Université de Montréal

Génération d'ombres floues provenant de sources de lumière surfaciques à l'aide de tampons d'ombre étendus

par

Jean-François St-Amour

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

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

Août 2004

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé

Génération d'ombres floues provenant de sources de lumière surfaciques à l'aide de tampons d'ombre étendus

présenté par

Jean-François St-Amour

a été évalué par un jury composé des personnes suivantes:

Yann-Gaël Guéhéneuc, président du jury
Pierre Poulin, directeur de recherche
Eric Paquette, codirecteur de recherche
Victor Ostromoukhov, membre du jury

Mémoire accepté le

Sommaire

Nous présentons une nouvelle méthode de pré-calcul d'ombres floues de haute qualité pouvant être projetées sur des objets dynamiques ajoutés à la scène. Nous stockons tous les événements de visibilité en combinant efficacement la visibilité calculée à partir de plusieurs *shadow maps* dans une structure de type *deep shadow map* (*DSM*) étendue. La structure résultante capture bien les changements d'atténuation dans chaque pixel du *DSM* et constitue donc une représentation précise de l'atténuation de la lumière, pour des sources de lumière de toute taille. Nous montrons comment cette structure peut être compressée et comment le matériel graphique récent peut être utilisé pour projeter en temps interactif des ombres floues complexes sur des scènes statiques et des objets animés à l'intérieur de ces scènes.

Mots-clés :
Infographie 3D, image de synthèse, rendu, illumination, matériel graphique, visibilité, ombre.

Abstract

We present a new method of precomputing high-quality soft shadows that can be cast on dynamic objects added to the scene. We store all visibility events by efficiently merging the visibility computed from many shadow maps into an extended deep shadow map (DSM) structure. The resulting structure effectively captures the changes of attenuation in each DSM pixel, and therefore constitutes an accurate representation of light attenuation from extended light sources of all sizes. We show how it can be compressed and how modern programmable graphics hardware technology can be used to cast real-time complex soft shadows on static scenes and on animated objects within those scenes.

Keywords :

3D computer graphics, image synthesis, rendering, illumination, graphics hardware, visibility, shadow.

Table des matières

Liste des tableaux

Table des figures

Liste des Algorithmes

Remerciements

J'aimerais d'abord et avant tout remercier mes deux directeurs de recherche : Pierre Poulin et Eric Paquette. Ça n'a pas dû être facile, mais ils ont réussit à faire de moi un chercheur et j'ai énormément apprécié travailler avec eux.

De plus, il est essentiel que je remercie les étudiants du LIGUM, qui ont rendu ces deux années plus faciles et combien plus intéressantes. Chacun d'eux a enrichi ma vie à sa façon. En particulier, merci aux deux personnes qui m'ont tenu compagnie dans mon local : Martin Côté pour nos séances musicales intenses et Philippe Beaudoin pour nos séances de PS2 beaucoup trop intenses et pour avoir été mon unique compagnon de *hardware* au labo.

Merci aussi à mes parents, Jean-Paul et Diane, ainsi que ma soeur Annie, de m'avoir supporté (pour ne pas dire enduré) pendant les moments de stress plus élevés.

Finalement, merci au financement du CRSNG qui a rendu mes études possibles.

Droits d'auteur

Il aurait souvent été souhaitable de présenter des images des autres travaux sur la détérioration. Cependant, les droits d'auteur contraignent la possibilité de reproduire ces images. Tout de même, certaines images de ce document ont pu être tirées d'articles publiés dans des journaux ou des conférences scientifiques. La provenance des images est clairement indiquée dans les figures et la référence au travail est également fournie. Voici la note de droits d'auteur pour les travaux tirés d'une publication d'ACM :

> ACM COPYRIGHT NOTICE. Copyright © 1982-2002 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept, ACM Inc., fax +1 (212) 869-0481, or permissions@acm.org.

Voici la note de droits d'auteur pour les travaux tirés de *Graphics Interface* :

> Copyright © 2001 par l'Association canadienne de l'informatique. Il est permis de citer de courts extraits et de reproduire des données ou tableaux du présent compte rendu, à condition d'en identifier clairement la source.

Voici la note de droits d'auteur pour les travaux tirés de *Eurographics* :

Notes on the assignment of copyright

1. In assigning copyright, the author retains his/her personal right to re-use the material in future collections of his/her own work without fee to the publisher. Acknowledgement of prior publication is the only requirement in such cases.

2. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than the publisher must be honoured and acknowledged.

3. Permission will not be given to any third party to reprint material without the author's consent (which is assumed to have been given if we receive no response from him/her within thirty days after writing to the last known address).

4. In cases where permission is sought to reproduce the existing typesetting (e.g. photocopying, microfilm or other forms of electronic reproduction), the author's consent will not normally be sought before granting permission to a third party.

5. Should the copyright be held by someone other than the author, the Publisher requires that assignment be made by the copyright owner. Correspondence relating to permissions will nevertheless be conducted with the author, who is presumed to be authorised by the copyright owner to deal with such questions on his/her behalf.

Chapitre 1

Introduction

There is nothing more difficult to take in hand, more perilous to conduct or more uncertain in its success than to take the lead in the introduction of a new order of things.

Niccolo Machiavelli,
The Prince

Les ombres procurent d'importants indices sur les relations spatiales des différents éléments de la scène, *i.e.,* des objets entre eux et des objets par rapport aux lumières [KMK94, MKK98, HWSB99] (figure 1.1). De plus, elles donnent de l'information sur les sources de lumière elles-mêmes : forme, intensité, distance et orientation. C'est donc pourquoi une quantité imposante de recherche se fait à leur sujet depuis longtemps [WPF90, CCOD+04]. Les ombres floues sont les plus désirables, car elles ajoutent un réalisme précieux aux images synthétiques.

Avant d'en arriver aux motivations et aux contributions de ce mémoire, quelques concepts de base seront exposés pour faciliter le reste de la lecture.

FIG. 1.1 – Perception de la relation spatiale entre les différents éléments d'une scène, grâce aux ombres (image tirée de [HLHS03]).

1.1 Concepts de base à propos des ombres

1.1.1 Qu'est-ce qu'une ombre ?

Considérons une scène contenant une source de lumière S. Un point P de la scène est considéré dans ce que nous appelons l'*umbra* si aucune partie de S n'est visible de ce point, *i.e.,* s'il ne reçoit aucune lumière directement de la source de lumière.

Si S est partiellement visible pour P, P est dans la *pénombre*. L'union de l'*umbra* et de la *pénombre* forme l'ombre. Les objets directement éclairés sont appelés des bloqueurs, car ils bloquent la lumière des objets derrière eux (les receveurs). Il est à noter que cette distinction n'est pas binaire : un objet peut être à la fois bloqueur et receveur. Par exemple, un même objet faisant de l'ombre sur lui-même (*self-shadowing*) est ainsi à la fois bloqueur et receveur.

Cette distinction entre *receveurs* et *bloqueurs* sera importante plus tard, car certains objets de notre scène ne seront que des *receveurs*.

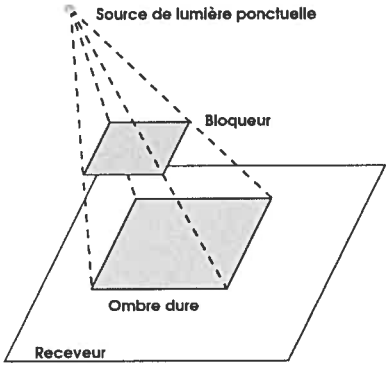


FIG. 1.2 – Ombres dures (image adaptée de [HLHS03]).

1.1.2 Ombres dures vs ombres floues

Le concept d'ombre est souvent vu comme un concept binaire, *i.e.,* un point est dans l'ombre ou il ne l'est pas. Ceci correspond à des ombres dures, comme celles produites par une lumière ponctuelle ou directionnelle. En effet, une lumière ponctuelle ne peut être que visible ou cachée d'un point receveur (figure 1.2). Il est par contre à noter que les lumières ponctuelles n'existent pas dans le monde réel et donc que les ombres dures donnent une apparence synthétique aux images.

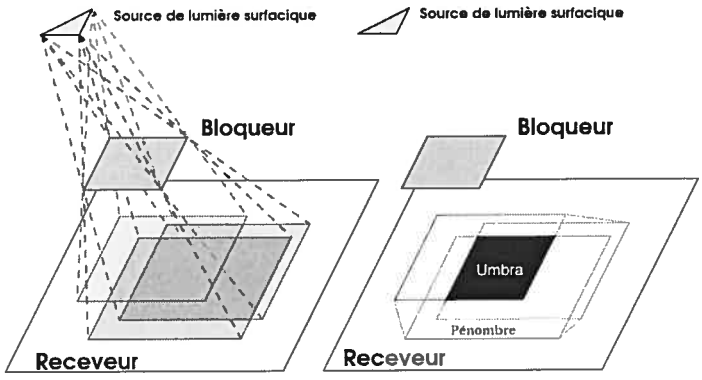


FIG. 1.3 – Ombres floues (image adaptée de [HLHS03]).

Dans le cas plus réaliste d'une lumière surfacique, un point sur un *receveur* pourrait

ne voir qu'une fraction de la lumière. C'est ici que se fait la distinction entre la zone complètement éclairée, la *pénombre* et la zone de *umbra*. Le calcul exact de ces zones (figure 1.3) est particulièrement difficile (et donc habituellement plus long), mais les ombres floues donnent des images beaucoup plus près de la réalité (figure 1.4).

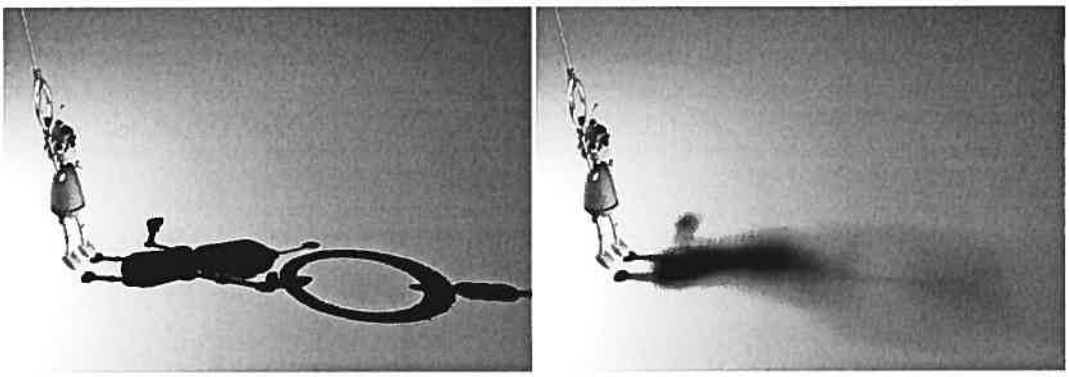


FIG. 1.4 – Ombre dure vs. ombre floue (image tirée de [HLHS03]).

1.2 Motivation

Comme nous l'avons mentionné précédemment, le réalisme des ombres floues vient souvent au prix d'un temps de calcul beaucoup plus considérable que sa contrepartie des ombres dures. Pour cette raison, leur génération a traditionnellement été le fief de techniques non interactives. Par contre, certaines de ces méthodes permettent, après des temps de pré-calcul variant de quelques secondes à quelques heures, un rendu interactif de la scène, à la condition que la scène demeure statique, *i.e.,* que tous les éléments (objets et lumières) demeurent immobiles.

Les récentes innovations dans le domaine du matériel graphique ont rendu possible de nouvelles méthodes de génération interactive d'approximations d'ombres. Bien que ces méthodes donnent souvent de bons résultats pour un nombre limité d'objets dynamiques, elles ne possèdent pas le très haut niveau de qualité des méthodes en pré-calcul, traditionnellement utilisées en production d'images synthétiques.

Ces deux classes de techniques à rendu interactif, avec et sans pré-calcul, sont intéressantes, car elles apparaissent immédiatement comme étant complémentaires. D'une part, les algorithmes en pré-calcul permettent le rendu interactif d'ombres de très haute qualité, mais pour des scènes statiques uniquement. De l'autre, les algorithmes en temps réel permettent une génération d'ombres floues interactive et ce même pour des objets dynamiques, au prix d'une qualité moindre et d'une limite sur la complexité et-ou la quantité des objets générant de l'ombre. Dans un contexte de rendu en temps réel de mondes virtuels (*e.g.,* jeux vidéo ou applications de type *walkthrough*), où on ajoute quelques objets dynamiques (personnages, véhicules, *etc.*) dans un monde statique souvent énorme, il apparaît intéressant de pouvoir utiliser les deux types de techniques conjointement.

1.3 Contributions

Ce mémoire propose une méthode de génération d'ombres floues, basée sur la structure du *deep shadow map (DSM)* [LV00], qui comble le vide entre les techniques avec et sans pré-calcul. Notre méthode permet :

- des ombres floues pré-calculées de haute qualité, pouvant être rendues à l'aide de matériel graphique moderne et donc pouvant être intégrées dans la plupart des moteurs graphiques ;
- l'ajout d'objets dynamiques à la scène, et ce *après* le pré-calcul des ombres.

Avec une structure du type *DSM*, nous pouvons évaluer la fonction de visibilité pour n'importe quel point dans l'espace, ce qui permet de traiter correctement l'ombre sur les objets dynamiques ajoutés après le pré-calcul de l'ombre. Bien que ces objets ne feront pas automatiquement de l'ombre (*i.e.,* ils ne seront que des receveurs), des études [Wan92] ont montré que les humains sont peu sensibles à la qualité des ombres d'objets dynamiques. Il serait donc acceptable de générer ces ombres à l'aide des récentes méthodes interactives qui seront discutées au chapitre 2. Cet aspect de notre

technique, ainsi que la qualité accrue des images en résultant, nous positionnent de façon favorable face aux travaux de Agrawala *et al.* [ARHM00], dont certains éléments sont similaires au travail présenté ici. Comme il sera discuté au cours de ce mémoire, notre technique peut se positionner tant au niveau du rendu de production de haute qualité que du rendu interactif.

En résumé, les contributions de ce mémoire sont :

1. La construction du *DSM* pour des ombres floues de sources étendues de lumière ;
2. Le rendu interactif de ces ombres de haute qualité.

Chapitre 2

Travaux antérieurs

Who so neglects learning in his youth,
Loses the past and is dead for the future.

Euripides,
Phrixus

Bien que le calcul des ombres soit relié au calcul de visibilité [COCSD03], un problème omniprésent en infographie, il a ses particularités, ce qui a mené au développement de plusieurs algorithmes et structures dédiés à la résolution du problème du calcul d'ombres [WPF90]. Les techniques de calcul d'ombres peuvent être, dans la majorité des cas, regroupées en quatre catégories : les techniques basées sur le lancer de rayons, les techniques d'illumination globale, les techniques basées sur les volumes d'ombre et finalement, celles basées sur les tampons de profondeur (*depth buffers* ou *shadow maps*). Nous discuterons brièvement des trois premières catégories, pour principalement nous intéresser à la dernière, plus étroitement reliée à notre approche. La plupart des méthodes sont décrites plus en détail dans différents livres et *surveys* [WPF90, AMH02, HLHS03, CCOD+04].

2.1 Méthode du lancer de rayons

Le calcul d'ombre à l'aide de la méthode du lancer de rayons [Whi80] est très simple et générale (elle n'est pas limité aux modèles polygonaux). À partir du point où nous désirons calculer l'ombre, il suffit de lancer un ou plusieurs rayons vers la lumière et d'accumuler la proportion de ces rayons qui intersectent une surface avant d'atteindre la lumière. Le nombre de rayons lancés détermine le réalisme et la qualité de l'ombre. Un nombre important de rayons est souvent nécessaire afin de bien échantillonner la surface de la lumière. Plusieurs articles ont été écrits sur le lancer de rayons ; les livres de Glassner et de Shirley sont de bonnes introductions au sujet [Gla89, Shi00].

Malheureusement, cette simplicité vient au prix d'un temps de calcul élevé et le lancer de rayons n'est donc normalement pas capable d'effectuer un rendu d'images en temps interactif, bien que beaucoup de recherche se fasse encore à ce sujet [WDP99, PBMH02, CHH02, WDS04].

2.2 Méthodes d'illumination globale

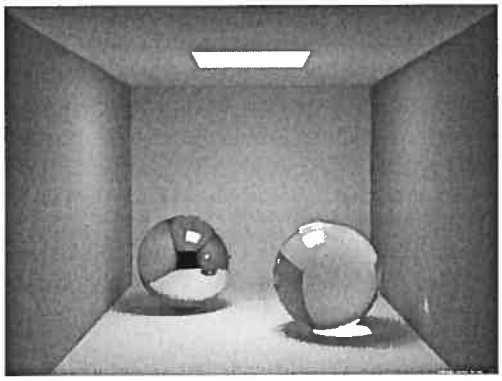


FIG. 2.1 – Exemple d'illumination globale à l'aide du *photon map* (image tirée de [Jen96]).

