



**HAL**  
open science

# Partition de volumes d'ombres : une alternative pour le rendu d'ombres en temps réel

Julien Gerhards

► **To cite this version:**

Julien Gerhards. Partition de volumes d'ombres : une alternative pour le rendu d'ombres en temps réel. Géométrie algébrique [math.AG]. Université de Limoges, 2017. Français. NNT : 2017LIMO0083 . tel-01701895

**HAL Id: tel-01701895**

**<https://theses.hal.science/tel-01701895>**

Submitted on 6 Feb 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

**UNIVERSITÉ DE LIMOGES**

ÉCOLE DOCTORALE Science et Ingénieries pour l'information

FACULTÉ DES SCIENCES ET TECHNIQUES

Année : 2017

**Thèse**

pour obtenir le grade de

**DOCTEUR DE L'UNIVERSITÉ DE LIMOGES**

**Discipline : Informatique et Application**

présentée et soutenue par

**Julien Gerhards**

le 15 novembre 2017

**Partition de volumes d'ombre : une alternative pour  
le rendu d'ombres en temps réel**

**Thèse dirigée par Djamchid Ghazanfarpour**

**Co-encadrée par Frédéric Mora**

**JURY :**

<b>Christophe RENAUD</b>	Professeur, Université de Dinkerque	Président
<b>Kadi BOUATOUCH</b>	Professeur, Université de Rennes	Rapporteur
<b>Mateu SBERT</b>	Professeur, Université de Gérone	Rapporteur
<b>Djamchid GHAZANFARPOUR</b>	Professeur	Examineur
<b>Frédéric MORA</b>	Maitre de conférence	Examineur



« *Fidèle comme une ombre...* »

Expression populaire



# Table des matières

Table des figures . . . . .	4
Liste des tableaux . . . . .	8
Introduction générale . . . . .	10
<b>Chapitre 1 . Ombres temps réel : État de l’art . . . . .</b>	<b>12</b>
1.1 L’ombre . . . . .	14
1.1.1 Définition . . . . .	14
1.1.2 L’importance de l’ombre . . . . .	14
1.1.3 L’ombre du point vue géométrique . . . . .	15
1.2 État de l’art des techniques de rendu temps réel d’ombre dure . . . . .	17
1.2.1 La Shadow Map . . . . .	18
1.2.1.1 Le principe . . . . .	18
1.2.1.2 Artefacts et aliassage . . . . .	19
1.2.1.3 Shadow Mapping : Premier bilan . . . . .	21
1.2.1.4 Les sources de lumières omnidirectionnelles . . . . .	24
1.2.1.5 Limitation du biais . . . . .	24
1.2.1.6 Perspective Shadow Map . . . . .	24
1.2.1.7 Cascade de cartes d’ombre ou cascading Shadow Maps . . . . .	26
1.2.1.8 Shadow Map par objet . . . . .	27
1.2.1.9 Shadow Mapping : conclusion . . . . .	27
1.2.2 Les volumes d’ombre ou Shadow Volumes . . . . .	29
1.2.2.1 Le principe . . . . .	29
1.2.2.2 Le Z-pass . . . . .	29
1.2.2.3 Artefact . . . . .	31
1.2.2.4 Le Z-fail . . . . .	31
1.2.2.5 Volumes d’ombre par extraction des silhouettes . . . . .	32
1.2.2.6 Limitation de la saturation de la bande passante mémoire . . . . .	33
1.2.2.7 Shadow Volumes : conclusion . . . . .	34
1.2.3 Les techniques hybrides . . . . .	34
1.2.3.1 Shadow Map irrégulière . . . . .	35
1.2.3.2 Volume d’ombre par triangle . . . . .	36
1.2.3.3 Rastérisation irrégulière et lancer de rayon . . . . .	39
1.2.3.4 Arbre BSP de volume d’ombre ou SVBSP . . . . .	39
1.3 Conclusion . . . . .	42
<b>Chapitre 2 . Partition de Volumes d’ombre . . . . .</b>	<b>43</b>
2.1 Inadaptation des SVBSP au traitement sur GPU . . . . .	44
2.2 Introduction de la partition ternaire des objets . . . . .	45
2.3 Partition de volumes d’ombre . . . . .	45
2.3.1 Construction d’une partition de volumes d’ombre . . . . .	47
2.3.2 Parcours d’une partition de volumes d’ombre . . . . .	50
2.3.3 Extension à la transparence . . . . .	55
2.3.4 Implémentation . . . . .	57
2.3.5 Résultats . . . . .	59
2.4 Conclusion . . . . .	67
<b>Chapitre 3 . Analyse et optimisations . . . . .</b>	<b>68</b>
3.1 Analyse des partitions de volumes d’ombre . . . . .	69
3.2 Prise en compte de la profondeur par rapport à la lumière . . . . .	71
3.3 Un parcours sans pile . . . . .	72
3.4 Parcours hybride . . . . .	79
3.5 Implémentation . . . . .	80
3.6 Résultats . . . . .	82
3.7 Limitations . . . . .	91
3.8 Conclusion . . . . .	93

**Conclusion** . . . . . 98

# Table des figures

1.1	Le système visuel humain. . . . .	14
1.2	Le tapis volant. . . . .	15
1.3	L'homme qui lévite. . . . .	15
1.4	Information sur le positionnement relatif des objets. . . . .	15
1.5	Information sur la forme du receveur. . . . .	15
1.6	Qui est-ce? . . . . .	15
1.7	Et lui? . . . . .	15
1.8	Une source de lumière lointaine. . . . .	16
1.9	Une source de lumière proche. . . . .	16
1.10	Ombre dure avec une source ponctuelle en 2 dimensions. . . . .	16
1.11	Ombre dure avec une source ponctuelle en 3 dimensions. . . . .	16
1.12	Une source de type spot et une source omnidirectionnelle. . . . .	17
1.13	Une scène. . . . .	18
1.14	Le tampon de profondeur associé. . . . .	18
1.15	Principe de la Shadow Map. . . . .	19
1.16	Artefact d'auto-ombrage. . . . .	19
1.17	Acné. . . . .	20
1.18	Effet Peter Pan. . . . .	20
1.19	Rendu des faces arrières. . . . .	21
1.20	Aliassage de perspective. . . . .	21
1.21	Aliassage de perspective en pratique. . . . .	21
1.22	Aliassage de projection. . . . .	22
1.23	Shadow Map 512 x 512. . . . .	23
1.24	Shadow Map 2048 x 2048. . . . .	23
1.25	Précision du tampon de profondeur. . . . .	24
1.26	Frustum adaptatif. . . . .	25
1.27	Rendu sans et avec frustum adaptatif. . . . .	25
1.28	Déformation perspective de la Shadow Map. . . . .	26
1.29	Rendu avec une Shadow Map de 512 x 512, à gauche avec une shadow classique et à droite avec une perspective Shadow Map. . . . .	26
1.30	Frustum de vue de la caméra. . . . .	27
1.31	Découpe du frustum de vue de la caméra . . . . .	27
1.32	Rendu sans (à gauche) et avec (à droite) une Shadow Map spécifique pour la main et l'arme au premier plan. . . . .	27
1.33	Ombre par Shadow Volume dans Doom 3. . . . .	29
1.34	Volumes d'ombre générés par 2 triangles. . . . .	30
1.35	Les différentes étapes de l'application des shadows volumes. . . . .	30
1.36	De grands shadow quads. . . . .	30
1.37	Rendu incorrect. . . . .	31
1.38	Rendu correct. . . . .	31
1.39	Du Z-fail au Z-pass. . . . .	32
1.40	Simplification de la géométrie grâce à la silhouette. . . . .	32
1.41	Remplissage du compteur par pixel avec un objet fermé. . . . .	33
1.42	Remplissage du compteur par pixel avec un objet non fermé. . . . .	33
1.43	Correction du remplissage du compteur par pixel avec un objet non fermé. . . . .	33
1.44	Élimination des objets cachés du point de vue de la lumière. . . . .	33
1.45	Limitation de la zone de rendu des volumes d'ombre. . . . .	33
1.46	Élimination des objets caché du point de vue de la caméra. . . . .	33
1.47	Pour un même objet, un point de vue favorable (à gauche) et un point de vue très défavorable (à droite) pour les méthodes basées Shadow Volumes. . . . .	35
1.48	Projection des pixel sur une Shadow Map . . . . .	35
1.49	Transformation des pixels sous forme de hiérarchie de frustums. . . . .	36
1.50	Traversée par un Shadow Volume de la hiérarchie de frustums. . . . .	37
1.51	Changement des Frutrum . . . . .	38
1.52	Un arbre de volumes d'ombre en 2D et sa construction. . . . .	40
1.53	Découpe et duplication des informations géométriques dans un arbre BSP. . . . .	40
1.54	Parcours d'un arbre BSP de volumes d'ombre. . . . .	41

2.1	Cas pathologique pour les SVBSP	44
2.2	Introduction d'une structure de donnée ternaire	45
2.3	Vue d'ensemble	46
2.4	Représentation d'un volume d'ombre avec un arbre ternaire.	46
2.5	Le cas d'une bande de triangles	54
2.6	Volume englobant du sous-arbre intersection	55
2.7	Bunny - 69 k triangles - 8.43 Mo : on simule l'ombre d'un lapin semi-transparent	55
2.8	Implémentation GPU	57
2.9	Stockage	58
2.10	Rendu indirect	59
2.11	Trex - 20 k triangles - 2.44 Mo	60
2.12	Bunny - 69 k triangles - 8.43 Mo	60
2.13	Géodesic - 200 k triangles - 24.42 Mo	60
2.14	Fairy - 174 k triangles - 21.24 Mo	60
2.15	Citadel issue de Unreal Engine de - 393 k triangles - 48 Mo	60
2.16	Raptor - 1000 k triangles - 122 Mo	60
2.17	Comportement rapporté à la résolution	62
2.18	Comportement en fonction de la complexité géométrique	63
2.19	CITADEL	64
2.20	NEW VILLA	64
2.21	FUZZY	64
2.22	TREX : 3.39 ms (construction 0.30 ms et parcours 3.09 ms)	65
2.23	BUNNY : 3.67 ms (construction 0.72 ms et parcours 2.95 ms)	65
2.24	GEODESIC : 8.41 ms (construction 1.99 ms et parcours 6.42 ms)	66
2.25	RAPTOR : 33.90 ms (construction 15.78 ms et parcours 18.12 ms)	66
3.1	Coût en noeuds	69
3.2	Coût en noeuds 2	69
3.3	Traversée d'un point illuminé	70
3.4	Un parcours avec distance	71
3.5	Effet de l'optimisation de distance	72
3.6	Architecture d'un GPU	72
3.7	Parcours sans pile	75
3.8	Un parcours sans pile	75
3.9	Comparaison des différents parcours	76
3.10	Utilisation de la pile	79
3.11	Comparaison des coût en noeuds	80
3.12	Stockage	80
3.13	Frustum adaptatif	81
3.14	Processus persistants	82
3.15	PLAYGROUND - 131 k triangles (16 Mo)	82
3.16	EPIC CITADEL - 394 k triangles issue du SDK de Unreal Engine (48 Mo)	82
3.17	CLOSED CITY - 623 k triangles (76 Mo)	83
3.18	SHIP - 956 k triangles (117 Mo)	83
3.19	ESCAVATOR - 1300 k triangles(138 Mo)	83
3.20	Comparaison des parcours sur 5 scènes	84
3.21	Playground - 131K triangles	86
3.22	Epic Citadel - 393K triangles	87
3.23	Closed Citadel - 623K triangles	88
3.24	Ship - 956K triangles	89
3.25	Escavator - 1300K triangles	90
3.26	Impact de la taille de la pile pour un parcours hybride	91
3.27	SAN MIGEL avec ses 7 millions de triangles	92
3.28	HAIR BALL	92



# Liste des tableaux

2.1	Construction d'une partition de volumes d'ombre . . . . .	48
2.2	Construction d'une partition de volumes d'ombre . . . . .	49
2.3	Traversée d'une partition de volumes d'ombre par un point à l'ombre . . . . .	51
2.4	Traversée d'une partition de volumes d'ombre par un point éclairé . . . . .	52
2.5	Traversée d'une partition de volumes d'ombre par un point éclairé . . . . .	53
2.6	Résultats : la colonne <i>CS</i> indique le temps de construction de la partition de volumes d'ombre, et la colonne <i>FS</i> concerne le parcours de la structure par les fragments. Les lignes <i>No opt</i> , <i>No rand</i> et <i>No FFC</i> correspondent respectivement aux mesures sans l'optimisation minimisant la visite des sous-arbres intersection, sans aléa et sans élimination des triangles faisant face à la lumière. . . . .	61
2.7	Résultats : en vert pour les ombres transparentes, avec le nombre de nœuds moyen, le temps de construction <i>CS</i> , le temps de parcours <i>FS</i> et le temps total. À titre de comparaison, on rappelle en bleu les résultats du calcul de l'ombre dure sur les mêmes scènes et mêmes parcours de 1000 images. . . . .	65
3.1	Résultats . . . . .	85

# Introduction générale

En synthèse d'image, une étape cruciale réside dans le calcul de visibilité pour déterminer par exemple quel objet est visible sur un point de l'image. Plus précisément, on cherche quel polygone du maillage de l'objet est visible en un point de l'image afin d'avoir les informations nécessaires au calcul de l'apparence finale. Un second problème de visibilité se pose alors : la source de lumière est elle visible ou masquée depuis le point dont on souhaite faire le rendu ? Le calcul par force brute étant inefficace, les solutions mises en place, consistent à construire des structures accélératrices qui permettent de diminuer le nombre de tests d'intersection à réaliser. Les données géométriques d'une scène sont réparties dans une structure permettant de subdiviser l'espace. Une des subdivision les plus communes est celle des partitions binaires de l'espace où arbre BSP et sa version améliorée, le Kd-Tree qui vise à proposer une partition optimisée de l'espace en fonction de la densité géométrique de la scène ; alors que l'arbre BVH construit une hiérarchie de volumes englobant définissant une partition de l'espace objet. Ce type de structure est utilisé massivement dans le calcul de rendu qualitatif, notamment le domaine du lancer de rayon.

Dans le cadre du rendu temps réel, on utilise la rasterisation, accélérée matériellement par la carte graphique, qui est à ce jour la technique la plus rapide pour calculer ce qui est visible depuis une caméra : elle permet de transformer les objets vectoriels 2D ou 3D en images matricielles. Il reste à déterminer si les points de l'image sont éclairés par une ou des sources lumineuses. Deux familles de techniques sont utilisées pour répondre à cette question : d'un côté la Shadow Map de Williams [4], solution standard dans l'industrie du jeu vidéo pour sa stabilité et sa vitesse de rendu mais qui donne par nature un résultat inexact ; et de l'autre côté les volumes d'ombre de Crow [5] qui permettent d'obtenir des ombres exactes, mais dont les performances sont très variables suivant le point de vue et fortement dépendantes de la quantité de géométrie. Le choix d'une méthode d'ombrage se ramène à choisir entre qualité et performance.

Dans ce mémoire, on va se focaliser sur le calcul d'ombre en temps réel pour proposer une méthode rapide avec un résultat exact par pixel en supportant des modèles à la géométrie complexe. L'objectif est de proposer une alternative aux techniques classiques basées Shadow Map ou volumes d'ombre en conciliant qualité et performance, y compris sur des modèles à la géométrie complexe.

Ce mémoire se décompose en 3 parties :

- le premier chapitre est un état de l'art du rendu des ombres en temps réel ;
- dans le second chapitre, nous revisitons la technique des arbres BSP de volumes d'ombre, idée abandonnée à cause de ses nombreux défauts, pour proposer une nouvelle structure de données, la partition de volumes d'ombre. Elle se base sur un arbre ternaire pour organiser les volumes d'ombre de la scène. Ensuite, il suffit d'interroger cet arbre pour déterminer l'ombrage d'un point de la scène. Cette nouvelle structure a l'avantage de posséder une empreinte mémoire fixe en fonction de la quantité de géométrie ;
- par la suite, on s'intéresse à l'analyse des performances des partitions de volumes d'ombre afin d'en détecter les faiblesses et de proposer de nouvelles optimisations et parcours permettant de rendre la méthode plus rapide, plus stable tout en supprimant le seul paramètre sensible de la méthode.

## Chapitre 1 .

# Ombres temps réel : État de l'art



*« Il n'y a pas d'ombre sans lumière. »*

Argon

---

*« Si tu vis dans l'ombre,  
tu n'approcheras jamais le soleil. »*

Jacque Mesrine



*« Jamais le soleil ne voit l'ombre. »*

Léonard de Vinci

---

*« J'apprécie peu la lumière,  
et je passe le plus clair de mon temps  
à tenter de l'assombrir. »*

Boris Vian



*« L'homme qui tire plus vite que son ombre. »*

René Goscinny

## 1.1 L'ombre

### 1.1.1 Définition

« **Ombre** (nom, féminin, du latin *umbra*) : Zone sombre résultant de l'interception de la lumière ou de l'absence de lumière : "Au soleil couchant, la vallée est dans l'ombre". Silhouette sombre, plus ou moins déformée, que projette sur une surface un corps qui intercepte la lumière : "Les ombres des arbres s'allongent vers le soir." »

Dictionnaire Larousse

On peut définir les ombres simplement : ce qui est visible depuis une source lumineuse est éclairé, ce qui ne l'est pas est ombré. Du point de vue géométrique, cela revient à déterminer s'il existe, entre une source lumineuse  $s$  et un point  $p$  d'une scène, un élément géométrique qui intersecte le segment  $[ps]$ .

### 1.1.2 L'importance de l'ombre

Pour bien comprendre l'importance de l'ombre, particulièrement en synthèse d'images, il faut revenir à la base du fonctionnement de la perception visuelle humaine. Ainsi pour percevoir un objet

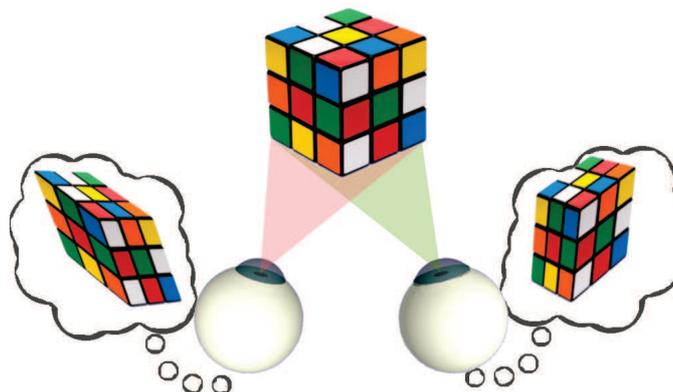


FIGURE 1.1 – Le système visuel humain.

tridimensionnel, le cerveau fusionne les 2 images bidimensionnelles issues des 2 yeux pour générer une perception unique avec du relief. Dans le cas où l'on regarde une image plane comme sur un écran, le cerveau ne peut pas extraire d'information sur le relief (cf. figure 1.1). Il doit alors déduire le relief et les distances entre les objets par d'autres moyens. Observé par un individu privé de vision binoculaire, une image en deux dimensions donnera des perceptions fausses, comme sur les figures 1.2 et 1.3. Dans ces 2 exemples, c'est la mauvaise interprétation des ombres réelles ou fausses qui aboutit à la vision de situations non réalistes.

En pratique, l'ombre permet au cerveau de déterminer un certain nombre d'informations :

- la position relative des objets les uns par rapport aux autres, comme sur la figure 1.4 : l'ombre donne une estimation de la distance entre le personnage et le sol ;
- le relief d'un objet sur lequel une ombre se projette comme l'ondulation du sol (cf. figure 1.5) ;
- la perception d'un objet absent du champ visuel, mais dont l'ombre est visible comme sur les figures 1.6 et 1.7 ;



FIGURE 1.2 – Le tapis volant.



FIGURE 1.3 – L'homme qui lévite.



FIGURE 1.4 – Information sur le positionnement relatif des objets.  
Source : [1] page 3

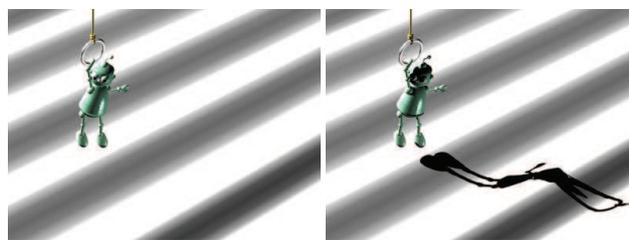


FIGURE 1.5 – Information sur la forme du receveur.  
Source : [1] page 3

— la direction et la distance d'une source lumineuse absente du champ visuel, comme sur les figures 1.8 et 1.9.

L'ombre est donc un élément essentiel pour le réalisme d'une image de synthèse et son interprétation correcte par le cerveau.



FIGURE 1.6 – Qui est-ce ?



FIGURE 1.7 – Et lui ?

### 1.1.3 L'ombre du point vue géométrique

La problématique du calcul d'ombre est une question de visibilité : un point non visible depuis la lumière sera dans l'ombre. Il faut donc être capable de savoir s'il existe un élément géométrique qui coupe le segment de droite reliant la source lumineuse et un point de la scène. Dans ce travail, on se limite au cas des ombres dures.

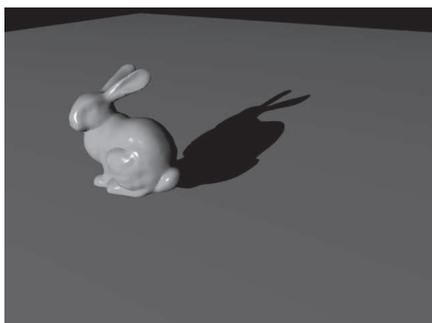


FIGURE 1.8 – Une source de lumière lointaine.

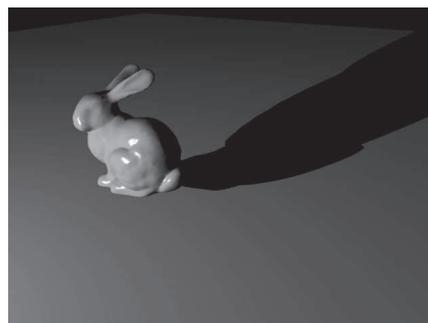


FIGURE 1.9 – Une source de lumière proche.

Les ombres dures sont les plus simples, car on se limite au cas d'une source ponctuelle. On considérera par exemple que, proportionnellement à la distance Terre-soleil, une petite ampoule ou le soleil sont assimilables à des sources ponctuelles. Un élément géométrique va générer une zone d'ombre comme sur

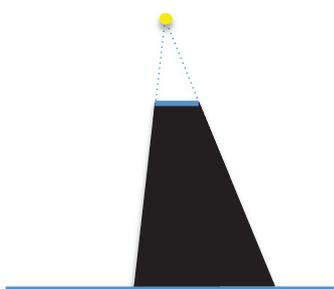


FIGURE 1.10 – Ombre dure avec une source ponctuelle en 2 dimensions.

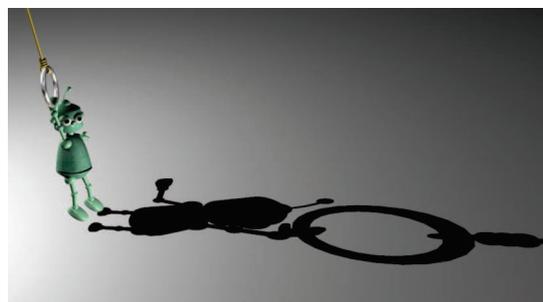


FIGURE 1.11 – Ombre dure avec une source ponctuelle en 3 dimensions.

Source : [1] page 4

la figure 1.10, et le calcul de l'ombre se ramène à un choix binaire : totalement dans l'ombre ou totalement éclairée. On peut d'ores et déjà décrire l'algorithme 1 basique pour obtenir un résultat exact : on teste

---

**Algorithm 1** Principe de l'algorithme de calcul d'ombre dure

---

```

1: CalculerOmbreDure(Point  $p$ , source de lumière  $s$ , Géométrie  $g$ )
2: {
3:   Pour tout élément  $e$  de  $g$ 
4:   {
5:     Si(TesterRayonGeometrie( $p,s,e$ ))
6:       Alors retourner Faux
7:       // Le point est dans l'ombre
8:   }
9:   retourner Vrai
10:  // Le point est dans la lumière
11: }
```

---

l'intersection du rayon lumière-point avec toute la géométrie de la scène, et dès qu'on trouve un élément qui coupe ce rayon on sait que le point est dans l'ombre ; sinon une fois que toute la géométrie est testée on peut conclure que le point est illuminé.

L'application de ce genre d'algorithmes est inenvisageable pour du rendu temps réel : il devient vite très coûteux de tester toute la géométrie pour chaque pixel d'une image. Nous verrons par la suite les différentes stratégies utilisées pour contourner ce problème. En synthèse d'images en temps réel sur GPU

(Graphics Processing Unit ou carte graphique), on représente la géométrie d'une scène avec  $t$  triangles. Pour une image de hauteur  $h$  et de largeur  $l$  et une scène de  $t$  triangles, le nombre de tests à faire pour une image est :

$$h \times l \times t \quad (1.1)$$

## 1.2 État de l'art des techniques de rendu temps réel d'ombre dure

Cette partie se focalisera sur les méthodes de rendu en temps réel des ombres dures. La problématique du rendu des ombres s'est posée dès l'apparition des premières méthodes de rendu. Quasiment tous les travaux dans le domaine se basent sur des techniques imaginées à la fin des années 70 :

- une méthode basée image utilisant le tampon de profondeur, connue sous sa dénomination anglo-saxonne de Shadow Mapping ou technique des Shadow Maps [4] ;
- une méthode basée sur la géométrie, la technique des volumes d'ombre ou Shadow Volumes en anglais [5] ;
- quelques approches alternatives pouvant mélanger plus ou moins les deux premières approches.

Pour classer et comparer les diverses méthodes, il faut d'abord définir plusieurs critères.

- **L'exactitude** : le calcul de la visibilité de la source lumineuse en chaque point de l'image est-il fidèle aux données géométriques ?
- **Le comportement face à la complexité géométrique** : comment se comporte une méthode en fonction du nombre  $t$  de triangles ?
- **Le comportement face à une hausse de la résolution image** : quels sont les coûts d'une méthode face lorsque la résolution de l'image augmente ?
- **Le coût en mémoire** : quel est le coût en mémoire  $m$  d'une méthode ? Est-il fixe ? Est-il prévisible ? La méthode nécessite-t-elle beaucoup de trafic mémoire ?
- **Le type de source supportée** : est-il possible de gérer les sources de type spot et omnidirectionnelle présentées sur la figure 1.12 ?



FIGURE 1.12 – Une source de type spot et une source omnidirectionnelle.

- **Le paramétrage** : y a-t-il des paramètres à choisir qui modifient la qualité du rendu, le temps de calcul, la consommation mémoire ? Est-ce que ces paramètres sont aisément déterminables ou faut-ils les trouver empiriquement ?
- **La stabilité** : est-ce que le temps de calcul de l'ombre peut fortement varier d'une image à l'autre ? Y a-t-il des cas particuliers susceptibles d'augmenter le temps de calcul ?

Les sections suivantes vont expliquer le fonctionnement des méthodes basées Shadow Maps ou Shadow Volumes, leurs évolutions respectives ainsi que leurs avantages et défauts.

### 1.2.1 La Shadow Map

Le concept de Shadow Map a été introduit par Williams Lance [4] il y a 38 ans. C'est la technique qui est massivement utilisée par l'industrie du jeu vidéo, pour des raisons de vitesse liées au fait que les différentes étapes nécessaires bénéficient toutes de l'accélération matérielle des cartes graphiques. Cette méthode jouit ainsi d'une bonne stabilité et d'une faible sensibilité à la complexité géométrique en dépit des nombreux défauts qui seront présentés par la suite.

#### 1.2.1.1 Le principe

Pour obtenir une ombre rapidement, la technique des Shadow Map repose sur une technique de rendu : la *rastérisation*. Lors de la rastérisation d'une scène, on ne sait pas dans quel ordre vont être transformés en fragments les différentes primitives présentes. Le tampon de profondeur par pixel, inventé par Edwin Catmull [6], permet de faire un tri en profondeur pour ne conserver que les éléments qui sont les plus proches du point de vue et autoriser une élimination des fragments qui seraient cachés par d'autres. On peut voir une scène 3D sur la figure 1.13 et le tampon de profondeur associé visualisé sous forme de niveaux de gris (sombres pour les éléments lointains et clairs pour ceux qui sont proches).

On utilise le tampon de profondeur en 2 étapes successives pour mettre en place la technique de la



FIGURE 1.13 – Une scène.



FIGURE 1.14 – Le tampon de profondeur associé.

Shadow Map :

- on place une caméra virtuelle pour faire une rastérisation de la scène du point de vue de la lumière, dont on ne conserve que les informations du tampon de profondeur stockées dans une texture 2D : la Shadow Map (en orange sur la figure 1.15) ;
- on fait une rastérisation de la scène depuis le point de vue de la caméra et pour chaque pixel, on compare simplement la distance pixel-lumière à celle enregistrée dans la Shadow Map : si le fragment est plus éloigné que la profondeur lue dans la Shadow Map, c'est qu'il existe des éléments géométriques qui bloquent le passage de la lumière comme sur la figure 1.15.

La rastérisation et le tampon de profondeur sont implémentés matériellement dans les cartes graphiques ou *GPU* ; c'est ce qui permet au Shadow Mapping d'être une technique très rapide, et vu les performances de rastérisation des GPU modernes, on peut considérer que la complexité géométrique influe peu sur les performances.

