UNIVERSITÉ DE STRASBOURG

N° d'ordre : 1248

École Doctorale Mathématiques, Sciences de l'Information et de l'Ingénieur

UdS – INSA – ENGEES

THÈSE

présentée pour obtenir le grade de

Docteur de l'Université de Strasbourg
Discipline : Informatique

par

Alexandre Ancel

Éclairage pour visualisation volumique haute-performance

Soutenue publiquement le 4 novembre 2011

Membres du jury :

Directeur de thèse :	M. Jean-Michel Dischler, Professeur, Université de Strasbourg
Co-Directeur de thèse :	Mme. Catherine Mongenet, Professeur, Université de Strasbourg
Rapporteur externe :	M. Kadi Bouatouch, Professeur, Université de Rennes 1
Rapporteur externe :	M. Bruno Raffin, Chargé de recherches, INRIA, Grenoble
Examinateur externe :	M. Jean-Philippe Nominé, Ingénieur de recherche, CEA
Examinateur interne :	M. Pascal Schreck, Professeur, Université de Strasbourg

Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection UMR 7005

Remerciements

Je tiens en premier lieu à remercier Jean-Michel Dischler et Catherine Mongenet d'avoir accepté d'encadrer cette thèse, et de m'avoir aidé et soutenu au sein des différentes étapes de ce travail.

Je remercie Kadi Bouatouch et Bruno Raffin d'avoir accepté d'être rapporteurs pour ce manuscrit de thèse, ainsi que Jean-Philippe Nominé et Pascal Schreck pour avoir accepté d'être examinateurs.

Je souhaite aussi remercier l'ensemble des membres de l'ICPS pour m'avoir accueilli pendant ces 3 années de thèse, ainsi que de m'avoir fourni le matériel pour travailler. Merci à eux aussi pour leur bonne humeur et les nombreux pots/soirées qui ont eux lieu !

Je souhaite aussi remercier les personnes qui m'ont aidé dans mon travail pendant ces 3 années : Stéphane Marchesin pour m'en avoir appris beaucoup sur le rendu volumique, Amel Guétat et Guillaume Gilet, avec qui j'ai aussi eut le plaisir de travailler. Je remercie aussi Ogier Maitre pour m'avoir aidé matériellement, ainsi que pour ses conseils (notamment en CUDA).

Finalement et pour m'avoir soutenu dans les moments difficiles, je souhaite aussi remercier ma famille et l'ensemble de mes amis de Colmar et de Strasbourg.

Résumé

Le rendu volumique direct est une technique permettant la visualisation de données volumiques scalaires. L'objectif de cette technique est de permettre l'analyse des structures présentes dans ces données en leur associant des propriétés optiques (couleur, opacité) avec l'aide d'une fonction de transfert. L'ajout d'éclairage au rendu volumique direct permet d'apporter des indices visuels supplémentaires pour l'analyse des données. En effet, l'éclairage joue un rôle fondamental dans la manière dont nous percevons notre environnement, car il permet d'évaluer le positionnement et la forme des objets qui nous entourent.

Nous présentons dans cette thèse différentes techniques permettant d'améliorer l'intégration et l'utilisation d'éclairage dans le rendu volumique direct. Nous proposons ainsi une méthode d'interpolation non-linéaire qui permet d'améliorer la précision numérique des méthodes d'éclairage basées sur une normale dans le contexte du rendu volumique direct pré-intégré. Dans un second temps, nous nous intéressons à une technique d'éclairage spécifique : l'Ambient Occlusion (ou occultation ambiante en français). Nous introduisons pour celle-ci un découpage en structures qui permet d'améliorer la perception des structures internes en améliorant le contraste dans l'image finale. Afin de réduire les temps de pré-calcul inhérents à cette méthode, nous la parallélisons sur une architecture multi-GPU, en intégrant la gestion de l'équilibrage de charge et s'appuyant sur deux approches différentes de répartition : statique et dynamique.

Abstract

Direct volume rendering allows the visualization of scalar volume data. This technique aims at helping the analysis of the different features in datasets by giving them optical properties (color, opacity) using a transfer function. Adding lighting to direct volume rendering allows to bring additional and important visual cues for data analysis. Indeed, lighting plays a fundamental role in the way we perceive our environment, because it helps evaluating the relative positions and shapes of objects surrounding us.

In this thesis, we introduce different techniques improving the integration and the use of lighting methods in direct volume rendering. We first propose a non-linear interpolation method for the gradient. It allows to numerically improve the integration of gradient based lighting methods in the context of direct volume rendering. We then put the focus on a specific lighting method: Ambient Occlusion. By introducing a feature separation in the pre-computation step, we improve the perception of internal features by improving picture contrast. In order to reduce the pre-computation times induced by this method, we parallelize it on a multi-GPU architecture while integrating load balancing mechanisms and relying on two data repartition strategies: static and dynamic.

Table des matières

Liste des algorithmes

Table des figures

Liste des tableaux

Chapitre 1

Introduction

De nos jours, le recours à l'informatique est indispensable dans de nombreux domaines scientifiques. L'informatique permet d'aider à la simulation de phénomènes de plus en plus complexes, ainsi qu'à l'interprétation des résultats de ces simulations. La visualisation est un moyen d'interprétation, qui a recours à une représentation graphique de l'information pour l'étudier et la transmettre.

En matière de visualisation et de représentation tridimensionnelle, la communauté scientifique utilise principalement la visualisation surfacique, qui consiste à représenter des objets 3D par leurs bords à l'aide de polygones. Dans les années 80 est apparu une autre forme de visualisation 3D, la visualisation volumique directe, qui se voulait mieux adaptée à certains types de phénomènes naturels volumétriques comme les nuages, la fumée, ... Cependant, elle a aussi été mise en avant pour satisfaire des besoins liés notamment à l'apparition de la tomodensitométrie et de l'imagerie à résonance magnétique (IRM) dans le domaine médical. La visualisation volumique directe est utilisée pour apporter différentes représentations sous forme d'images de ces objets et données 3D qui sont essentiellement des champs scalaires 3D. Depuis, l'imagerie à résonance magnétique a évolué avec l'introduction de l'IRM fonctionnelle, qui permet de mesurer les variations des propriétés du flux sanguin ajoutant une dimension temporelle complexifiant encore la visualisation de telles données.

Au delà de l'imagerie médicale, la visualisation de champs scalaires 3D s'est exportée à d'autres domaines scientifiques. Ainsi en géologie elle est utilisée pour visualiser des carottes sédimentaires. En physique, elle permet la visualisation de résultats de simulations numériques, comme par exemple les simulations de phénomènes naturels que sont les aurores boréales, la fusion nucléaire ou encore les nébuleuses, afin de mieux les comprendre et de valider les modèles numériques proposés. La visualisation volumique de champs scalaires 3D est devenue une discipline à part entière et fait partie du domaine plus vaste, qu'est celui de la visualisation scientifique, dont un aperçu plus complet peut être consulté dans les ouvrages de Johnson et Hansen [JH04] et Nielson et al. [NHM97].

Dans la suite de cette introduction, nous présentons les principes de la visualisation de champs scalaires volumiques en section 1.1, les problèmes que pose encore le rendu volumique direct à la section 1.2, et un résumé des contributions de cette thèse à la section 1.3.

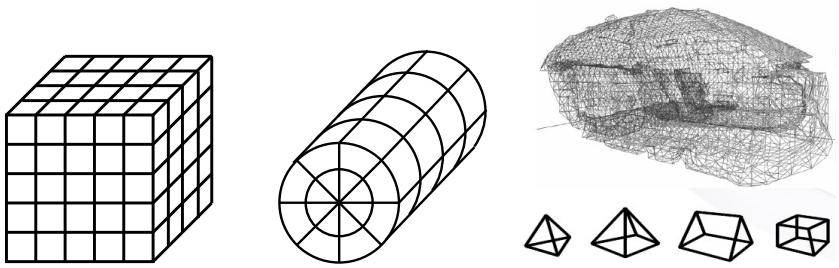


FIGURE 1.1 – A gauche et au milieu deux exemples de maillages structurés : cubique et toroïdal. A droite, un maillage non structuré formé de polyèdres adjacents.

1.1 Principes de la visualisation de champs scalaires 3D

Nous présentons ici une introduction aux données volumiques et aux méthodes qui permettent de visualiser de telles données. Parmi ces méthodes, on s'intéresse plus particulièrement au Rendu Volumique Direct (ou DVR de l'anglais "Direct Volume Rendering") et à l'amélioration de la précision numérique de cette méthode.

1.1.1 Champs volumiques

De façon générale, on peut voir un jeu de données scientifiques comme une fonction associant à une position dans l'espace un ensemble d'informations s :

$$\begin{aligned} f : (x, y, z) &\mapsto \{s_i\}^n = s \\ x, y, z &\in \mathbb{R} \\ s_i &\in \mathbb{R} \end{aligned}$$

En pratique, les données volumiques consistent en l'association d'une (ou plusieurs) information(s) à chaque élément d'un maillage tridimensionnel. Ces données peuvent être des scalaires, dont la nature varie selon le domaine d'application : température, pression, densité ... Il peut aussi s'agir de données plus complexes : vectorielles ou tensorielles : vitesse, déformation ...

Le maillage porteur de l'information est limité à un nombre fini de points. Par exemple, les informations issues d'IRM sont capturées par des capteurs à résolution finie, de même pour les simulations physiques/astronomiques, pour lesquelles on se restreint à utiliser un maillage plus ou moins précis, regroupant les informations de plusieurs éléments de base pour des raisons de temps de calcul et de capacités de stockage.

Le maillage porteur des informations peut être de deux types : structuré ou non structuré. La notion de structure est liée à la connexité du maillage, i.e. la relation d'adjacence entre les cellules du maillage. Si cette relation est implicite, les données sont dites "structurées". Si la connexité est définie explicitement, les données sont dites non structurées (cf. Figure 1.1).

Les maillages étant par nature discrets, il est nécessaire de reconstruire l'information en tout point pour pouvoir définir des champs de l'espace 3D et ainsi les visualiser. L'idéal est de compléter les données en utilisant un filtre de reconstruction. Comme la reconstruction peut se révéler très coûteuse, on utilise en visualisation plutôt des filtres simples comme les filtres en boîte ou en tente, qui ne considèrent

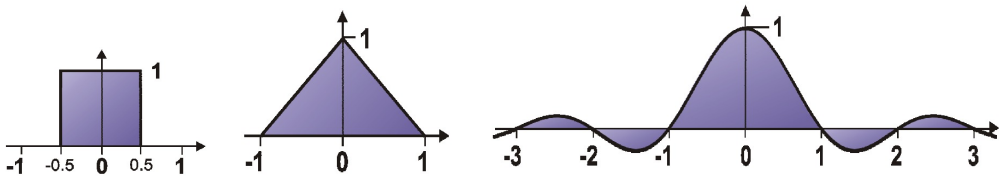


FIGURE 1.2 – De gauche à droite, différents filtres de reconstruction (boîte, tente, sinus cardinal) [EHK+03].

qu'un voisinage restreint autour de l'échantillon (cf. Figure 1.2). De plus, ces deux types de filtres sont intégrés nativement dans les cartes graphiques quand on utilise les structures spécifiques que sont les textures. On peut ainsi profiter d'une reconstruction matériellement optimisée.

HADWIGER ET AL [HTHG01], ainsi que SIGG ET HADWIGER [SH05] ont étudié des reconstructions de meilleure qualité en utilisant respectivement l'application de plusieurs textures et la programmabilité des accélérateurs graphiques récents pour réaliser des filtres à réponse impulsionnelle finie arbitraire. Ces techniques sont cependant beaucoup plus coûteuses.

Dans la suite, nous travaillons essentiellement sur des données scalaires définies sur un maillage structuré de forme parallélépipèdique. Les domaines produisant de telles données sont très variés : médecine, géologie, physique, archéologie, etc. Le filtrage n'a pas été une problématique étudiée dans le cadre de cette thèse. Nous nous sommes basés sur le filtrage linéaire matériellement optimisé des cartes graphiques.

1.1.2 Visualisation de données volumiques scalaires

La visualisation de données volumiques scalaires (un champ scalaire) peut se faire au moyen de trois principales méthodes :

– **Tranches 2D**

Cette méthode consiste à extraire une ou plusieurs tranches 2D des données à visualiser (cf. Figure 1.3 gauche). On peut extraire de telles tranches en les choisissant alignées sur l'objet, c'est-à-dire qu'on visualise les données en prenant des plans orthogonaux à l'un des 3 axes x, y, ou z. On peut aussi visualiser les données en choisissant des plans de coupe quelconques (non alignés sur l'objet).

Ce type de visualisation peut être illustré avec l'exemple du médecin visionnant les différentes coupes issues d'un appareil d'imagerie à résonance magnétique (IRM). Les tranches sont dans ce cas alignées sur l'objet.

Un inconvénient de cette technique est que l'on ne dispose que d'un point de vue local sur les données, car on extrait seulement une sous-partie sous la forme d'un plan 2D. Avec ce type de visualisation, la forme 3D des objets est plus difficile à percevoir.

– **Isosurface**

Cette méthode consiste à extraire une isosurface, c'est-à-dire l'ensemble des points du champ correspondant à une valeur scalaire spécifique (cf. Figure 1.3 milieu). Cet ensemble de points est en fait approché par une surface polygonale. L'algorithme des "marching cubes" de LORENSEN ET CLINE [LC87] est un exemple d'algorithme permettant d'extraire des isosurfaces d'un champ scalaire 3D.

Par rapport à la visualisation par tranches 2D, cette technique permet une meilleure perception des formes 3D. L'inconvénient est que cette technique ne permet de visualiser qu'une partie des données (un sous-ensemble).

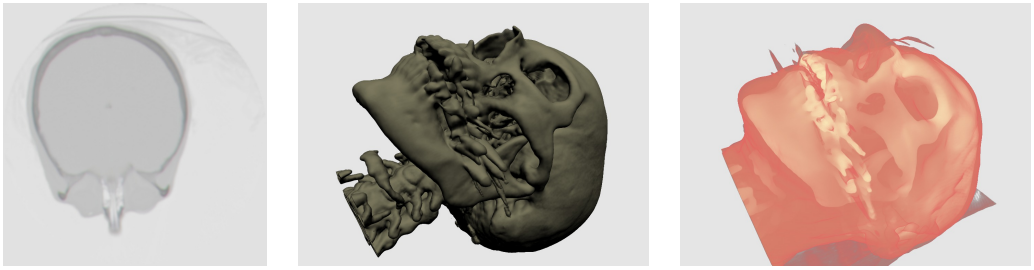


Figure 1.3: Les 3 types de visualisation de données volumiques: tranches 2D (à gauche), isosurface (au milieu) et rendu volumique direct (à droite).

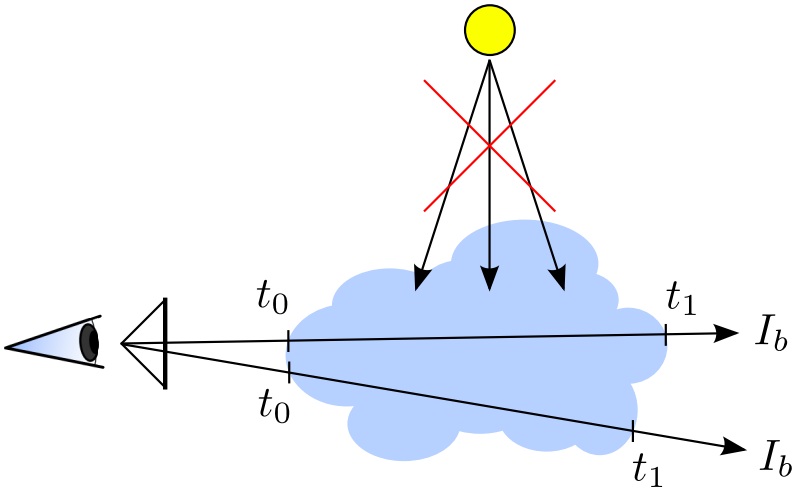


FIGURE 1.4 – On ne considère ici que les contributions issues de l'émission propre du volume, et non les potentielles contributions liées à des sources extérieures.

– **Rendu volumique direct**

Contrairement aux deux méthodes précédentes, le rendu volumique direct consiste à produire une image en considérant l'intégralité du volume de données, i.e. l'ensemble des données contribue à l'image (cf. Figure 1.3 droite). La section suivante décrit plus précisément cette technique dont cette thèse fait l'objet.

1.1.3 Rendu volumique direct (DVR)

La méthode de rendu volumique direct (en anglais DVR, pour "direct volume rendering") consiste à considérer le volume de données comme un corps participatif, c'est-à-dire un corps constitué d'un ensemble de particules qui absorbent, émettent et diffusent la lumière en son sein. Dans les modèles suivants, on ne considère que l'émission propre de ce corps et non une illumination issue de sources extérieures, comme illustré en figure 1.4.

Pour établir une projection d'un volume de données sur un plan image, plusieurs modèles d'interactions lumineuses à l'intérieur du milieu ont été exposés par MAX [Max95] :

– Absorption uniquement : on considère une intensité lumineuse initiale I_b provenant de l'arrière plan. Les particules contenues dans le volume absorbent une partie de cette lumière avant d'atteindre le point de vue. I_b est donc essentiellement atténuée par semi-transparence.

- Émission uniquement : on considère que les particules du volume sont elles-mêmes émettrices de lumière. Chaque particule va ainsi contribuer à la lumière que sera accumulée le long des rayons traversant le volume et perçue par l'observateur.
- Émission plus Absorption : Les deux effets précédents sont combinés. On a donc une lumière provenant d'un arrière plan, partiellement absorbée, ainsi que l'émission propre des particules du volume également partiellement absorbée. C'est le modèle communément utilisé en rendu volumique direct. Il est suffisamment simple pour être implémenté facilement.

Ces modèles classiques peuvent être étendus à des modèles plus conformes à la réalité physique en matière d'illumination d'un corps participatif. On peut ainsi considérer des phénomènes de diffusion, dans lesquels la lumière émise par les particules se propage sur les particules voisines selon une certaine fonction de phase et pas uniquement le long du rayon provenant du point de vue. L'ajout de sources lumineuses externes fera l'objet d'une description plus avancée dans le chapitre 3.

1.1.4 Modèle Émission plus Absorption

En utilisant le modèle optique de l'émission plus absorption, le plus largement utilisé en DVR, la perception qu'un observateur a de ce milieu est obtenue par l'ensemble des rayons lumineux qui traversent le volume. Chaque rayon traverse le volume qui absorbe une partie de la lumière et ajoute sa propre contribution sous forme de couleur.

L'expression décrivant la progression de l'intensité lumineuse portée par un rayon est appelée l'intégrale du rendu volumique. Pour cela, on définit la variation d'intensité lumineuse le long d'un rayon :

$$dI = I_{emise} - I_{absorbee} \tag{1.1}$$

De là, on déduit une équation différentielle, qui développée s'écrit alors sous la forme suivante (l'obtention et le développement sont décrit par *Max* [Max95]) :

$$I = I_b \cdot e^{-\int_{t_0}^{t_1} \tau(t)dt} + \int_{t_0}^{t_1} c(t) \cdot \tau(t) \cdot e^{-\int_{t_0}^{t} \tau(u)du} dt \tag{1.2}$$

La formule 1.2 s'appelle **l'intégrale du rendu volumique**. On considère un rayon de lumière traversant le volume du point d'entrée t_0 au point de sortie t_1. I correspond à l'intensité lumineuse atteignant l'observateur et I_b à l'intensité initiale provenant de l'arrière plan. Les termes $\tau(t)$ et $c(t)$ décrivent respectivement la profondeur optique et la couleur émise au point t. Le premier terme de cette équation décrit l'atténuation de la lumière provenant de l'arrière plan I_b après la traversée du volume. Le second terme décrit, quant à lui, l'ensemble de la lumière émise en chaque point sur le rayon en tenant compte du facteur d'atténuation, autrement dit la composition de l'ensemble des couleurs associées à chaque scalaire s sur le chemin du rayon.

Pour calculer cette intégrale, on recourt à une approximation numérique. On se base sur une somme de Riemann qui consiste à passer de $\int h(x)dx$ à $\sum_{i=1}^{n} h(x_i)\Delta x$ en découpant l'intervalle de t_0 à t_1 en n intervalles t_i.

En utilisant ce principe, on obtient l'expression suivante :

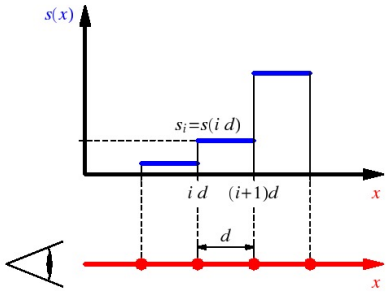


FIGURE 1.5 – Approximation de l'intégrale par la somme de Riemann [SKLE03].

$$I \approx I_b \cdot e^{-\sum_{i=1}^{n} \tau(t_i)\Delta t} + \sum_{i=1}^{n} c(t_i) \cdot \tau(t_i) \cdot e^{-\sum_{j=1}^{s} \tau(t_j)\Delta j} \Delta t \tag{1.3}$$

En remarquant que $e^{a+b} = e^a \cdot e^b$ et en approximant l'opacité du k^e segment de rayon par un facteur α_k comme suit $\alpha_k \approx 1 - e^{-\tau(t_k)\Delta t}$, on obtient :

$$I \approx I_b \cdot \prod_{i=1}^{n}(1-\alpha_i) + \sum_{i=1}^{n} c(t_i) \cdot \tau(t_i) \cdot \Delta t \cdot \prod_{j=1}^{s-1}(1-\alpha_j) \tag{1.4}$$

Dans la section suivante, nous présentons la manière algorithmique de calculer I.

1.1.5 Composition

En considérant la couleur $c(t_i)$ émise au segment t_i comme déjà pré-multipliée par l'atténuation $\tau(t_i)\Delta t$, on peut simplifier l'expression précédente, et ainsi obtenir l'expression de composition utilisant l'opérateur "over" (ou "au-dessus") défini par PORTER ET DUFF en 1984 [PD84] :

$$\begin{aligned} I &\approx I_b \cdot (1-\alpha_1) \cdot \ldots \cdot (1-\alpha_n) \\ &+ c_1 + c_2 \cdot (1-\alpha_1) + \ldots + c_n \cdot (1-\alpha_1) \cdot \ldots \cdot (1-\alpha_n) \\ &\approx I_b \cdot (1-\alpha_1) \cdot \ldots \cdot (1-\alpha_n) \\ &+ c_1 \, over \, c_2 \, over \, \ldots \, over \, c_n \end{aligned} \tag{1.5}$$

avec $c_i = c(t_i) \cdot \tau(t_i) \cdot \Delta t$.

L'équation (1.4) revient donc à composer des échantillons avec l'opérateur "over" et ceci peut être réalisé de deux manières différentes : d'avant en arrière ou d'arrière en avant.

La méthode "back-to-front" (d'arrière en avant) consiste à parcourir les données d'arrière-en-avant par rapport à l'observateur et à accumuler les contributions pour définir I sous la forme d'une suite **inversée** telle que :

$$I \approx I_0$$

avec :

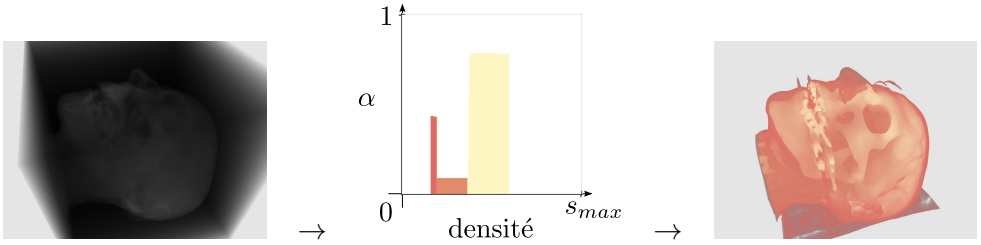


FIGURE 1.6 – Le rendu volumique des densités (à gauche) combiné à une fonction de transfert (au centre) nous donne le rendu volumique direct (à droite).

$$\begin{cases} I_n &= I_b \\ I_{k-1} &= (1-\alpha_k)\cdot I_k + c_k \end{cases}$$

Cette technique requiert de parcourir l'intégralité des n échantillons en partant du dernier (le n^{e}) jusqu'au premier. Il est nécessaire de parcourir l'intégralité des échantillons car le dernier (c_1, α_1) peut être totalement occultant et annuler toutes les contributions accumulées précédemment.

La méthode "front-to-back" (d'avant en arrière) permet aussi l'évaluation de la formule 1.4, mais en parcourant les données d'avant en arrière par rapport à l'observateur. En parcourant les échantillons sous la forme de la suite suivante :

$$I \approx I_n + (1 - A_n)\cdot I_b$$

avec :

$$\begin{cases} I_k &= I_{k-1} + (1-A_{k-1})\cdot c_k \\ A_k &= A_{k-1} + (1-A_{k-1})\cdot \alpha_k \\ I_0 &= 0 \\ A_0 &= 0 \end{cases}$$

Cette seconde méthode requiert la mémorisation d'une opacité accumulée A_k le long du rayon. Elle a l'avantage majeur de permettre d'arrêter le parcours dès que l'opacité maximale est atteinte, c'est-à-dire $A_k = 1$ (à ce stade plus aucun échantillon ne pourra contribuer à la valeur finale). En effet, dès que A_k vaut 1, A_{k+1} reste à 1 et I_{k+1} reste égal à I_k.

1.1.6 Fonction de transfert

L'attribution de propriétés optiques c et α aux valeurs scalaires d'un volume de données est une étape très importante. Elle permet de décider de la visibilité et de la couleur affectées aux différentes plages de valeurs. Cette étape s'appelle aussi l'étape de **classification**.

La fonction de transfert prend classiquement la forme d'une table unidimensionnelle associant à chaque valeur discrète de densité un quadruplet (r, g, b, α), où r, g et b forment respectivement la proportion des couleurs rouge, verte et bleue de c, et α désigne la transparence (cf. Figure 1.6). Notons que cet outil permet aussi d'extraire des isosurfaces sans avoir à générer de géométrie en associant une opacité totale à l'iso-densité que l'on souhaite visualiser.

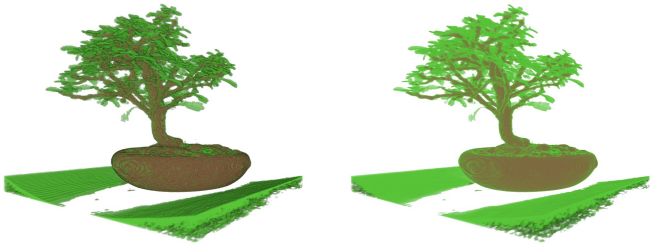


FIGURE 1.7 – Exemple de pré-classification (à gauche) et post-classification (à droite).

De nombreuses propositions ont été faites pour enrichir cet outil, dont le principe de base est simple. Généralement, les fonctions de transfert sont définies empiriquement par l'utilisateur, ce qui peut représenter un travail fastidieux. HE ET AL. [HHKP96] proposent une technique permettant de générer automatiquement des fonctions de transfert en utilisant une méthode d'optimisation stochastique. Le choix peut être guidé par l'utilisateur avec une étape intermédiaire proposant des images produites avec les fonctions générées. KINDLMANN ET DURKIN [KD98] proposent une méthode de génération semi-automatique de fonctions de transfert en visant plus spécifiquement les jeux de données pour lesquels les régions d'intérêt sont celles situées à l'interface entre deux matériaux. KNISS ET AL. [KKH02] proposent d'utiliser la norme du gradient comme dimension supplémentaire dans la spécification des fonctions de transfert. Cela leur permet ainsi de discerner des zones non seulement sur la base de leur densité, mais par rapport aux variations de densité. CORREA ET MA [CM09a] se basent sur une méthode de calcul de facteurs d'occultation et les utilisent pour ajouter une dimension supplémentaire à la fonction de transfert. Ceci leur permet de définir des paramètres optiques en fonction de l'occultation associée à une certaine densité. CORREA ET MA [CM09b] introduisent le concept d'histogramme de visibilité. Il permet d'évaluer la visibilité des différentes plages de densités pour un point de vue et une fonction de transfert donnés.

En ce qui concerne l'application d'une fonction de transfert, nous avons vu dans la section 1.1.1 qu'il était nécessaire de reconstruire l'information définie par le maillage pour obtenir un champ continu. L'introduction de la fonction de transfert pose la question de savoir où placer cette reconstruction. L'ordre d'application donne ainsi lieu à deux classifications différentes (illustrées en Figure 1.7), car ces deux opérations ne commutent pas (à moins que la fonction de transfert ne soit une fonction constante ou l'identité) :

– la pré-classification interpole les propriétés optiques :

$$s \rightarrow (r, g, b, \alpha) \rightarrow interpolation$$

– la post-classification interpole les données scalaires :

$$s \rightarrow interpolation \rightarrow (r, g, b, \alpha)$$

On constate empiriquement que le rendu final lié à la pré-classification est moins précis que celui issu d'une post-classification, on préférera donc généralement cette seconde méthode. MAX [Max95] indique que la différence entre pré-classification et post-classification est analogue à la différence entre l'ombrage de Gouraud (interpolation des couleurs ombragées) et l'ombrage de Phong (interpolation des normales) pour le calcul de l'éclairage local d'une surface polygonale.

D'un point de vue pratique, la pré-classification requiert de pré-classifier les données avant d'en effectuer le rendu. Ceci a pour effet d'augmenter la consommation mémoire, car au lieu de stocker le tableau des densités sur la carte graphique, on stocke un tableau dans lequel on substitue les densités par le quadruplet (r, g, b, α) correspondant. Quatre fois plus de mémoire est donc utilisée dans ce cas si les densités et les propriétés optiques sont stockées dans le même format. La post-classification requiert seulement le tableau des densités et la fonction de transfert, mais nécessite un accès texture supplémentaire pour chaque échantillon, afin d'obtenir le quadruplet (r, g, b, α) correspondant. Ce qui est plus coûteux en temps de calcul.

1.2 Principaux verrous dans le rendu volumique

On peut identifier différents verrous concernant le rendu volumique direct. Ci-dessous nous présentons trois principaux verrous sans être exhaustif :

Fonction de transfert

Un premier verrou est la création et définition de fonctions de transfert. Ce point est critique, car il permet de décider quelles structures sont mises en valeur lors de la visualisation. La création d'une fonction de transfert est souvent un travail délicat, car des zones que l'on voudrait différencier peuvent avoir la même densité et donc correspondre à une même couleur et une même opacité, d'où l'impossibilité de cacher l'une sans cacher l'autre. Pour aider à la résolution de ce problème, on peut segmenter les données au préalable ou utiliser, par exemple, des fonctions de transfert multi-dimensionnelles. Le problème de la détermination de la fonction de transfert n'est pas abordé dans cette thèse. Nous les avons définies empiriquement dans la suite de ce travail.

Éclairage

Le choix du type d'éclairage et de ses paramètres est un autre point critique qui permet d'amener une meilleure compréhension des données lors de la visualisation. La création de modèles d'éclairage plus proche de la physique de la lumière permet ainsi d'obtenir un rendu photoréaliste des données. Ou au contraire en s'éloignant du réalisme, on peut modifier l'éclairage afin qu'il mette en valeur certaines caractéristiques des données. Dans ce dernier cas, on parle plutôt de rendu expressif ou non-photoréaliste. La problématique de l'éclairage est abordée dans cette thèse et fait l'objet des chapitres 5 et 6 de la partie contributions.

Performance

Un autre verrou concernant le rendu volumique est le problème de la performance. En particulier, l'utilisation de méthodes d'éclairage sophistiquées peut altérer la performance de la visualisation. De même, l'utilisation de grands volumes de données requiert des techniques spécifiques de visualisation, du fait des limitations liées à la capacité mémoire des cartes graphiques d'aujourd'hui. Améliorer les performances consiste donc à améliorer les algorithmes existants ou à utiliser le parallélisme de plus en plus présent dans les machines actuelles. L'usage du parallélisme fait l'objet des travaux présentés dans le chapitre 7.

1.3 Résumé des contributions

Les contributions réalisées dans ce travail de thèse se situent sur deux axes : l'amélioration de la qualité visuelle et l'augmentation des performances de la visualisation volumique.

1.3.1 Amélioration de la qualité visuelle

Deux méthodes ont été développées pour améliorer la qualité visuelle.

Amélioration de l'éclairage de Phong pour rendu volumique direct pré-intégré avec interpolation non-linéaire du gradient

En rendu volumique, le sous-échantillonnage, lié à la contrainte de performance temps-réel, se manifeste sous la forme de discontinuités visibles sur les surfaces, car on ne capture pas l'information qui se trouve entre deux échantillons consécutifs. Ces artefacts sont grandement réduits en utilisant une technique de pré-intégration qui suppose une variation linéaire de couleur et d'opacité entre paires d'échantillons, plutôt qu'en considérant chaque échantillon séparément. L'ajout d'éclairage avec cette technique est approché soit par le calcul de la moyenne des valeurs d'éclairage en chaque échantillon, soit par l'utilisation d'une variation linéaire des gradients entre les deux échantillons considérés. Ces deux approches induisent des erreurs dans le calcul de l'éclairage. Pour réduire ces erreurs, nous proposons une technique qui s'appuie sur une variation non-linéaire du gradient. Cette contribution est présentée au chapitre 5.

Elle a fait l'objet d'une collaboration avec Amel Guetat et Stéphane Marchesin, et a été publiée dans le cadre de la conférence *IEEE Vis* en 2010 [GAMD10].

Introduction d'une méthode d'Ambient Occlusion par structure

Les données volumiques brutes forment un ensemble de couches de densités différentes et les propriétés optiques de chacune sont définies par l'utilisation de la fonction de transfert. Lorsqu'on utilise la technique de l'Ambient Occlusion (ou occultation ambiante en français) pour l'éclairage de données volumiques, le choix de la fonction de transfert fera que les couches visibles s'obscurciront plus ou moins par rapport à leurs positions relatives. Ainsi, la présence de multiples couches fera qu'elles s'entre-assombriront, ce qui résultera en une perception décroissante pour les différentes couches de la plus externe vers la plus interne : ce type de phénomène est par exemple visible si on souhaite visualiser une image obtenue par IRM et que l'on associe une couleur pour la peau, l'os et le cerveau. Dans ce cas, les structures internes (le cerveau) seront assombries par la participation des couches extérieures que sont l'os et la peau.

Dans cette thèse, nous proposons une méthode d'Ambient Occlusion par structure qui aide à distinguer ces différentes structures en ne considérant que les phénomènes d'occultation propre, et donc en ignorant les interactions inter-structures dans le calcul de l'Ambient Occlusion. Pour cela, nous découpons la fonction de transfert en sous-fonctions représentant des structures significatives et calculons des facteurs d'Ambient Occlusion pour chacune de ces sous-fonctions. Une dernière phase réassemble les résultats obtenus pour chaque structure en un volume final qui constitue le volume d'occultation. Cette méthode réduit significativement l'assombrissement

résultant du calcul de l'Ambient Occlusion. Cependant, elle a un coût supplémentaire dû au fait que l'on doit considérer et calculer plusieurs volumes d'Ambient Occlusion. Cette seconde contribution sera présentée au chapitre 6.

Elle a fait l'objet d'une publication dans le cadre de la conférence *Volume Graphics* de 2010 [ADM10].

1.3.2 Performance : Optimisation et parallélisation

La deuxième partie de cette thèse porte sur l'amélioration de la performance de la méthode de calcul de l'Ambient Occlusion, et est présentée au chapitre 7. Le calcul de l'Ambient Occlusion nécessite le parcours de l'ensemble du volume, et pour chaque voxel, le lancement d'un ensemble de rayons. Le nombre de rayons lancés doit être suffisant pour échantillonner correctement le voisinage. Cette phase est coûteuse en temps (de l'ordre de quelques secondes), et elle est d'autant plus coûteuse avec l'Ambient Occlusion classifié, car ces calculs doivent se faire pour plusieurs structures. Ces temps de calcul se révèlent donc être pénalisants si l'on souhaite obtenir une visualisation en temps réel. En effet, pour explorer les données volumiques, un utilisateur est amené à manipuler la fonction de transfert, or les valeurs d'Ambient Occlusion dépendent de cette fonction. Il est donc nécessaire de recalculer ces valeurs à chaque changement de fonction de transfert. Pour améliorer les performances, deux approches ont été étudiées : l'élimination de zones vides et la parallélisation basées sur plusieurs cartes graphiques.

Élimination des zones vides

Cette méthode calcule des valeurs uniquement dans les sous-volumes que l'on considère utiles, c'est-à-dire les parties du volume qui seront réellement affichées à l'écran. Pour cela, nous partons du principe que toutes les valeurs de la plage de densité ne sont pas "affichées". En effet, de nombreuses plages sont rendues transparentes par le choix de la fonction de transfert, lorsqu'elles contiennent du bruit ou du contenu non important pour l'analyse de l'image finale. Pour déterminer les zones vides, on utilise une technique de "bricking" (ou regroupement en briques) en découpant le jeu de données initial en briques de taille inférieure.

Pour déterminer si une brique est vide, chaque voxel de la brique est analysé pour évaluer s'il est ou n'est pas transparent. Si tous les voxels sont transparents, la brique est vide et aucun calcul d'Ambient Occlusion n'est nécessaire pour cette brique. Le temps de calcul de l'Ambient Occlusion est donc réduit aux seules briques non vides. Pour préserver la qualité du rendu et éviter les discontinuités d'ombrage au bord des structures, le découpage en briques intègre une zone de recouvrement (overlap). Nous avons évalué différentes tailles de briques, afin de sélectionner celle qui avait le meilleur impact sur la réduction du temps de calcul tout en restant peu coûteuse en temps de calcul.

Parallélisation multi-GPU

La seconde approche sur laquelle nous avons travaillé est l'utilisation de machines multi-GPU [1] pour réduire encore le temps de calcul. Répartir les calculs sur différentes cartes graphiques permet de réduire l'empreinte mémoire sur chacune, ainsi que le temps de calcul global, mais nécessite la mise en place d'une gestion plus avancée du parallélisme CPU/GPU. Pour cela, nous nous sommes basés sur l'utilisation

1. GPU signifie "Graphics Processing Unit".

de l'environnement de travail sur GPU de NVIDIA : CUDA, ainsi qu'OpenMP, qui permet la parallélisation côté CPU. En combinant ces approches, on peut ainsi utiliser plusieurs cartes graphiques présentes dans un même ordinateur, afin de les faire calculer parallèlement.

Nous avons développé une technique de parallélisation de l'Ambient Occlusion, qui intègre la gestion de l'équilibrage de charge en s'appuyant sur deux approches de répartition : statique et dynamique. Dans la méthode statique, les données à calculer sont découpées en autant de blocs qu'il y a de GPUs, tandis que dans l'approche dynamique, les données sont découpées en sous-blocs de taille identique qui sont distribués à la volée aux différents GPUs.

Nous avons expérimenté ces deux techniques de parallélisation sur une machine équipée de 4 GPUs, en produisant des résultats à partir d'une série de jeux de données.

1.4 Plan de thèse

La thèse est structurée en trois grandes parties : un état de l'art, les contributions de la thèse et une conclusion présentant également des perspectives à ce travail.

La première partie présente un état de l'art en trois chapitres. Nous présentons tout d'abord différentes méthodes de visualisation en rendu volumique direct au chapitre 2. Le chapitre 3 introduit ensuite des travaux sur différentes méthodes d'éclairage utilisées en rendu volumique direct, dont l'Ambient Occlusion en section 3.2. Puis nous concluons cet état de l'art par un aperçu de méthodes utilisant le parallélisme et notamment les approches multi-GPU au chapitre 4.

La seconde partie présente les contributions de cette thèse. Elle est structurée selon les différents thèmes présentés en section 1.3.

La troisième partie clôt la thèse par une conclusion et un ensemble de perspectives de continuation et d'amélioration de ces travaux.

Première partie

État de l'art

Chapitre 2

Rendu volumique direct en temps réel

Historiquement, le rendu volumique direct a été introduit avec les travaux sur la simulation des nuages de Blinn [Bli82] et Kajiya et Von Herzen [KH84]. Sabella [Sab88] a poursuivi ces travaux en utilisant de la couleur pour mettre en exergue certaines propriétés dans des champs scalaires 3D. Tous ces travaux ont été implémentés en utilisant des processeurs centraux (CPU). L'arrivée des cartes graphiques et de leurs processeurs de traitement, les GPUs, a permis la mise en place de nouvelles techniques pour visualiser les champs scalaires 3D en temps réel. Nous présentons ici les deux principales méthodes de rendu volumique direct : la méthode par géométrie de proximité en section 2.1 et la méthode se fondant sur un lancer de rayon sur carte graphique en section 2.2.

2.1 Méthode par géométrie de proximité

Cette méthode consiste à exploiter la notion de textures. Une texture peut être vue comme un tableau 1D, 2D ou 3D de valeurs. Implémentés matériellement par les cartes graphiques, ces tableaux sont optimisés pour certaines opérations spécifiques telles que les interpolations linéaires dans chaque dimension. A la base, les textures ont été créées pour "habiller" les polygones en visualisation surfacique, en apposant des morceaux d'image sur les polygones.

La première étape dans la visualisation volumique basée sur une géométrie de proximité consiste à définir une géométrie de support ("proxy geometry" en anglais). Elle est construite à partir de l'intersection entre une suite de plans parallèles, et le parallélépipède correspondant aux bords des données volumiques. La texture 3D contenant le champ de données scalaires est alors appliquée sur cette géométrie de support. Les textures étant définies comme des tableaux, il faut donc que le maillage portant le champ scalaire soit une grille régulière de voxels. La suite de plans est en principe définie parallèle au plan image (cf. Figure 2.1).

Les travaux de Cabral et al. [CCF94] et de Cullip et Neumann [CN94] sont les premiers à utiliser les textures 3D et la composition alpha (en anglais, "alpha-blending") pour le rendu volumique direct, exhibant par la même occasion une augmentation drastique des performances de rendu par rapport à une approche CPU.

L'utilisation de plans parallèles à l'espace image introduit une variation du pas d'échantillonnage par rapport à une projection perspective (cf. Figure 2.2). Ceci peut

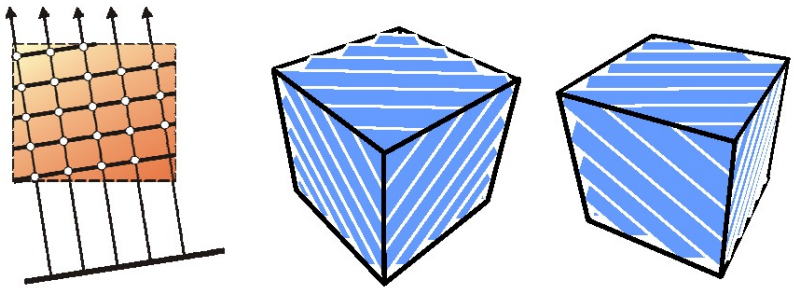


FIGURE 2.1 – Géométrie de support alignée sur le plan image [EHK+03].

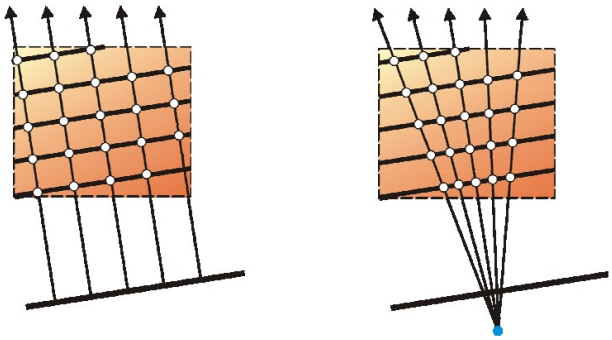


FIGURE 2.2 – Différences entre projection orthographique et perspective [EHK+03].

conduire à des artefacts. Pour palier ce défaut, LAMAR ET AL. [LHJ99] introduisent les coupes sphériques, afin de minimiser ces artefacts et assurer un échantillonnage régulier (cf. Figure 2.3).

Plus récemment et partant de l'idée que toutes les cartes graphiques ne disposent pas encore du support des textures 3D, par exemple dans le cas des cartes graphiques intégrées (ordinateurs et téléphones portables), KRÜGER [Krü10] propose d'utiliser les textures 2D et de générer une géométrie hybride utilisant, en fonction de la position de l'observateur, un jeu de textures 2D alignés sur l'un des 3 axes de coordonnées.

L'approche basée sur les textures 3D et une géométrie de proximité planaire est encore aujourd'hui utilisée, car elle offre de meilleures performances sur de nombreuses cartes graphiques que la technique du lancer de rayons, plus moderne, présentée dans la section suivante. Cependant, elle est moins précise.

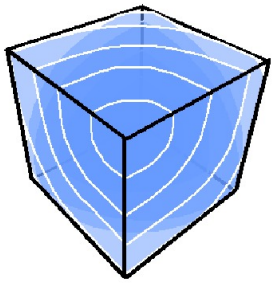


FIGURE 2.3 – Géométrie de support de type "coupes sphériques" [Mar07].

2.2 Lancer de rayons sur GPU

Depuis le début des années 2000, la programmabilité des cartes graphiques évolue constamment : tout d'abord avec les "register combiners"[1], puis les "vertex/fragment shaders"[2] et des langages de plus haut niveau tels que HLSL/GLSL, et finalement la généralisation plus récente avec le GPGPU et CUDA/Stream ou OpenCL. Cette évolution, que nous développons dans la section 4.2 du chapitre 4, permet d'implanter un lancer de rayons directement sur carte graphique.

Carr et al. [CHH02] sont parmi les premiers à vouloir exploiter les "fragment shaders" pour effectuer un lancer de rayons. Pour cela, ils déportent le calcul des intersections rayons-triangles sur le GPU, en laissant le reste du processus de lancer de rayons sur le CPU.

Purcell et al. [PBMH02] se sont intéressés à l'implémentation du lancer de rayons sur GPU. En prévoyant l'évolution des cartes graphiques et en proposant des solutions pour différents modèles : multi-passes (effectuer plusieurs rendus et les combiner) ou à branchements dynamiques (introduction d'instructions de branchement dans les GPUs), ils ont proposé des modèles et réalisé des simulations de fonctionnement. Pour cela, ils se sont attachés à découper l'algorithme du lancer de rayons en noyaux pouvant travailler sur des flots d'informations : modèle parfaitement en adéquation avec le modèle de flot utilisé dans les GPUs programmables.