Les techniques d'illumination globale tiennent compte non seulement de la lumière

venant directement de la source de lumière, mais aussi de la lumière provenant de sources "indirectes", comme par exemple l'interréflexion (souvent très douce) entre les murs d'une pièce (voir figure 2.1). Deux méthodes d'illumination globale très populaires sont la radiosité [CW93, SP94] et le *photon mapping* [Jen01]. Ces deux méthodes traitent implicitement les ombres, mais le *photon mapping* peut aussi les traiter explicitement à l'aide de *shadow photons* [JC95]. Comme ces techniques traitent l'illumination d'une scène de façon globale, elles produisent des ombres floues de deux types : celles causées par les lumières surfaciques et celles causées par les réflexions multiples de l'illumination globale. Comme le lancer de rayons, les techniques d'illumination globale ne permettent que rarement un rendu interactif, malgré de récents avancements à ce niveau [WPS+03, WGS04, GWS04].

2.3 Méthodes basées sur les volumes d'ombre

Une façon intuitive de penser aux ombres est de façon purement géométrique. Cette approche a d'abord été décrite par Crow [Cro77] et ensuite implantée à l'aide de matériel graphique par Heidmann [Hei91].

L'algorithme consiste d'abord à trouver les silhouettes des bloqueurs (du point de vue de la lumière*), puis à faire une extrusion des silhouettes le long de la direction de la lumière, créant ainsi les *volumes d'ombre* (voir figure 2.2). Les objets se trouvant à l'intérieur d'au moins un volume d'ombre sont dans l'ombre, ceux à l'extérieur sont illuminés.

Les volumes d'ombre sont calculés comme suit :

- La première étape consiste à trouver la silhouette des bloqueurs, vue de la lumière. La façon la plus simple est d'identifier les segments partagés par un polygone face à la lumière et un autre dans la direction opposée.
- Nous construisons ensuite les polygones d'ombre en faisant l'extrusion de chaque

*L'algorithme de volumes d'ombre traite les lumières ponctuelles et directionnelles.

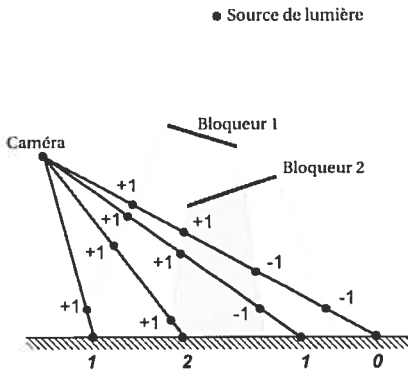


FIG. 2.2 – Illustration en 2D des volumes d'ombre (image adaptée de [HLHS03]).

segment de silhouette le long de la direction depuis la lumière, formant ainsi un long polygone (quadrilatère) d'ombre. Toutes ces extrusions définissent un volume, et savoir si un point est dans l'ombre revient à savoir si ce point est dans le volume.

– Pour chaque pixel de l'image, nous comptons le nombre de polygones d'ombre traversés du plan image jusqu'à l'objet à rendre. Les polygones d'ombre faisant face à la caméra incrémentent le compteur, les autres le décrémentent. Si le compteur est positif à la toute fin, le point est dans l'ombre (voir figure 2.2).

Le rendu avec l'aide du matériel graphique se fait aisément à l'aide du *stencil buffer* : les polygones d'ombre incrémentent ou décrémentent les pixels du *stencil buffer* selon leur orientation. Lors d'un rendu subséquent, seuls les pixels où le compteur du *stencil buffer* est à zéro seront traités. L'algorithme complet de rendu à l'aide des volumes d'ombre est donc :

1. faire le rendu de la scène avec l'illumination ambiante seulement ;
2. calculer et faire le rendu des volumes d'ombre dans le *stencil buffer* (toujours avec le test du z activé) ;
3. faire le rendu de la scène complète avec le test du *stencil buffer* activé : seuls les points où la valeur du *stencil buffer* est zéro sont rendus, tous les autres ne sont

pas modifiés, conservant seulement leur couleur ambiante.

L'algorithme de volumes d'ombre a beaucoup d'avantages :

- génération et rendu interactif pour scènes dynamiques ;
- permet l'utilisation de lumières omnidirectionnelles ;
- rendu d'ombres à la précision du pixel de caméra ;
- le *self-shadowing* est traité correctement.

FIG. 2.3 – Image provenant de *Doom 3* (produit par id Software).

Il a aussi plusieurs problèmes :

- le temps de calcul augmente rapidement avec la complexité de la scène ;
- requiert le calcul de la silhouette des bloqueurs ;
- au moins deux passes de rendu sont nécessaires ;
- malgré des améliorations récentes [GLY+03, LWGM04, CD04] à la technique, le rendu des polygones d'ombre consomment une grande quantité de *fillrate* sur

la carte graphique. Par exemple, prenons le cas de le moteur graphique du jeu *Doom 3*. Il s'agit d'un moteur très populaire, à la fine pointe de la technologie, traitant entre autres du *bump mapping*, du *shading* en précision pixel, de l'illumination dynamique, des effets atmosphériques et un système de particules (voir figure 2.3). Malgré toute cette complexité, le seul calcul des ombres à l'aide de volumes d'ombre compte pour 50% du temps de rendu. La compagnie produisant le moteur a même annoncé que, malgré des débuts prometteurs, les volumes d'ombre ne seraient plus utilisés dans leurs futurs projets.

– si des polygones d'ombre sont *en avant* du plan image[†], les valeurs du *stencil buffer* sont fausses. Everitt et Kilgard [EK02] proposent la méthode dite du *z-fail*, qui corrige le problème.

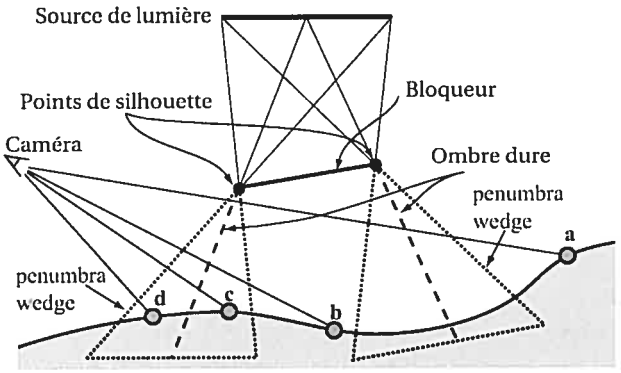


FIG. 2.4 – Diagramme illustant l'algorithme de *penumbra wedges* (image adaptée de [AAM03]).

Une extension récente aux volumes d'ombre, soit les *penumbra wedges* [AMA02, AAM03], produit des ombres floues de qualité de façon interactive (voir figures 2.4 et 2.5), mais les besoins en *fillrate* sont grandement augmentés, ce qui rend la technique inutilisable pour le rendu d'une grande quantité de polygones. De plus, bien que les résultats soient plutôt convaincants, leur technique reste approximative [‡].

[†] *i.e.*, entre la position de la caméra et le *front clipping plane*.

[‡] Les silhouettes sont calculées à partir du centre de la lumière, ce qui est faux pour toute lumière

FIG. 2.5 – Résultat de l'algorithme de *penumbra wedges* (image tirée de [AAM03]).

2.4 Méthodes basées sur le tampon de profondeur

2.4.1 Shadow mapping

Le calcul d'ombre consiste à identifier les parties de la scène qui sont cachées de la source de lumière. Cela revient donc intrinsèquement à un calcul de visibilité selon le point de vue de la lumière. La technique du *shadow mapping* [Wil78] exploite ce parallèle.

L'algorithme de *shadow mapping* se fait en deux passes de rendu :

1. Premièrement, la scène est rendue du point de vue de la source de lumière. Les valeurs contenues dans le *z-buffer* sont alors sauvegardées, elles forment le tampon de profondeur des ombres (*shadow map*) (voir figure 2.6).
2. Ensuite, un rendu de la scène est fait du point de vue de la caméra, en utilisant le *shadow map* pour déterminer les parties éclairées ou dans l'ombre. Pour déterminer si un point est dans l'ombre, on le projette dans le *shadow map* de la lumière et on compare la profondeur résultant de cette projection à la profondeur contenue dans le *shadow map*. Si la profondeur du *shadow map* est plus petite, le point est dans l'ombre, sinon il est éclairé (voir figure 2.7).

surfacique. De plus, la fonction d'atténuation est aussi approximative.

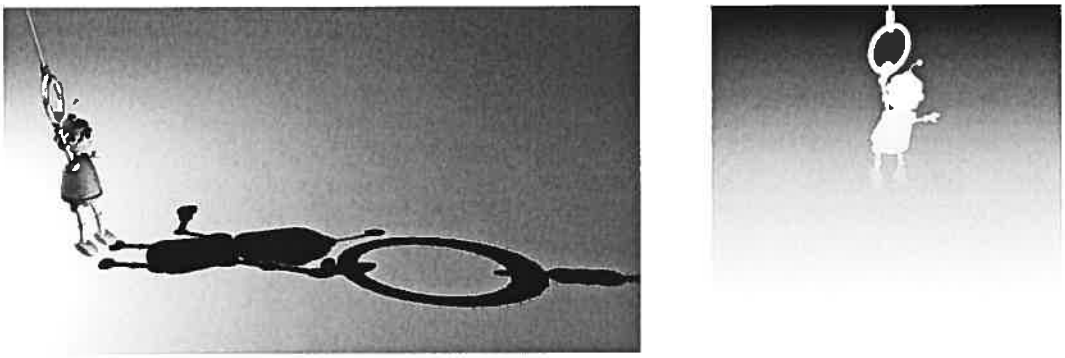


FIG. 2.6 – *Shadow mapping.* À gauche, vue de la caméra. À droite, tampon de profondeur vu de la lumière, encodé en niveaux de gris, où pâle représente une distance plus rapprochée de la lumière (image tirée de [HLHS03]).

L'algorithme du *shadow mapping* est implanté dans le matériel graphique en utilisant les textures [SKvW+92] à l'aide, par exemple, d'extensions à *OpenGL* disponibles sur les cartes nVidia à partir de la GeForce 3 et sur les cartes ATI à partir de la Radeon 9500. Cette approche est en fait une extension du système de textures standard, dans lequel on utilise le *shadow map* comme une texture projective, mais en appliquant un test spécifique au *shadow mapping* (*i.e.,* de comparer le z du *fragment* en espace lumière au z présent dans la texture). Ceci permet à la technique d'atteindre des vitesses de rendu très élevées. La technique du *shadow map* est aussi intéressante car elle n'est pas limitée aux modèles polygonaux, ce qui la rend plus générale que la technique des volumes d'ombre. En effet, elle ne requiert qu'une façon de rendre le modèle géométrique à l'aide d'un algorithme de type *z-buffer.*

Le problème majeur de l'algorithme est la possibilité qu'un aliassage très visible apparaisse au niveau des contours des ombres (voir l'image de gauche de la figure 2.8). Plusieurs techniques ont donc été proposées pour atténuer ce problème. Comme cet aliassage ne sera présent qu'aux frontières de l'ombre, un réflexe intuitif serait de filtrer le résultat du *shadow mapping*, de façon à masquer l'aliassage. Reeves *et al.* proposent leur algorithme de filtrage, le *Percentage Closer Filtering (PCF)* [RSC87], qui effectue

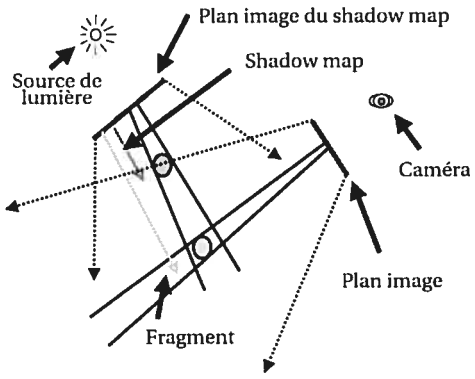


FIG. 2.7 – Diagramme illustrant l'algorithme de *shadow mapping*.

le test du z sur plusieurs pixels voisins du *shadow map*, de façon à obtenir des tons de gris aux frontières de l'ombre. Le *PCF* diminue de beaucoup l'aliassage, mais pour des tailles de filtre assez coûteuses. Le matériel graphique moderne implante une version simplifiée de cet algorithme, sous la forme de filtrage bilinéaire des textures de profondeur.

Il est aussi possible de minimiser l'aliassage, au lieu de le masquer. Fernando *et al.* proposent les *adaptive shadow maps* [FFBG01] qui subdivisent en *quadtree* le tampon de profondeur de façon à permettre un échantillonnage plus raffiné aux frontières de l'ombre et où plusieurs objets projettent dans le même pixel du *shadow map*. Malheureusement, la complexité des structures de données impliquées empêche une implantation matérielle de l'algorithme. D'autres techniques [SCH03, CD04] utilisent la précision des *shadow volumes* aux frontières de l'ombre. Plusieurs techniques ont été proposées pour calculer le *shadow map* en espace perspective [SD02, WSP04, MT04], stockant ainsi plus de détails dans les parties du *shadow map* proches de l'oeil (voir figure 2.8).

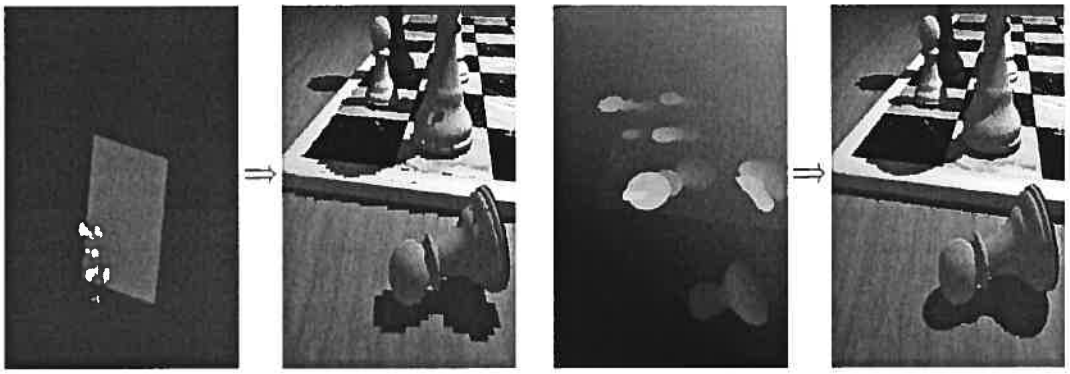


FIG. 2.8 – Le tampon de profondeur en espace perspective. À gauche, un tampon de profondeur traditionnel avec ses problèmes d'aliassage. À droite, en espace perspective (image tirée de [SD02]).

2.4.2 Ombres floues approximatives avec tampons de profondeur

Il existe aussi des techniques basées sur les tampons de profondeur qui permettent le calcul et le rendu d'ombres floues en temps interactif. La performance de ces techniques est supérieure aux *penumbra wedges*, basés sur les volumes d'ombre, mais la qualité des ombres générées est inférieure.

Les *smoothies* [CD03] étendent la silhouette des bloqueurs, de façon à créer une texture projective de pénombre (voir la figure 2.9 pour plus de détails). Les *penumbra maps* [WH03] sont similaires, mais utilisent des plans inclinés et des cônes rendus dans une texture projective de pénombre. Les deux techniques atteignent des temps de rendu interactifs, mais comme elles sont toujours basées sur les textures projectives, elles ne peuvent que représenter la *pénombre externe* (voir la figure 2.10). La pénombre interne est plus difficile à traiter, car elle se trouve *cachée* derrière les bloqueurs. Elle n'est donc tout simplement pas visible du centre de la lumière.

