REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider – BISKRA
Faculté des sciences exactes et des sciences de la nature et de la vie
Département d'informatique

Mémoire en vue de l'obtention du diplôme de

Magister en Informatique

Option : Synthèse d'Image et Vie Artificielle

Intitulé

VOLUME D'OMBRE EN RENDU TEMPS RÉEL

Par
ABD EL MOUMÉNE ZERARI
Soutenu le : 18 / 09 /2011

Devant le jury :			
Djedi Noureddine	**Professeur**	**Université de Biskra**	**Président**
Babahenini M^{ed} Chaouki	**Maître de Conférences**	**Université de Biskra**	**Rapporteur**
Moussaoui Abdelouahab	**Maître de Conférences**	**Université de Sétif**	**Examinateur**
Cherif Foudil	**Maître de Conférences**	**Université de Biskra**	**Examinateur**

Remerciements

« Avant toute chose, je remercie Dieu de m’avoir donner toute cette force»

Je remercie, avant toute personne, mon encadreur Monsieur le Docteur Mohamed Chaouki Babahenini, pour ses précieux conseils et surtout pour sa disponibilité et sa grande aide. Je lui exprime également toute ma reconnaissance pour sa compréhensibilité, sa bienveillance et la pertinence de ses interventions scientifique s.

Je tiens à exprimer mes vifs remerciements au Professeur DJEDI Nour Eddine d’avoir accepté d’être le rapporteur de ce mémoire. Je remercie également les membres du jury : Mr. le Docteur Cherif Foudil et Mr le Docteur Moussaoui Abdelouahab d'avoir accepté l’évaluation de ce travail.

Mes sincères remerciements s’adressent à tous les enseignants du département d’informatique, qui m’ont formé durant la période d’étude à l’université Mohamed Khider – Biskra.

Je souhaite également saluer tous les membres du laboratoire LESIA de l’université de Biskra, et particulièrement les collègue s Ali Beddiaf, Mebarek Boucetta et Nadir Henni pour leur générosité.

Dédicaces

Je dédie ce modeste travail à mes chers et bien aimés parents, pour leur soutient moral et pour leur encouragements et à tous les membres de la famille; plus particulièrement mes frères et sœurs.

A mon épouse, pour son soutien, à mes deux petites filles Aicha et Rafif.

A tous mes collègues de l'EPSP de Sidi okba : Mon Directeur Mr Zeroual Rachid et mon ex-directeur Mr Bouzahar, Mr Benziadi, Mr Heraki, Mr Trad et mon chère Mr Sahraoui Abdelali.

A tous mes amis; particulièrement : Elhani Djenaihi, Mahdi, Fouzi, Dakhi, Ali, Alla, Okba Djadda, Okba Zerari.

Table de matières

Liste de figures

Résumé

La modélisation et la simulation des interactions lumière/matière sont des aspects essentiels du rendu photoréaliste. Bien que de nombreux algorithmes de calcul d'illumination aient été proposés au cours des dernières décennies, ces méthodes ont généralement des coûts élevés en termes de mémoire et temps de calcul. Pour ce sujet de magister, nous nous sommes intéressés au calcul rapide de volume d'ombre (shadow volume) dans des scènes 3D. A cette fin, nous utilisons les performances des processeurs graphiques récents.

Le principe des Shadow Volumes consiste à calculer le volume d'ombre généré par un objet occultant une source de lumière. Une fois que ce calcul soit effectué, la sélection de certaines parties de ce volume permet l'affichage de l'ombre projetée. Cette technique est très intéressante, car bien implémentée, elle peut être utilisée dans de nombreuses applications. Les objets peuvent projeter leurs ombres sur eux-mêmes, contrairement à des techniques comme la projection plane ou la projection de texture.

L'objectif principal de notre étude est de proposer une nouvelle solution au problème important du rendu des volumes d'ombre en temps réel. Pour cela, nous avons proposé une approche basée sur l'intégration des GPU aux volumes d'ombre en utilisant le Geometry Shader.

Abstract

The modelling and the simulation of the interactions light/matter are essential aspects of photorealist rendering. Although many calculation algorithms of illumination were proposed during the last decades, these methods generally have costs high in terms of memory and computing time. For this subject of magister we are interested in fast calculation of shadow volume scenes 3D. For this purpose, we use the performances of the recent graphics processors.

The principle of Shadow Volumes consists of with to calculate the shadow volume generated by an object occulting a source of light. Once this calculation carried out, the selection of certain parts of this volume allows the posting of the projected shadow. This technique is very interesting, because it's well implemented, it can be used in many applications. The objects can project their shad ows on themselves, contrary to techniques like plane projection or the projection of texture.

The main objective of our study is to propose a new solution with the important problem of rendering shadow volumes in real-time. For that, we proposed an approach based on the integration of the GPU to shadow volumes by using Geometry Shader.

ملخص

تمثل النمذجة والتصنيع لتفاعلات بين الضوء والمواد جوانب جوهرية من تقديم حقيقي لصورة وواقعية تمثيلها.

على الرغم من أن العديد من الخوارزميات, قد اقترحت الإضاءة في العقود الأخيرة ، إلا أن هذه الطرق عادة ما تكون مرتفعة التكاليف من حيث الذاكرة والوقت الحوسب. يعالج هذه الرسالة موضوع حساب سريع لحجم الظل في المشاهد ثلاثية الأبعاد باستخدام معالجات الرسومات المتطورة و السريعة الأداء.
يتطلب حساب حجم الظل الذي تم إنشاؤه بواسطة جسم حاجب مصدر للضوء. عندما يتم هذا الحساب ،فإن اختيار أجزاء معينة من هذا الحجم يسمح بعرض الظل. هذه التقنية مثيرة للاهتمام, إذا تم تنفيذها بشكل جيد ، يمكن استخدامها في العديد من التطبيقات. يمكن للأجسام أن تظلل نفسها و أن تظلل غيرها من الأجسام، وعلى عكس تقنيات مثل الإسقاط المستوي.

والهدف الرئيسي لدراستنا هو اقتراح حل جديد لمشكلة عرض كميات كبيرة من الظل في الوقت الحقيقي. لهذا ، اقترحنا نهج يقوم على دمج وحدات المعالجة الرسومية لحجم الظل باستخدام شادر الهندسة.

Introduction générale

À l'heure actuelle, l'imagerie en trois dimensions est de plus en plus présente dans notre quotidien. L'industrie du divertissement, qui inclut les domaines d'activités liés aux développements des jeux vidéos, ou à la création de films ayant recours à des technologies informatiques, en est un exemple fort. Le domaine de l'informatique graphique, et plus particulièrement celui de la visualisation de données et du rendu de celles-ci, représente un secteur très vaste d'applications pouvant couvrir des domaines extrêmement variés qui s'étendent de l'exploration de données, provenant de sources diverses (résultats de simulations physiques, données topographiques, ...) ou de divertissements (jeux vidéos, film d'animations, ...). Ces différents domaines d'application ont des besoins de plus en plus importants, en effet, ils nécessitent des outils toujours plus développés afin de visualiser les différentes données qu'ils ciblent. Cette complexité découle, en partie, des masses de données manipulées sans cesse plus importantes et plus complexes.

Cependant, afin de fournir des résultats visuels qui soient en adéquation avec les attentes des différents utilisateurs, il est nécessaire de traiter les données en fonction de leurs propriétés. Ces dernières peuvent varier suivant leur provenance et de leur usage final. Ainsi, afin d'adapter au mieux les outils de visualisation, il existe un grand nombre de types de représentations des données, ayant chacune leurs spécificités et leurs domaines d'applications. On peut, à titre d'exemple, citer les données volumiques utilisées dans le domaine médical, ou la représentation de surfaces en trois dimensions via des maillages pour les jeux vidéos.

Le domaine de la synthèse d'image s'intéresse aussi d'assez prés aux ombres. Les énormes progrès accomplis ces dernières années dans la création d'images synthétiques permettent d'obtenir des images de plus en plus réalistes. Mais les utilisateurs sont devenus de ce fait de plus en plus exigeants. Les ombres présentent dans ce cadre un des critères permettant d'évaluer la qualité d'une image synthétique.

Dans ce cadre de la synthèse d'images en général, des méthodes plus spécifiques permettant de traiter les ombres ont récemment vu le jour. En effet, les incessants progrès matériels combinés aux dernières techniques de synthèse d'images permettent aujourd'hui la création d'images de très grande qualité. Cependant cela a aussi engendré une demande de plus en plus exigeante sur ces images, au niveau de la qualité bien sûr mais aussi au niveau des temps de calcul. Or, la création d'images réalistes impose encore des coûts de calcul et de stockage des données très importants. C'est pourquoi des travaux sont effectués afin de diminuer les coûts de calcul de ces simulations et d'obtenir une bonne maîtrise des coûts de stockage. Dans les scènes dynamiques qui autorisent le déplacement des objets de la scène, le critère temps est notamment un critère fondamental.

Pour obtenir une image réaliste, il est particulièrement important de simuler correctement les régions ombrées. Elles jouent un rôle prépondérant pour la compréhension humaine d'un environnement. 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. 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], [CCOD04], [WAL08]. Le problème le plus important qui ce pose est l'optimisation des volumes d'ombre pour aboutir au temps réel.

Objectifs du mémoire.

À la lumière de la montée subite et récente de l'intérêt pour des applications interactives autours des environnements virtuels, beaucoup de recherches ont été consacrées à la résolution de problèmes des ombres dans des scènes

tridimensionnelles, et particulièrement aux volumes d'ombre. Cependant, la plupart des approches pour la génération des volumes d'ombre en temps réel exigent des pré-calculs ou de multiples passes de rendu. Cette défaillance, réduit et limite inexorablement leur temps de calcul dans les scènes 3D complexe.

Notre travail concerne l'étude du rendu de volumes d'ombre en temps réel, cette problématique intéresse de plus en plus les laboratoires de recherche en synthèse d'images. Néanmoins, la qualité visuelle qu'ils offrent est très souvent associée à une complexité dans la prise en charge des accélérations matérielles.

Notre objectif principal est de proposer des algorithmes optimisés et efficaces (temps réel) et d'aboutir à un aspect simulation le plus rapide possible.

Organisation du mémoire

Nous avons organisé notre mémoire en quatre chapitres. Le premier chapitre expose les techniques de l'illumination et les techniques pour la génération d'ombre, nous présentons les termes les plus répondus dans notre domaine d'étude, tels que le rendu des ombres en temps réel.

Le deuxième chapitre présente, une description détaillée de l'état de l'art sur la génération des volumes d'ombre. La synthèse effectuée sur ces travaux, nous permettra d'établir une proposition afin d'améliorer le rendu en temps réel des volumes d'ombre.

Le troisième chapitre traite notre contribution basée sur les nouvelles performances des processeurs graphiques. Nous allons ainsi présenter, la technique utilisée pour rendre une telle représentation en utilisant les shaders et précisément le Geometry Shader, ce qui permettra d'améliorer le rendu des volumes d'ombre.

Enfin, le dernier chapitre permettra d'évaluer notre proposition et de présenter les premiers résultats. Nous montrons dans un premier temps les démarches de notre algorithme en augmentant la complexité de notre scène 3D, et dans un second temps nous comparons notre proposition avec d'autres modèles.

Nous terminons notre mémoire par une conclusion dans laquelle nous citons les perspectives de notre travail.

Chapitre I :

Illumination et Ombres

Illumination et Ombres

Dans ce chapitre, nous introduisons le domaine de l'illumination et des ombres, en donnant des définitions sur les termes les plus utilisés tels que la lumière, l'ombrage et l'ombre et en présentant également une brève description sur les modèles d'éclairement et sur les types de sources lumineuses existantes. Nous détaillons ensuite les techniques pour la génération des ombres, et nous ferons une synthèse sur les travaux de recherche liés à ce domaine. Le choix des travaux, est basé essentiellement sur leur popularité, et sur le fait qu'ils soient réalisés récemment. A la fin de ce chapitre, nous proposons un bilan de ces travaux selon des critères que nous définirons par la suite.

1. Introduction

La lumière à travers son meilleur représentant, le soleil, est source de vie et de création. Mais d'un aspect plus pratique, elle nous est aussi complètement indispensable dans la vie de tous les jours. Sans elle nous sommes aveugles et c'est pourquoi de nombreuses sources de lumière artificielle ont été développées [Pin02].

L'éclairage donne une personnalité à une pièce. La présence de lumières ou d'ombres résultant de l'éclairage permet de mieux distinguer leur forme ou leur donner une apparence plus dramatique. Les ombres jouent un rôle important dans la perception d'une scène 3D. Elles fournissent des informations importantes sur la forme et la position des objets [Ron09].

2. Illumination

Lors de la formation d'une image, le calcul de la couleur et de l'intensité lumineuse s'effectue généralement en deux étapes.
Une première étape qui consiste à estimer globalement les échanges lumineux dans la scène, afin de connaître la quantité de lumière (intensité et couleur) parvenant jusqu'à chaque point de chaque surface.

Une seconde étape qui calcule localement la manière avec laquelle la surface réagit à la lumière reçue. Ce calcul est effectué par le modèle d'illumination locale. A cette fin, celui-ci utilise les propriétés locales du matériau (couleur, brillance, etc ...), qui peuvent varier le long de la surface [Lef06].

2.1. Les modèles d'illumination

Un modèle d'illumination comprend tout le processus de transport de la lumière, qui comprend son émission, sa réflexion, réfraction, diffusion ou absorption. Il permet d'obtenir une solution approchée de l'illumination d'une scène. Deux types de modèles existent : les modèles d'illumination locale qui ne considèrent que la contribution directe des sources de lumière et les modèles d'illumination globale qui comprennent en plus la lumière interréfléchie entre les différentes surfaces de la scène. Dans ce processus, un modèle de réflexion se concentre sur la lumière qui atteint une surface et sur la lumière qui est réfléchie [Pin02].

2.2. Illumination locale

Le calcul de l'illumination locale d'un objet consiste à déterminer son éclairement direct par une source lumineuse sans tenir compte de l'influence que peut avoir un objet voisin sur cet éclairement. La définition d'un modèle d'illumination locale découle des lois de l'optique géométrique et est affinée de manière empirique en fonction des résultats que l'on veut obtenir. Les différents angles entrant en jeu dans le calcul de l'illumination locale sont décrits par la (figure 1) [Pau95], la lumière provient directement des sources de lumière.

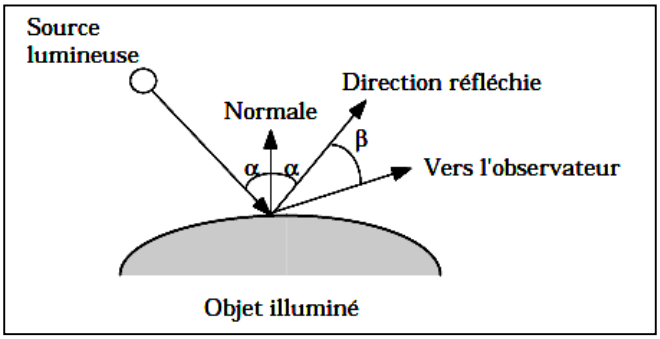


Figure 1 : Géométrie pour l'illumination locale [Pau95].

2.2.1. Modèles d'éclairement

Un modèle d'éclairement définit la réaction de la lumière rencontrant une surface. En fonction de la complexité des modèles utilisés, les différents effets du comportement de la lumière seront plus ou moins bien reproduits [Cha06].

2.2.2. Lumière ambiante

Le modèle le plus simple consiste à prendre en compte une source lumineuse non ponctuelle qui émet de manière constante dans toutes les directions et sur toutes les surfaces une lumière d'intensité Ia. Alors pour un point P de la surface, l'intensité lumineuse sera : $Ip = Ka \times Ia$; qui est constante en tous points de la surface, et où Ka est un facteur qui détermine la quantité de lumière ambiante réfléchie par la surface et est fonction des propriétés matérielles de la surface ($0 \leq Ka \leq 1$) [Boy07].

2.2.3. Modèle de réflexion

Un modèle de réflexion décrit l'interaction entre la lumière et une surface en fonction des propriétés du matériau constitutif de la surface ainsi que de la nature de la lumière et de son incidence.

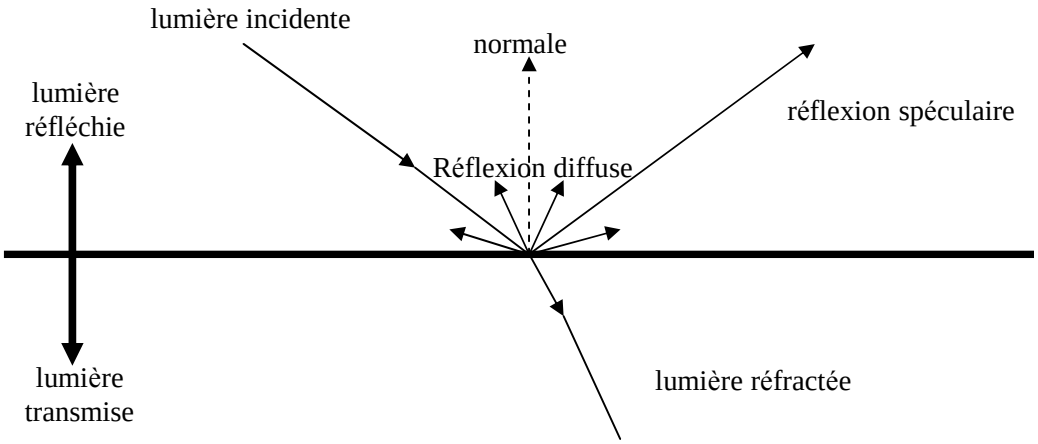


Figure 2 : Une partie du flux lumineux est absorbée par le matériau.

2.2.4. Modèle diffus

Le modèle de réflexion diffus est régi par la loi de Lambert: l'intensité de la lumière réfléchie augmente au fur et à mesure que la direction de la source devient colinéaire avec la normale à la surface au point considéré. L'intensité est donc proportionnelle au cosinus de l'angle que fait la normale avec la direction de la source de lumière incidente.

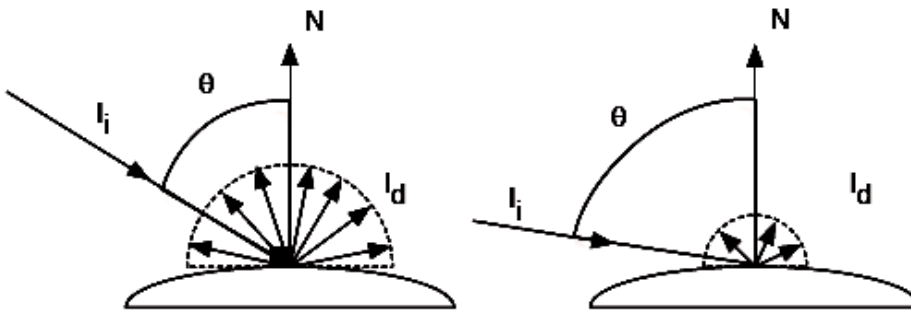


Figure 3 : Le modèle Lambertien [Cha06].

Le modèle de Lambert s'écrit de la façon suivante : $Id = Ii \times K_{diff} \times \cos\theta$

- ***Id*** : intensité diffuse, c'est-à-dire l'intensité de lumière sortante (quelle que soit la direction, par isotropie) ;
- ***Ii*** : intensité incidente ;

- K_{diff} : paramètre de texture paramétrant le comportement. Un faible coefficient traduit un objet absorbant, tandis qu'un fort coefficient traduit une forte réémission ;
- θ : angle d'incidence de la lumière, calculé par rapport au vecteur normal à la surface [Cha06].

2.2.5. Spécularité

La spécularité d'une surface représente sa capacité à se comporter comme un miroir, c'est à dire à réémettre la lumière dans une direction privilégiée.

Le modèle de Phong (Figure 4) est un modèle spéculaire classique ; il s'écrit de la façon suivante :

$$I_s = I_i \times K_s \times (\cos\theta)^n$$

- **Is** : intensité spéculaire, c'est-à-dire l'intensité de lumière sortante dans une direction donnée ;
- **Ii** : intensité incidente ;
- **Ks** : paramètre de texture paramétrant le comportement. Un faible coefficient traduit un objet à faible composante spéculaire ;
- **θ** : angle mesuré entre la direction de sortie considérée et le rayon réfléchi idéal ;
- **n** : paramètre de texture caractérisant le poli de l'objet. Un n grand traduit une spécularité aiguë, le point brillant se resserrant autour de la direction du rayon réfléchi idéal.

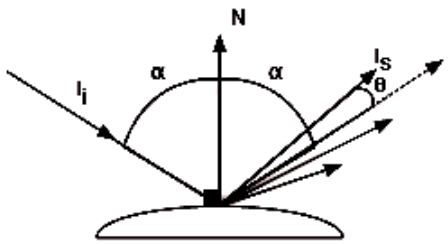


Figure 4 : Le modèle spéculaire de Phong : la réémission est plus ou moins intense autour du rayon réfléchi idéal [Cha06].

2.2.6. Modèle d'éclairement de Phong

Le modèle d'illumination locale le plus utilisé est le modèle défini par B.T. PHONG en 1975 [Pho75]. Il définit l'intensité lumineuse réfléchie en un point *P* d'une surface en fonction de l'intensité des sources lumineuses. Ce modèle est une généralisation de la loi de Lambert sur la réflexion de la lumière. Cette loi, aussi appelée loi du cosinus, décrit l'intensité diffusée par un point d'une surface mate en fonction de l'intensité et de la direction de la source prise en compte.

$$I(P)_{diffusée} = K_d .cos\ \alpha .I_{source}$$

Equation 1. Loi de Lambert.

où : $I(P)_{diffusée}$ = Intensité diffusée au point P $k_d(P)$= Coefficient de diffusion au point *P*

I_{source} = Intensité de la source α= Angle entre la normale et la direction de la source

De même, pour un point situé sur une surface brillante, on peut écrire la loi de réflexion suivante (équation 2.):

$$I(P)_{réfléchie} = K_s(\alpha) . cos^{n}\ \beta . I_{source}$$

Equation 2. Loi de réflexion sur une surface brillante.

où : $I(P)_{réfléchie}$ = Intensité réfléchie au point *P* I_{source} = Intensité de la source

$k_s(\alpha)$ = Coefficient de réflexion au point *P* (dépend de l'angle α)

α = Angle entre la normale et la direction de la source

β = Angle entre la direction réfléchie et la direction de l'observateur

n = Exposant de Phong

Le modèle de PHONG complet se déduit des équations 1 et 2 et s'écrit, pour plusieurs sources lumineuses :

$$I(P) = K_d\ I_{ambiant} + \sum_{i \le N} K_d .cos\ \alpha_i .I_i + \sum_{i \le N} K_s(\alpha_i). cos^{n}\ \beta_i . I_i$$

où : $I_{ambiant}$ = Estimation de l'intensité ambiante

N = Nombre de sources lumineuses dans la scène

2.3. Illumination globale

La lumière provient directement des sources de lumière ainsi que de la lumière distribuée entre les surfaces (lumières secondaires). La recherche d'un modèle représentant l'illumination globale d'une scène passe par la modélisation de toutes les interactions entre les objets de cette scène. Ces interactions sont dépendantes de la réaction locale d'un objet face à une onde lumineuse. La formulation du modèle représentant l'illumination globale d'une scène en des termes physiques est ass ez lourde. Aussi, une formulation plus abstraite du problème est nécessaire, surtout si l'on veut en développer une solution algorithmique. [Pau95]. Pour des exemples d'algorithmes utilisés dans l'illumination globale, on peut citer notamment : la radiosité, le lancer de rayon, photon mapping, le beam tracing, le cone tracing, la path tracing, le metropolis light transport et l'ambiant occlusion.

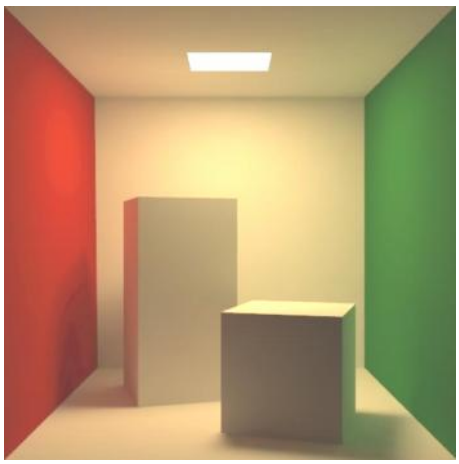


Figure 5 : Ombres avec l'algorithme d'illumination globale [Rog07].

2.4. Sources de lumière

Les sources lumineuses peuvent être regroupées en plusieurs catégories selon leurs caractéristiques intrinsèques ou le type de distribution de la lumière émise.

On distingue généralement plusieurs types de sources de lumière [Mig04]:

1. **la lumière ambiante** : C'est une lumière qui éclaire toute la scène uniformément et qui est uniquement caractérisée par son intensité. C'est aussi une lumière diffuse (non directionnelle) dont il n'est pas possible de déterminer la source. Tout objet de la scène est plongé dans cette lumière .

1. **les sources directionnelles** : Une source de lumière est dite directionnelle si tous les rayons de lumière sont parallèles et provien nent de la même direction. Tout se passe comme si la source lumineuse était infiniment loin.

   Elle se caractérise par son intensité I et sa direction **U**. En un point A quelconque, la direction de cette source est $\mathbf{L} = -\mathbf{U}$.

   Le soleil est bien approxim é par une source directionnelle car tous les rayons semblent parallèles pour des objets à l'échelle humaine. En règle générale, une source ponctuelle éloignée pourra être approximée par une source directionnelle.

1. **les sources ponctuelles** : ce sont des sources de lumière, supposées placées en un point précis et qui rayonnent la lumière radialement; elles sont donc caractérisées par leur intensité et leur position.

   Elle se caractérise par son intensité I et sa positi on P. En un point A quelconque, la direction de cette source ponctuelle est obtenue avec le vecteur $L = AP/|AP|$.

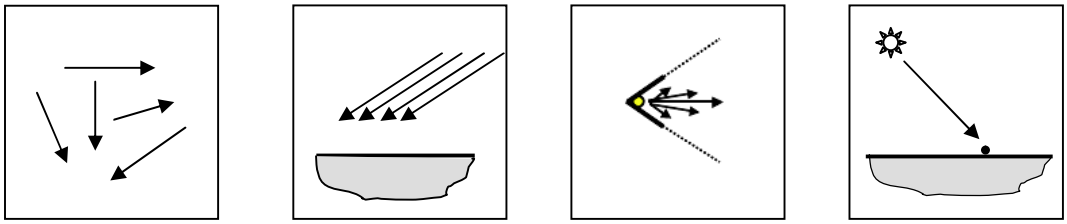
   On distingue parmi les sources ponctuelles :

   – les sources isotropes qui diffusent radialement ses rayons dans toutes les directions (figure 6 (a)) et avec la même intensité.
   – les sources anisotropes qui diffusent ses rayons dans une direction privilégiée (figure 6 (b)) et dont l'intensité dépend de l'écartement à cette direction. Un spot est un exemple classique de source ponctuelle anisotrope.

(a) source ponctuelle (isotrope). (b) source anisotrope.

Figure 6 : Les deux types de sources ponctuelles couramment utilisées [Mig04].

1. **les projecteurs ou spots** : ce sont des sources de lumière caractérisées par leur position, leur direction et un facteur de concentration.



a- Lumière ambiante b- Source directionnelle c- Projecteur ou spot d- Source ponctuelle

Figure 7 : Les types de sources de lumières.

3. Répartition de la lumière

L'ombrage (shading) est un terme trompeur en synthèse d'images. Dans un contexte d'illumination, il désigne la méthode de remplissage des polygones en fonction des normales utilisées. On distingue trois méthodes [Fau02] :

- l'ombrage plat (flat shading)
- l'ombrage de Gouraud (Gouraud shading)
- l'ombrage de Phong (Phong shading), à ne pas confondre avec le modèle d'illumination de Phong.

3.1. Ombrage plat

La méthode d'ombrage la plus simple pour les facettes polygonales est l'ombrage plat (ou constant). L'idée est de calculer une seule valeur d'illumination pour l'ensemble de la facette. Par exemple au point milieu de la facette en prenant pour normale à la surface celle du plan contenant la facette [Boy07], pour cela on suppose que :

- la source lumineuse est à l'infini,
- la projection est orthographique,
- la surface est composée de facettes polygonales uniquement.

16 x 16 facettes 32 x 32 facettes 64 x 64 facettes

Figure 8 : Ombrage plat de la sphère pour : 16 x 16 facettes [Boy07].

3.2. Ombrage de Gouraud

La méthode développée par Gouraud en 1971 élimine les discontinuités d'intensité sur une facette polygonale par interpolation des valeurs d'intens ité aux sommets de la facette. Cette méthode est largement utilisée et se retrouve dans la majorité des matériels graphiques existants (librairies, cartes graphiques).

Cette méthode requiert la connaissance de la normale à la surface aux sommets des facettes polygonales. Lorsque les normales sont connues, les intensit és aux sommets des facettes polygonales sont calcul ées. L'interpolation peut ensuite être effectuée à l'aide de l'algorithme de balayage de ligne utilisée pour le remplissage de polygone et le z-buffer [Fau02].

3.3. Ombrage de Phong

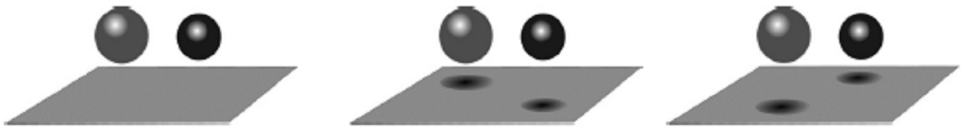
L'ombrage de Phong [Pho75] consiste à interpoler les normales au lieu d'interpoler les couleurs. La couleur du point est alors calculée en fonction de la normale interpolée.