Dans le cadre d'une implémentation multi-passes, Krüger et Westermann [KW03] ont ensuite proposé une méthode de rendu volumique utilisant le lancer de rayons. Ils utilisent pour cela un minimum de 3 passes. Les deux premières consistent en le rendu des faces avant et des faces arrière d'un cube représentant le bord des données volumiques (cf. Figure 2.4), afin de récupérer le point d'entrée et la direction de chaque rayon. Puis de 1 à M passes sont effectuées avançant chacune le rayon d'un nombre fini de pas (Le nombre M dépend du nombre de pas nécessaires pour traverser l'ensemble du volume). Le résultat de chacune de ces passes est composé avec les passes précédentes dans une texture à deux dimensions. Des passes intermédiaires sont réalisées en rendant les faces avant, afin de tester l'opacité de la texture résultante pour chaque rayon, et ainsi arrêter les rayons dont l'opacité est maximale (fin de rayon prématurée ou "early ray termination"). La mise en place d'un octree permet l'utilisation d'une seconde technique : le saut d'espace vide (ou "empty space skipping"), qui consiste à éviter de consacrer des passes au rendu des espaces totalement transparents.

La même année, un article de Roettger et al. [RGW+03] propose une solution de lancer de rayon similaire à la solution précédente. Elle s'appuie aussi sur une technique d'arrêt des rayons et de saut d'espaces vides.

En 2004, la société NVIDIA [NVI04] propose une démonstration technologique implémentant en une passe un lancer de rayons pour la visualisation de données volumiques en utilisant le branchement dynamique des dernières générations de cartes graphiques.

Stegmaier et al. [SSKE05] ont, par la suite, proposé un framework plus complet pour le lancer de rayons sur GPU. Toujours en se basant sur les améliorations des GPUs, et notamment l'ajout de branchements dynamiques et de boucles, ils ont eux aussi pu produire une implémentation d'un lancer de rayons en une passe, comme l'avait prédit Purcell et al. [PBMH02]. Leur formalisation du lancer de rayons sur GPU est donnée dans l'algorithme 1. De plus, ils ont amélioré leur mé-

1. Première version des "fragment shaders".
2. Programmes exécutés sur des unités spécifiques des cartes graphiques.

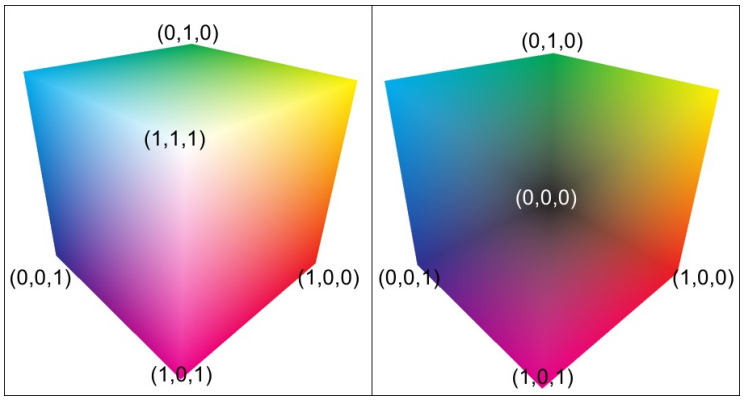


FIGURE 2.4 – Rendu des faces avant (à gauche) et arrière (à droite) [KW03].

thode avec des optimisations déjà présentes dans les implémentations multi-passes et ont décrit des méthodes de rendu plus photoréalistes se basant par exemple sur la réfraction ou déviation du rayon, qui n'est plus rectiligne.

Algorithme 2.1 Pseudo-code d'un fragment shader pour lancer de rayon GPU en une passe [SSKE05]

```
Calculer le point d'entrée dans le volume de données
Calculer le vecteur de direction du rayon
Tant que l'on est dans le volume
    Récupérer la densité à la position courante
    Récupérer (couleur, opacité) à partir _
        de la fonction de transfert et de la densité
    Accumuler couleur et opacité
    Avancer le long du rayon
```

Toujours la même année, KLEIN ET AL. [KSSE05] améliorent encore les performances du lancer de rayon en exploitant la cohérence spatio-temporelle entre les différentes images générées pour améliorer le saut des espaces vides ("empty-space skipping"). FANGERAU ET KRÖMKER [FK10] proposent une implémentation optimisée du lancer de rayon en CUDA.

L'optimisation de la définition des points d'entrée des rayons dans le volume a aussi fait l'objet de travaux. MENSMANN ET AL. [MRH08] génèrent par exemple une géométrie de proximité composée de polyèdres se rapprochant des données qui seront effectivement affichées. Ainsi, ils réduisent le temps de rendu de moitié. LIU ET AL. [LCD09] proposent d'utiliser une géométrie de support basée sur un ensemble de sphères. Ces sphères englobent les blocs de données qui sont effectivement affichés. Cela leur permet ainsi de réduire jusqu'à 15 fois le temps de rendu pour certains jeux de données. La Figure 2.5 illustre le changement de géométrie de proximité proposé par ces deux techniques.

Un autre thème de travaux dans le cadre du lancer de rayons est la visualisation de données de grande taille, typiquement des données dont la taille dépasse l'espace disponible sur une carte graphique. GOBETTI ET AL. [GMI08] et CRASSIN ET AL. [CNLE09] utilisent tous deux un octree permettant de ne charger que les sous-volumes effectivement affichés lors de la phase de rendu. Lors de cette phase, ils extraient des informations, notamment d'occultation, afin d'optimiser les blocs

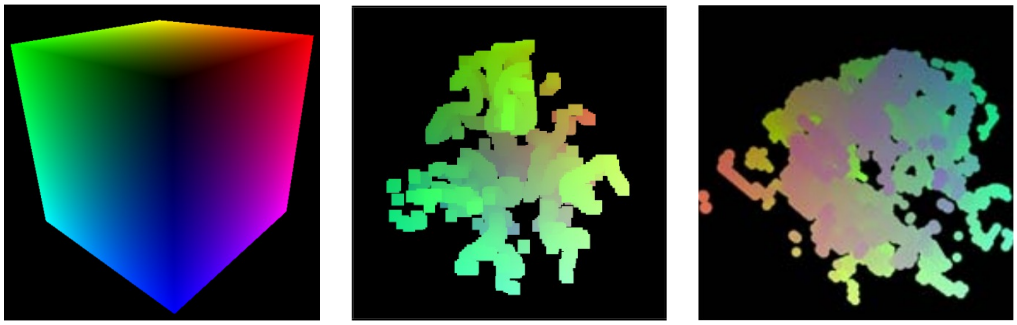


FIGURE 2.5 – Premiers points d'intersection avec le volume de données. De droite à gauche, en considérant la boite englobante des données, la technique utilisant des polyèdres de MENSMANN ET AL. [MRH08] et la technique basée sur les sphères de LIU ET AL. [LCD09], avec une sphère englobant 8^3 voxels.

effectivement chargés en mémoire.

Pour résumer, le lancer de rayon sur GPU se décompose en trois tâches :

- L'initialisation des rayons, qui définit les points d'entrée des rayons dans le volume, typiquement par calcul d'intersection,
- L'échantillonnage des rayons,
- L'intégration numérique des échantillons.

Les capacités de calcul parallèle des cartes graphiques sont ici utilisées pour les deux dernières étapes.

Cette méthode de rendu est aujourd'hui très répandue dans le domaine de la visualisation volumique, car elle est simple à mettre en place et permet d'échantillonner précisément les rayons lancés.

2.3 Autres méthodes de rendu volumique direct

D'autres techniques de rendu volumique existent. Nous donnons un bref aperçu des principales autres méthodes.

La projection de primitives, introduite par SHIRLEY ET TUCHMAN [ST90], consiste à exploiter une capacité spécifique des cartes graphiques, celle d'afficher des triangles. Cette méthode est principalement utilisée dans le cadre des maillages non structurés, car elle permet la visualisation des données non-structurées sans avoir à ré-échantillonner les données sous forme structurée pour stockage en textures 3D. Le principe de base consiste à décomposer le maillage tridimensionnel en tétraèdres (simplexes de dimension 3, i.e. on peut décomposer n'importe quel maillage tridimensionnel en tétraèdres). Les tétraèdres sont ensuite projetés sur l'espace écran sous la forme de triangles, formant leurs empreintes sur le plan de projection. Un inconvénient de cette méthode est qu'il est nécessaire de trier les tétraèdres pour obtenir un ordre et pouvoir réaliser la composition d'avant en arrière ou d'arrière en avant.

Le "splatting" (aplatissement) a aussi donné lieu à des implémentations en rendu volumique, comme dans les travaux de WESTOVER [Wes90] et ZWICKER ET AL. [ZPvBG02]. Chaque voxel est représenté par une "empreinte" (typiquement image 2D), puis ces empreintes sont projetées et composées sur l'espace écran.

LACROUTE ET LEVOY [LL94] proposent une technique qu'il nomme "Shear-warp". Dans cette technique, ils transforment l'espace des tranches du volume par

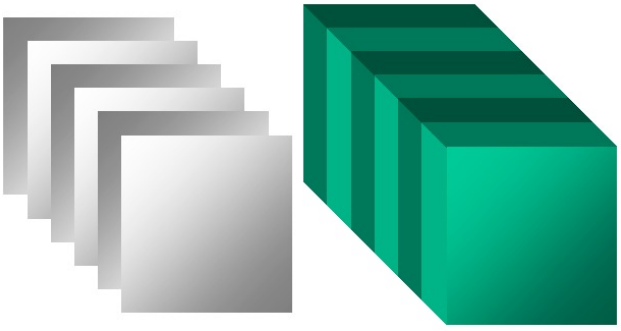


FIGURE 2.6 – Tranches (à gauche) et couches (à droite) [EHK+03].

translation, afin que les rayons issus du point de vue soient orthogonaux aux tranches des données (opération "Shear"). L'ensemble des tranches transformées est ensuite accumulé dans un buffer hors-écran. Ce buffer est finalement transformé, afin que les rayons issus du point de vue soient replacés à leur origine.

2.4 Amélioration de la précision numérique

L'amélioration de la précision numérique est un point crucial afin de permettre des rendus sans artefacts et sans augmenter de manière considérable les temps de calcul. La présence d'artefacts peut en effet mener à de mauvaises interprétations des données, du fait que l'on ajoute une information qui n'était pas initialement présente dans les données.

Une solution permettant d'augmenter la précision numérique de l'approximation de l'intégrale du rendu volumique est l'utilisation d'un grand nombre de tranches (principe du sur-échantillonnage). Un seuil théorique peut-être déterminé avec le théorème d'échantillonnage de Nyquist-Shannon. Ainsi l'approximation discrète de l'intégrale du rendu volumique doit converger vers le bon résultat si la distance entre les tranches tend vers 0. ROETTGER ET AL. [RGW+03] proposent par exemple de sur-échantillonner le lancer de rayon dans les zones de fortes variations scalaires.

Cependant on peut aussi partir du principe qu'au lieu de tranches de données, on considère des couches (cf. Figure 2.6) dans lesquelles on prend en compte l'émission et l'atténuation de la lumière en leur sein.

Proposée initialement par ROETTGER ET AL. [RKE00] pour la projection de tétraèdres en visualisation de données non structurées, ENGEL ET AL. [EKE01] ont ensuite appliqué ce même principe aux méthodes de rendu volumique utilisant un pas constant, telles que le lancer de rayons. Pour introduire la notion de couches, les échantillons ne sont plus pris séparément, mais par paires : (s_f, s_b), où s_f est la valeur scalaire dans la tranche commençant la couche, et s_b la valeur scalaire en sortie de couche (cf. Figure 2.7). Le principe de la pré-intégration consiste à séparer l'intégration numérique en deux intégrations : une première pour le champ de données scalaires et une seconde pour les valeurs de couleur et d'opacité.

L'opacité d'une couche k, d'épaisseur l, notée $\alpha(l)$, est ainsi approchée par :

$$\alpha_k(l) \approx 1 - e^{-\int_{t_k}^{t_{k+1}} \tau(t) l dt}$$

Par changement de variable et en notant s_f et s_b les valeurs scalaires de t_k et t_{k+1}, et en supposant une variation linéaire de la valeur scalaire, on obtient :

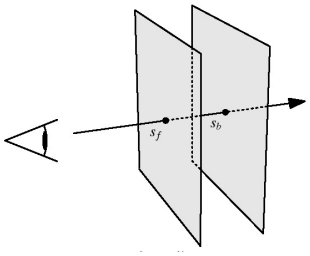
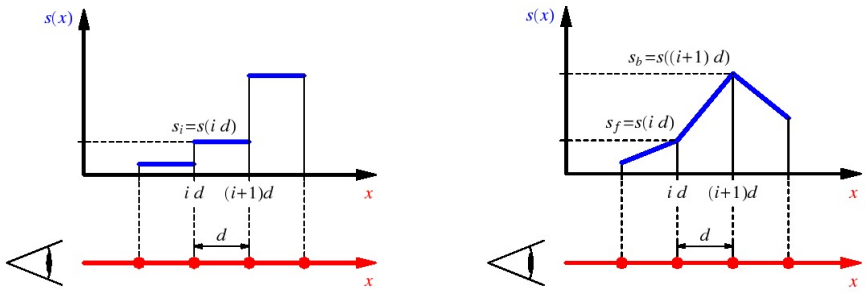


FIGURE 2.7 – Point d'entrée et de sortie d'une couche [EKE01].



a) Intégration par somme de Riemann b) Pré-intégration

FIGURE 2.8 – Comparaison des méthodes d'intégration [SKLE03].

$$\alpha(s_f, s_b, l) \approx 1 - e^{-\int_0^1 \tau((1-\omega)\cdot s_f + \omega \cdot s_b) l d\omega}$$

La couleur émise par ce même segment (s_f, s_b) est approximée de la même manière :

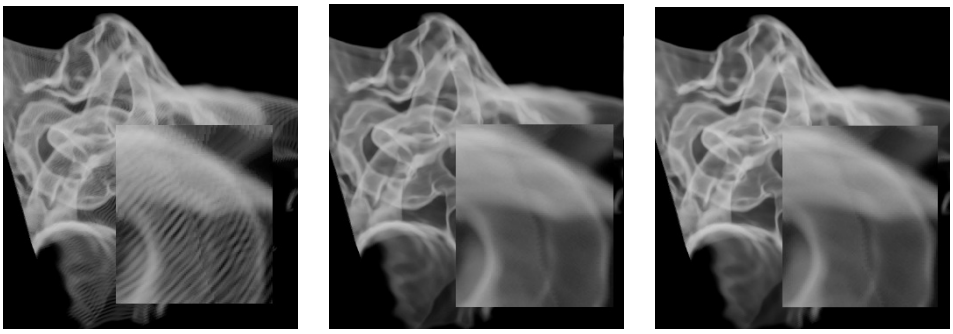
$$c(s_f, s_b, l) \approx \int_0^1 c((1-\omega)\cdot s_f + \omega \cdot s_b) \cdot \tau((1-\omega)\cdot s_f + \omega \cdot s_b) \cdot l \cdot e^{-\int_0^\omega \tau((1-\omega')\cdot s_f + \omega' \cdot s_b) l d\omega'} d\omega$$

L'intégrale du rendu volumique prend alors la forme suivante :

$$\begin{aligned} I &\approx I_b \cdot \prod_{i=1}^{n} (1 - \alpha(s_f(i), s_b(i), l(i)) \\ &+ \sum_{i=1}^{n} c(s_f(i), s_b(i), l(i)) \cdot \prod_{j=1}^{s-1} (1 - \alpha(s_f(j), s_b(j), l(j)) \end{aligned} \quad (2.1)$$

Cette nouvelle approximation de l'intégrale du rendu volumique correspond à la méthode des trapèzes, plutôt qu'une approximation par des rectangles pour la somme de Riemann, comme illustré par la Figure 2.8.

En pratique, cette méthode est implémentée sous la forme d'un tableau à deux dimensions pré-calculé à partir de la fonction de transfert. En effet, pour une épaisseur de tranche fixée l, c et α ne dépendent que de s_f et s_b, or en pratique s_f et s_b prennent des valeurs discrètes de 0 à 255 pour un codage des données sur 8 bits (respectivement 0-65535 pour 16 bits). Ce tableau est pré-calculé et ensuite stocké



a) Riemann, 128 tranches | b) Riemann, 284 tranches | b) Pré-intégration, 128 couches

FIGURE 2.9 – Visualisation d'un jeu de données (128 x 128 x 30) en post-classification [EKE01].

sous forme d'une texture 2D sur la carte graphique. Au moment de la visualisation, la couleur et l'opacité sont récupérées dans cette texture pour chaque couple d'échantillon (s_f, s_b).

Par la méthode des trapèzes, le signal reconstruit est plus proche de celui d'origine. Par ailleurs, l'utilisation de la pré-intégration capture les hautes fréquences de la fonction de transfert. La Figure 2.9 illustre respectivement un rendu sans pré-intégration en utilisant une géométrie de proximité composée de 128 tranches, puis un rendu sans pré-intégration avec 284 tranches et un rendu pré-intégré avec 128 couches. Le premier rendu laisse apparaître des artefacts dû à un échantillonnage insuffisant. Les deux derniers produisent un résultat équivalent, mais en divisant par deux le nombre de tranches utilisées pour le rendu pré-intégré. On obtient ainsi une meilleure performance.

Néanmoins, cette méthode dispose d'inconvénients. Un premier inconvénient est que la table calculée l'est pour un échantillonnage à pas constant. Si le pas n'est pas constant, on doit utiliser une table 3D. Ce qui est plus coûteux en mémoire et en temps pour le pré-calcul. De plus, cette technique n'est pas applicable si les données volumiques sont au format flottant, car il faut disposer d'une représentation discrète des données scalaires.

LUM ET AL. [LWM04] proposent une méthode permettant d'optimiser le temps de pré-calcul de la table de pré-intégration 2D. Plus récemment, EL-HAJJAR ET AL. [HMDM08] proposent une méthode permettant de faire une intégration du second ordre sur les données scalaires, permettant ainsi de capturer des variations non-linéaires des valeurs scalaires entre échantillons consécutifs. Cette méthode impose cependant l'utilisation d'une table de pré-intégration de dimension 3.

Chapitre 3

Éclairage et ombrage

Au delà de la qualité du rendu au sens numérique du terme, il est aussi important de pouvoir identifier les structures présentes dans un jeu de données : c'est l'objectif même de la visualisation scientifique. La nature semi-transparente de la visualisation volumique ne facilite effectivement pas cette identification. En effet, quand on projette le volume sur l'espace image, chaque pixel porte l'information de tous les échantillons accumulés le long d'un rayon. Cette quantité importante d'information peut entraîner une dégradation de la compréhension de l'information. Ce trop plein d'information peut être partiellement corrigé en spécifiant précisément la fonction de transfert, mais cela constitue souvent un travail laborieux. C'est dans cette perspective d'amélioration de la perception des structures internes que des techniques d'éclairage ont été développées.

3.1 Éclairage local

L'éclairage permet d'améliorer sensiblement la perception en visualisation volumique. En effet, l'oeil humain est habitué à distinguer les formes et les surfaces grâce à l'éclairage. Nous avons vu précédemment en Section 1.1.4 que l'on utilise le modèle émission/absorption pour visualiser des données volumiques. Pour rappel, on considère un ensemble de rayons qui traversent le corps participatif (le champ scalaire 3D). Suivant le modèle émission/absorption, les particules contenues dans le volume vont contribuer à la lumière accumulée le long de chaque rayon.

Avec l'éclairage, on rajoute une ou plusieurs sources de lumière externes, qui vont participer à l'illumination du volume. En traversant le volume, ces rayons de lumière sont soumis aux mêmes phénomènes d'émission et d'absorption que les rayons utilisés pour produire une image (cf. Figure 3.1).

Éclairage de Phong

En visualisation surfacique, l'éclairage local consiste souvent à utiliser par simplicité les modèles empiriques de Phong [Pho75] ou de Blinn [Bli77]. Ces modèles se basent tous deux sur l'utilisation de la normale au point considéré, de la position de l'observateur et de la source lumineuse. Ne disposant pas de surface à proprement parler dans le cas volumique, la normale est remplacée par un gradient calculé par différence centrale (cf. Formule 3.1). Le gradient peut ainsi être pré-calculé ou calculé à la volée pendant le rendu.

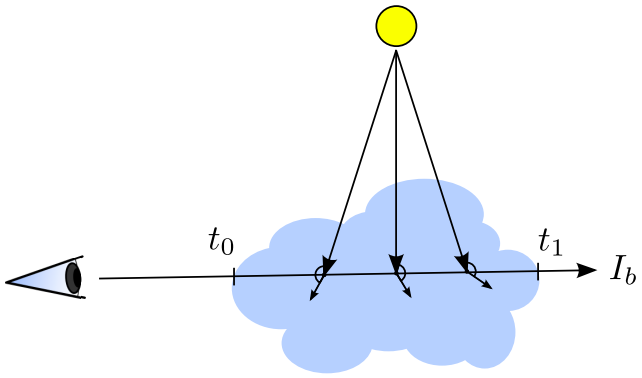


FIGURE 3.1 – Éclairage direct par une source externe.

$$g(x,y,z) = \begin{pmatrix} \frac{1}{2} \times f(x-1,y,z) & + & \frac{1}{2} \times f(x+1,y,z), \\ \frac{1}{2} \times f(x,y-1,z) & + & \frac{1}{2} \times f(x,y+1,z), \\ \frac{1}{2} \times f(x,y,z-1) & + & \frac{1}{2} \times f(x,y,z+1) \end{pmatrix} \tag{3.1}$$

Par souci de simplification et d'efficacité de calcul, on néglige la diffusion et l'absorption au sein du volume de données pour les rayons en provenance de la source lumineuse. L'éclairage de Phong pour une source lumineuse est intégré dans l'intégrale du rendu volumique de la manière suivante :

$$\begin{aligned} I(t_0,t_1) &= \int_{t_0}^{t_1} [K_a \cdot c(t) + K_d \cdot c(t) \cdot (\frac{\nabla g(t)}{\|\nabla g(t)\|} \cdot L)) \\ &+ c_s(t) \cdot K_s (V \cdot R)^{n_\delta}] \cdot \tau(t) \cdot e^{-\int_{t_0}^{t} \tau(u) du} dt \end{aligned} \tag{3.2}$$

avec K_a, K_d et K_s, respectivement les coefficients ambiant, diffus et spéculaire ; $\frac{\nabla g(t)}{\|\nabla g(t)\|}$, le gradient normalisé au point t considéré ; L, la direction de la source lumineuse ; V, la direction de l'observateur ; R, la direction d'un rayon de lumière parfaitement réfléchi sur la surface et n_δ, le coefficient de brillance associé au reflet spéculaire. De plus, on associe la couleur du point courant $c(t)$ aux composantes ambiante et diffuse. On peut choisir d'attribuer une couleur $c_s(t)$ spécifique, celle de la source lumineuse, pour le reflet spéculaire, mais on recourt généralement au blanc. L'intégration de l'éclairage est donc très simple une fois le gradient calculé : pour chaque échantillon, il suffit de calculer les 3 composantes : ambiante, diffuse et spéculaire, puis de les sommer. L'éclairage de Phong permet pour un coût négligeable en terme de temps de calcul supplémentaire d'apporter une bien meilleure perception en visualisation volumique (cf. Figure 3.2).

Cependant, cette méthode d'éclairage n'est pas parfaite : le gradient étant calculé à partir des données, il se peut qu'il soit faussé par du bruit présent dans les données. De plus, les plages homogènes de données engendrent également un gradient nul.

Concernant l'intégration de l'éclairage dans la pré-intégration, on peut procéder en considérant un gradient moyen pour une couche, calculé à partir de la moyenne des gradients au point d'entrée et au point de sortie, comme le suggèrent ENGEL ET AL. [EKE01]. Cette méthode n'assure cependant pas la continuité de l'éclairage d'une couche à l'autre. LUM ET AL. [LWM04] proposent une méthode permettant de mieux intégrer l'éclairage dans une couche. Ils décomposent la table nécessaire à la

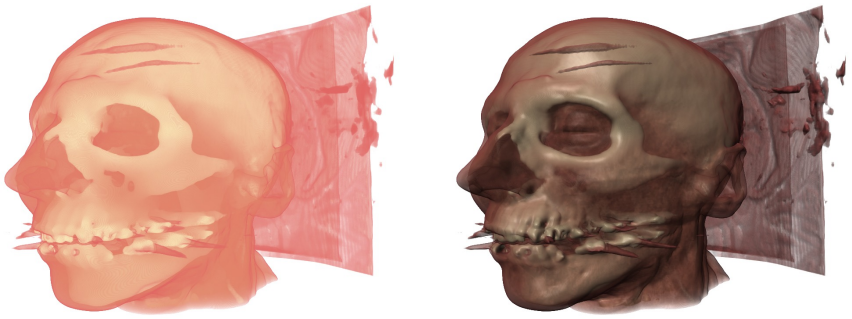


FIGURE 3.2 – Exemple de visualisation volumique sans éclairage (à gauche) et avec éclairage de Phong (à droite).

pré-intégration en deux paires de tables : deux pour la partie diffuse et deux pour la partie spéculaire. Chaque table représente la variation de la couleur possible entre chaque paire d'échantillon pondéré vers le premier ou vers le second échantillon. Cette approche revient à supposer une variation linéaire du vecteur gradient entre deux échantillons consécutifs.

Ombrage

Pour améliorer encore le "réalisme" de l'éclairage, on peut considérer l'absorption de la lumière émise par une source lumineuse quand elle traverse le volume. Cette technique est appelée *ombrage*. BEHRENS ET RATERING [BR98] proposent d'ajouter des ombres au rendu volumique basé sur les textures. Pour cela, ils utilisent des buffers auxiliaires pour accumuler successivement les tranches de textures en les décalant selon un axe spécifique donné par la direction de la lumière. Le résultat est utilisé pour atténuer les contributions lors de la visualisation. Toujours dans le cadre du rendu volumique basé sur les textures, KNISS ET AL. [KKH02] proposent de réaliser le rendu en utilisant des tranches obtenues par intersection entre le volume et le plan orthogonal au demi-vecteur entre la direction de l'observateur et la direction de la lumière, de telle sorte que l'ensemble des tranches soit visible par la source lumineuse et l'observateur. Ils utilisent une approche multi-passes et un buffer hors-écran (espace de rendu hors-écran sur la carte graphique) pour enregistrer l'atténuation de la lumière sur les tranches de 0 à i et utilisent ce résultat pour le rendu de la $(i+1)$ ème tranche côté observateur. Ils améliorent leur technique [KPHE02, KPH$^+$03] en ajoutant un buffer d'accumulation pour la lumière, ce qui leur permet de simuler une partie de la diffusion de lumière au moyen d'une opération de floutage ("blurring" en anglais).

Dans le cadre du rendu volumique par lancer de rayons, HADWIGER ET AL. [HKSB06] utilisent aussi le point de vue de la lumière pour générer de l'ombrage. Ils effectuent en fait un lancer de rayons depuis la source lumineuse, puis pour chaque échantillon successif, ils stockent le résultat intermédiaire dans une texture, qui est accédée dans la phase de rendu pour réaliser l'ombrage. La qualité de leur ombrage dépend donc de l'échantillonnage choisi. Ropinski et al. [RKH08] comparent les performances de différentes techniques d'ombrages dans le cadre du lancer de rayon GPU. La Figure 3.3 illustre des exemples de rendus pour différentes techniques.

FIGURE 3.3 – De gauche à droite, exemple de rendus avec les techniques d'ombrage de BEHRENS ET RATERING [BR98], de KNISS ET AL. [KKH02, KPHE02, KPH+03] et de HADWIGER ET AL. [HKSB06].

3.2 Éclairage par facteurs d'occultation ("Ambient occlusion")

3.2.1 Description générale

Les techniques d'Ambient Occlusion ont été introduites par ZHUKOV ET AL. [ZIK98] et IONES ET AL. [IKSZ03] avec la notion d"'Obscurance" dans le contexte de la visualisation surfacique. Cette technique consiste à analyser la géométrie au voisinage d'un point P en tenant compte de la distance d à laquelle elle se trouve par rapport à ce point P. Cette distance d influe sur l'éclairage en P : si d est petit, P recevra moins de lumière, et inversement si d est grand. Elle se traduit par la formule 3.3.

$$Obs_p = \frac{1}{\pi} \int_{\omega \in \Omega} \rho(d(P,\omega))(N \cdot \omega) d\omega \tag{3.3}$$

avec :

- $\rho(d(P,\omega))$, une fonction de la distance variant entre 0 et 1, indiquant l'intensité de la lumière provenant de la direction ω,
- $d(P,\omega)$, la distance du point P au premier élément de géométrie intersecté,
- $(N \cdot \omega)$, le produit scalaire entre la normale au point P et la direction considérée ω.

Par la suite, BREDOW [Bre02] et LANDIS [Lan02] ont simplifié ce modèle en supprimant la notion de distance. Ce nouveau modèle, appelé l'Ambient Occlusion, a notamment été utilisé dans le cadre du moteur de rendu utilisé par Pixar [Bre02, Lan02]. L'Ambient Occlusion est présentée comme l'intégration d'une fonction de visibilité dans l'hémisphère centré sur la normale N et décrit par l'angle solide Ω au point considéré P :

$$AO_p = \frac{1}{\pi} \int_{\Omega} V(P,\omega)(N \cdot \omega) d\omega \tag{3.4}$$

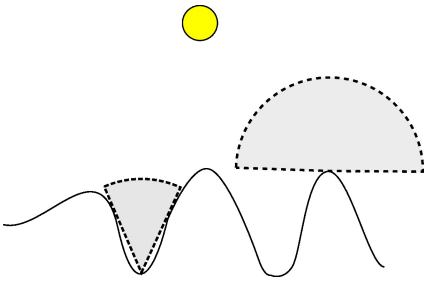


Figure 3.4: Valeur d'Ambient Occlusion pour un point situé dans une vallée et sur une crête.

On simplifie la formule 3.3 en remplaçant la fonction de distance ρ par une fonction de visibilité V, qui vaut 0 si P est occulté dans la direction ω ou 1 sinon. Méndez-Feliu et Sbert [MFS09] proposent une étude de l'ensemble des techniques d'Obscurance/Ambient Occlusion appliqué au rendu surfacique et volumique.

Visuellement parlant, l'Ambient Occlusion est une technique permettant d'obtenir un ombrage similaire à celui d'une journée nuageuse. Cela consiste à projeter la géométrie entourant un point P sur la sphère centrée sur la normale N en ce point, puis à évaluer la portion de l'hémisphère couverte. La figure 3.4 illustre le fait qu'un point situé dans une "vallée" de la géométrie (cf. Figure 3.4 gauche) recevra moins de lumière qu'un point situé sur une crête (cf. Figure 3.4 droite).

L'Ambient Occlusion permet d'avoir une bonne perception de la forme des objets à travers les occultations propres, avec par exemple les "vallées" ou creux dans la géométrie, et les occultations inter-structures ou ombres de contact, par exemple quand deux surfaces sont proches l'une de l'autre. Il existe des techniques très rapides et travaillant en espace image permettant de calculer une approximation de l'Ambient Occlusion en visualisation surfacique, notamment le *Screen-Space Ambient Occlusion*, proposé par Mittring [Mit07] pour le jeu *Crysis*. Cette technique consiste à examiner la distance entre un point de l'espace 3D et le pixel correspondant dans l'espace image. Pour un pixel considéré p, si ses voisins dans l'espace image sont plus loin en terme de distance, p est le projeté d'un point sur une crête, et inversement si tous ses voisins sont plus proches, il est situé dans une vallée.

De plus, des études de perception ont été menées par Langer et Bülthoff [LB99, LB00] soutenant que la perception était améliorée avec l'usage d'une lumière diffuse ou lumière de ciel nuageux par rapport à un éclairage directionnel.

3.2.2 Utilisation en visualisation volumique

Nous décrivons dans ce qui suit l'usage de l'Ambient Occlusion en rendu volumique direct, en introduisant d'abord les méthodes utilisant une phase de pré-calcul, puis les méthodes le calculant à la volée.

Méthodes avec phase de pré-calcul

Les méthodes suivantes se basent sur un pré-calcul des facteurs d'Ambient Occlusion/Obscurance. Les facteurs sont placés dans un volume de taille identique à

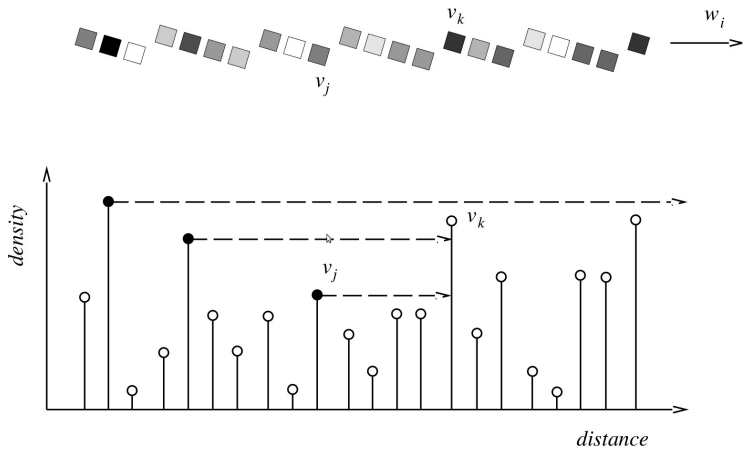


FIGURE 3.5 – Illustration du lancer d'un rayon et des voxels occultants par STEWART [Ste03].

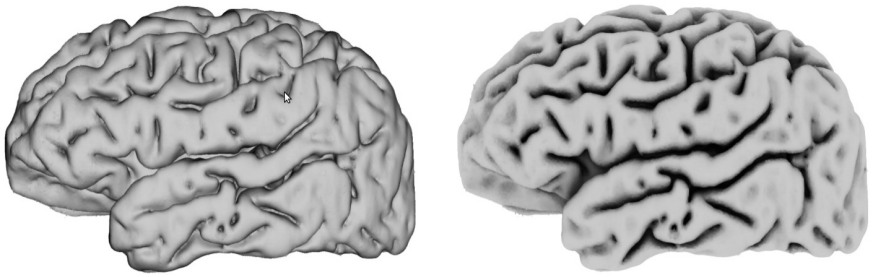


FIGURE 3.6 – Exemple de visualisation volumique d'un cortex cérébral, converti en données volumiques depuis un modèle polygonal. L'image de gauche illustre l'éclairage de Phong, celle de droite le "Vicinity Shading" de STEWART [Ste03]

la taille initiale des données (appelé "light volume"), puis utilisés pour pondérer la couleur au moment de la phase de rendu volumique direct.

STEWART [Ste03] est le premier à utiliser une technique d'occultation en visualisation volumique. La technique, baptisée "Vicinity Shading", propose de calculer un volume de luminosité en utilisant un lancer de rayons. Calculé côté CPU, cette technique combine le résultat obtenu pour plusieurs directions. Chaque rayon est lancé en utilisant un algorithme de Bresenham [Bre65] étendu en 3D, afin de garantir que chaque voxel n'est utilisé que pour un rayon. STEWART se base sur la densité des voxels rencontrés sur un rayon lancé pour déterminer si un voxel est occulté, i.e. une densité supérieure signifie un voxel occultant. La Figure 3.5 illustre le lancer de rayons le long d'une ligne de Bresenham 3D pour une direction w_i (en haut). Chaque voxel est considéré comme occulté dès que l'on rencontre un voxel de densité supérieure le long de cette ligne, avec l'exemple de v_j qui est occulté par v_k. Il utilise aussi la pondération basée sur la distance du voxel initial au voxel occultant (Obscurance). La figure 3.6 illustre la mise en valeur des vallées dans la visualisation. Notons que cette méthode est relativement coûteuse, puisque les résultats présentés pour un jeu de données de taille 256^3 annoncent un temps de pré-calcul d'environ 2.7 secondes pour chaque direction d'échantillonnage w_i.

RUIZ ET AL. [RBV+08] se basent sur la méthode proposée par STEWART [Ste03] et étudient l'impact de différentes fonctions ρ de pondération en fonction de la distance. Ils étudient les fonctions suivantes :

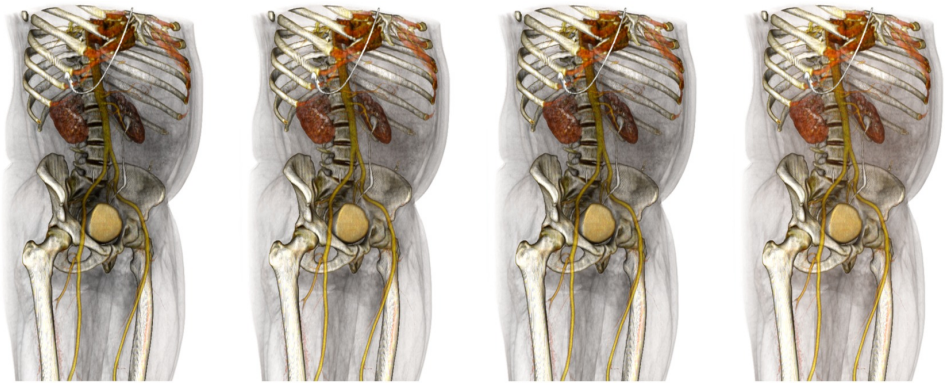


FIGURE 3.7 – Utilisation de différentes fonctions de pondération en fonction de la distance par RUIZ ET AL. [RBV+08]. De gauche à droite, fonction tout ou rien, linéaire, exponentielle et racine carrée.

- tout ou rien : $\rho(d) = 0$
- linéaire : $\rho(d) = \frac{d}{d_{max}}$, la fonction utilisée par STEWART [Ste03]
- exponentielle : $\rho(d) = 1 - \exp(-\frac{d}{d_{max}})$
- racine carrée : $\rho(d) = \sqrt{\frac{d}{d_{max}}}$

Selon eux, les fonctions tout-ou-rien et exponentielles produisent des résultats similaires. La fonction linéaire lisse le résultat comparé à la fonction tout ou rien, mais elle considère un nombre de voxels trop important. La fonction racine carrée semble selon eux le meilleur compromis, car elle permet d'obtenir les meilleurs résultats en terme de perception. Elle permet de mieux prendre en compte le voisinage proche et limite l'assombrissement. La Figure 3.7 illustre l'utilisation de ces différentes fonctions de pondération.

HERNELL ET AL. [HLY07] proposent une méthode utilisant les capacités de calcul des cartes graphiques. Cette technique consiste à calculer un Ambient Occlusion local (en anglais, "Local Ambient Occlusion" ou LAO), en considérant pour chaque voxel une sphère dans laquelle on échantillonne par lancer de rayons. Pour cela, ils calculent une texture contenant les facteurs d'Ambient Occlusion. Ils itèrent sur chaque tranche de texture en effectuant un rendu hors-écran, et calculent les contributions pour un ensemble de rayons. Ils accélèrent leur technique en se basant sur une approche multi-résolution, et ils entrelacent les phases de calcul et celles de rendus en accumulant progressivement les contributions. Ils ajoutent de plus une notion d'émissivité pour les voxels, ce qui leur permet de rendre brillantes certaines structures. La Figure 3.8 illustre la technique de LAO, avec notamment la projection des veines sur l'os du crâne. En terme de performance, la méthode de HERNELL ET AL. [HLY07] est meilleure que celle proposée par STEWART [Ste03], puisqu'ils annoncent 51 millisecondes par rayon pour un jeu de données de taille 256^3. Cependant elle ne permet toujours pas l'édition interactive de la fonction de transfert. Notons que l'ajout de paramètres utilisés dans la phase de précalcul peut présenter des inconvénients, car il faut en maîtriser les effets. Ce point est discuté dans le chapitre 6.

RITSCHEL [Rit07] utilise des harmoniques sphériques pour évaluer la fonction de visibilité sur un jeu de données. Il utilise le GPU pour accélérer les pré-calculs, et obtenir au final des ombres douces. Le changement de la fonction de transfert requiert un recalcul de l'ordre de quelques secondes. DESGRANGES ET ENGEL [DE07]

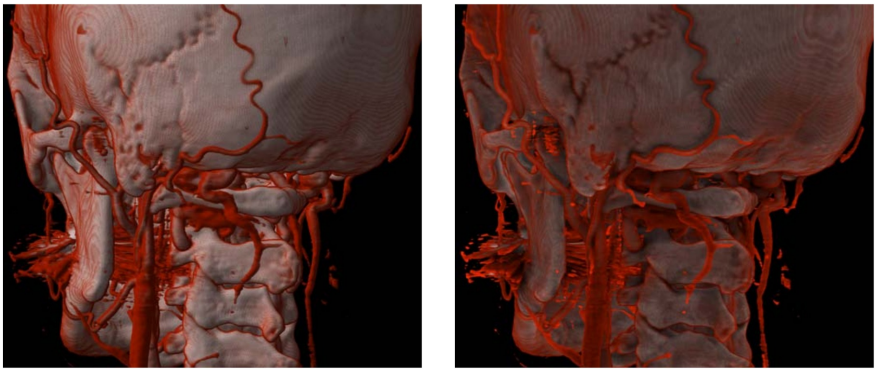


Figure 3.8: Exemple de visualisations en utilisant l'éclairage de Phong avec uniquement la composante diffuse à gauche et la méthode de "Local Ambient Occlusion" de Hernell et al. [HLY07] à droite.

ont publié une méthode de calcul de l'Ambient Occlusion au travers d'un brevet. Pour cela, ils parcourent l'ensemble du volume en utilisant des lignes de balayage, afin de compter les sauts (un nombre n de voxels supérieur à 0) entre chaque paire de voxels pleins, i.e. un voxel au dessus d'une certaine opacité après application de la fonction de transfert. Ces données sont placées dans un histogramme, et les sauts les plus fréquents sont détectés. Chaque taille de saut t significative est ensuite utilisée pour générer le "light volume" correspondant. Pour chaque voxel, les voxels contenus dans la boite englobant la sphère de rayon t sont sommés. Les différents "light volumes" sont finalement combinés par "blurring" pour obtenir un volume d'occultation. Une pondération est appliquée pendant cette phase, afin d'éviter de perdre trop en détails. Au niveau des performances, ils traitent un volume de 512^3 en 0.7 secondes.

Ropinski et al. [RMSD+08] considèrent deux hémisphères autour de chaque voxel : celui centré sur la normale et l'hémisphère opposé. Ils établissent pour chaque hémisphère un histogramme représentant l'influence qu'a chaque voxel dans l'hémisphère sur le voxel considéré en fonction de sa distance et de sa densité. Ils réduisent ensuite le nombre d'histogrammes en utilisant une méthode de "clustering", puis utilisent les histogrammes restant pour moduler la couleur de chaque voxel pendant la phase de rendu. En utilisant des histogrammes basés sur les densités, ils se rendent ainsi indépendants de la fonction de transfert. Ceci leur permet de réaliser des effets d'Ambient Occlusion et de "color bleeding", ou diffusion de la couleur sur les structures avoisinantes. Cependant, l'opération de clustering introduit des temps de pré-calculs importants, de l'ordre de plusieurs dizaines de minutes. Mess et al. [MR10] améliorent ce dernier travail en se basant sur CUDA. Ils déportent la génération des histogrammes et une partie de la phase de "clustering" pour réduire les temps de pré-calcul induits par la méthode. Selon eux, le processus est accéléré jusqu'à 10 fois.

Méthodes avec calcul à la volée

Ces méthodes n'utilisent pas de pré-calcul pour calculer l'occultation en rendu volumique. Le calcul est directement effectué à la volée pendant la phase de rendu en utilisant des méthodes en espace image.

Díaz et al. [DYV08, DVND10] proposent une technique similaire à la méthode employée par Mittring [Mit07] décrite en section 3.2.1. Ils proposent de générer une carte de profondeur à partir d'un rendu initial permettant de définir le bord

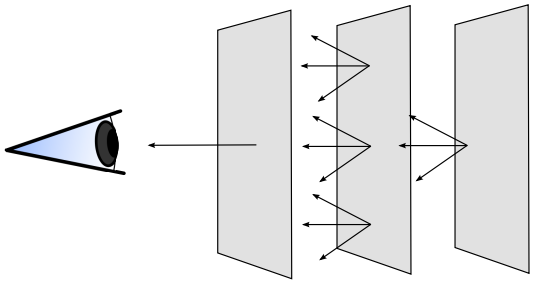


Figure 3.9: Méthode utilisée par Schott et al. [SPH+09].

des données à visualiser. Générer cette carte de profondeur impose de rencontrer une limite d'opacité et fait donc de cette technique une technique plus adaptée à la visualisation d'isosurfaces. Ils utilisent ensuite la technique des "summed-area tables" introduite par Crow [Cro84] sur la carte de profondeur. Cela leur permet d'évaluer pour un pixel considéré à quelle profondeur sont situés ses voisins. Ainsi si ses voisins sont plus proches de l'observateur le pixel considéré correspond à un creux, sinon il correspond à une crête. Ils utilisent ensuite cette information pour pondérer la couleur en ce pixel au moment de la visualisation.

Schott et al. [SPH+09] proposent une technique basée sur l'utilisation des textures 3D et de tranches alignées sur l'espace écran. Ils maintiennent un buffer d'occultation correspondant à l'opacité des n tranches précédemment traitées, et l'utilise pour la $(n+1)^{e}$ en cherchant pour chaque voxel contributeur l'occultation qui sera générée par l'accumulation des précédentes tranches (cf. Figure 3.9). Un des inconvénients de cette méthode est qu'elle ne permet pas de représenter les occultations inter-structures qui sont dans un même plan aligné sur l'espace image. De plus, cette méthode affecte les performances de la phase de rendu.

Šoltészová et al.[ŠPBV10] étendent la technique proposée par Schott et al. [SPH+09]. Ils permettent ainsi le placement interactif de la source lumineuse, fixée à la position de l'observateur dans l'article original.

Récapitulatif

Le tableau 3.1 présente un récapitulatif de différentes caractéristiques des méthodes décrites précédemment. On présente ainsi les caractéristiques suivantes :

- Le fait que le calcul dépend ou non de la fonction de transfert. Si oui, cela implique le recalcul des facteurs d'occultation en cas de changement de fonction de transfert,
- Le type d'implémentation réalisé. Si l'implémentation a été réalisée côté CPU, GPU ou une approche hybride,
- Le type de méthode : pré-calcul ou à la volée,
- Le type d'échantillonnage utilisé,
- L'espace de travail : espace objet (travail sur les données directement) ou espace écran (travail lors génération de l'image).

