Compression collaborative de rayons de lumière pour le rendu distribué de Monte Carlo et applications

Sylvain Rousseau

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2020IPPAT030

Thèse de doctorat

Compression collaborative de rayons de lumière pour le rendu distribué de Monte Carlo et applications

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 23 Septembre 2020, par

SYLVAIN ROUSSEAU

Composition du Jury :

Mme Isabelle Bloch Professeur, LTCI, Télécom Paris	Présidente
M. Guillaume Lavoué Professeur, LIRIS, Ecole Nationale d'Ingénieurs de Saint-Etienne	Rapporteur
M. Frédéric Mora Maître de conférences, XLIM, Université de Limoges	Rapporteur
M. François Goulette Professeur, Mines Paristech	Examinateur
M. Tamy Boubekeur Professeur, LTCI, Télécom Paris, Adobe 3D & Immersive	Directeur de thèse

Compression collaborative de rayons de lumière pour le rendu distribué de Monte Carlo et applications

THÈSE DE DOCTORAT

M. Sylvain Rousseau

Octobre 2016 - Septembre 2020

Résumé

Cette thèse s'inscrit dans le domaine de l'informatique graphique en étudiant un élément clé, à savoir les vecteurs unitaires. Nous proposons un nouvel espace de représentation d'ensemble de vecteurs unitaires avant de montrer plusieurs applications adaptant celles-ci à différents types de données.

Dans une première partie, nous proposons une méthode de compression d'ensembles de vecteurs unitaires désordonnées. Cette méthode, nommée `UniQuant`, permet de réaliser une compression des données de manière collaborative, en générant de la cohérence puis en l'exploitant pour changer l'espace de représentation des données. Celle-ci est ensuite exploitée au travers d'une première application, permettant de compresser des ensembles de nuages de points munis de normales, et permet ainsi de réaliser la compression des données à la volée.

Nous proposons ensuite une application à un élément clé du rendu de Monte Carlo : le rayon de lumière. Celui-ci est la structure de donnée de base permettant de réaliser la simulation du transport de la lumière dans une scène virtuelle 3D, en construisant des chemins de lumière représentés à l'aide de polylignes 3D, reliant le capteur virtuel (caméra) aux différentes sources de lumière. L'application de la compression est utilisée dans le cas distribué, où un moteur construit pour exploiter un ensemble de machines sur des réseaux distants est utilisé. Les architectures matérielles de ce type sont devenues de plus en plus populaires avec l'apparition de projets tels que SETI@Home. Elles pourraient facilement être étendues pour exploiter les machines présentes dans les institutions publiques ou dans les entreprises et utilisées moins de la moitié du temps. Cela permettrait ainsi d'exploiter la puissance de calcul perdue. La technique proposée utilise la multitude de rayons disponibles dans le cas d'un moteur distribué exploitant des portails de lumière pour réaliser une compression collaborative, permettant d'accélérer les vitesses de transfert de données sur un réseau non local. La compression des directions est étendue à celle des origines pour examiner l'impact de la baisse de précision sur les rendus. Nous montrons également que la précision de la compression des directions peut être corrélée aux matériaux rencontrés.

Enfin, nous présentons `qfib`, une adaptation d'`UniQuant` à d'autres types de données présentant le même type de contraintes mathématiques que les ensembles de rayons : des tractogrammes. Ceux-ci sont couramment utilisés en neurosciences pour visualiser les zones d'influence neuronales dans le cerveau. Ils permettent aux neurochirurgiens de prédire les effets possibles d'une opération, et aux chercheurs de mieux comprendre le fonctionnement du cerveau. L'utilisation de ce type de données est complexe du fait de leur taille, les rendant difficiles à visionner, traiter, stocker ou même échanger. L'algorithme introduit permet de diviser cette taille par 10 en quelques secondes pour des jeux de données typiquement utilisés, tout en assurant une perte inférieure à la précision des IRM ayant permises d'obtenir les jeux de données.

Remerciements

Cette thèse a été financée dans le cadre du Projet PAPAYA par la Direction Générale des Entreprises (DGE) via les Investissements d'avenir et par la Direction Générale de l'Armement (DGA) dans le cadre d'une demi-bourse DGA.

Je souhaite remercier Isabelle Bloch, Guillaume Lavoué, Frédéric Mora et François Goulette pour avoir accepté de faire partie de mon jury, et pour leurs remarques et conseils.

Je remercie chaleureusement Tamy Boubekeur pour m'avoir guidé et soutenu durant toutes ces années. Il m'a toujours suffit de petites réunions pour être inspiré et remonté pour plusieurs semaines. Je suis chanceux d'avoir pu travailler dans le groupe de recherche qu'il a créé, où bonne humeur et entraide ont toujours été au rendez-vous. Merci infiniment !

Merci à Jean-Marc pour ses conseils, et nos nombreuses discussions. Mais également pour toutes les diverses idées partagées sur des potentiels projets de recherche sans malheureusement avoir eu le temps de s'y pencher. Merci à Isabelle Bloch pour tout le temps qu'elle nous a généreusement accordé, ses commentaires et sa gentillesse, dans le projet de recherche que nous avons mené ensemble.

Merci à tous mes collègues, actuels et passés. Nos longues discussions ont toujours été un plaisir et la bonne ambiance du laboratoire, qui va bien au delà du travail, et participe à la réussite de tout le monde.

Je remercie tout particulèrement Malik Boughida, mon premier mentor, et Gilles Laurent qui m'ont tant appris en informatique et en rendu durant les longues journées passées au tableau et durant les soirées passées à manger de nombreux sushis.

Merci à Thibault, collègue du premier jour, pour son grand savoir du C++ qu'il a toujours généreusement partagé, pour nos nombreux débats, pour son temps et son chauvinisme breton. Merci à Corentin pour le projet réalisé ensemble. Ça a été un grand plaisir de pouvoir collaborer ensemble. Merci également à Victor pour cet incroyable voyage en Thaïlande, Hélène pour sa bonne humeur, Émilie pour ses conseils avant le début de la thèse, Norbert pour m'avoir accompagné dans le projet Papaya, Thibaud L. pour tout ses conseils, Elie, Jérémie, Théo, Adrien et Christophe pour toutes les discussions et Kiwon,

Alban, Chloé et Thong pour avoir formé un groupe de recherche vraiment unique ! J'espère que nos *Friday beers*, *Game Jams*, sessions d'escalade et diverses soirées pourront bientôt reprendre dès que le monde sera revenu à la normale ! Merci aux membres de passages pour toute la fraîcheur qu'ils ont apporté, Manuele, Fabrizio, Thibault G.. Merci au groupe IMAGES, pour toutes les discussions du midi, et les cafés. Thierry, Sylvain 2D, Guillaume, Christian, Alessandro, Hélène 2D, Nicolas, Tim, Clément, Xiangli, Emanuele, Mateus et tous les autres.

Merci aussi à Alexandre, Marwan, Arnaud, Juan, les éternels soutiens !

Je tiens enfin remercier, mes parents, mon frère et Laure pour leur soutien, leurs encouragements, les relectures, les conseils, et pour les nombreuses discussions jusqu'à tard dans la nuit.

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

Introduction

FIGURE 1.1 – Rendu réaliste de la couronne impériale d'Autriche réalisé avec Hurricane, le moteur de rendu discuté dans le chapitre 2

L'informatique graphique est devenue en un demi-siècle une science incontournable dans la vie de tous les jours. Elle est utilisée aussi bien dans la recherche scientifique pour la visualisation de données, que dans l'industrie pour la conception d'objets. Pour le grand public, on la retrouve dans des domaines aussi variés que le cinéma d'animation ou les effets spéciaux. Cette progression fulgurante a été rendue possible grâce à l'explosion de la puissance calculatoire des machines, ayant permis la représentation de mondes virtuels toujours plus complexes et réalistes. Nous nous intéressons ici, dans un contexte particulier, au rendu réaliste pour le cinéma d'animation et pour les effets spéciaux. C'est-à-dire aux méthodes de synthèse d'images photo-réalistes basées sur une simulation du transport de la lumière et notamment aux méthodes de tracer de chemin de Monte Carlo.

1.1 Contexte

Le processus de synthèse d'images est coûteux en calculs. Il nécessite parfois des dizaines d'heures sur des machines surpuissantes pour obtenir une seule image. Il faut un peu plus de 100 000 images pour une heure de film, rendant impossible de tels calculs sur des architectures non dédiées

Le but de la thèse est de développer de nouvelles méthodes de rendu de scènes de production, telles que celles utilisées dans les grands studios de production cinématographiques, pour des studios ou des laboratoires n'ayant pas accès à des ressources matérielles dédiées.

Dans ce contexte, dans la philosophie des projets tels que SETI@Home qui permettent d'accéder à des machines peu puissantes en très grand nombre pour faire des calculs coûteux, nous nous sommes intéressés à l'utilisation de clusters composés de centaines d'ordinateurs de tous les jours. Cela peut être par exemple les ordinateurs de salles de travaux pratiques d'une école, ou encore les ordinateurs utilisés par les employés d'une entreprise. Ces machines non spécialisées à ce type de calculs soulèvent des défis scientifiques et techniques pour pouvoir être exploitées efficacement. A terme, cela aura pour but d'exploiter la puissance de calcul perdue de ces ordinateurs. Ceux-ci sont disponibles en nombre dans les bureaux et sont typiquement utilisés moins de la moitié du temps. Cela aurait l'avantage de limiter les besoins en maintenance et en investissement matériel, et permettrait de conduire de nouveaux types de recherches, ou de servir d'architecture matérielle temporaire pour la génération d'images pour des petits studios.

Ces ordinateurs sont de faible puissance (comparativement aux ordinateurs utilisés dans les fermes de rendu), disposent de peu de RAM, et sont interconnectés avec des réseaux de faible puissance. La manière d'aborder la simulation du transport de la lumière doit donc être adaptée.

1.2 Informatique graphique et rendu réaliste

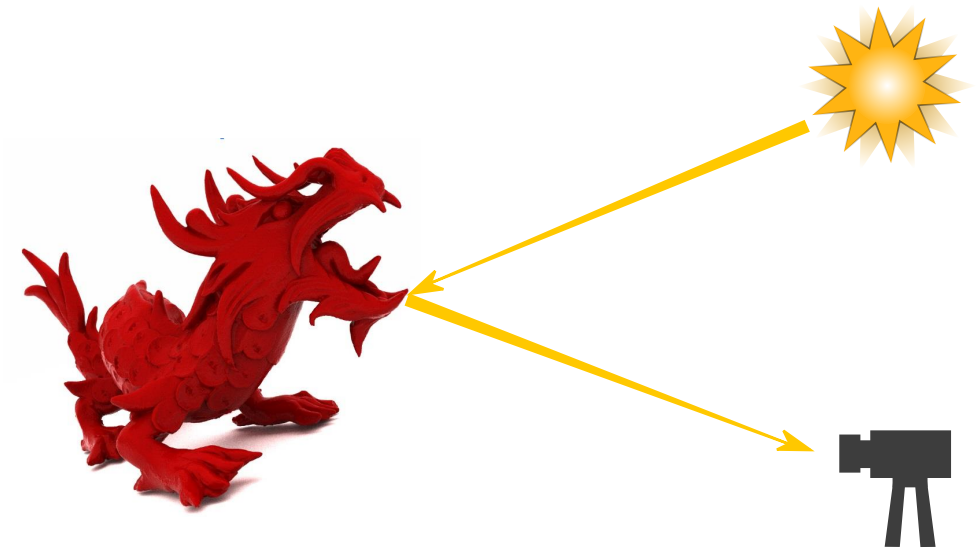


FIGURE 1.2 – Description d'une scène virtuelle 3D et de son rendu selon les méthodes de Monte Carlo.

Cette thèse se situe dans le domaine de l'informatique graphique 3D. C'est le domaine de l'informatique s'intéressant à la création, la modification, l'interaction, et la représentation de l'information tridimensionnelle. Le sous-domaine nous intéressant plus particulièrement est celui du rendu, étudiant les algorithmes permettant d'obtenir une image à partir d'une scène virtuelle 3D. Cette dernière est composée de géométries, de matériaux, de sources de lumières et d'un capteur virtuel. La représentation de celle-ci sous forme d'image peut varier en fonction du type d'application visé. Alors que la visualisation de données scientifiques cherchera à mettre en valeurs certaines données, sacrifiant le réalisme au bénéfice de la compréhensibilité, les effets spéciaux quant à eux auront pour but d'obtenir des images réalistes. Au delà du critère de réalisme, la vitesse d'exécution est

également un facteur important. Une application interactive telle qu'un logiciel de modélisation, ou un jeu vidéo nécessitera des dizaines d'images par seconde. Dans le cas d'un film d'animation, il sera possible d'attendre plusieurs heures pour calculer chaque image avant de les assembler pour obtenir le résultat final. Dans ce dernier cas, l'image réaliste est obtenue en réalisant une simulation du transport de la lumière dans la scène entre les différentes sources de la lumière et le capteur. Cela est fait avec les méthodes de Monte Carlo en construisant des chemins de lumière correspondant à des *rebonds* de la lumière sur la géométrie de la scène, comme décrit dans la figure 1.2.

Dans le cadre de cette thèse, nous nous intéressons aux rendus utilisés dans la production des films. Les contraintes sont alors les suivantes : nous disposons d'un budget de temps important (typiquement quelques heures pour une seule image), et cherchons à réaliser une simulation aussi précise que possible pour obtenir une image réaliste. Pour effectuer un tel rendu réaliste, ou basé physique, il est nécessaire de calculer les interactions lumineuses entre les différents éléments d'une scène virtuelle 3D composée de géométrie et de matériaux, et éclairée par des sources de lumières.

Ces interactions sont décrites à partir de l'équation du rendu (Équation 1.1) [Kajiya, 1986], ou équation de transport lumineux, qui est une équation intégrale récursive, dont la solution exacte reste actuellement impossible à calculer dans les cas non triviaux. Cependant, de nombreuses techniques ont été proposées pour en approximer une solution.

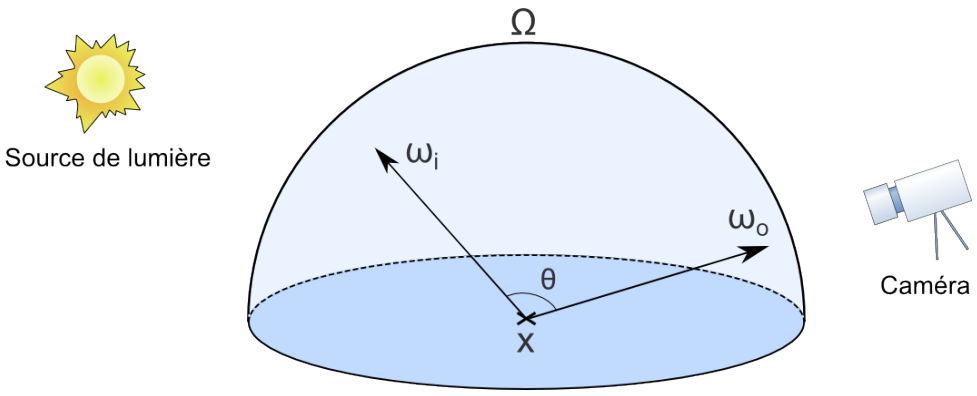


FIGURE 1.3 – Notations utilisées dans l'équation du rendu [Kajiya, 1986]

$$L_o(x, \omega_0) = L_e(x, \omega_o) + \int_\Omega L_i(x, \omega_i) fr(x, \omega_i, \omega_o) \cos\theta \, d\omega_i \qquad (1.1)$$

Cette équation décrit la luminance énergétique (ou radiance) dans la direction d'émission ω_o depuis le point x en prenant en considération le comportement de la lumière lorsqu'elle est en présence de matériaux. La luminance énergétique s'exprime en $W.m^{-2}.sr^{-1}$. Elle est alors définie par la somme de la radiance émise et de l'ensemble des radiances réfléchies. La radiance émise $L_e(x, \omega_o)$, par la surface au point x dans la direction ω_o sera typiquement importante dans le cas ou x se situe sur la surface d'une source de lumière primaire. L'intégrale permet quant à elle de prendre en compte les réflexions sur la surface des luminances $L_i(x, \omega_i)$ reçues depuis l'ensemble des directions Ω et réfléchies vers ω_o. Cette interaction est décrite au travers d'une fonction de réflectivité bidirectionnelle, ou BRDF (pour *Bidirectional Reflectance Distribution Function*) $fr(x, \omega_i, \omega_o)$ permettant de prendre en compte les propriétés physiques du matériau rencontré (albédo, rugosité, etc.). Elle dépend également de l'angle entre le rayon lumineux entrant et réfléchi $\cos\theta$. Nous pouvons alors remarquer que $L_i(x, \omega_i)$ est elle-même définie par une équation du rendu, rendant cette équation récursive. Dans les cas non triviaux, cette récursion est infinie.

La Figure 1.3 montre les notations utilisées pour décrire l'équation du rendu. Dans le cas de rendu avec un budget de temps élevé dit hors ligne (*offline*), les techniques utilisant l'approche de Monte Carlo sont devenues les plus populaires au cours du temps. Aujourd'hui, elles sont les plus utilisées dans les plus grands studios de production de films [Christensen and Jarosz, 2016].

La méthode de Monte Carlo permet d'approximer la valeur de l'intégrale d'une fonction à l'aide d'un échantillonnage des valeurs de celle-ci. L'estimateur de Monte Carlo se trouve être également un estimateur non biaisé de l'espérance. Pour obtenir des valeurs de l'équation du rendu aux échantillons aléatoires tirés, des chemins (polylignes 3D) sont tracés dans la scène 3D, depuis la caméra virtuelle, pour échantillonner et reconstruire la réponse couleur perçue par le capteur en chaque pixel. On peut représenter cette approximation comme essayant de trouver les chemins que des photons auraient pu emprunter pour aller de la source de lumière vers la caméra virtuelle.

1.3 Problématiques et contributions

Dans cette thèse, nous nous intéressons à la représentation de données pour effectuer de la compression de données à la volée. Nous nous sommes intéressés tout particulièrement à la représentation des rayons de lumière pour réduire au maximum leur empreinte mémoire dans le but d'accélérer les échanges réseau, et de ce fait, la rapidité des moteurs de rendus distribués exploitant les méthodes de Monte Carlo. Nous étendons également les contributions réalisées au domaine de l'informatique graphique au domaine des neurosciences en adaptant les outils méthodologiques ainsi développés. Après un état de l'art sur le rendu distribué de Monte Carlo et sur la compression de données permettant d'expliquer le choix des thématiques de recherche et d'introduire le domaine, nous introduisons dans le chapitre 3 une première contribution sous la forme d'un outil méthodologique permettant de réaliser la compression collaborative d'ensembles de vecteurs unitaires non ordonnés. Cette méthode est nommée `UniQuant` et réalise une transformation de l'espace de représentation en utilisant la cohérence entre les données. Nous présentons ensuite dans le chapitre 4 une analyse de l'impact de cette méthode quand elle est appliquée à des chemins de lumière. Nous montrons qu'il est possible de prendre en compte les propriétés des matériaux, en combinant cela avec la discrétisation des portails utilisés en rendu distribué pour arriver à un pipeline complet de compression d'ensemble de rayons. Enfin, nous présentons une contribution, dans le domaine des neurosciences, en appliquant `UniQuant` à des tractogrammes. Les tractogrammes du cerveau montrent les fibres qui connectent les zones d'influence neuronale, permettant aux neurochirurgiens de planifier les opérations, et aux chercheurs d'étudier le fonctionnement du cerveau. L'adaptation de `UniQuant` à la compression de tractogrammes se nomme `qfib`, et permet en quelques secondes de compresser des jeux de données de quelques dizaines de gigaoctets, là où les précédentes méthodes de l'état de l'art nécessitaient des dizaines de minutes pour réaliser la même opération.

1.4 Publications et Productions

1.4.1 Publications

Cette thèse a donné lieu aux productions scientifiques suivantes :

— **Rendu de Monte Carlo élastique**

Sylvain Rousseau et Tamy Boubekeur
JFIG 2016, pages 1–4 (publication conférence)
[Rousseau and Boubekeur, 2016]

— **Fast Lossy Compression of 3D Unit Vector Sets**
Sylvain Rousseau et Tamy Boubekeur
SIGGRAPH Asia Technical Brief 2017, pages 1–4, No. : 23 (publication conférence)
[Rousseau and Boubekeur, 2017]

— **QFib : Fast and Accurate Compression of White Matter Tractograms (Poster)**
Sylvain Rousseau*, Corentin Mercier*, Pietro Gori, Isabelle Bloch et Tamy Boubekeur
Organization on Human Brain Mapping 2019 (OHBM2019)
[Rousseau et al., 2019]

— **QFib : Fast and Efficient Brain Tractogram Compression**
Corentin Mercier*, Sylvain Rousseau*, Pietro Gori, Isabelle Bloch et Tamy Boubekeur
Volume 18, Pages 627–640 (2020)
NeuroInformatics
(publication journal)
[Mercier et al., 2020]

— **Unorganized Unit Vectors Sets Quantization**
Sylvain Rousseau et Tamy Boubekeur
Volume 9, No. 3, pages 92–107
Journal on Computer Graphics Technics (JCGT)
(publication journal)
[Rousseau and Boubekeur, 2020]

Certaines publications contiennent des co-premiers auteurs qui sont indiqués avec le symbole : *

1.4.2 Contributions logicielles

Le code de `UniQuant` (Chapitre 3) et de `qfib` (Chapitre 5) sont disponibles sur GitHub.

— UniQuant : `https://github.com/superboubek/UniQuant`
— Qfib : `https://github.com/syrousseau/qfib`

CHAPITRE 2

État de l'art

FIGURE 2.1 – Scène *Asian Dragon*, rendue en utilisant Hurricane.

2.1 Rendu de production

Comme expliqué dans l'introduction, durant la dernière décennie, le rendu de Monte Carlo s'est imposé comme la méthode standard dans les studios d'effets spéciaux et d'animations pour le rendu d'images réalistes [Christensen and Jarosz, 2016]. Bien que cette technique apparaisse comme une évidence de nos jours, la transition s'est faite très progressivement, au début des années 2000, partant d'un écosystème où de nombreux algorithmes de rendus différents cohabitaient. Dès la fin des années 90, le moteur de rendu *Arnold* [Georgiev et al., 2018] a très rapidement exploité la puissance du lancer de chemin. *RenderMan*, alors très largement utilisé dans de nombreux studios, reposait quant à lui sur l'algorithme *Reyes* [Cook et al., 1987] avec des cartes d'ombres (*Shadow Maps*) [Williams, 1978] et de réflections (*Reflection Maps*). Au début des années 2000, celui-ci a évolué pour exploiter des méthodes hybrides, combinant *Reyes* avec d'autres algorithmes tels que l'éclairage global par points (*Point Based Global Illumination*) [Christensen, 2008] pour améliorer le réalisme. Les méthodes de Monte Carlo reposent sur le lancer de chemin, algorithme simple à écrire, basé sur le lancer de rayons [Appel, 1968], permettant d'échantillonner l'espace des chemins possibles entre les sources de lumière et le capteur d'une scène virtuelle. Il se base sur les lois de la physique [Kajiya, 1986] lui permettant d'être robuste à une grande variété de configuration de scènes possibles et de prendre en compte une large variété d'effets. Cependant, durant cette même décennie, la résolution des images générées, la complexité des matériaux simulés, et la taille des scènes exploitées n'ont cessé de croître. Les sources de lumière se sont enrichies. La prise en compte d'effets physiques tels que l'iridescence ou les caustiques, est devenue la norme et a augmenté considérablement le nombre de calculs à réaliser pour obtenir des images toujours plus réalistes et riches artistiquement. Cette augmentation du nombre de calculs est partiellement compensée par l'augmentation de la puissance des machines. De nos jours, les plus grands studios utilisent des super-calculateurs rivalisant avec les machines les plus puissantes pour réaliser leurs films. De tels ordinateurs sont appelés des fermes de calculs et se composent de milliers de processeurs dédiés spécifiquement au rendu. Il aura par exemple fallu environ 6 mois de calculs sur un ordinateur disposant de 55 000 cœurs pour faire le rendu de *Les Nouveaux Héros* de Disney en 2014. Cela représente plus de 200 millions d'heures cœurs. Cette quantité de calculs peut

être effectuée par les plus grands studios, mais reste illusoire pour les plus petits d'entre eux qui ne peuvent réaliser de si grands investissements matériels. Il devient alors nécessaire de chercher une source alternative de puissance calculatoire pour permettre aux plus petits studios ou aux laboratoires de recherche publique de travailler sur des images approchant cette qualité. Dans cette quête de puissance, on s'intéresse alors aux sources de puissance de calculs "perdues". Dans les universités et dans les entreprises, les ordinateurs sont couramment utilisés moins de la moitié du temps. Ceux-ci sont peu puissants et disposent de peu de mémoire vive. Cependant, ils sont disponibles en grande quantité. Un seul de ces ordinateurs ne pourrait pas charger l'ensemble d'une scène, et nécessiterait de nombreuses heures pour faire le calcul d'une simple image. Une machine pourrait n'être disponible que quelques minutes. Le rendu de Monte Carlo distribué apparaît alors comme une réponse possible pour exploiter ce type d'infrastructure matérielle. Cette méthode permet également d'exploiter du matériel tel que celui fourni par des projets comme SETI@Home, ou les radiateurs de calcul permettant de chauffer les immeubles créés par Qarnot Computing. Cependant, pour exploiter ce type de machines, de nombreuses contraintes apparaissent concernant la conception du moteur de rendu. Les machines du cluster ne sont pas toutes de la même puissance, le cluster est donc dit hétérogène. Elles ne se situent pas sur un réseau local, impliquant une connexion lente par rapport aux connexions utilisées dans des fermes de rendu. Enfin, une machine peut tomber en panne, être éteinte, ou devoir être libérée pour que son propriétaire puisse l'utiliser. On doit donc lui attribuer des tâches petites, ou avoir une gestion intrinsèque des pannes.

2.2 Lancer de Chemin de Monte Carlo

Les méthodes de Monte Carlo [Metropolis and Ulam, 1949] sont une famille d'algorithmes ayant recours à des tirages de valeurs aléatoires pour calculer des valeurs numériques approchées. Celles-ci sont couramment utilisées dans les domaines où il est typiquement nécessaire de calculer des valeurs d'intégrales en haute dimension.

Comme expliqué dans le chapitre d'introduction, l'équation du rendu [Kajiya, 1986] (équation 1.1) est une équation récursive dont la solution exacte ne peut être calculée dans les cas non triviaux. La composante $L_i(\omega_i)$ est elle même définie par l'équation du rendu. Nous sommes donc en présence d'une équation de dimension infinie.

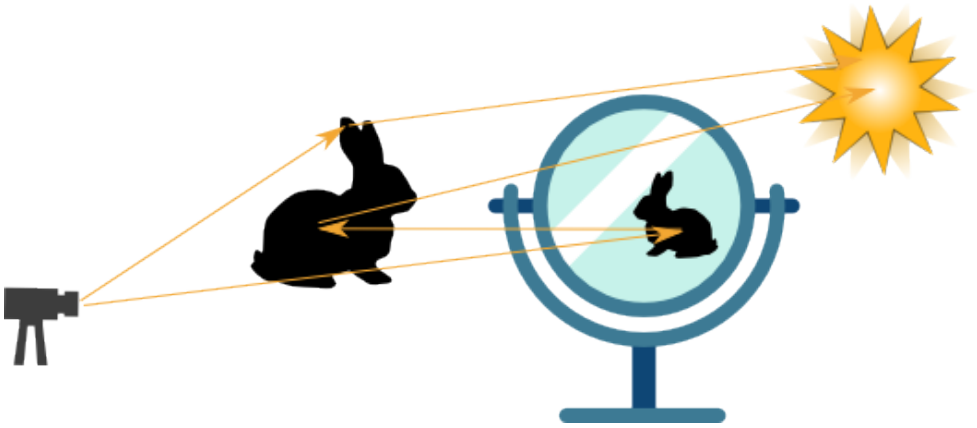


FIGURE 2.2 – Exemples de chemins lumineux

Pour l'approcher selon la méthode de Monte Carlo, des échantillons aléatoires sont alors construits en tirant aléatoirement une direction ω_i pour chaque dimension. Ce processus peut être visualisé comme étant celui de la construction de chemins explorant l'espace de manière stochastique sous la forme d'une polyligne 3D composée d'une suite de segments dont l'origine est la caméra virtuelle comme illustré dans la figure 2.2. L'avantage de cette technique est de ne s'intéresser qu'aux chemins lumineux qui atteignent de manière certaine la caméra virtuelle.

