Université de Montréal

Rendu de matériaux semi-transparents hétérogènes en temps réel

par
Éric Blanchard

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

juin 2012

Université de Montréal
Faculté des arts et des sciences

Ce mémoire de maîtrise intitulé

Rendu de matériaux semi-transparents hétérogènes en temps réel

présenté par
Éric Blanchard

a été évalué par un jury composé des personnes suivantes :

Président :	Jean Meunier
Directeur de recherche :	Pierre Poulin
Membre :	Derek Nowrouzezahrai

Sommaire

On retrouve dans la nature un nombre impressionnant de matériaux semi-transparents tels le marbre, le jade ou la peau, ainsi que plusieurs liquides comme le lait ou les jus. Que ce soit pour le domaine cinématographique ou le divertissement interactif, l'intérêt d'obtenir une image de synthèse de ce type de matériau demeure toujours très important. Bien que plusieurs méthodes arrivent à simuler la diffusion de la lumière de manière convaincante à l'intérieur de matériaux semi-transparents, peu d'entre elles y arrivent de manière interactive.

Ce mémoire présente une nouvelle méthode de diffusion de la lumière à l'intérieur d'objets semi-transparents hétérogènes en temps réel. Le coeur de la méthode repose sur une discrétisation du modèle géométrique sous forme de voxels, ceux-ci étant utilisés comme simplification du domaine de diffusion. Notre technique repose sur la résolution de l'équation de diffusion à l'aide de méthodes itératives permettant d'obtenir une simulation rapide et efficace. Notre méthode se démarque principalement par son exécution complètement dynamique ne nécessitant aucun pré-calcul et permettant une déformation complète de la géométrie.

Mots clefs : BSSRDF, équation de diffusion, voxélisation, rendu temps réel, matériaux semi-transparents hétérogènes.

Abstract

We find in nature several semi-transparent materials such as marble, jade or skin, as well as liquids such as milk or juices. Whether it be for digital movies or video games, having an efficient method to render these materials is an important goal. Although a large body of previous academic work exists in this area, few of these works provide an interactive solution.

This thesis presents a new method for simulating light scattering inside heterogeneous semi-transparent materials in real time. The core of our technique relies on a geometric mesh voxelization to simplify the diffusion domain. The diffusion process solves the diffusion equation in order to achieve a fast and efficient simulation. Our method differs mainly from previous approaches by its completely dynamic execution requiring no pre-computations and hence allowing complete deformations of the geometric mesh.

Keywords : BSSRDF, diffusion equation, voxelization, real-time rendering, translucent materials, subsurface scattering.

Table des matières

Table des figures

Liste des tableaux

Remerciements

J'aimerais en premier lieu, offrir un merci particulier à mon directeur de recherche, Pierre Poulin, pour son support (moral, technique, financier), ses conseils et sa toujours très grande disponibilité, même lorsque celui-ci se trouve à plusieurs milliers de kilomètres. Merci aussi à tous les membres du LIGUM, en particulier Gilles-Philippe Paillé, Dorian Gomez, Mohamed Yengui, Nicolas Rous et Marc-Antoine Desjardins qui m'ont permis de passer deux très belles années (et plus) au laboratoire.

J'ai aussi grandement apprécié mes discussions avec certains vétérans du labo, en particulier Luc Leblanc (au niveau technique) et Jocelyn Houle (au niveau philosophique). Merci Luc de m'avoir permis d'assister à la conclusion de plus de sept ans de recherche au doctorat !

Merci aussi au CRSNG et à l'organisation GRAND pour le soutien financier qu'ils m'ont apporté tout au long de mes études.

En terminant, je voudrais remercier les membres du comité Derek Nowrouzezahrai et Jean Meunier et saluer l'ensemble de la brigade française ayant fait un arrêt au LIGUM : Anthony Pajot, Arnaud Rosenblatt et Aurélien Yol.

Chapitre 1

Introduction

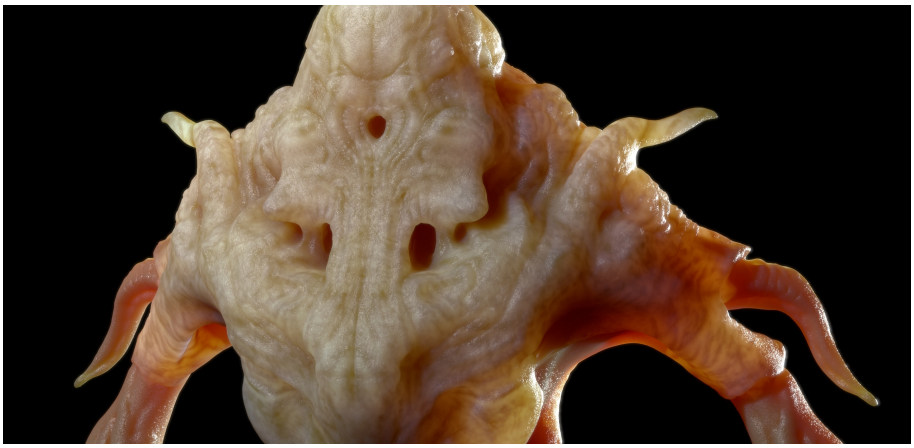

FIGURE 1.1 – Matériau semi-transparent rendu à partir de la méthode de D'Eon et Irving [DI11].

La simulation du transport de la lumière à l'intérieur de matériaux semi-transparents a reçu son lot d'intérêt lors de la dernière décennie. Plusieurs matériaux affichant ces propriétés tels la peau, les liquides ou le jade, se retrouvent fréquemment dans la composition de scènes pour le cinéma ou le divertissement interactif. Obtenir une simulation du transport de la lumière juste et précise de ce type de matériau est donc essentiel pour s'assurer du réalisme et de la qualité de l'image de synthèse. Les figures 1.1 et 1.2 montrent quelques exemples de rendus de matériaux semi-transparents réalisés à partir de méthodes non-interactives.

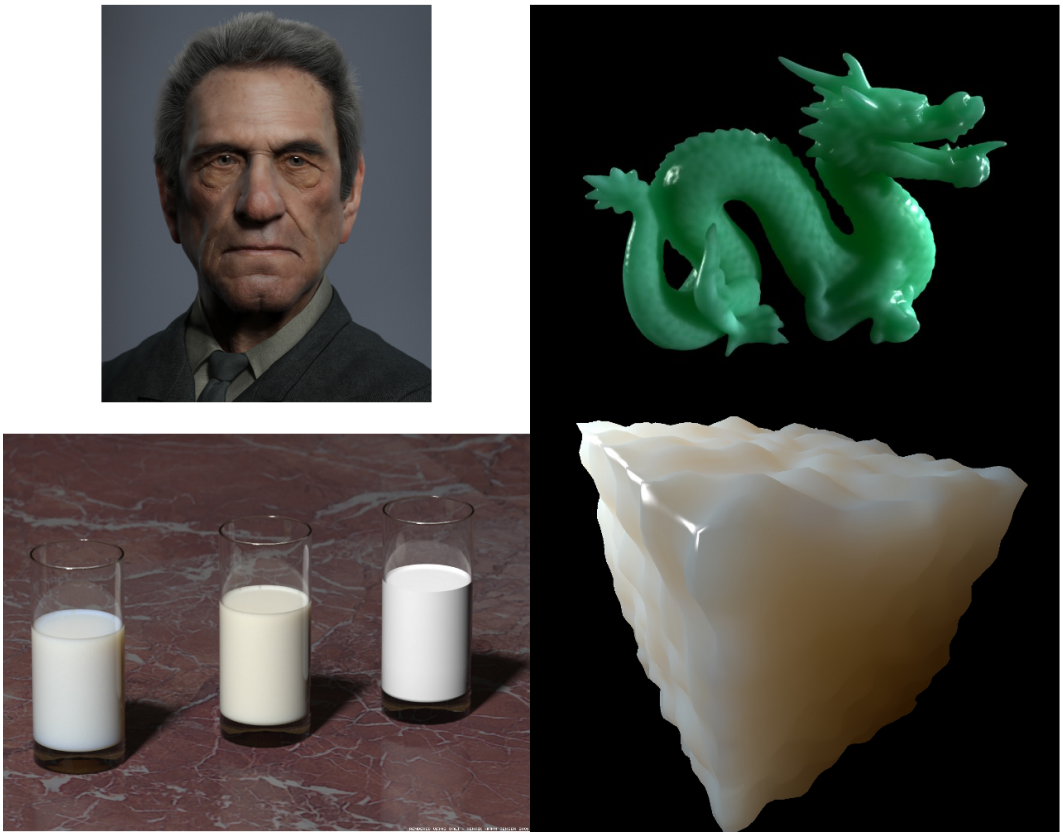

FIGURE 1.2 – Rendus de matériaux semi-transparents. Rendus du haut réalisés par le traceur de rayons *Arnold*, en bas à gauche par Jensen et al. [JMLH01], en bas à droite par D'Eon et Irving [DI11].

1.1 Motivation

Généralement, pour simplifier le calcul du transport de lumière, le milieu dans lequel sont placés les objets de la scène est considéré comme étant un vide total. Ces objets représentent donc les seule interactions pouvant influencer la distribution de la lumière simplifiant ainsi de beaucoup la simulation.

Lorsque que nous voulons simuler des milieux participatifs (e.g., le brouillard, la fumée, les particules dans l'air) ou des matériaux semi-transparents (e.g., la peau, le jade, le lait, la cire), cette simplification ne tient plus. La complexité additionnelle du calcul du transport de la lumière pour ce type de scène provient principalement du déplacement non-linéaire des photons à l'intérieur du matériau. Lorsque la lumière se déplace dans un milieu participatif, celle-ci peut être absorbée ou déviée de sa trajectoire

initiale par les particules composant ce milieu. Pour les objets semi-transparents, ces particules peuvent prendre la forme de bulles d'air ou d'impuretés présentes dans le matériau. Il devient alors rapidement très complexe de simuler exactement le transport de la lumière étant donnée l'explosion des chemins potentiels que celle-ci peut prendre. De plus, la composition même du milieu peut être différente en plusieurs points, que l'on pense aux différentes couches de la peau (épiderme, derme, hypoderme) ou à la densité des nuages, on parle alors de milieu hétérogène ajoutant encore à la complexité de la simulation.

Ces dernières années plusieurs approximations ont été proposées permettant de simuler en temps réel le transport de la lumière à l'intérieur de matériaux semi-transparents. L'intérêt d'obtenir une image en temps réel est évident pour les applications de divertissements interactifs, mais il peut l'être tout autant pour les outils de création 3D (*Autodesk 3ds Max*, *Autodesk Maya*, etc.). Un artiste pourrait par exemple, éditer en temps réel les propriétés de matériaux et avoir instantanément un aperçu à l'écran, contrairement aux rendus traditionnels pouvant prendre plusieurs minutes et même des heures sous certaines configurations.

1.2 Contributions

Le travail présenté dans ce mémoire s'intéresse principalement à la simulation en temps réel du transport de la lumière à l'intérieur de matériaux semi-transparents hétérogènes. Le but principal est d'obtenir une simulation complètement dynamique en exploitant les capacités des processeurs graphiques modernes. Contrairement aux méthodes antérieures, notre technique ne requiert aucun pré-calcul et permet non seulement de changer dynamiquement les propriétés des matériaux mais aussi la déformation des objets tout en gardant une simulation valide.

1.3 Organisation du mémoire

Nous ferons, en premier lieu, un survol théorique du rendu volumique et de ses composants. L'équation de transfert, la *Bidirectional Surface Scattering Reflectance Distribution Function* (BSSRDF) et l'équation de diffusion seront brièvement introduites et serviront de base à la description de notre technique. Nous reverrons aussi les plus récents

travaux ayant été réalisés dans ce domaine en accordant davantage d'intérêt aux simulations en temps réel. Pour sa part, le chapitre 4 décrira en détail notre technique pour le rendu de matériaux semi-transparents en se concentrant sur les étapes de voxélisation, d'initialisation du système, de diffusion et de rendu. Par la suite, nous présenterons une analyse approfondie des performances de notre technique et des résultats selon différents scénarios. Nous terminerons en mentionnant les limitations et les perspectives futures possibles de notre projet.

Chapitre 2

Équation de transfert et approximation de diffusion

Dans ce chapitre, nous ferons un bref survol théorique des éléments décrivant la diffusion de la lumière à l'intérieur de matériaux semi-transparents. Cette introduction servira de base à l'explication des travaux antérieurs ainsi qu'à la description de notre technique.

D'un point de vue général, en synthèse d'images, lorsque nous étudions l'interaction de la lumière avec un matériau, il peut être utile de séparer l'ensemble des matériaux en deux grandes catégories : les métaux et les matériaux isolants. Les métaux ont la particularité d'absorber entièrement la lumière transmise (l'énergie étant généralement convertie en chaleur), l'apparence de ceux-ci est donc déterminée uniquement par la lumière réfléchie à leur surface. La quantité de lumière transmise ou réfléchie peut dépendre tout autant du matériau, de la longueur d'onde de la lumière ainsi que de l'angle d'incidence de celle-ci. La figure 2.1 montre la réflectance de Fresnel du cuivre. La couleur orangée de ce dernier est causée par l'absorption des longueurs d'onde de basses fréquences (le bleu), les plus hautes fréquences réfléchies à la surface donnant la couleur orangée typique du cuivre. On remarque aussi, qu'à angle rasant (> 70 degrés par rapport à la normale), la quasi-totalité de la lumière est réfléchie plutôt qu'absorbée, donnant une réflexion (*highlight*) de couleur blanche au cuivre.

Pour leur part, l'apparence des matériaux isolants est déterminée principalement par le transport de la lumière à l'intérieur de ceux-ci. Lorsque la lumière se déplace à l'intérieur de l'objet, les photons peuvent être absorbés ou dispersés dépendemment

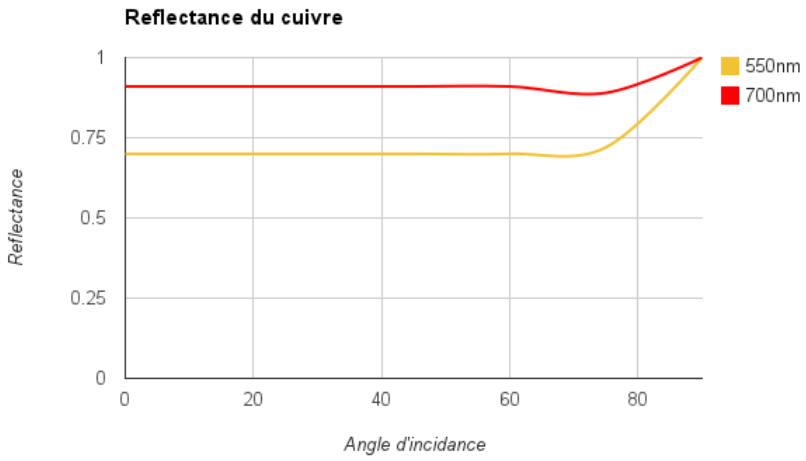

FIGURE 2.1 – Réflectance du cuivre.

de la structure atomique du matériau et de sa composition. La couleur perçue est donc dépendante de la lumière sortant de l'objet après l'absorption et la dispersion à l'intérieur de celui-ci.

Les matériaux semi-transparents faisant partie de la catégorie des isolants, nous nous attarderons donc davantage à leur comportement.

2.1 BRDF de matériaux isolants

La *Bidirectional Reflectance Distribution Function* (BRDF) est utilisée pour décrire la quantité de lumière réfléchie sur un élément de surface d'un matériau :

$$f(\vec{\omega_i}, \vec{\omega_o}) = \frac{dL_o(\vec{\omega_o})}{dE(\vec{\omega_i})}.$$

La BRDF décrit l'apparence du matériau par le rapport entre la radiance en sortie $dL_o(\vec{\omega_o})$ et l'irradiance en entrée $dE(\vec{\omega_i})$. Les angles solides $\vec{\omega_i}$ et $\vec{\omega_o}$ étant défini comme les directions d'entrée et de sortie respectivement. La BRDF est utilisée par l'équation du rendu pour obtenir la couleur finale du matériau (la radiance $L_o(\vec{\omega_o})$) :

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + \int_{2\pi} f(\vec{\omega_i}, \vec{\omega_o}) L_i(\mathbf{x}, \vec{\omega_i})(\vec{\omega_i} \cdot \vec{\omega_o}) d\vec{\omega_i}.$$

Dans ce travail, la composante émissive des matériaux $L_e(\mathbf{x}, \vec{\omega_o})$ sera omise, celle-ci étant rarement présente dans les matériaux semi-transparents (à l'exception de certains matériaux organiques tels les algues et les coraux).

2.1.1 BRDF lambertienne

La BRDF la plus simple pour représenter des matériaux isolants peut être dérivée à partir de la loi des cosinus de Lambert. La loi de Lambert stipule que l'intensité de la lumière observée sur une surface lambertienne est directement proportionnelle au cosinus de l'angle entre la normale à la surface et l'angle d'incidence de la lumière. Le facteur cosinus étant déja présent dans l'équation du rendu, la BRDF lambertienne peut donc se résumer en une valeur constante représentant l'énergie absorbée et dispersée par le matériau au point $\mathbf{x}$ (la valeur c_{diff} étant la couleur diffuse du matériau) :

$$f(\mathbf{x}, \vec{\omega_i}, \vec{\omega_o}) = \frac{c_{\text{diff}}}{\pi}.$$

La figure 2.2 montre le parcours de la lumière entrant en un point $\mathbf{x}$, se dispersant uniformément à l'intérieur du matériau et ressortant à son point d'origine également dans toutes les directions. Chaque vecteur sortant de la surface à la figure 2.2 est de longueur c_{diff}, représentant l'énergie absorbée par le matériau et dispersée à sa sortie. Cette valeur étant dépendante de la longueur d'onde de la lumière, on la retrouve généralement encodée sous forme de vecteur RGB.

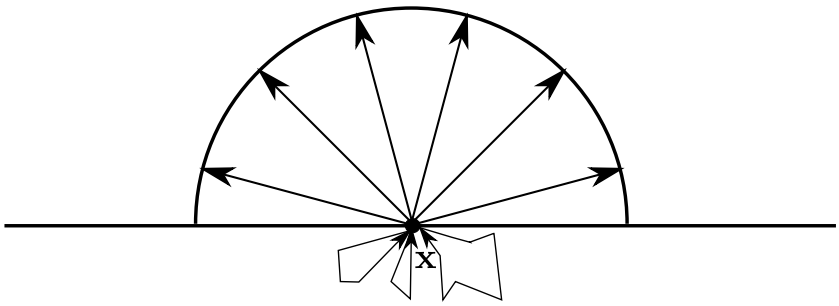

FIGURE 2.2 – BRDF lambertienne.

Pour simplifier son application, la constante c_{diff} peut être encodée à l'intérieur d'une texture appliquée sur la géométrie d'un objet pour simuler la couleur de celui-ci (comme le montre la figure 2.3). L'on peut donc considérer la valeur c_{diff} comme étant un pré-calcul de la quantité de lumière absorbée et dispersée à l'intérieur d'un objet à la position $\mathbf{x}$.

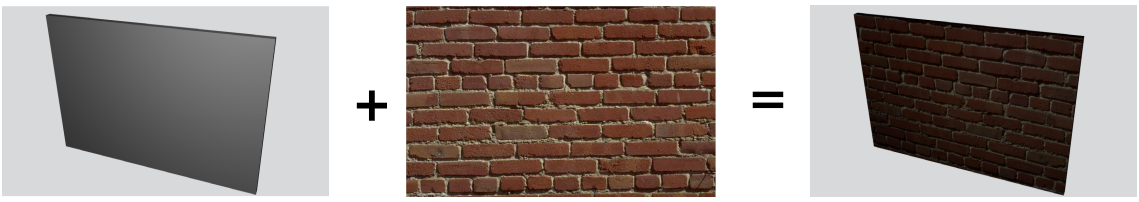

FIGURE 2.3 – Texture contenant la constante de la BRDF lambertienne.

2.2 La BSSRDF

Comme nous l'avons vu précédemment, le calcul de la BRDF suppose que la radiance est intégrée en entrée et en sortie aux alentours d'un même voisinage différentiel $d\mathbf{x}$ de la surface. Par contre, pour les matériaux semi-transparents, cette supposition ne tient plus étant donnée la plus grande distance parcourue par la lumière à l'intérieur du volume de l'objet (Voir la figure 2.4).

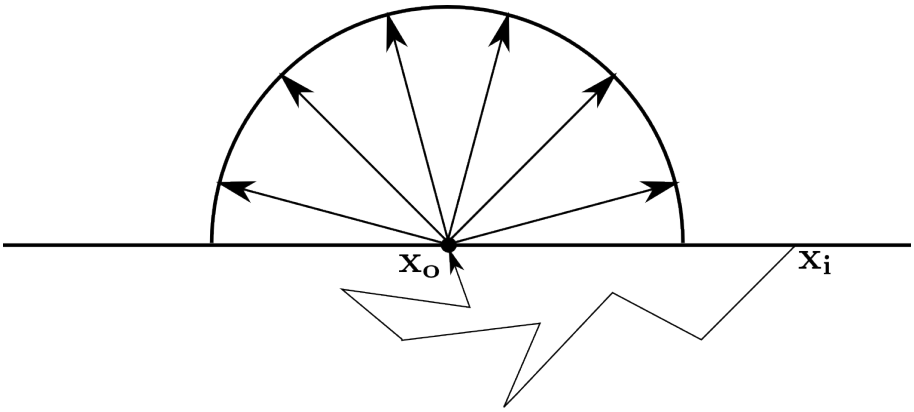

FIGURE 2.4 – BSSRDF.