La majorité des méthodes de calcul de l'Ambient Occlusion présentées dépendent de la fonction de transfert, ce qui implique que la modification de cette dernière provoque un nouveau calcul des facteurs d'occultation. C'est pour cela que ce temps de calcul doit être minimisé. Les méthodes calculant à la volée ne sont quant à elles pas touchées par ce phénomène, car le calcul de d'occultation est directement intégré dans la phase de rendu et c'est donc sur ce plan que l'impact de performance

Méthode	Dépend de la fonction de transfert	Implémentation	Méthode
Stewart [Ste03]	Non	CPU	Pré-calcul
Hernell et al. [HLY07]	Oui	GPU	Pré-calcul
Ritschel [Rit07]	Oui	GPU	Pré-calcul
Desgranges et Engel [DE07]	Oui	CPU/GPU	Pré-calcul
Ropinski et al. [RMSD+08]	Non	CPU	Pré-calcul
Díaz et al. [DYV08]	Oui	GPU	A la volée
Schott et al. [SPH+09]	Oui	GPU	A la volée
Šoltészová et al.[ŠPBV10]	Oui	GPU	A la volée
	Échantillonnage	Espace	
Stewart [Ste03]	Lignes de Bresenham 3D	Objet	
Hernell et al. [HLY07]	Sphère autour d'un voxel	Objet	
Ritschel [Rit07]	Sphère autour d'un voxel	Objet	
Desgranges et Engel [DE07]	Boite englobante d'une sphère autour du voxel	Objet	
Ropinski et al. [RMSD+08]	Les 2 hémisphères autour d'un voxel	Objet	
Díaz et al. [DYV08]	Voisinage du voxel projeté sur le plan image	Écran	
Schott et al. [SPH+09]	Cône en direction de l'observateur	Écran	
Šoltészová et al.[ŠPBV10]	Cône en direction modifiable	Écran	

Table 3.1: Tableau récapitulatif des caractéristiques des méthodes d'occultation.

se fait. Certaines méthodes ont essayé de s'abstraire de la fonction de transfert en se basant sur les densités, mais elles induisent des temps de calcul plutôt de l'ordre des minutes, voire de la dizaine de minutes.

Concernant le type d'échantillonnage réalisé, il y a plusieurs approches pour les méthodes travaillant en espace objet. Certaines se basent sur le lancer de rayons et permettent de conserver la position relative des voxels dans le voisinage d'un voxel considéré. D'autres comme la méthode basé sur les histogrammes de Ropinski et al. [RMSD+08] et celle de Desgranges et Engel [DE07] font perdre cette notion de position relative des voxels, ce qui peut mener à des ombres de moins bonne qualité qu'avec un lancer de rayons classique.

Concernant la performance des différentes méthodes, les méthodes travaillant en espace objet se basent toutes sur un pré-calcul et n'affectent donc pas la performance du rendu final, contrairement aux méthodes travaillant en espace écran.

3.3 Rendu volumique direct expressif

Afin d'améliorer la perception des structures dans les données, on peut aussi recourir au rendu expressif ou non-photoréaliste. Dans ce contexte, on modifie l'échantillonnage et/ou la manière dont sont combinés les échantillons de sorte à mettre en valeur certaines caractéristiques des données : structures particulières, courbure, ... On n'essaie donc plus de simuler la physique de la lumière et d'obtenir un rendu photoréaliste, mais d'obtenir un rendu expressif et de donner du sens à l'image finale.

Une des premières techniques de rendu volumique non-photoréaliste est la projection de l'intensité maximale ou "Maximum Intensity Projection" (MIP) qui consiste à n'afficher que l'intensité maximale rencontrée sur un rayon. Elle a été initialement introduite par Wallis et al. [WMLK89]. Avec cette technique, le rapport information par pixel est dans ce cas de 1, mais l'intensité maximale projetée ne correspond pas forcément aux structures que l'on souhaite analyser. De plus, on perd le contexte spatial de l'information capturée, notamment en matière de profondeur, ce qui peut mener à des visualisations prêtant à confusion comme l'illustre l'image de gauche sur la Figure 3.10. Les travaux récents de Diàz et al. [DV10] améliorent ce point en proposant de considérer les échantillons dans un intervalle autour de l'intensité maximale rencontrée sur chaque rayon. Ils pondèrent chaque échantillon par leur profondeur réincorporant ainsi leur position relative (cf. Figure 3.10 droite). On peut aussi utiliser le rendu additif, qui consiste à additionner les couleurs pré-multipliées des échantillons sur les rayons lancés.

Dans le cadre du rendu non-photoréaliste, Gooch et al. [GGSC98, GSG+99] développent des techniques non-photoréalistes pour la visualisation surfacique, dont l'ombrage de Gooch. Celui-ci consiste à modifier l'éclairage d'une surface en ne faisant plus varier la teinte des couleurs du sombre vers le clair, mais d'une couleur froide à une couleur chaude, par exemple de bleu à jaune (cf. Figure 3.11). Le contour ou silhouette est une autre composante importante des techniques non-photoréalistes. Celui-ci peut, par exemple, être calculé en considérant le produit scalaire entre la direction du rayon lancé depuis l'observateur et la normale au point considéré. Cette valeur est ensuite utilisée pour assombrir la couleur quand le résultat tend vers 0. Ceci a pour effet de mettre en valeur les bords des surfaces pour le point de vue courant (cf. Figure 3.11 droite). Lum et al. [LM02] proposent l'intégration des deux techniques précédentes dans un framework accéléré matériellement sur un cluster.

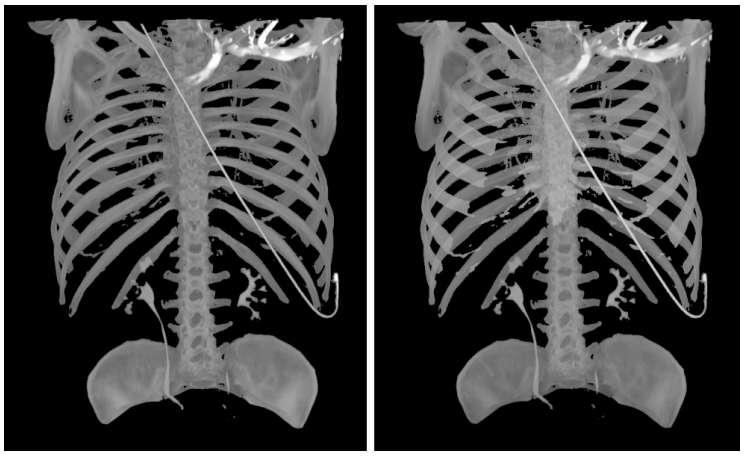


FIGURE 3.10 – Comparaison de la méthode MIP (à gauche) et de la méthode de MIP avec utilisation de la profondeur de DIÀZ ET AL. [DV10] (à droite).

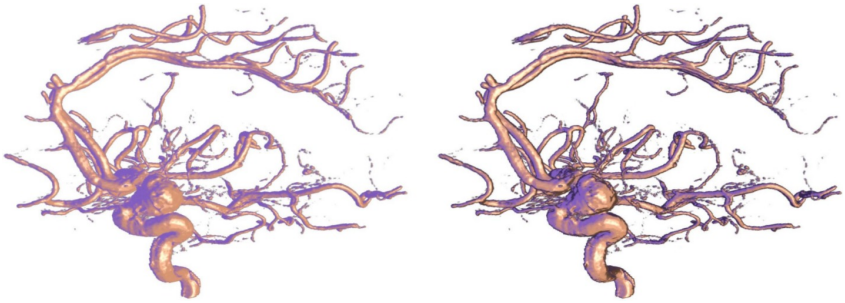


FIGURE 3.11 – Exemple de rendu avec modification de la variation de teinte d'une couleur froide vers chaude à gauche. L'image de droite ajoute une silhouette à la visualisation. Les images sont obtenues par la méthode de LUM ET AL. [LM02].

EBERT ET RHEINGANS [ER00] introduisent un framework permettant d'améliorer le rendu volumique en utilisant des techniques non-photoréalistes : mise en valeur des bords, ombrage avec variation de teinte d'une couleur froide vers chaude, diminution de la contribution en couleur en fonction de la distance à l'observateur, ...

D'autres techniques travaillent à modifier l'équation du rendu volumique pour améliorer la perception des données. HAUSER ET AL. [HMBG00] se basent sur des informations de segmentation pour appliquer une technique de rendu différente à des structures différentes. Ils utilisent pour cela le rendu volumique direct, qui permet de combiner différentes informations par pixel et le rendu MIP. KINDLMANN ET AL. [KWTM03] calculent une courbure afin d'appliquer plusieurs effets au rendu volumique, comme le contrôle de l'épaisseur des contours ou encore la mise en valeur des vallées et crêtes. VIOLA ET AL. [VKG04, VKG05] définissent un critère d'importance donné aux structures présentes dans un jeu de données. Ils utilisent ensuite ce critère pour afficher les structures moins importantes de manière plus transparentes que les structures importantes. Cela implique que les structures définies comme peu importantes n'occulteront pas les structures importantes, comme illustré en Figure 3.12.

MARCHESIN ET AL. [MDM07] proposent une technique adaptant l'opacité d'un échantillon sur un rayon en fonction du nombre d'échantillons utiles sur ce rayon.

FIGURE 3.12 – Exemple d'utilisation de la méthode utilisant le critère d'importance de VIOLA ET AL. [VKG04, VKG05].

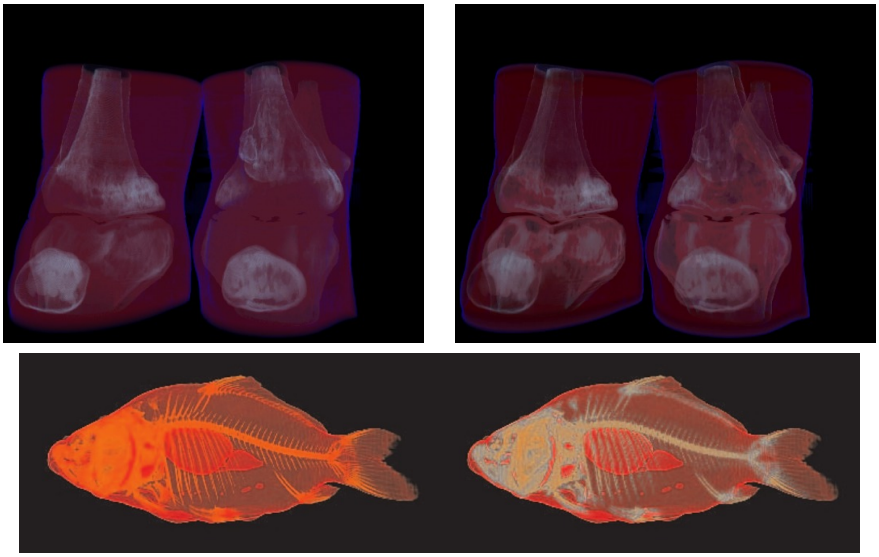


FIGURE 3.13 – La première ligne illustre la méthode adaptant l'opacité des échantillons de MARCHESIN ET AL. [MDM07] avec un rendu volumique normal à gauche et leur méthode à droite. La seconde ligne présente la méthode de CHUANG ET AL. [CWM09b] basée sur la préservation de la teinte de couleur, avec un rendu volumique normal à gauche et leur méthode à droite.

Pour cela, ils utilisent une segmentation binaire pour séparer les données rendues transparentes par l'application de la fonction de transfert des données rendues non-transparentes ou dites utiles. Dans un second temps, ils comptent le nombre d'échantillons utiles par pixel et adaptent l'opacité suivant ce nombre. CHUANG ET AL. [CWM09b] se basent sur l'hypothèse que l'on utilise la couleur pour regrouper des structures entre elles. Ils proposent alors un nouvel opérateur de composition, applicable au rendu volumique, et qui permet de préserver la teinte de la couleur, afin de pouvoir toujours bien identifier les différentes structures.

CHAN ET AL. [CWM+09a] proposent une méthode se basant sur trois critères de perception : la visibilité, la forme et la transparence. Ils optimisent ensuite la perception des structures dans la visualisation selon ces trois critères.

Chapitre 4

Accélération matérielle et parallélisation

4.1 Introduction

L'augmentation de la précision des capteurs (IRM, CT scan) et le besoin de simulations plus précises ont induit une augmentation de la taille des données volumiques. Cette augmentation s'est faite tant au niveau de la précision des scalaires, allant jusqu'à des flottants en double précision, que sur la taille des maillages porteurs des éléments de densité. On peut ainsi vouloir visualiser des simulations de très grande taille comme une flamme d'hélium, utilisée par [FCS+10], de taille 8192^3, soit 512 gigaoctets de données à visualiser si on considère une précision scalaire limitée à un octet. La visualisation ou l'extraction d'informations d'une telle masse de données requiert donc une puissance de calcul importante. Le parallélisme permet au travers de l'utilisation de plusieurs processeurs de déployer la puissance de calcul nécessaire. Dans le cadre de la visualisation, un des travaux pionniers est celui mené par Molnar et al. [MCEF94]. Ils proposent une taxonomie des méthodes possibles pour le rendu parallèle. Les deux méthodes applicables au rendu volumique sont les rendus dits "sort-first" et "sort-last". La première méthode consiste à associer une portion de l'espace écran à chaque nœud de calcul (cf. Figure 4.1). Il faut donc distribuer les données en fonction du découpage écran. Un problème induit par cette technique est qu'une modification du point de vue force à une redistribution des données entre les différents nœuds de calcul et cause ainsi beaucoup de transferts. Le rendu "sort-last" consiste à découper et distribuer d'abord le volume de données entre les différents processeurs (cf. Figure 4.2). Chaque processeur réalise ensuite la projection sur l'espace image de son sous-ensemble de données, et les images sont recomposées pour former l'image finale. Cette seconde approche est celle privilégiée dans la visualisation volumique de gros volumes de données, car elle évite les coûteuses redistributions de données de l'approche "sort-first".

Dans la suite du chapitre, nous présentons tout d'abord les évolutions récentes dans le domaine des cartes graphiques, puis les travaux dans le domaine de la visualisation volumique se basant sur des clusters et architectures multi-GPU. Enfin, nous terminons par une brève description de l'environnement CUDA et de l'architecture sur laquelle nous nous basons.

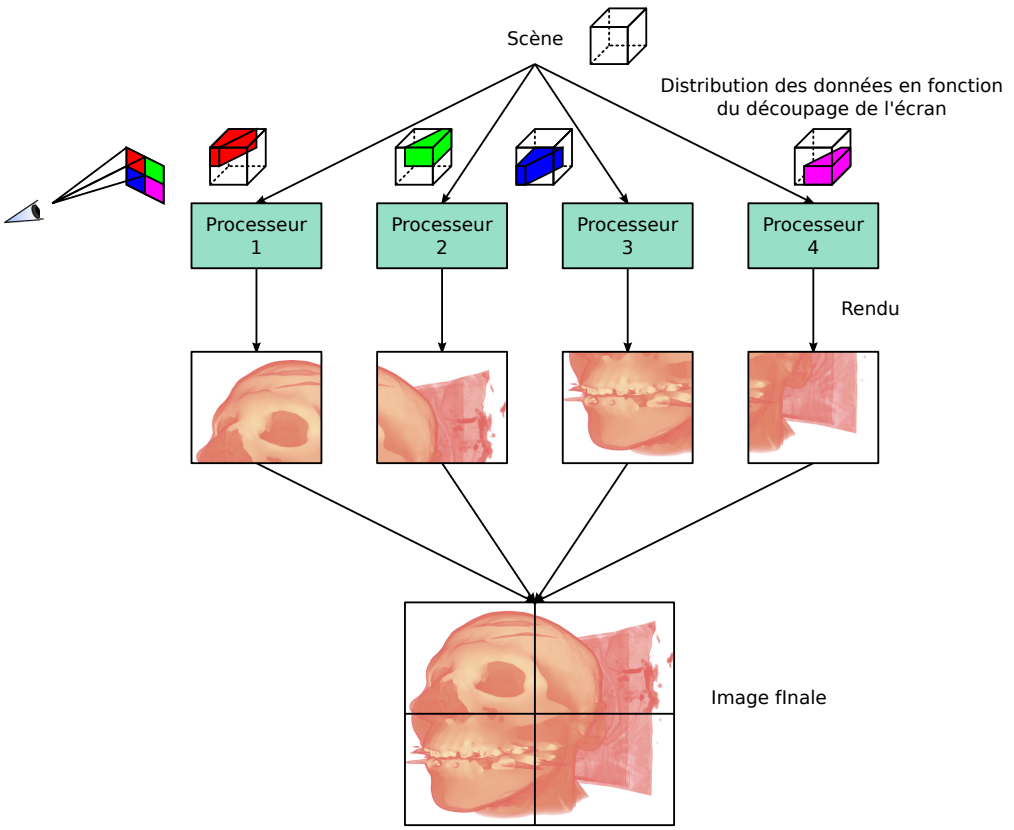


Figure 4.1: Rendu “sort-first” : on décompose l’espace image en autant de parties qu’il y a de processeurs. Chaque processeur est ensuite responsable du rendu de sa partie.

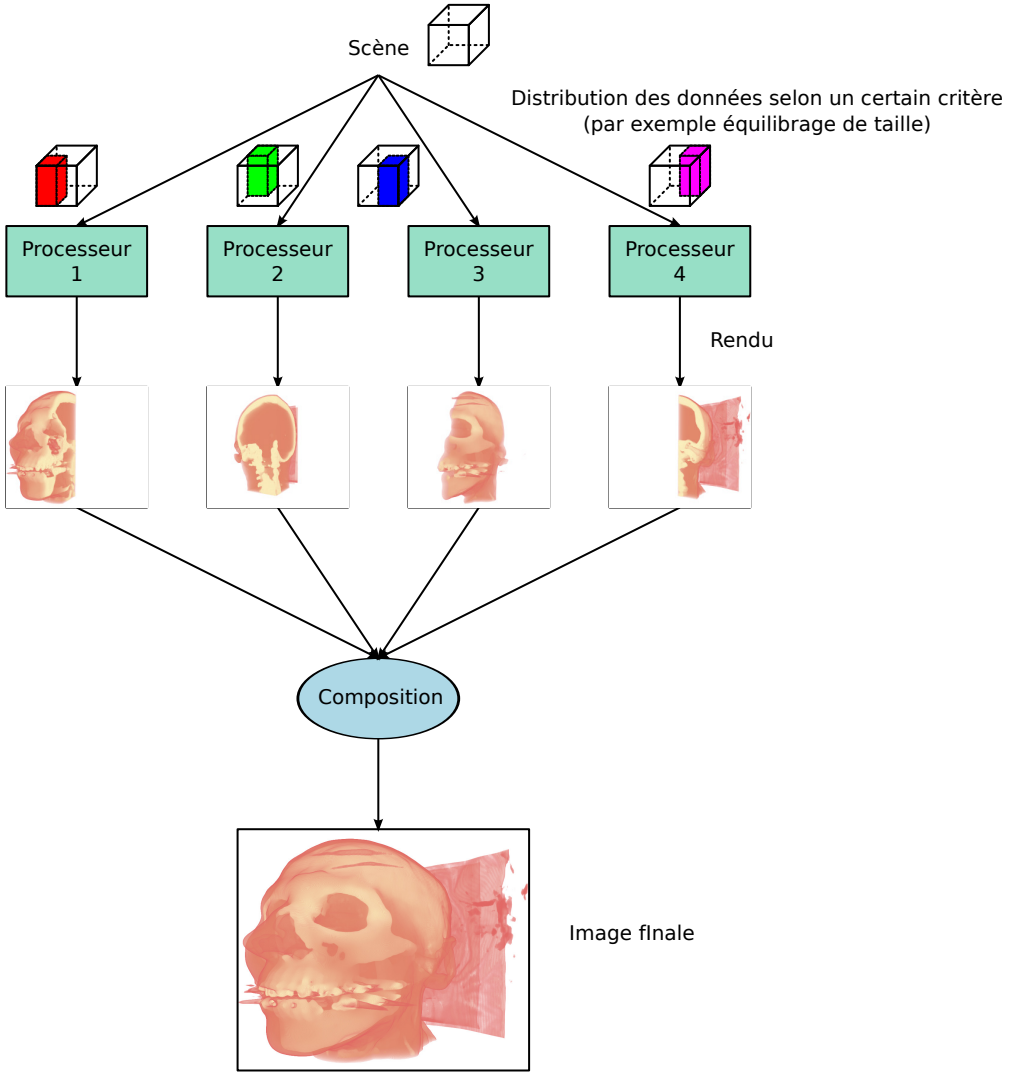


Figure 4.2: Rendu “sort-last” : on décompose l’espace des données en autant de parties qu’il y a de processeurs. Chaque processeur est ensuite responsable du rendu de sa partie.

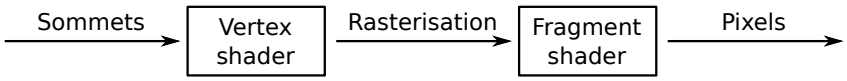


FIGURE 4.3 – Schéma d'utilisation des Vertex et Fragment shaders utilisés dans le cadre des API graphiques pour les GPUs. Les sommets sont d'abord traités par les vertex shaders. La phase de rasterisation permet de transformer les sommets en fragments, qui correspondent aux pixels de l'écran. Ces fragments sont ensuite traités par les fragments shaders avant d'être effectivement affichés.

4.2 Cartes graphiques

Avant les années 90, les cartes graphiques étaient destinées à des usages spécifiques du monde professionnel, comme par exemple le multi-écran proposé par les cartes graphiques Matrox. Les traitements graphiques dans le monde de la recherche étaient plutôt menés par des stations dédiées comme les machines SGI. C'est l'industrie du jeu vidéo qui a stimulé les innovations dans le monde des cartes graphiques grand public, qui par la suite allaient gagner en généricité. Les jeux vidéos se sont ainsi mis a réellement utiliser ces matériels dédiés vers le milieu des années 90 avec notamment l'arrivée des cartes graphiques Voodoo de 3dfx et des Riva/TNT de NVIDIA. Se succèdent alors 5 générations de Voodoo et 3 générations de cartes NVIDIA jusqu'à l'année 2000. L'année 2000 a été marquée par l'arrivée des premières GeForces de NVIDIA : la GeForce 256. NVIDIA a d'ailleurs vendu ce produit comme le tout premier GPU (Graphics Processing Unit), regroupant au sein d'une même puce les unités de transformation, d'éclairage et de traitement des triangles. Avec cette architecture, NVIDIA a notamment introduit un premier mécanisme rudimentaire de traitement des textures : Les "registers combiners" [NVIb], qui constituent une première version des "fragment shaders". RESK-SALAMA ET AL. [RSEB+00] ont utilisé ce mécanisme, ainsi que le multi-texturing pour accélérer et améliorer la qualité en rendu volumique direct basé sur les textures 2D.

En 2001, un nouveau pas important est franchi avec l'introduction des "shaders" dans les spécifications de DirectX 8.0, l'API référence de développement dans le monde du jeu vidéo, et d'OpenGL 1.4. Ces derniers sont des petits programmes écrits dans un language de haut-niveau et sont destinés à être exécutés par la carte graphique sur les unités de traitement spécifiques que sont les vertex et pixels shaders (cf. Figure 4.3). Ces derniers permettent ainsi respectivement d'effectuer des opérations par sommet ou par pixel. Les premières cartes implémentant ces mécanismes sont la troisième génération de GeForce chez NVIDIA et le RV200 d'ATI. L'ensemble des cartes graphiques sorties depuis sont de plus compatibles avec l'utilisation des Vertex/Fragment shaders, et disposent de nouvelles fonctionnalités à chaque génération : rendu HDR, ... On caractérise d'ailleurs les capacités des cartes graphiques selon les versions de "shader model" qu'elles supportent. Par exemple les dernières versions sont la version 4.1 pour OpenGL et 3.0 pour DirectX.

Les "shader languages" et l'apparition des techniques de rendu hors-écran, dont les "framebuffer objects", ont été les premiers pas vers l'utilisation des cartes graphiques pour effectuer des calculs massivement parallèles à destination autre que l'informatique graphique. En effet, il existe une manière typique de détourner le pipeline graphique pour effectuer des calculs autres que graphiques. Cela consiste à définir un "buffer" (ou tableau de pixels) hors-écran d'une taille (x, y), qui correspond à la taille d'un tableau contenant $x \times y$ éléments que l'on souhaite calculer. Dans un second temps, on effectue le rendu d'un rectangle couvrant l'ensemble de

l'espace écran de sorte que chaque pixel corresponde à un élément du "buffer" hors écran. On charge dans les "fragment shaders" le code parallèle que l'on souhaite utiliser. Puis ce dernier sera exécuté lors du traitement des pixels, remplissant ainsi le buffer hors-écran.

La dernière innovation majeure a été l'unification des unités de traitement "vertex shaders" et "fragment shaders" au sein des spécifications DirectX 10 et OpenGL 3.3. Il n'y a donc plus d'unités spécifiques dédiées au traitement des pixels ou des fragments. Ce changement a été introduit avec le G80 de NVIDIA et le R600 d'ATI. Depuis, l'utilisation des GPUs s'est répandue hors du domaine graphique avec l'apparition de frameworks dédiés tels que CUDA pour NVIDIA ou Stream pour ATI, tous deux sortis en 2007. Ces frameworks vont même jusqu'à laisser le programmeur décider de la manière dont se répartissent finement les "threads" sur les cartes et de gérer la hiérarchie mémoire. Un des objectifs de cette thèse est d'utiliser le multi-GPU, nous nous sommes donc focalisés sur ce type d'API. Il est en effet beaucoup plus pratique d'utiliser des API dédiées et facilitant l'accès aux cartes graphiques plutôt que l'API OpenGL, qui se destine plutôt aux environnements mono-carte. Dans le cas d'OpenGL, la présence de plusieurs cartes est gérée au niveau du driver et non au niveau applicatif, afin de soulager le travail de programmation pour les développeurs d'applications graphiques.

En 2008, un nouvel environnement, OpenCL, a été proposé par Khronos Group, consortium déjà en charge d'OpenGL. Il se veut unificateur des approches propriétaires proposées par ATI et NVIDIA, en s'appuyant sur leurs principes de programmation. Dans le cadre du travail de cette thèse, le choix s'est plutôt porté sur CUDA qu'OpenCL. Ce choix a été guidé par le fait qu'OpenCL était juste émergent au début de cette thèse et qu'il n'y avait pas encore de drivers stables pour les cartes NVIDIA. L'approche de programmation reste néanmoins très similaire.

En matière de rendu volumique direct, le parallélisme peut être utilisé dans différents contextes : pour la visualisation elle-même ou pour réduire les temps de pré-calcul. Ces contextes partagent les mêmes contraintes qui constituent les principaux challenges du parallélisme dans le cadre du rendu volumique : les transferts et l'équilibrage de charge entre les différents processeurs. Ces points deviennent de plus en plus cruciaux avec l'augmentation de la taille des jeux de données.

4.3 Clusters

Nous considérons les architectures de type cluster avec un ensemble de machines interconnectées par un réseau rapide, cf. Figure 4.4.

Dans le domaine de la visualisation volumique, de nombreux travaux ont déjà été menés pour permettre la visualisation interactive en s'appuyant sur des clusters. Magallon et al. [MHE01] utilisent un cluster de 4 machines équipées de Geforce 2 pour répartir les calculs de rendu en utilisant une stratégie sort-last, i.e. où les données sont réparties sur les noeuds. Ils essayent ensuite des stratégies de "blending" (ou fusion) différentes pour recombiner les rendus. Strengert et al. [SMW+04] basent leur travaux sur ceux de Magallon et al. [MHE01]. Ils optimisent la phase de blending en ne transférant que la zone utile de chaque GPU. Ils proposent d'utiliser une compression utilisant les ondelettes, afin d'améliorer les performances de transfert, en se basant sur les travaux de Guthe et al. [GWGS02]. Lee et al. [LSH05] combinent l'utilisation d'un octree et d'un arbre BSP. L'octree est tout d'abord utilisé pour déterminer les zones utiles dans les données. L'arbre BSP est ensuite appliqué pour découper le résultat obtenu par l'octree en autant de sous-

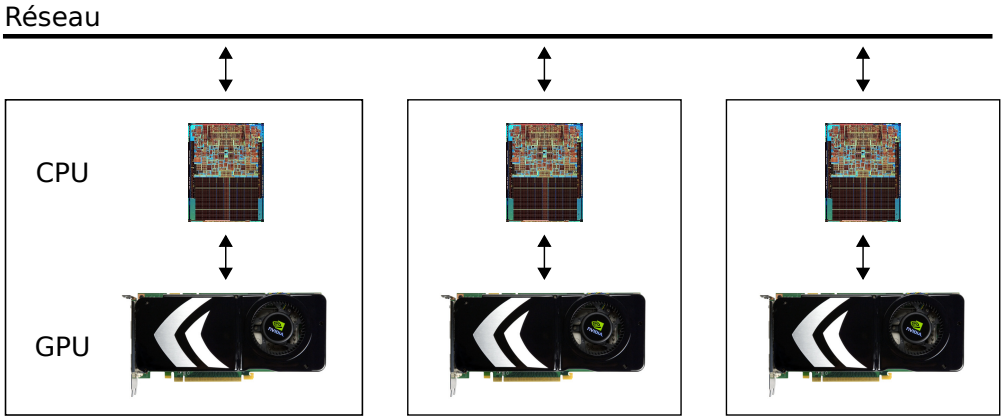


Figure 4.4: Schéma d'un cluster.

parties qu'il y a de nœuds de calcul. Marchesin et al. [MMD06] proposent une technique d'équilibrage basée sur un découpage en briques en formant un cache de données. Ils utilisent les temps de rendus précédents comme métrique d'équilibrage, afin d'équilibrer dynamiquement leur solution. Peterka et al. [PYRM08] utilisent le Blue Gene d'IBM pour montrer que le rendu volumique parallèle est viable sur une machine massivement parallèle, allant jusqu'à 4096 cœurs de calcul. Cela permet ainsi de n'entrevoir aucune limite au niveau de la taille possible des données à visualiser. Marchesin et Ma [MM10] utilisent un cluster de CPU pour effectuer le rendu. L'originalité de leur technique réside dans le fait qu'ils partagent une information d'occultation entre les cœurs des processeurs sur un même CPU et entre les machines. Celle-ci leur permet d'éliminer les blocs en attente dans le cache et occultés par des blocs situés plus en avant pour un point de vue considéré. Dans le contexte des architectures CPU multi-coeurs, Tchiboukdjian et al. [TDR10] proposent de prendre en compte les différents niveaux de cache des processeurs et leur hiérarchie pour accélérer deux algorithmes d'extractions d'isosurfaces. En réduisant le nombre de défauts de cache, ils permettent d'améliorer le passage à l'échelle sur plusieurs cœurs.

4.4 Multi-GPU

L'idée d'utiliser plusieurs unités de calculs en parallèle n'est pas une idée récente. Dans le domaine des cartes graphiques grand public, elle a été introduite par 3DFX avec la carte graphique Voodoo 2 en 1998. Elle consistait à installer 2 cartes graphiques dans un même ordinateur et à faire calculer la moitié des lignes de l'image à afficher par chaque carte ("Split Frame Rendering"), doublant ainsi le nombre d'images par seconde. La société 3DFX, toujours, a proposé en 2000 pour la Voodoo 5 d'utiliser plusieurs processeurs (le VSA-100) sur une même carte, allant jusqu'à 4 VSA-100 sur la Voodoo 5 6000. Malheureusement, cette carte n'a pas eu le succès escompté. Suite au rachat de 3DFX par NVIDIA en 2002, l'utilisation de plusieurs cartes graphiques pour le rendu (ou SLI [NVIa] pour NVIDIA) est réintroduit en 2004 en y ajoutant d'autre possibilités : l'Alternate Frame Rendering, qui consiste à réaliser le rendu des images paires par une carte et impaires par l'autre, ou encore le SLI Antialiasing, permettant de répartir le processus d'Antialiasing sur 2 cartes. En 2005, AMD/ATI ont aussi introduit un mécanisme multi-GPU, nommé Crossfire.

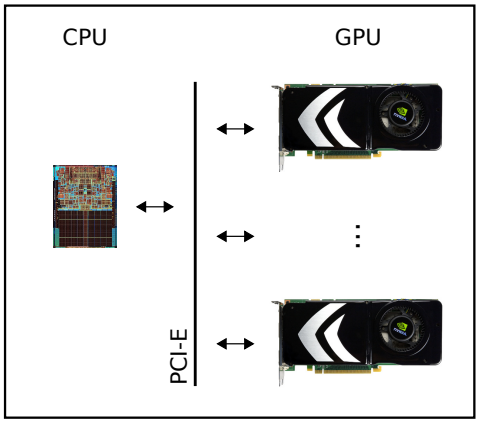


Figure 4.5: Schéma d'une architecture Multi-GPU.

La machine multi-GPU, telle que nous la concevons est donc composée d'une seule unité centrale avec un ou plusieurs processeurs centraux et de plusieurs GPUs interconnectés par le bus haut-débit PCI Express, voir Figure 4.5. Ces GPUs peuvent physiquement être sur la même carte, comme avec la 9800 GX2 ou la GTX 295 de NVIDIA.

Marchesin et al. [MMD08] utilisent une architecture multi-GPU pour effectuer une visualisation parallèle "sort-last" de données volumiques. Cette architecture se place ainsi entre la machine mono-GPU et le cluster en permettant la visualisation de données, dont la taille ne permettrait pas normalement le rendu sur une machine mono-GPU. Cette architecture est parfaitement compatible avec les clusters, on peut donc évidemment avoir des machines multi-GPU dans les clusters. Fogal et al. [FCS+10] étendent les travaux précédents en se basant sur un cluster de machines multi-GPU. Ils étudient ainsi les performances sur des machines avec un nombre important de GPU (256) et sur des jeux de données de taille importante (8192^3).

La génération d'isosurfaces pour le rendu-volumique a aussi fait l'objet de travaux sur les architectures parallèles. On peut ainsi citer les travaux de Martin et al. [MSM10], qui ont travaillé sur des clusters multi-CPU et multi-GPU pour accélérer la génération d'isosurfaces en exhibant une utilisation CPU et GPU supérieure à 85%.

Concernant le domaine spécifique de l'Ambient Occlusion, il n'y a pas à notre connaissance de travaux utilisant des architectures multi-CPU/GPU pour l'amélioration des temps de pré-calcul. Les enjeux de la parallélisation utilisée dans le cadre de cette thèse sont de réduire les temps de pré-calcul important de l'Ambient Occlusion. Notre propos n'a pas été de paralléliser le lancer de rayons "classique" (au sens de calculer une image d'une scène 3D de grande taille de façon interactive comme pour les travaux de Bigler et al. [BSP06]), même si cela reste parfaitement intégrable dans la méthode que nous décrivons au chapitre 7.

4.5 CUDA

Dans cette partie, nous donnons un aperçu de l'environnement de programmation CUDA et des contraintes liées à la programmation des cartes graphiques. L'utilisation de CUDA restreint le parc des cartes compatibles aux cartes NVIDIA. Cette partie décrira donc plus spécifiquement l'architecture NVIDIA. Néanmoins, OpenCL offre un modèle très proche, tout en proposant la compatibilité inter-constructeur.

FIGURE 4.6 – Comparaison schématique entre architecture CPU et GPU [NVI11].

4.5.1 Architecture

L'architecture des cartes graphiques est une architecture optimisée pour le graphisme. Ce type de calcul requiert d'effectuer un ensemble d'opérations identiques sur des sommets, puis les pixels, afin d'être affichés à l'écran. Les GPUs sont donc plutôt dédiés au traitement hautement parallèle et gourmand en calcul, c'est donc tout naturellement que le modèle choisi est un modèle de type parallélisme de données ou SIMD dédiant plus de transistors au traitement des données qu'au cache et contrôle de flux (cf. Figure 4.6).

Les GPUs sont en effet équipés d'un ensemble de multi-processeurs (*streaming multiprocessors*, SM), dont le nombre est variable selon le type de carte. Chacun de ces multiprocesseurs contient un ensemble de processeurs de flux (*streaming processor*, SP). Le nombre de processeurs de flux est variable selon la carte, de 8 pour les plus anciennes à 48 pour les plus récentes. De plus, chaque multiprocesseur dispose d'un ensemble de registres et d'une petite mémoire locale, appelée mémoire partagée.

Plus généralement, les caractéristiques et capacités de chaque carte sont définies au travers de la *Compute Capability*. Celle-ci change généralement de version à chaque nouvelle génération de cartes. Elle permet ainsi de déterminer par exemple si la génération utilisée est capable d'utiliser des opérations atomiques ou si elle est capable d'effectuer des opérations plus avancées telles que *printf* ou *malloc*.

4.5.2 Mémoire

Pour soutenir l'architecture multi-processeur, une mémoire sur plusieurs niveaux est répartie sur la carte (cf. Figure 4.7). Au plus proche des processeurs de flux se trouvent les **registres** (*registers*), qui ont le plus faible temps d'accès. Vient ensuite la **mémoire partagée** (*shared memory*), qui est commune à un multiprocesseur et est aussi très rapide d'accès. Elle est généralement utilisée pour précharger des données que l'on va utiliser plusieurs fois, ainsi que pour échanger des données entre threads. Dans les dernières générations de cartes, une partie de cette mémoire partagée peut être dédiée à la création d'un cache pour la mémoire globale.

La **mémoire des constantes** est une mémoire en lecture seule pour les processeurs de la carte. Elle réside en mémoire globale, mais un cache des constantes est présent sur chaque multi-processeur. De ce fait, cette mémoire est très rapide d'accès si on n'a pas de défauts de cache. Elle est typiquement utilisée pour stocker des valeurs constantes que tous les threads accèdent en même temps.

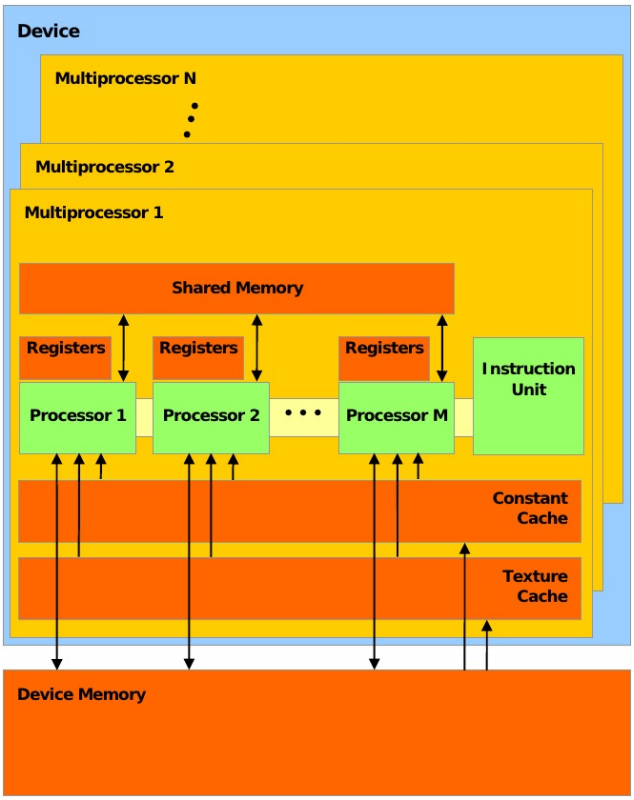


FIGURE 4.7 – Architecture mémoire d'un GPU [NVI11].

La **mémoire de texture** est une mémoire spécifique à l'informatique graphique. Elle dispose d'un cache et implémente matériellement certaines opérations telles que les interpolations.

Finalement, le dernier type de mémoire est la **mémoire globale**. Elle est la plus lente d'accès (400-600 cycles) et doit donc être évitée pour maintenir un bon niveau de performance.

4.5.3 Exécution

L'exécution d'un programme sur une carte graphique passe d'abord par l'écriture des fonctions qui seront exécutées sur les multiprocesseurs, appelées **noyaux** (kernels). Elles sont écrites dans un langage très proche du C, et compilées à l'aide du compilateur NVIDIA maison *nvcc* en langage intermédiaire PTX (compatible avec toutes les cartes graphiques NVIDIA) ou Cubin, un assembleur spécifique à la carte utilisée et dépendant des capacités de la carte. Le langage de programmation permet l'utilisation de conditionnelles et de boucles. Cependant, elles doivent être utilisées avec précaution, car une divergence sur une conditionnelle entre des threads exécutés en même temps provoquera la sérialisation des exécutions. Chaque partie de la conditionnelle sera donc exécutée uniquement par les threads concernés, laissant ainsi une partie des processeurs de flux inactifs à chaque divergence et sérialisation.

L'exécution sur les multi-processeurs est organisée en **threads**, **blocs** et **grilles**. On définit un bloc comme un ensemble de threads pouvant prendre la forme d'un tableau 1D, 2D, ou 3D. Les blocs sont ensuite organisés dans une grille à 1 ou 2 dimensions. La position du thread courant dans un bloc, ainsi que la position du bloc

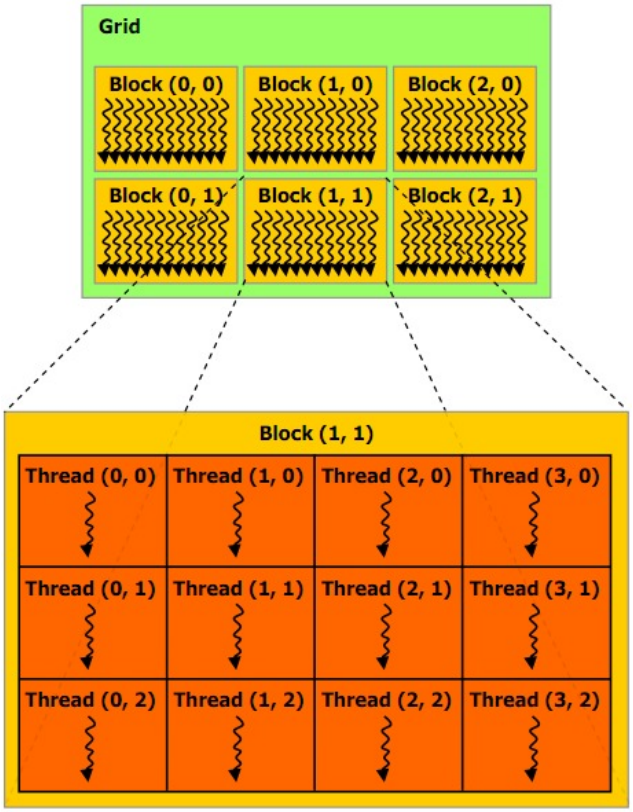


FIGURE 4.8 – Modèle d'exécution parallèle avec CUDA [NVI11]

courant dans la grille, peuvent être récupérées à tout moment à l'aide de variables spécifiques. Généralement, on utilise ensuite ces identifiants pour se repérer dans les données à traiter. La géométrie des blocs/grille permet de répartir les threads sur les ensembles de données à traiter.

Une fois les blocs et grille configurés, l'ensemble des blocs est envoyé à la carte et réparti entre les différents multiprocesseurs pour exécution. Sur chaque multiprocesseur, un bloc est découpé en petite unité de 32 threads pour exécution, appelée **warp**.

4.5.4 APIs

NVIDIA propose deux APIs de programmation pour CUDA : *Runtime* et *Driver*. Ces deux API mutuellement exclusives proposent chacune un niveau de programmation différent : l'API Runtime étant construite au dessus de l'API Driver. Cette dernière offre plus de contrôle que la première sur l'initialisation et le chargement des modules, ... Dans le cadre de ce travail, nous avons expérimenté l'utilisation de ces deux APIs et finalement décidé d'utiliser l'API Driver.

La première raison qui nous a guidé vers cette API est la possibilité de compilation à la volée. En effet, l'API Runtime oblige la compilation des noyaux CUDA au sein du même exécutable que le programme, or nous verrons au chapitre 6 que nous avons besoin de la compilation à la volée. De plus dans l'API Runtime, les textures doivent être définies statiquement pour être utilisées au sein des noyaux. L'API Driver permet quand à elle de charger des noyaux définis dans des modules externes sous la forme de PTX, un assembleur intermédiaire indépendant de l'archi-

tecture finale ou de Cubin, un “binaire” compilé spécifiquement pour l'achitecture courante. Elle permet aussi d'instancier et de lier dynamiquement des textures. Sa gestion des noyaux et des textures dynamiques en fait donc un meilleur candidat que l'API Runtime pour le rendu volumique.

De plus, l'API Driver permet de gérer finement les contextes CUDA. Ces contextes contiennent l'environnement et l'ensemble des structures définies pour un GPU spécifique. Avec l'API Runtime, un contexte est créé par l'appel à la fonction cudaSetDevice puis détruit à la fin du thread CPU courant. Si on utilise par exemple OpenMP pour créer des sections parallèles, on doit ainsi recréer les contextes à chaque nouvelle section. L'API Driver permet dynamiquement d'attacher et de détacher les contextes des threads CPU. On peut ainsi et sans interrompre les contextes GPU modifier à quel thread CPU est associé quel GPU.

Finalement pour faire le lien avec OpenCL, l'API Driver est plus proche de ce dernier. Le portage d'une application écrite avec l'API Driver vers OpenCL est donc plus facile que si l'application avait été écrite avec l'API Runtime.

L'ensemble des éléments précédents nous a donc conduit à utiliser l'API Driver. Cependant, les évolutions récentes de ces APIs montrent que certaines fonctionnalités de l'API Driver sont maintenant intégrées dans l'API Runtime, notamment celles permettant d'interroger l'état de la carte. On peut donc imaginer qu'à terme d'autres fonctionnalités encore spécifiques à l'API Driver soient elles aussi intégrées dans l'API Runtime, la rendant ainsi plus complète et utilisable à la place de l'API Driver utilisée dans le contexte de cette thèse.

Deuxième partie

Contributions

Chapitre 5

Rendu volumique direct pré-intégré avec interpolation non-linéaire des gradients

5.1 Introduction

La pré-intégration proposée par Engel et al. [EKE01] permet d'améliorer la précision numérique de l'intégration réalisée lors de la phase de rendu, comme nous l'avons vu en section 2.4 du chapitre 2. Pour rappel, la technique travaille sur l'échantillonnage du lancer de rayon. Elle consiste à supposer une variation linéaire de la densité entre les échantillons. Pour cela, on considère l'intervalle entre deux échantillons consécutifs ayant pour valeurs scalaires/densités (s_f, s_b). On précalcule l'accumulation de la couleur et de l'opacité entre paires d'échantillons en utilisant l'opérateur OVER de Porter et Duff [PD84], et ceci pour toutes les paires possibles (s_f, s_b) en rapportant le résultat à la longueur de l'intervalle D. En théorie, cette technique génère donc des tables à 3 dimensions (s_f, s_b, D). Néanmoins, dans le cas du lancer de rayons sur GPU, on peut limiter cette table à deux dimensions, car on utilise un pas D constant. Pour intégrer l'éclairage dans cette méthode, il faudrait 4 valeurs pour chaque échantillon s_f et s_b : (s, N_x, N_y, N_z), avec s, la valeur scalaire et (N_x, N_y, N_z), les composantes du gradient à l'échantillon considéré. Ce qui revient à générer une table à 8 dimensions, impraticable dans les GPUs d'aujourd'hui.