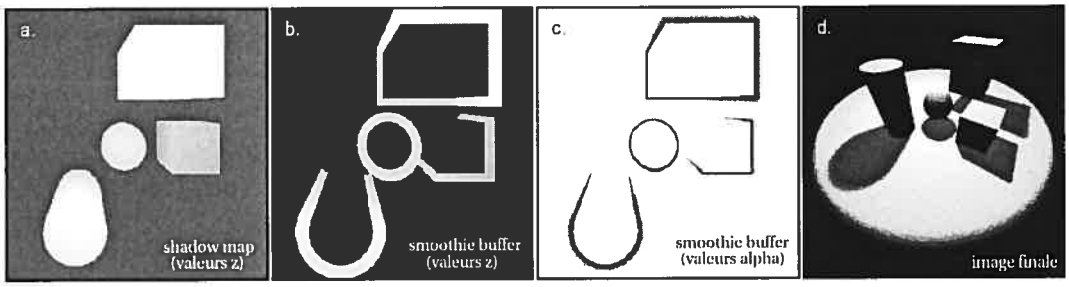


FIG. 2.9 – Algorithme des *smoothies*. (a) Création d'un *shadow map* standard. Puis, construction de primitives (*smoothies*) aux silhouettes. (b) Rendu des *smoothies* dans un tampon de profondeur. (c) Pour chaque pixel du *smoothie buffer*, stockage d'un alpha qui dépend du ratio de distances entre la source de lumière, le bloqueur et le receveur. (d) Finalement, rendu de la scène du point de vue de la caméra avec comparaison avec les valeurs du *shadow map* et du *smoothie buffer* (image adaptée de [CD03]).

2.4.3 Ombres floues avec multiples tampons de profondeur

Comme il a été mentionné dans la section sur le lancer de rayons, il est possible de calculer des ombres floues en échantillonnant la visibilité à plusieurs points sur la source de lumière et en intégrant leurs contributions. Brotman et Badler [BB84] ont été les premiers à adapter cette idée d'échantillonnage au *shadow mapping*. Ils calculent plusieurs tampons de profondeur, pris de points différents sur la source de lumière, puis effectuent le test du *shadow map* sur chacun d'eux en intégrant le résultat. Par exemple, si 25% des tests disent que le point est dans l'ombre, alors le facteur d'atténuation est aussi de 25%. Le problème avec cette technique est que le stockage de ces tampons de profondeur demande beaucoup d'espace mémoire (potentiellement plusieurs giga-octets[§]), ce qui rend la technique inutilisable pour un nombre d'échantillons élevé (souvent nécessaire, voir figure 2.11 pour exemple).

Par contre, les données contenues dans ces tampons de profondeur est très cohé-

[§]Par exemple, pour l'image de droite de la figure 2.11, supposons 1024 échantillons de *shadow map* de taille 2048 × 2048 à 24 bits par pixel. Ceci nous donnerait un total mémoire de 12 giga-octets.

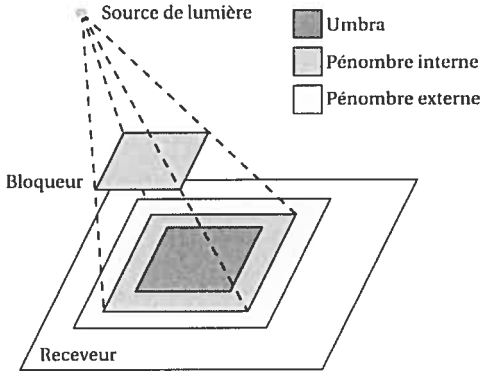


FIG. 2.10 – Pénombre interne et externe (image adaptée de [HLHS03]). La partie interne de la pénombre est celle qui n'est pas visible à partir d'une lumière ponctuelle.

rente. Agrawala *et al.* [ARHM00] regroupent tous ces tampons de profondeur dans une seule structure 3D, appelée *Layered attenuation map (LAM)*. Le *LAM* est construit de la façon suivante :

1. Pour chaque échantillon sur la lumière, une vue est calculée dans la direction de la normale de la lumière.
2. Pour chaque pixel de l'image résultante :
   (a) La coordonnée (x, y, z) du pixel est projetée dans la caméra centrale, avec comme résultat la nouvelle coordonnée (x', y', z').
   (b) Cette nouvelle coordonnée est insérée dans le *LAM*, *i.e.*, qu'au pixel (x', y') du *LAM*, un événement à la profondeur z' est ajouté.
3. Une fois tous les échantillons traités, une passe d'accumulation des événements est faite sur le *LAM*, de façon à calculer le pourcentage d'atténuation à tous les endroits où des événements ont été insérés.

Les résultats sont intéressants (voir la figure 2.12), mais leur technique de projection a toutefois deux désavantages importants :

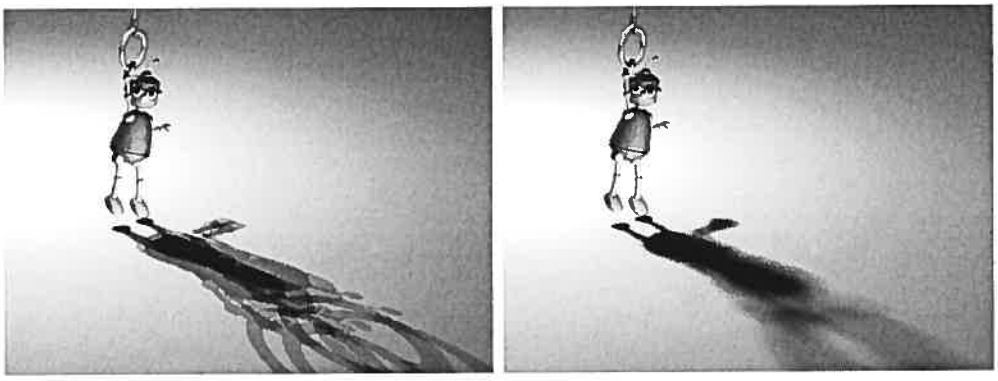


FIG. 2.11 – Effet d'un nombre différent d'échantillons sur la lumière. Gauche : 4 échantillons. Droite : 1024 échantillons (image tirée de [HLHS03]).

- elle ne donne de bons résultats que pour de petites sources de lumière. Pour de grandes sources, les échantillons seront trop distants et des trous apparaîtront dans l'ombre ;
- elle ne contient de l'information qu'aux endroits où se trouve une surface de la scène utilisée pour la construction, ce qui empêche l'ajout d'objets dynamiques dans la scène après le calcul du *LAM*.

2.4.4 Deep Shadow Map

Avec le *DSM*, Lokovic et Veach [LV00] introduisent une structure de visibilité capable de capturer l'occlusion cumulative due à des structures minuscules ou semi transparentes (en particulier, les cheveux, figure 2.13). Une fonction contenant l'augmentation de l'atténuation est construite et stockée pour chaque pixel d'un *shadow map* (voir figure 2.14 pour plus de détails). Kim et Neumann [KN01] construisent efficacement une approximation du *DSM* avec l'aide du matériel graphique.

Un avantage majeur du *deep shadow map* est qu'il effectue un filtrage de plusieurs *shadow maps* en une seule structure. En effet, la fonction finale contenue dans un pixel du *DSM* est en fait une intégration de plusieurs fonctions échantillonnées dans ce pixel.

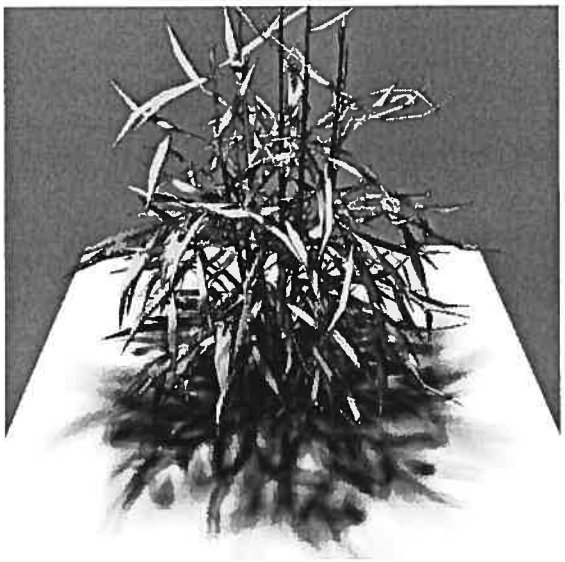


FIG. 2.12 – *Layered attenuation map* (image tirée de [ARHM00]).

Ceci a pour effet d'obtenir avec un *DSM* des résultats qui demanderaient une très grande résolution avec un *shadow map* normal (voir figure 2.13). De plus, comme les fonctions d'atténuation sont généralement assez douces (par morceaux), un algorithme de compression peut être utilisé de façon à réduire la quantité de mémoire nécessaire au stockage des fonctions (voir section 3.2 pour plus de détails).

La structure du *DSM* est particulièrement intéressante car elle permet d'évaluer la fonction d'atténuation n'importe où dans l'espace.

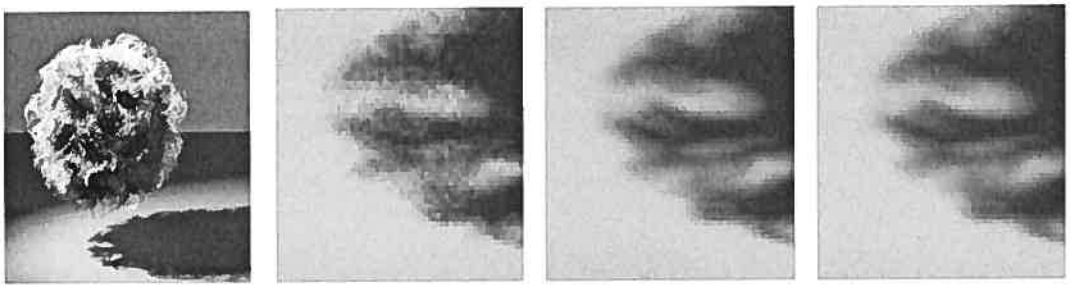


FIG. 2.13 – *Deep shadow map*. De gauche à droite : Boule de 50 000 cheveux, *shadow map* de 512 × 512, *shadow map* de 4096 × 4096, *DSM* de 512 × 512 (image tirée de [LV00]).

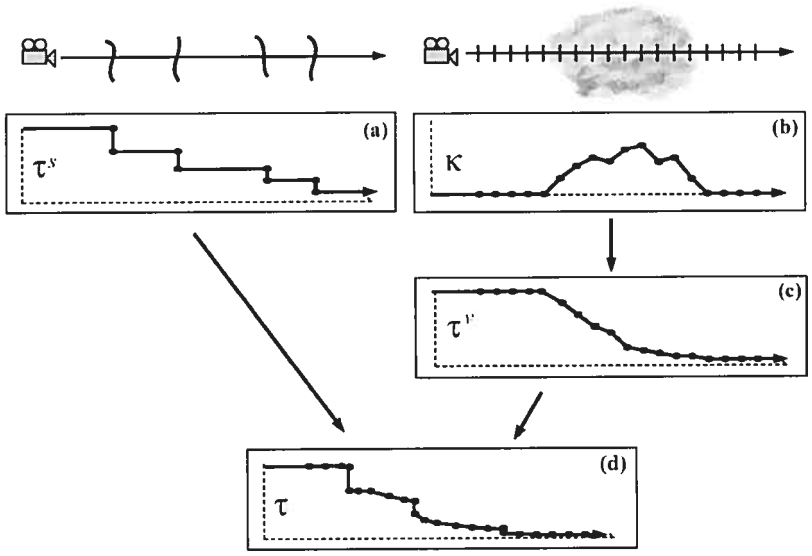


FIG. 2.14 – Construction de la fonction d'un pixel de *deep shadow map*. (a) Les intersections d'objets le long d'un rayon (pixel du *DSM*) donnent la fonction de transmission de surface τ^s, qui a des discontinuités à la profondeur de chaque surface. (b) La fonction d'extinction κ est obtenue en échantillonnant la densité atmosphérique à intervalles réguliers le long du rayon. (c) La fonction d'extinction est intégrée et la transmission de volume τ^v est calculée à partir de la fonction d'extinction. (d) Les transmissions de surface et de volume sont multipliées pour obtenir la fonction de transmission finale τ pour chaque rayon (image tirée de [LV00]).

Chapitre 3

Penumbra Deep Shadow Maps

Look not mournfully into the Past. It comes not back again. Wisely improve the Present. It is thine. Go forth to meet the shadowy Future, without fear, and a manly heart.

Henry Wadsworth Longfellow

Dans ce chapitre, nous discutons l'adaptation nécessaire à la structure du *DSM* pour capturer l'atténuation d'une source de lumière étendue. L'utilisation d'un *DSM* pour stocker de l'information par rapport à des ombres floues a deux avantages majeurs. Premièrement, comme nous justifierons plus tard, cela permet d'éviter les problèmes de trous se présentant lors de la projection de *shadow map* [ARHM00], ce qui résulte en des ombres de meilleure qualité. De plus, le *DSM* nous permet d'évaluer la fonction d'atténuation partout dans l'espace 3D. Ceci rend possible l'insertion de nouveaux objets dans la scène, recevant les ombres correctement, sans avoir à recalculer le *deep shadow map*. Nous ne discutons dans ce chapitre que de la création du *penumbra deep shadow map*, notre extension aux *DSM*, le rendu d'ombres à l'aide de cette structure sera présenté au chapitre 4.

3.1 Construction

Lors du rendu d'un point 3D, la fraction de lumière atteignant directement ce point est nécessaire. Cette information est encodée par le *DSM* pour tous les points 3D dans sa pyramide de vue. Chaque pixel du *DSM* correspond à un rayon 3D, débutant au centre de projection (CDP) du *DSM*, traversant le centre du pixel. Un pixel du *DSM* capture la fonction décrivant la fraction de lumière bloquée tout au long de ce rayon. Pour construire un *DSM* contenant de l'information de pénombre, ce que nous appellerons désormais un *penumbra deep shadow map* (*PDSM*), une technique similaire à celle de Agrawala *et al.* [ARHM00] est utilisée. Comme dans Agrawala *et al.*, nous désirons ajouter la contribution de plusieurs *shadow maps* à une structure 3D, mais dans notre cas il s'agit d'un *deep shadow map*. Nous cherchons donc à calculer comment chaque *shadow map* (*i.e.*, chaque échantillon) affecte la visibilité le long de chacun des rayons du *PDSM*. L'algorithme 3.1 détaille la méthode de construction.

Algorithme 3.1 : Algorithme de génération du *PDSM*.

```
1  Génération de points aléatoires sur la source de lumière.
   pour chaque point (échantillon) faire
2      Calcul d'un shadow map.
       // Insertion de l'information du SM dans le PDSM.
       pour chaque pixel du PDSM faire
3          Calcul du rayon 3D associé.
4          Projection de ce rayon dans le SM.
           pour chaque pixel du SM traversé par le rayon faire
5              si changement de visibilité alors
6                  Insertion d'un événement dans le PDSM.
```

Le calcul du facteur d'atténuation débute par la sélection de points aléatoires sur la source de lumière étendue (étape 1). Nous utilisons un *échantillonnage stratifié* sur

la surface de la lumière pour assurer une variance réduite. Un *shadow map* est ensuite calculé pour chacun des points d'échantillonnage (étape 2). Bien que nous n'avons pas implanté cette fonctionnalité, d'autres distributions, ainsi que différentes fonctions d'importance, pourraient être utilisées [ARBJ03, ODJ04], surtout si l'émission sur la surface de la lumière n'est pas uniforme.

Afin de calculer l'occlusion contribuée par un point échantillonné sur la lumière, nous projetons chaque rayon 3D du *PDSM* dans le *shadow map* du point (étape 4) et calculons ensuite les sections du rayon qui sont dans l'ombre (étape 5). Ces sections sont ensuite combinées avec les sections d'ombre déjà présentes dans le rayon du *PDSM* (étape 6). Il est à noter que c'est cette façon d'échantillonner la pyramide de vue de la lumière qui évite l'apparition de trous dans les ombres créées avec notre technique. Les étapes 5 et 6 sont illustrées dans les figures 3.1 et 3.2. Conceptuellement, notre technique est plutôt simple. Il y a cependant deux étapes qui comportent des difficultés, soient les étapes 5 et 6, que nous élaborons maintenant plus en détail.

La détermination des portions ombragées d'un rayon du *PDSM* projeté est obtenue en faisant la *scan-conversion* du rayon dans le *shadow map* (figure 3.3). Ensuite, l'algorithme garde en mémoire les parties du rayon échouant le test standard du *shadow map*, *i.e.,* les parties du segment de ligne qui seraient ombragées dans un contexte de rendu standard. Pour améliorer l'efficacité, il est seulement nécessaire de garder les points d'entrée et de sortie de chaque section ombragée. De plus, il est plus efficace de traiter les événements d'entrée/sortie comme de simples événements de visibilité, sans aucune connection entre les deux, avec des valeurs de ± 1 dépendant du type d'événement (entrée ou sortie), car ceci permet l'utilisation d'un arbre binaire efficace pour le stockage et l'insertion triée des événements. Une fois le rayon du *PDSM* construit, nous pouvons traiter ces événements (triés en z) séquentiellement pour créer la fonction d'atténuation.

Lorsqu'un pixel contenant un événement de visibilité a été identifié, nous devons déterminer à quelle profondeur le long du rayon du *PDSM* cet événement s'est produit.