L'intérêt de cette approche par rapport à l'ombrage de Gouraud réside principalement dans sa capacité à traiter les réflexions spéculaires. Gouraud ne permet pas, en effet, de prendre en compte les réflexions spéculaires lorsque celles -ci sont localisées au centre d'une facette [Fau02].

4. Les Ombres

4.1.1. Généralité

La présence des ombres dans une image fournit des informations essentielles pour la compréhension d'une scène et notamment sur la position relative des objets entre eux et des sources lumineuses (figure 9). Elles sont nécessaires au réalisme, le placement des ombres détermine totalement la position des sphères relativement au plan .



(a) scène sans ombre. Laquelle des deux sphères est la plus proche ?

(b) Celle de droite ?

(c) Celle de gauche ?

Figure 9 : Importance des ombres pour la compréhension d'une scène [Mig04].

4.1.2. Qu'est-ce qu'une ombre ?

Une ombre se définit comme l'obscurité que cause un corps opaque en interceptant la lumière. Elle est une conséquence particulière de l'illumination dans une scène. Les ombres font partie intégrante de notre environnement et sont présentes dans beaucoup de scènes naturelles, dés qu'une source de lumière est prédominante par rapport à la lumière ambiante. Elles sont le résultat d'une interaction entre une composante géométrique et une composante photométrique [Pin02].

Les ombres ont toutes la même origine, une obstruction par un objet d'une source de lumière. Il existe cependant autour de nous une multitude d'ombres différentes, tant par leurs formes que par leurs couleurs. Il est alors nécessaire de définir quelques caractéristiques communes de ces ombres afin de mieux les maîtriser. Les ombres se classent tout d'abord en trois catégories bien distinctes (figure 10) [Pin02] :

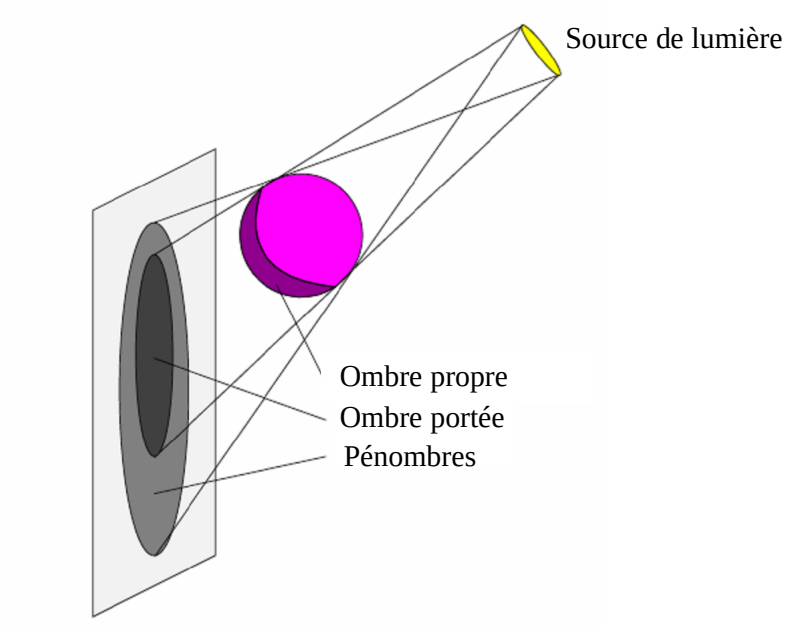


Figure 10 : Différentes ombres présentes dans une scène [Pin02].

- **les ombres propres** sont les parties des objets non éclairées par une source lumineuse ;
- **les ombres portées** sur une surface sont le résultat de l'obstruction de la lumière par un objet ;
- **la pénombre** se situe à la périphérie des ombres portées, quand un point n'est éclairé que par une partie de la source lumineuse. Elle constitue la transition entre une zone éclairée et une ombre.

4.2. Typologie des ombres

Au regard des ombres, les méthodes de rendu en synthèse d'images se divisent en deux catégories [Mig04] :

– celles qui résolvent (ou approximent) l'équation d'illumination globale. Dans ce cas, les ombres font naturellement parties de la solution trouvée.

– celles qui, par souci de performance, effectuent une résolution locale et ne gèrent que les éclairages directs ou les trajets des rayons (potentiellement) les plus énergétiques. Dans ce cas, il devient nécessaire de traiter explicitement le calcul des ombres (*Z*-buffer, lancé de rayons élémentaires).

En effet, (voir aussi figure 11),

- soit on considère que les rayons lumineux interceptés en question sont les rayons lumineux directs, et dans ce cas, la zone sombre située sous la sphère (figure 11 (b)) n'est pas une ombre.
- soit on parle également des rayons indirects, mais dans ce cas, les interréflexions impliquent qu'il faudrait alors considérer toutes les zones sombres comme des ombres.
- le mot opaque est également utilisé : donc l'ombre engendrée par la sphère transparente à gauche n'est pas une ombre (figure 11 (a)).

(a) Variétés des ombres. (b) Ombres indirectes.

Figure 11 : Exemples d'ombres réalistes obtenues avec une méthode d'illumination globale. Figure (b) : les ombres sous la sphère sont engendrées par l'éclairage indirect (issues des interactions entre objets) [Mig04].

On peut donc commencer à classer les différents types d'ombres :

- **les ombres directes** sont les zones sombres résultant de l'occultation directe d'une source de lumière par un objet.
- **les ombres indirectes** sont les zones sombres résultant de l'occultation de rayonnements indirects par un objet (figure 11 (b)).

Dans les deux cas, la disparition de l'objet occultant doit également faire disparaître l'ombre qui y est associée.

Le phénomène d'occultation peut prendre également deux formes :

- soit l'objet est opaque, les rayons touchants l'objet sont déviés par sa surface.

- soit l'objet est transparent, et dans ce cas, les rayons subissent soit une absorption partielle (diminuant ainsi leurs intensités), soit une focalisation (en concentrant les rayons dans une région au sortir de l'objet), soit les deux.

On n'oubliera pas le cas des objets non convexes qui peuvent engendrer des auto-ombrages (ombres crées par une partie de l'objet sur une autre partie du même objet; voir aussi figure 13).

Par ailleurs, pour les ombres directes, on peut encore différentier deux types d'ombres (figure 13) :

- **les ombres franches (dures)** (hard-shadow) : 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. 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 [Amo04].
- **les ombres douces** (soft-shadow) engendrées par les sources étendues. On a alors trois cas : la source est totalement visible, la source est partiellement visible (zone de pénombre), la source est complètement occultée (zone d'ombre). Voir aussi la (figure 14) pour comprendre la dépendance entre la taille de la source et l'étendue de la pénombre (plus la taille de la source lumineuse est importante, plus la largeur de la zone de transition est grande). 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 est particulièrement difficile (et donc habituellement plus long), mais les ombres douces donnent des images beaucoup plus près de la réalité (figure 12) [Amo04].

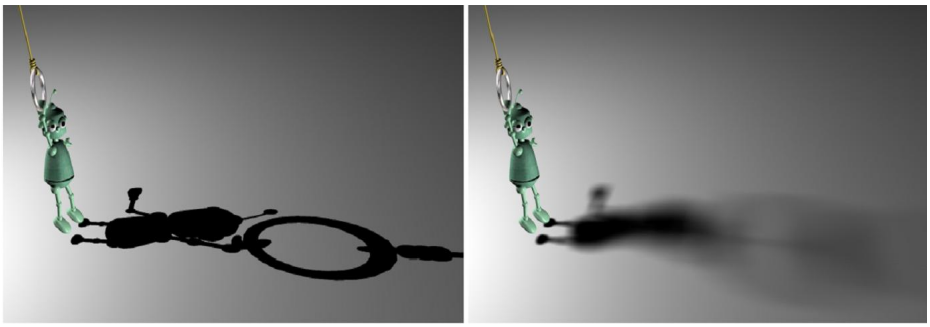
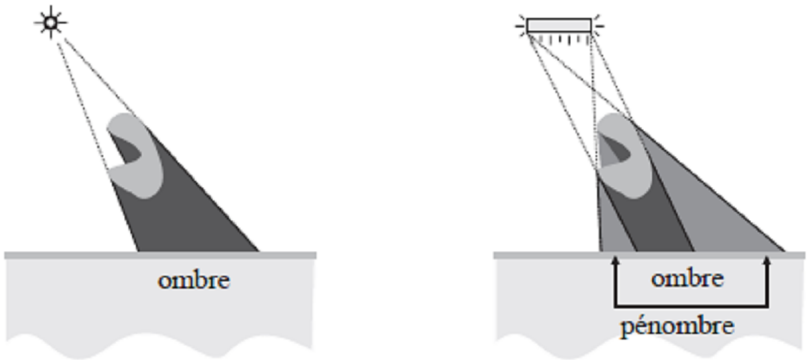


Figure 12 : Ombre dure vs. Ombre douce [HLHS03].



(a) Ombre franche créée par une source ponctuelle.

(b) Ombre douce créée par une étendue.

Figure 13 : Ombres directes générées en fonction du type de source. L'objet étant non convexe, on a également fait apparaître l'auto-ombrage de l'objet [Mig04].

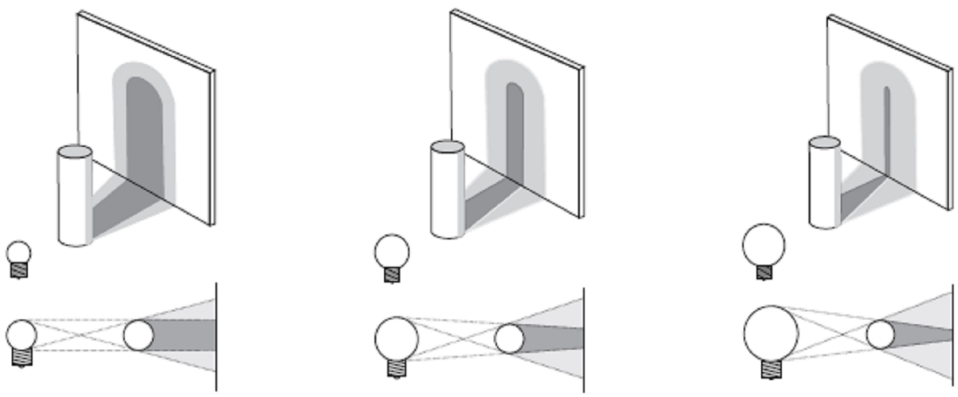


Figure 14 : Comportement de la pénombre en fonction de la taille de la source. En gris clair, la zone de pénombre. En gris foncé, la zone d'ombre [Mig04].

4.3. Comment générer des ombres portées

Les sources de lumière jouent un rôle fondamental dans le réalisme des images, cependant, il faut remarquer qu'à moins de placer une source unique de lumière à l'emplacement de l'observateur, il est indispensable de tenir compte de l'ombre portée. Malheureusement, le calcul de l'ombre portée est très coûteux en termes de temps-machine. C'est pour cette raison que les réalisateurs de films d'animation choisissent souvent une illumination très diffuse (ciel couvert, par exemple) pour limiter l'effet de manque d'ombre portée. En fait, la plupart des algorithmes de production d'ombres portées ont de sévères limitations sauf le lancer de rayons qui est une technique elle-même coûteuse en temps CPU [Tha03].

D'après Daniel Thalmann [Tha03] les techniques de production d'ombres portées sont classées en quatre catégories:

1. Techniques traditionnelles ; les deux principaux algorithmes sont ceux de Nishita et Nakamae (1974) et de Atherton et al. (1978).
2. La technique des volumes d'ombre introduite par Crow (1977) [Cro77].
3. Les techniques basées sur un algorithme de mémoire de profondeur (z-buffer); les deux principaux algorithmes sont ceux de Williams (1978) et de Brotman et Badler(1984) [Wil78].
4. La technique de lancer de rayons.

Les ombres dans l'infographie sont le résultat de l'interaction entre trois composants : sources lumineuses, récepteurs d'ombre et un occultant (figure 15).

Une source lumineuse est un objet qui modélise une source de lumière primaire, c-à-d., une certaine entité physique qui produit la lumière. Les sources lumineuses dans le monde réel émettent les particules énergiques (photons) qui voyage à travers l'espace et peuvent rencontrer des obstacles physiques. Selon la surface de l'obstacle, elles sont absorbées ou reflétées. Les objets qui bloquent partiellement ou complètement des photons d'atteindre un récepteur d'ombre s'appellent les occultants ou bloqueurs de lumière (Shadow Casters), puisqu'ils projettent des ombres sur les récepteurs d'ombre. Si un objet projette une ombre sur lui-même, il produit de l'ombre propre (self-shadowing) [Ste06].

Toutes les sources lumineuses du monde réel sont des sources lumineuses étendues (figure 15), ainsi il signifie qu'elles ont une ampleur physique. Dans l'infographie, cependant, les sources lumineuses ponctuelles (figure 16) sont employées souvent pour rapprocher des sources lumineuses étendue s. Ils n'ont aucune aire physique mais peuvent être décrites mathématiquement comme singularité (un point dans l'espace) [Ste06].

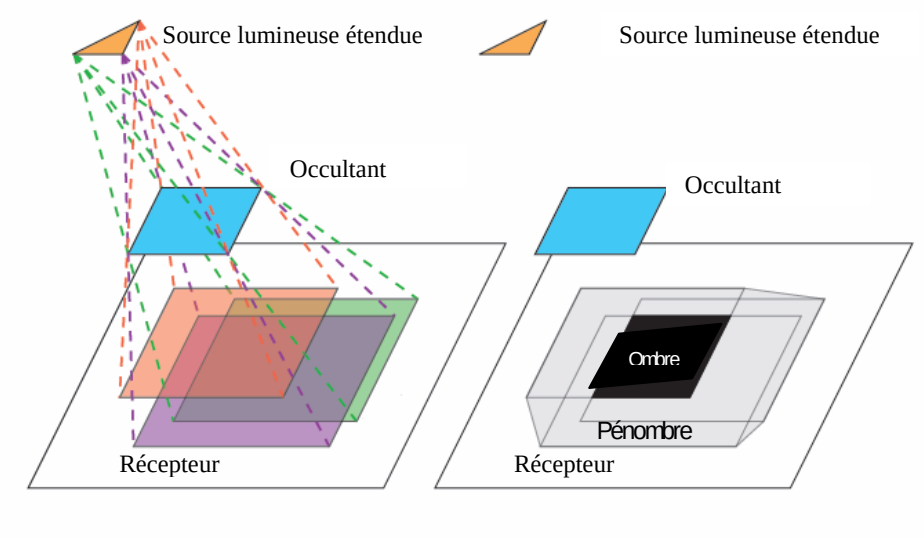


Figure 15: Régions d'ombre et de pénombre d'une source lumineuse étendue [HLHS03].

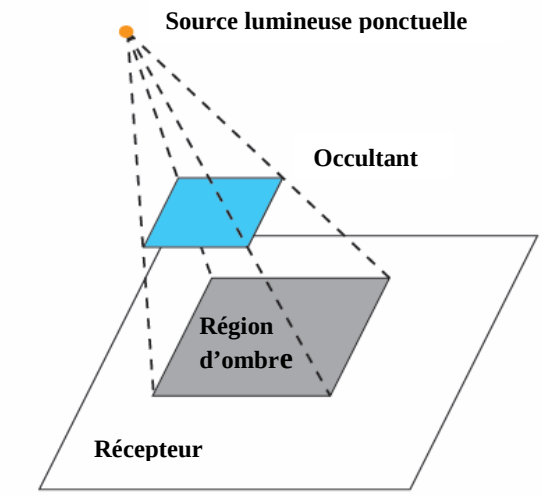


Figure 16: Région d'ombre d'une source lumineuse ponctuelle [HLHS03].

4.4. Ombre en temps réel

Au cours des dix dernières années, des progrès significatifs ont été accompli dans le domaine de l'infographie, particulièrement dans le rendu en temps réel. La recherche des nouvelles méthodes pour rendre les jeux vidéo, environnements virtuels et applications graphiques interactives plus réalistes, a vu un essor récent de travaux portant sur la génération d'ombres en temps-réel [Ber86]; [WPF90]; [Kil01]; [EK03]; [HLHS03]; [AAM04]; [AW04]; [CD04]; [LWGM04] ; [Ste06]; [Sch09] ; [Schw09].

Le calcul d'ombre dans une application d'infographie est une tâche très coûteuse en temps de calcul. Cela dépend de la méthode de rendu, l'algorithme d'ombre utilisé, le matériel disponible, la qualité de l'ombre, et la complexité de la scène. La génération d'une image ombragée peut être une tâche très longue, elle peut prendre plusieurs millisecondes jusqu'à plusieurs minutes ou même heures. Le matériel d'aujourd'hui est capable de rendre diverses techniques de génération d'ombre en temps réel [Schw09]. L'ombre en temps réel est un vrai défi en infographie, et il n'y a aucune solution générale qui peut être appliquée sur n'importe quelle scène [Schw09].

Les méthodes pour générées les ombres telles que : le lancer de rayons et la radiosité ne sont pas adapté au temps réel, car elles sont très coûteuses en temps de calcul. Mais les méthodes pour générées les ombres comme le Shadow map (basée image) et Shadow volume (basée objet) sont adapté au temps réel.

5. Les techniques de calcul d'ombres

Les techniques de calcul d'ombres peuvent être, dans la majorité des cas, regroupées en plusieurs catégories : les techniques traditionnelles, les techniques basées sur le lancer de rayons, les techniques d'illumination globale, les techniques basées sur les tampons de profondeur (depth buffers ou shadow maps) et finalement, celles basées sur les volumes d'ombre qui est l'objectif de notre travail.

5.1 Les techniques traditionnelles

Les algorithmes pour calculer les ombres en temps réel peuvent être classifiés comme des algorithmes d'espace de l'objet ou espace de l'image. Les méthodes d'espace de l'objet, également désignées sous le nom des algorithmes précision de

l'objet, fonctionnent dans le monde d'espace tridimensionnel et produisent des frontières d'ombre exactes, la plupart du temps ils utilisent des calculs géométriques. Parmi les algorithmes d'ombre en espace de l'objet ont peut cités les ombres projeter et les volumes d'ombre. Contrairement aux techniques de précision de l'objet, les algorithmes d'ombre en espace image fonctionnent dans un espace bidimensionnel, typiquement après qu'un certain genre de projection ait été appliqué. Les méthodes d'espace image sont généralement plus rapides que les méthodes d'espace objet. La technique d'ombre d'espace image la plus célèbre est les cartes d'ombre (shadow maps) [Ste06].

5.1.1. Fausse ombre

Une manière plus simple de créer une ombre est d'ajouter au-dessous de l'objet un polygone qui représente l'ombre. La position et la forme de ce polygone n'est pas exactement calculée [Ski01], [Bli88].

Figure 17: Jeux de Tomb RaiderII.

Les fausses ombres ont été utilisées dans beaucoup de jeux il y a quelques années, les ombres ont des formes quelconques et pas précises, mais c'est beaucoup mieux que rien. C'est une manière très simple de faire l'ombre. Malheureusement, elle est très limitée parce qu'elle exige que la surface de la plateforme doit être plate, l'ombre est estimée par un polygone qui ressemble à l'objet qui produit de l'ombre.

5.1.2. Ombres polygonales

En 1978, Atherton, Weiler, et Greenberg ont présenté une méthode générale pour la génération des ombres en utilisant un algorithme basé sur le détourage (clipping) de polygone dans l'espace d'objet [FE99] ; [PWG78]. Leur algorithme est fondé sur le fait qu'aucune ombre ne peut être vue du point de vue de la lumière. D'abord, la caméra est placée à l'endroit de la source lumineuse. Tous les polygones sont alors détourer d'une manière telle que tous les polygones visibles soient séparés de tous les polygones non visibles. Après cette étape, les polygones visibles correspondent aux polygones qui ne sont pas dans l'ombre, tandis que les polygones non visib les sont des polygones d'ombre [Ste06]. Un inconvénient avec cette technique est le fait que ses mesures de performance avec le nombre de polygone dans la scène, le rendant pratiquement impossible à l'employer dans des grandes scènes sans l'utilisation d'un certain genre d'optimisation pour réduire le nombre de triangles qui participent au processus de clipping.

5.1.3. La projection planaire

Chaque objet créant de l'ombre dans la scène est rendu deux fois, une fois normalement et une fois sous la forme d'un objet《ombre》 plane, qui une fois colorie en noir donnera l'ombre de l'objet [Coh04]. Pour ce faire, on calcule une nouvelle matrice monde en se basant sur les propriétés (type, position...) de la lumière et sur celles du plan receveur de l'ombre de l'objet [EMN08] ; [Tha89].

Considérons un vertex v, « illumine » par une source lumineuse L située en l, et dont la projection sur le plan d'équation :

π : n . x +d =0, se nomme p.

La matrice monde à appliquer est :

$$\mathbf{M} = \begin{pmatrix} \mathbf{n}\cdot\mathbf{l} + d - l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\ -l_y n_x & \mathbf{n}\cdot\mathbf{l} + d - l_y n_y & -l_y n_z & -l_y d \\ -l_z n_x & -l_z n_y & \mathbf{n}\cdot\mathbf{l} + d - l_z n_z & -l_z d \\ -n_x & -n_y & -n_z & \mathbf{n}\cdot\mathbf{l} \end{pmatrix}$$

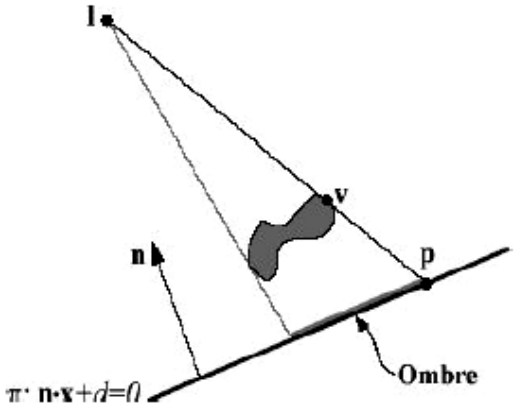


Figure 18: Représentant les différents éléments du problème. Elle correspond à la projection depuis la source lumineuse L de tout vertex sur le plan π [EMN08].

On rend donc tout d'abord toute la scène, l'occultant excepté. On rend ensuite l'objet sans illumination et de couleur sombre : le résultat est qu'un objet ayant la même forme et la même place que l'ombre apparaît sur le plan. Enfin on rend l'occultant lui-même, en ayant soin d'avoir remis la matrice monde a son état normal [Coh04].

Figure 19: Résultat du rendu de l'ombre d'une boite par projection planaire [Coh04].

5.1.4. La projection sur des surfaces courbes

Nous l'avons vu, une des limitations de la technique de la projection planaire est qu'une ombre ne se plaque que sur une surface plane. Une extension de cette méthode a été proposée [Ngu99] afin de pouvoir projeter une ombre sur tout type de surface [EMN08].

La technique consiste à rendre dans une texture l'occultant vue depuis la source lumineuse en noir ; les zones non ombrées par le l'occultant sont quant a elles blanches.

Figure 20: Les 3 étapes de la projection sur des surfaces courbes [EMN08].

On rend ensuite les objets recevant l'ombre en plaquant dessus la texture précédemment obtenue, projetée en utilisant un mode de projection et une matrice de texture calculé à partir des propriétés de la lumière source.
Enfin, on rend dans un dernier temps le bloqueur pour compléter la scène [EMN08].

5.2. 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 intérsectent 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 [Shi00].

Figure 21: Exemple de simulation du lancer de rayons [Rog07]

Malheureusement, cette simplicité se fait 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 sur ce sujet [WDP99] ; [PBMH02] ; [CHH02] ; [WDS04] ; [Amo04].

5.3. Méthodes d'illumination globale

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 entre les murs d'une pièce (figure 22). Les deux méthodes d'illumination globale les plus populaires sont la radiosité [CW93] ; [SP94] et le photon mapping [Jen01], elles 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].

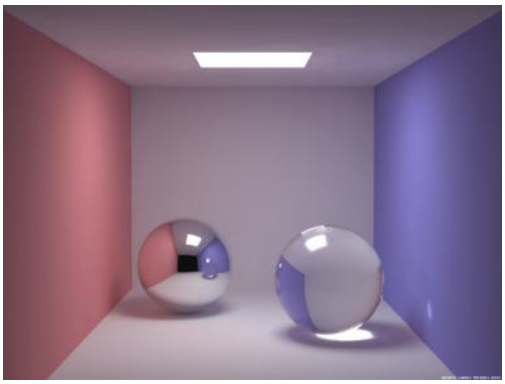


Figure 22 : Exemple d'illumination globale à l'aide du photon map [Jen96]).

5.4. Méthodes basées sur le tampon de profondeur

5.4.1 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 [Amo04] :

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) (figure 23).

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 celles qui sont 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 24).

Figure 23: Shadow mapping. À gauche, vue de la caméra. À droite , tampon de profondeur vu de la lumière, encodé en niveaux de gris [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 u ne 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 s hadow 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 25). Plusieurs techniques ont donc été proposées pour atténuer ce problème. Comme cet aliassage ne sera présent qu'aux frontières de l'ombre, un ré flexe 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 Filt ering (PCF) [RSC, 87], qui effectue 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 coût euses. Le matériel graphique moderne implante une version simplifiée de cet algorithme, sous la forme de filtrage bilinéaire des textures de profondeur.

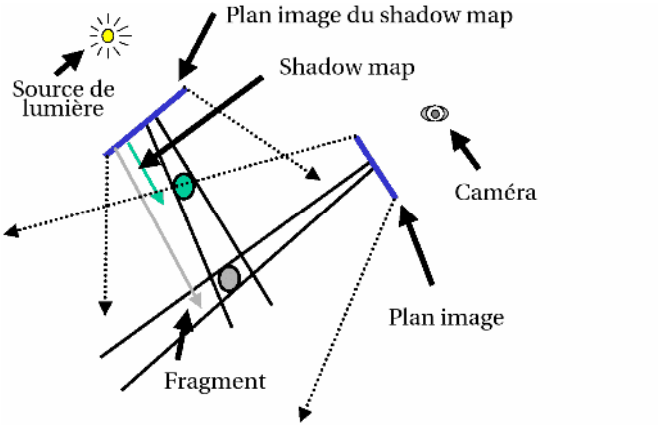


Figure 24: Diagramme illustrant l'algorithme de shadow mapping.

Il est aussi possible de minimiser l'aliassage, au lieu de le masquer. Fernando et al. [FFBG01] proposent les " adaptive shadow maps " qui subdivisent en quadtree le tampon de profondeur de façon à permettre un échantillonnage plus raf finé 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 (figure 25).

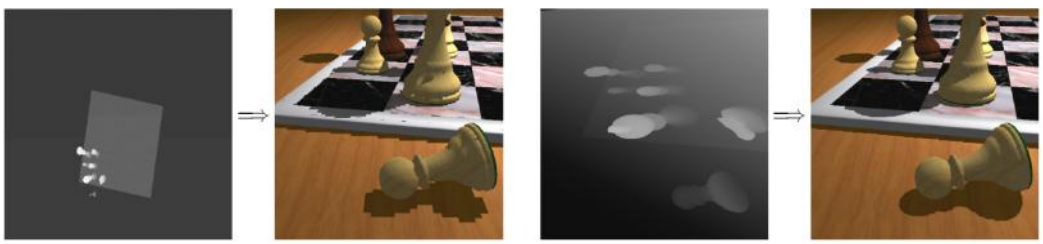


Figure 25 : Le tampon de profondeur en espace perspective. À gauche, un tampon de profondeur traditionnel avec ses problèmes d'aliassage. À droite, en espace perspective [SD02].

5.4.2 Ombres douces 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 douces 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 [Amo04].

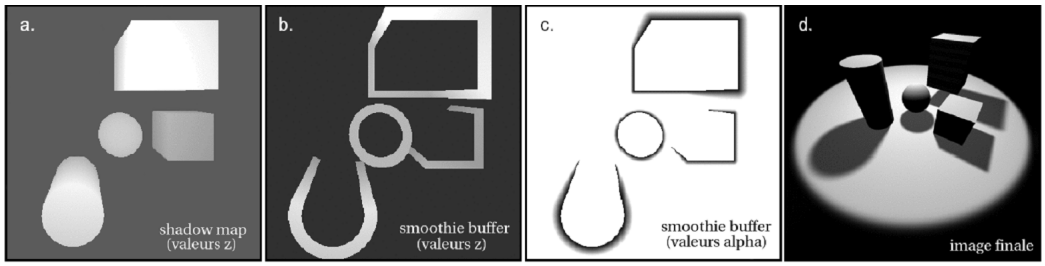


Figure 26: Algorithme des smoothies. (a) Création d'un shadow map standard. Puis, construction de primitives (smoothies) aux silhouettes. (b) Rendu des smoothies dans un tampon de profondeur. (c) Pour chaque pixel du smoothie buffer, stockage d'un alpha qui dépend du ratio de distances entre la source 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 [CD03].

Les smoothies [CD03] étendent la silhouette des bloqueurs, de façon à créer une texture projective de pénombre (voir la figure 26 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 (figure 27). 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.