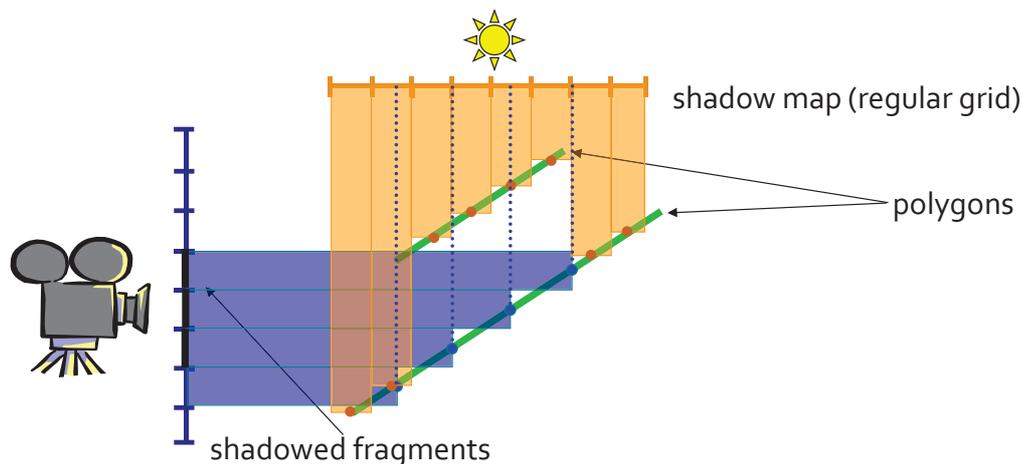


FIGURE 1.15 – Principe de la Shadow Map.  
Source : [7] diapositive 2

### 1.2.1.2 Artefacts et aliassage

Sous cette apparente simplicité, se cachent en pratique de nombreux problèmes : la Shadow Map est une technique basée image, et souffre donc des problèmes d'aliassage liés à l'usage de textures discrétisées. L'application directe de la méthode donne un ombrage avec beaucoup d'artefacts, comme par exemple des problèmes d'auto-ombrage sur les figures 1.16 et 1.17. On peut voir sur la figure 1.17 l'exemple

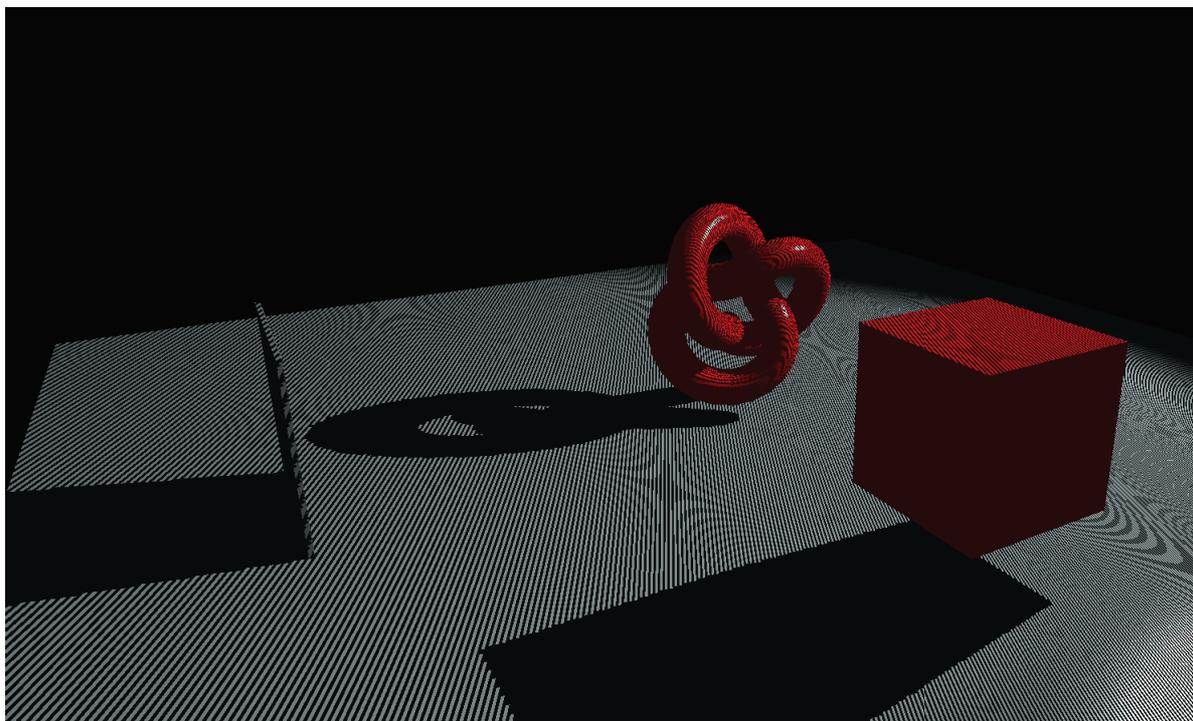


FIGURE 1.16 – Artefact d'auto-ombrage.

d'une surface en noir et en jaune ainsi que les profondeurs lues dans la Shadow Map correspondante : le résultat est une alternance de zones illuminées et ombrées là où il ne devrait pas y avoir d'ombre. On corrige généralement ce problème, dit *shadow acné*, en introduisant un biais : si la distance à la lumière

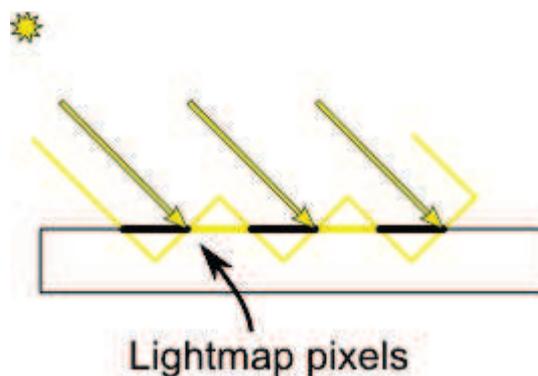


FIGURE 1.17 – Aca.

d'un point de la scène est supérieure au biais plus la profondeur correspondante dans la Shadow Map, on considérera que le point se situe dans l'ombre. On utilise un biais adaptatif qui est fonction de la direction normale du fragment à ombrer et de la direction de la lumière. Il faut noter que l'ajout de ce biais va générer un léger décalage de l'ombre appelé effet Peter Pan, comme sur la capture 1.18, qui peut faire apparaître une zone comme éclairée alors qu'elle devrait être dans l'ombre. Pour éviter cet effet,

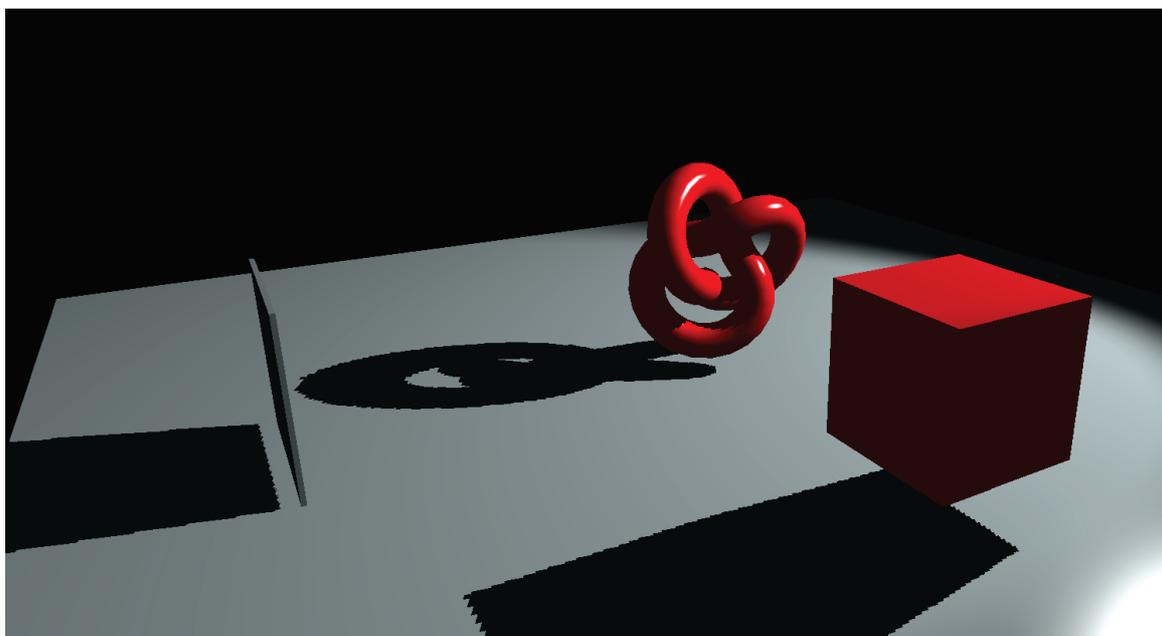


FIGURE 1.18 – Effet Peter Pan.

L'ombre du mur semble indiquer qu'il flotte légèrement au dessus de la surface, ce qui n'est pas le cas sur le modèle géométrique.

la solution proposée par Wang et al.[8] consiste à utiliser des modèles géométriques avec une épaisseur, et lors du rendu de la Shadow Map de ne rendre que les faces arrière des objets comme sur la figure 1.19, ce qui permet aussi de faire disparaître l'effet d'acné. Cette solution impose quelques ajustements particuliers :

- l'épaisseur des modèles géométriques doit être supérieure au biais utilisé ;
- les modèles géométriques doivent avoir des faces orientées correctement.

Le Shadow Mapping souffre d'autres défauts.

- L'aliasage de perspectives : les objets proches de la caméra auront besoin de davantage

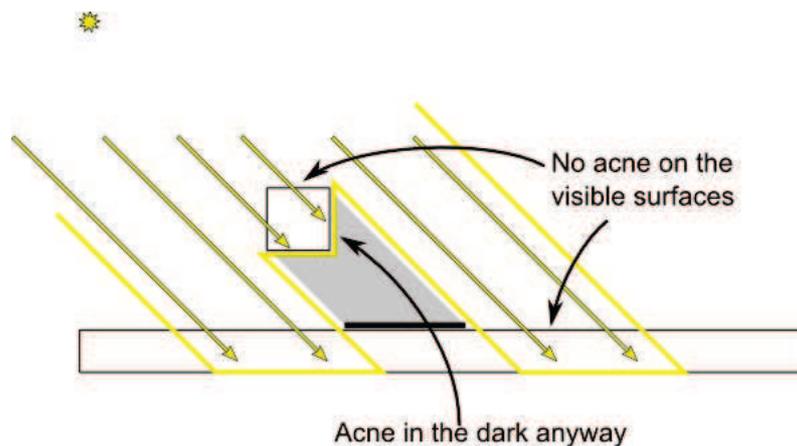


FIGURE 1.19 – Rendu des faces arrières.

d'informations de profondeur que les objets éloignés comme sur la figure 1.20 ; or la SM étant une grille uniforme, on va manquer de détails pour les objets proches, tout en ayant trop d'informations pour les objets éloignés comme sur la figure 1.21.

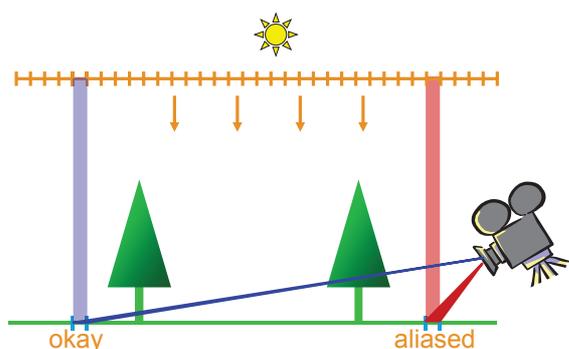


FIGURE 1.20 – Aliassage de perspective.  
Source : [7] diapositive 13



FIGURE 1.21 – Aliassage de perspective en pratique.  
Source : [7] diapositive 13

- L'aliassage de projection : plus les rayons lumineux seront parallèles à une surface, plus cette surface sera échantillonnée sur un faible nombre de texels de la Shadow Map, ce qui génère des erreurs sur l'ombrage et de possibles clignotements dans l'ombrage si la caméra bouge. On peut voir sur la figure 1.22 qu'un texel de la Shadow Map se projette sur plusieurs texels de l'image caméra.
- Les imprécisions du tampon de profondeur : la profondeur est stockée dans le tampon de profondeur sous la forme de  $\frac{1}{z}$ . En résulte une précision importante pour les éléments proches de la caméra et une imprécision croissante lorsqu'on s'éloigne de la caméra.

### 1.2.1.3 Shadow Mapping : Premier bilan

- **L'exactitude** : imprécisions numériques ;
- **Le comportement face à la complexité géométrique** : du point de vue théorique, l'algorithme est linéaire quelque soit le nombre de triangles, mais le fort parallélisme des GPU et l'implémentation hardware de la rasterisation font que la Shadow Map supporte très bien la hausse de complexité géométrique, car la rasterisation a un coût très faible sur GPU.

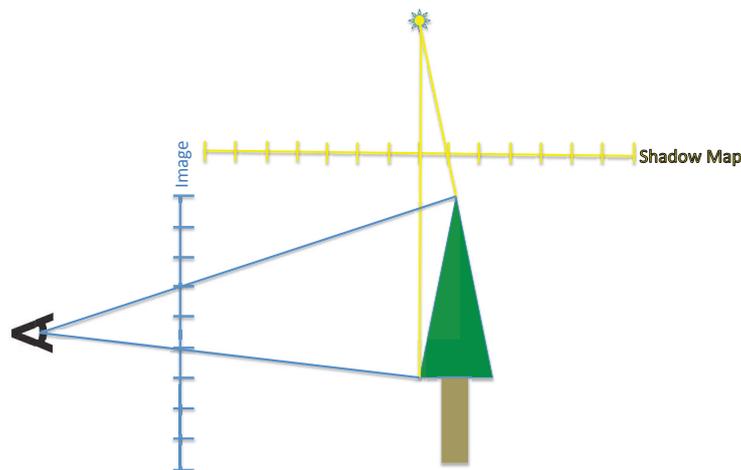


FIGURE 1.22 – Aliassage de projection.

- **Le comportement face à une hausse de la résolution** : à qualité de rendu équivalente, la hausse de résolution implique une hausse de résolution de la Shadow Map et donc une hausse de la consommation mémoire.
- **Le coût en mémoire** : le coût en mémoire est proportionnel à la résolution choisie pour la Shadow Map, résolution qui conditionne la qualité du rendu. À titre d'exemple, les moteurs 3D de référence comme le Unreal Engine utilisent des Shadow Maps d'une résolution de  $16000 \times 16000$  pour un rendu en  $1920 \times 1080$  en haute qualité avec un coût en mémoire de l'ordre du giga-octet, ce qui n'est pas négligeable, les GPU haut de gamme étant doté de 8 go de VRAM. La tendance récente à une hausse de résolution des écrans, avec le 4K et maintenant le 8K, entraîne une augmentation de la taille des Shadow Maps et un coût en mémoire qui dépasse la quantité de mémoire disponible sur les GPU existants à qualité de rendu égal. Il est cependant possible de limiter la consommation mémoire en adoptant un rendu de l'image définitive par portion ou *Tile rendering*, ce qui permet de diminuer la quantité de mémoire allouée simultanément au prix de multiples passes de rendu. En 2016, Scandolo et al.[9] proposent d'utiliser un précalcul de très grandes Shadow Maps de plus de  $1\text{ M} \times 1\text{ M}$  pixels, compressées sous forme de quad-tree pour permettre un rendu de bonne qualité tout en limitant l'explosion du coup mémoire.
- **Le type d'ombrage possible** : la technique des Shadow Maps permet de simuler l'ombre d'une source de type spot ou une lumière directionnelle (comme le soleil). Pour gérer une source directionnelle, il faut répéter le processus 6 fois en rendant 6 Shadow Maps pour obtenir un cube de Shadow Map autour de la source lumineuse.
- **Le paramétrage** : le paramètre principal est celui de la taille de la Shadow Map et des biais utilisés. Le choix de la taille de la Shadow Map est important car c'est le paramètre qui va faire varier la qualité du rendu et la consommation mémoire. A titre d'exemple, les captures 1.23 et 1.24 montrent le résultat de l'ombrage avec une Shadow Map de  $512 \times 512$  et  $2048 \times 2048$ .
- **La stabilité** : c'est un bon point pour les Shadow Map qui s'avèrent être particulièrement stables avec un temps quasiment constant. En effet, la rasterisation qui permet de générer la Shadow Map est faiblement sensible à la complexité géométrique et l'interrogation de la Shadow Map se limite à un accès mémoire par pixel.

Il existe beaucoup d'autres publications scientifiques sur les Shadow Maps et leurs diverses évolutions visant à combler les défauts de la technique originelle. Un état de l'art complet a été proposé par Eisenmman et al en 2011 [10]. Nous exposons à présent les principales techniques permettant d'améliorer

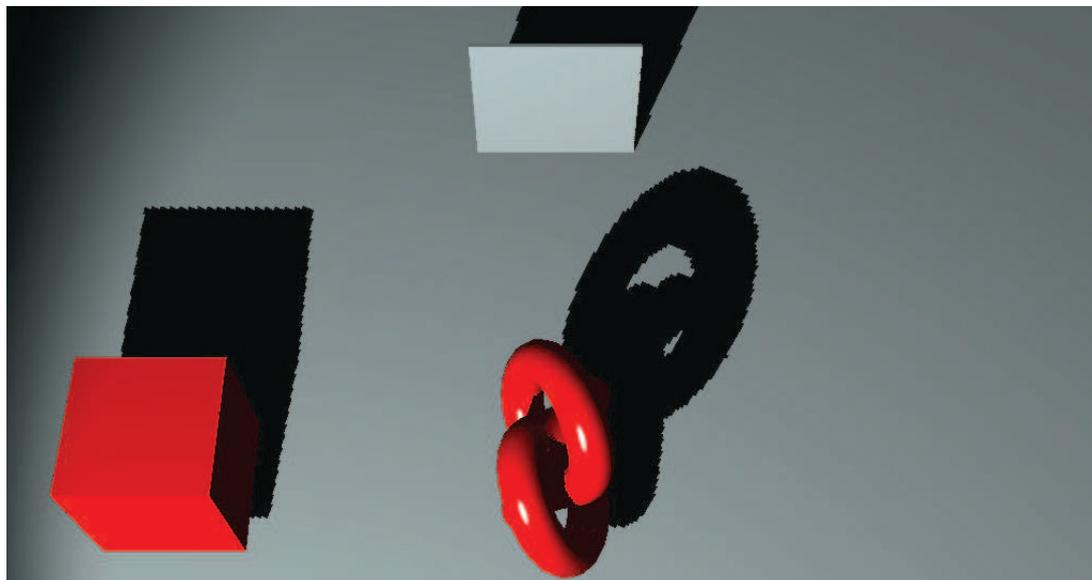


FIGURE 1.23 – Shadow Map 512 x 512.

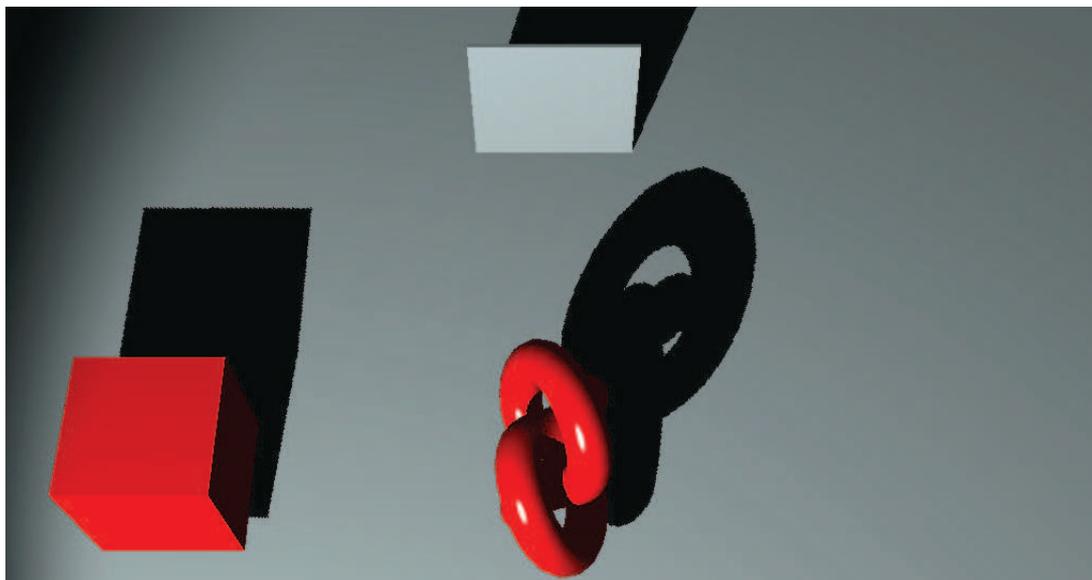


FIGURE 1.24 – Shadow Map 2048 x 2048.

le Shadow Mapping.

#### 1.2.1.4 Les sources de lumières omnidirectionnelles

Stefan Brabec et al. proposent [11] de déformer l'espace lors du calcul de la Shadow Map, afin qu'une Shadow Map couvre un hémisphère. Ainsi avec 2 passes de rasterisation et les 2 Shadow Maps hémisphériques générées, il est possible de gérer une source de lumière omnidirectionnelle. Comme le résultat de cette approche génère des rendus avec des précisions variables suivant les directions, elle a été laissée de côté jusqu'à maintenant. C'est donc le rendu « cubique » qui reste la solution utilisée par l'industrie.

#### 1.2.1.5 Limitation du biais

Plusieurs techniques ont été proposées afin de limiter l'utilisation du biais nécessaire pour éviter les artefacts d'auto-ombrage.

- Jean-Charles Hourcade et al. [12] conservent l'index du triangle qui se projette dans la Shadow Map, au lieu de garder la profondeur du tampon de profondeur. Ainsi il suffit de comparer l'index du triangle se projetant sur un pixel à celui contenu dans cette nouvelle Shadow Map pour savoir si le pixel est éclairé ou non. Malheureusement cette technique produit de nouveaux artefacts à la jointure entre 2 triangles adjacents ainsi que des erreurs lorsque des triangles projetés sont inférieurs à la taille d'un pixel ;
- Stefan Brabec et al. [13] font remarquer que la représentation des profondeurs dans le tampon de profondeur n'est pas linéaire comme sur la figure 1.25 : la solution proposée est de linéariser le calcul des profondeurs pour minimiser le risque d'imprécision.

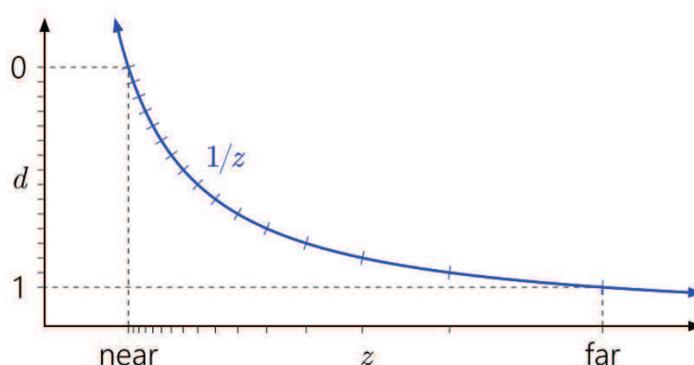


FIGURE 1.25 – Précision du tampon de profondeur.

Ce qui est proche de la caméra bénéficie d'une bonne précision, alors que ce qui est loin en a beaucoup moins. En effet, le GPU stocke les profondeurs en utilisant des valeurs du type  $1/z$ . C'est ici une source potentielle d'erreurs, particulièrement lorsque la source de lumière est loin de la caméra.

- Stefan Brabec et al. [13] proposent enfin d'utiliser un frustum adaptatif (cf. figure 1.26) qui permet d'améliorer la qualité du rendu comme le montrent les captures de la figure 1.27.

#### 1.2.1.6 Perspective Shadow Map

On sait que la résolution d'une Shadow Map est insuffisante pour ce qui est proche de la caméra, et trop dense pour ce qui en est éloigné. L'idée de la technique proposée par Stamminger et al. [14] est

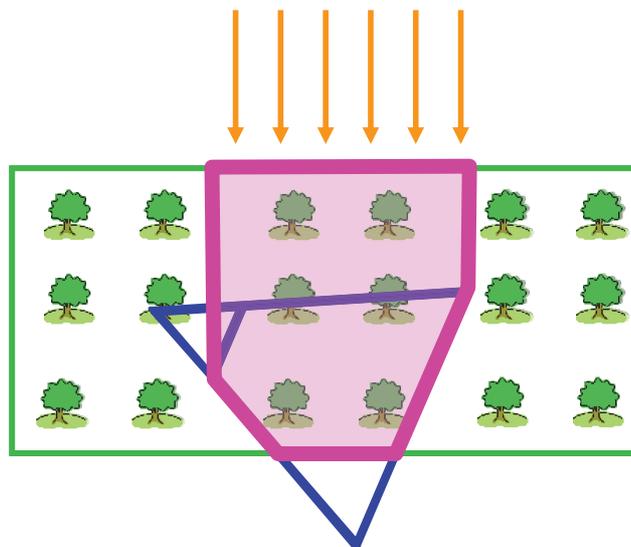


FIGURE 1.26 – Frustum adaptatif.

Source : [7] diapositive 7

Le principe est de maximiser la portion de la Shadow Map utile. Pour obtenir un tel résultat, il faut calculer un frustum de vue pour la caméra virtuelle qui génère la Shadow Map, afin de le restreindre à la géométrie visible depuis la caméra et à celle susceptible de générer de l'ombre dans le frustum caméra.

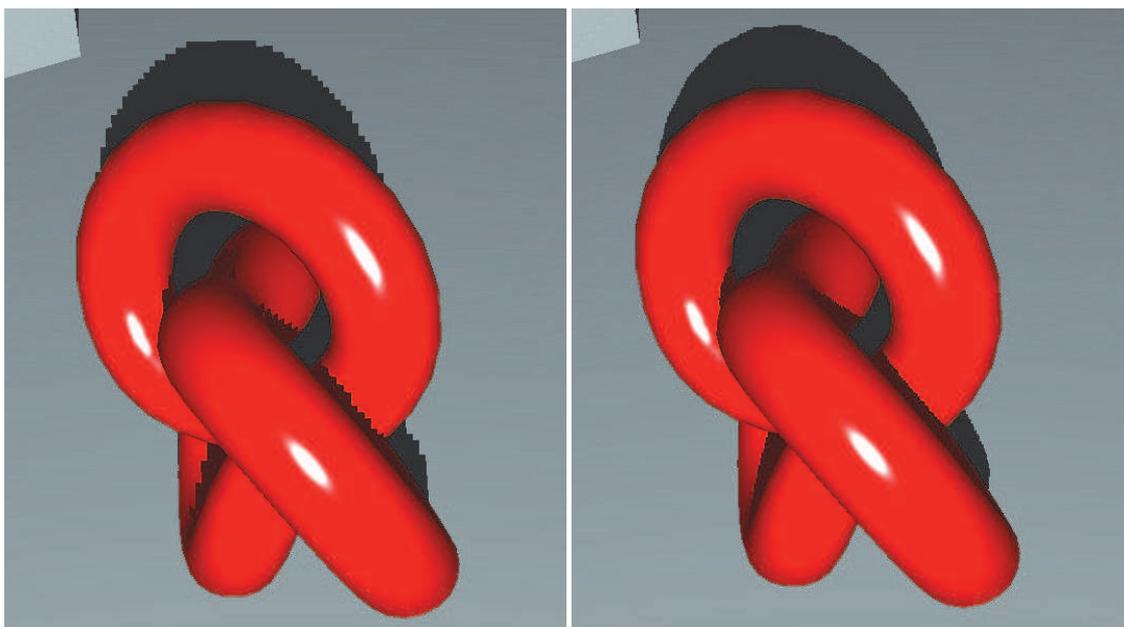


FIGURE 1.27 – Rendu sans et avec frustum adaptatif.

de modifier la matrice de projection utilisée pour le rendu de la Shadow Map, afin d'agrandir la zone où se projette ce qui est proche de la caméra tout en diminuant la zone où se projette ce qui est loin de la caméra. L'amélioration du rendu est très nette sur la figure 1.29 ; en revanche, l'augmentation de la qualité du rendu des ombres des objets proches se paye par une diminution de la qualité des ombres des objets éloignés de la caméra car, la projection perceptuelle a tendance à déformer les éléments les plus éloignés.

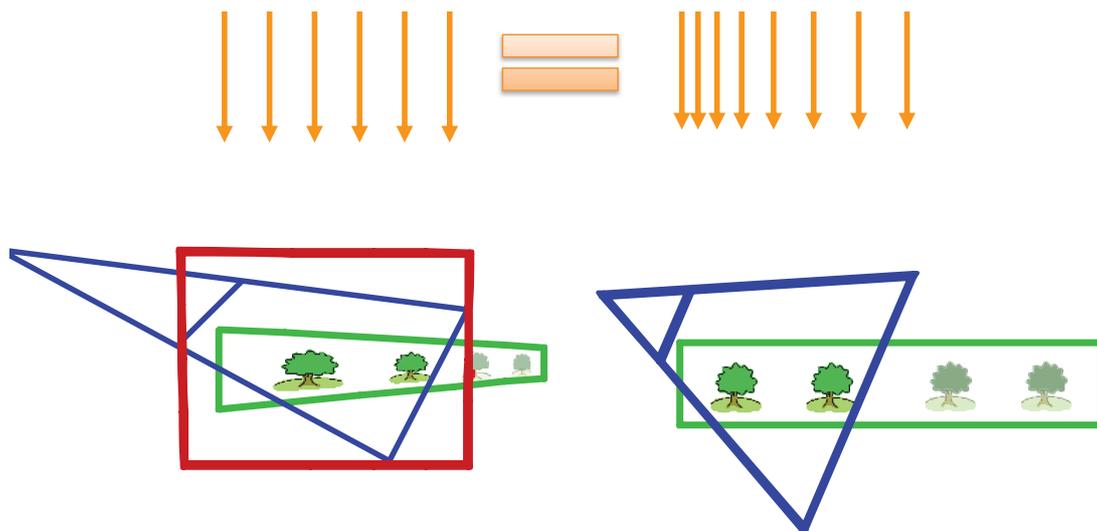


FIGURE 1.28 – Déformation perspective de la Shadow Map.  
Source : [7] diapositive 20



FIGURE 1.29 – Rendu avec une Shadow Map de 512 x 512, à gauche avec une shadow classique et à droite avec une perspective Shadow Map.

