delta_{p>2\%}$	0%	#VPL/pix.	1936.94
	RMSE	7.12	RMSE	2.45	#VPL	29336
Coupe ant. (N=64K)	SSIM	0.9835	SSIM	0.9928	%pix./VPL	1.44%
	$\Delta_{p>2\%}$	6.5%	$\Delta_{p>2\%}$	1.53%	#VPL/pix.	424.20
	RMSE	9.50	RMSE	6.63	#VPL	7404
Coupe ant. (N=16K)	SSIM	0.9678	SSIM	0.9747	%pix./VPL	1.5%
	$\Delta_{p>2\%}$	15.62%	$\Delta_{p>2\%}$	10.97%	#VPL/pix.	111.02
	RMSE	14.74	RMSE	12.7	#VPL	954
Coupe ant. (N=2K)	SSIM	0.8923	SSIM	0.9041	%pix./VPL	1.3%
	$\Delta_{p>2\%}$	39.7%	$\Delta_{p>2\%}$	36.63%	#VPL/pix.	12.51

TABLE 4.2 – Table de résultats, coupes antérogrades continues

Pour un ν donné, c'est-à-dire un niveau « continu » choisi pour un *VPL*, le support de $a \mapsto \tilde{s}(\nu, a)$ est alors le segment $[\nu_{\text{sup}}^{-1}(\nu), \nu_{\text{inf}}^{-1}(\nu)]$, cf. figure 4.11. Or l'application réciproque ν_{sup}^{-1} correspond à une fonction exponentielle. Cela signifie que les bornes de la zone d'influence d'un *VPL* évolue exponentiellement par rapport à ν . Cela prolonge de manière cohérente l'évolution géométrique que nous avons donnée pour la définition des aires discrètes $(a_k)_k$ à la section précédente, cf. formule 4.28, justifiant ainsi le lien entre la densité d'échantillonnage surfacique ν et l'aire a des disques imaginaires portés par les *VPL*.

4.3.5 Résultats

La figure 4.12 illustre les résultats de notre formulation continue. Pour pouvoir comparer les résultats obtenus avec cette nouvelle formulation et ceux de la solution discrète, cf. §4.2.6, il nous faut adapter les paramètres, c'est-à-dire $\lambda^{\mathcal{L}}$ et $\lambda^{\mathcal{D}}$. Pour cela, nous transposons tout d'abord l'argument de loi géométrique pour obtenir une loi exponentielle équivalente, ce qui donne $\lambda^{\mathcal{L}} = -\ln(q_N) \approx 1.1$. Le second paramètre de notre paramétrisation, $\lambda^{\mathcal{D}}$ est censé représenter l'évolution de la distance d'influence des fonctions de support. Il correspond donc aux paramètres a_1 et q_a . Empiriquement, nous avons évalué ce paramètre pour faire en sorte que l'approche continue ait le même gain que l'approche discrète, c'est-à-dire que nous avons cherché à faire correspondre la valeur des %pix./VPL. De cette façon, on obtient une évolution équivalente de la mesure SSIM en fonction du gain.

La formulation continue permet ainsi de décrire très simplement une classe de solution à l'aide de deux paramètres. On obtient de plus les mêmes résultats que ceux de la version discrète. Nous pensons, par ailleurs, qu'avec le formalisme que nous avons introduit, dans de futurs travaux, on pourrait contrôler la famille de solutions de manière plus fine et appropriée à la géométrie locale. Cela permettrait alors d'accroître davantage le gain de performance.

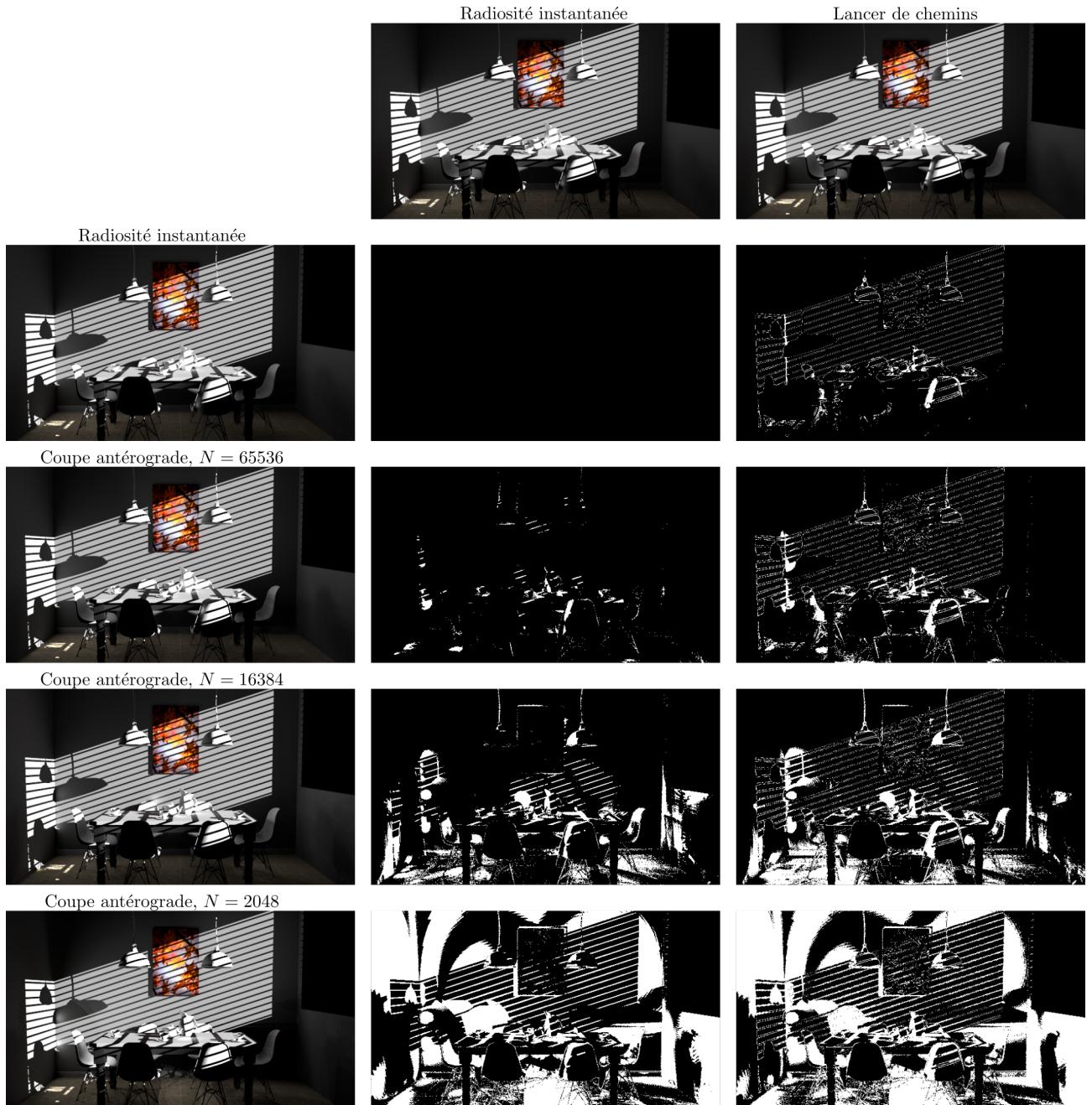


FIGURE 4.12 – Comparaison des coupes antérogrades continues sur une scène d'intérieur. En haut, nos deux références évaluées par radiosité instantanée et par lancer de chemins. À gauche, les images sont produites par notre algorithme en faisant varier N , le nombre de VPL. Au centre, les pixels blancs correspondent aux pixels ayant une différence supérieure à 2% par rapport à la référence.

4.4 Discussion

Dans ce chapitre nous avons introduit les coupes antérogrades de lumières. Pour cela nous nous sommes inspirés des approches antérieures de lumières multiples et de radiosité instantanée dans l'idée de substituer une scène complexe constituée de triangles par un échantillonnage de points. Ces échantillons, alors nommés *VPL*, permettent de capturer l'éclairage incident afin de fournir une réponse approchée d'un rebond lumineux. Cependant, là où les précédentes approches ont recours à des structures accélératrices sur cette nouvelle représentation pour pouvoir passer à l'échelle, nous avons proposé ici une méthode travaillant directement sur les *VPL* sans ajouter d'information supplémentaire sur la géométrie.

Afin de tout de même être en mesure traiter la densité des *VPL* générés, nous avons reformulé l'approche multi-échelle du transport lumineux à l'aide d'une description probabiliste en distribuant des zones d'influences sur nos *VPL*. De plus, contrairement aux approches par lancer de rayons, où chaque pixel est calculé indépendamment des autres de manière probabiliste, notre approche éclaire les pixels affectés par un même *VPL* de manière corrélée. Cela nous permet, dès lors, de supprimer le bruit haute fréquence typique des rendus de Monte-Carlo par lancer de rayons. Nous avons alors évalué nos résultats sur le cas diffus du transport lumineux et avons réussi à réduire la quantité de calcul nécessaire sur plusieurs scènes tout en conservant une distance à la vérité terrain acceptable.

Enfin dans ce chapitre nous nous sommes restreints à l'analyse des interactions diffuses. Nous pensons que notre approche peut s'étendre aux *BRDF* plus brillantes en modifiant la nature des fonctions de support. On constate également que ces fonctions ont une forme relativement homogène sur l'intégralité de la scène, notamment dans les zones d'obscurité; trouver un meilleur critère de répartition des *VPL* permettrait sans doute d'optimiser encore la pertinence des calculs. Nous laisserons en revanche cela à des travaux futurs.

Dans ce chapitre, nous avons ainsi démontré que le transport lumineux peut efficacement être approché à l'aide de sources lumineuses ponctuelles même en nombre modéré. Nous allons ainsi, dans le chapitre suivant, exploiter ces résultats afin de proposer une implémentation temps réel des coupes antérogrades de lumière dans le but de l'appliquer aux scénarios dynamiques sans pour autant se contraindre sur la taille des données à traiter.

RÉÉCHANTILLONNAGE DYNAMIQUE PAR POINTS POUR L'ÉCLAIRAGE GLOBAL

Dans ce chapitre nous nous intéresserons à la mise en œuvre d'un pipeline graphique permettant de transformer un maillage en nuage de points sans avoir à faire d'hypothèse forte sur la géométrie. L'algorithme que nous décrivons peut être entièrement implémenté sur le pipeline de *Shader*, s'adaptant ainsi à tout type de scénario dynamique sans recourir à une utilisation prohibitive de mémoire supplémentaire. Enfin, après avoir démontré les capacités de rééchantillonnage de notre algorithme dans l'objectif de transformer un maillage complexe en une représentation plus simple, nous l'utilisons pour proposer une implémentation temps réel des *coupes antérogrades de lumière* présentées au chapitre précédent.

5.1 Introduction

Comme nous l'avons vu au cours des chapitres précédents, la simulation du transport lumineux constitue un composant important du réalisme des images. Malheureusement, son calcul en temps réel demeure, aujourd’hui encore, un défi. Il nécessite, en effet, de nombreux calculs s'avérant peu compatibles avec les scénarios dynamiques. La plupart des implémentations temps réel partent du constat que l'éclairage diffus indirect est un phénomène de basse fréquence, et ce d'autant plus que l'émetteur et le receveur sont éloignés. À partir de là, il est courant d'adopter une approche hiérarchique sur un ensemble de *VPL* pour estimer l'éclairage indirect, cf. §3.2.5. Cependant, pour cette catégorie de méthodes, la gestion de scènes dynamiques constituées de géométries massives implique au moins deux limitations majeures. Tout d'abord la structure d'accélération sur les *VPL* doit être recalculée à chaque trame. En effet, le fait de détecter et de mettre à jour les seules parties de la structure ayant été altérées est souvent incompatible avec le calcul parallèle, et s'avère, en général, plus lent. De plus, l'ensemble des *VPL* nécessaires pour représenter le champ lumineux peut être extrêmement volumineux. Le traitement de données de cette taille peut alors ne pas être réalisable en temps réel.

Pour cela, nous proposons ici d'exploiter nos résultats sur les *coupes antérogrades*, développées au chapitre précédent, et d'en proposer une implémentation temps réel. Nous développons ainsi une approche en deux temps. Tout d'abord, nous observons que les triangles de la scène, eux-mêmes, peuvent être utilisés pour porter les *VPL*. Nous introduisons alors deux fonctionnalités : l'une amplifiant la géométrie et l'autre la décimant. Elles sont implémentées à l'aide du *Tessella-*



FIGURE 5.1 – Rendu temps réel du premier rebond lumineux avec notre algorithme. La scène, constituée de 7 millions de triangles, est rendue en 1080p en 16 ms et n'a pas nécessité de préparation.

tion Shader et du *Geometry Shader* et nous servent à rééchantillonner, à la volée, la géométrie pour construire notre ensemble de *VPL*. Nous raffinons les parties de la scène trop grossières pour correctement capturer la luminance, et nous simplifions la géométrie là où la densité de triangles est trop forte, cf. §5.5. Ensuite, nous distribuons sur notre échantillonnage, selon une loi de probabilité, les fonctions de support que nous avons développées au chapitre précédent. Grâce à cela, nous construisons une répartition multi-échelles de nos *VPL*, permettant d'adapter leurs contributions et, surtout, de borner en espace image le nombre de pixels qu'ils influencent. En couplant la génération en temps réel des *VPL* à nos fonctions de support, nous fournissons une implémentation des *coupes antérogrades* entièrement réalisée sur processeur graphique, cf. §5.6. Enfin, nous démontrons l'applicabilité de notre algorithme sur des scénarios dynamiques dont les scènes mélagent à la fois des zones très détaillées et très grossières, cf. §5.7.

5.2 Travaux Antérieurs

Au chapitre 3 nous avons présenté un ensemble de techniques visant à simuler l'éclairage global et à le calculer en temps réel. Nous n'allons donc pas détailler de nouveau l'ensemble de ces méthodes ici. En revanche, nous nous intéressons à deux grandes approches fondées sur les *caches* auxquels nous nous comparons en fin de chapitre, cf. §5.7.

Comme nous l'avons vu précédemment, cf. §3.2.4, les approches en espace image constituent la classe de solutions la plus populaire pour approcher le calcul de l'éclairage global en temps réel. En effet, le fait d'utiliser les pixels du point de vue principal permet d'échantillonner sans surcoût la géométrie et de fournir une approximation de l'éclairage indirect [Ritschel et coll., 2009b, Soler et coll., 2010]. Cependant, malgré les performances temps réel de ces approches et leur capacité à gérer des scènes complètement dynamiques, une quantité importante d'information est perdue par l'échantillonnage en espace écran. En effet, les zones en dehors du champ de vision ainsi que les parties cachées par d'autres objets de la scène ne sont pas échantillonnées. Cela peut être problématique car ils pourraient, tout de même, avoir une influence sur les pixels visibles ; en particulier lorsqu'une région avec une forte influence devient subitement visible ou cachée.

Dans notre approche, nous travaillons dans l'espace des objets pour éviter ce genre d'artefacts. Cependant, construire l'échantillonnage dans l'espace des objets est moins évident que de construire le tampon géométrique des approches en espace image. Néanmoins, nous avons vu, cf. §3.2.3,

qu'en distribuant un ensemble de *VPL* sur la scène, il est possible d'approcher un rebond lumineux [Keller, 1997]. Les *VPL* étant construits indépendamment du point de vue, leur contribution reste stable quelles que soient les parties visibles de la scène. Cependant, comme le montrent les approches hiérarchiques [Walter et coll., 2005, Christensen, 2008, Ritschel et coll., 2009a], calculer la contribution de tous les *VPL* sur tous les pixels est extrêmement lent et, le plus souvent, ce n'est pas nécessaire. En revanche, pour mettre en œuvre ces approches, il faut avoir recours à des structures d'accélération peu compatibles avec le calcul sur processeur graphique. Ainsi, pour exploiter le calcul parallèle à grain fin, nous proposons d'utiliser notre formulation des *coupes antérogrades*, cf. chapitre 4, pour calculer dynamiquement et de manière indépendante, l'ensemble des *VPL* ainsi que leurs différents niveaux de résolution.

Enfin, Nalbach et coll. [Nalbach et coll., 2014] proposent une approche exploitant les avantages des deux mondes. Ainsi, comme pour les approches en espace objet, ils proposent de construire les *VPL* à la surface de la géométrie même si celle-ci est en dehors du champ de vision; mais comme pour les approches en espace écran, ils tirent profit des accélérations matérielles du processeur graphique, en utilisant l'unité de tessellation (à la place de l'unité de rastérisation) pour échantillonner la surface par des *VPL*. Cependant, même si cette approche permet de gérer des scènes de taille modérée, elle ne permet pas de passer à l'échelle sur des scènes massives où les contraintes de temps réel imposent un besoin de décimation de la géométrie plutôt qu'un raffinement. Notre technique fait un pas dans cette direction en permettant de soit raffiner, soit simplifier la géométrie portant les *VPL*, grâce à une stratégie en deux temps. Nous exploitons ainsi, à la fois, l'unité de tessellation et le *Geometry Shader* pour ajuster la résolution de notre échantillonnage en travaillant sur des scènes de plusieurs dizaines de millions de triangles. En particulier, on est en mesure d'atteindre des performances temps réel en calculant, en une seule passe, un échantillonnage à plusieurs résolutions sans avoir à construire ni maintenir de structure de données au cours du temps.

5.3 Vue d'ensemble

Notre algorithme, que nous illustrons par la figure 5.2, possède la structure classique d'un pipeline fondé sur les *VPL* composée de deux principales étapes : la génération des *VPL* et le calcul de l'éclairage avec ces sources lumineuses. Dans ce chapitre, nos contributions se focalisent sur le fait de construire efficacement le nuage de *VPL* et d'utiliser cet échantillonnage pour calculer les *coupes antérogrades* que nous avons introduites précédemment. On peut alors les résumer de la manière suivante :

- au chargement de la scène, nous associons à chaque sommet un attribut entier ; cette valeur est ensuite utilisée pour construire un nombre aléatoire au niveau du triangle qui reste cohérent au cours des trames, même si la géométrie subit des transformations dynamiques, cf. §5.5.1 ;
- l'ensemble de tous les triangles est alors aléatoirement partitionné, en plusieurs niveaux de résolution, selon une distribution de probabilité ; pour chaque triangle, un *VPL* est généré de manière stochastique de sorte que sa luminance soit seulement déterminée par le niveau dans lequel il tombe, $(\mathcal{L}^1, \dots, \mathcal{L}^K)$, cf. §5.4.9 ; pour répartir le travail sur les unités de calcul, les fonctions de support, w , des *coupes antérogrades*, cf. chapitre 4, sont distribuées sur les *VPL* de sorte que les échantillonnages denses, (\mathcal{L}^1, \dots) , aient une zone d'influence restreinte

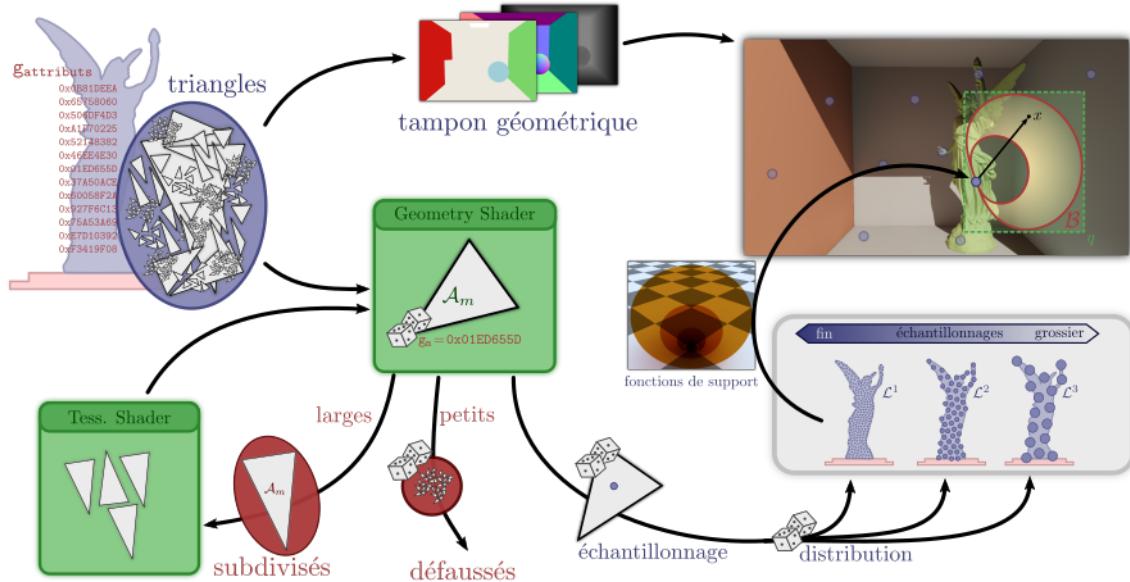


FIGURE 5.2 – Vue d’ensemble de notre algorithme d’éclairage global. La géométrie initiale est constituée de triangles de toutes tailles. Pour les partitionner le plus uniformément possible, il nous est nécessaire de construire une variable aléatoire par triangle. On réalise ce calcul au *Geometry Shader* à l’aide d’attributs entiers ajoutés sur les sommets et calculés au chargement du modèle. Ensuite, en fonction de la taille du triangle, plusieurs choix sont possibles : les larges triangles sont stockés dans un tampon intermédiaire pour être ultérieurement subdivisés, les petits sont aléatoirement défaussés proportionnellement à leur taille pour ne pas suréchantillonner une partie de la scène, et les autres triangles sont distribués aléatoirement en plusieurs groupes pour former un échantillonnage à plusieurs résolutions. Enfin, selon le niveau de résolution, une fonction de support, définie par nos *coupes antérogrades*, est construite sur le *VPL*. Le triangle portant le *VPL* construit est alors transformé, par le *Geometry Shader*, en quadrilatère englobant le support, de sorte à accumuler, pour tous les pixels ainsi touchés, la contribution lumineuse de ce *VPL*.

et que les échantillonnages grossiers, (\dots, \mathcal{L}^K) , aient une zone d’influence lointaine, cf. §5.6.2 ;

- par ailleurs, pour significativement réduire le nombre de triangles à traiter, tout en préservant la qualité de l’échantillonnage, une partie des triangles (majoritairement composée de petits triangles) est complètement défaussée ;
- de plus, pour satisfaire aux restrictions matérielles liées à l’amplification de la géométrie au *Geometry Shader*, nous nous limitons à construire, au plus, un *VPL* par triangle, cf. §5.5.2 ; en conséquence, les gros triangles ne peuvent plus être convenablement échantillonnés selon la stratégie que nous venons d’introduire, ce qui produit d’importants artéfacts visuels ; nous stockons donc leurs coordonnées dans un tampon intermédiaire avant de les subdiviser dans un second temps via l’unité de tessellation, cf. §5.5.3 ;
- enfin, en calculant les bornes, en espace image, des fonctions de support, on accumule la contribution des *VPL* dans un mécanisme semblable au rendu différé ; cette dernière étape est possible directement après avoir construit l’échantillon, en transformant, grâce au *Geometry Shader*, le triangles portant le *VPL* en un quadrilatère perpendiculaire à l’axe de vue de la caméra, qui englobe son support en espace image, cf. §5.6.3.

Dans la suite nous procérons de la manière suivante pour décrire notre algorithme. Dans un premier temps, cf. §5.4, nous décrivons le problème abstrait consistant à construire un échantillonnage sur le maillage dans le but de calculer une intégrale quelconque. Dans un second temps, cf. §5.5, nous proposons une implémentation temps réel, fondée sur les étapes de traitement géométrique des *Shaders*, pour construire, à la volée, un échantillonnage à partir de n'importe quel maillage. Puis, cf. §5.6, nous faisons le lien entre le calcul d'intégrale décrit plus tôt et le calcul d'éclairage par *coupes antérogrades*. Enfin, cf. §5.7, nous évaluons notre approche, sur un jeu de modèles constitués de maillages massifs.

5.4 Construction à grain fin d'échantillonnages uniformes pour le calcul d'intégrales

Dans le but d'être facilement intégré à un moteur de rendu existant, notre mécanisme d'échantillonnage est fondé sur un pipeline standard de rastérisation. En effet, notre algorithme ne requiert pas plus de préparation sur la géométrie que celle nécessaire à un rendu classique de la scène, par exemple pour le calcul du tampon géométrique ou de l'éclairage direct, cf. §2.2.4. Par ailleurs, afin d'exploiter au maximum le parallélisme à grain fin du processeur graphique, nous proposons de générer chaque échantillon indépendamment les uns des autres selon une densité surfacique définie par l'utilisateur.

Enfin, grâce à notre utilisation du pipeline graphique et de la création indépendante des échantillons, à aucun moment il ne nous est nécessaire de connaître l'intégralité des échantillons. Ainsi, en application, cf. §5.6, il nous est possible de consommer l'échantillonnage directement dans un *Fragment Shader* sans avoir besoin de le stocker dans une mémoire intermédiaire. Nous pré-servons ainsi à la fois la consommation mémoire et les transferts de données entre la mémoire principale et les coeurs de calcul, cf. §2.2. Tout cela nous permet finalement de faire passer notre algorithme à l'échelle en fonction de la taille de la géométrie et de gérer les scènes dynamiques.

5.4.1 Définition du problème

Dans cette première partie, nous nous donnons une scène \mathcal{S} constituée de M triangles $\{t_1, \dots, t_M\}$ pour laquelle nous chercherons un échantillonnage de N points $\mathcal{L} = \{y_1, \dots, y_N\}$. En se donnant par ailleurs une application $f : \mathcal{S} \rightarrow \mathbb{R}$ notre but est d'approcher le calcul de l'intégrale suivante :

$$I_f = \int_{y \in \mathcal{S}} f(y) d\mathcal{A}_y \simeq \frac{1}{N} \sum_{y_n \in \mathcal{L}} \frac{f(y_n)}{\mu^{\mathcal{S}}(y_n)} = \hat{I}_f(\mathcal{L}), \quad (5.1)$$

où $\mu^{\mathcal{S}}(y_n)$ est une densité de probabilité à la surface du maillage. Dans la suite de tout ce chapitre nous faisons l'hypothèse simplificatrice que cette densité de probabilité est uniforme sur tout le maillage. En notant $|\mathcal{S}|$ l'aire totale du maillage, on a donc :

$$\mu^{\mathcal{S}}(y_n) = \frac{1}{|\mathcal{S}|}.$$

De plus, en définissant l'aire moyenne $\eta = \frac{|\mathcal{S}|}{N}$ on a enfin :

$$\hat{I}_f(\mathcal{L}) = \sum_{y_n \in \mathcal{L}} f(y_n) \eta.$$

Il nous reste à présent à définir un tel échantillonnage ainsi qu'un critère qui nous permet de dire si $\hat{I}_f(\mathcal{L})$ est une approximation « satisfaisante » de I_f . Cependant, étant donné que l'on cherche à produire cet échantillonnage en temps réel en utilisant au mieux la puissance du processeur graphique, nous devons tout d'abord établir l'ensemble des contraintes que cela pose.

5.4.2 Contraintes

Nous avons pour objectif ici d'échantillonner la surface de la scène en temps réel en utilisant les capacités du processeur graphique. Or, afin d'en avoir une utilisation efficace, les noeuds de calcul des processeurs graphiques doivent pouvoir travailler, en parallèle, sur des tailles de données les plus petites possibles. Ainsi, puisque nos échantillons dépendent d'un critère surfacique, la plus petite primitive envisageable est le triangle. Ce dernier nous permet d'avoir accès à l'aire locale tout en garantissant que l'échantillon construit appartient bien à la surface. Ces deux propriétés sont plus difficiles à garantir en travaillant sur d'autres types de primitive comme les sommets par exemple. De plus, d'un point de vue matériel, le *Geometry Shader* que nous utilisons est parfaitement adapté pour travailler à cette granularité.

Une seconde contrainte induite par le calcul massivement parallèle est l'extrême inefficacité des branches divergentes, c'est-à-dire qu'il est *a priori* très inefficace d'effectuer une quantité de calcul différente d'une primitive à l'autre. Dans notre cas, cela signifie qu'il est grandement préférable de produire la même quantité d'échantillons pour chaque triangle traité par un cœur de calcul. De plus, étant donnée l'architecture sous forme de pipeline, l'exécution des programmes graphiques ne permet pas d'accéder à une grande quantité de mémoire. C'est pourquoi on s'impose de construire, au plus, un échantillon par triangle au niveau du *Geometry Shader*. Les unités de tessellation, en revanche, sont particulièrement adaptées pour amplifier la donnée. Nous restreignons ainsi la subdivision de la géométrie à celles-ci.

5.4.3 Solution Naïve

Au vu des contraintes ci-dessus, il semble possible d'imaginer un algorithme naïf y répondant, cf. figure 5.3. L'idée consiste à remarquer qu'un triangle avec une aire $\mathcal{A}_{triangle}$ proche de η est un bon candidat pour contenir un échantillon de \mathcal{L} . En effet, si, idéalement, le maillage n'était constitué que de triangles de cette taille, produire un échantillon par triangle reviendrait à construire un échantillonnage globalement équidistribué sur la surface. Cependant, comme notre maillage est en général composé de triangles de toute sorte, notre idée de base consiste alors à filtrer tous les triangles en fonction de leur aire, de sorte à ne conserver que les triangles dont l'aire est « proche » de η . Sur chacun de ces triangles nous construisons un unique échantillon.

5.4.4 Angle d'attaque

Bien que l'algorithme décrit ci-dessus présente l'avantage d'être extrêmement simple, cette approche ne peut pas fonctionner pour au moins trois raisons.