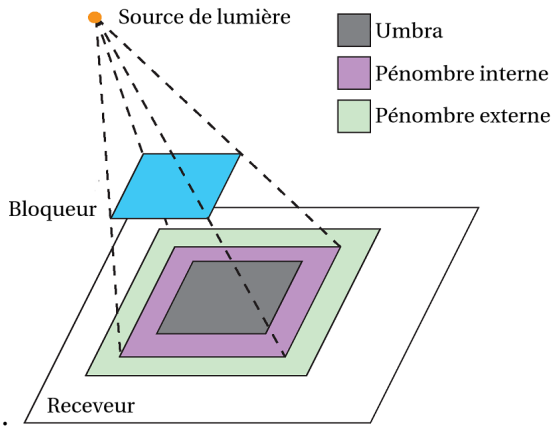


Figure 27: Pénombre interne et externe [HLHS03]. La partie interne de la pénombre est celle qui n'est pas visible à partir d'une lumière ponctuelle.

5.5 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 en 1977 [Cro77] et ensuite implantée à l'aide de matériel graphique par Heidmann en 1991 [Hei91]. Nous exposons en détail cette partie dans le prochain chapitre.

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 28). 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 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 28).

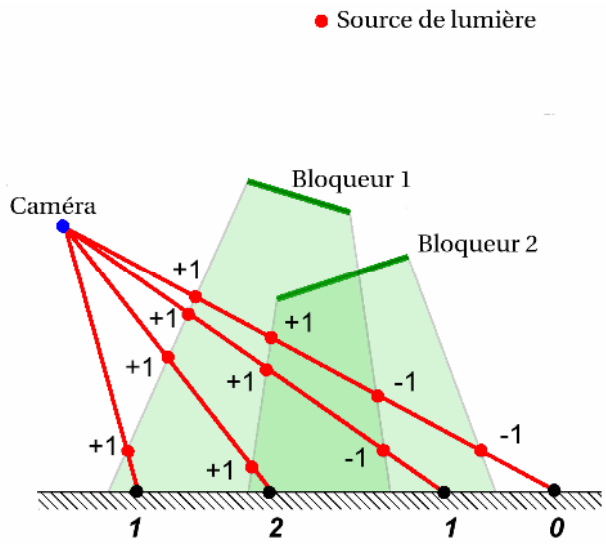


Figure 28: Illustration en 2D des volumes d'ombre [HLHS03].

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, 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.

6. Bilan

Nous présentons dans ce qui suit une synthèse des travaux présentés dans ce chapitre, en mentionnant leurs catégories communes, les points forts de chacun, les nouveautés introduites par chaque type (Tableau 2), ainsi que leurs avantages et leurs limites (Tableau 1).

	Avantages	Inconvénients
shadow volume	- génération et rendu interactif pour des scènes dynamiques ; - Traite les sources omni-directionnelles - Plus grande précision des ombres - Traite les cas d'auto-ombrage (self-shadowing) - Ombres précises - Positions quelconques lumière/ caméra - Robuste si bien programmé,	- requiert le calcul de la silhouette des bloqueurs ; - consomme une grande quantité de *fillrate* sur la carte graphique. - Temps de calcul dépend de la complexité des occultants (calcul de la silhouette, extrusion) - Création de gros polygones (extrusion) → diminue le fillrate de la carte graphique - Calcul de la silhouette (sur CPU très long) - Scènes spécifiques : modèles fermés, formés de convexes - Deux rendus de la scène, plus rendu du volume
shadow map	- Implémentation entièrement avec le hardware graphique - Création de la shadow map relativement rapide - Traite la plupart des cas d'auto-ombrage (self-shadowing) - Très simple à implémenter, code compact - Marche toujours (scène quelconque) - Prix indépendant de la complexité de la scène - Nombreuses variantes pour améliorer la qualité	- Problème d'aliasing - Ne traite pas les sources omnidirectionnelles (angle de la source limitée) - Au moins 2 passes sont nécessaires pour le rendu - Ne pas regénérer systématiquement la shadow map, seulement si la source se déplace - Besoin d'extensions OpenGL (disponibles ?) - Problèmes d'échantillonnage (*xy* et *z*)

Tableau 1: Synthèse sur les techniques de shadow Map et shadow volume

	type de données	forme du récepteur	gestion de l'auto-ombrage	vitesse	implémentation	espace
shadow volume	géométrique	quelconque	oui	- Le coloriage de l'ombre est lent (3 passes : 2 pour le calcul du stencil, 1 pour le remplissage). - Déterminer la silhouette est coûteux.	hardward	Objet
La projection planaire	géométrique	plan	non	très rapide : 1 passe	hardward	Objet
La projection sur des surfaces courbes	quelconque	quelconque	non	correcte : il faut rendre 2 fois les bloqueurs	hardward	Image
shadow map	quelconque	quelconque	oui	correcte (2 passes), indépendante de la complexité de la scène.	hardward	Image

Tableau 2: Synthèse sur les travaux de génération d'om bres

7. Conclusion

Le long de ce chapitre nous avons présenté une brève étude concernant l'illumination, nous pouvons aussi conclure la distinction entre le modèle d'ombrage et les ombres portées. L'utilisation des shadow Maps ou des volumes d'ombre aboutissent à la génération d'ombres en temps réel, bien que les shadow Maps souffrent de problème d'aliasing, donc l'utilisation des volumes d'ombre est le plus exigé.

Dans le prochain chapitre nous faisons une représentation détaillée sur les volumes d'ombre ainsi que les travaux relatifs.

Chapitre II :

Volume d’ombre

Volume d'ombre

Dans ce chapitre, nous allons définir les volumes d'ombre. De même, nous allons présenter quelques travaux sur celles-ci. Concernant le choix des travaux, celui-ci est basé sur leur popularité, ces travaux ayant été fait récemment. A la fin de ce chapitre, nous nous sommes proposé de présenter une synthèse de ses travaux selon un certain nombre de critères que nous définirons.

1. Rendu en temps réel

Le rendu en temps réel est une partie du domaine de l'infographie. Tandis que le rendu en synthèses d'images s'occupe à la suite de l'apparition des images produites, le rendu en temps réel se concentre principalement sur la création des images assez rapide pour maintenir l'observateur immergé dans la scène. Pour atteindre ce but, les images doivent être créées à un taux qui ne permet pas à l'observateur de distinguer les différentes images.

Le terme frame-rate est employé pour décrire le taux auquel des images sont affichées sur l'écran. Son unité est des images par seconde (fps). Haines et Akenine Möllher [AMH02] classifier une application en tant que temps réel si son frame-rate est au moins 15 fps.

2. Volumes d'ombre

Les volumes d'ombre (Shadow Volumes) sont proposé la première fois par Frank Crow en 1977 [Cro77], l'approche consiste à générer une ombre polygonale à partir

des objets occulteurs (bloqueurs) de la scène vis-à-vis d'une source de lumière ponctuelle, puis à afficher uniquement certaines parties de ces volumes, classiquement, celles comprises entre les faces avant (dont la normale serait orientée vers l'observateur) et les faces arrière (dont la normale serait orientée vers le sens opposé) des volumes d'ombre, tel qu'illustré sur la figure 29. L'algorithme consiste d'abord à trouver les arêtes silhouettes (i.e. les arêtes incidentes à deux polygones (ou triangles), l'un orienté vers la source, l'autre non) 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. Les objets se trouvant à l'intérieur d'au moins un volume d'ombre sont dans l'ombre, ceux à l'extérieur sont illuminés [Fré06].

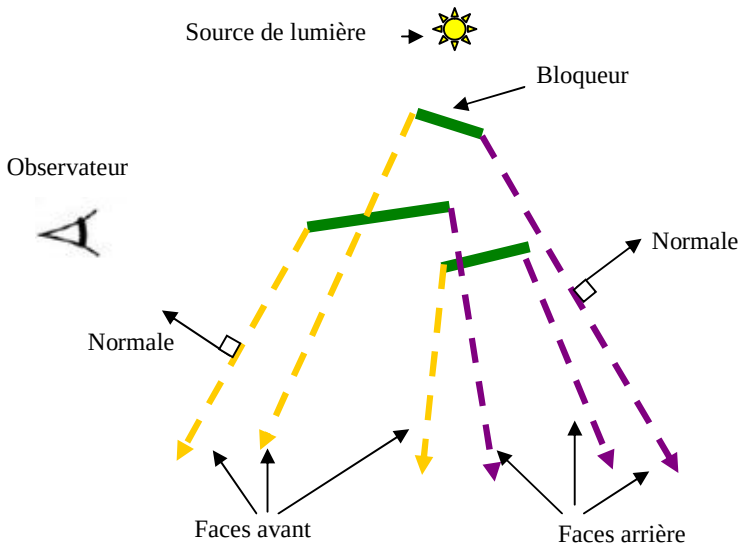


Figure 29: Illustration des faces avant et des faces arrière d'un volume d'ombre

Pendant la phase de rendu, pour déterminer si un point P ce trouve ou non à l'intérieur d'un volume d'ombre, nous traçons une demi-droite partant de P et passant par le point de vue de l'observateur. Pour tous les points où la demi-droite intersecte une face avant d'un volume d'ombre, nous incrémentons un compteur correspondant à ce point. Pour tous les points où la demi-droite intersecte une face arrière du volume d'ombre, nous décrémentons ce compteur. A l'issue de cette phase, si notre compteur est différent de 0, alors P est ombré, sinon P ne l'est pas, il

est dans la lumière. Nous réitérons cette procédure pour chaque point visible de la scène. Dans une utilisation temps réel, il est inutile d'utiliser un lancer de rayon explicite. Cette méthode peut être implémentée sur les cartes graphiques modernes en faisant usage du « Stencil-Buffer » [HLHS03].

Il existe un buffer particulier dans OpenGL : le Stencil-Buffer, c'est un masque de calcul qui permet d'éliminer d'un calcul les pixels marqués. L'idée est de construire un masque représentant l'ombre telle qu'elle est vue depuis la caméra et d'utiliser ce masque pendant les calculs d'éclairement.

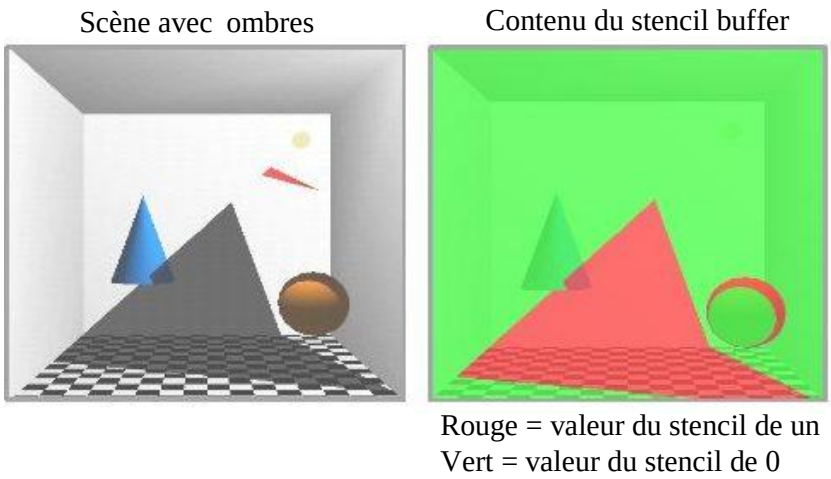


Figure 30: Utilisation du Stencil-Buffer

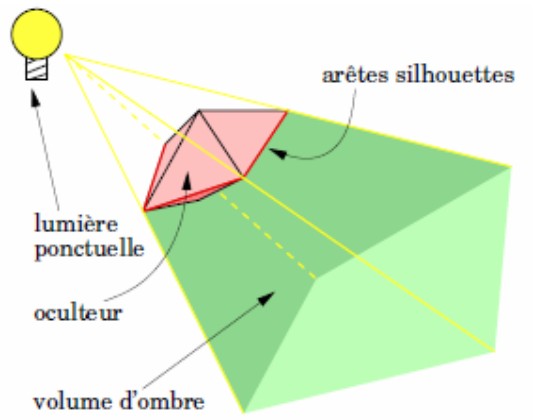


Figure 31 : Illustration d'un volume d'ombre [Hor06].

Dans la pratique, chaque point visible de la scène peut être assimilé à un pixel à l'écran, et il est courant de gérer ce système de compteurs par l'intermédiaire du

Stencil-Buffer. Tester l'intersection des demi-droites par des moyens purement géométriques est en effet fastidieux [DW04].

Mais l'algorithme de volume d'ombre exige que l'objet qui projette l'ombre doit être sous forme de maillage polygonal fermé. Chaque bord du maillage doit être partagé exactement par deux polygones. Il est également utile de se limiter aux mailles triangulaires, parce que le matériel graphique est optimisé pour le rendu de triangle.

La règle la plus contraignante est au niveau de la modélisation des objets 3D. Tous les modèles 3D ne sont pas candidats pour être shadow-caster (lanceur d'ombre: c'est l'objet qui va bloquer la lumière et créer l'ombre proprement dite). Pour pouvoir projeter une ombre volumique, les meshes (mailles) constituant un modèle doivent être fermés. Mais qu'est-ce donc un mesh fermé?

Un mesh est une surface composée d'un maillage triangulaire. Chaque triangle représente une face et est composé de 3 sommets et 3 arêtes. Le mesh le plus simple que l'on puisse faire est un triangle [JJG05]. Un mesh est dit fermé si chaque arête (edge) est partagé par 2 faces (chaque arête possède 2 faces adjacentes). La sphère est l'exemple parfait d'une surface fermée. Nous ne pouvons pas y trouver de côté qui n'appartient qu'à une seule face. Un plan, au contraire, est l'exemple même d'une surface non fermée (ou surface ouverte): chaque côté des bords appartient à une face et une seule.

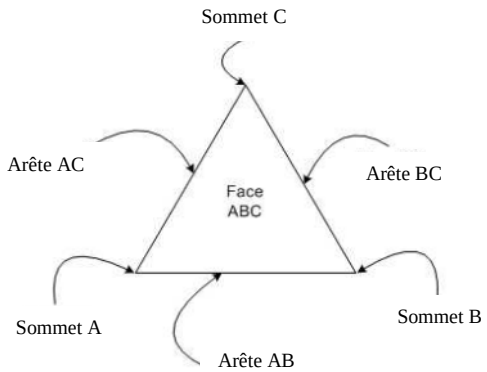


Figure 32 : Le triangle composant de base d'un mesh

Un côté qui ne possède qu'une seule face est appelé un crack. Donc un mesh contenant des cracks ne pourra pas projeter correctement des ombres volumiques.

Cette limitation provient de l'algorithme de détection de silhouette qui nécessite de s surfaces fermées. Par exemple le modèle de la théière, que l'on peut générer rapidement avec 3D Studio MAX, possède des cracks. Le couvercle de la théière est une surface non fermée. Le résultat de la présence de ces cracks est visible sur la figure 33. Donc la modélisation en vue de projeter des ombres volumiques doit se faire en gardant à l'esprit la contrainte de surface fermée .

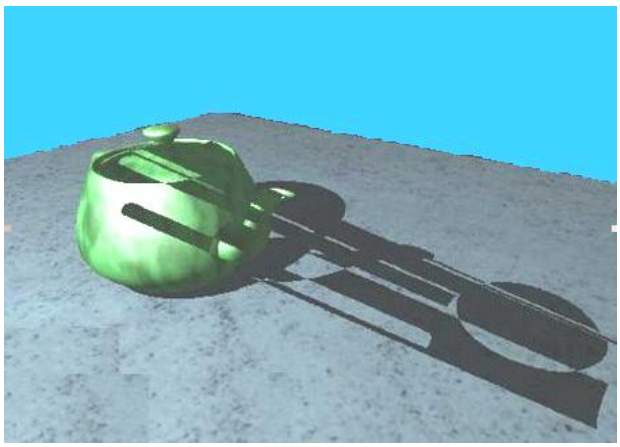


Figure 33: Les surfaces non fermées provoquent des défauts visuels [JJG05]

3. Calcul du volume d'ombre

De nombreuses méthodes ont été développées pour permettre le calcul rapide des parties ombrées d'une séquence d'images. Une de ces méthodes est connue sous le nom de Stencil Shadow Volumes. Elle utilise la géométrie de la scène 3D pour en extraire les volumes de l'espace non éclairé par une source lumineuse ponctuelle. Ces volumes sont ensuite dessinés dans une image appelée « Stencil-Buffer » pour y obtenir un masque indiquant quels pixels sont dans l'ombre. Il existe deux techniques de base pour dessiner ces volumes d'ombre dans le Stencil-Buffer [Hor06]. Elles sont connues sous les noms de Z-Pass [Hei91] et Z-Fail [Car00]. La première technique (Z-Pass) présente quelques défauts dont la correction est faite par la deuxième technique appelée Z-Fail.

L'algorithme consiste d'abord à trouver les silhouettes des occulteurs du point de vue de la lumière, la façon la plus simple est d'identifier les segments (arêtes) partagés par un polygone face avant et un autre face arrière, puis à faire une

extrusion des silhouettes le long de la direction de la lumière (figure 31), créant ainsi les volumes d'ombre, et savoir si un point est dans l'ombre revient à savoir si ce point est dans le volume. Les objets se trouvant à l'intérieur d'au moins un volume d'ombre sont dans l'ombre, ceux à l'extérieur sont illuminés. 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 du volume d'ombre (les faces avant et fac es arrière) peuvent incrémenter ou décrémenter le compteur par rapport à la technique utiliser Z-Pass ou Z-Fail. Si le compteur est positif à la fin, le point est dans l'ombre.

3.1. La technique de base : Z-pass

En 1991, Tim Heidmann a présenté la technique de base pour le rendu des volumes d'ombre est communément appelée méthode Zpass [Hei91]. Ce nom provient de l'utilisation de la fonction Z-Pass lors de la modification du tampon de stencil (ou Stencil-Buffer) [Car00].

Déroulement de l'algorithme Z-Pass

La scène est rendue de la façon suivante :

- L'ensemble de la scène est dessiné dans le Frame-Buffer en utilisant uniquement la lumière ambiante. Les informations de couleurs sont stockées dans le tampon de couleur. Pour chaque pixel, la profondeur associée au point correspondant est stockée dans le Z-Buffer.
- Le Stencil-Buffer est initialisé à 0. Les mises à jour du Z-Buffer et du tampon de couleur sont désactivées.
- Les faces avant des volumes d'ombre sont dessinées. Quand un point d'une face avant d'un volume d'ombre passe le test de profondeur, nous incrémentons le compteur du pixel correspondant, dans le Stencil-Buffer.
- Les faces arrière des volumes d'ombre sont dessinées. Quand un point d'une face arrière d'un volume d'ombre passe le test de profondeur, nous décrémentons le compteur du pixel correspondant, dans le Stencil-Buffer.
- La scène est à nouveau dessinée vers le Frame-Buffer, mais seulement aux endroits où le Stencil-Buffer est à 0. Seul les composantes diffuses et spéculaires de la lumière sont activées. Ce rendu s'effectue sur le rendu de l'étape 1, et a pour but d'illuminer les parties non ombrées de la scène .

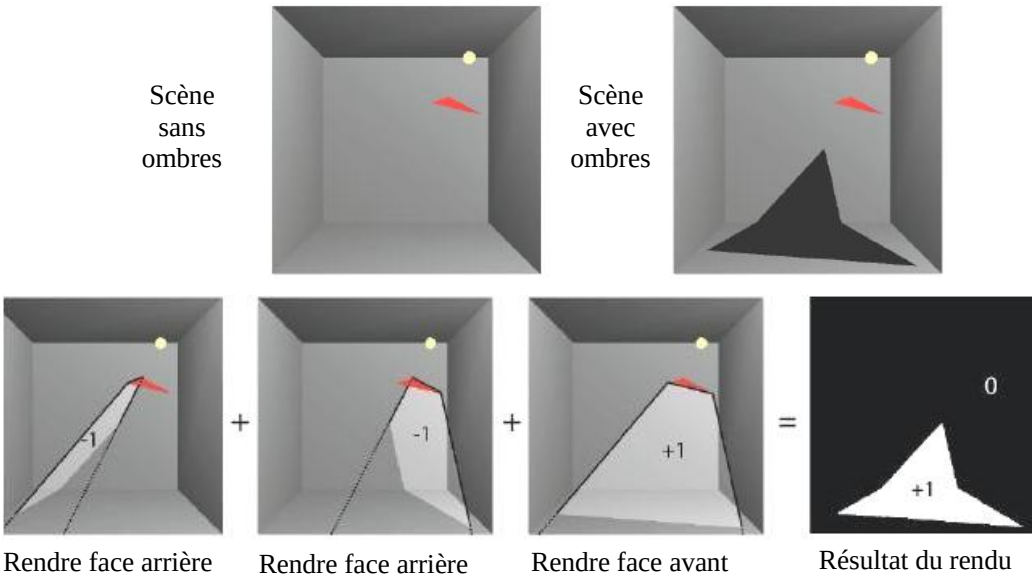


Figure 34 : Le déroulement de l'algorithme de Z-Pass [EASW10].

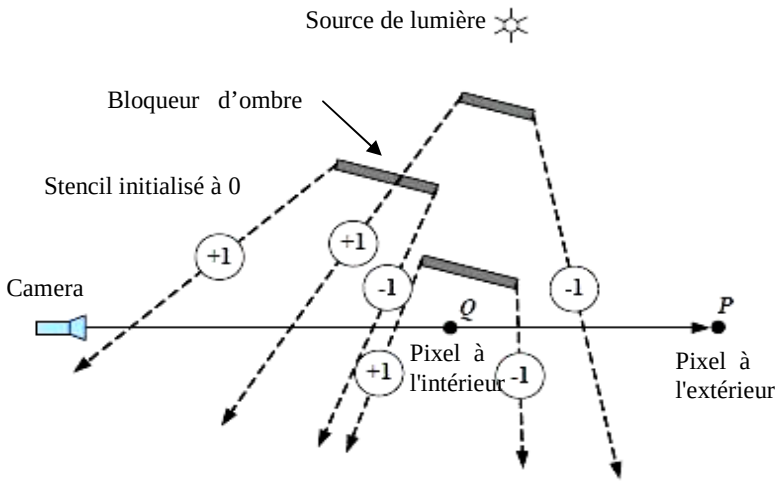


Figure 35 : Illustration en 2D des volumes d'ombre Méthode Z-Pass [DZY08].

Par exemple dans la figure 35 la valeur du stencil du pixel Q est +1+1-1+1= 2, signifie que Q est dans l'ombre, et la valeur du stencil du pixel P est +1+1-1+1-1-1=0 signifie que P est dans la lumière.

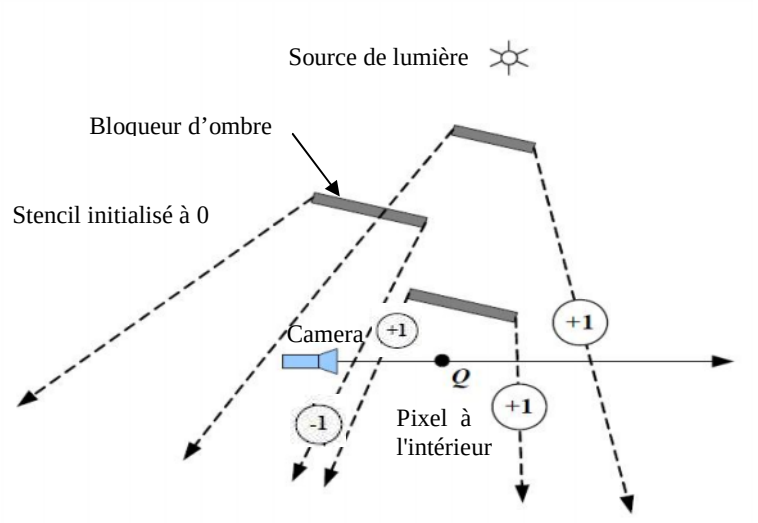


Figure 36 : Méthode Z-Pass, la camera dans le volume d'ombre [DZY08].

Cette technique permet un rendu de scène avec de multiples sources de lumières mais a pour inconvénient le fait qu'elle ne produit pas de bon résultat lorsque l'observateur (caméra) se trouve dans une zone d'ombre (figure 36), la valeur du stencil du pixel Q est -1+1= 0, signifie que Q est en dehors de l'ombre, mais en réalité il est à l'intérieur de l'ombre.

La technique de Z-pass ne marche plus aussi quand le volume d'ombre traverse le plan avant (du système de visualisation) et plus précisément, le rectangle avant (la figure 37).

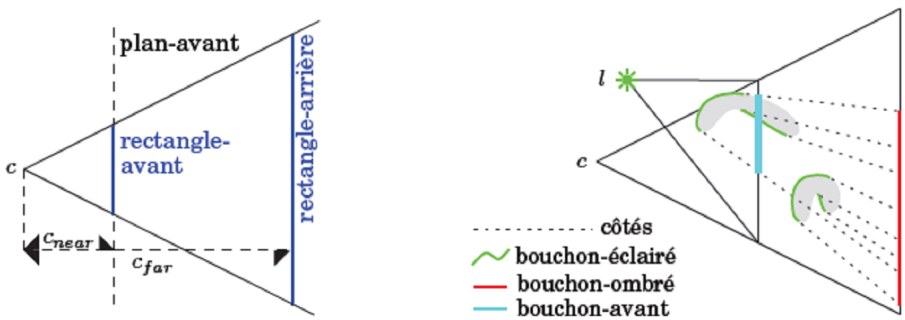


Figure 37: Terminologie utilisée. c : est la position de la caméra. l : est la source lumineuse ponctuelle. L'image de droite montre les différents éléments qui composent un volume d'ombre : le bouchon-éclairé est l'ensemble des polygones faisant face à la lumière. Le bouchon-ombré est l'ensemble des polygones qui

«tournent le dos » à la caméra et projetés sur le plan-arrière. Le bouchon-éclairé et le bouchon-ombré sont utilisés par la méthode Z-fail. Le bouchon-avant est l'intersection du volume d'ombre avec le plan-avant de la caméra. Les côtés du volume d'ombre (sides) sont utilisés dans les deux méthodes Z-Pass et Z-Fail.

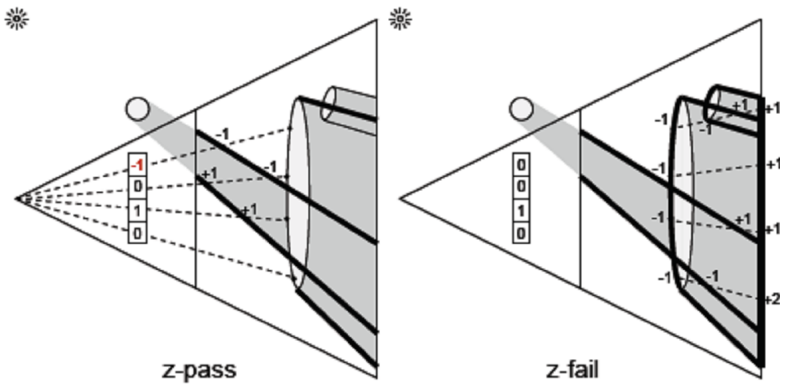


Figure 38: La méthode Z-Pass dessine d'abord la scène dans le Z-buffer, puis dessine les volumes d'ombre (en gras), en incrémentant ou décrémentant le stencil buffer quand un fragment passe le test en Z. Z-Pass ne donne pas une ombre correcte quand le volume d'ombre est coupé par le plan avant du système de visualisation.

En effet, Z-Pass suppose implicitement que ce plan avant est en dehors de toute ombre. Dans l'exemple de la figure 38 (gauche), le fragment le plus en haut est classifié incorrectement comme étant dans l'ombre car le volume d'ombre créé par le cercle est coupé par le plan avant. Ainsi, une partie du volume d'ombre n'est pas détectée par le rayon lancé depuis la caméra et le compte des entrées/sorties devient faux. De nombreuses méthodes ont été proposées pour boucher le volume d'ombre coupé à l'aide de polygones additionnels [BJ99], [Dif96], [Kil01]. Mais, elles s'avèrent coûteuses en temps de calcul et souffrent toutes de problèmes de robustesse qui font apparaître des artefacts très visibles [Hor06].

Ce problème de coupe par le plan avant a mené certains, dont John Carmack [Car00] en 1999, à examiner la méthode dite « Z-Fail » qui modifie le stencil buffer pour les fragments du volume d'ombre qui échouent au test de profondeur, voir figure 38 (droite). Nous verrons que Z-Fail déplace le problème du plan avant au plan arrière. Mais, cette spécificité permet de le traiter de manière robuste en déplaçant le plan arrière à l'infini [EK03] (pour Z-Pass, il n'est pas possible de

déplacer le plan avant à distance nulle de l'observateur). Cependant, cette robustesse est obtenue aux dépens des performances.

3.2. La technique : Z-Fail

Elle consiste en un test de stencil différent: l'incrémenta tion et la décrémentation du tampon de stencil n'est effectué qu'en cas d'échec du test de profondeur, d'où son nom " Z-Fail ". Cette méthode d'ombrage est dite robuste car elle est utilisable quelle que soit la position de la caméra ou de la lumière, que se soit pour une scène en intérieur ou en extérieure, et cette technique est plus connu sous le nom de (Carmack's Reverse) [Car00].

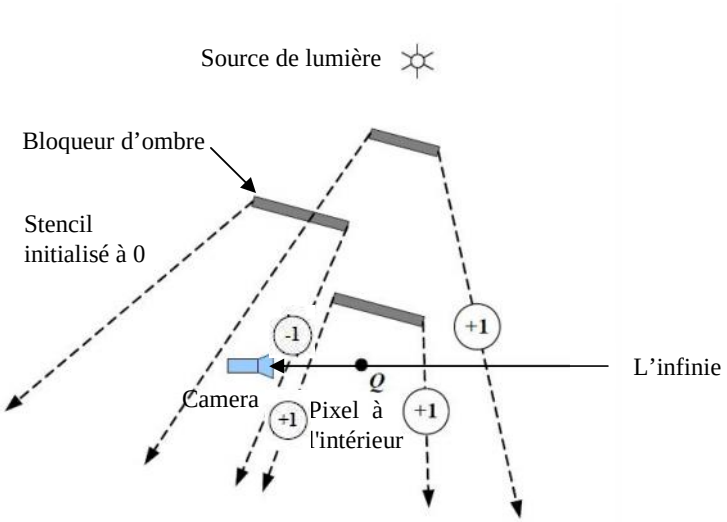


Figure 39 : Méthode Z-Fail, la camera dans le volume d'ombre [DZY08].