Source : [14] page 1

### 1.2.1.7 Cascade de cartes d'ombre ou cascading Shadow Maps

Une autre approche, proposée par Dimitrov et al. [15] pour diminuer la problématique de l'aliasage, consiste à calculer plusieurs Shadow Maps, chacune servant à ombrer une portion du frustum de vue de la caméra comme sur les figures 1.30 et 1.31, et ainsi obtenir une précision adaptée à la portion correspondante du frustum . Cette technique est très utilisée par l'industrie des jeux vidéos. Il faut noter que cette amélioration est compatible avec les améliorations déjà présentées, mais implique encore une complexification de la technique des Shadow Maps et rajoute un nouveau paramètre : combien de Shadow Maps différentes faut-il calculer ? Comment séparer de manière optimale les différentes Shadow Maps ? Même si les GPU actuels sont rapides, il faut à nouveau rajouter  $n$  passes de rendu pour  $n$  cascades et dans le cas d'une lumière omnidirectionnelle, il faudra multiplier par 6 le nombre de passes de rendu nécessaires. Généralement le nombre de cascades utilisées en pratique est de 4 ; donc dans ce cas, il faut 24 passes de rendu pour les 24 Shadow Maps à générer. Ajoutons que dans le cas d'une lumière omnidirectionnelle et selon sa position dans la scène, les tailles des différentes Shadow Maps peuvent être variables en fonction de la face du cube de Shadow Maps à générer.

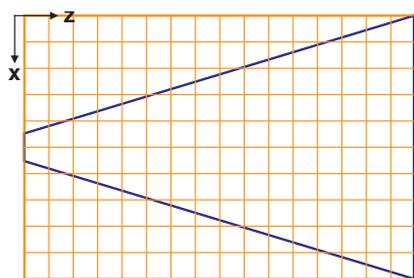


FIGURE 1.30 – Frustum de vue de la caméra.

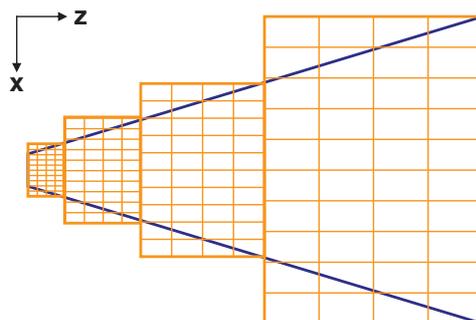


FIGURE 1.31 – Découpe du frustum de vue de la caméra

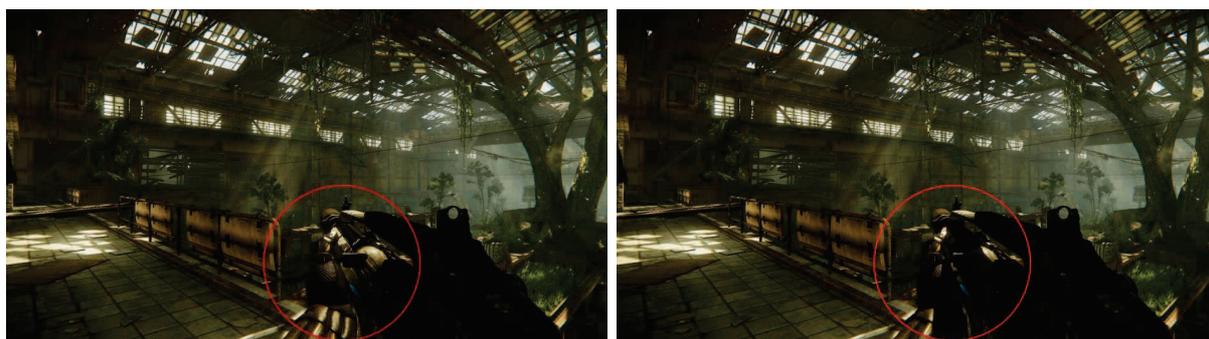


FIGURE 1.32 – Rendu sans (à gauche) et avec (à droite) une Shadow Map spécifique pour la main et l'arme au premier plan.

Source : [16] diapositive 20

### 1.2.1.8 Shadow Map par objet

Le moteur Cryengine de Crytek [16] utilise une pratique qui consiste à utiliser une Shadow Map pour un seul objet de la scène 3D, typiquement pour les objets très détaillés et proches de la caméra, comme par exemple l'arme visible au premier plan. Cette approche permet un auto-ombrage meilleur que l'application d'une Shadow Map globale. En effet, une Shadow Map par objet se limite à la zone de projection de cet objet, ce qui permet une densification des informations de profondeur de la Shadow Map sur l'objet lui-même. Sur la figure 1.32, on peut aisément constater la différence entre le rendu avec ou sans Shadow Map par objet : le rendu classique des Shadow Maps ne permet pas d'avoir un auto-ombrage correct, notamment sur l'ombre de l'arme sur le bras du personnage.

### 1.2.1.9 Shadow Mapping : conclusion

La Shadow Map part d'un principe simple, mais demande en pratique de cumuler les diverses améliorations présentées pour obtenir un résultat sans artefact. Le contrôle de ce cumul nécessite de gérer un grand nombre de paramètres pour optimiser les différents frustums utilisés, ainsi que les déformations de perspective qui rendent la méthode plus complexe à implémenter, et ne font que repousser les limites d'apparition d'artefacts tout en introduisant de nouveaux. En pratique, la solution choisie par l'industrie consiste à utiliser des cascades de perspective Shadow Maps avec de très grandes résolutions pour obtenir des résultats sans artefact perceptible et plausible. Pour un résultat voisin de l'exactitude, il faut pousser la méthode plus loin avec les Shadow Maps par objet afin d'obtenir un auto-ombrage correct des objets proches. Une des qualités indéniables des approches par Shadow Map est la

stabilité dont le coût en temps est quasiment constant. La hausse de la définition d'image entraînera une hausse équivalente de celles des Shadow Maps ; même si la rasterisation est rapide sur les GPU actuels, la multiplication de nombre de passe de rendu dans des textures sans cesse croissante affectera obligatoirement les performances en augmentant la consommation de mémoire et la quantité d'accès mémoire pour le rendu d'une image. C'est dans cette optique que Scandolo et al. [9] et Sintorn et al. [17] orientent leurs recherches vers des structures pré-calculées permettant de compresser les Shadow Maps.

## 1.2.2 Les volumes d'ombre ou Shadow Volumes



FIGURE 1.33 – Ombre par Shadow Volume dans Doom 3.

C'est une technique proposée par Crow en 1977 [5], et qui permet d'obtenir un résultat exact pour chaque pixel. Après une explication sur son principe, il faudra montrer pourquoi, malgré ses diverses tentatives d'améliorations, cette méthode est délaissée par l'industrie, sa seule utilisation notable étant dans le jeu Doom 3 d'IDsoftware (cf. capture 1.33).

### 1.2.2.1 Le principe

C'est une approche géométrique qui va consister à générer par triangle le volume qui correspond à toute la zone cachée de la lumière. Ce volume d'ombre ou Shadow Volume est une pyramide tronquée infinie, qui est délimitée par le triangle lui-même et par les 3 plans que l'on peut construire à partir de la source de lumière et chaque arête du triangle. Une fois que tous les volumes d'ombre d'une scène ont été construits, il reste à déterminer si un pixel se trouve ou non dans un ou plusieurs volumes d'ombre. Pour y parvenir, on compte le nombre de plans de volume d'ombre traversés pour aller de la caméra au pixel, comme sur la figure 1.34 : on incrémente un compteur lorsque l'on rentre dans un volume d'ombre, et on le décrémente quand on en sort. Si à la fin de ce décompte la valeur est 0, alors le pixel est illuminé ; dans tous les autres cas, le pixel est dans un ou plusieurs volumes d'ombre.

### 1.2.2.2 Le Z-pass

Il faudra attendre 1991 pour que Heidmann et al. [19] proposent une première implémentation des volumes d'ombre sur GPU à la suite de l'apparition de compteurs par pixel ou *stencil buffer* sur les GPU,

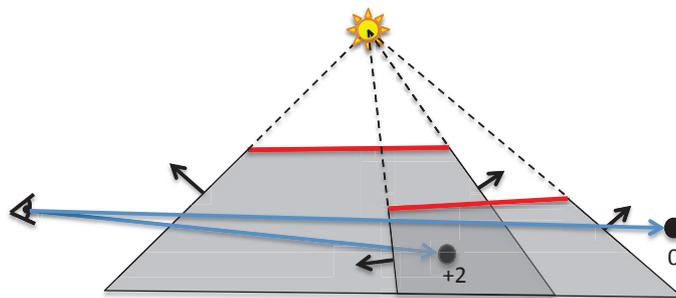


FIGURE 1.34 – Volumes d'ombre générés par 2 triangles.  
Source : [18] diapositive 20

comme sur la figure 1.35. Il faut passer par plusieurs étapes :

- un rendu de la scène en gardant les valeurs du tampon de profondeur (en haut à gauche) ;
- un rendu des 3 plans infinis de la pyramide, en incrémentant ou décrémentant les valeurs du compteur par pixel selon leur orientation par rapport à la caméra, en conservant le test de profondeur de la première étape de rendu (en bas) ;
- le rendu final de la scène (en haut à droite) en utilisant les valeurs du compteur par pixel (en bas à droite).

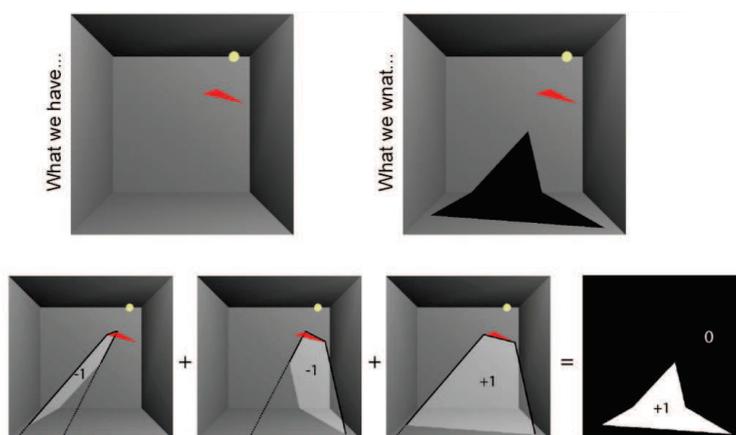


FIGURE 1.35 – Les différentes étapes de l'application des shadows volumes.  
Source : [18] diapositive 27

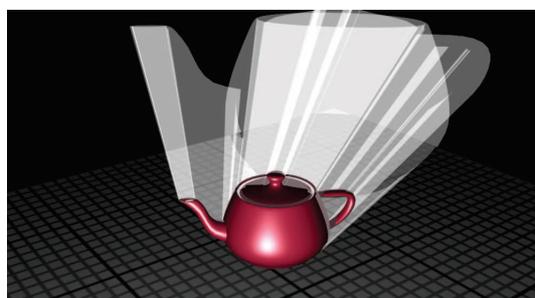


FIGURE 1.36 – De grands shadow quads.

Tout comme les Shadow Maps, les Shadow Volumes reposent sur la rasterisation. La phase de

rastérisation est rapide lorsque l'on projette de petits triangles ; mais de grands triangles ou quads, utilisés pour représenter les plans infinis des volumes d'ombre, comme sur la figure 1.36, impliquent beaucoup plus d'accès mémoires pour la comparaison de profondeur et la mise à jour du compteur par pixel. C'est le principal problème des techniques basées Shadow Volumes : elles sont très fortement dépendantes de la complexité géométrique, qui elle sature la bande passante mémoire du GPU : cette bande passante représentant la quantité maximale de pixels ou texels qu'une carte graphique est capable d'écrire dans la mémoire video par seconde.

### 1.2.2.3 Artefact

La technique de heidemann [19] pose problème lorsque la caméra se trouve elle-même dans un ou plusieurs volumes d'ombres : le masque d'ombrage final donne alors des ombres fausses qui peuvent être plus ou moins inversées, comme le montre la figure 1.37 . Le problème est que les compteurs devraient être initialisés au nombre de volumes d'ombre contenant la caméra. Pour y remédier, Eisemann et al [10] proposent une solution potentiellement inexacte, mais qui est efficace en pratique. Il s'agit de faire une deuxième passe de rendu de Shadow Volumes un peu particulière, puisqu'il s'agit d'un rendu de 1 pixel à la position de la caméra (il faudra pour la lecture du compteur par pixel tenir compte de la valeur comptée pour la caméra). Sur les GPU au du début des années 90, il y avait aussi une limitation importante liée



FIGURE 1.37 – Rendu incorrect.



FIGURE 1.38 – Rendu correct.

au compteur par pixel qui utilisait seulement 8 bit soit 256 valeurs : lorsque de trop nombreux volumes d'ombre se projetaient sur un même pixel, on saturait la capacité du compteur par pixel qui indiquait alors de fausses valeurs. Aujourd'hui les GPU utilisent un compteur de 32 bits par pixel qui élimine tout risque de dépassement.

### 1.2.2.4 Le Z-fail

Le Z-fail est la technique utilisée dans Doom 3 ; son objectif consiste à supprimer les artefacts liés à la présence ou non de la caméra dans un ou des volumes d'ombre. Quasiment à la même époque, Bilodeau [20] et John Carmack [21] proposent d'inverser le sens de traversée des volumes d'ombre (cf figure cf. figure ??).

La correction de l'artefact de caméra a ici un prix non négligeable : le test de profondeur étant là aussi inversé, c'est la géométrie masquée qui génère potentiellement plus de Shadow Volumes qu'avec le Z-pass. En pratique, le Z-fail est plus lent que le Z-pass : c'est une des raisons du mécontentement des utilisateurs

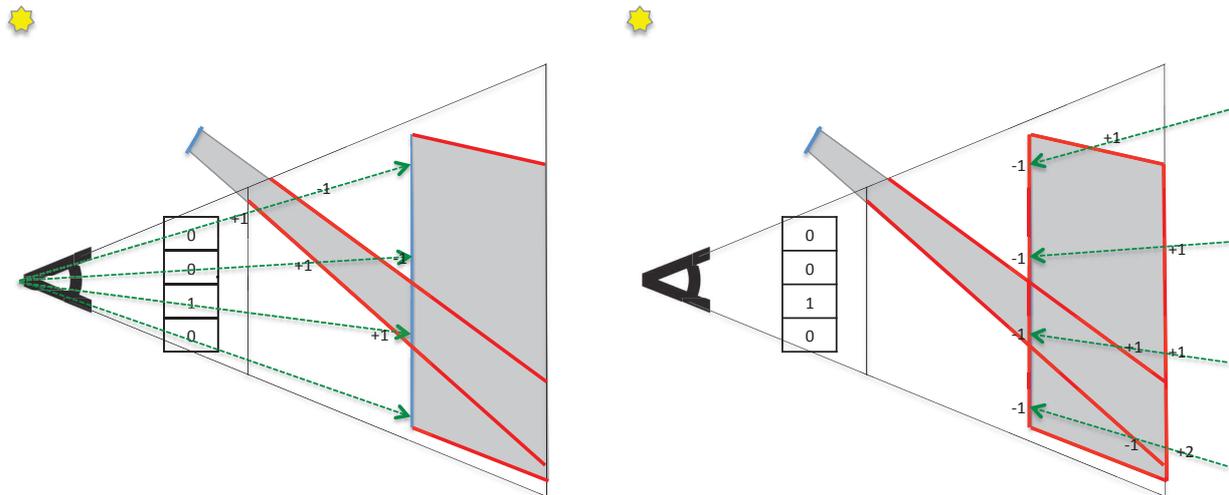


FIGURE 1.39 – Du Z-fail au Z-pass.

A gauche, on peut voir le remplissage du tampon de profondeur pour la technique du Z-fail en comptant les volumes d'ombre entre la caméra et un pixel ; à droite, le comptage inversé du Z-pass implique une modification du rendu des volumes d'ombre : il faut maintenant fermer les volumes d'ombre afin de pouvoir commencer le décompte des volumes d'ombre depuis un point qui ne se trouve dans aucun volume d'ombre.

de Doom 3 qui ne pouvaient pas dépasser les 60 images par seconde, alors même que l'ensemble du jeu était pensé pour limiter à 4 le nombre d'ennemis présents simultanément afin de réduire le temps de rendu des Shadow Volumes.

### 1.2.2.5 Volumes d'ombre par extraction des silhouettes

Dans cette partie et la suivante, nous ferons référence au figures de du cours Sigraph [18]. Au lieu de rendre un volume d'ombre par triangle, Tomas Akenine-Möller et al. [22] proposent de se focaliser sur la silhouette de l'ombre (cf. figure 1.40) et de l'extruder. Cette technique permet de diminuer drastiquement le nombre de triangles à rendre, lors du calcul du masque d'ombre avec le compteur par pixel. Ainsi la complexité se limite à celle de la silhouette de l'ombre, indépendamment du nombre de triangles.

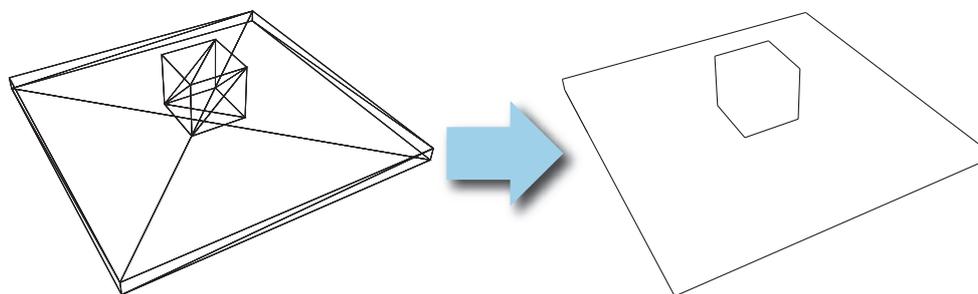


FIGURE 1.40 – Simplification de la géométrie grâce à la silhouette.

Le calcul de la silhouette permet de gagner en performance, mais à 2 conditions :

- connaître les informations d'adjacences des triangles, afin de pouvoir déterminer le moment où une arête devient silhouette, lorsqu'un seul des 2 triangles partageant cette arête fait face à la lumière ;

cette limitation implique que la topologie des objets de la scène ne change ;

- respecter les impératifs géométriques qui évitent les erreurs de calcul d'ombre : les modèles géométriques doivent être fermés. On peut voir une erreur de calcul du compteur de volumes d'ombre sur la figure 1.42 lorsque l'objet n'est pas fermé, contrairement à celui de la figure 1.41.

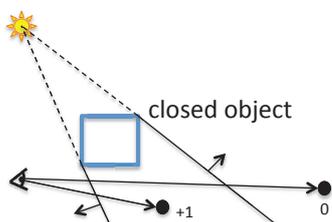


FIGURE 1.41 – Remplissage du compteur par pixel avec un objet fermé.

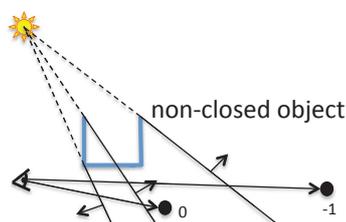


FIGURE 1.42 – Remplissage du compteur par pixel avec un objet non fermé.

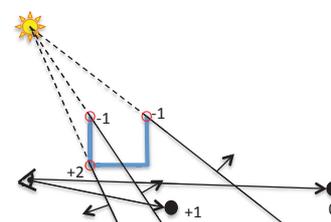


FIGURE 1.43 – Correction du remplissage du compteur par pixel avec un objet non fermé.

Heureusement, Bergeron et al. [23] apportent une légère modification de la méthode de comptage, qui permet d'outrepasser cette limitation comme sur la figure 1.43 :

- les arêtes silhouettes appartenant à 2 triangles incrémenteront ou décrémenteront le compteur de 2 ;
- celles n'appartenant qu'à un seul triangle incrémenteront ou décrémenteront le compteur de 1.

### 1.2.2.6 Limitation de la saturation de la bande passante mémoire

Pour améliorer les performances des volumes d'ombre, il y a 2 axes :

- limiter le nombre de volumes d'ombre à rendre ;
- limiter autant que possible la zone de projection des volumes d'ombre, afin de soulager la bande passante mémoire du GPU.

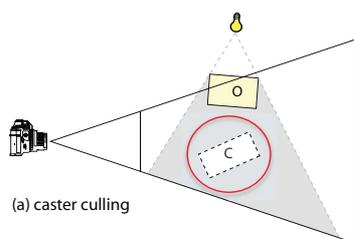


FIGURE 1.44 – Élimination des objets cachés du point de vue de la lumière.

Source : [18] diapositive 40

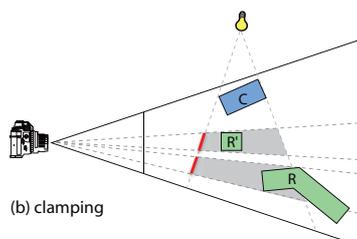


FIGURE 1.45 – Limitation de la zone de rendu des volumes d'ombre.

Source : [18] diapositive 41

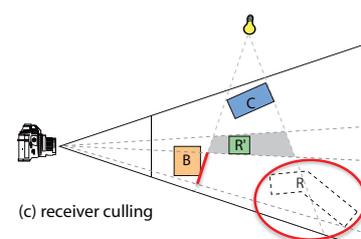


FIGURE 1.46 – Élimination des objets cachés du point de vue de la caméra.

Source : [18] diapositive 42

Dans cette optique, Alexander Keller [24] et Stich [25] proposent de ne pas rendre les volumes d'ombre d'un objet qui se retrouverait déjà caché par un autre, comme sur la figure 1.44 :

- une première passe consiste à rendre une Shadow Map depuis le point de vue de la lumière ;
- une seconde passe permet un rendu des objets de la scène, toujours du point de vue de la lumière mais en utilisant un test de profondeur et un compteur par pixel pour marquer les zones où le test de profondeur a échoué, ce qui donne les zones totalement ombrées ;

- les boîtes englobantes de chaque objet de la scène vont elles aussi être rendues : si toute une boîte englobante se trouve dans l'ombre, alors il ne sera pas nécessaire de générer les volumes d'ombre de l'objet associé.

Une autre amélioration, due à Lloyd [26] et Elmar Eisenmann [27], consiste à limiter autant que possible les zones où l'on va réellement rendre les volumes d'ombre en partant du constat qu'il est inutile d'en faire le rendu dans les zones où il n'y a pas d'objet susceptible de recevoir l'ombre comme sur la figure 1.45. De même, il est inutile de rendre des volumes d'ombre qui cacheraient de la lumière un objet déjà invisible depuis la caméra (figure 1.46).

### 1.2.2.7 Shadow Volumes : conclusion

En résumé, les évolutions de la technique des volumes d'ombre étudiées ici améliorent le comportement général, mais peuvent toutes être mises à mal :

- l'utilisation de la silhouette perd toute son efficacité dans le cas d'une silhouette complexe, comme le feuillage d'un arbre où il faudra presque rendre un volume d'ombre par triangle ;
- la limitation de la zone de projection des volumes d'ombre et la suppression des objets cachés du point de vue de la lumière et de la caméra impliquent de nombreuses passes de rendu supplémentaires. Dans le pire des cas, si tous les objets sont visibles et génèrent de grandes ombres, ces optimisations deviennent inutiles et peuvent même dans le pire des cas devenir plus lentes que la simple utilisation de la silhouette. En reprenant les critères d'analyse utilisés précédemment pour les Shadow Maps, on obtient les conclusions suivantes :
- **l'exactitude** : exacte par pixel ;
- **le comportement face à la complexité géométrique** : risque de saturation de la bande passante mémoire des GPU ;
- **le comportement face à une hausse de la résolution** : amplification des problèmes de bande passante mémoire ;
- **le coût en mémoire** : négligeable car la méthode n'a pas besoin de données persistantes ; dans le cas des méthodes avec extraction de silhouette, il faut stocker les informations d'adjacence ;
- **le type d'ombrage possible** : omnidirectionnel où directionnel ;
- **la paramétrage** : faible ;
- **la stabilité** : très instable ; selon le point de vue la bande passante mémoire sature plus ou moins vite (cf. figure 1.47).

### 1.2.3 Les techniques hybrides

Elles se répartissent en 2 familles :

- celles qui s'attaquent à la nature même de l'aliassage des Shadow Maps : l'échantillonnage sur une grille régulière ne permet pas d'avoir toutes les informations de profondeur nécessaires, et ne permet donc pas un ombrage correct ; l'idée générale est de générer un échantillon de profondeur correspondant exactement à chaque pixel de l'image finale ;
- celles qui cherchent à diminuer le coût de la rasterisation des volumes d'ombre à l'aide de structures accélératrices.

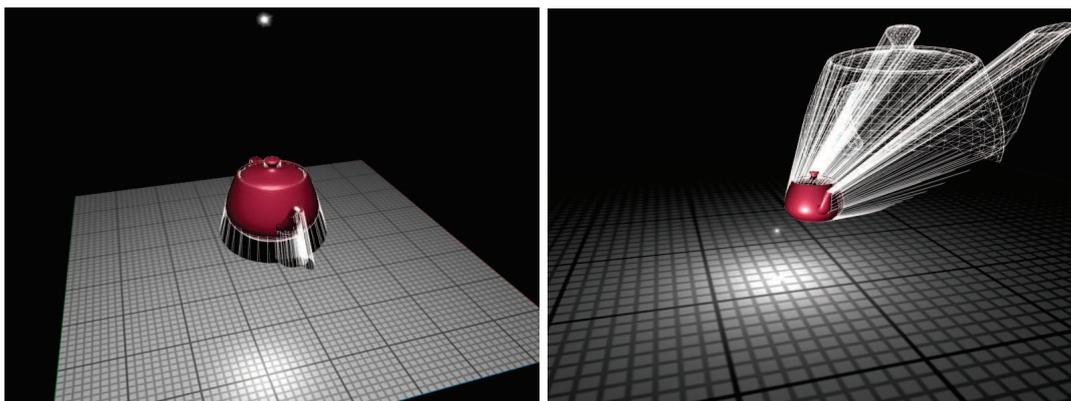


FIGURE 1.47 – Pour un même objet, un point de vue favorable (à gauche) et un point de vue très défavorable (à droite) pour les méthodes basées Shadow Volumes.

Sur l'image de gauche, les volumes d'ombre se projettent sous formes de tous petits volumes d'ombre, et beaucoup ne seront pas rendus car cachés par l'objet lui-même. Au contraire sur l'image de droite, presque tous les volumes d'ombre de la silhouette doivent être rendus sur une grande surface avec une ombre qui s'étire : les volumes d'ombre sont donc longs et étirés, ce qui entraîne de très nombreuses lectures et écritures dans le compteur par pixel.

### 1.2.3.1 Shadow Map irrégulière

En partant de l'observation que la densité des échantillons de profondeur nécessaires pour un ombrage sans aliassage ne rentre pas dans une grille régulière (cf. figure 1.48), Aila et al. [28] proposent de

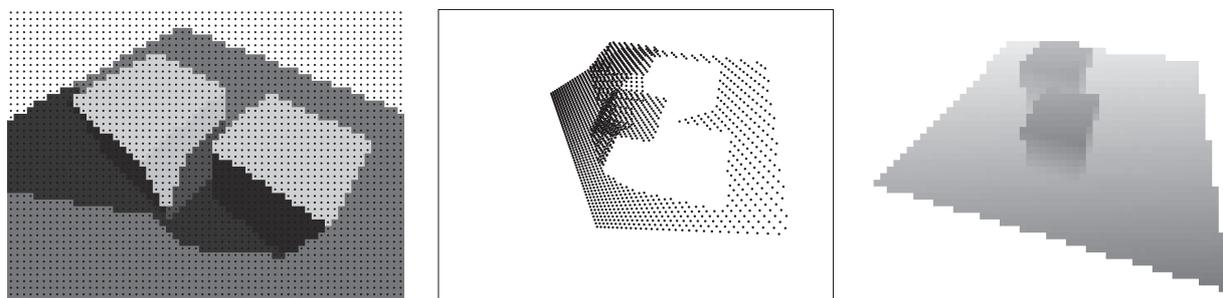


FIGURE 1.48 – Projection des pixel sur une Shadow Map

Une scène du point de vue de la camera à gauche, la reprojection des pixels camera du point de vue de la lumière et la répartition des informations dans une Shadow Map classique. Source : [28] page 3

générer tous les échantillons nécessaires dans une Shadow Map sur une grille irrégulière ou un tampon de profondeur irrégulier. Cette génération consiste à :

- faire un rendu depuis la caméra ;
- exprimer la position du centre de chaque pixel du premier rendu dans l'espace lumière ;
- les échantillons idéalement placés dans la grille irrégulière sont stockés dans un arbre BSP 2D ;
- la géométrie est rendu du point de vue de la lumière pour générer une liste de triangles par échantillon ;
- on évalue la visibilité d'un échantillon en testant l'intersection avec les triangles de sa liste.

Le résultat est un ombrage sans aliassage, mais une bonne partie du rendu est ici prise en charge par le CPU : les performances de l'époque rendaient la méthode inutilisable en temps réel, et tout juste acceptable pour du temps interactif.

Même si les performances ne sont pas au rendez-vous, cette approche est la première à permettre un résultat exact sur la base des Shadow Maps.