5.2 Intégration de l'éclairage de Blinn-Phong dans le rendu volumique pré-intégré

Avec un éclairage de Blinn-Phong au sein de l'intégrale du rendu volumique 1.2, on définit la somme suivante entre deux échantillons t_k et $t_{k+1} = t_k + D$:

$$\begin{aligned} I(t_k, t_{k+1}) &= \int_0^D [c(t+t_k) \cdot (K_a + K_d(\frac{\nabla g(t+t_k)}{\|\nabla g(t+t_k)\|} \cdot L)) \\ &+ c_s(t+t_k) \cdot K_s(H \cdot \frac{\nabla g(t+t_k)}{\|\nabla g(t+t_k)\|})^{n_\delta}] \\ &\cdot \tau(t+t_k) \cdot e^{-\int_{t_k}^{t} \tau(u)du} dt \end{aligned} \tag{5.1}$$

avec K_a, K_d, K_s et n_δ, respectivement les coefficients ambiant, diffus, spéculaire et

l'exposant spéculaire ; $\frac{\nabla g(t)}{\|\nabla g(t)\|}$, le gradient normalisé ; L, la direction de la lumière ; H, le demi-vecteur entre le vecteur V, direction de l'observateur et L. La couleur associée aux termes ambiant et diffus est la couleur issue de la fonction de transfert $c(t)$ au point considéré dans le volume. On introduit une couleur spéculaire $c_s(t)$, souvent remplacée par la couleur blanche et donc supprimée de l'équation. De plus, on néglige ici le terme $I_b \cdot e^{-\int_{t_0}^{t_1} \tau(t)dt}$, car il n'est pas affecté par l'éclairage (composante de fond).
Cependant, on ne peut pas utiliser directement cette équation lors de la phase de visualisation. En effet, dans le cas où un des produits scalaires venait à être négatif, c'est-à-dire avec des angles entre vecteurs supérieurs à 90°, on retire de l'énergie dans l'équation. Pour éviter cela, on peut recourir à l'utilisation de la fonction maximum avec 0 (éclairage d'un côté) ou inverser le vecteur gradient (éclairage des deux côtés).

Dans le cadre du rendu volumique pré-intégré, on doit tenir compte de deux gradients pour chaque échantillon de chaque paire (s_f, s_b) considérée. Engel et al. [EKE01] proposent d'utiliser la moyenne des gradients aux extrémités de l'intervalle. Avec cette technique, les variations brusques des gradients d'une tranche à l'autre causent des artefacts visuels sous la forme de discontinuités d'éclairage (cf. Figure 5.1 gauche).

5.3 Interpolation linéaire du gradient

Une meilleure solution consiste à interpoler linéairement le gradient le long du segment entre les paires d'échantillons : $N(t) = N_f \cdot (D-t)/D + N_b \cdot t/D$, avec $t \in [0, D]$, $t = 0$ représentant le premier échantillon et $t = D$, le second. Avec cette interpolation, l'équation 5.1 devient pour une tranche d'épaisseur D :

$$\begin{aligned} \tilde{C} &= \int_0^D [c(t+t_k) \cdot (K_a + K_d(\frac{N_f \cdot (D-t) + N_b \cdot t}{D} \cdot L)) \\ &+ c_s(t+t_k) \cdot K_s(H \cdot \frac{N_f \cdot (D-t) + N_b \cdot t}{D})^{n_\delta}] \\ &\cdot \tau(t+t_k) \cdot e^{-\int_{t_k}^{t} \tau(u)du} dt \end{aligned} \tag{5.2}$$

Cette technique correspond aux travaux de Lum et al. [LWM04]. Ils proposent ensuite de découpler le calcul de l'éclairage pour les deux gradients. On obtient deux intégrations séparées pondérées par un facteur linéaire croissant de 0 à 1 pour la première et décroissante de 1 à 0 pour la seconde :

$$\begin{aligned} \tilde{C} &\approx \int_0^D [c(t+t_k) \cdot (K_a + K_d(N_f \cdot L)\frac{(D-t)}{D})) \\ &+ c_s(t+t_k) \cdot K_s(H \cdot N_f)^{n_\delta} \frac{(D-t)}{D}] \cdot \tau(t+t_k) \cdot e^{-\int_{t_k}^{t} \tau(u)du} dt \\ &+ \int_0^D [c(t+t_k) \cdot (K_a + K_d(N_b \cdot L)\frac{t}{D}) \\ &+ c_s(t+t_k) \cdot K_s(H \cdot N_b)^{n_\delta} \frac{t}{D}] \cdot \tau(t+t_k) \cdot e^{-\int_{t_k}^{t} \tau(u)du} dt \end{aligned} \tag{5.3}$$

Notons que c'est l'éclairage qui est interpolé pour la partie spéculaire et non pas le gradient. Ceci permet, moyennant une erreur plus importante, de s'affranchir de la puissance n_δ pour le terme linéaire.

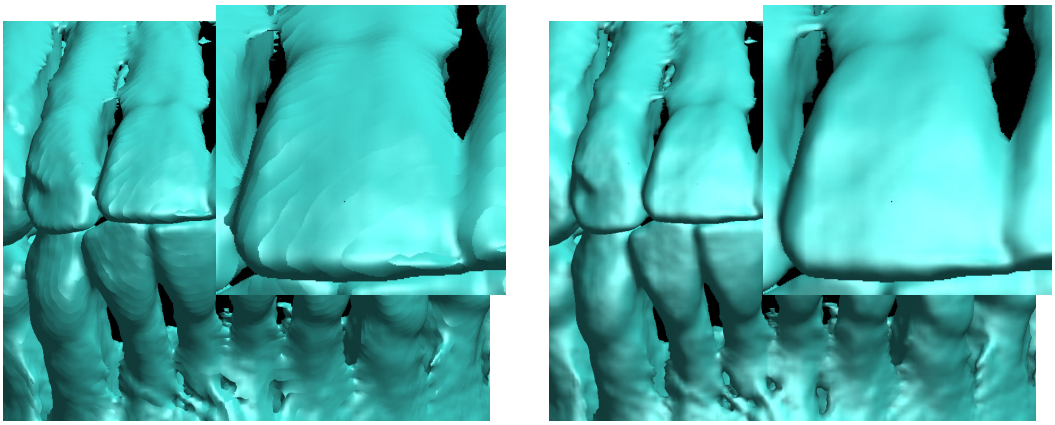


FIGURE 5.1 – Comparaison d'éclairage en rendu volumique direct pré-intégré selon ENGEL ET AL. [EKE01] à gauche et selon LUM ET AL. [LWM04] à droite.

Ils créent donc deux paires de tables de pré-intégration : une paire pour la composante diffuse et une paire pour la composante spéculaire. La paire pour la composante diffuse est créée en utilisant la couleur et l'opacité de la fonction de transfert. La paire spéculaire utilise seulement l'opacité de la fonction de transfert, et remplace la couleur par du blanc. Pour chaque paire, la couleur de la première table est pondérée linéairement de manière décroissante depuis la première valeur de densité vers 0. La couleur de la seconde table est pondérée linéairement de manière croissante depuis 0 vers la seconde valeur de densité. La somme des couleurs des deux tables correspond ainsi à la table de pré-intégration classique.

L'éclairage peut être calculé séparément pour chaque paire d'échantillon (s_f, s_b), car $N_f \cdot L$ et $N_b \cdot L$ sont constants et peuvent sortir de l'intégrale de l'équation 5.3. En pratique le calcul lors du rendu se fait ainsi : On récupère pour s_f, la couleur pondérée vers le premier échantillon dans les tables diffuse Kd_f et spéculaire Ks_f correspondantes, et pour s_b, la couleur pondérée vers le second échantillon dans les tables diffuse Kd_b et spéculaire Ks_b correspondantes. On récupère les gradients en ces deux points N_f et N_b, puis on calcule les contributions en couleur avec éclairage C_f et C_b en utilisant Kd_f, Ks_f et N_f pour C_f et Kd_b, Ks_b et N_b pour C_b. La somme de C_f et C_b donne la couleur finale pour la tranche (s_f, s_b). On combine ensuite les échantillons en utilisant la valeur d'opacité calculée avec la méthode de pré-intégration classique, dans la mesure où l'opacité n'est pas affectée par l'éclairage.

Cette technique revient à interpoler linéairement l'éclairage entre les deux échantillons de manière similaire à l'ombrage de Gouraud. Si l'angle entre les gradients de ces deux échantillons est important, cela peut causer des erreurs d'éclairage. C'est ce que nous proposons de corriger dans la suite.

La figure 5.1 compare l'approche utilisant un gradient calculé comme la moyenne des gradients des deux échantillons (à gauche) avec la méthode de LUM ET AL. [LWM04] (à droite), utilisant une interpolation linéaire de l'éclairage entre paires d'échantillons successifs. L'approche de LUM ET AL. [LWM04] permet de supprimer les discontinuités d'éclairage qui apparaissent d'une tranche à l'autre par rapport à la méthode proposée par ENGEL ET AL. [EKE01].

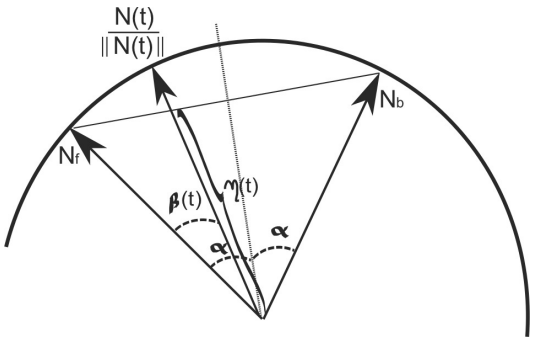


FIGURE 5.2 – Évaluation du terme de normalisation $\eta(t)$ entre N_f et N_b.

5.4 Interpolation non-linéaire du gradient entre les échantillons

La méthode de LUM ET AL. [LWM04] propose d'interpoler linéairement l'éclairage entre deux échantillons successifs. Le point de départ est une interpolation linéaire des gradients. Cependant, comme N_f et N_b sont normalisés, la norme du vecteur résultant d'une interpolation linéaire sera toujours inférieure à 1 (sauf en $t = 0$ et $t = D$). Pour corriger cela, on introduit un terme de normalisation $\eta(t) = \|N(t)\|$ pendant l'intégration de l'échantillon avant vers l'échantillon arrière en divisant $N(t)$ par $\eta(t)$. L'équation corrigée est donc :

$$\begin{aligned} \tilde{C} &= \int_0^D [c(t+t_k) \cdot (K_a + K_d(\frac{N_f \cdot (D-t) + N_b \cdot t}{D \cdot \eta(t)} \cdot L)) \\ &+ c_s(t+t_k) \cdot K_s(H \cdot \frac{N_f \cdot (D-t) + N_b \cdot t}{D \cdot \eta(t)})^{n_\delta}] \\ &\cdot \tau(t+t_k) \cdot e^{-\int_{t_k}^{t} \tau(u)du} dt \end{aligned} \tag{5.4}$$

Supposer que l'on interpole linéairement le gradient le long du segment entre les deux échantillons revient à considérer que $\eta(t)$ est toujours égal à 1. Or cette approche introduit une perte d'énergie, le gradient sera en fait interpolé le long de la ligne reliant N_f à N_b, et non le long de l'arc de cercle (cf. Figure 5.2). La perte d'énergie résultante est proportionnelle à l'angle entre N_f et N_b : l'erreur augmente avec la diminution de la valeur du produit scalaire $N_f \cdot N_b$, c'est-à-dire avec l'augmentation de l'angle formé par N_f et N_b. La Figure 5.2 illustre cette perte d'énergie, où le vecteur interpolé ne suit pas l'arc de cercle entre les deux gradients. Cette erreur est d'autant plus importante dans le cas d'une composante à haute fréquence avec l'éclairage spéculaire par exemple.

En divisant $N(t)$ par $\eta(t)$, le vecteur interpolé suit l'arc de cercle entre les deux gradients N_f et N_b, ce qui correspond donc à une interpolation non-linéaire du gradient. La valeur maximale de $\eta(t)$ est atteinte pour le demi-vecteur entre N_f et N_b. On peut calculer $\eta(t)$ par l'expression suivante :

$$\eta(t) = \frac{\cos(\alpha)}{\cos(\beta(t) - \alpha)}$$

avec $\alpha = \arccos(N_f \cdot N_b)/2$, l'angle entre N_f, N_b et leur demi-vecteur. $\beta(t)$ varie linéairement de 0 à 2α avec l'interpolation linéaire, d'où $\beta(t) = 2\alpha t/D$.

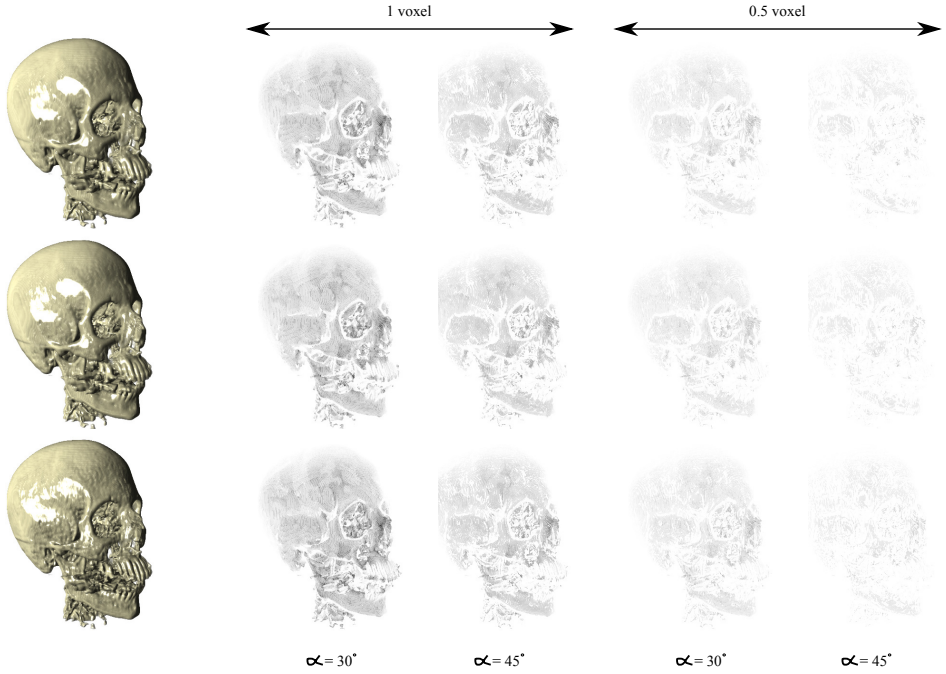


FIGURE 5.3 – Variation des gradients entre 2 échantillons consécutifs pour le jeu de données CT Head en utilisant 3 méthodes de calcul des gradients et deux pas d'échantillonnage différents.

Pour estimer s'il est utile d'introduire une nouvelle méthode d'interpolation du gradient entre paires d'échantillons, nous devons d'abord évaluer quel est l'ordre de grandeur des valeurs des angles entre échantillons consécutifs pour un certain échantillonnage du lancer de rayon. La Figure 5.3 montre cette évaluation pour le jeu de données "CT Head". La variation des angles dépend principalement de la technique utilisée pour calculer le gradient, ainsi que du pas d'échantillonnage (plus ce pas sera important, plus la possibilité que les gradients soient différents entre le premier et second échantillon d'une paire est grande). Sur cette figure, les trois lignes montrent des résultats pour trois méthodes de calcul du gradient différentes. La colonne de gauche montre une image de rendu du jeu de données. Les colonnes du milieu et de droite montrent la variation du gradient le long du rayon pour des pas d'échantillonnage de respectivement 1 et 0.5 voxels, et pour des angles α plus grands que 30 et 45 degrés respectivement (cela signifie respectivement des angles de 60 et 90 degrés entre le premier et second échantillon d'une paire). Les zones sombres indiquent le nombre d'échantillons ayant de tels angles. Plus les zones sont sombres, plus on trouve d'échantillons consécutifs avec des angles respectivement supérieurs à 60 et 90 degrés le long du rayon. La première ligne utilise des gradients calculés avec un algorithme de Sobel (différences centrales). La seconde ligne illustre l'utilisation d'une interpolation cubique. La dernière ligne illustre un éclairage exagéré selon les travaux de RUSINKIEWICZ ET AL. [RBD06]. Dans notre cas, pour exagérer la perception des détails, nous avons augmenté pour chaque échantillon l'angle entre le gradient calculé et un gradient moyen dans un voisinage donné autour de cet échantillon. On note N_m le gradient moyen autour de l'échantillon (i, j, k) dans un certain voisinage et N le gradient en (i, j, k), on remplace ce dernier par $N' = N + \epsilon(N - N_m)$,avec $\epsilon \geq 0$, l'amplitude de l'exagération (0 signifiant aucune exagération). Pour l'exemple en Figure 5.3, nous avons utilisé $\epsilon = 1$ et un voisi-

nage de 5^3 voxels. L'éclairage exagéré améliore la perception des petits détails et augmente la variation du gradient entre le premier et second échantillon d'une paire considérée. Tous ces exemples montrent que les jeux de données peuvent contenir des variations du gradient localement importantes entre le premier et second échantillon d'une paire (ces variations sont plus accentuées quand le pas d'échantillonnage est grossier, ou que les détails sont exagérés). Pour de tels cas, il semble important d'appliquer une interpolation correcte du gradient pour éviter d'introduire des erreurs dans l'éclairage.

Pour intégrer une variation non-linéaire des gradients, on peut ajouter une dimension supplémentaire à la table de pré-intégration sous la forme de l'angle α utilisé. Ce qui résulte en une table de pré-intégration de dimension 3. Ces tables ne sont pas pratiques à mettre à jour en temps réel. Elles sont de plus consommatrices d'espace mémoire, alors que le jeu de données accompagné de ses gradients peuvent déjà occuper beaucoup d'espace.

Pour conserver les tables de pré-intégration en 2 dimensions, nous proposons d'exploiter le fait qu'une normale correctement interpolée se déplace le long d'un cercle, comme exposé en Figure 5.2. Un arc de cercle pouvant être approché par un polynôme du second degré, nous proposons d'utiliser l'approximation suivante pour $\eta(t)$:

$$\begin{aligned} \eta(t) &= \frac{\cos(\alpha)}{x^2(\cos(\alpha)-1)+1} \\ x &= \frac{(\beta(t)-\alpha)}{\alpha} \\ &= \frac{2t}{D}-1 \end{aligned} \tag{5.5}$$

Avec cette formule, on a :

$$\begin{cases} t=0 & \Rightarrow \quad \beta{=}0, \eta{=}1 \\ t=0.5D & \Rightarrow \quad \beta{=}\alpha, \eta{=}\cos(\alpha) \\ t=D & \Rightarrow \quad \beta{=}2\alpha, \eta{=}1 \end{cases}$$

Les résultats correspondent à ceux attendus pour le facteur de normalisation $\eta(t)$. En utilisant ce terme de normalisation, on minimise l'erreur en se rapprochant de l'arc de cercle par rapport à l'interpolation linéaire ($\eta = 1$).

En utilisant la formule 5.5, le calcul du gradient normalisé entre le premier et le second échantillon d'une couche devient :

$$\frac{\nabla g(t)}{\|\nabla g(t)\|} \approx \frac{((2\frac{t}{D}-1)^2(\cos(\alpha)-1)+1)(\frac{\nabla g(0)}{\|\nabla g(0)\|}t + \frac{\nabla g(D)}{\|\nabla g(D)\|}(D-t))}{D\cdot\cos(\alpha)} \tag{5.6}$$

Sur trois échantillons successifs, la Figure 5.4 représente trois cas avec différents gradients. Pour chaque cas, trois bandes (a), (b) et (c) illustrent l'éclairage le long du rayon en utilisant une méthode d'interpolation du gradient différente à chaque fois. Ainsi, les différentes bandes présentent respectivement :

(a) l'interpolation linéaire de l'éclairage telle qu'elle est réalisée par Lum et al. [LWM04],

(b) le gradient précisément interpolé et normalisé : l'image de référence,

(c) notre approche d'interpolation non-linéaire avec un courbe quadratique.

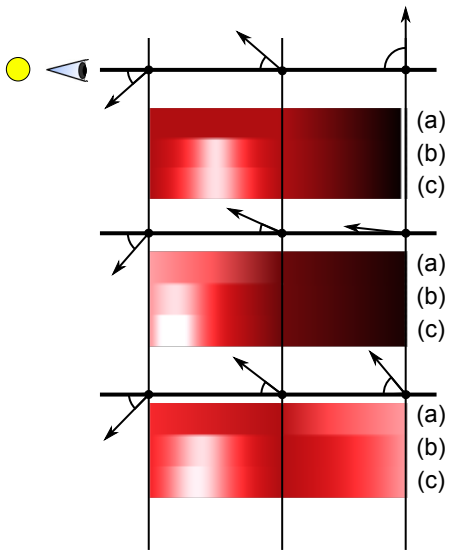


FIGURE 5.4 – Comparaison des résultats pour différentes techniques d'interpolation entre différents échantillons. Les bandes (a), (b) et (c) représentent respectivement l'interpolation linéaire du gradient, l'interpolation précise et normalisée du gradient et la technique que nous proposons, basée sur une interpolation non-linéaire.

Le premier et le dernier exemple illustrent le fait que le reflet spéculaire est absent en utilisant la méthode basée sur une interpolation linéaire de l'éclairage. Notre méthode (c) est plus proche de la bande de référence (b), mais le reflet est plus large, ce qui est dû à la formulation quadratique. Pour le second exemple, le reflet spéculaire est placé au bon endroit dans les trois cas, mais notre méthode donne encore un résultat plus proche de l'image de référence. L'interpolation quadratique du gradient permet donc de réduire les erreurs d'éclairage et d'obtenir des reflets spéculaires plus proches de l'interpolation correcte.

La Figure 5.5 illustre l'erreur commise sur le gradient par l'interpolation linéaire (en vert) et notre méthode (en rouge). L'erreur est donnée en fonction de la position t sur l'intervalle entre les deux échantillons et la valeur de l'angle α. On constate effectivement que pour l'interpolation linéaire et donc sans normalisation du gradient, l'erreur devient maximale lorsqu'on atteint la moitié de la valeur de l'angle (cf. la courbe verte). En utilisant une normalisation basée sur la formulation quadratique proposée, l'erreur est beaucoup moins importante et même nulle au milieu de l'intervalle (cf. la courbe rouge). Pour un angle α de 45°, l'erreur maximale est de 0.3% pour l'interpolation quadratique par rapport aux 41% de l'interpolation linéaire. En effet, au demi-vecteur on a $1/\eta = 1/\cos(\pi/4) = 1.414$.

5.5 Pré-intégration avec interpolation non-linéaire du gradient

Dans cette section, nous commençons par expliquer comment intégrer l'interpolation non-linéaire de la section précédente dans le cadre du calcul de la partie diffuse et spéculaire de l'éclairage de Blinn-Phong. Finalement, nous présentons l'utilisation de cette interpolation dans le cadre d'un autre modèle d'éclairage : l'éclairage non-photoréaliste de Gooch. On considère dans la suite que le terme ambiant est

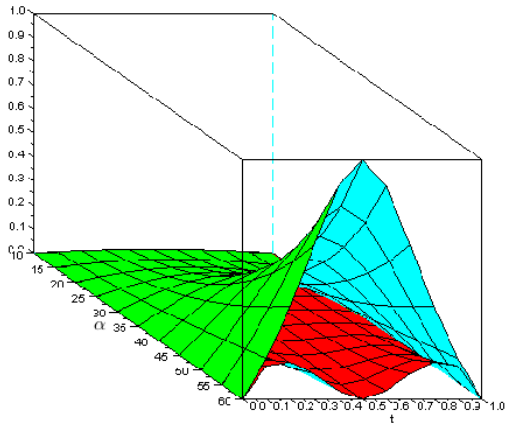


FIGURE 5.5 – Graphes représentant l'erreur commise sur le gradient par l'interpolation linéaire (en vert) et la méthode d'interpolation non-linéaire proposée (en rouge). L'erreur est donnée en fonction de la position t entre les deux échantillons et la valeur de l'angle α.

obtenu à l'aide d'une table de pré-intégration simple, dans la mesure où le calcul de la composante ambiante de l'éclairage ne dépend pas du gradient.

5.5.1 Partie diffuse

En ne considérant que la partie diffuse de l'équation de Blinn-Phong décrite avec l'équation 5.3 et le modèle d'interpolation non-linéaire du gradient (Formule 5.6), la composante diffuse pour une tranche d'épaisseur D devient :

$$\begin{aligned}\tilde{C}_d &= K_d \int_0^D [c(t+t_k)\frac{(2\frac{t}{D}-1)^2(\cos(\alpha)-1)+1}{\cos(\alpha)} \\ &\quad \cdot\frac{(N_b t + N_f(D-t))}{D}\cdot L)]\cdot\tau(t+t_k)\cdot e^{-\int_{t_k}^{t}\tau(u)du}dt\end{aligned}$$

Par substitution de variable $s' = t(s_b - s_f)/D + s_f$, $ds' = (s_b - s_f)/D \cdot dt$, ce qui suppose une variation linéaire de la densité entre les échantillons (comme d'habitude pour la pré-intégration), on replace l'équation dans l'intervalle des échantillons $[s_f, s_b]$:

$$\begin{aligned}\tilde{C}_d(s_f,s_b) &= K_d d \int_{s_f}^{s_b} [c(s')\frac{(2s-1)^2(\cos(\alpha)-1)+1}{\cos(\alpha)} \\ &\quad \cdot(N_b\cdot Ls + N_f\cdot L(1-s))]\cdot\tau(s')\cdot e^{-d\int_{s_f}^{s'}\tau(u)du}ds' \qquad (5.7)\end{aligned}$$

et $s = (s' - s_f)/(s_b - s_f)$.

En utilisant la notation suivante avec k, exposant de s, et en notant :

$$I_k(s_f,s_b) = d\int_{s_f}^{s_b} c(s')\cdot s^k\cdot\tau(s')\cdot e^{-d\int_{s_f}^{s'}\tau(u)du}ds'$$

Puis en séparant les termes constants, linéaires (s), quadratiques (s^2) et cubiques (s^3), on obtient l'équation suivante :

$$\begin{aligned}\tilde{C}_d(s_f, s_b) &= K_d(a_0(L)I_0(s_f, s_b) + a_1(L)I_1(s_f, s_b) \\ &+ a_2(L)I_2(s_f, s_b) + a_3(L)I_3(s_f, s_b)) \end{aligned} \tag{5.8}$$

avec :

$$\begin{aligned} a_0(L) &= N_f \cdot L \\ a_1(L) &= N_b \cdot L - (1+r)N_f \cdot L \\ a_2(L) &= r(2N_f \cdot L - N_b \cdot L) \\ a_3(L) &= r(N_b \cdot L - N_f \cdot L) \\ r &= \frac{4(\cos(\alpha)-1)}{\cos(\alpha)} \end{aligned}$$

Tous les termes I_k de cette équation ne dépendent que de s_f et s_b, ils peuvent par conséquent être pré-calculés sous la forme de tables 2D. Pendant la phase de rendu, on accède aux quatre tables correspondant aux termes I_k, les constantes a_k sont calculées et on obtient finalement la valeur de la composante diffuse en utilisant la formule 5.8. Cela permet d'avoir une erreur négligeable par rapport à l'interpolation le long de l'arc de cercle même pour un angle important entre N_f et N_b, de 90° par exemple.

Comme mentionné dans la section 5.4, l'équation 5.8 n'est pas utilisée telle quelle à cause des produits scalaires qui peuvent devenir négatifs et ainsi retirer de l'énergie. On doit soit utiliser un maximum, soit inverser le gradient pour obtenir des valeurs positives. Pour le premier cas, on doit introduire une fonction maximum $max(\_, 0)$ dans l'intégrale. Cette fonction n'étant pas triviale à intégrer, on préfère distinguer 3 cas :

- Si $N_f \cdot L > 0$ et $N_b \cdot L > 0$, on peut utiliser la formule sans maximum.
- Si $N_f \cdot L < 0$ ou $N_b \cdot L < 0$, on recourt à une approximation qui consiste à augmenter ou diminuer progressivement la valeur non négative, c'est-à-dire que $(N_b \cdot Ls + N_f \cdot L(1-s))/\eta(t)$ est remplacé par $N_b \cdot L_s$ ou $N_f \cdot (1-s)$. On obtient respectivement $\tilde{C}(s_f, s_b) = K_d N_f \cdot L(I_0(s_f, s_b) - I_1(s_f, s_b))$ et $\tilde{C}(s_f, s_b) = K_d N_b \cdot LI_1(s_f, s_b)$.
- Si $N_f \cdot L < 0$ et $N_b \cdot L < 0$, le résultat est 0.

Pour le second cas, il suffit d'inverser le gradient responsable de la valeur négative et d'appliquer l'interpolation. Cela introduit néanmoins une approximation dans le cas où un des gradients (avant ou arrière) est correctement orienté et que le second est inversé par rapport au vecteur L. En effet, dans ce cas, l'interpolation correcte ne consiste plus en un arc de cercle, mais deux arcs de cercles se rejoignant à l'endroit où le gradient s'inverse. Appliquer une interpolation avec un gradient inversé n'est donc pas tout à fait correct et ajoute de l'énergie.

5.5.2 Partie spéculaire

Considérons maintenant la partie spéculaire de l'éclairage de Blinn-Phong. En partant des mêmes hypothèses que pour la partie diffuse, on a la formule suivante :

$$\begin{aligned}\tilde{C}_s(s_f, s_b) &= K_s d \int_{s_f}^{s_b} [\frac{(2s-1)^2(\cos(\alpha)-1)+1}{\cos(\alpha)} \\ &\cdot (N_b s + N_f(1-s)) \cdot H)]^{n_\delta} \cdot \tau(t) \cdot e^{-\int_{s_f}^{s} \tau(u)du} ds' \end{aligned} \tag{5.9}$$

La puissance n_δ rend l'évaluation numérique de cette intégrale délicate. Pour la simplifier, nous proposons de remplacer le lobe spéculaire par un cône spéculaire, en utilisant l'approximation suivante : $(1+a)^n = 1 + na + o(x)$. Cette approximation modifie légèrement la forme du reflet spéculaire sans modifier les apports visuels. En appelant $N \cdot H = \cos(\theta)$, on remplace $\cos(\theta)^{n_\delta}$ par $1 + n_\delta(\cos(\theta) - 1)$. La Figure 5.6 montre que cela n'affecte pas la perception des reflets spéculaires. La colonne de gauche représente l'utilisation du lobe spéculaire, celle de droite le cône spéculaire. On peut noter une différence sur une sphère parfaite (deuxième ligne), mais cela représente un cas particulier. En effet, dans le cas où on a des variations dans la surface, la différence visuelle devient quasiment imperceptible. L'avantage d'utiliser un cône est que cela permet une intégration plus aisée.

En utilisant un cône au lieu d'un lobe pour le terme spéculaire, on obtient :

$$\begin{aligned}\tilde{C}_s(s_f, s_b) &= K_s d \int_{s_f}^{s_b} [1 + n_\delta(\frac{(2s-1)^2(\cos(\alpha)-1)+1}{\cos(\alpha)} \\ &\quad \cdot(N_b s + N_f(1-s)) \cdot H - 1)] \cdot \tau(t) \cdot e^{-\int_{s_f}^{s} \tau(u)du} ds'\end{aligned}$$

Comme dans le cas diffus, on introduit la notation suivante :

$$I'_k(s_f, s_b) = d \int_{s_f}^{s_b} s^k \cdot \tau(s') \cdot e^{-d\int_{s_f}^{s'} \tau(u)du} ds'$$

En séparant les termes constants, linéaires (s), quadratiques (s^2) et cubiques (s^3), l'équation devient :

$$\begin{aligned}\tilde{C}_s(s_f, s_b) &= K_s(I'_0(s_f, s_b) + n_\delta(a_0(H)I'_0(s_f, s_b) + a_1(H)I'_1(s_f, s_b) \\ &+ a_2(H)I'_2(s_f, s_b) + a_3(H)I'_3(s_f, s_b)) - n_\delta I'_0(s_f, s_b) \qquad (5.10)\end{aligned}$$

Comme dans le cas diffus, on peut avoir des valeurs négatives en fonction des valeurs des produits scalaires et de n_δ. On peut donc choisir d'inverser les gradients ou d'utiliser une fonction maximum $max(\_, 0)$. Les valeurs deviennent négatives quand le produit scalaire entre H et le gradient interpolé $N(s')$ est inférieur à $seuil = (n_\delta - 1)/n_\delta$. Pour la fonction maximum, on distingue les 3 cas suivants :

- Si $N_f \cdot H > seuil$ et $N_b \cdot H > seuil$, les valeurs sont positives, on les utilise donc directement.
- Si $N_f \cdot H < seuil$ ou $N_b \cdot H < seuil$, on utilise une approximation qui consiste à augmenter ou diminuer progressivement la valeur positive, c'est-à-dire que l'on utilise $\tilde{C}_s(s_{f,}s_b) = K_s(I_0(1-n_\delta) + n_\delta N_b \cdot H I_1)$ ou $\tilde{C}_s(s_{f,}s_b) = K_s(I_0(1 + n_\delta(N_f \cdot H - 1)) - n_\delta N_f \cdot H I_1)$ respectivement.
- Si $N_f \cdot H < seuil$ et $N_b \cdot H < seuil$, le résultat ne sera pas 0 comme dans le cas diffus, puisqu'il peut y avoir une réflection entre le premier et second échantillon. Pour calculer la position t' où la valeur spéculaire maximale est atteinte, on peut utiliser la dérivée de la forme cubique, une équation quadratique que l'on va résoudre. On peut ainsi vérifier si la solution t' est dans $[0; 1]$ ou non, et calculer la valeur du terme spéculaire correspondant. Si il est positif, il y a un maximum pour t', et un reflet entre le premier et second échantillon. Un moyen de réaliser cette intégration serait d'augmenter la valeur spéculaire de 0 à t', puis de la diminuer de t' à 1. Cette solution n'est

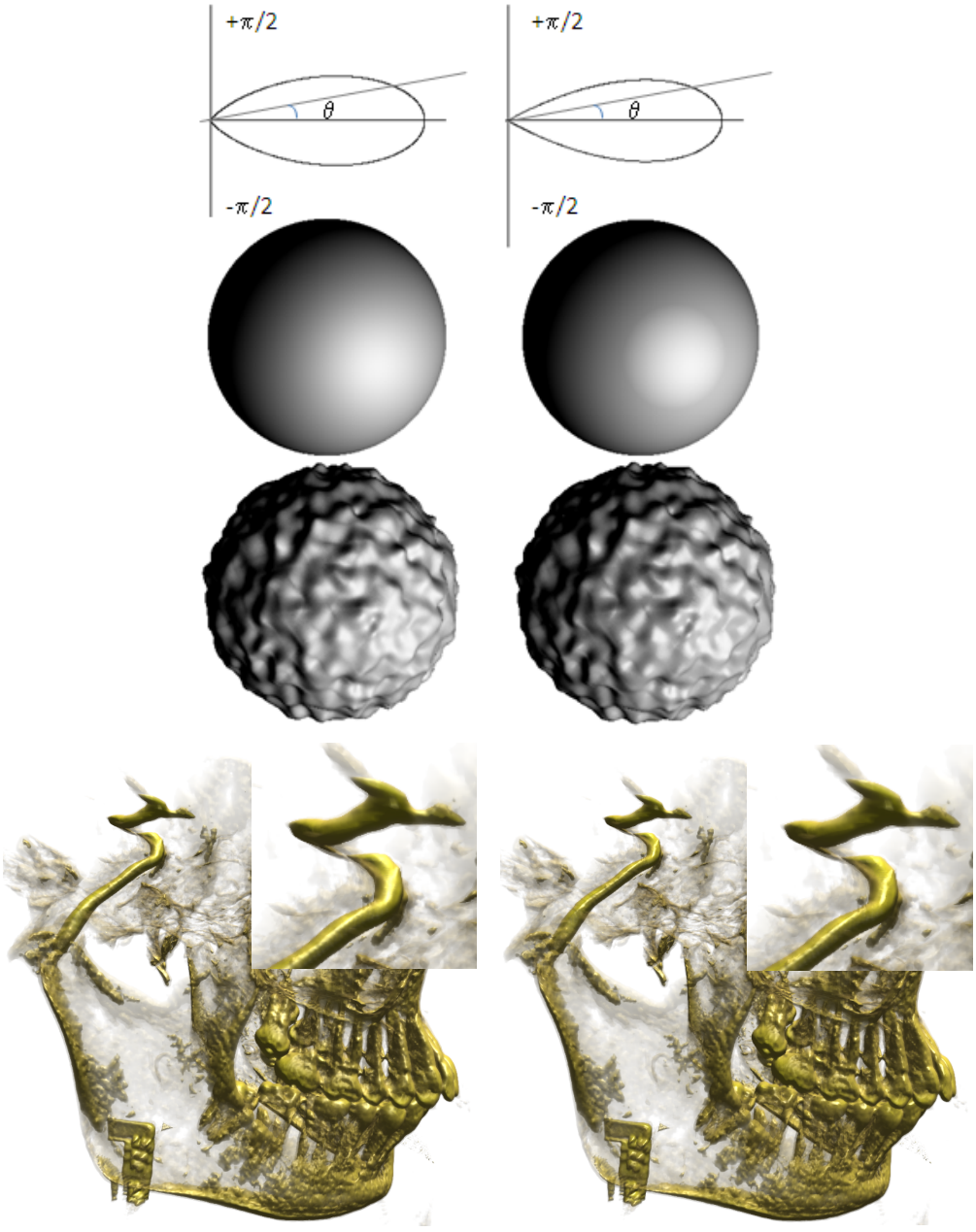


FIGURE 5.6 – Remplacement du lobe spéculaire (à gauche) par un cône spéculaire (à droite) pour $n_\delta = 10$. La différence visuelle sur la surface 3D reste minime, et les reflets spéculaires restent bien perceptibles.

néanmoins pas réalisable, car il faudrait autant de tables que de positions de t'. Nous proposons donc d'introduire une approximation supplémentaire sous la forme d'une table additionnelle I'' :

$$I''(s_f, s_b) = d\int_{s_f}^{s_b} (s < 0.5\,?\,2s\,:\,2-2s)\cdot\tau(s')\cdot e^{-d\int_{s_f}^{s'}\tau(u)du} ds'$$

Cette approximation part de l'hypothèse que la valeur spéculaire maximale est atteinte à mi-chemin entre le premier et second échantillon, c'est-à-dire pour $t' = 0.5$. Ceci permet d'être sûr d'avoir une contribution spéculaire entre les deux échantillons, mais pas forcément à la position exacte du reflet dans l'intervalle entre les échantillons.

5.5.3 Autres modèles d'éclairage

Nous avons montré ici l'exemple de l'intégration de l'interpolation non-linéaire du gradient pour l'éclairage de Blinn-Phong. On peut néanmoins appliquer cette interpolation à d'autres techniques d'éclairage utilisant le gradient, tant que celle-ci peut s'exprimer sous forme polynomiale. On peut par exemple prendre l'exemple de la technique de rendu expressif de Gooch et al. [GGSC98].

L'équation 5.1 est modifiée de la manière suivante pour suivre ce modèle :

$$\begin{aligned} I(t_k, t_{k+1}) &= \int_{t_k}^{t_{k+1}} [\frac{1}{2}(c_{blue} + a\cdot c(t)K_d)(1 + N\cdot L) \\ &+ \frac{1}{2}(c_{yellow} + b\cdot c(t)K_d)(1 - N\cdot L)]\cdot\tau(t)\cdot e^{-\int_{t_k}^{t}\tau(u)du} dt \end{aligned}$$

avec a et b, des coefficients pour contrôler l'importance de la couleur de l'objet, et c_{blue} et c_{yellow}, respectivement la couleur froide et chaude utilisée dans le modèle de Gooch.

Sous sa forme pré-intégrée, la partie diffuse de cette équation peut être écrite sous la forme suivante :

$$\begin{aligned} \tilde{C_d}(s_f, s_b) &= a_0(L)I_0^{Gooch}(s_f, s_b) + a_1(L)I_1^{Gooch}(s_f, s_b) \\ &+ a_2(L)I_2^{Gooch}(s_f, s_b) + a_3(L)I_3^{Gooch}(s_f, s_b)) \end{aligned}$$

avec :

$$\begin{aligned} I_k^{Gooch}(s_f, s_b) &= \frac{d}{2}\int_{s_f}^{s_b} (c_{blue} - c_{yellow} + K_d c(s')(a-b)) \\ &\cdot s^k\cdot\tau(s')\cdot e^{-d\int_{s_f}^{s'}\tau(u)du} ds' \end{aligned}$$

Notons que pour ce modèle d'éclairage, les valeurs résultantes sont toujours positives. Les valeurs du produit scalaire $N \cdot L$ ne doivent donc pas être traités spécialement.

5.6 Résultats

Les résultats ont été produits avec une implémentation OpenGL et GLSL de la méthode. La configuration utilisée est un Intel Quad Core Q9300 avec 4 Go de RAM et une NVIDIA GeForce GTX 280 (avec 1Go de mémoire). Le rendu volumique direct est fait sous la forme d'un lancer de rayons GPU sans optimisation excepté la terminaison de rayon. Les tables de pré-intégration utilisent une précision de 16 bits. Dans la suite, nous présentons des comparaisons d'images avec les techniques existantes.

La Figure 5.7 montre les résultats obtenus pour différents jeux de données et différentes méthodes de pré-intégration. De gauche à droite, les jeux de données utilisés sont deux genoux ($379 \times 229 \times 305$), une tête ($256 \times 256 \times 225$) et un bonsai ($256 \times 256 \times 256$). Les paramètres d'éclairages utilisés pour le modèle de Phong sont $Ka = 0$, $Kd = 1$, $Ks = 2$, $n_\delta = 50$ pour le jeu de données du bonsai et $Ka = 0$, $Kd = 1$, $Ks = 3$, $n_\delta = 80$ pour ceux de la tête et les genoux. L'ensemble des exemples présentés utilisent un cône au lieu d'un lobe pour le reflet spéculaire. Les lignes illustrent respectivement les techniques suivantes :

(a) L'image de référence, produite sans pré-intégration avec une interpolation correcte du gradient entre échantillons en utilisant un échantillonnage élevé,

(b) Le gradient calculé comme la moyenne des gradients du premier et second échantillon de l'intervalle,

(c) Le gradient interpolé linéairement, selon la méthode de Lum et al. [LWM04],

(d) Notre approche : l'interpolation non-linéaire du gradient.

Cette figure montre des gros plans, ainsi qu'une image de différence par rapport à l'image de référence (a) pour les cas (b), (c) et (d) : une faible différence résulte en une couleur noire. Comme illustré dans la Figure, (c) et (d) sont plus proches de l'image de référence que (b). Avec notre approche (d), la différence est nulle presque partout (les images de différence sont presque complètement noires). Il y a cependant des différences visibles notamment sur le jeu de données des genoux. En effet, notre approche tend a exagérer l'intensité autour des reflets spéculaires. Ceci est du au fait que la courbe de degré 2 utilisée pour l'interpolation n'est pas exactement un arc de cercle. Le terme de normalisation peut être trop petit, ce qui résulte en des vecteurs ayant des normes supérieures à 1 et donc un ajout d'énergie. Le rendu réalisé ici se base sur un pas d'échantillonnage de 2 voxels pour le bonsai et d'un voxel pour la tête et les genoux. Globalement les différences entre les 3 méthodes de calcul du gradient sont difficilement perceptibles. Les zooms permettent cependant de mettre en valeur trois bénéfices de l'interpolation non-linéaire du gradient. Le premier est qu'en utilisant un grand pas d'échantillonnage, par exemple 2 voxels pour le bonsai, les bandes sont moins évidentes grâce à la variation plus douce de l'éclairage. Notons que la pré-intégration n'enlève pas nécessairement toutes les bandes visibles, surtout dans le cas d'un grand pas d'échantillonnage. La variation plus douce de l'éclairage permet de réduire la visibilité de ces bandes, comparé à l'utilisation d'un gradient constant ou de l'interpolation linéaire du gradient. Un second bénéfice de cette méthode (deuxième colonne, jeu de données de la tête) est qu'elle permet de réduire un bruit qui apparaît quand la fréquence des reflets spéculaires est importante (petits points brillants), ceci est aussi du à la variation plus douce de l'éclairage. Finalement, le dernier bénéfice est que les contours des reflets spéculaires sont plus précisément définis. Ce point est important pour une bonne compréhension visuelle de la forme des objets sous-jacents. Concernant l'impact sur le temps de rendu, notre approche

est de 10 à 20 % plus lente que l'interpolation linéaire du gradient. Ceci s'explique par le fait que l'on doive réaliser deux accès textures supplémentaires : seules deux tables supplémentaires sont utilisées. On considère ici que l'on utilise la couleur blanche pour le reflet spéculaire, on peut donc utiliser un unique canal de couleur et placer les 3 couleurs de la composante diffuse plus celle de la composante spéculaire dans une même texture à 4 canaux.

Pour mieux percevoir la différence entre les trois méthodes de pré-intégration, nous avons utilisé un jeu de données contenant un objet de faible courbure, de sorte que la lumière varie doucement, mettant en valeur les erreurs dans l'éclairage. Le jeu de données du moteur ($256 \times 256 \times 128$) correspond à ce critère. La Figure 5.8 est une étude comparative sur ce jeu de données. L'image (a) est l'image de référence obtenue avec un pas d'échantillonnage de 0.1 voxel sans pré-intégration. Les images sont respectivement : (b) la variation du gradient, (c) notre méthode, (d) l'interpolation linéaire de l'éclairage de Lum et al. [LWM04] et (e) le gradient moyen. La première colonne illustre un pas d'échantillonage d'un voxel, la seconde un pas d'un demi voxel. Dans ce cas, les différences entre les différentes méthodes de pré-intégration sont mieux perceptibles. Ces différences diminuent avec l'augmentation de la fréquence d'échantillonnage, comme la variation des gradients devient plus petite (ceci est dû à l'interpolation trilinéaire sur les voxels). Même avec un pas d'un demi-voxel, notre approche (c) reste plus proche de l'image de référence (a) (voir le coin haut du moteur).

La Figure 5.9 montre deux exemples de rendu expressif avec la méthode d'éclairage de Gooch et al. [GGSC98] en utilisant deux jeux de données : celui du haut est un anévrisme ($256 \times 256 \times 256$) et celui du bas est la tête ($256 \times 256 \times 225$). La colonne de gauche montre l'image de référence et celle de droite le résultat obtenu avec notre méthode et un pas d'échantillonnage d'un demi-voxel. Quasiment aucune différence n'est perceptible entre notre résultat et l'image de référence. Pour l'anévrisme, l'image de référence est visualisée à 1.8 image par seconde à une fréquence d'échantillonnage de 0.1 voxel et notre résultat (à droite) est visualisé à 16.3 images par seconde pour une fréquence d'échantillonnage d'un demi voxel. Concernant le jeu de données de la tête, la visualisation s'effectue à 3.3 et 18.7 images par seconde respectivement pour l'image de référence échantillonnée à 0.1 voxel et notre résultat échantillonné à un demi-voxel.