Déroulement de l'algorithme Z-FAIL

- 1ère étape : On affiche notre scène dans son intégralité mais sans ombre. Cet affichage met à jour le tampon de profondeur qui sera utilisé dans l'étape 3.
- 2ème étape : On désactive la mise à jour du tampon de profondeur et du tampon de couleur. On active le tampon de stencil avec une fonction de comparaison qui laisse passer tous les fragments.
- 3ème étape : On affiche le volume d'ombre créé par les objets éclairés en deux passes :

- Lors de la 1ère passe, on affiche les faces arrière du volume d'ombre en incrémentant le stencil-buffer lorsque le test de stencil échoue.
- Lors de la 2éme passe, on affiche les faces avant du volume d'ombre en décrémentant le stencil-buffer lorsque le test de stencil échoue.
- Une fois ce rendu effectué, seul les zones d'ombres de la scène ont une valeur différente de la valeur de référence dans le tampon de stencil.

➢ 4ème étape : Comme dans la méthode classique, on réactive le tampon de couleur. On affiche un rectangle semi-transparent qui couvre la totalité de l'écran par-dessus notre scène. Ce rectangle est de la couleur de l'ombre. On affiche tous les pixels du rectangle où la valeur de stencil est différente de la valeur de référence c'est-à-dire les parties non ombrés.

Dans la figure 39 partant de l'infinie jusqu'au pixel Q, la valeur du stencil du pixel Q est : +1+1= +2, signifie que Q est dans l'ombre. Dans ce cas le résultat est correct.

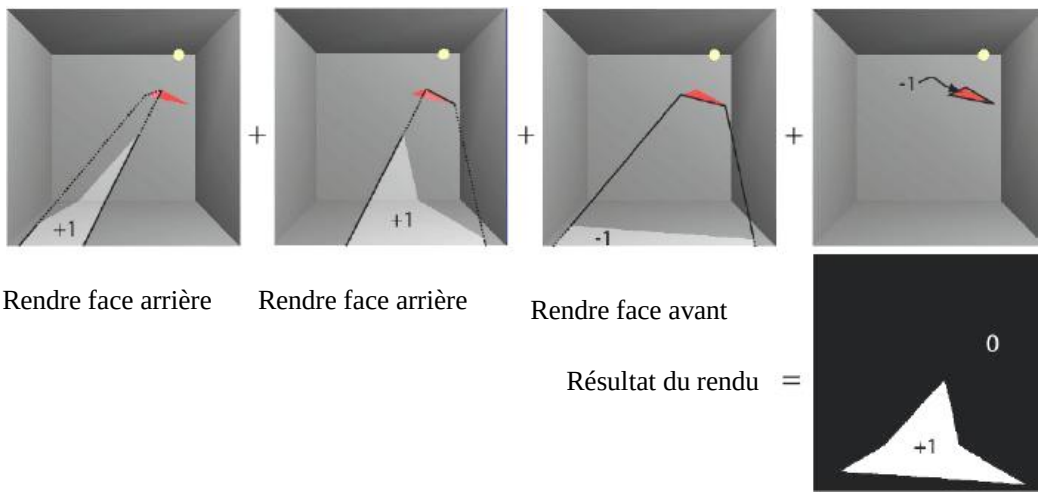


Figure 40: Le déroulement de l'algorithme de ZFail [EASW10].

Cet algorithme, supprime le problème lié au plan-avant mais le replace au niveau du plan-arrière. Conceptuellement, Z-Pass compte les entrées et sorties du volume d'ombre pour un rayon partant de la caméra vers l'infini (comme pour le ray-tracing). Le comptage, pour Z-Pass s'arête dès lors que le rayon atteint un objet de la scène. Symétriquement, Z-Fail compte ces entrées-sorties pour un rayon allant de l'infini vers la caméra, et s'arête dès lors que le rayon n'est plus invisible pour la

caméra. Cette symétrie fait clairement apparaître que le problème de coupe du volume d'ombre s'est maintenant déplacé vers le plan-arrière.

Un peu plus tôt, Bilodeau et Songy [BS99] avaient développé une idée similaire appelée maintenant reversed Z-Pass. Au lieu de compter les fragments qui ne passent pas le test de profondeur (comme Z-Fail), leur méthode inverse le test de profondeur et compte les fragments qui passent ce test inversé. Bien que très similaire, Z-Fail est plus efficace que la méthode du reversed Z-Pass sur les GPU actuels qui stoppent le traitement d'un fragment aussitôt que le test en profondeur échoue (Z-Fail).

Le problème de coupe du volume d'ombre par le plan-arrière nous empêche toujours d'obtenir le bon compte dans le stencil buffer. Comme illustré sur la figure 37 (à droite) et la figure 38 (à gouche), nous devons, pour Z-Fail, fermer le volume d'ombre à l'aide du bouchon-éclairé et du bouchon-ombré. Si le bouchon-ombré est coupé par le plan-arrière, l'intérieur du volume d'ombre est exposé et des valeurs incorrectes sont produites dans le stencil buffer car l'entrée du rayon (conceptuel) dans le volume d'ombre ne sera pas détectée pour certains pixels. Everitt et Kilgard [EK03] résolvent ce problème en repoussant le plan-arrière à l'infini grâce à une matrice de projection modifiée, et en poussant de même le bouchon-ombré à l'infini (opération simplement réalisée à l'aide de coordonnées homogènes). Cette méthode Z-Fail, bien que robuste, ajoute à Z-Pass le dessin, coûteux en temps, du bouchon-éclairé et du bouchon-ombré, qui doivent être traités séparément puisque le bouchon-ombré est éloigné à l'infini de sa position originale. Cette séparation des deux bouchons en fonction de l'orientation des polygones d'un objet, empêche de prendre avantage de la structure du maillage à l'aide, par exemple, de triangle-strips ou de Vertex Buffer Object[1].

4. Les optimisations

Les volumes d'ombre doivent être optimisés afin d'avoir des rendus en temps réel. Ces optimisations peuvent être réalisé par l'évitement de certains calculs, utiliser les extensions des OpenGL ou par l'utilisation des shaders.

[1] Un VBO (Vertex Buffer Objet) est un tableau de coordonnées 3D résidant en mémoire vidéo. Il bénéficie donc d'un accès plus rapide par le GPU.

Eviter certains calculs par des optimisations pour accélérer l'affichage des scènes ombrées. La première optimisation consiste à détecter la présence de la camera à l'intérieur du volume d'ombre pour pouvoir déterminer s'il est nécessaire ou non de fermer les volumes. Il n'est pas nécessaire de recalculer en permanence les silhouettes et les volumes d'ombre lorsque les lumières et les objets de la scène sont fixes.

Utilisation des shaders : L'extrusion des arêtes du volume d'ombre peut être aussi effectué grâce au vertex shaders, ou sous OpenGL. De même pour le calcul des silhouettes des ombres.

4.1. Optimisation par le Stencil

Récemment, de nombreux travaux ont tenté avec succès d'améliorer la méthode des volumes d'ombre. Lloyd et al. [LWGM04] proposent une technique pour réduire le taux de remplissage (fill-rate) nécessaire au dessin des volumes d'ombre en rejetant ceux qui n'ont pas d'influence sur le résultat final, et en les coupant (clipping) pour ne garder que la partie des volumes nécessaire au calcul correct des pixels ombrés. Aila et Akenine- Möller [AAM04] utilisent une méthode hiérarchique qui ne dessine les volumes d'ombre que sur les bords de l'ombre. De manière similaire, Chan et Durand [CD04] réduisent le taux de remplissage (fill-rate) en utilisant d'abord une carte d'ombre (shadow map) pour déterminer l'ensemble des pixels se trouvant proches du bord de l'ombre, où une plus grande précision est requise. Cet ensemble de pixels est alors utilisé comme un masque, qui indique les pixels devant être modifiés lors du dessin des volumes d'ombre. Enfin, Aldridge et Woods [AW04] proposent une méthode robuste pour produire des ombres correctes pour des objets non-manifold (qui ne possède pas un volume interne, comme par exemple un plan).

4.2. Utiliser les extensions d'OpenGL

Les cartes graphiques récentes de NVIDIA implémentent des extensions OpenGL permettant d'accélérer le rendu. GL_EXT_stencil_two_side permet de ne rendre les faces avant et arrière une seule fois dans le stencil buffer. GL_EXT_stencil_wrap évite la saturation du stencil buffer. L'extension GL_NV_depth_clamp qui, au lieu de supprimer les fragments coupés (Clipping) par les plans avant et arrière du frustum

(pyramide du vue), tronque leur valeur de profondeur à celle du plan de coupe le plus proche et dessine effectivement le fragment avec sa nouvelle profondeur. Cette extension serait très utile pour garantir que l'occultant n'est pas coupé par le plan-avant de la lumière. Cependant, en activant cette extension d'OpenGL, les fragments se trouvant derrière le plan-arrière de la lumière resteraient eux aussi visibles, ce que nous ne désirons évidemment pas. Nvidia vient aussi de créer la technologie Ultra Shadow qui permet de limiter la zone de rendu des volumes d'ombre. Malheureusement, ces extensions ne sont pas disponibles sur les cartes graphiques de tous les constructeurs.

4.3. ZPass+

Le nouvel algorithme ZP+ [HHL05] (Z-pass-plus), allie la robustesse de Z-Fail à une vitesse plutôt comparable à la méthode Z-Pass. La méthode ZP+, illustrée sur la figure 41 (droite), construit, lors d'une passe de dessin préliminaire, un frustum de vue cisaillé (sheared frustum) qui s'étend depuis la lumière jusqu'au rectangle-avant. Puisque le rectangle-arrière de ce nouveau frustum est aligné avec le rectangle-avant de la caméra, il contient exactement la portion de la scène qui génère un volume d'ombre coupé sur le rectangle-avant initial. Une fois ce frustum de vue mis en place, le fait de dessiner la scène projette ses fragments sur le plan-avant de la caméra exactement là où les volumes d'ombre étaient coupés. Ce dessin permet donc de réparer les erreurs induites par la méthode Z-Pass, et rend cette dernière robuste.

Mentionnons que suite à ces travaux, Samuli Laine [Sam05] propose une méthode pour combiner ZP+ et Z-Fail en utilisant une technique dans les zones de l'image où elle est plus efficace que l'autre. La technique de Laine subdivise l'image en carreaux de petite taille (8 × 8) et permet de décider dans chaque carreau, selon la configuration géométrique de la scène, s'il vaut mieux utiliser Z-Fail ou ZP+ pour minimiser le nombre de pixels touchés au cours du dessin des ombres.

Bien que ZP+ soit mathématiquement exacte, la nature discrète du calcul par ordinateur génère quelques petits artefacts. ZP+ est en général plus rapide que Z-fail et se comporte bien mieux lorsque les occulteurs sont complexes, mais ZP+ peut être plus lent que Z-fail dans certains cas.

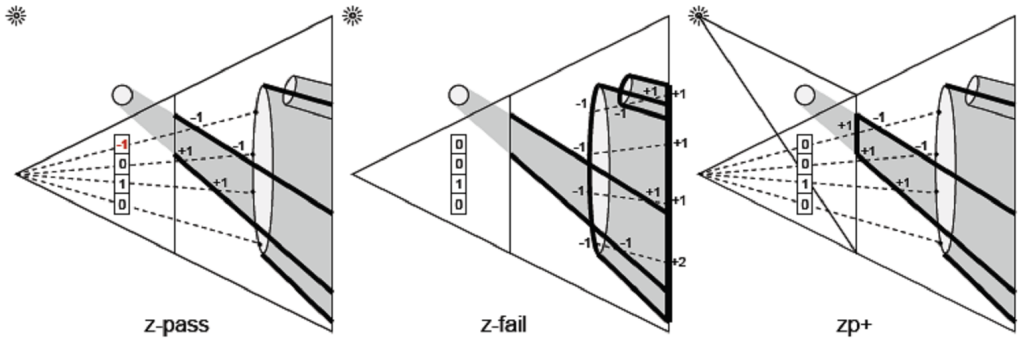


Figure 41: À droite, ZP+, initialise le stencil buffer en dessinant la scène vue depuis la lumière pour positionner le stencil à la valeur donnée par Z -pass en l'absence de plan-avant. Ainsi, ZP+ rend Z-Pass aussi robuste que Z-Fail.

4.4. Reconstruction du volume d'ombre des cartes en profondeur

Michael McCool était le premier à proposer un algorithme hybride combinant les shadow maps (cartes d'ombre) et les volumes d'ombre. Il a présenté son algorithme pour reconstruire des volumes d'ombre d'une carte en profondeur (depth map) en 2000 [MD00]. Celui-ci n'exige pas une représentation polygonale de la scène. Au lieu de cela, quelque peu semblable à la méthode de carte d'ombre, elle exige une carte en profondeur rendue depuis de source lumineuse. L'information de carte d'ombre est alors employée pour reconstruire une frontière polygonale de volume d'ombre qui peut être combinée avec une carte en profondeur vue depuis l'observateur en utilisant seulement un bit du Stencil-Buffer. La reconstruction est basée sur la détection d'arête dans la carte d'ombre. Les arêtes de silhouette identifiés sont alors utilisées pour construire une maille polygonale représentant les frontières du volume d'ombre. Cet algorithme n'a besoin d'aucune extension du matériel, par conséquent il peut être employé avec n'importe quel matériel qui produit de carte en profondeur correcte.

L'observation principale de l'algorithme est le fait que les échantillons de profondeur dans la carte d'ombre, en conjonction avec leurs coordonnées de pixel, décrivent des points de scène relativement à la position de lumière. Ces points peuvent être joints dans une maille polygonale puis transformés de nouveau dans

l'espace du monde, utilisant l'inverse de la projection du shadow map. Les surfaces qui sont construites de cette façon définissent les frontières entre l'ombre et la lumière dans la scène.

Dans l'algorithme du volume d'ombre, les frontières du volume d'ombre peuvent être englobés, par conséquent la nécessité d'additionner des polygones de face avant et de soustraire des polygones de face arrière. Avec la technique de McCool, les frontières du volume d'ombre se composent d'une surface en forme d'étoile simple. Dans ce cas-ci, il est seulement nécessaire de maintenir la parité du nombre de polygones du volume d'ombre devant la surface à chaque pixel. Si le nombre d'intersections de frontière du volume d'ombre le long d'un rayon de l'oeil au premier point de surface à un pixel est impair, le pixel est dans l'ombre. Autrement, il est illuminé. Pour maintenir la parité, un à bit unique dans le Stencil-Buffer il est suffisant. Ce peu est simplement inversé toutes les fois qu'un fragment de polygone d'ombre est dessiné sur lui.

L'algorithme hérite de tous les avantages de l'algorithme de carte d'ombre. Un avantage de cette approche hybride est que les volumes d'ombre produits sont en taille minimal. Ils s'étendent de la surface du bloqueur à la surface du récepteur d'ombre, tandis que les volumes d'ombre de Crow [Cro77] s'étendent vers l'extérieur du plan de vue lointain après le coupage (Clipping). Cependant, la méthode hybride hérite également de certains des inconvénients de l'algorithme de carte d'ombre. L'aliassage est toujours un problème, de même que le problème de précision de Z-buffer et de la carte d'ombre. Un autre inconvénient est le grand nombre de polygones d'ombre, qui est hérité de l'algorithme de base du volume d'ombre.

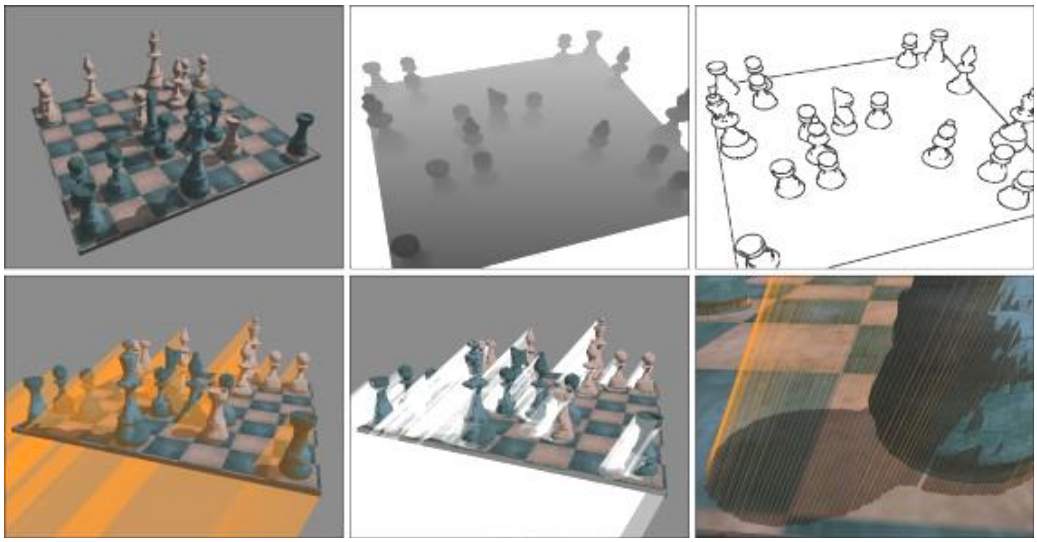


Figure 42: Reconstruction des volumes d'ombre des cartes en profondeur. De gauche à droite et jusqu'au bas : la scène, carte en profondeur créée de la position de lumière, bords a détecté dans la carte en profondeur, volumes d'ombre reconstruits, volumes d'ombre créés à partir de la géométrie des occulteurs de lumière (utilisant les bords potentiels de silhouette), modèle en fil des volumes d'ombre reconstruits en détail.

4.5. Un algorithme hybride efficace du rendu d'ombre

Avec les shadow maps, l'aliassage est seulement visible aux discontinuités entre les régions ombragées et illuminés, c.-à-d., aux silhouettes d'ombre [SCH03]. De l'autre côté, les volumes d'ombre calculent des ombres précises à chaque pixel, mais cette précision est en fait seulement nécessaire aux silhouettes d'ombre. Cette observation suggère un algorithme hybride qui utilise l'algorithme précis de volume d'ombre seulement aux silhouettes d'ombre et l'algorithme plus rapide de shadow maps partout dans la scène. En fait, cette approche évite non seulement des problèmes d'aliassage avec les shadow maps, mais également les coûts élevés du fill-rate de l'algorithme standard de volume d'ombre de Crow [Cro77].

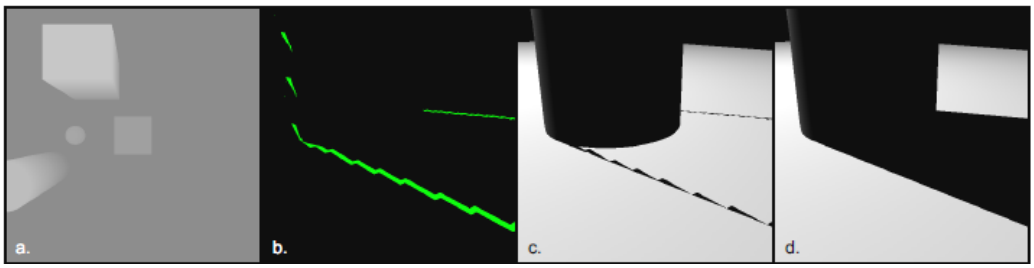


Figure 43: Dans la première étape, un chadow map est créée (a) pour identifier les pixels dans l'image qui se trouvent près des silhouettes d'ombre. Ces pixels, v us du point de vue de l'observateur, sont ombragés en vert (b). Après, nous rendons des volumes d'ombre seulement à ces pixels pour obtenir les bords précis d'ombre (c). Sinon nous employons le shadow map pour calculer des ombres partout, et le résultat final apparaît en (d) [CD04].

La technique présentée a été développée en 2004 par Eric Chan et Frédo Durand [CD04]. Dans la première étape, un shadow map ordinaire est crée. Ce shadow map sert à identifier des pixels de silhouette d'ombre et à calculer des ombres pour des pixels non-silhouette. Dans la deuxième étape, la scène est rendue du point de vue de l'observateur. Après, il suffit de rendre uniquement les volumes d'ombre à ces pixels pour obtenir les bords précis d'ombre, chaque fragment où sa profon deur est comparée par les quatre échantillons les plus proches de profondeur provenant d u shadow map [SCH03]. Si les résultats de comparaison conviennent, le fragment est un pixel de non-silhouette et est ombragé selon le résulta t de comparaison de profondeur. Sinon, le fragment est classifié comme silhouette. Pendant cette étape, le Z-Buffer standard est aussi bien exécuté, pour créer la carte en profondeur requise pour dessiner les volumes d'ombre dans la prochaine étape. Après que des pixels de silhouette aient été identifiés, des volumes d'ombre sont dessinés selon l'algorithme standard de volume d'om bre. La différence principale dans cette technique, est qu e les volumes d'ombre sont rasterizer seulement aux pixels de silhouette. La scène ombragée est rendue seulement aux pixels avec une valeur de stencil de 0, évitant de ce fait les régions ombragées dans la scène.

Cette technique se fonde fortement sur le matériel graphique pour jeter les fragments inutiles du volume d'ombre dès que possible dans le pipeline graphique. Pour limiter la rasterization des volumes d'ombre et des mises à jour d u stencil aux

régions contenant des pixels de silhouette, Chan et Durand proposent l'utilisation des masques de calcul. Un masque de calcul est un dispositif qui perm et le masquage des adresses spécifiques de frame-buffer, de sorte que le matériel puisse éviter de traiter des pixels à ces endroits [CD04]. Le matériel courant graphiques n'expose pas directement des masques de calcul. Cependant, l'extension EXT_depth_bounds_test d'OpenGL peut être employée pour simuler le comportement. L'idée est d'employer un pixel shader pour masquer les pixels en plaçant leurs valeurs de profondeur à une valeur constante en dehors des limites de profondeur.

4.6. CC Shadow Volumes

Lloyd et al en 2004 [LWGM04] présentent une méthode pour accélérer les volumes d'ombre dans les scènes. Cette méthode diminue le coût de la rasterisation par deux techniques différentes : volume d'ombre culling et clamping (CC) en réduisant le rendu inutile. De cette façon, la rasterisation des volumes d'ombre sur de grandes régions de l'espace vide est évité.

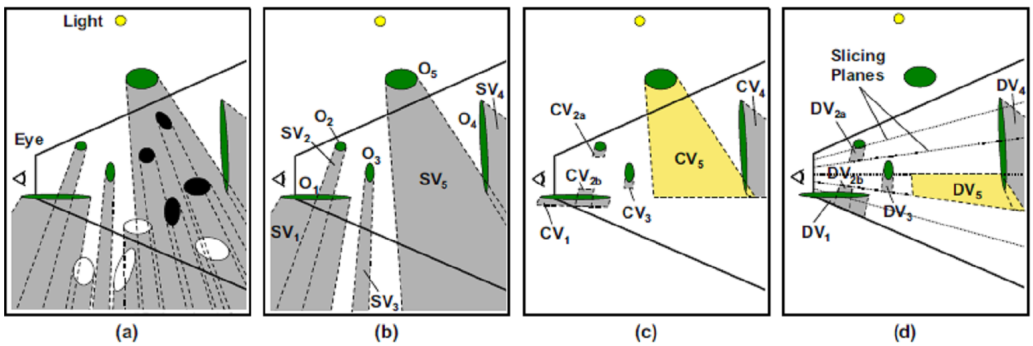


Figure 44 : CC Shadow Volumes accélèrent l'algorithme standard du volume d'ombre en culling d'abord les occulteurs inutiles et en maintenant ensuite des volumes d'ombre de survie pour s'adapter étroitement autour de la géométrie de scène. De gauche à droite : volumes standard d'ombre, volume d'ombre culling, volume d'ombre Clamping continu, volume d'ombre Clamping discret [LWGM04].

4.6.1. Culling

Ce que l'on appelle communément culling en programmation 3D, est le fait d'éliminer prématurément des parties de la scène qui ne seront pas visibles à l'écran ;

soit parce qu'elles sont cachées, soit parce qu'elles ne sont pas dans le ch amp de vision. Govindaraju et al. en 2003 [KSYM03] éliminent les occulteurs qui sont eux-mêmes complètement dans l'ombre, et éliminent également les occulteurs d'ombre dont les ombres ne sont pas visibles par l'oeil. Le culling de volume d'ombre est montré dans figure 44 (b). L'étape de culling des volumes d'ombre généralement produite en deux étapes figure 44 (a) et (b). D'abord, tous les objets qui sont visibles à partir de l'oeil sont détectés, fournissant l'ensemble de récepteurs d'ombre potentiel (PSR). De même, dans la deuxième étape on détecte tous les objets qui sont visibles du point de vue de la source lumineuse. Prenant les objets identifiés précédemment dans le PSR en considération, ces objets peuvent être en plus limités à ceux qui occluent réellement tous les objets visibles de l'oeil. La deuxième étape fournit l'ensemble de bloqueur potentielle (PSC).

Figure 45: Volumes standard d'ombre (gauche) vs des volumes d'ombre de CC (droits).

4.6.2. Clamping

La figure 44 (b) indique que même après l'étape de culling, les volumes d'ombre restant sont en grande partie inutile : seulement les petites parties des volumes d'ombre couvrent réellement toute la géométrie de la scène. Par le Clamping des volumes d'ombre aux régions qui contiennent des récepteurs d'ombre, nous pouvons augmenter la performance globale par la réduction du taux de remplissage.

Lloyd et al. emploient le clamping en utilisant deux techniques différentes. Le clamping continu fonctionne entièrement sur le CPU et essaye de clamper les volumes d'ombre à leurs limites en z dans l'espace de la lumière figure 44 (c).

Le clamping discret coupe le frustum de visionnement par plusieurs plans en sections et utilise le matériel graphique pour découvrir que lles sections sont occupées avec des récepteurs d'ombre. Comme le clamping continu, il limite alors les volumes d'ombre à ces sections figure 44 (d).

5. La Silhouette

La détection de silhouette est une étape principale pour le rendu de volume d'ombre, comme nous l'avons dit précédemment pour construire les volumes d'ombre il faut : trouver la silhouette des objets vus depuis la source, puis construire des quadrilatères infinis (extrusion de la silhouette) s'appuyant : sur la source et sur chaque arête de silhouette, puis compter les entrées/sorties en utilisant le stencil-buffer.

Les silhouettes sont utilisées dans de nombreuses applications tel que la conception architecturale et les atlas médicaux. Les courbes de silhouette d'un modèle polygonal sont utiles dans le rendu réaliste, dans des techniques interactives, et dans le rendu non-photoréaliste (NPR). Dans le rendu réaliste les silhouettes sont employées pour simplifier le calcul d'ombre [AB01].

5.1. Définition d'une silhouette

Soit E le vecteur qui représente la direction de vue, et N le vecteur normal à un point de la surface. Un point appartient à la silhouette si le produit scalaire entre E et N est égale a zéro, c'est-à-dire, l'angle entre E et N est de 90 degrés. Il est important de noter que l'ensemble de silhouette d'un objet dépend de la position de vue dans le cas du rendu non-photoréaliste (NPR) et dépend de la position de lumière dans le cas du rendu réaliste (calcul d'ombre).

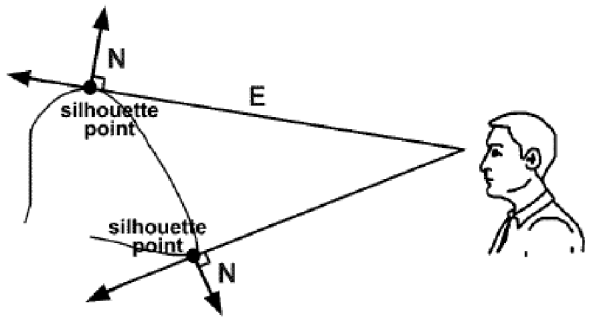


Figure 46: Silhouette avec des surfaces lisse [HHCG 03].

5.2. Silhouette pour le calcul d'ombre

Contrairement dans le rendu non-photoréaliste (NPR), la silhouette dépend de la position de la lumière, c'est-à-dire que la silhouette est vue par rapport à la source de lumière. La silhouette consiste à trouver le contour des objets vus depuis la source de lumière, en se basant sur les données géométriques de ceux-ci.

Soit E le vecteur qui représente la direction de la lumière, et N le vecteur normal à un point de la surface. Une arête silhouette est une arête où un triangle face avant et un triangle face arrière partagent la même arête. (Figure 47).

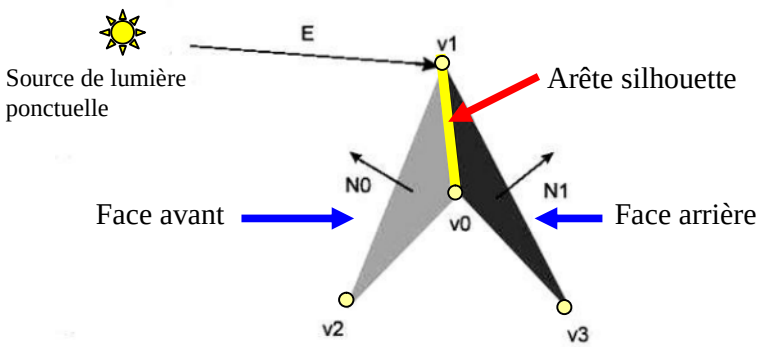


Figure 47 : Les arêtes silhouettes

Suivant les indications de la figure 47 : le triangle (v0,v1,v2) et le triangle (v0,v3,v1) partagent la même arête (v0,v1). Nous pouvons voir que le triangle (v0,v1,v2) est un triangle face avant et le triangle (v0,v3,v1) est un triangle face

arrière par rapport à la source de lumière, ainsi l'arête (v0,v1) est une arête de silhouette [Yaz06].