Les méthodes de Monte Carlo s'appuient sur la loi des grands nombres. Ainsi, avec cette technique, les premiers échantillons peuvent fournir une solution éloignée de la valeur réelle de l'équation du rendu, mais celle-ci converge ensuite à la vitesse de la racine carrée du nombre d'échantillons envoyés vers la solution exacte. Il est donc nécessaire d'envoyer pour chaque pixel de l'image que l'on souhaite générer de nombreux échantillons pour obtenir des images non bruités.

Il existe de nombreuses manières pour accélérer la convergence. La première d'entre-elles, décrite dans la méthode de Monte Carlo, est d'utiliser une fonction de densité de probabilité. Lors de la construction du chemin, nous savons que toutes les directions possibles pour le tirage aléatoire de la direction suivante n'ont pas la même chance de fournir une aussi grande intensité lumineuse. Nous connaissons par exemple les propriétés du matériau sur lequel le chemin se situe à chaque rebond, et l'absorption qui en résulte. Il est alors possible de fausser le tirage aléatoire de la prochaine direction pour prendre en compte cela, tout en divisant en conséquence la probabilité de

l'échantillon qui aura alors été tiré selon une loi non uniforme pour ne pas biaiser le résultat.

Des méthodes plus complexes existent pour trouver des meilleurs fonction de densité de probabilité, c'est notamment le cas du lancer de chemin guidé [Vorba et al., 2014] qui apprend les transferts lumineux dans la scène pour influencer la valeur de ces fonctions.

2.3 Rendu distribué

Lorsque l'on parallélise l'exécution d'un algorithme, on cherche à minimiser son *coût de parallélisation*. En effet, un algorithme s'exécutant en parallèle sur N machines ne sera pas N fois plus rapide car il est nécessaire d'exécuter une partie non parallèle pour contrôler la partie parallèle de l'algorithme. L'amélioration de la rapidité suit donc la loi d'Amdahl[Rodgers, 1985] indiquant que l'accélération A dépend de l'accélération de la partie parallélisée s et de la proportion de temps d'exécution du code qui est parallélisée p par rapport au code total. Elle se formule de la manière suivante.

$$A(s) = \frac{1}{(1-p) + \frac{p}{s}} \tag{2.1}$$

Le lancer de rayons et de chemins est un problème parfaitement parallèle, (*embarassingly parallel*) car la simulation du transport de la lumière est indépendante pour chaque chemin de lumière. Il est donc possible de paralléliser de manière simple le rendu en divisant la simulation par rayon ou par chemin de lumière, en assurant un coût de parallélisation minimum. Cependant, dans ce cas il est nécessaire de disposer de l'ensemble de la scène pour résoudre chaque chemin de lumière. Ceci pose alors le problème des scènes de production qui peuvent être trop lourdes pour pouvoir être gérées par des ordinateurs de faible puissance. Toutefois, il existe de nombreuses autres manières d'effectuer la parallélisation d'un rendu. Molnar et son équipe en ont proposé une classification dans leur état de l'art [Molnar et al., 1994]. Ils permettent de définir la distribution en fonction du type de données présentes sur chaque nœud de calculs.

2.3.1 Division de la simulation par rayon

La division du rendu par rayons ou par chemins lumineux apparaît comme la plus naturelle. C'est une approche qui a été appliquée à de multiples reprises, et que l'on retrouve dans les moteurs grands

publiques tels que Mitsuba [Nimier-David et al., 2019]. Elle est couramment appliquée de manière triviale, en divisant l'image finale en sous-images de petites tailles. L'ensemble de la scène (géométrie, lumières, matériaux, caméra) est envoyé sur chaque machine du cluster de calculs. Durant le rendu, les machines vont recevoir d'un ordinateur maître des coordonnées de sous-parties d'images à rendre. Une fois le calcul effectué et le résultat renvoyé, ils reçoivent alors de nouvelles sous-parties d'images à calculer. Cette architecture a l'avantage d'être simple et de s'adapter à des machines hétérogènes. En effet, un ordinateur de moindre puissance effectuera simplement le rendu d'un moins grand nombre de sous-parties de l'image. Nous montrons dans la partie suivante qu'il est possible d'exploiter cette stratégie en profitant des outils de calculs distribués mis au point dans le domaine du *cloud computing* en exploitant le patron d'architecture *MapReduce* [Dean and Ghemawat, 2004]. Celui-ci a été proposé par des ingénieurs de Google avec pour but d'effectuer des calculs sur des grandes quantités de données en exploitant la puissance d'un cluster. Il est devenu rapidement populaire, notamment grâce à l'apparition d'outils tels que Hadoop ou Spark.

2.3.2 Hurricane [Rousseau and Boubekeur, 2016]

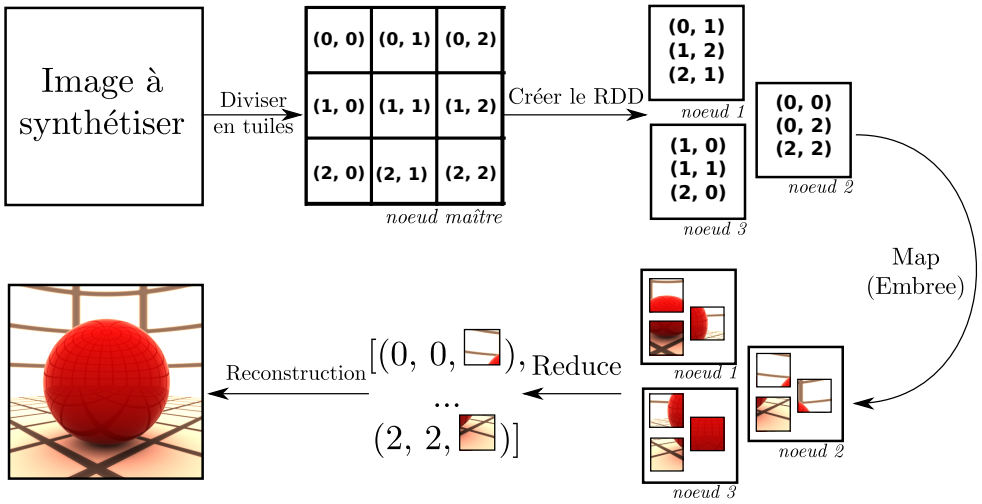


FIGURE 2.3 – Algorithme de rendu d'Hurricane

Pour pouvoir dégager des problématiques du rendu de Monte Carlo distribué, un premier travail a été réalisé pour exploiter des outils existants du calcul parallèle sur cluster de calculs. Nous avons ainsi développé Hurricane, un moteur de rendu reposant sur

Hadoop et exploitant une stratégie de parallélisation telle que décrite précédemment. L'implémentation utilise comme noyau de support Embree [Wald et al., 2014] mais pourrait bien entendu être adaptée à n'importe quel autre moteur de rendu.

Le système, décrit dans la Figure 2.3, est construit selon le patron d'architecture *MapReduce*. À ce titre, il s'appuie sur deux opérateurs issus de la programmation fonctionnelle : *map* et *reduce*. Avant l'exécution de ces deux opérateurs, une première étape d'initialisation est effectuée afin de diviser le travail en *éléments* qui sont envoyés sur les nœuds $\{T_i\}$. Ensuite, la fonction *map* applique une transformation à chaque élément. Cette transformation est effectuée de manière autonome et asynchrone par le nœud T_i. Enfin, la fonction *reduce* agrège les différents résultats produits indépendamment par chaque nœud pour obtenir le résultat final. En formulant l'algorithme selon *MapReduce*, la parallélisation est d'une part transparente pour le programmeur, et d'autre part permet d'exploiter un support d'exécution dédié et agnostique à l'algorithme opéré.

Dans le contexte de la synthèse d'une image, l'initialisation prend la forme d'un partitionnement de l'image à synthétiser en régions rectangulaires, que nous appellerons *tuiles*. Nous formulons ensuite l'opérateur *map* avec la séquence suivante :

1. Identification des coordonnées du sous-domaine de la tuile dans l'image ;
2. Génération, pour chaque pixel de la tuile, d'un ensemble de chemins et stockage de la moyenne des échantillons de radiance ainsi récoltés ;
3. Servir la tuile remplie.

Notre opérateur *reduce* agrège quant à lui les tuiles dans une liste désordonnée, qui une fois remplie est triée et utilisée pour reconstruire l'image finale directement sur le noeud maître.

Le système a alors l'avantage de pouvoir être rapidement déployé sur une grande flotte de machines hétérogènes.

Il est résistant aux pannes des nœuds travailleurs et passe à l'échelle comme le montre l'évolution des temps de calculs en fonction du nombre de nœuds travailleurs dans la figure 2.4. Dans le cas d'application visé, nous souhaitons faire le rendu de scènes de production. Avec les moteurs de rendu divisant le rendu selon les chemins lumineux, il est nécessaire d'avoir l'ensemble de la scène sur chaque nœud de calcul, ou un accès à une mémoire partagée [Wald et al., 2001, Ize et al., 2011, Rui and Yue, 2014] ou à une connexion

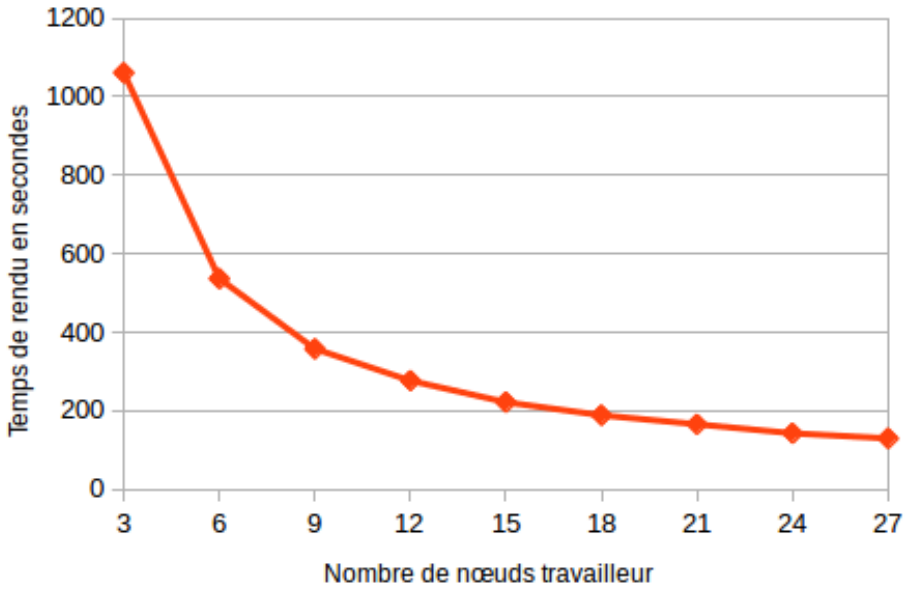


FIGURE 2.4 – Temps de rendu de la scène *Asian Dragon* (Figure 2.1)

performante [Gao et al., 2005, Peterka et al., 2008] comme *Hyperion* qui réalise des tris de rayons dans le but de minimiser les erreurs de cache [Eisenacher et al., 2013]. Les scènes de production peuvent peser des centaines de gigaoctets, qui ne peuvent pas être chargées par des ordinateurs de faible puissance, disposant de peu de mémoire vive, et potentiellement d'un disque dur trop petit et/ou trop lent pour avoir recours efficacement à des méthodes hors cœur.

Nous nous sommes donc intéressés aux méthodes ayant recours à une division de la scène.

2.3.3 Division de la scène et rendu basé portails

Les divisions de scènes sont couramment utilisées en visualisation de données scientifiques où l'on utilise beaucoup les grilles de voxels qui sont représentées sous forme d'octree ou selon d'autres structures hiérarchiques. La division est alors réalisée simplement en distribuant les branches de l'arbre. Un état de l'art complet de ce type de méthodes décrit le cas des rendus distribués de grilles de voxels [Beyer et al., 2015].

S'inspirant également du patron d'architecture MapReduce, des moteurs de lancer de chemins ayant recours à des divisions de scènes ont été développés. On notera notamment l'existence du moteur

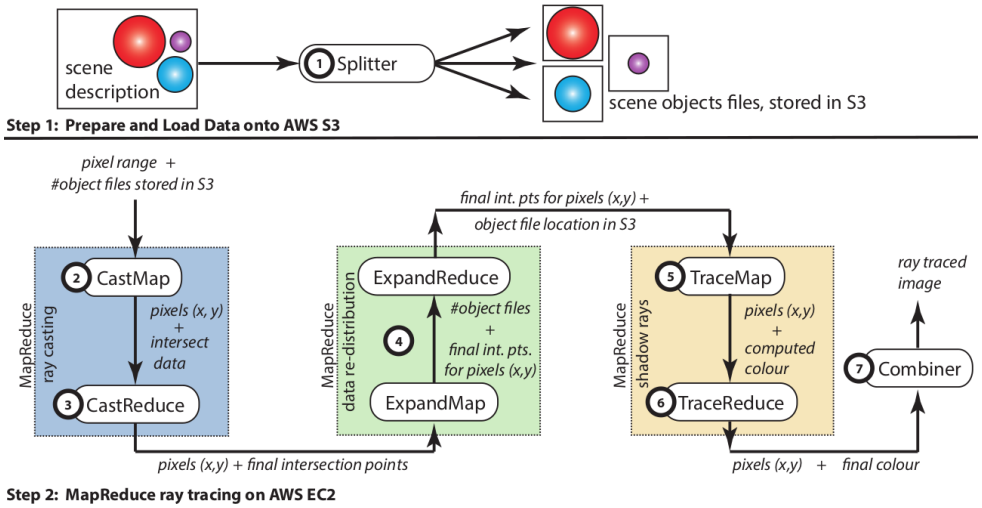


FIGURE 2.5 – Pipeline utilisé dans [Northam et al., 2013] (figure extraite de l'article)

proposé par l'équipe de Northman [Northam et al., 2013]. Dans cet algorithme présenté en Figure 2.5, la scène est d'abord divisée entre les nœuds travailleurs du cluster en attribuant aléatoirement les triangles à chaque machine. Durant le rendu, chaque rayon est ensuite envoyé à chaque machine qui effectue le test d'intersection rayon/triangles dans la phase de *map* et retourne l'intersection la plus proche. Durant la phase de *reduce*, l'intersection finale est calculée par un simple tri. Cet algorithme nécessite donc d'envoyer chaque rayon sur chaque ordinateur du cluster. Cela est possible en lancer de rayon, car peu de rayons sont envoyés pour générer l'image finale. Dans le cadre du lancer de chemin de Monte Carlo, des milliards de rayons sont nécessaires pour générer une seule image et peuvent représenter une plus grande quantité de données que la scène complète. Cette méthode ne peut donc pas fonctionner dans notre cas où le réseau peut être de mauvaise qualité. On trouvera des approches similaires dans le moteur de rendu de Square Enix, Kilauea [Kato and Saito, 2002] qui fonctionne également sur un réseau local.

Pour éviter d'envoyer chaque rayon à chaque machine, et donc pouvoir faire du rendu réaliste distribué, il est possible d'introduire la notion de portail. Chaque ordinateur contient alors une sous-partie de la scène contenue dans une enveloppe. Chaque nœud va traiter la simulation du transport de la lumière dans sa propre sous-partie de la scène, et lorsqu'un rayon intersecte un portail, il est envoyé au nœud contenu derrière le portail de la manière décrite dans la figure 2.6.

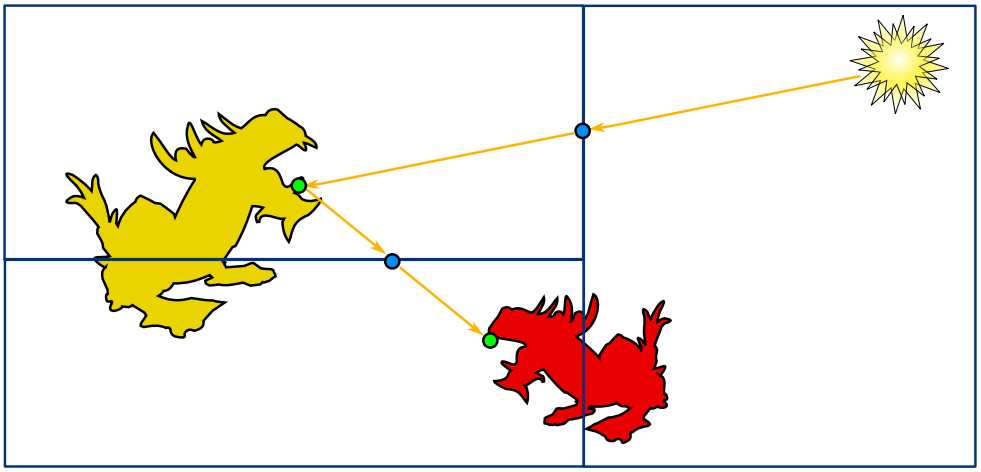


FIGURE 2.6 – Quand un photon intersecte un portail (en bleu) il est envoyé au nœud travailleur contenant la sous-scène derrière le portail via le réseau. Quand il intersecte un objet (en vert), l'intersection est gérée en interne.

On retrouve cet algorithme sous différentes formes en lancer de photons [Günther and Grosch, 2014] et en lancer de chemins [Somers and Wood, 2012]. Cette distribution permet de réduire au maximum le nombre d'échanges de rayons entre les nœuds du cluster pour un algorithme de rendu donné en maximisant le nombre de simulations traitées entièrement localement.

2.4 Nature des données

2.5 Compression de données

Pour aller plus loin dans le rendu distribué, notamment exploitant une architecture composée de très nombreux ordinateurs peu puissants distribués dans des lieux divers avec des connexions de faible puissance, il est nécessaire de s'intéresser à la réduction de la quantité de données échangées.

Pour ce faire, nous pouvons réduire le nombre de rayons envoyés ce qui est déjà le cas avec des portails bien choisis. Nous pouvons également utiliser les dernières techniques de rendu telles que le lancer de chemin guidé [Vorba et al., 2014].

Il est également possible de s'intéresser à la réduction de la taille des données transitant sur le réseau, à savoir, réaliser de la compression.

La compression de données est le domaine de l'informatique s'intéressant à la représentation minimale en espace mémoire d'un jeu de données donné par la complexité de Kolmogorov [Kolmogorov, 1963].

Les algorithmes de compression de données peuvent être catégorisés en fonction des métriques qu'ils cherchent à optimiser [Uthayakumar et al., 2018]. Ici, l'objectif est de fournir des nouveaux outils pour le rendu distribué. Dans cette optique, le but est d'optimiser la vitesse de la réduction de la taille des données. En effet, s'il est plus long de compresser puis d'envoyer les données que d'envoyer directement les données originales sur le réseau, la technique ne présente pas d'intérêt. Les méthodes permettant de réaliser une réduction de la taille des données en respectant ces contraintes sont couramment appelés méthodes de compression *à la volée*. Dans le cas de données générales, *LZ4* [Collet, 2011] une méthode populaire, permet d'obtenir d'excellentes performances en s'adaptant à une grande variété de données. Elle exploite des méthodes par dictionnaire permettant d'identifier et de supprimer les redondances. Cependant, cette flexibilité ne lui permet pas d'exploiter les contraintes mathématiques spécifiques dont nous pouvons disposer, notamment dans le cas de rayons lumineux. La première d'entre-elles étant que pour la majorité des composantes, les nombres à virgule flottante sont utilisés. Cette contrainte rend difficile l'utilisation des méthodes par dictionnaire et l'utilisation de méthodes plus spécifiques est alors nécessaire. Dans le cas de données à virgule flottante générales, *ZFP* [Lindstrom, 2014] est l'état de l'art actuel. Cette dernière reste cependant générale à tout type de données à virgule flottante et ne peut pas exploiter les contraintes géométriques.

2.5.1 Méthodes de compression spécifiques à l'informatique graphique

En informatique graphique 3D, la dimentionalité des données les rend lourdes en mémoire et a inspiré le développement de méthodes de compression pouvant exploiter les propriétés géométriques et mathématiques des données étudiées. S'agissant de compression de points dans l'espace, les méthodes vont de l'exploitation d'octree [Huang et al., 2006] aux méthodes plus adaptatives exploitant une division récursive de l'espace [Smith et al., 2012]. Ces méthodes exploitent toutes des arbres de partitionnement et remplacent les points par le point situé au centre de la cellule de la feuille dans

laquelle le point à compresser est contenu.

Les nuages de points viennent de scans d'objets et représentent des formes. Une classe de méthodes réalise la compression en exploitant la similarité géométrique entre les zones d'un nuage de points pour supprimer les redondances statistiques [Digne et al., 2014].

Les scènes utilisées en rendu exploitent plus souvent des maillages triangulaires, dont la compression a intéressé des équipes de recherche dès le milieu des années 90 [Deering, 1995]. L'encodage de la connectivité [Touma and Gotsman, 1998] a été prouvée comme étant quasi-optimale pour une compression à une résolution donnée.

Depuis, les avancées se sont faites au niveau des approches progressives [Maglo et al., 2012], ou à l'aide de partitionnements de l'espace [Peyré and Mallat, 2005]. Ces dernières méthodes ont l'avantage de permettre le traitement de maillages ayant des topologies plus complexes au détriment du taux de compression. De nouvelles méthodes ont émergé depuis et le domaine reste très actif en recherche [Maglo et al., 2015].

Ici, nous nous intéressons à la compression de rayons pour leur transfert sur le réseau. Alors que l'état de l'art en compression de points dans l'espace ou en nuage de points peut être utilisé pour la compression des origines de rayons, il apparaît que ces méthodes ne sont pas adaptées pour la compression des directions.

2.5.2 Nature des données

Dans cette thèse, nous nous intéressons à la compression des directions des rayons dans le cas du rendu de Monte Carlo distribué se reposant sur des portails de lumière. Dans ce contexte, les données traitées sont des ensembles de vecteurs unitaires donc les propriétés statistiques des rayons qui ont une intersection avec les portails sont difficiles à connaître a priori sans effectuer un pré-calcul qui correspondrait lui-même à un rendu. En effet, leurs propriétés dépendent de la géométrie et des matériaux de la scène, mais également des sources de lumière présentes. Lorsqu'aucune distribution de probabilité ne peut être considérée, la distribution uniforme est alors la meilleure distribution possible. Dans de tels cas, on écarte les méthodes classiques de l'état de l'art telles que le codage entropique.

2.5.3 Méthodes de compression de vecteurs unitaires

Une des composantes du rayon à laquelle nous nous intéressons est la direction. Celle-ci est un vecteur unitaire qui peut être représenté mathématiquement par un point sur la surface d'une sphère unitaire. Cette information permet de limiter de manière conséquente la taille de l'espace de représentation des données, en passant de l'ensemble de $\mathbb{R}^3$ à une partie de $\mathbb{R}^2$. S'agissant de la compression de l'origine de rayons lumineux, *ZFP* s'avère être très puissant, mais dans le cas de la direction, il est primordial d'exploiter la nature unitaire des vecteurs. L'état de l'art en matière de compression de vecteurs unitaires repose sur les méthodes de quantification [Cigolle et al., 2014]. Il en existe une large variété permettant de s'adapter à de nombreux cas d'applications. Elles sont rapides et permettent d'obtenir des hauts taux de compression moyennant une petite perte de données. Nous évaluons l'impact de ces pertes sur le rendu de Monte Carlo dans le Chapitre 4. Dans le Chapitre 3, nous exploitons deux méthodes couramment utilisées : la quantification par ensemble de points sphériques de Fibonacci [Keinert et al., 2015] qui permet d'obtenir la meilleure précision en contrepartie d'une complexité calculatoire importante et la quantification octaédrique [Meyer et al., 2010], méthode très populaire de par son rapport entre la précision de la quantification et son très faible coût calculatoire. Bien que l'évaluation soit réalisée sur ces deux méthodes, il est à noter que `UniQuant` est utilisable avec n'importe quelle méthode de quantification. Nous noterons notamment l'existence d'autres méthodes populaires, telle que HealPix [Górski et al., 2005], développée dans le domaine de l'astro-physique, permettant d'obtenir des cellules disposant d'une surface égale pour chaque cellule de quantification, ou l'optimisation de coordonnées sphériques pour l'obtention d'un ensemble de points plus adapté à la quantification [Smith et al., 2012].

Principe de fonctionnement de la quantification par ensemble de points sphériques de Fibonacci. [Keinert et al., 2015]

Les méthodes de quantification de vecteurs unitaires définissent un ensemble de points sur la surface de la sphère unitaire, et compressent les vecteurs en les remplaçant par des points de cet ensemble. Pour obtenir l'erreur de quantification (angle entre le vecteur d'origine et celui ayant été compressé et décompressé) la plus faible, il est donc nécessaire de trouver l'ensemble de points ayant la distribution la plus

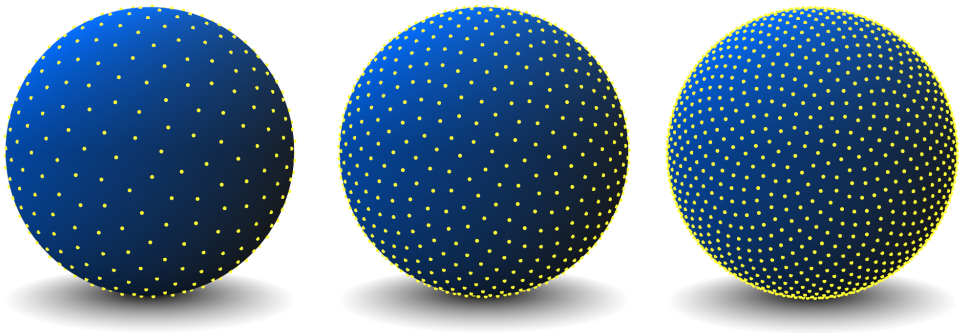


FIGURE 2.7 – Ensembles de points sphériques de Fibonacci avec 500, 1000 et 2000 points.

uniforme possible sur la surface de la sphère unitaire. C'est-à-dire, un ensemble de points dont les cellules de Voronoï sur la surface de la sphère de Gauss sont d'aires les plus égales possible en ayant une forme s'approchant d'une calotte sphérique. La quantification par ensemble de points sphériques de Fibonacci [Keinert et al., 2015] est actuellement l'état de l'art en terme de précision de la quantification. Elle repose sur l'ensemble de points sphériques de Fibonacci [González, 2010], dont la distribution est quasi-uniforme sur la surface de la sphère unitaire. Les coordonnées sphériques (ϕ, θ) des points de la suite peuvent être aisément calculés. Pour un point j de l'ensemble contenant K points, les coordonnées peuvent être obtenues à l'aide de l'équation suivante.

$$\begin{cases} \theta_j & = arccos(1 - \frac{2j+1}{K}) \\ \phi_j & = 2j\pi((3 - \sqrt{5})/2) \end{cases} \tag{2.2}$$

Cette formule permet d'effectuer la décompression d'un point dont la version quantifiée (compressée) est donnée par l'identifiant j correspondant au point le plus proche du vecteur d'origine dans la suite. La figure 2.7 permet d'observer cet ensemble de points pour différentes valeurs de K. Lorsque l'on compresse un vecteur avec une précision de M bits, le nombre de points utilisés dans la suite sphérique de Fibonacci sera de $K = 2^M$.

Alors qu'à partir d'un vecteur compressé avec l'identifiant j, il est aisé de trouver le vecteur décompressé avec l'équation 2.2, trouver l'identifiant j à partir d'un point sur la surface de la sphère unitaire n'est pas trivial. Les points se suivent en s'enroulant autour de la sphère selon une spirale, et une cellule contenant le point à compresser est entouré par 4 d'entre eux pouvant être plus ou

moins écartés dans la suite de points. Une méthode par dichotomie est envisageable, mais coûteuse algorithmiquement. L'équipe de Keinert [Keinert et al., 2015] a proposé une méthode permettant de faire cela avec une complexité algorithmique constante. L'idée est d'arriver à identifier la cellule entourée de 4 points de la suite dans laquelle le vecteur unitaire se trouve. Il suffit ensuite de comparer chacune des 4 distances pour trouver le plus proche voisin. Cette méthode nécessite de faire un changement d'espace de représentation, de calculer 4 distances et de les comparer. Elle est donc relativement lourde malgré sa faible complexité algorithmique théorique constante par rapport à d'autres algorithmes de quantification. Il est également à noter que cette méthode ne permet pas d'effectuer des quantifications trop précises. Du fait des imprécisions numériques, la méthode est limitée à environ 8 millions de points de Fibonacci, à savoir une quantification sur 23 bits.