### 1.2.3.2 Volume d'ombre par triangle

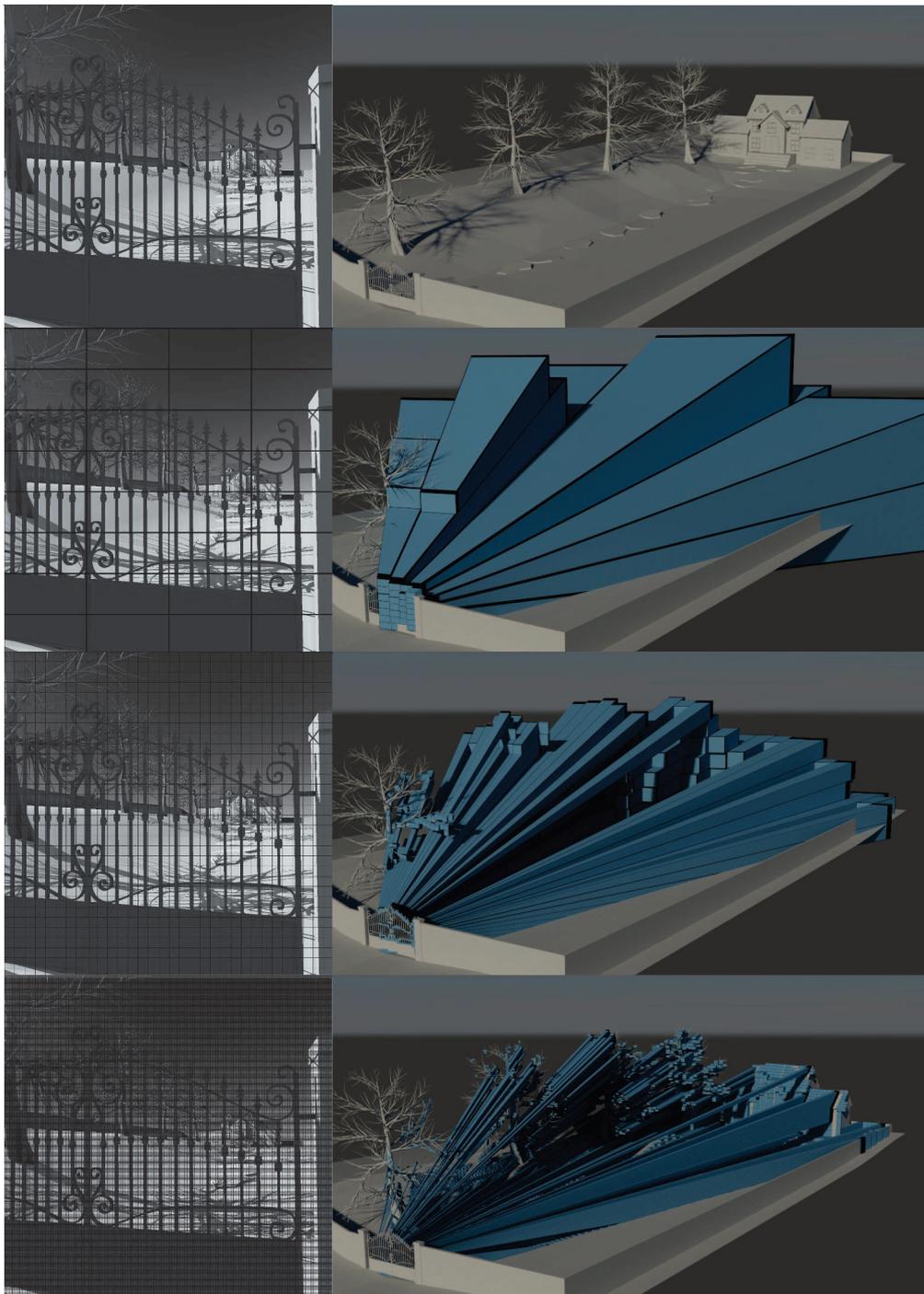


FIGURE 1.49 – Transformation des pixels sous forme de hiérarchie de frustums.

Source : [29] diapositives 7 à 10

Erik Sintorn et al. [30] s'attaquent au problème principal des techniques basées Shadow Volumes, la

saturation de la bande passante mémoire des GPU. Afin de dépasser cette limitation, cette technique va rendre les volumes d'ombre dans une structure accélératrice construite sur les points image depuis le point de vue de la caméra. Les pixels sont regroupés dans des frustums englobants ; ces frustums vont eux-même être regroupés hiérarchiquement sur plusieurs niveaux, comme sur la figure 1.49. Ensuite chaque triangle, sous forme du Shadow Volume correspondant, va traverser cette structure accélératrice pour ombrer ou non les différents pixels. Les différentes étapes de la traversée de la structure sont visibles sur la figure 1.50. Cette méthode a l'avantage de ne pas nécessiter d'information d'adjacence.

En 2014, une évolution de cette méthode est proposée par Erik Sintorn et al. [29]. Elle se base sur la

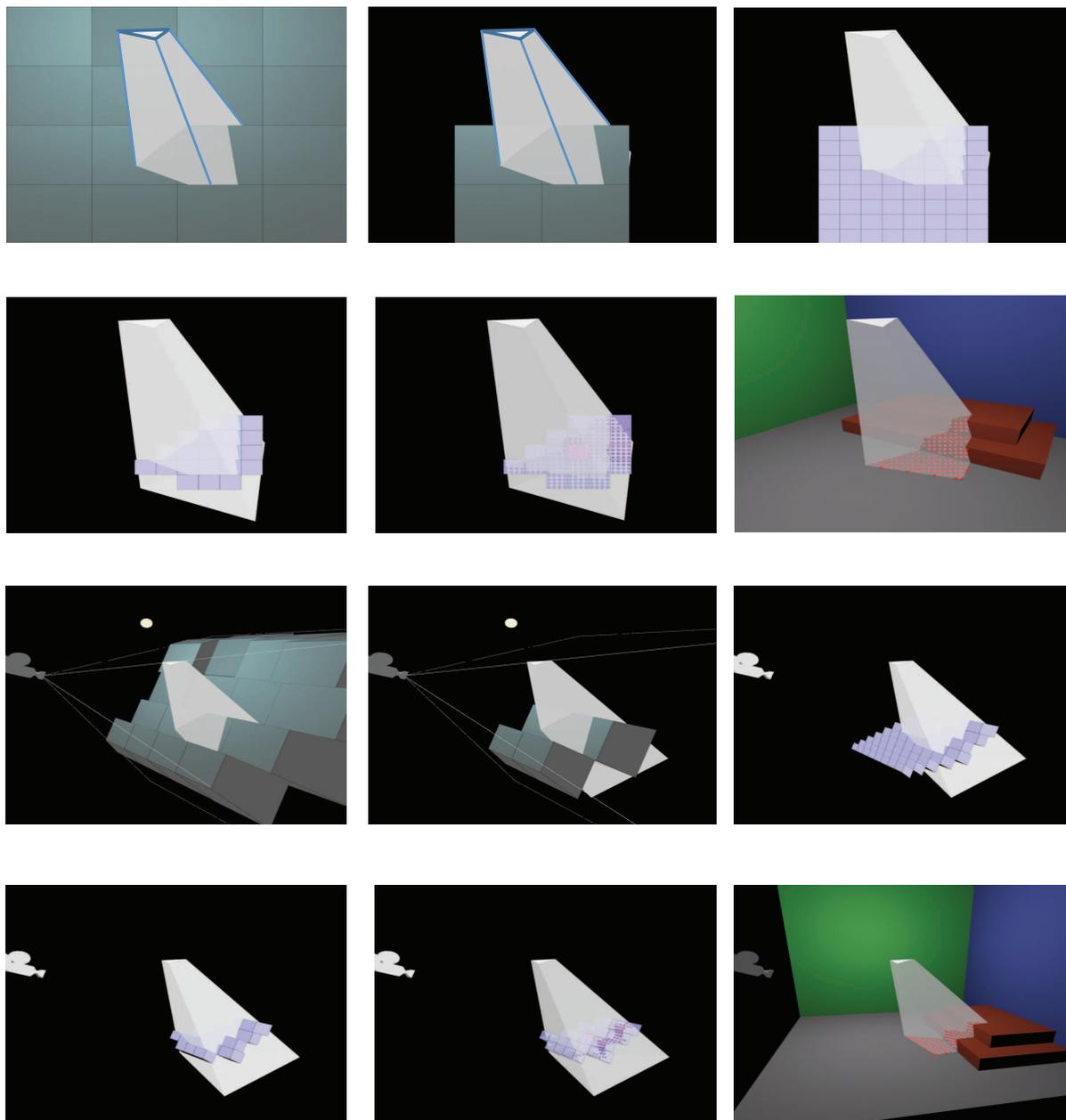


FIGURE 1.50 – Traversée par un Shadow Volume de la hiérarchie de frustums.

Les 6 premières images montrent la traversée du point de vue de la caméra et les 6 suivantes montrent la même traversée d'un point de vue différent. Source : [18] diapositives 46 à 50

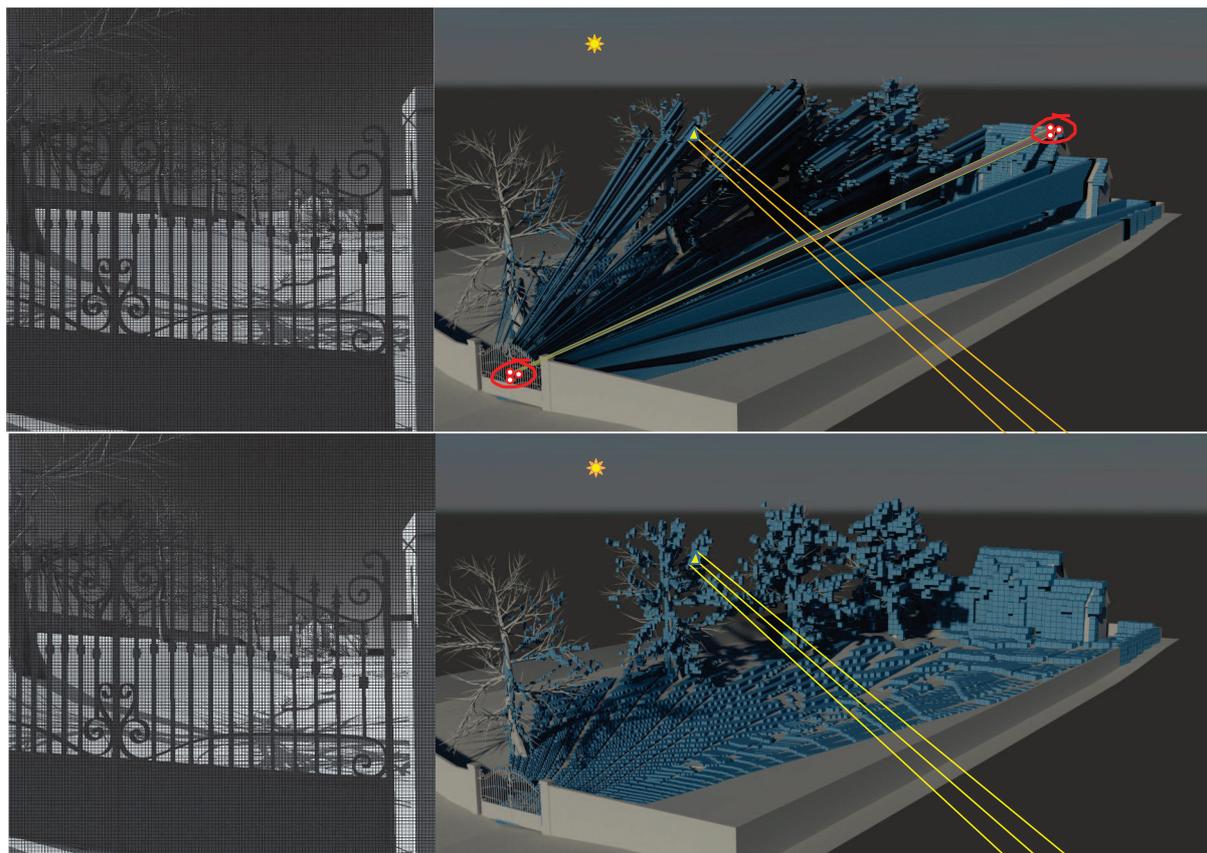


FIGURE 1.51 – Changement des Frustrums

En haut, de trop grands frustums menant à des tests inutiles ; en bas, la nouvelle découpe des frustums.  
Source : [29] diapositives 11 à 12

remarque que dans la première implémentation, les frustums de la hiérarchie peuvent être très profonds s'il y a une forte discontinuité de profondeur entre 2 pixels voisins. Ainsi sur la figure 1.51 en haut, on peut voir un long frustum qui devra être testé par rapport au volume d'ombre du triangle jaune, alors même que ce frustum concerne des points qui n'ont aucune chance de se trouver dans l'ombre de ce triangle. Partant de ce constat, les frustums vont aussi être découpés en profondeur pour devenir plus petits (cf. figure 1.51 en bas), ce qui réduira d'autant les tests de volumes d'ombre inutiles. Cette méthode permet également de gérer des ombres avec des triangles semi-transparentes ou texturés avec un alpha de transparence.

Bilan :

- **l'exactitude** : exacte par pixel ;
- **Le comportement face à la complexité géométrique** : comportement linéaire, chaque triangle doit traverser la structure accélératrice ;
- **Le comportement face à une hausse de la résolution** : comportement logarithmique dû à la structure accélératrice ;
- **Le coût en mémoire** : proportionnel à la résolution de rendu et à la profondeur de la hiérarchie ;
- **Le type d'ombrage possible** : directionnel ou spot et omnidirectionnel ;
- **Le paramétrage** : aucun ;
- **La stabilité** : le temps de rendu peut varier du simple au triple selon les points de vue.

### 1.2.3.3 Rastérisation irrégulière et lancer de rayon

Chris Wyman et al. [31] reprennent le concept de tampon de profondeur irrégulier pour proposer une implémentation sur les GPU modernes afin d'obtenir une ombre exacte. La méthode est une combinaison des approches de tampon de profondeur irrégulier et lancer de rayon. Elle consiste en quelques étapes :

- à partir de l'image caméra, on plonge les échantillons à ombrer dans l'espace lumière ;
- ces échantillons sont placés dans une grille régulière sous forme de liste chaînée ;
- dans cette grille la géométrie est rendue par rastérisation conservatrice, qui permet pour chaque texel d'obtenir la liste des triangles qui s'y projettent sans risque d'en manquer ;
- par force brute, les listes de triangles sont testés les uns après les autres par rapport aux liste d'échantillons pour parvenir au le résultat final, via des tests rayons/triangles ;

En prenant les critères déjà énoncés, on peut établir le bilan suivant :

- **l'exactitude** : garantie ;
- **le coût en mémoire** : imprévisible et variable selon le point de vue ;
- **le type d'ombrage possible** : directionnel ou spot ; comme pour les Shadow Map, il faudra multiplier le rendu par 6 dans le cas d'une lumière omnidirectionnelle avec un surcoût ;
- **le paramétrage** : pour obtenir de bons résultats, la méthode utilise diverses techniques de cascade et de Shadow Mapping, dont le paramétrage conditionne fortement les performances ; la taille de la Shadow Map est importante : si elle est trop petite, la taille des listes de triangles à tester augmente ; si elle est trop grande, des calculs inutiles sont générés.
- **La stabilité** : d'après l'auteur lui-même, la méthode est relativement stable malgré des variations pouvant aller du simple au double. Cette stabilité dépend beaucoup des valeurs des différents paramètres, qui doivent être choisis de manière empirique.

### 1.2.3.4 Arbre BSP de volume d'ombre ou SVBSP

Au début des années 90, Chin et Freiner[32] présentent un technique différente des approches de type volumes d'ombre, qui cherchent à résoudre les problèmes de visibilité depuis le point de vue caméra. Ici, la problématique est traitée depuis le point de vue de la lumière. Partant du principe qu'un volume d'ombre est un polyèdre ouvert délimité par des plans, un arbre BSP (Binary Space Partition, ou Partition binaire de l'espace ) est construit à partir de ces plans. Cette construction utilise des opérations CSG (Constructive solid geometry) proposées par Naylor et al.[33].

Une fois la structure BSP de volumes d'ombre ou SVBSP construite comme sur la figure 1.52, il suffit de localiser les points de l'image à rendre dans l'arbre BSP comme sur la figure 1.54 : si un point arrive dans une feuille vide positive de l'arbre, alors le point est visible depuis la lumière ; dans le cas d'une feuille négative, le point se trouve dans la pyramide d'un volume d'ombre et un dernier test par rapport au plan du triangle permet de déterminer s'il se situe devant le triangle ou dans le volume d'ombre.

Cette méthode offre théoriquement une ombre juste. Elle souffre cependant, dans sa mise en pratique, de nombreuses imprécisions numériques au rendu final. En effet, la construction du SVBSP implique, lors des opérations de CSG, de découper des polyèdres à de multiples reprises. Ces opérations de découpe peuvent aboutir à des polyèdres dégénérés, comme sur la figure 1.53, et elles sont sujettes à des erreurs numériques. Elles posent aussi un problème de consommation mémoire, car une seule surface peut en générer plusieurs lors de la découpe, multipliant ainsi le nombre d'éléments à stocker. La mémoire nécessaire pour stocker un SVBSP n'est donc pas prédictible, et elle varie selon les points de vue. Dans le pire des cas, la consommation mémoire explose avec une complexité en  $O(n^2)$ .

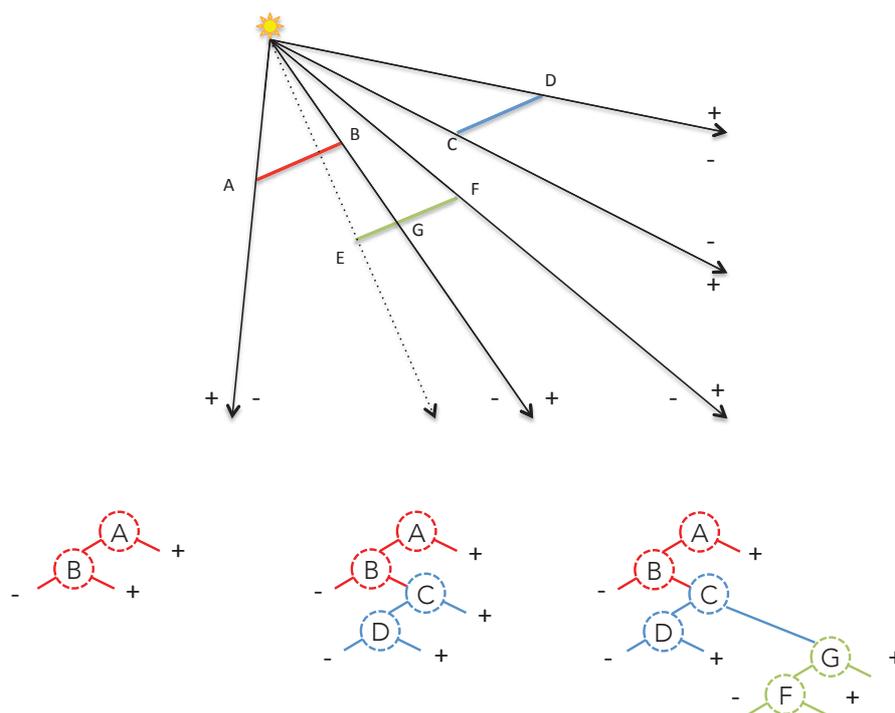


FIGURE 1.52 – Un arbre de volumes d'ombre en 2D et sa construction.

Cet exemple en 2 dimensions comprend 3 segments qui génèrent 3 volumes d'ombre. Chaque volume d'ombre est délimité par un segment et par les 2 droites passant par la lumière et chaque extrémité du segment. Chacune de ces droites est orientée positivement vers l'extérieur du volume d'ombre. La construction de l'arbre SVBSP correspondant se déroule comme suit :

- on insère dans un arbre vide les 2 droites (A) et (B) qui définissent la pyramide du volume d'ombre du segment AB ;
- on place logiquement le segment CD du côté positif de la droite (B) ;
- pour le segment EF, la situation est différente : la méthode implique de fusionner les volumes d'ombres ; la partie EG du segment EF qui se trouve dans le volume d'ombre de AB est supprimée : ce n'est pas le segment EF qui va être inséré dans l'arbre, mais seulement la partie GF.

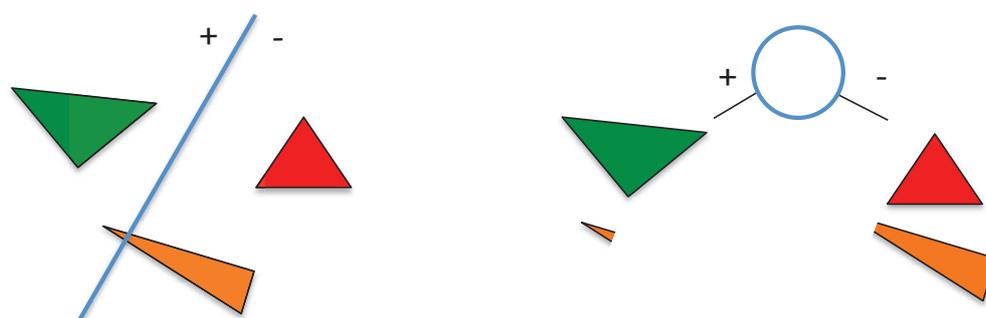


FIGURE 1.53 – Découpe et duplication des informations géométriques dans un arbre BSP. Le triangle orange doit être réparti dans les 2 fils du noeuds BSP. Non seulement il y a une duplication des données dans l'arbre, mais en plus le fils négatif est maintenant un quadrilatère et non un triangle.

Il est intéressant de noter que la construction des SVBSP est indépendantes du point de vue caméra ; ainsi dans une scène statique il suffit de construire une fois le SVBSP pour pouvoir ensuite se déplacer

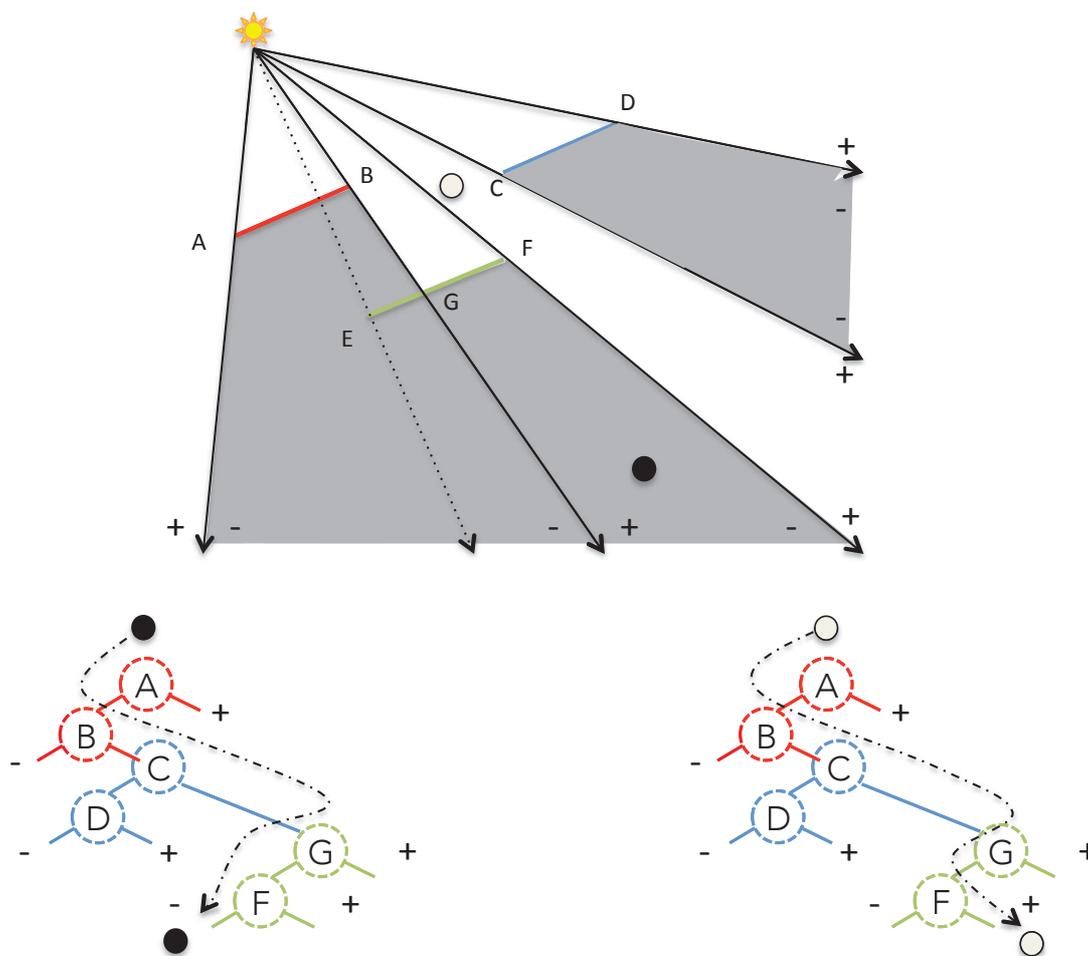


FIGURE 1.54 – Parcours d'un arbre BSP de volumes d'ombre.

On peut voir le parcours de l'arbre déjà présenté par un point hors de l'ombre à droite, et le cas d'un point qui finit dans un volume à gauche, dont on doit tester la position face à la droite (EF) avant de conclure qu'il est dans l'ombre.

dans la scène. Un autre aspect intéressant est la complexité de la requête dans le SVBSP, qui est en  $O(\log n)$  pour une scène avec  $n$  triangles.

Présentant des problèmes de mémoires et d'approximations numériques, cette méthode a été délaissée, bien que Chrysanthou et al. [34] en présentent une amélioration permettant de gérer des scènes dynamiques de taille modérée, avec une mise à jour rapide de la structure des SVBSP.

### Comparaison des méthodes

Les techniques basées Shadow Map ont les avantages d'être rapides et stables. Ces avantages sont inhérents au fonctionnement des Shadow Maps : la première passe de rasterisation de la Shadow Map, accélérée sur GPU, est quasiment insensible à la complexité géométrique ; ensuite le test de visibilité se limite en principe à un accès mémoire par fragment de l'image finale. Le résultat est approximé et aliasé ; sa qualité va dépendre de la taille de Shadow Map utilisée, donc du budget mémoire alloué, ainsi que l'adaptation du paramétrage à la variante utilisée. Cependant, même si la rasterisation est rapide, l'industrie a choisi de masquer les défauts de la méthode par la combinaison de cascades, de

déformations perspectives et de Shadow Maps de très grande résolutions, ce qui peut générer un coût mémoire important.

Pour les volumes d'ombre, le résultat est exact, mais la stabilité du temps de rendu est incertaine ; car le rendu des volumes d'ombres variables en tailles et en nombres selon le point de vue, génère un trafic mémoire inégal. Ce trafic mémoire augmente avec la quantité de géométrie, et sature rapidement la bande passante du GPU. Les diverses améliorations proposées tentent d'améliorer la stabilité mais échouent à résoudre les cas pathologiques.

La technique de Sintorn [30] n'utilise pas la rasterisation matérielle des GPU pour rendre les volumes d'ombre, mais se base sur une rasterisation hiérarchique logicielle pour limiter le coût mémoire du rendu des volumes d'ombres. De son côté, Wymann [31] utilise une approche hybride, mêlant cascade de tampons de profondeur irréguliers et lancer de rayon pour obtenir une ombre exacte par pixel, au prix d'un paramétrage complexe.

La tendance à la hausse des résolutions des écrans avec le 4K et bientôt le 8K, ainsi que la faible amélioration des performances de la mémoire des GPU, ne sont favorable ni aux volumes d'ombre avec une hausse du trafic mémoire lié aux hautes résolutions, ni aux Shadow Maps qui vont voir leur consommation mémoire augmenter. L'ombrage en temps réel reste donc encore une problématique ouverte.

### 1.3 Conclusion

L'ombre, élément crucial en synthèse d'images, est un domaine de recherche ouvert à la fin des années 70 par Williams, avec les Shadow Maps et Crow pour les volumes d'ombre. Une grande majorité des publications sur le sujet se basent sur ces 2 méthodes. Après 40 ans de recherche sur ces 2 familles de techniques, les volumes d'ombres, malgré l'exactitude de leur résultat, ont été délaissés par l'industrie à cause de leur instabilité ainsi que leur difficulté à gérer, en temps réel, des scènes géométriquement complexes en dépit de la simplification par extraction silhouette de Tomas Akenine-Möller [22] et des améliorations proposées par Lloyd [26] et Elmar Eisenmann [27] visant à rendre la méthode plus stable. Ce sont les Shadow Maps qui sont massivement utilisées par l'industrie, notamment vidéoludique, pour leur rapidité et leur stabilité. Pour minimiser autant que possible l'imprécision inhérente à la méthode, c'est généralement une combinaison des optimisations de Brabec [13] avec la déformation perspective de Stamminger [14] et les cascades de Dimitrov [15] qui est utilisée. Ces 2 techniques peuvent être considérées comme matures. Il y a peu d'espoir d'y apporter des améliorations significatives. Pour preuve, les dernières publications marquantes de Sintorn [30] et Wymann [31] s'orientent vers des approches hybrides avec d'un côté une rasterisation hiérarchique logicielle et de l'autre une combinaison de lancer de rayon et de tampons de profondeur irréguliers.

Il reste le cas des SVBSP de Chin et al. [32], une méthode originale qui se démarque en proposant d'obtenir un résultat exact, comme les volumes d'ombres, tout en ayant une propriété intéressante : avoir un comportement logarithmique vis à vis de la complexité géométrique. Malheureusement les opérations CSG utilisées pour construire les SVBSP introduisent des imprécisions numériques qui rendent la méthode peu robuste avec de potentiels cas pathologiques menant à une explosion du coût mémoire. L'objectif des travaux présentés dans ce mémoire est de proposer une méthode robuste et adaptée au GPU, inspirée des SVBSP, qui permettent d'obtenir un résultat exact par pixel tout en gardant le comportement logarithmique vis à vis de la complexité géométrique afin de proposer une technique plus stable mais aussi précise que celle des volumes d'ombre.

## Chapitre 2 .

# Partition de Volumes d'ombre

Ce travail, publié dans la revue *Computer Graphics Forum* [42] suite à la conférence *Eurographics 2015*, prend pour base les arbres SVBSP [32]. Dans un premier temps, il faudra expliquer pourquoi cette méthode, en plus d'être sujette à des instabilités numériques, est peu adaptée au traitement parallèle des GPU modernes. Par la suite, on présentera une nouvelle structure accélératrice plus efficace que les SVBSP.