L'algorithme qui calcule si le triangle est une face avant ou pas est :

Etant donné un point de vue et un triangle (v0,v1,v2).

1. Calculer la normale du triangle (v0, v1, v2).

   N = cross (v1-v0, v2-v1) /* cross : Le produit vectoriel */

2. Calculer le vecteur de vue.

   E = v0 – eye /* eye : Point de vue */

3. If dot(E, N) < 0 /* dot : Le produit scalaire */

   Triangle (v0, v1, v2) est en face avant

   Else

   Triangle (v0, v1, v2) est en face arrière

5.2.1. Algorithme de silhouette à base de CPU

Il y a quelques conditions de base pour le format du modèle (objet dans la scène). Le modèle devrait au moins avoir une liste de sommet, une liste d'arêtes et la liste normale de triangle pour fournir les informations adjacentes [Yaz06]. Les silhouettes indiquent spécifiquement les silhouettes de l'espace d'objet pour les modèles polygonaux fermé. Un modèle polygonal est fermé si le modèle se compose uniquement de triangles et chaque bord a deux et seulement deux triangles adjacents.

L'algorithme traditionnel pour extraire la silhouette fonctionne par les deux étapes suivantes.

1. Pour chaque triangle, calculer si le triangle est un face avant ou pas.

2. Pour chaque arête, si un triangle adjacent est face avant et un autre est de face arrière, marquer cette arête comme une arête de silhouette.

Une première méthode pour déterminer la silhouette de l'objet pour l'extrusion du volume d'ombre, est donnée en utilisant l'algorithme suivant :

Pour chaque face :

Si elle est éclairée

Pour chaque arête :

On détermine si cette arête est partagée par une face voisine éclairée

Si ce n'est pas le cas :

Cette arête fait partie de la silhouette.

Fin Si

Fin Pour

Fin Si

Fin Pour

Les silhouettes ou l'information de silhouette est extrudé par le CPU avant le rendu. Dans le deuxième cas, le GPU est employé pour établir les structures de données nécessaires pour le rendu de silhouette .

5.2.2. Algorithme de silhouette à base de GPU

Plusieurs travaux de recherches ont été réalisés afin de détect er la silhouette par le GPU et de trouver une manière efficace de faire ceci en temps réel .

Quand on parte de GPU on parle d'u tilisation des shaders : l'extrusion des arêtes silhouettes peut être aussi effectué grâce au vertex shaders, sous OpenGL.

En 2006, Gunter Wallner implémenta l'étape d'extrusion de silhouette par le GPU dans un code shader qui est comme suit:

Code Vertex shader : Le code OpenGl passe comme paramètre la position de la source de lumière au code Vertex Shader.

→ Pour chaque vertex calculer la direction de la lumière : (light_direction = light_position - position.xyz)

→ Calculer le produit scalaire entre la normal e du sommet et la direction de la lumière et prendre leur max : NdotL = max(dot(gl_Normal, light_direction), 0.0);

→ On vérifie que la normale ne fait pas face à la lumière avant l'expulsion :

→ Si (NdotL <= 0.0)

→ {

/* On l'extrude */

→ position = position + (position - vec4(light_position, 1.0)) * EXTRUSION_FACTOR;

/* Placer le sommet expulsé à l'infini*/

→ position.w = 0.0;

→ }

→ Sinon Ne faire rien.

Code Fragment de shader :

→ Affectation de la couleur aux sommets, ainsi que la texture.

En 2009, P. Hermosilla & P.P. Vázquez [HPV09] suite à leurs études sur la détection de silhouette, avaient remarqué que la plupart des algorithmes exigent un pré-calcul, et/ou des multiples passages de rendu. Ils conclurent alors :

- L'information de la silhouette est extraite dans le CPU avant le rendu.
- Le GPU est employé pour établir les structures de données nécessaires pour le rendu de silhouette, parfois utilisé le multi-passage de rendu.
- Souvent utilise une certaine sorte de pré-calcul pour stocker l'information de contiguïté dans les sommets.

Nous pouvons conclure que dans les cas traditionnels il n'y a aucune géométrie qui est à base de silhouette et que le pré-calcul pour stocker l'information de contiguïté (nécessaire pour détecter la silhouette) est toujours présent.

6. Bilan

Dans cette partie, nous allons faire une synthèse des travaux présentés dans ce chapitre en mentionnant leurs catégories communes et les points forts de chacun ainsi que les nouveautés introduites.

6.1. Avantages des volumes d'ombre

L'utilisation des volumes d'ombre dans les scènes 3D, nous donne des ombres précises indépendamment des positions des sources et/ou caméra. Les volumes d'ombre s'ils sont bien programmés, génèrent des scènes robustes et même presque réalistes. L'algorithme est relativement générique, il fonctionne dans toutes les configurations avec les sources, les occulteurs et les receveurs d'ombre.

6.2. Problèmes des volumes d'ombre

Suite à notre étude sur les volumes d'ombre nous avons constaté que ces derniers exigent que l'objet qui projette d'ombre doit être fermé (chaque bord du maillage doit être partagé exactement par deux polygones donc limiter à des modèles fermés). Ils nécessitent aussi plusieurs passes de rendu pour générer les volumes d'ombre , ce qui conduit à la consommation d'un taux très élevé du Fill-Rate (le taux de remplissage) qui conduit à alourdir le système. Le calcul de silhouette peut charger le CPU pour des scènes dynamiques. La complexité de la géométrie des volumes d'ombre est supérieure à celle de la scène. Le problème principal est que l'information d'adjacence entre les primitives de base (calculés par le CPU) est aussi exigée. Le calcul de la silhouette sur CPU est très long.

6.3. Techniques d'optimisation

Les volumes d'ombre peuvent être améliorer en ce basant sur l'optimisation du fill-rate, le Culling, le clipping, et le Clamping (figure 48), ou par l'optimisation du rendu de volume d'ombre en utilisant le shaders [WAL08]. On peut aussi optimiser par la détection de la silhouette. Classiquement la détection de la silhouette exige un prétraitement qui consiste à rechercher l'information d'adjacence entre les primitives de base (triangle ou arêtes) de la géométrie des objets dans la scène.

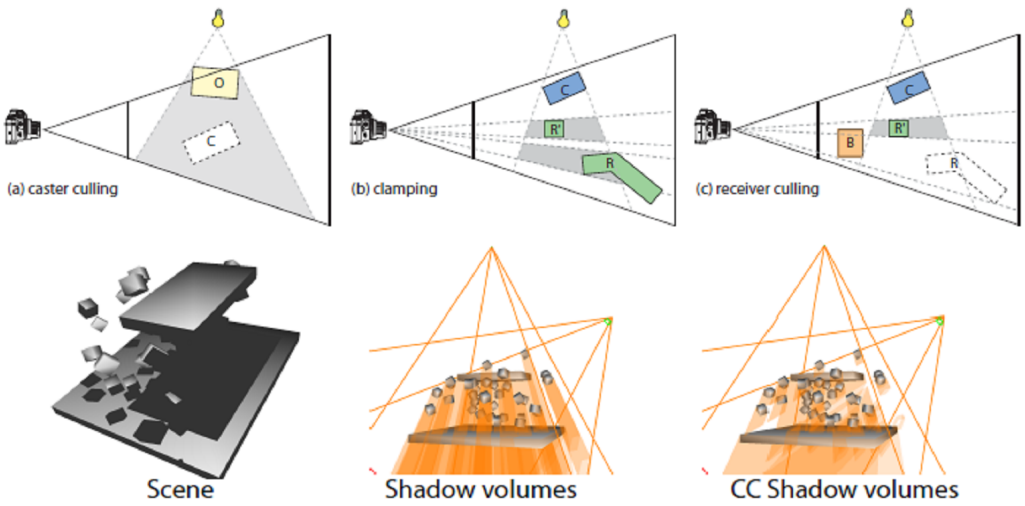


Figure 48: Les optimisations des volumes d'ombre [EASW09].

7. Conclusion

En conclusion, nous avons, à l'occasion du présent chapitre, passé en revue l'essentiel des travaux dédiés à la génération des volumes d'ombre. Le point commun entre tous ces travaux consiste en l'utilisation de la technique Z -Fail pour rendre les volumes d'ombre en temps réel et l'exigence d'un prétraitement des objets à rendre pour générer l'information d' adjacence entre les primitives de base des objets. Dans le prochain chapitre nous faisons une représentation détaillée sur notre contribution dans le domaine qui se basera sur les limites des précédents travaux, en proposant une nouvelle méthode pour optimiser le calcul des volumes d'ombre en temps réel par l'utilisation du matériel graphique récent.

Chapitre III :

Le modèle proposé

Le modèle proposé

Le problème fondamental du rendu des volumes d'ombre en temps réel est de tro uver de quelle manière on peut optimiser les calculs, en gardant la présence du réalisme et de l'esthétique.

Ce problème a longtemps été la matière principale de beaucoup d'études scientifiques. C'est également le cas en ce qui concerne notre mémoire. Cet dernièr est consacrée à l'optimisation des volumes d'ombre en utilisant les performances récentes du matériel graphique.

Le fort accroissement des besoins des applications graphiques, est, en partie, le résultat des développements très actifs des disposi tifs matériels d'acquisition, de traitements et de visualisations de s données. Durant ces dernières années, les possibilités offertes par les matériels graphiques ont été décuplées, aussi bien en termes de puissance brute de calculs que de flexibilité, ouv rant la voie à de nouvelles applications toujours plus sophistiquées, permettant de repousser les limites des techniques de visualisation.

Dans ce chapitre nous allons expliquer en détail la solution que nous proposons pour prendre en charge le problème du volume d'ombre.

1. Problématique générale et motivations

Les techniques de génération des volumes d'ombre en temps réel introduisent un certain nombre de difficultés quant à leur visualisation. Le problème principal réside dans l'interactivité car il faut calculer en temps réel le rendu de plusieurs images à

entrelacer. En général, ces calculs dépassent les capacités de traitement des cartes graphiques actuelles dans le cas de scènes complexes qui contiennent un nombre important de polygone. Les méthodes ainsi basées sur des maillages produisent des résultats assez satisfaisants, en termes de qualité et de performances, mais se montrent globalement complexes et très lourdes à mettre en place en ayant, par exemple, recours à de longues phases de pré-calculs implémenter sur le CPU, comme pour la génération d'information de connectivité entre les primitives de base qui constituent la maille de l'objet à rendre afin de générer le volume d'ombre. Pour résoudre ce problème, il est possible de faire appel aux récentes innovations dans le domaine du matériel graphique afin d'optimiser le calcul des volumes d'ombre.

Dans le cadre des travaux menés précédemment (voir chapitre II), nous avons constaté qu'il n'existe aucune maille qui est à base de silhouette . En s'appuyant sur la nouvelle génération des GPU, qui permettent, pour chaque pixel affiché à l'écran, de réaliser des opérations programmables, accroissant ainsi le réalisme en minimisant le coût, nous avons proposé de générer l'information d'adjacence en utilisant le Geometry Shader.

2. Contributions

Les travaux vus dans le chapitre précédent ont débouché sur un certain nombre de contributions qui sont résumées dans cette section. D'une façon globale, les méthodes de génération de volume d'ombre développées exigent des calculs supplémentaires qui alourdissent le processus de génération de l'image .

Classiquement avant de rendre les volumes d'ombre il faut premièrement trouver l'information de contiguïté de chaque objet constituant la scène qui projette de l'ombre, jusqu'à ce jour cette étape est implémenté sur le CPU.

Notre travail est basé sur la méthode de volume d'ombre de Gunter Wallner [WAL08] qui est basée sur la technique de Z-FAIL, et l'extrusion de la silhouette est implémenté par des programmes GPU (shader), mais dans ce travail l'étape de détection de l'information de contiguïté est implémenté sur le CPU, c'est-à-dire qu'elle exige un pré-calcul de la géométrie, car l'information d'adjacence entre les primitives est absente à cette étape du rendu.

Notre analyse faite sur les limites de la méthode de Gunter Wallner [WAL08] nous a permis de proposer une méthode plus rapide pour la détection de l'information de contiguïté entre les primitives de base qui constituent le modèle 3 D et l'extrusion de silhouette, par une amélioration à l'aide des nouvelles performances des GPUs. Plusieurs travaux de recherche pour trouver une manière efficace de détection de la silhouette en temps réel [JLCD02], [RAS01], [ASH04], ont été élaborés, bien que la plupart des algorithmes exigent un pré-calcul, et/ou les multi-passes de rendu.

Avec la nouvelle étape de le Geometry Shaders de GPUs, une nouvelle possibilité est ouverte, puisque l'information d'adjacence est disponible. Grâce aux Geometry Shaders nous pouvons produire de la géométrie de silhouette. Notre contribution consiste à implémenter l'étape de calcul de l'information d'adjacence entre les primitives de base nécessaires pour le calcul de silhouette sur le GPU et de détecter et d'extruder la silhouette en utilisant les Geometry Shaders.

3. L'environnement choisi.

Pour poursuivre notre simulation et pour accomplir notre but qui est la génération du volume d'ombre en temps réel assurant la robustesse et l'interactivité dans notre environnement tridimensionnelle, nous avons choisi un environnement 3D qui est un espace plat avec quelques objets. Il y'a lieu de signaler, à ce niveau, que bien que notre scène soit tridimensionnelle, notre objectif n'est pas de lui trouver l'esthétique mais plutôt d'accélérer la génération des ombres dans un environnement tridimensionnel.

4. Bibliothèques graphiques.

4.1. OpenGL.

OpenGL (Open Graphics Library) est une API (Application Programming Interface) multi plate-forme pour le développement d'applications gérant des images 2D et 3D. Elle a été développée en 1989 par Sillicon Graphics, puis portée sur d'autres

architectures en 1993. Depuis 1992, sa spécification est surveillée par l'OpenGL Architecture Review Board (ARB).

Elle été implémentée sur la plupart des plates-formes informatiques actuelles (Linux et Unix, Windows, MacOs, BeOS...) et est disponible pour de nombreux langages de programmation (C, C++, Java, Fortran, Ada...).

Son interface regroupe environ 250 fonctions différentes qui peuvent être utilisées pour afficher des scènes tridimensionnelles complexes à partir de simples primitives géométriques. Du fait de son ouverture, de sa souplesse d'utilisation et de sa disponibilité sur toutes les plates-formes, elle est utilisée par la majorité des applications scientifiques, industrielles ou artistiques 3D et certaines applications 2D vectorielles [BP07].

OpenGL est maintenu par un groupe indépendant : l'ARB (Architecture Review Board), composé des plus grands acteurs du marché (Compaq, Microsoft, SGI, IBM, 3DLabs, HP, Intel, Evans & Sutherland). C'est l'ARB qui contrôle donc les propositions d'évolution de l'API, les étudie et les approuve.

Il existe deux niveaux de portabilité lorsque l'on parle d'OpenGL :

- La portabilité au niveau de la plateforme : il existe des implémentations d'OpenGL pour Windows, Linux, Mac... Il est possible de créer un programme sur un système et ensuite faire tourner ce même programme sous un autre système.
- La portabilité au niveau du hardware : chaque vendeur de carte graphique pouvant ajouter ses propres extensions, il est tout à fait possible avec OpenGL de créer des programmes qui ne tournent que sur un type de carte graphique donné. De même, un programme tournant sur une générati on de carte a peu de chances de tourner sur une génération plus ancienne si jamais elle utilise des extensions apportées par la nouvelle génération. Pour éviter les problèmes de portabilité entre cartes graphiques, il est donc conseillé de n'utiliser que les extensions ARB ou EXT (bien que les extensions EXT ne garantissent pas d'être présentes sur toute les cartes, elle sont généralement implémentées par la plupart).

4.2. Extensions

La norme OpenGL permet à différents fabricants d'ajouter de nouvelles fonctionnalités sous forme d'extensions. Une extension est distribuée en 2 parties : un fichier d'en-têtes qui contient les fonctions prototypes de l'extension et les drivers du fabricant. Chacun d'eux possède une abréviation qui est utilisée pour nommer leurs nouvelles fonctions et constantes. Par exemple, l'abréviation de nVidia (« NV ») est utilisée pour définir leur fonction propriétaire « glCombinerParameterfvNV() » et leur constante « GL_NORMAL_MAP_NV ». Il peut arriver que plus d'un fabricant implémente la même fonctionnalité. Dans ce cas, l'abréviation « EXT » est utilisée. Il peut également arriver que l'ARB officialise une extension. Celle -ci devient alors un standard et l'abréviation « ARB » est utilisée. La première extension ARB fut GL_ARB_multitexture [BP07].

Dans notre travail, nous nous sommes intéresser aux extensions suivantes : GL_STENCIL_TEST_TWO_SIDE_EXT permet de ne rendre que les faces avants et arrières une seule fois dans le stencil buffer. GL_EXT_stencil_wrap évite la saturation du stencil buffer. L'extension GL_NV_depth_clamp qui, au lieu de supprimer les fragments coupés (Clipping) par les plans avant et arrière du frustum (pyramide du vue).

4.3. Shaders Model 4.

La révision 4 du modèle des shaders telle qu'elle est décrite dans Direct X 10 et implémentée sur le G80 est exposée dans OpenGL via les extensions EXT_gpu_shader4 (pour le GLSL[2]) et NV_gpu_program4 (pour l'assembleur NVidia). Elles fournissent ainsi un jeu d'instructions unifiées pour l'ensemble des shaders qui supportent maintenant tous totalement les branchements dynamiques par exemple ou l'accès aux textures sans restrictions [BP07].

Cette révision ajoute principalement la gestion totale des entiers pour les opérations arithmétiques et logiques, les attributs, ainsi que pour l'adressage dans les

[2] GLSL : Pour OpenGL Shading Langage est un langage permettant la programmation GPU de scènes OpenGL.

textures et l'accès aux valeurs stockées en entiers (gestion via l'extension EXT_texture_integer).

Un nombre plus important d'attributs peu vent être passés entre vertex shader et fragment shader. Le fragment shader a maintenant également la possibilité de choisir le mode d'interpolation des attributs qu'il reçoit.

5. Contexte matériel

Afin de développer des méthodes de rendu et de visualisation performantes, aussi bien en terme de qualité visuelle que de vitesse, il faut désormais exploiter au maximum le matériel graphique et, en particulier, exploiter au maximum les capacités de programmation offertes par les processeurs graphiques actuels. Pour cela, une brève description de ce contexte matériel est nécessaire.

5.1. Chaîne de traitements graphiques (pipeline graphique)

La chaîne de traitements graphiques est restée très longtemps composée de différentes étapes dont le comportement ne pouvait être modifié que très peu, par un simple ajustement de seulement quelques paramètres. Désormais, certaines étapes de cette chaîne de traitements sont devenues entièrement programmables au prix d'un certain nombre de contraintes liées au type même des processeur s graphiques (appelés également GPU pour Graphics Processing Unit). En effet, les processeurs graphiques sont des processeurs de type SIMD (Single Instruction, Multiple Data), appliquant, en parallèle, le même ensemble de traitements à un ensemble de données sans possibilité de dépendances entre les traitements des données [Amm10].

Concrètement dans le cas d'un GPU, cela se traduit par un ensemble de traitements identiques appliqués à tous les pixels d'une image, mais le traitement d'un pixel ne peut pas s'appuyer sur le résultat du pixel voisin.

Avec différents modèles de processeurs graphiques apparus vers 2007 (Processeurs graphiques de type NVDIA G80), la chaîne de traitements graphiques se décompose suivant le schéma de la figure 49. Les données, permettant de débuter les traitements définis par cette architecture, sont les sommets de la géométrie à

afficher. Ils sont, dans un premier temps, traités individuellement (au niveau du vertex shader) puis assemblés afin de former les différentes primitives géométriques (étapes d'assemblage de primitives et du Geometry Shader). Les parties de la géométrie ne rentrant pas dans la pyramide de vision sont éliminées avant que les parties valides soient rastérisées pour générer les pixels ou fragments (À ce niveau de la chaîne de traitements graphique, on parle plutôt de fragments. En effet, un pixel est un fragment qui fait partie de l'image finale : un fragment n'est pas obligatoirement un pixel, et peut être évincé de l'image finale.) qui composent l'image finale.

Afin de calculer leur couleur, un dernier traitement est appliqué sur chaque fragment (au niveau du fragment shader). La chaîne se termine par différents tests sur les fragments (profondeur, transparence, ...) avant leur écriture dans l'espace mémoire associée à la zone de dessin de l'écran (frame buffer) utilisée pour l'affichage.

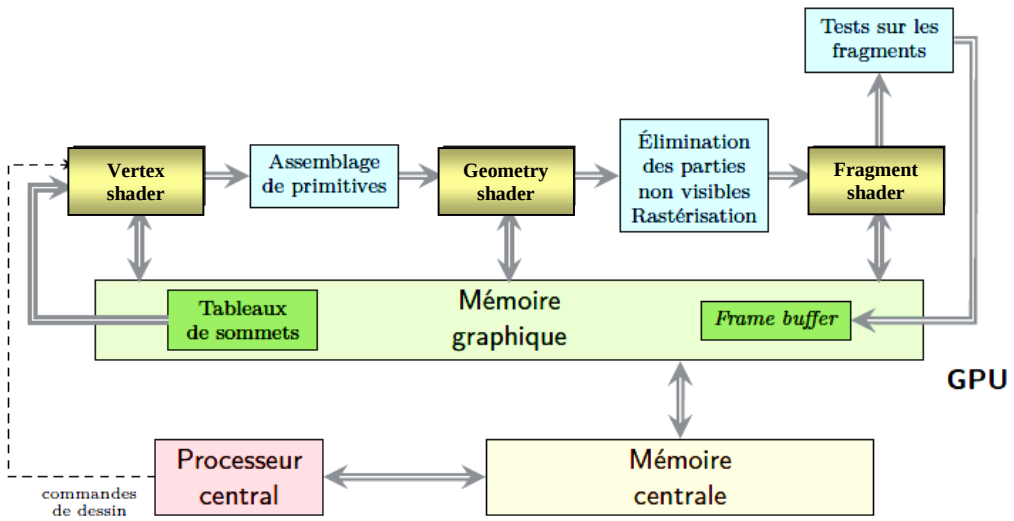


Figure 49: Chaîne de traitements graphiques. Le point d'entrée de la chaîne se situe au niveau du vertex shader et est commandée par le processeur central. En jaune, les étapes programmables de cette chaîne [Amm10].

On peut noter qu'il existe des interactions fortes entre le processeur central (ou CPU pour Central Processing Unit), la mémoire centrale, la mémoire graphique et le processeur graphique. En effet, le processeur central envoie les commandes de dessin

au GPU et gère les échanges mémoires, entre mémoire centrale et mémoire graphique. Ces derniers constituent d'ailleurs généralement un goulot d'étranglements qui peut pénaliser les performanc es des méthodes exploitant ce type d'architecture.

Chaque étape programmable de la chaîne de traitements graphiques est définie par un ensemble de programmes appelés shaders (ou programme de shaders). Parmi les différentes étapes programmables, on compte :

Vertex shader reçoit en entrée les sommets composant la géométrie à afficher. Chaque sommet peut être associé à différentes valeurs (couleurs, coordonnées de textures, normales à ces sommets, ...). Ce shader applique un traitement sur chacun des sommets reçus.

Geometry Shader situé après l'assemblage des sommets en différentes primitives (triangles, lignes, ...), il applique un traitement sur chacune des primitives reçues. Il est possible d'annuler le traitement de certaines primitives ou d'en créer de nouvelles.

Fragment shader situé après l'étape de rastérisation, il applique un traitement sur chacun des fragments (pixels) générés par la rastérisation des primitives issues du Geometry Shader.

À l'heure actuelle, de nouvelles architectures graphiques (Processeurs de type NVIDIA fermi) sont apparues et ajoutent une étape programmable supplémentaire à la chaîne de traitements graphiques. Cette nouvelle étape, appelée Tesselation Shader, permet d'effectuer plus simplement la subdivision des primitives sur le G PU. Ce nouveau shader est situé juste avant le Geometry Shader.

5.2. Choix des objets 3D

Nous avons vu dans le chapitre précédent que le volume d'ombre exige des modèles polygonaux fermés, donc le choix des objets 3D est une étape indispensable pour le calcul des volumes d'ombre. Comme l a détection de la silhouette exige un pré-calcul (sur le CPU) de la géométrie des objets dans la scène, il est alors nécessaire de construire les structures de données adéquates permettant de calculer et tracer rapidement les volumes d'ombre (utilisé un pré-calcule pour stocker l'information contiguë). Pour cela nous avons besoin d'une liste d'arêtes et d'informations

supplémentaires sur les faces. Toutes ces structures de données supplémentaires conduisent à alourdir le calcul. Donc les volumes d'ombre sont limités aux modèles polygonaux fermées. Le maillage des billboards, des systèmes de particule ou une maille texturée avec de l'alpha matte (telle qu'une feuille d'arbre) etc., ces dernières n'ont pas une maille qui représente exactement leur forme, ils produisent des ombres basées sur leurs mailles réelles, qui ne correspondent pas comment les objets apparaissent réellement. C'est pour cette raison que la majorité des programmeurs de volume d'ombre utilisent des mailles fermées (Modèles 3D sous format : MD2, MD5, OBJ, MDL, 3dsmax etc.), puisque avec ces derniers on peut représente exactement leur forme donc obtenir des ombres réalistes.

Ces modèles sont très utiles pour le rendu des scènes 3D, car pour faire rentrer l'ensemble des coordonnées des points à la main est fastidieux rien que pour dessiner un cube alors imaginer pour une maison ou un personnage. C'est pourquoi il existe des modeleurs 3D permettant de dessiner les modèles 3D puis de générer des fichiers contenant l'ensemble des données pré-formatés pour être utilisable par la carte graphique.

Les modeleurs 3D peuvent générer des fichiers contenant les informations 3D (primitives de bases) de la scène dessiné (position des sommets, indices, triangles, textures ….), mais ces fichiers ne contiennent pas l'information de contiguïté (adjacence) entre les primitives de bases [HPV09].

Nous dans notre travail nous avons choisi le modèle MD2 parce que c'est un modèle assez simple pour charger des modèles en 3D depuis un fichier.

Comme nous l'avons vu dans le chapitre précédent qu'il n'y a aucune géométrie qui est à base de silhouette [HPV09].

5.2.1. Les modèles MD2

Les modèles MD2 sont les modèles composés : de données géométriques, d'animations par frame et de « commandes OpenGL » [Hen04].

Les données géométriques sont les triangles et les sommets du modèle, ainsi que leurs coordonnées de texture.

Les animations du modèle sont des animations par frame. Ce ne sont pas des animations squelettiques. C'est pour ça aussi que ce format est simple. Chaque frame

contient toutes les données géométrique du modèle (triangles, etc.) dans une certaine position. Donc en réalité, un modèle MD2 est composé d'une multitude de modèles qui, affichés successivement et rapidement, lui donnent un effet d'animation.

Les commandes OpenGL sont des données structurées de sorte que l'on puisse dessiner le modèle uniquement à l'aide des primitives GL_TRIANGLE_STRIP et GL_TRIANGLE_FAN. Ces données ont donc été triées puis rangées de sorte à ce que le rendu du modèle se fasse rapidement et simplement. Mais ce modèle comme tous les autres modèles n'est pas à base de silhouette, donc l'information de contiguïté entre ces données n'est pas disponible.

6. Approche proposée

La méthode que nous retiendrons, pour l'intégration d'ombrage, est celle des volumes d'ombre. Cette méthode nous semble la plus robuste à l'heure actuelle. Bien que coûteuse, du point de vue temps de calcul CPU, les capacités des machines actuelles nous permette raisonnablement de l'envisager.

Après avoir défini les conventions que nous utiliserons, nous présenterons les grandes lignes de notre algorithme.

Afin de rendre des volumes d'ombre de la façon la plus rapide et la plus précise possible, nous nous sommes basé essentiellement sur l'algorithme et les optimisations proposées par Nvidia dans l'article " Fast, Practical and Robust Shadows " [EK03] et sur l'article " Geometry of Real Time Shadows" [WAL08] de Gunter Wallner.

6.1. Algorithme général

Nous avons vu dans le chapitre précédent, que pour construire les volumes d'ombre dans une scène contenant des objets 3D, il faut :

- Trouver la silhouette des objets vus depuis la source, puis construire des quadrilatères infinis (extrusion de la silhouette) en s'appuyant : sur la source et sur chaque arête de silhouette, puis compter les entrées/sorties en utilisant le Stencil-Buffer.
- Pour trouver la silhouette des objets il faut construire les structures de données adéquates permettant de calculer et tracer rapidement les volumes

d'ombre (utilisé un pré-calcul pour stocker l'information contiguë qui est nécessaire pour la détermination de silhouette). Pour cela nous avons besoin d'une liste d'arêtes et d'informations supplémentaires sur les faces. Cette tâche est implémentée dans le CPU.