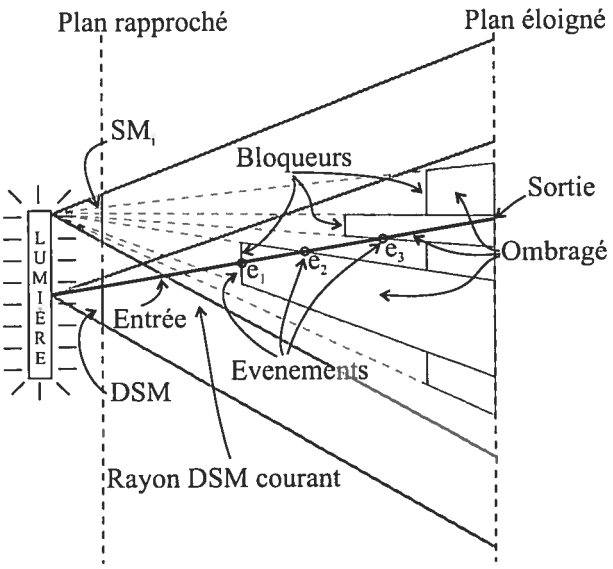


FIG. 3.1 – Un rayon correspond à un seul pixel du *PDSM*. Le rayon du *PDSM* est utilisé pour accumuler l'information d'atténuation calculée à partir du *shadow map* d'un échantillon. Les événements d'ombre (e_1, e_2, e_3, arrière-plan) sont insérés dans ce pixel du *PDSM*.

Dans un contexte de calcul d'ombres, il est important que la précision en z soit la plus grande possible*. Le *shadow map* et l'algorithme de *scan-conversion* du rayon nous procurent de l'information à ce sujet, chacun avec ses avantages et ses inconvénients. Le *shadow map* a l'avantage d'avoir la plus grande précision possible, car nous avons directement la profondeur de la surface, mais a le désavantage majeur de pouvoir nous fournir de l'information fausse quand le rayon passe derrière un objet, comme illustré sur la figure 3.4 (b). La *scan-conversion* quant à elle ne fournit jamais d'information erronée, mais est de précision variable et parfois pauvre. En effet, la précision de l'information fournie par la *scan-conversion* est directement proportionnelle au nombre de

*Si la profondeur est trop petite, des artefacts de *self-shadowing* peuvent apparaître. Si elle est trop grande, l'ombre semble se détacher des objets, ce qui est particulièrement dérangeant au point de contact entre deux objets, en particulier au point de contact entre un objet et le sol.

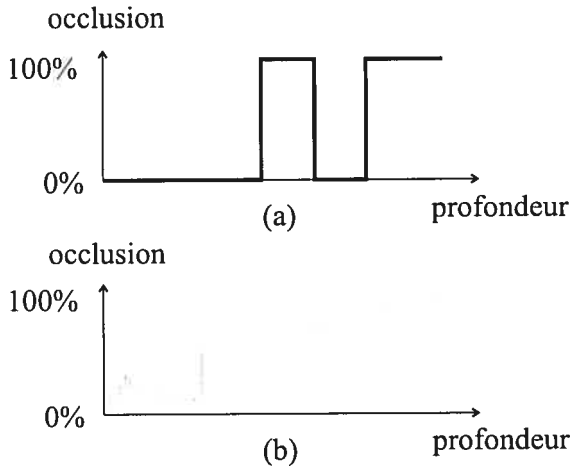


FIG. 3.2 – (a) Visibilité le long d'un rayon pour *un* échantillon. (b) La fonction d'atténuation finale le long du rayon du *PDSM*, *i.e.*, la somme des contributions de chacun des *shadow map* échantillons. Il est à noter que cette fonction n'est pas strictement croissante, contrairement à Lokovic et Veach [LV00].

pixels qu'occupe le rayon dans le *SM*. Plus ce nombre est petit, moins l'information est précise.

Heureusement, nous pouvons utiliser les deux méthodes conjointement, de façon à obtenir de meilleurs résultats. Il est possible d'utiliser l'information de la *scan-conversion* pour valider l'information provenant du *shadow map*, de façon à n'utiliser l'information de la *scan-conversion* que si l'information du *shadow map* est fausse. En effet, la *scan-conversion* nous fournit des bornes inférieures et supérieures sur le z que peut avoir l'événement de visibilité, cet événement se devant d'être entre la profondeur du pixel courant (de la *scan-conversion*) et celle du pixel précédent. Si le z du SM n'est pas entre les bornes du scanconversion, alors l'information provenant du *shadow map* est fausse. La figure 3.4 résume le tout : (a) quand utiliser l'information du *shadow map*, (b) quand utiliser l'information de l'algorithme de *scan-conversion* et (c), comment calculer la profondeur des événements de *sortie*. De plus, nous pouvons intuitivement observer que les deux techniques seront précises de façon complémentaire. Le *shadow*

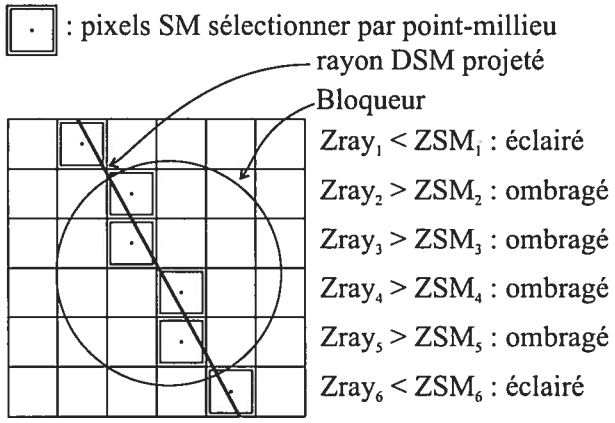


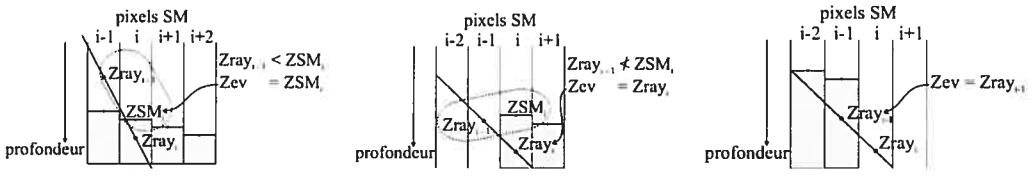
FIG. 3.3 – Le rayon 3D du *PDSM* est projeté dans le *shadow map* et les pixels utilisés pour déterminer la visibilité du rayon sont identifiés en faisant la *scan-conversion* du rayon en utilisant l'algorithme du point milieu ([FvDFH90] pp. 74-81).

map sera probablement plus précis lorsque le point échantillon est proche du centre de la lumière (*i.e.,* du point central et donc du *PDSM)*, alors que c'est l'inverse pour la *scan-conversion*, qui sera plus précise pour des échantillons loin du point central où les rayons du *PDSM* couvriront plus de pixels lors de la *scan-conversion*.

Les rayons du *PDSM* se projetant dans un seul pixel sont traités comme un cas spécial. Une section d'ombre est insérée de la profondeur du *shadow map* jusqu'à l'arrière-plan.

Lors de la combinaison de l'information du *shadow map* avec celle du *PDSM*, les profondeurs du *shadow map* sont transformées en des profondeurs du *PDSM* pour insertion. Pour une source de lumière de forme arbitraire, où les espaces 3D des échantillons[†] et celui du *PDSM* ne sont pas cohérents entre eux, ceci nécessite de déprojeter l'événement de visibilité en *espace monde*, puis de le projeter dans l'espace du *PDSM*. Techniquement, ceci revient à multiplier un point 3D par deux matrices, ce qu'il est très coûteux de faire à chaque événement (*e.g.,* la scène du Dragon de la figure 3.6 contient

[†]L'espace 3D d'un échantillon correspond en fait à l'espace 3D du *shadow map* associé à cet échantillon.



(a) (b) (c)

FIG. 3.4 – Les pixels du *shadow map* correspondent aux régions verticales *i-2, i-1, i, etc.* ; les points représentent l'information de profondeur au centre de chaque pixel du *shadow map* ; la profondeur du rayon du *PDSM* croît en traversant le *shadow map*. Choix entre la profondeur du rayon du *PDSM* et celle du *shadow map* au pixel i : (a) la profondeur du *shadow map* est correcte, (b) la profondeur du *shadow map* est incorrecte, (c) illustration du choix de la profondeur du pixel précédent pour les événements de *sortie*.

plus de 25 millions événements de visibilité). Ce calcul peut être grandement simplifié si la lumière est planaire et si les *shadow maps* sont parallèles entre eux et parallèles à la source de lumière [ARHM00]. Dans ce contexte, les profondeurs dans le *shadow map* correspondent exactement aux profondeurs dans le *PDSM* ; aucun calcul n'est donc nécessaire pour le z (voir figure 3.5). De plus, nous connaissons directement les coordonnées (x, y) de l'événement dans le *PDSM*, puisque nous savons le rayon qui est présentement traité. Nous avons donc directement les coordonnées 3D de l'événement dans le *PDSM*, sans aucun calcul coûteux.

Puisque la résolution en profondeur du *shadow map* est finie, plusieurs événements provenant de différents *shadow maps* peuvent se produire à la même profondeur ou être très proches. Les événements se produisant dans un intervalle de profondeur s'approchant de la résolution en profondeur du *shadow map* sont tout simplement consolidés.

Une fois le *PDSM* construit, nous avons une série d'événements avec leur profondeur et leur valeur d'atténuation pour chaque pixel du *PDSM*. Puisqu'il s'agit d'une représentation de la lumière provenant d'une source étendue, cette fonction n'est pas strictement croissante. Une telle fonction d'atténuation est illustrée dans la figure 3.2 (b).

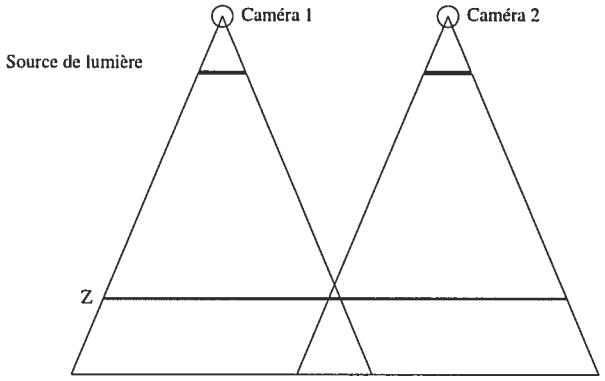


FIG. 3.5 – Tous les points le long de la ligne Z ont la même profondeur, tant dans la caméra de la vue 1 que de la vue 2.

Il est à noter que si l'ajout d'objets dynamiques dans la scène n'est pas désiré, il n'est pas nécessaire de garder tous les événements. Comme Agrawala *et al.* [ARHM00], il est possible de ne conserver que les valeurs d'atténuation proches des surfaces, réduisant ainsi de beaucoup le nombre d'événements et la quantité de mémoire requise.

Comme avec la technique de *shadow mapping* standard, il est nécessaire d'utiliser un biais lors de l'utilisation d'un *PDSM*. Ce biais peut être insérer lors de la construction du *PDSM* ou lors de l'évaluation de la fonction d'atténuation pendant le rendu. Nous avons choisi la première option, qui est le choix standard lors de l'utilisation de *shadow maps*.

3.2 Compression

Les fonctions d'atténuation construites avec notre méthode tendent à avoir beaucoup de sommets (événements), dépendant de la taille de la lumière et du nombre d'échantillons. Heureusement, ces fonctions sont en général assez lisses et donc facilement compressibles. Il est cependant important que la méthode de compression conserve la profondeur des événements importants, car même des petites erreurs en z peuvent causer des artefacts importants de *self-shadowing*.

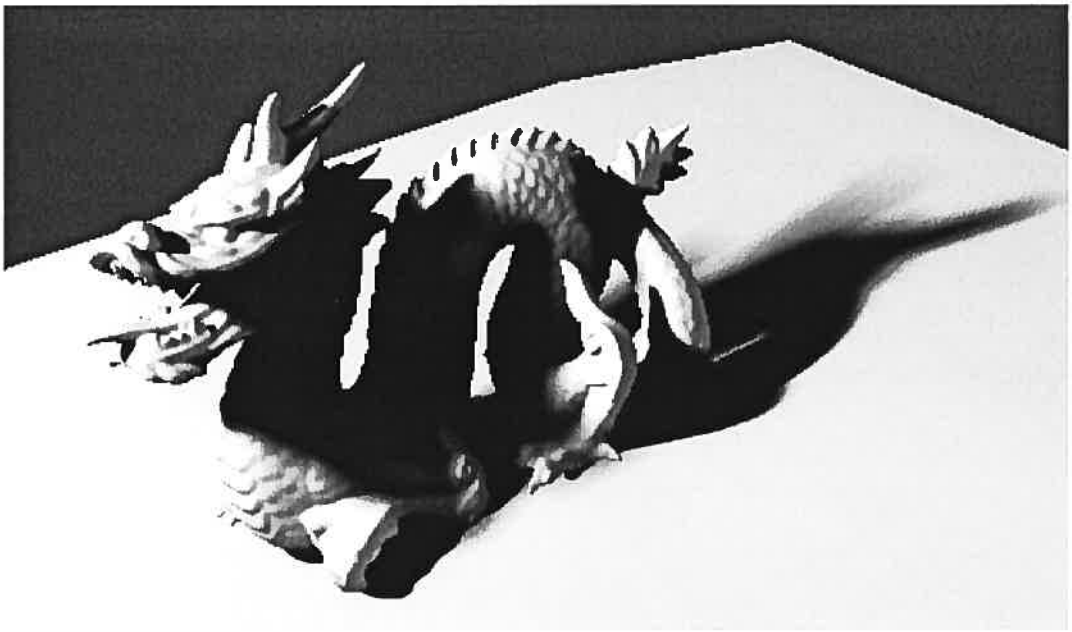


FIG. 3.6 – Rendu du modèle de Dragon de Stanford avec un *PDSM* de 512×512 et 256 échantillons.

Les fonctions d'atténuation du *PDSM* sont compressées d'une façon similaire à l'implantation originale du *DSM* [LV00]. Premièrement, une borne sur l'erreur ϵ est déterminée, de façon à limiter l'erreur causée par la compression (figure 3.7 (a)). Ceci permet de définir la sortie de l'algorithme comme étant V' tel que

$$|V'(z) - V(z)| \leq \epsilon \quad \forall z$$

où V est la fonction d'atténuation originale et où V' a typiquement beaucoup moins de sommets que V (figure 3.7 (d)).

L'idée générale est qu'à chaque étape, l'algorithme trace le segment de ligne le plus long possible sans sortir des bornes d'erreur. L'origine du segment est fixe et nous n'avons qu'à choisir la direction et la longueur du segment.

Soit (z'_i, V'_i) l'origine du segment courant (de sortie). À chaque pas, nous gardons l'intervalle de pentes permissibles $[m_{lo}, m_{hi}]$ pour le segment. Chaque nouveau point

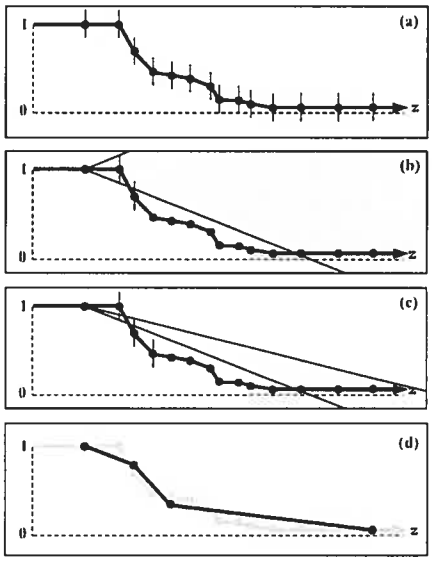


FIG. 3.7 – Algorithme de compression original de Lokovic et Veach [LV00]. (a) Courbe linéaire par morceaux avec sa borne sur l'erreur. (b) Chaque sommet de la fonction restreint la pente du prochain segment. (c) L'interval de pente courante (définie par ϵ) est intersectée avec l'intervalle de pente acceptable courant, jusqu'à ce que cette intersection donne l'ensemble vide. (d) Le segment courant est défini jusqu'au z courant avec une pente égale à la médiane de l'intervalle de pente courante. Le processus est ensuite répété.

de contrôle (z_j, V_j) de la fonction en entrée impose une contrainte sur l'intervalle de pentes courant, en forçant le segment à traverser la fenêtre (segments verticaux dans la figure 3.7) définie par les deux points $(z_j, V_j \pm \epsilon)$. L'intervalle initial de pentes est initialisé à $[-\infty, \infty]$ et intersecté avec chaque fenêtre jusqu'à ce que l'intersection devienne vide (figure 3.7 (c)). Nous traçons ensuite un segment de ligne avec la pente $(m_{lo} + m_{hi})/2$, se terminant au z du dernier point de contrôle visité (z_{j-1}). La fin de ce segment devient l'origine du suivant et l'algorithme est ensuite répété.