## 2.1 Inadaptation des SVBSP au traitement sur GPU

### Problèmes de mémoire

Le calcul sur GPU demande de respecter certaines règles spécifiques pour rester efficace. Ainsi il faut autant que possible :

- minimiser les échanges entre CPU et GPU ;
- minimiser les allocations mémoire ;
- tenir compte de la quantité de mémoire disponible, les GPU n'intégrant pas de mécanisme de mémoire virtuelle en cas de surcharge.

Or il est impossible de prévoir la consommation mémoire d'un SVBSP sans l'avoir construit. Le cas pathologique de la figure 2.1 illustre le problème du coût mémoire maximal en  $O(n^2)$  pour la création des SVBSP. Sachant que les GPU ne supportent pas l'allocation mémoire dynamique, il faudrait donc systématiquement allouer de la mémoire sur GPU pour le pire cas en  $O(n^2)$ , ce qui serait prohibitif.

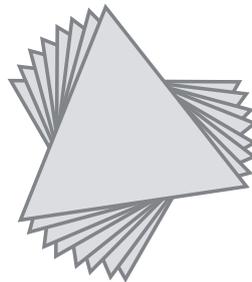


FIGURE 2.1 – Cas pathologique pour les SVBSP

Si on construit un SVBSP avec cette configuration géométrique du point de vue de la lumière, chaque arête et plan d'ombre correspondant coupent tous les autres triangles : en plus de générer des instabilités numériques, la consommation mémoire explose en  $O(n^2)$  pour  $n$  triangles.

### Problèmes de parallélisme

Pour être efficace, le traitement en parallèle sur GPU, doit tenir compte de la faible quantité de mémoire disponible par processus et, vérifier que les processus d'un groupe d'exécution suivent les mêmes instructions. Si jamais un ou des processus doivent suivre des instructions différentes, les autres sont mis en pause le temps de pouvoir synchroniser à nouveau tous les processus donc le traitement devient moins efficace. Or les opérations CSG nécessaires à la création d'un SVBSP impliquent de gérer des polygones avec un nombre de sommets variable et non prévisible, ce qui introduit de la divergence d'exécution et de la pression sur les registres. On parle de pression sur les registres quand un algorithme a besoin de garder en mémoire plus de variables que le nombre de registres du processeur qui l'exécute.

## 2.2 Introduction de la partition ternaire des objets

Pour éviter les écueils inhérents aux SVBSP, il faut proposer une nouvelle structure accélératrice qui évite la découpe géométrique, afin de supprimer les imprécisions numériques et l'explosion du coût mémoire. On va donc s'éloigner de la partition de l'espace pour aller vers une partition ternaire des objets.

### Principe

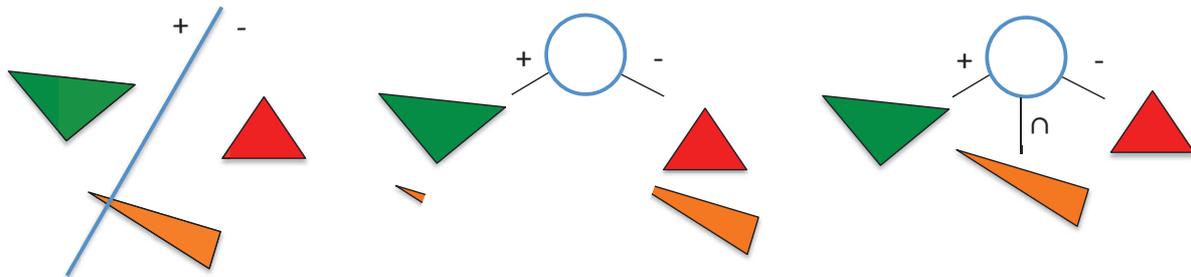


FIGURE 2.2 – Introduction d'une structure de donnée ternaire

A gauche, un plan de séparation qui filtre de la géométrie, au centre la représentation BSP et à droite la représentation avec un nœud d'arbre ternaire : le fils intersection permet d'éviter de découper le triangle orange.

L'idée consiste à conserver une structure proche de l'arbre BSP, en gardant un fils + et un fils - mais en rajoutant un troisième fils dit « intersection », qui contient toute la géométrie intersectant le plan d'un nœud. On évite alors toute découpe : la géométrie qui serait découpée dans un arbre BSP va ici dans le fils intersection comme sur la figure 2.2. Avec cette nouvelle structure, chaque élément géométrique n'est représenté qu'une seule fois dans l'arbre qui sera construit. Ce changement de structure de données implique un parcours différent de celui d'un arbre BSP : le parcours des fils positifs et négatifs est équivalent à celui d'un arbre BSP, mais il faut tenir compte du fils intersection qui, s'il n'est pas vide, doit être visité systématiquement puisqu'il comporte de la géométrie qui se trouve des 2 cotés du plan de séparation.

Avec cette nouvelle structure de données, les opérations CSG ne sont plus nécessaires, ce qui supprime la réplication de données et les imprécisions numériques inhérentes aux SVBSP. Il est temps d'introduire la technique de partition de volumes d'ombre basée sur les partitions ternaires des objets.

## 2.3 Partition de volumes d'ombre

La technique des partitions de volumes d'ombre repose sur 2 étapes pour le rendu d'une image, ainsi que le montre la figure 2.3. Il faut noter que dans les résultats qui seront présentés la phase de construction est systématiquement recalculée à chaque image ; mais en pratique si la source lumineuse et la géométrie ne bougent pas, il n'est pas nécessaire de mettre à jour la structure qui est indépendante du point de vue.

Avant de détailler les 2 étapes rapidement présentées, il faut d'abord voir comment représenter un volume d'ombre avec une partition ternaire.

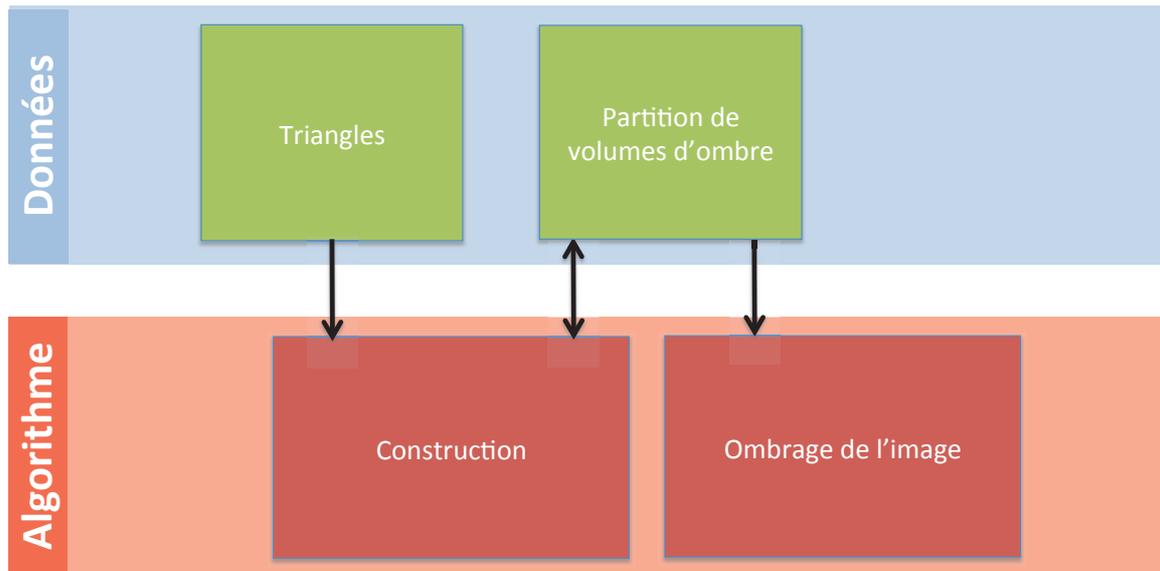


FIGURE 2.3 – Vue d'ensemble

Toute la géométrie de la scène est utilisée lors de la construction de la partition de volumes d'ombre. Ensuite, les fragments de l'image vont traverser la structure pour déterminer l'ombrage.

### Représentation d'un volume d'ombre par un arbre ternaire

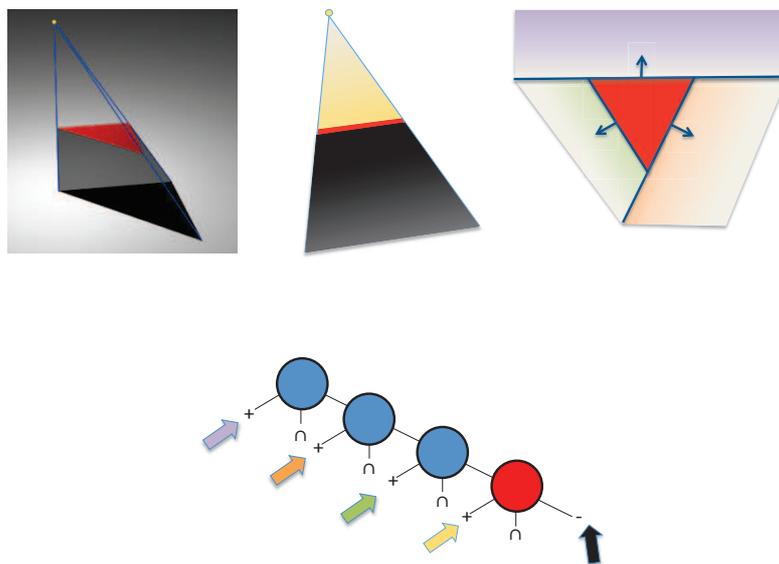


FIGURE 2.4 – Représentation d'un volume d'ombre avec un arbre ternaire.

En haut à gauche un volume d'ombre généré par un triangle; en haut au milieu et à droite, le même volume d'ombre vu en coupe et du dessus. En bas la représentation de ce volume d'ombre sous la forme de 4 nœuds d'arbre ternaire. Chaque flèche permet de voir à quelle zone de l'espace correspond chaque nœud positif et négatif. Les nœuds intersections contiendront la géométrie qui intersecte chacun des plans.

Sur la figure 2.4, on peut voir, à gauche : un volume d'ombre généré par le triangle rouge, qui est composé des 3 plans issus de la lumière, et des 3 arêtes du triangle et du plan du triangle lui-même; au centre, le même volume d'ombre vue en coupe; et à droite enfin une représentation depuis la lumière, qui

illustre les différents espaces couverts par chacune des feuilles de la représentation ternaire. En bas de la figure 2.4, la représentation d'un volume d'ombre avec des nœuds d'arbre ternaire donne une suite de 4 nœuds représentant les 4 plans composant un volume d'ombre. De cette façon, si un point se trouve du côté négatif de chacun des 4 plans, alors il est dans le volume d'ombre.

Il est temps maintenant de faire une présentation de la méthode des partitions de volume d'ombre.

### 2.3.1 Construction d'une partition de volumes d'ombre

#### Principe

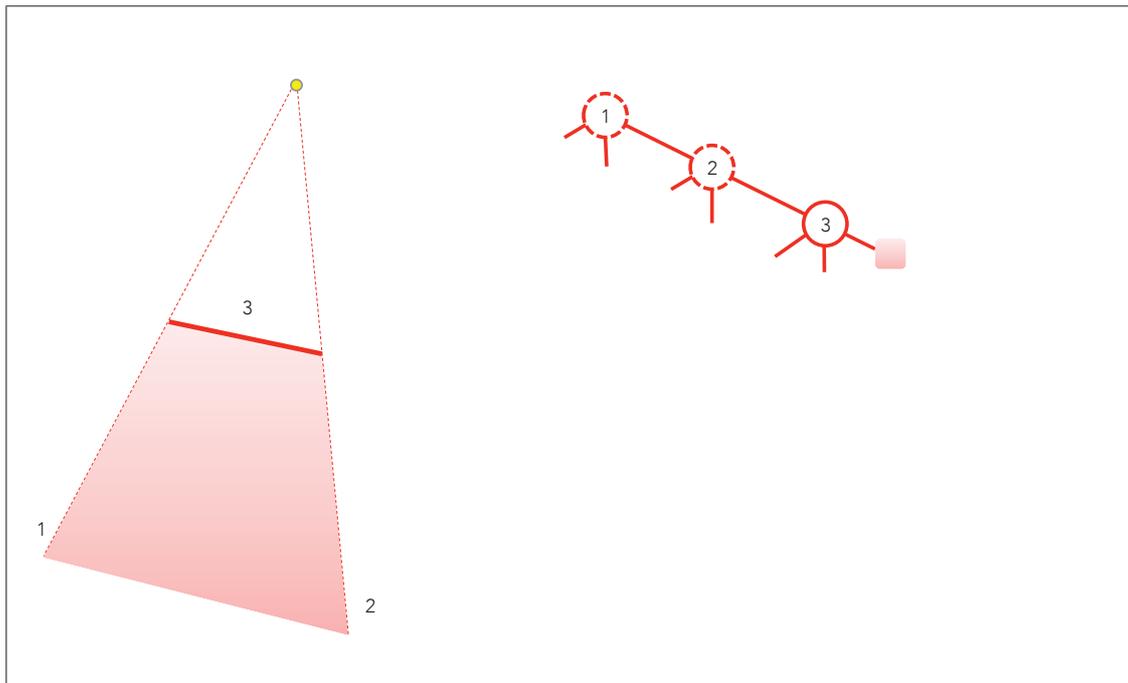
Lorsqu'on prend la lumière comme origine du repère, un triangle et son volume d'ombre sont projectivement confondus; donc classer un volume d'ombre par rapport à un plan revient simplement à localiser la position du triangle correspondant. La construction commence avec un premier triangle inséré dans une partition de volume vide sous la forme de 4 nœuds contenant les 4 plans décrivant le volume d'ombre correspondant. Ensuite, pour ajouter un nouveau volume d'ombre issu d'un triangle, on fait traverser la partition de volumes d'ombre, depuis sa racine, par ce triangle jusqu'à trouver un fils vide où insérer son volume d'ombre; il suffit de tester la position du triangle par rapport au plan stocké dans le nœud courant pour déterminer dans quel fils le triangle se trouve. Pour simplifier la représentation, on prendra un exemple en 2 dimensions, où un volume d'ombre est représenté par seulement 3 nœuds au lieu de 4 en 3D. Les tableaux 2.1 et 2.2 montrent les différentes étapes d'une construction de partition de volumes d'ombre, avec un exemple simple de 4 surfaces en suivant l'algorithme 2 (sans les optimisations).

Avec une telle construction, la complexité pour insérer un triangle est en  $O(\log(n))$ ; donc pour  $n$  triangles, la complexité est en  $O(n \log(n))$ . Dans le cas 3D, on insère seulement les triangles qui du point de vue de la lumière ont une orientation négative. Ainsi on diminue le nombre de nœuds dans la partition de volumes d'ombre, tout en diminuant le trafic mémoire nécessaire. La construction est parallélisée, ainsi chaque processus va traiter un triangle. Le fait que la partition de volumes d'ombre ne nécessite pas d'opération CSG permet d'éviter les imprécisions numériques, et permet de borner la consommation mémoire de la structure :  $n$  triangles seront représentés par  $4n$  nœuds.

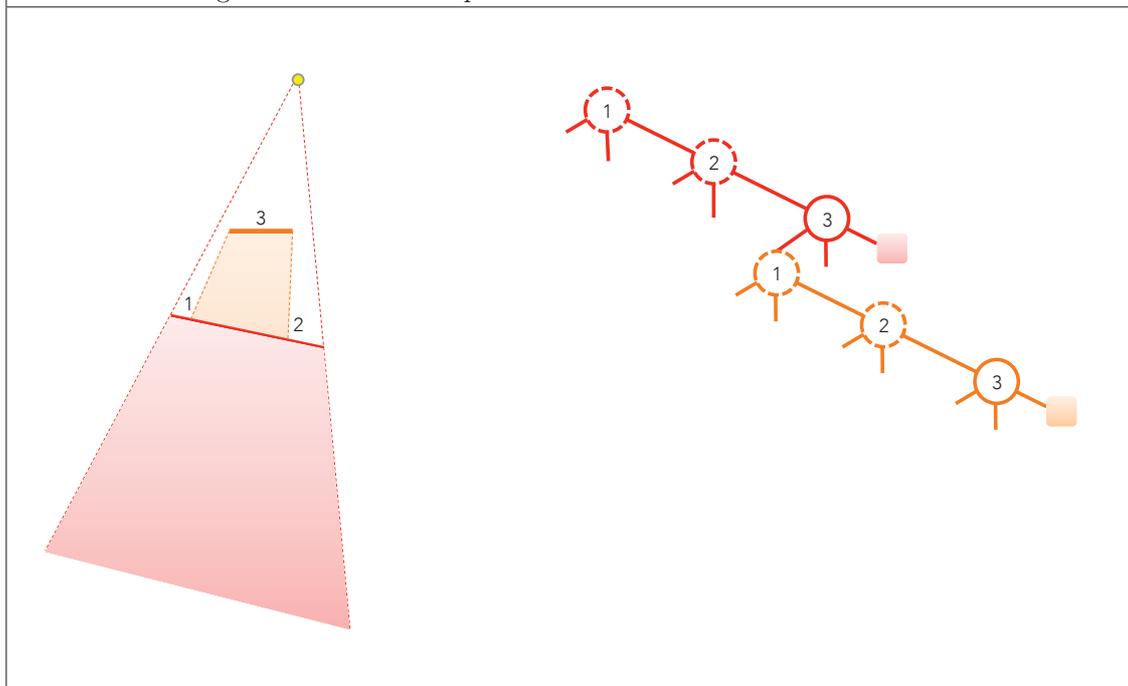
#### Équilibrage de la structure

Lors de la création d'une structure accélératrice sous forme d'arbre, on cherche généralement à équilibrer l'arbre pour permettre un parcours aussi logarithmique que possible; un arbre ternaire idéalement équilibré serait un arbre dont tous les fils intersection seraient vides et équilibré entre les fils positifs et négatifs, ce qui reviendrait à un arbre BSP équilibré. Un cas aussi idéal ne peut se limiter qu'à des configurations géométriques et des points de vue particuliers. Dans un cas plus général, pour équilibrer un arbre ternaire, il faut minimiser et le nombre de sous-arbres intersection, et équilibrer au mieux les fils positifs et négatifs. Pour éviter de recourir à une heuristique complexe, coûteuse et difficile à mettre en place dans une construction en parallèle, la solution consiste à introduire de l'aléa dans l'ordre d'insertion des triangles pour éviter une construction très déséquilibrée (cf. figure 2.5). Cette approche est analogue à certaines implémentations du tri rapide ou *quicksort* : dans le pire des cas, la complexité du tri est en  $O(n^2)$ , mais l'introduction d'aléas permet de tendre vers un comportement moyen en  $O(\log(n))$ . La comparaison du temps de parcours d'un arbre construit avec aléa à celui d'un arbre équilibré construit sur CPU montre des différences très faibles en pratique.

Il reste le cas de la figure 2.1, qui est une configuration pathologique : chaque arête de chaque triangle

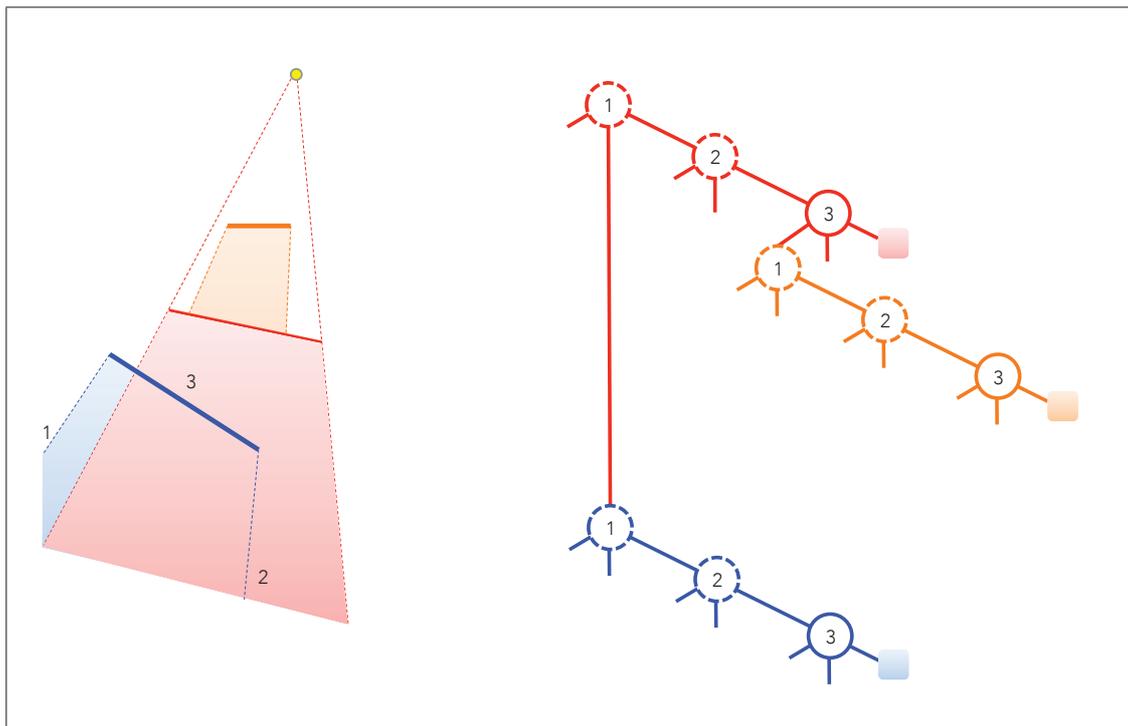


Première surface : on insère dans un arbre vide les 3 nœuds correspondants au volume d'ombre de la surface rouge avec les différents plans la délimitant.

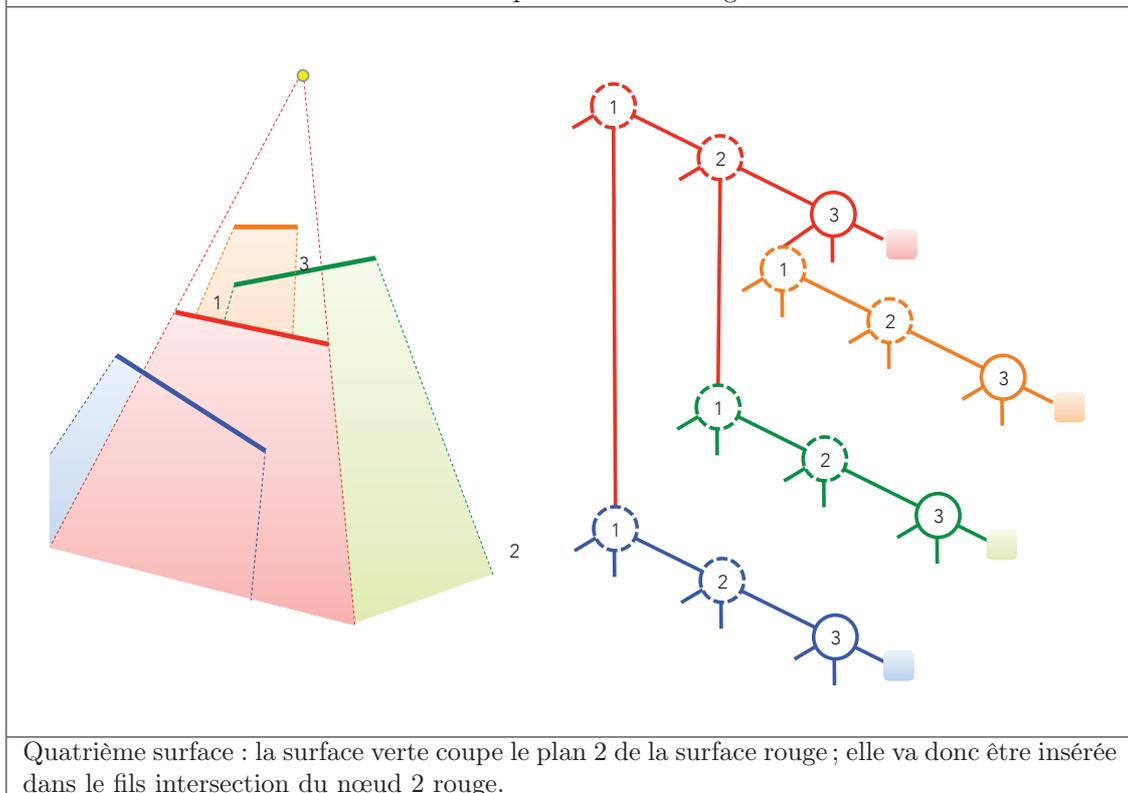


Seconde surface : la surface orange est du côté négatif des plans 1 et 2 de la surface rouge, mais se trouve devant la surface rouge. Elle est insérée dans le fils positif du nœud rouge 3.

Tableau 2.1 – Construction d'une partition de volumes d'ombre



Troisième surface : la surface bleue est intersectée par le premier plan de la surface rouge ; elle va être insérée dans le fils intersection du premier nœud rouge.



Quatrième surface : la surface verte coupe le plan 2 de la surface rouge ; elle va donc être insérée dans le fils intersection du nœud 2 rouge.

Tableau 2.2 – Construction d'une partition de volumes d'ombre

---

**Algorithm 2** Algorithme d'ajout d'un volume d'ombre dans une partition de volumes d'ombre : un triangle  $t$  va parcourir l'arbre existant afin de trouver la feuille libre lui correspondant, et y insérer sa représentation ternaire. Les lignes bleutées correspondent à une optimisation présentée ultérieurement.

---

```

1: Node {
2:   // un plan d'ombre ou plan support du triangle
3:   Plan plan
4:   // lien vers les fils positif, negatif et intersection
5:   Node positif, intersection, negatif
6:   // on stocke l'angle limitant le fils intersection
7:   flottant angle
8: }
9:
10: AjoutVolumeOmbre(Node racine, Triangle t, lumière l)
11: Node n ← racine
12: while n n'est pas une feuille do
13:   flottant location ← position(n.plan, t)
14:   if location > 0 then
15:     n ← n.positif
16:   else if location < 0 then
17:     n ← n.negatif
18:   else
19:     // le triangle t est coupé par le plan n.plan
20:     if n.plane est issue d'une arête then
21:       // On met à jour l'angle si le triangle est hors de l'angle déjà stocké
22:       MiseAJourAngleIntersection(n, t)
23:     end if
24:     n ← n.intersection
25:   end if
26: end while
27:
28: RemplacerFeuilleParVolumeOmbre(n, t, l)

```

---

coupe tous les autres triangles. Dans ce cas, l'arbre construit est une liste chaînée par les fils intersection des volumes d'ombre. Contrairement au cas du SVBSP, cette situation ne mène pas à une explosion du coût mémoire, qui reste fixe avec les partitions de volumes d'ombre.

### 2.3.2 Parcours d'une partition de volumes d'ombre

#### Principe

Maintenant que la structure accélératrice est prête, voyons comment l'interroger en reprenant l'arbre construit précédemment. Le parcours de la partition de volumes d'ombre est proche de celui d'un arbre BSP : pour chaque nœud, on visite le fils positif ou négatif suivant la position d'un point par rapport au plan du nœud. Toutefois il ne faut pas oublier d'aller visiter le fils intersection. En pratique, on va parcourir la structure comme un BSP, et utiliser une pile pour stocker tous les sous-arbres intersections à visiter par la suite. La requête s'achève dès qu'un point est trouvé dans un volume d'ombre, ou lorsque la pile se retrouve vide et qu'il ne reste aucun volume d'ombre susceptible de contenir le point traité. Les tableaux 2.3, 2.4 et 2.5 illustrent les parcours pas à pas d'un point illuminé et d'un point ombré dans la partition ternaire, en suivant l'algorithme de parcours 3.