5.7 Conclusion

Dans ce chapitre, nous avons introduit une nouvelle méthode non-linéaire d'interpolation du gradient. Utilisée en conjonction avec la pré-intégration, elle permet d'améliorer la précision de l'éclairage entre chaque paire d'échantillon. Les améliorations sont visibles lorsqu'il y a de fortes variations des gradients ou la présence de reflets spéculaires. Pour le terme spéculaire de Phong, nous avons utilisé un cône spéculaire au lieu du lobe pour une intégration plus facile dans l'éclairage. D'un point de vue visuel, le cône est proche du lobe et permet de préserver des apports visuels similaires. Notre technique est aussi compatible avec d'autres techniques d'éclairage, tant qu'elles peuvent être exprimées sous forme polynomiale.

Notre approche requiert l'utilisation de tables supplémentaires par rapport à l'interpolation linéaire du gradient, ce qui résulte en des accès textures supplémentaires lors de la phase de visualisation. L'impact sur la performance est néanmoins minime. De plus, les tables restent bi-dimensionnelles, ce qui permet de les mettre à jour plus facilement en cas de modification de la fonction de transfert.

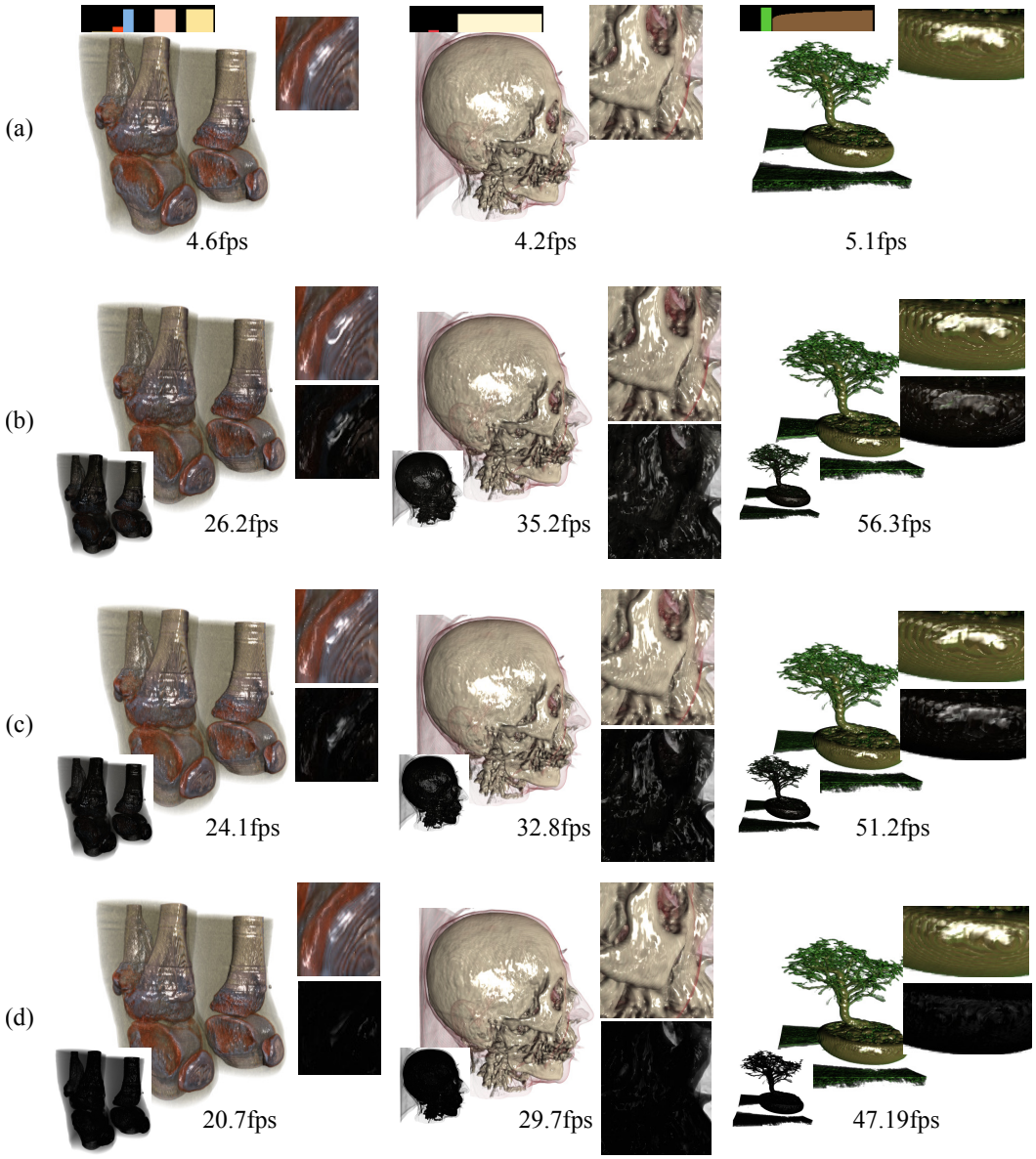


FIGURE 5.7 – Comparaison de différentes méthodes de pré-intégration : (a) interpolation du gradient précise en utilisant un sur-échantillonnage et pas de pré-intégration, (b) gradient calculé en utilisant la moyenne des gradients du premier et second échantillon, (c) interpolation linéaire du gradient et (d) notre approche.

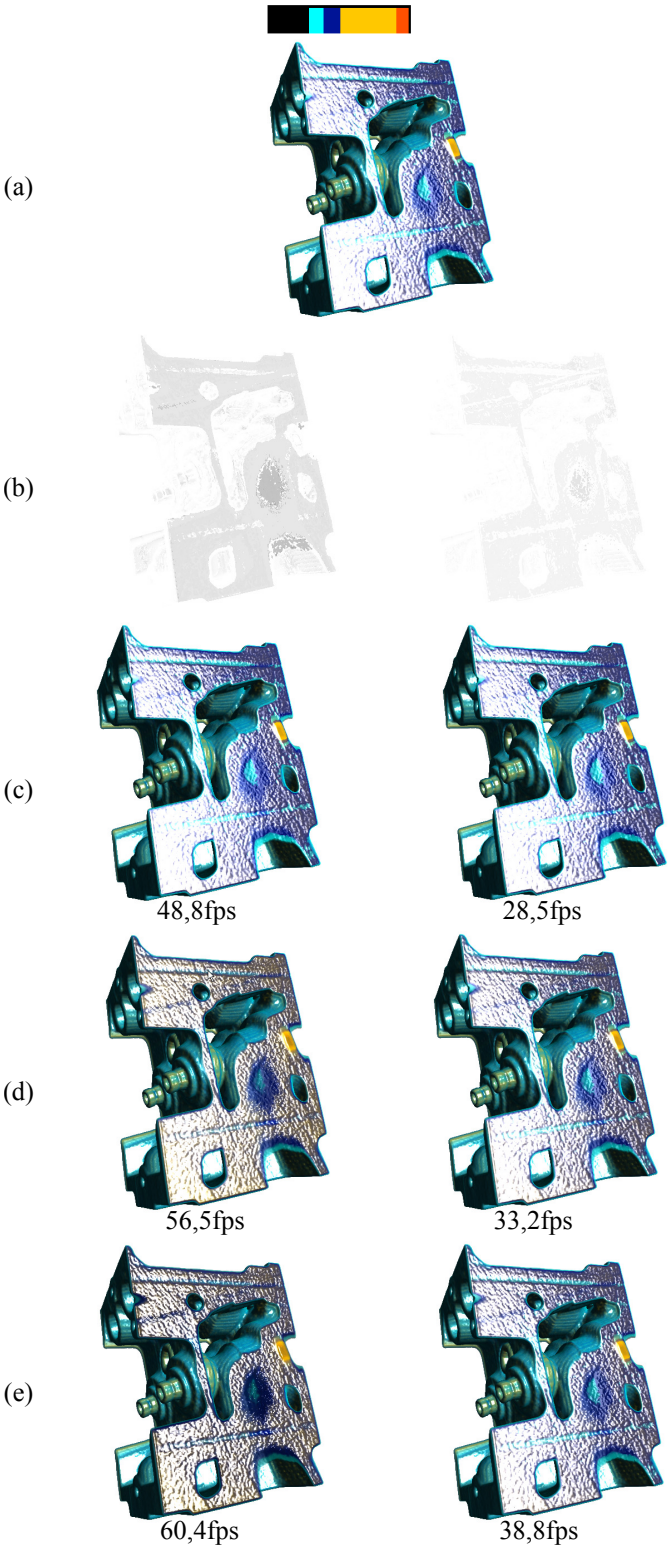


FIGURE 5.8 – Visualisation du jeu de données du moteur avec différentes méthodes de pré-intégration. La première colonne utilise un pas d'échantillonnage d'un voxel, la seconde un pas de 0.5 voxel.

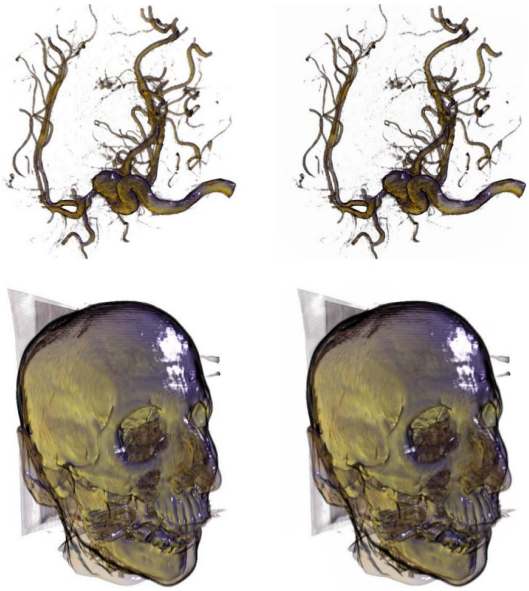


FIGURE 5.9 – Deux exemples de l'intégration de l'interpolation non-linéaire du gradient dans la méthode de rendu expressif de Gooch. La colonne de gauche présente l'image de référence (sans pré-intégration avec une fréquence d'échantillonnage importante), celle de droite l'éclairage expressif avec notre méthode d'interpolation et un pas d'échantillonnage d'un demi-voxel.

En utilisant une interpolation non linéaire du gradient, on augmente le degré de reconstruction du signal pour l'éclairage. Cette méthode permet ainsi de réduire les bandes de discontinuités dans l'éclairage et d'améliorer la variation de l'éclairage comparé à l'utilisation d'un gradient moyen ou à un gradient interpolé linéairement. Cependant, comme pour toutes les méthodes de pré-intégration, cette technique suppose que le signal sous-jacent est linéaire par morceau pour calculer l'occultation et les composantes de la couleur. Comme cela a déjà été montré dans des travaux précédents, il peut rester des bandes visibles dans le cas d'un pas d'échantillonnage trop grand. Notre approche permet de réduire la visibilité de telles bandes, mais ne les supprime pas. Pour améliorer encore les résultats, il faut améliorer la reconstruction du signal entre les paires d'échantillons. La problématique se porte alors plus sur l'augmentation de la dimension des tables de pré-intégration à utiliser.

Chapitre 6

Ambient Occlusion par structure pour rendu volumique direct

6.1 Introduction et problématique

Comme nous l'avons vu précédemment, l'éclairage est très important pour améliorer la perception des structures en rendu volumique. Il devient encore plus critique lorsqu'on a un ensemble de structures semi-transparentes qui s'entremêlent. Utiliser l'Ambient Occlusion dans ce contexte a de multiples intérêts :

- L'éclairage de Phong n'est pas toujours à même de donner des indices visuels suffisant pour une bonne interprétation des données, particulièrement en ce qui concerne la position relative des structures. Les techniques d'Ambient Occlusion apportent des ombres douces, testées empiriquement comme aussi bonne que la meilleure configuration en illumination directionnelle par Langer et al. [LB00].
- L'Ambient Occlusion ne dépend pas de normales/gradients et est donc moins touchée par le bruit potentiel dans les données.
- Elle requiert de configurer moins de paramètres d'éclairage qu'en illumination directe (nombre de sources lumineuses, positions des sources, directions, intensités, propriétés des matériaux, coefficients ambiant et diffus, ...). De plus, avec l'éclairage de Phong, l'utilisateur doit souvent changer les paramètres, en changeant de point de vue par exemple. L'Ambient Occlusion est indépendant du point de vue. Par conséquent, elle est plus accessible aux non-experts en visualisation, qui veulent visualiser les données sans avoir à configurer de nombreux paramètres.

Néanmoins, l'Ambient Occlusion est plus une technique utilisée dans le contexte de la visualisation par isosurfaces. Appliquer cette technique au rendu volumique direct n'est pas une question simple. En effet, comme en rendu volumique direct on considère le volume de données comme un milieu participatif, il n'y a pas à proprement parler de surface définie. L'information d'occultation est alors obtenue soit par l'utilisation des propriétés intrinsèques aux données (densités), soit par l'utilisation de la fonction de transfert (opacité). L'Ambient Occlusion d'un voxel v est alors évaluée en considérant le voisinage de ce voxel. Par commodité et simplicité, on choisit d'utiliser une demi-sphère centrée sur le gradient ou une sphère si on souhaite rendre l'Ambient Occlusion indépendant du bruit potentiellement présent dans les données.

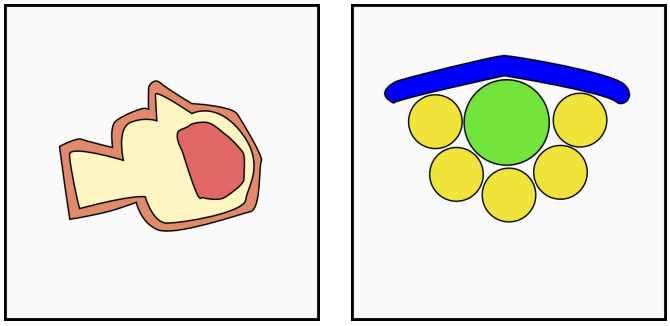


FIGURE 6.1 – Exemples de structuration des couches en rendu volumique direct.

En ayant recours au rendu volumique direct, on est donc susceptible d'obtenir un ensemble de structures volumiques imbriquées ou proches les unes des autres, comme les exemples illustrés en Figure 6.1.

Si on utilise sur une mauvaise paramétrisation ou si une grande partie des structures est rendue semi-transparente ou opaque, il va en résulter un assombrissement important de la visualisation. En effet, si on considère le modèle de gauche de la Figure 6.1, dans lequel on a plusieurs couches imbriquées les unes dans les autres, cela va résulter selon l'opacité des différentes couches en une perte de visibilité croissante vers le centre du jeu de données. Le modèle de droite sur la figure 6.1 illustre un autre cas où un ensemble de structures très proches les unes des autres vont s'entre-projeter des ombres de contact, ce qui induira une perte de visibilité sous la forme d'un assombrissement prononcé en ces lieux de projection.

La Figure 6.2 illustre le fait qu'augmenter la luminosité ne permet pas de réduire l'assombrissement. La première ligne présente respectivement l'utilisation de l'éclairage de Phong et de l'Ambient Occlusion. La seconde ligne montre la fonction de transfert utilisée. La troisième ligne illustre l'augmentation de luminosité ($\times 2$), la réduction de l'impact des facteurs d'occultation en les multipliant ($\times 2$), puis en ajoutant une constante (0.2), comme le proposent HERNELL ET AL. [HLY07]. On constate qu'aucun des trois cas n'améliore réellement la visibilité des structures internes quelques soient les valeurs utilisées. Pour modifier plus efficacement les facteurs d'Ambient Occlusion, il faut travailler plus en amont.

Toutefois, malgré ce défaut de luminosité (cf. Figure 6.2), on constate que l'Ambient Occlusion permet d'obtenir une meilleure perception de la position relative des structures dans les données. En effet, dans le cas de l'éclairage de Phong, on distingue l'os et la peau, mais leurs positions relatives sont plus difficiles à évaluer qu'en utilisant l'Ambient Occlusion. L'objectif visé est de permettre de conserver les apports visuels de l'Ambient Occlusion, mais de réduire l'effet d'assombrissement dû à la présence de multiples structures. Pour cela, nous ne souhaitons pas utiliser une approche naïve qui consisterait par exemple à modifier artificiellement les valeurs d'occultation par le biais de paramètres supplémentaires.

Dans la section 6.2, nous rappelons les détails théoriques de la méthode de HERNELL ET AL. [HLY07], puis nous discutons de détails d'implémentation de cette méthode. Nous détaillons ensuite notre méthode de calcul de l'Ambient Occlusion par structures en section 6.3. Nous discutons de la performance et des apports visuels de notre méthode dans la section 6.4.

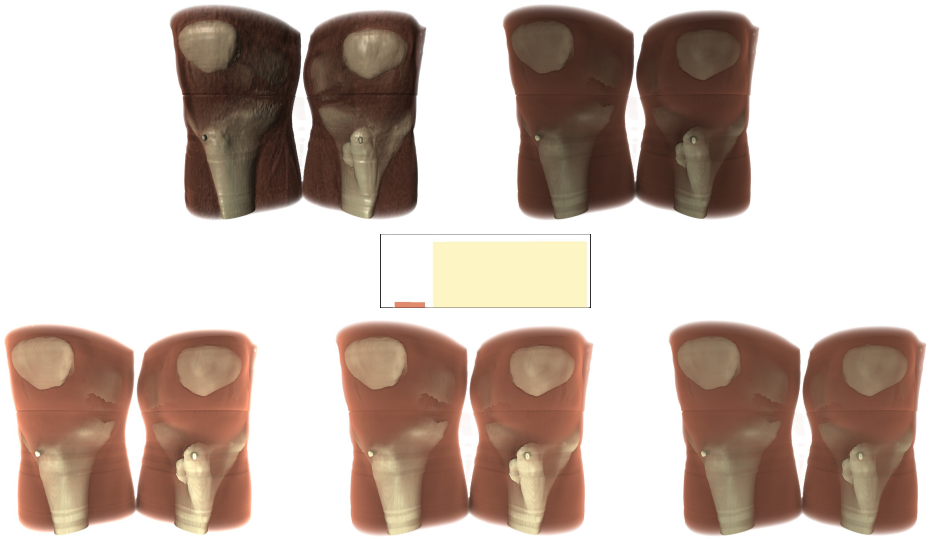


FIGURE 6.2 – La première ligne ligne illustre la visualisation d'un jeu de données avec éclairage de Phong et Ambient Occlusion. La seconde ligne montre la fonction de transfert utilisée. La troisième ligne illustre l'augmentation de luminosité ($\times 2$), la réduction de l'impact des facteurs d'occultation en les multipliant par une constante ($\times 2$), puis en ajoutant une constante (0.2), comme le proposent HERNELL ET AL. [HLY07].

6.2 Ambient Occlusion

Dans cette première section, nous rappelons les bases théoriques de la méthode proposée par HERNELL ET AL. [HLY07], puis nous discutons l'influence des différents paramètres inhérents à cette méthode. Nous concluons cette section par des détails d'implémentation.

6.2.1 Base théorique

Dans ce travail de thèse, nous nous sommes appuyés sur les travaux utilisant le lancer de rayon de HERNELL ET AL. [HLY07]. Pour cela, on considère un rayon de lumière arrivant au centre x_v d'un voxel v de la direction l avec la formule :

$$I_l(x_v) \quad = \quad \int_a^{R_\Omega} \frac{1}{R_\Omega - a} \cdot \exp\left(- \int_a^s \tau(u)\, du \right) ds \tag{6.1}$$

avec a, un décalage ("offset") initial, qui permet de réduire l'influence des plus proches voisins, R_Ω le rayon de la sphère d'influence considérée et τ la profondeur optique. Cette équation définit une région d'occultation sous la forme d'une sphère d'épaisseur $(R_\Omega - a)$ centrée sur chaque voxel. La formule est approchée en utilisant M échantillons pour une composition d'avant-en-arrière avec une opacité α_i, obtenue via la fonction de transfert :

$$I_l(x_v) \quad = \quad \sum_{m=0}^{M} \frac{1}{M} \prod_{i=0}^{m-1} (1 - \alpha_i) \tag{6.2}$$

Pour obtenir un facteur d'Ambient Occlusion par voxel v, il suffit donc d'effectuer un lancer de rayons pour un ensemble de rayons L, puis de combiner les contributions accumulées par rayon de la manière suivante :

$$AO(v) = \frac{1}{L}\sum_{l}^{L} I_l(x_v)$$

A chaque rayon lancé, sa contribution est ajoutée à celle des rayons précédents, puis normalisée en divisant la somme par le nombre de rayons lancés. Nous nous basons donc sur cette approche en utilisant une répartition pseudo-aléatoire des rayons basée sur les coordonnées polaires. Pour générer un rayon v aléatoirement selon une distribution équiprobable, nous utilisons la formule suivante :

$$\begin{aligned} \alpha &= 2 \times \pi \times rnd() \\ \beta &= \frac{\pi}{2} - \arccos(1 - 2 * rnd()) \\ v &= (\cos(\alpha) \times \cos(\beta), \sin(\beta), \sin(\alpha) \times \cos(\beta)) \end{aligned}$$

avec rnd(), une fonction renvoyant un nombre pseudo-aléatoire dans l'intervalle $[0; 1]$. Un rayon est donc généré et utilisé pour l'ensemble du jeu de données à chaque fois.

Le résultat obtenu en calculant l'Ambient Occlusion est un tableau tridimensionnel de même taille que les données initiales. Ce dernier contient pour chaque voxel une valeur réelle dans $[0.0; 1.0]$. Ces valeurs sont ensuite utilisées pour pondérer la couleur en chaque échantillon avec 0.0, pour un échantillon totalement occulté et 1.0, pour un échantillon totalement non occulté.

6.2.2 Influence des paramètres

Les différents paramètres précédemment évoqués ont une influence directe sur la précision et qualité de la méthode : le nombre de rayons lancés L, la distribution de ces rayons dans la sphère, l'offset initial a, le nombre de pas M et finalement le rayon de la sphère considérée R_Ω. Alors que les premiers sont intuitifs, car ils permettent de contrôler la précision et l'antialiasing associé à l'ombrage, le dernier est plus difficile à définir adéquatement, car il peut être responsable de sur-occultation s'il est choisi trop grand. S'il est à l'opposé choisi trop petit, cela résultera en un mauvais ombrage au sens où les ombres portées ne sont pas assez visibles, car on ne capture pas suffisamment d'information sur le voisinage. Pour résoudre ce problème de choix, Desgranges et Engel [DE07] ont par exemple proposé de choisir différentes valeurs pour le rayon de la sphère, appelé rayon d'influence, et de calculer l'Ambient Occlusion pour chacun d'eux. Puis ils combinent ces différents volumes en les pondérant. Cette solution risque cependant de mener à une moyenne des valeurs et ainsi de perdre une partie de l'information des différents rayons. De plus, elle requiert de définir des coefficients de pondération additionnels. C'est pourquoi nous nous sommes concentrés sur l'utilisation d'un seul rayon.

Le tableau 6.1 illustre l'évolution de l'ombrage pour une isosurface en fonction du rayon de la sphère d'influence choisi R_Ω et du nombre de rayons lancés. En dessous de chaque image est indiqué le temps de pré-calcul en secondes des facteurs d'occultation associés. L'augmentation du nombre de rayons lancés permet d'obtenir des ombres plus précises, alors que l'augmentation du rayon R_Ω permet de saisir plus

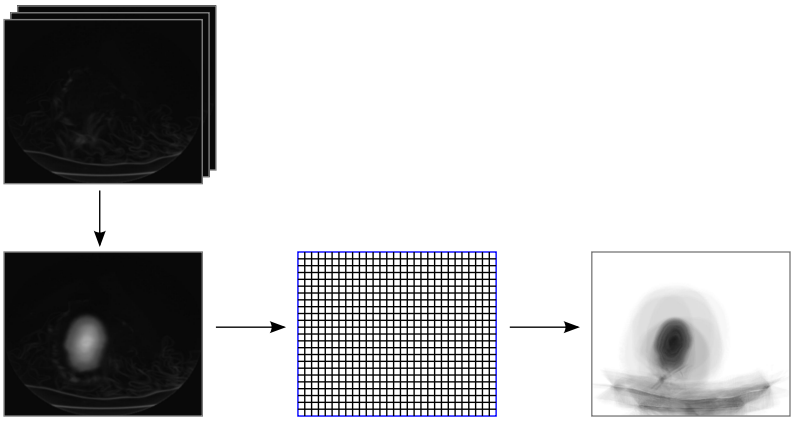


Figure 6.3: Génération des facteurs d'Ambient Occlusion en utilisant OpenGL.

d'informations sur le voisinage et d'accentuer les ombres ou d'obtenir des ombres de contact. En effet, on passe d'un ombrage quasiment directionnel pour 2 rayons lancés à un éclairage considérant un voisinage plus important et produisant des ombres plus douces à partir de 16 rayons. On distingue aussi mieux la position relative des différentes parties du jeu de données en passant d'un rayon R_Ω de 4 voxels à 32 voxels. Passer à 32 voxels permet aussi d'assombrir les cavités dans les structures et ainsi de mieux appréhender leur forme.

Le tableau 6.2 illustre aussi l'influence du rayon de la sphère d'influence R_Ω et du nombre de rayons lancés, mais dans le cas du rendu volumique direct. Comme pour le tableau précédent, les temps de pré-calcul des facteurs d'occultation sont indiqués en secondes en dessous des images. On constate ici que l'augmentation du nombre de rayons n'est pas facteur d'assombrissement de la visualisation. Par contre, le rayon R_Ω contribue beaucoup au sur-assombrissement de la visualisation. Il est en effet très difficile de choisir un rayon R_Ω adapté, car celui-ci dépend de la fonction de transfert utilisée et de la répartition des données effectivement affichées dans le jeu de données. On constate ainsi qu'à partir d'un rayon de 16 voxels et de 8 rayons lancés, les couches extérieures s'assombrissent et laissent moins entrevoir les structures internes.

6.2.3 Implémentation

La technique sur laquelle nous nous sommes basés pour ce travail a été introduite par Hernell et al. [HLY07]. A la base implémentée en OpenGL, nous en avons fait une version CUDA.

La version OpenGL est implémentée en utilisant des buffers de rendu hors écran 3D ("Framebuffer objects"). Le rendu est initié en dessinant sur l'espace écran un rectangle de la taille d'une tranche du buffer. Chaque pixel de ce rectangle est traité par un "fragment shader" et la valeur calculée est écrite dans le buffer hors écran. La figure 6.3 illustre ce processus. On extrait une tranche de données que l'on associe à l'espace écran en faisant correspondre un pixel à un voxel. Puis on obtient une image correspondant au calcul de l'Ambient Occlusion pour cette tranche.

En CUDA, on ne dispose plus de notion d'espace image et de buffer hors-écran. On peut cependant créer des tableaux 3D, destinés à recevoir les valeurs d'Ambient Occlusion. Cependant, on doit définir la distribution des différents voxels à calculer sur les multi-processeurs sous la forme de blocs/grilles CUDA, là où OpenGL gérait

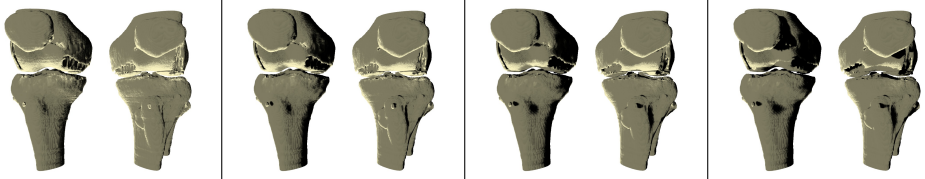
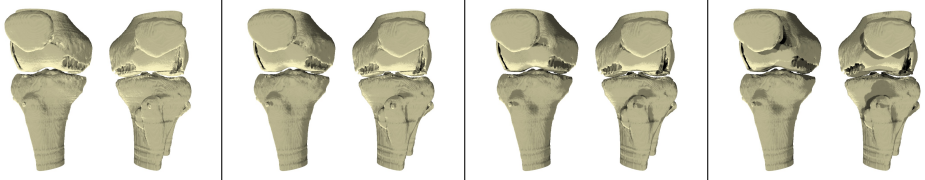
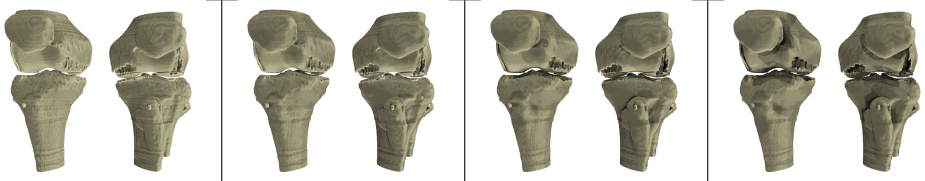
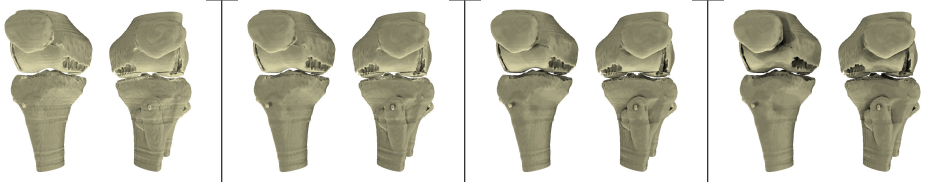
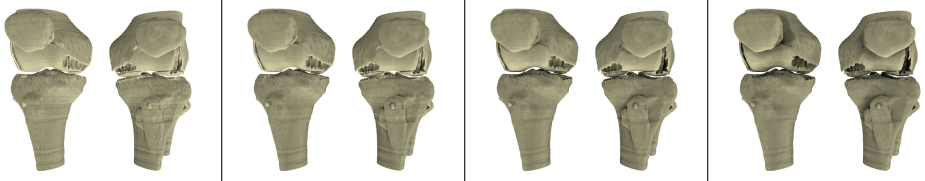
N	$R_\Omega = 4$	$R_\Omega = 8$	$R_\Omega = 16$	$R_\Omega = 32$
2				
	0.186	0.200	0.254	0.360
4				
	0.211	0.242	0.348	0.569
8				
	0.264	0.322	0.534	0.973
16				
	0.367	0.477	0.898	1.758
32				
	0.574	0.791	1.629	3.322

Table 6.1: Visualisation d'isosurface en faisant varier le rayon de la sphère d'influence (en colonne) de 0 à 32 voxels et le nombre de rayons lancés (en ligne) de 2 à 32. Le nombre associé à chaque image indique le temps de pré-calcul en secondes.

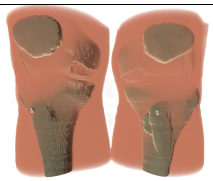
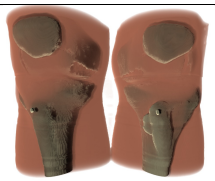
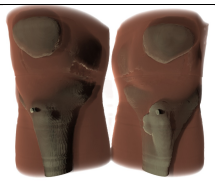
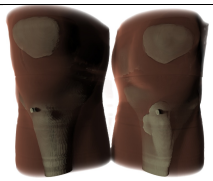
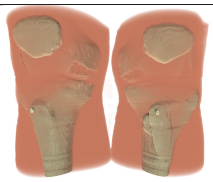
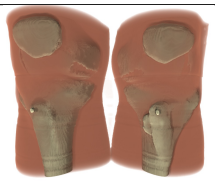
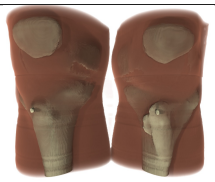
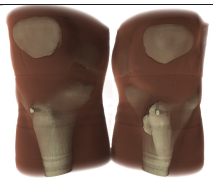
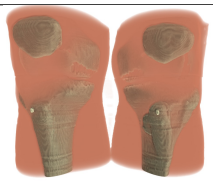
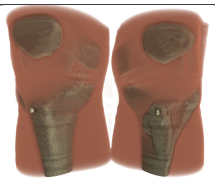
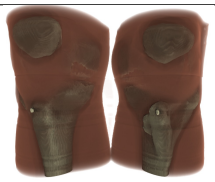
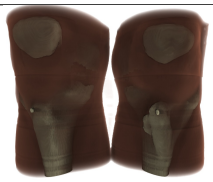
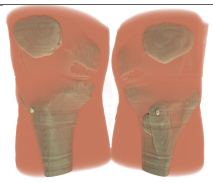
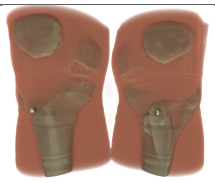
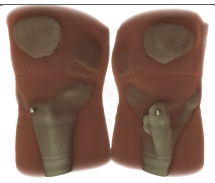
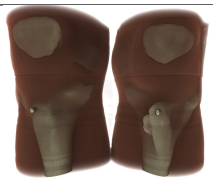
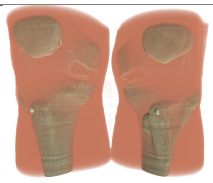
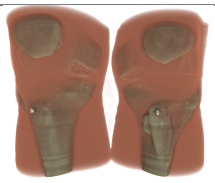
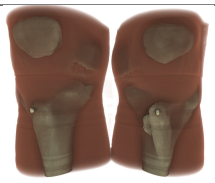
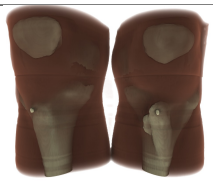
N	$R_\Omega = 4$	$R_\Omega = 8$	$R_\Omega = 16$	$R_\Omega = 32$
2				
	0.185	0.200	0.254	0.362
4				
	0.212	0.242	0.349	0.570
8				
	0.264	0.321	0.535	0.973
16				
	0.367	0.477	0.899	1.758
32				
	0.574	0.792	1.631	3.321

Table 6.2: Rendu volumique direct en faisant varier le rayon de la sphère d'influence (en colonne) de 0 à 32 voxels et le nombre de rayons lancés (en ligne) de 2 à 32. Le nombre associé à chaque image indique le temps de pré-calcul en secondes.

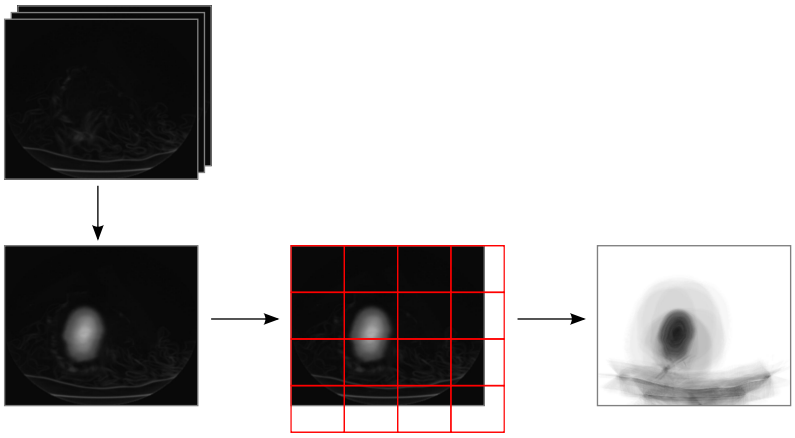


Figure 6.4: Génération des facteurs d'Ambient Occlusion en utilisant Cuda.

cette répartition automatiquement. On définit ainsi une répartition des données sous forme de blocs (matérialisés en rouge dans la figure 6.4). Chacun de ces blocs sera attribué à un multi-processeur sur la carte graphique pour le calcul de l'ensemble des voxels qu'il contient. Les différentes dimensions des blocs n'étant pas nécessairement un multiple de la taille d'une tranche, cela résulte en des threads inactifs lors de l'exécution sur les multi-processeurs.

La définition de la taille de ces blocs est expérimentale. Selon la documentation [NVI11], il faut s'assurer que le nombre de données à traiter dans ces blocs soit un multiple du nombre de threads par warps, comme les warps sont l'unité de traitement de base des multiprocesseurs. La documentation ne donne pas d'indication sur l'impact du nombre de dimensions choisi et précise que l'on doit expérimenter différentes tailles pour obtenir de bons résultats, un bon intervalle d'essai est de 128 à 256, toujours selon la documentation.

On souhaite répartir les threads de calcul dans une configuration de blocs et grille, dont la taille maximale pour notre GPU (GeForce GTX 295) est respectivement $(512, 512, 64)$ et $(65535, 65535, 1)$. A cela s'ajoute des contraintes supplémentaires. Il y a par exemple une limite matérielle, toujours pour l'architecture utilisée, de 512 threads au total par bloc. De plus, le noyau utilisé limite aussi le nombre total de threads par bloc pour prendre en compte ses propres contraintes d'utilisation en terme de nombre de registres et d'utilisation de mémoire partagée (pour le passage des arguments par exemple). Nous avons choisi ici d'utiliser des blocs de taille (16, 8, 1) par défaut, soit 128 éléments, car cela permet de traiter les jeux de données utilisés.

Pour un volume de taille (x, y, z) et si les contraintes sont respectées, on peut ainsi répartir les données avec le couple (bloc, grille) suivant $((16, 8, 1), (\lceil x/16 \rceil \times \lceil y/8 \rceil, z, 1))$, soit $((16, 8, 1), (16 \times 32, z, 1))$ pour 256^3. On prend bien garde à inclure la totalité des données en utilisant la partie entière supérieure. La position de chaque voxel est ensuite calculée en fonction de cette taille de bloc. Pour ce faire, on détermine le nombre de blocs par ligne et on calcule la position du voxel correspondant avec l'algorithme 6.1.

Un problème avec la définition de la taille des blocs est que la fonction d'accès aux données, telle que celle utilisée dans l'algorithme 6.1, est définie statiquement dans le code compilé sur la carte. L'ajout de dimensions dans le traitement des blocs implique donc que l'on modifie les fonctions d'accès pour l'ensemble des noyaux utilisant cette nouvelle fonction, ce qui implique de recompiler l'ensemble des noyaux

Algorithme 6.1 Calcul de la position d'un voxel en utilisant un découpage bidimensionnel des données.

```
int2 debutBlock;
int3 posVoxel;

debutBlock.x = (blockIdx.x % blocksParLigne) * blockDim.x
debutBlock.y = (blockIdx.x / blocksParLigne) * blockDim.y

posVoxel.x = debutBlock.x + threadIdx.x
posVoxel.y = debutBlock.y + threadIdx.y
posVoxel.z = blockIdx.z
```

correspondants. Ce côté statique, la séparation du code GPU/CPU et l'ensemble des contraintes matérielles à gérer rendent souvent complexe l'utilisation de CUDA.

En s'abstrayant de ces détails d'implémentation CUDA et en revenant au calcul des facteurs d'Ambient Occlusion, le calcul de la valeur de chaque voxel est alors implémenté sous la forme du noyau décrit par l'algorithme 6.2. Cet algorithme prend en paramètres les informations nécessaires pour le lancer de rayon : un offset o, un pas p, une taille de rayon R_Ω, ainsi qu'une direction de rayon d. Ce code échantillonne le long d'un rayon dont la direction est d, puis accumule les opacités. Le facteur d'occultation pour le rayon courant est ensuite ajouté aux facteurs calculés précédemment dans un tableau tridimensionnel de même taille que les données. Afin de garantir la meilleure précision possible le buffer d'accumulation est un buffer de valeurs flottantes, ainsi on ne quantifie pas les valeurs à chaque accumulation et on dispose donc d'une précision en flottant 32 bits.

Algorithme 6.2 Algorithme de calcul d'un rayon pour la technique d'Ambient Occlusion.

```
fonction AccumulerAO(tAO:tableau, dir:vecteur,
                     a:réel, pas:réel, Ro:réel)
        calculer position P du voxel courant

        ao = 1.0
        v = a * dir
        v_pas = pas * dir

        tant que norme(v) < Ro
                densite = texture(données, P + v)
                (couleur, opacité) = texture(f_t, densite)
                ao = ao * (1.0 - opacité)

                v += v_pas
        fin tant que

        tAO[P] += ao
fin fonction
```

Une fois l'ensemble des rayons parcourus, il reste à faire une dernière passe sur

le tableau en normalisant l'accumulation de chaque voxel en divisant la résultat par le nombre de rayons. Cette phase est aussi l'occasion de réaliser une quantification dans un format moins gourmand en mémoire, nous utilisons ici les entiers 8 bits.

Concernant les possibilités d'optimisation de l'algorithme du lancer de rayons, l'algorithme est très simple et ne peut guère être davantage optimisé spécifiquement pour CUDA. Une stratégie souvent utilisée dans l'optimisation de codes CUDA est l'utilisation de la mémoire partagée. Or cette dernière se base sur la réutilisation des données dans les noyaux et dans ce cas, il n'y en a pas d'un thread à un autre dans un même bloc. De plus, les accès à la mémoire sont déjà optimisés du fait que nous utilisons les textures qui sont mises en cache au niveau des multi-processeurs. On pourrait tenter d'optimiser plus le code, mais cela ne s'avère pas toujours payant. Par exemple, en recourant à des conditionnelles, on risque de créer des divergences dans l'exécution des blocs et ainsi de ralentir le code plutôt que de l'accélérer.

6.3 Ambient Occlusion par structure

Afin d'éviter le phénomène d'assombrissement dû aux ombres de contact entre les différentes structures de données volumiques, nous introduisons une méthode de calcul de l'Ambient Occlusion par structure. Cette méthode permet de limiter l'influence du choix d'une mauvaise longueur de rayon R_Ω en 3 étapes :

- La définition des structures (cf. sous-section 6.3.1),
- Le calcul des facteurs d'Ambient Occlusion par structure (cf. sous-section 6.3.2),
- La recombinaison des différents facteurs (cf. sous-section 6.3.3).

6.3.1 Définition des structures

La définition des structures n'a pas été le propos de ce travail. Cette opération est donc basée sur un découpage des fonctions de transfert "à la main" en autant de fonctions de transfert que l'on souhaite définir de structures, comme illustré en Figure 6.5. Pour chaque structure identifiée par un ou plusieurs intervalles de densité, on crée alors une nouvelle fonction de transfert dans laquelle on aura recopié l'information de couleur et d'opacité associée aux intervalles correspondants.

Pour cette opération de découpage, on peut aussi s'appuyer sur des fonctions de transfert prédéfinies notamment dans le domaine médical, où les capteurs font correspondre certaines valeurs de densité à des zones bien définies. On peut aussi générer des fonctions de transfert en utilisant des fonctions bidimensionnelles ou les définir à partir d'autres paramètres tels que la visibilité ou la taille des structures. La définition de la structure est obtenue pour l'ensemble des voxels dont la densité transformée par la fonction de transfert donne une opacité supérieure à 0, autrement dit tous les voxels non transparents.

6.3.2 Calcul par structure

Rappelons que le calcul de l'Ambient Occlusion est effectué sur l'ensemble des voxels contenus dans le jeu de données volumiques. On peut alors penser dans un premier temps à calculer des valeurs d'occultation uniquement pour les voxels qui seront effectivement utilisés lors de la visualisation, c'est-à-dire les voxels dont l'opacité est supérieure à 0 pour une structure spécifique. En faisant cela, on se retrouve

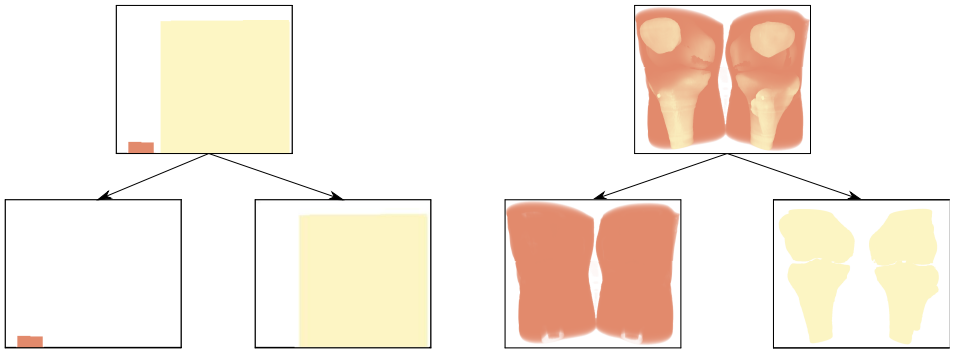


FIGURE 6.5 – Découpage de la fonction de transfert en sous-fonctions de transfert correspondant aux structures dans le cas du jeu de données "Knee" ($379 \times 229 \times 305$). La première ligne représente la fonction de transfert et la visualisation initiale associée. La seconde ligne représente les fonctions de transfert découpées et les visualisations associées à chaque structure (ici au nombre de deux).

(a) (b) (c)

FIGURE 6.6 – Présence d'artefacts quand on ne considère que les voxels dont l'opacité est supérieure à 0 pour une structure donnée, en écrivant respectivement la valeur 0 (a) et 1 (b) pour les voxels transparents. Puis en calculant la valeur d'occultation pour l'ensemble des voxels, même les voxels transparents (c).

en présence d'artefacts sur les structures avec une haute opacité, comme illustré en Figure 6.6.

Si l'utilisation de cette technique permet de réduire effectivement les temps de calcul, elle est donc la cause d'artefacts sur le bord des structures. Le problème vient du fait que l'on doit attribuer des valeurs par défaut aux voxels transparents, car on ne leur calcule pas de facteur d'occultation. Or ces valeurs par défaut seront utilisées au moment de la visualisation sur le bord des structures, en raison de l'interpolation trilinéaire. La Figure 6.6 illustre le fait de mettre les voxels transparents à 0.0 (totalement occulté (a)), à 1.0 (non occulté (b)) et en calculant l'Ambient Occlusion sur l'ensemble des voxels (c). On constate effectivement des artefacts de visualisation non présents en calculant les valeurs d'occultation pour l'ensemble des voxels. Calculer des valeurs d'occultation pour l'ensemble des voxels du jeu de données permet d'éviter les artefacts.

Concernant le calcul de l'Ambient Occlusion par structure, on peut imaginer le réaliser en une passe. En partant du voxel de départ, on identifie à quelle structure celui-ci appartient, puis on effectue le lancer de rayons en ne considérant que la fonction de transfert associée à cette structure. Le problème des discontinuités réapparaît alors, car pour un voxel transparent au bord d'une structure, on ne peut pas identifier si il appartient à la structure voisine, à moins d'examiner le voisinage de

celui-ci, ce qui se révèle coûteux en temps. On est donc incapable de déterminer la fonction de transfert correspondant à la bonne structure voisine et à l'utiliser dans ce cas.