La *Bidirectional Surface Scattering Distribution Function* (BSSRDF) permet de généraliser le concept de la BRDF en prenant compte l'énergie entrant sur l'ensemble de la surface :

$$S(\mathbf{x}_o, \vec{\omega_o}, \mathbf{x}_i, \vec{\omega_i}) = \frac{dL_o(\mathbf{x}_o, \vec{\omega_o})}{dE(\mathbf{x}_i, \vec{\omega_i})}.$$

Pour obtenir la radiance en sortie à un point donné $\mathbf{x}_o$, nous devons donc intégrer sur l'ensemble de la surface la radiance en entrée et en modulant celle-ci par la BSSRDF :

$$L_o(\mathbf{x}_o, \vec{\omega_o}) = \int_A \int_{2\pi} S(\mathbf{x}_o, \vec{\omega_o}, \mathbf{x}_i, \vec{\omega_i}) L_i(\mathbf{x}, \vec{\omega_i})(\vec{\omega_i} \cdot \vec{\omega_o}) d\vec{\omega_i} dA.$$

Le travail présenté dans ce mémoire se penche donc exclusivement sur les techniques de résolution rapide de cette double intégrale en se basant sur un ensemble

d'approximations exécutées sur processeur graphique.

2.3 Équation de transfert volumique

Jusqu'à présent, nous avons vu comment la lumière interagit avec le matériau à la surface de l'objet. Cette section s'intéressera au transport de la lumière à l'intérieur de l'objet. Les concepts présentés ici serviront de base à notre méthode de diffusion volumique sur processeur graphique.

2.3.1 Absorption

Lorsque la lumière se déplace à l'intérieur d'un matériau, une partie de son énergie est absorbée par les particules composant le matériau. Une plus grande densité du matériau entraînera une plus grande absorption d'énergie mais sans changer la direction de la lumière comme le montre la figure 2.5.

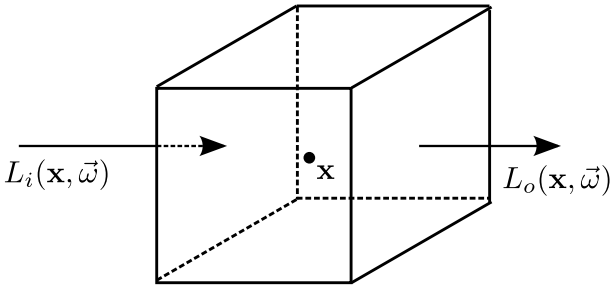

FIGURE 2.5 – Absorption à l'intérieur d'un élément différentiel du volume ($\mathbf{x}$ étant au centre du volume).

La quantité différentielle de radiance absorbée est définie linéairement à partir de la radiance en entrée :

$$dL_o(\mathbf{x}, \vec{\omega}) = -\sigma_a L_i(\mathbf{x}, \vec{\omega}) dt.$$

Le valeur σ_a est le coefficient d'absorption du matériau, définissant la quantité d'énergie absorbée par unité de distance (m^{-1}). Pour les matériaux hétérogènes, la valeur de ce coefficient peut varier à l'intérieur du volume. Pour obtenir l'absorption totale le long d'un chemin de distance d, nous devons résoudre cette équation différentielle et intégrer la solution sur la distance d :

$$\exp\left(-\int_0^d \sigma_a(\mathbf{x} + t\vec{\omega}) dt\right).$$

2.3.2 Dispersion en sortie

Le second phénomène contribuant à la réduction de la radiance est dû à la dispersion de la lumière. Cette dispersion est principalement causée par la collision de la lumière avec les particules du matériau, le changement de densité ou de structure du matériau et la présence de discontinuités telles des bulles d'air ou des particules étrangères (voir la figure 2.6).

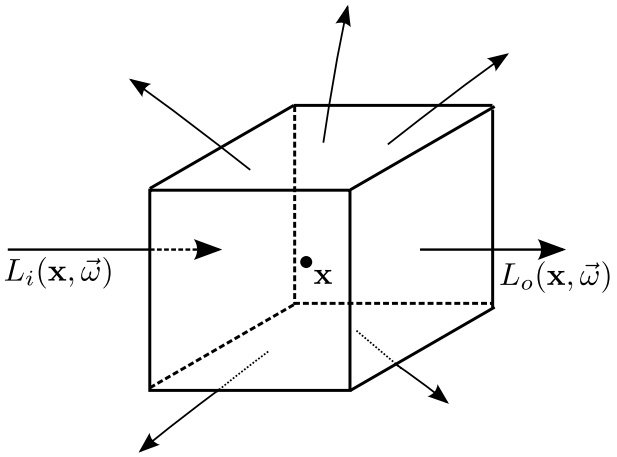

FIGURE 2.6 – Dispersion en sortie à l'intérieur d'un élément différentiel du volume.

La quantité de radiance perdue due à ce phénomène est décrite par le coefficient de dispersion σ_s, décrivant la probabilité d'un évènement de dispersion par unité de distance (m^{-1}). Nous pouvons décrire la quantité différentielle d'énergie perdue par une équation similaire au phénomène d'absorption :

$$dL_o(\mathbf{x}, \vec{\omega}) = -\sigma_s L_i(\mathbf{x}, \vec{\omega}) dt.$$

En combinant les coefficients d'absorption et de dispersion nous obtenons le coefficient d'extinction σ_t, décrivant l'énergie totale perdue par unité de distance :

$$\sigma_t = \sigma_a + \sigma_s.$$

Pour obtenir la perte de radiance totale sur un chemin de longueur d, il suffit donc de résoudre l'équation différentielle à l'aide du coefficient d'extinction σ_t donnant :

$$\exp\left(-\int_0^d \sigma_t(\mathbf{x} + t\vec{\omega}) dt\right).$$

Pour les matériaux homogènes, le calcul de la transmittance se simplifie comme suit : $e^{-\sigma_t d}$ et se nomme la loi de *Beer*.

2.3.3 Émission

Certains matériaux peuvent aussi augmenter l'apport en radiance causé par la conversion d'énergie en lumière. Cette augmentation de la radiance est représentée par la composante émissive du matériau et est définie comme suit :

$$dL_o(\mathbf{x}, \vec{\omega}) = L_e(\mathbf{x}, \vec{\omega}).$$

Généralement, la lumière émise est constante dans toutes les directions $\vec{\omega}$ mais peut varier selon sa position $\mathbf{x}$. Comme nous l'avons mentionné précédemment, cette partie émissive sera ignorée dans nos calculs, cette composante étant rarement présente pour les matériaux semi-transparents.

2.3.4 Dispersion en entrée

Le dernier phénomène présenté augmente l'énergie du volume différentiel en tenant compte de la radiance dispersée au voisinage de ce dernier (voir la figure 2.7).

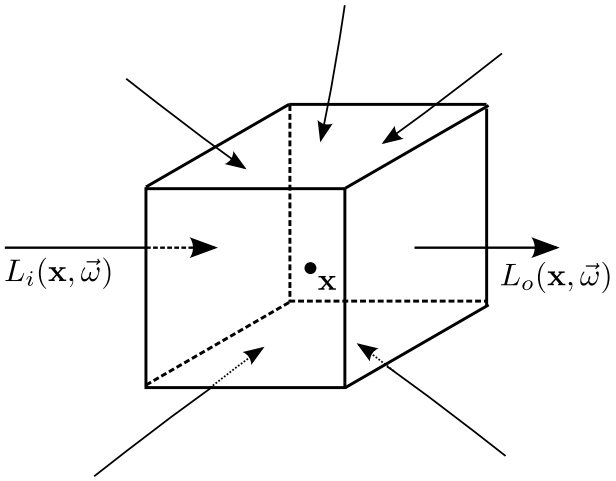

FIGURE 2.7 – Dispersion en entrée à l'intérieur d'un élément différentiel du volume.

Pour obtenir cet apport en énergie, nous devons intégrer l'énergie entrant par rapport à l'ensemble des directions :

$$dL_o(\mathbf{x}, \vec{\omega}) = \sigma_s \int_{4\pi} p(\mathbf{x}, \theta) L_i(\mathbf{x}, \vec{\omega}_i) d\vec{\omega}_i.$$

L'énergie en entrée $L_i(\mathbf{x}, \vec{\omega_i})$ est d'abord modulée par le coefficient de dispersion σ_s car seule la quantité dispersée doit être prise en compte. Étant donné que le coefficient

est constant à l'intérieur d'un élément de volume différentiel, celui-ci peut être sorti de l'intégrale.

Une fois la quantité d'énergie dispersée connue, nous devons calculer la proportion de lumière dispersée en direction $\vec{\omega}$. Cette proportion est définie à l'aide de la fonction de phase $p(\mathbf{x}, \theta)$.

Fonction de phase

La fonction de phase se définit comme étant la probabilité de déviation de la lumière vers une certaine direction $\vec{\omega}$ par rapport à sa direction initiale $\vec{\omega}_i$: $p(\vec{\omega}_i \rightarrow \vec{\omega})$. La fonction de phase peut s'apparenter au comportement de la BRDF en ce sens où la radiance en entrée est modulée par une fonction de distribution dépendante de ses directions d'entrée **et** de sortie. La fonction de phase par contre, représente une distribution de probabilité ; l'intégrale sur l'ensemble des directions doit donc donner un résultat unitaire :

$$\int_{4\pi} p(\vec{\omega}_i \rightarrow \vec{\omega}) d\vec{\omega} = 1.$$

Pour une fonction de phase isotropique, fonction ayant une probabilité de dispersion constante sur l'ensemble des directions, nous obtenons une fonction de phase à valeur constante :

$$\begin{aligned} \int_{4\pi} p(\vec{\omega}_i \rightarrow \vec{\omega}) d\vec{\omega} &= 1 \\ p \int_{4\pi} d\vec{\omega} &= 1 \\ p &= \frac{1}{4\pi}. \end{aligned}$$

Plutôt que d'utiliser la direction d'entrée $\vec{\omega}_i$ et la direction de dispersion $\vec{\omega}$, il est généralement plus simple de référer à la fonction de phase à partir de l'angle entre ces deux directions. La fonction de phase devient une fonction 1D $p(\theta)$. Cette simplification est un comportement isotropique de la fonction, soit que la probabilité est identique peu importe la direction initiale ω_i (à ne pas confondre avec la fonction de phase isotropique $p = \frac{1}{4\pi}$).

En infographie, le modèle d'Henyey-Greenstein est fréquemment utilisé pour décrire la fonction de phase. Un seul paramètre décrivant la symétrie de la distribution est

nécessaire permettant ainsi un encodage compact de la fonction :

$$p(\theta) = \frac{1 - g^2}{4\pi(1 + g^2 - 2g\cos(\theta)^{3/2})}.$$

Le paramètre g, se situant à l'intérieur de l'interval $[-1, 1]$, permet de spécifier la tendance de la fonction. Par exemple, une valeur négative décrit un matériau ayant une probabilité de diffusion plus élevée dans le sens inverse de sa diffusion initiale tandis qu'une valeur nulle décrit une fonction de phase isotropique. La figure 2.8 montre le tracé des paramètres $g = 0$ et $g = -0.5$.

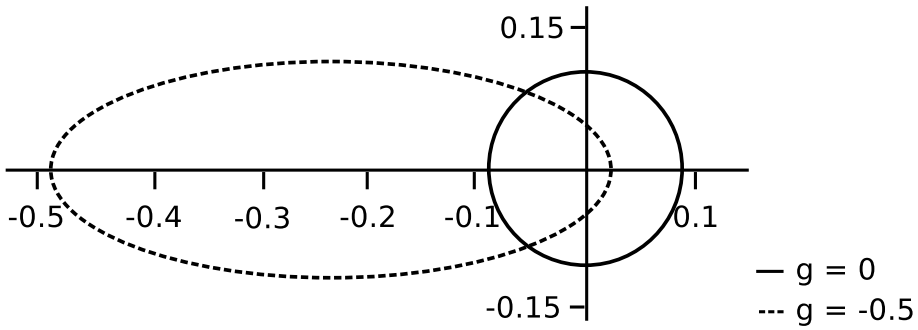

FIGURE 2.8 – Fonction de phase de Henyey-Greenstein. Les axes représentent la probabilité de l'évènement selon l'angle θ.

2.3.5 Équation de transfert

Les quatre concepts décrits précédemment : l'émission, la dispersion en entrée, la dispersion en sortie et l'absorption, permettent de décrire le comportement général de la lumière à l'intérieur d'un milieu participatif. L'équation résultante donnant la valeur différentielle de la radiance en un point donné du volume et dans une direction spécifique $\vec{\omega}$ se nomme l'équation de transfert :

$$(\vec{\omega} \cdot \vec{\nabla})L(\mathbf{x}, \vec{\omega}) = -\sigma_t L(\mathbf{x}, \vec{\omega}) + L_e(\mathbf{x}, \vec{\omega}) + \sigma_s \int_{4\pi} p(\mathbf{x}, \theta) L(\mathbf{x}, \vec{\omega_i}) d\vec{\omega_i} + Q(\mathbf{x}, \vec{\omega}). \quad (2.1)$$

La valeur $Q(\mathbf{x}, \vec{\omega})$ représente l'apport en énergie dû aux évènements de dispersion unique. Comme le montre la figure 2.9, cette valeur représente la quantité d'énergie due à la source lumineuse n'ayant pas encore été diffusée à l'intérieur du matériau. Pour calculer l'apport $Q(\mathbf{x}, \vec{\omega})$ en direction $\vec{\omega}$, nous devons intégrer l'énergie non-diffusée en provenance de l'ensemble des directions $\vec{\omega}'$:

$$Q(\mathbf{x}, \vec{\omega}) = \sigma_s \int_{4\pi} p(\mathbf{x}, \theta) L_{ri}(\mathbf{x}, \vec{\omega}') d\vec{\omega}'. \quad (2.2)$$

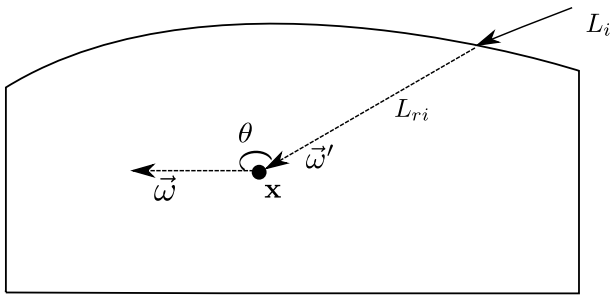

FIGURE 2.9 – Représentation d'un évènement de diffusion unique.

L'équation de transfert servira de base à l'évaluation du transport de la radiance à l'intérieur de matériaux hétérogènes (où les valeurs σ_a, σ_s et p peuvent être différentes en chaque point d'un volume).

2.4 Approximation de diffusion de la lumière

L'équation de transfert peut s'avérer d'une grande complexité à résoudre. Les méthodes de Monte-Carlo sont une avenue possible mais elles demeurent toutefois trop coûteuses pour obtenir une résolution en temps réel si on désire une approximation sans bruit. Il est tout de même possible de simplifier l'équation en contraignant quelque peu la classe de matériaux semi-transparents aux matériaux ayant une valeur d'albédo élevée. L'albédo est défini comme étant le ratio entre le coefficient de diffusion et le coefficient d'extinction :

$$\alpha = \frac{\sigma_s}{\sigma_t}.$$

Une valeur élevée d'albédo α indique un matériau ayant une plus grande propension à disperser la lumière plutôt qu'à l'absorber. Le principe de similarité stipule que les matériaux ayant un albédo élevé peuvent être représentés par une fonction de phase isotropique. Ce principe se base sur le fait qu'après plusieurs évènements de diffusion à l'intérieur d'un matériau la composante directionnelle de la lumière diminue de plus en plus au profit d'une distribution homogène.

La dérivation de l'approximation de diffusion est basée sur ce principe (voir la thèse de Donner [Don06] pour la dérivation détaillée). Étant donnée l'approximation à une fonction de phase isotropique, la fonction utilise un coefficient de dispersion réduit décrit comme suit : $\sigma'_s = \sigma_s(1-g)$ (à ne pas confondre avec le g de la fonction de phase de Henyey-Greenstein). Intuitivement le paramètre g permet de manipuler la densité

du matériau. Prenons par exemple, une valeur de g négative indiquant un matériau ayant une propension à la diffusion vers l'arrière et une direction de diffusion $\vec{\omega}$ vers l'intérieur du matériau. Ce comportement est incorporé dans le coefficient de dispersion réduit en simulant une plus grande densité du matériau comme le montre la figure 2.10.

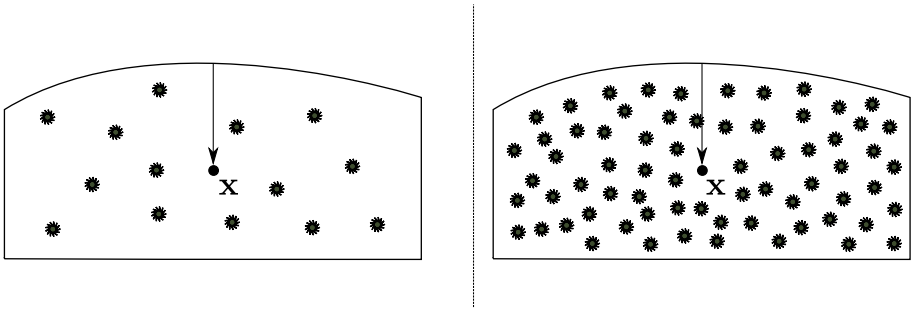

FIGURE 2.10 – Effet de g sur la réduction du coefficient σ_s. À gauche, $g = 0$, et à droite $g < 0$. Nous remarquons que donner une valeur négative à g revient à simuler une substance de plus grande densité.

Le coefficient d'extinction est aussi modifié pour tenir compte du coefficient de dispersion donnant le coefficient d'extinction réduit :

$$\sigma_t' = \sigma_a + \sigma_s'.$$

La simplification de l'équation de transfert requiert une seconde approximation cette fois-ci à partir de la définition de la radiance en un point du volume. Pour ce faire, la radiance au point $\mathbf{x}$ est encodée à partir du flux radiatif et de l'irradiance directionnelle en un point (on peut aussi d'une manière équivalente définir la radiance à partir des quatre premiers termes des harmoniques sphériques) :

$$L(\mathbf{x}, \vec{\omega}) = \frac{1}{4\pi}\phi(\mathbf{x}) + \frac{3}{4\pi}\vec{E}(\mathbf{x}) \cdot \vec{\omega}. \tag{2.3}$$

Le flux radiatif est calculé en intégrant la radiance $L(\mathbf{x}, \vec{\omega})$ sur l'ensemble des directions $\vec{\omega}$:

$$\phi(\mathbf{x}) = \int_{4\pi} L(\mathbf{x}, \vec{\omega}) d\vec{\omega}.$$

L'irradiance directionnelle pour sa part, est calculée d'une manière similaire mais en modulant la radiance par le vecteur directionnel $\vec{\omega}$:

$$\vec{E}(\mathbf{x}) = \int_{4\pi} L(\mathbf{x}, \vec{\omega}) \vec{\omega} d\vec{\omega}.$$

Il est intéressant de remarquer qu'avec cet encodage, la directionnalité de la radiance est presque disparue, seule l'irradiance directionnelle donne une approximation grossière de la fonction originale. Pour obtenir l'approximation de diffusion, l'équation 2.3 est injectée dans l'équation 2.1 donnant (voir le livre d'Ishimaru [Ish78]) :

$$\vec{\nabla} \cdot (\kappa_d(\mathbf{x})\vec{\nabla}\phi(\mathbf{x})) = \sigma_a(\mathbf{x})\phi(\mathbf{x}) - Q_0(\mathbf{x}) + 3\kappa_d(\mathbf{x})\vec{\nabla} \cdot Q_1(\mathbf{x}). \tag{2.4}$$

$\kappa_d(\mathbf{x})$ est simplement une redéfinition des coefficients d'absorption et d'extinction introduite par l'injection dans l'équation 2.1 :

$$\kappa_d(\mathbf{x}) = \frac{1}{3\sigma_t'(\mathbf{x})}.$$

$Q_0(\mathbf{x})$ représente l'intégrale sur l'ensemble des directions de l'apport en énergie dû aux évènements de diffusion unique $Q(\mathbf{x}, \vec{\omega})$ défini sur l'ensemble des directions $\vec{\omega}$ (voir l'équation 2.2) :

$$Q_0(\mathbf{x}) = \int_{4\pi} Q(\mathbf{x}, \vec{\omega}) d\vec{\omega}.$$

Pour simplifier la résolution de l'équation de diffusion, ce facteur peut être ignoré et calculé séparément (par tracé de rayons par exemple ou à l'aide d'une méthode de *depth peeling*[1]).

Pour sa part, la valeur $Q_1(\mathbf{x})$ donne le terme de premier ordre du facteur de source $Q(\mathbf{x}, \vec{\omega})$:

$$Q_1(\mathbf{x}) = \int_{4\pi} Q(\mathbf{x}, \vec{\omega})\vec{\omega} d\vec{\omega}.$$

Par principe de similarité, nous savons qu'avec des matériaux à valeur élevée d'albédo il est valide de remplacer la fonction de phase par une fonction de phase isotropique. Dans ce cas, les valeurs de $Q(\mathbf{x}, \vec{\omega})$ seront identiques sur l'ensemble des directions, donnant donc une évaluation nulle de l'intégrale. Pour le calcul des valeurs du flux radiatif internes, nous obtenons l'équation simplifiée suivante :

$$\vec{\nabla} \cdot (\kappa_d(\mathbf{x})\vec{\nabla}\phi(\mathbf{x})) = \sigma_a(\mathbf{x})\phi(\mathbf{x}). \tag{2.5}$$

Par contre, pour le calcul à la frontière de l'objet nous devons tenir compte de l'équation 2.4 de diffusion complète car au voisinage de la surface de l'objet, la composante de la lumière L_{ri} est toujours présente et non-diffusée.

[1] Voir la section 3.3.

Chapitre 3

Travaux antérieurs

Dans ce chapitre nous ferons un survol des travaux antérieurs appliqués au rendu et à la simulation de matériaux semi-transparents. Une attention particulière sera portée aux méthodes de résolution temps réel sur processeur graphique.

3.1 Méthodes *offline*

Bien que le sujet de ce mémoire porte sur la résolution de l'équation de diffusion en temps réel, il peut être utile de faire un survol des méthodes dites *offline* étant donnée l'influence qu'ont eu ces dernières sur le développement des méthodes interactives.