Tous les algorithmes pour générer les volumes d'ombre reposent sur l'algorithme de base suivant :

Algorithme de base du volume d'ombre

DEBUT

- **Pour chaque** Objet de la scène **Faire**
  - construire un volume d'ombre
- **Pour chaque** fragment dessiné **Faire**
  - compter combien de fois on entre/sort d'un volume (compteur)
    - **Si** le compteur > 0 **alors** dans l'ombre
    - **Si** le compteur = 0 **alors** dans la lumière

FIN

Les étapes de l'algorithme sont détaillées comme suit :

a) construire les volumes d'ombre
   - trouver la silhouette des objets vus depuis la source
   - construire des quadrilatères infinis s'appuyant
     - sur la source
     - sur chaque arête de silhouette

b) compter les entrées/sorties en utilisant le Stencil Buffer

Les étapes de l'algorithme pour trouver la silhouette

- **Pour chaque** arête du modèle **faire**
  - Identifier les faces avant / arrière et leurs normales
  - Calculer les produits scalaires normales/vecteur vers la source
  - Marquer comme silhouette si de signe différent

Parmi les inconvénients de l'algorithme pour trouver la silhouette :

➲ Requiert les infos d'adjacence du maillage (jusqu'à ce jour implémenté sur le CPU).

➲ Calculer un sur-ensemble de la silhouette sur le CPU.

6.2. Approche proposée pour la détection et l'extrusion de silhouette

L'approche que nous allons essayer de proposer est bas ée sur les performances des GPUs.

Nous allons calculer la détection et l'extrusion de silhouette, en utilisant le Geometry Shader (GPU) sans avoir besoin à calculer l'information d'adjacence entre les primitives de base des mailles 3D, donc nous n'avons plus besoin de pré-calcul implémenté dans le CPU pour stocker l'information de contiguïté qui est nécessaire pour la détermination de silhouette. Les Geometry Shaders ont l'avantage d'accéder aux primitives de voisinage des mailles 3D, donc le calcul d'info rmation d'adjacence sera réalisé par des programmes shaders sur GPU.

6.2.1. Silhouette à base de Geometry Shader

Les programmes Shaders sont des programmes exécutés par le GPU. Ils servent à remplacer les fonctions figées (fonctions câblées) implémentées dans la carte graphique par d'autres fonctions écrites par le programmeur [JRL09]. Ils interviennent à différents niveaux du pipeline graphique (Figure 50). Les Shaders les plus utilisés sont le Vertex et Fragment Shaders. Dans les cartes graphiques modernes, l'étape Geometry Assembly du pipeline est devenue programmable ce qui donne aux programmeurs plus de flexibilité pour gérer les sommets et leurs connexités (ajout ou suppression de sommets ou d'arêtes).

6.2.1.1. Geometry Shader

Ces dernières années, les vertex et fragment Shaders ont bénéficié d'un f ort engouement de la part de la communauté graphique. En comparaison des CPU ou des anciennes versions des GPU, ils ont l'avantage de permettre d'accélérer les temps de rendu et d'améliorer la qualité du rendu. Le vertex shader ne reçoit pas les informations de connectivité (triangles ou quads), il ne reçoit que les coordonnées des points, indépendamment les uns des autres.

La quatrième version du Shader Model a introduit un nouveau genre de traitement nommé Geometry Shader [BP07], qui gèrent la connectivité des polygones ainsi que leurs subdivisions ou simplifications. Ils sont disponibles sur OpenGL, ce dernier prend place entre le vertex shader et la phase de clipping, comme illustré sur la figure 50. L'objectif de ce shader est de pouvoir manipuler, sur GPU, certaines primitives en offrant la possibilité de les transformer, dupliquer ou bien d'agir sur les sommets. Étant donné un ensemble d'éléments en entrée, le Geometry Shader permet d'obtenir des primitives comme des points, des lignes ou des triangles.

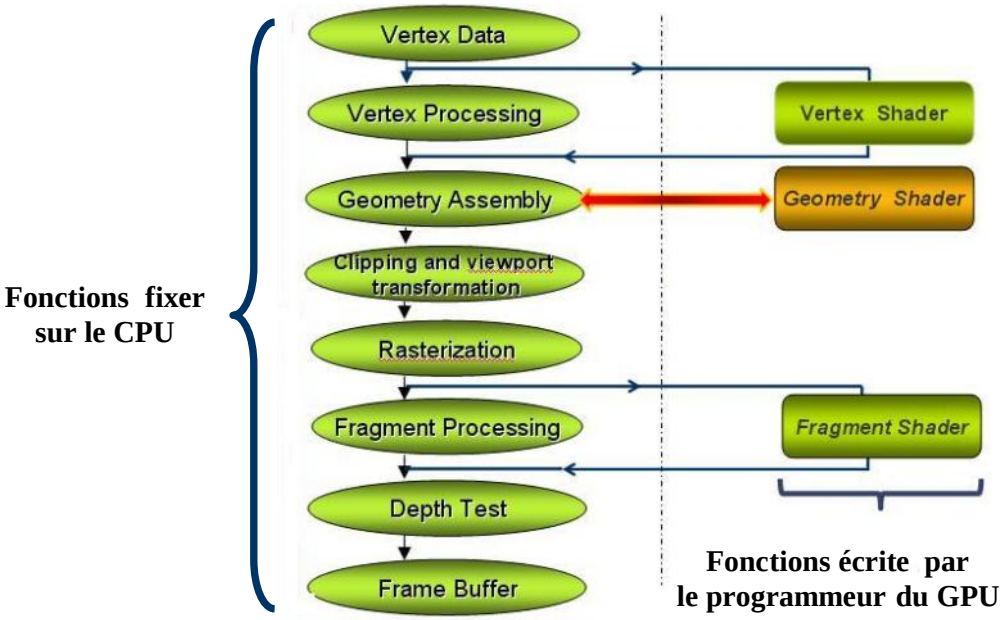


Figure 50 : Le pipeline OpenGL avec l'introduction de la version 4 du Shader model.

Avant son entrée dans la norme OpenGL 2.0, la programmation en GLSL était possible à l'aide des extensions GL_ARB_vertex_shaderet GL_ARB_fragment_shader.

La programmation des Geometry Shaders n'est possible qu'avec l'extension GL_EXT_geometry_shader. La figure 50 présente le pipeline graphique des cartes.

Le Vertex Processing : est une unité programmable disponible avec l'extension GL_ARB_vertex_shader pour des versions d'OpenGL inférieure à 2.0 et disponible en standard depuis la norme 2.0. Il traite un sommet entrant et ses attributs

(coordonnées homogènes, couleur, normale, coordonnées de texture, . . .). La sortie du Vertex Processing pour un sommet est normalement : les coordonnées du sommet dans le repère caméra, les attributs du sommet (couleur, normale, texture . . .)

Le Geometry Assembly : est une unité programmable disponible dans certaines cartes récentes avec l'extension GL_EXT_geo metry_shader4. Il traite une primitive entrante et ses données (i.e. plusieurs sommets avec leurs attributs). La sortie du Geometry Assembly consiste en une ou plusieurs primitives sortantes avec les attributs de sommets. Une primitive entrante et une prim itive sortante doivent être déclarées dans le programme OpenGL.

Les primitives entrantes autorisées sont:

GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP,
GL_LINES_ADJACENCY_EXT, GL_LINE_STRIP_ADJACENCY_EXT,
GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANG LE_FAN,
GL_TRIANGLES_ADJACENCY_EXT,
GLTRIANGLE_STRIP_ADJACENCY_EXT.

La primitive entrante d'un Geometry Assembly n'est pas obligatoirement celle utilisée pour l'envoi de la géométrie par OpenGL. Les primitives sortantes autorisées sont : GL_POINTS, GL_LIN ES_TRIP et GLTRIANGLE_STRIP.

Le Fragment Processing : est une unité programmable disponible avec l'extension GL_ARB_fragment_shader pour des versions d'OpenGL inférieure à 2.0 et disponible en standard depuis la norme 2.0. Il traite un fragment entrant et ses données (couleur, profondeur, . . .). L'entrée est le fragment dont on ne peut pas modifié la position (x, y). La sortie du Fragment Processing est le fragment (couleur, profondeur, . . .).

Le Geometry Shader a des n ouvelles primitive s de contiguïté qui sont maintenant disponibles comme suit :

- GL_LINES_ADJACENCY_EXT
- GL_LINE_STRIP_ADJACENCY_EXT
- GL_TRIANGLES_ADJACENCY_EXT
- GL TRIANGLE STRIP ADJECENCY EXT

Triangles avec la contiguïté (GL_TRIANGLES_ADJACENCY_EXT)

La figure 51 représente les six sommets définit comme suit :

Les points 0, 2, et 4 définissent le triangle courant.

Les points 1, 3, et 5 indiquent la position des triangles adjacents.

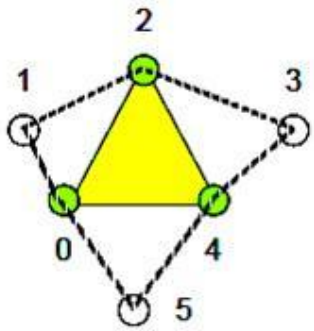


Figure 51 : Triangles avec la contiguïté (GL_TRIANGLES_ADJACENCY_EXT)

La silhouette d'une maille de triangle est l'ensemble des arêtes où un des triangles adjacents est en face avant tandis que l'autre est en face arrière.

Afin de détecter une silhouette dans une maille triangulée, nous devons traiter n'importe quelle triangle, ainsi que les triangles qui partagent une arête avec lui. Afin de pouvoir accéder à l'information de contiguïté, le Geometry Shader exige que les index de triangle soit comme illustrée dans figure 52

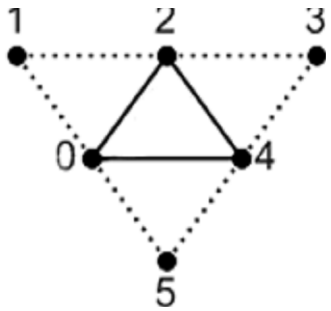


Figure 52: Le triangle actuellement (au centre) traité est celui déterminée par les sommets (0.2.4). Ceux déterminé par les sommets (0.1.2), (2.3.4) et (4.5.0) sont les triangles qui partagent une arête avec le triangle courant.

Cette nouvelle étape nous donne la possibilité de traiter une maille au niveau primitif avec l'information d'adjacence (triangles voisins), ce qui permettra de

produire une nouvelle géométrie en sortie (voir la figure. 50). C'est l'information que nous devons détecter pour créer les arêtes de la silhouette.

Nous considérons une maille fermée de triangle avec les triangles orientés régulièrement. L'ensemble de triangles est noté, $T_i \ldots T_N$., l'ensemble de sommets est $v_i \ldots v_n$ dans R^3, et les normales des triangles sont donnés par: n_t qui représente la normale d'un triangle $T_t = [v_i , v_j , v_k]$ [CMJ08]. Cette normale de triangle est définie comme normalisation du vecteur $(v_j - v_i) \times (v_k - v_i)$. Étant donné un observateur à la position $X \in R^3$, nous pouvons dire qu'un triangle est en face avant dans v si $(v - x) \cdot n \leq 0$, sinon il est en face arrière.

Cependant il existe uniquement deux fonctions disponibles sur les Geometry Shaders, EmitVertex() et EndPrimitive() dont le fonctionnement est comme suit : EmitVertex() est alors l'équivalent de glVertex () et EndPrimitive() l'équivalent de glEnd(). glBegin() qui est dans ce cas implicite, lorsque l'on rentre dans un Geometry shader, on peut considérer qu'un appel de ce type est lancé, et aussi après chaque appel de EndPrimitive() [RIC07].

6.2.1.2. Détection de silhouette et création de la géométrie

Nous pouvons détecter si une arête de triangle appartient à la silhouette en vérifiant si, pour les deux triangles partageant une arête, l'un est en face avant et l'autre en face arrière. Si nous appliquons cette méthode comme elle est, elle produira la même arêtes deux fois, une pour chaque triangle qui partage la même l'arête. Afin de corriger ceci, nous produisons uniquement une arête si le triangle qui est en face avant est au centre, celui qui est traité réellement par l e Geometry Shader.

Une fois qu'un bord de silhouette a été détecté, nous produisons la nouvelle géométrie pour la silhouette. Ceci est effectué en créant un nouveau polygone, en extrudant l'arête le long de la perpendiculaire de vecteur à la direction de l'arête. Cependant, cette approche produit des discontinuités dans le début et l'extrémité des arêtes. Ceux-ci deviennent particulièrement visibles avec l'augmentation de la largeur de la silhouette, suivant les indications de figure 53.

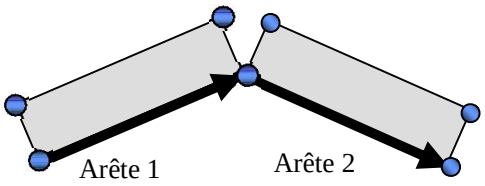


Figure 53: Quand en extrude l'arête des discontinuités apparaissent.

Afin de corriger ceci, nous adoptons l'approche de McGuire et Hughes [MJ04], mais nous faisons ceci à l'étape de Geometry Shader qui exige trois passages de rendu. Nous créons deux nouveaux triangles au début et à la fin d e l'arête, le long de la projection de la normale du sommet dans l'espace d'écran. Ceci assure la continuité le long de la silhouette, comme représenté sur la figure 56.

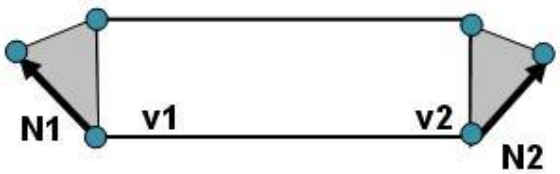


Figure 54: v1 et v2 sont les sommets de l'arête, et N1 et N2 sont les normales des sommets.

Dans certains cas, cette solution peut produire une erreur quand la direction d'extrusion a une direction différente que la normale projetée. Afin de résoudre ceci, nous avons conçu deux solutions possibles :

- Inverser la direction d'extrusion dans ce sommet.
- Inverser la direction de la normal e projetée.

Ces deux solutions sont illustrées sur les figures 55(a) et 55 (b). Pendant que ce problème se pose habituellement quand des arêtes sont cachés par un polygone de face avant, en renversant la direction d'expulsion les bords cachés sont indiqués à travers la géométrie visible, et donc un arte fact apparaît. Par conséquent, nous avons décidé d'employer la deuxième soluti on car elle est celle qui produit des résultats plus agréables.

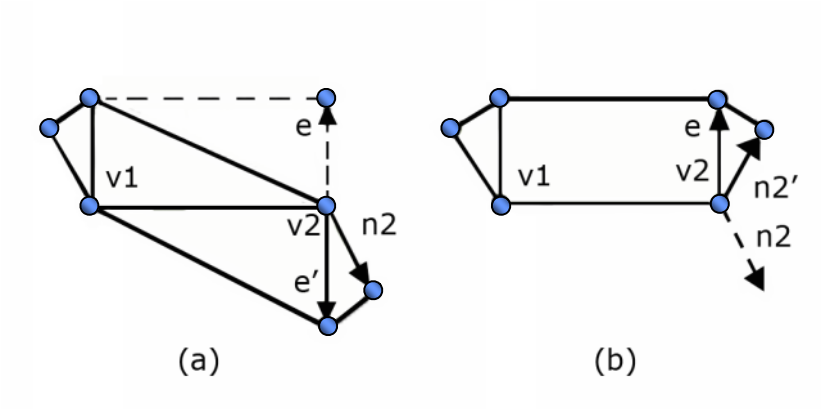


Figure 55 : v1 et v2 sont les sommets d'arête, e est la direction d'extrusion, n2 est la normale du sommet, e' est la direction d'extrusion inversé et le n2' est la normale projetée en l'inverse.

6.2.2. Proposition de calcul d'information d'adjacence et l'extrusion de la silhouette par les Geometry Shaders

Pour calculer les bords de silhouette d'une maille, le Geometry Shader doit avoir accès à l'information de contiguïté des triangles. Dans OpenGL, nous pouvons passer dans des sommets supplémentaires par triangle en utilisant le nouveau mode de GL_TRIANGLES_ADJACENCY_EXT pour le glBegin. En ce mode nous avons besoin de six sommets, trois pour définir le triangle en question, trois autres servant à spécifier les sommets voisins des bords. La figure 52 montre la disposition de sommet d'un triangle avec des voisins.

Le pseudo code du Geometry Shader qui extrude l'arête de silhouette est le suivant :

Calculer la normale du triangle central et le vecteur de position de vue

Si le triangle est en face avant **alors**

Pour tous les triangles adjacents **faire**

Calculer la normale

Si le triangle est en face arrière **alors**

Extruder la silhouette

Émettre les sommets

Fin si

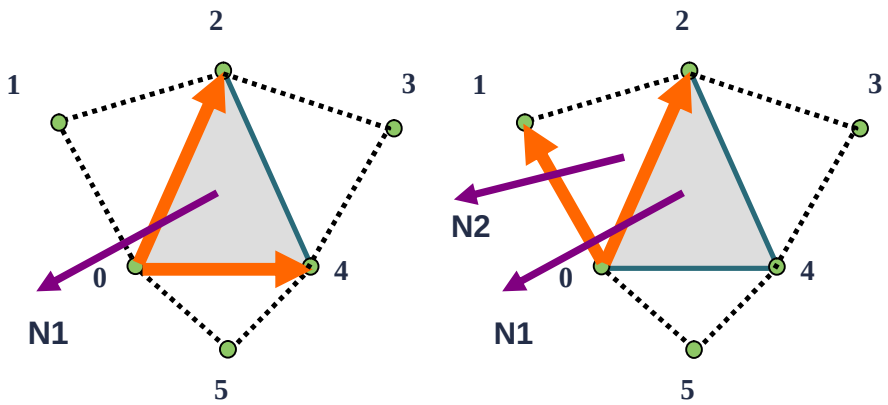
Fin pour

Fin si

La démarche de cet algorithme (figure 56):

Calculer le vecteur de position de vue eyeVec, Calculer la normale de triangle central (0.2.4) N1, si le triangle central (0.2.4) est en face avant, calculer si le triangle voisin (0.1.2) est en face arrière (calculer la normale N2 du triangle (0.1.2)), c'est-à-dire que Si (dot(eyeVec,N1) > 0 et dot(eyeVec,N2) < 0) cela signifie que l'arête [0,2] est une arête silhouette, sinon voir les deux autre voisin (2.3.4) et (4.5.0).

Si (dot(eyeVec,N1) > 0 et dot(eyeVec,N2) < 0) alors faire extruder l'arête silhouette [0,2] (comme illustré dans la figure 57).



(a) calcul de la normale N1 du triangle central (0.2.4)

(b) calcul de la normale N2 du triangle voisin (0.1.2)

Figure 56: Illustration de l'algorithme à base de Geometry Shader qui détecte l'arête de silhouette.

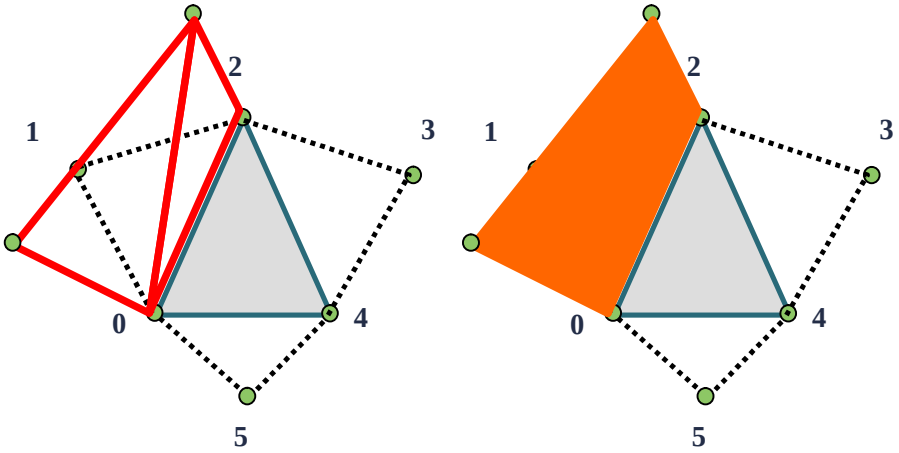


Figure 57: Illustration de l'algorithme à base de Geometry Shader qui extrude l'arête de silhouette.

Après avoir calculé la silhouette il faut extruder (projeter) les bords de silhouette à une distance importante. Dans notre application nous avons implémenté cette tâche grâce au Geometry Shader sur le GPU.

Nous avons implémenté l'étape de détection et d'extrusion de la silhouette dans un seul code shader et plus précisément dans le Geometry Shader.

6.3. Amélioration grâce à l'utilisation des extensions d'OpenGL.

Nous pouvons améliorer notre programme grâce à des extensions OpenGL qui permettent d'accélérer le rendu. GL_STENCIL_TEST_TWO_SIDE_EXT permet de ne rendre les faces avant et arrière une seule fois dans le stencil buffer. GL_EXT_stencil_wrap évite la saturation du stencil buffer. L'extension GL_NV_depth_clamp qui, au lieu de supprimer les fragments coupés (Clipping) par les plans avant et arrière du frustum (pyramide du vue). Mais d'autres extensions peuvent servir.

6.4. Amélioration par le Clipping.

Dans notre cas, il convient de mettre en place des plans de clipping afin de restreindre l'ombre au seul plan, et ne pas la dessiner dans le vide. De nos jours, beaucoup de cartes graphiques proposent l'utilisation d'un stencil, communément appeler Stencil Buffer.

Comme son nom l'indique, nous allons nous servir de ce stencil pour ne dessiner que l'ombre sur le plan.

Ce buffer est communément utilisé comme Stencil ou bien comme compteur d'événements. Les opérations sur le stencil sont des opérations arithmétiques simples telles que l'incrémentation, la décrémentation, le remplacement par un e valeur, ... Dans notre cas, il suffit de remplir le stencil à 1 là ou nous dessinons notre plan, et par la suite ne dessiner l'ombre que là où le stencil est à 1, on obtient ainsi la restriction au plan (figure 58).

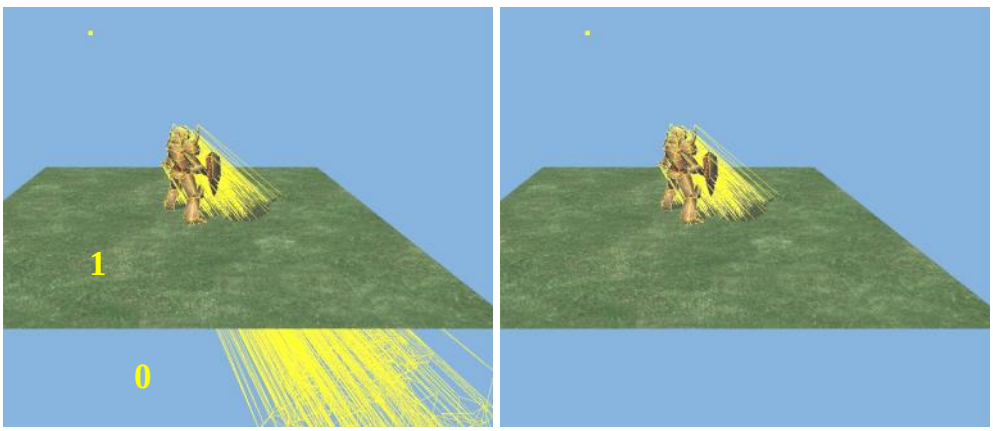


Figure 58: Principe d'utilisation du stencil, l'ombre n'apparaît que sur le plan

7. Conclusion.

Tout au long de ce chapitre, nous avons proposé une amélioration et une accélération de l'algorithme du volume d'ombre en temps réel. Ainsi, nous avons, dans un premier temps, choisi le Geometry Shader dont l'existence de l'information d'adjacence entre les primitives de bases et qui est simulée grâce aux Shaders. Ces

derniers permettent d'obtenir un niveau élevé du rendu, mais surtout le niveau d'accélération nécessaire au temps réel.

Pour conclure, il y'a lieu de noter que le système a été conçu dans le but d'aboutir à un rendu des volumes d'ombre en temps réel et ce dans un environnement complexe. Pour ce faire, nous avons défini un processus , nous permettant d'améliorer le calcul de volume d'ombre dans un environnement tridimensionnel contenant certains objets qui sont positionnés sur le sol. Ce processus est tout à fait réaliste est similaire au processus réel car il intègre toutes les caractéristiques des ombres comme elles sont dans la réalité.

Dans le chapitre suivant, nous allons discuter et donner les détails de l'implémentation de notre modèle ainsi que les résultats obtenus .

Chapitre IV :

Implémentation, bilan et résultats

Implémentation, bilan et résultats

Ce chapitre a pour but de valider expérimentalement, à travers un ensemble de simulations, le modèle proposé dans le chapitre précédent. Dans cette partie nous allons présenter un ensemble de résultats permettant de valider le modèle utilis er pour la génération des volumes d'ombre en temps réel sur une surface tridimensionnelle. Les résultats sont obtenues en utilisant un Intel(R) Pentium(R) D CPU 3.00GHz équipé d'une mémoire 1 Go DDR et une carte graphique GeForce 9400M GT avec une mémoire de 1 Go, version OpenGl Core: 3.3 et version GLSL: 3.30 .

Nous mettons, également, en exergue dans ce chapitre les résultats des expérimentations que nous avons réalisées. La première consistant à faire optimiser le rendu du volume d'ombre en utilisant le G eometry Shader [ZB+11a], [ZB+11c]. Quant à la seconde, elle s'intéresse à éliminé le temps de pré-calcule pour la détection de l'information de contiguïté entre les primitives de base des modèles 3D (triangles) en utilisant le Geometry Shader [ZB+11b], [ZB+11d].

1. Langage de programmation utilisé.

Techniquement, le volume d'ombre est réalisé par le biais d'une application implémentée dans le langage de programmation C++.

Nous avons choisi l'environnement de programmation Visual C++ pour, d'abord la qualité offerte par le C++, telle que la programmation orientée objet et pour l'interface qu'il fournit mais surtout pour sa compatibilité et ses capacités d'interfaçage avec le langage des shaders. Les shaders étant destiné, dans notre travail, à l'implémentation des volumes d'ombre.

De nombreux langages permettent la programmation des processeurs graphiques programmables. Il existe des langages de bas niveau comme l'assembleur défini dans les extensions GL_ARB_vertex_program et GL_ARB_fragment_program d'OpenGL [BP07].

Il existe également des langages de haut niveau spécifiques à des logiciels (RenderMan, Gelato), ou spécifiques à des API (DirectX High -Level Shading Language, Cg Shader Language, le langage GLSL).

HLSL : DirectX High-Level Shader Language, s'utilise avec Direct3D, la librairie de DirectX permettant de faire de la 3D temps réel (en pratique, on a parfois tendance à utiliser Direct3D et DirectX de façon interchangeable, car Direct3D est la partie la plus connue de DirectX). HLSL et DirectX ont été dével oppés par Microsoft. On les retrouve notamment dans les systèmes Windows, ou dans les Xbox.

Cg : Cg programming language, développé par NVIDIA. Langage très proche de HLSL car développé en parallèle ; Cg fonctionne par contre à la fois sous Direct3D et OpenGL. CgFx est une extension de Cg ; il permet notamment d'encapsuler plusieurs shaders dans un même fichier ; on obtient donc un "bloc" qui contient tout ce qui est nécessaire à l'effet 3D voulu. HLSL dispose aussi d'une extension similaire.

GLSL: OpenGL shading language, s'utilise avec OpenGL. Comme OpenGL, il est donc multiplateforme : fonctionne sous Linux, Mac OS, Windows [BP07].

Dans notre travail nous allons utiliser le langage GLSL car il permet la programmation GPU avec OpenGL.

1.1. Langage GLSL.

GLSL est un langage permettant la programmation GPU de scènes OpenGL. Il a été développé par 3D Lab et approuvé par l'ARB (Architecture Review Board), l'organisme chargé de la standardisation d'OpenGL.

La programmation GPU se pratique au moyen de trois types éléments : les vertex shader, la géométrie shader et le fragment shader. Les Shaders sont généralement écrit dans des fichiers textes en dehors du code mais il est tout a fait possible d'imaginer d'autres formes tant que les Shaders sont accessibles sous forme de chaînes de caractères [JJG05].

Un vertex shader réalise des opérations d'un vertex alors qu'un fragment shader réalise des opérations sur un fragment. Il est possible d'envoyer des informations du programme C/C++ vers le programme GLSL mais par d ans le sens inverse, mise à part le résultat final rendu à l'écran.

Précisons le vocabulaire au niveau de deux termes très utilisés : « Shader » et « Programme ». Le terme « shader » indique du code GLSL, alors que le terme « programme » indique soit des programmes GPU de bas niveau soit du code GLSL compilé et lié, ces deux derniers éléments n'étant pas vraiment différent [JJG05].
Le principe pour travailler avec un shader GLSL :

1. On crée un programme source en GLSL
2. On fait compiler ce programme source par OpenGL
3. On transfère le shader sur la carte graphique : liaisons entre le shader et le programme OpenGL
4. On passe des paramètres au shader : des images de texture, des valeurs numériques...
5. On active le shader pour dessiner certains polygones.

1.2. Stencil-Buffer.

Stencil-Buffer est un tampon particulier dans OpenGL permettant de n'afficher que certaines portions de l'écran déterminées par l'utilisateur, il est utilisé dans toute sorte d'effets spéciaux dont les ombres volumiques [Nev08].