<p>Le point part de la racine et se situe du coté négatif du premier nœud rouge. On ajoute le premier fils intersection à la pile pour le visiter éventuellement. Plus tard, on continue vers le second nœud rouge.</p>
<p>Le point se situe du coté négatif du second plan rouge ; il faut ajouter le fils intersection dans la pile et continuer vers le troisième nœud rouge.</p>
<p>Le point se situe derrière la surface : on peut en conclure que le fragment est dans le volume d'ombre rouge.</p>

Tableau 2.3 – Traversée d'une partition de volumes d'ombre par un point à l'ombre

<p>Le début du parcours du point est identique à celui du point à l'ombre jusqu'au troisième nœud rouge. Le point étant entre la lumière et la surface rouge, il doit visiter le volume d'ombre orange.</p>
<p>Le point se situe du côté négatif des 2 premiers plans du volume d'ombre orange et est du côté positif du troisième nœud et sort du volume d'ombre.</p>
<p>Il faut vider la pile et le point doit visiter le volume d'ombre vert. Le point est rejeté par le troisième nœud vert, et se trouve donc hors du volume d'ombre vert.</p>

Tableau 2.4 – Traversée d'une partition de volumes d'ombre par un point éclairé

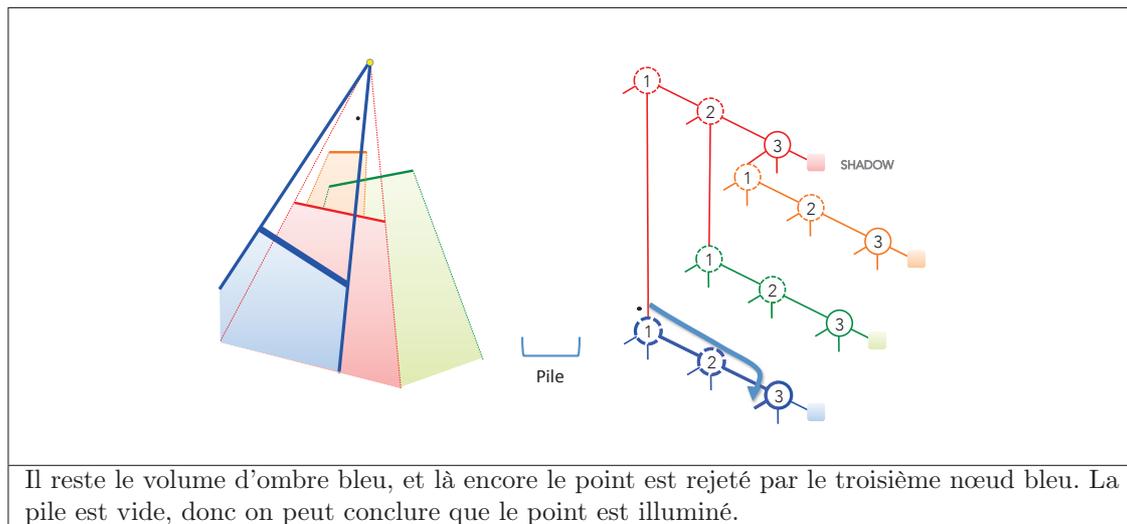


Tableau 2.5 – Traversée d'une partition de volumes d'ombre par un point éclairé

**Algorithm 3** Interrogation d'une partition de volumes d'ombre

```

1: Interrogation(Node racine, Point p)
2: PileDeNœud pile
3: Ajouter(pile, racine)
4: while pile n'est pas vide do
5:   Node b ← Depiler(pile)
6:   if n n'est pas une feuille then
7:     Entier location ← signe(n.plan.p)
8:     if n.plan est plan d'ombre then
9:       // on est avec un plan issu d'une arête du triangle
10:      // Il faut ajouter dans la pile le fils intersection
11:      // est ce que le point p est contenu dans l'angle n.angle
12:      flottant pAngle ← CalculAngle(n.plan.p)
13:      if pAngle < n.angle then
14:        Empiler(pile, n.intersection)
15:      endif
16:      // et le positif ou négatif dans le quel se trouve le point
17:      Empiler(pile, location>0 ? n.positif : n.négatif)
18:    else
19:      // le plan est le plan d'un triangle
20:      if location<0 then
21:        // Le point dans un volume d'ombre, la requête se termine
22:        Retourner(0)
23:      end if
24:      Empiler(pile, n.intersection)
25:      Empiler(pile, n.positif)
26:    end if
27:  end if
28: end while
29: // la pile est vide, il n'y a plus de volume d'ombre pouvant contenir p qui est donc dans la lumière
30: Retourner(1)
    
```

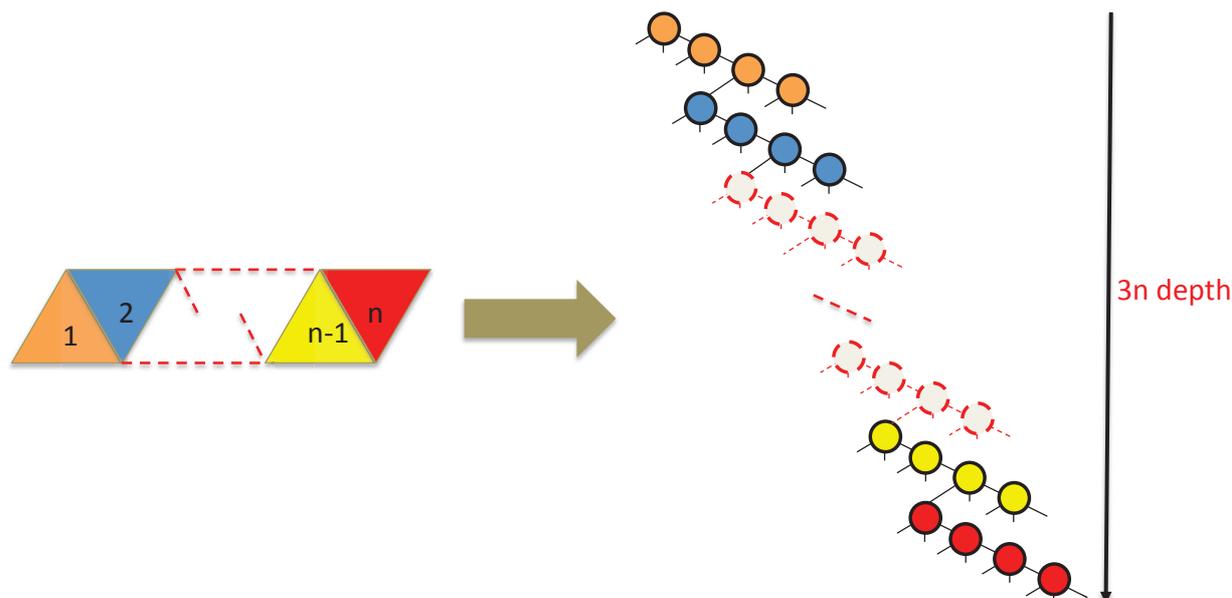


FIGURE 2.5 – Le cas d'une bande de triangles

Pour des raisons de performance et d'économie de mémoire, les modèles géométriques voient leurs triangles organisés sous forme de bandes de triangles voisins comme sur la gauche. Si les triangles sont insérés dans l'arbre ternaire dans le même ordre que leur position dans la bande de triangles, la structure devient une liste chaînée des différents volumes d'ombre, perdant ainsi toute son efficacité. La construction devient coûteuse, chaque nouveau triangle inséré devant parcourir tout l'arbre pour trouver une feuille disponible, et le parcours consiste alors à tester tous les volumes d'ombre.

### Optimisation

À ce stade, la méthode fonctionne et permet d'obtenir des résultats justes. Mais les performances ne sont pas satisfaisantes, et pour le comprendre il faut revenir sur le concept d'arbre ternaire de partition des objets. Il permet comme vu précédemment de garantir une empreinte mémoire fixe, et d'éviter les opérations CSG, mais il y a un prix à payer : il faut visiter pour chaque nœud testé le fils + ou - et le fils intersection, soit 2 fils sur 3. Cela a pour conséquence de devoir visiter énormément de nœuds avec potentiellement le besoin d'une grosse pile pour stocker les nœuds intersections et des accès mémoires accrus. Pour éviter ces surcoûts, il faut restreindre autant que possible l'accès au sous-arbre intersection. Afin d'y arriver, on peut déterminer, comme sur la figure 2.6, un volume englobant l'ensemble d'un sous-arbre intersection qui permet, par exemple, au fragment vert d'économiser la visite d'un sous-arbre intersection. Cette optimisation permet de limiter le nombre de nœuds à visiter pour tendre autant que possible vers un parcours proche de celui d'un arbre BSP, et donc aller vers une complexité de la traversée de la structure en  $O(\log(n))$ .

Elle implique de légères modifications dans les algorithmes de construction et de parcours :

- pour la construction (cf. algorithme 2 avec les lignes bleutées), il faut stocker pour chaque fils intersection un angle qui doit être mis à jour à chaque fois qu'un triangle descend dans le sous-arbre correspondant ;
- pour le parcours (cf. algorithme 3 avec les lignes bleutées), avant d'ajouter dans la pile un sous-arbre intersection, il faut vérifier que le point à ombrer se situe bien dans le volume englobant défini par le plan du nœud, la lumière et l'angle stocké à la construction. Cette optimisation permet de minimiser le nombre de sous-arbres intersection à visiter, et donc le nombre de nœuds à stocker

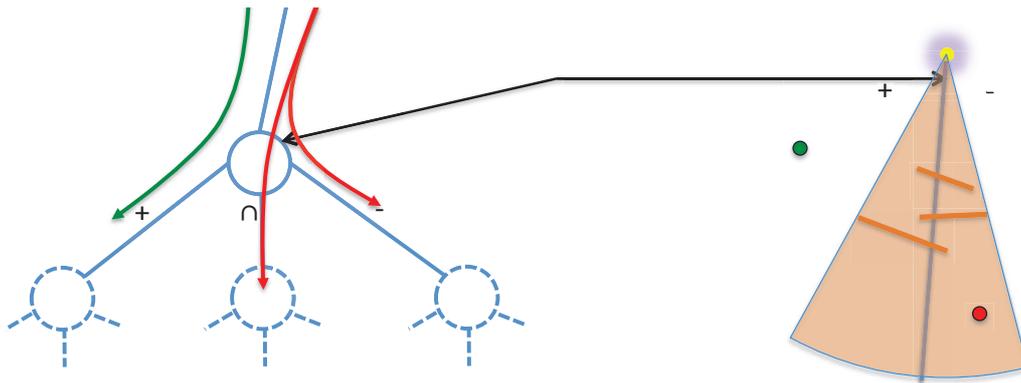


FIGURE 2.6 – Volume englobant du sous-arbre intersection

Pour tous les nœuds qui représentent des plans d'ombre, les 3 premiers d'un volume d'ombre, on délimite le volume englobant toute la géométrie du sous arbre-intersection par deux angles, un pour le coté positif et un pour le coté négatif. En pratique, on ne stocke que l'angle le plus grand des deux pour garder une structure compacte. Lorsqu'un fragment arrive dans ce nœud, une simple comparaison angulaire permet de déterminer s'il doit ou non visiter le sous-arbre.

dans la pile.

### 2.3.3 Extension à la transparence

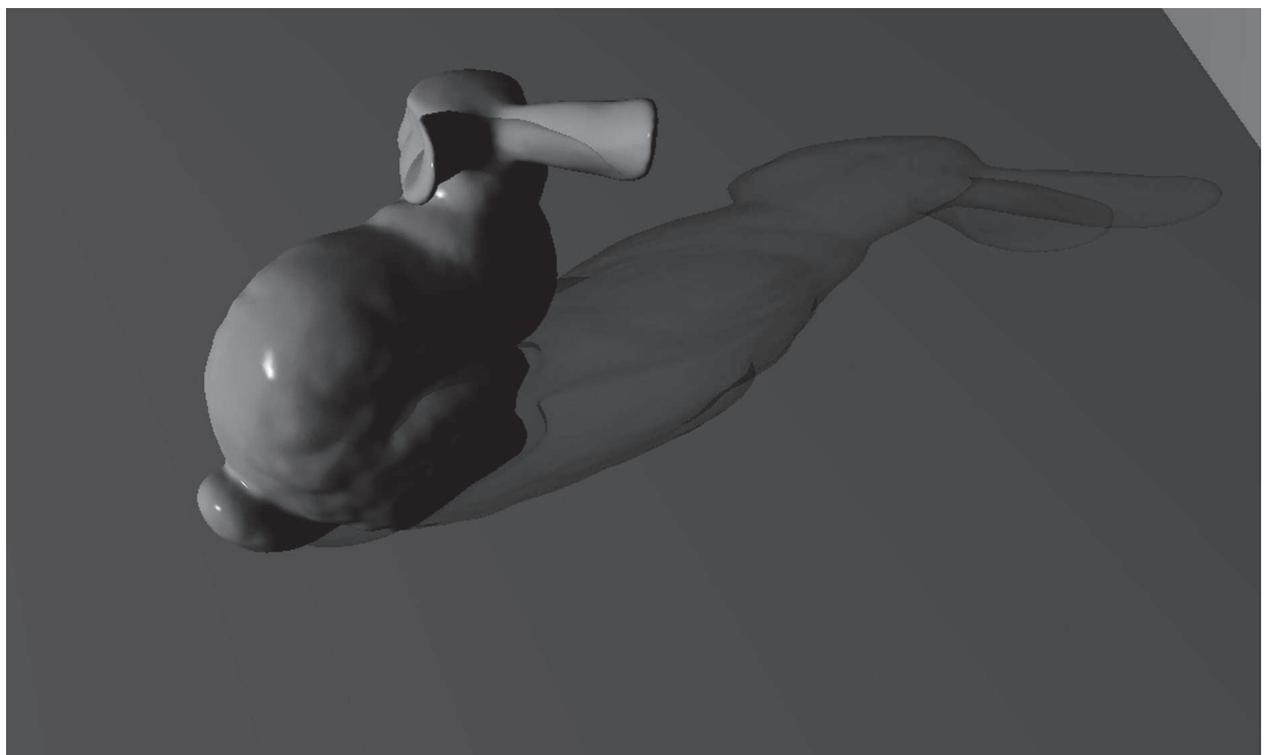


FIGURE 2.7 – Bunny - 69 k triangles - 8.43 Mo : on simule l'ombre d'un lapin semi-transparent

Avec une légère modification de la construction et du parcours de la structure accélératrice, on peut gérer des ombres translucides : la structure permet de trouver tous les triangles traversés par

un rayon partant de la lumière vers un fragment. On peut ainsi gérer le cas d'objets transparents ou semi-transparentes comme sur la figure 2.7. Pour arriver à un type de rendu qui ressemble à une vision par rayon  $X$  comme en radiographie, il faut prendre les précautions suivantes :

- lors de la construction de la structure, tous les triangles de l'objet générant de l'ombre doivent être insérés dans la structure : il faut donc supprimer l'optimisation de suppression des faces avant par rapport à la lumière ;
- le parcours est modifié (voir l'algorithme 4) : on ne peut plus terminer la traversée dès que le fragment est masqué par un volume d'ombre, il faut finir de visiter les sous-arbres empilés.

Un problème subsiste : la structure ne permet pas de déterminer l'ordre dans lequel le rayon partant de la caméra vers le fragment traverserait les triangles de la scène. Pour calculer une atténuation de la lumière, il faut une méthode qui soit décorrélée de l'ordre de traversée de la géométrie. Carpenter et al. [35] ont proposé en 1984 une première méthode d'une famille de techniques dite de transparence indépendante de l'ordre ou « *order independent transparency* » ou encore « *OIT* ». En s'inspirant de cette idée, on utilise un calcul d'atténuation fonction de l'angle entre la normale du triangle traversé et la direction de la droite définie par le fragment et la lumière.

---

**Algorithm 4** Interrogation pour une ombre translucide
 

---

```

1: Interrogation(Node racine, Point p, Lumiere l)
2: PileDeNœud pile
3: flotant attenuation  $\leftarrow$  0
4: Empiler(pile, racine)
5: while pile n'est pas vide do
6:   Node b  $\leftarrow$  Depiler(pile)
7:   if n n'est pas une feuille then
8:     Entier location  $\leftarrow$  signe(n.plan, p)
9:     if n.plan est plan d'ombre then
10:      // on est avec un plan issue d'une arête du triangle
11:      // Il faut ajouter dans la pile le fils intersection
12:      // est ce que le point p est contenu dans l'angle n.angle
13:      flotant pAngle  $\leftarrow$  CalculAngle(n.plan, p)
14:      if pAngle < n.angle then
15:        Empiler(pile, n.intersection)
16:      endif
17:      // et le positif ou négatif dans le quel se trouve le point
18:      Empiler(pile, location>0 ? n.positif : n.négatif)
19:    else
20:      // le plan est le plan d'un triangle
21:      if location<0 then
22:        // Il faut calculer et mettre à le facteur d'atténuation
23:        CalculAttenuation(attenuation, p, l, n.plan)
24:      end if
25:      Empiler(pile, n.intersection)
26:      Empiler(pile, n.positif)
27:    end if
28:  end if
29: end while
30: // la pile est vide, il n'y a plus de volume d'ombre pouvant contenir p qui est donc dans la lumière
31: Retourner(attenuation)

```

---

### 2.3.4 Implémentation

L'implémentation se gère entièrement sur GPU, comme le montre la figure 2.8 : dans une première étape, on va construire la partition de volumes d'ombre stockée dans la mémoire du GPU ; l'étape suivante concernera le parcours des fragments de l'image pour déterminer l'ombrage de la scène. Afin d'optimiser le rendu de l'ombre, on utilise un rendu indirect pour éviter des calculs inutiles (ce procédé sera détaillé par la suite). Les codes sources des différents parcours et constructions associées sont disponibles sur GitHub : <https://github.com/PSVcode/EG2015>.

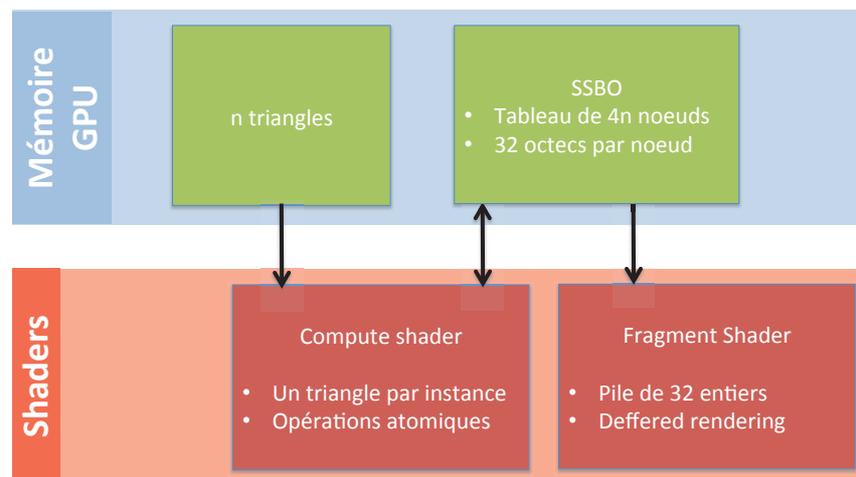


FIGURE 2.8 – Implémentation GPU

En bleu la mémoire GPU qui contiendra la géométrie ainsi que la partition de volumes d'ombre sous la forme d'une tableau de noeuds ; et en orangé les programmes de construction et de parcours.

#### Stockage

Pour stocker la partition de volumes d'ombre sur GPU, on utilise un *shader buffer object* ou SSBO dont la structure est détaillée sur la figure 2.9.

#### Construction

L'étape de construction utilise des *compute shaders* : introduits en 2012 dans la version 4.3 d'OpenGL, ils offrent la possibilité d'utiliser les unités de calcul du GPU pour exécuter parallèlement des programmes qui ne sont pas forcément liés à l'affichage. Ainsi on lancera une instance de compute shader par triangle à insérer dans la structure en suivant l'algorithme 2. Le mélange des triangles au chargement de la scène combiné avec l'ordonnancement des processus sur GPU reproduit l'aléa recherché. Pour éviter les conflits d'écriture au cas où 2 triangles arriveraient dans une même feuille en même temps, on utilise des opérations atomiques lors de l'insertion d'un triangle sous sa forme de volume d'ombre. L'arrivée d'un triangle dans un fils dont la feuille vaut 0 signifie qu'une place est disponible ; l'instruction *atomicCompSwap* permet de remplacer le 0 par la position dans le tableau du premier des 4 noeuds du volume d'ombre correspondant.

Il y a un deuxième cas d'accès concurrent possible lors de la mise à jour de la valeur angulaire du fils intersection. Ici, la valeur angulaire est stockée dans un entier à l'aide de la fonction *floatBitsToUint*. Il n'existe pas d'implémentation standardisée pour opération atomique sur des valeurs flottantes, mais

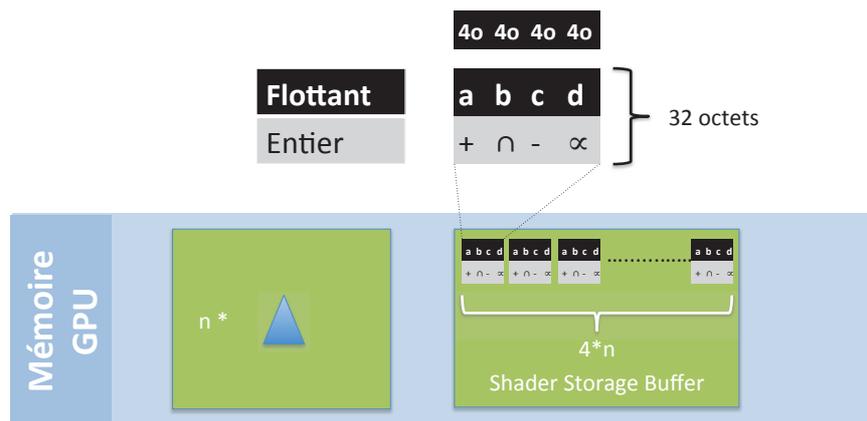


FIGURE 2.9 – Stockage

On écrit dans le SSBO les nœuds sous forme d'un tableau à une dimension. On peut repérer les nœuds par leurs indices. La représentation des nœuds est compacte : la taille est un multiple de 16 octets afin de garantir un accès optimisé aux données sachant que le GPU peut accéder en un seul transfert à 16 ou 32 octets de données alignées. Chaque nœud comprend l'équation du plan d'ombre, (ici a,b,c,d) et les liens vers les différents fils. L'angle  $\alpha$  englobant la géométrie du fils intersection est stocké dans un entier à l'aide de la fonction *floatBitsToUint*.

heureusement la représentation sous la forme d'entier permet de conserver les relations d'ordre tant qu'il s'agit de valeurs positives : ainsi la fonction *atomicMax*, sur des flottants encodés dans des entiers, se comporte comme une fonction atomique maximum sur des flottants positifs. Les opérations atomiques sont gérées efficacement par les dernières générations de GPU. De plus, les accès concurrents en écriture concernent surtout le début de l'arbre ; ensuite, au fur et à mesure de la descente dans l'arbre, les conflits deviennent de moins en moins nombreux.

## Parcours

Ce sont les unités de traitement par pixel ou *fragment shaders* qui vont interroger notre structure en respectant l'algorithme 3. On utilise ici une pile de 32 entiers pour stocker les racines des sous-arbres intersection à explorer. La taille de la pile doit être choisie avec attention : si elle est trop importante, elle va saturer la faible capacité mémoire des unités de traitement et générer des échanges avec la mémoire globale du GPU qui sont à limiter au maximum, car ces échanges sont très lents. Si la pile est trop petite, on risque de faire face à des débordements de pile qui bloquent l'exécution. D'après nos tests, une pile de 32 entiers suffit à gérer des modèles géométriques finement détaillés allant jusqu'au million de triangles.

## Rendu indirect ou *Deffered rendering*

On utilise un rendu indirect : lors de la rasterisation, les triangles de la scène sont projetés sur l'image ; c'est une comparaison avec le tampon de profondeur qui permet de déterminer quel fragment restera sur l'image finale. Cela implique que plusieurs fragments peuvent arriver sur un même pixel. Or avec un rendu direct, l'apparence de chaque fragment sera systématiquement calculée, qu'il fasse parti ou non de l'image finale, ce qui génère un trafic mémoire inutile. Le rendu indirect permet ici d'éviter de calculer la visibilité de la source de lumière pour des éléments qui seront finalement invisibles sur l'image définitive. Le rendu indirect consiste à rendre la scène depuis la caméra et à stocker dans une texture les positions des fragments, dans notre cas, dans l'espace monde ; puis dans un deuxième temps à utiliser les données de

cette texture pour calculer l'image définitive et interroger la structure de partition de volumes d'ombre. Là où pour un rendu direct, le nombre de fragments à ombrer dépend de la complexité géométrique, le rendu indirect permet de ramener ce nombre à la quantité de pixels de l'image finale. La figure 2.10 montre le type de rendu indirect utilisé dans notre implémentation.

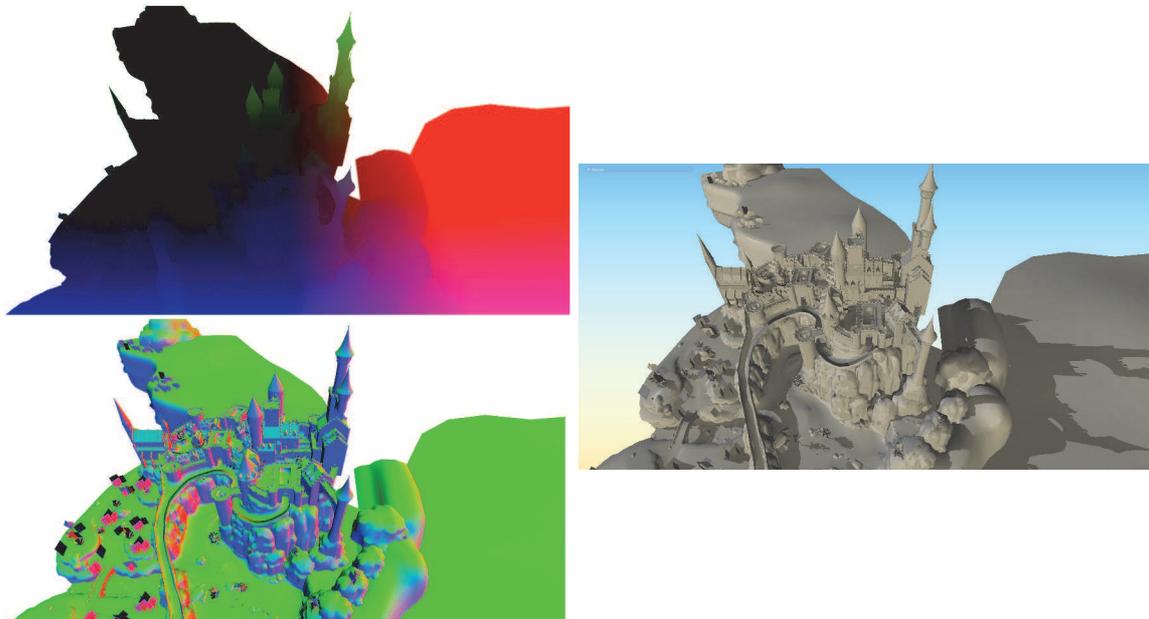


FIGURE 2.10 – Rendu indirect

Les positions dans l'espace monde et les normales des pixels sont stockées dans une texture. Ensuite on utilise ces informations pour faire le calcul d'illumination finale.

### 2.3.5 Résultats

Pour mesurer les performances, les tests ont été effectués sur une NVIDIA GTX 980 équipée de 4 Go de mémoire, et avec une résolution de  $1024 \times 1024$ . En considérant qu'on se trouve dans un cas où la géométrie et la lumière sont susceptibles de bouger, on recalcule la structure accélératrice à chaque image. On mesure par image le temps de construction et le temps de parcours de la structure par les pixels.

Les figures 2.11 à 2.16 montrent les différentes scènes utilisées, classées de la plus simple à la plus complexe avec leur nombre de triangles, ainsi que la mémoire nécessaire pour les partitions d'ombre correspondantes.

Pour tous les tests, on supprime lors de la construction les triangles orientés positivement par rapport à la lumière. Les mesures se portent sur la construction et le parcours de la structure (en millisecondes par image) ainsi que sur le nombre moyen de nœuds visités. Les résultats donnés dans le tableau 2.6 représentent le comportement moyen sur un millier d'images issues d'un parcours de la scène.

#### Optimisation angulaire

C'est l'optimisation la plus efficace, sans laquelle la méthode ne serait pas performante :

- la construction est légèrement plus rapide, d'un facteur variant de 0.77 à 0.99, car elle évite de calculer et stocker la valeur angulaire des fils intersections ; ce gain reste marginal car le temps de

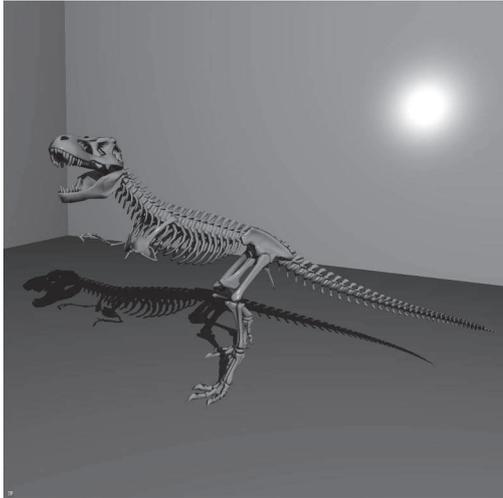


FIGURE 2.11 – T-rex - 20 k triangles - 2.44 Mo

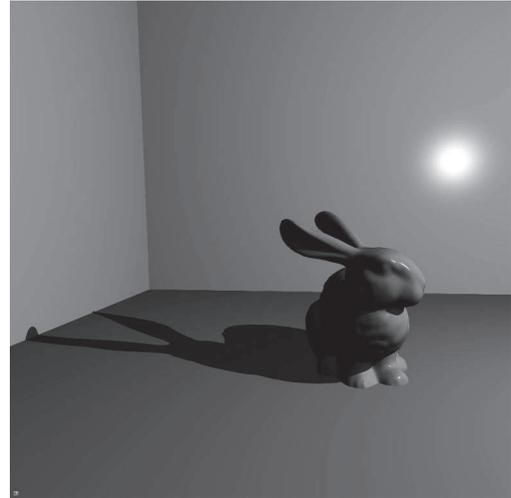


FIGURE 2.12 – Bunny - 69 k triangles - 8.43 Mo

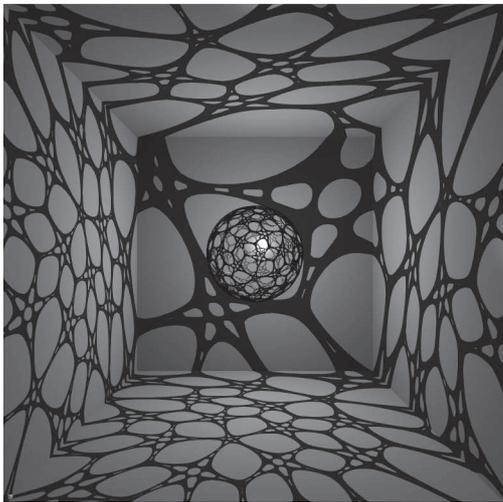


FIGURE 2.13 – Géodesic - 200 k triangles - 24.42 Mo



FIGURE 2.14 – Fairy - 174 k triangles - 21.24 Mo

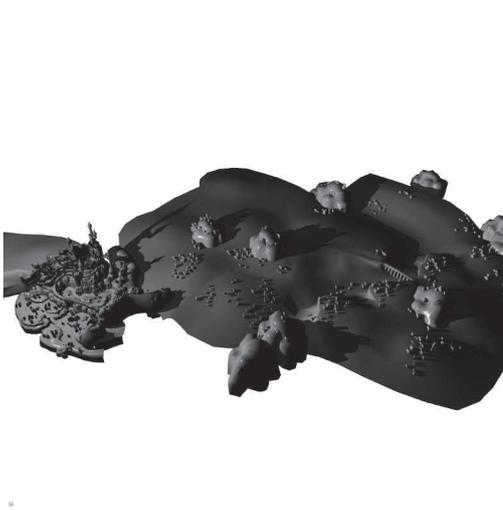


FIGURE 2.15 – Citadel issue de Unreal Engine de - 393 k triangles - 48 Mo



FIGURE 2.16 – Raptor - 1000 k triangles - 122 Mo

Tableau 2.6 – Résultats : la colonne *CS* indique le temps de construction de la partition de volumes d'ombre, et la colonne *FS* concerne le parcours de la structure par les fragments. Les lignes *No opt* , *No rand* et *No FFC* correspondent respectivement aux mesures sans l'optimisation minimisant la visite des sous-arbres intersection, sans aléa et sans élimination des triangles faisant face à la lumière.