Il nous faut tout d'abord remarquer qu'en ne produisant qu'au plus un échantillon par triangle, les gros triangles, c'est-à-dire ceux ayant une grande surface, ne peuvent pas être correctement échantillonnés alors que leur contribution est importante. Il nous faut donc définir un critère per-

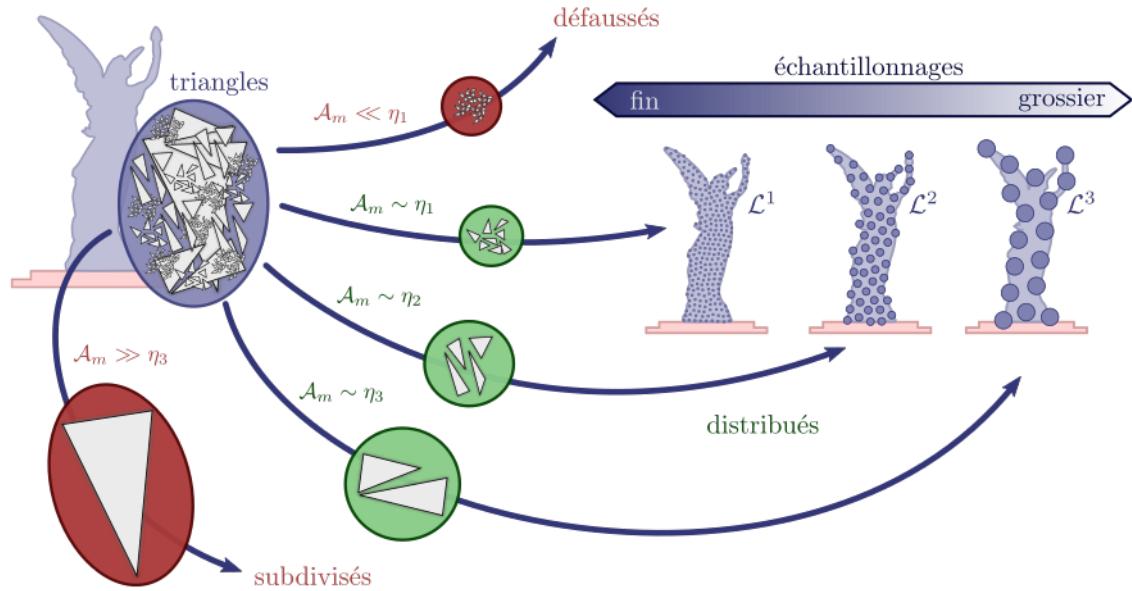


FIGURE 5.3 – Algorithme de multi-échantillonnage naïf. La géométrie est composée de toutes sortes de triangles. L'idée consiste à distribuer les triangles en fonction de leur taille de sorte à ce que leur aire, \mathcal{A}_m , soit comparable à l'aire moyenne η_k représentée par les échantillons de \mathcal{L}^k . En faisant porter un échantillon par chaque triangle tombant dans l'un des ensembles \mathcal{L}^k , on espère extraire ici trois échantillonnages dont la densité surfacique est $\frac{1}{\eta_k}$. Dans cette situation, les très petits triangles, c'est-à-dire ceux étant plus petits que toutes les densités cibles, sont exclus de l'échantillonnage. De même, les très gros triangles ne peuvent être attribués à aucun ensemble.

mettant de les détecter ainsi que de les raffiner dynamiquement. Cette étape doit être réalisée dans un processus à part afin de préserver la charge de travail répartie équitablement sur les coeurs de calcul, cf. §5.4.5.

De plus, les triangles de petite taille risquent d'être complètement ignorés par cette approche. En effet, il est tout à fait probable qu'une région de la scène dense en triangles puisse avoir une aire totale importante tandis que, pris séparément les uns des autres, les triangles constituant cette région aient une aire négligeable et soient défaussés. Afin de prendre en compte l'importance collective de ce type de triangles, sans perdre le caractère hautement parallèle de notre approche, nous utilisons un critère stochastique, dépendant de l'aire du triangle, pour déterminer, d'un point de vue individuel, si celui-ci doit être échantillonné ou pas, cf. §5.4.7.

Enfin, on note que l'expression « \mathcal{A}_m proche de η » mérite une précision plus formelle. Cependant, quand bien même on pourrait, dans un premier temps du moins, lui attribuer le sens commun intuitif, on constate que ce critère n'a de sens que si le maillage est majoritairement constitué de triangles de tailles homogènes. Or ce critère de taille étant dépendant de facteurs extrinsèques au maillage, c'est-à-dire η défini par l'utilisateur. Il est donc impossible que la condition soit satisfaisable en général. Cependant, si la condition d'homogénéité était satisfaite, cette approche permettrait de produire une répartition uniforme sur le maillage. Nous voyons dans la suite que, grâce à un critère de discrimination non déterministe dépendant de l'aire des triangles, nous parvenons à nous rapporter, en espérance, à ce cas d'utilisation quelle que soit la répartition initiale des tailles de triangles, cf. §5.4.8.

En s'inspirant de cette solution naïve et, en prenant en compte les défauts que nous venons d'identifier, nous redéfinissons à présent formellement notre problème d'échantillonnage dans le but d'adapter et d'étudier notre solution.

5.4.5 Échantillonnage par triangle

En distribuant le calcul de l'échantillonnage à l'échelle du triangle, tel que nous le décrivons dans notre approche naïve, il ne nous est, dès lors, plus possible de construire un estimateur de Monte-Carlo au sens où nous l'avons défini, *cf.* §2.1.5. Cependant, nous pouvons tout de même réécrire I_f , *cf.* formule 5.1, comme la somme des intégrales sur les triangles constituant la scène, c'est-à-dire :

$$\forall f, \quad I_f = \sum_{m \in [1, M]} \int_{y \in t_m} f(y) d\mathcal{A}_y. \quad (5.2)$$

On se propose alors de construire un échantillonnage visant à approcher le calcul de cette somme d'intégrales. Pour cela on se donne une paire de variables aléatoires indépendantes l'une de l'autre (Y_m, A_m) à valeurs dans $t_m \times \mathbb{R}^+$ pour chaque triangle t_m de la scène. De plus, on impose Y_m comme uniformément répartie sur le triangle t_m et A_m comme devant vérifier :

$$\forall m \in [1, M], \quad \mathbb{E}[A_m] = \mathcal{A}_m. \quad (5.3)$$

En posant alors :

$$\hat{I}_f(Y_m) = f(Y_m)A_m, \quad (5.4)$$

on définit notre nouvel estimateur pour l'échantillonnage $\mathcal{L} = \{Y_1, \dots, Y_M\}$ par :

$$\hat{I}_f(\mathcal{L}) = \sum_{m \in [1, M]} \hat{I}_f(Y_m) = \sum_{m \in [1, M]} f(Y_m)A_m. \quad (5.5)$$

En construisant Y_m et A_m indépendantes l'une de l'autre, $\hat{I}_f(\mathcal{L})$ constitue alors bien un estimateur non biaisé de I_f . En effet, on remarque tout d'abord :

$$\mathbb{E}[f(Y_m)] = \frac{1}{\mathcal{A}_m} \int_{y \in t_m} f(y) d\mathcal{A}_y,$$

puis :

$$\mathbb{E}[\hat{I}_f(\mathcal{L})] = \sum_{m \in [1, M]} \mathbb{E}[f(Y_m)A_m] = \sum_{m \in [1, M]} \mathbb{E}[f(Y_m)] \mathbb{E}[A_m] = I_f. \quad \square$$

On note enfin que, pour obtenir la propriété non biaisée, il suffit que, pour chaque m , Y_m et A_m soient indépendantes. En outre, si pour deux indices différents, m_1 et m_2 , A_{m_1} et A_{m_2} sont corrélés, ou Y_{m_1} et Y_{m_2} , ou encore A_{m_1} et Y_{m_2} , le calcul de l'espérance de $\hat{I}_f(\mathcal{L})$ reste inchangé. Cette remarque nous permet à présent de lier cet estimateur à la définition première de notre problème, *cf.* §5.4.1.

5.4.6 Validité de l'échantillonnage

À partir de cette définition, bien que $\hat{I}_f(\mathcal{L})$ soit un estimateur non biaisé de I_f , on ne peut plus le considérer comme une approximation « fiable » en invoquant l'argument de la loi forte des grands nombres. En effet, même si l'on produit un grand nombre de *VPL* pour un estimateur $\hat{I}_f(\mathcal{L})$, ceux-ci ne sont pas construit selon la même loi de probabilité. Cela confirme que l'échantillonnage \mathcal{L} utilisé pour construire $\hat{I}_f(\mathcal{L})$ n'est *a priori* pas un échantillonnage valide pour une méthode de Monte-Carlo. Cependant, on est tout de même en mesure de fournir deux propriétés intéressantes sur cet échantillonnage.

Tout d'abord le fait que $\hat{I}_f(\mathcal{L})$, et plus précisément $\hat{I}_f(Y_m)$, soit un estimateur non biaisé signifie que si on est en mesure de produire N échantillonnages $(\mathcal{L}^n = \{Y_1^n, \dots, Y_M^n\})_n$ tels que, pour tout triangle t_m , les variables $\{Y_m^1, \dots, Y_m^N\}$ soient indépendantes et identiquement distribuées selon une loi uniforme sur le triangle, alors la moyenne empirique :

$$\bar{I}_f^N = \frac{1}{N} \sum_{n \in [1, N]} \hat{I}_f(\mathcal{L}^n) = \sum_{m \in [1, M]} \underbrace{\frac{1}{N} \sum_{n \in [1, N]} \hat{I}_f(Y_m^n)}_{\xrightarrow{N \rightarrow \infty} \mathbb{E}[f(Y_m)] \mathbb{E}[A_m]}$$

forme un estimateur convergeant vers l'intégrale I_f par la loi des grands nombres. On note en outre que cette affirmation demeure vraie si deux échantillonnages \mathcal{L}^{n_1} et \mathcal{L}^{n_2} sont corrélés mais que pour un triangle donné, les échantillons construits sont, quant à eux, indépendants. De cette façon on reste en mesure d'approcher notre intégrale aussi précisément que l'on veut en tirant davantage d'échantillons.

Par ailleurs, nous n'oublions pas que l'on cherche ici à produire un échantillonnage dans le cadre d'un scénario temps réel. Cela signifie qu'il est acceptable d'avoir une solution approximative du calcul de cette intégrale. Cependant, on note tout de même que la façon de procéder qui consiste à échantillonner par un point aléatoire chaque partie d'une subdivision de l'espace est proche de ce que l'on nomme « échantillonnage par perturbations » (ou *jittered sampling* en anglais). Cette approche est notamment étudiée dans les approches dites d'échantillonnage à bruit bleu et de basse discrépance [Cook, 1986]. Nous ne développons pas ici d'analyse sur la nature de l'échantillonnage que nous produisons. Nous nous contenterons en effet d'une analyse empirique démontrant qu'en moyenne, pour l'objectif que nous visons, notre stratégie d'échantillonnage produit des résultats comparables à ceux que l'on obtient avec un échantillonnage uniforme.

5.4.7 Répartition stochastique uniforme

Dans ce paragraphe, nous supposons que tous les triangles du maillage ont une aire inférieure à un seuil η_0 défini par l'utilisateur. Nous voyons au paragraphe suivant, *cf.* §5.5.3, comment nous rapporter à cette situation quel que soit le maillage initial.

Dans le paragraphe précédent, nous proposons de construire l'échantillonnage en construisant exactement un échantillon par triangle et en évaluant en chaque point Y_m la fonction f multipliée par la variable aléatoire A_m . Cependant, tel quel, cela ne permet pas de résoudre le problème du suréchantillonnage dans les zones très dense en triangles. En revanche, nous avons laissé une

certaine liberté sur la variable aléatoire A_m , n'imposant qu'une contrainte sur son espérance. C'est pourquoi, on se propose ici de la définir par :

$$A_m = \eta_0 \mathbb{1}_{[0,u_m]}(U_m), \quad (5.6)$$

où U_m est une variable uniforme sur $[0, 1]$ et $\mathbb{1}_{[0,u_m]}(U_m)$ est une variable aléatoire de Bernoulli d'argument $u_m \in [0, 1]$. En d'autres termes, on conserve le triangle avec une probabilité u_m et on le défausse avec une probabilité $1 - u_m$.

Comme nous avons posé une contrainte sur l'espérance de A_m , cf. formule 5.3, et que $\mathbb{E}[A_m] = \eta_0 u_m$, il vient :

$$u_m = \frac{\mathcal{A}_m}{\eta_0}. \quad (5.7)$$

Comme on a supposé que $\mathcal{A}_m < \eta_0$, on a bien le paramètre $u_m < 1$. Ainsi, en choisissant u_m proportionnel à l'aire du triangle \mathcal{A}_m , nous sommes en mesure de répartir de manière uniforme nos échantillon sur toute la surface. Les petits triangles, c'est-à-dire ceux correspondant à une zone dense en triangles, ont très peu de chance d'être sélectionnés tandis que les triangles dont l'aire est proche de l'aire moyenne cible η_0 ont une forte chance d'être choisis.

Par ailleurs, grâce à cette formulation on peut à présent définir l'échantillonnage \mathcal{L}^* comme le sous-ensemble de \mathcal{L} tel que A_m est différente de 0, c'est-à-dire :

$$\mathcal{L}^* = \{Y_m \in \mathcal{L}, U_m < u_m\}. \quad (5.8)$$

De plus, en écrivant $\eta_0 = \frac{|\mathcal{S}|}{N_0}$, on se propose de reparamétriser notre échantillonneur grâce au nombre N_0 représentant le nombre moyen d'échantillons produit par cet algorithme. En effet, si on note $\hat{N} = |\mathcal{L}^*|$, la variable aléatoire correspondant au nombre d'échantillons produits par ce procédé, on a :

$$\begin{aligned} \mathbb{E}[\hat{N}] &= \sum_{m \in [\mathbb{I}, M]} \underbrace{\mathbb{E}[\mathbb{1}_{[0,u_m]}(U_m)]}_{= u_m} \\ &= \sum_{m \in [\mathbb{I}, M]} \frac{\mathcal{A}_m}{\eta_0} \\ &= \underbrace{\frac{1}{\eta_0}}_{= \frac{N_0}{|\mathcal{S}|}} \times \underbrace{\sum_{m \in [\mathbb{I}, M]} \mathcal{A}_m}_{= |\mathcal{S}|} = N_0. \quad \square \end{aligned} \quad (5.9)$$

Ainsi, là où un échantillonneur uniforme sur la surface produit N_0 échantillons indépendants, notre approche propose d'en produire un nombre comparable en choisissant $\eta_0 = \frac{|\mathcal{S}|}{N_0}$.

5.4.8 Solution du problème

Nous avons à présent défini toutes les contraintes pour fournir une solution au problème du calcul d'intégrale que nous avons formulé plus tôt cf. §???. Nous allons à présent résumer cette solution.

Tout d'abord, nous avons supposé que tous les triangles de la scène avaient une aire inférieure à un seuil défini par l'utilisateur η_0 . Nous devions ainsi, dans un premier temps, fournir un certain

échantillonnage de la géométrie, noté \mathcal{L}^* . Pour cela, nous avons proposé de travailler au sein de chaque triangle t_m sur lesquels nous avons construit les paires de variables aléatoires indépendantes (Y_m, U_m) ; Y_m étant uniformément répartie à la surface du triangle t_m et U_m uniformément répartie sur $[0, 1]$, cf. §5.4.5. À partir de là, afin de construire l'échantillonnage, nous n'avons besoin que d'un paramètre défini par l'utilisateur, η_0 , correspondant à l'aire moyenne représentée par chaque échantillon, cf. §5.4.7. Avec la donnée de ce paramètre il ne nous est pas nécessaire de calculer l'aire totale de la géométrie. En revanche, si celle-ci est connue, on a alors la relation $|S| = N_0 \eta_0$ où N_0 représente le nombre d'échantillons moyen construit par notre approche. En fonction de l'aire A_m de chaque triangle, nous construisons l'échantillonnage \mathcal{L}^* comme l'ensemble des Y_m pour lesquels la variable U_m vérifie une condition proportionnelle à l'aire du triangle, cf. formule 5.8.

En posant ainsi :

$$\hat{I}_f(\mathcal{L}^*) = \sum_{y^* \in \mathcal{L}^*} f(y^*) \eta_0, \quad (5.10)$$

qui est rigoureusement égal à $\hat{I}_f(\mathcal{L})$ tel que nous l'avons défini précédemment, cf. formule 5.5, nous obtenons bien à la fois un échantillonnage et un estimateur permettant d'approcher le calcul de l'intégrale définie par notre problème.

En outre, comme $\hat{I}_f(\mathcal{L}) = \hat{I}_f(\mathcal{L}^*)$, notre estimateur hérite de toutes les propriétés que nous avons énoncées plus tôt, cf. §5.4.6. Cela nous permet ainsi de conclure que pour chaque tirage d'un échantillonnage, l'estimateur a, d'une part, de bonnes raisons d'être « proche » de l'intégrale. D'autre part, tant que, pour un triangle, on est en mesure de construire N couples de variables indépendantes, la moyenne de N estimateurs $\hat{I}_f(\mathcal{L}_n^*)$ converge en loi vers l'intégrale.

5.4.9 Échantillonnage multi-échelles

L'algorithme que nous venons de décrire permet ainsi de produire pour un paramètre donné η_0 , un échantillonnage \mathcal{L}^* , dont les échantillons représentent, en moyenne, une aire η_0 de la surface. Il est parfaitement possible de choisir un second paramètre, $\eta_1 > \eta_0$, et de relancer notre algorithme pour produire un second échantillonnage \mathcal{L}^1 qui serait alors moins raffiné que le premier et ainsi de suite pour K valeurs $(\eta_k)_{k \in [\![1, K]\!]}$.

Par ailleurs, en se donnant n'importe quelle famille de fonctions (f_1, \dots, f_K) , le problème visant à calculer $I_{f_1 + \dots + f_K}$ peut se réécrire de la façon suivante :

$$I_{f_1 + \dots + f_K} = I_{f_1} + \dots + I_{f_K}.$$

En se donnant alors K échantillonnages, $\mathcal{L}^1, \dots, \mathcal{L}^K$, on peut alors construire K estimateurs, $I_{f_1}(\mathcal{L}^1), \dots, I_{f_K}(\mathcal{L}^K)$. On peut également définir un estimateur de $I_{f_1 + \dots + f_K}$ par :

$$\hat{I}_{f_1 + \dots + f_K} = \sum_{k \in [\![1, K]\!]} \hat{I}_{f_k}(\mathcal{L}^k). \quad (5.11)$$

Cet estimateur est ainsi non biaisé. En effet :

$$\mathbb{E} \left[\hat{I}_{f_1 + \dots + f_K} \right] = \sum_{k \in [\![1, K]\!]} \underbrace{\mathbb{E} \left[\hat{I}_{f_k}(\mathcal{L}^k) \right]}_{= I_{f_k}} = I_{f_1 + \dots + f_K}.$$

On remarque alors qu'il n'est pas nécessaire que les échantillonnages \mathcal{L}^k soient indépendants les uns des autres pour que le caractère non biaisé de l'estimateur soit préservé. En se donnant ainsi, pour chaque triangle t_m , une variable aléatoire $\kappa_m \in [0, K]$ telle que,

$$\begin{aligned}\forall k \in [1, K], \quad \mathbb{P}(\kappa_m = k) &= \frac{\mathcal{A}_m}{\eta_k} \\ \mathbb{P}(\kappa_m = 0) &= 1 - \sum_{k \in [1, K]} \mathbb{P}(\kappa_m = k),\end{aligned}\tag{5.12}$$

on peut construire l'échantillonnage \mathcal{L}^k comme l'ensemble des Y_m pour lesquels la variable $\kappa_m = k$. L'ensemble \mathcal{L}^0 correspond ici à l'ensemble des triangles qui sont défaussés par notre algorithme.

De plus, pour construire les $\mathcal{L}^1, \dots, \mathcal{L}^K$, il est nécessaire que la définition de la loi de κ_m soit valide, c'est-à-dire :

$$\begin{aligned}\sum_{k \in [0, K]} \mathbb{P}(\kappa_m = k) &= 1 \\ \Rightarrow \quad \sum_{k \in [1, K]} \frac{\mathcal{A}_m}{\eta_k} &< 1 \\ \Rightarrow \quad \mathcal{A}_m &< \eta^*,\end{aligned}$$

où

$$\eta^* = \frac{1}{\sum_{k \in [1, K]} \frac{1}{\eta_k}}.\tag{5.13}$$

Ainsi, pour construire ce multi-échantillonnage, il est nécessaire que tous les triangles aient une aire inférieure à η^* .

De cette façon, nous pouvons construire K échantillonnages en une seule passe de notre algorithme, ce qui limite le traitement géométrique à réaliser. Enfin, nous avons présenté au chapitre précédent, *cf.* §4.2.4, un algorithme permettant de construire une variable κ_m définie avec les mêmes contraintes. De plus, au même paragraphe, nous développons un calcul nécessitant un échantillonnage multi-échelles tel que celui que nous venons de décrire. Nous détaillons plus tard dans ce chapitre, *cf.* §5.6, comment mettre en pratique cela pour le calcul de l'éclairage global.

5.4.10 Lois non-uniformes

Il serait possible de construire un échantillonnage selon une loi μ^S quelconque définie sur la surface de la géométrie. Cela nécessiterait en revanche de reformuler u_m en conséquence ainsi que de rendre η_0 dépendant du triangle de sorte à ce que $u_m < 1$. Bien que cela resterait compatible avec la suite de notre pipeline de subdivision, nous ne nous sommes pas focalisés sur l'analyse d'une autre densité de probabilité que la loi uniforme. Nous considérons cette problématique comme orthogonale, et laissons cela pour de futurs travaux.

5.5 Implémentation de l'échantillonneur

Au paragraphe précédent, nous avons montré qu'il était possible de fournir l'approximation d'une intégrale en travaillant indépendamment sur chaque triangle d'un maillage. L'unique contrainte étant de pouvoir construire une variable aléatoire par triangle. Afin de produire efficacement cet

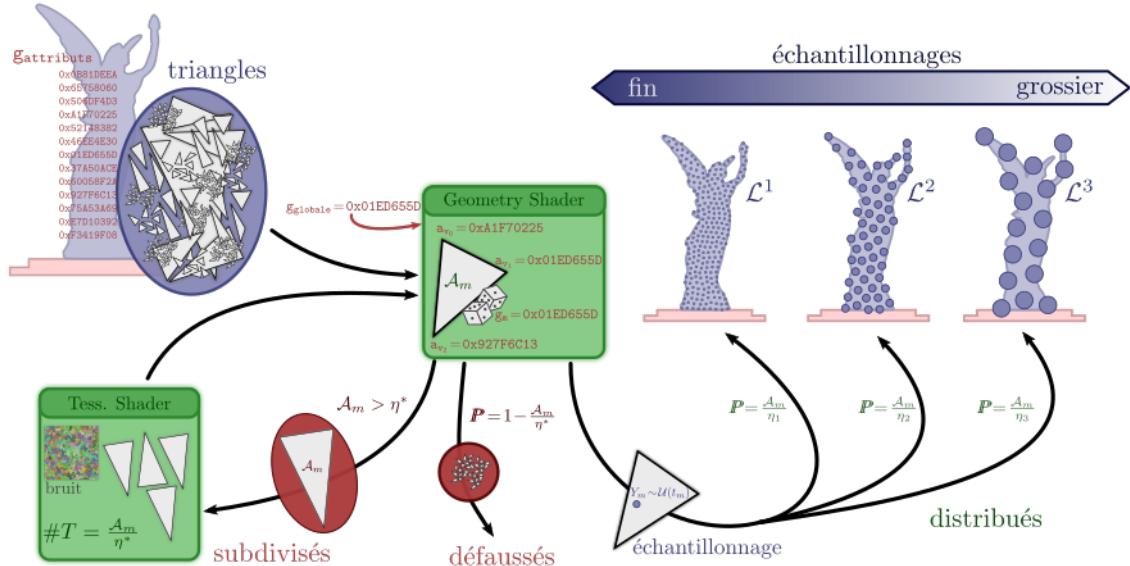


FIGURE 5.4 – Pipeline d'échantillonnage global. On cherche à construire K échantillonnages en une seule itération de notre algorithme. Pour construire la variable aléatoire nécessaire à l'échantillonnage, on ajoute un attribut par sommet $g_{\text{attributs}}$ et une variable aléatoire globale g_{globale} . Au *Geometry Shader*, on peut construire g_m , une variable aléatoire portée par le triangle t_m , en combinant g_{globale} et les 3 attributs des sommets. Ensuite, à l'aide de cette variable, si le triangle a une aire inférieure au seuil η^* , celui-ci est aléatoirement défausssé ou conservé avec une probabilité proportionnelle à son aire, ce qui aura tendance à faire défausser préférentiellement les petits triangles. Les triangles conservés sont distribués dans les différents échantillonnages \mathcal{L}^k de sorte à produire K densités d'échantillonnage distinctes. Les gros triangles, quant à eux, c'est-à-dire ceux dont l'aire est supérieure au seuil η^* , sont subdivisés de sorte à ce que les sous-triangles générés soient suffisamment petits, puis sont réinjectés dans le premier pipeline.

échantillonnage, nous décrivons à présent un algorithme fondé sur le pipeline de traitement géométrique du processeur graphique, à savoir le *Vertex Shader*, le *Tessellation Shader*, et le *Geometry Shader*. La figure 5.4 résume l'ensemble des points que nous développons dans ce paragraphe.

Dans un premier temps, nous décrivons une façon de produire une variable aléatoire au sein d'un triangle, cf. §5.5.1, puis, après avoir montré comment générer les échantillons à partir de ces variables, cf. §5.5.2, nous décrivons notre pipeline de subdivision adaptative, cf. §5.5.3.

5.5.1 Variable aléatoire par triangle

Sur un ordinateur, il est relativement difficile, si ce n'est impossible, de produire un nombre véritablement aléatoire. En pratique, afin d'obtenir une variable qui se rapproche d'un comportement aléatoire on utilise des séquences dites pseudo-aléatoires. Celles-ci sont initialisées par une graine et leur évaluation dépend directement de l'évaluation précédente. Ainsi, en choisissant convenablement la fonction f_{alea} , la suite de nombres $(g^n)_n$, générée par $g^{n+1} = f_{\text{alea}}(g^n)$, reproduit le comportement d'une variable aléatoire.

Cette façon de procéder n'est, en revanche, pas compatible avec le calcul parallèle. En effet, le fait de dépendre de la dernière valeur, g^n , pour calculer la suivante, g^{n+1} , implique de devoir synchroniser les appels à la fonction génératrice, f_{alea} , au sein des coeurs de calcul. Cela casse

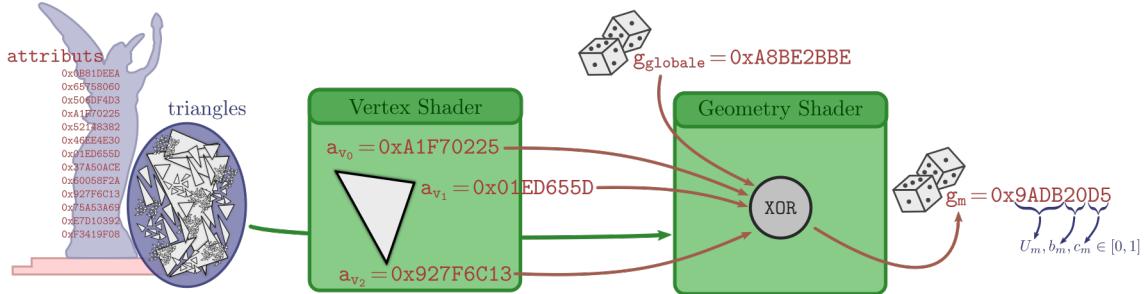


FIGURE 5.5 – Construction de variable aléatoire par triangle. Au niveau du triangle, pour générer plusieurs variables aléatoires, on commence par construire un attribut entier uniformément réparti sur $[0, 2^{32}]$, en combinant, via l'opérateur logique *xor*, les quatre nombres issus des sommets et la variable globale. Ensuite, on extrait le nombre de bits souhaité de cet entier pour construire les variables U_m , a_m et b_m dans l'intervalle $[0, 1]$.

le parallélisme du processeur graphique. Afin de produire des nombres aléatoires en parallèle, il semble donc nécessaire d'utiliser une fonction aléatoire propre à chaque unité de calcul. En pratique, il suffit ainsi de choisir une graine g_m^0 , différente sur chaque unité.

Dans notre cas, la granularité de nos calculs étant celle du triangle, on cherche à produire une graine g_m^0 pour chaque triangle. Cependant, le pipeline graphique n'offre pas de mécanisme simple pour identifier un triangle à partir d'un indice. Grâce à cela, on aurait pu pré-calculer une table de graines aléatoires uniformes et les associer à chaque triangle. Bien que ce ne soit pas nécessaire selon l'algorithme que nous avons décrit précédemment, cf. §5.4.8, cela nous permettrait d'avoir une répartition uniforme sur l'ensemble des triangles et ainsi prévenir des cas pathologiques particulièrement ennuyeux. En effet, si par exemple, dans un cas extrême, tous les triangles avaient la même graine et une même probabilité de recevoir un échantillon, alors il existerait uniquement deux échantillonnages possibles : l'un où tous les triangles sont choisis, et l'autre où aucun triangle n'est choisi. On se rend alors bien compte qu'aucun de ces deux scénarios n'est souhaitable. Cependant, on peut tout de même minimiser le risque de produire de tels échantillonnages dégénérés en essayant de minimiser la corrélation entre les graines des triangles. Encore une fois, on rappelle que nous cherchons à produire un échantillonnage « acceptable » avec le budget de temps dont on dispose. Quel que soit le résultat obtenu, on a montré par ailleurs, cf. §5.4.6, qu'il est toujours possible de ré-échantillonner la géométrie de sorte à faire converger notre algorithme si cela est souhaité.

Attribut aléatoire par sommet Comme le pipeline graphique nous offre la possibilité de travailler à cette échelle, nous proposons ici de construire nos graines à partir de chaque sommet du maillage, cf. figure 5.5. Pour cela, nous ajoutons un tableau d'attributs d'entiers, $(a_i)_i$, de 32 bits chacun, propre à chaque sommet, v_i , du maillage. Nous construisons ce tableau une seule fois au moment où le maillage est chargé en choisissant des nombres aléatoires. Dans le pipeline géométrique nous récupérons cette valeur que nous faisons transiter du *Vertex Shader* au *Geometry Shader*. À cette étape nous travaillons à l'échelle d'un triangle t_m et c'est là que nous construisons la graine g_m qui lui est propre. Nous y combinons ainsi les 3 attributs pré-calculés, a_{v_0} , a_{v_1} , a_{v_2} , de ses 3 sommets v_0 , v_1 , v_2 .

$\mathbb{P}(a = \cdot \wedge b = \cdot)$	a	b	$a \text{ XOR } b$	$\mathbb{P}(a \text{ XOR } b = \cdot)$
$1/4$	0	0	0	$1/2$
$1/4$	1	1	0	
$1/4$	0	1	1	$1/2$
$1/4$	1	0	1	

TABLE 5.1 – Table de vérité du *ou exclusif*. Pour deux variables booléennes a et b , prenant l’ensemble des combinaisons possibles, le nombre d’occurrences ou le résultat de $a \text{ XOR } b$ vaut 0 est égale à celui où il vaut 1. Ainsi, si a et b sont uniformément distribués sur $\{0, 1\}$, alors $a \text{ XOR } b$ l’est également.