Quelques modifications sont cependant nécessaires pour adapter cette technique au *penumbra deep shadow map*. Comme nos fonctions d'atténuation peuvent avoir de

Nombre		*PDSM*		
de *shadow maps*	non compressé	2%	5%	10%
4	4.6 MB	4.6 MB	1.6 MB	1.3 MB
16	9.1 MB	5.3 MB	1.6 MB	1.3 MB
36	12.1 MB	5.5 MB	1.8 MB	1.4 MB
64	15.2 MB	5.6 MB	1.9 MB	1.4 MB
256	28.0 MB	5.1 MB	1.9 MB	1.4 MB

TAB. 3.1 – Compression des fonctions d'atténuation d'un *PDSM* de 512 × 512 pour différents nombres d'échantillons et différents pourcentages d'erreur permis (scène du Dragon de Stanford, voir figure 3.6).

Nombre		*PDSM*		
de *shadow maps*	non compressé	2%	5%	10%
4	16.3 MB	12.2 MB	5.1 MB	4.9 MB
16	36.8 MB	15.8 MB	5.3 MB	4.9 MB
36	47.0 MB	16.2 MB	5.4 MB	5.1 MB
64	60.6 MB	16.4 MB	5.5 MB	5.2 MB
256	103.9 MB	16.2 MB	5.5 MB	5.2 MB

TAB. 3.2 – Compression des fonctions d'atténuation d'un *PDSM* de 512 × 512, pour différents nombres d'échantillons et différents pourcentages d'erreur permis (scène Nature, voir figure 4.2 (b)).

Nombre de	Résolution *PDSM*		
shadow maps	256×256	512×512	1024×1024
4	< 1	2	9
16	3	10	37
36	6	21	77
64	11	37	133
256	43	147	529

TAB. 3.3 – Temps de construction (incluant la compression), en secondes, de *PDSM* à différentes résolutions et pour différents nombres d'échantillons (*SM* de 512×512) (scène du Dragon de Stanford, voir figure 3.6).

plus grandes pentes que celles de l'article original du *deep shadow map* [LV00], le fait d'orienter ϵ (l'erreur) directement dans l'axe de V peut causer des fenêtres trop petites, diminuant l'efficacité de l'algorithme en le forçant à insérer des points qui seraient ignorés si la pente était moins grande. Ce problème peut être réduit en orientant ϵ dans une direction perpendiculaire au segment joignant les deux points étant traités, et non seulement dans la direction de l'*atténuation* comme dans Lokovic et Veach [LV00]. Ce changement modifie le ϵ donné plus haut, mais en pratique, la différence est minime. Les tableaux 3.1, 3.2, ainsi que la figure 3.8, montrent les résultats de notre algorithme de compression. Avec 256 échantillons et une erreur faible, l'algorithme de compression fonctionne légèrement mieux que pour un nombre d'échantillons plus faible. Ceci est probablement dû à une réduction du *bruit* dans la fonction d'atténuation.

3.3 Détails d'implantation

3.3.1 Temps de calcul et consommation mémoire

La construction d'un *penumbra deep shadow map* peut être une opération très coûteuse. L'augmentation de la complexité de la scène, de la taille de la lumière, de la taille du *deep shadow map* et des *shadow maps* ainsi que du nombre d'échantillons demandé peuvent rapidement faire grimper le temps de pré-calcul et l'utilisation mémoire de l'algorithme (voir tableaux 3.3, 3.4 et 3.5). Bien qu'il puisse être assez élevé (voir tableau 3.3) [‡], le temps processeur n'est jamais prohibitif [§]. Par contre, la colonne de gauche (non optimisé) du tableau 3.6 montre bien que l'utilisation mémoire peut rapidement devenir problématique. Ces chiffres montrent l'utilisation mémoire d'une implantation naïve de l'algorithme de construction. Dans cette approche, nous ajoutons la contribution de chaque échantillon à tous les rayons du *PDSM* en une seule étape, comme dans l'algorithme 3.1. Ceci est en fait l'approche idéale, car elle ne requiert le calcul de chaque *shadow map* qu'une seule fois. Cependant, il est nécessaire d'attendre que *tous* les *shadow maps* aient été ajoutés au *PDSM* avant de pouvoir le compresser. Comme pour des scènes complexes et-ou des nombres élevés d'échantillons, le nombre d'événements par pixel du *PDSM* peut être très élevé (de l'ordre de plusieurs milliers), cette approche est inutilisable pour des scènes arbitrairement complexes et des tailles élevées de *PDSM*.

La technique à l'autre extrémité du spectre serait de traiter chaque rayon du *PDSM* séparément, *i.e.,* d'ajouter la contribution de tous les *shadow maps* à un seul pixel à la fois, de le compresser, puis de passer au pixel suivant du *PDSM*. Cette approche est en fait la plus économique en mémoire, car elle ne requiert qu'assez de mémoire

[‡]Tous les tests de construction de ce chapitre ont été effectués sur un PC avec deux processeurs Intel Xeon 2.4GHz, 1Go de RAM et une carte vidéo nVidia GeForce 4 Ti 4200. Il est à noter que ces processeurs utilisent la technologie *Hyperthread* et apparaissent donc au système comme deux processeurs chacun. Sauf lorsque spécifié autrement, tous les tests ont été effectués en mode parallèle.

[§]Il ne se compte jamais dans l'ordre de jours, semaines ou années.

Résolution *shadow map*	Temps de construction
128 × 128	55
256 × 256	72
512 × 512	147
1024 × 1024	217
2048 × 2048	554

TAB. 3.4 – Temps de construction, en secondes, d'un *PDSM* de 512 × 512 avec 256 échantillons de différentes résolutions (scène du Dragon de Stanford, voir figure 3.6).

Taille de la lumière	Temps de construction
1	67
2	88
3	109
4	137
5	146
6	162

TAB. 3.5 – .

Temps de construction, en secondes, d'un *PDSM* de 512 × 512 avec 256 échantillons (512 × 512), pour différentes tailles de source de lumière (scène du Dragon de Stanford, voir figure 3.6). Pour référence, le dragon a une taille d'environ 6 unités.

Nombre	*PDSM*	
shadow maps	non optimisé	optimisé
4	19 Mo	6 Mo
16	57 Mo	17 Mo
36	125 Mo	35 Mo
64	208 Mo	59 Mo
256	558 Mo	126 Mo

TAB. 3.6 – Utilisation maximale de mémoire, en mégaoctets, de la construction d'un *PDSM* de 256 × 256 et pour différents nombres d'échantillons (256 × 256) (scène du Dragon de Stanford, voir figure 3.6).

pour pouvoir stocker un seul pixel non compressé du *PDSM* et non *tous* les pixels. Évidemment, cette technique est très inefficace car elle demande de recalculer les *shadow maps* pour chaque pixel traité. Par exemple, pour un *PDSM* de 1024 × 1024 avec 1024 échantillons, cette technique demande le calcul de plus d'un milliard de *shadow maps*, alors que la technique naïve, gourmande en mémoire, n'en demande que 1024.

Un équilibre entre les deux techniques est donc souhaitable. Idéalement, nous voudrions construire le plus de pixels du *PDSM* possible à la fois, sans dépasser les capacités mémoire du système et en minimisant le nombre de *shadow maps* à calculer. Comme il est difficile de déterminer le nombre précis de pixels que nous pouvons traiter avec une quantité finie de mémoire, nous découperons le *PDSM* en un nombre arbitraire de parties, assez petites pour qu'il soit certain que la quantité de mémoire soit suffisante, mais les plus grosses possibles pour minimiser le nombre de *shadow maps* à recalculer. L'algorithme 3.2 illustre les différences avec l'algorithme original, les lignes différentes étant marquées par des flèches. Comme le montre le tableau 3.6, cette approche réduit substantiellement la quantité de mémoire nécessaire à la construction, au prix d'une légère augmentation en coût de calcul (tableau 3.7). Pour un nombre élevé d'échantillons, les résultats semblent même suggérer que la gestion mémoire de la version non opti-

Nombre de *shadow maps*	*PDSM* non optimisé	*PDSM* optimisé
4	6.3	6.9
16	16.7	17.3
36	34.6	39.9
64	63.7	65.3
256	325.3	259.8

TAB. 3.7 – Temps de construction, en secondes, d'un *PDSM* de 256 × 256 et pour différents nombres d'échantillons (256 × 256) (scène du Dragon de Stanford, voir figure 3.6).

misée devient plus coûteuse que le léger calcul supplémentaire de la version optimisée. Ces résultats ont été obtenus en divisant le *PDSM* en tranches de 128 rangées de pixels, un choix adéquat pour nos scènes de tests.

3.3.2 Implantation parallèle

Avec les chiffres du chapitre précédent, il est clair que l'algorithme peut devenir assez coûteux en temps CPU, surtout avec une grande résolution pour le *penumbra deep shadow map* et un nombre élevé d'échantillons sur la source de lumière. Heureusement, la nature de l'algorithme le rend facilement parallélisable. En effet, lors de l'ajout d'un échantillon (*i.e.,* de l'ajout de la contribution d'un *shadow map*), il est possible de traiter chaque rayon indépendamment. Comme les rayons ne dépendent pas l'un de l'autre, il est possible d'effectuer la *scan-conversion* de plusieurs en parallèle, *i.e.,* sur différents processeurs. De façon à minimiser le faible coût encouru lors de l'utilisation de plusieurs *threads*, nous avons choisi de faire le découpage au niveau des *rangées* de pixels du *PDSM*. Chaque rangée est donc traitée indépendamment des autres, sur le premier processeur disponible. Comme chaque *thread* n'utilise le *shadow map* qu'en lecture,

Algorithme 3.2 : Algorithme optimisé de génération du *PDSM*.

```
Génération de points aléatoires sur la source de lumière.
→ pour chaque tranche du PDSM faire
    pour chaque point (échantillon) faire
        Calcul d'un shadow map.
        // Insertion de l'information du SM dans le PDSM.
→       pour chaque pixel contenu dans la tranche faire
            Calcul du rayon 3D associé.
            Projection de ce rayon dans le SM.
            pour chaque pixel du SM traversé par le rayon faire
                si changement de visibilité alors
                    Insertion d'un événement dans le PDSM.
→   Compression des pixels de la tranche du PDSM.
```

il n'y a pas de problème de blocage d'accès. De plus, comme nous traitons les rayons séquentiellement, il est fort probable que les *threads* accèdent aux mêmes régions en mémoire, favorisant une utilisation efficace de la mémoire tampon du processeur. Le tableau 3.8 montre l'amélioration importante (de l'ordre de 100% à 125%) qu'une implantation parallèle apporte à notre algorithme[¶].

[¶]Bien que surprenante, l'amélioration de parfois plus de 100% s'explique par la technologie Hyper-thread présente dans nos processeurs.

Nombre de *shadow maps*	*PDSM* 1 CPU	*PDSM* 2 CPU
4	5	2
16	30	10
36	43	21
64	76	37
256	293	147

TAB. 3.8 – Temps de construction, en secondes, d'un *PDSM* de 512 × 512 et pour différents nombres d'échantillons (512 × 512) (scène du Dragon de Stanford, voir figure 3.6).

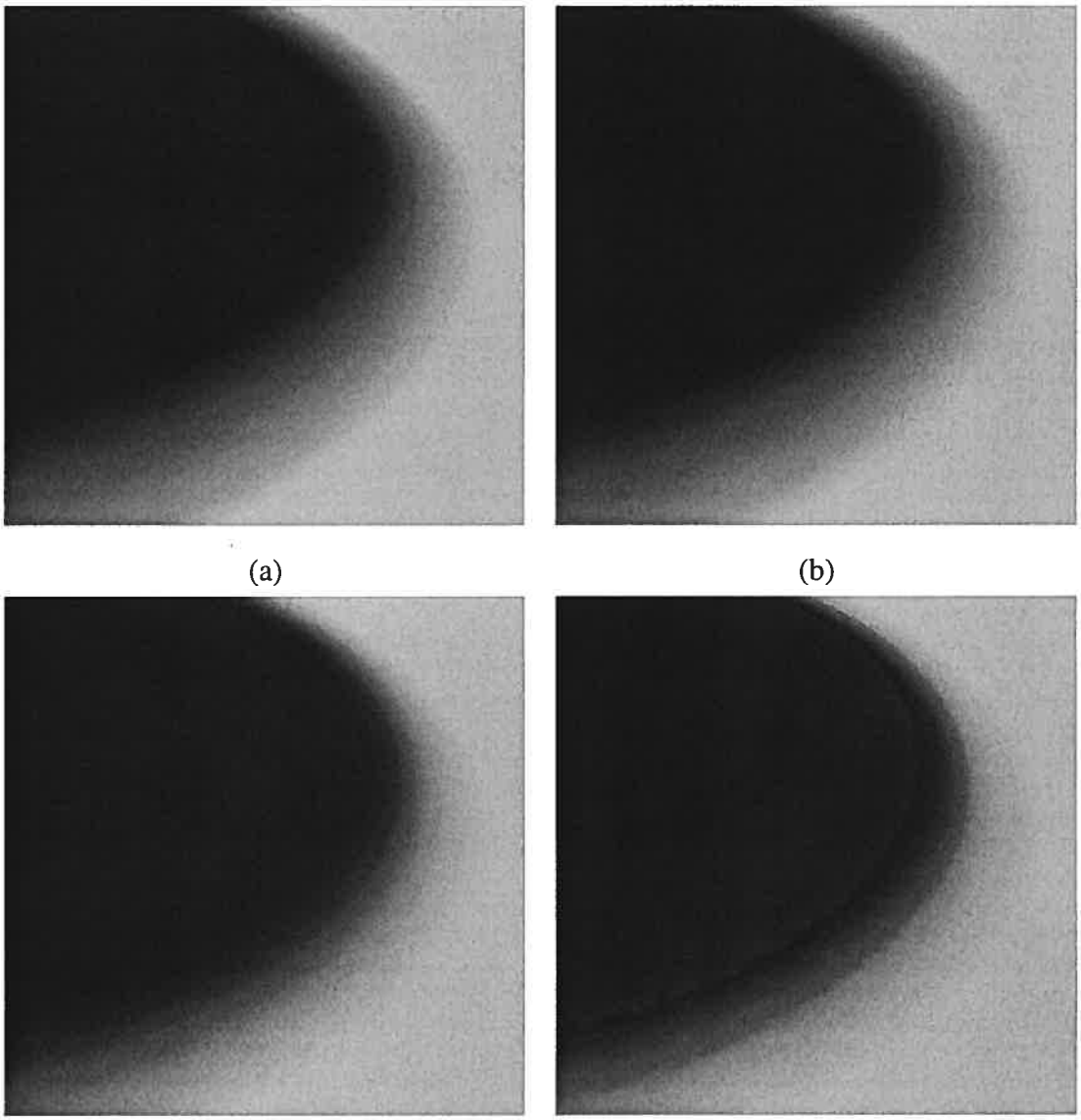


FIG. 3.8 – Impact visuel des différents niveaux de compression : (a) Aucune compression, (b) 2% d'erreur alloué, (c) 5% et (d) 10%. La différence est particulièrement notable aux frontières de la pénombre dans les deux dernières images.

Chapitre 4

Rendu à l'aide de PDSM

Drawing is the honesty of the art. There is no possibility of cheating. It is either good or bad.

Salvador Dali

Dans la plupart des moteurs de rendu, temps réel ou hors-ligne (*offline*), les ombres sont calculées indépendamment du *shading* et ensuite utilisées pour moduler ce dernier*. En adoptant cette stratégie, nous pouvons aisément ajouter des ombres à une scène. En effet, la structure du *deep shadow map* a été adaptée spécifiquement pour sa capacité à nous fournir le pourcentage d'atténuation de tous les points 3D d'une scène†. Une simple projection dans l'espace 3D du *penumbra deep shadow map* nous donne directement le facteur d'atténuation pour le point (algorithme 4.1).

Nous proposons deux approches pour le rendu à l'aide d'un *penumbra deep shadow map* : une approche logiciel générale et une approche accélérée à l'aide du matériel graphique, plus limitée mais beaucoup plus rapide.