Plusieurs articles ont été publiés sur la résolution de l'équation de transfert par méthodes de Monte-Carlo [HK93, DEJ+99, PH00]. Ces méthodes permettent d'obtenir des résultats de simulation très convaincants et d'une grande précision au détriment d'un temps de calcul nécessitant souvent plusieurs heures de traitements. Le coût élevé de la simulation provient principalement du grand nombre de rayons devant être lancés pour obtenir des résultats valides et non-bruités (un problème commun à ces méthodes). Bien que certaines améliorations peuvent être apportées comme l'échantillonnage par importance ou à basses divergences (méthodes quasi-Monte-Carlo), les temps de simulation restent toujours très loins des performances interactives.

Pour améliorer ces temps de calcul, Jensen et Christensen [JC98] ont adapté la technique de cartes de photons pour la simulation du transport de la lumière au rendu de matériaux semi-transparents. Cette technique permet de réduire considérablement

les temps de calcul mais reste tout de même limitée aux matériaux à faible albédo. Pour contrer cette limitation, Jensen et al. [JMLH01] proposent d'utiliser une méthode basée sur l'approximation de l'équation de diffusion à l'aide d'une distribution de lumières ponctuelles à deux pôles (une lumière positive à l'intérieur de l'objet et l'autre négative à l'extérieur) sur l'ensemble de la surface de l'objet. Pour calculer la lumière diffuse sortant en un point, la contribution de chaque dipôle est prise en compte, ceux-ci étant encodés à l'intérieur d'une structure accélératrice [JB05] permettant un regroupement efficace des valeurs.

3.2 Approximation de diffusion par rendu volumique

Cette section s'attardera aux travaux antérieurs portant sur les méthodes de résolution volumique de l'équation de diffusion.

3.2.1 Diffusion sur grille homogène

Stam [Sta95] est le premier à utiliser l'approximation de l'équation de diffusion pour le rendu de milieux participatifs en infographie. Bien que cette méthode ne soit pas interactive et concerne exclusivement les milieux participatifs homogènes, il est tout de même pertinent de s'y attarder étant données la similitude et la grande influence qu'a eue cette dernière sur les techniques temps réel.

Stam propose d'utiliser avec cette technique une méthode de résolution itérative basée sur la discrétisation de l'équation de diffusion sur une grille 3D. La grille représente un ensemble d'éléments de volume de même taille contenant chacun une valeur du flux radiatif. Ce flux radiatif est par la suite mis à jour itérativement à l'aide d'une méthode numérique de différences finies. Pour accélérer la convergence de sa méthode, Stam utilise plusieurs niveaux de grilles de tailles différentes en initialisant les niveaux supérieurs à partir des résultats de simulation des niveaux à basse résolution (appelée technique multi-résolution en calcul numérique).

3.2.2 Carte d'ombrage de materiaux semi-transparents

Une méthode intéressante utilisant une extension aux cartes d'ombrage (*shadow map*) a été proposée par Dachsbacher et Stamminger [DS03] pour le rendu de matériaux semi-transparents. Tout comme l'algorithme conventionnel de carte d'ombrage, le rendu de

l'objet se fait d'abord du point de vue de la lumière. L'utilisation de plusieurs tampons de destination (*render target*) permet non seulement de stocker la position du fragment mais aussi l'irradiance transmise dans le matériau. Par la suite, un second rendu, cette fois-ci à partir du point de vue de la caméra, est effectué en échantillonnant la carte d'ombrage pour calculer l'énergie sortante à la position du fragment résultant. Bien que cette méthode soit contrainte aux matériaux homogènes, l'idée d'accumuler l'énergie transmise dans un tampon du point de vue de la lumière est intéressante et sera utilisée dans la méthode que nous proposons.

3.2.3 Rendu de matériaux semi-transparents hétérogènes en temps réel

Cette section s'intéressera aux plus récentes méthodes de la littérature pour le rendu de matériaux semi-transparents hétérogènes en temps réel. Ces techniques formeront la base sur laquelle notre méthode, décrite au prochain chapitre, sera construite.

La méthode de Stam [Sta95] décrite précédemment a été reprise et adaptée par Wang et al. [WZT+08] pour le rendu de matériaux semi-transparents. L'approche est basée principalement sur la construction d'une grille nommée *polygrid* construite à partir d'un maillage triangulaire. Après sa construction, cette grille est chargée dans un ensemble de textures en mémoire graphique. Ces textures permettent de garder non seulement la topologie de la grille mais aussi le flux radiatif en chaque élément de la grille. L'équation de diffusion est par la suite résolue sur processeur graphique à l'aide d'une méthode de différences finies de manière similaire à celle de [Sta95]. La méthode de construction *ad hoc* de la *polygrid* constitue un facteur limitant de cette méthode étant donnés les résultats variant selon la construction de la grille et du maillage triangulaire en entrée.

Pour contrer cette limitation, Wang et al. [WWH+10] proposent de remplacer la construction de la grille par une tétraédrisation du maillage. La tétraédrisation se construit en remplissant le volume défini par la frontière du maillage par un ensemble de tétraèdres où les faces triangulaires s'apparentent le plus possible au maillage original. Cette tétraédrisation doit s'effectuer lors d'une étape de pré-calcul, ce qui empêche les déformations sévères du maillage en temps réel. De plus, le maillage original étant modifié par cette tétraédrisation, il peut s'avérer difficile de récupérer les attributs de

points (*vertex attributes*) définis par les artistes lors de la création du maillage (coordonnées de texture, vecteurs tangent et bi-tangent, etc.). L'intérêt principal d'utiliser une tétraédrisation est d'obtenir une conformité du domaine de résolution de l'équation de diffusion par rapport au maillage original, les deux volumes étant presque identiques. Par contre, en pratique, l'utilité de cette conformité est débattable étant donnée la nature diffuse des matériaux à albédo élevé.

Une technique très performante proposée par Borlum et al. [BCK+11], offre la possibilité d'effectuer une simulation de la diffusion de la lumière de manière complètement dynamique. Cette méthode permet donc une déformation complète du maillage, un changement de positions et des caractéristiques des sources de lumière en temps réel ainsi qu'une modification dynamique des propriétés hétérogènes du matériau. Pour ce faire, ils simplifient au maximum le domaine de diffusion en ignorant complètement la frontière du maillage. L'énergie est tout d'abord initialisée à la manière des cartes d'ombrage de Dachsbacher et Stamminger [DS03] décrite précédemment, le transport de la lumière s'effectuant par la suite sur une grille sans frontière en gardant une composante directionnelle encodée par harmoniques sphériques. Cette technique donne des résultats peu précis étant donnée la frontière inexistante du domaine de résolution mais le rendu reste très efficace et convaincant.

3.3 *Depth peeling* et ses variantes

De manière générale, un processeur graphique moderne peut se résumer par ses deux grandes actions : le traitement de la géométrie suivi du traitement des fragments résultants. Les fragments sont générés à partir de la projection de la géométrie sur un plan 2D représentant le point de vue de la caméra. C'est l'échantillonnage de ce plan 2D qui est responsable de la génération des fragments. Généralement, l'échantillonnage se fait au centre des pixels résultant de la discrétisation du plan. Chaque fragment représente donc une couche de géométrie à un pixel donné. Certains algorithmes, comme ceux apparentés au rendu d'objets transparents ou semi-transparents, doivent traiter chacun de ces fragments pour effectuer un rendu valide. Les algorithmes de *depth peeling* permettent de trier et de traiter l'ensemble de ces fragments dans l'ordre désiré.

Dans cette section, nous nous attarderons aux méthodes de *depth peeling* utilisées pour la simulation de diffusion de la lumière par évènement unique.

3.3.1 Algorithme classique

L'algorithme classique de *depth peeling* a été proposé par Mammen [Mam89] et adapté sur processeur graphique par Everitt [Eve01]. L'ensemble des fragments est obtenu en effectuant plusieurs passes de rendu à l'aide de deux tampons de profondeur. L'algorithme peut se résumer en trois grandes étapes :

1. Rendu de la scène pour remplir le tampon de profondeur 0.

2. Second rendu de la scène cette fois-ci en utilisant le tampon de profondeur 1. Le tampon 0 est pour sa part utilisé comme texture en entrée permettant le rejet des fragments ayant une profondeur inférieure ou égale à son contenu.

3. Les tampons de profondeur 0 et 1 sont interchangés et les étapes 1 et 2 sont répétées tant qu'il reste des fragments à traiter.

Avec cet algorithme, nous obtenons en plusieurs passes l'ensemble des fragments représentant chaque pixel. Cette méthode a pour avantage de pouvoir être exécutée sur presque tous les processeurs graphiques étant donné qu'elle ne requiert aucune capabilité particulière. Par contre, le nombre de passes de rendu arbitraire demeure son défaut le plus flagrant.

3.3.2 Variantes de l'algorithme

Aux cours des dernières années, les avancées au niveau du matériel graphique ont permis d'améliorer grandement l'efficacité de cette technique. Pour certaines méthodes approximatives, il peut être utile de garder seulement la profondeur minimum et celle maximum d'un fragment. Pour ce faire, Bavoil et Myers [BM08] proposent d'utiliser la fonctionnalité **min**/**max** du mélangeur de sortie (*blending stage*). À partir d'une texture de format point flottant `RG` et de l'option **max** du mélangeur de sortie, il est possible d'obtenir les fragments de profondeurs minimum et maximum en redirigeant la profondeur z dans les composants R et G de la façon suivante :

$$[R, G] \leftarrow [z, -z].$$

Le mélangeur de sortie ne gardera que les valeurs maximum et minimum de profondeur et ce en une seule passe de géométrie. Voici un exemple en initialisant le tampon de sortie à $[R, G] \leftarrow [1.0, 1.0]$ et en utilisant la fonction **min** du mélangeur de sortie :

- Arrivé du fragment $z = 0.6 : [0.6, -0.6] \leftarrow [\mathbf{min}(0.6, 1.0), \mathbf{min}(-0.6, 1.0)]$

- Arrivé du fragment $z = 0.2 : [0.2, -0.6] \leftarrow [\mathbf{min}(0.2, 0.6), \mathbf{min}(-0.2, -0.6)]$

- Arrivé du fragment $z = 0.3 : [0.2, -0.6] \leftarrow [\mathbf{min}(0.3, 0.2), \mathbf{min}(-0.3, -0.6)]$

La distance minimum est obtenue en prenant la composante $R \rightarrow z_{min} = 0.2$ et la distance maximum en prenant la valeur absolue de la composante $G \rightarrow z_{max} = \mathbf{abs}(-0.6)$.

Par contre, pour obtenir plus de deux couches, des passes de géométrie supplémentaires doivent être exécutées. Cette technique permet tout de même de réduire de moitié le nombre de passes nécessaires à l'obtention de l'ensemble des fragments.

Pour réduire cette limitation, Liu et al. [LHLW09] proposent d'utiliser la possibilité d'écrire vers plusieurs textures de sortie (8 avec les processeurs graphiques actuels) à partir du nuanceur de fragment. En premier lieu, l'écart maximal de profondeur est déterminé et séparé en 32 intervalles distincts (8 textures de sortie $\times$ 4 composants par texture `RGBA` = 32). Par la suite la technique **min/max** de Bavoil et Myers [BM08] est utilisée en redirigeant le fragment vers le bon intervalle. Cette méthode permet d'obtenir 32 couches en une seule passe. Par contre, les intervalles étant fixes, il est possible de manquer certaine profondeurs si l'on retrouve plus de deux fragments par intervalle.

La dernière méthode, proposée par Gruen et Thibieroz [GT10], offre une solution complètement générique au problème. Il est maintenant possible, avec les tout derniers processeurs graphiques (de classe DX11/OGL4), d'implémenter des listes chaînées en mémoire graphique. Cette nouvelle possibilité ouvre la porte à l'implémentation d'un grand nombre d'algorithmes impossibles auparavant (les *ABuffer* par exemple). Pour le *depth peeling*, les listes chaînées sont utilisées pour stocker un nombre arbitraire de fragments par pixel permettant d'obtenir l'ensemble des profondeurs exactes par pixel. Étant donné qu'il est toujours impossible d'allouer dynamiquement de la mémoire vidéo à partir des nuanceurs, l'emplacement mémoire où sont alloués les noeuds des listes chaînées doit être créé préalablement à l'exécution de la technique. Malgré cette limitation, cette technique demeure la plus efficace et la plus complète.

3.3.3 Rendu de matériaux semi-transparents homogènes en temps réel

Une des applications des méthodes de *depth peeling*, outre le rendu d'objets transparents, est la simulation de la transmittance de la lumière à l'intérieur d'objets de forme arbitraire. Cette technique utilise la différence de profondeur entre deux couches géométriques pour évaluer la quantité de lumière absorbée par le matériau. En utilisant la couche la plus près et la plus éloignée du point de vue de la lumière, nous pouvons obtenir la quantité de lumière absorbée à l'aide de la loi de Beer [Gre04] :

$$L_o(\mathbf{x}) = e^{-\sigma_t d} L_i(\mathbf{x_i}).$$

Par contre, cette méthode reste contrainte aux matériaux homogènes et ne garde que deux profondeurs pour le calcul de la transmittance, il est donc possible de manquer certains détails de la géométrie comme le montre la figure 3.1.

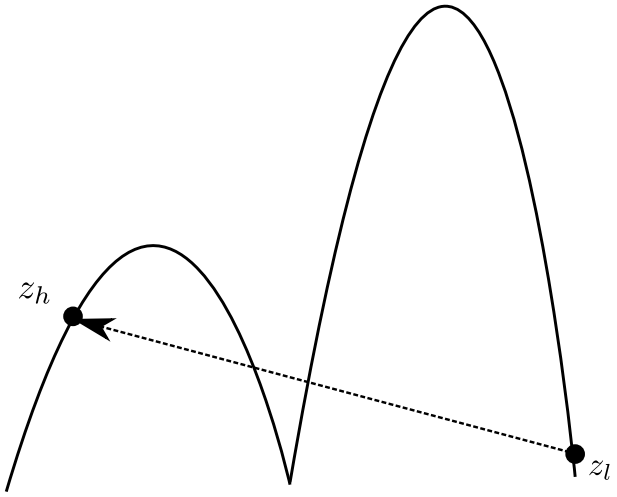

FIGURE 3.1 – Limitation avec deux couches géométriques.

La figure 3.2 montre que cette technique arrive tout de même à donner des résultats assez intéressants.

FIGURE 3.2 – Rendu avec la méthode de Green [Gre04].

Chapitre 4

Approximation de diffusions multiples sur *GPU*

Ce chapitre sera consacré à la description de notre méthode de diffusion à évènements multiples sur processeur graphique. Notre technique peut être considérée comme une extension des méthodes mentionnées aux chapitres précédents, en réduisant les limitations de ces dernières tout en conservant une simulation en temps réel. Les caractéristiques de notre technique se résument ainsi :

- Support de maillage complètement déformable.
- Utilisation de sources de lumière dynamiques (position, intensité, etc.).
- Édition des paramètres du matériau en temps réel.
- Résolution de l'équation du diffusion sur une voxélisation du maillage, permettant d'obtenir une simulation efficace tout en tenant compte des frontières.

4.1 Survol du pipeline de diffusion

La simulation de diffusion est entièrement réalisée sur processeur graphique, en utilisant un ensemble hétérogène de noyaux et de nuanceurs *OpenCL*/*OpenGL*. La figure 4.1 montre les différentes étapes du pipeline de diffusion en partant du maillage triangulaire original jusqu'à la couleur diffuse finale.

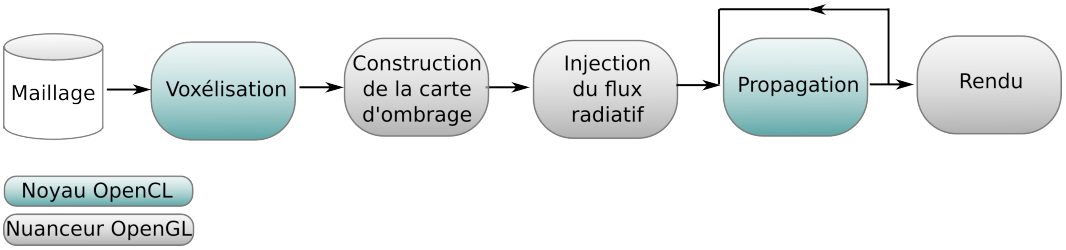

FIGURE 4.1 – Pipeline de rendu.

Chaque bulle de la figure 4.1 représente une ou plusieurs exécutions d'un noyau *OpenCL*/*OpenGL*. Les prochaines sections décriront en détail chacune des étapes du pipeline de rendu. Le prochain chapitre sera consacré à une analyse des performances et des résultats du rendu.

4.2 Voxélisation du maillage triangulaire

La première étape de notre pipeline de diffusion consiste en la voxélisation du maillage triangulaire en entrée. Ce maillage représente l'objet semi-transparent dans lequel le processus de diffusion sera appliqué. Comme le montre la figure 4.2, la voxélisation permet de simplifier le maillage en discrétisant le volume occupé par ce dernier.

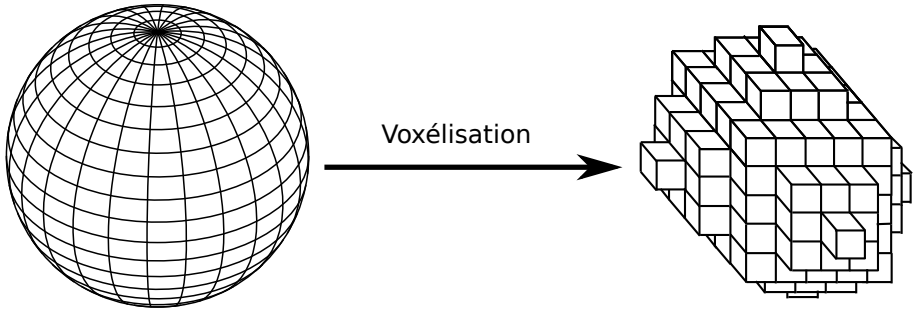

FIGURE 4.2 – Voxélisation d'une sphère.

Étant donnée la structure orthogonale résultant de cette voxélisation, l'étape de diffusion s'en retrouvera de beaucoup simplifiée. De plus, en exploitant le parallélisme des processeurs graphiques modernes, il est possible d'obtenir une voxélisation du maillage en temps réel, permettant ainsi de supprimer l'étape de pré-calcul des techniques précédentes [WZT+08, WWH+10].

Comme le montre la figure 4.3, l'opération de voxélisation est composée de deux étapes distinctes. La première consiste en la voxélisation du maillage au niveau de raffi-

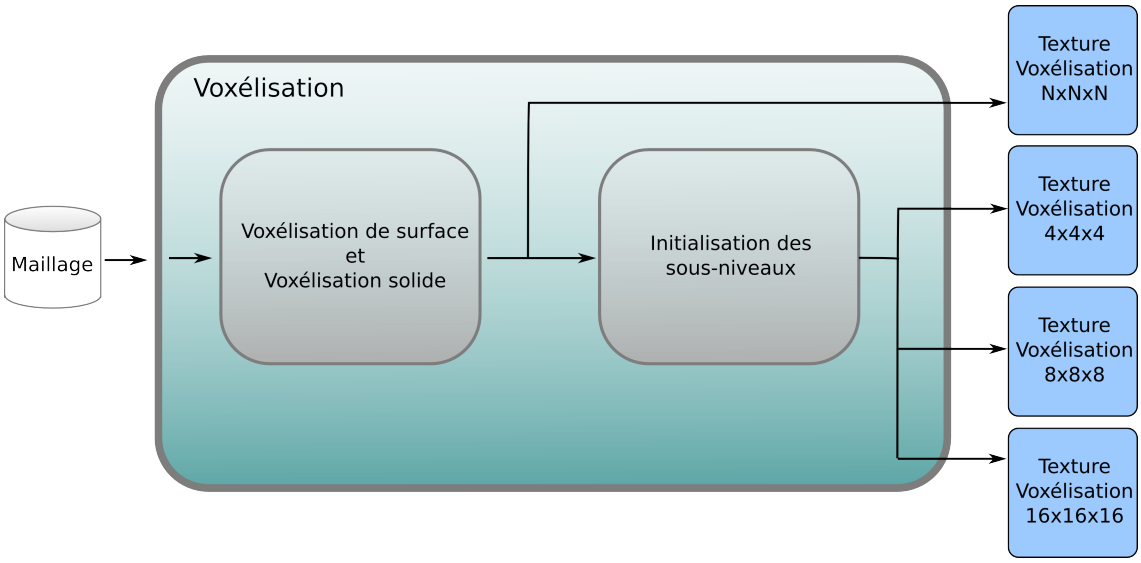

FIGURE 4.3 – Pipeline de voxélisation.

nement souhaité. Par la suite, la seconde étape initialise les sous-niveaux de voxélisation à partir de la voxélisation haute résolution. Ces sous-niveaux seront utilisés par l'étape de diffusion pour permettre de résoudre l'équation à l'aide de la méthode de différences finies multi-résolution.

4.2.1 Algorithme de voxélisation

L'algorithme de voxélisation se base sur la technique proposée par Schwarz et Seidel [SS10]. Celle-ci se divise en deux étapes distinctes : la voxélisation de la surface et la voxélisation de l'intérieur du maillage.

Voxélisation de surface

L'algorithme traite indépendemment chaque triangle T du maillage, identifié à partir de ses sommets $\mathbf{v}_0$, $\mathbf{v}_1$, $\mathbf{v}_2$ et sa normale $\vec{\mathbf{n}} = (\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_2 - \mathbf{v}_0)$, l'enroulement (*winding*) des sommets étant supposé dans le sens anti-horaire. Pour obtenir une voxélisation conservatrice, le triangle T doit donc vérifier son chevauchement avec chaque voxel V défini par le coin minimum $\mathbf{p}$ et l'extension au coin maximum $\Delta\mathbf{p}$ (voir la figure 4.4). Pour qu'un triangle T soit considéré comme chevauchant un voxel V celui-ci doit satisfaire les deux conditions suivantes :

1. Le plan P dans lequel se retrouve T doit chevaucher V.

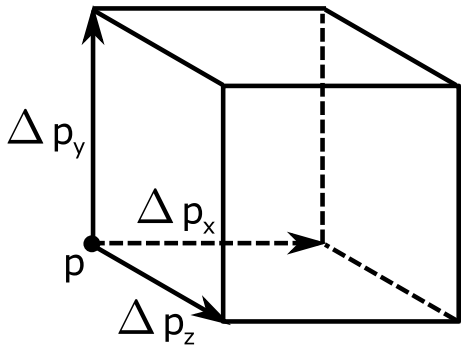

FIGURE 4.4 – Mesure du voxel.