Cette construction expose un certain nombre de propriétés avantageuses. Tout d’abord on note que la graine g_m ainsi construite est indépendante de la position du modèle dans l’espace. Ainsi quelles que soient les transformations rigides subies, l’échantillonnage résultant reste cohérent. Par ailleurs, comme nous travaillons indépendamment d’un triangle à l’autre, si une forte modification est faite sur une partie du maillage, par exemple une déformation non rigide, même si l’aire des triangles est modifiée, les graines, elles, ne dépendent que de la topologie du maillage. Il vient que, d’une part, seuls les échantillons présents dans la région concernée sont affectés, laissant tout autre partie de l’échantillonnage inchangée. D’autre part, au vu de notre stratégie de sélection de triangle, le critère $U_m < \frac{\mathcal{A}_m}{\eta}$, basé sur un seuil proportionnel à l’aire du triangle, cf. formule 5.7, implique qu’une petite variation sur la forme du triangle n’aura que très peu de chances de modifier la réussite ou l’échec du test. Tout ceci permet finalement d’obtenir un échantillonnage globalement stable par rapport à la plupart des transformations habituellement réalisées sur un maillage.

Combinaison d’attributs On considère un nombre entier codé sur N bits. Ses bits sont issus de N tirages aléatoires indépendants dont les valeurs, 0 ou 1, sont choisies de manière équiprobable. On montre alors qu’un tel processus permet de construire un nombre entier uniformément choisi dans $[0, 2^N]$. En effet, pour tout nombre entier, $i \in [0, 2^N]$, il existe une unique représentation binaire (b_1, \dots, b_N) tel que $i = \sum_k b_k 2^k$, et $b_k \in \{0, 1\}$. Ainsi, la probabilité d’obtenir i par un tel tirage est égale à la probabilité d’obtenir la séquence de sa représentation binaire. Or, comme chaque séquence est équiprobable avec ce procédé, et qu’il y a exactement 2^N séquences possibles, la probabilité d’obtenir l’entier $i \in [0, 2^N]$ est bien $\frac{1}{2^N}$.

Par ailleurs, afin de construire nos graines, g_m , par triangle, nous utilisons l’opérateur de *ou exclusif*, XOR, qui permet de préserver le caractère équiprobable indépendant de chaque bit du nombre calculé. En effet, on note tout d’abord que la valeur de chaque bit du nombre construit est calculable indépendamment des autres bits. On peut donc se concentrer sur le caractère équiprobable d’une seule variable booléenne. De plus grâce à une table de vérité sur les valeurs possibles de cette opération, cf. table 5.1, on observe que la probabilité d’obtenir un 1 est égale à celle d’obtenir un 0. Ainsi, si deux nombres i_a et i_b sont indépendants uniformément choisis sur $[0, 2^N]$ alors la variable résultant de l’opération $i_a \text{ XOR } i_b$ l’est également. De même, si l’une des deux variables est déterministe, le résultat reste une variable aléatoire uniforme. En outre, le résultat de 3 variables uniformes, et par extension de n’importe quel nombre, mélangées via l’opérateur XOR forme, lui aussi, une variable aléatoire uniforme.

Nous exploitons cette dernière propriété pour ajouter une graine globale, g_{globale} , accessible par le *Geometry Shader* et avec laquelle on construit à chaque trame la véritable graine, g_m , du triangle

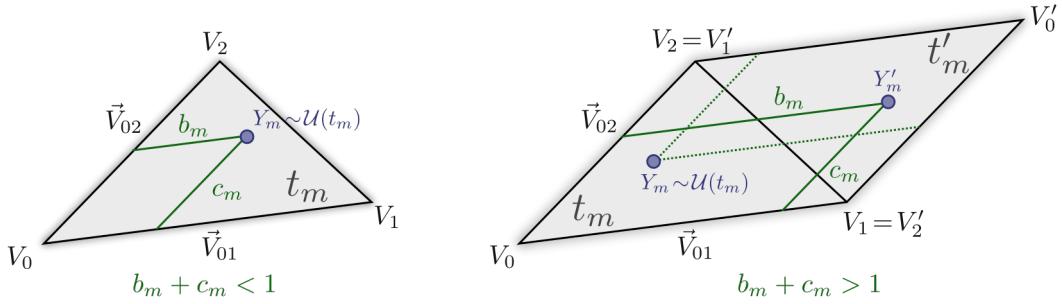


FIGURE 5.6 – Échantillonnage uniforme du triangle. À partir de deux variables b_m et c_m uniformément réparties sur $[0, 1]$, deux cas de figure se présentent. Si, $b_m + c_m < 1$ alors le point $Y_m = b_m \vec{V}_{01} + c_m \vec{V}_{02}$ parcourt le triangle t_m uniformément. En revanche, si $b_m + c_m > 1$, alors la variable Y'_m parcourt uniformément le triangle t'_m qui est le symétrique de t_m comme sur le dessin, c'est-à-dire par symétrie centrale autour du milieu de l'arête commune. Pour produire un échantillon réparti uniformément sur le triangle t_m , quelque soit b_m et c_m , il suffit de prendre, dans ce second, la position symétrique de Y'_m dans t_m .

t_m . Celle-ci est ainsi le résultat de la combinaison des 3 graines des sommets et de la graine globale

$$g_m = g_{\text{globale}} \text{ xor } a_{v_0} \text{ xor } a_{v_1} \text{ xor } a_{v_2}. \quad (5.14)$$

Enfin, en s'autorisant à moduler de manière aléatoire g_{globale} , il est possible de produire un tout nouvel échantillonnage sur la géométrie sans avoir besoin de modifier les valeurs des graines par sommet. On note alors que, d'une trame à l'autre, les graines résultantes de tous les triangles sont extrêmement corrélées, mais, qu'au niveau du triangle, la graine g_m correspond à une variable aléatoire uniforme. Cela nous permet ainsi de remplir la condition développée au paragraphe précédent, cf. §5.4.6, justifiant le fait que, en moyennant plusieurs évaluations de notre estimateur \hat{I}_f , le résultat converge vers l'intégrale.

Une fois g_m donnée, il est ais  de construire une variable al atoire sur $[0, 1]$. Dans notre cas, nous avons besoin d'au moins trois variables al atoires sur $[0, 1]$, ´a savoir deux pour la position de l' chantillon sur le triangle b_m, c_m , cf. §5.5.2, et une, U_m , pour d terminer si le triangle est conserv  ou non cf. §5.4.7. Dans nos applications, nous extraîrons 8 bits pour b_m et c_m et 16 bits pour U_m . Si par ailleurs, on souhaite obtenir davantage de pr cision, il est toujours possible de fournir une seconde variable globale $\tilde{g}_{\text{globale}}$ ind pendante de g_{globale} . Le nombre \tilde{g}_m , produit de mani re analogue ´a g_m , cf. formule 5.14, est lui aussi ind pendant de g_m . Ses bits peuvent ainsi  tre utilis s pour accroître la pr cision des variables b_m et c_m par exemple.

5.5.2 Algorithme d' chantillonnage global

Dans ce paragraphe, nous continuons de faire l'hypoth se que tous les triangles ont une aire inf rieure ´a η . Dans le but de d crire la strat gie de r - chantillonnage, nous nous pla ons dans le *Geometry Shader*, cf. figure 5.4, lors du traitement du triangle t_m , d'aire \mathcal{A}_m . En utilisant l'ap roche d crite au paragraphe précédent, nous supposons que nous sommes en possession de trois variables al atoires uniformes sur $[0, 1]$ que l'on note a_m, b_m et c_m .

Dans un premier temps nous testons la condition de d cimation d crite plus haut, cf. formule 5.7 :

$$a_m \eta < \mathcal{A}_m.$$

Si cette condition n'est pas vérifiée, alors le traitement s'arrête là pour ce triangle et aucune primitive n'est émise. Dans le cas contraire, on utilise les deux variables, b^m et c^m , pour construire la position de l'échantillon Y_m .

$$Y_m = \begin{cases} V_0 + b_m \vec{V}_{01} + c_m \vec{V}_{02} & \text{si } b^m + c^m < 1 \\ V_0 + (1 - b_m) \vec{V}_{02} + (1 - c_m) \vec{V}_{01} & \text{sinon} \end{cases},$$

où V_0 , \vec{V}_{01} et \vec{V}_{02} sont définis comme sur la figure 5.6. Cette dernière formule permet de construire le point Y_m uniformément réparti sur la surface du triangle. À partir de là, notre mécanisme de rééchantillonnage est terminé : nous venons de produire une position sur le maillage pour laquelle un cœur de calcul est actif, en l'occurrence le *Geometry Shader*. Nous verrons dans la partie suivante, cf. §5.6, comment mettre à profit cette situation afin d'engendrer d'autre calculs à l'aide de l'émission de primitives, de l'unité de rastérisation et de l'étape de *Fragment Shader*.

On note que l'on pourrait également enregistrer cette position dans un tableau afin de stocker l'échantillonnage et d'éviter d'avoir à le recalculer. Cependant, selon nos expérimentations, cette étape d'échantillonnage est rarement le facteur limitant, tandis que, stocker l'intégralité des échantillons peut consommer beaucoup de mémoire. Ainsi, le fait d'utiliser le pipeline de rendu pour reconstruire à la volée cet échantillonnage permet à notre approche de passer relativement bien à l'échelle lorsque le nombre de triangles augmente. Enfin, puisque notre approche se fonde sur le pipeline de rendu classique, si des données massives, telles que des textures, sont nécessaires pour décrire la géométrie ou les matériaux, celles-ci peuvent être accédées de manière cohérente sans nécessiter de préparation supplémentaire.

5.5.3 Pipeline de subdivision

Dans les paragraphes qui précèdent, nous avons à chaque fois supposé que notre géométrie était constituée de triangles dont l'aire est forcément inférieure à un certain seuil η . Cependant, il n'y a, *a priori*, aucune raison que ce soit le cas. De plus, il s'agit d'une contrainte relativement forte et, qui plus est, dépendante du choix de l'utilisateur. En pratique, nous avons même constaté que cette condition n'est jamais vérifiée ; d'autant que rien ne contraint l'utilisateur de choisir un η aussi petit qu'il le souhaite. En revanche, nous constatons que, dans les scénarios que nous avons testés, il n'est pas nécessaire que η soit trop petit, et, que dans cette situation, le nombre de triangles violant cette condition demeure relativement faible, cf. §5.7.

Notre implémentation étant entièrement réalisée au sein du pipeline graphique, nous avons dans un premier temps analysé les limitations des différentes étapes de traitement géométrique. L'algorithme que nous décrivons nécessite deux types de traitements. Un premier type, que nous avons décrit au paragraphe précédent, cf. §5.5.2, qui consiste à décimer une partie des triangles. Nous avons ainsi montré que cette première étape peut être efficacement réalisée au sein d'un *Geometry Shader* pour distribuer efficacement le traitement sur les cœurs de calcul.

Dans ce paragraphe, nous nous intéressons à un deuxième type de traitement. Celui-ci consiste à raffiner la géométrie, c'est-à-dire subdiviser la géométrie dans les zones constituées de larges triangles. Cela permet ainsi de réduire la taille des triangles pour finalement se rapporter à la situation du paragraphe précédent. Pour ce faire, nous proposons d'utiliser les capacités de tessellation accélérées matériellement, à savoir les étapes de *Tessellation Shader*, cf. §2.2.2. Cette fonctionnalité travaille en deux temps ; tout d'abord au niveau des patchs, dans notre cas les triangles, en

permettant de définir leur niveau de tessellation, puis, dans un second temps, au niveau des sommets construits par la subdivision. Nous exploitons ces deux niveaux afin d'à la fois subdiviser la géométrie de sorte à respecter la condition sur la taille des triangles, et également d'étendre la construction de la graine aléatoire sur les nouveaux triangles.

Raffinement par subdivision La première étape de l'unité de tessellation correspond au *Tesselation Control Shader*. Au sein de celle-ci, on connaît les trois sommets du triangle et donc son aire. On peut alors déterminer si elle est supérieure à η et auquel cas subdiviser le triangle en conséquence. En effet, c'est toujours au cours de cette étape que l'on peut paramétriser l'unité de tessellation et dire en combien de sous-triangles on souhaite subdiviser le patch. Grâce à l'aire du triangle, il est alors aisément de déterminer le nombre minimal de subdivisions, $\#T_m$, nécessaires pour que l'aire de chaque sous-triangle ainsi construit soit inférieure à η , c'est-à-dire :

$$\#T_m = \left\lceil \frac{\mathcal{A}_m}{\eta} \right\rceil, \quad (5.15)$$

où $\lceil \cdot \rceil$ représente la fonction partie entière supérieure.

Par ailleurs, la stratégie de décimation que nous avons décrite plus tôt, cf. §5.5.2, est optimale lorsque l'aire des triangles est égale à η car aucun triangle n'est alors défaussé. Il est bien entendu important de minimiser le nombre de défausses car, même lorsque qu'ils sont défaussés, le fait de déterminer s'ils doivent ou non porter un échantillon coûte du travail. Grâce à la condition ci-dessus, on optimise ainsi la charge de travail du processeur graphique.

Cependant, le schéma de subdivision étant codé matériellement [Kessenich et coll., 2016] on ne peut pas définir exactement le nombre de sous-triangles que l'on souhaite obtenir. Au lieu de cela, on peut contrôler le nombre de segments $\#S_m$ en lesquels on souhaite subdiviser chaque arête du triangle. Ainsi, en inversant la formule liant le nombre de subdivisions au nombre de triangles construits, on obtient :

$$\#S_m = \min \left(2 \left\lceil \sqrt{\frac{\#T_m}{6}} \right\rceil, 1 + \left\lceil \sqrt{1 + 2 \frac{\#T_m - 1}{3}} - 1 \right\rceil \right). \quad (5.16)$$

Extension de l'aléatoire sur la subdivision Une fois que le triangle de grande taille a été transformé en plusieurs petits, il reste encore à interpoler les attributs. Pour la plupart des attributs, par exemple la position, la normale ou encore les coordonnées de textures, la seconde étape de tessellation offre un mécanisme relativement simple pour interpoler les attributs à l'aide des coordonnées barycentriques des nouveaux sommets. Cependant, ce mécanisme d'interpolation ne convient pas aux attributs entiers, et notamment à notre attribut graine, g_m , car une interpolation linéaire naïve ne préserve pas, par exemple leur uniforme distribution. De plus, il n'est pas possible d'identifier les sommets générés autrement que par leurs coordonnées barycentriques ; en particulier, ils ne sont pas numérotés par un indice. Cela nous empêche donc d'utiliser un tableau de valeurs pré-calculées pour associer la valeur correspondant à leur indice. Pour ces raisons, nous proposons d'utiliser une texture 2D de bruit dont chaque pixels contient un nombre entier aléatoire pré-calculé, cf. figure 5.7. Ainsi, pour chaque sommet subdivisé V_m^s du triangle t_m , on se sert de deux de ses coordonnées barycentriques, $(\alpha_m^s, \beta_m^s, \gamma_m^s)$, afin de calculer la position d'un pixel dans la texture de bruit. On récupère ainsi une graine $g_{\text{sub}}(\alpha_m^s, \beta_m^s)$ propre à ce sommet.

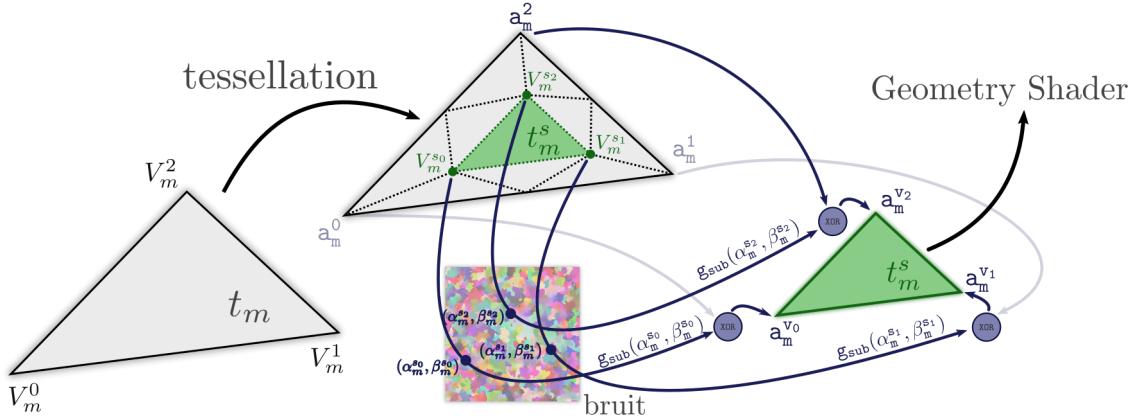


FIGURE 5.7 – Prolongement de la variable aléatoire sur la subdivision. Le triangle t_m^s est issu de la subdivision du triangle initial t_m . Pour construire, sur ce nouveau triangle, les attributs $a_m^{s_i}$ de ses sommets, on utilise leurs coordonnées barycentriques $(\alpha_m^{s_i}, \beta_m^{s_i})$ pour récupérer dans une texture de bruit les valeurs g_{sub} correspondantes. Celle-ci est alors combinée, via l'opérateur logique *xor* avec les trois attributs a_m^i du triangle initial. De cette façon, la variable g_m^s , construite au *Geometry Shader* qui suit, mélangeant alors ces attributs émergents via l'opérateur *xor*, se retrouve avoir le minimum de corrélation avec celle des sous-triangles issus de la subdivision, et également avec celle des triangles extérieurs à cette subdivision.

Nous construisons alors pour le triangle subdivisé t_m^s , l'interpolation de sa graine aléatoire, g_m^s , en mélangeant la graine du triangle t_m et celles des trois sommets subdivisés s_0, s_1, s_2 , c'est-à-dire :

$$g_m^s = g_m \text{ xor } g_{sub}(\alpha_m^{s_0}, \beta_m^{s_0}) \text{ xor } g_{sub}(\alpha_m^{s_1}, \beta_m^{s_1}) \text{ xor } g_{sub}(\alpha_m^{s_2}, \beta_m^{s_2}) \quad (5.17)$$

Le fait d'utiliser à la fois la graine, g_m , issue du patch initial ainsi que celles de la texture de bruit, fait que d'un patch à l'autre, de même qu'au sein du patch lui-même, les graines générées ne seront pas corrélées. De cette façon, le triangle qui est construit se comporte indifféremment, dans l'étape de *Geometry Shader* qui suit, selon si le patch a été subdivisé ou non.

Filtrage des triangles en deux temps Avec l'ajout du pipeline de subdivision, notre algorithme d'échantillonnage pourrait être complet en l'état. Cependant, pour deux raisons principales, il n'est pas préférable d'activer la stratégie de subdivision sur l'intégralité du maillage. La première raison est que pour pouvoir utiliser l'unité de tessellation, il est nécessaire que le maillage soit décrit sous la forme de liste de triangles. Bien que cela soit une contrainte relativement faible, il est tout de même fréquent, dans le but de compresser les données, de trouver de la géométrie décrite par des triangles en rayures (ou *triangle strip* en anglais), et ce d'autant que la géométrie est massive. Dans un tel scénario, notre algorithme ne fonctionnerait pas sans devoir transformer la géométrie. De même, l'unité de tessellation peut très bien être déjà employée dans un tout autre but, par exemple pour le calcul de niveaux de détail d'une carte de hauteur. Pour ce dernier cas, il serait possible de combiner les deux algorithmes afin qu'ils s'exécutent en même temps, mais cela rendrait l'intégration de notre approche autrement plus complexe. Le second obstacle à l'activation systématique de la subdivision est que l'unité de tessellation induit un surcoût de traitement d'autant plus grand que la géométrie est massive. Ceci est problématique car, en pratique, seul un nombre modéré de triangles nécessite réellement d'être raffiné.

Pour ces deux raisons, nous proposons de traiter l'échantillonnage des triangles massifs en deux temps. Dans un premier temps, nous désactivons l'unité de tessellation, et donc le pipeline de sub-

division, et faisons tourner notre algorithme sur la totalité de la géométrie. Au sein du *Geometry Shader*, si le triangle possède une aire inférieure à η alors nous continuons notre algorithme tel que décrit au paragraphe précédent, cf. §5.5.2. Dans le cas contraire, nous enregistrons les trois sommets du triangle dans une liste et ne générerons aucun échantillon pour ce triangle. L'insertion dans cette liste se fait à l'aide d'un compteur atomique global qui est incrémenté lorsque qu'un triangle massif a besoin d'être stocké. Ainsi nous réservons dans un tableau, implémenté en *OpenGL* à l'aide d'un *Shader Storage Buffer*, la mémoire pour inscrire de manière contiguë les attributs des trois sommets.

Dans une seconde étape, nous interprétons le tableau comme une liste de triangles et nous l'utilisons comme géométrie d'entrée de notre algorithme. À l'aide de la fonctionnalité de rendu indirect, *glDrawArrayIndirect*, aucune synchronisation entre le processeur graphique et le processeur n'est requise. C'est alors seulement pour ce dernier rendu de la géométrie que le pipeline de subdivision est activé. De cette façon, on est certain que les triangles sont, d'une part, convenablement préparés sous forme de liste et donc utilisables par le *Tessellation Shader*, et, d'autre part, que la partie de la géométrie qui est subdivisée se résume aux seuls triangles qui ont réellement besoin d'être subdivisés.

5.6 Vers l'éclairage global diffus temps réel

Au cours des deux paragraphes précédents, nous avons construit un algorithme permettant de produire un échantillonnage à partir de n'importe quelle surface dans le but d'approcher en temps réel le calcul d'une intégrale. Cela n'est pas sans rappeler le besoin que nous avons évoqué au chapitre précédent pour construire nos *coupes antérogrades de lumières*. Ainsi, dans ce paragraphe nous utilisons l'échantillonnage produit afin d'interpréter chaque échantillon comme une source de lumière *VPL*, et, nous utilisons les variables aléatoires construites par triangle pour calculer les fonctions de support associées. Nous montrons alors, après avoir décrit notre pipeline complet, cf. §5.6.1, qu'en utilisant une technique de rendu d'éclairage différé, cf. §5.6.3, nous sommes en mesure d'approcher en temps réel les résultats que nous avons démontrés au chapitre précédent.

5.6.1 Pipeline de rendu complet

La figure 5.8 illustre le pipeline que nous proposons afin de résoudre l'éclairage global. Dans notre algorithme d'échantillonnage, nous n'utilisons que les étapes de traitement géométrique du pipeline de *Shaders*. Cela nous permet, à présent, d'exploiter la dernière étape, à savoir le *Fragment Shader*, pour réaliser directement les calculs d'éclairage. Cette approche s'avère ainsi particulièrement bien adaptée pour calculer la contribution des *VPL* à l'aide du rendu différé, cf. §2.2.4.

Nous commençons donc par construire en espace écran le tampon géométrique (ou *G-Buffer* en anglais). Cette première partie est en général très rapide à calculer, et est très fréquemment réalisée dans un moteur de rendu. Dans un second temps, nous activons notre pipeline d'échantillonnage dans le but de produire, pour chaque triangle, au plus un *VPL* au *Geometry Shader*, tel que nous venons de le décrire. Pour construire chaque *VPL*, on a utilisé trois variables aléatoires, cf. §5.5.2. Nous avons également développé une extension permettant de produire un multi-échantillonnage à partir de ces trois seuls nombres, cf. §5.4.9, c'est-à-dire qu'à chaque *VPL* on associe également

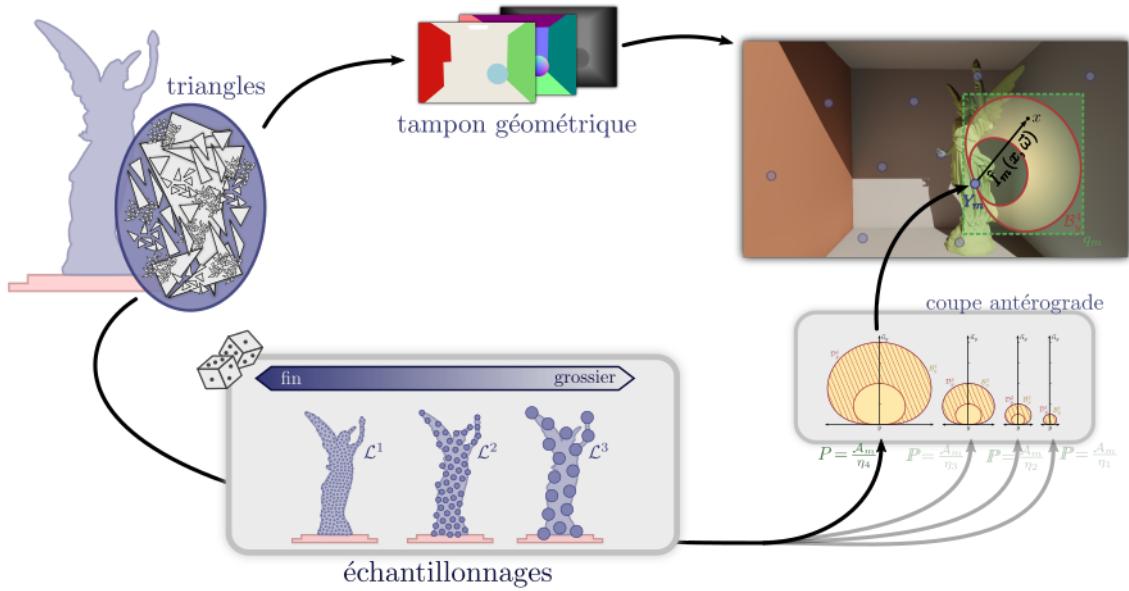


FIGURE 5.8 – Pipeline d’approximation de l’éclairage global. Une fois que notre algorithme d’échantillonage a produit un échantillon à partir d’un triangle, nous branchons le mécanisme de *coupes antérogrades* juste après. Le choix de la fonction de support est alors fait grâce à une variable aléatoire dans le *Geometry Shader*, et le triangle est transformé en rectangle englobant cette fonction en espace image. Au *Fragment Shader* qui suit, on accumule sur chaque pixel affecté par le *VPL* sa contribution lumineuse.

un niveau. Ce niveau nous permet ici de construire la *coupe antérograde* en déterminant la fonction de support du *VPL*, que nous avons introduite au chapitre précédent, cf. chapitre 4, et que nous allons maintenant appliquer, cf. §5.6.3. Par ailleurs, comme cette fonction de support est non nulle uniquement sur une zone bornée dans l’espace, il est possible de calculer sa zone d’influence en espace image, et plus précisément son rectangle englobant qui correspondent aux pixels possiblement affectés par la lumière. On utilise alors le *Geometry Shader* pour transformer le triangle portant le *VPL* en une autre primitive correspondant à ce rectangle englobant. Pour chaque pixel touché par ce rectangle, on calcule enfin la contribution lumineuse du *VPL* multipliée par la fonction de support de la coupe antérograde. Le résultat est ensuite accumulé sur le pixel, en activant l’opération de mélange additif (*color blend* en *OpenGL*) sur la carte de couleurs. En accumulant la contribution de tous les *VPL* sur chacun des pixels, on obtient ainsi l’approximation d’un rebond lumineux.

5.6.2 Lien avec les coupes antérogrades

Comme nous l’avons décrit au début de ce chapitre, l’échantillonnage que nous avons construit nous permet de calculer l’approximation \hat{I}_f d’une certaine intégrale I_f sur la surface du maillage S , cf. formule 5.2. Afin de nous rapporter au calcul par coupe antérograde défini au chapitre précédent, cf. formule 4.30, il nous faut dans un premier temps voir I_f comme la réponse lumineuse d’un pixel soumis à l’éclairage d’un rebond de la lumière. On pose alors :

$$f : (y, x, \vec{\omega}) \mapsto \left(\int_{\nu \in \mathcal{V}} \mathcal{W}(\nu, y, x) d\nu \right) L(y \rightarrow x) h(y, x, \vec{\omega}). \quad (5.18)$$

En définissant f ainsi, I_f s'identifie à \mathcal{T}_L , et donc, \hat{I}_f correspond à une « bonne » approximation de l'opérateur de transport. Pour chaque VPL , Y_m , on applique alors la formule que nous avons développée plus haut, *cf.* formule 5.4, pour définir sa contribution lumineuse :

$$\hat{I}_m(x, \vec{\omega}) = \hat{I}_f(Y_m, x, \vec{\omega}) = \hat{f}_m(x, \vec{\omega}) \mathcal{A}_m. \quad (5.19)$$

où \hat{f}_m est donnée à partir des fonctions de supports, \mathcal{W} , définies précédemment, *cf.* §4.3.2 et *cf.* §4.3.4, c'est-à-dire :

$$\hat{f}_m : (x, \vec{\omega}) \mapsto \frac{w(d_m, Y_m, x)}{\mu^I(d_m)} L(Y_m \rightarrow x) h(Y_m, x, \vec{\omega}). \quad (5.20)$$

Si l'on ne conserve que les deux derniers termes de la formule ci-dessus, on retombe sur l'expression classique de la contribution d'un VPL uniformément réparti sur la surface. $L(Y_m \rightarrow x)$ correspondant alors à la luminance du VPL Y_m , en direction du pixel x et $h(Y_m, x, \vec{\omega})$ à la fonction de transfert point à point telle que nous l'avons introduite, *cf.* formule 2.1.4.

Dans cette formule, en revanche, nous avons le terme inspiré de la fonction de support, $\frac{w(d_m, Y_m, x)}{\mu^I(d_m)}$, dépendant d'une variable aléatoire d_m , de densité de probabilité μ^I , indépendante de Y_m . Notre mécanisme d'échantillonnage permet de produire de telles variables. Cependant, en fonction du choix de la solution discrète ou continue, il nous faut adapter l'interprétation de ces termes.

Distribution des fonctions de support Dans le chapitre précédent nous avons proposé deux familles de solutions selon si nous étions dans la formulation discrète, *cf.* formule 4.36, ou continue, *cf.* formule 4.41. Pour produire l'estimateur de ces solutions, il est nécessaire de construire une variable aléatoire $\kappa_m \in \{1, \dots, K\}$ pour la version discrète (resp. $\nu_m \in [0, K+1]$ pour la version continue) et dont la loi de probabilité $\tilde{\mu}_{(\text{CAL})}^{\mathcal{L}}(\kappa_m)$ (resp. $\tilde{\mu}^{\mathcal{L}}(\nu_m)$) est donnée par la formule 4.33 (resp. formule 4.42). Pour produire numériquement ces variables, nous pouvons procéder de la même façon que pour les a_m , b_m et c_m au paragraphe précédent, *cf.* §5.5.1. On se propose alors de se placer au *Geometry Shader* pour construire d_m , s'identifiant à κ_m ou ν_m , selon la situation, et suivant la densité de probabilité correspondante. Dans cette situation, on construit un échantillonnage constitué d'un unique niveau \mathcal{L}^* paramétré par η_0 , *cf.* §5.4.8. Il faut alors interpréter la distribution $\mu^I(d_m)$ de la formule 5.20 comme, la probabilité de générer le VPL , Y_m , suivie de la probabilité de choisir le niveau d_m (c'est-à-dire κ_m ou ν_m selon la situation). Comme les deux variables définissant ces événements sont indépendantes, on peut écrire :

$$\mu^I(\kappa_m) = \mathbb{P}(Y_m \in \mathcal{L}^*) \tilde{\mu}_{(\text{CAL})}^{\mathcal{L}}(\kappa_m) = \frac{\mathcal{A}_m}{\eta^*} \tilde{\mu}_{(\text{CAL})}^{\mathcal{L}}(\kappa_m) \quad (5.21)$$

$$\mu^I(\nu_m) = \mathbb{P}(Y_m \in \mathcal{L}^*) \tilde{\mu}^{\mathcal{L}}(\nu_m) = \frac{\mathcal{A}_m}{\eta^*} \tilde{\mu}^{\mathcal{L}}(\nu_m). \quad (5.22)$$

On note alors, que dans le calcul de la contribution du VPL , *cf.* formule 5.19, l'aire du triangle \mathcal{A}_m portant ce point se simplifie avec celui porté par la densité de probabilité μ^I . On peut finalement interpréter cela comme si l'échantillonnage \mathcal{L}^* était alors constitué de VPL ayant tous une aire représentative égale à η^* .