*Cette approche, bien qu'approximative et parfois fausse pour certaines configurations, donne dans la grande majorité des cas de bons résultats et est normalement jugée appropriée.

†En fait, on parle ici de tous les points compris dans la pyramide de vue du *penumbra deep shadow map*.

Algorithme 4.1 : Survol de l'algorithme de rendu avec *PDSM*.

pour chaque *point 3D à illuminer* **faire**

Shading standard (détermination de la couleur du point).

Transformation du point dans l'espace 3D du *PDSM*.

Recherche du facteur d'atténuation associé avec la profondeur du point dans le *PDSM*.

Modulation de la couleur du point par ce facteur d'atténuation.

fin

4.1 Approche générale de rendu

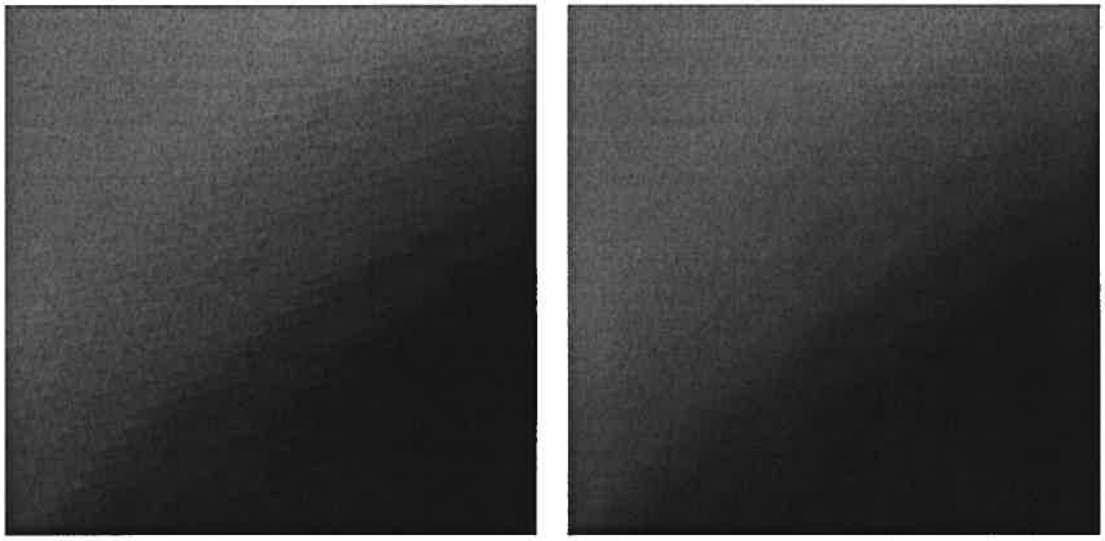
Comme dans l'algorithme original du *DSM*, il est trivial d'ajouter des ombres à l'aide du *PDSM* dans un système de rendu hors-ligne ayant déjà un support pour les *shadow maps* de base, par exemple dans les moteurs de rendu *RenderMan* [Ups89] (dans lequel les *DSM* sont d'ailleurs implantés) et *Mental Ray* [Ray04]. En effet, les coordonnées de texture à utiliser pour une structure du type *DSM* sont les mêmes que pour l'algorithme du *shadow map* original[‡]. Dans un contexte de *DSM*, la composante z de la coordonnée de texture ne sera cependant pas utilisée pour une simple comparaison de profondeur, mais bien comme paramètre pour évaluer la fonction d'atténuation contenue au pixel (x, y). L'évaluation de cette fonction donne le facteur d'atténuation, qui est ensuite multiplié avec la couleur du point courant[§].

Comme pour les textures ordinaires (et les *shadow maps*), il est aussi possible d'effectuer du filtrage 2D sur un *PDSM*. Il suffit de filtrer les résultats de l'évaluation de la fonction d'atténuation aux différents pixels contenus dans le filtre. Comme pour les *shadow maps*, il est important de filtrer les résultats et non les profondeurs [RSC87]. Dans cette approche, tous les types et grosseurs de filtres sont permis. Cependant, comme l'évaluation de la fonction d'atténuation peut être assez coûteuse pour des pixels de

[‡]Rappel : Ces coordonnées de texture sont en fait la position du point 3D à texturer, mais exprimée dans l'espace du *shadow map* et non du point de vue de la caméra (voir la section 2.4.1).

[§]En fait, on multiplie l'illumination sans le terme ambiant par (1 – facteur d'atténuation).

PDSM contenant des fonctions complexes, il est préférable de restreindre la taille du filtre. C'est pour cette raison que nous utilisons un simple filtrage bilinéaire, avec de bons résultats (voir la figure 4.1). Évidemment, l'ajout du filtrage vient au coût d'un temps de rendu plus élevé (voir le tableau 4.1).



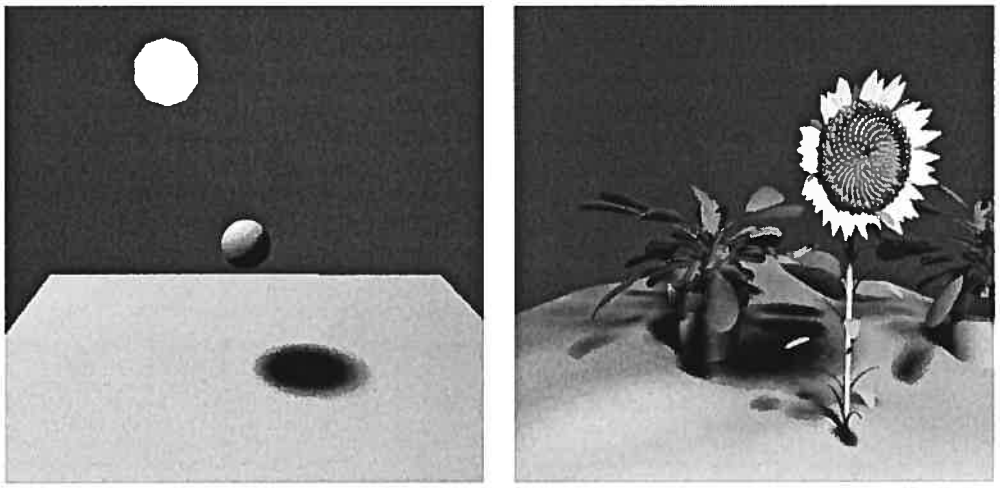
(a) (b)

FIG. 4.1 – Comparaison des résultats (a) sans filtrage et (b) avec filtrage bilinéaire.

Nous avons simulé un moteur de rendu hors-ligne à l'aide de *OpenGL*. En effet, après un rendu d'image (sans ombres) à l'aide de *OpenGL*, nous avons le même type d'information qui nous serait disponible lors d'un rendu hors-ligne. Nous avons la couleur du point auquel nous voulons ajouter de l'ombre et nous avons sa position (x, y, z) en coordonnée image[¶]. En construisant nous-mêmes la matrice de projection de texture, nous pouvons obtenir les coordonnées du point dans l'espace du *PDSM* et donc le facteur d'atténuation. Il suffit ensuite de multiplier la couleur courante du pixel par ce facteur, et ce pour tous les pixels. Cette approche, bien que plus lente qu'un moteur sophistiqué comme *RenderMan*, donne une bonne idée des résultats possibles, tels

[¶]Les composantes x et y sont simplement les coordonnées du pixel à moduler et la coordonnée z est obtenue en lisant le tampon de profondeur du *framebuffer*.

qu'illustrés dans la figure 4.2.



(a) (b)

FIG. 4.2 – Exemple des résultats possibles avec notre technique (*PDSM* de 1024×1024, 256 échantillons et filtrage bilinéaire).

4.2 Approche de rendu accéléré par matériel graphique

L'approche de rendu accéléré par *graphics processing unit (GPU)* est très similaire à l'approche générale (logiciel) décrite ci-haut. En fait, les deux approches sont conceptuellement identiques. Elles utilisent toutes deux la technique de texture projective pour ensuite évaluer la fonction d'atténuation du pixel du *PDSM* correspondant. Deux différences importantes sont toutefois à noter : le *PDSM* doit être stocké sur le *GPU* et nous devons ensuite évaluer la fonction d'atténuation, toujours sur le *GPU*.

4.2.1 Stockage du PDSM sur GPU

Pour avoir accès à l'information du *PDSM* lors d'un rendu accéléré, cette dernière doit se trouver sur le *GPU*. Pour se faire, nous encodons le *PDSM* (une structure 3D)

dans plusieurs textures 2D qui sont beaucoup plus communes. Une texture 3D aurait aussi été une option viable, mais pour être efficace, le système où roule l'algorithme doit supporter plusieurs extensions très récentes à *OpenGL*, d'où le choix des textures 2D. Chaque fonction d'atténuation est encodée comme un tableau de paires (profondeur, atténuation), les profondeurs avec 16 bits de précision, les atténuations avec 8 bits. Pour minimiser le nombre de textures nécessaire, les récentes instructions de *compactage* sur *GPU* sont utilisées. Ces dernières permettent l'encodage de deux nombres flottants de 16 bits dans un format 32 bits IEEE, ou de quatre entiers de 8 bits dans ce même format 32 bits. En utilisant le format de texture *RGBA32* (4 canaux de 32 bits chacun), il est donc possible de stocker en mémoire texture un *PDSM* d'une profondeur maximale de 16 événements avec seulement 3 textures[‖].

4.2.2 Évaluation de la fonction d'atténuation sur GPU

Le matériel graphique étant très spécialisé, nous sommes limités à des approches très simples pour évaluer la fonction d'atténuation. En effet, jusqu'à très récemment, les programmes pour *GPU* ne supportaient pas le branchement conditionnel et étaient limités à un faible nombre d'instructions assembleurs (de 96 à 1024).

Nous avons choisi d'évaluer la fonction d'atténuation par section. Il est d'abord nécessaire de charger tous les sommets de la fonction. Ensuite, nous évaluons dans quel segment de la fonction se trouve la composante z de la coordonnée de texture projective. Une fois le segment trouvé, nous interpolons linéairement entre les valeurs d'atténuation des deux sommets. Ceci nous donne directement le facteur d'atténuation, qui sera multiplié avec la couleur du pixel (*fragment* dans la nomenclature *GPU*) avant son écriture dans le *framebuffer*.

[‖]Dans un texel RGBA32 de 128 bits, nous pouvons stocker 8 profondeurs de 16 bits ou 16 facteurs d'atténuation de 8 bits. Pour équilibrer, nous combinons deux textures de profondeurs avec une texture d'atténuation.

	Logiciel		GPU	
Scène	non filtré	bilinéaire	non filtré	bilinéaire
Sphère	2.7 fps	0.9 fps	72.3 fps	24.5 fps
Plante	1.9 fps	0.6 fps	53.0 fps	18.3 fps

TAB. 4.1 – Vitesse de rendu pour différentes scènes (*PDSM* de 512 × 512 avec 256 échantillons). Obtenu sur un PC avec un processeur AMD Athlon 64 3200+, 1.5Go de RAM et une carte vidéo nVidia GeForce 6800GT.

4.2.3 Vitesse et qualité du rendu

Comme le montre le tableau 4.1, la version accélérée de l'algorithme de rendu est beaucoup plus rapide que la version logicielle jusqu'à plus de 25 fois plus rapide. Cette rapidité a toutefois un prix, non pas directement dans la qualité des images, mais plutôt dans la généralité des *PDSM* pouvant être utilisés. En effet, pour un *PDSM* n'excédant pas les capacités du *GPU*, la qualité des images rendues par l'algorithme accéléré sera pratiquement équivalente à celle de l'algorithme général (voir la figure 4.3 pour la comparaison).

4.2.4 Limitations et extension possibles

La version sur *GPU* de l'algorithme impose les limitations suivantes :

- la taille en (x, y) du *PDSM* ne peut dépasser la taille maximale des textures 2D supportées par le *GPU* (présentement 4096×4096 sur la plupart des cartes). Pour la plupart des scènes, cette taille est adéquate, mais il s'agit tout de même d'une limite potentielle.
- la profondeur maximale (en z) du *PDSM* est aussi limitée par le nombre de textures disponibles. Ce nombre est normalement limité à 8 ou 16 sur les *GPU* courants, ce qui limite la profondeur maximale de notre *PDSM* à 32 ou 80 événements respectivement (*i.e.*,, avec 6 et 12 textures pour le *PDSM*). Bien que nos

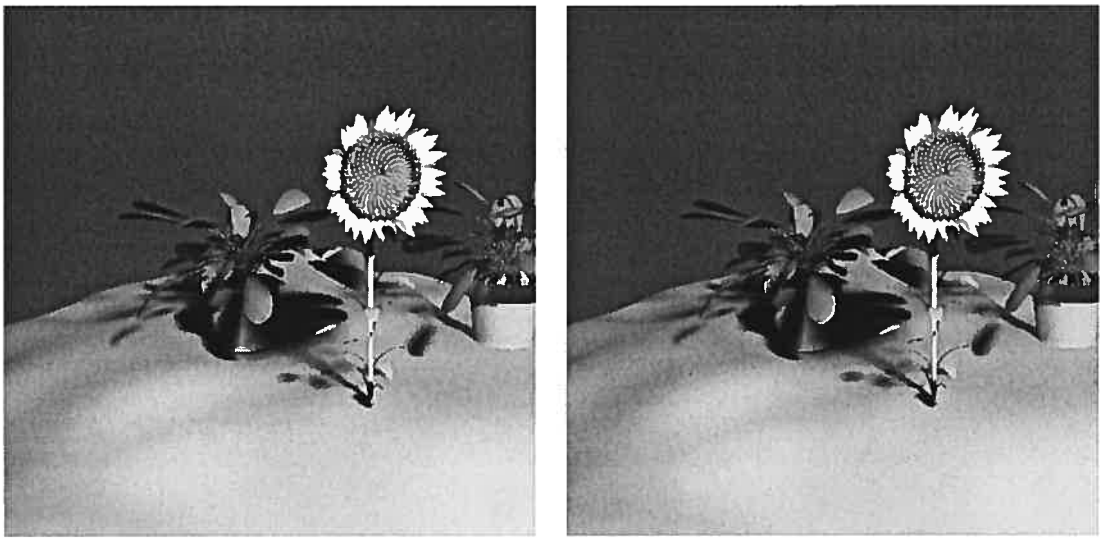


FIG. 4.3 – Comparaison des résultats de l'algorithme général (gauche) et de l'algorithme accéléré (droite).

scènes de test n'aient jamais dépassées la barre des 32 événements, il est facile d'imaginer des scènes où cela se produirait. Dans ce cas, une approche multi-passes pourrait être adoptée.

- le type de filtre utilisé pendant le rendu, ainsi que sa largeur, est limité par la longueur maximale des *programmes fragments* du *GPU*. De plus, comme dans le cas de la technique logicielle, les performances décroissent rapidement avec l'élargissement du filtre et un filtrage bilinéaire reste le meilleur compromis entre qualité et performance.

Deux problèmes plus sérieux de l'algorithme sur *GPU* se présentent cependant. Premièrement, le stockage d'un *PDSM* dans une structure 3D à taille fixe dans les trois dimensions cause un gaspillage de mémoire potentiellement énorme. En effet, que ce soit avec une texture 3D ou de multiples textures 2D, nous devons garantir que la structure soit assez profonde pour contenir la plus longue des fonctions d'atténuation du *PDSM*. Comme la plupart des pixels du *PDSM* contiennent des fonctions vides ou avec un faible nombre d'événements, une grande partie de la mémoire allouée pour

ces pixels est gaspillée. Par exemple, pour la figure 4.3, la fonction la plus complexe contient 31 sommets, nous obligeant donc à utiliser 6 textures pour stocker le *PDSM*, pour un total de mémoire texture de 24 Mo. Le problème se situe avec la moyenne de sommets par fonction dans le *PDSM*, qui est de 2,3. Nous n'avons donc réellement besoin que d'approximativement 5,5 Mo de mémoire pour stocker le *PDSM* (résultat provenant de la figure 3.2).

Le problème du gaspillage de mémoire pourrait être résolu en encodant le *penumbra deep shadow map* sous forme linéaire. L'approche qui semble la plus prometteuse serait d'avoir une structure à deux niveaux :

1. une première texture, de la taille du *PDSM*, qui ne contient en fait que l'information d'indexation, *i.e.*, l'index du début de la liste de sommets (dans une autre texture) et le nombre de sommets à lire.
2. une deuxième texture contenant les sommets de toutes les fonctions d'atténuation, encodés séquentiellement. Dans le cas de *PDSM* de grandes tailles, il est possible d'imaginer plus d'une texture de ce type. L'information d'indexation se devrait donc aussi d'inclure le numéro de texture.