Principe de fonctionnement de la quantification octaédrique. [Meyer et al., 2010]

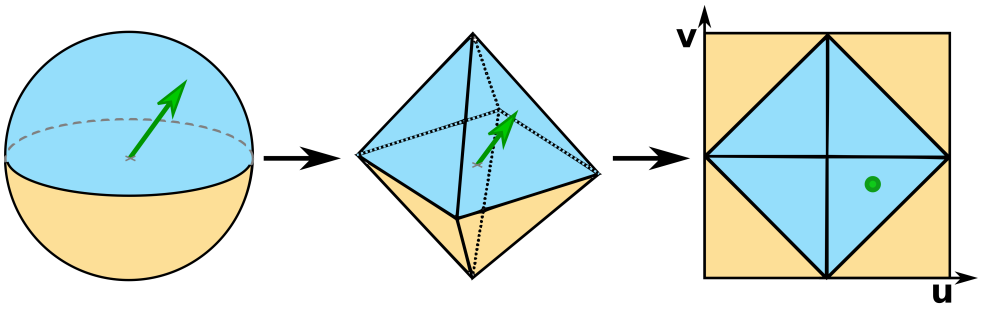


FIGURE 2.8 – La quantification octaédrique ([Meyer et al., 2010]) projette le vecteur unitaire sur un octaèdre, puis sur le carré unitaire pour encoder la discrétisation des coordonnées 2D résultantes.

La quantification octaédrique a été conçue pour utiliser efficacement les capacités matérielles. Ceci est fait en réalisant deux transformations : une transformation des points représentatifs des vecteurs unitaires depuis la surface de la sphère unitaire vers un octaèdre, puis l'exploitation d'une paramétrisation de cet octaèdre sur le carré unitaire 2D. Le vecteur compressé est alors donné par la discrétisation des coordonnées. La transformation depuis la sphère vers l'octaèdre est réalisée avec la norme L1. La paramétrisation dans le carré unitaire 2D est quant à elle réalisée selon le schéma illustré dans la figure 2.8. Les coordonnées (u, v) résultantes sont ensuite discrétisées avant

d'être encodées. Lorsque l'on effectue une quantification sur M bits, chaque coordonnée est discrétisée sur $M/2$ bits. Le vecteur peut alors être décompressé en utilisant les formules suivantes ([Meyer et al., 2010]), avec σ la fonction de signe donnant $\sigma(x) = 1$ pour $x \geq 0$ et $\sigma(x) = -1$ dans les autres cas :

$$\begin{aligned} z &= 1 - |u| - |v| \\ [x, y] &= \begin{cases} [u, v]^T & \text{si } z \geq 0 \\ [\sigma(v) - v, \sigma(u) - u]^T & \text{si } z < 0 \end{cases} \end{aligned} \tag{2.3}$$

Cette méthode permet d'obtenir un très bon ratio entre la précision de la compression et sa rapidité. Elle est donc un très bon candidat pour les cas d'applications de compression de données à la volée.

2.6 Conclusion

Pour maximiser le nombre de contraintes respectées par rapport à notre problématique, le rendu distribué se basant sur une division de la scène exploitant des portails apparaît comme la meilleure solution. Elle permet d'assurer que la scène pourra être chargée par des machines peu puissantes, en divisant suffisamment celle-ci. De plus, elle assure une minimisation des échanges réseau en effectuant autant que possible les simulations du transport de la lumière localement. Le nombre d'échanges réseau à réaliser lors d'une simulation du transport de la lumière complète reste cependant conséquent, tout particulièrement dans le cas du lancer de chemins dans lequel il est nécessaire d'envoyer à chaque rebond de la lumière des rayons d'ombre vers chaque source de lumière. Ces derniers parcourent de longues distances, et augmentent énormément le nombre d'échanges sur le réseau qui peuvent rapidement devenir le goulot d'étranglement de l'ensemble de la simulation. Le lancer de photons ne présente pas cette contrainte liée aux rayons d'ombre, mais converge plus lentement sur les cas de scènes les plus répandus. Pour limiter les échanges réseau, il est possible d'adopter deux stratégies orthogonales. On peut diminuer au maximum le nombre de rayons échangés sur le réseau, en utilisant les dernières méthodes permettant une convergence plus rapide telle que le lancer de chemins guidés [Vorba et al., 2014]. On peut également exploiter des méthodes de compression de données à la volée. Dans cette thèse, nous nous intéressons à la deuxième possibilité, et plus particulièrement à la représentation des rayons de lumière dans le but de réduire au maximum leur empreinte mémoire pour des échanges réseau plus efficaces.

CHAPITRE 3

Représentation d'ensembles de vecteurs unitaires

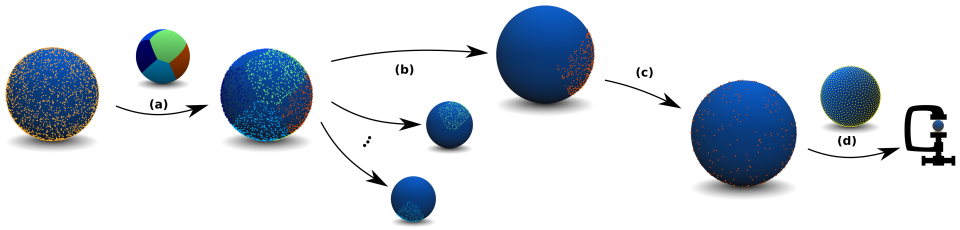


FIGURE 3.1 – (a) Les vecteurs unitaires représentés ici à l'aide de leurs points représentatifs sur la surface de la sphère unitaire sont d'abord regroupés. (b) Chaque groupe est ensuite traité indépendamment en parallèle. (c) Pour chacun d'eux, on applique une *transformation uniforme* déplaçant les vecteurs unitaires depuis la surface des sous-parties vers l'ensemble de la surface de la sphère unitaire. (d) Les vecteurs peuvent alors être quantifiés avec des méthodes classiques de quantification de vecteurs unitaires avec une précision améliorée.

3.1 Objectif

Les rayons de lumière employés dans les différents moteurs de rendu contiennent un minimum d'information géométrique. Celle-ci est composée d'une origine, d'une direction et peut être complétée par d'autres informations sur la micro-structure des surfaces. Dans ce chapitre, nous nous intéressons à l'amélioration des méthodes de compression pour l'une de ces composantes : la direction, couramment exprimée à l'aide d'un vecteur de $\mathbb{R}^3$ normalisé.

Cette écriture, bien que nécessaire pour l'efficacité des calculs, est éloignée de l'espace de représentation minimal de ces vecteurs, donné par la complexité de Kolmogorov [Kolmogorov, 1963]. En effet, étant normalisées, leurs coordonnées cartésiennes tridimensionnelles correspondent à des points sur la surface de la sphère unitaire. On appelle ces points des points représentatifs du vecteur unitaire sur la surface de la sphère unitaire. Ils sont facilement représentables à l'aide d'un vecteur de $\mathbb{R}^2$ avec les coordonnées sphériques. Pour un vecteur unitaire indépendant aléatoire, cela correspond à l'espace de représentation minimal.

Dans notre cas d'application, un moteur de rendu distribué, nous travaillons avec des milliards de rayons, et nous nous intéressons donc à la compression d'ensemble de données. Nous introduisons dans ce chapitre `UniQuant`, un modèle de représentation d'ensemble de vecteurs unitaires permettant d'exploiter efficacement la cohérence des données pour en réduire l'emprunte mémoire. Cette cohérence peut être préexistante, lorsque l'on utilise un algorithme de rendu effectuant un tri des rayons tel que celui utilisé dans Hyperion [Eisenacher et al., 2013], ou générée à l'aide d'une méthode de regroupement des données.

Pour évaluer l'efficacité de la méthode proposée, nous avons étudié deux types de données. Des données théoriques composées d'ensembles de vecteurs unitaires générées artificiellement et des données réelles, des nuages de points disposant de normales tels que l'on peut en obtenir lorsque l'on réalise un scan 3D d'un objet réel. Nous avons également étudié l'impact de l'utilisation de divers schémas de regroupements permettant, en fonction du type d'application visé, de privilégier la rapidité ou la précision de la compression. Ces derniers créent des groupes de vecteurs unitaires dont les points représentatifs sur la surface de la sphère de Gauss sont contenus dans des sous-parties approximables par des calottes sphériques.

L'idée clé de la méthode de représentation proposée est d'appliquer une fonction de transformation permettant de déplacer les points depuis ces sous-parties, vers l'ensemble de la surface de la sphère unitaire, domaine de support des méthodes de quantification des vecteurs unitaires.

Ces méthodes forment actuellement l'état de l'art en compression de vecteurs unitaires et la précision accrue fournie par `UniQuant` permet alors à précision égale de réduire le nombre de bits nécessaire pour l'encodage de la même quantité information. Ceci est fait avec une méthode de faible complexité algorithmique permettant de rester efficace dans le contexte d'algorithmes de compression à la volée.

Ce travail a fait l'objet d'un papier court, présenté lors de la conférence SIGGRAPH Asia 2017 : **Fast Lossy Compression of 3D Unit Vector Sets**, Sylvain Rousseau and Tamy Boubekeur, *ACM SIGGRAPH Asia 2017 Technical Brief.* [Rousseau and Boubekeur, 2017] et a été accepté comme papier long dans *Journal of Computer Graphics Techniques*.

3.2 Algorithme

3.2.1 Idée générale

Chaque vecteur unitaire est représenté à l'aide de son point représentatif sur la surface de la sphère unitaire. La méthode se résume alors avec les trois étapes suivantes :
1 - Regrouper les points représentatifs en fonction de leur localisation spatiale sur la surface de la sphère unitaire
2 - Traiter individuellement chaque groupe de vecteurs. Transformer chaque point représentatif dont l'espace de représentation est limité par l'enveloppe convexe du groupe auquel il appartient en un autre point contenu sur la surface de l'ensemble de la sphère unitaire.
3 - Quantifier les points représentatifs transformés avec une méthode de quantification de vecteurs unitaires.

Le principe de la méthode proposée est d'adapter l'ensemble des méthodes de quantification de vecteurs unitaires indépendants existantes à des groupes de vecteurs. Pour ce faire, nous prenons en entrée un ensemble de vecteurs unitaires. Ceux-ci peuvent être de deux formes différentes. Ils peuvent soit disposer d'une cohérence spatiale par construction des données, auquel cas la première étape

de la méthode proposée peut être ignorée, ou générer cette dernière au travers d'algorithmes tels que ceux qui sont utilisés dans certains moteurs de rendu sur les rayons [Novák et al., 2010, Van Antwerpen, 2011, Eisenacher et al., 2013, Laine et al., 2013]. Dans ce dernier cas, une première étape de regroupement (Section 3.2.2) est donc appliquée pour créer la cohérence des données. Dans le but de garder ce chapitre indépendant, nous montrerons les résultats en y ajoutant l'étude de l'étape du regroupement. Elle permet de former des ensembles de vecteurs spatialement cohérents délimitant des sous-parties de la sphère unitaire.

En utilisant la méthode que nous proposons, les points représentatifs appartenant à ces sous-parties sont transformés en d'autres points définis sur la surface de l'ensemble de la sphère unitaire, domaine de support des méthodes de quantification de vecteurs unitaires. En effet, un groupe ne contenant que des vecteurs sur une sous-partie de la surface de la sphère unitaire, une grande partie des points de quantification ne seront jamais exploités. Cette transformation peut également se visualiser comme étant la transformation des points de quantification depuis l'ensemble de la surface de la sphère unitaire, vers la surface du groupe de vecteur. Cela conduit à la création d'un ensemble de quantification spécifiquement adapté à un sous-ensemble de données dont la densité sera accrue, et l'erreur induite réduite.

Cette transformation est réalisée à l'aide d'une fonction de transformation introduite dans la section 3.2.3. Elle s'assure de conserver l'uniformité des distributions de points dans le but de minimiser l'erreur moyenne et maximale de l'étape de quantification en conservant les bonnes propriétés des ensembles de points. Chaque vecteur est ensuite encodé selon le schéma défini dans la section 3.2.5. Conséquemment, l'algorithme fournit le tableau contenant les groupes de données compressés et, dans le cas de données ayant été réordonnées, un second tableau permettant de retrouver l'indice de chaque vecteur dans le tableau d'origine.

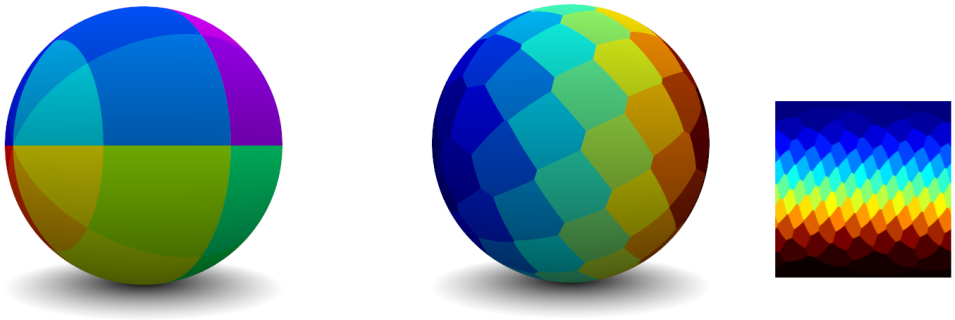
Dans le type d'application souhaité, à savoir un moteur de rendu distribué, l'ordre des rayons n'est pas important (chaque rayon vient d'un différent chemin de lumière). De plus il est courant d'envoyer un identifiant unique avec le chemin de lumière. Le second tableau peut donc être supprimé dans ces cas. Dans d'autres cas, ce tableau sera conservé par le noeud maître pour prendre en compte de manière correcte les résultats des calculs qui auront été effectués sur un noeud travailleur.

3.2.2 Regroupement des vecteurs unitaires

La méthode que nous proposons, baptisée `UniQuant`, s'inspire des méthodes de compression exploitant la cohérence dans des petits groupes de données. Elles fonctionnent en encodant les échantillons à l'aide de leur décalage par rapport à des valeurs moyennes. [CCITT, 1992]

Dans le contexte d'un moteur de rendu, l'ordre dans lequel les rayons sont envoyés n'est pas important. Ils peuvent donc être réordonnées pour créer des petits groupes de rayons partageant des directions fortement cohérentes. Dans le cas d'un nuage de points avec des normales, l'ordre des données n'est également pas important car il n'encode pas de relation de voisinage ou de connexité. Il est donc également possible de les réordonner pour obtenir des grands groupes de données spatialement cohérentes. De plus, dans certains moteurs de rendu, les rayons sont triés durant le rendu et la cohérence ainsi créée peut être exploitée à la place de notre étape de regroupement [Novák et al., 2010, Van Antwerpen, 2011, Eisenacher et al., 2013, Laine et al., 2013]. `UniQuant` effectue son étape de transformation en approximant chaque groupe par une calotte sphérique. La méthode de regroupement la plus efficace devrait donc fournir des groupes d'aires aussi égales que possible et de forme approchant au maximum d'une calotte sphérique. Dans le cas contraire, les points de quantification contenus entre le groupe et la calotte sphérique englobante seront inutilisés, et la précision de la quantification sera en conséquence diminuée.

Nous présentons et étudions ici deux méthodes de regroupement possibles, plus ou moins précises et plus ou moins coûteuses en terme de calculs. La première exploite une paramétrisation de la sphère de Gauss et permet de réaliser cette étape rapidement avec un très faible coût mémoire en utilisant les coordonnées sphériques discrétisées. La seconde se dérive sous deux formes distinctes utilisant toutes deux le point le plus proche de l'ensemble de points sphériques de Fibonacci pour déterminer l'appartenance d'un vecteur à un groupe. C'est la méthode la plus précise, mais son coût calculatoire est important. C'est la raison pour laquelle nous étudions également l'utilisation d'une carte de mise en correspondance permettant de pré-calculer l'appartenance à un groupe. Diminuer la qualité du groupement et en augmenter la quantité de mémoire nécessaire permet de réduire grandement les temps de calculs nécessaires



(a) (b)

FIGURE 3.2 – Les vecteurs unitaires peuvent être regroupés à l'aide de différentes méthodes. Ici, nous étudions l'utilisation de deux d'entre elles. (a) Les **c**oordonnées **s**phériques **d**iscretisées (CSD) sont les plus rapides, mais fournissent des groupes de tailles irrégulières dont la forme s'écarte d'une calotte sphérique qui serait optimale pour notre méthode. (b) Le plus proche voisin de l'ensemble de **p**oint **s**phérique de **F**ibonacci (PSF) permet d'obtenir la plus petite erreur maximale et moyenne, au détriment d'un plus grand coût calculatoire. Il est également possible d'avoir recours à cette méthode en utilisant une table de mise en correspondance pré-calculée, donnant la correspondance entre les points de la surface de la sphère unitaire et le point sphérique de Fibonacci correspondant. Cela améliore les performances en coût calculatoire, en augmentant légèrement les erreurs minimales et maximales, au détriment d'un plus grand coût mémoire lors du calcul. Cette variante est dénotée PSF-MC dans les résultats.

Regroupement selon les coordonnées sphériques discrétisées (CSD)

Une méthode simple et efficace pour regrouper des points est de construire des ensembles partageant les mêmes bits de poids forts du code de Morton [Morton, 1966] de leurs coordonnées sphériques discrétisées. Le code de Morton est construit en entrelaçant les bits des deux valeurs binaires. Des instructions sont présentes sur les processeurs modernes pour réaliser cela efficacement, via le jeu d'instruction BMI2. Le calcul de l'appartenance à un groupe peut donc s'effectuer de manière très simple et efficacement en ayant recours uniquement à des opérations sur les bits. La Figure 3.2.a montre une représentation colorisée de ce regroupement. La complexité linéaire en nombre de vecteurs, la faible quantité de mémoire utilisée et la

rapidité d'exécution de cette méthode en font un très bon candidat, malgré la non-uniformité des groupes réduisant la qualité de la compression. De plus, avec cette méthode, il est très facile de calculer le vecteur moyen, nécessaire par la suite. L'ensemble des vecteurs d'un groupe partagent les mêmes N bits de poids fort. Nous appelons ces N bits la *clé* du groupe. Le vecteur moyen peut alors être approximé par le vecteur dont le code de Morton est *clé*$+((\sim masque)/4)$ en nommant $masque$ la valeur binaire avec ses N bits de poids fort à 1 et ses autres bits à 0 (masque de bits). Ses coordonnées cartésiennes sont ensuite calculées en effectuant successivement les opérations suivantes :

(i) extraction des coordonnées sphériques discrétisées du code de Morton.

(ii) inversion de la normalisation de celles-ci.

(iii) transformation des coordonnées sphériques en coordonnées cartésiennes.

Le vecteur moyen peut ainsi être calculé à partir de n'importe quel vecteur du groupe. Avec cette stratégie de regroupement, il est à noter que les valeurs paires de N fournissent des groupes plus uniformes et proches des caractéristiques recherchées. Elles doivent donc être favorisées du fait d'un effet de pas.

Regroupement selon l'ensemble de points sphériques de Fibonacci (PSF)

Comme indiqué précédemment, dans l'étape de transformation les groupes sont approximés par des calottes sphériques. Pour réduire l'erreur maximale autant que possible, il faut que la taille de la plus grande des calottes sphériques soit la plus petite possible. Une manière pour trouver une bonne solution à ce problème est de répartir le centre des groupes aussi uniformément que possible. Cela revient au même problème que celui de la quantification de vecteurs unitaires pour lequel l'ensemble de points sphériques de Fibonacci est la solution de l'état de l'art. Nous effectuons donc le regroupement en assignant chaque vecteur au point sphérique de Fibonacci le plus proche grâce à la méthode de l'équipe de Keinert [Keinert et al., 2015]. Cependant, cette méthode s'avère être trop lente pour une compression à la volée. Nous proposons alors une alternative pré-calculant une table de mise en correspondance comme illustré dans la figure 3.2b. Elle permet d'accélérer les calculs moyennant une légère perte en précision et un surcoût en mémoire durant l'exécution.

3.2.3 Transformation uniforme des calottes sphériques

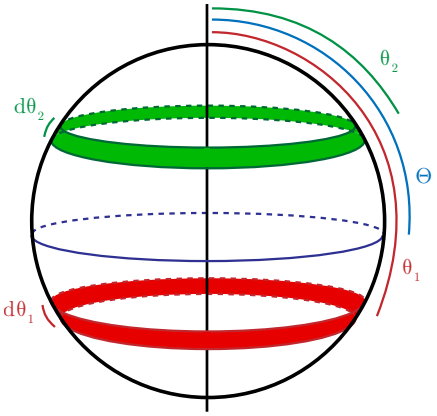


FIGURE 3.3 – Notations utilisées dans Section 3.2.3 pour définir la fonction de transformation.

Notre schéma de compression est basé sur les méthodes de quantification de vecteurs unitaires. Ces méthodes correspondent à une compression avec perte dont l'erreur est définie par l'angle entre le vecteur décompressé et le vecteur d'origine. Dans le cadre de données dont la distribution est inconnue, la quantification minimisant l'erreur est donnée par un ensemble de points distribués uniformément sur la surface de la sphère unitaire, en associant un vecteur unitaire au point le plus proche dans cet ensemble de points. En utilisant des méthodes de regroupement précédemment introduites (Section 3.2.2), ou des données ayant une forte cohérence, il est possible de définir par groupe des sous-surfaces de sphère de Gauss que l'on nomme *surface d'un groupe*. L'espace de quantification est exploité à pleine puissance lorsque le vecteur est défini à la surface de l'ensemble de la sphère unitaire, or, dans notre cas, celle-ci n'est pas utilisée dans son ensemble mais est restreinte à la surface du groupe. Nous proposons ici une fonction de transformation, permettant de changer l'espace de représentation des vecteurs à quantifier depuis la surface du groupe vers celle de l'ensemble de la surface de la sphère unitaire. Une fois celle-ci appliquée, il est alors possible d'utiliser plus efficacement les méthodes de quantification.

Cette méthode doit être introduite en respectant un certain nombre de propriétés mathématiques permettant de minimiser l'erreur. Pour ce faire, nous introduisons tout d'abord comme indiqué dans la figure 3.3 les constantes mathématiques utilisées pour définir la transformation.

Nous définissons ensuite les aires des surfaces rouges (dS_1) et vertes (dS_2) dans l'équation 3.1.

$$\begin{aligned} dS_1 &= 2\pi r^2 \sin(\theta_1) d\theta_1 \\ dS_2 &= 2\pi r^2 \sin(\theta_2) d\theta_2 \end{aligned} \tag{3.1}$$

Dans l'équation 3.2, nous définissons S_{cap}, la surface d'une calotte sphérique (région de la sphère définie par l'ensemble des points de la surface formant un angle inférieur à une valeur Θ par rapport à un vecteur donné). Dans la suite, nous nommerons *vecteur moyen* le vecteur situé au centre de la surface d'un groupe.

$$\begin{aligned} S_{cap} &= 2 \int_{\theta \in [0,\Theta]} \pi r^2 \sin\theta d\theta \\ &= 2\pi r^2 (1 - \cos\Theta) \end{aligned} \tag{3.2}$$

Pour respecter une conservation de la distribution des données sur la surface, permettant d'éviter de créer des accumulations ou sous-densités de vecteurs sur la surface, il est nécessaire de trouver un mapping permettant de conserver le rapport entre la taille de la surface de groupe par rapport à la surface d'origine et la taille de la surface de groupe transformée sur la taille de la surface de la sphère :

$$\frac{dS_1}{S_{sphere}} = \frac{dS_2}{S_{cap}} \tag{3.3}$$

Il est alors possible d'effectuer une réécriture de chacun des termes de ces rapports :

$$\frac{dS_1}{S_{sphere}} = \frac{2\pi r^2 \sin\theta_1 d\theta_1}{4\pi r^2} = \frac{\sin\theta_1 d\theta_1}{2}$$

$$\frac{dS_2}{S_{cap}} = \frac{2\pi r^2 \sin\theta_2 d\theta_2}{2\pi r^2 (1 - \cos\Theta)} = \frac{\sin\theta_2 d\theta_2}{1 - \cos\Theta}$$

La fonction de transformation cherchée s'exprime en fonction de l'angle d'origine et de l'angle d'arrivée par rapport au vecteur moyen, et nous cherchons $h : \theta_1 \to \theta_2$. En utilisant l'équation 3.3, on obtient alors :

$$\sin\theta_2 d\theta_2 = \frac{1 - \cos\Theta}{2} \sin\theta_1 d\theta_1$$

$$-d(\cos\theta_2) = \frac{1 - \cos\Theta}{2} (-d(\cos\theta_1))$$

$$\cos\theta_2 = \frac{1 - \cos\Theta}{2} \cos\theta_1 + c$$

En utilisant le seul vecteur dont nous pouvons connaître la position finale, nous définissons alors le fait que $h(0) = 0$. Cela implique que le vecteur moyen sera transformé en lui-même.

$$c = 1 - \frac{1 - \cos\Theta}{2}$$

On peut alors calculer θ_2 en fonction de θ_1 :

$$\cos\theta_2 = 1 + \frac{1 - \cos\Theta}{2}(\cos\theta_1 - 1)$$

$$\theta_2 = \text{acos}\left(1 - \frac{1 - \cos\Theta}{2}(1 - \cos\theta_1)\right) \tag{3.4}$$

L'équation 3.4 nous donne la transformation depuis θ_1 vers θ_2 comme illustré dans la Figure 3.3. Nous la réécrivons de la manière suivante :

$$h(\theta_1) = \text{acos}\left(1 - \frac{1 - \cos\theta_1}{k}\right) \text{ avec } k = \frac{2}{1 - \cos\Theta}$$

On remarque alors que la transformation inverse a la forme suivante :

$$h^{-1}(\theta_2) = \text{acos}(1 - k(1 - \cos\theta_2)) \tag{3.5}$$

Il est alors intéressant de remarquer dans l'équation 3.5 que la même fonction de transformation peut être utilisée pour la compression et la décompression, en utilisant $1/k$ pour la compression, et k pour la décompression.

La fonction h nous fourni une transformation d'un angle vers un autre angle. Nous appliquons cette transformation au vecteur unitaire directeur. Soit $\mathbf{x}$ le vecteur en entrée, $\mathbf{x}'$ le vecteur transformé et $\mathbf{P_0}$ le vecteur moyen normalisé. Nous définissons alors :

$$\begin{aligned} l_0 &= \text{acos} < \mathbf{P_0}, \mathbf{x} > \\ l_1 &= h(l_0) \\ &= \text{acos}\left(1 - \frac{1 - < \mathbf{P_0}, \mathbf{x} >}{k}\right) \end{aligned}$$

Soit $\mathbf{P_1}$, un vecteur orthogonal à $\mathbf{P_0}$, dans le plan contenant $\mathbf{x}$ et $\mathbf{P_0}$.

$$\mathbf{P_1} = \frac{\mathbf{x} - < \mathbf{x}, \mathbf{P_0} > \mathbf{P_0}}{||\mathbf{x} - < \mathbf{x}, \mathbf{P_0} > \mathbf{P_0}||}$$

En utilisant $\mathbf{P_0}$ et $\mathbf{P_1}$ comme base orthonormée, le vecteur $\mathbf{x}'$ est alors défini de la manière suivante :

$$\mathbf{x}' = \cos(l_1)\mathbf{P_0} + \sin(l_1)\mathbf{P_1}$$

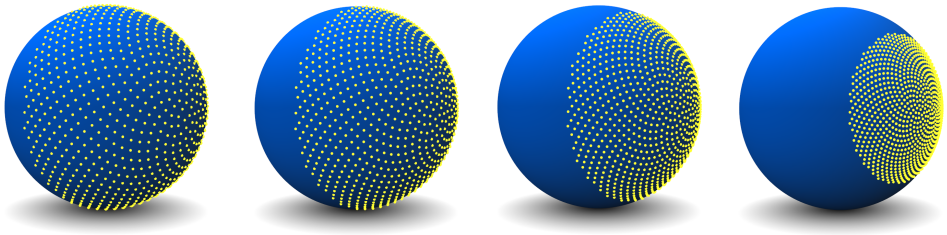


FIGURE 3.4 – Fonction de transformation appliquée à une hémisphère de l'ensemble de points sphériques de Fibonacci. De gauche à droite : ensemble de points originaux et transformation uniforme avec k = 0.75, 0.5 et 0.25.

nous donnant la transformation d'un point $\mathbf{x}$ en un point $\mathbf{x}'$:

$$\mathbf{x}' = c\mathbf{P_0} + \sqrt{1-c^2}\mathbf{P_1} \text{ avec } c = 1 - \frac{1 - < \mathbf{P_0}, \mathbf{x} >}{k} \tag{3.6}$$

Cette transformation peut être aisément implémentée comme montré dans l'algorithme 3.1. Son utilisation sur l'ensemble de points sphériques de Fibonacci est illustrée dans la figure 3.4.