Distribution par multi-échantillonnage Alternativement à la définition que nous venons de donner pour μ^I , on se propose ici d'exploiter le multi-échantillonnage que nous avons construit plus

tôt, *cf.* §5.4.9. En revanche, nous nous limitons ici à la solution discrète des coupes antérogrades. De cette façon, on propose de définir les aires significatives $\{\eta_1, \dots, \eta_K\}$ évoluant de la même façon que les aires $(a_k)_k$ au chapitre précédent, *cf.* formule 4.28, c'est-à-dire :

$$\forall k \in [1, K], \quad \eta_k = a_k = \eta_1 q_a^{k-1}.$$

En produisant alors le multi-échantillonnage, on peut alors directement interpréter la densité $\mu^I(\kappa_m)$ comme la probabilité d'un triangle t_m de produire un échantillon de niveau κ_m . Ainsi, d'après la formule 5.12, on peut ici définir μ^I par :

$$\mu^I(\kappa_m) = \frac{\mathcal{A}_m}{\eta_{\kappa_m}}. \quad (5.23)$$

Pour construire ce multi-échantillonnage, il nous faut ainsi subdiviser les triangles de sorte à ce que leurs aires soient inférieures à η_0 , *cf.* formule 5.13. La loi de la variable k_m est alors induite par notre algorithme de multi-échantillonnage. Cette construction possède plusieurs propriétés intéressantes. Tout d'abord, en espérance, l'estimateur de coupe antérograde est identique à celui défini plus tôt, *cf.* formule 5.21. De plus, comme nous l'avons mentionné à la construction du multi-échantillonnage, ce mécanisme permet de conserver une certaine stabilité dans le choix du niveau lorsque l'aire des triangles varie légèrement. Cette dernière propriété est particulièrement importante lorsque la géométrie est modifiée (par une animation ou une homothétie, par exemple). En effet, si un triangle d'aire \mathcal{A}_m porte un *VPL* d'un niveau k , et, si sa taille est légèrement modifiée, alors, la plupart du temps rien ne change. Si, en revanche, le choix du niveau devait changer, alors celui-ci serait nécessairement proche du niveau précédent. En outre, cela nous permet de gérer les scénarios animés en limitant le scintillement dans le rendu final.

5.6.3 Aplatissement en espace image

Quel que soit le choix de la fonction de support déterminée au-dessus, nous avons formulé les w de sorte à ce qu'ils soient contenus dans un domaine borné, \mathcal{B}_k , introduit au chapitre précédent, *cf.* figure 4.3. Ces domaines peuvent alors être englobés par une sphère, c'est-à-dire une géométrie plus simple, qui elle-même peut être facilement encadrée par un quadrilatère en espace image, *cf.* figure 5.9. Cette dernière primitive est particulièrement intéressante car l'étape de *Geometry Shader* permet de transformer efficacement un triangle en carré. De cette façon, on exploite directement notre pipeline d'échantillonnage. En effet, une fois que le triangle a été sélectionné pour produire un *VPL* au *Geometry Shader*, celui-ci envoie alors à l'unité de rastérisation une primitive transformée, en l'occurrence un carré, sans briser le mécanisme de pipeline. Il n'est donc pas nécessaire de stocker les échantillons produits, ces derniers étant alors consommés tout de suite après avoir été produits.

Par ailleurs, l'aplatissement (ou *splatting* en anglais), c'est-à-dire le fait de calculer, en espace image, une primitive simple englobant l'ensemble des pixels affectés par une source lumineuse, présente de nombreux avantages. Tout d'abord, cela permet de grandement réduire le nombre de calculs en éliminant, *a priori*, une grande partie de l'image sur laquelle on est sûr que le *VPL* n'a pas d'influence. On s'épargne ainsi le fait de devoir calculer la distance entre le *VPL* et les pixels en dehors de la primitive englobante. Par ailleurs, l'aplatissement permet également d'exploiter les fonctionnalités accélérées matériellement par l'unité de rastérisation. En effet, lorsque nous construisons notre primitive, nous lui donnons comme profondeur celle du point de la sphère

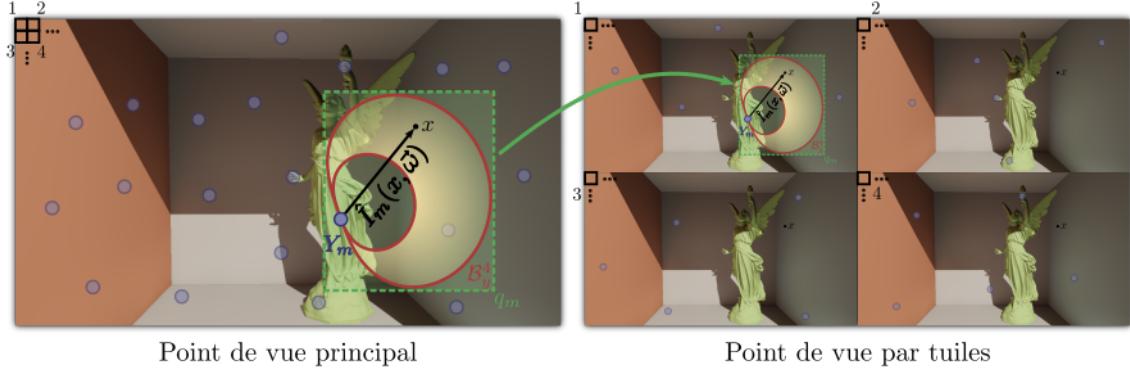


FIGURE 5.9 – Aplatissement en espace image. À gauche, le *VPL* Y_m a pour intensité lumineuse \hat{I}_m . Celui-ci a pour support l’ensemble \mathcal{B}_{y_m} défini par les *coupes antérogrades*. Pour calculer efficacement la contribution lumineuse de ce *VPL*, on commence par calculer le rectangle q_m englobant ce support de sorte à ce qu’un *Fragment Shader* soit appelé pour chaque pixel x touché par la primitive q_m . Pour chacun d’eux, on teste si la position correspondante est bien dans l’ensemble \mathcal{B}_{y_m} puis on calcule la contribution lumineuse du *VPL*, $\hat{I}_m(x, \vec{\omega})$. Pour tous les pixels en dehors de q_m aucun calcul n’est réalisé car aucun *Fragment Shader* n’est invoqué pour ce *VPL*.

À droite, le point de vue principal est subdivisé en 4 sous-tuiles de résolution 4 fois moindre. Pour construire ces tuiles, on distribue les pixels du point de vue principal en envoyant un pixel sur 4 dans l’une d’entre elles, selon le schéma de la figure. Dans cette situation, pour calculer la contribution lumineuse de Y_m , on choisit aléatoirement l’une des sous-tuiles et on réalise le même calcul que dans le point de vue principal. Or, comme la résolution est 4 fois moindre, il y a 4 fois moins de calculs à réaliser.

englobante le plus proche de la caméra. Ainsi, en activant le test de profondeur, si, lors de la rastérisation de la primitive, un fragment généré se retrouve derrière le pixel contenu dans la carte de profondeur de la scène, le *Fragment Shader* correspondant n’a pas besoin d’être invoqué. Comme ce test est réalisé nativement sur le processeur graphique, il se fait sans surcoût et de manière extrêmement efficace.

5.6.4 Rendu Entrelacé et Filtrage

D’après nos observations, cf. §5.7, le principal goulot d’étranglement de notre algorithme de simulation d’éclairage reste le calcul de la contribution lumineuse au niveau du pixel. En effet, nous avons constaté que notre technique est fortement limitée, d’une part, par la quantité d’opérations numériques que l’on doit réaliser, et d’autre part, par le nombre d’accès mémoire nécessaires pour reconstruire la géométrie au niveau du pixel touché. Par ailleurs, le phénomène que l’on cherche à rendre est le transport diffus de la lumière, qui demeure de basse fréquence spatiale. Il est alors classique de réduire la résolution de l’image afin de réduire de manière drastique les besoins en calcul. Les pixels de l’image finale sont ensuite interpolés à partir du rendu de basse résolution en prenant en compte les caractéristiques disponibles dans le tampon géométrique. Ce mécanisme d’interpolation est appelé filtrage bilatéral [Tomasi et Manduchi, 1998] ou filtrage bilatéral guidé [He et coll., 2013]. Cependant, bien que le mécanisme de sous échantillonnage et d’interpolation fonctionne convenablement dans les zones relativement homogènes, les régions avec beaucoup de variations spatiales (comme les feuillages, la géométrie dense, ou un bord de mur par exemple) produisent de forts artefacts de reconstruction.

Afin de réduire ces artéfacts, nous utilisons une approche de rendu entrelacé, [Keller et Heidrich, 2001, Laine et coll., 2007], qui consiste à construire des sous-tuiles du point de vue principal, *cf.* figure 5.9. On construit alors $N_{\text{tuiles}} = 4, 16$ ou 64 sous-tuiles dont chacune est constituée de N_{tuiles} fois moins de pixels, de sorte que chaque pixel du point de vue principal se retrouve dans exactement une sous-tuile. Pour évaluer la contribution d'un *VPL*, on choisit alors aléatoirement l'une des tuiles, et on calcule la projection en espace image de la primitive englobante. Comme cette tuile est de résolution N_{tuiles} fois moindre par rapport au point de vue principal, il y a donc N_{tuiles} fois moins de calculs et d'accès mémoire réalisés. On observe ainsi un gain proportionnel à N_{tuiles} , *cf.* §5.7. Enfin, le rendu final est calculé en désentrelaçant les sous-tuiles et en filtrant l'image ainsi obtenue à l'aide d'un filtre bilatéral guidé.

5.6.5 Rendu Progressif

Grâce à notre échantillonnage, le plus uniforme possible, et nos fonctions de support lisses, bien que le nombre de *VPL* soit relativement faible, nous sommes en mesure de produire à chaque instant un résultat lisse et visuellement plausible. Nous proposons néanmoins, ici, une extension possible de notre approche permettant de faire converger le résultat visuel vers la solution de l'intégrale.

Comme nous l'avons montré plus tôt, notre technique permet d'itérer sur plusieurs trames en faisant varier la graine aléatoire globale, *cf.* §5.5.1. En pratique, celle-ci reste fixe d'une trame à l'autre pour conserver un échantillonnage cohérent quelle que soit la transformation de la caméra ou de la géométrie. Cependant, si le point de vue demeure fixe, on peut alors calculer la moyenne de plusieurs échantillonnages indépendants, tel que nous l'avons formulé plus tôt, *cf.* §5.4.6. Cela permet ainsi d'étendre le champ d'utilisation de notre technique en faisant converger l'image vers l'intégrale I_f . Quitte à s'autoriser une latence dans le rendu, on peut alors envisager des scénarios plus complexes constitués de toutes sortes de géométries, avec, par exemple, une forte variation spatiale comme des textures très contrastées, ou des cartes de normales sur la géométrie. Chaque rendu est alors moyenné avec ceux des trames précédentes tant que la caméra reste immobile. De plus, comme le rendu est suffisamment lisse à chaque trame, aucun artéfact de bruit ne vient altérer l'image lors d'un mouvement contrairement aux méthodes classique de Monte-Carlo par lancer de rayons.

5.7 Résultats et Évaluation

Nous avons testé notre algorithme sur un ordinateur standard équipé d'un processeur graphique Quadro M6000. Nos observations montrent que les performances globales de notre approche dépendent de deux principaux facteurs : la résolution de l'image de sortie et le nombre de triangles qui composent la scène.

Ainsi, lorsque le rendu n'est pas limité par la taille du modèle, le facteur limitant est le nombre de *VPL* générés. Afin de réduire la quantité de calculs, on avons entrelacé les pixels de manière à réduire la résolution tout en conservant la totalité de l'information du point de vue, *cf.* §5.6.4. On remarque néanmoins que la dégradation du rendu engendrée par le sous-échantillonnage des pixels de l'image parvient à être efficacement compensée si l'on augmente par ailleurs le nombre de *VPL*. Cela nous permet donc d'ajuster la résolution du rendu tout en conservant un résultat

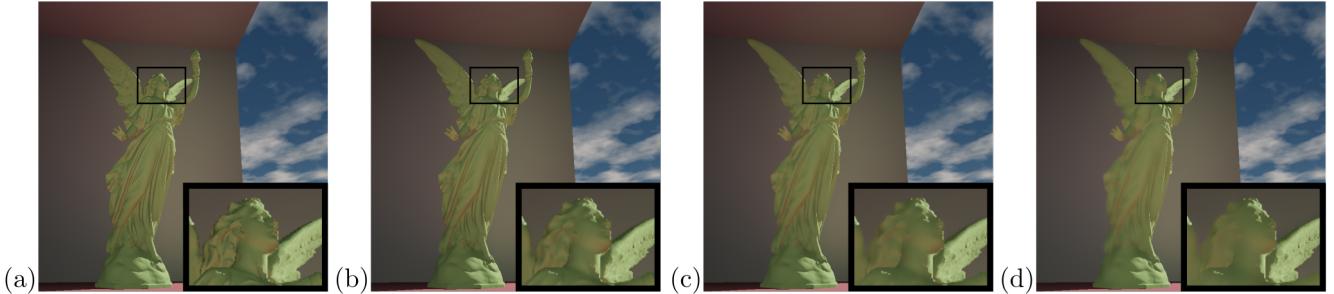


FIGURE 5.10 – Influence du sous-échantillonnage de l'image sur la statue de Lucy (28M triangles). La résolution est divisée par deux dans chaque dimension et le nombre de sous-tuiles est multiplié par quatre. Allant d'une tuile de 1024×1024 pixels (a) jusqu'à 64 tuiles de 128×128 pixels (d). À cause de l'interpolation, les détails du visage de la statue sont de plus en plus flous au fur et à mesure que la résolution est réduite. Quant à la durée de calcul, elle passe de 93 ms (a) à 48 ms (b), puis à 33 ms (c, d). Le temps de rendu de (c) et (d) sont identiques car le goulot d'étranglement de notre algorithme est alors la taille de la géométrie et non plus la résolution de l'image.

	Sponza	Plateforme	Statue de Lucy		
N_{tuiles}	64	4	1	4	16
Direct	4 ms	1 ms	18 ms	18 ms	18 ms
Indirect	4 ms	13 ms	75 ms	30 ms	15 ms
pipeline direct	1.7 ms	9.8 ms	17.6 ms	15.5 ms	12.8 ms
pipeline de subdivision	1.1 ms	2.8 ms	57.4 ms	14.1 ms	1.4 ms
filtrage	1.2 ms	0.4 ms	0 ms	0.4 ms	0.8 ms
Trame	8 ms	14 ms	93 ms	48 ms	33 ms

TABLE 5.2 – Temps de constructions des différentes étapes du pipeline.

convaincant avec des performances correctes, *cf.* figure 5.10. La table 6.1 reporte le temps des différentes étapes de notre algorithme.

Comparaisons Nous avons comparé notre approche avec deux autres algorithmes que nous avons réimplémentés : la méthode largement reconnue *Screen Space Directional Occlusion (SSDO)* [Ritschel et coll., 2009b] ainsi qu'une approche plus récente *Deep Screen Space (DSS)* [Nalbach et coll., 2014]. Ces deux algorithmes partagent les mêmes propriétés que notre méthode. Premièrement, ils tournent en temps réel sans nécessiter d'étape de préparation de la géométrie, c'est-à-dire sur des scènes complètement dynamiques. Deuxièmement, ils cherchent à simuler le premier rebond diffus de la lumière sans prendre en compte l'occlusion indirecte. Nos comparaisons ont été faites sur cinq scènes présentant des types de structures, nombres de triangles et conditions d'éclairage différents. La table 5.3 présente les résultats visuels obtenus avec les trois approches et rapporte le temps d'exécution de la trame complète, c'est-à-dire l'éclairage direct et indirect.

On observe tout d'abord que les résultats produits avec notre approche sont visuellement similaires, voire meilleurs que pour *DSS* tandis que nous sommes significativement plus rapides. Cette tendance est d'autant plus marquée lorsque le nombre de polygones augmente. Cela peut s'ex-

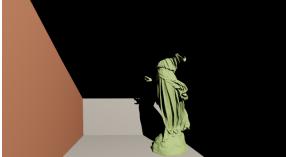
Modèles	Éclairage direct	<i>SSDO</i>	DSS	Notre approche
Sponza (270k tri.)				
Plateforme pétrolière (700k tri.)				
Voiture (3m tri.)				
San Miguel (12m tri.)				
Lucy (28m tri.)				

TABLE 5.3 – Comparaisons à la résolution 1280x720 pixels.

plier par notre stratégie en deux temps de subdivision, permettant de réduire significativement la charge de travail de l’unité de tessellation. Par ailleurs, on observe que contrairement à nous, *SSDO* est souvent mis à défaut lorsque les régions de la scène réfléchissant la lumière se trouvent en dehors du point de vue principal. Bien sûr, *SSDO* étant une approche fondée sur l’espace image, ses performances demeurent élevées quelle que soit la complexité de la géométrie. Cependant, le résultat visuel reste faible au regard de l’éclairage direct. Au contraire, notre approche, comme *DSS*, parvient à produire un résultat plus cohérent au cours du temps et permet de prendre en compte les interactions lumineuses sur de grandes distances.

De plus, la figure 5.11 illustre l’exécution de notre technique en activant le rendu progressif. En effet, en faisant varier la graine aléatoire uniforme, *cf.* §5.6.5, on est capable de générer, à chaque trame et sans aucun coût supplémentaire, une série de rendus dont la moyenne converge. On peut alors juger que, au prix d’un rendu plus long, cette approche permet de faire complètement dispa-

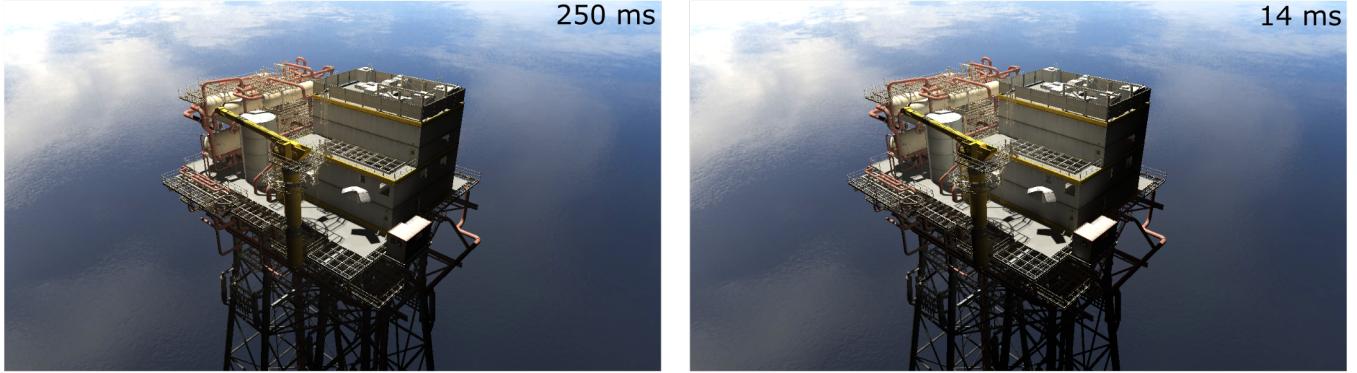


FIGURE 5.11 – Comparaison de la version progressive de notre approche. (Gauche) Rendu progressif moyennant 50 images (250ms). (Droite) Rendu temps-réel (14 ms).

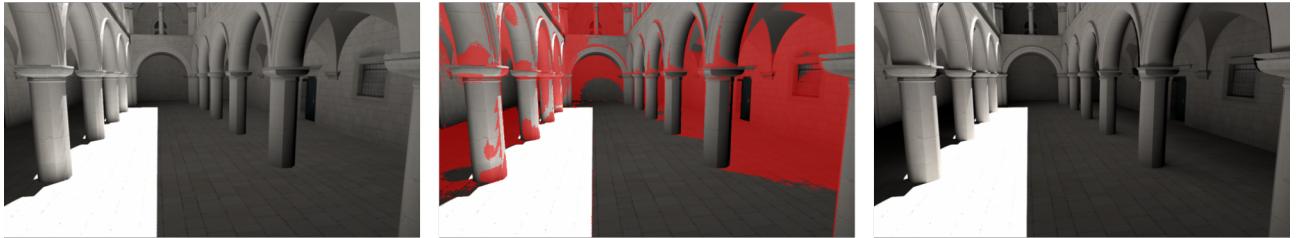


FIGURE 5.12 – Comparaison entre notre technique (gauche) et la vérité terrain (droite). Celle-ci est calculée pour le premier rebond diffus de la lumière par lancer de chemins avec 8196 échantillons par pixel. Entre ces deux images, les erreurs sont $RMSE = 20.10$ et $SSIM = 0.8761$. Au centre, les pixels rouges (25% de l'image) correspondent aux régions dont la différence est supérieure à 6%. On observe le manque d'ombre de contact clairement visible au pied des piliers sur la droite. Les murs sous les voûtes de droite, bien plus clairs dans notre approche que dans la référence, révèlent également ce manque d'occlusion. En effet, la totalité de la façade de gauche reflète la lumière alors que celle-ci aurait du être stoppée par le premier étage du bâtiment.

raître les artéfacts inhérents à la technique, comme ceux issus de la singularité du terme géométrique.

5.8 Discussion et Ouverture

Nous avons présenté ici, une nouvelle technique de rendu d'éclairage indirect diffus en temps réel. Notre méthode géométrique génère des VPL de manière stochastique grâce à un processus d'échantillonnage qui permet, dans un premier temps, de raffiner la géométrie trop grossière, ou dans un second temps, de la simplifier dans les régions trop denses. Cette approche ne requiert aucun traitement préalable de la scène ni ne nécessite de maintenir au cours temps des structures de données complexes. Elle est également adaptée aux scènes massives composées de plusieurs millions de triangles qui peuvent être complètement dynamiques et s'intègre naturellement dans les moteurs de rendu modernes. Enfin, nous avons évalué notre technique sur un ensemble de scènes allant des modèles de CAO aux données scannées et nous parvenons à obtenir des performances interactives à chaque fois.

L'algorithme que nous venons de décrire ne parvient pas à simuler la projection d'ombre issue de l'éclairage indirect, *cf.* figure 5.12, ni ne permet de simuler davantage de rebonds au delà du premier. Malheureusement, il n'existe pas, à notre connaissance, de moyen permettant d'étendre trivialement notre stratégie d'accumulation de l'éclairage en espace image afin de gérer les rebonds multiples. En ce qui concerne le calcul de la visibilité entre les pixels et les *VPL*, en revanche, nous proposons dans la suite un algorithme visant à répondre à ce problème.

CARTES D'HYPEROVoxELS SPHÉRIQUES

Dans ce chapitre nous présentons les *Cartes d'HyperVoxels Sphériques*, une nouvelle technique permettant de calculer de manière approximative mais très efficace la visibilité entre deux points arbitraires d'une scène. Ce terme est responsable, dans l'opérateur de transport, de la simulation des occlusions secondaires et constitue très souvent le goulot d'étranglement dans le calcul de l'éclairage. Dans notre système, nous étendons l'approche classique de *marche de rayons* en exploitant davantage la mémoire mise à disposition par les nouvelles générations de processeurs graphiques afin d'optimiser les étapes critiques de l'algorithme. Ainsi, à partir d'un jeu de voxelisations grossières de la géométrie calculées à la volée selon un grand nombre de directions, nous réalisons les requêtes de visibilité en un nombre minimal d'étapes de *marche de rayons*, contrairement aux approches traditionnelles qui peuvent être arbitrairement longues. Nous exploitons alors notre paramétrisation de l'espace des directions pour retrouver en quelques opérations la grille de voxelisation la plus appropriée à ce calcul. Enfin, nous présentons le potentiel de notre approche simple mais efficace sur plusieurs applications dont le calcul des ombres douces, l'occlusion ambiante, ainsi que l'éclairage d'environnement sans recourir à la moindre préparation de la géométrie. Cela nous permet ainsi d'être utilisable sur des scénarios complètement dynamiques.

6.1 Introduction

Étant donné une scène et un segment de l'espace défini par deux points, la fonction de visibilité renvoie 1 ou 0 selon si ce segment intersecte ou non la géométrie. Malheureusement, évaluer cette fonction, c'est-à-dire effectuer une requête de visibilité, s'avère être une tâche complexe en pratique et les approches classiques ont très souvent recourt à des structures d'accélération telles que des arbres de partitionnement de l'espace, cf. §3.1, ou bien des *cartes d'ombre* [Williams, 1978], cette dernière étant notamment très populaire dans les applications temps-réel. Le choix de la structure d'accélération est primordial pour concilier à la fois les performances de l'application et la qualité visuelle souhaitée. Dans ce sens, contrairement aux approches hiérarchiques, les algorithmes dérivant des *cartes d'ombre* permettent d'effectuer des requêtes de visibilité en temps constant tout en limitant le nombre d'indirections et donc d'accès mémoire, cf. §2.2.3. Cependant, cette efficacité vient avec un coût. En effet, si les cartes sont calculées très efficacement c'est parce qu'elles tirent profit de l'accélération matérielle offerte par la rastérisation et, comme nous l'avons vu précédemment, cf. §2.2.1, les architectures actuelles ne permettent la rastérisation que pour un jeu de rayons issus d'une même origine, potentiellement à l'infini. Cela signifie qu'avec une *carte d'ombre*, la visibilité ne peut pas être enregistrée de manière globale. Il est donc, en général,

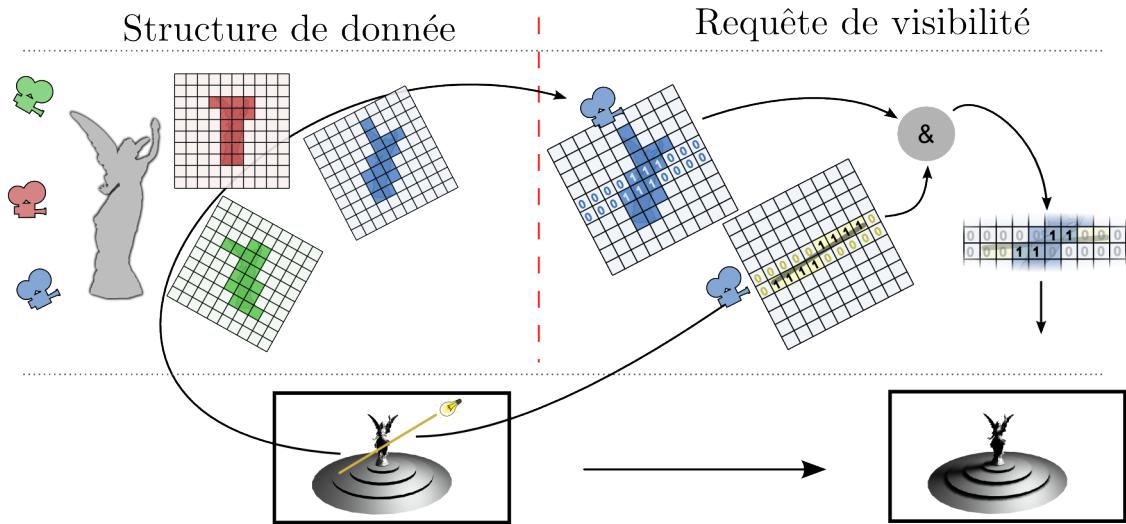


FIGURE 6.1 – Pipeline temps réel des cartes d'hypervoxel sphérique. De gauche à droite : étant donné une scène 3D, nous en calculons dans un premier temps la voxélisation selon un jeu de directions et nous enregistrons le résultat dans un tableau de texture grâce à une représentation binaire compacte. Les 1 représentent les voxels contenant de la géométrie et les 0 pour les zones vides. Au moment du rendu, pour réaliser une requête de visibilité entre deux points de l'espace, on commence par chercher parmi les directions calculées au préalable celle qui est la plus alignée avec ces deux points et on extrait de la grille associée les voxels correspondant au segment. Puis nous calculons le masque de bits pour ce segment projeté dans ce point de vue. Enfin, nous répondons à la requête en temps constant en calculant le *et* bit à bit entre la donnée récupérée et le masque du segment. Les deux points sont ainsi mutuellement visibles si et seulement si le résultat de cette opération est zéro.

impossible de calculer efficacement la visibilité entre deux points arbitraires de l'espace à partir d'une telle carte. De même, les volumes d'ombre (ou *Shadow Volumes* en anglais) [Crow, 1977] forment une autre classe d'algorithme permettant de calculer les ombres projetées d'une source ponctuelle de manière exacte en utilisant le pipeline de rastérisation. Malgré tous, cet algorithme résout la visibilité en construisant une structure fortement liée au point de vue et à la position de la source lumineuse.

Dans notre approche nous conservons les avantages des *cartes d'ombre* tout en étendant le domaine de définition. Ainsi, pour calculer la visibilité, nous nous contentons d'un nombre très restreint d'accès mémoire et d'opérations tout en permettant de répondre pour n'importe quelle segment de l'espace. Pour ce faire, nous proposons d'enrichir la paramétrisation bidimensionnelle de l'espace projectif des *cartes d'ombre* parallèles avec deux dimensions supplémentaires correspondant à un sous-échantillonnage de l'ensemble des directions, c'est-à-dire la sphère unité. À chacun des points de cette paramétrisation quadridimensionnelle est associée une unique droite que nous échantillonons alors par des hypervoxels de dimension cinq. Nous montrons par la suite que cette structure de données peut efficacement être calculée sur les architectures modernes à partir de la rastérisation.