2. Pour chacun des trois plans xy, xz et yz, les projections 2D de T et V sur ces plans doivent se chevaucher.

Vérification de la condition 1 :

Pour vérifier la condition 1, nous aurons besoin du point critique du voxel V défini à partir de la normale du triangle :

$$\mathbf{c} = \left(\left\{ \begin{array}{ll} \Delta\mathbf{p}_x, & \mathbf{n}_x > 0 \\ 0, & \mathbf{n}_x \leq 0 \end{array} \right\}, \left\{ \begin{array}{ll} \Delta\mathbf{p}_y, & \mathbf{n}_y > 0 \\ 0, & \mathbf{n}_y \leq 0 \end{array} \right\}, \left\{ \begin{array}{ll} \Delta\mathbf{p}_z, & \mathbf{n}_z > 0 \\ 0, & \mathbf{n}_z \leq 0 \end{array} \right\} \right).$$

Le point critique est utilisé pour évaluer la distance entre les points extrêmes du voxel V par rapport au plan P :

$$\begin{array}{ll} d_1 = & \vec{\mathbf{n}} \cdot (\mathbf{c} - \mathbf{v}_0) \\ d_2 = & \vec{\mathbf{n}} \cdot ((\Delta\mathbf{p} - \mathbf{c}) - \mathbf{v}_0). \end{array}$$

Une fois les distances d_1 et d_2 connues, nous pouvons vérifier le chevauchement du triangle T au plan P en s'assurant que les points $\mathbf{p}$ et $\mathbf{p} + \Delta\mathbf{p}$ ne se retrouvent pas dans le même demi-espace défini par P. Le prédicat suivant exprime cette condition (la position $\mathbf{p}$ étant utilisée comme vecteur) :

$$(\vec{\mathbf{n}} \cdot \vec{\mathbf{p}} + d_1)(\vec{\mathbf{n}} \cdot \vec{\mathbf{p}} + d_2) \leq 0. \tag{4.1}$$

Vérification de la condition 2 :

Une fois le sous-ensemble de voxels chevauchant le plan P connu, nous devons par la suite restreindre ce sous-ensemble au triangle T (la condition 1 étant appliquée au plan P) en vérifiant le chevauchement des projections 2D du triangle T et du voxel V. Pour ce faire nous utiliserons les fonctions d'arêtes du triangle T pour mesurer la distance aux coins du voxel sur les plans de projection xy, xz et yz. Les arêtes du triangle T

étant définies comme suit,

$$\begin{aligned}\vec{\mathbf{e}}_0 &= v_1 - v_0\\ \vec{\mathbf{e}}_1 &= v_2 - v_1\\ \vec{\mathbf{e}}_2 &= v_0 - v_2.\end{aligned}$$

et leurs normales (exemple par rapport au plan xy) :

$$\begin{aligned}\vec{\mathbf{n}}^{xy}_{e_0} &= (-\mathbf{e}_{0,y}, \mathbf{e}_{0,x}, 0) \cdot \left\{ \begin{array}{ll} 1, & \mathbf{n}_z \geq 0 \\ -1, & \mathbf{n}_z < 0 \end{array} \right\}\\ \vec{\mathbf{n}}^{xy}_{e_1} &= (-\mathbf{e}_{1,y}, \mathbf{e}_{1,x}, 0) \cdot \left\{ \begin{array}{ll} 1, & \mathbf{n}_z \geq 0 \\ -1, & \mathbf{n}_z < 0 \end{array} \right\}\\ \vec{\mathbf{n}}^{xy}_{e_2} &= (-\mathbf{e}_{2,y}, \mathbf{e}_{2,x}, 0) \cdot \left\{ \begin{array}{ll} 1, & \mathbf{n}_z \geq 0 \\ -1, & \mathbf{n}_z < 0 \end{array} \right\}.\end{aligned}$$

La figure 4.5 montre la signification de ces valeurs sur le triangle T.

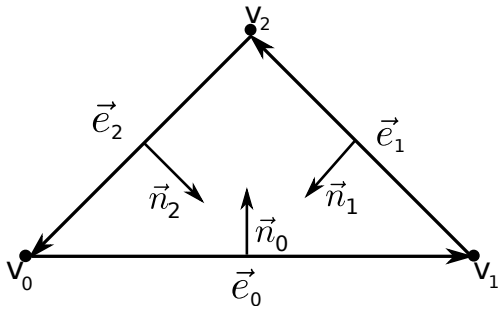

FIGURE 4.5 – Propriétés de voxélisation du triangle T.

Par la suite, la distance de l'arête au point extrême du voxel V est déduite comme suit (la position du sommet $\mathbf{v}$ étant vectorisée : $\vec{\mathbf{v}} = \mathbf{v} - (0,0,0)$) :

$$d^{xy}_{e_i} = -(\vec{\mathbf{n}}^{xy}_{e_i} \cdot \vec{\mathbf{v}}_{i,xy}) + \max(0, \Delta\mathbf{p}_x \vec{\mathbf{n}}^{xy}_{e_i,x}) + \max(0, \Delta\mathbf{p}_y \vec{\mathbf{n}}^{xy}_{e_i,y}). \tag{4.2}$$

Il est possible de déduire le chevauchement en vérifiant que la distance du point $\mathbf{p}$ dans le plan de projection 2D additionnée à la distance calculée par l'équation 4.2 soit supérieure à 0 et ce pour l'ensemble des arêtes. Cette condition s'exprime comme suit :

$$\bigwedge_{i=0}^{2} (\vec{\mathbf{n}}^{xy}_{e_i} \cdot \mathbf{p}_{xy} + d^{xy}_{e_i} \geq 0). \tag{4.3}$$

Donc, pour connaître si un voxel V touche au triangle T, nous devons vérifier les predicats 4.1 et 4.3 (pour chaque plan xy, yz et xz) déterminant ainsi la voxélisation de surface.

Voxélisation solide

L'algorithme de voxélisation de l'intérieur de l'objet s'exécute de manière similaire à la voxélisation de surface. En premier lieu, le chevauchement de la projection du triangle et du voxel sur le plan xy est vérifié à partir d'une équation similaire à l'équation 4.3.

$$\bigwedge_{i=0}^{2}(\vec{\mathbf{n}}_{e_i}^{xy} \cdot \mathbf{p}_{xy} + d_{e_i}^{xy} + f_{e_i}^{xy} > 0). \tag{4.4}$$

Au lieu d'effectuer le test à partir des coins extrêmes du voxel, l'équation 4.4 utilise plutôt le centre du voxel défini comme suit :

$$\mathbf{p}_{xy} = (\mathbf{p}_x + \Delta\mathbf{p}_x, \mathbf{p}_y + \Delta\mathbf{p}_y, 0).$$

Pour sa part, le facteur $f_{e_i}^{xy}$ est utilisé pour éviter de voxéliser plusieurs fois les arêtes partagées entre les triangles. Pour ce faire, la règle *haut-gauche* est utilisée, en tenant compte seulement des arêtes hautes ($\vec{\mathbf{n}}_{e_i,x}^{xy} = 0 \wedge \vec{\mathbf{n}}_{e_i,y}^{xy} < 0$) et gauches ($\vec{\mathbf{n}}_{e_i,x}^{xy} > 0$) d'un triangle (voir la documentation Direct3D pour plus de détails sur la règle *haut-gauche* utilisée pour la *rasterization* de triangles). Le facteur $f_{e_i}^{xy}$ est défini comme suit :

$$f_{e_i}^{xy} = \begin{cases} \textbf{FloatMin}, & \vec{\mathbf{n}}_{e_i,x}^{xy} = 0 \wedge \vec{\mathbf{n}}_{e_i,y}^{xy} < 0 \\ 0, & otherwise. \end{cases}$$

La valeur **FloatMin** représente la plus petite valeur en représentation point flottant sur l'architecture courante.

Une fois le voxel identifié sur le plan xy, le centre de celui-ci est projeté sur l'axe des z permettant ainsi d'obtenir sa position $\mathbf{p}_{vz}$. Cette position est par la suite utilisée pour créer un masque de bits permettant d'obtenir la voxélisation interne du maillage. Le masque de bits est défini en activant chacun des bits représentant une profondeur de voxel en z (n étant égal à 32 car nous utilisons des entiers 32 bits) :

$$masque \leftarrow 0, 0, 0, ..., \lfloor \mathbf{p}_{vz} + 0.5 \rfloor, 1, 1, 1, ..., n - 1.$$

Une opération *ou-exclusive* est par la suite appliquée entre le masque et la valeur actuelle de voxélisation comme le montre la figure 4.6.

Il est important de mentionner que l'algorithme de voxélisation interne n'est valide que pour un maillage fermé (*watertight*). Si celui-ci est appliqué sur une frontière ouverte, l'opération *ou-exclusive* sera incomplète et nous obtiendrons une voxélisation erronée.

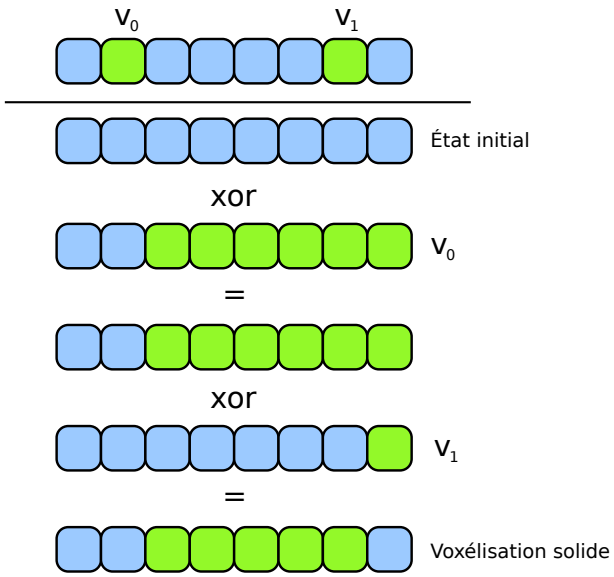

FIGURE 4.6 – Voxélisation solide.

4.2.2 Création des sous-niveaux

La création de sous-niveaux de voxélisation est nécessaire pour l'exécution de la méthode de différences finies multi-résolution. Cette étape utilise la voxélisation créée lors de l'étape précédente pour générer une voxélisation à différents niveaux de résolution comme le montre la figure 4.7.

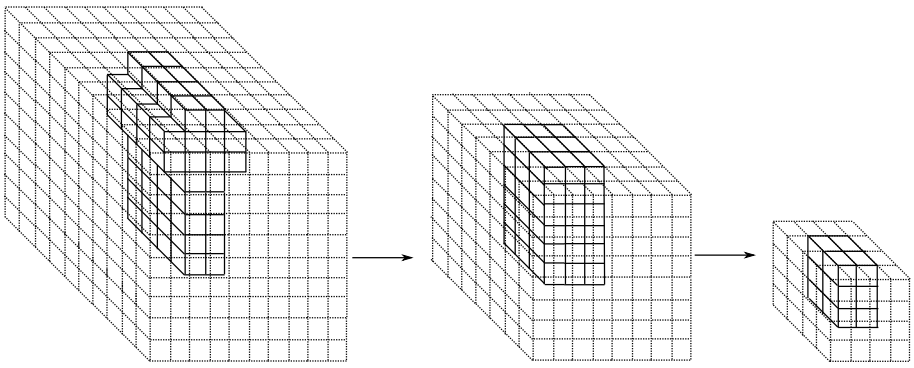

FIGURE 4.7 – Initialisation des sous-niveaux.

L'algorithme remplissant les sous-niveaux est simple et se résume ainsi : un voxel de la sous-résolution est considéré actif si son volume englobant contient un voxel actif de la voxélisation haute résolution.

4.2.3 Implémentation

La voxélisation du maillage est réalisée entièrement sur *GPU* à partir des triangles contenus à l'intérieur de tampons de sommets. Comme le montre la figure 4.8, chaque triangle est traité indépendemment par l'exécution d'un noyau *OpenCL* vérifiant les conditions de voxélisation décrites précédemment.

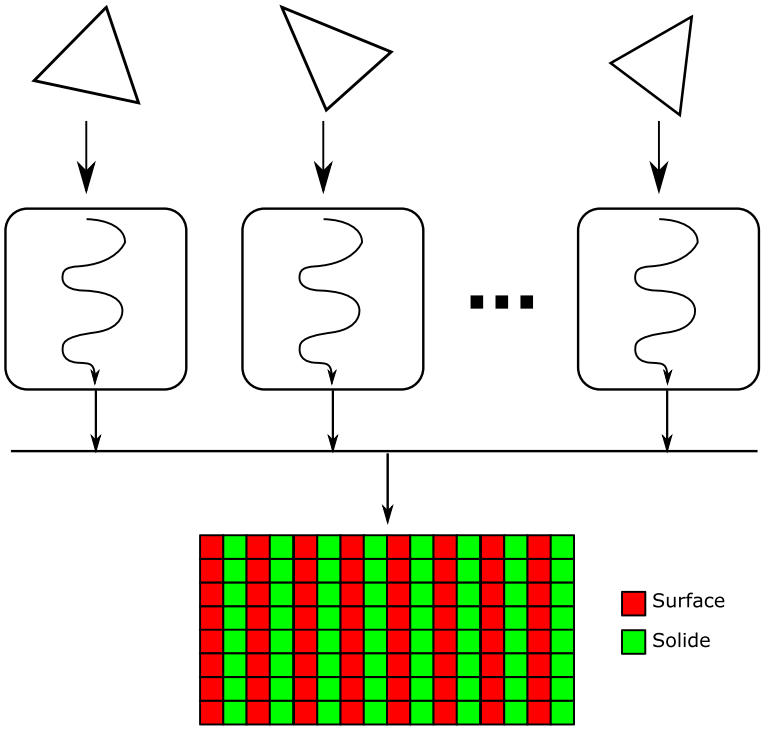

FIGURE 4.8 – Implémentation de la voxélisation.

La voxélisation résultante est encodée sur une texture 2D d'entiers 32 bits de format `RG` et de dimension $N \times N$ (N étant la dimension de la voxélisation), la composante `R` contenant l'état de la voxélisation de surface tandis que la composante `G` l'état de la voxélisation solide. Comme le montre la figure 4.9, chaque bit d'un texel contient l'état de voxélisation de l'espace de l'objet.

Présentement, notre système est limité à une dimension de $32 \times 32 \times 32$, les processeurs graphiques actuels ne supportant pas les textures d'entiers 64 bits. Par contre, il est tout de même possible de créer plusieurs textures 2D permettant d'obtenir une voxélisation au niveau souhaité.

Comme nous l'avons mentionné précédemment, chaque triangle est traité indépendemment par un coeur du processeur graphique. Étant donnée cette exécution massivement pa-

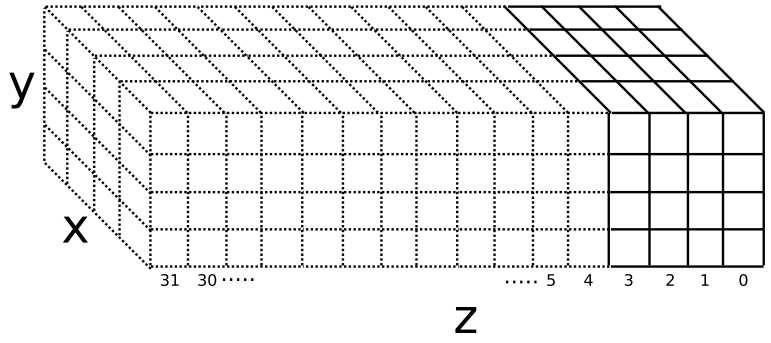

FIGURE 4.9 – Format de la texture de voxélisation.

rallèle, l'écriture à la texture de voxélisation doit être sérialisée pour s'assurer d'éviter les conditions de course (*race conditions*) à l'aide d'opérations d'écriture atomique.

4.3 Construction de cartes d'ombrage de semi-transparence

Pour simuler la diffusion à l'intérieur de l'objet semi-transparent, nous devons tout d'abord calculer l'énergie incidente sur la frontière de l'objet. Comme le montre la figure 4.10, cette étape utilise la voxélisation de l'objet pour calculer l'énergie en entrée sur chacune des faces visibles de l'objet.

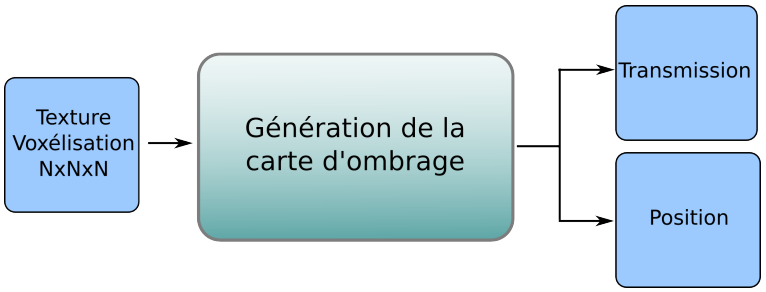

FIGURE 4.10 – Pipeline de création de la carte d'ombrage semi-transparente.

Pour ce faire, nous utilisons une variante de la méthode proposée par Dachsbacher et Stamminger [DS05] se basant sur le calcul de l'irradiance transmise à la frontière d'un objet à l'aide d'un procédé similaire à la carte d'ombrage. Pour chaque lumière de la scène, nous effectuons un rendu de la voxélisation du point de vue de la lumière en sauvegardant l'irradiance transmise et la position du voxel le plus rapproché sur deux textures cibles différentes (voir la figure 4.11).

Plutôt que de sauvegarder la position dans le repère monde, nous nous servons

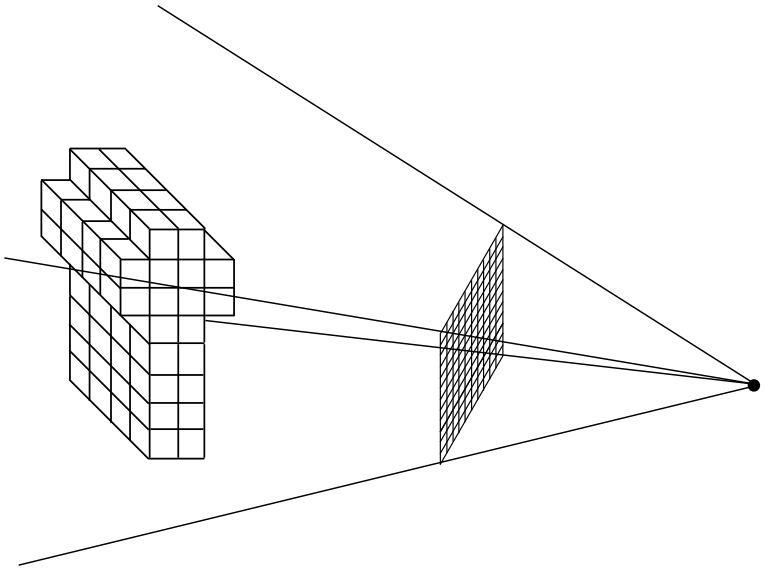

FIGURE 4.11 – Initialisation de l'irradiance incidente par carte d'ombrage pour chaque point de lumière.

plutôt d'un indice identifiant le voxel de destination et la face sur laquelle la lumière est transmise. L'indice de la face de destination est pour sa part encodé à partir de son référentiel (voir la figure 4.12 et le tableau 4.1).

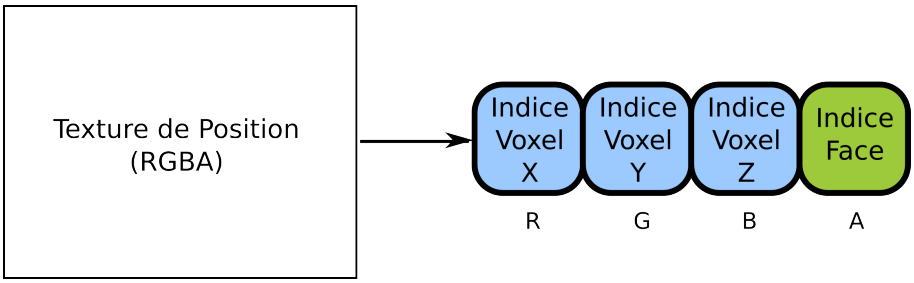

FIGURE 4.12 – Format de la texture de la carte d'ombrage semi-transparente.

Effectuer l'initialisation de l'irradiance en entrée sur le voxélisation plutôt que sur le maillage permet de simplifier de beaucoup l'étape de diffusion, l'énergie étant discrétisée dans le même domaine que celui de l'étape de diffusion.

Pour calculer le flux radiatif transmis à l'intérieur du matériau nous intégrons sur l'hémisphère d'entrée et de sortie au point $\mathbf{p}$:

$$\phi(\mathbf{p}) = A_p \int_{\Omega_o} \left(\int_{\Omega_i} f_t(\mathbf{p}, \vec{\omega}_i, \vec{\omega}_o) L_i(\vec{\omega}_i)(\vec{n} \cdot \vec{\omega}_i) d\vec{\omega}_i \right) (\vec{n} \cdot \vec{\omega}_o) d\vec{\omega}_o. \tag{4.5}$$

La valeur A_p représente l'aire d'un pixel de la carte d'ombrage et $f_t(\mathbf{p}, \vec{\omega}_i, \vec{\omega}_o)$ la

Face	Indice
z_-	0
z_+	1
x_-	2
x_+	3
y_-	4
y_+	5

TABLE 4.1 – Identifiant des faces d'un voxel.

BTDF (*Bidirectional Transmission Distribution Function*) décrivant la distribution de la lumière transmise au point **p**.