Listing 3.1 – code C++ pour la fonction de transformation

```
vec3 mapping(vec3 & x, vec3 & p0, double ratio)
{
  double k = ratio;
  double d = dot(x, p0);
  vec3 p1 = normalize(x - d * p0);
  double c = (1 - ((1 - d) / k));
  return  p1 * sqrt(1 - (c * c)) + (c * p0);
}
```

3.2.4 Ratio

Pour un groupe de points contenu dans une calotte sphérique C , le paramètre k de la transformation uniforme (Équation 3.6) doit être égale à la valeur du ratio entre la taille de la projection de C sur l'axe $\overrightarrow{OP_0}$ et la projection de l'ensemble de la sphère sur ce même axe (le diamètre). Lorsque l'on utilise le groupement CSD, les groupes ne sont pas de forme et de taille uniformes. Pour un groupe donné disposant du vecteur moyen P_0 et de bords du quadrilatère sphérique représentant ses frontières B_i avec $i \in 1, 2, 3, 4$, le ratio est de : $\frac{1-(min(\vec{A}\cdot\vec{B_i}))}{2}$. Cette valeur peut être légèrement sous-estimée à cause de l'erreur de quantification pouvant amener à des points en

dehors de la zone et à des imprécisions numériques. Pour palier à ce problème, dans le cas général, nous ajoutons une faible valeur ϵ dans notre implémentation. Quand nous utilisons le groupement selon l'ensemble de points sphériques de Fibonacci, k est calculé une fois au début et est utilisé pour l'ensemble des groupes.

3.2.5 Schéma de compression

L'étape précédente transforme des régions correspondant à des sous-parties de la surface de la sphère unitaire vers l'ensemble de celle-ci. Nous pouvons alors utiliser n'importe quelle méthode de quantification de vecteurs unitaires existante avec une précision améliorée spécifiquement pour ce groupe de vecteurs. Si l'erreur minimale est recherchée, la quantification selon l'ensemble de points sphériques de Fibonacci est recommandée, car elle est l'état de l'art en terme de précision. Comme indiqué en introduction, l'exploitation de cette méthode de quantification est réalisée grâce à la méthode proposée par Keinert et collaborateurs [Keinert et al., 2015]. A cause des imprécisions numériques, en réalisant les calculs en précision double, cette méthode est limitée à une quantification sur 23 bits. Lorsque la rapidité d'exécution est le critère principal, comme cela est le cas dans le cadre du rendu distribué, la quantification octaédrique [Meyer et al., 2010] permet d'obtenir un bon rapport entre rapidité et précision de la quantification comparativement à d'autres méthodes [Cigolle et al., 2014]. Étant donnée une méthode de quantification, on stocke pour chaque groupe le nombre de vecteurs compressés, ainsi que les vecteurs transformés avec `UniQuant` puis quantifiés. L'identifiant (position dans le tableau) du groupe de vecteur donne la *clé* du groupe. Comme les vecteurs moyens peuvent être retrouvés grâce à cette clé (voir Section 3.2.2), et le ratio avec cette clé et le nombre de groupes, ils ne nécessitent pas d'être stockés dans le tableau. L'algorithme complet de compression peut alors être résumé de la manière suivante :

(i) On commence par construire des groupes de vecteurs spatialement cohérents, c'est-à-dire étant contenus dans les mêmes sous-parties de la sphère unitaire.

(ii) Pour chaque groupe, on enregistre le nombre de vecteurs unitaires présents et on applique à chacun d'entre eux la fonction de transformation permettant de les replacer sur l'ensemble de la surface de la sphère unitaire.

(iii) On enregistre enfin la quantification des vecteurs transformés

Le schéma de quantification est illustré dans la Figure 3.5.

Groupe 1			...	Groupe n	
N_1	$C_{1\,1}$	... $C_{1\,N_1}$	...	N_n	$C_{n\,1}$ ... $C_{n\,N_n}$

FIGURE 3.5 – Schéma de compression des groupes. N_i (respectivement C_i) est le nombre de vecteurs unitaires compressés (respectivement les vecteurs compressés) dans le i^{ieme} groupe.

3.3 Implémentation et résultats

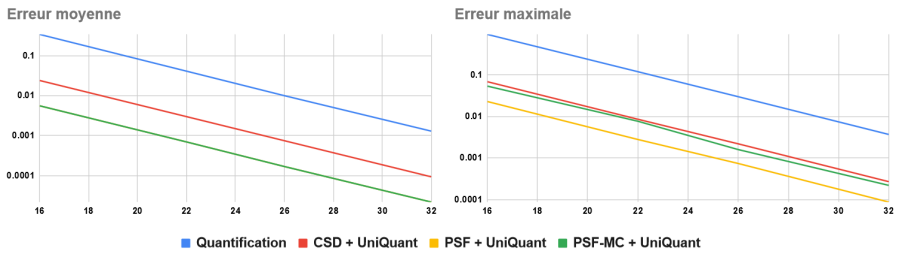
3.3.1 Implémentation

L'implémentation a été réalisée sur le CPU en C++ en utilisant GLM pour les calculs sur les vecteurs et OpenMP pour la parallélisation. Les tests ont été réalisés sur un processeur Intel Xeon E5-1630 v4 (quad core, 3.7 GHz). Quand le groupement basé sur les coordonnées sphériques (DSC) est utilisé, nous utilisons LibMorton pour calculer de manière efficace le code de Morton grâce au jeu d'instruction BMI2. À cause des erreurs de précision numérique, la transformation est effectuée en précision double. Chaque test est réalisé sur 10 millions de vecteurs unitaires générés avec une distribution aléatoire uniforme. Pour la quantification octaédrique, l'implémentation proposée par Cigolle et al. [Cigolle et al., 2014] a été utilisée. L'implémentation de la quantification selon l'ensemble de points sphériques de Fibonacci a été réalisée, et est disponible avec le code de `UniQuant` en ligne sur GitHub et en matériel additionnel des papiers publiés sur le sujet.

3.3.2 Performances et erreurs

Nous mesurons les erreurs selon l'angle en degré entre le vecteur original et celui qui est compressé puis décompressé. Cette métrique d'erreur est celle utilisée pour évaluer les méthodes de compression dans l'état de l'art sur les méthodes de quantification de vecteurs unitaires indépendants réalisé par l'équipe de Cigolle [Cigolle et al., 2014].

Dans le tableau 3.1, nous montrons l'impact de la méthode de regroupement choisie sur la précision de la quantification, ainsi que l'apport de la méthode par rapport à une quantification seule. Ces résultats sont présentés pour 2^{13} groupes. C'est à dire 2^{13} points



Méthode	Quantification		CSD et *UniQuant*	
Erreur	moyenne	maximale	moyenne	maximale
16 bits	3.4×10^{-1}	9.5×10^{-1}	2.4×10^{-2}	6.9×10^{-2}
22 bits	4.1×10^{-2}	1.2×10^{-1}	3.0×10^{-3}	8.6×10^{-3}
26 bits	1.0×10^{-2}	3.0×10^{-2}	7.5×10^{-4}	2.2×10^{-3}
32 bits	1.3×10^{-3}	3.7×10^{-3}	9.4×10^{-5}	2.7×10^{-4}
Méthode	**PSF et *UniQuant***		**PSF-MC et UniQuant**	
Erreur	**moyenne**	**maximale**	**moyenne**	**maximale**
16 bits	5.6×10^{-3}	2.3×10^{-2}	5.6×10^{-3}	5.4×10^{-2}
22 bits	7.0×10^{-4}	2.8×10^{-3}	7.0×10^{-4}	7.7×10^{-3}
26 bits	1.7×10^{-4}	7.4×10^{-4}	1.7×10^{-4}	1.6×10^{-3}
32 bits	2.2×10^{-5}	8.7×10^{-5}	2.2×10^{-5}	2.2×10^{-4}

TABLE 3.1 – Erreur de quantification en degrés sur 10 millions de vecteurs unitaires aléatoires uniformément distribués, en fonction de la précision de la quantification, et de la méthode utilisée pour effectuer le regroupement avant la transformation. Ce regroupement a été effectué en utilisant 13 bits pour la clé des CSD et le nombre équivalent de groupe pour PSF/PSF-MC. PSF-MC correspond à PSF appliqué avec une table de mise en correspondance de 500×500. La méthode de quantification appliquée est l'octaédrique.

pour PSF et une clé de 13 bits pour CSD. Nous évaluons les erreurs moyennes et maximales.

Nous observons alors que la méthode de regroupement des vecteurs a un grand impact sur l'efficacité de la transformation. Par exemple, alors que l'utilisation de CSD permet d'obtenir un gain d'un peu plus de 6 bits par rapport à une quantification classique comme illustré par les cases surlignées en vert, PSF permet un gain de plus de 10 bits comme indiqué par les cases surlignées en rouge.

Comme indiqué précédemment, un groupe est approximé par une calotte sphérique, et la zone contenue entre le groupe et la calotte sphérique englobante contiendra des points de quantification qui seront inutilisés justifiant cette différence. Cependant, alors que PSF est capable de traiter les 10 millions de points en 3.86 secondes,

CSD est capable de regrouper les données en seulement 1 seconde. Dans un cas d'application à la volée comme visé ici, CSD sera alors privilégié car il permet de traiter 114 Mo/s de données alors que PSF ne pourra en traiter que 29 Mo/s. L'utilisation de la table de mise en correspondance permet d'améliorer grandement les performances, et lorsque celle-ci fait une taille de 500×500, le groupement peut être effectué en 1.35 secondes, soit une bande passante de 85 Mo/s. Elle réduit cependant la précision de la méthode de groupement. Malgré son temps de calculs plus important, cette méthode a un intérêt dans le cas d'un réseau non local peu puissant. L'étape de transformation prise seule est capable de traiter plus d'1 Go/s de données. L'utilisation du regroupement PSF permet de réduire à une précision donnée l'erreur moyenne par un facteur 60 et l'erreur maximale par un facteur 42.

Quelle que soit la méthode choisie pour le groupement, le nombre de groupes a un impact sur les performances. Un nombre plus faible de groupes diminue les temps de calculs mais diminue également l'efficacité de la transformation. De plus, pour chaque groupe, nous devons encoder le nombre de vecteurs. Il n'est donc pas possible de choisir un nombre trop important de groupes et il doit dépendre du nombre de vecteurs à encoder. La décompression est indépendante de la technique de groupement choisie et ne dépend que de la méthode de quantification choisie. Ici, sur les exemples montrés sur le tableau 3.1, c'est la quantification octaédrique qui est choisie. Elle réalise la décompression des 10 millions de vecteurs en 0.18 seconde.

3.4 Application de UniQuant à la compression de nuages de points avec normales pour la reconstruction de surfaces.

Dans cette partie, `UniQuant` est illustré sur un premier cas d'application simple : la compression de nuages de points disposant de normales. Dans ce contexte, le schéma de quantification peut être appliqué de manière directe.

3.4.1 Reconstruction de surfaces

Les scanners tridimensionnels permettent de numériser des objets réels pour en obtenir une représentation virtuelle. Différentes technologies existent pour réaliser ceci, allant de la reconstruction à partir d'ensembles de photos d'un objet selon différents points de vue, jusqu'aux scanners laser en passant par des méthodes non optiques permettant de reconstruire des objets transparents, hautement spéculaires, ou disposant de beaucoup d'auto-occlusions [Aberman et al., 2017]. Dans le cas des scanners optiques, on obtient couramment des nuages de points, donnant un échantillonage des surfaces ainsi que la normale à la surface. Dans de nombreux cas d'applications (affichage, simulation physique, etc.), il est préférable de disposer d'un maillage triangulaire. Le passage du nuage de points à un maillage triangulaire s'appelle une reconstruction de surface [Berger et al., 2017]. La reconstruction de surface par les méthodes de Poisson exploitent les points ainsi que les normales. Ces dernières sont utilisées comme une mesure locale de l'espace tangent de la surface recherchée.

3.4.2 Compression de nuages de points

Bien que `UniQuant` ne puisse pas rivaliser avec les méthodes de compression de nuages de points hors ligne [Cao et al., 2019] telles que la quantification par répartition spatiale récursive [Smith et al., 2012], ou des méthodes plus poussées comme l'exploitation de la similarité de surfaces [Digne et al., 2014], notre méthode présente l'avantage de pouvoir traiter les points à la volée durant le scan, sans avoir besoin d'attendre d'avoir de nombreux points pour commencer la compression. Ces ensembles de points sont couramment utilisés pour faire de la reconstruction de surface. Nous étudions l'impact de `UniQuant` sur la quantification de points avec normales sur le cas d'application typique : la reconstruction de surface par les méthodes de Poisson. Pour se faire, nous appliquons notre algorithme de compression sur différents ensembles de données dont les caractéristiques sont présentées dans la Figure 3.6.

3.4.3 Résultats

Nous évaluons la qualité de la compression en comparant celle-ci à une reconstruction faite à partir de l'ensemble de points n'ayant pas subit de compression. Cette erreur quadratique est montrée pour la

Octocat
47k points

Dragon
900k points

Lucy
14M points

FIGURE 3.6 – Nuages de points utilisés pour l'évaluation de l'impact de la méthode de compression sur les méthodes de reconstruction de surface à partir de nuage de points.

Modèle	Octocat		Dragon		Lucy	
UniQuant	Oui	Non	Oui	Non	Oui	Non
2 bits	$3.6.10^{-5}$	$1.5.10^{-3}$	$4.3.10^{-4}$	$1.2.10^{-2}$	$8.2.10^{-5}$	$1.6.10^{-3}$
4 bits	$1.7.10^{-5}$	$2.3.10^{-4}$	$9.2.10^{-5}$	$2.4.10^{-3}$	$6.9.10^{-5}$	$2.5.10^{-4}$
8 bits	$1.4.10^{-5}$	$2.2.10^{-5}$	$2.3.10^{-5}$	$2.3.10^{-4}$	$6.8.10^{-5}$	$7.1.10^{-5}$
16 bits	$1.4.10^{-5}$	$1.4.10^{-5}$	$2.3.10^{-5}$	$2.5.10^{-5}$	$6.8.10^{-5}$	$6.8.10^{-5}$

TABLE 3.2 – Erreur quadratique de la reconstruction de surface de Poisson entre la reconstruction appliquée sans compression des normales, et celle appliquée après compression et décompression avec `UniQuant` et la quantification octahédrique.

quantification octaédrique, combinée ou non avec `UniQuant` dans le tableau 3.2.

Nous observons alors que l'erreur quadratique converge rapidement vers l'erreur minimale atteignable avec une quantification sur 8 bits lorsque nous utilisons le groupement CSD avec une clé de 13 bits. Sans transformation, il est nécessaire d'avoir recours à une quantification sur 16 bits.

3.5 Perspectives

Nous avons introduit `UniQuant`, une méthode pour compresser des ensembles de vecteurs unitaires non ordonnés. Cette méthode peut être appliquée à toute sorte de données n'ayant pas besoin de

conserver d'ordre durant les calculs, et avons démontré son utilité dans le cas de la compression de nuages de points munis de normales, issus de scanners 3D. Ce type de données est cependant courant en informatique graphique et dans d'autres domaines de l'informatique. Dans cette thèse, nous nous intéressons tout particulièrement au rendu de Monte Carlo distribué. Nous examineront dans le prochain chapitre l'utilisation possible de cet algorithme avec cet objectif. Nous montrerons également dans le Chapitre 5 que l'exploitation de cette méthode peut s'appliquer à d'autres domaines que celui de l'informatique graphique, et qu'avec une adaptation, il est possible d'exploiter cette technique en imagerie médicale.

CHAPITRE 4

Compression de rayons de lumière pour le rendu de Monte Carlo distribué

FIGURE 4.1 – Scènes utilisées pour étudier l'impact de UniQuant sur le rendu de Monte Carlo. De gauche à droite : *staircase*, une scène principalement diffuse, *glass* principalement spéculaire, and *living room*, avec un mélange de types de matériaux. Les scènes sont issues du répertoire de partage de Benedikt Bitterli [Bitterli, 2016]

4.1 Rendu distribué de Monte Carlo

Le rendu distribué de Monte Carlo consiste à utiliser une architecture matérielle composée de plusieurs ordinateurs pour effectuer le rendu d'une image unique. Dans le cadre de cette thèse, nous nous intéressons à ce type d'architecture en exploitant un grand nombre de machines peu puissantes sur des réseaux non locaux. Comme expliqué dans le Chapitre 2, pour exploiter ce type de machines, il est préférable d'utiliser des moteurs reposant sur une division de la scène avec des portails de lumière. Dans de tels cas, la

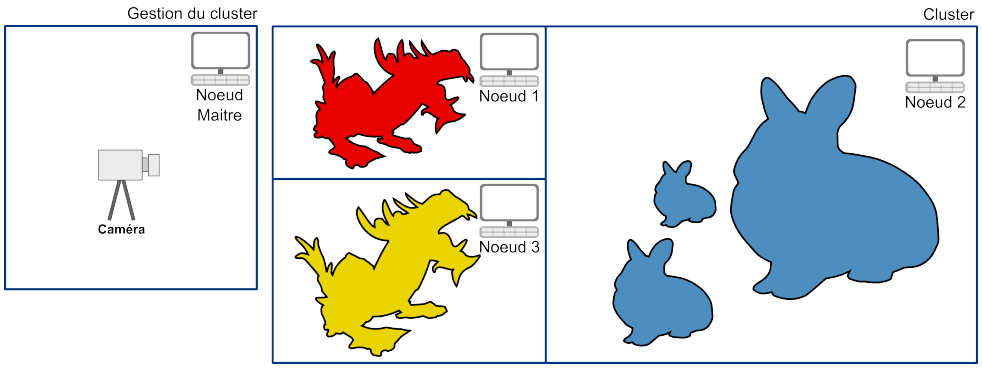


FIGURE 4.2 – Architecture matérielle pour le rendu distribué de Monte Carlo. Chaque partie de la scène est attribuée à un noeud du cluster selon une division spatiale.

scène est décomposée en sous-ensembles, et des portails sont placés sur les enveloppes du domaine spatial de chaque sous-scène comme illustré dans la figure 4.2. La structure de portail en rendu de Monte Carlo fournit à n'importe quel instant du rendu une discrétisation du champ de lumière sur lequel on applique une réduction de dimensions. Chaque machine dispose alors d'une sous-scène. Les chemins lumineux sont ensuite traités de manière classique sur chaque machine, mais, dès qu'un portail est rencontré en place de géométrie classique, le rayon et les statistiques déjà accumulées sont envoyées au noeud du cluster qui est lié au portail. Cela est représenté dans la figure 4.3.

Avec un grand nombre de machines, on va augmenter en conséquence le nombre de portails et les échanges réseau. Comme le montre le tableau 4.2, il est nécessaire d'envoyer plusieurs centaines de millions de rayons pour une simple image haute définition. Des milliards sont nécessaires pour une image pleinement haute résolution (1920 x 1080 pixels) et nous nous dirigeons à grand pas vers la 4K. Cela représente des quantités phénoménales de données qui deviennent vite problématiques à transférer sur un réseau non local. Il est alors nécessaire de s'intéresser à réduire le nombre de données transférées sur le réseau, soit en réduisant le nombre de rayons envoyés, avec les dernières méthodes de rendu tel que le lancer de chemin guidé [Vorba et al., 2014], soit en les compressant. Nous nous intéressons ici à cette dernière problématique. La compression de données se divise en deux grandes classes de méthodes, à savoir celles avec pertes et celles sans pertes. Nous nous intéressons ici à la compression de données avec pertes qui nous permet d'obtenir des

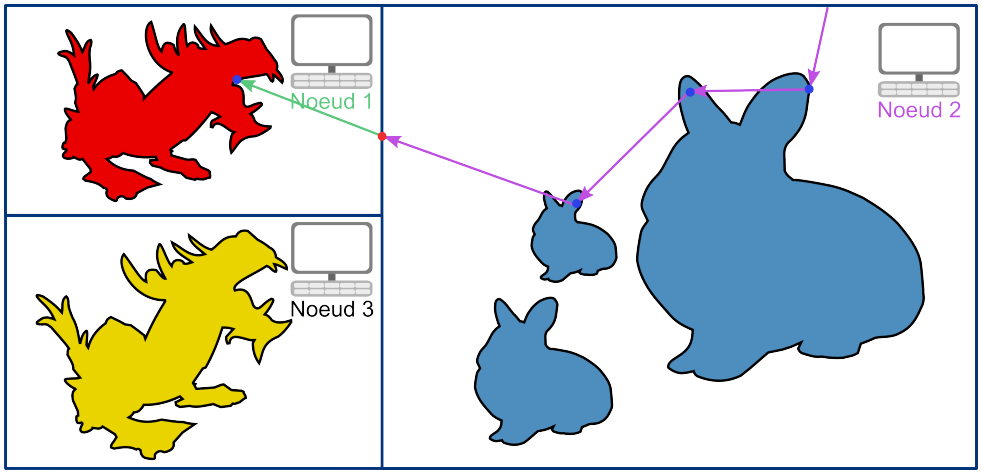


FIGURE 4.3 – Quand un chemin de lumière rencontre la géométrie de sa sous-scène (point bleu), il est traité localement. Lorsqu'il rencontre un portail (point rouge), il est envoyé avec toutes les statistiques déjà accumulées sur l'ordinateur contenant la sous-scène contenue derrière le portail pour continuer son chemin.

grands taux de compression avec une grande rapidité d'exécution, qui correspond à notre cas d'application où la vitesse de réduction des données doit être plus rapide que la vitesse de transfert réseau. Le rendu de Monte Carlo est un processus d'intégration par tirage aléatoire permettant d'estimer la valeur de l'équation du rendu (équation 1.1). Cette part aléatoire le rend très sujet au bruit, surtout lorsque peu d'échantillons sont utilisés. La moindre variation dans la direction ou l'origine d'un rayon peut alors avoir un impact important sur la couleur finale d'un pixel. Cependant, il est intéressant de voir si ce biais a un impact fort sur la convergence vers la valeur exacte de l'équation du rendu. Nous étudions donc ici une adaptation de Uniquant, la méthode présentée précédemment, à des rayons lumineux et étudions son impact sur la qualité des images générées dans le cadre du rendu distribué.

4.2 Objectif

Au Chapitre 3, nous avons défini une méthode permettant de réaliser la compression d'ensembles de vecteurs unitaires désordonnés. Dans le cas du rendu de Monte Carlo, l'ordre dans lequel sont envoyés les rayons n'est pas important, chaque chemin lumineux étant parfaitement indépendant l'un de l'autre. Nous

pouvons donc appliquer `UniQuant` au rendu de Monte Carlo, tout particulièrement, comme indiqué en introduction, au rendu de Monte Carlo distribué. Dans ce chapitre, nous commençons donc par appliquer cette méthode directement aux rayons d'un moteur de rendu complet (PBRTv3 [Pharr et al., 2016]) avant d'en améliorer l'approche en prenant en compte les matériaux. Enfin, la direction ne représentant qu'une partie des rayons, nous nous sommes aussi intéressés à leur origine. Pour ce faire, nous nous reposons sur les portails de lumière utilisés sur les moteurs de rendus distribués visés par cette thèse.

4.3 Méthode

Durant le rendu de Monte Carlo distribué avec des portails, on accumule des chemins à envoyer sur le réseau au niveau de ces derniers. Il est alors possible de les envoyer par grands groupes ouvrant alors la possibilité d'avoir recours à des méthodes de compression d'ensembles de rayons. En effet, les chemins sont alors composés de statistiques accumulées, et de rayons de lumière. Nous étudions ici la compression par groupe de rayons lumineux désordonnés. Nous utilisons pour la compression des directions `UniQuant`, avec un regroupement sur 13 bits, et nous étudions également la discrétisation 2D des origines des rayons sur la surface des portails. Cela permet d'obtenir une compression complète du rayon (origine + direction) dont l'impact de la perte est évaluée sur des rendus effectués sur diverses scènes.

4.4 Impact de UniQuant sur le rendu de Monte Carlo distribué

Pour étudier l'erreur introduite par `UniQuant` sur le rendu de Monte Carlo quand il est appliqué aux chemins lumineux, c'est-à-dire à l'ensemble des rayons sauf les rayons d'ombres qui peuvent profiter d'autres méthodes de compression, nous avons implémenté le schéma de quantification dans un moteur de lancer de chemins complet. Nous avons fait varier les paramètres du rendu pour étudier la convergence vers l'image de référence. Nous avons utilisé PBRTv3 [Pharr et al., 2016] sur les trois scènes montrées dans la figure 4.1 allant de la plus spéculaire (*glass*) à la plus diffuse (*staircase*).

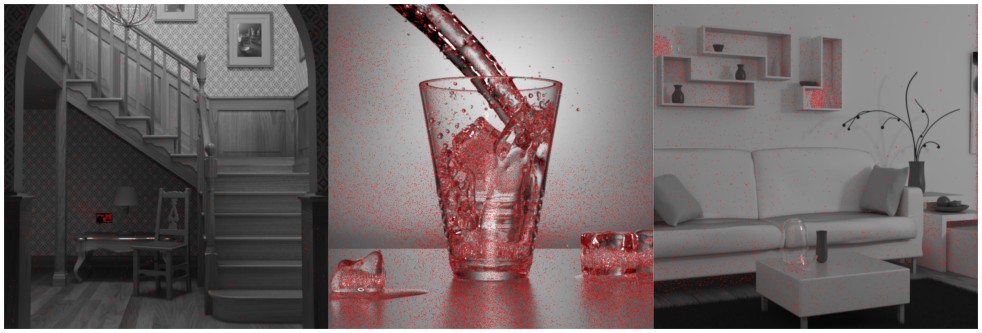


FIGURE 4.4 – **Localisation des erreurs de compression sur les rendus.** Les images sont les rendus avec la quantification octaédrique sur 16 bits et le mapping. Les pixels rouges correspondent à ceux ayant une grande différence par rapport au pixel du rendu avec le même nombre d'échantillons sans quantification.

Les rendus ont été effectués à 512 échantillons par pixel avec différents niveaux de quantification, en utilisant ou non `UniQuant`. Cela permet de montrer son impact positif sur la qualité des images obtenue par rapport à l'utilisation de méthodes de quantification. Enfin, nous avons également fait les mêmes rendus non quantifiés à 512 échantillons par pixel, ainsi que ceux des images que nous supposerons avoir convergées. On nomme ces dernières *valeurs exactes*, aussi couramment appelées *groundtruth* dans la littérature, même si elles ne correspondent qu'à une approximation de la valeur exacte réelle qui reste elle inatteignable dans les cas non triviaux. Elles ont été calculées avec 65536 échantillons par pixel. Pour les comparaisons, nous utilisons la similarité structurelle (*Structural Similarity* ou SSIM)[Wang et al., 2004] qui permet d'obtenir une différence perceptuelle entre deux images, avec des valeurs allant de 0 pour des images complètement différentes à 1 pour des images identiques, et une mesure de similarité basé sur le PSNR (*Peak Signal-to-Noise Ratio*) représentant un pourcentage (entre 0 et 100) de similarité calculé selon la méthode des moindres carrés qui est très sensible au bruit de Monte Carlo et pour lequel deux images identiques auront un PSNR de 100. Ces deux métriques ont été choisies de par leur popularité dans le milieu du rendu de Monte Carlo. On notera cependant l'existence de travaux permettant d'approcher au plus près la métrique de comparaison d'images de la perception humaine [Adhikarla et al., 2017].