La stratégie que nous avons employée est de générer séparément des facteurs d'Ambient Occlusion pour chaque structure. Chaque structure est identifiée par une fonction de transfert, qui correspond à une sous-partie de la fonction de transfert initiale (voir Figure 6.5). On génère successivement des facteurs d'occultation pour l'ensemble des voxels et pour l'ensemble des structures. Pour cela, on utilise le même noyau de calcul que pour l'Ambient Occlusion de base, mais en associant dynamiquement la bonne fonction de transfert à l'identifiant de texture générique représentant la fonction de transfert. On obtient donc n volumes d'Ambient Occlusion pour n structures. Une fois tous ces facteurs d'occultation calculés, il faut les recombiner en un volume final qui sera utilisé lors de la phase de rendu pour moduler l'éclairage. Ceci peut être résumé en modifiant la formule 6.2 du calcul de $I_l(x_v)$ de la manière suivante :

$$\begin{aligned} I_l(x_v) &= \odot_{k=1}^{n_f} I_l(x_v, k) \\ avec\ I_l(x_v, k) &= \sum_{m=0}^{M} \frac{1}{M} \prod_{i=0}^{m-1} (1 - \alpha_{i,k}) \end{aligned}$$

avec $\odot$, un opérateur de recombinaison, n_f le nombre de structures et $\alpha_{i,k}$, l'opacité en fonction de la structure k (i correspond au i^{e} échantillon le long des rayons et k, la structure utilisée).

6.3.3 Recombinaison

Pour la phase de recombinaison, la première difficulté provient du fait que le nombre de structures que l'on va définir dans un jeu de données n'est pas défini à l'avance et qu'on peut vouloir le modifier lors de l'exécution du programme. Il faut alors pouvoir générer un noyau qui permettra de recombiner les différentes structures. L'approche classique CUDA en se basant sur l'API Runtime ne permet pas de telles manipulations, car l'ensemble du code est compilé dans le binaire du programme, rendant ainsi impossible les modifications lors de l'exécution. Une solution pour contourner ce problème serait d'utiliser un buffer d'accumulation et de combiner successivement chaque texture au sein de ce buffer. Cependant, nous avons choisi d'utiliser l'API Driver qui permet de générer du code durant l'exécution du programme et de le charger pour exécution sur la carte graphique. Pour cela, nous créons un squelette de programme qui est complété au moment d'effectuer la fusion des valeurs pour les différentes structures. Une fois complété ce programme est compilé à la volée à l'aide du compilateur *nvcc* de CUDA et lié au programme courant, afin de pouvoir être exécuté.

Dans un second temps, on définit l'opérateur de recombinaison utilisé pour rassembler les différentes valeurs des structures calculées. Nous avons expérimenté différents opérateurs, présentés en Figure 6.7.

La Figure 6.7 présente une comparaison des différents opérateurs de recombinaison testés. La première ligne présente une vue en coupe dans l'axe (x ;y) du volume d'occultation généré par structure pour les deux structures du jeu de données "Knee" de la figure 6.5. Chacune des lignes présente une vue en coupe du volume d'occultation après recombinaison (à gauche) et une image de rendu final à droite. Ces lignes représentent respectivement :

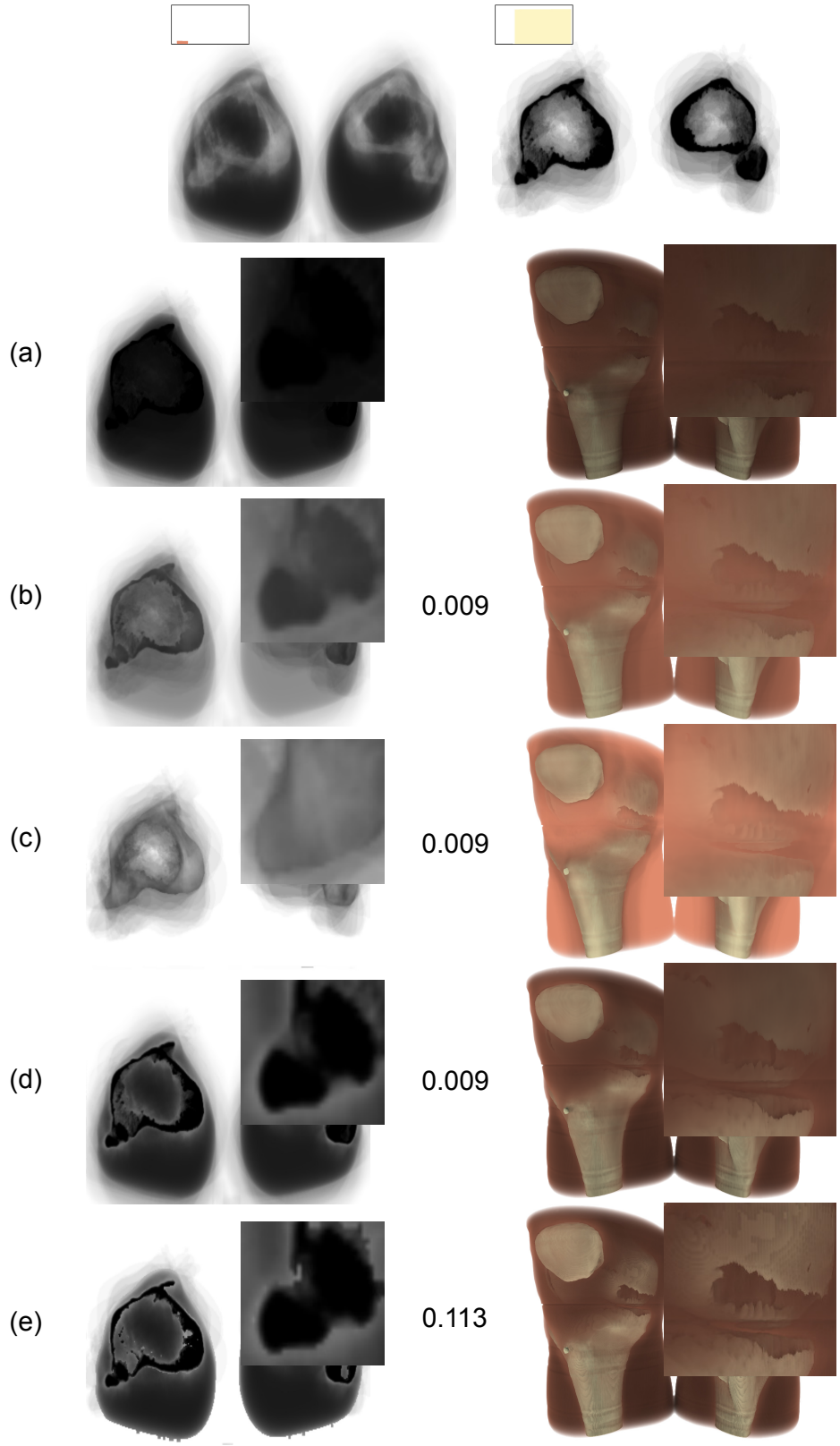


FIGURE 6.7 – Comparaison d'opérateurs de recombinaison. La première ligne représente une coupe des volumes d'occultation correspondant aux deux structures de la Figure 6.5. Pour chacune des lignes suivantes, une coupe du volume d'occultation (à gauche) et le rendu final (à droite) sont présentés. Les lignes (a), (b), (c), (d) et (e) représentent respectivement l'Ambient Occlusion classique, puis les recombinaisons avec moyenne, maximum, minimum et 26-voisinage. La colonne du milieu présente les temps de recombinaison en secondes.

(a) L'Ambient Occlusion classique sur l'ensemble de la fonction de transfert,

(b) L'opérateur *moyenne* pour la recombinaison. Pour un voxel v en position (x, y, z), on récupère l'ensemble des facteurs d'occultation à cette position (x, y, z), puis on calcule une moyenne de ces facteurs.

(c) L'operateur *maximum*. De manière similaire à l'opérateur moyenne, on récupère l'ensemble des facteurs d'occultation correspondant à un voxel et on choisit la valeur maximale.

(d) L'opérateur *minimum*. Traitement similaire au maximum.

(e) L'opérateur basé sur le *voisinage*. Dans le 26-voisinage d'un voxel considéré v, on compte le nombre de voxels appartenant à chaque structure. Puis on choisit la valeur d'occultation correspondant à la structure qui comptabilise le plus de voxels.

La colonne du milieu présente les temps de recombinaison mesurés pour chacun des opérateurs. On constate déjà que les opérateurs *minimum*, *moyenne* et *maximum* s'exécutent en des temps similaires. Seul l'opérateur basé sur le *voisinage* prend plus de temps, ce qui est du au fait qu'il faut parcourir les 26 voisins de chaque voxel, et faire autant d'accès texture. Globalement, les opérateurs restaurant le plus de visibilité au sein des structures internes sont le *minimum* et l'opérateur basé sur le *voisinage*. Avec ces opérateurs, on distingue par exemple une fente en haut à gauche sur le zoom, ainsi qu'une cassure en bas à gauche qu'il est difficile de voir sur les autres captures et notamment sur celle avec l'Ambient Occlusion classique (a) en raison du sur-assombrissement. L'opérateur *minimum* assure une meilleure continuité des facteurs que celui basé sur le *voisinage*. Il donne finalement les meilleurs résultats, en réduisant les artefacts et en améliorant le contraste. Nous expliquons cela par le fait que la valeur d'occultation devient plus grande en se rapprochant d'une certaine structure. Comme la meilleure valeur d'occultation pour un voxel est donnée par la structure avoisinante de plus grande influence, le minimum permet d'atteindre cet objectif. Notons que l'utilisation de cet opérateur étend l'influence des structures opaques aux structures voisines non opaques. De ce fait, les valeurs d'occultation utilisées pour des structures plus transparentes peuvent être erronées. Cependant, nous n'avons pas constaté d'artefacts visibles, car les structures plus transparentes sont aussi moins visibles.

6.4 Résultats, performance et occupation mémoire

L'ensemble des résultats suivants a été produit avec l'implémentation CUDA du lancer de rayon. La configuration utilisée est un Intel Quad Core Q8200 avec 4 Go de RAM, une NVIDIA GeForce GTX 295. Cette carte graphique est particulière, car elle est composée de deux GPU avec chacun 896 Mo de RAM. La puissance de chacun est équivalente à celle d'une GeForce GTX 275. Nous n'utilisons ici qu'un seul GPU sur les deux pour les calculs et le rendu.

Nous présentons dans la suite les résultats obtenus pour plusieurs jeux de données : un pied ("Foot") de taille $256 \times 256 \times 256$, une tête ("CT Head") de taille $256 \times 256 \times 225$, une orange ("Orange") de taille $256 \times 256 \times 64$ et des genoux ("Knee") de taille $379 \times 229 \times 305$. Concernant les paramètres de l'Ambient Occlusion, nous utilisons un pas d'échantillonnage et un offset initial d'1 voxel. Le nombre de rayons lancés pour échantillonner la sphère est de 32 et le rayon de la sphère R_Ω est de 32 voxels.

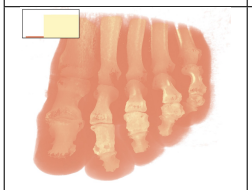
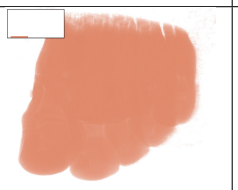
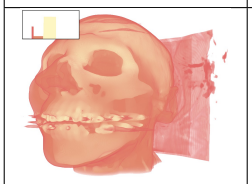
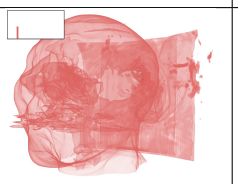
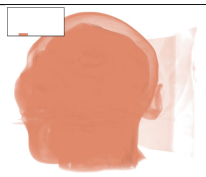
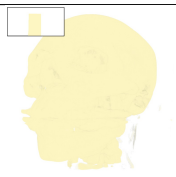
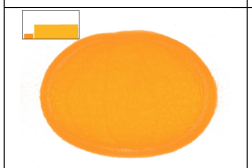
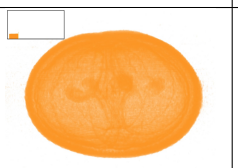
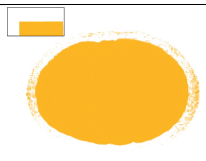
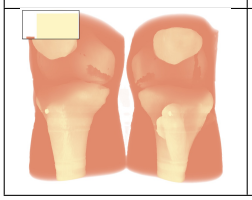
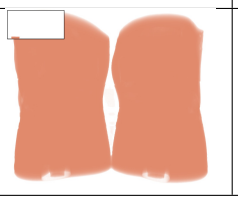
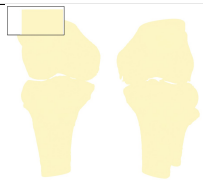
Totalité	Structure 1	Structure 2	Structure 3
			
			
			
			

TABLE 6.3 – Tableau illustrant les différentes structures dans chaque jeu de donnée utilisé.

Les jeux de données ont été pré-segmentés en un nombre défini de structures en utilisant une segmentation empirique de la fonction de transfert. Néanmoins dans certains domaines scientifiques, comme l'imagerie médicale, les scientifiques ont l'habitude de travailler avec des données spécifiques, et ont du coup créé des fonctions de transfert adaptées à ces données. En utilisant ces fonctions de transfert, le découpage en structures peut être obtenu plus facilement. Un exemple de découpage, utilisé ici, est illustré en Table 6.3.

Résultats

Nous présentons dans un premier tableau 6.4, un ensemble de résultats obtenus pour différents jeux de données. Pour chacun, nous présentons des images sans éclairage, avec éclairage de Phong, Ambient Occlusion normal et notre Ambient Occlusion par structure.

Les images n'utilisant aucun éclairage fournissent peu d'informations sur la forme des structures et la position relative dans les données, voir par exemple l'orange. Pour l'utilisation de l'éclairage de Phong, nous avons empiriquement défini une direction d'éclairage permettant d'apporter une bonne perception dans les données. Les gradients utilisés pour la visualisation ont été pré-calculés et combinés à la texture contenant les données. Le fait que l'Ambient Occlusion soit indépendant d'une direction d'éclairage permet de simplifier la tâche de l'observateur, qui n'a plus besoin de rechercher une "bonne" direction. On peut constater que l'Ambient Occlusion conduit à un sur-assombrissement des structures internes. Notre méthode permet de restaurer une partie de la visibilité pour ces structures internes, si les structures externes ne sont pas totalement opaques. Elle est donc bien adaptée à la

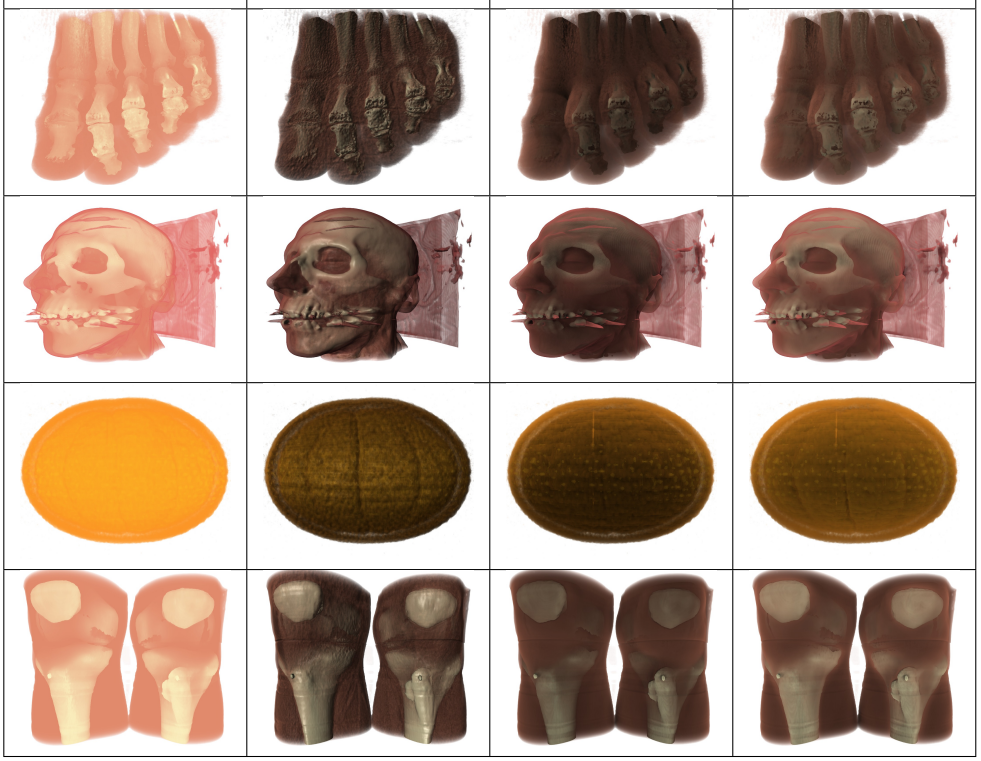
Normal	Phong	AO	AO par Structure
			

TABLE 6.4 – Comparaison d'images pour différents jeux de données.

visualisation en utilisant la semi-transparence.

Le nombre de paramètres à gérer est peu important. Il n'y a pas de paramétrisation de l'éclairage à faire. Ceci est un avantage, car en rendu volumique les utilisateurs ne souhaitent pas perdre de temps à configurer des paramètres notamment d'éclairage. Pour l'Ambient Occlusion, un paramètre important est le rayon d'influence R_Ω. Dans notre cas, l'impact d'un mauvais choix de ce paramètre est réduite. Il est en effet assez difficile de trouver une valeur correcte pour ce paramètre tant il dépend du jeu de données et de la proximité des différentes structures considérées. Si on choisit un rayon R_Ω trop grand, on risque de capturer de l'information en provenance de structures voisines pour lesquelles on ne souhaite, par exemple, pas avoir de projection d'ombres de contact. La figure 6.8 montre que l'on réduit effectivement l'influence de R_Ω. La première ligne montre que l'Ambient Occlusion classique est sensible à ce paramètre. Avec notre méthode en seconde ligne, la différence entre les valeurs de R_Ω est réduite.

Performance

La performance en temps de calcul de cette méthode est liée au nombre de structures que l'on a identifié avec la fonction de transfert. A chaque nouvelle structure, on va calculer un volume contenant des facteurs d'occultation pour cette structure. Si le volume d'occultation d'un jeu de données de taille $n \times m \times o$ est calculé en k secondes et qu'on le découpe en l structures, le temps total de calcul T sera de : $T \approx l \times k$, en négligeant ici le temps de recombinaison, qui est de l'ordre de quelques millisecondes. La table 6.5 illustre les temps de calculs de notre méthode appliquée aux différents jeux de données des tables 6.3 et 6.4. Les temps de calcul sont à la

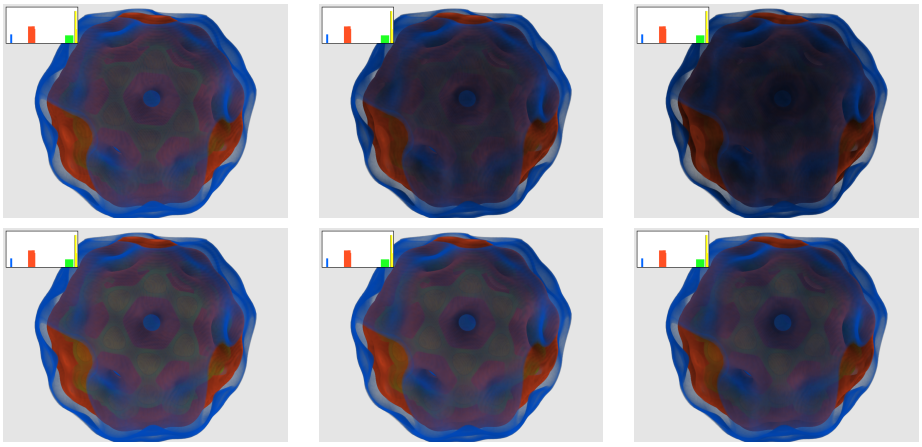


Figure 6.8: Influence de R_Ω sur le jeu de données du "Bucky Ball" (128^3) : La première ligne correspond à l'Ambient Occlusion classique, la seconde à notre méthode. De gauche à droite, R_Ω a pour les valeurs 8, 16 et 32 voxels.

	Foot	Head	Orange	Knee
Struct. 1	2.044	1.786	0.481	3.267
Struct. 2	2.043	1.789	0.481	3.270
Struct. 3	X	1.786	X	X
Recomb	0.005	0.005	0.001	0.009
Total	4.092	5.366	0.963	6.546

TABLE 6.5 – Tableau des temps de calculs de la méthode par structures pour différents jeux de données.

fois proportionnels à la taille du jeu de données et au nombre de structures.

Occupation mémoire

Concernant l'empreinte mémoire du processus, elle peut être importante si on laisse les valeurs d'occultation de chaque structure en mémoire. On peut ainsi avoir une occupation mémoire Om définie par :

$$Om = (N+1) \times x_d \times y_d \times z_d \times b_d$$

avec N, le nombre de structures considérées ; x_d, y_d, z_d et b_d, respectivement la taille des données en x, y, z et le nombre d'octets occupés par une valeur de densité d. Le fait d'utiliser $N+1$ dans la formule est lié à une contrainte CUDA qui ne permet pas d'écrire directement dans une texture 3D depuis un noyau. Si on souhaite écrire dans une texture 3D depuis un noyau, la seule manière à l'heure actuelle est d'écrire dans un buffer intermédiaire, puis de copier ensuite son contenu dans la texture 3D. On doit donc considérer un buffer supplémentaire de taille : $x_d \times y_d \times z_d \times b_d$, pour stocker les valeurs intermédiaires de fusion avant la recopie dans la texture 3D finale. Cette dernière opération est très peu coûteuse, car la copie en mémoire interne d'une carte est très rapide.

Une solution pour réduire l'occupation mémoire peut être utilisée dans le cadre d'opérateurs ne nécessitant pas de conserver l'ensemble des facteurs d'occultation pour chaque structure. C'est le cas avec les opérateurs minimum, maximum et moyenne. On peut utiliser un seul buffer. En veillant à l'initialiser correctement

au départ : 0.0 pour le maximum, 1.0 pour le minimum, 0.0 pour la moyenne, on va pouvoir au fil des calculs des valeurs d'occultation par structure, raffiner la valeur présente dans le buffer pour arriver à la valeur finale.

6.5 Conclusion

Dans ce chapitre, nous avons présenté une méthode permettant de faciliter l'application de l'Ambient Occlusion dans le cadre du rendu volumique direct. Pour cela, nous proposons de nous baser sur une séparation des données en structures en utilisant la fonction de transfert. Puis nous calculons les facteurs d'occultation par structure et les recombinons au sein d'un seul volume pour la visualisation finale. Notre méthode permet ainsi de restaurer de la visibilité en évitant les projections d'ombres de contact entre les structures. Elle se révèle utile lorsqu'on a des structures proches, potentiellement imbriquées les unes dans les autres, et que l'on souhaite pouvoir analyser la surface de chacune d'elles sans que l'autre n'interfère dans l'ombrage.

Cette méthode a cependant un coût mémoire et en temps de calcul plus important que l'Ambient Occlusion de base comme nous l'avons vu en section 6.4. C'est pourquoi nous nous sommes intéressés à la parallélisation de l'Ambient Occlusion comme présentée au chapitre suivant.

Chapitre 7

Ambient Occlusion accéléré par Multi-GPU

7.1 Introduction et problématique

Nous avons choisi d'utiliser une méthode se basant sur un pré-calcul de l'Ambient Occlusion dans le cadre de cette thèse, un des objectifs étant de ne pas affecter la performance de la visualisation. Néanmoins, le temps de pré-calcul de l'ordre de quelques secondes n'est pas négligeable, surtout si on souhaite éditer la fonction de transfert en temps interactif. Dans cette partie, nous proposons de réduire les temps de calcul induits par la méthode d'Ambient Occlusion en nous basant sur l'utilisation d'un système multi-GPU. La problématique est donc de répartir correctement les calculs d'Ambient Occlusion entre les différents processeurs, afin de réduire les temps de calcul.

Notre parallélisation s'appuie sur une architecture composée de n GPU situés dans une même machine et reliées au processeur par le bus PCI-express. On suppose que les GPUs sont identiques au niveau des spécifications. Dans la suite de ce travail, nous ne gérons pas l'hétérogénéité potentielle des cartes en terme de génération, de capacité de calcul ou de nombre d'unités de calcul, même si c'est un facteur que l'on peut prendre en compte avec des heuristiques basées sur ces propriétés. L'ensemble des résultats et des tests a été réalisé sur une architecture munie de GPUs identiques.

Comme nous sommes toujours dans un contexte de visualisation, une fois les facteurs d'Ambient Occlusion calculés, une image doit être produite. L'objectif de la thèse n'étant pas d'optimiser la phase de rendu, la méthode utilisée a été la suivante. Parmi les n GPUs, on en désigne arbitrairement un comme celui qui sera en charge de la génération de l'image finale. On suppose donc que celui-ci dispose constamment du jeu de données complet, ainsi que de la fonction de transfert utilisée en mémoire. Notons toutefois que la phase de calcul de l'Ambient Occlusion est indépendante de la phase de rendu. On pourrait donc interfacer la méthode de calcul de l'Ambient Occlusion parallèle avec des techniques de rendu parallèle existantes, telles que celles proposées par Gobetti et al. [GMI08] et Crassin et al. [CNLE09] pour la visualisation de gros volumes de données, ou un rendu de type sort-last comme proposé par Marchesin et al. [MDM07].

Pour l'ensemble des résultats de ce chapitre, la configuration utilisée est identique et composée d'un Intel Quad Core Q8200 avec 4 Go de RAM et de deux NVIDIA GeForce GTX 295 identiques et de même marque : les fréquences de fonctionnement sont ainsi identiques. Chacune des deux cartes est équipée de deux GPUs, équivalent

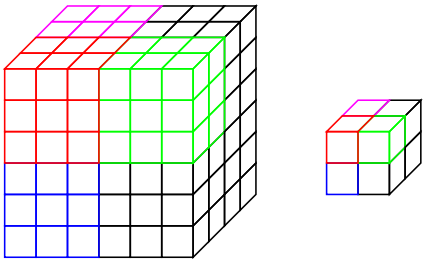


FIGURE 7.1 – Illustration du processus de bricking, dans lequel on regroupe des blocs de voxels connexes en briques.

de GeForce GTX 275, embarquant chacun 896 Mo de mémoire interne. Ce qui fait donc un total de 4 GPUs. La mémoire interne des deux GPUs d'une même carte n'étant pas partagée, on se retrouve dans la même situation qu'avec un vrai multi-GPU, car pour transférer des données d'un GPU à l'autre, on doit passer par le CPU.

Concernant la configuration du processus d'Ambient Occlusion, nous avons utilisé la même dans l'ensemble des résultats de ce chapitre, c'est-à-dire un pas d'échantillonnage et un offset d'un voxel, un rayon de sphère R_Ω de 32 voxels et 32 rayons pour échantillonner la sphère autour de chaque voxel.

Nous détaillons dans la suite les différentes approches d'optimisation mises en place : la section 7.2 décrit une première approche de réduction des temps de calcul : le "bricking" ou découpage en briques. Puis la section 7.3 décrit les différentes approches de parallélisation possibles, avant de décrire les deux méthodes implantées : statique en section 7.4 et dynamique en section 7.5.

7.2 Bricking

Une première approche d'optimisation mise en place est l'utilisation d'une méthode de *bricking*. Cette méthode part du principe que chaque voxel ne contribue pas à l'image finale. En effet, il y a souvent une grande partie des voxels qui sont rendus transparents par la fonction de transfert et ne contribuent ainsi pas à la visualisation. Cette technique va permettre d'éviter de calculer des facteurs d'occultation pour ces voxels. Pour cela, on définit un groupement de voxels sous la forme de briques de taille identiques (cf. Figure 7.1). A chacune de ces briques, on associe une valeur booléenne qui indique si tous les voxels de la brique sont ou non transparents : 0 indique une brique vide qui ne contribue pas au rendu final (tous ses voxels sont transparents), 1 indique une brique pour laquelle on va calculer des valeurs d'occultation pour ses voxels (il y a au moins 1 voxel non transparent). Comme pour les facteurs d'Ambient Occlusion, le contenu d'une brique dépend de la fonction de transfert, la valeur d'une brique doit donc être recalculée à chaque changement de fonction de transfert.

Calcul de la valeur des briques – La valeur binaire associée à chaque brique est calculée sur le GPU en parcourant pour chaque brique l'ensemble des voxels qui la constitue jusqu'à trouver un voxel non transparent (dans ce cas la valeur associée est 1) ou jusqu'à ce que tous les voxels soient parcourus (la valeur 0 est utilisée dans ce cas). Chacun des n GPUs reçoit du CPU le même nombre de briques à calculer. Ce dernier est obtenu en divisant le jeu de données en n blocs B_i de même taille. Le résultat de ce calcul est une texture binaire associant à chaque brique 0 ou 1 selon

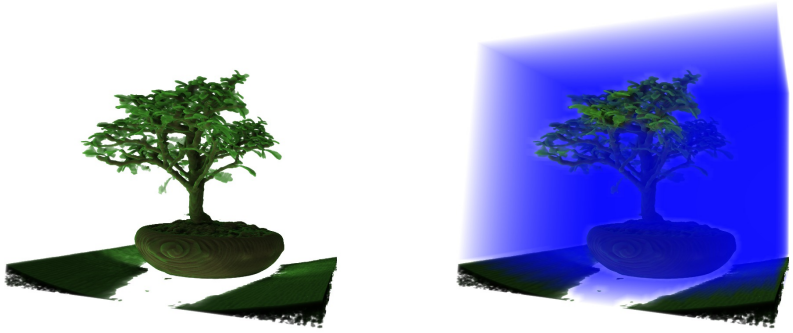


FIGURE 7.2 – Exemple de bricking en utilisant des briques de taille 4^3 sur le bonsai (256^3). Les briques vides, qui ne sont donc pas traitées, sont représentées par la couleur bleue.

qu'elle est vide ou pleine. Si la taille du jeu de données n'est pas un multiple de la taille d'une brique, des voxels fantômes sont simulés au bord du jeu de données. Ceci est réalisé en utilisant l'opération de "clamping" des textures. Cette opération consiste à maintenir les coordonnées des textures dans l'intervalle $[0; 1]$: si la valeur d'une coordonnée est inférieure à 0, celle-ci est ramenée à 0 au moment de l'accès à la texture ; de la même manière avec une valeur supérieure à 1 qui sera ramenée à 1 au moment de l'accès texture. Notons que ce calcul supplémentaire est très rapide et ne représente qu'une partie minime du temps total de calcul de l'Ambient Occlusion.

Artefacts – Le calcul des valeurs des briques doit être fait avec une attention particulière pour les voxels situés au bord des briques. En effet, considérons deux briques voisines b_1 et b_2 avec b_1, marquée comme pleine et b_2 comme vide. Si un voxel v_1, appartenant à b_1, est voisin d'un voxel v_2, appartenant à b_2, on calcule une valeur d'occultation pour v_1, mais pas pour v_2. Or si un échantillon lors de la phase de lancer de rayon passe entre v_1 et v_2, ceci résulte en la présence d'artefacts liée à l'interpolation trilinéaire qui sera effectuée entre la valeur d'occultation calculée de v_1 et la valeur indéfinie qui sera dans v_2. Ceci se manifeste par la présence de discontinuités dans l'ombrage. Pour remédier à cela, nous considérons un dépassement pour chaque brique, afin de tenir compte des voxels immédiatement voisins des briques adjacentes.

Utilisation – La texture obtenue après traitement des briques est ensuite utilisée dans la phase de pré-calcul de l'Ambient Occlusion. Pour chaque voxel à calculer, on récupère la valeur de la brique dans laquelle il est contenu. Si la valeur indique une brique pleine, on calcule le facteur d'occultation de ce voxel, sinon on quitte le noyau. La figure 7.2 illustre les briques vides qui ne seront pas traitées (en bleu) pour le jeu de données du bonsai.

Résultats – Les jeux de données utilisés dans la suite sont un pied de taille 256^3 ("Foot"), un bonsai de taille 256^3 ("Bonsai"), une tête de taille 256^3 ("CT Head") et des genoux de taille $379 \times 229 \times 305$ ("Knee"), tous obtenus par tomodensitométrie (ou "CT-Scan"). Ils sont illustrés en Figure 7.3.

La Figure 7.4 présente 4 graphes illustrant les temps de calcul du volume d'Ambient Occlusion pour chaque jeu de données en fonction de la taille de brique. Les différentes tailles de briques choisies sont 1^3, 2^3, 4^3, 8^3, 16^3 et 32^3. La taille de brique 0 correspond à l'absence de bricking. Les graphes sont produits en n'utilisant qu'un seul des 4 GPUs, afin de d'abord constater les bénéfices du bricking avant même de paralléliser le processus. Les courbes bleues, vertes et rouges représentent respecti-

FIGURE 7.3 – Les différents jeux de données utilisés dans le cadre de l'amélioration du temps de calcul de l'Ambient Occlusion.

vement le temps total de calcul (T_{total}), incluant tous les précalculs de bricking et les transferts mémoires, le temps de calcul du lancer de rayons uniquement pour calculer l'Ambient Occlusion en utilisant le bricking (T_{RC}) et le temps de calcul du bricking uniquement (T_B). On constate une réduction du temps total allant de un tiers à près de la moitié selon le jeu de données. Le meilleur compromis de taille de brique, en terme de temps de précalcul/réduction de temps, est situé à 2^3 ou 4^3 sur l'ensemble des jeux de données utilisés. Nous avons choisi d'utiliser une taille de brique de 4^3 pour la suite.

Par ailleurs, on peut se demander si la réduction du temps de calcul est directement liée au pourcentage de briques vides/pleines effectivement présentes dans le jeu de données pour une fonction de transfert donnée. Les graphes en Figure 7.5 présentent ces deux informations pour les jeux de données précédents. La première est le pourcentage de briques pleines dans chaque jeu de données pour une fonction de transfert donnée avec la courbe bleue. La courbe verte représente quant à elle la réduction du temps de calcul pour chaque taille de brique, c'est-à-dire le rapport entre les temps cumulés pour le lancer de rayon avec bricking sur les temps cumulés sans bricking. On constate une différence de l'ordre de 10 à 20% entre la réduction de temps espérée, correspondant au nombre de briques vides dans le jeu de données, et la réduction effective. Ceci peut s'expliquer par le fait que pour chaque voxel traité, le bricking ajoute un accès texture supplémentaire, ainsi qu'un test pour vérifier si le voxel considéré appartient à une brique vide, ce qui peut causer des divergences dues aux branchements conditionnels.

Le bricking fait aussi apparaître des différences de temps de calcul suivant la fonction de transfert utilisée. La Figure 7.6 illustre un exemple de la variation du temps de calcul pour le jeu de données du pied. La première fonction de transfert fait apparaître une structure moins compacte que la seconde et par conséquent le bricking permettra d'éliminer moins de briques dans le premier cas que dans le second, d'où une différence dans le temps de lancer de rayons de l'ordre de 300 ms.

Cette technique de bricking permet donc de diminuer le temps de pré-calcul en fonction des données effectivement rendues non transparentes par la fonction de transfert. Cette optimisation est dépendante de la fonction de transfert, par conséquent le gain sera moindre si cette dernière fait apparaître beaucoup de plages de densité pour un jeu de données.

Pour gagner davantage en efficacité, nous étudions donc dans la suite de ce chapitre différentes techniques de parallélisation du calcul de l'Ambient Occlusion.

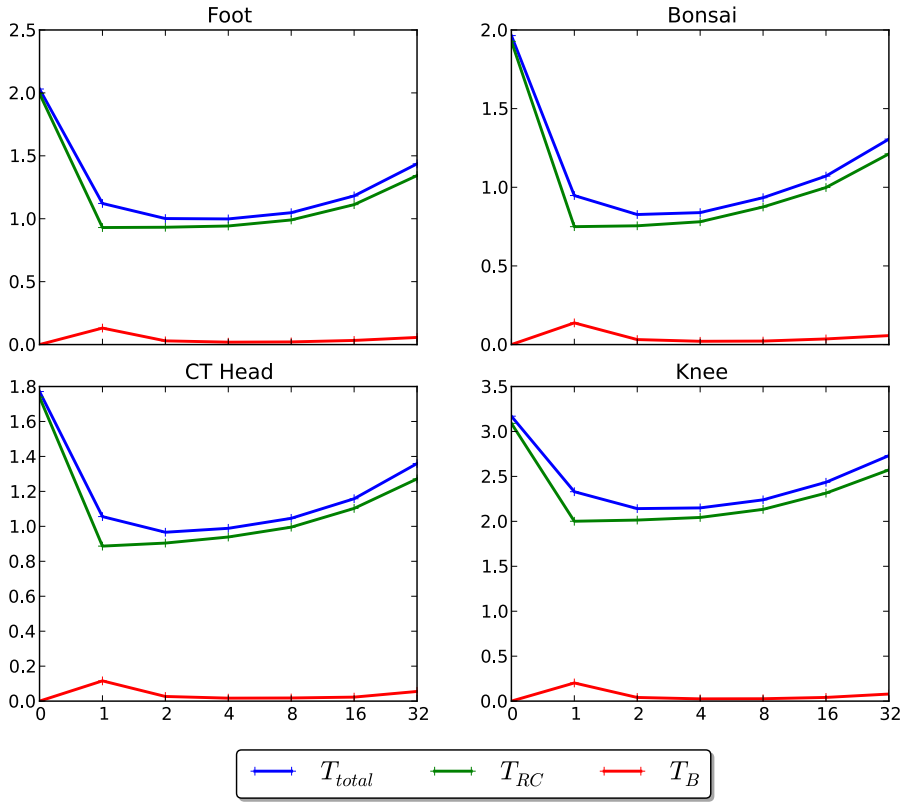


FIGURE 7.4 – Comparaison des temps de calcul en utilisant un seul GPU avec et sans bricking pour différents jeux de données. En abscisse se trouve la taille de brique utilisée (0 signifie sans Bricking) et en ordonnée, le temps de calcul mesuré. Les courbes bleue, verte et rouge correspondent respectivement au temps total mesuré (T_{total}), au temps du lancer de rayon (T_{RC}) et au temps de calcul de la structure du bricking (T_B).

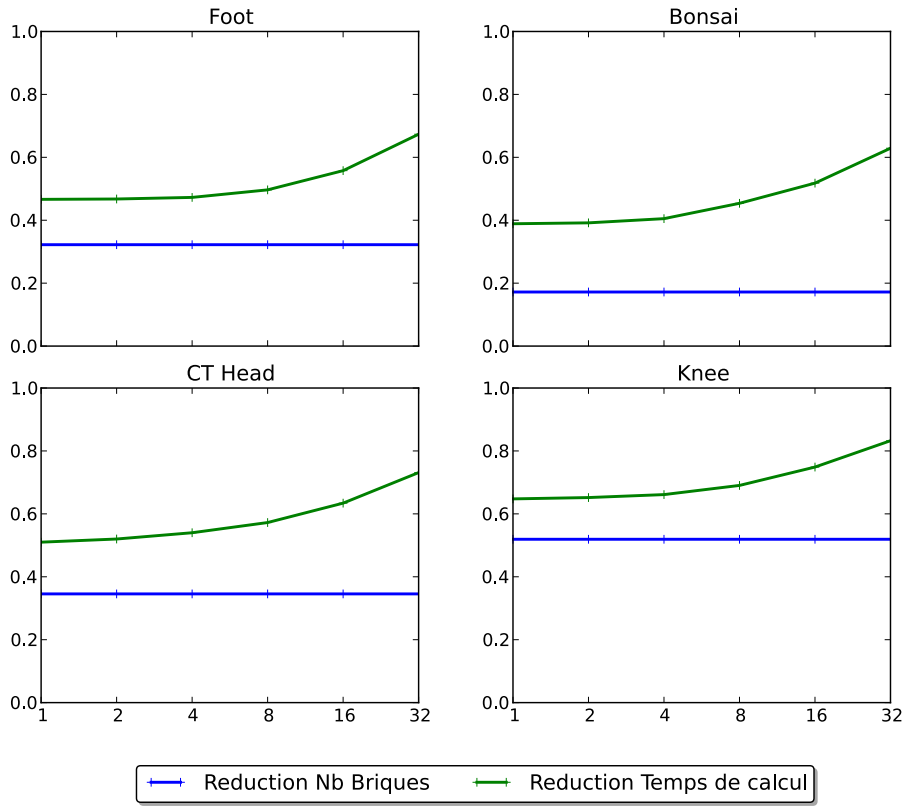
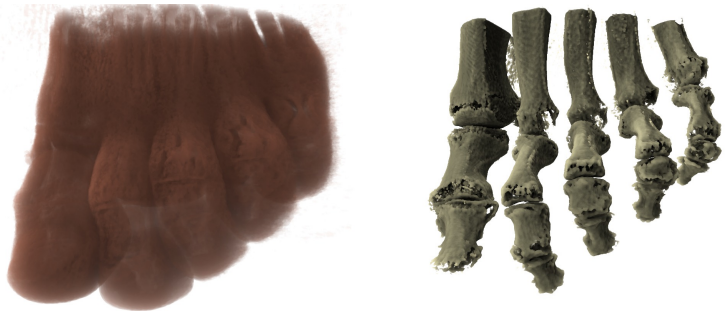


FIGURE 7.5 – Pourcentages ramenés entre 0 et 1 représentant pour une fonction de transfert spécifique le pourcentage de briques pleines dans le jeu de données (courbe bleue) et le rapport entre le temps de calcul du lancer de rayon avec bricking sur le temps de calcul initial (vert).



$T_{RC} = 0.941$, $T_B = 0.019$ $T_{RC} = 0.644$, $T_B = 0.022$

FIGURE 7.6 – Comparaison des temps de calcul du lancer de rayon (T_{RC}) et de calcul de la structure du bricking (T_B) sur deux fonctions de transfert différentes pour un même jeu de données.

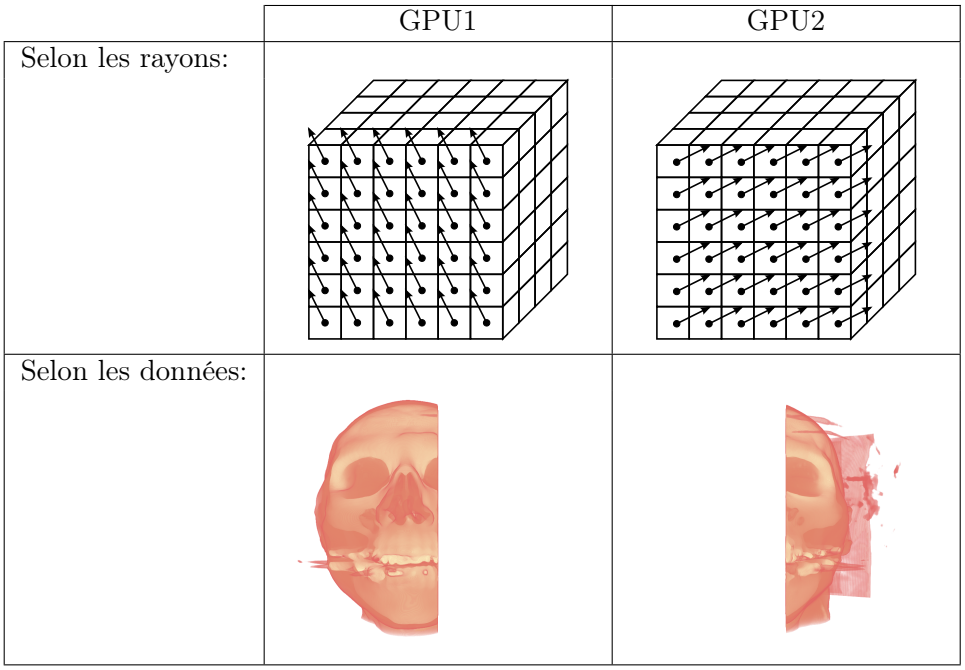


FIGURE 7.7 – Parallélisation selon les rayons ou selon les données.

7.3 Approches de parallélisation

Pour identifier les différentes possibilités de parallélisation, on identifie tout d'abord les différents niveaux de boucles du processus de calcul de l'Ambient Occlusion, soit trois principaux niveaux de boucles. On itère tout d'abord sur l'ensemble des rayons que l'on souhaite calculer (32 dans notre cas), puis sur l'ensemble des voxels du jeu de données et finalement sur chaque échantillon du rayon. La première étape est réalisée côté CPU, les suivantes côté GPU.

Algorithme 7.1 Différents niveaux de boucles pour le lancer de rayon.

```
CPU:  Pour toutes les directions de rayons
GPU:    Pour chaque voxel du volume de données
GPU:      Pour chaque échantillon d'un rayon
GPU:        accumulation de l'opacité le long d'un rayon
```

Un des avantages du calcul de l'Ambient Occlusion est qu'il n'y a pas de dépendance de calcul d'un voxel à l'autre. On peut donc paralléliser les deux premiers niveaux de boucle sans risque de modifier la correction du résultat. Concernant la dernière boucle, il y a une dépendance d'ordre entre les échantillons successifs que l'on choisit, depuis le voxel et le long du rayon. Cette dépendance est due au fait que l'on utilise l'opérateur OVER de PORTER ET DUFF [PD84] pour combiner les échantillons. Il faut donc respecter cet ordre de recomposition, si on choisit de paralléliser selon cette boucle. Les stratégies de répartition sur les différents GPUs varient donc en fonction du niveau de boucle que l'on va paralléliser avec les GPUs (cf. Figure 7.7).

Si on choisit de paralléliser selon les rayons, chaque GPU va donc être en charge

du calcul d'une sous-partie des rayons. Ainsi si on dispose de n GPUs et que l'on souhaite faire calculer m rayons, on constitue n ensembles de $\lfloor n/m \rfloor$ rayons (si $k = n \,\%\, m$ est différent de 0, on répartit alors les k rayons restants dans les m ensembles). Chaque GPU est ensuite en charge du calcul d'un des n ensembles de rayons. Une fois ces ensembles calculés, il faut recourir à une phase de recombinaison qui va générer des temps de transfert et de calcul additionnels. Si le CPU est en charge de cette phase, il y aura alors transfert de chaque volume d'occultation calculé pour les n ensembles et le CPU aura la charge de recombiner ces ensembles en un unique volume. Pour cela, on somme les m valeurs associées à chaque voxel v et on les divise par le nombre de rayons. Si un des GPUs est désigné, l'ensemble des données des $n-1$ autres GPUs devra être transféré sur le CPU, puis vers le GPU désigné pour effectuer le même processus qu'évoqué précédemment sur le CPU. On peut aussi envisager des approches hiérarchiques entre GPUs pour la combinaison des rayons. Néanmoins, cette première stratégie a pour inconvénient de devoir transférer un volume de données équivalent à la taille du jeu de données pour chaque GPU, ainsi que de nécessiter une phase de recombinaison supplémentaire.