	Version	Nœuds	GTX 980		
			CS	FS	Total
	<b>PSV</b>	16.48	0.18	1.85	<b>2.03</b>
	No opt	× 11.05	× 0.94	× 14.98	× 13.73
	No rand	× 1.06	× 0.83	× 1.0	× 0.99
	No FFC	× 1.41	× 1.67	× 1.12	× 1.17
	<b>PSV</b>	16.17	0.4	1.65	<b>2.05</b>
	No opt	× 10.15	× 0.95	× 14.91	× 12.19
	No rand	× 1.00	× 1.03	× 1.89	× 1.02
	No FFC	× 1.12	× 1.80	× 1.40	× 1.40
	<b>PSV</b>	23.93	1.24	2.90	<b>4.14</b>
	No opt	× 14.35	× 0.77	× 10.41	× 7.53
	No rand	× 1.38	× 1.01	× 1.34	× 1.24
	No FFC	× 1.44	× 1.37	× 1.37	× 1.37
	<b>PSV</b>	14.13	1.04	1.23	<b>2.27</b>
	No opt	× 21.15	× 0.97	× 49.93	× 24.25
	No rand	× 2.54	× 1.38	× 1.17	× 1.27
	No FFC	× 1.66	× 1.91	× 3.17	× 2.59
	<b>PSV</b>	68.02	3.00	6.4	<b>9.4</b>
	No opt	× 4.34	× 0.86	× 8.49	× 6.06
	No rand	× 1.21	× 0.95	× 1.18	× 1.10
	No FFC	× 1.17	× 1.57	× 1.30	× 1.39
	<b>PSV</b>	20.38	5.81	3.15	<b>8.96</b>
	No opt	× 14.64	× 0.90	× 16.72	× 6.46
	No rand	× 4.27	× 2.11	× 3.37	× 2.55
	No FFC	× 1.09	× 2.72	× 4.24	× 3.25

parcours prédomine sur le temps de construction ;

- le parcours systématique des sous-arbres intersection provoque l'explosion du nombre de nœuds à visiter, multiplié par un facteur allant de 4 à 21 selon les modèles et le temps de parcours ;
- il y a davantage de sous-arbres intersection à stocker dans la pile, donc la pile doit avoir une taille plus importante (64 pour les modèles testés contre 32 avec l'optimisation).

## Aléa

L'introduction d'aléa entraîne des effets variables selon le type de modèle : sur les modèles simples, son influence est inexistante ; mais sur le modèle du RAPTOR qui est finement tassé, elle apporte par une diminution d'un facteur 3.37 du nombre de nœuds à visiter et du temps de parcours ainsi qu'une construction 2 fois plus rapide.

## Suppression des faces avant par rapport à la lumière

On peut estimer que l'élimination des faces avant supprime, selon les points de vue, environ 50 % de la géométrie. Bien que le nombre de nœuds visités augmente peu (au maximum × 1.66), grâce au comportement logarithmique des partitions de volumes d'ombre, les temps de construction et de parcours s'accroissent : plus la scène contient de triangles, plus le surcoût devient important. L'augmentation du

nombre de triangle entraîne celle du nombre de nœuds au sein des partitions de volumes d'ombre, donc des accès mémoire pour la construction. Pour le parcours, même si le nombre de nœuds change peu, l'augmentation de la taille de la structure accélératrice va minimiser l'effet de la mémoire cache du GPU (la mémoire cache contient les informations récemment utilisées pour y permettre un accès plus rapide par la suite). Le cas de la scène TREX en est un très bon exemple : la partition de volumes d'ombre de 2.44 Mo qui est proche des 2 Mo de mémoire cache de la GTX 980, la suppression des faces avant par rapport à la lumière ne change presque pas le temps de parcours. Au contraire, sur les modèles de taille importante comme RAPTOR, le surcoût devient extrêmement lourd.

Maintenant que les premiers résultats ont été commentés, il est temps de s'intéresser au comportement des partitions de volumes d'ombre en fonction de la résolution avant de vérifier les complexités pratiques de la construction et du parcours.

### Comportement face à la résolution image

Le comportement face à la hausse de résolution est linéaire, comme le montre le graphique 2.18. Pour chaque modèle, la barre bleue représente le temps de rendu en  $1024 \times 1024$  comme valeur de base à 1. En multipliant par 4 le nombre de pixels à rendre en  $2048 \times 2048$ , le temps de calcul est un peu moins de 4 fois plus coûteux ; le passage à 16 millions de pixels en  $4096 \times 4096$  consomme 12 fois plus de temps qu'un rendu de 1 million de pixels. On peut remarquer que le coût diminue légèrement quand la quantité de pixels à traiter augmente. Cet effet peut s'expliquer par le fait que des pixels voisins ont des chances d'avoir des parcours très proches, ce qui améliore l'efficacité des mécanismes de mémoires caches du GPU. En densifiant la résolution, on se place dans des conditions qui favorisent ces mécanismes.

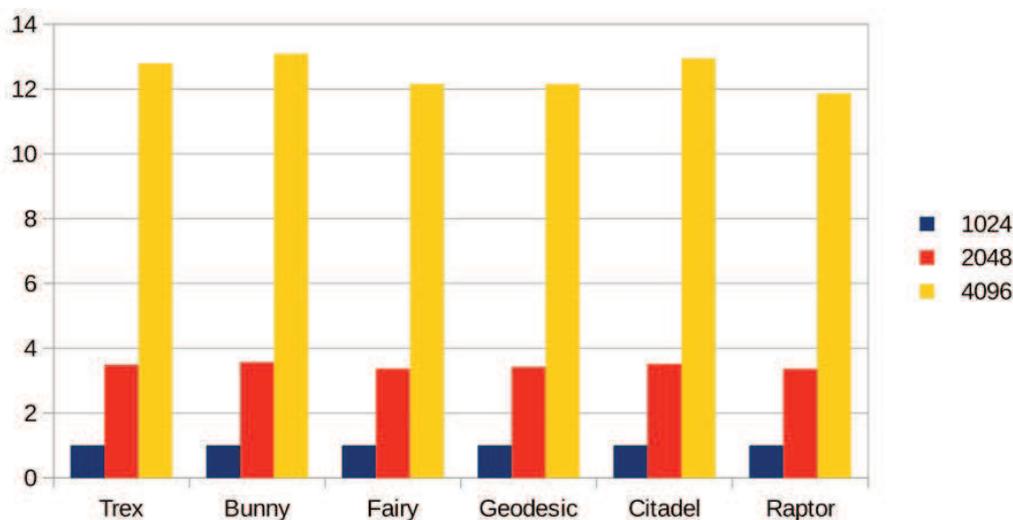


FIGURE 2.17 – Comportement rapporté à la résolution

Le comportement est quasiment linéaire par rapport au nombre de pixels à ombrer. On peut noter une légère diminution du coût sur les hautes résolutions, lié au mécanisme de cache.

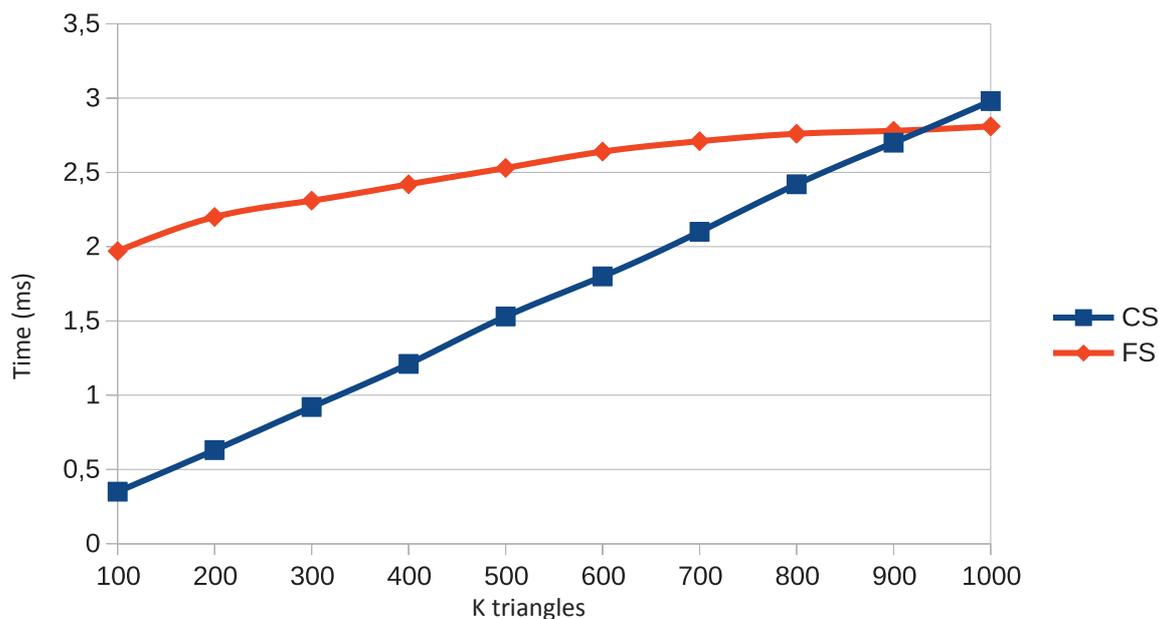


FIGURE 2.18 – Comportement en fonction de la complexité géométrique

Avec en bleu le temps de construction et en rouge le temps de parcours pour le RAPTOR décimé à différentes résolutions, de 100k à 1000k triangles.

### Complexité pratique d'une partition de volumes d'ombre par rapport à la complexité géométrique

Pour mesurer la complexité pratique, il faut utiliser des scènes dont l'ombre aura une complexité comparable : le modèle RAPTOR, dont la géométrie a été décimée pour avoir une plage de 10 modèles allant de 100k à 1000k triangles, permet d'obtenir le graphique 2.18. Conformément aux attentes, on peut constater que la complexité du parcours est en  $O(\log(n))$ , et que la complexité de la construction est quasiment linéaire, en  $O(n \log(n))$ .

### Comparaison avec l'état de l'art

Lors de la publication des partitions de volume d'ombre [42], la seule méthode exacte au pixel près et capable de gérer des scènes avec plusieurs centaines de milliers de triangles en temps réel était la technique d'Erik Sintorn [29] considérée dans ce domaine comme la plus rapide en 2014. La méthode de Sintorn et les partitions de volumes d'ombre ont une approche duale de la problématique des volumes d'ombre :

- Sintorn construit une structure accélératrice sur les points  $p$  de l'image, puis il fait traverser les  $n$  triangles de la scène dans sa structure ;
- les partitions de volumes d'ombre construisent une structure accélératrice avec les volumes d'ombre des  $n$  triangles de la scène, puis les  $p$  points de l'image  $y$  sont localisés.

Dans ces 2 méthodes, c'est le temps de traversée des structures accélératrices qui prédomine. Pour  $p$  pixels et  $n$  triangles, Sintorn doit mener  $n$  interrogations de structure en  $O(\log(p))$ , alors que les partitions de volumes d'ombre impliquent  $p$  interrogations en  $O(\log(n))$ . Par comparaison indirecte, à partir des données de Sintorn [29], en utilisant les mêmes modèles, le même GPU (une Nvidia Titan) et la même

résolution de  $1024 \times 1024$  on obtient les résultats suivant, publiés dans [42] :

- sur le modèle SMALL CITADEL de 60K triangles, la partition de volumes d'ombre est 10 fois plus lente ;
- sur le modèle NEW VILLA de 88K triangles, la partition de volumes d'ombre est plus lente d'un facteur de 3 à 5 ;
- sur le modèle FUZZY de 400K triangle, la partition de volumes d'ombre est un peu plus lente d'un facteur de 1,3 à 1,5.



FIGURE 2.19 – CITADEL



FIGURE 2.20 – NEW VILLA



FIGURE 2.21 – FUZZY

Une erreur de mesure a faussé les tests de comparaison avec la méthode de Sintorn lors de la publication de l'article : la pile utilisée avait une taille de 64 entiers, ce qui a été défavorable aux partitions de volumes d'ombre. Avec une pile de 32 entiers, sur une NVIDIA GTX 980 dont les performances sont proches de celles de la NVIDIA Titan utilisée par Sintorn, les résultats sont bien meilleurs :

- sur le modèle SMALL CITADEL de 60K triangles, la partition de volume d'ombre est plus lente d'un facteur de 2 à 5 ;
- sur le modèle NEW VILLA de 88K triangles, la partition de volume d'ombre est plus lente d'un facteur de 1,25 à 1,7 ;
- sur le modèle FUZZY de 400K triangles, la partition de volume d'ombre est légèrement plus rapide.

Ces résultats illustrent bien les différences de complexité entre les méthodes : pour une résolution image fixe, l'augmentation de la géométrie favorise les partitions de volumes d'ombre et défavorise l'approche de Sintorn.

L'approche de Sintorn est systématiquement plus consommatrice en mémoire, soit 198 Mo pour une résolution de  $1024 \times 1024$ , alors que les partitions de volumes d'ombre requièrent 7,33 Mo (CITADEL), 10,74 Mo (VILLA) et 48,8 Mo (FUZZY) d'après [29]. Moyennant un léger surcoût d'environ 5 %, il est possible de diminuer la consommation mémoire de la technique de Sintorn pour la contenir dans un intervalle de 33 à 58 Mo.

### Ombres translucides

Les figures 2.21 à 2.24 montrent les résultats obtenus une fois la transparence activée et un rendu avec une couleur par triangle afin d'illustrer la possibilité d'utiliser une texture alpha pour définir le facteur de transparence. Les divers temps de rendu sont présentés dans le tableau 2.7. Pour obtenir ces résultats il est nécessaire d'utiliser une pile de 64 entiers, car il faut visiter tous les volumes d'ombre avant de terminer la requête : il y a davantage de sous-arbres intersection à stocker dans la pile.

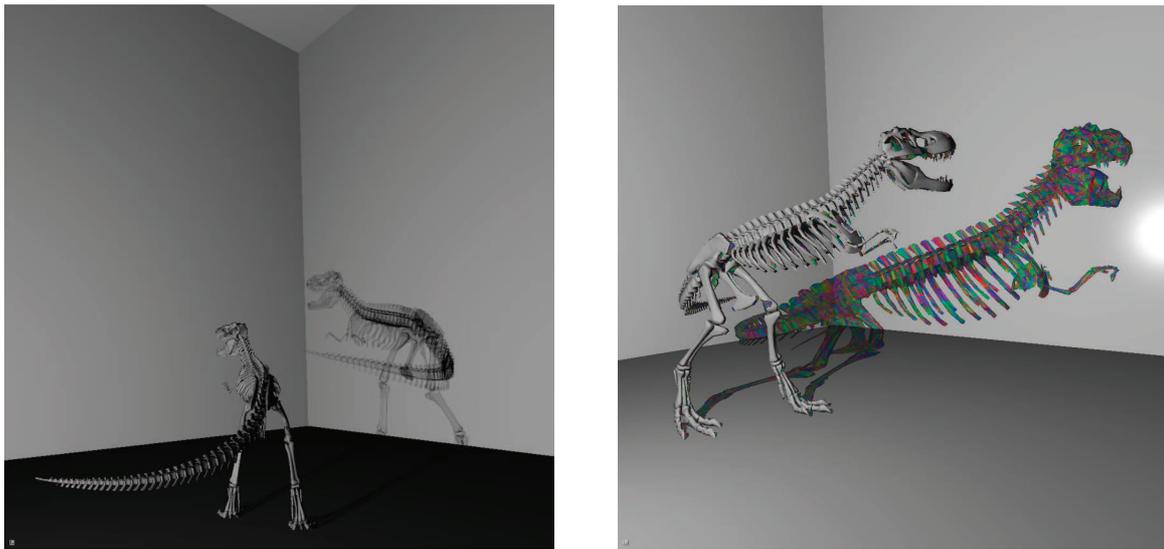


FIGURE 2.22 – TREX : 3.39 ms (construction 0.30 ms et parcours 3.09 ms)

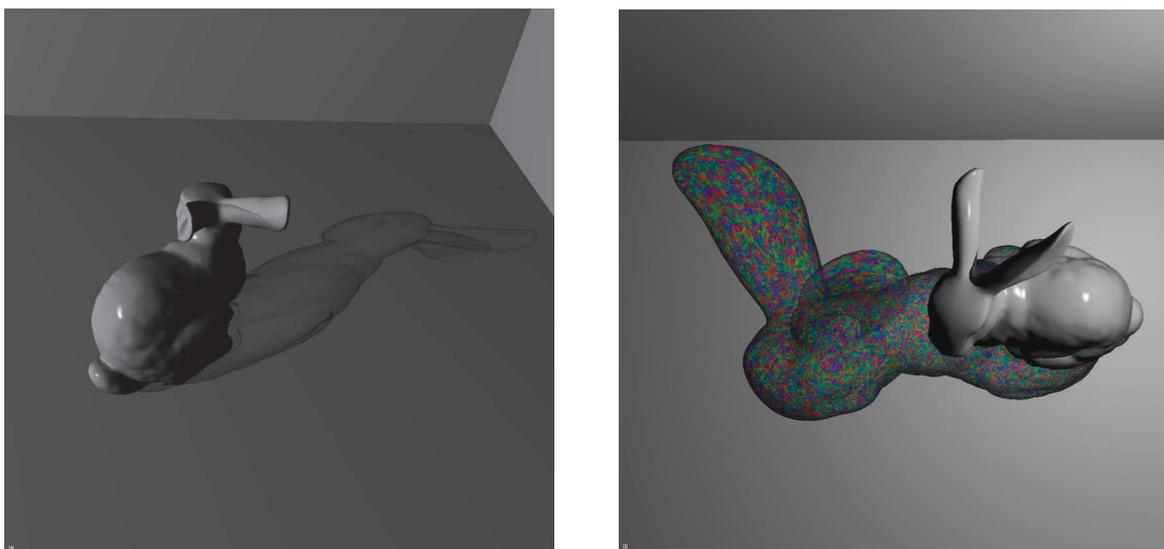


FIGURE 2.23 – BUNNY : 3.67 ms (construction 0.72 ms et parcours 2.95 ms)

Tableau 2.7 – Résultats : en vert pour les ombres transparentes, avec le nombre de nœuds moyen, le temps de construction *CS*, le temps de parcours *FS* et le temps total. À titre de comparaison, on rappelle en bleu les résultats du calcul de l'ombre dure sur les mêmes scènes et mêmes parcours de 1000 images.

	Ombre translucide				Ombre dure			
	Nœuds	CS	FS	Total	Nœuds	CS	FS	Total
Trex	23.25	0.30	3.09	3.39	16.48	0.18	1.85	2.03
Bunny	22.95	0.72	2.95	3.67	16.17	0.4	1.65	2.05
Geodesic	25.63	1.99	6.42	8.41	14,13	1.04	2.90	4.14
Raptor	28.67	15.78	18.12	33.90	20.28	5.81	3.15	8.96

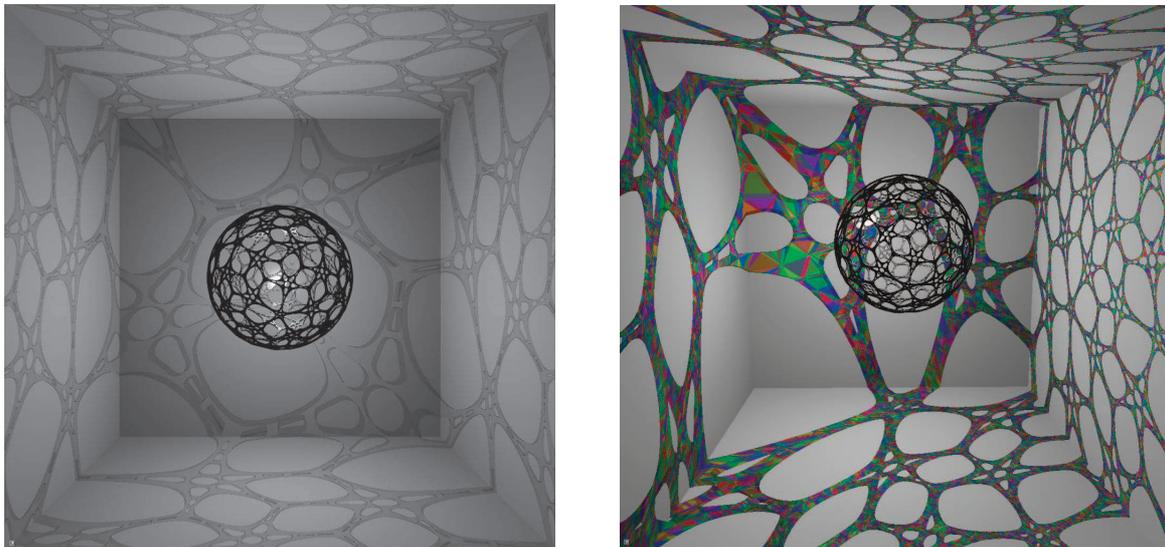


FIGURE 2.24 – GEODESIC : 8.41 ms (construction 1.99 ms et parcours 6.42 ms)

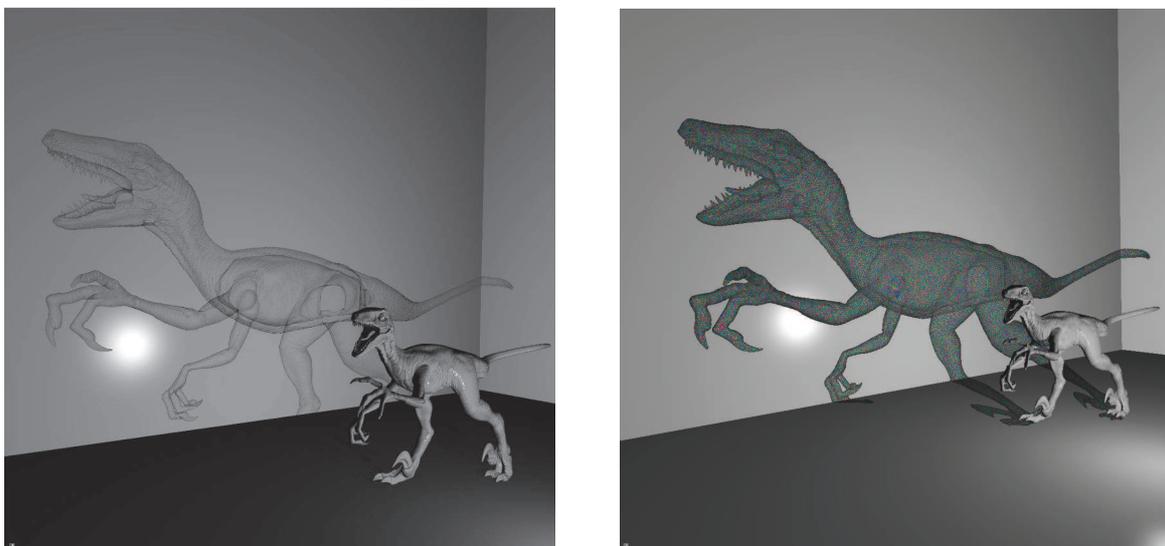


FIGURE 2.25 – RAPTOR : 33.90 ms (construction 15.78 ms et parcours 18.12 ms)

### Analyse des ombres transparentes

Les résultats du tableau 2.7 permettent de montrer l'efficacité des partitions de volumes d'ombre : alors que la quantité de géométrie dans la structure est environ le double de ce qui est nécessaire à la gestion des ombres dures, et qu'il faille trouver potentiellement plusieurs volumes d'ombre par pixel, le nombre de nœuds à visiter augmente d'un facteur 1,4 sur tous les modèles (sauf GEODESIC où le facteur est 1,8). Le temps de construction est plus ou moins doublé à cause de la géométrie supplémentaire, sauf sur RAPTOR où la construction est 3 fois plus lente. Quant au temps de parcours, il augmente d'un facteur allant de 1.7 sur TREX à 5.5 sur RAPTOR ce qui s'explique d'une part par la hausse de la taille de pile utilisé, et d'autre part par la diminution de l'efficacité du système de cache mémoire lié a des partitions contenant plus de géométrie que leur équivalent en ombre dure.

### Bilan de la méthode

- **L'exactitude** : exact par pixel ;
- **Le comportement face à la complexité géométrique** : le parcours en  $O(\log(n))$  permet de bien gérer la hausse en complexité géométrique, tant que le temps de construction ne devient pas prohibitif ;
- **Le comportement face à une hausse de la résolution image** : le comportement est quasiment linéaire tout en étant légèrement favorable ;
- **Le coût en mémoire** : linéaire par rapport au nombre de triangle ;
- **Le type d'ombrage possible** : omnidirectionnel ;
- **Le paramétrage** : il faut bien choisir la taille de la pile, car une grosse pile diminue les performances tandis qu'une petite peut provoquer des blocages ;
- **La stabilité** : temps de rendu variable selon les points de vue.

## 2.4 Conclusion

La partition ternaire des objets permet de s'affranchir des imprécisions numériques inhérentes aux opérations CSG utilisées par les SVBSP. À partir de cette nouvelle structure de données qui évite toute réplication d'information, on peut construire et parcourir, entièrement sur GPU grâce à une empreinte mémoire prévisible, un arbre ternaire basé sur la géométrie des volumes d'ombre de la scène. Face à la complexité géométrique, cette nouvelle technique bénéficie d'un parcours logarithmique qui lui permet d'être proche de l'état de l'art sur des scènes complexes allant jusqu'au million de triangles. Il faut cependant noter une stabilité relative des performances selon les points de vue. Le choix de la taille de la pile utilisée pour le parcours de la structure est un paramètre important qui va influencer les performances ; une taille inadaptée provoquera des erreurs ou des blocages, et malheureusement il n'est pas possible de déterminer par avance la taille optimale. Le fait que les performances des partitions de volumes d'ombre soient déjà comparables avec le meilleurs de l'état de l'art montre que la voie originale initiée par les SVBSP mérite d'être approfondie. Comme les partitions de volumes d'ombre sont limitées par la bande passante mémoire du GPU, l'amélioration de la méthode nécessite d'envisager de minimiser de nombre de nœuds et envisager des parcours différents qui permettraient de s'affranchir du paramétrage de la pile. Il faudra aussi s'intéresser à la stabilité des performances et aux façons de les améliorer.

# Chapitre 3 .

## Analyse et optimisations

Dans ce troisième chapitre, on va chercher à améliorer la stabilité et l'efficacité des partitions de volumes d'ombre. Dans un premier temps, on se focalisera sur l'analyse du parcours de la structure, car c'est la phase la plus coûteuse de la méthode. Puis on cherchera une solution à la problématique de la pile nécessaire au parcours de la structure. Ces travaux ont été publiés [43]

### 3.1 Analyse des partitions de volumes d'ombre

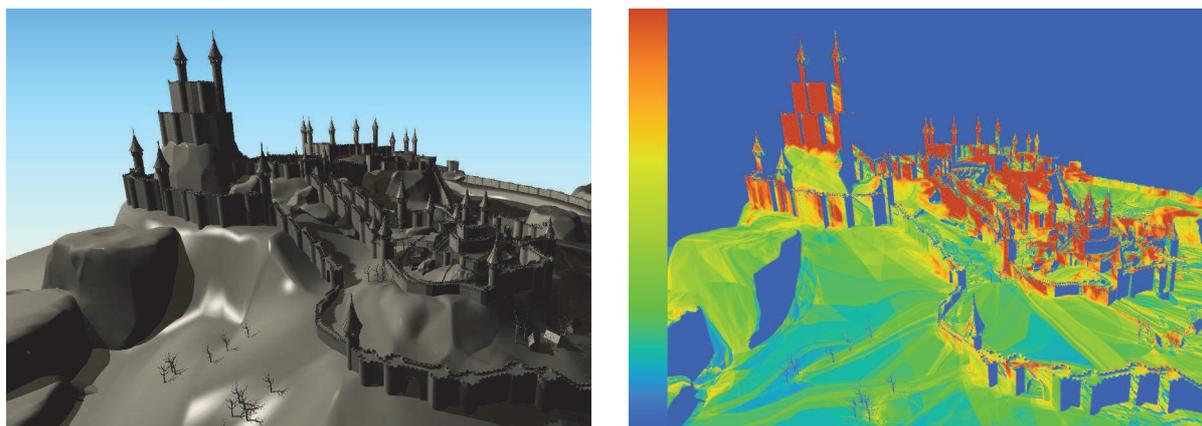


FIGURE 3.1 – Coût en noeuds

À gauche, le modèle CLOSED CITADEL composé de 393 k triangles ; à droite, on peut voir sous forme de carte de couleurs le nombre de noeuds à visiter pour déterminer l'ombrage de la scène : du bleu pour 0 noeud au rouge lorsque l'on dépasse les 250 noeuds.