Les figures 4.5 et 4.6 montrent alors les différences de SSIM et de PSNR entre des rendus quantifiés ou non pour le même nombre

Quantification	16 bits		24 bits		32 bits		Sans
mapping	Avec	Sans	Avec	Sans	Avec	Sans	Sans
staircase	0.983	0.592	0.984	0.975	0.984	0.984	0.984
glass	0.838	0.566	0.872	0.808	0.873	0.873	0.873
living room	0.941	0.721	0.942	0.934	0.942	0.941	0.942

TABLE 4.1 – **Comparaison à la valeur exacte.** SSIM entre les images rendues avec quantification avec 512 échantillons par pixel et la valeur exacte. La quantification utilisée est la quantification octaédrique et les vecteurs sont regroupés selon le regroupement PSF avec 2^{13} groupes. La dernière colonne du tableau montre le résultat de la comparaison des rendus non compressés avec la valeur exacte.

d'échantillons par pixel par rapport à un rendu non quantifié. Nous observons que réduire l'erreur de quantification moyenne et maximale permet d'obtenir des images qui s'approchent fortement des rendus non-quantifiés. Cela est d'autant plus vrai selon la métrique SSIM, ce qui s'explique par le fait qu'un faible changement d'angle dans le rayon peut radicalement changer le bruit généré par le rendu de Monte Carlo, et que cette différence n'est pas nécessairement perceptible à l'oeil humain. On observe également que même avec un nombre très faible de bits de quantification, notre méthode est capable de fournir des images se rapprochant fortement des images produites par des rendus non quantifiés, tout particulièrement pour les scènes diffuses telles que *staircase*. Pour les scènes montrant de multiples rebonds spéculaires, telles que *glass*, il est nécessaire d'utiliser plus de bits de quantification pour obtenir des résultats corrects (une SSIM proche de 1, ou un PSNR proche de 100). Nous observons que les rendus effectués avec 16 bits de quantification sans notre méthode ne permettent même plus de discerner la scène 3D, l'utilisation de notre méthode permet d'obtenir des images correctes. Si nous diminuons trop le nombre de bits de quantification, nous voyons apparaître le même type d'artefacts, comme le montre le rendu utilisant la quantification octaédrique sur 8 bits avec `UniQuant` sur la scène *Glass* dans la figure 4.7. Il est également intéressant de voir si la compression écarte l'image générée de la version non quantifiée en ayant un impact sur la convergence des rendus vers la valeur exacte.

Dans la figure 4.4, nous observons que les zones de l'image ayant les plus grandes différences sont celles présentant des matériaux très spéculaires (le chandelier de *Staircase* par exemple), mais également les zones nécessitant de multiples rebonds de lumière pour être éclairées comme l'intérieur des étagères de *Living Room*.

Dans le tableau 4.1, nous observons qu'avec 512 échantillons par pixel, c'est-à-dire un paramètre de rendu qui pourrait être utilisé en production, même si les images obtenues avec notre méthode ne sont pas les mêmes que celles obtenues avec un rendu non quantifié, elle ont des valeurs de SSIM très proches de celles obtenues avec un rendu non quantifié par rapport à la valeur exacte. Tout particulièrement, il est possible d'utiliser une quantification sur 24 bits avec notre méthode sans perte de rapidité de convergence dans les cas montrés. Toutefois, cette première partie nous a permis de comprendre qu'au delà du nombre de bits de quantification utilisés pour représenter un rayon, certaines scènes sont plus complexes à traiter que d'autres. Ainsi, alors qu'il est simple de compresser les scènes majoritairement diffuses sans perdre en qualité, lorsque cela est effectué sur une scène principalement spéculaire, l'impact est bien plus important, notamment vis à vis de la convergence par rapport à la valeur exacte. Nous avons ensuite examiné s'il est possible d'utiliser une approche adaptative aux matériaux rencontrés par un chemin lumineux.

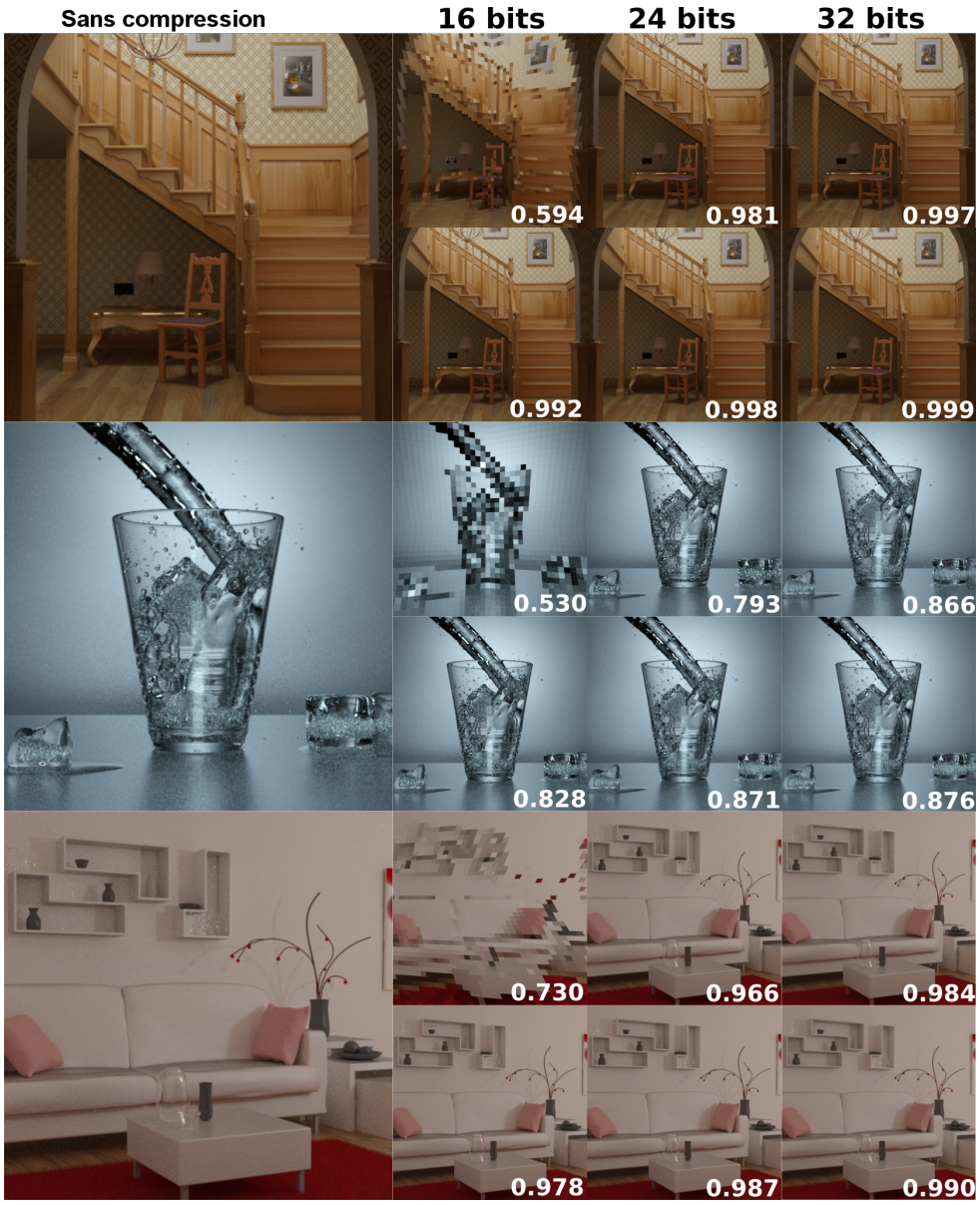


FIGURE 4.5 – **Impact de la quantification sur le rendu de Monte Carlo (SSIM)**. Toutes les images ont été rendues en utilisant 512 échantillons par pixel. Pour chaque scène présentée dans la Figure 4.1, cette figure présente sur la première ligne les rendus réalisés en utilisant une quantification octaédrique sur 16, 24 et 32 bits. La seconde ligne montre les rendus réalisés avec la même quantification, combinée à la transformation uniforme exploitant 2^{13} groupes selon la métrique CSD présentée précédemment. Pour chaque image, la valeur de SSIM (*Structural Similarity*) par rapport au rendu réalisé sans quantification est montré. Les rendus quantifiés sont obtenus en quantifiant les rayons des chemins, mais pas les rayons d'ombres.

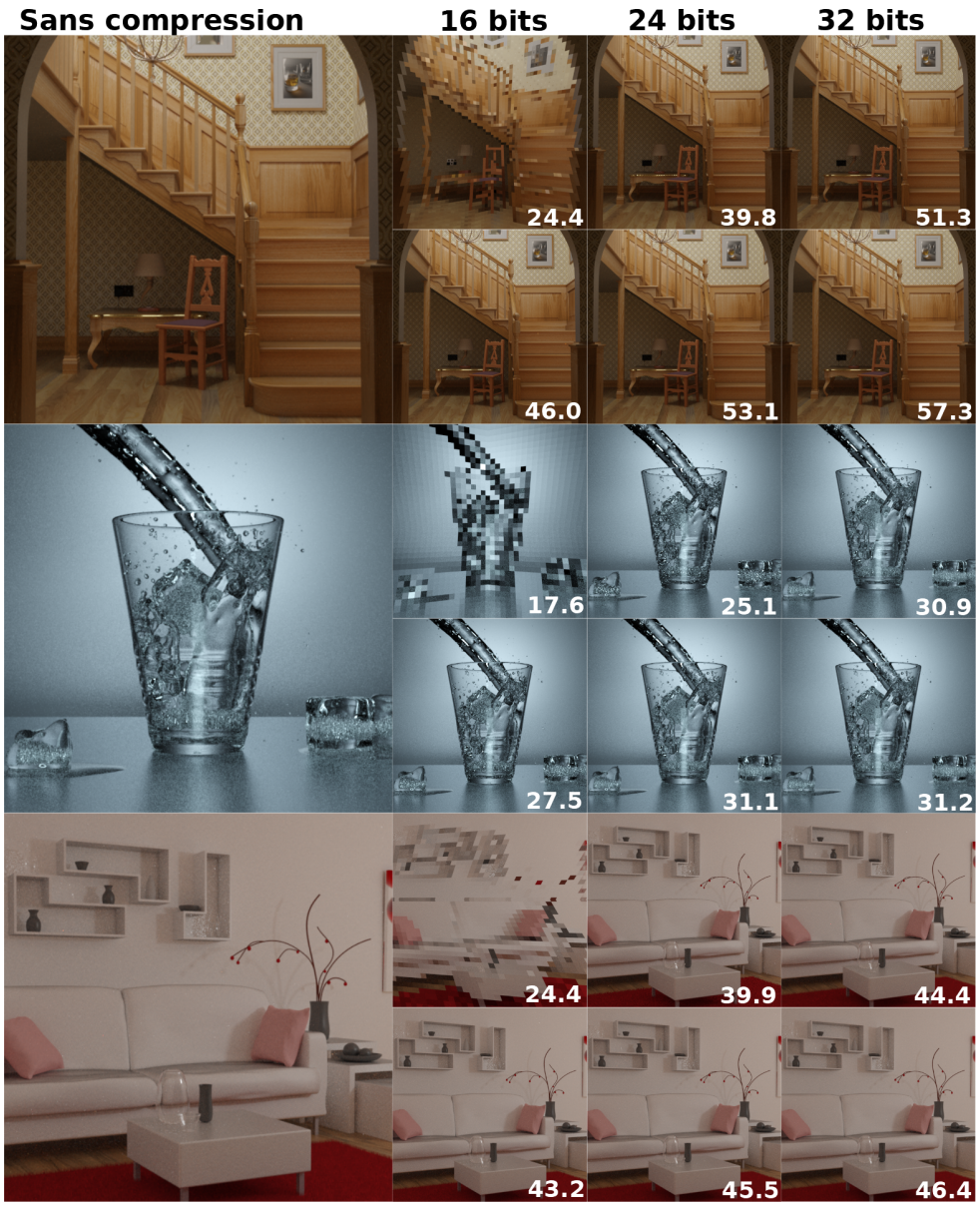


FIGURE 4.6 – **Impact de la quantification sur le rendu de Monte Carlo (PSNR)**. Cette figure montre les mêmes exemples que la figure 4.5 avec les valeurs de mesure de similarité basées sur le PSNR indiquées en blanc

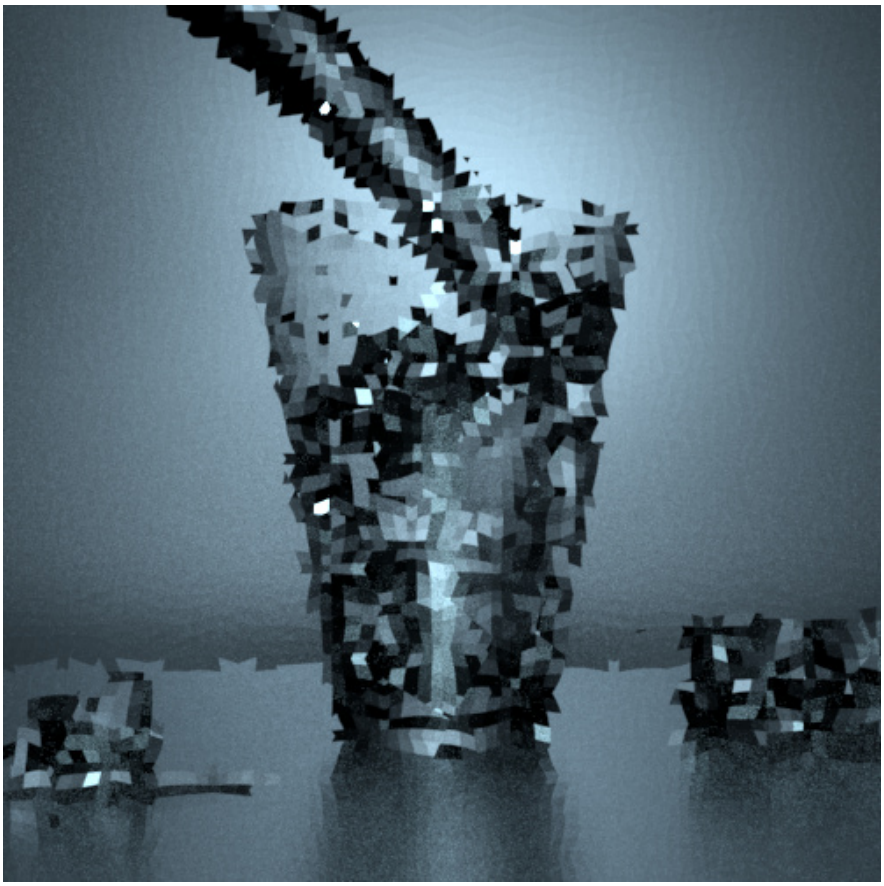


FIGURE 4.7 – Scène *Glass* rendue avec une quantification octaédrique sur 8 bits combiné avec `UniQuant`.

Scène	Nombre de rayons spéculaires	Nombre de rayons diffus	Pourcentage de rayons diffus
glass	509M	0M	0
staircase	414M	225M	35.1
living room	287M	385M	57.3

TABLE 4.2 – Nombre de rayons générés suite à un rebond avec un matériau diffus ou spéculaire lors du rendu des 3 images montrées dans la figure 4.1.

Quantification	*Living room*	*Staircase*
16-24 bits	1.52 Go	1.58 Go
16-32 bits	1.78 Go	1.96 Go
24-32 bits	2.14 Go	2.17 Go
16 bits	1.25 Go	1.19 Go
24 bits	1.88 Go	1.78 Go
32 bits	2.50 Go	2.38 Go
Sans	7.50 Go	7.14 Go

TABLE 4.3 – Taille totale des directions des rayons traités durant le rendu en fonction de la compression choisie.

4.5 Quantification adaptative des directions en fonction des matériaux

La classification des matériaux en deux catégories distinctes, diffus et spéculaires, est un problème de recherche difficile, notamment pour la détection des zones difficiles du rendu, comme les parties de l'image qui pourraient présenter des reflets spéculaires. N'étant pas le sujet de cette étude, nous nous reposons sur les taux de spécularité explicitement fournis par les modèles de matériaux numériques de la scène. Dans PBRTv3, les matériaux disposent d'indicateurs permettant de savoir s'ils sont diffus ou non. Nous nous intéressons tout d'abord au ratio rayons classifiés comme diffus après un rebond. Cela nous indique si un rendu pourrait profiter ou non d'une méthode adaptative. Nous affichons ces statistiques sur les trois scènes présentées précédemment dans le tableau 4.2. Nous observons alors que la scène *Glass* est bien uniquement composée de matériaux spéculaires. S'intéressant aux méthodes adaptatives aux matériaux rencontrés, nous l'éliminons donc, car l'algorithme se contenterait de compresser l'ensemble des rayons avec la précision maximale.

En réalisant une quantification avec `UniQuant` distincte pour les rayons diffus et spéculaires, nous obtenons les quantités de données montrées dans le tableau 4.2.

Nous mettons alors en rapport ces tailles de données avec les SSIM comparant les rendus (à 512 échantillons par pixel) avec compression

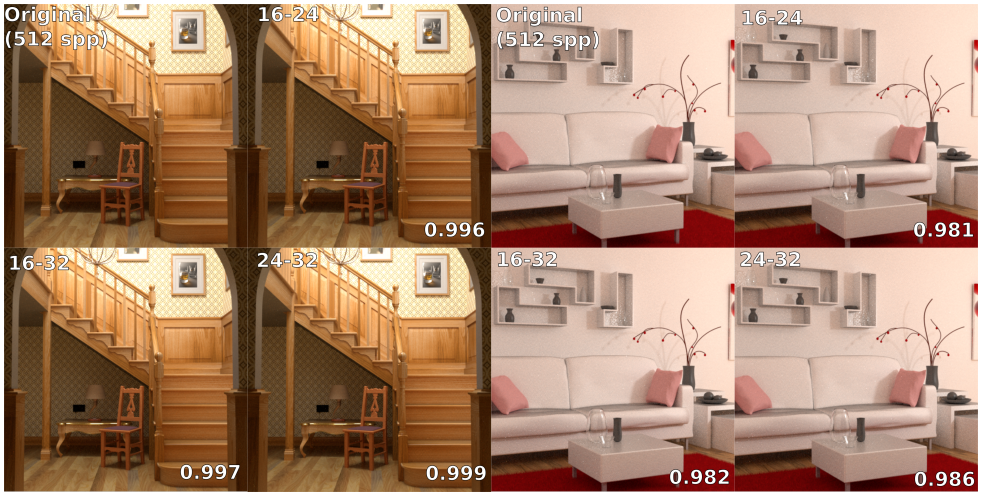


FIGURE 4.8 – SSIM entre les rendus non quantifiés et les rendus avec rendu adaptatif.

adaptative au rendu effectué sans compression. Comme montré dans la figure 4.8, nous observons alors que la méthode permet dans certains cas d'économiser de l'espace mémoire, comme avec *staircase*, où les mêmes résultats sont obtenus avec une quantification avec mapping sur 32 bits ou un quantification adaptative de 24-32 bits, permettant une économie de 9% de mémoire. Dans le cas de *Living Room*, comme nous pouvons l'observer sur la figure 4.4, l'erreur est principalement présente dans des zones où le matériau est diffus mais où les chemins lumineux doivent faire de nombreux rebonds avant d'atteindre une lumière. De tels cas ne peuvent pas être détectés à priori et restent problématique pour cette méthode alternative. Dans ces cas, la méthode fournit tout de même des taux de compression alternatifs aux taux de compression imposés par les précisions de quantification fixées par le mot machine.

4.6 Quantification des origines

Nous avons proposé une méthode de compression de directions collaboratives qui peut être exploitée dans le cadre du rendu de Monte Carlo distribué, notamment lorsque l'on exploite un système de division de scènes, et de portails pour téléporter les chemins de lumières entre les différents noeuds d'un cluster de calcul. Un rayon dispose cependant d'au moins deux composantes, la direction et l'origine, toutes deux de même taille. Nous nous intéressons donc à la

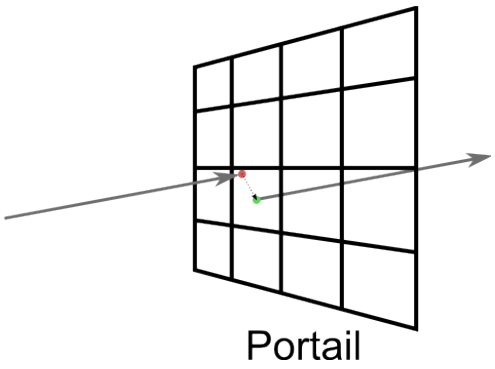


FIGURE 4.9 – Quand un rayon rencontre un portail (point rouge), l'origine de celui-ci est décalée au centre du rectangle (point vert) issu de la discrétisation 2D de la surface du portail. Celui-ci peut être encodé efficacement avec les coordonnées bilinéaires.

compression de ces dernières pour proposer un pipeline complet de compression d'ensemble rayons sur des portails de lumière.

4.6.1 Méthode

Comme avec les directions, nous étudions les méthodes de quantification pour les origines. Les scènes utilisées sont normalisées. Elles sont alors divisées en sous-parties, nommées cellules, selon une grille régulière 3D. Nous faisons varier la précision de la division de la scène avec des grilles contenant respectivement 5^3 et 10^3 cellules. Cela permet d'observer l'impact du nombre de cellules sur la quantification.

Sur la surface des cellules, se situent les portails, eux-mêmes divisés selon une grille régulière 2D. Pour quantifier une origine, on la représente par le centre du rectangle issu de la discrétisation 2D du portail que le rayon a rencontré lors de son intersection avec la sous-scène, comme illustré dans la figure 4.9. En effet, lorsqu'un rayon rencontre un portail, son origine peut être déplacée à son niveau, car cela indique qu'il n'y a eu aucune intersection précédemment. La géométrie du portail étant partagée entre les voisins, le noeud de calcul suivant disposera alors de la géométrie nécessaire à la reconstruction de l'origine compressée. Seul l'identifiant de la cellule doit être transmis. Il est également possible de compresser sans perte en exprimant l'origine selon ses coordonnées bilinéaires sur le quadrilatère du portail, réduisant à deux dimensions au lieu de trois la taille de l'origine. Dans notre cas, nous quantifions les coordonnées bilinéaires, ce qui fournit bien une quantification en grille sur la

surface du portail comme illustré dans la Figure 4.9.

4.6.2 Résultats

En quantifiant ainsi les origines, nous obtenons les résultats présentés dans la figure 4.10. Nous observons alors que la taille des portails a bien un impact sur la précision de la quantification des origines. Plus ils sont nombreux, plus ils sont petits, et plus la quantification est précise. Cela est cependant à mettre en perspective avec le fait qu'un plus grand nombre de portails implique un plus grand nombre d'échanges sur le réseau, et ceux-ci doivent donc être limités au maximum. Nous observons ensuite que des artefacts sont visibles pour des quantifications de 32 bits ou moins sur les scènes *glass* et *living room*. Nous pouvons observer avec les SSIM qu'une fois encore, les scènes les plus spéculaires sont les plus difficiles à rendre. Les quantifications sur 48 bits permettent sur les scènes les plus simples telles que *staircase* d'obtenir des résultats visuellement identiques à des images ayant été générées sans la méthode, et dans le cas de scènes légèrement plus complexes telles que *living room*, d'obtenir des scènes visuellement difficiles à différencier. L'oeil humain étant moins habitué à voir les différences dans les reflets spéculaires, la qualité d'images obtenues dans le cas de la scène *glass* pourra convenir pour certains projets, mais le cas spéculaire peut nécessiter l'utilisation de la compression sans perte, en envoyant directement les coordonnées barycentriques sans quantification. La quantification sur 48 bits permet d'économiser la moitié de la place par rapport à une origine encodée sur trois nombres à virgule flottante, et la compression sans perte permet d'économiser un tiers de l'espace mémoire. Nous observons enfin la localisation des erreurs sur l'image dans la figure 4.11. Comme dans le cas de la compression des direction, les zones de plus grandes erreurs sont celles qui sont les plus spéculaires (le verre), et celles nécessitant de nombreux rebonds pour atteindre une source de lumière ce qui accumule l'erreur de compression (en dessous des étagères par exemple).

4.7 Compression du rayon complet

Nous avons enfin combiné la compression de directions avec `UniQuant`, avec la compression d'origines avec la quantification de portails pour voir l'impact sur le rendu. Nous utilisons les meilleurs paramètres de qualité d'image parmi ceux étudiés précédemment

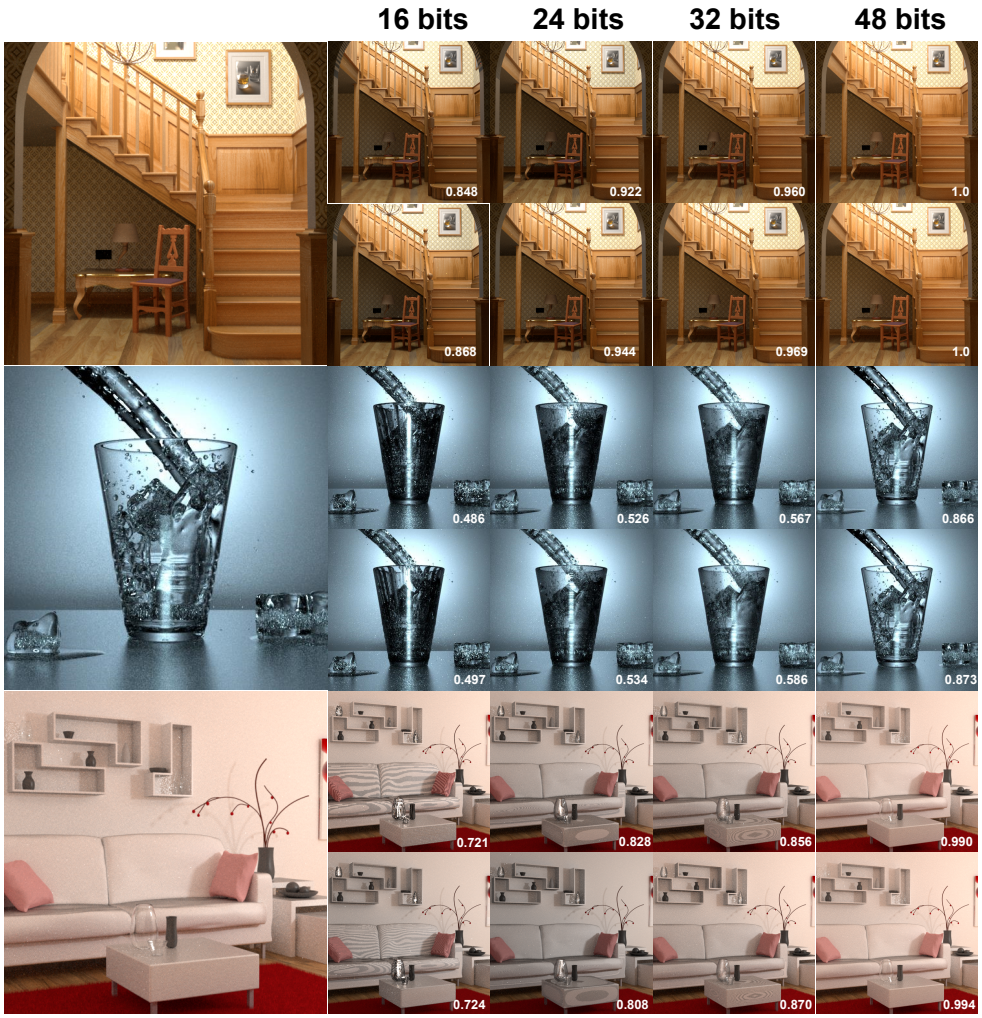


FIGURE 4.10 – SSIM entre les rendus avec origine quantifiés sur les portails et les rendus non quantifiés avec autant d'échantillons par pixel. Pour chaque scène, la première ligne est rendue en utilisant une division de scène en 125 cellules (première ligne) ou 1000 cellules (seconde ligne). Le nombre de bits pour chaque colonne correspond au nombre total de bits (soit pour la colonne 16 bits, 8 bits par dimension)

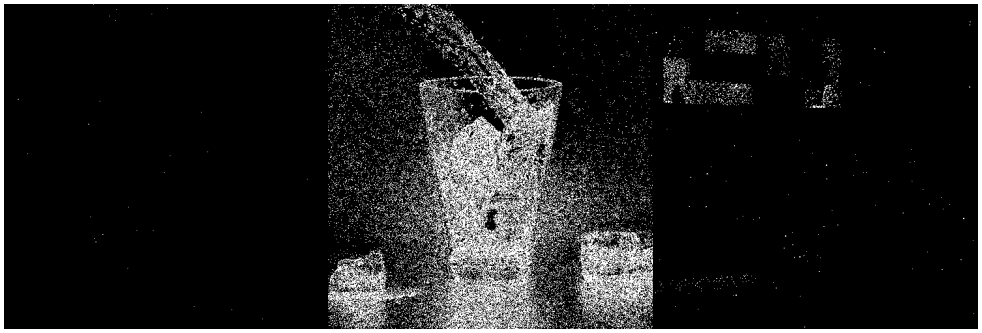


FIGURE 4.11 – Différence entre le rendu avec quantification des origines sur 48 bits et sans quantification pour les scènes (de gauche à droite) *staircase*, *glass*, et *living room*. En blanc apparaissent les pixels avec une différence entre deux couleurs RGB supérieures à 2%.