Cette approche élimine pratiquement le gaspillage, en n'allouant pour chaque fonction que l'espace nécessaire. Il est possible que de l'espace soit perdu dans l'éventualité où il est impossible d'allouer une texture 2D contenant exactement le bon nombre de texels (*e.g.*, si nous avons besoin de 163 texels, nous serons obligé d'allouer une texture de 4×41, gaspillant ainsi 1 texel). Ce gaspillage est évidemment minime, surtout par comparaison au gaspillage de la méthode courante. De plus, cette approche impose une limite sur le nombre total d'événements dans le *PDSM*, mais pas sur le nombre d'événements dans une fonction d'atténuation, comme c'est présentement le cas.

Le deuxième problème se trouve dans l'évaluation des fonctions d'atténuation, qui est inefficace sur *GPU*. Comme mentionné précédemment, l'évaluation des fonctions d'atténuation se fait en trouvant d'abord le segment de fonction approprié, puis en interpolant linéairement entre ses deux sommets. Malheureusement, comme la plupart des

GPU ne supportent pas le branchement conditionnel, tous les segments devront être lus et testés, même si le bon segment a déjà été trouvé. L'arrivée récente de *GPU* supportant le *Shader Model 3.0* de Microsoft, qui impose le support de réel branchement conditionnel, aide. La carte vidéo nVidia GeForce 6800GT utilisée pour nos tests est une de ces cartes. Nos données sont encore préliminaires, mais l'utilisation des nouvelles instructions de branchement conditionnel, par rapport aux anciennes instructions simulant le branchement conditionnel, donne un gain de performance d'environ 50%.

4.3 Ajout d'objets dynamiques

Comme nous l'avons mentionné précédemment, le fait de pouvoir évaluer la fonction d'atténuation pour tout point 3D de l'espace nous permet de projeter des ombres non seulement sur les objets présents lors de la construction du *penumbra deep shadow map*, mais aussi sur tout objet dynamique ajouté à la scène. Il est à noter que les objets dynamiques ne projetteront pas d'ombres automatiquement, comme illustré par la figure 4.4. Bien qu'il serait possible de recalculer le *PDSM* avec la position de ces nouveaux objets, ou même de les ajouter de façon incrémentale à deux *PDSM* (un statique et l'autre dynamique), le temps requis pour recalculer le *PDSM* est significatif par rapport au temps de rendu. Comme le nombre d'objets dynamiques dans une scène 3D est souvent restreint (*e.g.,* les caractères animés d'un jeu vidéo), une des méthodes de génération d'ombre en temps réel discutées au chapitre 2 serait plus appropriée, en particulier les techniques d'ombres floues basées sur les tampons de profondeur [CD03, WH03] et celles basées sur les volumes d'ombre [AMA02, AAM03]. Il est à noter qu'enlever un objet au *PDSM* est un problème beaucoup plus difficile.

4.4 Autres résultats

Comme il n'est pas facile de voir les éléments plus subtils des ombres dans un format imprimé, la figure 4.6 montre plusieurs gros plans sur l'ombre de la figure 4.5. Il est possible d'y remarquer les effets d'une variation dans la résolution du *PDSM* et dans le nombre d'échantillons. La figure 4.7 illustre l'effet du changement de taille de lumière.

FIG. 4.4 – Insertion d'un objet dynamique (la voiture) après la construction du *PDSM*. La voiture ne fait pas d'ombre puisqu'il s'agit d'un objet dynamique et qu'il ne fait donc pas partie du *PDSM*.

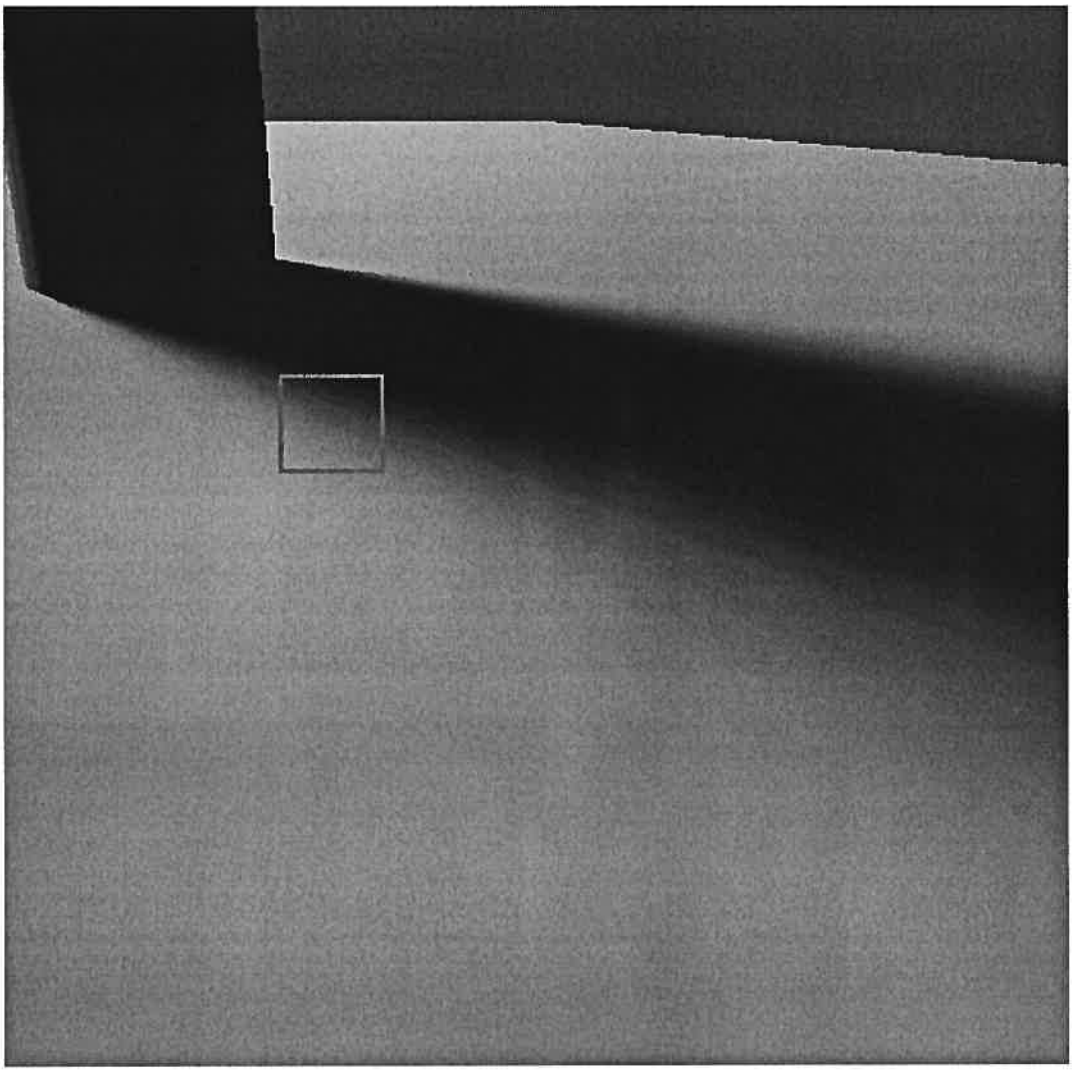


FIG. 4.5 – Rendu d'un cylindre (*PDSM* de 512 × 512 avec 256 échantillons). Le rectangle rouge est grossi dans la figure 4.6.

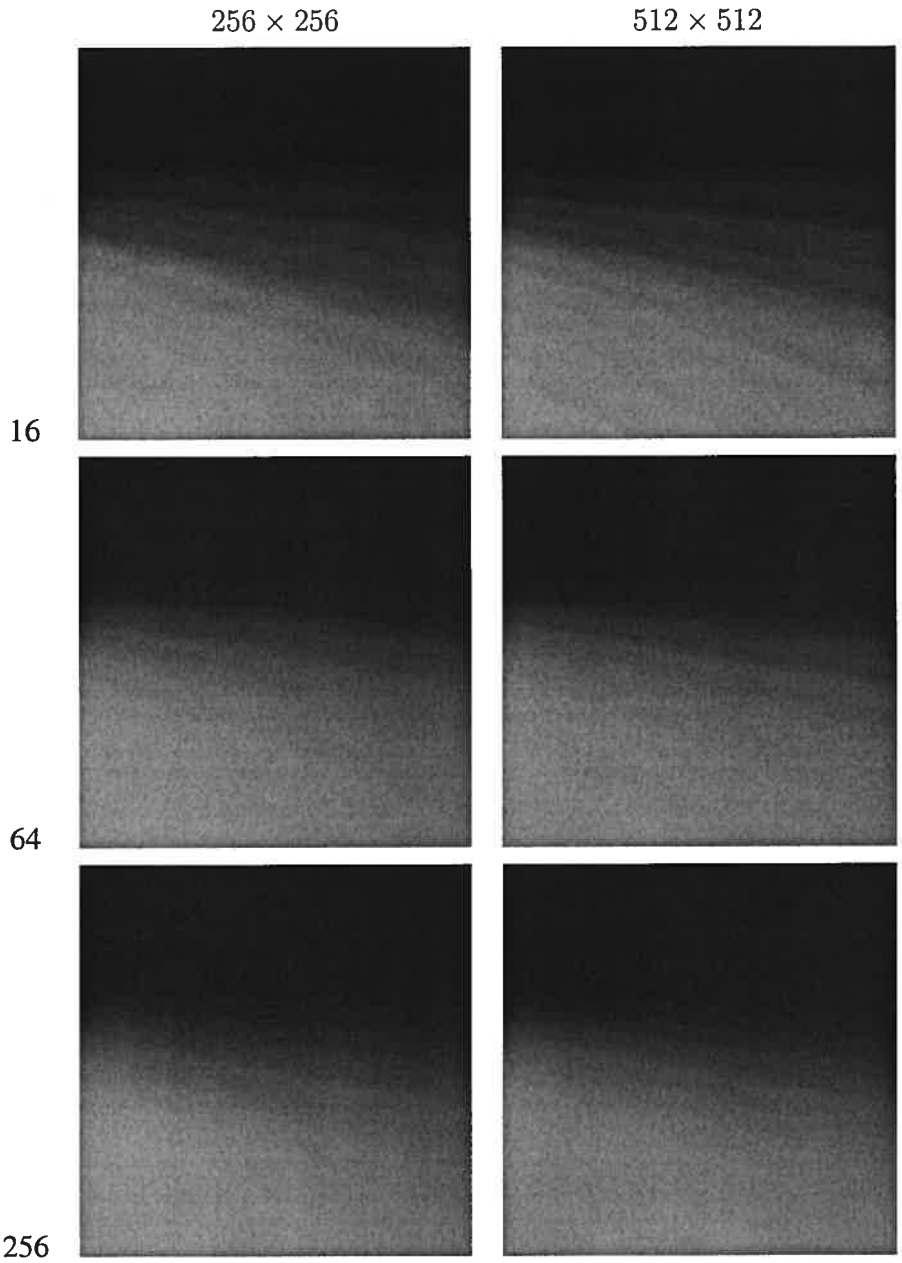


FIG. 4.6 – Résultats pour différentes résolution de *PDSM* (non compressés) et différents nombres d'échantillons. Colonne de gauche : *PDSM* de 256 × 256, à droite : 512 × 512. De haut en bas : 16 échantillons, 64 échantillons et 256 échantillons (se référer à la figure 4.5 pour l'image complète).

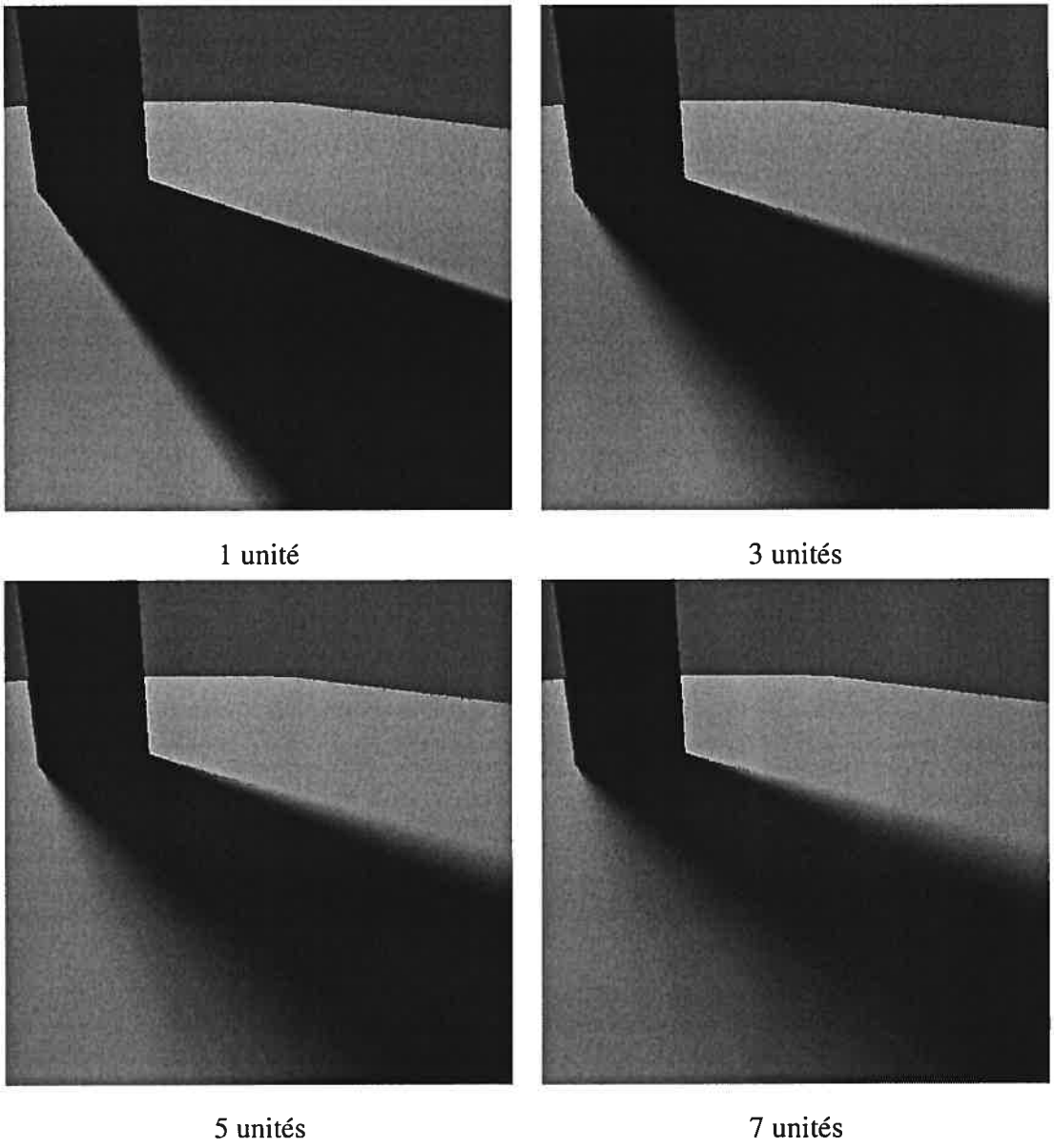


FIG. 4.7 – Résultats pour différentes tailles de source de lumière (*PDSM* de 512×512, 256 échantillons). Le cylindre a un rayon de 1 unité.

Chapitre 5

Conclusion

A conclusion is the place where
you got tired of thinking.

Martin H. Fischer

Nous avons présenté une méthode de qualité, mais toutefois efficace, de construction de *deep shadow maps* contenant l'information d'atténuation provenant de sources de lumière étendues, représentées par une distribution de sources de lumière ponctuelles. Chaque rayon du *PDSM* est tracé dans le *shadow map* de chaque source de lumière ponctuelle pour accumuler la distribution du facteur d'atténuation le long du rayon. La structure de type *DSM* résultante capture des ombres de haute qualité, autant dures que floues, pour des sources de lumière de tailles variables, allant d'une lumière ponctuelle à de larges sources étendues. De plus, nous avons proposé un algorithme de compression des données ainsi que des modifications à notre méthode de construction pour la rendre parallélisable et incrémentale.

Nous proposons deux méthodes de rendu avec l'aide cette structure. La première est une méthode générale, permettant une grande qualité d'ombres pour toute complexité de scène. Nous proposons cette approche pour les outils de rendu de haute qualité, *e.g.,* Maya de Alias, XSI de Softimage et 3D Studio Max de Discreet. Ils utilisent

normalement les options suivantes pour le traitement des ombres lors du rendu : *shadow maps* standards (avec ou sans *PCF*), lancer de rayons d'ombre ou intégralement dans le traitement de l'illumination globale avec le *photon map* [Jen01]. Notre approche se positionne bien, en temps de calcul et en qualité, entre l'algorithme du *shadow map* et celui du lancer de rayons. De plus, il pourrait même être utilisé dans des logiciels de rendu comme *RenderMan* [Ups89].