La voxélisation du maillage étant encodée sous forme d'un champ de bits à l'intérieur d'une texture 2D, nous utilisons le nuanceur de géométrie pour générer les voxels de l'objet comme le montre la figure 4.13.

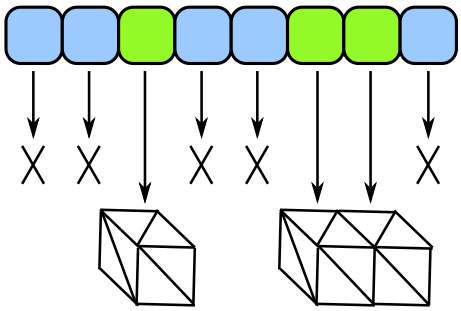

FIGURE 4.13 – Génération des voxels à partir de la carte de voxélisation.

L'état de la voxélisation étant encodé à l'intérieur d'une texture 2D de type tampon d'*OpenGL* (*texture buffer*), ceci permet de ré-utiliser celui-ci comme tampon de sommets en entrée du pipeline *OpenGL*. L'échantillonnage de la texture de voxélisation comme tampon de sommets permet un accès linéaire à celui-ci, offrant ainsi un usage maximum des caches d'entrée tout en évitant l'utilisation des unités de texture du processeur graphique. Le rendu se fait en utilisant la texture de voxélisation comme un ensemble de points de taille $N \times N$ (où N est la taille de la voxélisation) où chaque point représente une tranche du volume selon l'axe des z. Chaque bit actif est converti en maillage triangulaire par le nuanceur de géométrie en générant 12 triangles distincts (6 faces $\times$ 2 triangles par face) par voxel.

Pour des voxélisations compactes ayant plusieurs voxels actifs par tranche de profondeur, la quantité maximale de sommets pouvant être générés par le nuanceur géométrique

peut être rapidement atteinte. Pour contourner cette limitation, nous divisons la tranche de profondeur en plusieurs sous-intervalles où chacun est traité par une invocation différente du nuanceur de géométrie. Pour ce faire, nous initialisons le qualificateur *OpenGL* `invocation` au nombre de sous-intervalles de la tranche. Pour les processeurs graphiques ne supportant pas ce qualificateur (< *OpenGL* 4.0), il est possible d'utiliser les techniques d'instanciation géométrique pour obtenir un comportement similaire.

4.4 Injection du flux radiatif sur les faces de la voxélisation

L'étape précédente a permis d'échantillonner l'énergie transmise sur la surface représentée par la voxélisation du maillage. L'étape d'injection pour sa part, consiste à prendre l'ensemble de ces échantillons et de les distribuer sur les faces de la voxélisation. L'énergie accumulée sur ces faces sera par la suite utilisée pour calculer la diffusion à l'intérieur de l'objet.

La figure 4.14 montre les dépendances de cette étape et les textures résultant de l'opération d'injection. Le terme *RSM* (*reflective shadow map*) est utilisé pour garder la nomenclature introduite par Dachsbacher et Stamminger [DS05].

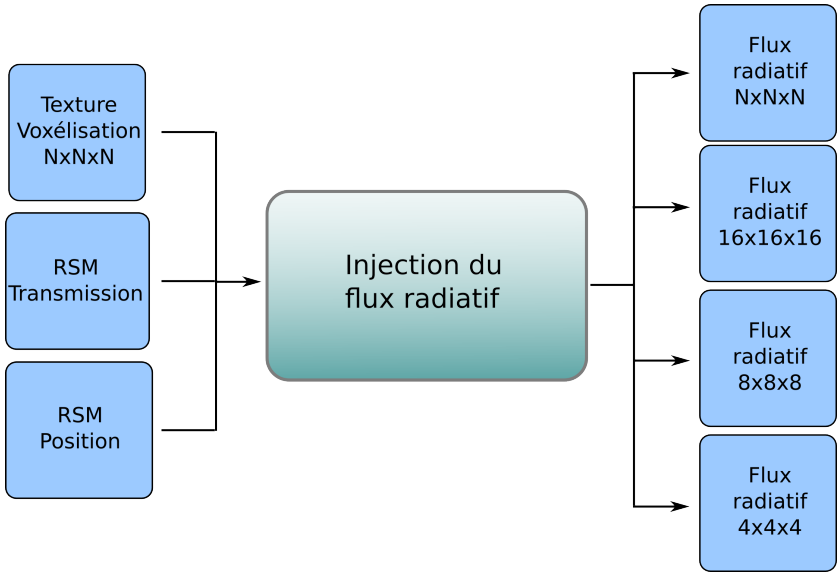

FIGURE 4.14 – Pipeline d'injection du flux radiatif.

Chaque pixel de la texture *RSM Transmission* contient la valeur du flux radiatif transmis à la surface de l'objet. La texture *RSM Position*, pour sa part, contient la position dans le repère local de l'objet et l'indice de la face sur laquelle l'énergie est

transmise. Pour accumuler le flux radiatif de chaque échantillon, nous traversons un à un chacun des texels des textures *RSM Transmission* et *RSM Position* en utilisant un rendu de primitive de type point. L'énergie de chacun des points est par la suite redirigée selon sa position et son indice de face vers le bon emplacement dans la texture de sortie ayant une structure décrite par la figure 4.15. Pour éviter d'utiliser une trop grande part de la mémoire graphique, nous utilisons un format RGB point flottant de 16 bits, qui est plus que suffisant pour contenir l'accumulation du flux radiatif.

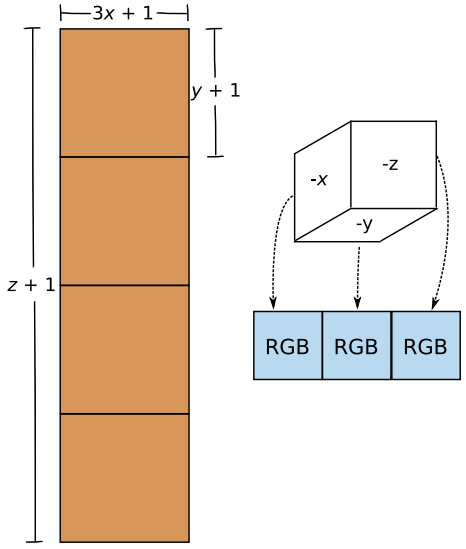

FIGURE 4.15 – Format de la texture du flux radiatif en entrée.

La position à l'intérieur de la texture de sortie est déterminée à partir du nuanceur de sommet. Il est donc important de s'assurer de générer une position valide dans le repère de coupure (*clip space*) car celle-ci sera utilisée par le *rasterizer* pour générer cette position. L'algorithme 1 montre le calcul nécessaire pour passer du repère local au repère de coupure.

Une fois la *rasterization* complétée, le flux radiatif est accumulé en utilisant la fonction d'addition du mélangeur de sortie. Comme le montre la figure 4.14, cette étape doit être répétée pour chacun des sous-niveaux de résolution de diffusion souhaité.

4.5 Propagation de la lumière à l'intérieur de l'objet

Cette section s'attarde au processus de diffusion de la lumière à l'intérieur du maillage voxélisé et constitue le coeur de l'algorithme de diffusion. La diffusion est calculée en

Algorithme 1: Calcul de la position dans le repère de coupure pour la face d'indice 0

```
// Obtention de la position dans le repère local
position := texture(RSMPosition);
// Obtention de l'indice du voxel
voxelPosition := position × voxelDim/tailleEspaceVoxel;
voxelIndex := ⌊voxelPosition⌋ ;
csPosX := (voxelIndex.x × 3 + 2) × 2/((voxelDim×3+1));
csPosY := (voxelIndex.z × (voxelDim+1) + voxelIndex.y) × 2/(voxelDim+1)^2;
```

utilisant une méthode des différences finies (MDF) appliquée aux équations de diffusion présentées au chapitre 2.

Comme le montre la figure 4.16, le procédé de diffusion est calculé hiérarchiquement sur un ensemble de domaines multi-résolution permettant ainsi de réduire le nombre d'itérations nécessaire à la convergence de la MDF.

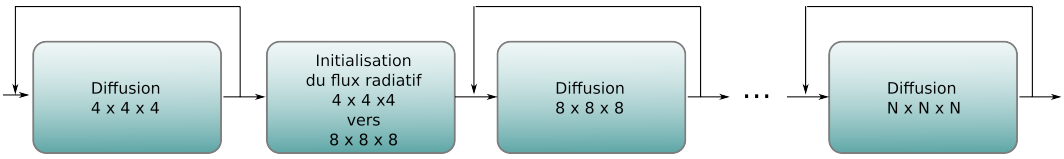

FIGURE 4.16 – Pipeline de diffusion.

Pour chaque niveau de résolution, le noyau de la MDF est exécuté n fois, dépendemment de la résolution du domaine et du niveau de précision souhaité. Pour accélérer la convergence, les valeurs initiales de chacun des niveaux sont initialisées à partir des résultats de diffusion à basses résolutions. Pour des domaines simples (inférieurs à 4×4×4), la méthode multi-résolution est désactivée et seul le noyau de diffusion est lancé.

Actuellement notre méthode permet de supporter un niveau de résolution maximal de 32×32×32 avec une exécution hiérarchique suivant les dimensions en puissance de deux. Par exemple, si notre domaine est de 22×22×22, nous procédons en exécutant d'abord le noyau sur un domaine de 4×4×4, suivi d'une exécution sur un domaine de 8×8×8 et de 16×16×16 en terminant par la résolution cible de 22×22×22. Le tableau 4.2 résume les dépendances d'exécution selon la résolution cible souhaitée.

La méthode multi-résolution nous permet de converger plus rapidement par rapport à l'utilisation d'une grille unique à haute résolution. Chaque voxel occupant un volume

Sous-résolution	Résolution cible			
	< 4	< 8	< 16	< 32
4×4×4		x	x	x
8×8×8			x	x
16×6×16				x

TABLE 4.2 – Dépendances des sous-résolutions.

plus important, le transport de la lumière se fait plus rapidement à travers l'objet. Les résolutions plus élevés nous permettent de rafiner la solution en tenant compte des détails supplémentaires apportés par une voxélisation plus fine. Le désavantage de cette méthode est bien sur le coût supplémentaire de stockage nécessité par l'ensemble des résolutions.

Notre système utilise présentement un volume constant dépendant du maillage en entré qu'il discrétise en un nombre égal de voxels selon les trois axes. Il pourrait être tout à fait réalisable de voxéliser davantage selon un axe en particulier dépendamment de la forme de la boîte englobante du maillage. Cette voxélisation adaptative nous permettrait d'obtenir une simulation plus précise suivant la forme du maillage. Bien que la limite actuelle de $32 \times 32 \times 32$ serait toujours présente, aucune modification à l'algorithme de diffusion serait nécessaire étant donné que celui-ci utilise la distance entre les voxels dans son calcul de diffusion.

4.5.1 Discrétisation de l'équation de diffusion

Comme nous l'avons vu au chapitre 2, nous devons traiter deux variantes de l'équation de diffusion. La première est utilisée pour calculer la diffusion à l'intérieur de l'objet, la seconde pour calculer la diffusion aux frontières de l'objet.

Diffusion interne

L'équation de diffusion interne est définie par l'équation suivante :

$$\vec{\nabla} \cdot (\kappa_d(\mathbf{x})\vec{\nabla}\phi(\mathbf{x})) = \sigma_a(\mathbf{x})\phi(\mathbf{x}).$$

Pour pouvoir résoudre cette équation avec la MDF, nous devons discrétiser celle-ci par rapport au flux radiatif $\phi(\mathbf{x})$ au centre du voxel. Pour ce faire, nous effectuons la

substitution suivante :

$$\vec{\nabla} \cdot (\kappa_d(\mathbf{x})\vec{\nabla}\phi(\mathbf{x})) = \frac{\kappa_d(\mathbf{x_n})\phi(\mathbf{x_n}) - \kappa_d(\mathbf{x})\phi(\mathbf{x})}{d_n^2}.$$

$\phi(\mathbf{x_n})$ représente le flux radiatif au voxel voisin et d_n la distance entre deux voxels adjacents. En substituant cette relation dans l'équation de diffusion, nous obtenons :

$$\frac{\kappa_d(\mathbf{x_n})\phi(\mathbf{x_n}) - \kappa_d(\mathbf{x})\phi(\mathbf{x})}{d_n^2} = \sigma_a(\mathbf{x})\phi(\mathbf{x}).$$

Isoler la valeur du flux radiatif $\phi(\mathbf{x})$ permet d'obtenir l'équation de diffusion interne discrétisée qui sera exécutée itérativement par notre noyau de diffusion :

$$\begin{array}{ll} \frac{\kappa_d(\mathbf{x_n})\phi(\mathbf{x_n})}{d_n^2} & = \frac{\kappa_d(\mathbf{x})\phi(\mathbf{x})}{d_n^2} + \sigma_a(\mathbf{x})\phi(\mathbf{x}) \\ \frac{\kappa_d(\mathbf{x_n})\phi(\mathbf{x_n})}{d_n^2} & = \phi(\mathbf{x})\left(\frac{\kappa_d(\mathbf{x})+d_n^2\sigma_a(\mathbf{x})}{d_n^2}\right) \\ \phi(\mathbf{x}) & = \left(\frac{d_n^2}{\kappa_d(\mathbf{x})+d_n^2\sigma_a(\mathbf{x})}\right)\left(\frac{\kappa_d(\mathbf{x_n})\phi(\mathbf{x_n})}{d_n^2}\right) \\ \phi(\mathbf{x}) & = \frac{\kappa_d(\mathbf{x_n})\phi(\mathbf{x_n})}{\kappa_d(\mathbf{x})+d_n^2\sigma_a(\mathbf{x})}. \end{array} \tag{4.6}$$

Diffusion surfacique

Comme nous l'avons vu dans la section 4.2 décrivant la voxélisation, une face d'un voxel ne possédant pas de voisin est considérée comme faisant partie de la frontière de l'objet. Dans ce cas, nous devons utiliser l'équation de diffusion à la surface de l'objet. L'équation de diffusion surfacique doit tenir compte non seulement de la transmission de Fresnel mais aussi de l'énergie en entrée sur la face du voxel. L'équation décrivant la diffusion surfacique est tirée de Arbree et al. [AWB11] et définie comme suit :

$$\phi(\mathbf{x}) + 2\kappa_d(\mathbf{x})\frac{1+F_{dr}}{1-F_{dr}}\vec{\nabla}\phi(\mathbf{x}) = \frac{4}{1-F_{dr}}Q(\mathbf{x_s}). \tag{4.7}$$

F_{dr}, la réflectance diffuse de Fresnel, est définie comme étant l'énergie totale réfléchie à la frontière de l'objet :

$$F_{dr} = \int_\Omega F_r(\eta)(\vec{\omega}_i \cdot \vec{n})d\vec{\omega}_i.$$

Pour calculer F_{dr}, tout en évitant d'évaluer cette intégrale, nous utilisons l'approximation de F_{dr} suivante de Jensen et al. [JMLH01] :

$$F_{dr} = -\frac{1.5}{\eta^2} + \frac{0.7099}{\eta} + 0.6681 + 0.0636\eta.$$

La valeur $Q(\mathbf{x_s})$ pour sa part, représente le flux radiatif en entrée sur la face $\mathbf{x_s}$ obtenue lors de l'étape d'injection décrite précédemment.

Tout comme l'équation de diffusion interne, nous devons discrétiser l'équation 4.7 en isolant $\phi(\mathbf{x})$:

$$\begin{aligned}
\phi(\mathbf{x}) + 2\kappa_d(\mathbf{x})\tfrac{1+F_{dr}}{1-F_{dr}}\tfrac{\phi(\mathbf{x_s})-\phi(\mathbf{x})}{d_s} &= \tfrac{4}{1-F_{dr}}Q(\mathbf{x_s}) \\
\phi(\mathbf{x})\left(1 - 2\kappa_d(\mathbf{x})\tfrac{1+F_{dr}}{(1-F_{dr})d_s}\right) &= \tfrac{4}{1-F_{dr}}Q(\mathbf{x_s}) - 2\kappa_d(\mathbf{x})\tfrac{1+F_{dr}}{1-F_{dr}}\tfrac{\phi(\mathbf{x_s})}{d_s} \\
\phi(\mathbf{x})\left(\tfrac{(1-F_{dr})d_s-2\kappa_d(\mathbf{x})(1+F_{dr})}{(1-F_{dr})d_s}\right) &= \tfrac{4}{1-F_{dr}}Q(\mathbf{x_s}) - 2\kappa_d(\mathbf{x})\tfrac{1+F_{dr}}{1-F_{dr}}\tfrac{\phi(\mathbf{x_s})}{d_s} \\
\phi(\mathbf{x}) &= \left(\tfrac{(1-F_{dr})d_s}{(1-F_{dr})d_s-2\kappa_d(\mathbf{x})(1+F_{dr})}\right)\left(\tfrac{4}{1-F_{dr}}Q(\mathbf{x_s}) - 2\kappa_d(\mathbf{x})\tfrac{1+F_{dr}}{1-F_{dr}}\tfrac{\phi(\mathbf{x_s})}{d_s}\right) \\
\phi(\mathbf{x}) &= \tfrac{4Q(\mathbf{x_s})d_s-2\kappa_d(\mathbf{x})(1+F_{dr})\phi(\mathbf{x_s})}{(1-F_{dr})d_s-2\kappa_d(\mathbf{x})(1+F_{dr})}.
\end{aligned} \tag{4.8}$$

Pour l'équation de diffusion surfacique 4.8, $\phi(\mathbf{x_s})$ représente le flux radiatif sur une face intérieure de l'objet. d_s est la distance entre le centre d'un voxel et l'une de ses faces.

4.5.2 Implémentation

Le processus de diffusion est exécuté entièrement sur *GPU*. Chaque voxel est assigné à un fil d'exécution s'occupant de calculer la valeur du flux radiatif en utilisant les équations 4.6 et 4.8 comme le montre la figure 4.17.

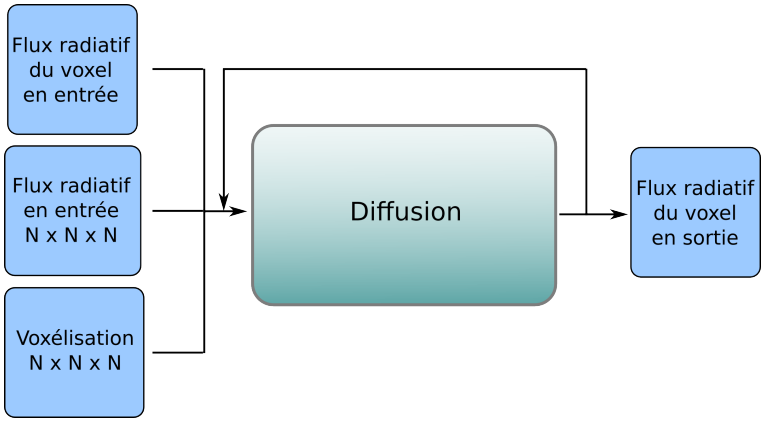

FIGURE 4.17 – Noyau de diffusion.

La méthode des différences finies étant un procédé itératif, chaque noyau est exécuté itérativement en basculant les textures d'entrée et de sortie après chaque itération (la texture de sortie d'une itération devenant la texture en entrée de l'itération suivante).

Comme le montre la figure 4.18, un voxel donné peut se présenter sous plusieurs configurations selon les voxels présents au voisinage de ce dernier.

Pour chacune des six faces du voxel, nous vérifions d'abord si un voxel avoisinant est actif. Si un voxel est actif, nous utilisons l'équation 4.6 en échantillonnant les valeurs du

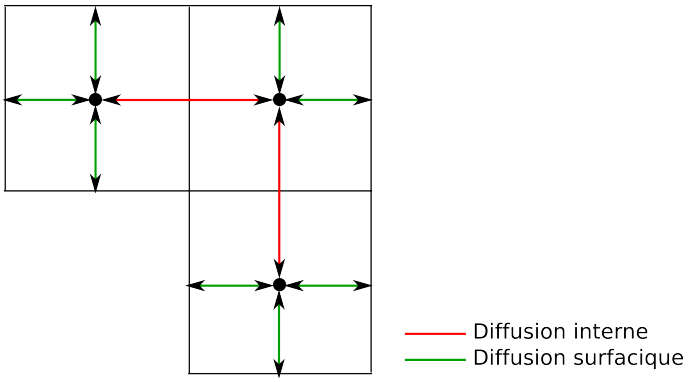

FIGURE 4.18 – Configuration de la diffusion.

Résolution	Nombre d'itérations	
	Simple	Multi-résolution
8×8×8	34	20
16×16×16	72	38
32×32×32	142	66

TABLE 4.3 – Itérations nécessaires en multi-résolution.

flux radiatif et les propriétés volumiques (σ_a et σ_s) du matériau. Dans le cas échéant, l'équation 4.8 est utilisée pour mettre à jour le flux radiatif.

Évidemment, plus la résolution de la voxélisation est grande et plus il est nécessaire d'effectuer d'itérations pour permettre au flux radiatif de se distribuer dans l'ensemble du volume. L'utilisation d'un procédé multi-résolution permet de réduire le nombre d'itérations nécessaire sur des volumes à haut niveau de voxélisation ($> 16\times16\times16$). Le tableau 4.3 montre un comparatif du nombre d'itérations nécessaire avec et sans le processus de multi-résolution.

4.6 Rendu de la lumière diffusée

L'étape précédente a permis de simuler la diffusion de la lumière à l'intérieur de l'objet. L'étape de rendu pour sa part, utilise les résultats de simulation pour calculer l'apparence finale de l'objet (voir la figure 4.19).

Pour obtenir la radiance en sortie de l'objet nous devons calculer la fraction du flux radiatif interne transmise à sa frontière. Cette étape s'apparente quelque peu au calcul du flux radiatif en entrée mais en suivant le chemin inverse (le calcul se fait de l'intérieur de l'objet vers l'extérieur).