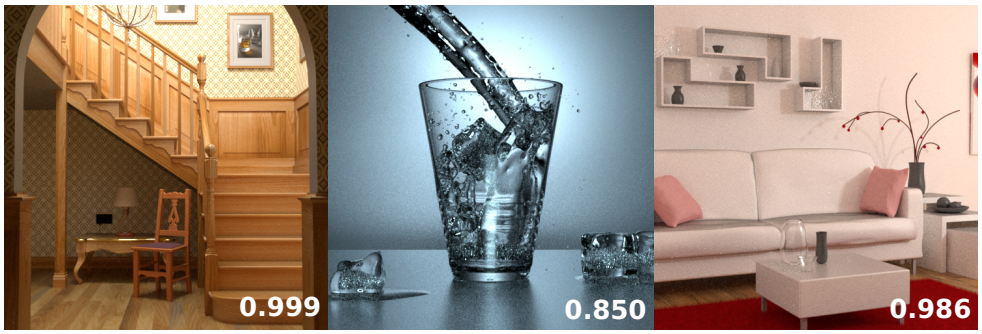


FIGURE 4.12 – SSIM des rendus utilisant le pipeline complet de compression de rayons (quantification octaédrique sur 32 bits pour les directions et quantification sur 48 bits pour les origines) par rapport aux rendus ne les utilisant pas avec le même nombre d'échantillons par pixel.

pour chaque technique, c'est à dire 32 bits pour les directions, et 48 bits pour les origines. Nous réduisons ainsi de 58% la taille des rayons à transmettre sur le réseau et nous obtenons alors les rendus présentés en figure 4.12.

Nous observons alors que les différences sont imperceptibles sur les images telles que *staircase* et *living room*. Cependant, une légère différence existe dans le cas de *glass*. Nous en déduisons que la technique pourra être utilisée sur des scènes classiques, mais que les scènes plus complexes, et hautement spéculaires devront être encore traités sur des fermes de rendus, ou en rendu distribué, sans avoir recours à des méthodes de compression avec perte. Nous observons

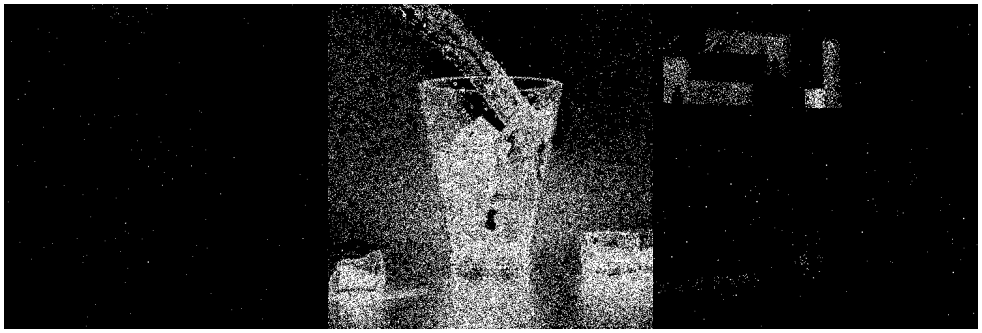


FIGURE 4.13 – Différence entre les rendus de la Figure 4.12 et les rendus réalisés sans compression

enfin la localisation sur les images des différences par rapport à des rendus réalisés sans compression dans la figure 4.13. La localisation des erreurs reste la même que sur les erreurs observées avec la compression de direction seule, et la compression des origines seules. Les zones difficiles sont les zones hautement spéculaires, ainsi que les zones très sombres nécessitant de nombreux rebonds pour atteindre une source de lumière.

4.8 Perspectives

Nous avons vu qu'il est possible d'utiliser la méthode `UniQuant`, proposée au précédent chapitre dans le cas du rendu de Monte Carlo distribué. Nous profitons alors des performances suffisamment intéressantes pour permettre un envoi sur un réseau non local pour accélérer les transferts réseau comme montré dans le chapitre 3. En fonction de la scène cible, l'utilisation d'une stratégie adaptative permet d'améliorer les résultats en adaptant la compression des rayons aux matériaux parcourus. Il permet également d'avoir une plus grande variété de choix de taux de compression. Cette adaptation est réalisée grâce à la description du taux de spécularité explicitement fournis par les modèles de matériaux numériques de la scène, et il serait intéressant dans le futur d'étudier une méthode automatique pour effectuer la séparation entre les rayons spéculaires et diffus pour le cas de la compression. La méthode peut être combinée à des méthodes de compression des origines pour obtenir une compression collaborative d'ensembles de rayons. Celle-ci est réalisée avec une quantification des origines sur les portails, interfaces de transfert réseau entre les noeuds du cluster. Il serait intéressant d'étudier des

approches plus sophistiquées pour la compression des origines, telle que l'utilisation d'une structure hiérarchique pour leur encodage. Il faudra alors s'assurer que la taille de la structure hiérarchique reste inférieure au gain qu'elle procure. Il faudrait également étudier l'impact de telles méthodes de compression sur les milieux participants, et, le cas échéant, adapter la méthode à ce type de scènes. Les scènes hautement spéculaires restent problématiques et nécessitent actuellement de baisser de manière importante les taux de compression pour obtenir des résultats acceptables. Les algorithmes d'apprentissage artificiel restent actuellement rares en rendu du fait de la haute dimensionalité des données traitées. Il serait alors intéressant d'étudier les auto-encodeurs pour formuler le problème de compression comme un réseau de neurones, en exploitant la discrétisation sous forme de portail comme réduction du nombre de dimensions. Dans le chapitre suivant, nous montrerons que l'outil méthodologique introduit peut être utilisé plus globalement dans différents domaines en l'utilisant dans la modélisation numérique 3D, et notamment dans la modélisation de données du domaine médical. Cette application se fait au travers de la compression de tractogrammes du cerveau, pour lesquels de grands ensembles de vecteurs unitaires jouent un rôle clé.

CHAPITRE 5

Compression de tractogrammes du cerveau

FIGURE 5.1 – Un tractogramme contient des millions de fibres décrivant les connections neuronales du cerveau. Elles sont construites à l'aide de données extraites à partir d'une IRM de diffusion du cerveau.

5.1 Objectif

Nous avons précédemment montré l'intérêt de la compression d'ensembles de vecteurs unitaires dans le cas de la synthèse d'images, en rendu de Monte Carlo. Cet outil méthodologique peut cependant s'appliquer à d'autres types de modèles partageant les mêmes contraintes mathématiques. Dans ce chapitre nous nous intéressons donc à la compression de données partageant des contraintes communes avec les rayons ou chemins lumineux. Cette ouverture permet d'illustrer la généralité des méthodes de compression développées durant cette thèse. Nous présentons `qfib`, une adaptation de `UniQuant`, la méthode de compression d'ensemble de vecteurs unitaires présentée dans le chapitre 3, appliquée à des données médicales : les tractogrammes. Les tractogrammes (Figure 5.1) sont des données couramment utilisées en neurosciences et neurochirurgie pour décrire les différentes connexions neuronales du cerveau humain. Leur utilisation est cependant actuellement fortement limitée par la taille de ce type de données. Ils sont composés d'un ensemble de lignes polygonales 3D permettant d'obtenir une représentation de l'organisation anatomique de la matière blanche du cerveau. Ils permettent aux neurochirurgiens de planifier des opérations en prévoyant l'influence des différentes zones du cerveau entre elles. Ils permettent également aux chercheurs en neurosciences de mieux comprendre le fonctionnement du cerveau. Ils sont construits à partir de données issues d'une IRM de diffusion, permettant de visualiser le champ de propagation des molécules d'eau. Le problème de ce type de données est leur taille. Un tractogramme de quelques millions de fibres atteint rapidement des dizaines de Go, rendant leur stockage, leur échange, leur traitement, ou même leur visualisation très difficile. Il existe actuellement très peu de méthodes de compression dans l'état de l'art et leurs performances n'ont pas permis de les rendre populaires dans la communauté. `qfib` exploite la puissance de `UniQuant` pour permettre de réaliser une compression à la volée de tractogrammes, en assurant des hauts taux de compression, un possible traitement hors coeur des données, des faibles temps de calculs et une faible erreur en dessous de la précision de l'IRM. Il permet de réduire de 80 à 90% la taille des données en quelques secondes pour des jeux de données standards.

Ce travail a été publié dans le journal NeuroInformatics et a été présenté sous forme de poster et de démonstration logicielle dans la conférence OHMB 2019 (Organization for Human Brain Mapping :

– **QFib : Fast and Efficient Brain Tractogram Compression**, *Corentin Mercier*, Sylvain Rousseau*, Pietro Gori, Isabelle Bloch, Tamy Boubekeur*, Neuroinformatics 2020 [Mercier et al., 2020]
– **QFib : Fast and efficient Fiber Tracking Dataset Compression (poster)**, *Sylvain Rousseau*, Corentin Mercier*, Pietro Gori, Isabelle Bloch, Tamy Boubekeur*, OHBM 2019. [Rousseau et al., 2019].
Il a été réalisé en collaboration avec Corentin Mercier.

5.2 Données

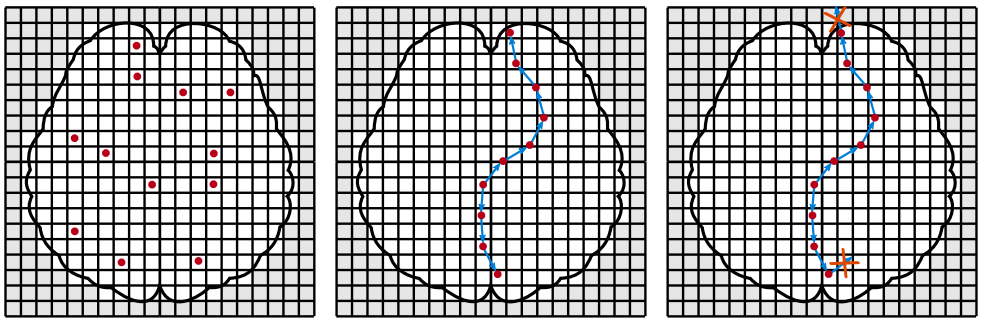


FIGURE 5.2 – Construction d'un tractogramme. (a) Premièrement, on échantillonne des graines aléatoirement au centre des cellules. (b) Ensuite, pour chaque graine, on avance à **pas constant** selon le champ de déplacement des molécules d'eau. (c) Enfin, la fibre est terminée si le point suivant mène en dehors de la matière blanche, ou que l'**angle formé entre deux segments consécutifs est trop important**.

Dans ce chapitre, nous nous intéressons tout particulièrement à la compression de tractogrammes du cerveau humain. Ils sont issus d'IRM de diffusions et offrent la seule méthode permettant d'obtenir de manière non invasive l'architecture de la matière blanche du cerveau humain. Ces données sont utilisées par les cliniciens, les neurochirurgiens et les chercheurs pour comprendre les connexions dans le cerveau. Elles peuvent ainsi aider les chercheurs à mieux comprendre le fonctionnement du cerveau humain, mais également aider en pré-opératoire à déterminer l'impact de l'ablation de certaines parties. Les tractogrammes sont composés de lignes de flux tridimensionelles, couramment appelées fibres, représentées par des polylignes tridimensionelles contenant chacune quelques centaines ou milliers de points 3D ordonnés. Les algorithmes de construction de tractographie peuvent obtenir des millions de fibres ([Tournier et al., 2011]). Celles-ci forment des fichiers de plusieurs dizaines de gigaoctets ([Rheault et al., 2017]). Pour les obtenir, on les construit à partir des données de l'IRM de diffusion, correspondant à une grille tridimensionnelle régulière, contenant les directions de diffusion des molécules d'eau dans le cerveau. A partir de celle-ci, on commence tout d'abord par échantillonner des graines aléatoires dans la grille, au centre des cellules (Figure 5.2.a). Ensuite, on propage chaque graine, en suivant le gradient dans les deux sens (Figure 5.2.b). Cela peut

s'effectuer de différentes manières, à l'aide de plusieurs heuristiques. Certaines sont déterministes et fournissent des fibres régulières, alors que d'autres sont probabilistes et peuvent fournir des fibres "bruitées". On étudie ici des tractogrammes générés selon une heuristique déterministe, et d'autres jeux de données ayant été construits selon une heuristique probabiliste. Plusieurs conditions d'arrêts existent (Figure 5.2.c). La propagation ne continue pas si l'angle formé en ajoutant un nouveau point forme un angle trop important par rapport au précédent segment de fibre, ou si le point mène en dehors de la matière blanche. Une fois ceci fait, les points sont réordonnés pour chaque fibre.

La taille de ce type de données, de plusieurs dizaines de Go, les rend difficiles à stocker, visualiser, analyser, traiter, ou même échanger. Par exemple, un fichier contenant 1 million de fibres composées de segments de polylignes faisant 0.1 mm peut peser jusqu'à 8.7 Go.

Il est à noter que les praticiens travaillent avec des ordinateurs de faible puissance ne disposant pas de beaucoup de mémoire vive ou même de stockage. C'est un frein au développement de l'utilisation de ce type de données, montrant l'importance du développement d'une meilleure représentation minimisant la taille de celles-ci avec une complexité algorithmique minime.

Nous introduisons ici un nouvel algorithme de compression spécialement conçu pour les tractogrammes, prenant en compte la manière dont ces données sont construites avec les méthodes les plus couramment utilisées. Cette technique se base sur `UniQuant`, la méthode de compression décrite dans le chapitre 3

5.3 Introduction

5.3.1 État de l'art en compression de tractogrammes

Les méthodes existantes permettant de réduire la taille des tractogrammes du cerveau peuvent être divisées en deux catégories ; celles compressant l'ensemble du tractogramme et celles s'intéressant à la représentation individuelle des fibres.

Compression du tractogramme complet

Ce type d'approche consiste à réduire le nombre de fibres du tractogramme. Une manière de le réaliser est de choisir aléatoirement

un sous-ensemble de fibres de l'ensemble de données originales [Gori et al., 2016]. Avec ce type de méthode, il est difficile de contrôler l'information perdue. Pour s'assurer la conservation des données importantes, il n'est pas possible de supprimer un trop grand nombre de fibres et de ce fait, les taux de compressions ne peuvent pas être très importants.

D'autres méthodes utilisent le regroupement de fibres similaires ([Alexandroni et al., 2017, Garyfallidis et al., 2012, Guevara et al., 2011, Liu et al., 2012, Maddah et al., 2007, Wassermann et al., 2010, Demir and ¸Cetingül, 2015]) pour supprimer les redondances statistiques dans les données. Dans ce cas, les groupes peuvent être représentés par une fibre unique de l'ensemble de données originales que l'on nomme prototype ([Guevara et al., 2011]), ou à l'aide d'une fibre créée pour représenter au mieux le groupe ([Garyfallidis et al., 2012]). Ces méthodes nécessitent de développer des heuristiques de regroupement, et peuvent être particulièrement adaptées à certains cas d'application. Des représentations géométriques des fibres groupées peuvent également être utilisées [Maddah et al., 2007]. Il existe par exemple les cylindres généralisés ([Mercier et al., 2018, Petrovic et al., 2007]), construit de manière à ce que l'étendue spatiale des fibres regroupées soit conservée. Les données sont alors plus simples à comprendre ou à visualiser. Selon certains critères (forme, connections, etc.) elles peuvent également être plus simple à analyser. Cependant, des données sont tout de même perdues et dans certains cas d'applications, il est préférable de conserver l'ensemble des fibres originales.

Compression des fibres

Dans le cas général où l'on cherche à garder autant d'informations que possible depuis l'ensemble de données originales, la compression des fibres individuelles, ou leur représentation est une bonne approche. Dans certains cas, ces méthodes peuvent également être combinées avec les méthodes de compression de tractogrammes.

Il est possible d'appliquer un algorithme général de compression avec ou sans perte au tractogramme. Cependant, comme montré dans le Chapitre 3 sur `UniQuant`, cette approche n'est pas la plus optimale, car ces algorithmes identifient et suppriment les redondances statistiques sans avoir de connaissances sur les données utilisées. Elles ne peuvent donc pas exploiter les contraintes mathématiques permettant de changer l'espaces de représentation. Dans le cas de tractogrammes du cerveau, les données sont exprimées par

des ensembles de points 3D en nombres à virgule flottante à précision simple. Une approche classique pour compresser chaque fibre individuellement consiste à réduire le nombre de points la composant. Cette méthode se nomme une linéarisation. En retirant les points colinéaires, on peut réaliser une compression sans perte, mais la méthode prend son intérêt lorsque l'on accepte une marge d'erreur [Presseau et al., 2015]. Cette dernière méthode fournit un nouveau tractogramme en sortie qui peut ensuite être compressé avec les méthodes de compression de tractogrammes. L'inconvénient de ce genre de méthodes est la perte de points. Certains traitements appliqués ensuite sont sensibles à ce type de perte de données [Soares et al., 2013]. Une autre approche consiste à construire un dictionnaire dans un espace de représentation dédié [Presseau et al., 2015, Kumar and Desrosiers, 2016, Moreno et al., 2017]). `zfib`, proposé par l'équipe de Presseau [Presseau et al., 2015] utilise un algorithme en trois étapes réalisant une (i) linéarisation, une (ii) quantification et enfin un (iii) encodage des points à l'aide d'un dictionnaire. Cette méthode est une approche similaire à la compression `zip` , justifiant le nom de l'algorithme. Cette méthode permet d'obtenir des bons taux de compression (dans les 90%) mais au détriment d'un grand temps de calculs (plus de 10 minutes pour un tractogramme d'un million de fibres). Du fait de l'étape de linéarisation, certains points sont perdus, perturbant l'utilisation des algorithmes basés sur les points que l'on pourrait souhaiter utiliser ensuite [Soares et al., 2013]. Les auteurs de cette méthode ont également essayé d'appliquer d'autres types de transformations, comme celle en ondelettes, sans améliorer le ratio de compression.

D'autres méthodes de compression dédiées à la tractographie utilisent des représentations parcimonieuses des lignes de flux [Chung et al., 2009, Kumar and Desrosiers, 2016, Moreno et al., 2017]. Dans [Chung et al., 2009], les fibres sont représentées à l'aide de fonctions cosinus paramétrées avec 60 paramètres, réduisant le coût mémoire. L'erreur moyenne introduite est de $0.26\ mm$, ce qui peut être trop important pour certains cas d'application.

Dans [Alexandroni et al., 2017] l'idée est d'utiliser un dictionnaire combiné avec un échantillonage de la densité de fibres. Cependant, cette technique supprime toutes les hautes fréquences en ne gardant que quelques coefficients non nuls par fibre. Les changements brusques dans le chemin de la fibre sont alors adoucis. L'erreur moyenne type avec ce genre de méthode est de l'ordre de $2\ mm$ sur des données ayant une résolution de $1.25\ mm^3$.

En considérant que les méthodes actuelles de tractographies sont appliquées sur des images de diffusion ayant une résolution inférieure à $1\ mm^3$ (entre 1.25 et $2\ mm^3$ en pratique), l'erreur de compression d'une fibre ne doit pas dépasser cette valeur de taille de voxel. Idéalement, il faut atteindre une erreur ayant un ordre de grandeur plus faible. De plus, ces méthodes basées dictionnaire ne permettent pas la compression et la décompression de parties ou sous-ensemble de fibres, réduisant grandement leur intérêt dans le cas de la visualisation ou de traitement des données.

Ces méthodes démontrent donc l'important besoin d'une nouvelle approche pour effectuer une compression rapide et efficace pour traiter les données de tractographie du cerveau. Pour combiner la rapidité de la compression avec un haut taux de compression, la possibilité de compresser individuellement chaque fibre, et de passer à l'échelle, nous nous intéressons directement à la représentation des données individuelles, les fibres.

Contraintes sur les données

Pour réaliser la compression de données, il est nécessaire de trouver un plus petit espace de représentation que celui utilisé originalement. Dans le cas de tractogrammes, il est possible d'exploiter la manière dont les données sont construites pour restreindre cet espace. Nous utilisons tout particulièrement deux contraintes : le pas constant δ entre les points successifs d'une fibre, et l'angle maximal ψ entre les segments consécutifs de n'importe quelle fibre du jeu de données. Les algorithmes de construction de tractogrammes les plus couramment utilisés fixent une **distance constante** entre les points d'une même fibre. C'est une propriété qui est par ailleurs utilisée dans certains post-traitement [Soares et al., 2013]. De plus, les fibres sont également terminées lorsque l'angle entre deux segments consécutifs est trop important (supérieur à un **angle maximum** commun à l'ensemble du tractogramme). L'utilisation de ces deux contraintes permet de considérer une fibre comme une succession de vecteurs unitaires dont l'angle entre des segments consécutifs est restreint. Nous proposons dans ce chapitre un nouveau modèle de représentation permettant de prendre en compte ces contraintes pour réduire l'espace de représentation. Nous proposons ensuite `qfib`, un nouvel algorithme de compression et de décompression basé sur cette représentation. Cette méthode possède les propriétés suivantes :

— haut taux de compression (entre 80 et 90%).
— compression et décompression rapide (quelques secondes pour les jeux de données types).
— faibles erreurs de compression.
— possibilité de compresser et décompresser des fibres individuelles.
— conservation du nombre de points de chaque fibre.
— possibilité d'accéder à une fibre aléatoirement dans le jeu de données compressés.

Une implémentation open-source C++ est disponible sur GitHub : https://github.com/syrousseau/qfib.

5.4 Méthode

Nous définissons δ le pas constant entre les points d'une fibre et ψ l'angle maximum entre deux segments consécutifs du tractogramme dans la suite de ce chapitre.

5.4.1 Représentation d'une fibre du cerveau

Une fibre f_a contenant N_a points est décrite par la polyligne 3D $f_a = \{p_1 \dots p_{N_a}\}$ par un ensemble de points ordonnés. Chaque fibre a son propre nombre de points N_a et une distance entre chaque point δ_a. Cette distance peut être différente pour chaque fibre. Le i^{ieme} point d'une fibre peut se calculer de la manière suivante : $p_i = p_{i-1} + \overrightarrow{p_{i-1}p_i}$ pour $i \in [2, N_a]$. La distance entre les points successifs d'une fibre étant constante, le point p_i peut alors être réécrit de la manière suivante :

$$p_i = p_{i-1} + \delta_a \, \hat{\overrightarrow{p_{i-1}p_i}} \tag{5.1}$$

avec $\hat{\overrightarrow{x}}$ le vecteur normalisé $\overrightarrow{x}/|\overrightarrow{x}|$. Cela implique que chaque polyligne 3D du jeu de données peut être représentée avec son premier point et un ensemble de vecteurs unitaires. La contrainte est d'avoir une distance constante entre les points d'une même fibre, ce qui est assuré par construction par la majorité des méthodes de construction de tractogrammes couramment utilisées.

5.4.2 Quantification de vecteur unitaires

La représentation avec un ensemble de vecteurs unitaires diminue déjà l'espace requis pour représenter un tractogramme, car il permet une représentation selon deux dimensions (avec les coordonnées

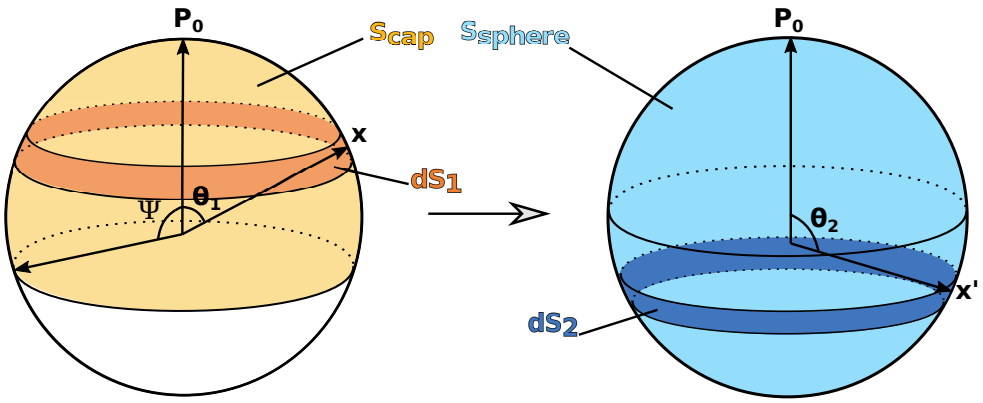


FIGURE 5.3 – `UniQuant` permet de transformer un point x défini sur une calotte sphérique centrée en P_0 d'angle maximal ψ représentée en jaune sur la surface de la sphère unitaire vers un autre point x', défini sur la surface de la sphère unitaire complète, colorée en bleu clair.

sphériques) au lieu de trois. Il permet une réduction de l'espace requis d'un peu moins d'un tiers de la taille originale.

Pour réduire encore plus cette taille, en s'assurant de ne pas perdre d'informations importantes, il est possible d'exploiter les connaissances sur l'acquisition des données. Celles-ci sont extraites à partir d'IRM de diffusion dont la précision est actuellement comprise entre 1.25 mm et 2 mm. Une compression avec perte dont la distance de déplacement des points sera d'une distance inférieure à l'ordre de grandeur de 1 mm n'aura donc pas d'impact sur les chiffres significatifs de la mesure physique. Nous utilisons donc la compression de vecteurs unitaires sur l'ensemble de vecteurs normalisés représentant les points de chaque fibre. Comme indiqué dans le chapitre 3 sur la compression de vecteurs unitaires, le choix de la méthode de quantification de vecteurs unitaires dépend du type d'application. Alors que pour la précision maximale, nous choisirons la quantification selon l'ensemble de points sphériques de Fibonacci [Keinert et al., 2015], pour privilégier la rapidité, nous choisirons la quantification octaédrique [Meyer et al., 2010].

5.4.3 Mapping

Nous avons montré dans la Section 5.4.2 que des vecteurs unitaires quantifiés peuvent être utilisés pour représenter les fibres. Cette représentation peut être considérée comme la représentation

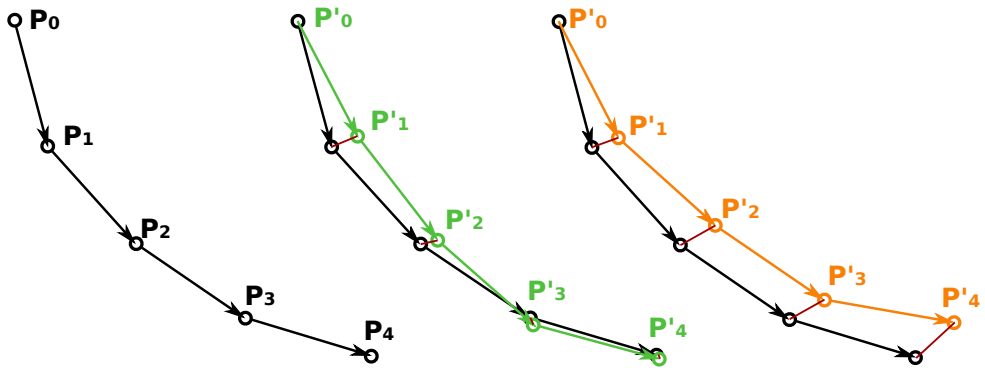


FIGURE 5.4 – Quantification avec (en vert) et sans (en orange) réduction de la propagation de l'erreur. L'erreur calculée est indiquée en rouge.

différentielle de la fibre. Avec cette représentation, la décompression est réalisée avec une intégration, en utilisant le premier point et la distance δ comme termes constants. Cela permet d'obtenir des bons taux de compression, mais il est possible d'améliorer cela en utilisant la contrainte sur l'angle maximal ψ entre deux segments consécutifs d'une même fibre. Nous utilisons à notre avantage cette contrainte en représentant chaque vecteur unitaire en fonction du précédent. Nous obtenons alors ce qui peut être vu comme la dérivée seconde de la fibre. La position relative d'un vecteur unitaire par rapport à un autre est limité à une petite calotte sphérique paramétrée par l'angle ψ (en jaune dans la figure 5.3). Tel quel, l'ensemble de points de quantification ne serait que partiellement utilisé, ce qui ne serait pas optimal. Nous nous rendons alors compte que nous nous ramenons au même problème que celui ayant été résolu par `UniQuant`. Nous utilisons alors le mapping précédemment défini dans le Chapitre 3, permettant de retransformer les calottes sphériques vers la surface de la sphère unitaire complète pour améliorer efficacement la précision de la quantification comme illustré dans la figure 5.3.

$$x' = cP_0 + \sqrt{1-c^2} \;\; avec \;\; c = 1 - \frac{1 - cos(\theta_1)}{k} \tag{5.2}$$

avec $k = \frac{1-cos(\psi)}{2} + \varepsilon$

5.4.4 Réduction de la propagation de l'erreur

A chaque fois qu'un vecteur unitaire est compressé, une petite erreur est introduite du fait du processus de quantification. Cela

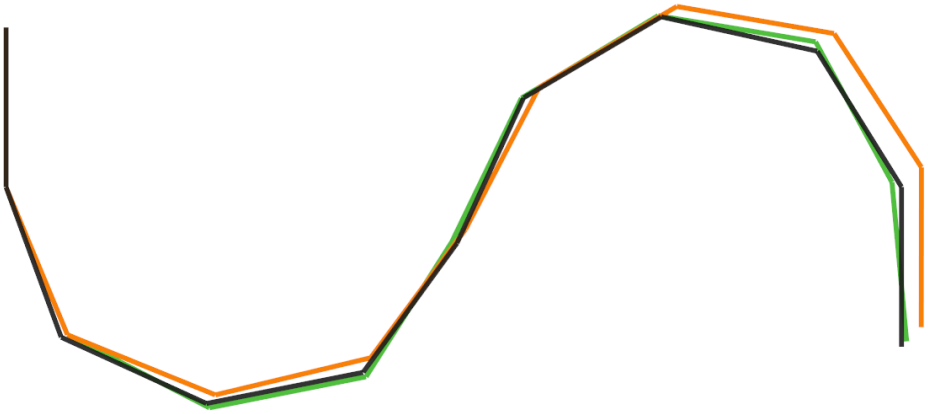


FIGURE 5.5 – Exemple de l'application de la stratégie de réduction de la propagation d'erreur en vert, sur la courbe originale en noire comparativement à une courbe ayant été compressée sans cette réduction en orange.

signifie que partant du point p_{i-1}, la position du point suivant décompressé sera p'_i et non p_i. En compressant les vecteurs unitaires les uns par rapport aux autres, en quantifiant $\overrightarrow{\hat{p_{i-1}p_i}}$, l'erreur va alors s'accumuler en permanence, et les derniers points de chaque fibre peuvent alors avoir une erreur cumulée très importante comme montré avec la courbe orange dans la figure 5.4. Pour l'éviter, il est possible de réduire la propagation de l'erreur en substituant le point p_{i-1} avec le point p'_{i-1}. La compression d'une fibre se fait point par point. Lors de la compression du point p_{i-1} nous aurons obtenu le point p'_{i-1} que nous pouvons décompresser (dé-quantifier puis dé-transformer) pour en obtenir le réel point p'_{i-1}. Au lieu de compresser le vecteur unitaire $\overrightarrow{\hat{p_{i-1}p_i}}$, nous compressons alors le vecteur unitaire $\overrightarrow{\hat{p'_{i-1}p_i}}$. Cette petite modification permet de compenser l'accumulation de l'erreur. Cette version est montrée à l'aide de la courbe verte sur la figure 5.4. Dans la figure 5.5, nous montrons une fibre d'exemple ayant été compressée et décompressée avec `qfib` avec (en vert) et sans (en orange) la méthode de réduction d'erreur de propagation. La fibre compressée est plus proche de la fibre originale en noire grâce à l'utilisation de cette stratégie de réduction de l'erreur.