Grâce à la compression du *PDSM*, il est possible d'exploiter le matériel graphique moderne pour un rendu en temps interactif. Notre méthode est bien adaptée aux applications interactives, telles les jeux vidéos et les applications de type *walkthrough*, où des ombres floues de haute qualité peuvent être projetées sur des éléments animés (*e.g.,* personnages et véhicules). Une méthode complémentaire de génération d'ombre en temps réel pourrait être appliquée à ces éléments animés pour s'assurer que leurs ombres soient projetées sur la scène.

Notre méthode n'est évidemment pas parfaite. Le pré-calcul nécessaire à la construction du *PDSM* peut être assez long. De plus, l'évaluation de la fonction d'atténuation n'est possible que dans la pyramide de vue du *PDSM*. Bien qu'il soit possible d'augmenter le champ de vue (*field of view*) de ce dernier pour couvrir une plus grande partie de la scène, des artefacts peuvent apparaître si la résolution du *PDSM* n'est pas augmentée en conséquence. De plus, les objets dynamiques ne projettent pas d'ombres avec notre méthode, bien qu'il soit possible d'en ajouter avec une autre méthode interactive.

Il y a plusieurs directions intéressantes pour améliorer nos travaux. Le traçage des rayons du *PDSM* dans chaque *shadow map* pourrait beaucoup mieux exploiter la cohérence inhérente à une telle structure, comme par exemple de tracer des "tranches" de la pyramide de vue du *PDSM* plutôt que des rayons individuels. La compression des fonctions d'atténuation n'est aussi faite que sur un rayon à la fois, alors que l'utilisation des fonctions des pixels voisins ainsi qu'une base perceptuelle des ombres serait préférable. Il serait aussi intéressant d'exploiter les approches plus adaptatives des récentes extensions aux *shadow maps* [FFBG01, SD02, SCH03, WSP04, MT04] et de

développer un algorithme adaptatif d'échantillonnage de la source de lumière étendue.

Une avenue très intéressante à explorer serait la construction d'un *penumbra deep shadow map* de haute qualité, mais avec un nombre très restreint d'échantillons. En effet, il semble possible d'insérer un faible nombre de *shadow maps* à la structure et d'ensuite filtrer leurs contributions entre les différents rayons. Il y a évidemment plusieurs difficultés à cette approche, entre autre d'avoir assez d'échantillons (et-ou une fonction d'échantillonnage assez intelligente pour bien les placer) pour bien capturer toutes les discontinuités importantes de l'ombre. La simple reconnaissance de ces discontinuités est en soit un problème difficile. Une approche complètement opposée, mais peut-être plus facile, serait de construire un *PDSM* de très haute résolution et d'ensuite le filtrer dans un *PDSM* de plus petite taille, comme suggéré par Lokovic et Veach [LV00].

La structure même du *deep shadow map* est très prometteuse et pourrait sûrement être utilisée à d'autres fins. L'idée d'avoir une structure permettant le stockage et l'évaluation d'une fonction pour n'importe quel point 3D de l'espace est intéressante. Une fois qu'une fonction à évaluer est trouvée, il suffit de construire une structure du type *DSM* pour la stocker. Nous aimerions explorer la possibilité d'utiliser cette structure pour d'autres effets, comme le stockage d'illumination globale et en particulier des effets plus directionnels de caustiques.

Annexe A

Glossaire

Understanding reduces the greatest to simplicity, and lack of it causes the least to take on the magnitude.

Raymond Holliwell

Ce glossaire donne la définition de certains termes utilisés mais non définis dans le présent document, ainsi que des termes anglophones dont la traduction est inexistante ou peu utilisée.

Bump mapping Effet de perturbation des normales d'une surface, de façon à simuler l'illumination d'une surface avec relief.

Échantillonnage stratifié Méthode d'échantillonnage qui regroupe les échantillons en sous-groupes lors de l'échantillonnage. Dans notre cas, la lumière est divisée en régions de tailles égales, puis chacune est échantillonnée séparément.

Moteur graphique Partie centrale d'un programme, qui se charge traditionnellement uniquement (et exclusivement) de la création de l'image, soit à l'écran ou dans un fichier.

Extrusion Création d'un objet à partir d'un autre objet de base (souvent d'une dimension inférieure) et d'une direction ou d'un chemin à suivre par l'objet de base.

Fillrate Terme utilisé à plusieurs effets en infographie. Réfère généralement à la quantité de données pouvant/devant être traitées par la carte graphique. Ces données sont habituellement des fragments. Lorsqu'on dit d'une technique qu'elle demande beaucoup de fillrate, c'est qu'elle génère beaucoup de fragments.

Fragment Un fragment est ce qui est produit par le *rasterizer* à partir des triangles dans le pipeline graphique. Souvent confondu avec un pixel, ce qui est faux car plusieurs fragments seront probablement générés pour un seul pixel. Le fragment qui gagne la bataille du test en z sera celui qui sera affiché, ou dans le cas d'une image avec *blending*, la contribution de chacun des fragments sera accumulée pour produire la couleur finale du pixel.

Framebuffer Partie de la mémoire vidéo allouée pour contenir l'information d'une image. Peut être "à l'écran" et donc directement affiché sur le moniteur ou "hors écran".

Lumière directionnelle Lumière se trouvant conceptuellement à l'infini. Tous ses rayons sont donc parallèles. Le soleil est souvent approximé par ce type de lumière.

Lumière ponctuelle Lumière à aire nulle, représentée par un point dans l'espace, qui émet également dans toutes les directions.

Lumière surfacique Lumière à aire non nulle. Peut avoir n'importe quelle forme 2D ou 3D. C'est ce type de lumière, plus réaliste que les lumières directionnelles et ponctuelles, qui produit de la pénombre.

Programme fragment Sur les cartes graphiques programmables, il est possible de remplacer toute la partie traditionnelle du traitement des fragments par un programme de ce type.

Rasterizer Entre le stage des sommets et celui des fragments, il transforme les triangles en de multiples fragments. Se charge aussi d'interpoler les propriétés des sommets aux différents fragments.

Self-shadowing On considère qu'un objet se fait du *self-shadowing* lorsqu'une partie

de cet objet fait de l'ombre sur une autre partie du même objet.

Shading Tout le processus de détermination de la couleur d'un point. Dans notre cas, nous n'y incluons pas le calcul d'ombre, mais cette séparation n'est pas fondamentalement nécessaire (et est en fait une approximation).

Stencil buffer Un masque sur le *framebuffer* qui détermine si la couleur d'un pixel peut être mise à jour ou pas.

Walkthrough Application interactive où un à plusieurs usagers se trouvent dans un monde virtuel qu'ils visitent sans pour autant interagir avec le monde ou les autres usagers.

Z-buffer Réfère typiquement au tampon de profondeur associé au *framebuffer*. Contient la profondeur de l'élément présent dans chaque pixel.

Bibliographie

[AAM03] Ulf ASSARSSON et Tomas AKENINE-MÖLLER : A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics*, 22(3):511–520, juillet 2003.

[AMA02] Tomas AKENINE-MÖLLER et Ulf ASSARSSON : Approximate soft shadows on arbitrary surfaces using penumbra wedges. *In Eurographics Workshop on Rendering*, pages 297–306, juin 2002.

[AMH02] Tomas AKENINE-MÖLLER et Eric HAINES : *Real-time Rendering*. AK Peters, 2e édition, 2002.

[ARBJ03] Sameer AGARWAL, Ravi RAMAMOORTHI, Serge BELONGIE et Henrik Wann JENSEN : Structured importance sampling of environment maps. *ACM Transactions on Graphics*, 22(3):605–612, juillet 2003.

[ARHM00] Maneesh AGRAWALA, Ravi RAMAMOORTHI, Alan HEIRICH et Laurent MOLL : Efficient image-based methods for rendering soft shadows. *In Proc. of ACM SIGGRAPH 2000*, pages 375–384, juillet 2000.

[BB84] Lynne BROTMAN et Normand BADLER : Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics & Applications*, 4(10): 71–81, octobre 1984.

[CCOD+04] Yiorgos L. CHRYSANTHOU, Daniel COHEN-OR, Frédo DURAND, Cláudio T. SILVA, Andrew WOO et Pierre POULIN : *Visibility and Occlusion :*

Rendering Acceleration and Shadow Computation. Morgan Kaufman, In preparation, 2004.

[CD03] Eric CHAN et Frédo DURAND : Rendering fake soft shadows with smoothies. *In Eurographics Symposium on Rendering*, pages 208–218, juin 2003.

[CD04] Eric CHAN et Frédo DURAND : An efficient hybrid shadow rendering algorithm. *In Eurographics Symposium on Rendering*, pages 185–195, juin 2004.

[CHH02] Nathan A. CARR, Jesse D. HALL et John C. HART : The ray engine. *In Graphics Hardware 2002*, pages 37–46, septembre 2002.

[COCSD03] Daniel COHEN-OR, Yiorgos L. CHRYSANTHOU, Cláudio T. SILVA et Frédo DURAND : A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, jul-sep 2003.

[Cro77] Franklin C. CROW : Shadow algorithms for computer graphics. *In Proc. SIGGRAPH 77*, volume 11, pages 242–248, juillet 1977.

[CW93] Michael F. COHEN et John R. WALLACE : *Radiosity and Realistic Image Synthesis*. Academic Press Professional, 1993.

[EK02] Cass EVERITT et Mark KILGARD : Practical and robust stenciled shadow volumes for hardware-accelerated rendering. Rapport technique, nVidia Corporation, mars 2002.

[FFBG01] Randima FERNANDO, Sebastian FERNANDEZ, Kavita BALA et Donald P. GREENBERG : Adaptive shadow maps. *In Proc. SIGGRAPH 2001*, pages 387–390, août 2001.

[FvDFH90] James D. FOLEY, Andries van DAM, Steven K. FEINER et John F. HUGHES : *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, Reading, Massachusetts, 1990.

[Gla89] Andrew S. GLASSNER, éditeur. *An Introduction to Ray Tracing*. Academic Press, London, 1989.

[GLY+03] Naga GOVINDARAJU, Brandon LLOYD, Sung-Eui YOON, Avneesh SUD et Dinesh MANOCHA : Interactive shadow generation in complex environments. *In Proceedings of SIGGRAPH*, 2003.

[GWS04] Johannes GÜNTHER, Ingo WALD et Philipp SLUSALLEK : Realtime Caustics using Distributed Photon Mapping. *In Rendering Techniques 2004, Proceedings of the Eurographics Symposium on Rendering*, pages 111–121, juin 2004.

[Hei91] Tim HEIDMANN : Real shadows real time. *IRIS Universe*, 18:28–31, novembre 1991.

[HLHS03] Jean-Marc HASENFRATZ, Marc LAPIERRE, Nicolas HOLZSCHUCH et François SILLION : A survey of real-time soft shadows algorithms. *In Computer Graphics Forum*, volume 22(4), 2003.

[HWSB99] Geoffre S. HUBONA, Philip N. WHEELER, Gregory W. SHIRAH et Matthew BRANDT : The role of object shadows in promoting 3d visualization. *ACM Transactions of Computer-Human Interactions*, 6(3):214–242, 1999.

[JC95] H. JENSEN et N. CHRISTENSEN : Efficiently rendering shadows using the photon map. *In Proceedings of Compugraphics*, pages 285–291, décembre 1995.

[Jen96] Henrik Wann JENSEN : Global illumination using photon maps. *In* Xavier PUEYO et Peter SCHRÖDER, éditeurs : *Eurographics Rendering Workshop 1996*, pages 21–30, New York City, NY, juin 1996. Eurographics, Springer Wein. ISBN 3-211-82883-4.

[Jen01] H.W. JENSEN : *Realistic Image Synthesis using Photon Mapping*. A.K. Peters, 2001.

[KMK94] Daniel KERSTEN, Pascal MAMASSIAN et David C. KNILL : Moving cast shadows and the perception of relative depth. Rapport technique, Max-Planck-Institut fuer biologische Kybernetik, 1994.

[KN01] Tae-Yong KIM et Ulrich NEUMANN : Opacity shadow maps. *In Eurographics Workshop on Rendering*, pages 177–182, juin 2001.

[LV00] Tom LOKOVIC et Eric VEACH : Deep shadow maps. *In Proc. SIGGRAPH 2000*, pages 385–392, juillet 2000.

[LWGM04] Brandon LLOYD, Jeremy WENDT, Naga GOVINDARAJU et Dinesh MANOCHA : Cc shadow volumes. *In Eurographics Symposium on Rendering*, 2004.

[MKK98] Pascal MAMASSIAN, David C. KNILL et Daniel KERSTEN : The perception of cast shadows. *Trends in Cognitive Science*, 2(8):288–295, 1998.

[MT04] T. MARTIN et T. TAN : Anti-aliasing and continuity with trapezoidal shadow maps. *In Eurographics Symposium on Rendering*, pages 153–160, 2004.

[ODJ04] Victor OSTROMOUKHOV, Charles DONOHUE et Pierre-Marc JODOIN : Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics*, 23(3):466–475, 2004. Proc. SIGGRAPH 2004.

[PBMH02] Timothy J. PURCELL, Ian BUCK, William R. MARK et Pat HANRAHAN : Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, juillet 2002.

[Ray04] Mental RAY : http ://www.mentalimages.com/, 2004.

[RSC87] William T. REEVES, David H. SALESIN et Robert L. COOK : Rendering antialiased shadows with depth maps. *In Proc. SIGGRAPH 87*, volume 21, pages 283–291, juillet 1987.

[SCH03] Pradeep SEN, Michael CAMMARANO et Pat HANRAHAN : Shadow silhouette maps. *ACM Transactions on Graphics*, 22(3):521–526, juillet 2003.

[SD02] Marc STAMMINGER et George DRETTAKIS : Perspective shadow maps. *ACM Transactions on Graphics*, 21(3):557–562, juillet 2002.

[Shi00] Peter SHIRLEY : *Realistic Ray Tracing*. A.K. Peters, 2000.

[SKvW+92] Mark SEGAL, Carl KOROBKIN, Rolf van WIDENFELT, Jim FORAN et Paul E. HAEBERLI : Fast shadows and lighting effects using texture mapping. *In Proc. SIGGRAPH 92*, volume 26, pages 249–252, juillet 1992.

[SP94] François SILLION et Claude PUECH : *Radiosity and Global Illumination*. Morgan Kaufmann, 1994.

[Ups89] Steve UPSTILL : *RenderMan Companion : A Programmer's Guide to Realistic Computer Graphics*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[Wan92] Leonard WANGER : The effect of shadow quality on the perception of spatial relationships in computer generated imagery. *In 1992 Symposium on Interactive 3D Graphics*, volume 25, pages 39–42, mars 1992.

[WDP99] Bruce WALTER, George DRETTAKIS et Steven PARKER : Interactive rendering using the render cache. *In Eurographics Workshop on Rendering*, volume 10, pages 235–246, juin 1999.

[WDS04] Ingo WALD, Andreas DIETRICH et Philipp SLUSALLEK : An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. *In Eurographics Symposium on Rendering*, pages 81–92, juin 2004.

[WGS04] Ingo WALD, Johannes GÜNTHER et Philipp SLUSALLEK : Balancing Considered Harmful – Faster Photon Mapping using the Voxel Volume Heuristic. volume 22, 2004.

[WH03] Chris WYMAN et Charles HANSEN : Penumbra maps : Approximate soft shadows in real-time. *In Eurographics Symposium on Rendering*, pages 202–207, juin 2003.

[Whi80] Turner WHITTED : An improved illumination model for shaded display. *CACM, 1980*, 23(6):343–349, 1980.

[Wil78] Lance WILLIAMS : Casting curved shadows on curved surfaces. *In Proc. SIGGRAPH 78*, volume 12, pages 270–274, août 1978.

[WPF90] Andrew WOO, Pierre POULIN et Alain FOURNIER : A survey of shadow algorithms. *IEEE Computer Graphics & Applications*, 10(6):13–32, novembre 1990.

[WPS+03] Ingo WALD, Timothy J. PURCELL, Joerg SCHMITTLER, Carsten BENTHIN et Philipp SLUSALLEK : Realtime Ray Tracing and its use for Interactive Global Illumination. *In Eurographics State of the Art Reports*, 2003.

[WSP04] Michael WIMMER, Daniel SCHERZER et Werner PURGATHOFER : Light space perspective shadow maps. *In Eurographics Symposium on Rendering*, 2004.