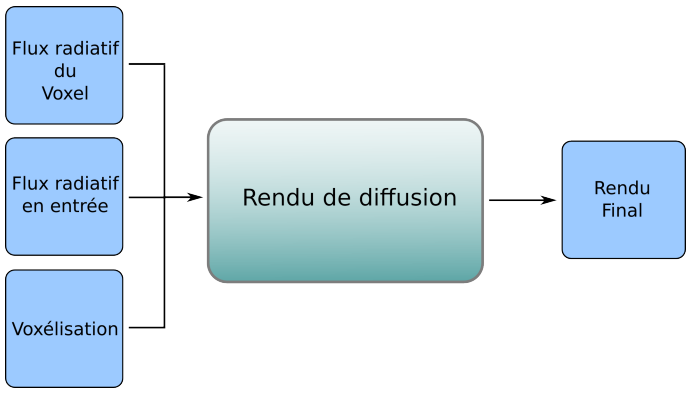

FIGURE 4.19 – Étapes de rendu de la diffusion.

Nous devons donc tenir compte de trois composantes : le flux radiatif à l'intérieur de l'objet, le flux radiatif entrant à la frontière et le facteur de transmission de Fresnel. Ces trois composantes se retrouvent dans l'équation suivante introduite par Arbree et al. [AWB11] :

$$L(\mathbf{x},\vec{\omega}) = \frac{F_t(\eta,\vec{\omega})}{4\pi}\left[\left(1+\frac{1-F_{dr}(\eta)}{1+F_{dr}(\eta)}\right)\phi_v(\mathbf{x}) - \frac{4(1-F_{dr}(\eta))}{F_{dt}(\eta)(1+F_{dr}(\eta))}\phi_i(\mathbf{x})\right]. \quad (4.9)$$

Dans cette équation, $\phi_v(\mathbf{x})$ représente le flux radiatif résultant de la simulation et $\phi_i(\mathbf{x})$ le flux radiatif en entrée à la frontière de l'objet.

4.6.1 Implémentation

La première étape pour effectuer le rendu de l'objet semi-transparent est de calculer les positions de celui-ci dans l'espace image. Par la suite, chaque position est utilisée pour échantillonner le flux radiatif contenu dans la voxélisation de l'objet (voir la figure 4.20).

Pour réduire l'aliassage dû à l'échantillonnage du domaine discrétisé, nous utilisons une interpolation tri-linéaire (en x, y et z) sur les valeurs du flux radiatif contenues dans chacun des voxels.

4.6.2 Intégration dans un système de rendu en différé

Pour un obtenir une image de synthèse convaincante, nous avons construit un système de rendu en différé intégrant la composante de diffusion décrite dans ce mémoire. La figure 4.21 montre les étapes composant ce système de rendu, l'étape de diffusion étant soulignée en jaune.

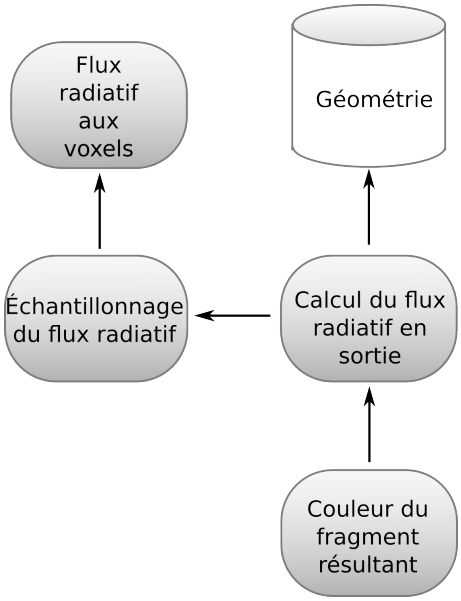

FIGURE 4.20 – Rendu de la simulation.

Un système de rendu en différé se distingue par un traitement minimal de la géométrie où l'ensemble des calculs sont réalisés dans l'espace image. Pour ajouter la composante de diffusion à un tel système, nous calculons pour chaque pixel de l'image finale, la contribution de la diffusion selon l'équation 4.9. L'image est donc composée en additionnant les résultats des étapes de lumière d'environnement, de sources de lumière ponctuelles et de diffusion à l'image finale.

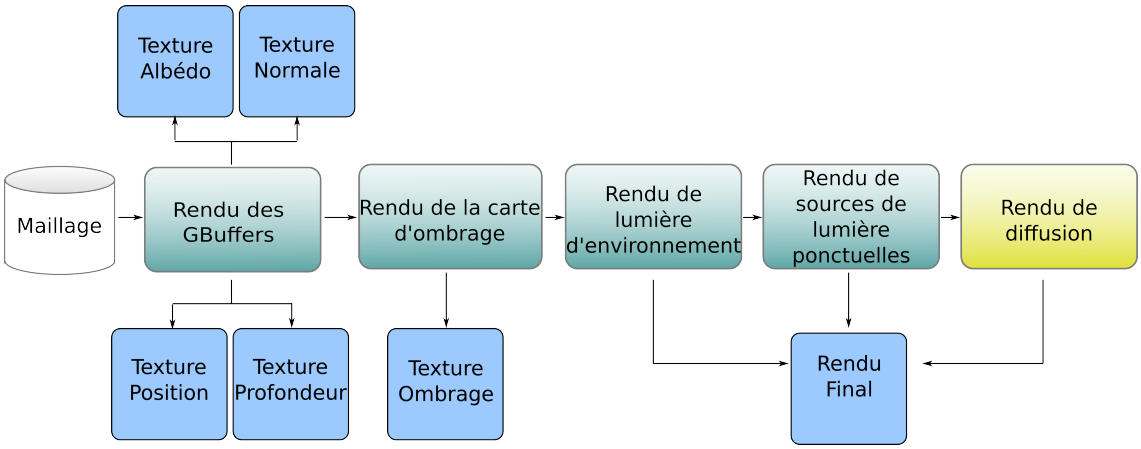

FIGURE 4.21 – Système de rendu.

Chapitre 5

Résultats

Ce chapitre sera consacré à l'analyse des performances de notre algorithme ainsi qu'à la présentation des résultats. L'ensemble des tests et des rendus ont été effectués sur un processeur graphique NVIDIA GTX 470 possédant 1.2 GB de mémoire vidéo.

5.1 Analyse de performance

Dans cette section, nous étudierons l'impact de chaque étape de notre pipeline de rendu sur les performances. Pour ce faire, nous avons utilisé un profileur *GPU* (*NVIDIA Parallel NSight*) pour extraire l'ensemble des données. La figure 5.1 montre la trace d'exécution de notre pipeline graphique à une résolution de 28×28×28 voxels sur le modèle *Lucy* comptant plus de deux millions de triangles. Il est intéressant de s'attarder particulièrement aux appels des noyaux *OpenCL* et aux nuanceurs *OpenGL*. Ces derniers sont présents sous la forme des *DrawCalls* sur cette figure. Un *DrawCall OpenGL* représente généralement une commande utilisée pour envoyer de la géométrie au *GPU* et par le fait même pour lancer une passe de rendu. Nous avons mis en évidence sur la figure 5.1 chaque étape de notre pipeline, en indiquant le numéro de l'étape près de sa partie correspondante dans la trace d'exécution, que ce soit un noyau *OpenCL* ou une passe de rendu *OpenGL* (les *DrawCall* mentionnés précédemment) :

1. **Voxélisation [*OpenCL*]** : Cette étape comprend l'étape de voxélisation et l'initialisation des sous-niveaux de voxélisation. Comme le montre la trace d'exécution de la figure 5.1, la voxélisation, identifiée par le nom *voxelize*, utilise la majeure

partie du temps d'exécution de notre pipeline de rendu. Nous analyserons en détail les performances de cette étape à la section suivante.

2. **Construction des cartes d'ombrage pour la semi-transparence [*OpenGL*]** : Création des cartes d'ombrage pour la semi-transparence à partir des sources de lumière de la scène. Pour simplifier la description, nous utiliserons le qualificatif PLV (points de lumière virtuels) pour décrire ces cartes. Cette étape est exécutée par le contexte *OpenGL* et se retrouve au numéro 2 sur la trace d'exécution de la figure 5.1.

3. **Injection [*OpenGL*]** : Traitement de chacun des pixels de la PLV et addition des valeurs du flux radiatif sur les faces des voxels. Comme le montre le numéro 3 sur la figure 5.1, cette étape comporte quatre parties, soient l'injection sur la voxélisation cible et l'injection sur les sous-voxélisations 4×4×4, 8×8×8 et 16×16×16. L'intervalle entre chaque lancement est identifiable par trois segments blancs dans le rectangle vert.

4. **Propagation [*OpenCL*]** : Propagation du flux radiatif à l'intérieur du volume. Notre système multi-résolution est clairement visible sur la trace (sous le nom *propagate*), où nous distinguons dans l'espace en brun, les sous-étapes de propagation de résolutions 4×4×4, 8×8×8, 16×16×16 ainsi que la résolution cible de 28×28×28. Il est intéressant de souligner la présence du noyau d'initialisation (nommé *initFluenceFromLowerLevel*) par un court segment de couleur bleue entre chacune des sous-étapes de propagation.

5. **Rendu [*OpenGL*]** : Cette dernière étape prend le résultat de propagation et calcule la valeur de radiance en sortie. Comme le montre le numéro 5 sur la trace, nous revenons au contexte *OpenGL* pour effectuer le rendu final de l'objet vers le tampon de destination.

Pour analyser l'impact de chacune de ces étapes, nous utiliserons leur profilage sur plusieurs maillages de tailles différentes et à des voxélisations variées. Les tableaux 5.1, 5.2 et 5.3 montrent ces résultats où chacune des colonnes est définie comme suit :

- **Maillage** : Le maillage triangulaire utilisé, le nombre entre parenthèses signifiant le nombre total de triangles constituant celui-ci.

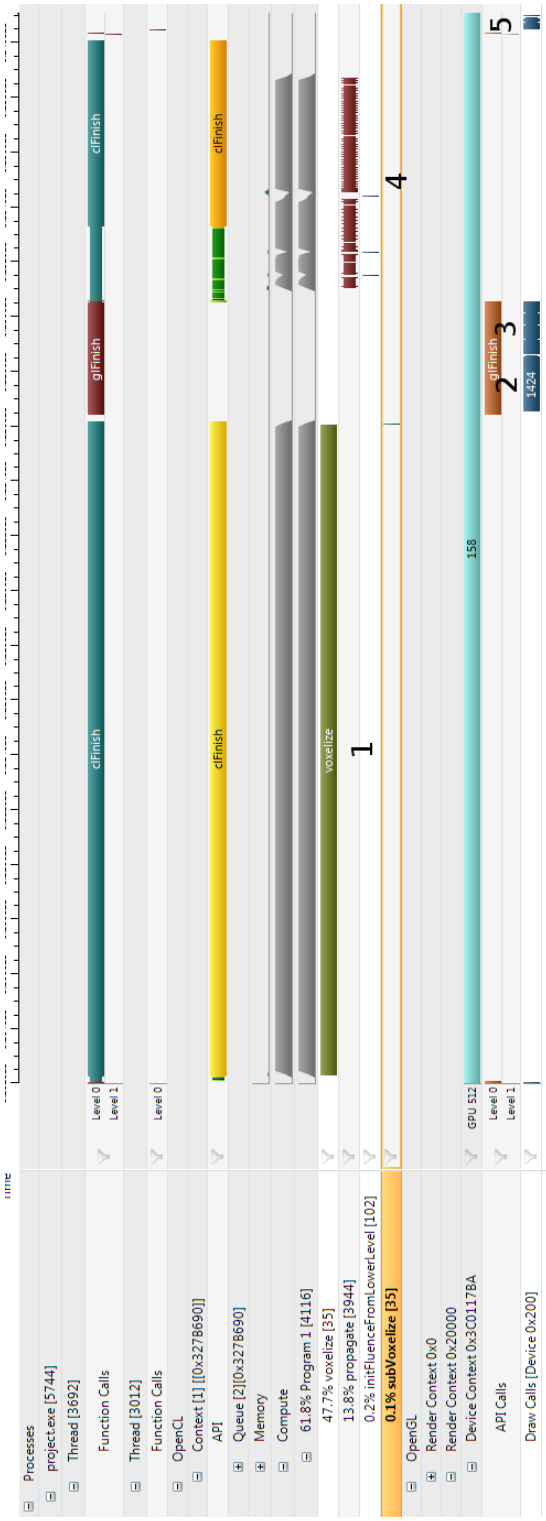

FIGURE 5.1 – Trace d'exécution sur *GPU*.

Maillage	Pipeline de diffusion (μs)						TPS
	Voxélisation	PLV	Injection	Propagation	Rendu	Total	
Suzanne (15k)	266	83	1095	364	227	2035	491
Armadillo (346k)	4906	89	1236	357	202	6790	147
Statue (800k)	9700	68	1214	287	178	11447	87
Lucy (2027k)	26401	62	1244	287	185	28179	35

TABLE 5.1 – Performances pour une voxélisation de 4×4×4. La propagation requiert 8 itérations.

Maillage	Pipeline de diffusion (μs)						TPS
	Voxélisation	PLV	Injection	Propagation	Rendu	Total	
Suzanne (15k)	257	424	1640	2432	269	5022	199
Armadillo (346k)	5046	568	1780	2432	255	1081	99
Statue (800k)	8789	454	1693	2315	218	13469	74
Lucy (2027k)	24822	286	1770	2274	220	29372	34

TABLE 5.2 – Performances pour une voxélisation de 12×12×12. La propagation requiert 52 itérations.

- **Pipeline de diffusion** : Temps d'exécution de chacune des étapes en micro-secondes.
- **TPS** : Trames par seconde.

5.1.1 Voxélisation

Comme le montre le tableau 5.4, pour des maillages complexes, l'étape de voxélisation peut s'avérer très coûteuse en temps de calcul. Ceci est compréhensible car cette étape agit comme un *rasterizer* volumique en traitant chacun des triangles composant le maillage. Même avec une voxélisation à haute résolution (28×28×28), cette étape occupe près de 70% du temps de calcul pour un maillage de plus de deux millions de triangles.

Maillage	Pipeline de diffusion (μs)						TPS
	Voxélisation	PLV	Injection	Propagation	Rendu	Total	
Suzanne (15k)	286	2148	2004	7958	337	12733	79
Armadillo (346k)	5208	2941	1932	8089	311	18481	54
Statue (800k)	8627	2198	1914	6644	288	19671	51
Lucy (2027k)	23705	2024	1932	6876	259	34796	29

TABLE 5.3 – Performances pour une voxélisation de 28×28×28. La propagation requiert 116 itérations.

Maillage	4×4×4	12×12×12	28×28×28
Suzanne (15k)	13%	5%	2%
Armadillo (346k)	72%	48%	28%
Statue (800k)	85%	65%	43%
Lucy (2027k)	94%	84%	68%

TABLE 5.4 – Pourcentage du temps de calcul de l'étape de voxélisation.

Il est intéressant de constater que le temps de calcul dédié à la voxélisation ne suit pas nécessairement la résolution de cette voxélisation. En fait, comme le montre la figure 5.2, le temps de calcul décroît avec les résolutions plus élevées.

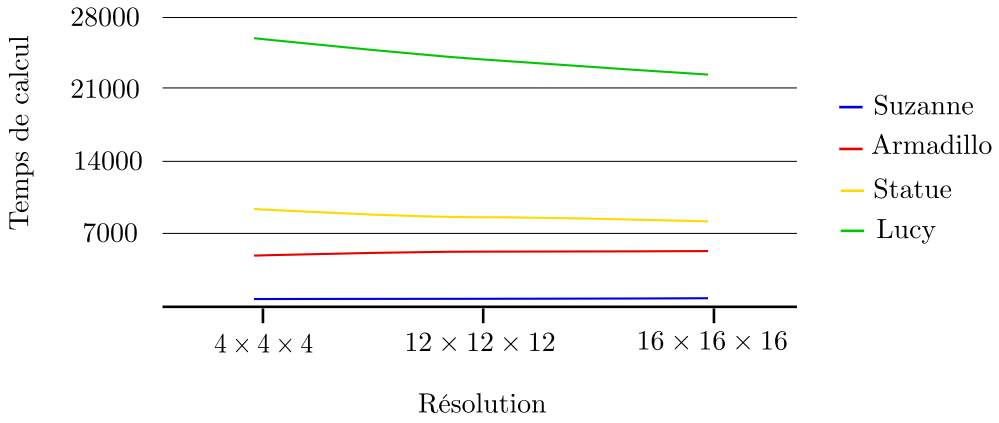

FIGURE 5.2 – Diminution du temps de voxélisation pour des résolutions supérieures.

Ceci s'explique principalement par les contraintes de synchronisation aux accès mémoire. Les états de voxélisation étant encodés dans chaque bit d'un mot et ceux-ci devant être accédés de manière atomique, il en résulte une plus grande latence. Une voxélisation à haute résolution permet de distribuer ces accès sur un ensemble plus grand d'espace mémoire, réduisant ainsi les délais de synchronisation.

Comme il a été mentionné au chapitre précédent, la voxélisation s'effectue en traitant indépendemment chacun des triangles. Nous obtenons donc des temps de voxélisation beaucoup plus importants sur des maillages complexes. Le tableau 5.4 montre bien la relation entre le temps de calcul et la complexité du maillage.

5.1.2 Génération des PLVs

Cette étape dépend entièrement du nombre de voxels actifs résultant de l'étape de voxélisation. Comme nous l'avons décrit au chapitre précédent, cette étape utilise le

Maillage	4×4×4	12×12×12	28×28×28
Suzanne (15k)	18%	48%	62.5%
Armadillo (346k)	5.3%	24%	43.8%
Statue (800k)	2.5%	17%	33.7%
Lucy (2027k)	1%	7.7%	19.8%

TABLE 5.5 – Pourcentage du temps de calcul de propagation par résolution de voxélisation.

nuanceur de géométrie pour instancier chaque voxel actif en effectuant un rendu de style carte d'ombrage. Il est donc normal d'obtenir des temps de calcul plus élevés avec une voxélisation à plus haute résolution. On le reconnaît par les nombres dans la colonne PLV, qui croissent entre les tableaux 5.1, 5.2 et 5.3. Même si le nombre de voxels actifs n'y est pas indiqué, il est directement relié à la résolution de la voxélisation.

5.1.3 Injection

L'étape d'injection consiste à prendre chacun des points de lumière virtuels et de les injectés sur les faces des voxels correspondants. Cette étape est donc dépendante de la résolution de la texture de PLVs ainsi que du nombre de sous-niveaux à initialiser. Nous constatons des temps de calcul similaires pour un même niveau de voxélisation. La différence entre les niveaux de voxélisation est due principalement à la taille de la voxélisation cible. Par exemple, pour une voxélisation de 12×12×12, nous devons initialiser non seulement la voxélisation cible mais aussi les sous-voxélisations de 4×4×4 et de 8×8×8.

5.1.4 Propagation

L'étape de propagation est la seconde étape la plus coûteuse du pipeline. Pour un plus grand nombre de voxels, les transferts de lumière demanderont plus d'itérations de simulation afin d'atteindre l'autre extrémité de la voxélisation, ainsi que pour se stabiliser. On le reconnaît par exemple en observant que pour 4×4×4, 12×12×12 et 28×28×28 voxels, le nombre d'itérations requises passent de 8 à 52 à 116 pour chacun des tableaux 5.1, 5.2 et 5.3. Comme le montre le tableau 5.5, pour des maillages simples et une voxélisation élevée, le temps de calcul occupé par l'étape de propagation peut représenter plus de la moitié du temps total d'exécution.

Il est intéressant de comparer les temps de calcul de propagation avec et sans les étapes de multi-résolution pour obtenir une diffusion équivalente. La figure 5.3 montre une comparaison des temps de calcul avec et sans multi-résolution.

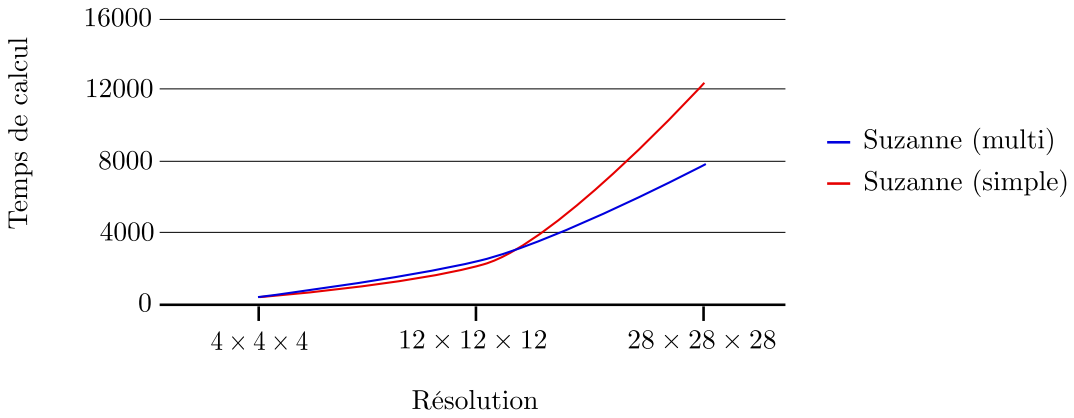

FIGURE 5.3 – Temps de propagation avec et sans multi-résolution.

Nous remarquons qu'à basse résolution les gains obtenus par la méthode multi-résolution sont éliminés par les coûts de mise en place des nuanceurs et d'initialisation des textures du flux radiatif. À haute résolution par contre, nous obtenons des gains en temps de calcul de plus de 25%. Nous pouvons aussi faire l'hypothèse que cet écart serait d'autant plus important avec des résolutions de voxélisation plus grandes, les temps de mise en place devenant négligeables par rapport au temps de calcul total.

5.1.5 Rendu

L'étape de rendu est intégrée à notre système de rendu en différé. Cette étape dépend donc en majeure partie du nombre de pixels occupés par l'objet semi-transparent sur la texture cible. Les tableaux 5.1 à 5.3 montrent tout de même des temps de calcul dépendants de la résolution de la voxélisation. Ceci s'explique principalement par l'utilisation des caches des échantillonneurs de textures où une résolution de voxélisation moins grande utilisera plus souvent le contenu du même voxel.