6.2 Vue d'ensemble

Nous donnons ici une vue d'ensemble de notre approche pour calculer en temps réel la visibilité entre deux points. Notre objectif premier consiste à utiliser ce calcul pour simuler les ombres indirectes en tant que troisième problématique évoquée au chapitre 3 pour l'éclairage indirect fondé sur les caches. Nous exploitons donc le fait que ce signal soit de basse fréquence pour justifier que l'on puisse se contenter d'une représentation approximative de la géométrie pour l'approcher, *cf.* figure 6.1. Nous pouvons décomposer notre approche selon les points suivants :

- dans un premier temps nous construisons une structure d'accélération fondée sur le pipeline de rastérisation ; il s'agit d'une multitude de voxelisations de la géométrie alignées selon de nombreuses directions ;
- pour optimiser cette construction, nous utilisons une représentation sous forme de nuage de points orientés de la géométrie ; nous les distribuons alors à la volée dans les différentes grilles ; le choix de la grille est réalisé préférentiellement par rapport à l'angle entre l'orientation de la grille et la normale à la surface ;
- on définit ensuite sur ces cartes une paramétrisation de l'espace à cinq dimensions des points orientés (trois pour l'espace et deux pour les directions d'où notre appellation d'hypervoxel sphérique) ;
- enfin, pour répondre à la requête de visibilité, nous tirons profit de notre représentation en déterminant parmi l'ensemble des directions de nos cartes, celle qui rend optimale l'étape de marche de rayons, c'est-à-dire celle qui est le plus alignée avec la direction de la requête ;
- notre paramétrisation des points orientés nous permet alors de stocker de manière contiguë en mémoire l'ensemble des hypervoxels nécessaires à la résolution d'une requête de visibilité, quelque soit sa direction ; cela minimise ainsi la quantité de mémoire utilisée pour répondre à la requête et constitue la clef de la performance de notre algorithme.

Dans la suite nous commençons par présenter l'algorithme de marche de rayons sur lequel se fonde notre approche, *cf.* §6.3. Puis dans un second temps, *cf.* §6.4, nous nous analysons un peu plus en détail les conditions qui rendent cet algorithme plus ou moins efficace. À partir de cette analyse, nous proposons les cartes d'hypervoxels sphériques, *cf.* §6.5, permettant d'utiliser la marche de rayons en se plaçant systématiquement dans son cas optimal d'utilisation et nous en détaillons l'implémentation, *cf.* §6.6. Enfin, avant de conclure, *cf.* §6.8, nous évaluons notre approche sur un jeu de scène nécessitant le calcul d'ombres douces, *cf.* §6.7.

6.3 Marche de rayons

L'algorithme de *marche de rayons* [Appel, 1968] (de l'anglais *ray marching*) vise à tester l'intersection entre un rayon et la géométrie à l'aide d'une structure d'accélération adaptée. Comme de nombreux algorithmes de rendu temps réel, il se décompose en deux grandes étapes : tout d'abord, la scène est échantillonnée et stockée dans une structure d'accélération ; puis, lors des étapes de rendu, celle-ci est lue pour reconstruire la géométrie à partir des informations qu'elle contient. Cependant, à la différence des approches fondées sur des arbres de partitionnement de l'espace,

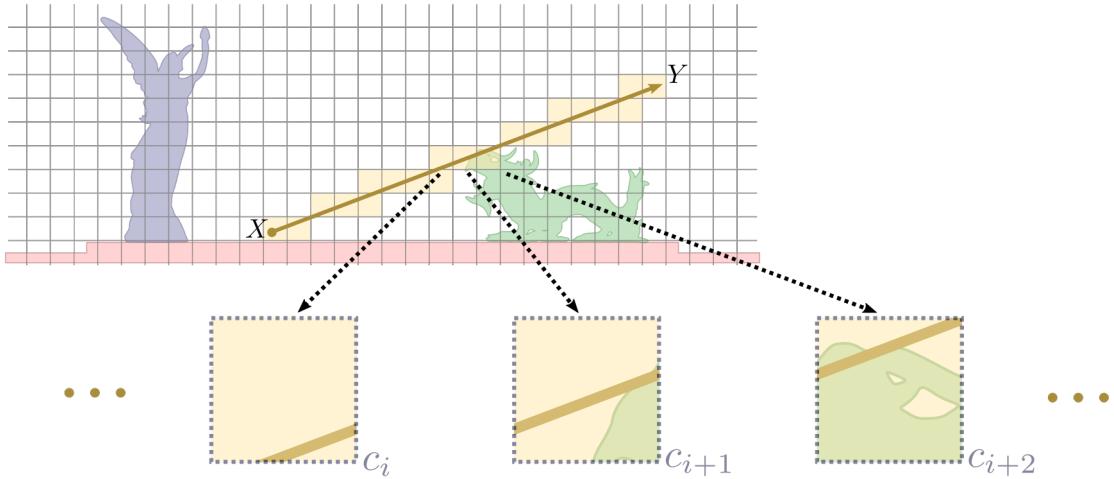


FIGURE 6.2 – Schéma haut niveau de l’algorithme générique de *marche de rayons*. Pour tester la visibilité entre les points X et Y , on commence par déterminer l’ensemble des cellules (en jaune) de la grille traversées par le segment correspondant. Le mécanisme de rastérisation de ligne permet d’itérer efficacement sur cette famille de cellules, $(c_i)_i$, de X vers Y . Pour chaque cellule, il suffit de reconstruire la géométrie sous-jacente grâce à l’information qu’elle contient et de la tester avec le rayon pour résoudre la requête. Dans le cas d’une information binaire indiquant par 1 ou 0 si une cellule contient de la géométrie, il suffit qu’au moins une des cellules c_i soit non nulle pour déterminer que les deux points ne sont pas mutuellement visibles.

cf. §3.1, la *marche de rayons* utilise des grilles régulières comme structure d’accélération. À la manière d’une table de hachage, celle-ci permet l’accès à une donnée en un temps théoriquement indépendant de la complexité de scène. De cette façon, le test d’intersection est réalisé en déterminant, dans un premier temps, l’ensemble des cellules par lesquelles passe le rayon, puis, dans un second temps, en itérant sur chacune de ces cellules tout en testant le rayon avec une reconstruction de la géométrie correspondante, *cf.* figure 6.2.

Deux problématiques majeures apparaissent alors. Tout d’abord il est nécessaire que la structure de données puisse être accédée rapidement et que l’on puisse déterminer efficacement l’ensemble des cellules traversées par un segment. Par ailleurs, il est nécessaire d’avoir une représentation de la géométrie sous-jacente à chaque cellule qui concilie à la fois la vitesse d’accès, l’occupation mémoire, ainsi que la fiabilité et la rapidité de la reconstruction.

Les algorithmes de *marche de rayons* sont rendus efficaces par le fait de pouvoir exploiter au maximum le calcul parallèle sur les nombreux coeurs des processeurs graphiques. En effet, les partitionnements de l’espace tels que ceux offerts par le processus de rastérisation, à savoir la projection orthoscopique ou perspective, rendent la construction des cartes particulièrement rapide et simple. De plus, une telle paramétrisation de l’espace implique naturellement la possibilité de déterminer aisément l’ensemble des cellules intersectant une droite, car ce processus correspond exactement à celui de la rastérisation de ligne.

À présent, nous nous intéressons aux différentes façons de partitionner l’espace pour calculer une marche de rayons au regard de leur intégration dans une pipeline graphique. Nous détaillons également quelles contraintes cette paramétrisation implique sur la nature des échantillons. Cependant, l’étude générale des problèmes liés à la visibilité et sa simulation en temps réel dé-

passent le cadre de cette thèse et nous référons le lecteur aux revues de Hasenfratz et coll. [Hasenfratz et coll., 2003] ainsi que Eisemann et coll. [Eisemann et coll., 2016] sur le sujet.

Cartes de profondeur Comme nous en avons parlé précédemment, cf. §3.2.4, les approches fondées sur l'espace image bénéficiant de nombreux avantages dont celui d'utiliser directement le pipeline classique de rastérisation et de ne pas nécessiter d'étape de calcul supplémentaire pour échantillonner la scène. En effet, la carte de profondeur utilisée pour la marche de rayon est quoi qu'il arrive calculée pour le rendu principal. De plus, l'étape d'itération sur le rayon est un processus relativement simple car il correspond à la rastérisation d'une ligne projetée à l'écran. Cependant, bien que la donnée enregistrée dans une telle carte corresponde à une distance depuis la caméra, ce qui constitue une évaluation très précise de la géométrie, seules les régions visibles depuis la caméra sont échantillonnées. On perd ainsi l'information d'une grande partie de la géométrie. Comme pour l'ensemble des approches en espace image, il est alors possible de capturer davantage de couches par l'épluchage de profondeur (ou *depth peeling*) mais cela complexifie la marche le long du rayon.

Voxélisation Contrairement aux approches en espace image, les méthodes par voxelisation considèrent l'intégralité de la scène discrétisée selon une collection de cellule. Ce partitionnement revient à pavier l'espace tridimensionnel de la scène en entier ou en partie et non plus à une projection de celui-ci. Il est cependant nécessaire d'adapter la donnée enregistrée dans chaque cellule afin de borner la consommation mémoire étant donné qu'il faut alors représenter toute une dimension supplémentaire. Ainsi, la représentation la plus simple possible consiste à enregistrer de façon binaire si une cellule, et donc un voxel, contient ou non de la géométrie ce qui permet de stocker un voxel sur un unique bit. De plus, comme le montrent Eisemann et Décoret [Eisemann et Décoret, 2006], la construction d'une grille correspondant à une telle représentation est tout à fait réalisable grâce au pipeline de rastérisation classique associé aux opérations atomiques sur les fragments. Dans le cas d'une grille régulière de voxelisation, la marche des rayons consiste alors à généraliser l'algorithme de rastérisation d'une ligne en dimension 3. Cependant, la précision offerte par cette représentation dépend à présent de la résolution des voxels, or seule une petite partie d'entre eux (ceux intersectant la surface de la géométrie) contiennent de l'information, les autres étant dans le vide. Pour éviter de gâcher de la mémoire, les tables de hachage parfaite [Lefebvre et Hoppe, 2006, Alcantara et coll., 2009] permettent d'adapter la structure de données à la géométrie en décrivant les voxels sans collisions, c'est-à-dire en définissant une fonction injective de l'ensemble des voxels non nuls vers les indices d'un tableau. Malheureusement, bien que cela produise une donnée compacte et cohérente en espace, la construction de cette fonction de hachage demeure compliquée et nécessite de longues opérations de traitement. Dans le même ordre d'idée, les approches hiérarchiques, Kämpe et coll. [Kämpe et coll., 2013], permettent d'adapter la résolution des voxel à l'aide principalement d'*octree*. Cependant, outre que la génération d'une telle structure peut s'avérer couteuse, le coût de la marche elle-même s'en retrouve grandement augmenté. En effet, la gestion du flux d'exécution des coeurs parallèles tendant à diverger, les accès mémoire deviennent rapidement incohérents, limitant alors les performances. De plus, la taille des données est bien supérieure pour représenter les noeuds d'un arbre. La voxelisation en tant que structure d'accélération peut ainsi être utilisée pour représenter de manière très précise la géométrie cependant, dans ce cas d'utilisation il ne semble plus être possible de calculer dynamiquement la structure de donnée ni de l'interroger de manière aussi efficace. Dans notre approche, nous considérons la voxelisation sur des grilles régulières car, pour notre cas d'application, il suf-

fit d'une représentation imprécise de la géométrie pour reconstruire le signal tandis que le test de visibilité est critique.

Visibilité sur le champ lumineux La visibilité peut être vue comme une application à 6 dimensions correspondant à un segment de l'espace. Ainsi, les structures telles que les grilles de voxelisation à trois dimensions doivent être évaluées en de nombreux points lorsque l'on souhaite synthétiser la réponse de visibilité. Pour espérer réduire ce nombre d'évaluations il est donc nécessaire de stocker l'information sur une structure de données possédant davantage de dimensions. En cela, les cartes d'ombres imparfaites de Ritschel et coll. [Ritschel et coll., 2008b] peuvent être interprétées comme une structure de données discrétilisant une fonction à 4 dimensions qui, à toute droite issue d'un point de la géométrie lui associe la distance à la géométrie la plus proche. Cependant, dans cette approche, l'objectif est d'évaluer la visibilité des *VPL*, c'est-à-dire des points définis pour lesquels les cartes sont calculées. Pour étendre et interpoler le calcul à des points quelconques de la surface, Ritschel et coll. [Ritschel et coll., 2008a] construisent alors une courbe parcourant l'intégralité de la surface de manière cohérente. Dans notre approche nous proposons une paramétrisation de l'espace des droites qui permet d'évaluer n'importe quelle segment de manière agnostique à la scène sous-jacente et ne nécessitant pas de traitement sur la géométrie. Dans le même ordre d'idée, Ritschel et coll. [Ritschel et coll., 2007] proposent de construire autour de chaque objets une collection de cartes de profondeur pour précalculer la fonction de visibilité de l'objet. Cependant, bien qu'ils proposent un mécanisme de compression sans perte de données, le fait d'enregistrer la profondeur est certes très précis, mais ne permet de répondre qu'aux requêtes dont l'un des deux points du segment est en dehors de l'enveloppe convexe de l'objet. Notre approche, en revanche, propose d'enregistrer l'information de position sur la totalité des segments car les *VPL* que nous considérons sont situés sur la surface, et nécessairement à l'intérieur de cette enveloppe.

Dans la suite, nous utilisons la voxelisation comme base de notre algorithme. Cependant, après avoir analysé le cadre dans lequel cette structure de données peut être utilisée de manière optimale, nous étudions pourquoi le cas normal d'utilisation n'est pas optimal. Nous proposons alors une nouvelle structure de données permettant de rendre optimal toutes les requêtes de visibilité.

6.4 Marche Idéale

La voxelisation est devenue une technique de moins en moins onéreuse et un bon choix de structure d'accélération. Pour en tirer profit, la stratégie classique consiste à utiliser la marche de rayons. Cependant cette stratégie passe difficilement à l'échelle lorsque l'on souhaite réaliser des millions de requêtes à cause des accès mémoire qui deviennent nombreux et incohérents. Les requêtes sont réalisées lors du rendu et constituent la partie critique de l'algorithme. Nous avons donc cherché à les rendre les plus efficace possible en proposant de minimiser le nombre d'accès mémoire pour y répondre sans pour autant augmenter la quantité de calculs. Il nous a donc fallu revoir la structure d'accélération de sorte à ce qu'un minimum d'accès mémoire soit requis pour synthétiser la réponse.

Nous allons commencer par regarder ce qu'il advient dans le cas idéal d'utilisation de la *marche de rayons* au regard de notre représentation mémoire. Puis dans un second temps, nous regardons ce qui se passe dans le cas réel et comment faire pour se rapprocher du cas idéal.

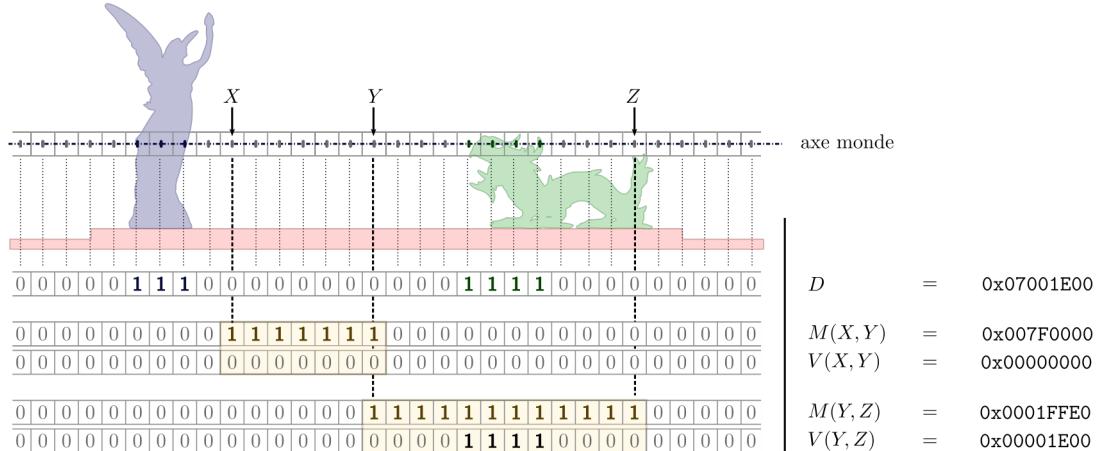


FIGURE 6.3 – *Marche de Rayons* en dimension un. Dans un premier temps, l’*axe monde* est subdivisé en cellules ce qui revient à construire un pavage de l’espace. Ensuite, en superposant la géométrie à ces cellules, on construit l’échantillonnage de la scène sous la forme d’un champ de bits initialisés à 0, et les assignant à 1 seulement si ils correspondent à une cellule à l’intérieur de la géométrie. En parallèle, la donnée correspondante (D) est enregistrée sous la forme compacte équivalente d’un entier (ici de 32 bits). Dans un second temps, pour répondre à la requête de visibilité entre les points X et Y , on détermine d’abord la plage couverte en mémoire (zone jaune) et on teste si au moins un bit correspondant dans la donnée D n’est pas nul. De la même manière que pour définir D , le champ de bits correspondant à cette plage peut être représenté par un entier $M(X, Y)$ que nous nommons le *masque de visibilité*. Grâce à lui, la réponse au test de visibilité revient à regarder le résultat d’un *ET* logique entre M et D . Ici, entre X et Y , $V(X, Y)$ est alors évalué à 0 et les points sont effectivement visibles. Entre Y et Z , $V(Y, Z)$ est différent de 0 signifiant qu’il y a de la géométrie entre eux.

6.4.1 Cas Unidimensionnel

Dans un cadre hypothétique optimal de l’algorithme de *marche de rayons*, l’intégralité du monde gît le long d’un axe, que nous nommerons l’*axe monde*, cf. figure 6.3. Dans ce contexte, on définit la structure de donnée comme un tableau unidimensionnel enregistrant un échantillonnage de la géométrie, où chaque case contient une information binaire de présence ou non de géométrie. Le tableau est alors encodé sous la forme compacte d’un champ de bits où à chaque bit correspond une case dont la valeur est égale à 1 si elle est à l’intérieur de la géométrie, et 0 sinon. Dans ce contexte, un segment de visibilité correspond donc à une collection de cases contiguës entre les deux extrémités du segment. Le calcul de la visibilité revient alors à examiner chacune de ces cases en retournant 1 si au moins une d’entre elles est non nulle et 0 dans le cas contraire.

Par ailleurs, grâce à la représentation compacte de la structure de données sous forme de champ de bits, au lieu d’itérer sur chaque cellule, il est possible d’utiliser quelques opérations arithmétiques basiques pour déterminer la visibilité entre deux points. En effet, comme nous venons de le voir un segment correspond à une collection de cellules. On peut ainsi décrire ces cellules par un masque entre deux points, c’est-à-dire par un champ de bits dont les cellules valent 1 si elles sont à l’intérieur du segment, et 0 autrement. Le test de visibilité revient alors à composer ce masque avec la structure de données en utilisant l’opérateur logique *ET*. Encore une fois, si le résultat est nul cela signifie que les deux points sont mutuellement visibles, sinon cela signifie qu’il existe de la géométrie entre eux, et donc qu’ils ne sont pas mutuellement visibles. L’algorithme en figure 6.4 et la figure 6.3 illustrent ce processus.

```

// Renvoie l'indice de la cellule contenant le point X
Entier CalculeIndiceDeCellule(Point1D X)

// Renvoie un entier dont les bits sont égaux à 1 seulement
// s'ils appartiennent au segment défini par [borneX, borneY]
Entier CalculeMasqueBinaire(Entier borneInf, Entier borneSup)
    Entier masqueInf = (1 << borneInf) - 1
    Entier masqueSup = (2 << borneSup) - 1
    Renvoie masqueSup & ~masqueInf

// Renvoie vrai si et seulement X et Y sont mutuellement visibles
// selon le champ de bits enregistré dans donnée
Booleen CalculeVisibilite1D(Entier donnée, Point1D X, Point1D Y)
    Entier borneX = CalculeIndiceDeCellule(X)
    Entier borneY = CalculeIndiceDeCellule(Y)
    Entier borneInf = min(borneX, borneY)
    Entier borneSup = max(borneX, borneY)
    Entier masque = CalculeMasqueBinaire(borneInf, borneSup)
    Renvoie (masque & donnée) == 0

```

FIGURE 6.4 – Algorithme de calcul de visibilité en dimension un.

On note que cet algorithme fonctionne une seule opération tant que la résolution d'échantillonnage est inférieure à la taille du registre, c'est-à-dire 32 bits. Nous considérons plus tard la situation où ce n'est plus le cas, *cf.* §6.6.

6.4.2 Représentation Mémoire

Dans le cas unidimensionnel, la représentation mémoire de la structure de données ne présente aucun problème. En effet, l'utilisation d'un tableau est relativement évidente car alors, l'agencement physique du stockage mémoire reflète exactement la paramétrisation du monde. Il en découle alors deux avantages. Premièrement, la structure de données est facile à construire car elle nécessite peu de calculs pour faire correspondre une cellule à un emplacement physique. Et deuxièmement, cela permet de rendre très efficaces les requêtes de visibilité. Comme nous venons de le voir, une requête est réalisée en inspectant une plage contiguë du tableau, et donc des cellules voisines en mémoire. Cela permet ainsi de minimiser la quantité de transactions mémoires qui constituent le principal goulot d'étranglement de l'algorithme.

En revanche, en dimension supérieure, il n'existe pas de façon triviale d'étendre la représentation précédente tout en conservant les avantages cités ci-dessus. Cela vient du fait que la mémoire physique reste accessible de manière unidimensionnelle alors que les données représentées, quant à elles, sont de dimension deux ou trois (voire plus). Ainsi, les bits pertinents pour une requête quelconque n'ont *a priori* aucune raison de représenter une région contiguë de la mémoire. Il est cependant nécessaire de choisir une paramétrisation de l'espace pour faire correspondre à chaque case mémoire une partie de l'espace. Or, parmi l'ensemble des choix possibles, il se trouve que le mécanisme de rastérisation accélère matériellement la construction de la structure de données d'une certaine catégorie de paramétrisations, à savoir la voxélisation. Ainsi, pour s'adapter à ce mécanisme, on fait le choix arbitraire d'une direction principale et, d'un plan (ou d'une droite pour la dimension deux) orthogonal à cette direction *cf.* figure 6.5. Ce plan (ou cette droite) est ensuite subdivisé en cellules, en l'occurrence en pixels, sur lesquelles sont construits un tableau

unidimensionnel représentant les bits de la géométrie stockés de manière contigüe. Chaque pixel contient alors la donnée d'un axe monde tel que nous l'avons défini en dimension un, cf. §6.4.1.

Nous traitons plus tard, cf. §6.5, le problème consistant à optimiser les accès à cette structure, mais avant cela, nous analysons les problèmes induits par cette représentation lorsque l'on généralise l'algorithme de *marche de rayons* en dimension supérieure.

6.4.3 Marche naïve en dimension deux et trois

Comme nous venons de le dire, en dimension deux ou trois, il est nécessaire de choisir une paramétrisation linéaire de l'espace pour la représenter en mémoire. La figure 6.5 illustre alors les trois différents scénarios qui peuvent se produire lorsque l'on généralise l'algorithme de *marche de rayons* en dimension deux. Ainsi, chaque pixel possède un ou plusieurs entiers dont les bits correspondent à la discrétisation de la géométrie. Au niveau de la requête de visibilité, il peut survenir plusieurs situations. Le cas le plus simple est celui où on peut trivialement se rapporter au cas unidimensionnel, c'est-à-dire lorsque le segment de visibilité est suffisamment aligné avec la direction principale pour que tous les voxels qu'il intersecte tombent dans sur le même pixel. Dans ce cas, que nous avons qualifié d'optimal sur la figure, il suffit de déterminer le pixel qui porte le rayon. Cette opération est relativement classique dans le cadre d'échantillonnage de texture et consiste à trouver l'adresse du pixel dont les coordonnées sont calculées par la projection d'une des deux extrémités du segment. Une fois que le champs de bits correspondant a été récupéré, on peut alors appliquer l'algorithme de test de visibilité que nous décrivions au paragraphe précédent, cf. figure 6.4.

Ce cas simple n'est malheureusement pas le seul qui peut se produire, et c'est d'ailleurs loin d'être le cas le plus fréquent. Le plus souvent, il se produit le cas, que nous avons nommé normal, pour lequel le segment appartient à plusieurs pixels. Dans ce cas l'algorithme se décompose en deux parties. Tout d'abord, il est nécessaire de déterminer l'ensemble des indices couvert par le segment. Dans le cas bidimensionnel, il suffit de connaître les indices des deux extrémités. La plage d'indices couverte correspond alors à tous ceux compris entre ces valeurs. Ensuite, pour chacun des pixels traversés par le segment, on calcule l'ensemble des bits qui contiennent effectivement le segment. Nous détaillons ce calcul dans le cas tridimensionnel dans un paragraphe ultérieur, cf. §6.5. En ayant ainsi subdivisé le segment initial, il nous est à présent possible de se rapporter au cas unidimensionnel en répondant 1 dès qu'au moins un des sous segments traverse un bit non nul.

On note que dans le cas précédent, il nous est nécessaire de réaliser davantage de calculs et d'accès mémoire que dans le cas optimal. De plus, bien que sur la figure que nous avons réalisé, le nombre de pixels à interroger reste faible dans le cas normal, il existe des situations où il est nécessaire d'interroger beaucoup de pixels avant de pouvoir répondre à la requête. Dans le pire cas, c'est-à-dire lorsque le segment est presque perpendiculaire à la direction principale, il est nécessaire de faire autant d'accès mémoire que la taille du segment. Par ailleurs, pour chaque sous segment, la quantité d'informations importantes ne correspond qu'à un seul bit de données. On est alors clairement dans un cas sous optimal d'utilisation de la structure de données.

De cette façon, on vient de voir que l'approche de *marche de rayons* classique présente des performances inégales en fonction du choix arbitraire d'une direction privilégiée. Cela est d'autant dommageable que les accès mémoire ainsi que les calculs supplémentaires qui en découlent sont les principaux goulets d'étranglement des applications temps réel. C'est pourquoi nous voyons à présent comment tirer profit de cette observation en tentant de minimiser les impacts du choix

Données :	D₀ = 0x08000000 D₁ = 0xC8000000 D₂ = 0x07800000 D₃ = 0x07000000 D₄ = 0x03000000 D₅ = 0x03000000 D₆ = 0x07001E00 D₇ = 0x07000FA0 D₈ = 0x07001FE0 D₉ = 0x07000F60 D₁₀ = 0xFFFFFFFF
Requête 1 : cas optimal	M₀ = 0x00000000 M₁ = 0x00000000 M₂ = 0x00000000 M₃ = 0x00000000 M₄ = 0x00000000 M₅ = 0x00000000 M₆ = 0x3FFFFFFF M₇ = 0x00000000 M₈ = 0x00000000 M₉ = 0x00000000 M₁₀ = 0x00000000
Requête 2 : cas normal	M₀ = 0x00000000 M₁ = 0x00000000 M₂ = 0x00000000 M₃ = 0x00000000 M₄ = 0x00000000 M₅ = 0x00000000 M₆ = 0x00000000 M₇ = 0x000000FF M₈ = 0x0007FF80 M₉ = 0x00FC0000 M₁₀ = 0x00000000
Requête 3 : pire cas	M₀ = 0x00040000 M₁ = 0x00040000 M₂ = 0x00040000 M₃ = 0x00040000 M₄ = 0x00040000 M₅ = 0x00040000 M₆ = 0x00040000 M₇ = 0x00040000 M₈ = 0x00040000 M₉ = 0x00040000 M₁₀ = 0x00000000

FIGURE 6.5 – Marche de rayons en dimension deux. De haut en bas : En dimension deux, la voxélisation de la scène consiste à définir un tableau d'axe monde dont chaque ligne, stockée dans un pixel (ici un entier), représente un échantillonnage de la scène. Dans le cas optimal d'une requête de visibilité, on peut calculer le champ de bits associé au segment sur un seul axe, ici M_6 . Pour tester la visibilité, le même processus qu'en dimension un est réalisé en comparant ce masque avec la donnée correspondante, c'est-à-dire D_6 . Dans le cas normal, en revanche, le segment ne peut pas être représenté par une seul ligne du tableau. On construit alors une collection de masques correspondant aux intersections du segment avec les pixels de la voxélisation M_7 , M_8 , M_9 . Cette fois-ci, le test de visibilité est un peu plus long car il faut alors comparer tous les masques avec leur donnée correspondante, c'est-à-dire D_7 , D_8 , D_9 . Enfin dans le pire cas, la procédure est la même que dans le cas nominal sauf que cette fois-ci pour une faible longueur de segment l'intégralité des indices de 0 à 9 doivent être lus et testés pour répondre à la requête.

arbitraire de cette direction privilégiée et ainsi optimiser globalement la vitesse de réponse des requêtes de visibilité.

On note enfin que bien que notre exemple illustre le cas bidimensionnel, les problématiques de représentation et d'accès mémoire dans le cas tridimensionnel sont similaires. Nous parlerons des détails propres à la 3D dans la section qui suit.

6.5 Cartes d'HyperVoxels Sphériques

6.5.1 Description Générale

Dans ce paragraphe nous décrivons notre approche pour calculer les requêtes de visibilité. Notre solution travaille sur deux points majeurs, à savoir la structure de données servant à représenter la géométrie et le calcul de requête de visibilité. Ces deux points sont extrêmement dépendants l'un de l'autre car la façon de représenter la géométrie détermine également comment les requêtes peuvent y accéder. En outre, en fonction du type de calcul et du nombre d'opérations nécessaires, comme nous l'avons souligné précédemment, la rapidité d'exécution de la requête s'en voit affectée. Ainsi, à partir des limitations que nous avons observées plus tôt avec l'approche naïve de *marche de rayons*, nous proposons ici de construire un axe monde, *cf.* §6.4.1, pour chaque droite de la scène afin de l'échantillonner. De cette façon, une requête de visibilité entre deux points revient à trouver la droite passant par ces deux points, récupérer les données échantillonées le long de cette droite et ensuite appliquer l'algorithme unidimensionnel optimal que nous avons décrit précédent.

Cela nous demande alors de paramétriser un espace à 5 dimensions que sont l'ensemble des positions de l'espace 3D auxquels on ajoute une directions. En d'autres mots, l'ensemble des points orientés. Notre objectif ici est ainsi de faire en sorte qu'un ensemble de points avec la même orientation puisse être stocké en mémoire de manière contigüe. De cette façon on espère qu'une requête de visibilité accède aussi efficacement que possible à la mémoire. Afin de construire notre structure de données, nous subdivisons cet espace pentadimensionnel en cellules, que nous nommons alors *HyperVoxels Sphériques* ou *HVS*, selon lesquels nous échantillonnons la géométrie, *cf.* §6.5.4. Cependant, nos *HVS* sont définis dans un espace de trois dimensions spatiales et de deux dimensions correspondant à la direction et bien que les trois premières dimensions ne posent *a priori* pas de problème pour leur discréétisation, le fait de discréétiser l'ensemble des directions nécessite un peu d'attention.