1.2.1. Test du Stencil

Le test du stencil consiste à comparer une valeur de référence à celle contenue dans chaque pixel du tampon [Nev08], à l'aide de la fonction glStencilFunc(GLenum func, GLint ref, GLuint mask) où :

- Ref : est la valeur de référence
- Mask : représente un masque de bits qui sera appliqué à la valeur de référence et à celle du tampon.
- Test du stencil: (ref & mask) func (stencil & mask).
- La variable func peut prendre ces différents états :
  - GL_NEVER : Le test est toujours faux;
  - GL_ALWAYS : le test est toujours vrai;
  - GL_LESS : le test est vrai si la valeur ref est strictement inférieure à celle du tampon;

- GL_GREATER : le test est vrai si la valeur ref est strictement supérieure à celle du tampon;
- GL_LEQUAL : le test est vrai si la valeur ref est inférieure ou égal à celle du tampon;
- GL_GEQUAL : le test est vrai si la valeur ref est supérieure ou égale à celle du tampon;
- GL_EQUAL : le test est vrai si la valeur ref est égale à celle du tampon;
- GL_NOTEQUAL : le test est vrai si la valeur ref est différente de celle du tampon.

1.2.2. Modification du Stencil

En fonction du résultat le tampon est modifié et le passage des fragments est affecté [Nev08]. La fonction void glStencilOp(GLenum fail, GLenum zfail, GLenum pass) permet de déterminer la manière dont le stencil buffer sera modifié en fonction du résultat du test, ceci s'effectue comme suit :

- Le paramètre fail est utilisé si le test échoue.
- Le paramètre zfail est utilisé si le test stencil et le test de profondeur échouent.
- Sinon c'est le paramètre pass qui est utilisé.

Les paramètres fail, zfail et pass peuvent prendre les valeurs suivantes :

- GL_KEEP : La valeur dans le tampon est conservée;
- GL_ZERO : la valeur dans le tampon devient nulle;
- GL_REPLACE : La valeur du tampon est remplacée par la valeur de référence;
- GL_INCR : la valeur du tampon est incrémentée de un;
- GL_DECR : la valeur du tampon est décrémentée de un;
- GL_INVERT : les bits de la valeur du tampon sont inversés.

2. Algorithmes utilisés.

L'application réalisée, dans le cadre de notre mémoire, est basée sur deux concepts très importants. Le premier étant le rendu des volumes d'ombre en temps réel basé sur les travaux de Gunter Wallner [WAL08] (version CPU) alors que le second aspect

est celui de l'optimisation de cette technique en utilisant le Geometr y Shader (version GPU).

L'algorithme du volume d'ombre exige 2 passage s de rendu : un premier rendu de la scène sans ombre et avec uniquement la lumière ambiante et un deuxièm e passage de la scène où elle est rendu à nouveau avec le test de Stencil-Buffer permettant de mettre à jour uniquement les pixels étiquetés par le volume d'ombre.

Les étapes de l'algorithme pour la génération du volume d'ombre sont :

1. Initialisation de l'environnement OpenGL et les shaders.
2. Chargement de toutes les données de la scène (Modèle MD2, texture, shader..).
3. Cette étape est réalisée par l'un e des deux méthodes suivantes:
   - Les méthodes de Gunter Wallner [WAL08]: construction de la structure d'arêtes : construire les structures de données adéquates permettant de calculer et tracer rapidement les volumes d'ombre (utilisé un pré-calcul pour stocker l'information de contiguïté (version CPU)).
   - Notre proposition : utilisation du Geometry Shader (version GPU) pour détecter l'information de contiguïté des primitives de base du modèle 3D, et détecter puis extruder la silhouette par le GPU.
4. Le rendu de la scène :

Le rendu de la scène s'effectue en suivant les étapes suivantes :

☞ **1ère étape :**

On affiche notre scène dans son intégralité mais sans ombre. Cet affichage met à jour le tampon de profondeur qui sera utilisé dans l'étape 3.

☞ **2ème étape :**

On désactive la mise à jour du tampon de profondeur et du tampon de couleur. On active le tampon de stencil avec une fonction de comparaison qui laisse passer tous les fragment s.

☞ **3ème étape :**

On affiche le volume d'ombre créé par les objets éclairés en deux passes en utilisant la technique de Z-Fail :

Lors de la 1ère passe, on affiche les faces arrière du volume d'ombre, en incrémentant le stencil buffer lorsque le test de sten cil échoue.

Lors de la 2éme passe, on affiche les faces avant du volume d'ombre en décrémentant le stencil buffer lorsque le test de stencil échoue.

Une fois ce rendu effectué, seules les zones d'ombres de la scène ont une valeur différente de la valeur de référence dans le tampon de stencil.

☞ **4ème étape :**

Comme dans la méthode classique, on réactive le tampon de couleur. On affiche un rectangle semi-transparent qui couvre la totalité de l'écran par-dessus notre scène. Ce rectangle est de la couleur de l'ombre. On affiche tous les pixels du rectangle où la valeur de stencil est différente de la valeur de référence c'est-à-dire les parties non ombrés.

Le code 1 représente le programme principal de notre application comme suit :

Code 1: Pseudo code du programme principal pour créer les volumes d'ombre

```
Void codePrincipal()
{
                    /* charger toutes les objets 3D et les textures */
 Load(const char* szFileName);
        /* Premier rendu de la scène sans ombre et que la lumière ambiante */
 PremRendu() ;
         /* Rendu des volumes d'ombre et appel aux programmes shaders */
 DeuxRendu();
 DessinOmbre() ;
}
```

2.1. Chargement des objets 3D

La fonction Load (const char* szFileName) s'occupe du chargement des objets 3D et des textures, elle s'effectue au démarrage de l'application. Nous commençons par charger le plan puis les objets 3D et leurs textures à partir des fichiers sous format MD2, et initialiser les variables.

Nous avons réutilisé le code pour charger les modèles MD2 de David HENRY téléchargeable à partir du lien :
http://tfcduke.developpez.com/tutoriel/format/md2_specs/fichiers/md2.c , afin d'afficher nos modèles et de les intégrer dans la scène 3D.

Code 2: Pseudo code pour le chargement des objets 3D et allocation de mémoire

```
void Load(const char* szFileName)
{
initialisé le nombres des variables (nFrames,    nVertices, nTexCoords,nTriangles ,
nOpenGLCmds);
Allocation de memoire;
return;
}
```

Figure 59: Chargement des objets 3D et du plan.

2.2. Premier rendu de la scène sans ombre et utilisant uniquement la lumière ambiante.

Le premier rendu consiste à rendre la scène dans son intégrité avec uniquement la lumière ambiante et sans ombres, et à initialiser la valeur du Stencil-Buffer à zéro.

Code 3: Pseudo code du rendu de la scène sans ombre et utilisant uniquement la lumière ambiante

```
void PremRendu()
{
glEnable(GL_LIGHTING);
                        /*source de lumière activé */
glEnable(GL_LIGHT0);
                        /* débuter le depth testing */
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);
```

```
                    /* Ne pas écrire dans Z-buffer */
glDepthMask(1);
                /* pas de stencil testing (dans ce passage) */
glDisable(GL_STENCIL_TEST);
                    /*mise a jour du color buffer*/
glColorMask(1,1,1,1);
                /* initialisé la valeur du stencil à zéro */
glClearStencil(0);
                    /* Vider le Tampon Stencil */
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |
GL_STENCIL_BUFFER_BIT);
                    /* Ne calculer que l'ambiant */
renderScene();
    }
```

Figure 60: Rendu de la scène avec la lumière ambiante et sans ombres.

2.3. Rendu des volumes d'ombre.

Concernant l'implémentation du rendu des volumes d'ombre, nous utilisons la technique Z-Fail pour décréter les parties ombrées, et nous nous s'intéressons à l'initialisation du Stencil-Buffer.

Nous désactivons la mise à jour du tampon de profondeur et du tampon de couleur. Nous activons le tampon de stencil avec une fonction de comparaison qui laisse passer tous les fragments.

On affiche le volume d'ombre créé par les objets éclairés, puis nous affichons les faces avants et arrières une seule fois (un seul passage de rendu grâce à l'extension GL_STENCIL_TEST_TWO_SIDE_EXT au lieu de deux passages de rendu) dans le

Stencil-Buffer. Lors du 1er passage, on affiche les faces arrières du volume d'ombre en incrémentant le stencil buffer lorsque le test de stencil échoue.

Lors du la 2éme passage, on affiche les faces avants du volume d'ombre en décrémentant le stencil buffer lorsque le test de stencil échoue.

Une fois ce rendu effectué, seul les zones d'ombres de la scène ont une valeur différente de la valeur de référence dans le tampon de stencil.

Code 4 : Pseudo code du rendu des volumes d'ombre

```
void DeuxRendu()
{
                /* --- Utiliser le Stencil-Buffer méthode ZFAIL --- */
                        /* Ne pas écrire ds Z-buffer*/
glDepthMask(GL_FALSE);
glDepthFunc(GL_LESS);
                        /* Activer le test du Stencil*/
glEnable(GL_STENCIL_TEST);
    /* rendre les faces avants et arrières une seule fois dans le stencil buffer */
glEnable(GL_STENCIL_TEST_TWO_SIDE_EXT);
                      /* Ne pas modifier framebuffer*/
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
                        /* quadrilatère face arrière */
glActiveStencilFaceEXT(GL_BACK);
glStencilOp(GL_KEEP, GL_DECR_WRAP_EXT, GL_KEEP);
                        /* quadrilatère Face avant */
glActiveStencilFaceEXT(GL_FRONT);
glStencilOp(GL_KEEP, GL_INCR_WRAP_EXT, GL_KEEP);
glStencilFunc(GL_ALWAYS, 0, 0xFFFFFFFF);
glStencilFunc(GL_ALWAYS, 0, 0xFFFFFFFF);
        /* Activer le Vertex Shader, Fragment shader et Geometry Shader */
            /* light_position : position courante de la source de lumière */
ActiverShader("light_position");
renderScene();
DéactiverShader();
}
```

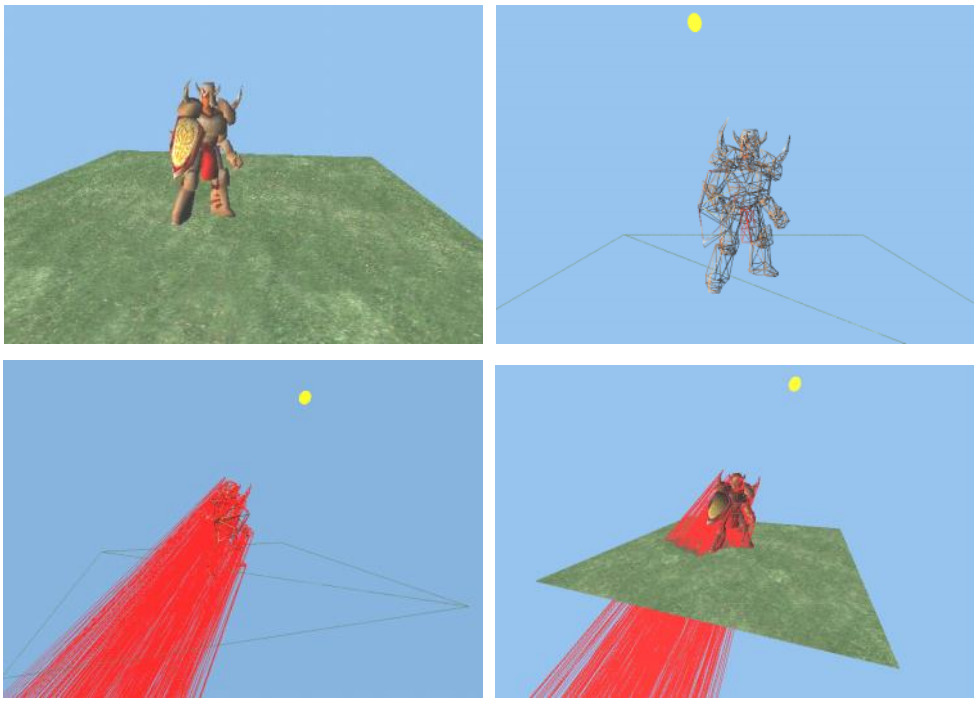


Figure 61: Rendu du volume d'ombre, et extrusion de silhouette en utilisant le Geometry Shader.

2.4. Deuxième rendu de la scène.

Comme dans la méthode classique, on réactive le tampon de couleur. On affiche un rectangle semi-transparent qui couvre la totalité de l'écran par-dessus notre scène. Ce rectangle est de la couleur de l'ombre. On affiche tous les pixels du rectangle où la valeur de stencil est différente de la valeur de référence c'est-à-dire les parties non ombrés.

Code 5 : Pseudo code du deuxième rendu des volumes d'ombre

```
void DessinOmbre()
{
glDepthFunc(GL_LEQUAL);
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glActiveStencilFaceEXT(GL_FRONT);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
glStencilFunc(GL_NOTEQUAL, 0, 0xFFFFFFFF);
glActiveStencilFaceEXT(GL_BACK);
```

```
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
glStencilFunc(GL_NOTEQUAL, 0, 0xFFFFFFFF);

    /*----- Mélanger (Blending) le volume d'ombre dans la scène en rendant un
  rectangle couvrant l'écran entier. Affichage du gris aux zones d'ombre -----*/

DessinerRectangle(); /* Figure 62 à gauche */
glDepthMask(GL_TRUE);
                        /* Désactiver le test du Stencil */
glDisable(GL_STENCIL_TEST);
glDisable(GL_STENCIL_TEST_TWO_SIDE_EXT);
}
```

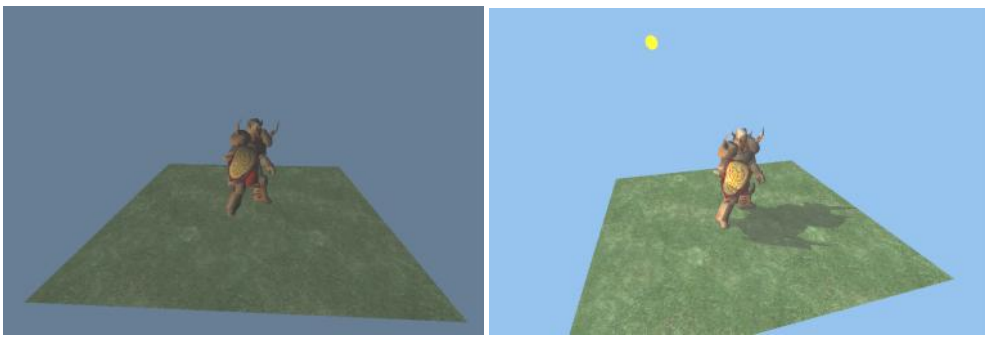


Figure 62: Affichage d'un rectangle semi-transparent qui couvre la totalité de l'écran (image de gauche), le rendu final du volume d'ombre (image de droite).

2.5. Activation des shaders.

L'activation des shaders consiste à les utiliser afin de détecter l'information d'adjacence entre les triangles formant les mailles des objets 3D, et pour la détection et l'extrusion de silhouette.

Code 5: Pseudo code pour l'activation des shaders

```
void ActiverShader (const char* light_position)
{
Triangle_principal( triangleadj VS_OUTPUT input[6]);
Triangle_voisin ( VS_OUTPUT v1, VS_OUTPUT v2, VS_OUTP UT vAdj);
}
/*----------------- Code Geometry Shader : le triangle au centre -----------------*/
```

```
void Triangle_principal( triangleadj VS_OUTPUT input[6])
{
                    /*  Calculer la normale du triangle  */
float3 N1 = cross( input[0].Position - input[2].Position, input[4].Position –
input[2].Position );
                    /*  Calculer le vecteur de vue  */
float3 eyeVec =  Eye – input[0].Position;
        /* si le triangle central est face avant, vérifier les autres triangles  */
If ( dot(N1, eyeVec)  > 0.0f )
{
Triangle_voisin(input[2],input[0],input[1]);
Triangle_voisin(input[4],input[2],input[3]);
Triangle_voisin(input[0],input[4],input[5]);
}
}
 /*  ------------------------------  les triangles voisins  ------------------------------- */

void Triangle_voisin ( VS_OUTPUT v1, VS_OUTPUT v2, VS_OUTPUT vAdj)
{
float3 N2 = cross( v1.Position - v2.Position, vAdj.Position - v2.Position );
float3 eyeVec = Eye - v1.Position ;

          /* si c'est une arête de silhouette, l'extruder dans 2 triangles */
if( dot(eyeVec,N2) < 0 )
{
GS_OUTPUT_FINS Out;
for(int v=0; v<2; v++)
{
Out.Position = mul(v1.Position + v*float4(v1.Normal,0)*length, WorldViewProj );
Out.Normal = mul( v1.Normal, World );
Out.TextureMesh = v1.Texture;
Out.TextureModel = float2(0,1-v);
Out.Opacity = opacity;

                    /* ajouter  les nouveaux sommets au  Stream */
TriStream.Append(Out);
TriStream.RestartStrip();
}
}
}
```

3. Résultats.

Dans cette partie nous allons présenter un ensemble de résultats permettant de valider le modèle utilisé pour le volume d'ombre dans une scène tridimensionnelle. Nous allons dans ce qui suit valider et tester notre application pour des cas particuliers où le volume d'ombre est généré de différentes manières.

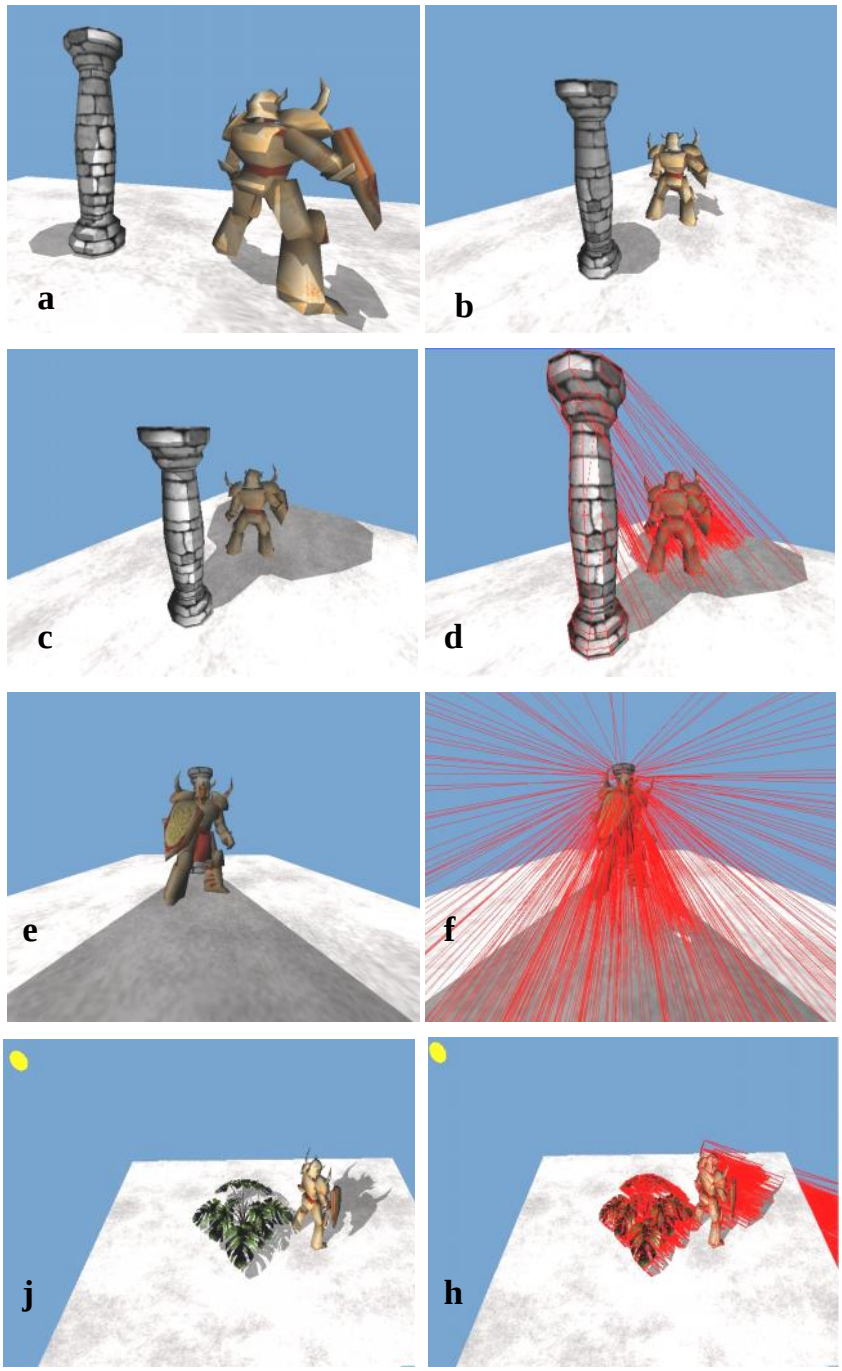


Figure 63: Six vues d'une scène rendue en utilisant l'algorithme du volume d'ombre (approche proposée).

1. Sur la figure 63 (a) nous pouvons voir clairement le volume d'ombre ainsi que l'auto-ombrage, c'est-à-dire que l'ombre crée par une partie de l'objet sur une autre partie du même objet.

2. La figure 63 (c et d) illustre l'influence des ombres des objets entre eux, c'est -à-dire que l'ombre crée par un objet sur un autre objet (la créature est dans l'ombre du pilier).

3. La figure 63 (e et f) présente le cas où la caméra est dans le volume d'ombre, dans cette situation il n'y pas de problème de caméra à l'intérieur du volume d'ombre.

4. La figure 63 (j et h) présente une scène représentée par une plante avec plusieurs feuilles, nous pouvons observer la génération du volume d'ombre sans aucune anomalie.

Dans ce qui suit, nous allons comparer notre méthode avec les techniques existantes de Gunter Wallner [WAL08] pour la génération du volume d'ombre en temps réel (telle que : Technique ZFail, extrusion et détection de silhouette par les shaders (vertex et fragment), et un pré-calcul pour calculer l'information d'adjacence entre les primitives de bases qui constituent le modèle), afin de valider les résultats que nous avons obtenus, les résultats sont récapitulés dans les graphiques de la figure (65, 66) et les images correspondantes listées dans les figures (64,67, 68, 69).

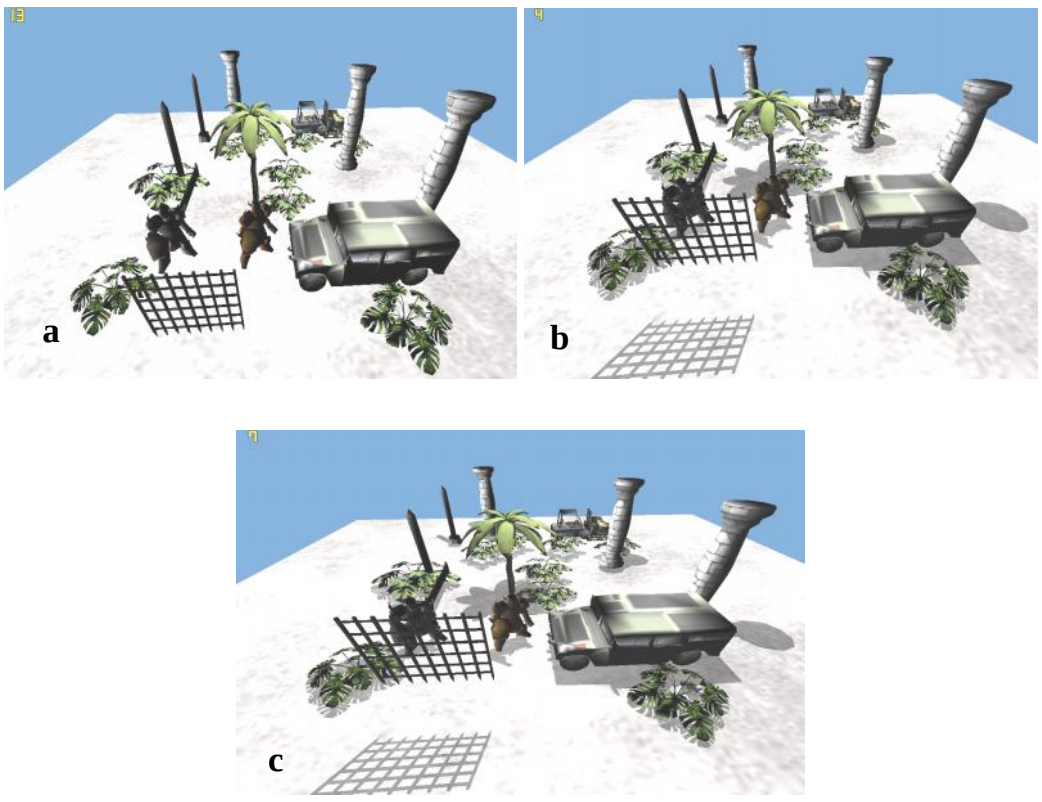


Figure 64: trois vues d'une scène rendue en utilisant l'algorithme du volume d'ombre avec une scène de 13200 triangles.

La figure 64 (a) représente une scène générée avec des ombres désactivés. Quant on active les ombres, la même scène est générée avec une fréquence d'affichage de 04 FPS par la méthode de Gunter Wallner (figure 64 (b)) et 7 FPS, en appliquant notre approche (figure 64 (c)).

3.1 Analyse des résultats.

Afin d'élaborer notre méthode, nous nous l'a soumis à un jeu de tests pour la mettre à l'échelle, ceci en variant le nombres d'objets à base de triangles de 365 à 200000 triangles tout en mesurant la fréquence d'affichage des images par seconde : FPS.

Les résultats obtenus, à l'issue des premières générations du volume d'ombre, étaient globalement acceptables et interactifs, mais si on augmente la complexité de la scène, nous aboutissons à des résultats non interactifs.

3.2 Discussion des résultats.

Dans cette section nous allons montrer quelques résultats et comparaisons des différentes méthodes implémentées. Nous terminerons cette section par un ensemble de recommandations pour le choix de la bonne méthode de rendu selon nos besoins en termes de performance.

Pour comparer les méthodes, nous avons choisi le critère FPS (Frame Per Second) pour évaluer les performances. Notre scène test est la scène 3D de la figure 67 contenant 80 objets de 200000 triangules.

Le graphique de la figure 65 montre les valeurs des FPS en fonction de nombre de triangles dans la scène. Cette même scène est rendu en utilisant trois méthodes différentes, la première en désactivant les volumes d'ombre (figure 67), la deuxième en activant les volumes d'ombre et en utilisant la méthode de Gunter Wallner [WAL08] (figure 68), et la troisième en activant les volumes d'ombre et en utilisant l'algorithme que nous avons proposé (figure 69). Les évaluations sont représentées dans le graphe de la figure 65.

Les travaux présentés dans ce chapitre montrent que le calcul des volumes d'ombre peut être effectué en tout point d'une image sans sacrifier la vitesse du rendu. L'efficacité de notre approche repose d'une part sur la vitesse du GPU. Contrairement

aux autres méthodes, où le calcul du volume d'ombre exige un pré-calcul de l'information de contiguïté (CPU).

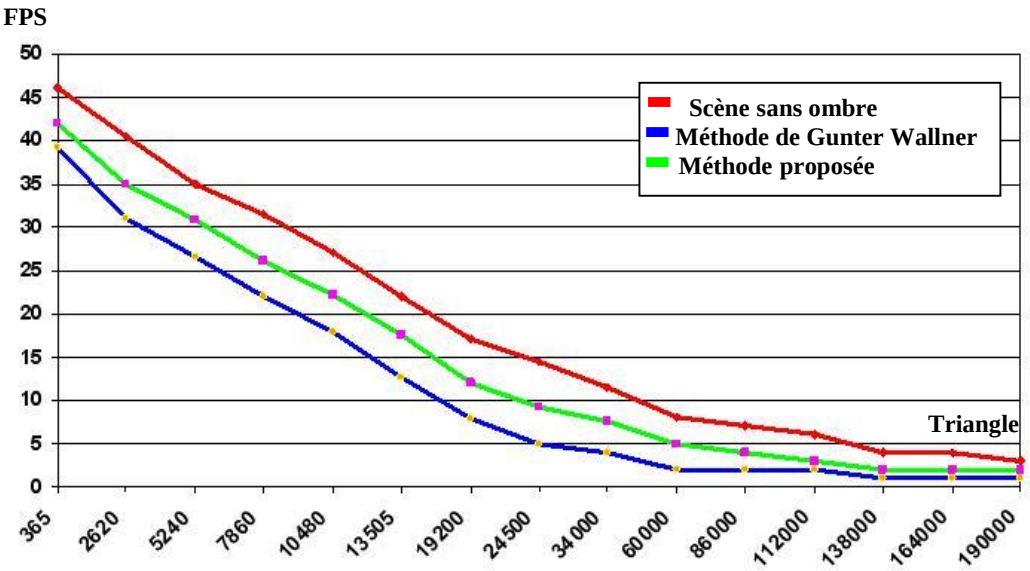


Figure 65 : Graphique des valeurs des FPS par rapport au nombre de triangles dans une scène, avec trois types de rendu différents (scène rendue sans ombres, scène rendue avec la méthode de Gunter Wallner, scène rendue avec la méthode proposée).