5.2 Résultats

Cette section sera consacrée entièrement à la présentation des rendus générés par notre méthode. Nous présenterons les résultats de simulations de matériaux homogènes et

hétérogènes en mettant l'emphase sur l'effet de la résolution de voxélisation. Nous terminerons cette section par une analyse des rendus avec déformation.

5.2.1 Matériaux homogènes

Les matériaux homogènes se caractérisent par une distribution de valeurs uniques pour σ_a et σ_s à l'intérieur du volume. Ce sont les matériaux les plus simples à simuler car ils ne requièrent aucun encodage ni paramétrisation particulière du volume. Simuler une pierre précieuse aquamarine par exemple, peut être réalisé en spécifiant une valeur de σ_a absorbant l'ensemble des longueurs d'onde à l'exception de celles du bleu. Généralement pour les rendus temps réel, ce spectre est encodé à l'aide d'un 3-tuple définissant les trois couleurs primaires rouge, vert et bleu. Sous cet encodage, le σ_a de l'aquamarine peut être défini comme suit : $(RGB) \leftarrow (0, 1, 1)$ spécifiant une absorption totale pour le rouge et une absorption égale pour les longueurs d'onde dans le bleu et le vert. Une comparaison entre le modèle de réflectance de Blinn-Phong et notre technique est donnée à la figure 5.4 pour le rendu de ce type de matériau.

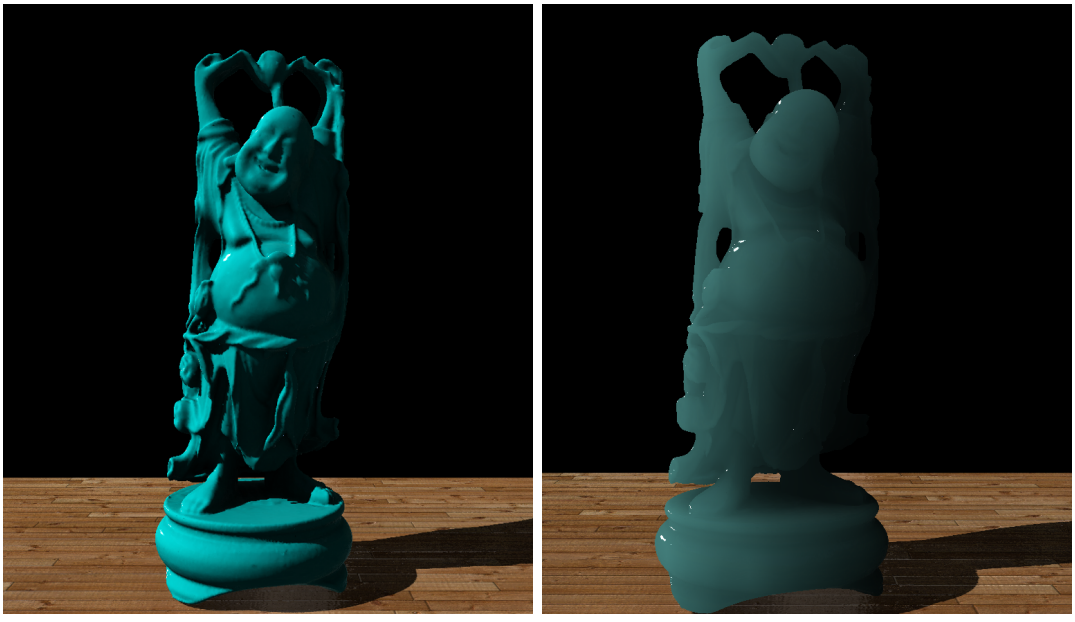

FIGURE 5.4 – Comparaison d'un rendu de matériau homogène de type aquamarine du modèle géométrique Buddha. Le modèle de réflectance de Blinn-Phong se trouve à gauche tandis que le résultat avec notre technique se trouve à droite.

Une variante simple du coefficient σ_a permet d'obtenir un matériau simulant le jade

en utilisant les composantes $(RGB) \leftarrow (0.1, 1.0, 0.1)$. Nous obtenons ainsi un coefficient σ_a laissant passer majoritairement les longueurs d'onde du vert. La figure 5.5 présente un rendu de cette variation. Il est important de mentionner que valeurs σ_a et σ_s ne sont pas des valeurs normalisées entre 0 et 1. Elles représentent plutôt la probabilité par unité de distance (m^{-1}) que la lumière soit aborbée (σ_a) ou diffusée (σ_s). Il pourrait être valide par exemple, de spécifier pour un matériau à faible densité une valeur de $\sigma_a = (50.0, 50.0, 50.0)$.

FIGURE 5.5 – Rendu de matériau homogène de type jade du modèle géométrique Suzanne.

Notre système de diffusion étant basé sur la discrétisation d'équations physiques connues, il peut être plus simple pour un artiste d'extraire les valeurs de σ_a et σ_s de la littérature scientifique et de les utiliser comme point de départ pour la définition d'un nouveau matériau. Les résultats étant obtenus en temps réel, il devient simple d'éditer ces valeurs et d'en observer leur impact directement sur le rendu du matériau.

Effet de la voxélisation sur les résultats

Comme il a été mentionné à la section 5.1 sur l'analyse de performance, la voxélisation peut s'avérer un procédé coûteux tant au niveau du temps de calcul de la voxélisation que du procédé de diffusion. Par contre, avoir une voxélisation plus fine permet d'obtenir une plus grande précision des résultats. Selon la configuration de la scène et la position des sources de lumière, l'impact d'une voxélisation à haute résolution peut être majeur sur le rendu final.

La figure 5.6 montre la différence entre un rendu à une résolution de voxélisation de 4×4×4 versus une résolution de 32×32×32.

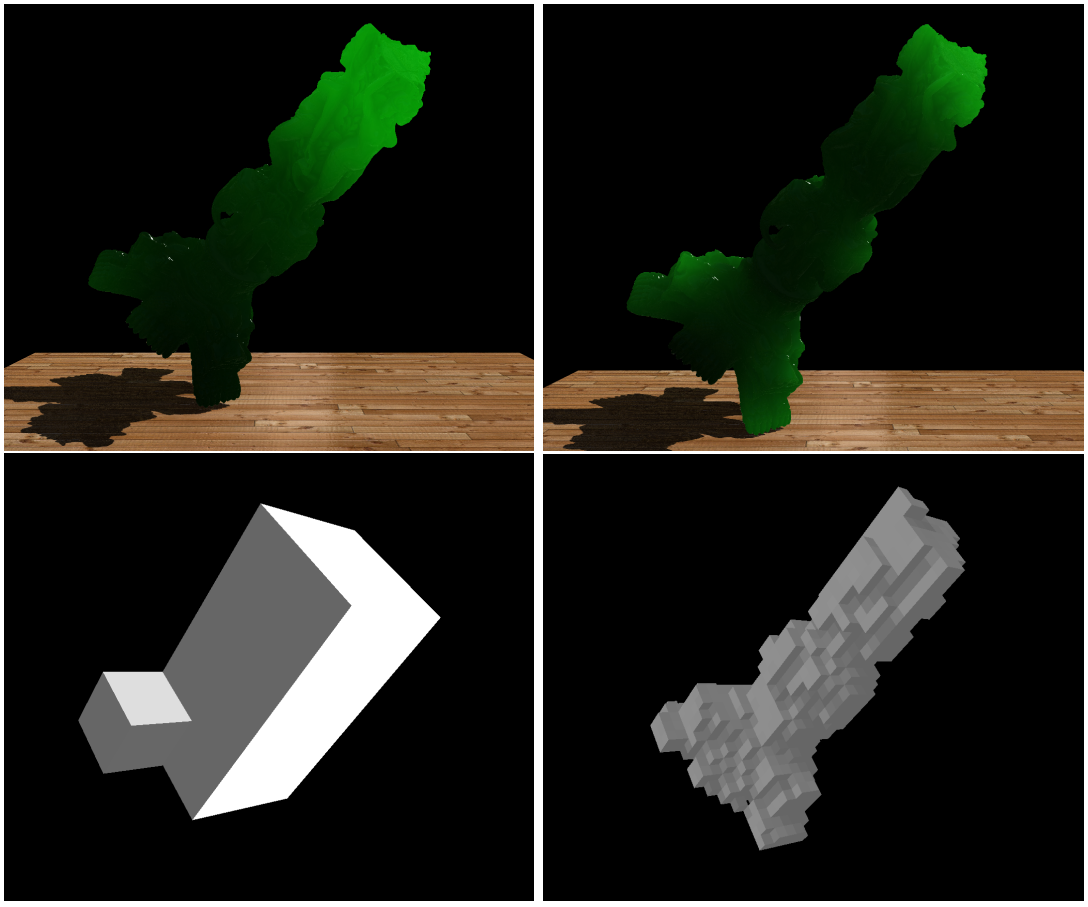

FIGURE 5.6 – Différences de rendu selon la résolution de la voxélisation pour un même nombre d'itérations. L'image de gauche représente une voxélisation de 4×4×4 et l'image de droite de 32×32×32. Leur voxélisation respective est montrée sur la rangée du bas. La source de lumière directionnelle provenant du coin supérieur droit.

Une voxélisation plus fine permet de bien traiter les détails géométriques au pied de la statue, ce qui est absent de la voxélisation 4×4×4. Le résultat donne une simulation plus réaliste et convaincante au détriment d'un coût de rendu supérieur (87 TPS vs. 51 TPS). Comme nous l'avons vu à la section précédente, ce coût de rendu supérieur provient du nombre plus élevé d'itérations devant être exécutées pour obtenir la convergence de la méthode de différences finies. La figure 5.7 montre les résultats de diffusion à différentes itérations de la méthode.

Le nombre d'itérations nécessaire peut varier selon la voxélisation du modèle géométrique

et ses paramètres de diffusion. Une voxélisation plus compacte ne demandera pas autant d'itérations qu'un objet couvrant un grand nombre de voxels. Pour vérifier la convergence de la méthode, la différence du flux radiatif entre deux itérations peut être calculée et comparée par rapport à un critère d'arrêt. Étant donné que nous avons un système en temps réel, il serait inefficace de transférer le tampon de flux radiatif de la mémoire vidéo à la mémoire système à chaque itération pour vérifier ce critère. Nous avons donc pré-calculé un nombre conservateur d'itérations assurant la convergence selon la résolution de la voxélisation (fourni au tableau 5.6). Ce pré-calcul se base sur la diffusion à l'intérieur d'un cube couvrant l'ensemble de la voxélisation avec le critère d'arrêt suivant : $\Delta\phi(x) < 0.05$.

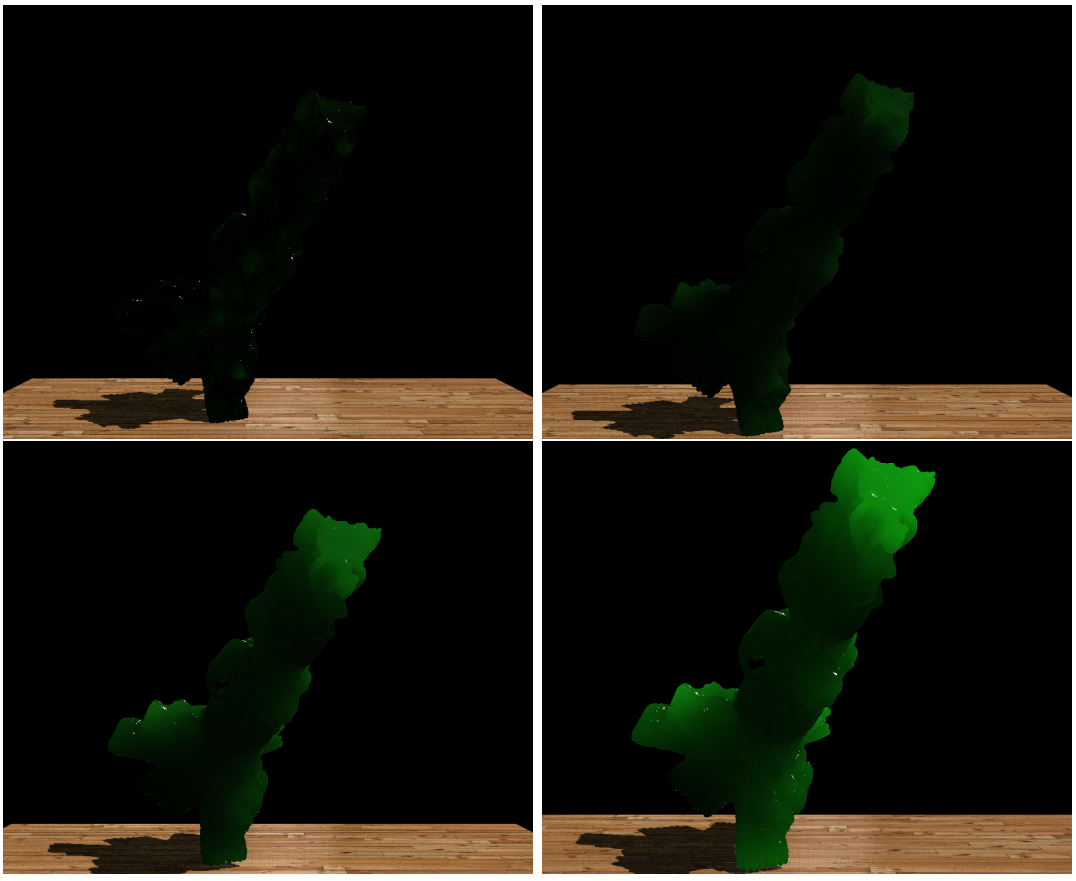

FIGURE 5.7 – Résultats de diffusion de la méthode de différences finies dans l'ordre habituel : après 1, 12, 36 et 72 itérations.

Résolution	Nombre d'itérations
8×8×8	16
16×16×16	38
32×32×32	66

TABLE 5.6 – Itérations nécessaires pour obtenir la convergence de la méthode des différences finies.

5.2.2 Matériaux hétérogènes

Dans la nature, il est rare, voire même impossible, de retrouver des matériaux complètement homogènes et bien que l'utilisation de ce type de matériau peut être suffisante pour certaines scènes, il est souvent insuffisant pour atteindre un niveau de réalisme élevé. Pour notre technique, un matériau hétérogène implique une distribution variant dans un volume des coefficients σ_a et σ_s. Une valeur différente de ces coefficients est associée à chaque voxel définissant le volume de l'objet.

La figure 5.8 montre un exemple de rendu de matériau hétérogène utilisant deux valeurs $\sigma_a = \{(1, 0, 1), (1, 1, 0)\}$ distribuées aléatoirement à l'intérieur du volume.

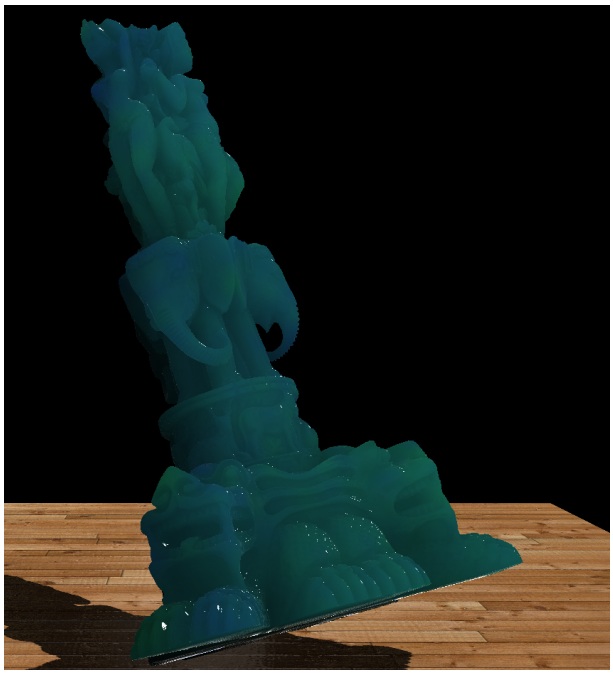

FIGURE 5.8 – Rendu de matériau hétérogène du modèle géométrique Statue.

Il est intéressant de constater la teinte de cyan résultant du processus de diffusion entre le vert et le bleu. On remarque aussi quelques zones à plus fortes densités où

plusieurs voxels avoisinants possèdent les mêmes valeurs de σ_a et σ_s. La figure 5.9 montre un second rendu de matériau homogène utilisant cette fois des valeurs de $\sigma_a = \{(1,0,1),(0,1,1)\}$ aléatoirement distribuées dans le volume.

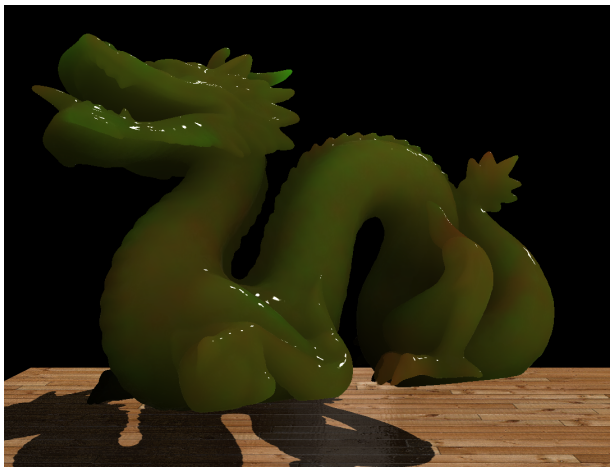

FIGURE 5.9 – Rendu de matériau hétérogène du modèle géométrique Dragon.

Effet de la voxélisation sur les résultats

Pour le rendu de matériaux hétérogènes, l'utilisation d'une haute résolution de voxélisation est essentielle pour bien traiter les subtilitées de la distribution des coefficients. Chacun des coefficients étant encodé à l'intérieur d'un voxel, augmenter la résolution permet d'accroître la fréquence à laquelle ces valeurs peuvent différer. La figure 5.10 donne un exemple probant des limitations qu'apporte une faible voxélisation par rapport à une voxélisation plus complexe pour le rendu de matériaux hétérogènes, à savoir une distribution complexe des paramètres de diffusion à l'intérieur du volume.

5.2.3 Déformations

Un des avantages de notre méthode est de permettre une déformation d'objets géométriques sans restriction tout en gardant une simulation valide. Comme il a été mentionné au chapitre 4, nous y parvenons en re-voxélisant le maillage pour chaque trame de rendu. La figure 5.11 montre un rendu après une transformation de mise à l'échelle selon l'axe des x et après une transformation de type *twist* selon l'axe des y.

Une difficulté supplémentaire s'ajoute lorsque l'on permet la déformation de maillage, soit de garder une paramétrisation valide de la distribution des coefficients σ_a et σ_s dans

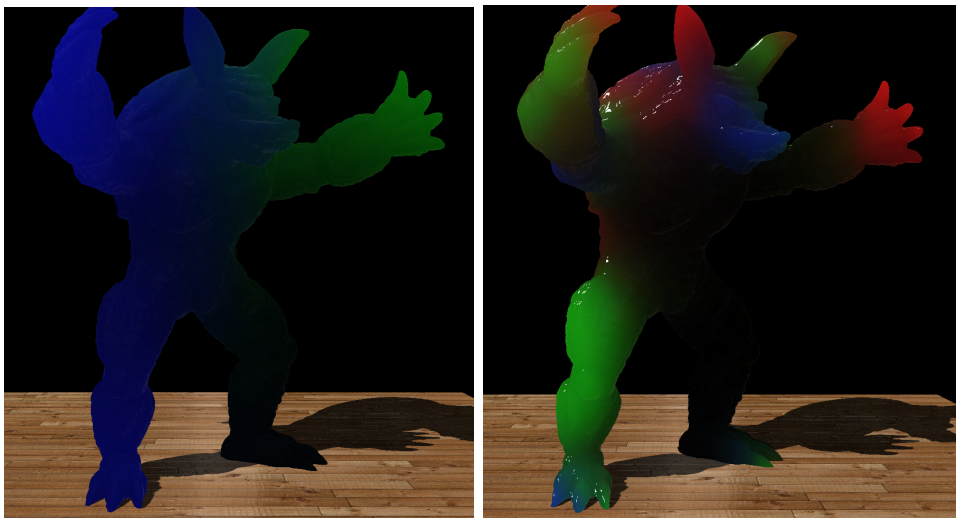

FIGURE 5.10 – Différences de rendu d'un matériau hétérogène selon la résolution de la voxélisation. L'image de gauche représente une voxélisation de 4×4×4 et l'image de droite de 32×32×32.

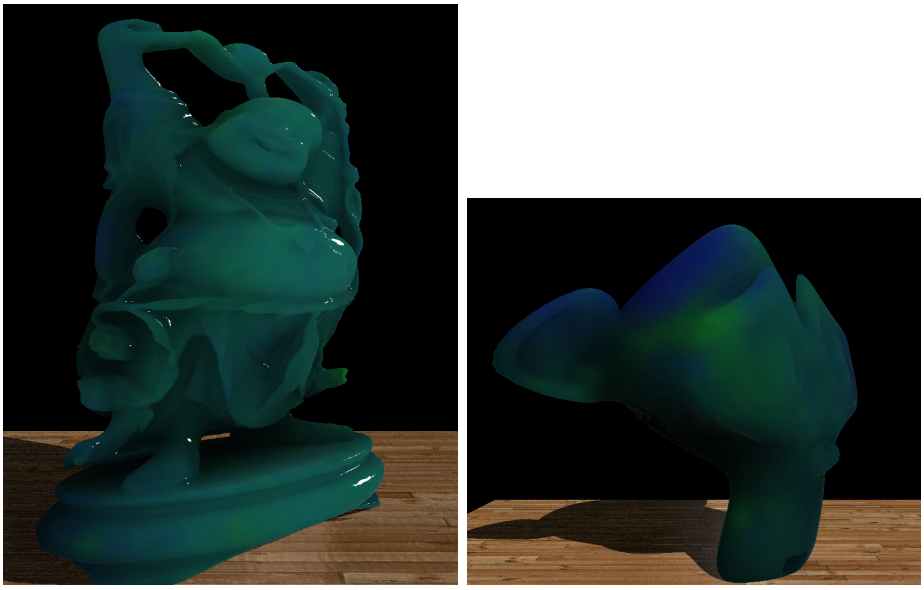

FIGURE 5.11 – Effet d'une mise à l'échelle et d'une déformation de type fuseau sur la diffusion de matériau hétérogène.

le volume avant et après la déformation. Étant donné que les positions après transformation sont différentes, nous ne pouvons plus simplement utiliser les positions des voxels comme indices pour l'échantillonnage des valeurs de coefficients. Une solution simple et efficace à ce problème est de toujours échantillonner par rapport à la position originale du voxel (avant transformation). Nous obtenons ainsi un échantillonnage suivant la déformation. Ce principe est le même que celui utilisé pour la paramétrisation 2D, où les coordonnées (u, v) sont attachées à un point particulier du maillage.