Dans la section qui suit nous proposons une paramétrisation de l'espace des droites nous permettant alors de convenablement définir nos *HVS*. En ce sens, nous définissons cette espace de sorte à ce qu'il soit possible de construire efficacement l'échantillonnage de la scène étant donné les outils à notre disposition, c'est-à-dire l'unité de rastérisation, tout en permettant de l'interroger efficacement. On souhaite donc être en mesure de décrire de manière non redondante l'ensemble des droites. Réciproquement, étant donné deux points, c'est-à-dire les extrémités d'un segments de visibilité, on souhaite calculer efficacement la droite associée.

6.5.2 Paramétrisation de l'espace des droites

Nous définissons ici notre façon de paramétriser l'espace des droites de l'espace 3D. Pour cela, nous commençons par discuter du cas du plan où l'ensemble des droites est de dimension deux puis nous prolongeons notre solution à l'espace 3D où l'ensemble des droites est de dimension quatre.

Variété Les ensembles dont nous parlons dans la suite, l'ensemble des droites par exemple, ne forment plus nécessairement des espaces vectoriels. On ne peut alors plus se contenter de les décrire par des ensembles de vecteur sans plus de précisions. C'est pourquoi, on parle de k -variété, c'est-à-dire un ensemble qui, à l'aide d'une application bicontinue, fait correspondre tout point de cet ensemble à l'espace vectoriel \mathbb{R}^k , au moins localement. Ainsi, lorsque nous disons que l'ensemble des droites du plan (resp. de l'espace) est de dimension deux (resp. quatre), cela signifie que l'on peut y faire correspondre à tout point, un point de \mathbb{R}^2 (resp. \mathbb{R}^4). En outre, c'est uniquement en connaissant une telle application, ce que l'on se propose de faire ici, que l'on peut décrire une droite du plan (resp. de l'espace) à l'aide de deux (resp. quatre) scalaires.

Formalisation du problème Nous commençons par supposer que l'intégralité de la géométrie peut être contenue dans le disque unité ou la boule unité selon la dimension. Les droites que l'on cherche à décrire sont donc toutes celles pouvant être générées par deux points distinct appartenant à cet ensemble. Le problème que nous cherchons à étudier est donc double :

- étant donnée deux scalaires (resp. quatre scalaires) comment représenter une droite du plan (resp. de l'espace) ?
- étant donnée deux points distincts A et B du disque unité (resp. de la boule unité), comment calculer les deux scalaires (resp. quatre scalaires) définissant la droite à laquelle appartient le segment $[A, B]$?

Cercle, Sphère et Directions Pour décrire les droites, nous nous contentons dans la suite de donner comme paramètre une direction, un point sur le cercle ou un point sur la sphère selon la dimension. Bien entendu, en dimension deux (resp. trois) un tel paramètre doit donc être décrit par deux (resp. trois) scalaires. Cependant, nous exploitons le fait que ces points gisent en réalité sur une 1-variété (resp. 2-variété) pour dire qu'ils peuvent être décrit par un (resp. deux) scalaires. Par exemple, en dimension deux, on peut repérer un point du disque par l'angle par rapport à un axe arbitraire et en dimension trois, à l'aide des coordonnées sphériques. Cela étant dit, dans la section suivante, cf. §6.5.4, nous décrivons un mécanisme, moins couteux en calcul, que nous utilisons pour définir la projection d'un point de la sphère sur $[-1, 1]^2$, ainsi que sa réciproque. Nous considérons donc, qu'un point du disque (resp. de la sphère) est représenté par un scalaire (resp. deux scalaires).

Droites du plan Dans le plan, il existe de nombreuses manières de paramétriser l'ensemble des droites. À l'aide de la figure 6.6, nous donnons ici trois façons de répondre à notre problème.

- 1) La première solution que nous proposons, et qui semble être l'approche la plus simple, consiste à choisir deux points X^- et X^+ sur le cercle englobant, puis de considérer la droite qui passe par ces deux points. (X^-, X^+) pouvant alors être repérer par un angle chacun, on décrit

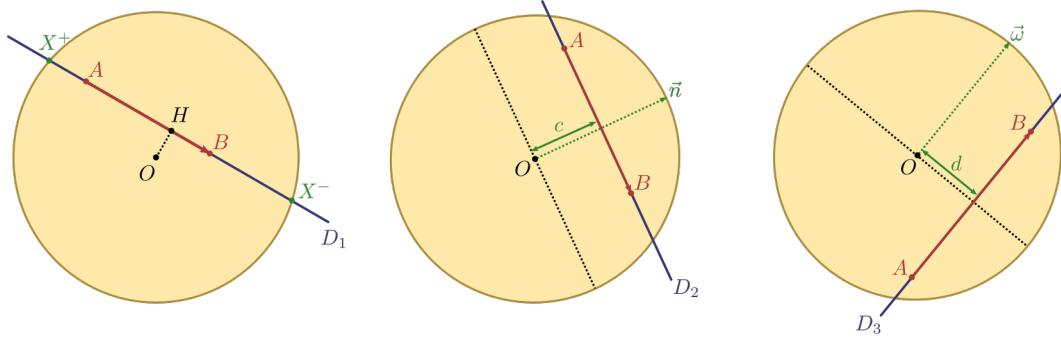


FIGURE 6.6 – Paramétrisations de l'ensemble des droites du plan. Les trois paramétrisations permettent de décrire l'ensemble des droites intersectant le disque unité. Dans les trois situations, ces droites sont décrites à l'aide de deux scalaires uniquement. Le problème inverse consiste à retrouver les paramètres de la droite aligné avec un segment $[A, B]$.

bien ici une 2-variété. Bien que simple à définir, cette solution présente quelques inconvénients. Tout d'abord, en supposant que l'on discrétise en sections égales le cercle, le centre du disque a tendance à être beaucoup plus échantillonné que la périphérie. Cela se trouve être gênant dans notre approche car, alors, il y a une disparité dans la qualité de la reconstruction de la géométrie. De plus, étant donné le segment $[a, b]$, le fait de déterminer les paramètres (X^-, X^+) définissant cette droite s'avère relativement coûteux en calcul. En effet, en notant H la projection orthogonale de O sur la droite, on a

$$X^\pm = H \pm \sqrt{1 - OH^2} \frac{\vec{AB}}{\|\vec{AB}\|}.$$

De plus, une telle paramétrisation permet difficilement de profiter d'accélération matérielle offerte par le mécanisme de rastérisation. En effet les droites proches lors de ce processus, c'est-à-dire les pixels voisins, se retrouvent éclatées en mémoire selon cette paramétrisation.

2) Une des limitations de la description précédente, est le manque de cohérence entre les droites dites proches. Une façon de résoudre ce problème consiste à regrouper les droites en fonction de leur direction. Pour cela, on commence par choisir une direction $\vec{n} = (a, b)$ et on trace la droite perpendiculaire à \vec{n} et qui est à une distance c de l'origine. Il s'agit de la paramétrisation cartésienne $ax + by - c = 0$. Cette paramétrisation ne privilégie *a priori* pas de zone de l'espace et en ce sens permet de mieux répartir les échantillons. Enfin pour cette paramétrisation, le calcul inverse est beaucoup plus simple ne faisant intervenir qu'une normalisation et un produit scalaire, c'est à dire :

$$\vec{n} = \frac{\vec{AB}_\perp}{\|\vec{AB}\|} \quad \text{et} \quad c = \langle \vec{n}, X \rangle,$$

où \vec{AB}_\perp est issue de la rotation de $\frac{\pi}{2}$ de \vec{AB} dans le sens trigonométrique qui se calcule trivialement dans le plan.

3) La dernière représentation que nous proposons ici est quasiment la même que la précédente. Pour cela, on choisit également une direction $\vec{\omega} = (b, -a)$ mais cette fois-ci, on choisit de tracer la droite parallèle qui est à une distance d de l'origine. Il s'agit alors de la droite définie par $ax + by + d = 0$. Pour le calcul inverse, on a alors :

$$\vec{\omega} = \frac{\vec{AB}}{\|\vec{AB}\|} \quad \text{et} \quad d = \det(\vec{\omega}, X).$$

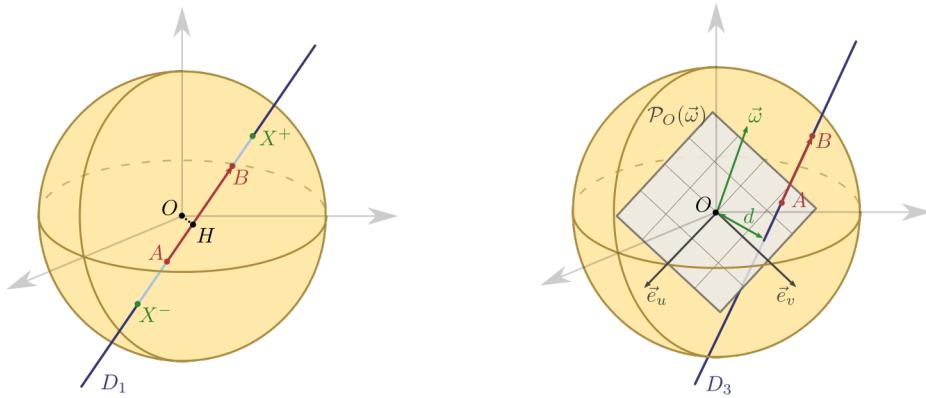


FIGURE 6.7 – Paramétrisations de l’ensemble des droites de l’espace. Les deux paramétrisations permettent de décrire l’ensemble des droites intersectant la sphère unité. Dans les deux situations, ces droites sont décrites à l’aide de quatre scalaires uniquement. Le problème inverse consiste à retrouver les paramètres de la droite alignée avec un segment $[A, B]$.

Bien que cette dernière ne présente presque aucune différence avec la stratégie précédente en terme de calcul et d’échantillonnage, nous allons voir que ce n’est plus le cas lors en dimension 3.

Droites de l’espace 3D Nous généralisons à présent la manière de décrire les droites en dimension trois à l’aide de la figure 6.7. Avant tout, il nous faut également généraliser les notions que nous utilisons. Ainsi, l’ensemble des directions qui s’identifiait au cercle unité est maintenant identifié à la sphère unité, c’est-à-dire que le fait de choisir une direction équivaut à choisir un point sur cette sphère.

1) La première stratégie que nous avons décrit s’étend trivialement en définissant une droite par deux points sur la sphère. Nous avons alors les mêmes calculs et les mêmes inconvénients que ceux énoncés plus tôt.

2) La seconde stratégie, quant à elle, ne permet pas de définir une droite lorsque l’on est dans l’espace. En effet, si on choisit une direction et que l’on considère l’ensemble des points qui lui sont perpendiculaire, on définit alors un plan. Il nous serait alors nécessaire de fournir un second plan non parallèle au premier pour pouvoir définir une droite.

3) La dernière approche, en revanche permet de décrire une droite en dimension trois. En effet, étant donné une direction $\vec{\omega}$ et le plan $\mathcal{P}_O(\vec{\omega})$ passant par l’origine et orthogonal à $\vec{\omega}$, on peut définir une unique droite parallèle à cette direction et qui intersecte le plan au point $d \in \mathcal{P}_O(\vec{\omega})$. La paramétrisation d’une droite revient ainsi à déterminer une direction et à fournir les deux coordonnées d’un point dans un plan. Cette stratégie est bien adaptée à notre problème car elle échantillonne convenablement l’ensemble des droites, dans le sens où aucune région de l’espace n’est sur-échantillonnée, et elle permet de paramétriser les droites de manière cohérente avec l’unité de rastérisation, cf. §6.6.1. De plus, le calcul inverse des paramètres de la droite à partir du segment de visibilité est relativement simple. En effet, comme dans le cas bidimensionnel $\vec{\omega}$ s’obtient en normalisant le vecteur \vec{AB} et les coordonnées du déplacement \vec{d} s’obtiennent en projetant l’un des

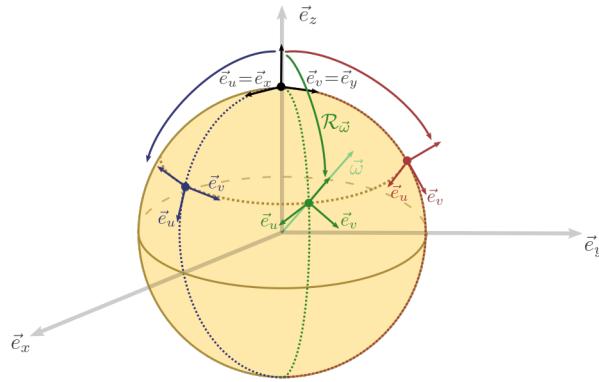


FIGURE 6.8 – Paramétrisation de l'espace tangent à la sphère, continue presque partout. En tout point $\vec{\omega}$ différent des pôles, on définit une unique rotation $\mathcal{R}_{\vec{\omega}}$ qui envoie le pôle nord vers ce point et qui est perpendiculaire au méridien passant par ces deux points. La base du plan tangent est issue de la transformation de la base canonique au pôle nord par la même rotation. Cette transformation n'est pas définie au pôle sud car il existe alors une infinité de méridiens.

sommets du segment sur le plan $\mathcal{P}_O(\vec{\omega})$. Si l'on en possède une base orthonormée (\vec{e}_u, \vec{e}_v) (ω), cette projection correspond à deux simples produits scalaires.

$$\vec{\omega} = \frac{\vec{AB}}{\|\vec{AB}\|} \quad \text{et} \quad \vec{d} = \begin{pmatrix} \langle \vec{e}_u(\vec{\omega}), \vec{OA} \rangle \\ \langle \vec{e}_v(\vec{\omega}), \vec{OA} \rangle \end{pmatrix}. \quad (6.1)$$

Pour complètement définir notre paramétrisation *via* cette stratégie il ne nous reste plus qu'à définir une telle base, ce que nous proposons dans le paragraphe suivant.

6.5.3 Repère tangent à la sphère

Tel que nous l'avons défini ci-dessus, $\mathcal{P}_O(\vec{\omega})$ est le plan passant par l'origine (O) et de normal $\vec{\omega}$. Cependant, définir ce plan revient à définir un plan tangent à la surface de la sphère, or il n'existe pas de façon triviale de définir une base pour un tel objet. Nous cherchons donc ici une expression pour les vecteurs d'une base orthonormée continue sur toute la surface de la sphère et qui soit simple à calculer. Le caractère continue de cette paramétrisation est important car, comme nous cherchons à discréteriser l'ensemble des paramètres en cellules, *cf.* §6.5.4, on souhaite que cela se traduise par le fait que chaque cellule corresponde à un ensemble de lignes proches les unes des autres. Malheureusement, le théorème de la boule chevelue, aussi appelé théorème du point fixe de Brouwer [Brouwer, 1911, Mawhin, 2007], montre que la condition de continuité ne peut pas être vérifiée sur l'ensemble de la sphère. Cependant, comme dans notre cas nous cherchons à décrire l'ensemble des droites, il n'est pas nécessaire de parcourir l'ensemble de toutes les directions. En effet, si $\vec{\omega}$ est un vecteur directeur de la droite alors $-\vec{\omega}$ en est un également. De cette façon, nous pouvons nous contenter de fournir une paramétrisation de l'hémisphère supérieur, pour qui la base que nous cherchons peut être construite.

La base que nous proposons, *cf.* figure 6.8, peut être définie de la façon suivante. Au pôle nord, c'est-à-dire $\vec{\omega} = \vec{e}_z$, la base s'identifie à la base canonique (\vec{e}_x, \vec{e}_y) . Par ailleurs, pour toute direction $\vec{\omega}$ différente du pôle nord et du pôle sud, on considère l'unique rotation $\mathcal{R}_{\vec{\omega}}$ dont l'axe est orthogonal à l'unique méridien passant par $\vec{\omega}$ et \vec{e}_z et qui envoie \vec{e}_z sur $\vec{\omega}$. On définit alors la base

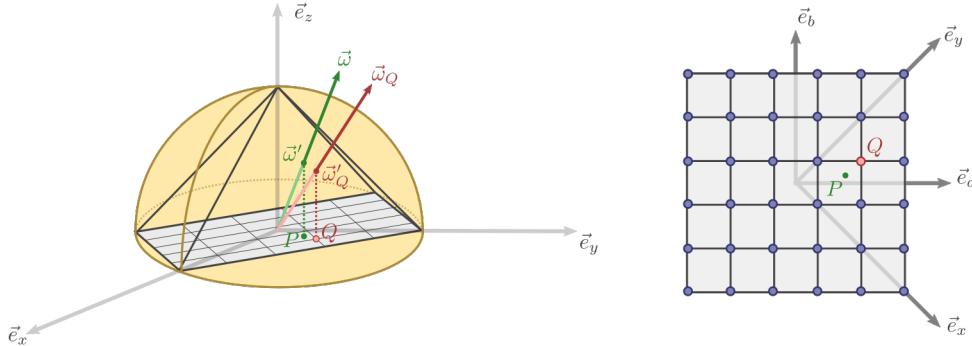


FIGURE 6.9 – Paramétrisation et quantification de l’hémisphère par projection octaédrique. À toute direction $\vec{\omega}$, on associe son intersection avec la partie supérieure de l’octaèdre, $\vec{\omega}'$ qui est alors projeté sur le plan horizontal pour définir le point P . Par cette projection, l’ensemble des directions décrit alors un carré que l’on peut quadriller (à droite). Parmi l’ensemble des nœuds de ce quadrillage, on trouve le point Q qui est le plus proche de P . Par le processus inverse, ce point correspond alors à une direction $\vec{\omega}_Q$ qui correspond à la direction quantifiée la plus proche de $\vec{\omega}$.

tangente à la sphère en $\vec{\omega}$, (\vec{e}_u, \vec{e}_v) ($\vec{\omega}$) comme l’image par $\mathcal{R}_{\vec{\omega}}$ de la base orthonormée (\vec{e}_x, \vec{e}_y) . On montre alors qu’il est possible de calculer ces vecteur grâce à quelques opérations polynomiales, directement à partir des coordonnées de $\vec{\omega}$:

$$\begin{aligned} \forall \vec{\omega} = (x, y, z) \text{ tel que } x^2 + y^2 + z^2 = 1 \text{ et } z \neq -1, \\ (\vec{e}_u, \vec{e}_v)(\vec{\omega}) = \left(\begin{pmatrix} 1 - \frac{x^2}{1+z} \\ -\frac{xy}{1+z} \\ -x \end{pmatrix}, \begin{pmatrix} -\frac{xy}{1+z} \\ 1 - \frac{y^2}{1+z} \\ -y \end{pmatrix} \right). \end{aligned} \quad (6.2)$$

Comme nous l’avons dit plus tôt, il n’est pas possible de construire de bases variant continûment et paramétrant les plans tangents de la sphère. Dans notre cas, comme nous appliquons une symétrie par rapport à l’équateur, nous avons une discontinuité des paramètres à cette endroit. Pour limiter les artefacts liés à cette discontinuité, nous prenons soin, dans la suite, de discréteriser la sphère des directions au niveau de l’équateur.

6.5.4 HyperVoxel Sphérique

La paramétrisation de l’espace des droites \mathcal{D}_3 , que nous venons de définir, nous permet de transformer le problème de discréterisation d’un espace compliqué en la discréterisation d’un espace plus simple : au lieu de quantifier les droites directement, on se propose de quantifier l’espace de ses paramètres, c’est-à-dire $\mathcal{S}_2 \times [-1, 1]^2$. En effet, nous avons besoin d’une direction $\vec{\omega} \in \mathcal{S}_2$ ainsi que de le déplacement $d \in [-1, 1]^2$ par rapport à l’origine dans le plan orthogonal. Ce dernier paramètre peut effectivement être borné au carré unité car nous avons supposé que l’intégralité de nos segments de visibilité sont inclus dans la sphère unité. À présent, il nous faut donc construire une discréterisation de la sphère unitaire ainsi que du carré unitaire.

Quantification de \mathcal{S}_2 Dans un premier temps il nous faut discréteriser le premier paramètre, à savoir $\vec{\omega} \in \mathcal{S}_2$. La problématique de discréterisation, aussi nommée quantification de la sphère, a été longuement étudiée [Cigolle et coll., 2014] et de nombreuses problématiques ont été identifiées et

résolues. Parmi celles-ci, nous avons besoin de quantifier la sphère de la façon la plus uniforme possible. Il nous faut également pouvoir inverser la quantification, c'est-à-dire trouver l'échantillon le plus proche pour une direction donnée, de la façon la plus efficace possible.

Au vu de nos contraintes, nous choisissons d'utiliser la paramétrisation octaédrique pour discréteriser notre ensemble de directions [Meyer et coll., 2010]. De plus, comme nous ne cherchons qu'à décrire l'hémisphère supérieur \mathcal{H} , cette paramétrisation nous permet de faire simplement correspondre l'ensemble des directions avec un carré unitaire, cf. figure 6.9.

$$\forall \vec{\omega} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathcal{H}, \quad P(\vec{\omega}) = \begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{|x| + |y| + |z|} \begin{pmatrix} x - y \\ x + y \end{pmatrix}, \quad (6.3)$$

et réciproquement,

$$\begin{aligned} \forall P = \begin{pmatrix} a \\ b \end{pmatrix} \in [-1, 1]^2, \quad \vec{\omega}'_P &= \begin{pmatrix} \frac{a+b}{2} \\ \frac{b-a}{2} \\ 1 - \max(|a|, |b|) \end{pmatrix} \\ \vec{\omega}_P &= \frac{\vec{\omega}'_P}{\|\vec{\omega}'_P\|}. \end{aligned} \quad (6.4)$$

De cette façon, pour construire notre discréétisation, on choisit une grille régulière sur le carré unité et pour chaque nœud de cette grille on discréteise la direction correspondante. Cela nous apporte deux avantages. Tout d'abord, cela rend trivial le fait de trouver la direction quantifiée la plus proche. En effet comme la projection octaédrique permet de construire une bijection de l'hémisphère sur ce carré, à chaque direction correspond un point dans le carré. Le calcul des coordonnées du nœud le plus proche dans la grille peut donc être classiquement réalisé à l'aide des fonctions partie entière. Bien qu'en toute rigueur ce calcul ne corresponde pas à la direction discréétisée la plus proche en terme de distance angulaire, il permet néanmoins d'être extrêmement proche tout en étant très simple à calculer. Un second avantage qu'offre cette paramétrisation est qu'elle permet de discréteriser l'équateur. Ainsi, lors du calcul de proximité que nous venons de décrire, comme les bords de chaque cellule de la grille sont composés de nœuds appartenant au même hémisphère, la discontinuité que nous avons introduite dans notre paramétrisation, cf. formule 6.2, ne nécessite alors plus de traitement particulier pour les directions se situant au bord de l'équateur. Cela implique néanmoins que les droites proches de l'équateur sont deux fois plus échantillonnées que les autres. On perd donc ici un peu de mémoire pour conserver une procédure d'échantillonnage simple.

Pavage de l'espace des points orientés Un point orienté est un couple $(\vec{\omega}, x) \in \mathcal{S}_2 \times \mathbb{R}^3$. De plus, grâce à notre quantification de la sphère via la projection octaédrique, on peut définir un pavage de l'ensemble des directions. Cela signifie que pour toute valeur $\vec{\omega} \in \mathcal{S}_2$, on est capable de lui associer un unique $\vec{\omega}_{Q_i} \in \mathcal{S}_2$ qui est la direction quantifiée d'indice i , la plus plus proche de $\vec{\omega}$, cf. figure 6.9.

Par ailleurs, grâce à notre définition d'un espace tangent muni d'une base orthonormée, cf. formule 6.2, on définit une grille régulière alignée sur cette base nous permettant alors de quantifier également l'ensemble des positions. Une cellule dans cette grille est ainsi donnée à partir de cinq coordonnées, deux pour la direction et trois pour la position. Pour une direction donnée, le calcul de proximité dans la grille régulière associée, c'est-à-dire le calcul de l'indice de la cellule

la contenant x , est un calcul classique à base de partie entière semblable à ce que nous venons de décrire pour la quantification de la sphère. De cette façon nous venons de complètement décrire un pavage de l'espace des points orientés de sorte à ce que pour tout élément $(\vec{\omega}, x) \in \mathcal{S}_2 \times \mathbb{R}^3$, il existe une unique cellule le contenant. Nous nommons ces cellules des *Hyper Voxels Sphériques* (HVS).

Nous aurions pu définir un HVS en commençant par quadriller l'espace 3D et ensuite associer à chaque case une discréétisation de l'ensemble des directions. Une telle cartographie permet également de construire un pavage de l'espace des points orientés, en notant tout de même que ces deux pavages ne se confondent pas. Cependant, nous souhaitons utiliser cette structure pour nos requêtes de visibilité en accédant aux HVS alignés, de manière linéaire. Or, le fait de séparer les directions dans plusieurs cellules différentes est incompatible avec cette idée. Ainsi, la première stratégie visant à regrouper les cellules alignées s'avère être beaucoup plus adaptée. Cela permet, de plus, à l'unité de rastérisation d'en tirer profit comme nous le décrivons ci-dessous.

6.6 Algorithme

Nous décrivons ici le pipeline de notre algorithme qui se divise en deux parties. Dans un premier temps nous construisons la structure de données à partir de la géométrie de la scène. À la manière d'une carte d'ombre, cette première étape est réalisée au début de la trame avant la phase de rendu principale. La structure de données est ensuite interrogée dans un second temps lors de la phase d'éclairage de la scène. Cette étape est la plus critique car il y est nécessaire d'effectuer plusieurs dizaines, voire centaines, de millions de requêtes de visibilité. La nature de ces requêtes influe donc grandement sur l'architecture de la structure de données afin que les accès soient rendus les plus optimaux possibles.

6.6.1 Construction des cartes

La construction des cartes d'HVS se décompose en deux parties principales. Tout d'abord, nous définissons l'ensemble des directions quantifiées. Dans un second temps, l'unité de rastérisation est utilisée pour remplir les grilles de voxels associées à ces directions, *cf.* figure 6.10.

Définition des cartes Afin de conserver un impact mémoire convenable, nous proposons de subdiviser la grille des directions en $K \times K$ cases régulières, *cf.* figure 6.9. Nous utilisons en pratique $K = 32$ et analysons l'impact de ce choix dans les résultats, *cf.* §6.7. Pour chacune de ces K^2 directions $(\vec{\omega}_{Q_i})_{i \in [\![1, K^2]\!]}$ nous pré-calculons alors le tableau contenant les bases associées dans une zone mémoire du processeur graphique, *cf.* formule 6.2. De même, pour stocker les grilles de voxels, nous utilisons un unique tableau de texture 2D de $N \times N$ pixels représentés par des mots entiers de N bits. En choisissant alors $N = 128$, on se permet de représenter chaque pixel par 4 entiers de 32 bits pour une consommation mémoire totale de $N^3 \times K^2 = 256$ Mo. A titre de comparaisons, cette mémoire représente également ce que nécessite une carte d'ombre de taille 8192×8192 enregistrée sur des nombres flottants. Nous sommes ainsi tout à fait comparable à ce que les processeurs graphiques modernes sont en mesure de gérer. Selon nos observations, *cf.* §6.7, ce choix de discréétisation permet un bon compromis entre la consommation mémoire et la précision. De plus, utiliser d'avantage de mémoire nécessiterait plus de calculs et donc plus de temps pour construire les cartes.

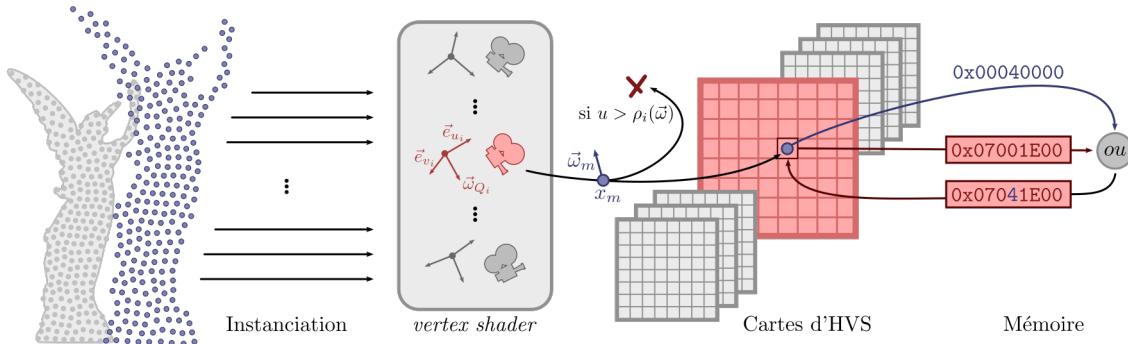


FIGURE 6.10 – Construction des cartes d’Hyper Voxels Sphériques. La scène est représentée sous forme d’un nuage de points orientés ($x_m, \vec{\omega}_m$). Ceux-ci sont dessinés en autant d’instances que de directions discrétisées. Au *Vertex Shader*, le numéro d’instance i du sommet permet de déterminer à quelle direction $\vec{\omega}_{Q_i}$ il est associé. Le repère correspondant ($\vec{e}_{v_i}, \vec{e}_{u_i}, \vec{\omega}_{Q_i}$) est calculé à partir de cet indice, et le sommet est projeté dans ce repère. Parallèlement, si l’angle entre la direction $\vec{\omega}_m$ du sommet orienté et la direction discrétisée $\vec{\omega}_{Q_i}$ est trop important, le point est défaussé avant le processus de rastérisation. Dans le cas contraire, le *Fragment Shader* correspondant à la projection du point dans la bonne carte calcule le masque binaire. Ce dernier est combiné de manière atomique par l’opérateur *ou* avec la donnée précédemment enregistrée dans ce pixel.

Remplissage des cartes Chaque carte à proprement parlé peut être construite par une approche classique de voxélisation de géométrie, cf. §6.3. Cependant nous souhaitons ici en construire 1024 en une seule fois sur des grilles de faible résolution. Il ne nous est donc pas nécessaire de travailler directement sur les triangles de la géométrie, et nous proposons d’utiliser à la place une représentation par nuage de points orientés de la scène. Nous travaillons en général sur des scènes de quelques centaines de milliers de points, ce qui nous permet de capturer l’essentiel de la géométrie.

Ensuite, le remplissage des cartes s’effectue grâce au pipeline classique de rendu en dessinant autant d’instances du nuage de points que de cartes à remplir. Au *Vertex Shader*, l’indice de l’instance nous informe sur la direction qui est en train d’être construite. En cherchant alors dans le tableau des directions pré-calculées, le repère correspondant, on calcule les coordonnées des points à dessiner dans ce repère. Par ailleurs, comme les cartes sont stockées sous forme de tableau de textures, le choix de la carte dans laquelle rendre peut également être réalisé au *Vertex Shader*. De cette façon, on définit une implémentation simple de la construction de nos cartes au niveau du traitement de la géométrie.