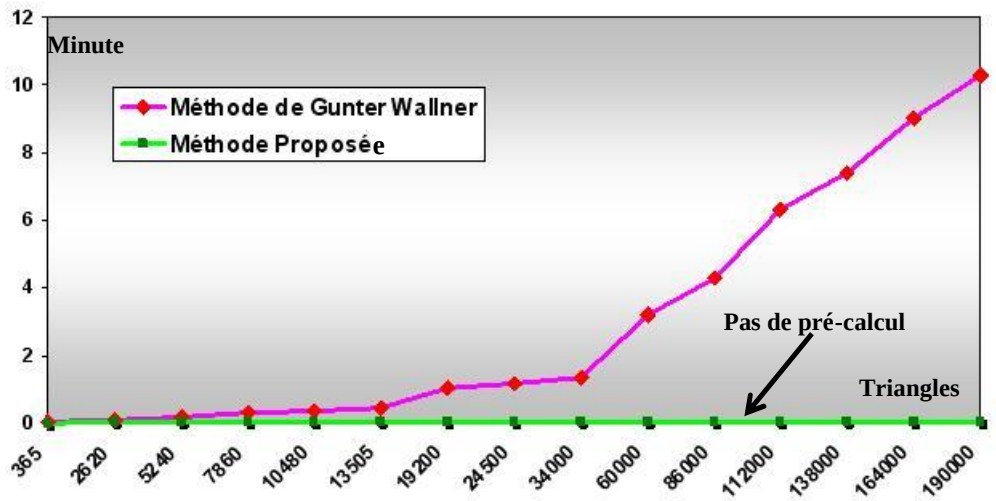


Figure 66: Graphique du temps de pré-calcul mesuré par rapport au nombre de triangles dans une scène, avec deux types de rendue différents (scène rendue avec la méthode de Gunter Wallner, scène rendue avec la méthode proposée).

Le graphique de la figure 66 illustre les variations du temps de pré-calcul par rapport aux nombres de triangles dans la scène, on observe que le volume d'ombre dans la version de Gunter Wallner, n'est pas coûteux en temps de pré-calcul quand on l'applique sur des objets 3D comportant moins de 1000 triangles . Et au-delà de ce seuil, il devient très coûteux et peut prendre jusqu'à 10 minutes quand on l'applique sur des objets 3D comportant plus de 200000 triangles.

Figure 67: Scène sans ombre (80 objets de 200000 triangles, 3 FPS).

La figure 68 représente le résultat de l'implémentation de l'algorithme du volume d'ombre de Gunter Wallner. Cette scène est interactive uniquement quand on l'applique sur des objets 3D comportant moins de 7000 triangles. Les volumes d'ombre ainsi que l'ombre propre sont clairement visible. Quand on l'applique sur une scène comportant 80 créatures, cette scène génère 1 FPS avec les ombres actives, et produit 3 FPS avec les ombres non actives.

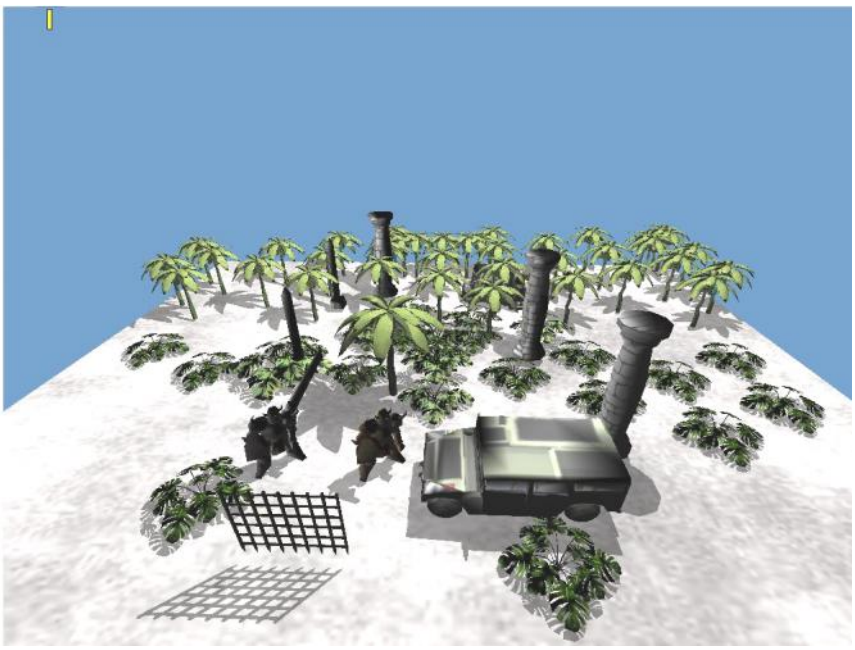


Figure 68: Scène avec ombres, implémentation de la méthode de Gunter Wallner. (80 objets de 200000 triangles, 01 FPS).

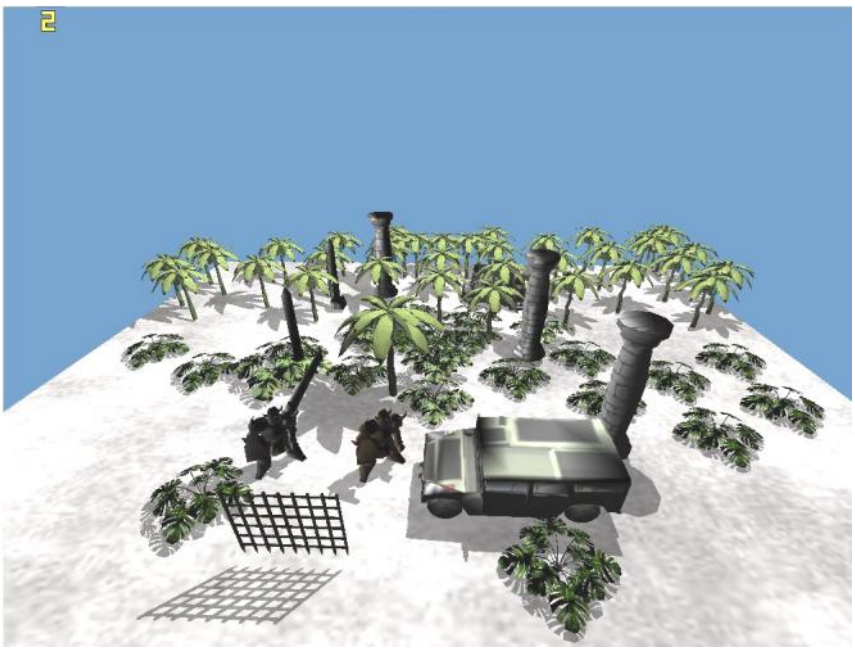


Figure 69: Scène avec ombres, implémentation de notre méthode (en utilisant le Geometry Shader), (80 objets de 200000 triangles, 2 FPS).

La figure 69 représente le résultat de notre algorithme en utilisant le Geometry Shader, et en utilisant les mêmes compositions que la scène précédente (80 objets). Cette scène est interactive uniquement quand on l'applique sur des objets 3D comportant moins de 12000 triangles. Dans ce cas notre scène génère 25 FPS avec les ombres actives, et produit 30 FPS avec les ombres non actives. Quand on l'applique sur des objets 3D comportant 80 créatures, dans ce cas notre scène génère 2 FPS avec les ombres actives, et produit 3 FPS avec les ombres non actives.

Dans le cas où le calcul du volume d'ombre est appliquée sur des objets 3D comportant moins de 10000 triangles, la scène est rendue dans sa version CPU avec un taux d'affichage d'environ 30 FPS sans activer les volumes d'ombre et elle est rendue avec un taux d'affichage de 20 FPS, si on active les volumes d'ombre. Si l'on utilise notre implémentation dans sa version GPU, la vitesse du rendu devient à environs 26 FPS avec les volumes d'ombre activés. Cette vitesse est tout à fait acceptable, et permet des interactions de l'utilisateur, qu'il s'agis se de déplacement de caméra ou de la lumière.

4. Limitations

Dés qu'on augmente la complexité de la scène, notre algorithme ne devient plus interactif, quelque soit le nombre de triangles, dans ce cas pour avoir de l'interactivité il ne faut pas dépasser un certain seuil de complexité de la scène.

Notre algorithme ne fonctionne que pour une seule lumière ponctuelle, de type projecteur. Il serait possible de l'adapter pour pouvoir traiter plusieurs sources lumineuses. Pour cela, pour chacune des lumières, il faut créer un volume d'ombre, et effectuer le calcul d'ombre. Ensuite, en combinant les résultats, il est possible d'obtenir les ombres pour l'ensemble des lumières.

Enfin, il faut noter que lors de l'utilisation de modèle volumineux (scène MD2 de quelques milliers de triangles), le comportement du programme tend à devenir assez hasardeux. Des blocages de quelques secondes se font ressentir, allant même jusqu'à la saturation du processeur graphique et à l'arrêt brutal du programme.

5. Comparaison des résultats

La comparaison des résultats obtenus par l'implémentation de la méthode Gunter Wallner [WAL08] et celle proposée (basée GPU) montre visiblement l'amélioration de la rapidité du rendu et l'optimisation du temps de calcul. Les résultats de la figure 68 donnent une fréquence d'affichage des images de 1 FPS, tandis que les résultats de la figure 69 donnes une fréquence d'affichage des images de 2 FPS, avec la même composition de la scène.

Finalement, les résultats obtenus par notre proposition avec l'accélération GPU en utilisant le Geometry Shader ont donné lieu à un temps de réponse interactif convenable aux usagers au contraire de l'implémentation CPU qui n'était pas interactive. Et les résultats obtenus de notre proposition avec le Geometry Shader n'ont pas besoin de pré-calcul, au contraire de l'implémentation dans sa version CPU qui exige un pré-calcul.

6. Conclusion et perspectives

Nous avons présenté une amélioration d'une méthode de volume d'ombre en rendu temps réel basée sur le Geometry Shader, capable d'effectuer des rendus en temps réel. Cette amélioration concerne la partie de détection et l'extrusion de silhouette, sans avoir besoin de prétraitement.

L'intégration des Geometry Shaders pour le calcul de la silhouette pour les volumes d'ombre à montré son efficacité dans l'optimisation du temps de calcul, ce qui nous a permis d'obtenir la génération des scènes en temps réel. La contribution que nous avons présentée dans ce papier constitue une première étape , ainsi nous envisageons dans le futur d'étendre notre étude pour d'autres scènes plus complexes, et la prise en charge des niveaux de détails dans la génération des ombres.

Les futurs développements devraient prendre en considération l'exploitation des avancements dans les techniques, certes plus complexes mais surtout plus efficaces. D'un autre coté, on pourra envisager l'optimisation des volumes d'ombre en faisant appel aux extensions d'OpenGL plus avancé tout en considérant la possibilité de faire évoluer les ombres vers des résultats toujours plus rapide.

Conclusion générale

A travers ce travail, nous avons présenté le rendu des volumes d'ombre en temps réel. Les ombres jouent un rôle fondamental dans le réalisme d'une scène 3D. Vu la puissance apporter par les processeurs graphiques moderne, des solutions pour l'accélération du calcul afin de générer des volumes d'ombre en temps réel peuvent être réalisé.

Les techniques de génération des volumes d'ombre en temps réel introduisent un certain nombre de difficultés quant à leur visualisation. Le problème principal réside dans l'interactivité car il faut calculer en temps réel le rendu de plusieurs images à entrelacer. En général, ces calculs dépassent les capacités de traitement des cartes graphiques actuelles dans le cas de scènes complexes qui contiennent un nombre important de polygone. Les méthodes ainsi basées sur des maillages produisent des résultats assez satisfaisants, en termes de qualité et de performances, mais se montrent globalement complexes et très lourdes à mettre en place en ayant, par exemple, recours à de longues phases de pré-calculs implémenter sur le CPU, comme pour la génération d'information de connectivité entre les primitives de base qui constituent la maille de l'objet à rendre afin de générer le volume d'ombre. Pour résoudre ce problème, il est possible de faire appel aux GPUs pour optimiser le calcul des volumes d'ombre.

Les algorithmes en temps réel permettent une génération de volume d'ombre interactive, 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 (jeux vidéo), où on ajoute quelques objets dynamiques (personnages,

véhicules, etc.) dans un monde souvent énorme, il apparaît intéressant de pouvoir optimiser les volumes d'ombre.

Le réalisme des ombres vient souvent au prix d'un temps de calcul considérable. Les récentes innovations dans le domaine du matériel graphique ont rendu possible de nouvelles méthodes de génération interactive du volume d'ombre. Bien que ces méthodes donnent souvent de bons résultats pour un nombre limité d'objets, elles ne possèdent pas le très haut niveau de qualité.

Après étude des travaux antérieurs, nous avons constaté que les volumes d'ombre exigent plusieurs critères qui conduisent à alourdir la scène. Un de ces critères est la détection d'information de connectivité entre les primitives de base du volume d'ombre qui exige un prétraitement implémenté sur le CPU, le fait que l'objet qui projette d'ombre soit sous forme de maillage polygonal fermé, et nous avons vu aussi qu'il n'existe aucune maille qui est à base de silhouette. En s'appuyant sur la nouvelle génération de cartes graphiques, les GPUs, permettent, pour chaque pixel affiché à l'écran, de réaliser des opérations programmables, accroissant ainsi le réalisme en minimisant le coût.

L'intégration des Geometry Shaders pour le calcul de la silhouette et pour la détection de l'information d'adjacence entre les primitives de base des mailles pour les volumes d'ombre a montré son efficacité dans l'optimisation du temps de calcul, ce qui nous a permis d'obtenir la génération des scènes en temps réel. La contribution que nous avons présenté dans ce mémoire constitue une première étape, on aura plus besoin dés aujourd'hui de calculer l'information d'adjacence entre les primitives de base qui constitue la maille des objets sur le CPU, car cette tâche est désormais implémenter par le Geometry Shader sur le GPU, ainsi nous envisageons dans le futur d'étendre notre étude pour d'autres scènes plus complexes, et la prise en charge des niveaux de détails dans la génération des ombres.

Nous pouvons enfin dire que la nouveauté apportée par notre travail, consiste à l'accélération du calcul des volumes d'ombre en utilisant les nouveautés apportées par les processeurs graphiques dons le but est de permettre un rendu en temps réel.

Bibliographie

[AAM04] Timo Aila et Tomas Akenine-Möller : "A hierarchical shadow volume algorithm". In Proceedings of Graphics Hardware 2004, pages 15–23. Eurographics, Eurographics Association, 2004.

[AB01] Gooch A. A., Gooch B.: "Non-Photorealistic Rendering". AK Peters, Ltd., July 1 2001. ISBN: 1568811330, 250 pages.

[AMH02] Tomas Akenine-Möller et Eric Haines: "Real-time Rendering". AK Peters, 2e édition, 2002.

[Amo04] Jean-François St-Amour , "Génération d'ombres floues provenant de sources de lumière surfaciques à l'aide de tampons d'ombre étendus" Mémoire présenté à la Faculté des études supérieures en vue de l'obtention du grade de Maître des sciences (M.Sc.) en informatique, l'université de Montréal, Août, 2004.

[Amm10] Lucas Ammann, "Visualisation temps réel de données à deux dimensions et demie", Thèse de doctorat, Université de Strasbourg, 17 septembre 2010.

[ASH04] Ashikhmin M.: "Image-space silhouettes for unprocessed models". In GI '04: Proceedings of Graphics Interface 2004 (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004), Canadian Human-Computer Communications Society, pp. 195–202.

[AW04] Graham Aldridge et Eric Woods: "Robust, geometry-independent shadow volumes". In Proc. 2nd International Conference on Computer graphics and Interactive Techniques in Australasia and Southeast Asia (Graphite), volume 2, pages 250–253. ACM, ACM Press, June 2004.

[Bli88] Jim Blinn. "Me and my (fake) Shadow". IEEE Computer Graphics and Applications, Vol. 8, No. 1, 1988, pp. 82-86

[Ber86] W. Bergeron. "A general version of crow shadow volumes". IEEE Computer Graphics and applications, 6(9) :17.28, 1986.

[BJ99] Harlen Costa Batagelo et Ilaim Costa Junior: "Real-time shadow generation using bsp trees and stencil buffers". In Proc. SIBGRAPI 99, pages 93–102, October 1999.

[BS99] Bill Bilodeau et Mike Songy : "Real time shadows". In Creative Labs Sponsored Game Developer Conference. Creative Labs Inc., May 1999.

[Boy07] Edmond Boyer " Rendu réaliste " cour Master Informatique 2007/2008 – Synthèse d'images, institut d'informatique et Mathématiques Appliquées de Grenoble, http://perception.inrialpes.fr/people/ Boyer/-Teaching/Master/c7.pdf.

[BP07] Lichtenbelt B., Brown P.: EXT_gpu_shader4 Extensions Specifi cations. NVIDIA, 2007.

[CD03] Eric Chan et Frédo Durand: "Rendering fake soft shadows with smoothies". In 14th Eurographics Symposium on Rendering, ACM Press, Leuven Belgium, pages 208–218, 25-27 juin 2003.

[CHH02] Nathan A. Carr, Jesse D. Hall et John C. Hart : "The ray engine". In Proceedings of SIGGRAPH / Eurographics Workshop on Graphics Hardware, pages 37–46, septembre 2002.

[CW93] Michael F. Cohen et John R. Wallace: "Radiosity and Realistic Image Synthesis". Academic Press Professional, Inc. San Diego, CA, USA, 1993.

[Cro77] Franklin C. Crow: "Shadow algorithms for computer graphics". Proceedings of the 4th annual conference on Computer graphics and interactive techniques SIGGRAPH 77, Association for Computing Machinery, Inc. volume 11, pages 242–248, juillet 1977.

[CD04] Eric Chan et Frédo Durand: "An efficient hybrid shadow rendering algorithm". In Proc. Eurographics Symposium on Rendering, pages 185–195. Eurographics, Eurographics Association, 2004.

[Coh04] Florent Cohen, "Ombrage et illumination pour le rendu 3D de forêts en temps réel", Institut National Polytechnique de Grenoble, Master Recherche Imagerie, Vision, Robotique, Juin 2004 .

[Cha06] Pierre Y. Chatelier " Une approche de la radiosité par voxels, application à la synthèse d'images " Thèse de doctorat, l'Université d'Auvergne Spécialité : Informatique, 4 décembre 2006.

[CMJ08] Dyken C., Reimers M., Seland J.: "Realtime gpu silhouette refinement using adaptively blended bézier patches". Computer Graphics Forum 27, 1 (Mar 2008), 1–12.

[Car00] John carmack, "on shadow volumes", 2000. URL http://developer.nvid-ia.com/attach/5628.

[CCOD04] 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.

[DW04] Damien Damour, Thomas Wittmann, "Shadow Volumes", Travail d'étude et de recherche Université Paris-Sud XI – 2004.

[Dif96] Paul J. Diefenbach : Multi-pass Pipeline Rendering : "Interaction and Realism through Hardware Provisions". Thèse de doctorat, University of Pennsylvania, 1996.

[DZY08] Chenguang Dai, Yongsheng Zhang, Jingyu Yang: "RENDERING 3D VECTOR DATA USING THE THEORY OF STENCIL SHADOW VOLUMES".The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. XXXVII. Part B2. Beijing 2008.

[EK03] Cass Everitt et Mark J. Kilgard : "Practical and robust stenciled shadow volumes for hardware-accelerated rendering". Rapport technique, nVIDIA, 2003.

[EMN08] Haines Eric., AKenine-Möller , AND Hoffman Naty. "Real-Time Rendering Third Edition" , A. K. Peters, Ltd., ISBN 978-1-56881-424-7, 2008.

[EASW10] Elmar Eisemann, Ulf Assarsson, Michael Schwarz, Michael Wimmer, "Shadow Algorithms for Real-time Rendering" Eurographics. 2010.

[EASW09] Elmar Eisemann, Ulf Assarsson, Michael Schwarz and Michael Wimmer "Casting Shadows in Real Time", SIGGRAPH Asia 2009 Course Notes.

[FFBG01] Randima Fernando, Sebastian Fernandez, Kavita Bala et Donald P. Greenberg : "Adaptive shadow maps". In SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, ACM Press, New York, NY, USA, 387–390. pages 387–390, août 2001.

[FE99] Benichou Fabien, and Gershon Elber, "Output Sensitive Extraction of Silhouettes from Polygonal Geometry", In Proceedings of the 7th Pacific Conference on Computer Graphics and Applications, Seoul, Korea, pp. 60-69, 1999.

[Fré06] Mora Frédéric. "Visibilité polygone à polygone : calcul, représentation, applications". Thèse Doctorat de l'université de Poitiers, 10 juillet 2006.

[Fau02] François Faure , "Illumination" , cour en informatique ,18 mars 2002. http://ebook-cours.com/cour/informatique/infogra/illumination.pdf .

[GWS04] Johannes Günther, Ingo Wald et Philipp Slusallek : "Realtime Caustics using Distributed Photon Mapping". In Rendering Techniques 2004, Proceedings of the 15th Eurographics Workshop on Rendering Techniques, Norköping, Sweden, pages 111–121, juin 2004.

[Hei91] Tim Heidmann : "Real shadows real time". IRIS Universe, volume 18 page 23–31. Silicon Graphics Inc., 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), pp. 753–774, 2003.

[Hor06] Samuel Hornus. " Maintenance de la visibilité depuis un point mobile, et applications", Thèse présentée pour l' obtention du titre de Docteur de l'Université Joseph Fourier, 22 mai 2006.

[HPV09] P. Hermosilla & P.P. Vázquez: "Single Pass GPU Stylized Edges", IV Iberoamerican Symposium in Computer Graphics - SIACG (2009).

[HHCG03] Hartner A., Hartner M., Cohen E., Gooch B.: "Object Space Silhouette Algorithms". Unpublished, 2003.

[HHL05] Samuel Hornus, Jared Hoberock, Sylvain Lefebvre, and John C. Hart. "ZP+: correct z-pass stencil shadows". In Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games). ACM, ACM Press, 2005.

[Hen04] David Henry. "Le format MD2", 2004. URL : http://tfcduke.developpez-.com/tutoriel/format/md2/.

[Jen96] HenrikWann Jensen: "Global illumination using photon maps". In Xavier PUEYO et Peter SCHRÖDER, éditeurs: Eurographics Rendering Workshop 1996, pages 21–30, New York City, NY, Eurographics, Springer Wein, juin 1996.

[Jen01] Henrik Wann Jensen: "Realistic Image Synthesis using Photon Mapping". A. K. Peters, Ltd. Natick, MA, USA ©2001

[JC95] Henrik Wann Jensen and Niels J. Christensen. "Efficiently Rendering Shadows Using the Photon Map". In Harold P. Santo, editor, Edugraphics + Compugraphics Proceedings, pages 285–291, Portugal, December 1995. GRASP- Graphic Science Promotions and Publications.

[JRL09] Randi J. Rost and Bill Licea-Kane. "OpenGL Shading Language Third Edition". Addison-Wesley, 2009.

[JLCD02] Mitchell J. L., Brennan C., Card D.: "Real-time image-space outlining for non-photorealistic rendering". In Siggraph 02 (2002).

[JJG05] Jérôme 'JeGX' Guinot. "Ombres Volumiques Stencil Shadow Volumes", 2005. URL http://www.ozone3d.net/tutorials/stencil_shadow_volumes.php .

[Kil01] Mark J. Kilgard: "Robust stencil shadow volumes". In CEDEC Presentation, Tokyo, 2001.

[KSYM03] Govindaraju, Naga K., Avneesh Sud, Sung-Eui Yoon, and Dinesh Manocha, "Interactive Visibility Culling in Complex Environments using Occlusion-Switches", Proceedings of the 2003 symposium on Interactive 3D graphics, pp. 103-112, 2003.

[LWGM04] Brandon Lloyd, Jeremy Wend, Naga K. Govindaraju, and Dinesh Manocha. "Cc shadow volumes". In Rendering Techniques, In Proc. Eurographics Symposium on Rendering. Eurographics, Eurographics Association, pages 197.206, 2004.

[Lef06] Sylvain Lefebvre, "Modèles d'habillage de surface pour la synthèse d'images", Thèse présentée pour l'obtention du titre de Docteur, Université Joseph Fourier Spécialité Informatique, 2006.

[MT04] T. Martin et T. Tan: "Anti-aliasing and continuity with trapezoidal shadow maps". School of Computing, National University of Singapore, In Eurographics Symposium on Rendering, pages 153 –160, 2004.

[MD00] McCool, Michael D., "Shadow Volume Reconstruction from Depth Maps", ACM Transactions on Graphics, vol. 19, pp. 1-26, 2000.

[Mig04] Pascal Mignot, Cours de Maîtrise d'informatique, Université de Reims (FRANCE) Année Uniersitaire 2004-2005.

[MJ04] Mcguire M., Hughes J. F.: "Hardware- determined feature edges". In NPAR '04: Proceedings of the 3rd international symposium on Nonphotorealistic animation and rendering (New York, NY, USA, 2004), ACM, pp. 35–47.

[Ngu99] H. Nguyen, 1999. "Casting Shadows on Volumes". Game Developer 6 (3): 44-53.

[Nev08] Marc Neveu "Les tampons : Frame Buffer, Stencil Buffer, Depth Buffer, Accumulation Buffer et autres buffers", UFR Sciences et Techniques Serveur pédagogique ; 2008. URL: http://ufrsciencestech.u-bourgogne.fr/m2via/MGSI/Stencil/Stencil%20Buffer.ppt.

[PBMH02] Timothy J. Purcell, Ian BUCK,William R. MARK et Pat HANRAHAN : Ray tracing on programmable graphics hardware. Proceedings of ACM SIGGRAPH 2002,San Antonio Texas USA , 21(3):703 –712, juillet 2002.

[Pin02] Jean-Marie Pinel , "Etude des conditions d'éclairement dans une séquence d'images et application à la composition et au codage de scènes vidéo" , Thèse de doctorat, Université de Rennes1, 28 novembre 2002.

[PWG78] Atherton Peter, Kevin Weiler, and Donald Gree nberg, "Polygon Shadow Generation", Program of Computer Graphics, pp. 275 - 281,1978.

[Pho75] B.T. Phong, "Illumination for computer generated pictures", Communication of the ACM, 18(6):311 -317, 1975.

[Pau95] Mathias Paulin " ALGORITHMES POUR LA RADIOSIT E :PARALLELISME ET ÉCHANTILLONNAGE " Thèse de doctorat, Université Paul Sabatier de Toulouse III, décembre 1995.

[Ron09] Frédérik Rozon" Peinture de lumière incidente dans des scènes 3D", Mémoire présenté à la faculté des études supérie ures et postdoctorales en vue de l'obtention du grade de Maître ès sciences (M.Sc.) en informatique,Université de Montréal, Août 2009.

[Rog07] David Roger " Lancer de rayons et anti -aliasing", Cours de rendu Master 2 en informatique, ARTIS Laboratoire LJK Rhône-Alpes Montbonnot, France, 24 novembre 2007. URL: http://artis.imag.fr/Members/David.Roger/coursRayTracing.pdf .

[RAS01] Raskar R.: "Hardware support for nonphotorealis tic rendering". In 2001 SIGGRAPH / Eurographics Workshop on Graphics Hardware (2001), ACM Press, pp. 41–46.

[RIC07] Christophe Riccio, "NVIDIA G80: Programmation OpenGL ", Tutorial Edité et Traduit par The oZone3D Team. Mars 2007. Available from: http://www.ozone3d.net/tutorials/g80_opengl_programming/p08.php?la ng=1.

[SD02] Marc Stamminger et George Drettakis: "Perspective shadow maps". SIGGRAPH 2002, San Antonio, Texa s, USA, 21(3):557–562, July 23-26, 2002. ACM 2002.

[Shi00] Peter Shirley : " Realistic Ray Tracing". A.K. Peters, 2000.

[SP94] François Sillion et Claude Puech : "Radiosity and Global Illumination". Morgan Kaufmann, 1994.

[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.

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

[Ste06] Christian Steiner: "Shadow Volumes in Complex Scenes". Master thesis, university of Vienna, May 2006.

[Ski01] Michael Skinner. "Shadows",2001 . URL: http://www.gamedev.net-/reference/articles/article1300.asp

[Sch09] Audric Schiltknecht. "Ombres douces en temps réel". Automne 2009.

[Schw09] Michael Schwärzler. "Accurate Soft Shadows in Real-Time Applications". Février 2009.

[Sam05] Laine, Samuli, Split-Plane Shadow Volumes, in Proceedings of Graphics Hardware 2005, pp. 23-32, 2005.

[Tha89] Tessman Thant, "Casting Shadows on Flat Surfaces", Iris Universe, pp. 16-19, 1989. Silicon Graphics, Inc.

[Tha03] Daniel Thalmann " INFOGRAPHIE " , Ecole Polytechnique Fédérale de Lausanne, Mars 2003.

[WPF90] Andrew Woo, Pierre Poulin et Alain Fournier: "A survey of shadow algorithms". IEEE Computer Graphics & Applications, 10(6):13–32, novembre 1990.

[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.

[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.

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

[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 etWerner purgathofer : "Light space perspective shadow maps". In Eurographics Symposium on Rendering, 2004.+

[WAL08] Gunter Wallner, "Geometry of Real Time Shadows", in international journals or conference proceedings, University of Applied Arts Vienna, Department of Geometry between 2005 and 2008.

[Yaz06] Shi Yazheng, "Performance comparison of CPU and GPU silhouette extraction in a shadow volume algorithm", Master's Thesis in Computing Science, 10 credits, Department of Computing Science SWEDEN, January 27, 2006.

[ZB+11a] A. Zerari, M.C. Babahenini, "Shadow Volume in real-time rendering", In Proceedings IEEE-2011 (CCCA'11) March 3-5, 2011 at Hammamet, Tunisia.

[ZB+11b] A. Zerari M.C. Babahenini, " Optimisation des volumes d'ombre pour un rendu temps réel", Rencontres sur la Recherche en Informatique Université Mouloud Mammeri Tizi-Ouzou 12-14 Juin 2011, Algérie.

[ZB+11c] A. Zerari, M.C. Babahenini, "Shadow Volume in real-time rendering", Accepted In Journal of Computer Technology and Application (ISSN: 1934-7332, USA), it will be published on Volume 2, Number 8, 2011.

[ZB+11d] A. Zerari, M.C. Babahenini, Optimisation des volumes d'ombre, accepté à TAIMA' 2011, Conférence sur le Traitement et l'Analyse de l'Information: Méthodes et Applications, Tunisie 03 au 08 octobre 2011.