La figure 5.12 montre un comparatif entre l'image rendue et la voxélisation après déformation. Une voxélisation plus fine permettra de mieux représenter les changements de diffusion dus à la déformation.

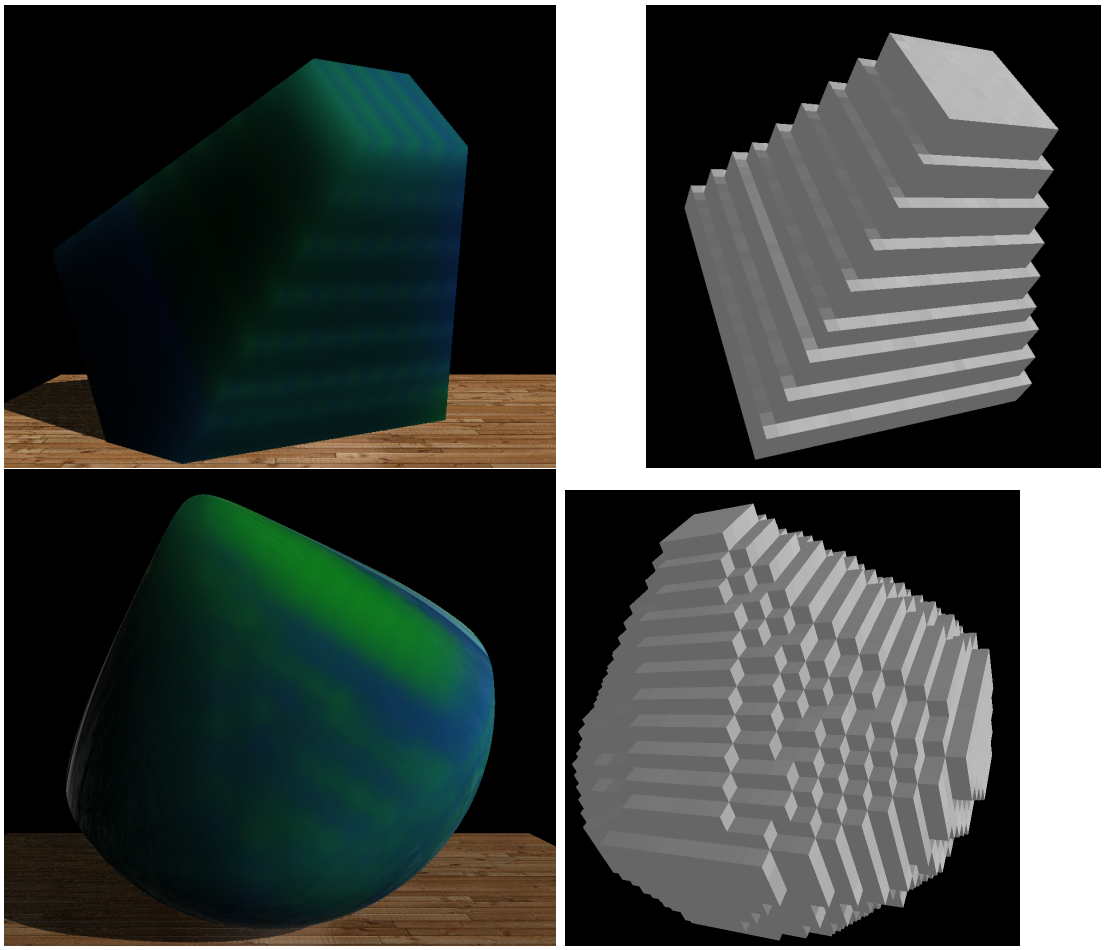

FIGURE 5.12 – Rendu et voxélisation résultant de la déformation d'un cube et d'une sphère.

L'image du bas de la figure 5.13 montre pour sa part l'importance de la robustesse

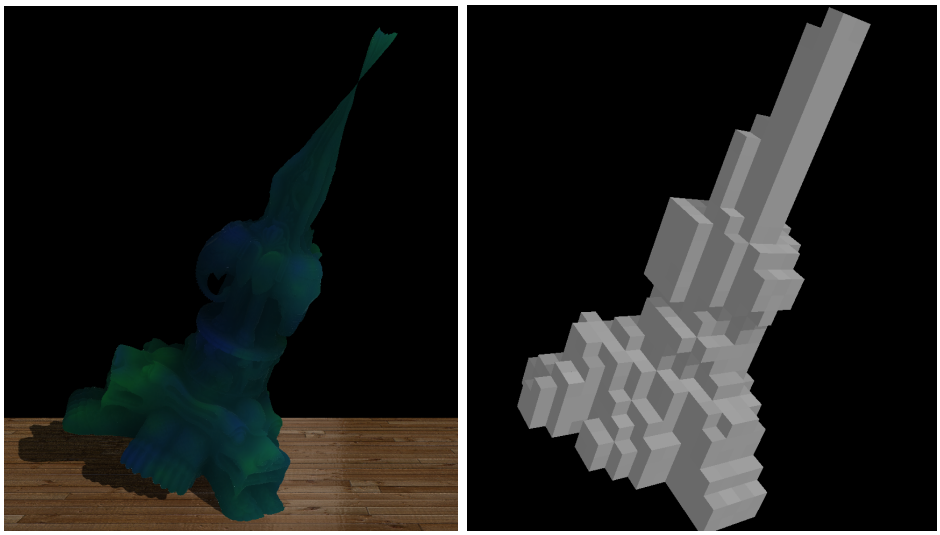

FIGURE 5.13 – Rendu et voxélisation résultant d'une déformation nous donnant un maillage invalide.

de l'algorithme de voxélisation pour garder un domaine de diffusion valide même après des déformations résultant en un maillage incongru.

Chapitre 6

Conclusion

Dans ce mémoire, nous avons abordé les méthodes de rendu efficaces de matériaux semi-transparents sur processeur graphique. Nous avons tout d'abord fait un survol de la théorie de la diffusion de la lumière ainsi que des plus récentes publications utilisant cette dernière dans le contexte de la génération d'images de synthèse.

L'approche décrite dans ce mémoire a été d'utiliser une simplification de l'équation de transport, appelée équation de diffusion, combinée à une résolution massivement parallèle sur *GPU*. Notre méthode se démarque des techniques précédentes en ne requérant aucun pré-calcul pour déterminer le domaine de diffusion. Pour ce faire, nous estimons l'espace volumique en voxélisant le maillage triangulaire en temps réel. Obtenir un domaine de résolution en temps réel permet d'éliminer certaines contraintes des méthodes antérieures, en particulier d'éviter d'avoir à pré-calculer le domaine de diffusion (étape qui peut être très coûteuse en temps de calcul) et ainsi permettre la déformation de maillages sans restriction.

Comme nous l'avons vu, l'ensemble des étapes du pipeline de rendu sont configurables et permettent de varier le niveau de qualité de la simulation selon les performances souhaitées. Pour le calcul de la lumière transmise, nous utilisons un procédé se rapprochant des cartes d'ombrage. En faisant varier la résolution de cette carte, il est possible de choisir le niveau de précision souhaité pour l'initialisation de la diffusion. En combinant ceci au niveau de voxélisation, il est possible de changer dynamiquement le niveau de précision de la simulation. Cette configurabilité peut être utile par exemple pour varier le niveau de précision selon la distance de l'objet à la caméra.

Nous avons aussi intégré notre technique à l'intérieur d'un pipeline de rendu en différé et avons terminé la présentation de notre approche, en montrant les performances obtenues et les résultats de rendus.

En résumé, les contributions de notre technique sont les suivantes :

- Le support de maillages entièrement déformables et de propriétés du matériau définies sur le volume permettant un rendu de matériaux hétérogènes.
- Les sources de lumière de la scène sont complètement dynamiques et donc aucun pré-calcul de lumière n'est nécessaire.
- L'utilisation d'une frontière calculée dynamiquement par voxélisation permettant d'obtenir une simulation complète de l'équation de diffusion.

6.1 Perspectives futures

Bien que nous obtenions des performances permettant l'affichage en temps réel d'objets semi-transparents, notre méthode comporte certaines limitations. La limitation la plus importante réside certainement au niveau de la voxélisation maximale de $32\times32\times32$. Augmenter le niveau de voxélisation offrirait un compromis raisonnable entre vitesse d'exécution et précision de la simulation même s'il serait probablement difficile de garder la composante temps réel du projet pour des résolutions supérieures à $128\times128\times128$. Par contre, il serait intéressant d'étudier la précision obtenue par rapport aux méthodes complètement *offline*, car les gains au niveau des temps de calcul seraient probablement toujours avantageux.

Une amélioration qu'il serait intéressant d'investiguer, serait de remplacer l'accumulation du flux radiatif sur les faces des voxels par un encodage par harmoniques sphériques aux centres de ceux-ci. Les processeurs graphiques actuels étant de très puissantes machines de calcul, ils sont néanmoins contraints par leur bande passante. Plutôt que d'échantillonner chacune des faces, il serait possible de reconstruire la distribution sphérique d'un voxel à l'aide d'un nombre réduit d'accès mémoire. De plus, les étapes d'initialisation et d'injection de la lumière transmise en seraient simplifiées en évitant d'avoir à rediriger le flux radiatif arrivant aux frontières de l'objet vers les faces des voxels.

Actuellement, la synchronisation entre les contextes *OpenGL* et *OpenCL* doit se faire explicitement à l'aide des appels `glFinish` (*OpenGL*) et `clFinish` (*OpenCL*). Ces deux fonctions sont des appels bloquants attendant que l'ensemble des commandes aient été exécutées sur leur contexte respectif avant de rendre la main à l'application. Ces appels bloquants ajoutent une latence d'exécution non-négligeable et empêchent l'agrégation de plusieurs commandes les unes à la suite des autres. L'impact de ces appels est clairement visible sur la trace d'exécution de la figure 5.1 présentée au chapitre précédent. Par exemple, si nous prenons l'appel `glFinish` (en jaune sur la trace), nous voyons que l'appel bloque l'application pour la durée complète de l'exécution de la voxélisation. Idéalement, nous devrions pouvoir mettre en queue les commandes *OpenGL* sans avoir à attendre la complétion du contexte *OpenCL*. Certaines extensions sont présentement disponibles permettant de créer des jetons de synchronnisation afin d'éviter l'attente active causée par les appels `glFinish` et `clFinish`. Il serait intéressant d'ajouter le support à ces primitives à notre projet et d'étudier leur impact sur la performance de notre pipeline de rendu.

Une autre extension qui pourrait s'avérer fort intéressante serait l'ajout de *shading* causé par la diffusion de la lumière à travers un objet semi-transparent. Une solution simple et efficace pourrait être d'ajouter une étape à la fin de notre pipeline créant une texture s'apparentant à une carte d'ombrage mais contenant la couleur diffuse à la sortie de l'objet. Cette texture pourrait par la suite être utilisée avec une technique similaire aux textures projectives décrite par Everitt et al. [ERC01] pour modifier la couleur des objets se trouvant derrière (par rapport à une source de lumière) l'objet semi-transparent.

Une solution plus complexe mais plus générale serait d'injecter la lumière sortant de l'objet à un système de rendu d'illumination globale. Récemment certaines techniques, comme celle proposée par Kaplanyan et Dashbacher [KD10], permettent le calcul de l'illumination globale à l'aide d'une voxélisation de l'espace de la scène. Il serait intéressant de pouvoir intégrer notre technique à un tel système et d'observer l'impact qu'aurait la lumière diffuse sortant de notre objet sur l'ensemble de la scène.

Néanmoins, comme nous l'avons mentionné, notre technique offre tout de même beaucoup d'avantages par rapport aux méthodes actuelles. En plus du support pour les scènes dynamiques et les objets déformables, le calcul du domaine de diffusion et la

résolution en temps réel en font actuellement le système offrant le meilleur compromis entre vitesse d'exécution et précision de simulation.

Chapitre 7

Glossaire

Albédo *albedo*. Ratio entre le coefficient de diffusion et le coefficient d'extinction.

Attributs de points *vertex attributes*. Données rattachées à chaque point d'un maillage (e.g., normale, coordonnées de texture, etc.).

Mélangeur de sortie *blending stage*. Étape du pipeline de rendu temps réel permettant de combiner ceraines opérations entre un fragment source et un pixel du tampon de destination.

BRDF *Bidirectional Reflectance Distribution Function*. Fonction de distribution à quatre dimensions décrivant la quantité d'énergie réfléchie sur une surface opaque par rapport à un angle solide d'entrée et un angle solide de sortie à un point donné **x** de la surface. La BRDF représente le ratio de la radiance en sortie par rapport à l'irradiance en entrée.

BSSRDF *Bidirectional Surface Scattering Distribution Function*. Généralisation du concept de la BRDF en prenant compte de l'énergie entrant sur l'ensemble de la surface. La BSSRDF décrit le transport de la lumière par rapport à deux points et deux angles solides donnés de la surface.

Carte d'ombrage *shadow map*. Carte de profondeur obtenue du point de vue de la lumière, utilisée pour le génération des ombres.

Enroulement des sommets *vertex winding*. Ordre (horaire ou anti-horaire) dans lequel sont définis les sommets.

Fenêtre de visualisation *viewport*. Espace 2D définissant la taille du rendu final à l'écran.

Flux radiatif *fluence*. Valeur obtenue par l'intégration de la radiance sur l'ensemble des directions au voisinage d'un point $\mathbf{x}$.

Rasterisation *rasterization*. Opération permettant de discrétiser un domaine 3D sous forme vectoriel en éléments 2D (fragments ou pixels).

Repère de coupure *clip space*. Espace de la position d'un sommet utilisé par le matériel graphique pour effectuer l'intersection avec la fenêtre de visualisation.

Tampon de destination *render target*. Texture de destination de rendu ou de calcul.

Tampon de sommets *vertex buffer*. Tampon contenant les sommets et les attributs d'un maillage.

Bibliographie

[AMHH08] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., 2008.

[Ash01] Michael Ashikhmin. Microfacet-based BRDFs. In *SIGGRAPH 2001 State of the Art in Modeling and Measuring of Surface Reflection course notes*, 2001.

[AWB11] Adam Arbree, Bruce Walter, and Kavita Bala. Heterogeneous subsurface scattering using the finite element method. *IEEE Transactions on Visualization and Computer Graphics*, 17(7) :956–969, 2011.

[BCK+11] Jesper Borlum, Brian Bunch Christensen, Thomas Kim Kjeldsen, Peter Trier Mikkelsen, Karsten Ostergaard Noe, Jens Rimestad, and Jesper Mosegaard. Sslpv : subsurface light propagation volumes. In *Proc. ACM Symposium on High Performance Graphics*, HPG '11, pages 7–14, 2011.

[BCL+07] Louis Bavoil, Steven P. Callahan, Aaron Lefohn, João L. D. Comba, and Cláudio T. Silva. Multi-fragment effects on the GPU using the k-buffer. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 97–104. ACM, 2007.

[BM08] Louis Bavoil and Kevin Myers. Order-independent transparency with dual depth peeling. *White Paper, NVIDIA*, 2008.

[DEJ+99] Julie Dorsey, Alan Edelman, Henrik Wann Jensen, Justin Legakis, and Hans Køhling Pedersen. Modeling and rendering of weathered stone. In *Proc. SIGGRAPH '99*, pages 225–234, 1999.

[DI11] Eugene D'Eon and Geoffrey Irving. A quantized-diffusion model for rendering translucent materials. In *SIGGRAPH '11 : ACM SIGGRAPH 2011 papers*, pages 1–14. ACM, 2011.

[DJ05] Craig Donner and Henrik Wann Jensen. Light diffusion in multi-layered translucent materials. *ACM Trans. Graph.*, 24 :1032–1039, Juillet 2005.

[DJ08] Craig Donner and Henrik Wann Jensen. Rendering translucent materials using photon diffusion. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, pages 4 :1–4 :9. ACM, 2008.

[dL08] Eugene d'Eon and David Luebke. *Advanced Techniques for Realistic Real-Time Skin Rendering.* Addison-Wesley, 2008.

[Don06] Craig Steven Donner. *Towards realistic image synthesis of scattering materials.* PhD thesis, 2006. AAI3226771.

[DRS07] Julie Dorsey, Holly Rushmeier, and Franois Sillion. *Digital Modeling of Material Appearance.* Morgan Kaufmann Publishers Inc., 2007.

[DS03] Carsten Dachsbacher and Marc Stamminger. Translucent shadow maps. In *Proceedings of the 14th Eurographics Workshop on Rendering*, EGWR '03, pages 197–201. Eurographics Association, 2003.

[DS05] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proc. 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 203–231, 2005.

[ERC01] Cass Everitt, Ashu Rege, and Cem Cebenoyan. Hardware shadow mapping. *White paper, NVIDIA*, 2001.

[Eve01] Cass Everitt. Interactive order-independent transparency. *White paper, NVIDIA*, 2001.

[Gre04] Simon Green. *Real-Time Approximations to Subsurface Scattering.* Addison-Wesley, 2004.

[GT10] Holger Gruen and Nicolas Thibieroz. OIT and indirect illumination using DX11 linked lists. *Tech. rep., AMD*, 2010.

[HK93] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. In *Proc. SIGGRAPH '93*, pages 165–174, 1993.

[HV04] Xuejun Hao and Amitabh Varshney. Real-time rendering of translucent meshes. *ACM Trans. Graph.*, 23 :120–142, April 2004.

[Ish78] A. Ishimaru. *Wave propagation and scattering in random media. Volume I - Single scattering and transport theory.* Academic Press, 1978.

[JB05] Henrik Wann Jensen and Juan Buhler. A rapid hierarchical rendering technique for translucent materials. In *ACM SIGGRAPH 2005 Classes*, 2005.

[JC98] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. In *Proc. SIGGRAPH '98*, pages 311–320, 1998.

[JDZJ07] Wojciech Jarosz, Craig Donner, Matthias Zwicker, and Henrik Wann Jensen. Radiance caching for participating media. In *ACM SIGGRAPH 2007 sketches*, SIGGRAPH '07. ACM, 2007.

[JMLH01] Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. In *Proc. SIGGRAPH '01*, pages 511–518. ACM, 2001.

[KD10] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10, pages 99–107. ACM, 2010.

[LHLW09] Fang Liu, Meng-Cheng Huang, Xue-Hui Liu, and En-Hua Wu. Efficient depth peeling via bucket sort. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 51–57. ACM, 2009.

[Mam89] Abraham Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Comput. Graph. Appl.*, 9(4) :43–55, July 1989.

[MKB+03] Tom Mertens, Jan Kautz, Philippe Bekaert, Hans-Peter Seidel, and Frank Van Reeth. Interactive rendering of translucent deformable objects. In *Proceedings of the 14th Eurographics Workshop on Rendering*, EGWR '03, pages 130–140. Eurographics Association, 2003.

[PH00] Matt Pharr and Pat Hanrahan. Monte Carlo evaluation of non-linear scattering equations for subsurface reflection. In *Proc. SIGGRAPH '00*, pages 75–84, 2000.

[PH04] Matt Pharr and Greg Humphreys. *Physically Based Rendering : From Theory to Implementation*. Morgan Kaufmann Publishers Inc., 2004.

[SRNN05] Bo Sun, Ravi Ramamoorthi, Srinivasa G. Narasimhan, and Shree K. Nayar. A practical analytic single scattering model for real time rendering. *ACM Trans. Graph.*, 24 :1040–1049, July 2005.

[SS10] Michael Schwarz and Hans-Peter Seidel. Fast parallel surface and solid voxelization on GPUs. In *ACM SIGGRAPH Asia 2010*, pages 179 :1–179 :10, 2010.

[Sta95] Jos Stam. Multiple scattering as a diffusion process. In *Proceedings of the 6th Eurographics Workshop on Rendering*, EGWR '95, pages 41–50, 1995.

[VBGS08] Romain Vergne, Pascal Barla, Xavier Granier, and Christophe Schlick. Apparent relief : a shape descriptor for stylized shading. In *NPAR '08 : Proceedings of the 6th International Symposium on Non-photorealistic Animation and Rendering*, pages 23–29. ACM, 2008.

[VPB+10] Romain Vergne, Romain Pacanowski, Pascal Barla, Xavier Granier, and Christophe Schlick. Radiance scaling for versatile surface enhancement. In *I3D '10 : Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 143–150. ACM, 2010.

[WCPW+08] Rui Wang, Ewen Cheslack-Postava, Rui Wang, David Luebke, Qianyong Chen, Wei Hua, Qunsheng Peng, and Hujun Bao. Real-time editing and relighting of homogeneous translucent materials. *Vis. Comput.*, 24(7) :565–575, July 2008.

[WWH+10] Yajun Wang, Jiaping Wang, Nicolas Holzschuch, Kartic Subr, Jun-Hai Yong, and Baining Guo. Real-time rendering of heterogeneous translucent objects with arbitrary shapes. *Computer Graphics Forum (Proceedings of Eurographics 2010)*, 2010.

[WZT+08] Jiaping Wang, Shuang Zhao, Xin Tong, Stephen Lin, Zhouchen Lin, Yue Dong, Baining Guo, and Heung-Yeung Shum. Modeling and rendering of heterogeneous translucent materials using the diffusion equation. *ACM Trans. Graph.*, 27 :9 :1–9 :18, March 2008.