5.4.5 Algorithme de compression et schéma d'encodage

Pour compresser la fibre $f_a = \{p_1, ..., p_{N_a}\}$ en une fibre f' pour laquelle les coordonnées décompressées sont $\{p'_1, ..., p'_{N_a}\}$, en combinant les étapes présentées précédemment, nous obtenons l'algorithme suivant :

— nous stockons les deux premiers points tels quels, avec leurs coordonnées cartésiennes ;
— pour les autres points, nous utilisons la transformation `UniQuant` sur les vecteurs unitaires $\overrightarrow{p'_{i-1}p_i}^{\wedge}$ avec pour axe de la calotte sphérique $\overrightarrow{p'_{i-2}p'_{i-1}}^{\wedge}$, et pour ratio, l'angle maximal ψ ;
— nous quantifions les vecteurs transformés, en utilisant par exemple la quantification octaédrique ([Meyer et al., 2010]) ou l'ensemble de points sphériques de Fibonacci ([Keinert et al., 2015]) en fonction du cas d'application visé.

Cet algorithme est appliqué indépendamment sur chaque fibre du jeu de données. Comme le premier vecteur unitaire est recalculé pour chaque fibre, cela permet d'avoir un différent δ_a pour chaque fibre, rendant la méthode utilisable pour les fibres ayant été ré-échantillonnées, tant que l'échantillonnage garanti une distance constante entre les points d'une même fibre.

5.5 Résultats et discussion

Nous avons testé notre algorithme sur des fibres construites à partir d'un sujet sélectionné aléatoirement dans la base du *Human Connectome Project* [Van Essen et al., 2012]. `qfib` nécessite l'utilisation d'un pas constant. Les méthodes utilisées pour la construction du tractogramme sont *SD_STREAM*, une méthode déterministe et *iFOD1* une méthode probabiliste de la suite logicielle *MRtrix* [Tournier et al., 2012]. Les tests ont été effectués sur 12 jeux de fibres différents faisant varier la taille du pas δ entre $0.1\ mm$ et $1\ mm$, et le nombre de fibres entre $60k$ et $3M$. L'angle maximal n'est pas spécifié quand les fibres sont générées, impliquant que leur valeur est déduite de la formule : $angle\ maximal = 90° \times \delta / voxelsize$. La taille des voxels (résolution de l'IRM) est de $1.25\ mm^3$. La longueur des fibres est contrainte entre $40\ mm$ et $256\ mm$. Toutes les configurations étudiées sont présentées dans le tableau 5.1. Les tailles de fichiers sont données pour le format tck.

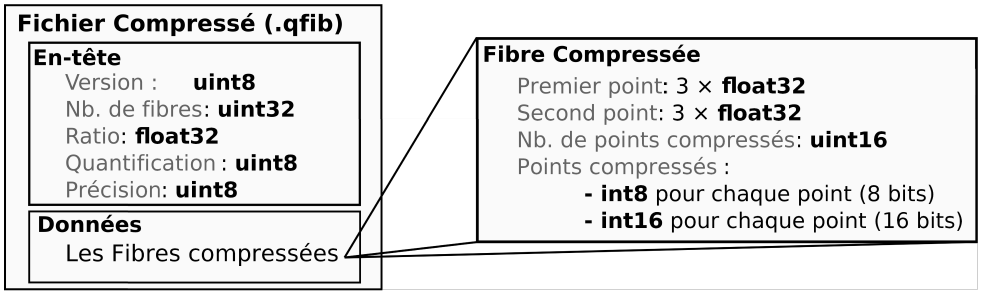


FIGURE 5.6 – `qfib` – Le format utilisé pour stocker les fibres compressées.

TABLE 5.1 – Taille des fichiers (tck).

Pas δ	0.1 mm		0.2 mm		0.5 mm	
Nb. de fibres	500k	3M	500k	3M	500k	3M
Taille de l'entrée (Go)						
Déterministe	3.80	22.8	1.92	11.5	0.84	5.01
Probabiliste	4.36	26.2	2.35	14.1	1.26	7.53

Notre méthode est générale et tout modèle de diffusion ou algorithme de tractographie pourrait être exploité, du moment que la contrainte de distance constante entre les points d'une même fibre δ_a est respectée. Tout comme `UniQuant`, même si la technique est montrée avec la quantification octaédrique et la quantification selon l'ensemble de points sphériques de Fibonacci, elle est utilisable avec n'importe quelle méthode de quantification de vecteurs unitaires actuelle ou future.

Les comparaisons sont faites avec l'état de l'art en compression de tractogramme : `zfib` ([Presseau et al., 2015]) après avoir montré son fonctionnement sur des exemples jouet. Nous montrons également que la méthode fonctionne en hors coeur (*out-of-core*), permettant de traiter des jeux de données limités uniquement par la taille du disque dur.

5.5.1 Erreur

Pour mieux illustrer l'erreur induite par la méthode de compression, nous montrons dans la figure 5.7 quatre différentes fibres (en noires) et leur version compressée et décompressée (en vert). Comme nous quantifions la direction relative à la précédente pour chaque segment, une ligne droite reste droite, sans introduire d'erreur (Figure 5.7(a)). Cependant, lorsque nous introduisons une courbure

plus importante, nous observons alors que l'erreur augmente en fonction de l'angle maximum comme le montre les exemples des figures 5.7(b), 5.7(c), 5.7(d), pour lesquels les angles sont respectivement 40.3°, 45.7° et 73.7°. Avec la courbe en spirale (Figure 5.7(d)), l'angle est trop important pour que la transformation soit efficace. Dans ce cas, l'erreur est plus importante et la différence avec la courbe d'origine est plus visible que sur les autres courbes. Cependant, cette fibre n'est pas anatomiquement viable, et une telle courbure ne doit pas apparaître dans un tractogramme.

Notre compression garde le nombre original de points. L'erreur peut donc être mesurée par point. Les erreurs sont calculées entre les courbes d'origine contenant les fibres $f_a = \{p_{a,1} \dots, p_{a,N_a}\}$ et les fibres compressées et décompressées $f'_a = \{p'_{a,1}, \dots, p'_{a,N_a}\}$ selon la formule suivante :

$$\begin{aligned} erreur\ moyenne &= \frac{\sum_{a=1}^{L} \sum_{b=1}^{N_a} ||p_{a,b} - p'_{a,b}||}{\sum_{a=1}^{L} N_a} \\ erreur\ max &= max\{||p_{a,b} - p'_{a,b}|| \mid a \in [1, L],\ b \in [1, N_a]\} \end{aligned} \tag{5.3}$$

où N_a est le nombre de points dans la fibre f_a, et L le nombre de fibres dans le jeu de données. Le tableau 5.3 montre ces erreurs pour chaque fichier décrit dans le tableau 5.1. L'augmentation de la précision de la quantification (nombre de bits utilisés pour la quantification) réduit grandement les erreurs grâce à l'augmentation du nombre de points de quantification. Cependant, cela réduit le ratio de compression.

Même s'il est recommandé d'utiliser un pas d'un dixième de la taille d'un voxel pour construire les fibres (entre $0.1\ mm$ et $0.2\ mm$ dans notre cas), il est possible d'utiliser des plus grandes valeurs à la construction. Dans ces cas, d'après les erreurs obtenues, il est préférable d'utiliser une quantification sur 16 bits. La méthode utilisée pour réaliser la quantification impacte l'erreur obtenue. La quantification selon l'ensemble de points sphériques de Fibonacci fournit une erreur plus faible que celle induite par la quantification octaédrique. Comme dans le cas de `UniQuant`, nous choisirons la méthode de quantification en fonction du cas d'application.

La Figure 5.8 évalue l'erreur en fonction de la longueur de la fibre (discrétisée tout les $10\ mm$). Les tractogrammes ont été calculés avec la méthode probabiliste et un pas de $0.1\ mm$. Ils ont été compressés avec la quantification octaédrique avec une précision de 8 bits. L'histogramme montre que l'erreur maximale semble indépendante de

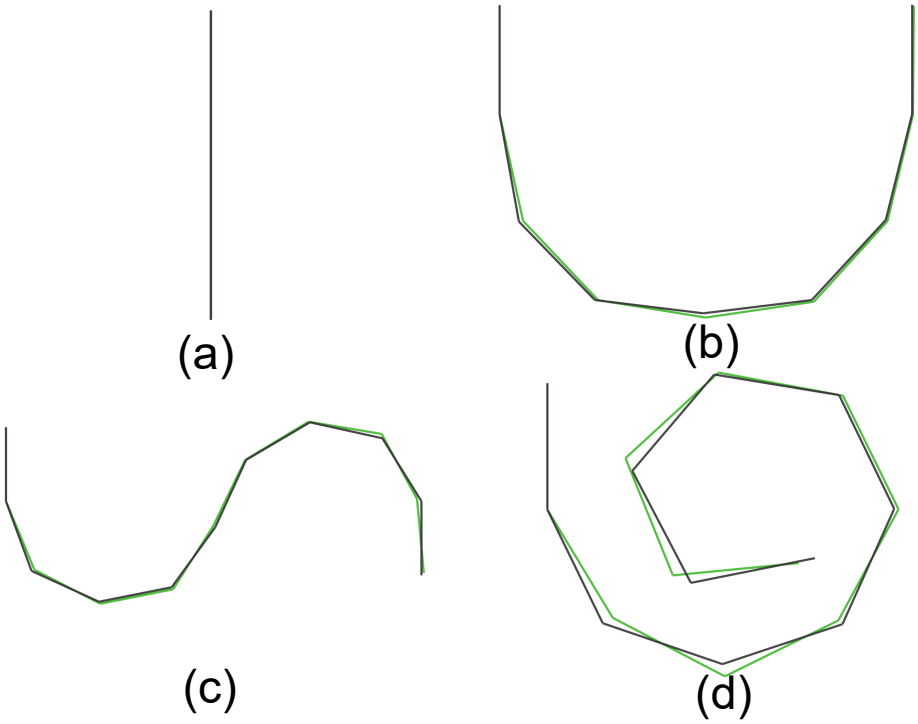


FIGURE 5.7 – Fibre originale (en noir), compressée et décompressée (en vert) avec la quantification octaédrique avec une précision de 8 bits.

la longueur de la fibre, par rapport à l'erreur moyenne qui augmente avec la longueur de la fibre.

Le tableau 5.2 montre l'erreur maximale et moyenne pour les points aux extrémités des fibres. Ces points sont importants pour les chercheurs en neurosciences car ce sont ceux qui sont utilisés pour construire des cartes mettant en correspondance les zones d'influence dans le cerveau. Ils représentent la connectivité de la fibre. Nous observons alors que pour un pas $\delta_a \leq 0.2\ mm$, une quantification octaédrique avec une précision de 8 bits est suffisante, alors qu'au delà, il est préférable d'avoir recours à une quantification sur 16 bits.

Pour visualiser ces erreurs, nous colorons la distance des fibres originales à leur version compressée et décompressée sur la figure 5.9. La quantification utilisée est la quantification octaédrique. L'échelle va de $0\ mm$ (bleu foncé) à $1.25\ mm$ (rouge foncé), soit la taille du voxel pour nos données. Les fibres sont calculées avec un pas de $1\ mm$ avec la méthode déterministe. Ce cas est l'exemple des situations où il est nécessaire d'utiliser une précision de 16 bits pour la quantification.

TABLE 5.2 – Erreur maximale et moyenne aux extrémités des fibres.

Pas δ		0.1 mm		0.2 mm		0.5 mm	
Nb. de fibres		500k	3M	500k	3M	500k	3M
Erreur maximale ($\times 10^{-2}\ mm$)							
Quantification	Précision	Déterministe					
Fibonacci	8 bits	4.91	5.28	10.3	10.6	32.9	33.5
	16 bits	0.50	0.57	0.28	0.28	0.36	0.38
Octaédrique	8 bits	7.53	8.03	15.9	16.5	46.1	50.7
	16 bits	0.50	0.56	0.26	0.29	0.47	0.46
Quantification	Précision	Probabiliste					
Fibonacci	8 bits	2.76	2.85	5.25	5.43	18.4	18.6
	16 bits	0.13	0.13	0.12	0.14	0.64	0.63
Octaédrique	8 bits	4.78	4.78	8.30	8.64	26.7	31.8
	16 bits	0.13	0.13	0.14	0.15	0.79	0.80
Erreur moyenne ($\times 10^{-3}\ mm$)							
Quantification	Précision	Déterministe					
Fibonacci	8 bits	19.9	19.9	42.2	42.2	109	109
	16 bits	0.44	0.44	0.32	0.32	1.17	1.18
Octaédrique	8 bits	29.0	29.0	61.4	61.4	161	161
	16 bits	0.44	0.44	0.34	0.34	1.31	1.31
Quantification	Précision	Probabiliste					
Fibonacci	8 bits	17.5	17.5	25.1	25.3	62.5	62.7
	16 bits	0.21	0.21	0.41	0.41	2.12	2.12
Octaédrique	8 bits	28.0	28.1	40.7	40.5	94.4	94.7
	16 bits	0.22	0.22	0.46	0.46	2.36	2.39

L'erreur obtenue avec la précision de 16 bits (Figure 5.9(b)) est presque invisible sur l'image comparativement à celle obtenue avec une quantification exploitant une quantification octaédrique avec une précision de 8 bits (Figure 5.9(a)).

5.5.2 Taux de compression

Le taux de compression est calculé selon la formule suivante :

$$taux\ de\ compression = 100 \times (1 - \frac{taille\ compressee}{taille\ originale}) \tag{5.4}$$

Notre ratio de compression Table 5.4 varie peu, autour de 90% avec une quantification de précision 8 bits et 82% avec une précision de 16 bits. Pour les mêmes valeurs d'erreurs, nos ratios de compression

TABLE 5.3 – Erreur maximale et moyenne de notre méthode en fonction du jeu de données, de la précision de la quantification et de la méthode de quantification utilisée.

Pas δ		0.1 mm		0.2 mm		0.5 mm	
Nb. de fibres		500k	3M	500k	3M	500k	3M
Erreur maximale ($\times 10^{-2}\ mm$)							
Quantification	Précision	Déterministe					
Fibonacci	8 bits	4.94	5.30	10.3	10.6	33.2	34.1
	16 bits	0.50	0.57	0.28	0.28	0.39	0.38
Octaédrique	8 bits	7.53	8.03	16.5	16.5	46.7	51.0
	16 bits	0.50	0.56	0.27	0.29	0.50	0.52
Quantification	Précision	Probabiliste					
Fibonacci	8 bits	3.04	2.95	5.86	5.91	19.7	20.6
	16 bits	0.13	0.14	0.14	0.16	0.69	0.72
Octaédrique	8 bits	4.79	4.90	8.55	9.38	29.8	31.7
	16 bits	0.13	0.14	0.17	0.19	0.87	0.93
Erreur moyenne($\times 10^{-3}\ mm$)							
Quantification	Précision	Déterministe					
Fibonacci	8 bits	2.54	0.52	9.10	2.04	55.8	9.76
	16 bits	0.10	0.03	0.10	0.03	0.47	0.11
Octaédrique	8 bits	3.12	0.55	12.4	2.29	66.8	18.8
	16 bits	0.10	0.03	0.10	0.03	0.64	0.15
Quantification	Précision	Probabiliste					
Fibonacci	8 bits	1.35	0.22	5.01	0.83	19.2	3.73
	16 bits	0.04	0.01	0.08	0.01	0.63	0.16
Octaédrique	8 bits	2.69	0.45	5.21	1.11	37.5	6.26
	16 bits	0.04	0.01	0.08	0.02	0.94	0.20

sont strictement meilleurs que `zfib` pour une quantification sur 8 bits avec des pas inférieurs à $0.5\ mm$, ce qui correspond aux valeurs de pas recommandées et couramment utilisées. Quand l'erreur de `zfib` est à 0.2 mm (leur valeur par défaut), nous avons soit un meilleur taux de compression, soit une erreur plus faible. Pour les fibres générées avec la méthode probabiliste, qui sont moins régulières et mettent à l'épreuve la phase de linéarisation de `zfib`, nous avons des meilleurs taux de compression avec la même valeur d'erreur sur l'ensemble des exemples examinés.

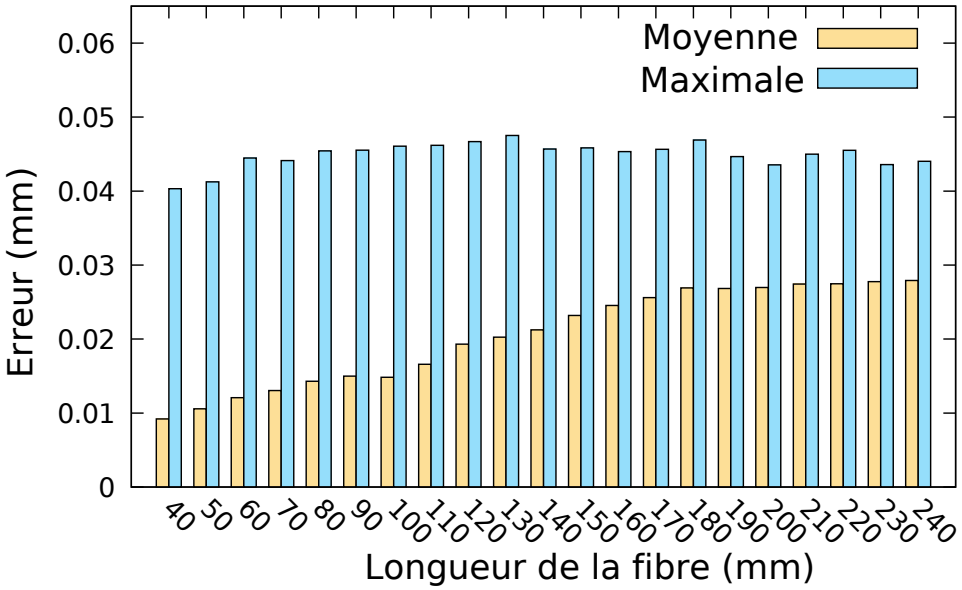


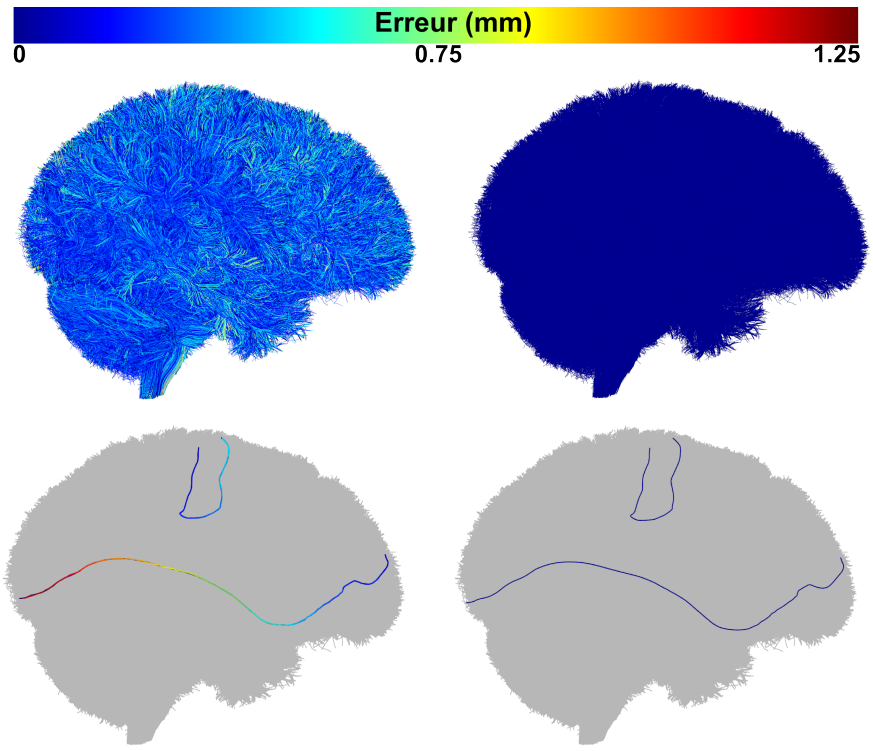
FIGURE 5.8 – Erreur en mm en fonction de la longueur de la fibre pour une quantification octaédrique avec une précision de 8 bits et un pas $\delta = 0.1\ mm$.

TABLE 5.4 – Taux de compression.

Taux de compression de `qfib` et `zfib`. Les valeurs N/A sont celles pour lesquelles l'algorithme n'a pas été capable d'effectuer la compression et la décompression.

Pas δ		0.1 mm		0.2 mm		0.5 mm	
Nb. de fibres		500k	3M	500k	3M	500k	3M
Taux de compression (en pourcentage)							
Méthode	Paramètres	Déterministe					
`qfib`	8 bits	91.4	91.4	91.1	91.1	90.4	90.4
	16 bits	83.1	83.1	82.8	82.8	82.3	82.2
`zfib`	même*	N/A	N/A	78.4	N/A	96.6	96.8
	0.2 mm	98.1	98.1	95.9	96.0	87.5	87.5
Méthode	Paramètres	Probabiliste					
`qfib`	8 bits	91.4	91.4	91.2	91.2	90.8	90.8
	16 bits	83.1	83.1	82.9	82.9	82.6	82.6
`zfib`	même*	N/A	N/A	78.1	N/A	87.1	N/A
	0.2 mm	96.0	N/A	88.7	N/A	69.9	N/A

même* : même erreur que `qfib` en utilisant une quantification octaédrique avec une précision de 8 bits (Table.5.3).



(a) Quantification Octahédrique, 8 bits

(b) Quantification Octahédrique, 16 bits

FIGURE 5.9 – Fibre dont la couleur représente l'erreur de compression locale. Les fibres indépendantes sont les plus défavorables extraites des deux jeux de données colorés complets montrés. Le jeu de données a été calculé avec un algorithme déterministe et avec un pas $\delta = 1\ mm$.

5.5.3 Temps de calculs

Dans le tableau 5.5, nous montrons les temps de compression et de décompression de `qfib` et de `zfib`. Ils ont été obtenus sur un ordinateur disposant d'un *Intel Xeon E5-1650v4* (6 cœurs, 12 threads, 3.6GHz). Avec chaque méthode, nous comptons uniquement les temps de compression et de décompression, sans les temps de lecture et d'écriture sur le disque dur. Nous définissons l'erreur de `zfib` à 0.2 mm car c'est leur erreur par défaut sur le code source fourni. Nous observons que sur la compression et la décompression, notre méthode est au moins deux ordres de grandeur plus rapide que `zfib`. De plus, durant nos essais avec le code source fourni par les auteurs, nous avons pu observer que `zfib` ne réussit pas à compresser des fichiers de trop grande taille alors que notre méthode passe à l'échelle.

TABLE 5.5 – Temps de calcul de `qfib` et `zfib`. Avec `zfib`, nous fixons l'erreur maximale à 0.2 mm. Les N/A sont les valeurs pour lesquelles l'algorithme n'a pas été capable d'effectuer la compression et la décompression.

Pas δ		0.1 mm		0.2 mm		0.5 mm	
Nombre de fibres		500k	3M	500k	3M	500k	3M
Temps de Compression (sec)							
Déterministe	`qfib` (fibo)	24.1	144	12.8	74.8	5.49	32.0
	`qfib` (octa)	7.83	46.5	3.81	22.6	1.67	9.76
	`zfib`	702	4243	387	2284	387	2373
Probabiliste	`qfib` (fibo)	27.9	167	15.3	90.6	8.27	49.7
	`qfib` (octa)	8.61	54.7	4.86	29.0	2.53	15.5
	`zfib`	910	N/A	1052	N/A	1418	N/A
Temps de décompression (sec)							
Déterministe	`qfib` (fibo)	4.98	30.1	2.61	15.4	1.14	6.79
	`qfib` (octa)	3.56	20.7	1.90	11.3	0.88	5.30
	`zfib`	12.1	72.7	12.9	77.1	17.3	103
Probabiliste	`qfib` (fibo)	5.77	34.9	3.23	18.9	1.75	10.3
	`qfib` (octa)	4.08	24.3	2.24	13.3	1.29	7.47
	`zfib`	28.5	N/A	43.0	N/A	60.8	N/A

5.5.4 Version hors coeur

Pour étendre le passage à l'échelle de notre méthode, nous avons implémenté une version hors coeur de notre algorithme. Les temps de compression sont présentés dans le tableau 5.6. Cette version est ralentie par rapport à la version classique dont les temps étaient rapportés sur le tableau 5.5. Cela peut être expliqué par la nécessité d'effectuer des lectures écritures en permanence durant la compression et la décompression sur le disque dur. Cet algorithme requiert seulement quelques mégaoctets de RAM pour travailler, quelle que soit la taille du jeu de données en entrée. Cela rend possible son utilisation sur n'importe quel ordinateur. Pour le démontrer, nous avons décidé de faire tourner l'algorithme sur un système embarqué avec un jeu de données très important. Nous avons généré un tractogramme de 10 millions de fibres, avec un pas de $0.1\ mm$ avec la méthode probabiliste, le fichier généré faisant 87.3 Go. Pour la partie matérielle nous avons utilisé un Raspberry Pi 2 disposant de 1 Go de RAM avec un processeur quad-core à 900 MH, les données étant stockées sur un disque dur externe connecté au Raspberry Pi avec l'interface USB 2.0. La compression a résulté en un fichier de 7.49

Go pour un taux de compression de 91.4% avec une quantification octaédrique d'une précision de 8 bits. La compression a pris 4 heures et la décompression 5.3 heures. Ces temps de calculs peuvent être expliqués par l'utilisation de l'interface USB 2.0, faisant exploser le temps de lecture et d'écriture. Cela explique que la décompression ait pris plus de temps que la compression, car l'écriture est plus lente que la lecture de données.