Au niveau du *Fragment Shader*, on utilise la profondeur calculée au *Vertex Shader* que l’on discréte sur l’une des 128 valeurs possibles. En calculant la valeur $z \in [0, 128]$, on construit la couleur entière du pixel comme étant égale à 2^z . En réalité, comme la couleur est composée de 4 entier, il nous faut décomposer ce nombre en 4 séquences de 32 bits. Enfin la valeur du pixel est écrite dans la texture correspondante, mais l’écriture ne se fait pas en écrasant l’ancienne valeur. Pour cela nous activons au préalable la fonctionnalité de mélange par l’opérateur logique OU. Ainsi la valeur finale enregistrée dans un pixel correspond au OU bit à bit de tous les fragments qui y ont écrit.

Décimation des points non pertinents Bien que l’utilisation de nuage de points, à la place des triangles, permette de fortement réduire à la fois la complexité et le temps d’exécution de cette

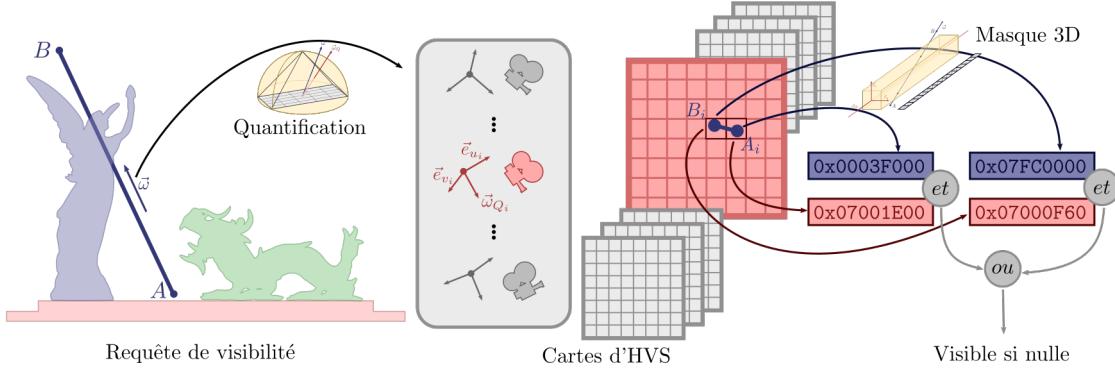


FIGURE 6.11 – Évaluation d'une requête de visibilité. Le segment de visibilité est défini par les deux points A et B . La direction correspondante $\vec{\omega}$ est alors quantifiée à l'aide de la projection octaédrique pour retrouver l'indice i dans la carte d'HVS. Dans cette carte on itère sur tous les pixels traversés par le segment projeté $[A_i, B_i]$ qui peuvent être calculés par une rasterisation conservative. À chaque pixels, le masque de bits du segment $[A, B]$ est calculé et comparé avec la donnée stockée dans la carte. Le résultat du test donne les deux points visible si et seulement si toutes les comparaisons donnent la valeur 0.

étape, le temps de construction des cartes demeure important. Le principal problème venant du nombre de transactions d'écriture dans les textures, nous cherchons à réduire la quantité de points à dessiner. Pour cela, on propose de défausser une partie des points à dessiner directement au niveau du *Vertex Shader*. En effet, on se rend compte que pour chaque direction quantifiée seul un petit ensemble de directions proches servent de support au segment de visibilité. En cela, les points orientés de la géométrie qui sont perpendiculaires à cette direction ont *a priori* peu d'importance tandis que les points orienté parallèlement à la direction ont de forte chance d'intersecter de tels segments. Ainsi, sans pour autant perdre en information, pour chaque point orienté par $\vec{\omega}$ on propose de le défausser de manière non déterministe, si une variable aléatoire u est supérieure à $\rho_i(\vec{\omega})$ où :

$$\rho_i(\vec{\omega}) = |\langle \vec{\omega}, \vec{\omega}_{Q_i} \rangle|. \quad (6.5)$$

Pour construire la variable aléatoire, nous utilisons alors une fonction de hachage sur l'indice du sommet et son numéro d'instance. Si l'on suppose que u est générée de manière uniforme, cette procédure élimine la moitié des points en espérance. Nous n'observons cependant qu'un tiers de gain au niveau du temps de construction car le temps passer à traiter les sommets au niveau du *Vertex Shader* n'est alors pas réduit. Une autre stratégie pour associer les points aux cartes pourrait alors être envisagée mais nous laissons cela à de futures recherches.

6.6.2 Requêtes optimales arbitraire

Une fois que les cartes sont construites, il est possible de s'en servir lors du rendu pour calculer la visibilité entre n'importe quel points de l'espace. Notre structure de données est suffisamment générique pour être interrogée en parallèle sans préparation particulière une fois que la carte est construite. L'algorithme que nous décrivons ici, permet de réaliser n'importe quelle requête de visibilité, par exemple pour calculer les ombres, cf. figure 6.11.

Ainsi, au *Fragment Shader*, lors du calcul de l'éclairage nous construisons le segment de visibilité entre la position du pixel A et l'une des sources lumineuses B . À l'aide de notre quantification des

directions, *cf.* §6.3, nous retrouvons la direction $\vec{\omega}_{Q_i}$ la plus proche de celle du segment ainsi que l'indice associé. On obtient ainsi le repère local et la carte HVS correspondante.

Il ne nous reste plus qu'à interroger la grille de voxels pour tester la visibilité. Pour cela, on interprète chaque pixel comme une boîte de longueur infinie selon la direction $\vec{\omega}_{Q_i}$ et dont les bits correspondent à des cubes. Trouver l'ensemble des cubes traversés par le segment consiste alors à, dans un premier temps, déterminer les boîtes qu'elle traverse, et dans un second temps, trouver les points d'intersection entre le segment et la boîte. Connaître l'ensemble des boîtes consiste à déterminer les pixels de la carte d'HVS, et connaître la position d'entrée et sortie du segment permet de construire le masque binaire du segment pour ce pixel, *cf.* figure 6.12. Une fois que ces deux informations sont connues, il ne nous reste plus qu'à combiner, par l'opérateur logique *et*, la donnée enregistrée par ces pixels et le masque ainsi calculé. Si, pour au moins un de ces pixels, le résultat de cette opération est différent de zéro, alors les deux points ne sont pas mutuellement visible, dans l'autre cas, on considère qu'ils le sont.

Par ailleurs, le fait de déterminer les boîtes traversées par le segment $[A, B]$ équivaut à calculer les pixels issus d'une rastérisation conservative de ce segment projeté dans le plan de la texture. Ainsi, pour itérer sur l'ensemble des pixels touchés, on réalise une boucle dans le *Shader* qui implémente la variante conservatrice de l'algorithme de Bresenham [Bresenham, 1965, Amanatides et coll., 1987]. Cette dernière étape est nécessaire car, faute de mémoire, nous ne pouvons pas discréteriser une infinité de directions pour construire nos cartes. Cependant, en choisissant la direction la plus alignée avec $[A, B]$, le segment projeté sur la texture est en pratique de très petite taille. Le nombre de pixels que l'on doit examiner est de ce fait très restreint, deux à trois en général. En outre, le coût engendré par le calcul du masque de bits de chaque pixel est également largement amorti par le fait qu'il représente un grand nombre de voxels dans la direction $\vec{\omega}_{Q_i}$. De cette façon nous transformons la rastérisation conservative d'une ligne tridimensionnelle de longueur quelconque en une rastérisation d'une ligne bidimensionnelle de longueur minimale. Nous montrons par la suite l'impact de cette approche sur la vitesse d'exécution, *cf.* §6.7.

6.6.3 Auto-occlusion

Bien que nos cartes puissent être utilisées pour calculer la visibilité entre deux points quelconques, la résolution d'échantillonnage demeure en pratique trop faible pour le permettre de manière très précise. De plus, de même que pour les cartes d'ombres classique, s'il l'on s'en fie uniquement à la géométrie reconstruite par les HVS, l'intégralité des points de la surface des triangles se situe strictement à l'intérieur d'un HVS. Cela signifie que les requêtes de visibilité se soldent toutes par un résultat négatif. Pour palier ce problème, il est classique de déplacer les bornes du segment en les décalant dans le sens de la normale à la géométrie, \vec{n}_A . Dans notre cas, nous proposons de décaler également dans la direction quantifiée $\vec{\omega}_{Q_i}$. Ce décalage est ainsi fait pour garantir que les bornes du segment soient en dehors du voxel contenant la géométrie, *cf.* figure 6.13. En se donnant $N \times N \times N$, la résolution de la grille, et $R_{\text{scène}}$ le rayon de la scène, on définit notre segment décalé $[A', B']$ par :

$$\begin{aligned} A' &= A + R_{\text{scène}} \left(\frac{\sqrt{3}}{N} \vec{n}_A - \frac{\lambda_A}{N} \vec{\omega}_{Q_i} \right), \\ B' &= B + R_{\text{scène}} \left(\frac{\sqrt{3}}{N} \vec{n}_B - \frac{\lambda_B}{N} \vec{\omega}_{Q_i} \right), \end{aligned} \quad (6.6)$$

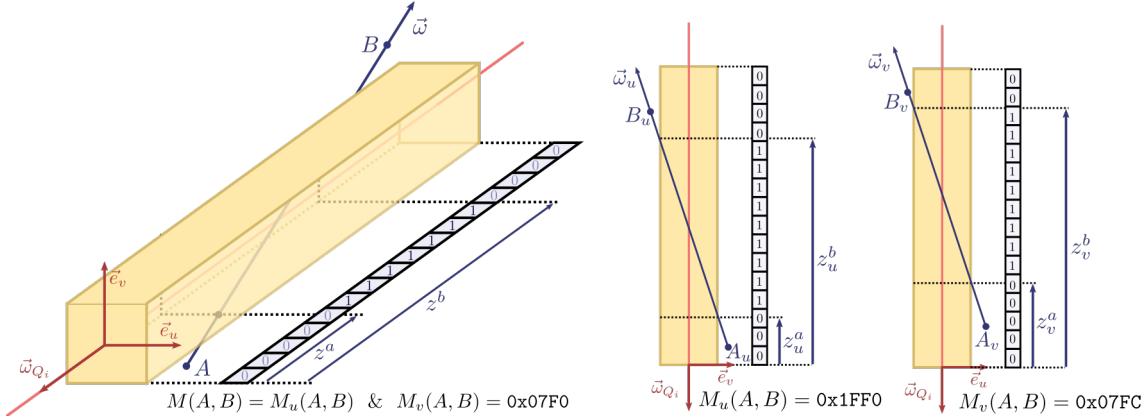


FIGURE 6.12 – Calcul du masque 3D. Chaque pixel d'une carte d'HVS est interprété comme une boîte de longueur infinie dans la direction $\vec{\omega}_{Q_i}$ et chaque bit de ce pixel correspond à une section. Le masque de bits du segment représente l'ensemble des sections traversées par la ligne. Pour cela le problème consiste à trouver les distances z^a et z^b représentant respectivement la profondeur d'entrée et de sortie du segment de la boîte. En quantifiant ici ces valeurs sur un entier compris entre 0 et $K - 1$, on reconstruit le masque à l'aide d'arithmétique sur les entiers. Le problème peut alors se décomposer en deux problèmes (à droite) de dimension deux et en construisant le masque 3D comme la conjonction par l'opérateur logique *ET* de ces deux masques 2D.

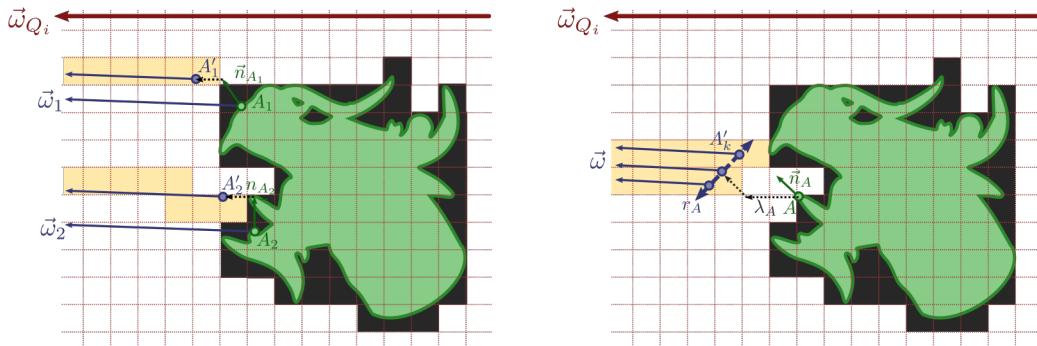


FIGURE 6.13 – Suppression de l'auto-occlusion et filtrage. À gauche, lorsque l'on réalise un test de visibilité depuis la surface, le point A_1 se situe nécessairement à l'intérieur d'un voxel plein. Pour garantir que l'origine du segment se situe en dehors de ce voxel on le déplace de la longueur d'une diagonale du voxel dans la direction de la normale. Cependant, ce mécanisme ne suffit pas toujours à construire un point en dehors de la géométrie, comme dans le cas du point A_2 où il demeure de la géométrie dans son voisinage dans la direction normale. Pour grandement réduire ce phénomène, on décale également dans la direction du rayon lumineux d'une longueur de un voxel.

À droite, on cherche à calculer une visibilité filtrée autour d'un disque de rayon r_A . Encore une fois, pour grandement limiter la probabilité que ce disque intersecte sa propre géométrie, on décale ce disque dans la direction du rayon lumineux, proportionnellement au rayon du disque.

en supposant que \vec{AB} et $\vec{\omega}_{Q_i}$ sont orientés dans le même sens, sinon on utilise $-\vec{\omega}_{Q_i}$ dans cette formule. Les paramètres λ_A et λ_B sont des facteurs d'échelle que nous laissons égaux à 1 pour le test de visibilité arbitraire.

6.6.4 Filtrage par pourcentage de visibilité

Encore une fois, puisque notre échantillonnage de la scène est relativement grossier, il nous est impossible de capturer des variations fines de l’ombrage. En revanche, nos requêtes étant rapides à calculer, nous proposons ici de calculer non plus une visibilité binaire mais un pourcentage de visibilité sur un substitut. Pour cela nous considérons un disque imaginaire autour de l’origine A du segment que nous échantillonons en plusieurs points. Pour déterminer ces points nous pré-calculons un échantillonnage de Poisson du disque unité [Cook, 1986]. Nous utilisons entre 4 et 8 échantillons. Ces points sont alors positionnés autour de A en construisant le repère tangent à \vec{n}_A à l’aide de la formule cité plus haut, *cf.* formule 6.2. On se propose alors de définir le rayon r_A du disque proportionnellement à la taille d’un pixel par

$$r_A = \frac{\lambda_A}{N}. \quad (6.7)$$

Dans cette situation, dans le but de limiter l’auto-occlusion avec la géométrie voisine, on se propose de déplacer la position du disque dans la direction quantifiée $\vec{\omega}_{Q_i}$, proportionnellement à λ_A tel que décrit dans la formule au paragraphe précédent, *cf.* formule 6.6. Ce choix est motivé par la raison suivante. Le fait de calculer la visibilité sur tout un disque signifie que l’on cherche à gommer l’influence des détails de la géométrie d’une taille inférieure à r_A . Ainsi, en décalant dans la direction de la visibilité, nous observons que l’on réduit alors l’influence de ces détails, *cf.* figure 6.13.

Enfin, pour calculer la visibilité, on compte alors le pourcentage de rayons qui passent le test binaire parmi ceux que nous venons de définir. Ce pourcentage définit alors notre valeur de visibilité comprise entre 0 et 1. De manière analogue, on se propose de construire un disque de récepteurs autour de la destination B . Le choix des facteurs λ_A et λ_B influent sur la performance de notre algorithme car si de grandes valeurs sont choisies, les rayons deviennent incohérents et risquent de nécessiter l’accès à des cartes différentes. Nous utilisons en pratique des λ variant en 1 et 4.

6.7 Résultats

Nous avons implémenté notre algorithme à l’aide de l’interface de programmation *OpenGL 4.5*. Nos mesures sont effectuées sur un ordinateur équipé d’un processeur graphique *Quadro P6000*. Nous présentons ici les résultats que nous avons obtenus sur des scénarios nécessitant le calcul de la visibilité. Les figures 6.14 et 6.15 illustrent les scènes en question.

Performances La table 6.1 reporte les différents temps des étapes de construction de notre algorithme. Celui-ci se décompose ainsi en deux grandes étapes. La première consiste à produire la structure d’accélération à partir d’un nuage de points orientés. Il est intéressant de noter que, de cette façon, le temps de construction de cette structure ne dépend pas de la complexité de la scène mais uniquement du nombre de points. Grâce à cela, nous sommes en mesure de construire nos cartes sur des modèles pouvant atteindre plus de cent millions de triangles en temps interactif. À notre connaissance, il n’y a qu’en utilisant le pipeline graphique et l’unité de rastérisation que l’on peut traiter de telles quantités de triangles avec ces performances tout en conservant une empreinte mémoire faible. La seconde étape correspond au calcul des requêtes de visibilité. Pour l’évaluer, nous avons travaillé sur différents scénarios.

	Veach	Sponza	7 statues de Lucy	
	éclairage direct	éclairage surfacique	occlusion ambiante	environnement infini
Tampon géométrique	< 1 ms	< 1 ms	50 ms	
# triangles	1K	150K	100M	
Construction des cartes	45 ms	56 ms	770 ms	100 ms
# points	100K	100K	800K	200K
Test de visibilité	20 ms	20 ms	140 ms	140 ms
# requêtes	44M	44M	512M	512M
Tout dynamique	60 ms	60 ms	1020 ms	300 ms
Semi dynamique	20 ms	20 ms	190 ms	190 ms

TABLE 6.1 – Temps de construction des différentes étapes du pipeline. Les images sont produites à la résolution de 1920×1080 pixels. Les cartes d’HVS échantillonnent 1024 directions sur des grilles de 128^3 voxels. *Tout dynamique* signifie que l’intégralité de la scène (géométries, lumières, point de vue) sont dynamiques. *Semi dynamique* signifie que la géométrie est fixe.



FIGURE 6.14 – Calcul de l’éclairage direct. À gauche, la scène de *Veach* contenant une unique source ponctuelle. À droite, la scène *Sponza* composée de trois sources sphériques dynamiques.

Éclairage Direct Dans un premier temps, nous évaluons notre approche pour le calcul de l’éclairage direct, cf. figure 6.14. Il est intéressant d’observer comment se comporte notre algorithme en présence d’une source ponctuelle, bien que ce scénario ne soit pas optimal pour notre technique (les cartes d’ombre, par exemple, permettent d’obtenir de bien meilleurs résultats). Comme on pouvait s’y attendre, les ombres qui sont générées par notre algorithme sont nécessairement douces. Cela vient du fait qu’il est nécessaire d’envoyer un grand nombre de rayons pour que la représentation grossière des cartes d’HVS ne génère pas d’artefact visible. En effet, lorsque les sources sont ponctuelles, la forme de la géométrie stockée dans la structure de données se projette très distinctement sur la scène.

Éclairage direct dynamique Le second scénario que nous avons testé consiste à positionner plusieurs sources lumineuses surfaciques dynamiques dans la scène *Sponza*. Dans cette situation, il est intéressant d’observer qu’une fois que nos cartes sont construites, la visibilité peut être calculée depuis n’importe où dans la scène. Cela nous permet ainsi de s’adapter naturellement à n’importe quel scénario dont l’environnement lumineux est dynamique. On note néanmoins une limitation

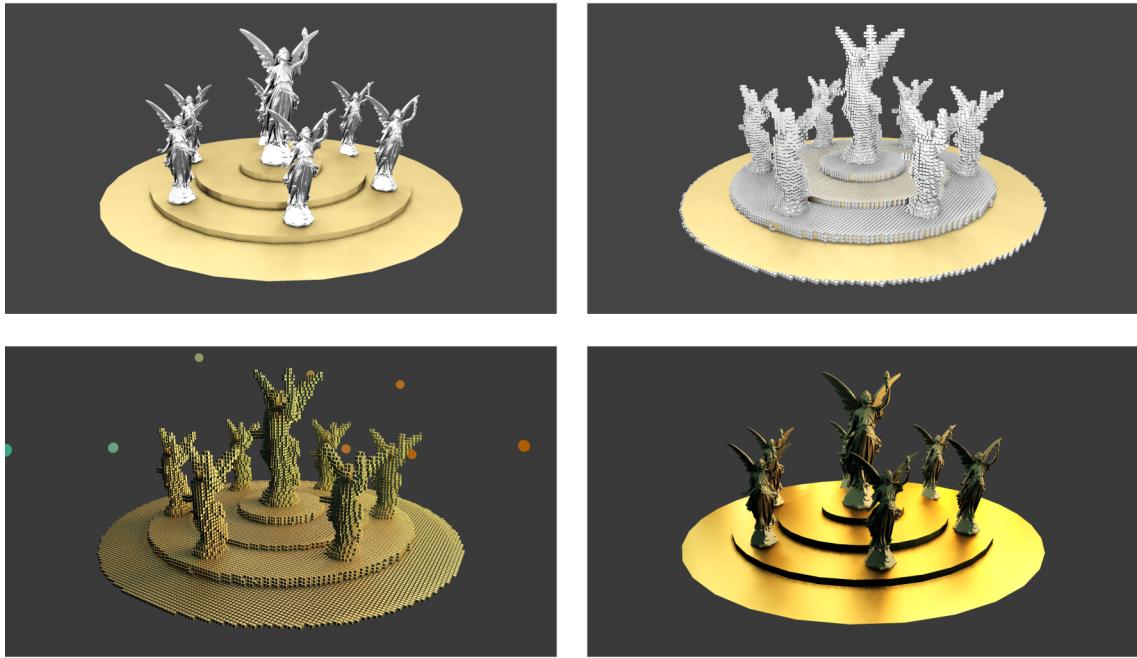


FIGURE 6.15 – Calcul de l’occlusion ambiante (en haut) et de la visibilité sur un environnement lumineux infini (en bas). Le matériau réfléchissant, utilisé pour le podium (en bas), permet de visualiser l’ombre spéculaire des statues, capturée par notre structure. En haut à droite et en bas à gauche, une des multiples voxélisations des cartes d’HVS. La scène, composée de 7 statues de Lucy, contient 100 millions de triangles et est visualisée en temps interactif (190 ms par trame).

de notre approche lorsqu’une source de forte intensité se trouve près d’un obstacle. À ce moment, la structure grossière des cartes d’HVS devient visible. Il s’agit, néanmoins là, d’un phénomène local pouvant être traité par d’autres approches. Nous pensons, en revanche, que notre algorithme possède un intérêt particulier pour calculer la visibilité entre des régions éloignées de l’espace.

Occlusion Ambiante Le dernier scénario que nous avons testé consiste à calculer l’occlusion ambiante sur une scène composée de plus de cent millions de triangles, *cf.* figure 6.15. Pour cela, nous avons disposé une dizaine de milliers de sources lumineuses autour de la scène. Nous avons alors utilisé nos cartes pour calculer sur chaque pixel la contribution de toutes ces sources. Par ailleurs, le calcul de l’occlusion ambiante est très similaire à celui d’un éclairage d’environnement lumineux à l’infini. Grâce à notre structure, on peut ainsi naturellement calculer la visibilité pour un environnement lumineux distant à haute dynamique. De cette façon, nous arrivons à produire un résultat en temps interactif, passant à l’échelle à la fois sur la taille de la géométrie et sur le nombre de sources lumineuses.

Prochaine Étape La prochaine étape serait d’appliquer ce calcul de visibilité à l’éclairage global. Lors du calcul de l’occlusion ambiante et de la visibilité de l’environnement infini, nous avons disposé des sources lumineuses autour de la scène. Pour le calcul de l’éclairage global, l’idée serait de faire de même avec les *VPL* construits par notre pipeline du chapitre précédent, *cf.* chapitre 5. Comme notre structure permet de répondre dynamiquement à n’importe quelle requête, il suffirait alors de l’interroger au moment où l’on calcule la contribution des *VPL* sur les fragments. Nous laissons cependant cela à de futurs travaux.

6.8 Discussion

Dans ce chapitre nous avons introduit une nouvelle approche pour calculer les requêtes de visibilité arbitraires. Pour cela, nous proposons de construire une représentation imprécise de la scène sous forme de voxels mais contrairement aux approches classiques, nous exploitons la mémoire mise à disposition des récentes architecture des processeurs graphiques, pour observer la scène sous un grand nombre de directions.

Nous avons tout d'abord analysé comment construire efficacement une grille de voxel grâce au mécanisme de rastérisation et les implications que cela formait sur l'agencement de la mémoire. Après avoir constaté que le choix d'une direction arbitraire pour cette voxélisation favorise certaines requêtes au dépend d'autres, nous avons introduit un mécanisme pour l'éviter. Nous avons alors défini les Hyper Voxels Sphériques comme une paramétrisation des points orientés de sorte à ce que des points alignés, ceux parcourus lors d'une requête de visibilité, puissent être enregistré linéairement en mémoire. Grâce à cette paramétrisation, nous avons optimiser le problème de marche de rayons. Ainsi, il nous suffit d'itérer sur quelques pixels et tester les voxels à l'aide de simples opérations arithmétiques sur les entiers.

Nous avons également proposé une implémentation permettant une construction dynamique de nos cartes. Nous montrons alors que notre structure permet de tester plusieurs milliards de rayons par seconde quelque soit la complexité de scène.

CONCLUSION

7.1 Résumé des contributions

Partant de scènes maillées haute résolution, typiques des scénarios de la CAO, nous avons travaillé dans l’optique de fournir une approximation en temps réel du calcul de leur éclairage global. En bornant alors le problème au transport diffus de la lumière, nous avons cherché à construire un pipeline complet de rendu s’inscrivant naturellement dans un moteur de rendu moderne. Nous avons alors considéré la représentation de la scène sous la forme de nuages de points nous permettant de travailler de manière indépendante de la façon dont la géométrie est construite. Nous avons dès lors identifié trois problèmes majeurs qui découlent de cette représentation pour permettre de construire notre pipeline. Nous avons ainsi cherché à modéliser le champ de luminance représenté par les *VPL*, puis, à construire le nuage de *VPL* en temps réel, et enfin à calculer la visibilité entre les *VPL* et les pixels.

Coupes antérogrades de lumière Pour approcher le calcul d’un rebond lumineux, les approches fondées sur des nuages de points interprètent ces derniers comme des sources lumineuses virtuelles capturant la lumière incidente et la retransmettant en fonction du matériau qui les porte. Une source de lumière ayant théoriquement une zone d’influence infinie, il est en pratique impossible de considérer un nombre important de ces sources au regard de la quantité de calculs requise. Cela nécessiterait, en effet, de calculer la contribution lumineuse entre toutes les paires d’émetteur-recepteur existantes. Cependant, bien que ces sources aient effectivement besoin d’être nombreuses pour permettre de fidèlement représenter le transport local de la lumière, le transport distant, quant à lui, peut se satisfaire d’un échantillonnage plus clairsemé de la géométrie pour produire une approximation convenable. Partant de cette observation, les méthodes hiérarchiques fondées sur des arbres permettent d’adapter efficacement les calculs aux régions qui en ont besoin. Avec ces approches, en revanche, il est nécessaire de maintenir une structure de données lourde peu compatible avec des scénarios dynamiques. Elles requièrent également de calculer à la volée de nombreuses coupes de lumière engendrant alors autant d’accès mémoire potentiellement lents et incohérents.

Nous avons ainsi proposé les *coupes antérogrades de lumière*. Celles-ci tirent profit d’une représentation multi-échelles du transport lumineux sans réellement avoir besoin de calculer de *coupe* de lumière. Le nuage de points est alors vu comme une distribution de sources lumineuses dont le support n’est plus nécessairement infini. De plus, en se fondant sur une approche stochastique imitant, en espérance, la forme du signal lumineux produit par les nœuds d’un arbre de lumière,

nous produisons une solution non biaisée et proche de la vérité terrain. Les avantages de notre approche sont alors multiples. Tout d'abord, puisque nous travaillons sur des variables aléatoires pour calculer l'éclairage et le support de nos sources virtuelles, nous pouvons distribuer efficacement ce calcul sur de nombreux cœurs en parallèle. Deuxièmement, bien que notre approche se fonde sur un estimateur de Monte-Carlo, contrairement au lancer de chemins, l'éclairage produit par des sources ponctuelles ne génère pas de bruit haute fréquence sur l'image. Enfin, en distribuant convenablement les régions d'influence des sources, nous optimisons la quantité de calculs pour produire le résultat le plus fidèle possible.

Ce chapitre a ainsi eu pour but de répondre à notre première problématique visant à modéliser mathématiquement le signal lumineux retransmis.

Rééchantillonnage dynamique par points pour l'éclairage global Comme nous voulions gérer des scènes 3D massives en temps réel, nous nous sommes naturellement tournés vers le calcul sur processeur graphique. Notre objectif a donc été de proposer une implémentation des *coupes antérogrades* fondée sur le pipeline graphique, en générant dans un premier temps un échantillonnage de la géométrie, puis dans un second temps en optimisant l'influence des *VPL* sur les pixels de l'écran. Nous avons ainsi cherché à répartir au mieux la charge de travail parmi tous les cœurs de calculs, tout en prenant en considération la nature potentiellement très disparate de la géométrie.

Comme cette dernière mélange des triangles de toutes tailles, la première étape de notre algorithme a pour but de générer uniformément des points à sa surface de sorte à s'abstraire de la représentation initiale ; nous avons ainsi nommé cette étape, le rééchantillonnage. Cependant, au vu des contraintes du calcul parallèle, il a fallu prendre soin de répartir le travail en empêchant que certaines étapes du pipeline se retrouvent surchargées et que d'autres se retrouvent en carence. C'est pourquoi nous avons dû efficacement utiliser les étapes de traitement géométrique du pipeline graphique pour, à la fois, décimer ou subdiviser les triangles en fonction de leurs tailles. Nous avons alors travaillé en deux temps, en traitant tout d'abord l'ensemble des triangles pour supprimer de manière aléatoire les très petits triangles et extraire les trop gros triangles. Ce dernier ensemble est, dans un second temps, réinjecté dans notre pipeline pour lequel on active l'unité de tessellation. De plus, comme nous n'avons utilisé que les étapes de traitement géométrique du pipeline pour construire l'échantillonnage, nous avons pu utiliser les étapes suivantes de la rastérisation pour calculer la contribution des *VPL*. Pour cela, nous avons exploité l'accélération matérielle afin de restreindre efficacement le calcul de l'éclairage aux seuls pixels effectivement influencés par un *VPL*. Ainsi, en accumulant la contribution des sources lumineuses de centaines de milliers de *VPL*, nous parvenons à simuler un rebond diffus de la lumière en temps réel sur des scènes entièrement dynamiques.