La seconde stratégie de parallélisation consiste à paralléliser selon les données. Si on a n GPUs, on découpe le jeux de données en n sous-volumes, et chaque GPU calcule les valeurs d'occultation pour un de ces sous-volumes et un même ensemble de rayons partagé entre les GPUs. Elle ne requiert donc de transférer qu'une sous-partie du volume vers le GPU qui sera en charge de la phase de rendu et permet ainsi d'alléger les temps de transfert en supprimant la phase de recombinaison.

On peut aussi imaginer des stratégies hybrides entre les deux approches évoquées, mais nous nous sommes concentrés sur la seconde approche qui consiste à paralléliser selon les données, vu l'avantage que semble avoir cette méthode.

Dans la suite, nous nous sommes tout d'abord intéressés à une approche de répartition des données statique en section 7.4. Dans cette première approche, on affecte à chaque GPU une partie des données sur laquelle il sera en charge de calculer les facteurs d'occultation. Dans un second temps, nous avons expérimenté une approche dynamique de répartition des données pour laquelle on découpe le jeu de données en sous-volumes de taille fixe (le nombre de sous-volumes est supérieur au nombre de GPUs). Chaque sous-volume à calculer est ensuite envoyé à la volée à un GPU disponible pour calcul. Cette seconde stratégie suit le modèle producteur-consommateur et est exposée en section 7.5.

7.4 Parallélisation statique

Dans cette première section, nous décrivons la parallélisation avec répartition statique des données. Celle-ci s'appuie sur l'utilisation conjointe d'OpenMP et de CUDA. L'API CUDA permet de créer un contexte associé à chaque GPU présent sur la machine de travail. Les sections parallèles d'OpenMP permettent ensuite de lancer plusieurs threads d'exécution auxquels on attache un contexte CUDA, si il est nécessaire d'effectuer des opérations sur les GPUs. La machine de test étant équipée d'un processeur quad-core, on peut effectuer des opérations sur les 4 GPUs en même temps.

7.4.1 Découpage initial

Comme évoqué dans la section 7.3, le découpage et la répartition des blocs à calculer vont être effectués en se basant sur les données. On découpe le bloc de

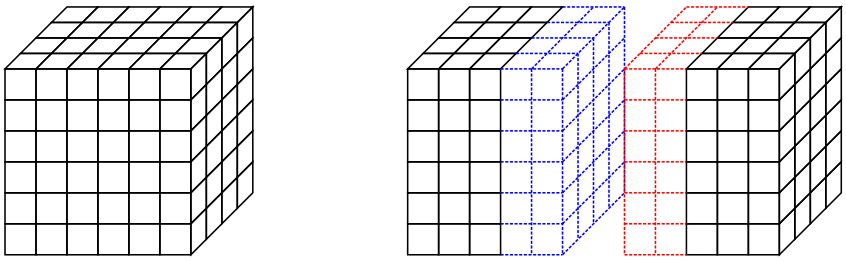


FIGURE 7.8 – En découpant le jeu de données en sous-volumes pour traitement par plusieurs GPUs (ici 2), on prend en compte un dépassement de la taille du rayon de sphère R_Ω considéré (ici 2 voxels).

données en n sous-blocs qui sont distribués aux n GPUs présents sur la machine. Chacun calcule les facteurs d'occultation pour le sous-bloc qui est pré-chargé dans sa mémoire.

Découpage des données – Le découpage des données en blocs peut être effectué de différentes manières. Nous avons choisi de subdiviser en 2 et récursivement le bloc de données initial jusqu'à atteindre un nombre de blocs égal au nombre de GPUs. A chaque étape, on peut subdiviser le bloc de données selon chacune des dimensions : X, Y ou Z. Nous avons choisi pour cette partie de subdiviser l'ensemble du jeu de données selon la direction Z. Les données étant stockées en mémoire dans l'ordre X, Y puis Z, découper les données selon l'axe Z permet de profiter au mieux des caches optimisés des cartes graphiques pour l'accès aux textures (notamment en deux dimensions).

Dépassement de voxels – Le point suivant est important par rapport au découpage des données : l'algorithme de calcul des facteurs d'Ambient Occlusion lance des rayons dans une sphère autour de chaque voxel et d'un certain rayon R_Ω pour des directions choisies aléatoirement. En découpant les données en sous-volumes, il faut donc s'assurer que pour tous les voxels du sous-bloc l'ensemble des voxels situés dans la sphère d'influence est accessible. Pour cela, il faut transférer les sous-volumes de données en considérant un dépassement ("overlap") de R_Ω voxels, comme illustré en Figure 7.8. Sur cette Figure, on découpe le jeu de données en deux selon l'axe vertical. Les voxels initiaux sont matérialisés par les cases noires et ceux de l'overlap par les cases bleues et rouges.

Ceci va donc générer une augmentation du temps de transfert liée à la taille du rayon R_Ω. Néanmoins, les temps de transfert résultants restent toujours marginaux par rapport aux temps de calcul de l'Ambient Occlusion, et n'ont donc pas une incidence sensible sur le temps total.

Parallélisation du bricking – Pour accélérer encore le processus, on parallélise aussi le calcul du bricking sur l'ensemble des GPUs à disposition, afin que chacun calcule la structure de Bricking pour le sous-volume dont il a la charge. Il n'y a pas de redondance dans ce cas, car on ne considère que les voxels utiles dans le calcul de chaque structure de bricking, c'est-à-dire les voxels qui n'appartiennent pas à l'overlap.

Processus global – La Figure 7.9 décrit le processus de parallélisation pour 2 GPUs. On répartit tout d'abord les sous-volumes du jeu de données sur les différents GPUs en tenant compte du rayon R_Ω. Puis on calcule la structure de Bricking. Les deux sous-volumes sont ensuite traités pour calculer les facteurs d'Ambient Occlusion. Une fois cette étape terminée, il faut rapatrier côté CPU l'ensemble des

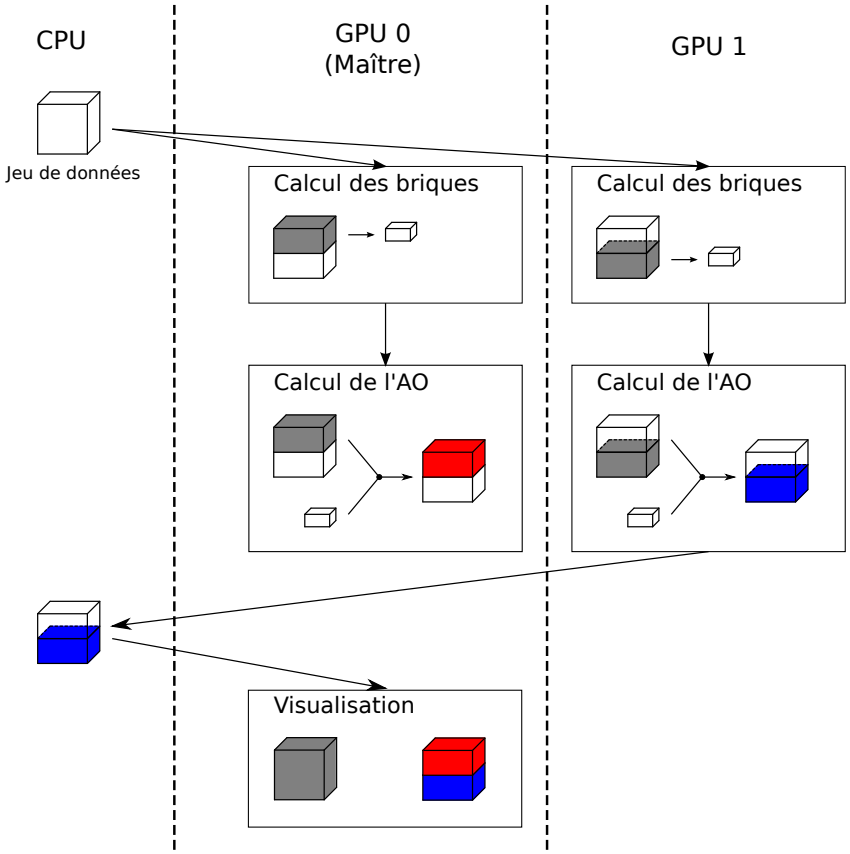


FIGURE 7.9 – Illustration du processus statique de calcul de l'Ambient Occlusion sur architecture multi-GPU dans le cas de deux GPUs.

valeurs d'occultation pour les sous-blocs qui ne sont pas sur le GPU en charge du rendu, puis les transmettre à ce dernier pour qu'il puisse générer une image en utilisant l'ensemble des facteurs d'occultation.

7.4.2 Résultats intermédiaires

La Figure 7.10 présente les temps mesurés pour cette répartition statique des données. Les 4 graphes du haut correspondent à la parallélisation sans bricking. Ils indiquent le temps total mesuré T_{total}, incluant l'ensemble du processus avec l'étape de lancer de rayons et les transferts de données, et les temps mesurés sur chaque GPU en parallèle ($T_{GPU0\,RC}$, $T_{GPU1\,RC}$, $T_{GPU2\,RC}$ et $T_{GPU3\,RC}$). Les 4 graphes du bas illustrent la parallélisation statique en utilisant le bricking, avec le temps total T_{total}, les temps par GPU, ainsi que le temps pour calculer la structure du bricking.

On constate que la parallélisation permet de réduire considérablement le temps de calcul de l'Ambient Occlusion dans l'ensemble des cas. Les temps totaux sont très proches des temps mesurés pour le lancer de rayon, ce qui indique que les temps de transferts ne sont pas très pénalisants. Ils restent de toute manière marginaux par rapport aux temps mesurés pour le lancer de rayon, tout comme le temps de calcul de la structure du bricking.

Le tableau 7.1 présente des résultats chiffrés liés aux graphes précédents. Ici les temps mesurés sont les temps cumulés du lancer de rayon. On constate qu'en passant de 1 à 4 GPU, on a une réduction quasiment linéaire du temps de calcul pour chaque rayon. Le bricking permet encore d'accentuer cette effet en réduisant

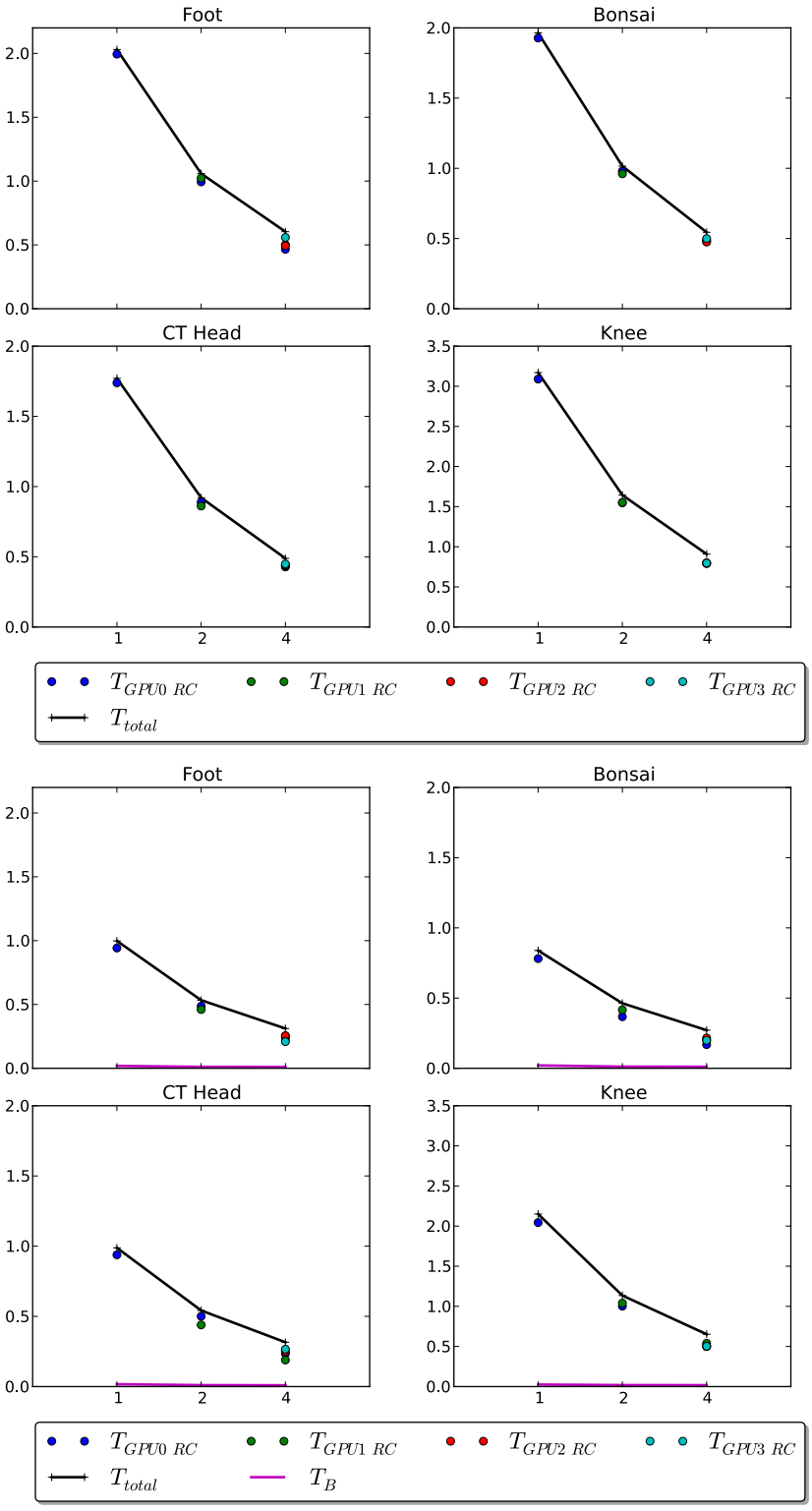


FIGURE 7.10 – Mesures de temps sans (4 graphes du haut) et avec bricking (4 graphes du bas) pour 1, 2 et 4 GPUs.

le temps de pré-calcul de l'Ambient Occlusion pour un jeu de données de taille 256^3 de l'ordre de 250 à 300 millisecondes.

On constate qu'une différence de calcul pour un rayon d'une milliseconde entre 2 GPUs introduit des variations dans les temps de calcul entre les GPUs considérés. Ce phénomène est du au fait que le processus est itératif selon les rayons. Ainsi si il y a une différence d'une milliseconde entre 2 GPU, on va multiplier cette différence par les n rayons que l'on va lancer, créant ainsi un écart dans les temps de calcul des différents GPUs. Cette différence est d'autant plus importante que la taille des données augmente.

7.4.3 Équilibrage de charge

En fonction de la classification obtenue par la fonction de transfert, il se peut que de grandes parties du jeu de données soient rendues transparentes. Le bricking va permettre d'éviter de calculer des valeurs d'occultation dans ces zones. La distribution de ces briques vides dépend du jeu de données et de la fonction de transfert. En subdivisant le jeu de données en sous-blocs, on obtient des temps de calcul entre blocs qui diffèrent, et un mauvais équilibrage de charge entre les différents processeurs, dans la mesure où certains sous-blocs contiennent plus de briques vides que d'autres.

Pour améliorer la répartition de la charge lors du processus de lancer de rayons, on peut envisager une approche statique, qui consiste à trouver un critère d'équilibrage et équilibrer avant d'effectuer le lancer de rayon, ou une approche dynamique, qui consiste à équilibrer progressivement la charge des différents GPUs en fonction des calculs précédents. Nous n'avons pas choisi d'utiliser la seconde solution, car elle implique une réorganisation des données sur les différents GPUs à chaque rayon lancé pendant la phase d'équilibrage, ce qui peut se révéler pénalisant en terme de temps de transfert et de combinaison avec des valeurs précédemment calculées.

Pour améliorer la répartition de charge entre les différents GPUs, nous proposons donc de modifier la distribution des blocs sur les différents GPUs, afin que le calcul de l'Ambient Occlusion prenne approximativement le même temps sur chaque GPU. Pour mettre en place l'équilibrage de charge, nous avons d'abord besoin d'un critère afin d'évaluer le coût de calcul d'un bloc donné. Ce critère est lié au nombre de briques pleines dans chaque bloc. Nous avons donc besoin de compter le nombre de briques dans un bloc donné et cette tâche peut être effectuée en parallèle sur chaque GPU. Une fois que ce critère est établi, on l'utilise pour calculer une subdivision qui résulte en une charge de travail équivalente pour chaque GPU. Les sous-sections suivantes décrivent le critère d'équilibrage choisi et comment l'évaluer.

Critère d'équilibrage

Description – Soit un bloc B_i contenant n_f^i briques pleines et n_e^i briques vides. Le temps de calcul d'un bloc vide est noté par T_e et le temps de calcul des facteurs d'Ambient Occlusion pour une brique pleine est noté par T_f. Pour chaque voxel traité, T_e est petit, car il correspond uniquement au temps de vérification requis pour savoir si une brique est vide ou non, alors que T_f est considéré comme constant et est dépendant des paramètres de calcul de l'Ambient Occlusion. Le temps de calcul de l'Ambient Occlusion pour un bloc B_i est alors $T = n_e^i \times T_e + n_f^i \times T_f$. Comme T_e n'est pas égal à 0, le critère d'équilibrage ne peut pas prendre en compte uniquement le nombre de briques pleines. Le critère que nous utilisons pour un bloc B_i est ainsi : $C_1 \times n_f^i + C_2 \times n_e^i$. Ce critère permet de tenir compte du nombre de briques vides lors

Sans Bricking												
	1 GPU		2 GPU			4 GPU						
	Total	1 rayon	Total		1 rayon		Total			1 rayon		
Foot	1.994	0.062	0.993	1.022	0.031	0.032	0.465	0.500	0.494	0.558	0.014	0.015	0.015	0.017
Bonsai	1.927	0.060	0.980	0.960	0.030	0.030	0.481	0.487	0.474	0.499	0.015	0.015	0.015	0.015
Head	1.738	0.054	0.889	0.861	0.027	0.027	0.428	0.435	0.442	0.448	0.013	0.013	0.014	0.014
Knee	3.090	0.096	1.548	1.551	0.048	0.048	0.798	0.790	0.797	0.796	0.025	0.025	0.025	0.025

Avec Bricking												
	1 GPU		2 GPU			4 GPU						
	Total	1 rayon	Total		1 rayon		Total			1 rayon		
Foot	0.942	0.030	0.486	0.460	0.015	0.014	0.234	0.252	0.255	0.209	0.007	0.008	0.008	0.006
Bonsai	0.781	0.024	0.367	0.416	0.011	0.013	0.168	0.201	0.215	0.198	0.005	0.006	0.007	0.006
Head	0.938	0.029	0.500	0.439	0.015	0.013	0.235	0.189	0.250	0.266	0.007	0.006	0.008	0.008
Knee	2.043	0.063	1.001	1.040	0.031	0.032	0.501	0.539	0.499	0.502	0.016	0.017	0.016	0.016

TABLE 7.1 – Tableau récapitulant les temps mesurés en secondes sans (en haut) et avec bricking de taille 4^3 (en bas) pour 1, 2 et 4 GPUs. Pour chaque configuration, "Total" signifie le temps total cumulé sur l'ensemble des rayons et "1 rayon", le temps moyen par rayon et par GPU.

de l'évaluation des temps de calcul. Nous avons expérimentalement trouvé 0.75 et 0.25 comme de bonnes valeurs pour C_1 et C_2 pour notre configuration multi-GPU. Avec ce critère, on peut évaluer le déséquilibre entre deux blocs différents B_i et B_j $(i \neq j)$ en calculant la différence : $D(B_i, B_j) = (C_1 \times n_f^i + C_2 \times n_e^i) - (C_1 \times n_f^j + C_2 \times n_e^j)$. Le but de l'équilibrage de charge est d'amener le résultat de cette formule à 0 pour toutes les paires (B_i, B_j) avec $(i, j) \in [1, n]^2$ et $i \neq j$. Mathématiquement, ceci peut être exprimé comme trouver l'ensemble des blocs $S = \{B_k | k \in [1, n]\}$ parmi $\wp$, l'ensemble de toutes les partitions possibles du jeu de données en n blocs, tel que on minimise la valeur maximale de $D(B_i, B_j)$ pour toutes les paires de blocs :

$$\begin{aligned} S &= \{B_k \mid k \in [1, n]\} \in \wp \mid M_S = \min\left\{M_{\{S_q\}} \mid S_q \in \wp\right\} \\ M_{\{S_q\}} &= \max\left\{|D(B_i, B_j)| \mid \forall (B_i, B_j) \in S_q^2,\, i \neq j\right\} \end{aligned} \tag{7.1}$$

L'équilibrage consiste donc à déterminer l'ensemble S selon l'équation 7.1. Ceci est un problème d'optimisation, qui ne peut pas être résolu facilement en utilisant par exemple la programmation dynamique, puisque le nombre de calculs serait important. Nous proposons d'utiliser un algorithme glouton basé sur des heuristiques en utilisant une procédure itérative.

Calcul – Afin d'éviter des transferts de données CPU-GPU, l'évaluation du critère est effectuée sur le GPU. Le calcul du nombre de briques pleines d'un bloc B_k est donc une opération demandée par le CPU aux différents GPUs. Comme décrit en section 7.2, chaque GPU a calculé les briques pour la partie des données dont il a la charge. Chacun est alors capable de calculer le nombre de briques pleines de son ensemble (ou une sous-partie de celui-ci). Comme un bloc B_k donné peut correspondre à des briques réparties sur un ou plusieurs GPUs, le calcul doit aussi être réparti pour tenir compte de cela. En d'autres termes, pour chaque bloc à traiter, le CPU répartit les calculs pour tenir compte de l'emplacement où se trouve les briques correspondantes. Pour effectuer ce calcul, nous utilisons un algorithme de réduction optimisé provenant du SDK CUDA de NVIDIA [Har08]. Pour déterminer le nombre de briques pleines d'un bloc de briques, il suffit alors de copier la sous-partie correspondante dans une nouvelle zone mémoire, et d'exécuter la réduction sur les valeurs de cette zone. L'équilibrage de charge que nous proposons requiert l'évaluation de beaucoup d'ensembles S_q de blocks B_k^q. Compte tenu de l'algorithme itératif que nous utilisons pour déterminer les blocs (cf. sous-section suivante), ils partageront un grand nombre de briques d'une étape à l'autre. Afin d'éviter de calculer plusieurs fois le nombre de briques pleines d'un même bloc, le CPU met à jour le nombre de briques pleines de manière incrémentale. Ceci est fait en demandant aux GPUs de ne calculer que la partie du nouveau bloc qui diffère par rapport au bloc précédent, et en ajoutant/soustrayant le nombre correspondant. Seule la première étape (le choix initial des blocs) requiert de calculer le nombre de briques de l'ensemble des briques sur chaque GPU.

Heuristique d'équilibrage

La première étape consiste à obtenir un ensemble initial de blocs $\{B_k | k \in [1, n]\}$. Le découpage est fait de telle sorte que l'on conserve les blocs sous la forme de parallélépipèdes, afin que les calculs suivants soient efficaces en terme d'accès aux données et de cache. Une structure d'arbre BSP est utilisée pour découper l'espace. Par défaut, nous découpons l'espace selon l'axe Z.

En se basant sur l'ensemble de blocs obtenus avec l'arbre BSP, le critère $D(B_i, B_j)$ est utilisé pour déplacer le plan de découpe entre blocs adjacents B_i et B_j correspondant à deux frères dans l'arbre BSP, afin de converger vers un ensemble de blocs proche de l'optimum défini dans l'équation 7.1. Le processus itère sur chaque nœud de l'arbre BSP de haut en bas. A chaque nœud, on calcule le critère d'équilibrage $D(B_i, B_j)$, où B_i et B_j sont les deux blocs associés aux fils du nœud courant. Si $|D(B_i, B_j)|$ est plus grand qu'une valeur limite, on considère alors les deux blocs comme en déséquilibre de charge, et le plan de découpe entre ces deux blocs est déplacé selon le signe de D. Le plan est déplacé d'une position dans l'espace des briques vers B_i si $D > 0$ ou vers B_j si $D < 0$. Ce processus est répété pour les blocs B_i et B_j jusqu'à ce que le signe de $D(B_i, B_j)$ change.

Résultats

Pour chaque jeu de données, le tableau 7.11 présente deux lignes de mesures qui indiquent respectivement des mesures avant et après équilibrage. On retrouve les jeux de données précédents : Foot (256^3), Bonsai (256^3), CT Head ($256\times256\times225$) et Knee ($379\times229\times305$), ainsi que des jeux de données de plus grande taille : Backpack ($512 \times 512 \times 373$) et Colon Phantom ($512 \times 512 \times 442$) (illustrés en Figure 7.12). Concernant les différentes colonnes, les deux premières définissent respectivement les temps maximum et minimum mesurés parmi l'ensemble des rayons lancés. Les deux colonnes suivantes montrent le temps total pour le lancer de rayon (32 rayons lancés) et la dernière colonne représente le temps total du calcul des facteurs d'Ambient Occlusion, incluant donc les temps de transfert, le calcul du bricking et de la phase d'équilibrage. On constate que l'équilibrage de charge permet de réduire globalement les temps de calcul. Sur les jeux de données de taille plus petite, on constate que la différence entre le maximum et le minimum du temps de lancer d'un rayon est petite, ce qui la rend plus difficile à corriger que pour les jeux de données de taille plus importante. Dans ces derniers cas, on part d'une différence de l'ordre de 15 à 20 millisecondes pour arriver à une différence de l'ordre de 4 à 5 millisecondes. Dans le cas des petits jeux de données, la réduction du temps global n'est pas significative du fait de l'introduction des calculs liés à l'équilibrage de charge. Elle est par contre plus significative pour les jeux de taille plus importante, même si on ne réduit le temps par rayon que de quelques millisecondes, puisqu'on arrive à des réductions de l'ordre de la centaine de millisecondes.

La Figure 7.13 illustre sous forme d'histogrammes les réductions de temps de calcul avec l'équilibrage de charge. Pour chaque jeu de données, les 4 histogrammes de gauche représentent les temps mesurés sans équilibrage et ceux de droite avec équilibrage sur les 4 GPUs. Sur les graphes de droite, les histogrammes rouges et verts représentent respectivement l'augmentation ou la réduction du temps de calcul par rapport aux histogrammes de gauche, donc après équilibrage. Les lignes bleues et vertes représentent les temps totaux sans et avec équilibrage. L'équilibrage permet dans ces cas d'avoir une répartition plus homogène des temps de calcul pour les différents jeux de données et de réduire globalement le temps d'exécution du processus complet.

	Un rayon		Total RC		Total
	min	max	min	max	
Foot	0.006538	0.007977	0.209205	0.255265	0.304479
	0.006776	0.007949	0.216847	0.254372	0.308103
Bonsai	0.005271	0.006741	0.168677	0.215699	0.264417
	0.005863	0.006419	0.187616	0.205404	0.272123
CT Head	0.005907	0.008335	0.189031	0.266706	0.316088
	0.006883	0.007954	0.220243	0.254532	0.304343
Knee	0.015619	0.016851	0.499810	0.539245	0.628329
	0.014774	0.017325	0.472755	0.554409	0.653940
Backpack	0.040594	0.060671	1.299018	1.941463	2.277240
	0.053448	0.057436	1.710347	1.837966	2.080352
Colon Phantom	0.050834	0.066926	1.626680	2.141636	2.444041
	0.061835	0.064096	1.978725	2.051084	2.337245

FIGURE 7.11 – Pour chaque jeu de données, les deux lignes représentent respectivement les mesures avant et après équilibrage. Les temps sont donnés en secondes.

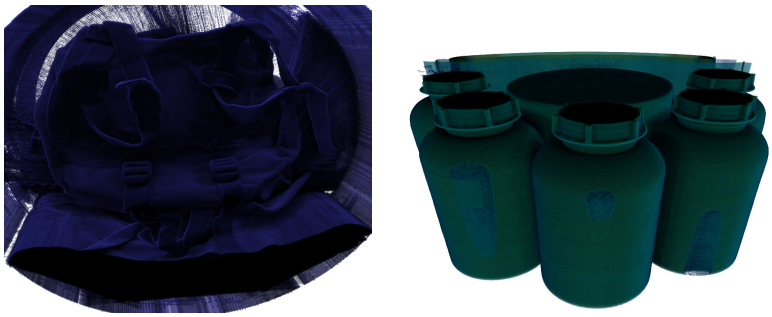


FIGURE 7.12 – Jeu de données additionnels de plus grande taille : Backpack ($512 \times 512 \times 373$) à gauche et Colon Phantom ($512 \times 512 \times 442$) à droite.

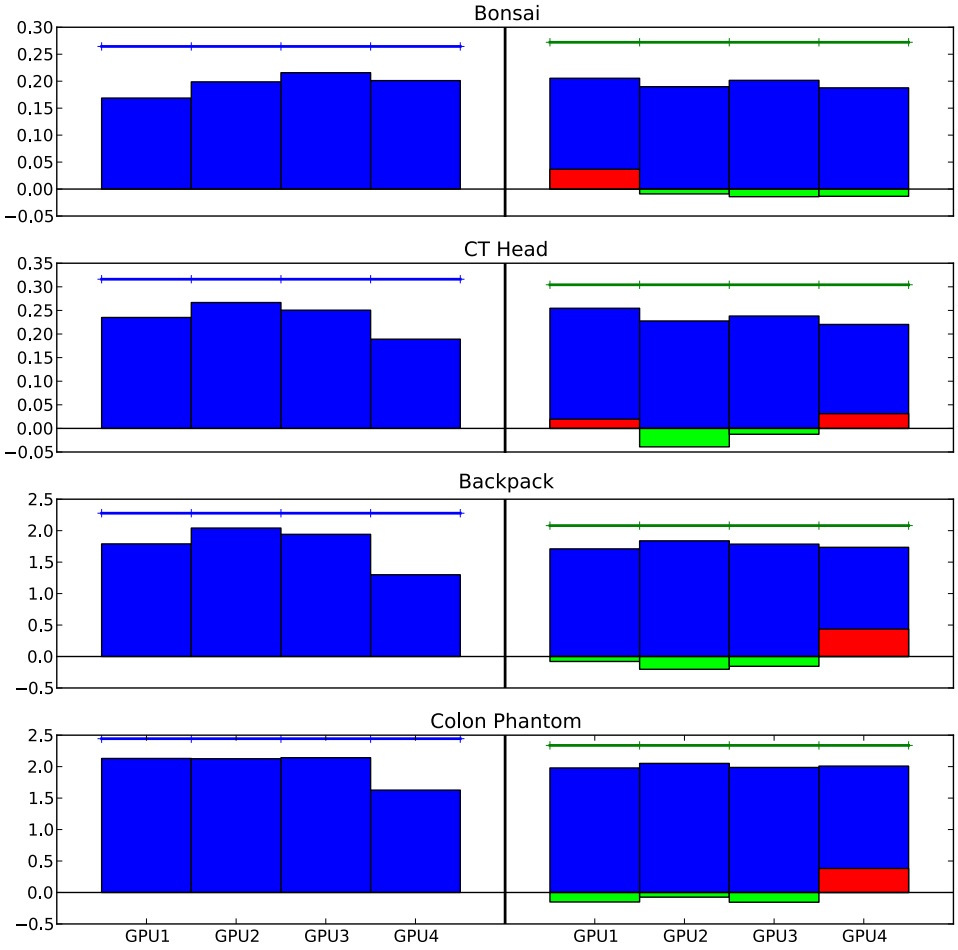


FIGURE 7.13 – Temps de calcul sans (à gauche) et avec (à droite) équilibrage de charge pour différents jeux de données.

7.5 Parallélisation dynamique

Dans un second temps, nous proposons une autre approche de parallélisation basée sur une répartition dynamique des blocs à calculer. Cette approche dynamique consiste à découper le jeu de données non plus par rapport à un nombre de blocs fixés égal au nombre de GPUs, mais par rapport à une taille de bloc. On définit pour cela une taille de bloc de base. Les blocs sont ensuite envoyés aux différents GPUs pour calcul en suivant un modèle producteur-consommateur. Cette méthode se base sur une implémentation similaire à celle de la méthode statique. OpenMP et CUDA sont toujours à la base, et les blocs sont distribués dynamiquement à l'aide d'une section critique.

Comme dans la méthode statique, la nécessité de définir un overlap de taille R_Ω pour chaque bloc est requise pour permettre le calcul des facteurs d'Ambient Occlusion. L'approche dynamique permet néanmoins de mieux prendre en compte les jeux de données de taille importante. On calcule ainsi successivement des sous-volumes qui sont stockables sur la carte graphique, alors qu'il est possible que le découpage du volume en un nombre de blocs égal au nombre de GPUs ne permet pas de placer ces blocs dans la mémoire interne des cartes graphiques du fait de leur trop grande taille. Avec cette approche, on peut ainsi adapter la taille des blocs à traiter aux capacités en terme de mémoire des GPUs disponibles, afin de pouvoir calculer l'Ambient Occlusion pour n'importe quelle taille de volume.

L'utilisation du bricking se fait de manière similaire. Nous avons choisi ici de pré-calculer l'ensemble des briques en découpant le bloc de données en autant de sous-blocs qu'il y a de GPUs. Les briques sont ensuite récupérées côté CPU. Lorsqu'on envoie un bloc de données au GPU, on envoie simultanément les briques correspondantes. La figure 7.14 résume l'ensemble des opérations réalisées pour la parallélisation dynamique.

Cette méthode permet de niveler naturellement les différences de temps de calcul entre les différentes cartes. En effet, en se basant sur le modèle producteur-consommateur, un nouveau bloc à calculer est envoyé aux cartes dès qu'elles ont fini les calculs sur leur précédent bloc. Cette méthode peut par conséquent être aussi utilisée dans une machine où les GPUs présents ne sont pas homogènes. On peut ainsi combiner des cartes puissantes avec des cartes moins puissantes, les plus puissantes seront utilisées pour traiter plus de blocs que les cartes moins puissantes.

7.5.1 Résultats intermédiaires

La Figure 7.15 présente 6 graphes. En abscisse, on trouve les tailles de blocs utilisées, allant de 32^3 à 256^3 et en ordonnée on trouve le temps de calcul en secondes. Pour chaque jeu de données, la courbe noire représente le temps total d'exécution du processus d'Ambient Occlusion. Les courbes labellisées $T_{GPUn\,RC}$ et $T_{GPUn\,Stream}$ représentent respectivement le temps du lancer de rayon pour le calcul de l'Ambient Occlusion, et le temps de transfert des blocs de données et des briques correspondantes pour le n^{e} GPU.

Une premier constat que l'on peut faire est que, même avec la parallélisation et le bricking, le lancer de rayon reste l'étape la plus coûteuse en temps de calcul. Le temps de transfert des blocs rapporté au temps de calcul global est faible lorsque la taille des blocs est suffisamment grande : 64^3 dans les exemples traités. Par contre, lorsque la taille des blocs est petite (32^3), la part du transfert devient plus importante. Ceci

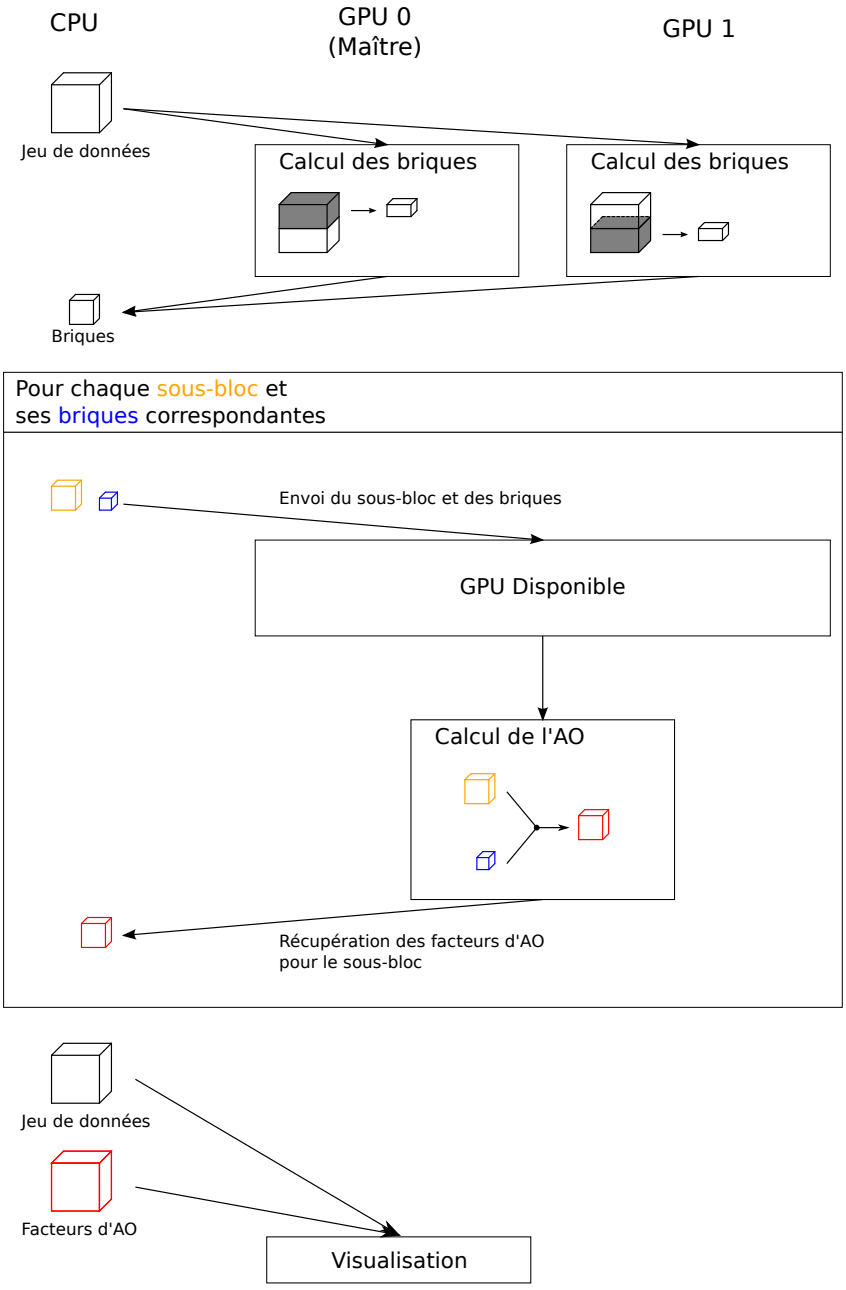


FIGURE 7.14 – Description du processus dynamique de calcul de l'Ambient Occlusion.

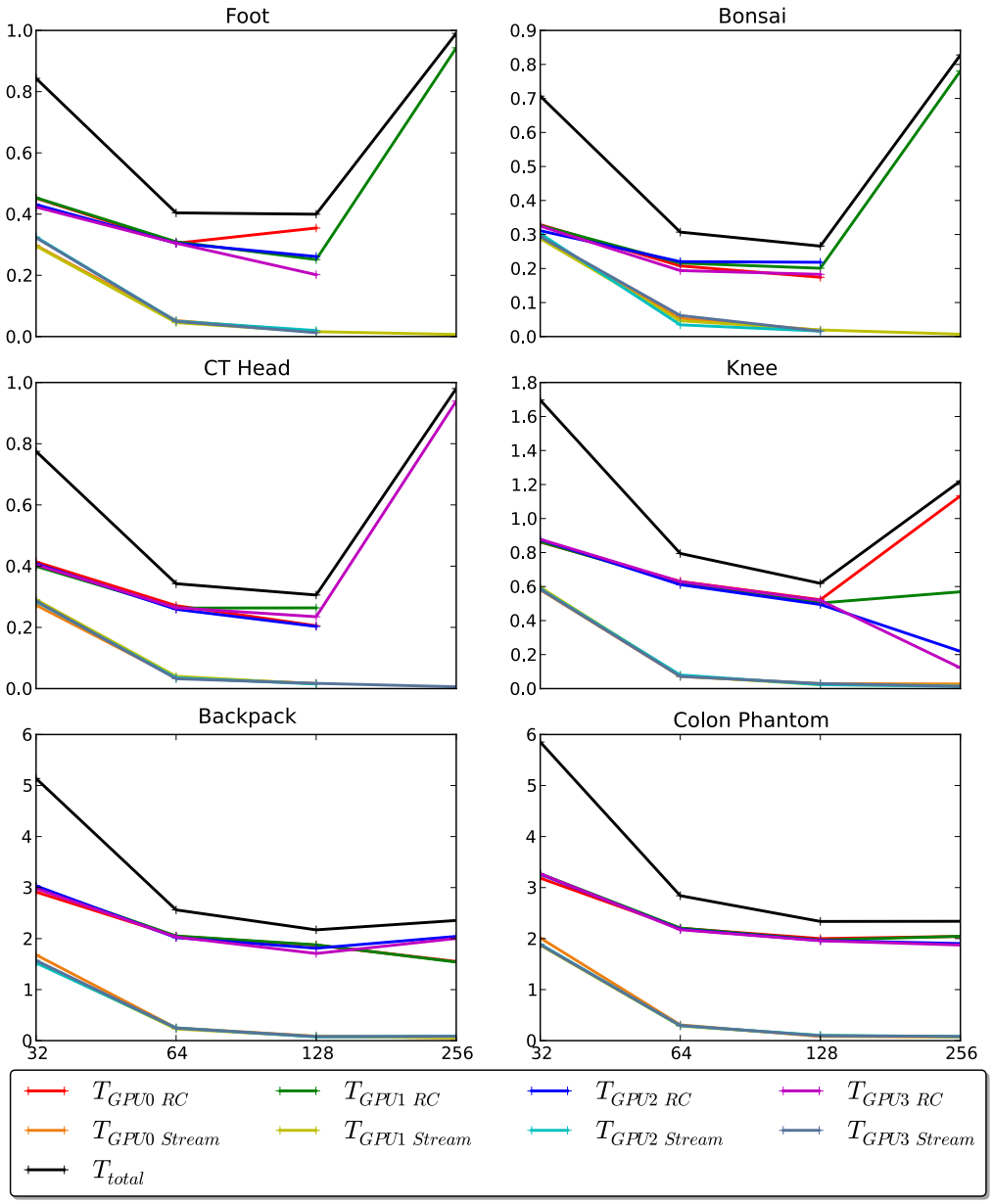


FIGURE 7.15 – Les graphes présentent l'utilisation de la méthode dynamique pour différentes tailles de blocs. Ttotal indique le temps total d'exécution du processus de calcul. $T_{GPUn\,RC}$ et $T_{GPUn\,Stream}$ indiquent respectivement le temps du lancer de rayon et le temps de transfert des blocs pour le n^{e} GPU.

est du au fait qu'un jeu de données de 256^3 donnera 64 blocs de 64^3 et 512 blocs de 32^3 auquel il faut rajouter l'overlap pour le lancer de rayon dû à R_Ω. Ceci se révèle pénalisant si on a un nombre important de blocs comme pour la taille de bloc 32^3. Pour certains jeux de données, nous n'avons qu'une mesure pour la taille 256^3, ceci est du au fait que le jeu de données a justement une taille de 256^3 ou inférieure, on a donc un unique bloc à traiter et un seul GPU en charge. C'est pour cela que le temps de calcul dans ces cas revient au temps mono-GPU avec bricking. Pour les jeux de données dont la taille s'approche de 512^3 (les deux dernières lignes), il devient intéressant d'utiliser une taille de 256^3, alors que pour les jeux de données plus petits, la taille 128^3 semble la plus appropriée.

7.6 Synthèse : méthodes statique et dynamique

La Figure 7.16 permet de comparer les temps globaux des méthodes statique et dynamique. La courbe verte ($T_{dynamic}$) représente le temps total de calcul pour différentes tailles de blocs dans l'approche dynamique. La courbe bleue (T_{static}) est le temps total mesuré pour la méthode statique en utilisant le bricking et 4 GPUs. On peut constater qu'elles proposent des temps similaires à partir des tailles de blocs 64^3 pour la méthode dynamique. La méthode dynamique surpasse même la statique pour certains cas avec des blocs de taille 128^3. Dans la méthode dynamique, la taille 256^3 n'est représentative que pour les jeux de données de taille supérieure à 256^3, car dans les autres cas, on utilise qu'un seul GPU. Il semblerait que le meilleur compromis dans la méthode dynamique soit atteint quand on découpe le jeu de données en 2 selon chaque dimension.

7.7 Parallélisation de la méthode d'Ambient Occlusion par structure par Multi-GPU

Le chapitre 6 introduit la technique d'Ambient Occlusion par structure, dont l'objectif est d'obtenir de meilleurs résultats visuels, c'est-à-dire des images permettant de conserver les ombres douces mettant en valeur crêtes et vallées dans les surfaces et en supprimant les ombres de contact inter-structures qui assombrissent la visualisation. Pour rappel, cette technique consiste à séparer l'information au sein du jeu de données en se basant sur un découpage de la fonction de transfert. L'Ambient Occlusion est ensuite calculée par structure, puis recombinée afin d'éliminer les occultations inter-structures. Cependant, cette technique a un coût en terme de temps de calcul plus important que l'Ambient Occlusion classique, car il est directement proportionnel au nombre de structures définies dans le jeu de données. Nous avons étudié la parallélisation multi-GPU de la méthode d'Ambient Occlusion par structure développée dans cette thèse.

La méthode d'Ambient Occlusion par structure est décrite par l'algorithme 7.2. Il présente les différents niveaux de boucles, ajoutant un niveau supplémentaire pour les structures à calculer.