TABLE 5.6 – Temps de compression de la méthode hors coeur (`qfib`) avec la quantification octaédrique et une précision de 8 bits.

Pas δ	0.1 mm		0.2 mm		0.5 mm	
Nb. de fibres	500k	3M	500k	3M	500k	3M
Temps de compression (sec)						
Déterministe	29.1	166	14.3	88.3	6.82	40.8
Probabiliste	33.0	192	17.9	103	9.99	57.9

5.5.5 Limitations

La limitation principale de cette méthode de compression est la nécessité d'avoir une distance constante entre les points d'une fibre. Il est possible d'adapter les données à cette contrainte en utilisant des méthodes de ré-échantillonage de fibres à pas constant, ce qui est couramment utilisé pour réduire la taille des données. L'algorithme fonctionne sur les tractographies du cerveau et utilise les contraintes qui existent sur ce type de données. Elles ne sont pas les mêmes sur d'autres types de données comme les tractographies du pelvis sur lesquelles il faudra adapter la méthode. En effet, sur ce type de données, l'angle maximal entre des segments consécutifs est plus important et la transformation ne sera pas forcément aussi efficace comme illustré dans la Figure 5.7. Dans un tel cas, il sera nécessaire d'utiliser une quantification sur 16 bits ou plus, réduisant ainsi le taux de compression.

5.5.6 Travaux Futures

Cette méthode est *parfaitement parallèle*. Cela signifie qu'elle peut être facilement parallélisée par fibre. De ce fait, il serait trivial de faire une version GPU et cela pourrait réduire de manière importante les temps de compression et de décompression. Ce faisant, il serait

intéressant d'exploiter la décompression à la volée pour réaliser la visualisation du jeu de données stockées de manière compressé sur la VRAM, en décompressant indépendamment chaque fibre à la volée. Cela réduirait la nécessité d'avoir une quantité importante de RAM ou de VRAM sur la machine, et améliorerait les temps de transfert entre la RAM et la VRAM. Cela permettrait à des médecins disposant de machines peu puissantes de pouvoir visualiser ou traiter des jeux de données bien plus grands. Il pourrait être intéressant de générer les données directement dans ce format, ce qui serait possible du fait de la méthode de construction utilisée, suivant le gradient du déplacement des cellules d'eau du cerveau.

Les autres méthodes de compressions ne doivent cependant pas être mises de côté. Il serait notamment intéressant d'utiliser une représentation plus parcimonieuse des données, avec par exemple des courbes de Bézier, des B-Splines, ou des NURBS.

5.6 Conclusion

Nous avons présenté un nouvel algorithme de compression de fibres pour les tractogrammes du cerveau, et son format d'encodage associé – `qfib`. Nous l'avons évalué et validé en l'utilisant sur une grande variété de configurations de tractogrammes avec des pas et des nombres différents de fibres. Cet algorithme fournit des erreurs plus faibles dans le cas général, et des meilleurs temps de compression que les méthodes existantes. Le taux de compression est important, entre 80% et 90%, avec des erreurs en dessous de la précision de l'IRM. Contrairement à d'autres méthodes, `qfib` ne retire aucun point du jeu de données original, ce qui est important pour les traitements basés sur les points que l'on pourrait vouloir appliquer par la suite. De plus, les étapes de compression et de décompression gèrent chaque fibre individuellement. Cela implique qu'elles peuvent être facilement parallélisées, et que `qfib` est efficace sur des jeux de données de toutes tailles contrairement aux méthodes basées dictionnaire. Cela implique également qu'il est possible de décompresser des fibres individuelles de la représentation compressée, permettant d'accéder directement à des sous parties du jeu de données, de faire des traitements à la volée, avec un impact mémoire et des temps de chargement négligeables. De ce fait, notre méthode ouvre la porte aux applications traitant les données à la volée dans lesquelles l'algorithme travaillerait avec les données compressées en mémoire et décompresserait des fibres ou des petits paquets de fibres avant de leur appliquer des traitements. Cela

réduit le besoin en RAM et VRAM, et nous avons également démontré que notre algorithme fonctionne en hors coeur pour le traitement des données les plus lourdes. Dans ce dernier cas, la taille des données est uniquement limitée par la taille du disque dur.

CHAPITRE 6

Conclusion

6.1 Rappel des contributions

Cette thèse s'axe autour de trois contributions majeures.

La première contribution, présentée dans le chapitre 3 est `UniQuant`, une méthode de compression à la volée avec perte d'ensemble non ordonnés de vecteurs unitaires. Cette dernière crée de la cohérence et l'exploite dans le but de compresser des petits ensembles de vecteurs pour lesquels un espace de représentation est spécifiquement créé à partir de la sphère unitaire grâce à une transformation. Son utilité est démontrée avec un premier exemple de compression à la volée de nuages de points disposant de normales. Cette première partie a fait l'objet de deux publications, et son code est accessible librement en ligne.

— **Fast Lossy Compression of 3D Unit Vector Sets** (Publication conférence)
Sylvain Rousseau et Tamy Boubekeur
SIGGRAPH Asia Technical Brief 2017
pages 1–4, article No. 23
[Rousseau and Boubekeur, 2017]

— **Unorganized Unit Vectors Sets Quantization** (Publication journal)
Sylvain Rousseau et Tamy Boubekeur
Journal on Computer Graphics Technics (JCGT)
Volume 9, No. 3, pages 92–107
[Rousseau and Boubekeur, 2020]

— **Code source** : `https://github.com/superboubek/UniQuant`

Dans un second temps, une deuxième contribution est présentée, sous la forme d'une étude de l'impact de la quantification des directions sur le rendu de Monte Carlo. Elle est complétée avec la quantification des origines des vecteurs sur la surface des portails

pour aboutir à une méthode complète de compression d'ensemble de rayons dans le cadre du rendu distribué de Monte Carlo avec portails pour leur transfert réseau.

Enfin, une dernière contribution est présentée, dans la modélisation de données 3D en imagerie médicale sur les tractogrammes du cerveau. Cette dernière contribution fournit aux neurochirurgiens et aux chercheurs une méthode permettant en quelques secondes de diviser par 10 la taille de leurs données grâce à une adaptation de `UniQuant` à ce type de données qui partagent de nombreuses contraintes mathématiques avec les chemins lumineux. Cette dernière contribution à fait l'objet d'une publication, d'un poster, et son code est librement accessible en ligne.

— **QFib : Fast and Accurate Compression of White Matter Tractograms (Poster)**
Sylvain Rousseau*, Corentin Mercier*, Pietro Gori, Isabelle Bloch et Tamy Boubekeur
Organization on Human Brain Mapping 2019 (OHBM2019)
[Rousseau et al., 2019]

— **QFib : Fast and Efficient Brain Tractogram Compression** (publication journal)
Corentin Mercier*, Sylvain Rousseau*, Pietro Gori, Isabelle Bloch et Tamy Boubekeur
Volume 18, pages 627–640 (2020)
NeuroInformatics
[Mercier et al., 2020]

— **Code source** : `https://github.com/syrousseau/qfib`

L'état de l'art a également été présenté, et notamment l'implémentation de Hurricane présenté dans le chapitre 2 lors d'une présentation publique.

— **Rendu de Monte Carlo élastique (publication conférence)**
Sylvain Rousseau et Tamy Boubekeur
JFIG 2016
[Rousseau and Boubekeur, 2016]

6.2 Perspectives

Cette thèse aura permis de réaliser le processus complet du développement d'un outil méthodologique permettant d'adapter l'espace de représentation de vecteurs unitaires à des petits ensembles cohérents puis d'en réaliser une application dans le domaine de

l'informatique graphique 3D et une ouverture à la modélisation numérique avec le domaine de l'imagerie médicale. Ces deux derniers chapitres ouvrent de nombreux sujets de recherche. La discrétisation du champ de lumière sous forme de portails de lumière correspond à une réduction du nombre de dimensions qui pourrait être exploitée en apprentissage artificiel pour la compression des rayons. La représentation avec `UniQuant` va également dans ce sens. Les auto-encodeurs présentent une piste prometteuse.

Les tractogrammes étaient jusqu'ici limités dans leur utilisation du fait de leur taille. L'algorithme proposé va permettre un plus large accès à ces données et développer les recherches dans le domaine. Cet intérêt a été démontré par la rapidité à laquelle les laboratoires externes ont commencé à l'utiliser. Il serait intéressant de revenir au rendu et d'utiliser la compression proposée pour faire de la décompression à la volée sur le GPU des données compressées. Considérant les quantités de mémoire vidéos sur les dernières cartes graphiques, cela permettrait alors de visualiser directement les jeux de données étudiés sans avoir à les simplifier. De nouveaux cas d'applications pourraient également être trouvés, dans les domaines ayant recours à des ensembles de vecteurs unitaires désordonnés.

Bibliographie

[Aberman et al., 2017] Aberman, K., Katzir, O., Zhou, Q., Luo, Z., Sharf, A., Greif, C., Chen, B., and Cohen-Or, D. (2017). Dip transform for 3D shape reconstruction. *ACM Trans. Graph.*, 36(4) :79 :1–79 :11. 3.4.1

[Adhikarla et al., 2017] Adhikarla, V. K., Vinkler, M., Sumin, D., Mantiuk, R., Myszkowski, K., Seidel, H.-P., and Didyk, P. (2017). Towards a quality metric for dense light fields. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 4.4

[Alexandroni et al., 2017] Alexandroni, G., Zimmerman Moreno, G., Sochen, N., and Greenspan, H. (2017). The fiber-density-coreset for redundancy reduction in huge fiber-sets. *NeuroImage*, 146 :246–256. 5.3.1

[Appel, 1968] Appel, A. (1968). Some techniques for shading machine rendering of solids. *AFIPS Conference Proceedings 32*, pages 37 – 45. 2.1

[Berger et al., 2017] Berger, M., Tagliasacchi, A., Seversky, L. M., Alliez, P., Guennebaud, G., Levine, J. A., Sharf, A., and Silva, C. T. (2017). A survey of surface reconstruction from point clouds. *Comput. Graph. Forum*, 36(1) :301–329. 3.4.1

[Beyer et al., 2015] Beyer, J., Hadwiger, M., and Pfister, H. (2015). State-of-the-art in GPU-based large-scale volume visualization. *Comput. Graph. Forum*, 34(8) :13–37. 2.3.3

[Bitterli, 2016] Bitterli, B. (2016). Rendering resources. https ://benedikt-bitterli.me/resources/. 4.1

[Cao et al., 2019] Cao, C., Preda, M., and Zaharia, T. (2019). 3D point cloud compression : A survey. In *The 24th International Conference*

on 3D Web Technology, Web3D '19, pages 1–9, New York, NY, USA. ACM. 3.4.2

[CCITT, 1992] CCITT (1992). Digital compression and coding of continuous-tone still images – requirement and guidelines. 3.2.2

[Christensen, 2008] Christensen, P. H. (2008). Point-based approximate color bleeding. *Pixar Technical Memo 08-01*. 2.1

[Christensen and Jarosz, 2016] Christensen, P. H. and Jarosz, W. (2016). The path to path-traced movies. *Foundations and Trends in Computer Graphics and Vision*, 10(2) :103–175. 1.2, 2.1

[Chung et al., 2009] Chung, M. K., Adluru, N., Lee, J. E., Lazar, M., Lainhart, J. E., and Alexander, A. L. (2009). Efficient parametric encoding scheme for white matter fiber bundles. *IEEE EMBS*, 2009 :6644–6647. 5.3.1

[Cigolle et al., 2014] Cigolle, Z. H., Donow, S., Evangelakos, D., Mara, M., McGuire, M., and Meyer, Q. (2014). A survey of efficient representations for independent unit vectors. *Journal of Computer Graphics Techniques (JCGT)*, 3(2) :1–30. 2.5.3, 3.2.5, 3.3.1, 3.3.2

[Collet, 2011] Collet, Y. (2011). LZ4 : Extremely fast compression algorithm. 2.5

[Cook et al., 1987] Cook, R. L., Carpenter, L., and Catmull, E. (1987). The reyes image rendering architecture. *SIGGRAPH Comput. Graph.*, 21(4) :95–102. 2.1

[Dean and Ghemawat, 2004] Dean, J. and Ghemawat, S. (2004). MapReduce : Simplified data processing on large clusters. In *OSDI'04 : Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA. 2.3.1

[Deering, 1995] Deering, M. (1995). Geometry compression. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 13–20, New York, NY, USA. Association for Computing Machinery. 2.5.1

[Demir and ¸Cetingül, 2015] Demir, A. and ¸Cetingül, H. E. (2015). Sequential Hierarchical Agglomerative Clustering of White Matter Fiber Pathways. *IEEE Trans. Biomed. Eng.*, 62(6) :1478–1489. 5.3.1

[Digne et al., 2014] Digne, J., Chaine, R., and Valette, S. (2014). Self-similarity for accurate compression of point sampled surfaces.

Computer Graphics Forum, 33(2) :155–164. 2.5.1, 3.4.2

[Eisenacher et al., 2013] Eisenacher, C., Nichols, G., Selle, A., and Burley, B. (2013). Sorted deferred shading for production path tracing. *CGF*, 32(4) :125–132. 2.3.2, 3.1, 3.2.1, 3.2.2

[Gao et al., 2005] Gao, J., Huang, J., Johnson, C. R., and Atchley, S. (2005). Distributed data management for large volume visualization. In *VIS 05. IEEE Visualization, 2005.*, pages 183–189. 2.3.2

[Garyfallidis et al., 2012] Garyfallidis, E., Brett, M., Correia, M. M., Williams, G. B., and Nimmo-Smith, I. (2012). QuickBundles, a Method for Tractography Simplification. *Frontiers in Neuroscience*, 6(175). 5.3.1

[Georgiev et al., 2018] Georgiev, I., Ize, T., Farnsworth, M., Montoya-Vozmediano, R., King, A., Lommel, B. V., Jimenez, A., Anson, O., Ogaki, S., Johnston, E., Herubel, A., Russell, D., Servant, F., and Fajardo, M. (2018). Arnold : A brute-force production path tracer. *ACM Trans. Graph.*, 37(3) :32 :1–32 :12. 2.1

[González, 2010] González, A. (2010). Measurement of Areas on a Sphere Using Fibonacci and Latitude-Longitude Lattices. *Mathematical Geosciences*, 42(1) :49–64. 2.5.3

[Gori et al., 2016] Gori, P., Colliot, O., Marrakchi-Kacem, L., Worbe, Y., Fallani, F. D. V., Chavez, M., Poupon, C., Hartmann, A., Ayache, N., and Durrleman, S. (2016). Parsimonious Approximation of Streamline Trajectories in White Matter Fiber Bundles. *IEEE Trans. Med. Imag.*, 35(12) :2609–2619. 5.3.1

[Górski et al., 2005] Górski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., and Bartelmann, M. (2005). HEALPix : A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere. *Astrophysical Journal*, 622 :759–771. 2.5.3

[Guevara et al., 2011] Guevara, P., Poupon, C., Rivière, D., Cointepas, Y., Descoteaux, M., Thirion, B., and Mangin, J. F. (2011).

Robust clustering of massive tractography datasets. *NeuroImage*, 54(3) :1975–1993. 5.3.1

[Günther and Grosch, 2014] Günther, T. and Grosch, T. (2014). Distributed out-of-core stochastic progressive photon mapping. *Computer Graphics Forum*. 2.3.3

[Huang et al., 2006] Huang, Y., Peng, J., Kuo, C.-C. J., and Gopi, M. (2006). Octree-based progressive geometry coding of point clouds. In *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, SPBG'06, page 103–110, Goslar, DEU. Eurographics Association. 2.5.1

[Ize et al., 2011] Ize, T., Brownlee, C., and Hansen, C. D. (2011). Real-time ray tracer for visualizing massive models on a cluster. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, EGPGV '11, pages 61–69. Eurographics Association. 2.3.2

[Kajiya, 1986] Kajiya, J. T. (1986). The rendering equation. *Proc. SIGGRAPH*, 20(4) :143–150. (document), 1.2, 1.3, 2.1, 2.2

[Kato and Saito, 2002] Kato, T. and Saito, J. (2002). "Kilauea" : Parallel global illumination renderer. In *Proc. EGPGV*, pages 7–16. 2.3.3

[Keinert et al., 2015] Keinert, B., Innmann, M., Sänger, M., and Stamminger, M. (2015). Spherical Fibonacci mapping. *ACM ToG*, 34(6) :193 :1–193 :7. 2.5.3, 2.5.3, 2.5.3, 3.2.2, 3.2.5, 5.4.2, 5.4.5

[Kolmogorov, 1963] Kolmogorov, A. N. (1963). On tables of random numbers. *Sankhyā Ser. A*, 25 :369–376. 2.5, 3.1

[Kumar and Desrosiers, 2016] Kumar, K. and Desrosiers, C. (2016). A sparse coding approach for the efficient representation and segmentation of white matter fibers. In *IEEE ISBI*, pages 915–919. 5.3.1

[Laine et al., 2013] Laine, S., Karras, T., and Aila, T. (2013). Megakernels considered harmful : wavefront path tracing on GPUs. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 137–143. ACM. 3.2.1, 3.2.2

[Lindstrom, 2014] Lindstrom, P. (2014). Fixed-rate compressed floating-point arrays. *IEEE TVCG*, 20(12) :2674–2683. 2.5

[Liu et al., 2012] Liu, M., Vemuri, B. C., and Deriche, R. (2012). Unsupervised automatic white matter fiber clustering using a

Gaussian mixture model. *IEEE International Symposium on Biomedical Imaging*, 2012(9) :522–525. 5.3.1

[Maddah et al., 2007] Maddah, M., Wells, W. M., Warfield, S. K., Westin, C.-F., and Grimson, W. E. L. (2007). Probabilistic Clustering and Quantitative Analysis of White Matter Fiber Tracts. In *IPMI*, volume 20, pages 372–383. 5.3.1

[Maglo et al., 2012] Maglo, A., Courbet, C., Alliez, P., and Hudelot, C. (2012). Progressive compression of manifold polygon meshes. *Computers & Graphics*, 36(5) :349 – 359. Shape Modeling International (SMI) Conference 2012. 2.5.1

[Maglo et al., 2015] Maglo, A., Lavoué, G., Dupont, F., and Hudelot, C. (2015). 3D mesh compression : Survey, comparisons, and emerging trends. 2.5.1

[Mercier et al., 2018] Mercier, C., Gori, P., Rohmer, D., Cani, M.-P., Boubekeur, T., Thiery, J.-M., and Bloch, I. (2018). Progressive and Efficient Multi-Resolution Representations for Brain Tractograms. In *EG VCBM*, pages 89–93. 5.3.1

[Mercier et al., 2020] Mercier, C., Rousseau, S., Gori, P., Bloch, I., and Boubekeur, T. (2020). QFib : fast and efficient brain tractogram compression. *Neuroinformatics*, 18 :627–640. 1.4.1, 5.1, 6.1

[Metropolis and Ulam, 1949] Metropolis, N. and Ulam, S. (1949). The Monte Carlo method. *Journal of the American Statistical Association*, 44 :335–341. 2.2

[Meyer et al., 2010] Meyer, Q., Süßmuth, J., Sußner, G., Stamminger, M., and Greiner, G. (2010). On floating-point normal vectors. In *Proc. EGSR*, pages 1405–1409. 2.5.3, 2.5.3, 2.8, 2.5.3, 3.2.5, 5.4.2, 5.4.5

[Molnar et al., 1994] Molnar, S., Cox, M., Ellsworth, D., and Fuchs, H. (1994). A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4) :23–32. 2.3

[Moreno et al., 2017] Moreno, G. Z., Alexandroni, G., Sochen, N., and Greenspan, H. (2017). Sparse Representation for White Matter

Fiber Compression and Calculation of Inter-Fiber Similarity. In *Computational Diffusion MRI*, pages 133–143. 5.3.1

[Morton, 1966] Morton, G. M. (1966). A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd, Ottawa, Canada. 3.2.2

[Nimier-David et al., 2019] Nimier-David, M., Vicini, D., Zeltner, T., and Jakob, W. (2019). Mitsuba 2 : A retargetable forward and inverse renderer. *ACM Trans. Graph.*, 38(6) :203 :1–203 :17. 2.3.1

[Northam et al., 2013] Northam, L., Smits, R., Daudjee, K., and Istead, J. (2013). Ray tracing in the cloud using MapReduce. In *Proc. HPCS*, pages 19–26. (document), 2.5, 2.3.3

[Novák et al., 2010] Novák, J., Havran, V., and Dachsbacher, C. (2010). Path regeneration for interactive path tracing. In *Eurographics (Short Papers)*, pages 61–64. 3.2.1, 3.2.2

[Peterka et al., 2008] Peterka, T., Yu, H., Ross, R. B., Ma, K.-L., et al. (2008). Parallel volume rendering on the IBM Blue Gene/P. In *EGPGV*, pages 73–80. 2.3.2

[Petrovic et al., 2007] Petrovic, V., Fallon, J., and Kuester, F. (2007). Visualizing Whole-Brain DTI Tractography with GPU-based Tuboids and LoD Management. *IEEE TVCG*, 13(6) :1488–1495. 5.3.1

[Peyré and Mallat, 2005] Peyré, G. and Mallat, S. (2005). Surface compression with geometric bandelets. *ACM Trans. Graph.*, 24(3) :601–608. 2.5.1

[Pharr et al., 2016] Pharr, M., Humphreys, G., and Jakob, W. (2016). PBRT version 3 rendering engine. 4.2, 4.4

[Presseau et al., 2015] Presseau, C., Jodoin, P.-M., Houde, J.-C., and Descoteaux, M. (2015). A new compression format for fiber tracking datasets. *NeuroImage*, 109 :73 – 83. 5.3.1, 5.5

[Rheault et al., 2017] Rheault, F., Houde, J.-C., and Descoteaux, M. (2017). Visualization, Interaction and Tractometry : Dealing with

Millions of Streamlines from Diffusion MRI Tractography. *Frontiers in Neuroinformatics*, 11 :42. 5.2

[Rodgers, 1985] Rodgers, D. (1985). Improvements in multiprocessor system design. *ACM SIGARCH Computer Architecture News*, 13(3) :225 – 231. 2.3

[Rousseau and Boubekeur, 2016] Rousseau, S. and Boubekeur, T. (2016). Rendu de Monte Carlo Élastique. (document), 1.4.1, 2.3.2, 6.1

[Rousseau and Boubekeur, 2017] Rousseau, S. and Boubekeur, T. (2017). Fast Lossy Compression of 3D Unit Vector Sets. In *SIGGRAPH Asia Tech. Briefs*, pages 23 :1–23 :4. 1.4.1, 3.1, 6.1

[Rousseau and Boubekeur, 2020] Rousseau, S. and Boubekeur, T. (2020). Unorganized unit vectors sets quantization. *Journal of Computer Graphics Techniques (JCGT)*, 9(3) :92–107. 1.4.1, 6.1

[Rousseau et al., 2019] Rousseau, S., Mercier, C., Gori, P., Bloch, I., and Boubekeur, T. (2019). QFib : fast and efficient fiber tracking dataset compression (poster). In *Organization on Human Brain Mapping*. 1.4.1, 5.1, 6.1

[Rui and Yue, 2014] Rui, L. and Yue, Z. (2014). An improved Monte Carlo ray tracing for large-scale rendering in Hadoop. *International Conference on Computer Science and Sevice System*, pages 609–613. 2.3.2

[Smith et al., 2012] Smith, J., Petrova, G., and Schaefer, S. (2012). Progressive encoding and compression of surfaces generated from point cloud data. *Computers & Graphics*, 36(5) :341 – 348. Shape Modeling International (SMI) Conference 2012. 2.5.1, 2.5.3, 3.4.2

[Soares et al., 2013] Soares, J., Marques, P., Alves, V., and Sousa, N. (2013). A hitchhiker's guide to diffusion tensor imaging. *Frontiers in Neuroscience*, 7 :31. 5.3.1

[Somers and Wood, 2012] Somers, B. and Wood, Z. J. (2012). FlexRender : a distributed rendering architecture for ray tracing huge scenes on commodity hardware. In *GRAPP/IVAPP*. 2.3.3

[Touma and Gotsman, 1998] Touma, C. and Gotsman, C. (1998). Triangle mesh compression. In *Proceedings of the Graphics Interface*

1998 Conference, June 18-20, 1998, Vancouver, BC, Canada, pages 26–34. 2.5.1

[Tournier et al., 2012] Tournier, J.-D., Calamante, F., and Connelly, A. (2012). MRtrix : Diffusion tractography in crossing fiber regions. *Int. J. of Imaging Systems and Technology*, 22(1) :53–66. 5.5

[Tournier et al., 2011] Tournier, J.-D., Mori, S., and Leemans, A. (2011). Diffusion tensor imaging and beyond. *Magnetic Resonance in Medicine*, 65(6) :1532–1556. 5.2

[Uthayakumar et al., 2018] Uthayakumar, J., Vengattaraman, T., and Dhavachelvan, P. (2018). A survey on data compression techniques : From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University - Computer and Information Sciences*. 2.5

[Van Antwerpen, 2011] Van Antwerpen, D. (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. 3.2.1, 3.2.2

[Van Essen et al., 2012] Van Essen, D., Ugurbil, K., Auerbach, E., et al. (2012). The Human Connectome Project : A data acquisition perspective. *NeuroImage*, 62(4) :2222–2231. 5.5

[Vorba et al., 2014] Vorba, J., Karlík, O., Šik, M., Ritschel, T., and Křivánek, J. (2014). On-line learning of parametric mixture models for light transport simulation. *ACM Trans. Graph.*, 33(4) :101 :1–101 :11. 2.2, 2.5, 2.6, 4.1

[Wald et al., 2001] Wald, I., Slusallek, P., and Benthin, C. (2001). Interactive distributed ray tracing of highly complex models. In Gortler, S. J. and Myszkowski, K., editors, *Rendering Techniques 2001*, pages 277–288, Vienna. Springer Vienna. 2.3.2

[Wald et al., 2014] Wald, I., Woop, S., Benthin, C., Johnson, G. S., and Ernst, M. (2014). Embree : A kernel framework for efficient CPU ray tracing. *ACM Trans. Graph.*, 33(4) :143 :1–143 :8. 2.3.2

[Wang et al., 2004] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment : from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4) :600–612. 4.4

[Wassermann et al., 2010] Wassermann, D., Bloy, L., Kanterakis, E., Verma, R., and Deriche, R. (2010). Unsupervised white matter fiber

clustering and tract probability map generation : Applications of a Gaussian process framework for white matter fibers. *NeuroImage*, 51(1) :228–241. 5.3.1

[Williams, 1978] Williams, L. (1978). Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3) :270–274. 2.1

ECOLE
DOCTORALE

Titre : Compression collaborative de rayons de lumière pour le rendu distribué de Monte Carlo et applications.

Mots clés : Informatique Graphique, Rendu de Monte Carlo, Compression de données, Vecteurs unitaires

Résumé : Cette thèse s'inscrit dans le domaine de l'informatique graphique et plus particulièrement de la synthèse d'images. Dans notre cas, nous exploitons la synthèse d'images photoréalistes, cherchant à imiter la physique pour obtenir les images les plus réalistes possibles. Dans ce contexte, nous nous intéressons à la compression du rayon de lumière. Cet élément est utilisé pour représenter les chemins reliant les sources de lumière à la caméra virtuelle. Nous présentons d'abord une méthode de compression de directions, une des composantes des rayons de lumière, avant de montrer son utilisation en combinaison avec une méthode de compression de positions afin de compresser des rayons complets dans le cadre d'un moteur de rendu fonctionnant en réseau avec de nombreux ordinateurs distants les uns des autres. Nous présentons enfin une adaptation de la méthode de compression développée pour des données médicales décrivant les zones d'influences neuronales du cerveau : les tractogrammes.

Title : Collaborative Light Ray compression for Distributed Monte Carlo Rendering and Applications.

Keywords : Computer Graphics, Monte Carlo Rendering, Data Compression, Unit Vectors

Abstract : This thesis is about computer graphics and specifically image synthesis. In our case, we use the algorithms of this subdomain that generate photorealistic images. In this context, take a closer look at light ray compression. This element is used to represent a light path connecting light sources to the virtual sensor (camera). We first present a new unit vector sets compression, that we use in combination with a positional compression algorithm to compress the whole light ray. We then apply it in the case of a rendering engine that works with numerous distant computers to generate single images. In the end, we present an adaptation of the first compression method to compress some medical data, used to describe the neuronal influences in the brain.

Institut Polytechnique de Paris
91120 Palaiseau, France