Au cours de ce chapitre, nous avons ainsi proposé un modèle d'utilisation du pipeline graphique tirant profit de toutes les étapes du traitement géométrique. De même, cela nous a également permis de répondre au second problème que nous avons posé visant à produire efficacement l'échantillonnage d'une scène par des *VPL*.

Cartes d'HyperVoxels Sphériques Le troisième problème qu'il nous a fallu résoudre pour pouvoir simuler le transport lumineux est le calcul de visibilité entre les nombreuses sources lumineuses et les pixels. Dans le but de s'intégrer au pipeline de construction et d'évaluation des *VPL* à la volée, nous avions besoin de construire une structure de donnée permettant de répondre à n'importe quelle requête de visibilité. Ainsi, à la manière des cartes d'ombre, il nous semblait né-

cessaire de pouvoir interroger notre structure en temps constant et en utilisant le minimum d'opération. Malheureusement, les cartes d'ombres ne permettent de répondre à ce problème qu'en se restreignant à une seule source de lumière. En parallèle, l'approche par marche de rayons sur une grille de voxels permet de calculer la visibilité générale, mais son exécution peut être arbitrairement longue. Cela est fortement pénalisant pour le calcul parallèle au niveau du *Fragment Shader*.

Avec les cartes d'hypervoxels sphériques, nous avons cherché à combiner les avantages de ces deux approches. Ainsi, en exploitant la mémoire mise à disposition par les architectures modernes, nous avons proposé de construire une multitude de voxelisations de la scène selon différentes directions. Chaque voxel de chaque grille contient alors une information binaire compacte enregistrant la présence ou non de la géométrie. De cette façon, nous représentons la scène de manière extrêmement redondante, mais en revanche, cela nous offre le moyen de réordonner la mémoire de sorte à ce que les requêtes y accèdent un minimum de fois. Pour calculer la visibilité entre deux points quelconques, il nous suffit alors de récupérer un champ de bits correspondant à un échantillonnage de la scène aligné avec ces deux points. La résolution de la requête peut ensuite être effectuée à l'aide de quelques opérations arithmétiques.

Notre modèle de structure de données nous a ainsi permis d'accélérer le calcul spécifique des tests de visibilité. Dans ce chapitre nous répondons donc au dernier problème lié aux approches fondées sur les *VPL*. De plus, comme notre structure peut être interrogée sans avoir besoin de préparer les requêtes, elle peut parfaitement être appelée au cours du pipeline temps réel que nous avons construit pour les coupes antérogrades, et plus précisément lors du calcul de la contribution des *VPL*.

Bilan Nous pensons qu'ensemble, ces trois chapitres définissent un algorithme complet pour simuler un rebond diffus de l'éclairage global. Dans notre problématique, nous devions prendre en compte des modèles massifs, mélangeant à la fois des géométries denses et parcimonieuses, et pouvant être complètement dynamiques. Nous avons ainsi choisi de s'abstraire de cette représentation pour travailler, à la place, sur des nuages de points. Enfin, à partir de ces points, nous avons cherché à modéliser le transport lumineux, à les construire dynamiquement sur le processeurs graphique, et à les utiliser efficacement pour construire une structure d'accélération adaptées aux requêtes de visibilité.

7.2 Perspectives

Finalement, il est important de noter que les travaux présentés dans cette thèse ouvrent la voie à d'intéressantes nouvelles recherches pour l'éclairage global en temps réel.

Dans un premier temps, nous avons modélisé le rebond diffus en s'inspirant des coupes de lumières. La famille de solutions que nous avons fournies pour les coupes antérogrades ont permis de distribuer la contribution des *VPL* en fonction de la distance du récepteur à la source. Cependant, notre méthodologie est générale et pourrait tout à fait prendre en compte n'importe quelle caractéristique du récepteur, par exemple la normale, la brillance *etc*. Cela permettrait alors d'étendre nos travaux à la gestion des réflexions brillantes, par exemple. De cette façon, les fonctions de support agiraient sur un espace de plus grande dimension que celui des positions. Il serait donc nécessaire de développer une toute nouvelle structure de données pour enregistrer les pixels de sorte à pouvoir déterminer efficacement ceux qui sont affectés par les *VPL*. Les cartes d'Hyper-

Voxels Sphériques font un pas dans cette direction, dans le sens où nous paramétrons un espace de dimension cinq pour pouvoir extraire rapidement, pour chaque requête, la fraction pertinente des informations de la scène. De même, en généralisant cette structure de données, il pourrait être envisageable d'y enregistrer autre chose que des pixels et ainsi permettre de simuler, en outre, de multiples rebonds. L'approche par coupe antérograde pourrait alors être utilisée pour simuler une solution complète du transport lumineux sans biais.

Enfin, pour définir nos *caches* de luminance, nous sommes partis d'une représentation de la scène sous forme de nuages de points. Il serait, selon nous, intéressant d'explorer d'autres représentations, comme des harmoniques sphériques ou des mixtures de gaussiennes, pour échantillonner la scène et représenter le signal lumineux. C'est par ailleurs déjà le cas dans les approches de type *PBGI* pour représenter la luminance des nœuds intérieurs. Il pourrait alors être possible de mélanger plusieurs classes de fonction dans notre formulation des coupes antérogrades, en représentant, par exemple, le transport local par un ensemble de points et le transport distant par des harmoniques sphériques (ou tout autre famille de fonctions). Pour cela néanmoins, il faudrait être en mesure de travailler à une échelle plus large que celle du triangle. En cela, de nouvelles fonctionnalités matérielles, tel que le *mesh shading* [Kubisch, 2018], proposent dès lors d'utiliser le pipeline graphique en supprimant une grande partie des contraintes de traitement de la géométrie. Nous pensons qu'il pourrait être intéressant de les exploiter en construisant un échantillonnage adapté pour représenter le transfert lumineux à différentes échelles.

BIBLIOGRAPHIE

- [Aila et coll., 2013] Timo Aila, Tero Karras, et Samuli Laine (2013). On quality metrics of bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 101–107. ACM.
- [Alcantara et coll., 2009] Dan A Alcantara, Andrei Sharf, Fatemeh Abbasinejad, Shubhabrata Sengupta, Michael Mitzenmacher, John D Owens, et Nina Amenta (2009). Real-time parallel hashing on the gpu. *ACM Transactions on Graphics (TOG)*, 28(5) :154.
- [Amanatides et coll., 1987] John Amanatides, Andrew Woo, et coll. (1987). A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10.
- [Annen et coll., 2004] Thomas Annen, Jan Kautz, Frédo Durand, et Hans-Peter Seidel (2004). Spherical harmonic gradients for mid-range illumination. In *Rendering Techniques*, pages 331–336. Citeseer.
- [Appel, 1968] Arthur Appel (1968). Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45. ACM.
- [Arikan et coll., 2005] Okan Arikan, David A Forsyth, et James F O’Brien (2005). Fast and detailed approximate global illumination by irradiance decomposition. *ACM Transactions on Graphics (TOG)*, 24(3) :1108–1114.
- [Arvo et coll., 1994] James Arvo, Kenneth Torrance, et Brian Smits (1994). A framework for the analysis of error in global illumination algorithms. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 75–84. ACM.
- [Baldwin et Weber, 2016] Doug Baldwin et Michael Weber (2016). Fast ray-triangle intersections by coordinate transformation. *Journal of Computer Graphics Techniques Vol*, 5(3).
- [Barák et coll., 2013] Tomá Barák, Jirí Bittner, et Vlastimil Havran (2013). Temporally coherent adaptive sampling for imperfect shadow maps. In *Computer Graphics Forum*, volume 32, pages 87–96. Wiley Online Library.
- [Basri et Jacobs, 2001] Ronen Basri et David Jacobs (2001). Lambertian reflectance and linear subspaces. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 383–390. IEEE.
- [Bavoil et Sainz, 2009] Louis Bavoil et Miguel Sainz (2009). Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH 2009 : Talks*, page 45. ACM.
- [Bavoil et coll., 2008] Louis Bavoil, Miguel Sainz, et Rouslan Dimitrov (2008). Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks*, page 22. ACM.
- [Bitterli, 2016] Benedikt Bitterli (2016). Rendering resources. <https://benedikt-bitterli.me/resources/>.
- [Blackwell, 1972] H Richard Blackwell (1972). Luminance difference thresholds. In *Visual Psychophysics*, pages 78–101. Springer.
- [Bresenham, 1965] Jack E Bresenham (1965). Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1) :25–30.
- [Brouwer, 1911] Luitzen Egbertus Jan Brouwer (1911). Über abbildung von mannigfaltigkeiten. *Mathematische Annalen*, 71(1) :97–115.
- [Buckley, 1927] H Buckley (1927). Lxvii. on the radiation from the inside of a circular cylinder. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 4(23) :753–762.
- [Chaitanya et coll., 2017] Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, et Timo Aila (2017). Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4) :98.
- [Christensen, 2008] Per Christensen (2008). Point-based approximate color bleeding. *Pixar Technical Notes*, 2(5) :6.
- [Christensen et Jarosz, 2016] P. H. Christensen et W. Jarosz (2016). The path to path-traced movies. In *Computer Graphics and Vision*, volume 10, pages 103–175. Foundations and Trends.

- [Cigolle et coll., 2014] Zina H Cigolle, Sam Donow, Daniel Evangelakos, Michael Mara, Morgan McGuire, et Quirin Meyer (2014). A survey of efficient representations for independent unit vectors. *Journal of Computer Graphics Techniques*, 3(2).
- [Cohen et Greenberg, 1985] Michael F Cohen et Donald P Greenberg (1985). The hemi-cube : A radiosity solution for complex environments. In *ACM SIGGRAPH Computer Graphics*, volume 19, pages 31–40. ACM.
- [Comaniciu et Meer, 2002] Dorin Comaniciu et Peter Meer (2002). Mean shift : A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5) :603–619.
- [Cook, 1986] Robert L Cook (1986). Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)*, 5(1) :51–72.
- [Cook et coll., 1984] Robert L Cook, Thomas Porter, et Loren Carpenter (1984). Distributed ray tracing. In *ACM SIGGRAPH computer graphics*, volume 18, pages 137–145. ACM.
- [Crassin et coll., 2011] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, et Elmar Eisemann (2011). Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, volume 30, pages 1921–1930. Wiley Online Library.
- [Crow, 1977] Franklin C Crow (1977). Shadow algorithms for computer graphics. In *Acm siggraph computer graphics*, volume 11, pages 242–248. ACM.
- [Dachsbacher et coll., 2014] Carsten Dachsbaucher, Jaroslav Krivánek, Miloš Hašan, Adam Arbree, Bruce Walter, et Jan Novák (2014). Scalable realistic rendering with many-light methods. In *Computer Graphics Forum*, volume 33, pages 88–104. Wiley Online Library.
- [Dachsbaucher et Stamminger, 2005] Carsten Dachsbaucher et Marc Stamminger (2005). Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231. ACM.
- [Dachsbaucher et Stamminger, 2006] Carsten Dachsbaucher et Marc Stamminger (2006). Splatting indirect illumination. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 93–100. ACM.
- [Dachsbaucher et coll., 2007] Carsten Dachsbaucher, Marc Stamminger, George Drettakis, et Frédo Durand (2007). Implicit visibility and antiradiance for interactive global illumination. In *ACM Transactions on Graphics (TOG)*, volume 26, page 61. ACM.
- [Davidovič et coll., 2014] Tomáš Davidovič, Jaroslav Krivánek, Miloš Hašan, et Philipp Slusallek (2014). Progressive light transport simulation on the gpu : Survey and improvements. *ACM Transactions on Graphics (TOG)*, 33(3) :29.
- [Dong et coll., 2009] Zhao Dong, Thorsten Grosch, Tobias Ritschel, Jan Kautz, et Hans-Peter Seidel (2009). Real-time indirect illumination with clustered visibility. In *VMV*, volume 9, pages 187–196.
- [Durand et coll., 2005] Frédo Durand, Nicolas Holzschuch, Cyril Soler, Eric Chan, et François X Sillion (2005). A frequency analysis of light transport. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 1115–1126. ACM.
- [Eisemann et Décoret, 2006] Elmar Eisemann et Xavier Décoret (2006). Fast scene voxelization and applications. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 71–78. ACM.
- [Eisemann et coll., 2016] Elmar Eisemann, Michael Schwarz, Ulf Assarsson, et Michael Wimmer (2016). *Real-time shadows*. AK Peters/CRC Press.
- [Eisenacher et coll., 2013] Christian Eisenacher, Gregory Nichols, Andrew Selle, et Brent Burley (2013). Sorted deferred shading for production path tracing. In *Computer Graphics Forum*, volume 32, pages 125–132. Wiley Online Library.
- [Garanzha et coll., 2011] Kirill Garanzha, Jacopo Pantaleoni, et David McAllister (2011). Simpler and faster hlbvh with work queues. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, pages 59–64. ACM.
- [Gautron et coll., 2005] Pascal Gautron, Jaroslav Krivánek, Kadi Bouatouch, et Sumanta Pattanaik (2005). Radiance cache splatting : A gpu-friendly global illumination algorithm. In *ACM SIGGRAPH 2005 Sketches*, page 36. ACM.
- [Goral et coll., 1984] Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, et Bennett Battaille (1984). Modeling the interaction of light between diffuse surfaces. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 213–222. ACM.

- [Gortler et coll., 1993] Steven J Gortler, Peter Schröder, Michael F Cohen, et Pat Hanrahan (1993). Wavelet radiosity. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 221–230. ACM.
- [Hanrahan et coll., 1991] Pat Hanrahan, David Salzman, et Larry Aupperle (1991). A rapid hierarchical radiosity algorithm. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 197–206. ACM.
- [Hasenfratz et coll., 2003] J-M Hasenfratz, Marc Lapierre, Nicolas Holzschuch, François Sillion, et Artis GRAVIR (2003). A survey of real-time soft shadows algorithms. In *Computer Graphics Forum*, volume 22, pages 753–774. Wiley Online Library.
- [He et coll., 2013] Kaiming He, Jian Sun, et Xiaou Tang (2013). Guided image filtering. *IEEE transactions on pattern analysis & machine intelligence*, (6) :1397–1409.
- [Hedman et coll., 2016] Peter Hedman, Tero Karras, et Jaakko Lehtinen (2016). Sequential monte carlo instant radiosity. In *Proceedings of the 20th ACM SIGGRAPH symposium on interactive 3D graphics and games*, pages 121–128. ACM.
- [Hollander et coll., 2011] Matthias Hollander, Tobias Ritschel, Elmar Eisemann, et Tamy Boubekeur (2011). Many-lods : Parallel many-view level-of-detail selection for real-time global illumination. In *Computer Graphics Forum*, volume 30, pages 1233–1240. Wiley Online Library.
- [Immell et coll., 1986] David S Immell, Michael F Cohen, et Donald P Greenberg (1986). A radiosity method for non-diffuse environments. In *ACM SIGGRAPH Computer Graphics*, volume 20, pages 133–142. ACM.
- [Kajiya, 1986] James T. Kajiya (1986). The rendering equation. *Proc. SIGGRAPH*, 20(4) :143–150.
- [Kaplanyan et Dachsbacher, 2010] Anton Kaplanyan et Carsten Dachsbaucher (2010). Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 99–107. ACM.
- [Karras et Aila, 2013] Tero Karras et Timo Aila (2013). Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 89–99. ACM.
- [Kautz et coll., 2002] Jan Kautz, John Snyder, et Peter-Pike J Sloan (2002). Fast arbitrary brdf shading for low-frequency lighting using spherical harmonics. *Rendering Techniques*, 2(291-296) :1.
- [Keller, 1997] Alexander Keller (1997). Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co.
- [Keller et Heidrich, 2001] Alexander Keller et Wolfgang Heidrich (2001). Interleaved sampling. In *Rendering Techniques 2001*, pages 269–276. Springer.
- [Kessenich et coll., 2016] John Kessenich, Graham Sellers, et Dave Shreiner (2016). *OpenGL Programming Guide : The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V*. Addison-Wesley Professional.
- [Kämpe et coll., 2013] V. Kämpe, E. Sintorn, et U. Assarsson (2013). High resolution sparse voxel dags. In *ACM Transactions on Graphics (TOG)*, volume 32, page 101. ACM.
- [Kubisch, 2018] Christoph Kubisch (2018). Introduction to turing mesh shaders.
- [Laine et coll., 2013] Samuli Laine, Tero Karras, et Timo Aila (2013). Megakernels considered harmful : wavefront path tracing on gpus. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 137–143. ACM.
- [Laine et coll., 2007] Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen, et Timo Aila (2007). Incremental instant radiosity for real-time indirect illumination. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 277–286. Eurographics Association.
- [Laurent et coll., 2017] Gilles Laurent, Cyril Delalandre, et Tamy Boubekeur (2017). Visibility function of a three-dimensional scene. EP17306930.3.
- [Laurent et coll., 2016a] Gilles Laurent, Cyril Delalandre, Grégoire de La Rivière, et Tamy Boubekeur (2016a). Forward light cuts : A scalable approach to real-time global illumination. *Computer Graphics Forum (Proc. EGSR 2016)*, 35(4) :79–88.
- [Laurent et coll., 2016b] Gilles Laurent, Cyril Delalandre, Grégoire de La Rivière, et Tamy Boubekeur (2016b). Rendering the global illumination of a 3d scene. EP 3 211 601 A1.

- [Lefebvre et Hoppe, 2006] Sylvain Lefebvre et Hugues Hoppe (2006). Perfect spatial hashing. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 579–588. ACM.
- [Lehtinen et coll., 2007] Jaakko Lehtinen, Matthias Zwicker, Janne Kontkanen, Emmanuel Turquin, François X Silion, et Timo Aila (2007). *Meshless finite elements for hierarchical global illumination*. PhD thesis, Publications in Telecommunications Software and Multimedia, Helsinki University of Technology.
- [Maletz et Wang, 2011] David Maletz et Rui Wang (2011). Importance point projection for gpu-based final gathering. In *Computer Graphics Forum*, volume 30, pages 1327–1336. Wiley Online Library.
- [Mara et coll., 2016] Michael Mara, Morgan McGuire, Derek Nowrouzezahrai, et David P Luebke (2016). Deep g-buffers for stable global illumination approximation. In *High Performance Graphics*, pages 87–98.
- [Mawhin, 2007] Jean Mawhin (2007). Le théorème du point fixe de brouwer : un siecle de métamorphoses. *Revue d'Histoire des Mathématiques,a paraître*.
- [McGuire et coll., 2017] Morgan McGuire, Mike Mara, Derek Nowrouzezahrai, et David Luebke (2017). Real-time global illumination using precomputed light field probes. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, page 2. ACM.
- [Meyer et coll., 2010] Quirin Meyer, Jochen Süßmuth, Gerd Sußner, Marc Stamminger, et Günther Greiner (2010). On floating-point normal vectors. In *Computer Graphics Forum*, volume 29, pages 1405–1409. Wiley Online Library.
- [Mittring, 2007] Martin Mittring (2007). Finding next gen : Cryengine 2. In *ACM SIGGRAPH 2007 courses*, pages 97–121. ACM.
- [Nalbach et coll., 2014] Oliver Nalbach, Tobias Ritschel, et Hans-Peter Seidel (2014). Deep screen space. In *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 79–86. ACM.
- [Nalbach et coll., 2015] Oliver Nalbach, Tobias Ritschel, et H-P Seidel (2015). The bounced z-buffer for indirect visibility. In *Proc. Vision, Modelling and Visualization*.
- [Nichols et coll., 2010] Greg Nichols, Rajeev Penmatsa, et Chris Wyman (2010). Interactive, multiresolution image-space rendering for dynamic area lighting. In *Computer Graphics Forum*, volume 29, pages 1279–1288. Wiley Online Library.
- [Nichols et coll., 2009] Greg Nichols, Jeremy Shopf, et Chris Wyman (2009). Hierarchical image-space radiosity for interactive global illumination. In *Computer Graphics Forum*, volume 28, pages 1141–1149. Wiley Online Library.
- [Nichols et Wyman, 2009] Greg Nichols et Chris Wyman (2009). Multiresolution splatting for indirect illumination. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 83–90. ACM.
- [Novák et coll., 2010] Jan Novák, Vlastimil Havran, et Carsten Dachsbacher (2010). Path regeneration for interactive path tracing. In *Eurographics (Short Papers)*, pages 61–64.
- [Olsson et coll., 2012] Ola Olsson, Markus Billeter, et Ulf Assarsson (2012). Clustered deferred and forward shading. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, pages 87–96. Eurographics Association.
- [Olsson et coll., 2014] Ola Olsson, Erik Sintorn, Viktor Kämpe, Markus Billeter, et Ulf Assarsson (2014). Efficient virtual shadow maps for many lights. In *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 87–96. ACM.
- [Pharr et coll., 2016] Matt Pharr, Wenzel Jakob, et Greg Humphreys (2016). *Physically based rendering : From theory to implementation*. Morgan Kaufmann.
- [Prutkin et coll., 2012] Roman Prutkin, Anton Kaplanyan, et Carsten Dachsbaucher (2012). Reflective shadow map clustering for real-time global illumination. In *Eurographics (Short Papers)*, pages 9–12.
- [Ramamoorthi et Hanrahan, 2001] Ravi Ramamoorthi et Pat Hanrahan (2001). On the relationship between radiance and irradiance : determining the illumination from images of a convex lambertian object. *JOSA A*, 18(10) :2448–2459.
- [Ritschel et coll., 2007] Tobias Ritschel, Thorsten Grosch, Jan Kautz, et Stefan Müller (2007). Interactive illumination with coherent shadow maps. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 61–72. Eurographics Association.

- [Ritschel et coll., 2008a] Tobias Ritschel, Thorsten Gorsch, Jan Kautz, et Hans-Peter Seidel (2008a). Interactive global illumination based on coherent surface shadow maps. In *Proceedings of Graphics Interface 2008*, pages 185–192. Canadian Information Processing Society.
- [Ritschel et coll., 2008b] Tobias Ritschel, Thorsten Gorsch, Min H Kim, H-P Seidel, Carsten Dachsbacher, et Jan Kautz (2008b). Imperfect shadow maps for efficient computation of indirect illumination. In *ACM Transactions on Graphics (TOG)*, volume 27, page 129. ACM.
- [Ritschel et coll., 2009a] Tobias Ritschel, Thomas Engelhardt, Thorsten Gorsch, H-P Seidel, Jan Kautz, et Carsten Dachsbacher (2009a). Micro-rendering for scalable, parallel final gathering. In *ACM Transactions on Graphics (TOG)*, volume 28, page 132. ACM.
- [Ritschel et coll., 2009b] Tobias Ritschel, Thorsten Gorsch, et Hans-Peter Seidel (2009b). Approximating dynamic global illumination in image space. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 75–82. ACM.
- [Ritschel et coll., 2011] Tobias Ritschel, Elmar Eisemann, Inwoo Ha, James DK Kim, et Hans-Peter Seidel (2011). Making imperfect shadow maps view-adaptive : High-quality global illumination in large dynamic scenes. In *Computer Graphics Forum*, volume 30, pages 2258–2269. Wiley Online Library.
- [Ritschel et coll., 2012] Tobias Ritschel, Carsten Dachsbacher, Thorsten Gorsch, et Jan Kautz (2012). The state of the art in interactive global illumination. In *Computer Graphics Forum*, volume 31, pages 160–188. Wiley Online Library.
- [Saito et Takahashi, 1990] Takafumi Saito et Tokiichiro Takahashi (1990). Comprehensible rendering of 3-d shapes. In *ACM SIGGRAPH Computer Graphics*, volume 24, pages 197–206. ACM.
- [Schröder et Hanrahan, 1993] Peter Schröder et Pat Hanrahan (1993). On the form factor between two polygons. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 163–164. ACM.
- [Segovia et coll., 2006] Benjamin Segovia, Jean Claude Iehl, Richard Mitanchev, et Bernard Péroche (2006). Bidirectional instant radiosity. In *Rendering Techniques*, pages 389–397.
- [Segovia et coll., 2007] Benjamin Segovia, Jean Claude Iehl, et Bernard Péroche (2007). Metropolis instant radiosity. In *Computer Graphics Forum*, volume 26, pages 425–434. Wiley Online Library.
- [Sloan et coll., 2002] Peter-Pike Sloan, Jan Kautz, et John Snyder (2002). Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 527–536. ACM.
- [Smits et coll., 1994] Brian Smits, James Arvo, et Donald Greenberg (1994). A clustering algorithm for radiosity in complex environments. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 435–442. ACM.
- [Soler et coll., 2010] Cyril Soler, Olivier Hoel, et Frank Rochet (2010). A deferred shading pipeline for real-time indirect illumination. In *ACM SIGGRAPH 2010 Talks*, page 18. ACM.
- [Stich, 2018] Martin Stich (2018). Introduction to nvidia rtx and directx ray tracing.
- [Stich et coll., 2009] Martin Stich, Heiko Friedrich, et Andreas Dietrich (2009). Spatial splits in bounding volume hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009*, pages 7–13. ACM.
- [Sun et Ramamoorthi, 2009] Bo Sun et Ravi Ramamoorthi (2009). Affine double-and triple-product wavelet integrals for rendering. *ACM Transactions on Graphics (TOG)*, 28(2) :14.
- [Tomasi et Manduchi, 1998] Carlo Tomasi et Roberto Manduchi (1998). Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE.
- [Tsai et Shih, 2006] Yu-Ting Tsai et Zen-Chung Shih (2006). All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 967–976. ACM.
- [Van Antwerpen, 2011] Dietger Van Antwerpen (2011). Improving simd efficiency for parallel monte carlo light transport on the gpu. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, pages 41–50. ACM.

- [Vardis et coll., 2013] Kostas Vardis, Georgios Papaioannou, et Athanasios Gaitatzes (2013). Multi-view ambient occlusion with importance sampling. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 111–118. ACM.
- [Wald et coll., 2001] Ingo Wald, Philipp Slusallek, Carsten Benthin, et Markus Wagner (2001). Interactive rendering with coherent ray tracing. In *Computer graphics forum*, volume 20, pages 153–165. Wiley Online Library.
- [Walter et coll., 2006] Bruce Walter, Adam Arbree, Kavita Bala, et Donald P Greenberg (2006). Multidimensional lightcuts. *ACM Transactions on graphics (TOG)*, 25(3) :1081–1088.
- [Walter et coll., 2005] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, et Donald P Greenberg (2005). Lightcuts : a scalable approach to illumination. *ACM Transactions on graphics (TOG)*, 24(3) :1098–1107.
- [Wang et coll., 2004] Zhou Wang, Alan C Bovik, Hamid R Sheikh, et Eero P Simoncelli (2004). Image quality assessment : from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4) :600–612.
- [Ward et Heckbert, 1992] Greg Ward et Paul S Heckbert (1992). Irradiance gradients.
- [Ward et coll., 1988] Gregory J Ward, Francis M Rubinstein, et Robert D Clear (1988). A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics*, 22(4) :85–92.
- [Whitted, 1979] Turner Whitted (1979). An improved illumination model for shaded display. In *ACM SIGGRAPH Computer Graphics*, volume 13, page 14. ACM.
- [Williams, 1978] Lance Williams (1978). Casting curved shadows on curved surfaces. In *ACM Siggraph Computer Graphics*, volume 12, pages 270–274. ACM.
- [Yamauti, 1926] Ziro Yamauti (1926). The light flux distribution of a system of interreflecting surfaces. *JOSA*, 13(5) :561–571.
- [Zwicker et coll., 2015] Matthias Zwicker, Wojciech Jarosz, Jaakkko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, et S-E Yoon (2015). Recent advances in adaptive sampling and reconstruction for monte carlo rendering. In *Computer Graphics Forum*, volume 34, pages 667–681. Wiley Online Library.

TABLE DES FIGURES

1.1	Synthèse d'images dans l'industrie	2
2.1	Modélisation d'une scène 3D pour la synthèse d'image	8
2.2	Grandeurs physiques de la synthèse d'image	9
2.3	Réflexion surfacique de la lumière	12
2.4	Mécanisme de rastérisation avec test de visibilité	17
2.5	Pipeline graphique programmable	18
3.1	Éclairage globale par lancer de chemins	22
3.2	Éclairage globale par éléments finis	25
3.3	Reconstruction par interpolation sur une grille régulière	27
3.4	Éclairage global par radiosité instantanée	29
3.5	Éclairage global en espace image	32
3.6	Approche hiérarchique pour l'éclairage global	33
3.7	Coupe de lumière	34
4.1	Coupe Rétrograde et Coupe Antérograde	39
4.2	Domaine de définition des fonctions de support	42
4.3	Surfaces iso en dimension 3 de l'application α_y	44
4.4	Solution \mathcal{C}^{-1}	46
4.5	Solution \mathcal{C}^0	46
4.6	Comparaisons des coupes antérogrades discrètes	51
4.7	Comparaison de fonction de support \mathcal{C}^0 et \mathcal{C}^{-1}	53
4.8	Plongement trivial de s_k en dimension 2	56
4.9	Prolongement linéaire de s_k en dimension 2	57
4.10	Prolongements continues de \tilde{s} représentés en niveaux de gris	58
4.11	Définition des bornes du prolongement linéaire lisse de \tilde{s}	60
4.12	Comparaisons des coupes antérogrades continues	62
5.1	Rendu temps réel de l'éclairage global	66
5.2	Vue d'ensemble de notre algorithme d'éclairage global	68
5.3	Algorithme de multi-échantillonnage naïf	71
5.4	Pipeline d'échantillonnage global	77
5.5	Construction de variable aléatoire par triangle	78
5.6	Échantillonnage uniforme du triangle	80
5.7	Prolongation de la variable aléatoire sur la subdivision	83
5.8	Pipeline d'approximation de l'éclairage global	85
5.9	Aplatissement en espace image	88
5.10	Influence du sous-échantillonnage	90
5.11	Comparaison entre rendu convergé et temps réel	92
5.12	Comparaison avec la vérité terrain	92
6.1	Pipeline temps réel des cartes d'HVS	96

6.2	Schéma haut niveau de l'algorithme générique de <i>marche de rayons</i>	98
6.3	Marche de rayon en dimension un	101
6.4	Algorithme de calcul de visibilité en dimension un	102
6.5	Marche de rayons en dimension deux	104
6.6	Paramétrisations de l'ensemble des droites du plan	107
6.7	Paramétrisations de l'ensemble des droites de l'espace	108
6.8	Paramétrisation de l'espace tangent à la sphère, continue presque partout.	109
6.9	Paramétrisation et quantification de l'hémisphère par projection octaédrique	110
6.10	Construction des cartes d'Hyper Voxels Sphériques	113
6.11	Évaluation d'une requête de visibilité	114
6.12	Calcul du masque 3D	116
6.13	Suppression de l'auto-occlusion et filtrage	116
6.14	Calcul de l'éclairage direct	118
6.15	Calcul de l'occlusion ambiante	119