On peut alors décider de paralléliser selon la boucle supplémentaire sur les structures. En n'utilisant pas le bricking, on peut indifféremment paralléliser selon les structures ou selon les données comme proposé en section 7.3. En effet, comme on calcule l'Ambient Occlusion pour l'ensemble des voxels à chaque fois, le temps de calcul de chaque structure est sensiblement le même et ne crée pas de différences entre les GPUs. En utilisant le bricking par contre, la figure 7.6 montre que l'on

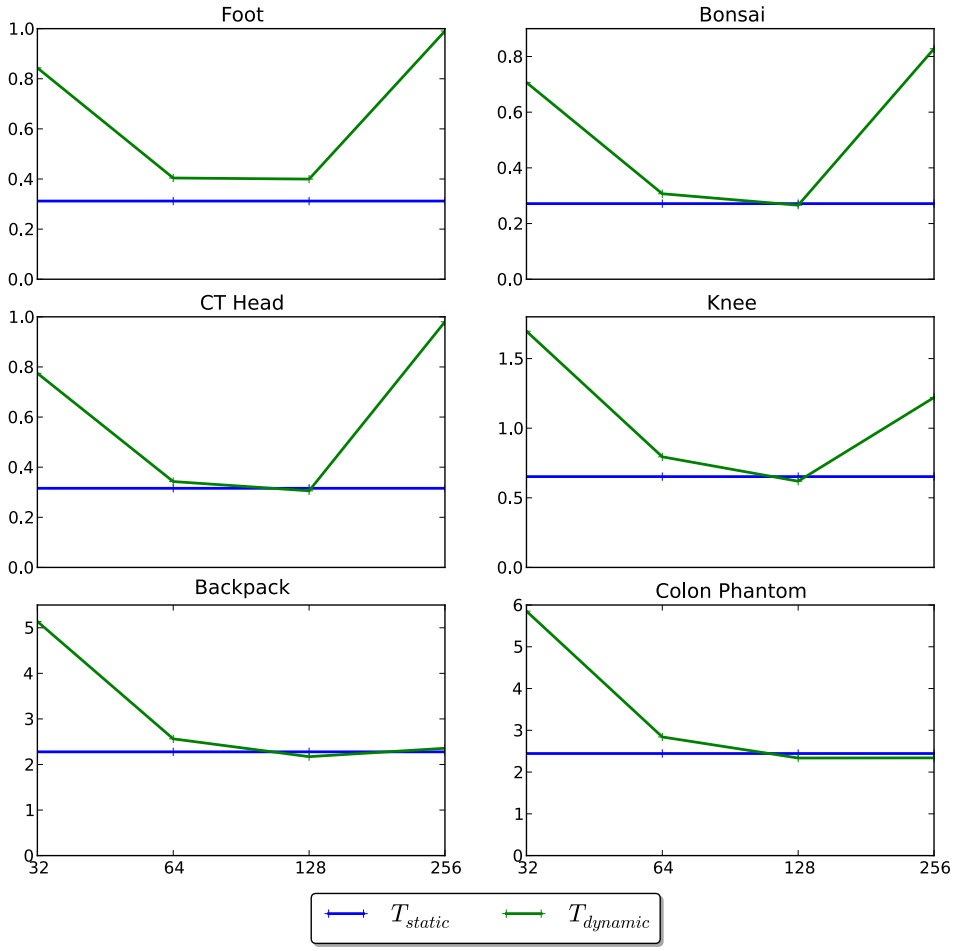


FIGURE 7.16 – Comparaison des temps de calcul entre méthode statique (T_{static}) et dynamique ($T_{dynamic}$).

Algorithme 7.2 Nouvelle version de l'algorithme 7.1 en incluant le calcul par structure.

```
CPU:  Pour toutes les stuctures
CPU:    Pour toutes les directions de rayons
GPU:      Pour chaque voxel du volume de données
GPU:        Pour chaque échantillon d'un rayon
GPU:          accumulation de l'opacité le long d'un rayon
```

obtient des temps de calcul différents selon la fonction de transfert. Pour assurer une bonne répartition de charge et ainsi obtenir des temps de calcul équivalents, il vaut mieux paralléliser selon les données.

7.7.1 Méthode statique

Dans le contexte de la méthode de parallélisation avec répartition statique, nous parallélisons selon les données comme l'approche décrite en section 7.4. Se rajoute alors le niveau de boucle supplémentaire sur les structures. La Figure 7.17 montre l'amélioration des performances que l'on peut attendre de l'utilisation du bricking et du multi-GPU. Sur le graphe du haut, chaque barre verticale est composée de plusieurs barres verticales représentant le temps de calcul pris pour chaque structure calculée (orange, pastel et vert clair) par GPU. Les lignes respectivement bleue, verte et rouge représentent le temps global de calcul de l'Ambient Occlusion, incluant les temps de transfert et de bricking, pour chaque configuration. La barre de gauche représente le temps mis pour un GPU sans utiliser de bricking, celle du milieu le temps mis par un GPU avec bricking, et les 4 de droite représentent le temps mis par chaque GPU dans une configuration à 4 GPUs avec bricking. Dans le cas de 2 ou 3 structures, l'utilisation du multiGPU et du bricking permettent de réduire considérablement les temps de pré-calcul de l'Ambient Occlusion par structure, puisqu'on passe d'un temps supérieur à plusieurs secondes à un temps autour de la seconde pour des jeux de données de taille 256^3.

Les graphes du bas permettent de comparer le temps de calcul de l'Ambient Occlusion classique avec l'Ambient Occlusion par structure. Pour un jeu de données de taille 256^3, on arrive à conserver un temps de calcul inférieur à la seconde même avec 3 structures (voir le jeu de données "CT Head"). On peut ainsi obtenir une meilleure perception dans les données en utilisant l'Ambient Occlusion par structure et réduire considérablement le temps de pré-calcul de cette technique avec le multi-GPU et le bricking.

7.7.2 Méthode dynamique

La méthode de parallélisation dynamique appliquée à l'Ambient Occlusion par structure s'appuie sur le même fonctionnement que celui de la méthode dynamique évoquée en section 7.5. On calcule des facteurs d'Ambient Occlusion pour des sous-blocs du jeu de données. De plus, nous avons choisi de traiter les structures séparément et ainsi de calculer l'ensemble du volume d'occultation pour une structure avant de passer à la suivante. Une alternative aurait pu être d'envoyer un bloc, puis de calculer ses facteurs d'occultation pour l'ensemble des structures identifiées, et finalement de renvoyer ces valeurs coté CPU dans la zone mémoire correspondant respectivement à chaque structure. Néanmoins vu le coût limité d'envoi des blocs, le gain ne serait pas globalement perceptible.

Le graphe du haut de la Figure 7.18 présente la réduction du temps de calcul de l'Ambient Occlusion par structure de la même manière que la Figure 7.17 pour un GPU sans bricking, un GPU avec bricking et 4 GPUs avec bricking. Le graphe du bas compare les Ambient Occlusion classique et par structure. On fait ici le même constat que pour l'approche statique. L'utilisation du multi-GPU et du bricking permettent avec l'approche dynamique de réduire les temps de calcul en deçà de la seconde, et de rendre beaucoup plus accessible la technique d'Ambient Occlusion par structure.

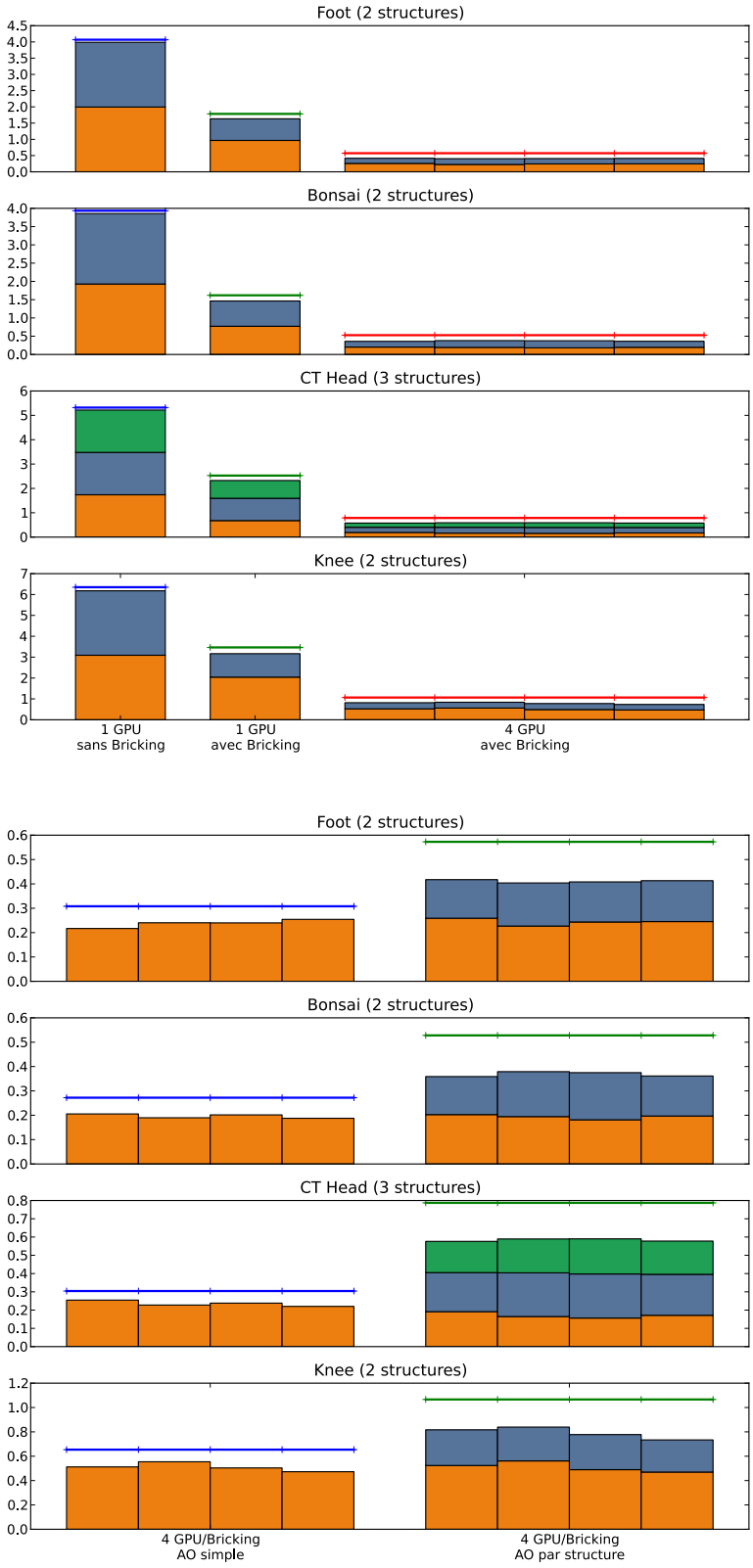


FIGURE 7.17 – Les graphes du haut présentent des histogrammes montrant l'utilisation du multiGPU dans le cadre de l'Ambient Occlusion par structure avec la méthode statique et différentes configurations. Les graphes du bas comparent l'Ambient Occlusion classique avec l'Ambient Occlusion par structure sur 4 GPUs en utilisant le bricking. Les temps sont donnés en secondes.

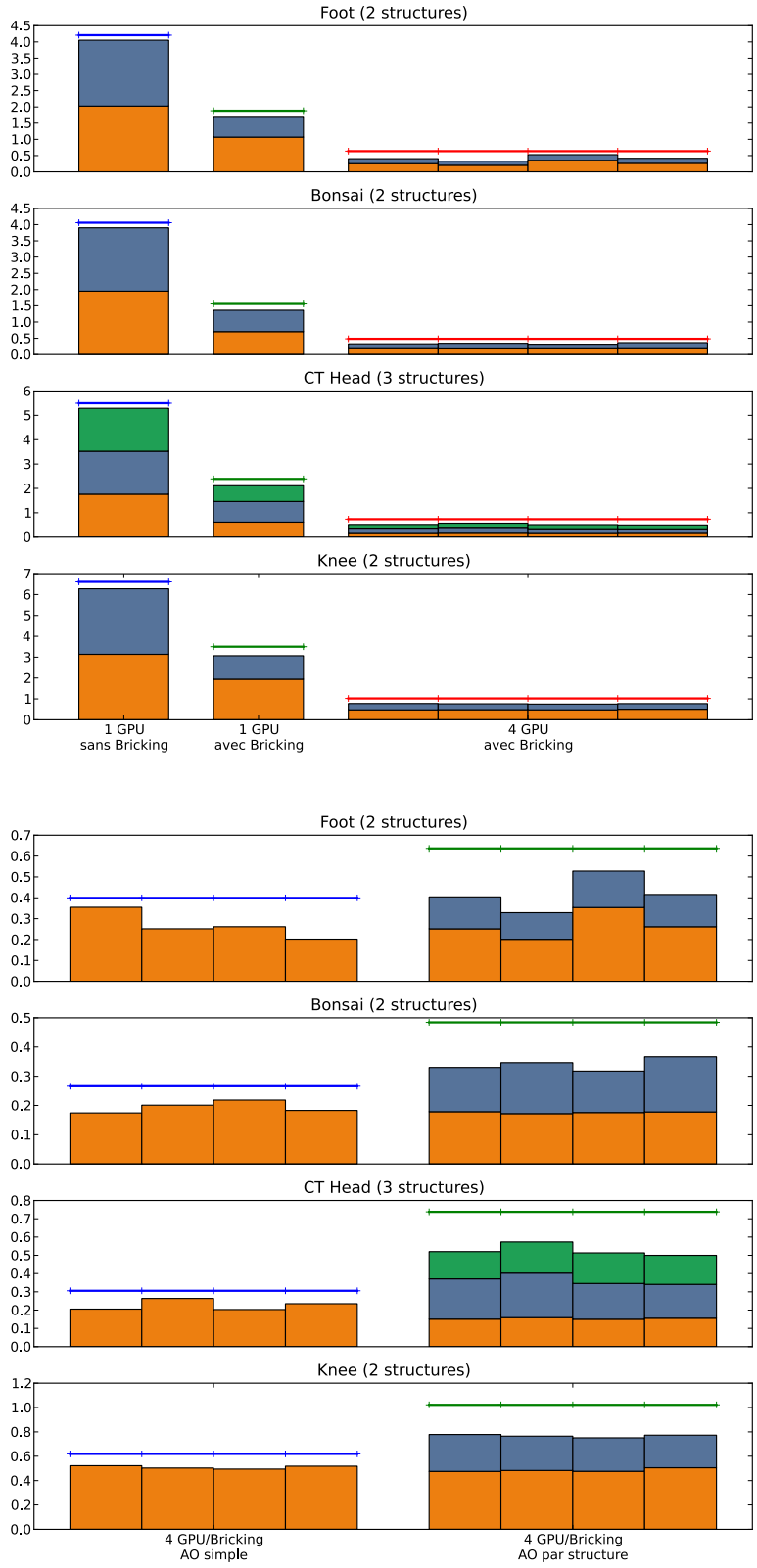


FIGURE 7.18 – Les graphes du haut présentent des histogrammes montrant l'utilisation du MultiGPU avec l'Ambient Occlusion par structure, la méthode dynamique (la taille de bloc utilisée est de 128^3) et pour différentes configurations. Les graphes du bas comparent l'Ambient Occlusion classique avec l'Ambient Occlusion par structure sur 4 GPU en utilisant le bricking. Les temps sont donnés en secondes.

7.8 Conclusion

Dans ce septième chapitre, nous avons présenté différentes techniques permettant d'accélérer le calcul de l'Ambient Occlusion. En introduisant un découpage en briques, le pré-calcul est accéléré en fonction de la proportion de briques pleines dans le volume. L'ensemble des pré-calculs est effectué côté GPU, afin de minimiser les temps de transfert. L'utilisation d'une architecture multi-GPU permet de réduire encore les temps de pré-calcul en répartissant les données à calculer sur les différents processeurs, soit de manière statique en attribuant une partie des données à chaque GPU, soit de manière dynamique en découpant les données en blocs et en distribuant aux différents processeurs pour calcul. On constate que ces deux approches produisent des résultats similaires pour l'Ambient Occlusion classique et par structure, rendant ainsi beaucoup plus accessible ces techniques. Il est à noter aussi que nous avons utilisé un nombre de rayons lancés et un rayon de la sphère d'influence assez élevé. En réduisant ces paramètres, on se rapprocherait encore plus de l'interactivité. Passer à des cartes graphiques plus récentes devrait aussi permettre une gain de performance supplémentaire. Nous avons également constaté un bon passage à l'échelle ce qui laisse supposer qu'une machine équipée de 8 GPUs permettrait déjà d'atteindre l'objectif visé : à savoir une manipulation interactive de la fonction de transfert.

Part III

Conclusion

Synthèse des contributions

Dans cette thèse, nous nous sommes intéressés à améliorer différentes techniques d'éclairages pour le rendu volumique direct. Ces améliorations se sont concentrés sur deux aspects : la qualité visuelle et la performance des méthodes.

Qualité visuelle

Les contributions en terme d'amélioration de la qualité visuelle se sont faites sur deux plans. Le premier plan est l'amélioration de l'éclairage de Phong en introduisant une nouvelle méthode d'interpolation du gradient dans le cadre du rendu volumique pré-intégré. Le second plan est l'introduction d'une méthode de calcul de l'Ambient Occlusion par structure.

Rendu volumique direct pré-intégré avec interpolation non-linéaire des gradients

Dans le cadre du rendu volumique pré-intégré, l'éclairage est un point délicat à traiter. En effet, on travaille avec des paires d'échantillons et pré-calculer l'éclairage requiert de calculer une table à 8 dimensions : pour chaque échantillon, nous avons besoin de la densité et des 3 composantes du gradient. Une première approche de Engel et al. [EKE01] propose de simplifier le calcul de l'éclairage en calculant la moyenne des gradients des deux échantillons, mais ceci résulte en des discontinuités d'éclairage d'une tranche à l'autre. Une seconde approche de Lum et al. [LWM04] propose d'interpoler linéairement le gradient entre les deux échantillons par l'ajout de tables de pré-intégration supplémentaires. Cette technique permet d'obtenir une meilleur continuité dans l'éclairage, mais l'utilisation de l'interpolation linéaire provoque une perte d'une partie de l'énergie lumineuse, notamment en considérant des angles importants entre les gradients. Nous proposons donc une nouvelle méthode consistant à interpoler le gradient le long d'une courbe approximant un arc de cercle avec un polynôme de second degré. Nous décrivons la manière d'intégrer cette méthode d'interpolation dans le calcul des composantes diffuses et spéculaires de l'éclairage de Blinn-Phong. Une simplification est néanmoins nécessaire pour l'intégration dans la composante spéculaire. Pour cela, on recourt à un cône spéculaire plutôt qu'un lobe. Cette méthode peut aussi être intégrée dans d'autres méthodes, telle que l'éclairage non-photoréaliste de Gooch et al. [GGSC98].

La meilleure interpolation du gradient implique que l'on perde moins d'énergie dans le calcul de l'éclairage. Ceci permet ainsi de mieux mettre en valeur les reflets spéculaires et d'avoir une meilleure continuité dans l'éclairage, en témoigne les comparaisons avec des images de références, consistant en un lancer de rayon suréchantillonné. Cette méthode a cependant un coût avec une baisse de performance de 10 à 20% lors de la visualisation, de par les accès textures additionnels qu'elle génère.

Ambient Occlusion par structure

Notre seconde contribution se place dans le contexte de l'Ambient Occlusion appliquée au rendu volumique. L'Ambient Occlusion est une technique permettant traditionnellement de mettre en valeur les "vallées" et "crêtes" dans les surfaces. En appliquant une telle technique au rendu volumique direct, dans lequel un ensemble de structures semi-transparentes sont imbriquées, cela résulte souvent en

des images sur-assombries. On perd en effet en visibilité pour les structures internes, car elles sont occultées par des structures plus externes ou par la projection d'ombres de contact. Nous proposons donc d'éliminer le phénomène d'occultation inter-structures. Pour cela, nous partons d'un découpage initial des données en structures, découpage ici réalisé à partir de la fonction de transfert. Puis nous calculons les facteurs d'Ambient Occlusion pour chaque structure, de sorte que les facteurs résultants ne dépendent que des structures elles-mêmes. Finalement, nous recombinons ces facteurs lors d'une dernière passe, afin d'avoir un unique volume de facteurs d'occultation utilisé lors du rendu volumique.

Nos résultats montrent que l'on améliore la visibilité des structures internes en préservant les caractéristiques de l'Ambient Occlusion. Cette amélioration a néanmoins un coût en utilisation mémoire et en temps de pré-calcul directement proportionnel au nombre de structures que l'on définit dans les données.

Performance

Concernant l'aspect performance, nous nous sommes intéressés à la réduction du temps de calcul de la méthode d'Ambient Occlusion. Pour cela, nous nous sommes basés sur une approche de bricking, consistant à regrouper des ensembles de voxels en briques. Chacune de ces briques mémorise une information booléenne indiquant si l'ensemble des voxels qu'elle contient sont transparents ou non pour une fonction de transfert donnée. Si l'ensemble des voxels est transparent, on peut alors ignorer cette brique dans la phase de calcul des facteurs d'Ambient Occlusion, et ainsi réduire les temps de pré-calcul.

Afin de réduire davantage les temps de calcul des facteurs d'Ambient Occlusion, nous proposons aussi de paralléliser le calcul en utilisant une architecture multi-GPU. Pour cela, nous proposons d'abord un découpage statique du jeu de données, qui alloue à chaque GPU disponible un bloc de données à calculer. Chaque GPU calcule alors des facteurs d'occultation pour une sous-partie des données initiales réduisant le temps global du processus. L'utilisation du bricking dans ce contexte peut créer des problèmes d'équilibrage de charge entre les différents GPUs. Nous proposons un critère basé sur le nombre de briques pleines dans chaque sous-bloc, afin d'équilibrer les temps de calcul entre les différents GPUs. Dans un second temps, nous proposons aussi une technique de parallélisation dynamique basée sur un modèle producteur-consommateur. On découpe ici les données en sous-blocs de taille pré-définie, puis on les distribue successivement aux GPUs disponibles. Cette seconde approche permet de mieux prendre en compte les jeux de données de taille importante. En effet en découpant en petit blocs de tels jeux de données, on peut calculer des facteurs d'occultation indépendamment de la capacité mémoire des cartes graphiques utilisées. Ce qui pouvait ne pas être le cas avec un seul GPUs ou une distribution statique avec autant de sous-blocs que de GPUs.

Ces deux modèles, ainsi que le bricking, permettent de réduire considérablement les temps de pré-calcul et de passer sous la barre de la seconde pour le pré-calcul des facteurs d'Ambient Occlusion d'un jeu de données de taille 256^3, alors qu'il fallait initialement plusieurs secondes. De plus, cette technique est aussi applicable à la méthode d'Ambient Occlusion par structure, permettant de réduire l'impact de la séparation en structures sur le temps de pré-calcul.

Perspectives

Concernant les perspectives de travail, nous aimerions tout d'abord intégrer la visualisation parallèle finale dans le contexte de l'Ambient Occlusion multi-GPU. Chaque GPU calcule ainsi une sous-partie du jeu de données, d'où une économie en espace, et réalise le rendu du bloc de données correspondant en utilisant les facteurs d'occultation qu'il aura calculé lui-même. Il suffit finalement de transférer ces parties d'image vers le GPU en charge et de recomposer les différentes sous-parties pour former l'image finale. Le fait de paralléliser la phase finale de rendu permet également de déterminer la visibilité effective de chaque sous-bloc et ainsi d'éviter de calculer des facteurs d'occultation pour des parties non visibles au final.

Dans un second temps et dans le cadre du rendu de jeux de données de taille importante, nous souhaitons expérimenter cette technique sur un cluster de PC multi-GPU, avec les contraintes additionnelles que cela implique notamment au niveau des temps de transfert. Si on combine le calcul de l'Ambient Occlusion parallélisé sur un plus grand nombre de GPUs et le rendu volumique parallèle, on peut viser l'édition interactive de la fonction de transfert et le recalcul en temps interactif des nouveaux facteurs d'occultation.

Nous souhaitons aussi explorer de nouvelles méthodes de calcul de l'Ambient Occlusion pour améliorer encore la perception des données. En effet, le fait de séparer les structures permet d'éviter le sur-assombrissement, mais il élimine par là même certains effets d'ombrage qui pourraient rester désirables. Une approche à mi-chemin entre l'Ambient Occlusion par structure et l'Ambient Occlusion classique pourrait se baser sur la méthode proposée par Desgranges et Engel [DE07] dont l'idée consiste à prendre en compte plusieurs rayons de sphère d'influence dans le calcul de l'Ambient Occlusion. Cependant au lieu de combiner de manière globale les différents volumes d'occultation en les pondérant, nous pourrions les combiner localement. Ceci pourrait être par exemple fait en évaluant quels rayons permettent d'apporter la meilleure perception aux différentes parties du jeu de données. Dans le cadre de l'Ambient Occlusion par structure, nous pourrions aussi utiliser le même principe avec différentes tailles de rayon ou sélectionner une taille spécifique par structure. On peut ainsi utiliser des rayons moins grands pour les structures opaques, et ainsi réduire l'assombrissement pour ces structures ; ou des rayons plus grands pour les structures transparentes. D'autres pistes d'évolution des méthodes d'Ambient Occlusion pourraient être d'utiliser des propriétés relatives à l'objet comme la courbure ou l'utilisation d'histogrammes, mais ces méthodes requièrent l'analyse du voisinage. Il faudrait alors évaluer le compromis entre qualité et performance.

A plus long terme, nous souhaitons combiner différentes approches d'éclairage. L'Ambient Occlusion permet d'avoir une bonne perception du relief des surfaces et de profiter d'ombres indiquant la proximité entre structures. L'éclairage de Phong permet quant à lui d'obtenir des reflets spéculaires, qui améliorent notamment le réalisme de la visualisation. Une bonne solution pourrait être de combiner ces modes d'éclairage pour profiter des avantages de chacune. La bonne combinaison et paramétrisation de chacune de ces méthodes constitue alors le cœur du problème, car on souhaite rendre la visualisation volumique plus facile pour l'utilisateur, tout en lui offrant des indices visuels lui permettant d'analyser les données.

Bibliographie

[ADM10] Alexandre Ancel, Jean-Michel Dischler, and Catherine Mongenet, *Feature-driven ambient occlusion for direct volume rendering*, International Symposium on Volume Graphics, IEEE/EG, may 2010.

[ADP10] James P. Ahrens, Kurt Debattista, and Renato Pajarola (eds.), *Eurographics symposium on parallel graphics and visualization, egpgv 2010, norrköping, sweden*, Eurographics Association, 2010.

[Bli77] James F. Blinn, *Models of light reflection for computer synthesized pictures*, SIGGRAPH '77, 1977, pp. 192–198.

[Bli82] ———, *Light reflection functions for simulation of clouds and dusty surfaces*, SIGGRAPH Comput. Graph. **16** (1982), no. 3, 21–29.

[BR98] Uwe Behrens and Ralf Ratering, *Adding shadows to a texture-based volume renderer*, VVS '98 : Proceedings of the 1998 IEEE symposium on Volume visualization (New York, NY, USA), ACM, 1998, pp. 39–46.

[Bre65] Jack Bresenham, *Algorithm for computer control of a digital plotter*, IBM Systems Journal **4** (1965), no. 1, 25–30.

[Bre02] Rob Bredow, *Renderman on film*, Course 16 : RenderMan in Production. ACM SIGGRAPH Course Notes, July 2002.

[BSP06] James Bigler, Abe Stephens, and Steven G. Parker, *Design for parallel interactive ray tracing systems*, in : Proceedings of IEEE Symposium on Interactive Ray Tracing, 2006, pp. 187–196.

[CCF94] Brian Cabral, Nancy Cam, and Jim Foran, *Accelerated volume rendering and tomographic reconstruction using texture mapping hardware*, VVS '94 : Proceedings of the 1994 symposium on Volume visualization (New York, NY, USA), ACM, 1994, pp. 91–98.

[CHH02] Nathan A. Carr, Jesse D. Hall, and John C. Hart, *The ray engine*, HWWS '02 : Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware (Aire-la-Ville, Switzerland, Switzerland), Eurographics Association, 2002, pp. 37–46.

[CM09a] Carlos D. Correa and Kwan-Liu Ma, *The occlusion spectrum for volume classification and visualization*, IEEE Transactions for Visualization and Computer Graphics **15** (2009), no. 6.

[CM09b] Carlos D. Correa and Kwan-Liu Ma, *Visibility-driven transfer functions*, PACIFICVIS '09 : Proceedings of the 2009 IEEE Pacific Visualization Symposium (Washington, DC, USA), IEEE Computer Society, 2009, pp. 177–184.

[CN94] Timothy J. Cullip and Ulrich Neumann, *Accelerating volume reconstruction with 3d texture hardware*, Tech. report, Chapel Hill, NC, USA, 1994.

[CNLE09] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann, *Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering*, feb 2009, to appear.

[Cro84] Franklin C. Crow, *Summed-area tables for texture mapping*, SIGGRAPH Comput. Graph. **18** (1984), no. 3, 207–212.

[CWM+09a] Ming-Yuen Chan, Yingcai Wu, Wai-Ho Mak, Wei Chen, and Huamin Qu, *Perception-based transparency optimization for direct volume rendering*, IEEE Transactions on Visualization and Computer Graphics **15** (2009), no. 6, 1283–1290.

[CWM09b] Johnson Chuang, Daniel Weiskopf, and Torsten Möller, *Hue-preserving color blending*, IEEE Transactions on Visualization and Computer Graphics **15** (2009), no. 6, 1275–1282.

[DE07] Philippe Desgranges and Klaus Engel, *Fast ambient occlusion for direct volume rendering*, United States Patent Application, no. #20070013696, 2007.

[DV10] José Díaz and Pere-Pau Vázquez, *Depth-enhanced maximum intensity projection*, Volume Graphics (Rüdiger Westermann and Gordon L. Kindlmann, eds.), Eurographics Association, 2010, pp. 93–100.

[DVND10] José Díaz, Pere-Pau Vázquez, Isabel Navazo, and Florent Duguet, *Technical section : Real-time ambient occlusion and halos with summed area tables*, Comput. Graph. **34** (2010), 337–350.

[DYV08] José Díaz, Héctor Yela, and Pere Pau Vazquez, *Vicinity occlusion maps : Enhanced depth perception of volumetric models*, Computer Graphics International (Istanbul, Turkey), 2008, pp. 56–63.

[EHK+03] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Aaron Lefohn, and Daniel Weiskopf, *Interactive visualization of volumetric data on consumer pc hardware*, `http://www.vis.uni-stuttgart.de/vis03_tutorial/`, 2003.

[EKE01] Klaus Engel, Martin Kraus, and Thomas Ertl, *High-quality pre-integrated volume rendering using hardware-accelerated pixel shading*, HWWS '01 : Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware (New York, NY, USA), ACM, 2001, pp. 9–16.

[ER00] David Ebert and Penny Rheingans, *Volume illustration : non-photorealistic rendering of volume models*, Proceedings of the conference on Visualization '00, 2000, pp. 195–202.

[FCS+10] Thomas Fogal, Hank Childs, Siddharth Shankar, Jens Krüger, R. Daniel Bergeron, and Philip Hatcher, *Large data visualization on distributed memory multi-gpu clusters*, Proceedings of the Conference on High Performance Graphics (Aire-la-Ville, Switzerland, Switzerland), HPG '10, Eurographics Association, 2010, pp. 57–66.

[FK10] Jens Fangerau and Susanne Krömker, *Facing the multicore-challenge*, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 143–153.

[GAMD10] Amel Guetat, Alexandre Ancel, Stéphane Marchesin, and Jean-Michel Dischler, *Pre-integrated volume rendering with non-linear gradient interpolation*, IEEE Trans. Vis. Comput. Graph. **16** (2010), no. 6, 1487–1494.

[GGSC98] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen, *A non-photorealistic lighting model for automatic technical illustration*, Proceedings of the 25th annual conference on Computer graphics and interactive techniques (New York, NY, USA), SIGGRAPH '98, ACM, 1998, pp. 447–452.

[GMI08] Enrico Gobbetti, Fabio Marton, and José Antonio Iglesias Guitián, *A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets*, The Visual Computer **24** (2008), no. 7-9, 797–806, Proc. CGI 2008.

[GSG+99] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld, *Interactive technical illustration*, Proceedings of the 1999 symposium on Interactive 3D graphics (New York, NY, USA), I3D '99, ACM, 1999, pp. 31–38.

[GWGS02] Stefan Guthe, Michael Wand, Julius Gonser, and Wolfgang Straßer, *Interactive rendering of large volume data sets*, Proceedings of the conference on Visualization '02 (Washington, DC, USA), VIS '02, IEEE Computer Society, 2002, pp. 53–60.

[Har08] Mark Harris, *Optimizing parallel reduction in cuda*, White paper, NVIDIA, 2008.

[HHKP96] Taosong He, Lichan Hong, Arie Kaufman, and Hanspeter Pfister, *Generation of transfer functions with stochastic search techniques*, Proceedings of the 7th conference on Visualization '96 (Los Alamitos, CA, USA), VIS '96, IEEE Computer Society Press, 1996, pp. 227–ff.

[HKSB06] Markus Hadwiger, Andrea Kratz, Christian Sigg, and Katja Bühler, *Gpu-accelerated deep shadow maps for direct volume rendering*, GH '06 : Proceedings of the 21st symposium on Graphics hardware, ACM, 2006, pp. 49–52.

[HLY07] Frida Hernell, Patric Ljung, and Anders Ynnerman, *Efficient ambient and emissive tissue illumination using local occlusion in multiresolution volume rendering*, Eurographics/IEEE-VGTC Symposium on Volume Graphics, 2007, pp. 1–8.

[HMBG00] Helwig Hauser, Lukas Mroz, Gian-Italo Bischi, and M. Eduard Gröller, *Two-level volume rendering — fusing mip and dvr*, Proceedings of the conference on Visualization '00, 2000, pp. 211–218.

[HMDM08] Jean-François El Hajjar, Stéphane Marchesin, Jean-Michel Dischler, and Catherine Mongenet, *Second order pre-integrated volume rendering*, IEEE Pacific Visualization Symposium (Issei Fujishiro, Hua Li, and Kwan-Liu Ma, eds.), march 2008, pp. 9–16.

[HTHG01] Markus Hadwiger, Thomas Theußl, Helwig Hauser, and Eduard Gröller, *Hardware-accelerated high-quality filtering on pc hardware*, Proceedings of Workshop on Vision, Modeling and Visualization (2001), 105–112.

[IKSZ03] Andrey Iones, Anton Krupkin, Mateu Sbert, and Sergey Zhukov, *Fast, realistic lighting for video games*, IEEE Comput. Graph. Appl. **23** (2003), 54–64.

[JH04] Christopher Johnson and Charles Hansen, *Visualization handbook*, Academic Press, Inc., Orlando, FL, USA, 2004.

[KD98] Gordon Kindlmann and James W. Durkin, *Semi-automatic generation of transfer functions for direct volume rendering*, Volume Visualization, 1998. IEEE Symposium on (19-20 Oct 1998), 79–86, 170.

[KH84] James T. Kajiya and Brian P. Von Herzen, *Ray tracing volume densities*, SIGGRAPH Comput. Graph. **18** (1984), no. 3, 165–174.

[KKH02] Joe Kniss, Gordon Kindlmann, and Charles Hansen, *Multidimensional transfer functions for interactive volume rendering*, IEEE Transactions on Visualization and Computer Graphics **8** (2002), no. 3, 270–285.

[KPH+03] Joe Kniss, Simon Premoze, Charles Hansen, Peter Shirley, and Allen McPherson, *A model for volume lighting and modeling*, IEEE Transactions on Visualization and Computer Graphics **09** (2003), no. 2, 150–162.

[KPHE02] Joe Kniss, Simon Premoze, Charles Hansen, and David Ebert, *Interactive translucent volume rendering and procedural modeling*, Proceedings of IEEE Visualization, 2002, pp. 109–116.

[Krü10] Jens Krüger, *A new sampling scheme for slice based volume rendering*, IEEE/EG International Symposium on Volume Graphics, 2010.

[KSSE05] Thomas Klein, Magnus Strengert, Simon Stegmaier, and Thomas Ertl, *Exploiting frame-to-frame coherence for accelerating high-quality volume raycasting on graphics hardware*, IEEE Visualization, IEEE Computer Society, 2005, p. 29.

[KW03] Jens Krüger and Rüdiger Westermann, *Acceleration techniques for gpu-based volume rendering*, VIS '03 : Proceedings of the 14th IEEE Visualization 2003 (VIS'03) (Washington, DC, USA), IEEE Computer Society, October 2003, p. 38.

[KWTM03] Gordon Kindlmann, Ross Whitaker, Tolga Tasdizen, and Torsten Moller, *Curvature-based transfer functions for direct volume rendering : Methods and applications*, Proceedings of the 14th IEEE Visualization 2003, 2003, p. 67.

[Lan02] Hayden Landis, *Production-ready global illumination*, Course 16 : RenderMan in Production. ACM SIGGRAPH Course Notes, July 2002.

[LB99] Michael S. Langer and Heinrich H. Bülthoff, *Perception of shape from shading on a cloudy day*, Tech. Report 73, Tübingen, Germany, October 1999.

[LB00] ———, *Depth discrimination from shading under diffuse lighting.*, Perception **29** (2000), 649–660.

[LC87] William E. Lorensen and Harvey E. Cline, *Marching cubes : A high resolution 3d surface construction algorithm*, SIGGRAPH Comput. Graph. **21** (1987), no. 4, 163–169.

[LCD09] Baoquan Liu, Gordon Clapworthy, and Feng Dong, *Accelerating volume raycasting using proxy spheres*, Comput. Graph. Forum **28** (2009), no. 3, 839–846.

[LHJ99] Eric LaMar, Bernd Hamann, and Kenneth I. Joy, *Multiresolution techniques for interactive texture-based volume visualization*, IEEE Visualization, 1999, pp. 355–361.

[LL94] Philippe Lacroute and Marc Levoy, *Fast volume rendering using a shear-warp factorization of the viewing transformation*, SIGGRAPH '94 : Proceedings of the 21st annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM, 1994, pp. 451–458.

[LM02] Eric B. Lum and Kwan-Liu Ma, *Hardware-accelerated parallel non-photorealistic volume rendering*, Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering (New York, NY, USA), NPAR '02, ACM, 2002, pp. 67–ff.

[LSH05] Won-Jong Lee, Vason P. Srini, and Tack-Don Han, *Adaptive and scalable load balancing scheme for sort-last parallel volume rendering on gpu clusters*, International Workshop on Volume Graphics, 2005.

[LWM04] Eric B. Lum, Brett Wilson, and Kwan-Liu Ma, *High-quality lighting and efficient pre-integration for volume rendering*, VisSym04 - Eurographics / IEEE TCVG Symposium on Visualization, May 2004, pp. 25–34.

[Mar07] Stéphane Marchesin, *Contributions à la visualisation volumique hautes performances*, Ph.D. thesis, Université Louis Pasteur-Strasbourg I, July 2007.

[Max95] Nelson Max, *Optical models for direct volume rendering*, IEEE Transactions on Visualization and Computer Graphics **1** (1995), no. 2, 99–108.

[MCEF94] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs, *A sorting classification of parallel rendering*, Computer Graphics and Applications, IEEE **14** (1994), no. 4, 23 –32.

[MDM07] Stephane Marchesin, Jean-Michel Dischler, and Catherine Mongenet, *Feature Enhancement using Locally Adaptive Volume Rendering* , Eurographics Association, 2007, pp. 41–48.

[MFS09] Àlex Méndez-Feliu and Mateu Sbert, *From obscurances to ambient occlusion : A survey*, The Visual Computer **25** (2009), no. 2, 181–196.

[MHE01] Marcello Magallón, Matthias Hopf, and Thomas Ertl, *Parallel volume rendering using pc graphics hardware*, Computer Graphics and Applications, 2001. Proceedings. Ninth Pacific Conference on, 2001, pp. 384 –389.

[Mit07] Martin Mittring, *Finding next gen : Cryengine 2*, ACM SIGGRAPH 2007 courses (New York, NY, USA), SIGGRAPH '07, ACM, 2007, pp. 97–121.

[MM10] Stéphane Marchesin and Kwan-Liu Ma, *Cross-node occlusion in sort-last volume rendering*, in Ahrens et al. [ADP10], pp. 11–18.

[MMD06] Stéphane Marchesin, Catherine Mongenet, and Jean-Michel Dischler (eds.), *Dynamic load balancing for parallel volume rendering*, EGPGV'06 : symposium on parallel graphics and visualization, Braga, Portugal, Eurographics, may 2006.

[MMD08] ———, *Multi-gpu sort last volume visualization*, EG Symposium on Parallel Graphics and Visualization (EGPGV'08), Eurographics, april 2008.

[MR10] Christian Meß and Timo Ropinski, *Efficient acquisition and clustering of local histograms for representing voxel neighborhoods*, IEEE/EG International Symposium on Volume Graphics (Rüdiger Westermann and Gordon Kindlmann, eds.), Eurographics Association, 2010, pp. 117–124.

[MRH08] Jörg Mensmann, Timo Ropinski, and Klaus Hinrichs, *Accelerating volume raycasting using occlusion frustums*, 2008, pp. 147–154.

[MSM10] Steven Martin, Han-Wei Shen, and Patrick McCormick, *Load-balanced isosurfacing on multi-gpu clusters*, EGPGV '10 : Proceedings of Eurographics Symposium on Parallel Graphics and Visualization 2010, May 2010, pp. 91–100.

[NHM97] Gregory M. Nielson, Hans Hagen, and Heinrich Müller (eds.), *Scientific visualization, overviews, methodologies, and techniques*, Washington, DC, USA, IEEE Computer Society, 1997.

[NVIa] NVIDIA, *Sli*.

[NVIb] ———, *Texture compositing with register combiners*.

[NVI04] ———, *Nvidia sdk 8.0*, 2004.

[NVI11] NVIDIA, *NVIDIA CUDA Programming Guide 4.0*, 2011.

[PBMH02] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan, *Ray tracing on programmable graphics hardware*, ACM Transactions on Graphics **21** (2002), no. 3, 703–712, ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[PD84] Thomas Porter and Tom Duff, *Compositing digital images*, SIGGRAPH Comput. Graph. **18** (1984), no. 3, 253–259.

[Pho75] Bui Tuong Phong, *Illumination for computer generated pictures*, Commun. ACM **18** (1975), no. 6, 311–317.

[PYRM08] Tom Peterka, Hongfeng Yu, Robert Ross, and Kwan-Liu Ma, *Parallel volume rendering on the ibm blue gene/p*, Proceedings of Eurographics Parallel Graphics and Visualization Symposium 2008 (Crete, Greece), 2008.

[RBD06] Szymon Rusinkiewicz, Michael Burns, and Doug DeCarlo, *Exaggerated shading for depicting shape and detail*, ACM SIGGRAPH 2006 Papers (New York, NY, USA), SIGGRAPH '06, ACM, 2006, pp. 1199–1205.

[RBV+08] Marc Ruiz, Imma Boada, Ivan Viola, Stefan Bruckner, Miquel Feixas, and Mateu Sbert, *Obscurance-based volume rendering framework*, Volume and Point-Based Graphics 2008, aug 2008, pp. 113–120.

[RGW+03] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser, *Smart hardware-accelerated volume rendering*, VISSYM '03 : Proceedings of the symposium on Data visualisation 2003 (Aire-la-Ville, Switzerland, Switzerland), Eurographics Association, May 2003, pp. 231–238.

[Rit07] Tobias Ritschel, *Fast GPU-based Visibility Computation for Natural Illumination of Volume Data Sets*, Short Paper Proceedings of Eurographics, September 2007, pp. 17–20.

[RKE00] Stefan Röttger, Martin Kraus, and Thomas Ertl, *Hardware-accelerated volume and isosurface rendering based on cell-projection*, VIS '00 :

Proceedings of the conference on Visualization '00 (Los Alamitos, CA, USA), IEEE Computer Society Press, 2000, pp. 109–116.

[RKH08] Timo Ropinski, Jens Kasten, and Klaus H. Hinrichs, *Efficient shadows for gpu-based volume raycasting*, Proceedings of the 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2008), 2008, pp. 17–24.

[RMSD+08] Timo Ropinski, Jennis Meyer-Spradow, Stefan Diepenbrock, Jörg Mensmann, and Klaus H. Hinrichs, *Interactive volume rendering with dynamic ambient occlusion and color bleeding*, Computer Graphics Forum **27** (2008), no. 2, 567–576.

[RSEB+00] Christof Rezk-Salama, Klaus Engel, Michael Bauer, Gunther Greiner, and Thomas Ertl, *Interactive volume on standard pc graphics hardware using multi-textures and multi-stage rasterization*, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware (New York, NY, USA), HWWS '00, ACM, 2000, pp. 109–118.

[Sab88] Paolo Sabella, *A rendering algorithm for visualizing 3d scalar fields*, SIGGRAPH '88 : Proceedings of the 15th annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM, 1988, pp. 51–58.

[SH05] Christian Sigg and Markus Hadwiger, *Fast third-order texture filtering*, GPU Gems 2 : Programming Techniques for High-Performance Graphics and General-Purpose Computation (2005), 313–329.

[SKLE03] Jürgen P. Schulze, Martin Kraus, Ulrich Lang, and Thomas Ertl, *Integrating pre-integration into the shear-warp algorithm*, VG '03 : Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics (New York, NY, USA), ACM, 2003, pp. 109–118.

[SMW+04] Magnus Strengert, Marcelo Magallon, Daniel Weißkopf, Stefan Guthe, and Thomas Ertl, *Hierarchical visualization and compression of large volume datasets using GPU clusters*, Proc. Eurographics Symposium on Parallel Graphics and Visualization (EGPGV) 2004, 2004, pp. 41–48.

[ŠPBV10] Veronika Šoltészová, Daniel Patel, Stefan Bruckner, and Ivan Viola, *A multidirectional occlusion shading model for direct volume rendering*, Computer Graphics Forum **29** (2010), no. 3, 883–891.

[SPH+09] Mathias Schott, Vincent Pegoraro, Charles D. Hansen, Kévin Boulanger, and Kadi Bouatouch, *A directional occlusion shading model for interactive direct volume rendering*, Computer Graphics Forum, vol. 28, 2009.

[SSKE05] Simon Stegmaier, Magnus Strengert, Thomas Klein, and Thomas Ertl, *A simple and flexible volume rendering framework for graphics-hardware-based raycasting*, Volume Graphics (Arie E. Kaufman, Klaus Mueller, Eduard Gröller, Dieter W. Fellner, Torsten Möller, and Stephen N. Spencer, eds.), Eurographics Association, 2005, pp. 187–195.

[ST90] Peter Shirley and Allan Tuchman, *A polygonal approximation to direct scalar volume rendering*, SIGGRAPH Comput. Graph. **24** (1990), no. 5, 63–70.

[Ste03] A. James Stewart, *Vicinity shading for enhanced perception of volumetric data*, Proceedings of the 14th IEEE Visualization Conference, 2003, pp. 355–362.

[TDR10] Marc Tchiboukdjian, Vincent Danjean, and Bruno Raffin, *Cache-efficient parallel isosurface extraction for shared cache multicores*, in Ahrens et al. [ADP10], pp. 81–90.

[VKG04] Ivan Viola, Armin Kanitsar, and Meister Eduard Gröller, *Importance-driven volume rendering*, Proceedings of IEEE Visualization'04, 2004, pp. 139–145.

[VKG05] ———, *Importance-driven feature enhancement in volume visualization*, 2005, pp. 408–418.

[Wes90] Lee Westover, *Footprint evaluation for volume rendering*, SIGGRAPH (Forest Baskett, ed.), ACM, 1990, pp. 367–376.

[WMLK89] Jerold W. Wallis, Tom R. Miller, Charles A. Lerner, and Eric C. Kleerup, *Three-dimensional display in nuclear medicine*, Medical Imaging, IEEE Transactions on **8** (1989), no. 4, 297 –230.

[ZIK98] Sergey Zhukov, Andrei Iones, and Grigorij Kronin, *An ambient light illumination model*, Rendering Techniques, 1998, pp. 45–56.

[ZPvBG02] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross, *Ewa splatting*, IEEE Transactions on Visualization and Computer Graphics **8** (2002), no. 3, 223–238.