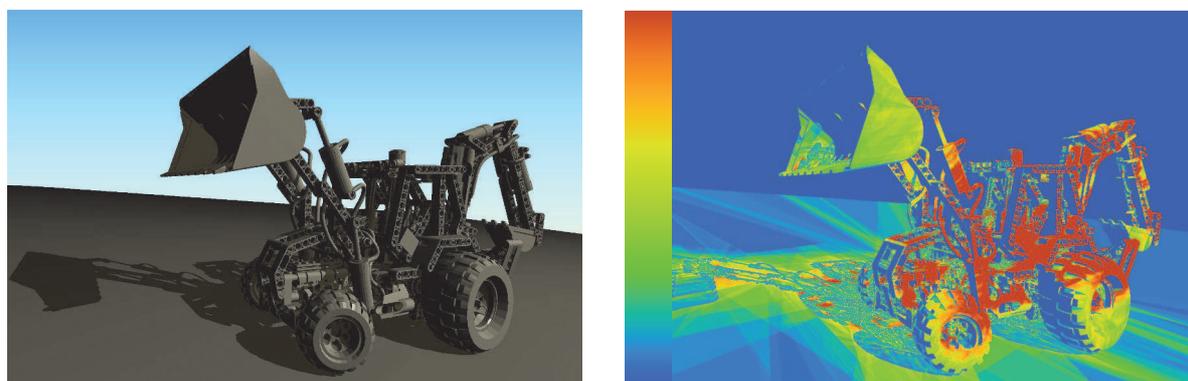


FIGURE 3.2 – Coût en noeuds 2

À gauche, le modèle ESCAVATOR composé de 1,1 M triangles ; à droite, on peut voir que les zones rouges coûteuses en noeuds sont éclairées et les plus proches de la lumière. L'échelle va du bleu pour 0 noeud au rouge lorsque l'on dépasse les 250 noeuds

Les résultats du tableau 2.6 du chapitre précédent montrent que le temps de construction des partitions de volumes d'ombre est inférieur au temps de parcours sur toutes les scènes sauf sur RAPTOR. C'est donc le parcours qui présente le coût prédominant, surtout si l'on utilise des définitions d'images courantes ( $1920 \times 1080$ ) qui comptent 2 fois plus de pixels que la résolution utilisée pour les mesures. On va donc analyser les performances des parcours et l'efficacité de l'optimisation angulaire, afin de repérer

les situations défavorables aux partitions de volumes d'ombre en se basant sur les résultats du second chapitre. Comme le coût et le temps du parcours dépendent du nombre de noeuds visités, on peut voir l'efficacité de l'optimisation angulaire, sur le modèle GÉODESIC avec la source lumineuse en son centre, qui divise par quasiment 50 le temps de parcours et par 21 le nombre de noeuds à visiter.

Toutefois, sur une scène comme CITADEL le gain de l'optimisation angulaire est beaucoup plus faible, avec seulement un facteur de l'ordre de 15 pour le temps de parcours et de 4 pour le nombre de noeuds. Cette différence s'explique en comparant la répartition de la géométrie par rapport à la lumière : GÉODESIC présente une géométrie distribuée uniquement de manière directionnelle, l'ensemble des triangles étant répartis sur une sphère. Dans ce cas, la partition de volumes d'ombre se montre très efficace puisque 3 noeuds sur 4 opèrent un tri suivant des plans passant par la lumière. Dans le cas de CITADEL, le problème est différent car le modèle s'étale en longueur avec des densités de géométrie très variables ; ainsi, du point de vue de la lumière une part importante de la géométrie est répartie dans une même direction et sur une importante profondeur. Le tri directionnel de la partition de volumes d'ombre se montre moins efficace dans ce cas, et le nombre de noeuds à visiter peut devenir très important comme on peut le voir sur la figure 3.1 : les zones à forte densité de géométrie en profondeur sont nettement visibles en rouge. On peut aussi remarquer une fait surprenant sur la figure 3.2 : les zones les plus coûteuses en noeuds correspondent très souvent à des zones éclairées. Là encore, ce sont des zones denses en géométrie le long de la demie droite issue de la lumière et passant par le point ; L'exemple de la figure 3.3 illustre ce problème. Pour limiter cette perte d'efficacité, il faut modifier notre structure afin de tenir compte de la notion de profondeur depuis la source lumineuse.

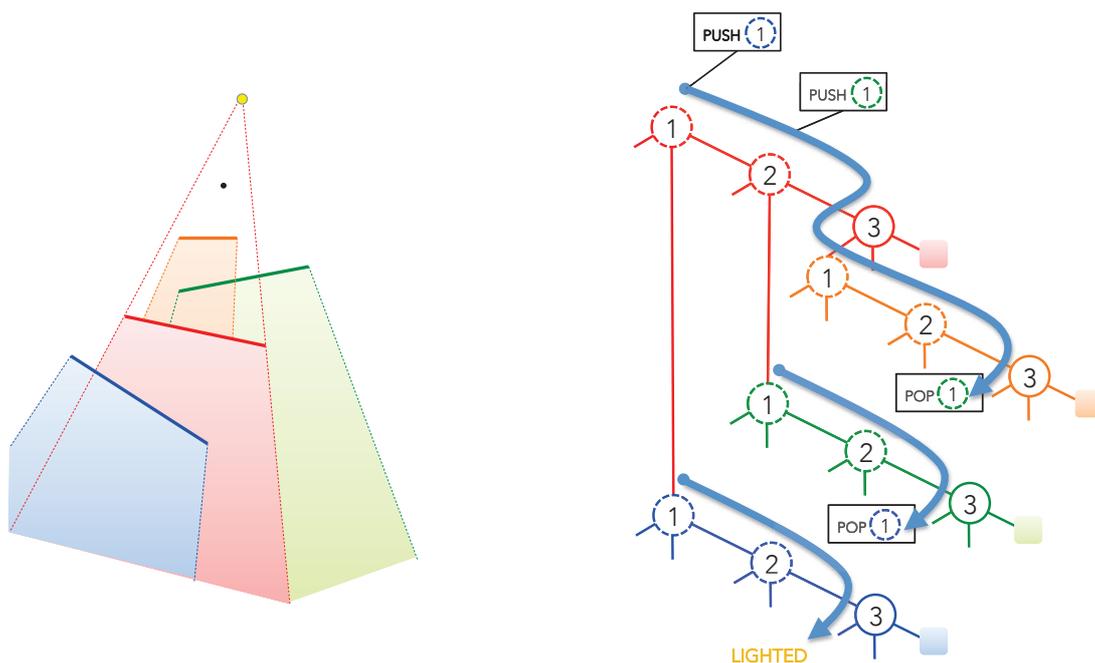


FIGURE 3.3 – Traversée d'un point illuminé

Ici un point éclairé qui doit être localisé dans la partition de volumes d'ombre. Les surfaces sont réparties sur des profondeurs différentes et leurs volumes d'ombre se superposent en partie. Dans cet exemple, l'optimisation angulaire est inefficace et le point doit visiter tous les noeuds de l'arbre alors même qu'il est beaucoup plus proche de la lumière que chacune des surfaces et donc forcément hors de l'ombre.

### 3.2 Prise en compte de la profondeur par rapport à la lumière

Pour remédier au problème analysé précédemment, il faut ajouter une information de profondeur à la partition de volumes d'ombre. L'idée est simple : on ajoute pour chaque nœud correspondant à un nouveau volume d'ombre une information sur la distance minimale entre la lumière et tous les volumes d'ombre présents dans ses fils. Ainsi, avant qu'un point commence à traverser un volume d'ombre, on peut comparer la distance point-lumière à celle qui est stockée dans le sous-arbre : si le point est plus proche de la lumière que la géométrie du sous-arbre courant, il n'est pas nécessaire de visiter cette portion de la partition de volumes (cf. figure 3.4). La figure 3.5 illustre la réduction du nombre de noeuds à visiter dans les zones qui étaient les plus coûteuses. Cette modification implique :

- une légère retouche de la construction, comme sur l'algorithme 5 : chaque fois qu'un triangle descend dans la partition de volume d'ombre et qu'il passe par le premier noeud d'un volume d'ombre, on met à jour la distance stockée si la distance lumière-triangle est inférieure à la distance déjà présente ;
- une modification du parcours, comme sur l'algorithme 6 : à chaque fois qu'on entre dans un volume d'ombre, si la distance point-lumière est inférieure à la distance contenue dans volume d'ombre, il n'est pas nécessaire de visiter les sous-arbres correspondants.

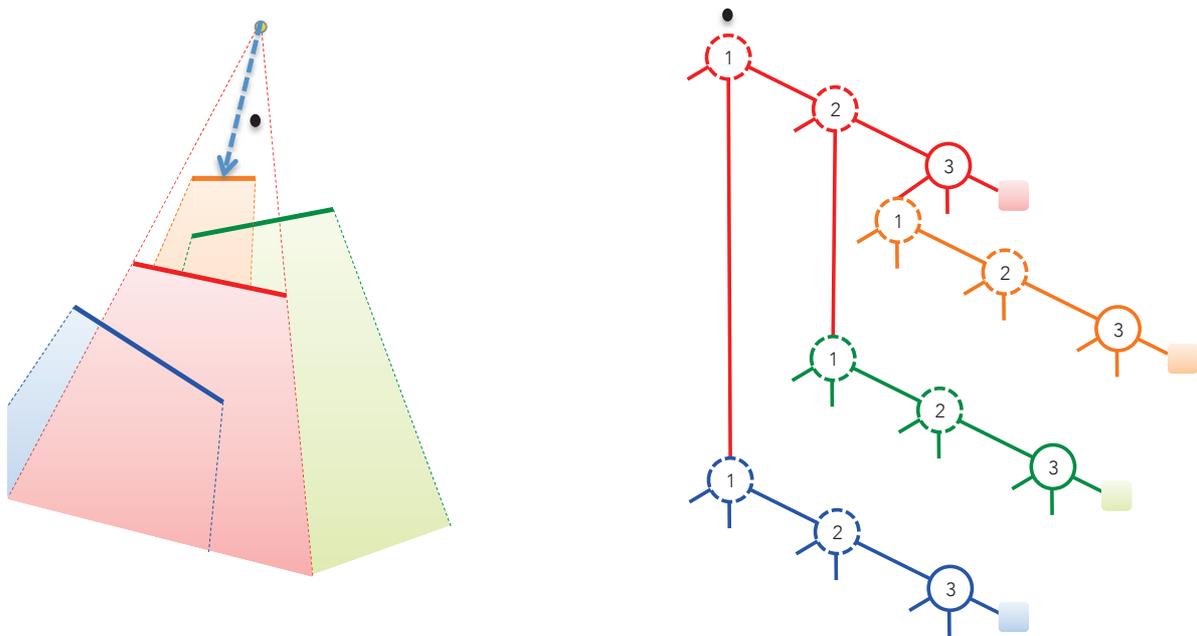


FIGURE 3.4 – Un parcours avec distance

Le point est plus proche de la lumière que la géométrie contenue dans l'arbre ternaire. Dès le passage dans le premier noeud rouge, on peut conclure qu'il est inutile de visiter l'arbre, et donc que le point est illuminé. Dans ce cas on passe d'une traversée nécessitant de visiter 12 noeuds (cf. figure 3.3) à une simple comparaison de distance.

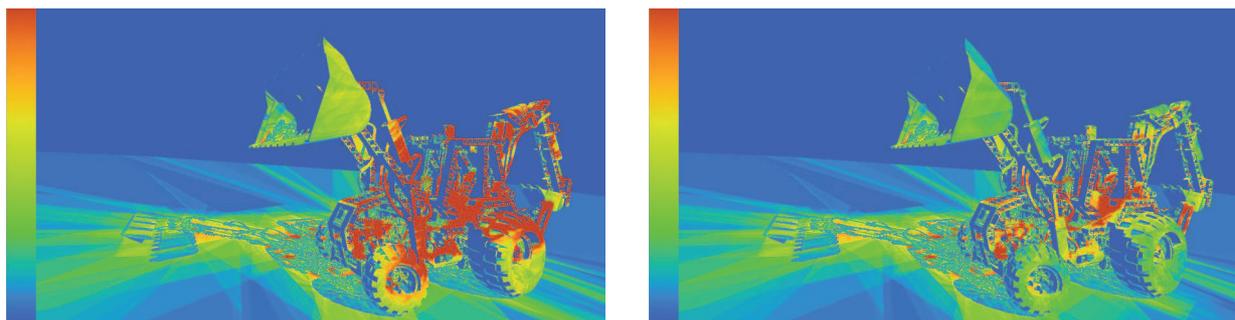


FIGURE 3.5 – Effet de l'optimisation de distance

À gauche, le nombre de noeuds sans l'optimisation de distance ; à droite, le nombre de noeud à visiter avec l'optimisation de distance : on peut voir une nette amélioration avec la diminution du nombre de noeuds à visiter, notamment pour les portions éclairées devant des zones denses en géométrie.

### 3.3 Un parcours sans pile

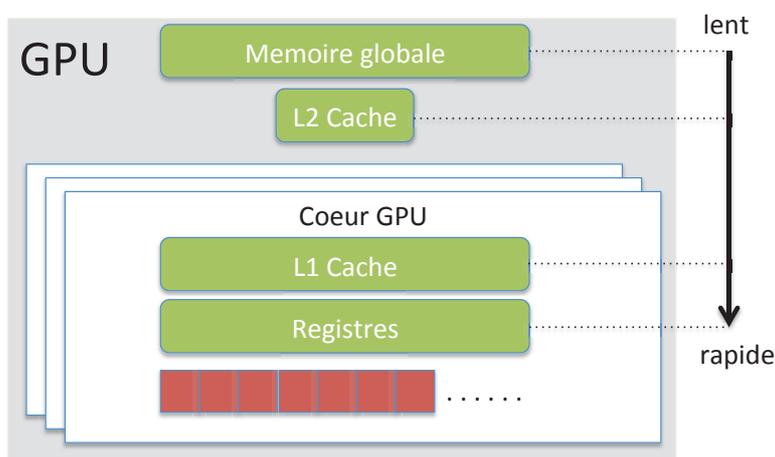


FIGURE 3.6 – Architecture d'un GPU

Avec en vert les différentes mémoires et en rouge les unités de calcul.

L'utilisation d'une pile dans le parcours d'une partition de volumes d'ombre présente 2 difficultés. La première est la taille de pile, nécessaire pour éviter des blocages d'exécution ou des erreurs d'ombrage (selon le comportement du GPU). La seconde concerne l'exécution d'un processus sur GPU, visible sur la figure 3.6 : les unités d'exécution disposent, par groupe de 32, d'une mémoire très rapide sous forme de registres en quantité très limitée, (255 registres pour les GPU les plus récents) ainsi que d'une mémoire dite locale qui est stockée dans la mémoire globale du GPU. Un processus utilise préférentiellement les registres qui ont des accès mémoire 10 fois plus rapides que la mémoire locale. Lorsque tous les registres sont occupés, le processus doit décharger et charger les données entre les registres et la mémoire locale. Or les échanges avec elle sont pénalisants ; car contrairement à ce que sa dénomination laisse penser, cette mémoire locale est allouée dans la mémoire globale du GPU qui est la plus lente. Il faut donc minimiser autant que possible son utilisation. Visible sur le listing 3.1, une petite portion du code assembleur du fragment shader du parcours avec une pile de 32 entiers montre l'utilisation de 48 blocs de 32 bits de mémoire locale, et donc une forte pression sur les registres.

---

**Algorithm 5** Algorithme de fusion d'un volume d'ombre avec l'optimisation de distance. En rouge apparaissent les ajouts permettant de prendre en compte l'optimisation de distance : chaque fois qu'un volume d'ombre est inséré dans la partition de volumes d'ombre, on met si nécessaire, à jour la distance stockée dans chaque premier noeud des volumes d'ombre traversés.

---

```

1: Node {
2:   Plan plan
3:   Node positif, intersection, negatif
4:   flottant angle
5:   flottant distance //on stocke la distance minimale géométrie-lumière du sous arbre p
6: }
7: FusionVolumeOmbre(Node racine, Triangle t, lumière l)
8:   Node n ← racine
9:   while n n'est pas une feuille do
10:    if n est le premier noeud d'un shadow volume then
11:      MiseAJourDistancePointTriangle(n,t,l) //maj si elle est plus petite que celle de n
12:    end if
13:    flottant location ← position(n.plan,t)
14:    if location > 0 then
15:      n ← n.positif
16:    else if location < 0 then
17:      n ← n.negatif
18:    else
19:      if n.plane est issue d'une arête then
20:        MiseAJourAngleIntersection(n,t)
21:      end if
22:      n ← n.intersection
23:    end if
24:  end while
25: RemplacerFeuilleParVolumeOmbre(n,t,l)

```

---

```

....
SHORT TEMP HC;
TEMP lmem[48];
OUTPUT result_color0 = result.color;
....

```

Listing 3.1 – Extrait du code assembleur du fragment shader du parcours avec une pile de 32 entiers.

Une problématique similaire existe en lancer de rayon sur GPU, car une pile est nécessaire pour faire traverser une structure accélératrice à un rayon. Afin d'éviter cet écueil, dans le domaine du lancer de rayon, des approches de parcours de structures accélératrice sans pile sur GPU existent :

- pour les arbres BVH ou *Bounding volume hierarchy*, un arbre binaire basé sur des boîtes englobantes, Hapala et al. [39] proposent d'ajouter dans chaque noeud un lien vers son parent afin de pouvoir remonter dans l'arbre ;
- encore pour les arbres BVH, Laine et al. [40] utilisent une approche hybride avec une petite pile : lorsque la pile est remplie, le rayon va recommencer à traverser la structure accélératrice et redémarrer un parcours avec une pile vide, non depuis la racine de l'arbre mais depuis un noeud marqué précédemment ;
- pour des arbres binaires, Barringer et al. [38] emploient un tableau de bits afin de garder en mémoire la trace du parcours effectué, en utilisant un 0 pour indiquer le passage d'un fils positif et un 1 pour celui négatif.

---

**Algorithm 6** Interrogation d'un arbre ternaire de volume d'ombre avec l'optimisation de distance. En rouge apparaissent les lignes spécifiques à l'optimisation de distance.

---

```

1: Interrogation(Node racine, Point p, Lumière l)
2:   PileDeNoeud pile
3:   Empiler(pile, racine)
4:   While pile n'est pas vide do
5:     Node b ← ViderUnElement(pile)
6:     if n est le premier noeud d'un Shadow Volume et n.distance > Distance(p,l) then
7:       SupprimerDernierElement(pile)
8:       Continue
9:     end if
10:    if n n'est pas une feuille then
11:      Entier location ← signe(n.plan,p)
12:      if n.plan est plan d'ombre then
13:        flottant pAngle ← CalculAngle(n.plan,p)
14:        if pAngle < n.angle then
15:          Empiler(pile, n.intersection)
16:        endif
17:        Empiler(pile, location>0 ? n.positif : n.negatif)
18:      else
19:        if location<0 then
20:          Retourner(0)
21:        end if
22:        Ajouter(pile, n.intersection)
23:        Ajouter(pile, n.positif)
24:      end if
25:    end if
26:  end while
27: Retourner(1)

```

---

Les deux dernières techniques ne s'appliquent pas aux partitions de volumes d'ombre qui sont des structures ternaires et non binaires, car l'utilisation d'un masque de bits n'est pas applicable avec une structure ternaire. De plus, ces techniques cherchent à connaître la distance de l'intersection la plus proche, alors que dans les partitions de volumes d'ombre ; on veut seulement savoir s'il y a une intersection. L'idée d'Hapala [39] d'ajouter des liens père fils entre chaque noeud peut se transposer aux partitions de volumes d'ombre.

Mais en pratique (comme sur la figure 3.7) cette approche est inefficace car elle conduit à visiter et revisiter de trop nombreux noeuds en générant un trafic mémoire supplémentaire pénalisant : le parcours se fait de noeud en noeud aussi bien en descendant qu'en montant dans l'arbre, donc de nombreux noeuds sont visités à plusieurs reprises.

Dans le cas d'une partition ternaire, on peut s'inspirer de l'approche d'Hapala : en se déplaçant non pas de noeuds en noeuds, mais directement en sautant au noeud parent des sous-arbres intersection. Cette modification implique de changer l'ordre de parcours par rapport à l'approche originelle, en visitant en premier les fils intersections pour traiter ensuite les fils positif et négatif (cf. figure 3.8). Une fois qu'un sous-arbre intersection a été visité, il faut pouvoir revenir au noeud père de ce sous-arbre afin de pouvoir continuer l'exploration d'un de ses fils positif ou négatif. Pour y arriver, il faut stocker dans tous les fils positifs vides d'un sous-arbre intersection l'indice de son noeud parent. Avec la nouvelle construction (cf. algorithme 7) et le parcours (cf. algorithme 8), on peut supprimer la pile du parcours. En regardant la figure 3.9, on peut remarquer sur l'image de droite que la version des partitions d'ombre sans pile

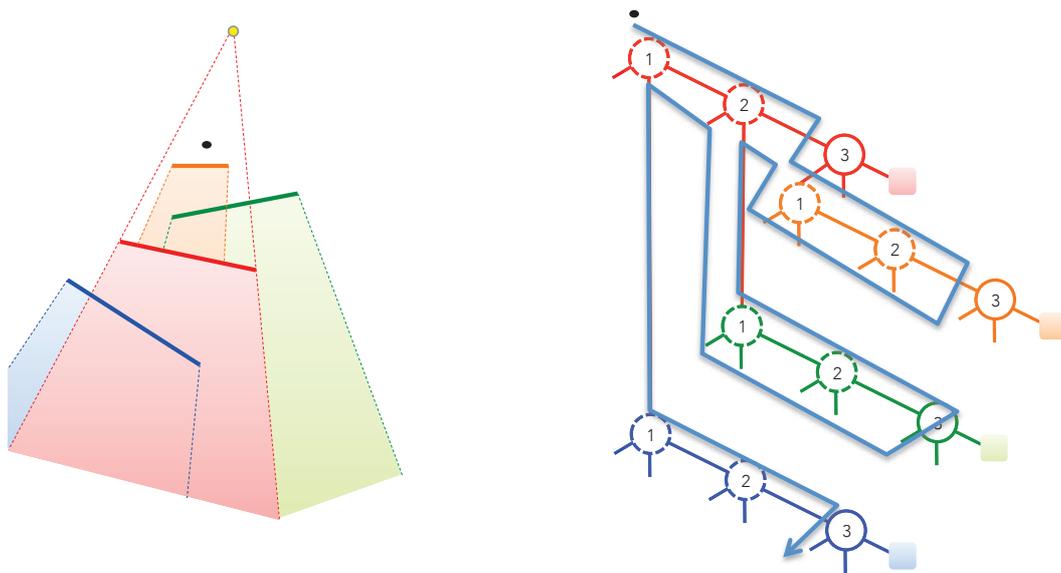


FIGURE 3.7 – Parcours sans pile

Avec un parcours sans pile en utilisant des liens père-fils entre chaque noeud, le nombre de noeuds à visiter explose : il faut visiter 20 noeuds là où la méthode de base des partitions d'ombre n'en nécessite que 12. On a même le cas pour le second noeud rouge, qui est visité 3 fois lors du parcours.

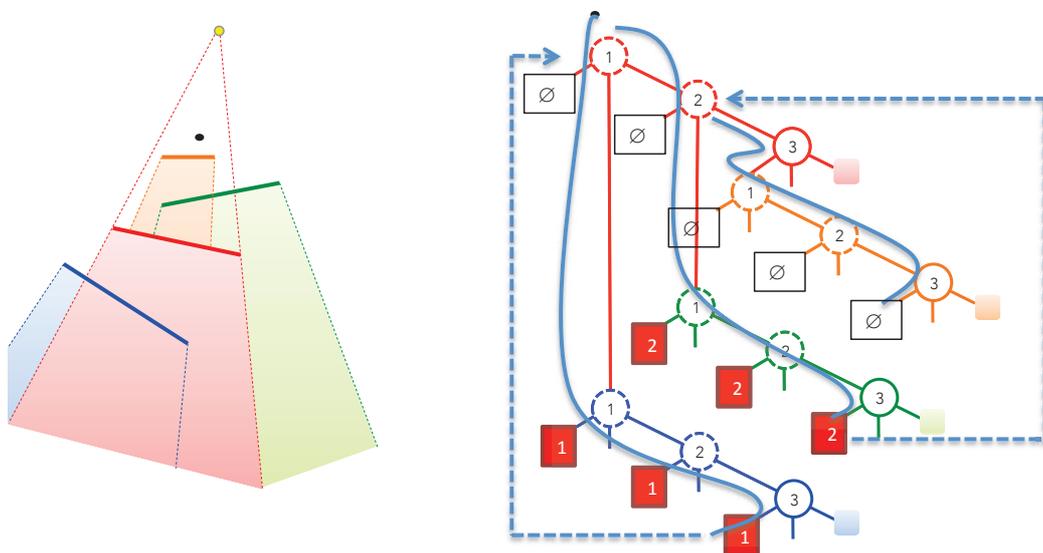


FIGURE 3.8 – Un parcours sans pile

On part de la racine pour aller directement visiter le volume d'ombre bleu. Le point arrive dans la feuille positive du troisième noeud bleu. Là un lien vers le noeud 1 rouge reprend la visite du volume d'ombre rouge. Là encore on va visiter en premier le fils intersection du deuxième noeud rouge et tester le volume d'ombre vert. En fin du volume d'ombre vert, on peut remonter au second noeud rouge pour finir de visiter les volumes d'ombre rouge et orange. Cette approche fait 14 noeuds.

demande de visiter davantage de noeuds que la méthode de base, malgré l'optimisation de distance : la pile n'est plus nécessaire, ce qui permet de faire diminuer la pression sur les registres lors du parcours de la structure : en revanche on visite plus de noeud car chaque noeud ayant un fils intersection sera visité 2 fois.

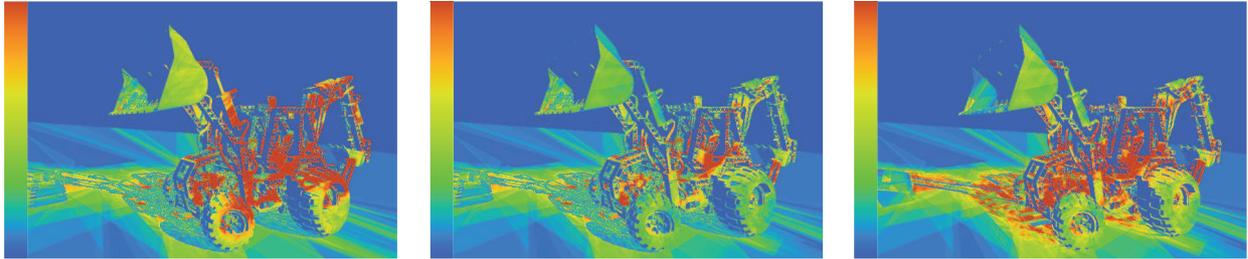


FIGURE 3.9 – Comparaison des différents parcours

De gauche à droite, on peut voir le nombre de noeuds visités pour les partitions de volumes d'ombre originelles, la version avec l'optimisation de profondeur et enfin la version avec parcours sans pile et avec profondeur.

**Algorithm 7** Algorithme de fusion d'un volume d'ombre avec l'optimisation de distance pour un parcours sans pile. En rouge apparaissent les ajouts permettant de prendre en compte l'optimisation de distance : chaque fois qu'un volume d'ombre est inséré dans la partition de volumes d'ombre, on met si nécessaire, à jour la distance stockée dans chaque premier noeud des volumes d'ombre traversés. Les modifications permettant de garder un lien vers le sous arbre-intersection père le plus proche sont en vert.

---

```
1: Node {
2:   Plan plan
3:   Node positif, intersection, negatif
4:   flottant angle
5:   flottant distance //on stocke la distance minimale géométrie-lumière du sous arbre p
6: }
7: FusionVolumeOmbre(Node racine, Triangle t, lumière l)
8: Node n ← racine
9: //on va garder un lien vers le plus proche ancêtre intersection qui est initialiser à rien
10: Node ancetre ← null
11: while n n'est pas une feuille do
12:   if n est le premier noeud d'un shadow volume then
13:     MiseAJourDistancePointTriangle(n, t, l) //maj si elle est plus petite que celle de n
14:   end if
15:   flottant location ← position(n.plan, t)
16:   if location > 0 then
17:     n ← n.positif
18:   else if location < 0 then
19:     n ← n.negatif
20:   else
21:     if n.plane est issue d'une arête then
22:       MiseAJourAngleIntersection(n, t)
23:     end if
24:     n ← n.intersection
25:     //le triangle va passer dans un fils intersection, il faut donc mettre à jour le lien
26:     ancetre ← n.intersection
27:   end if
28: end while
29: //Lors de l'insertion du triangle, chacune des feuilles positives va stocker le lien vers son ancêtre
   intersection
30: RemplacerFeuilleParVolumeOmbre(n, t, l, ancetre)
```

---

---

**Algorithm 8** Interrogation d'un arbre ternaire de volume d'ombre avec l'optimisation de distance pour un parcours sans pile. Les modifications pour prendre en compte l'optimisation de distance apparaissent en rouge : le test de la ligne 9 permet d'éviter de visiter le volume d'ombre courant et ses sous-arbres si la géométrie de cette portion de la partition de volumes d'ombre est plus éloignée de la lumière que le point à ombrer.

---

```

1: InterrogationPourPoint(Node racine, Point p, Lumière l)
2: // un booléen pour savoir si c'est la première fois qu'on visite un noeud
3: Booleen fait ← faux
4: // le calcul de la distance lumière point pour l'optimisation de distance
5: Flottant distance ← Distance(p, l)
6: Node node ← racine
7: While node ≠ null do
8: // optimisation de la distance pour éviter de visiter inutilement un sous-arbre
9:   if node est le premier noeud du volume d'ombre courant et distance < node.distance then
10: // p est plus proche de la lumière que la géométrie du sous-arbre
11: // il faut maintenant remonter au noeud intersection parent
12:   node ← node.ancetre
13:   fait ← vrai
14:   Continue
15: end if
16: // On visite le fils intersection en premier en vérifiant que p est bien compris dans la zone angulaire
    du fils intersection
17:   flottant pAngle ← CalculAngle(node.plan, p)
18:   if twice = faux et node.inter n'est pas une feuille et pAngle < node.angle then
19: // On visite le fils intersection
20:   node ← node.inter
21:   Continue
22: end if
23: // Ici le fils intersection a déjà été visité, on peut parcourir l'arbre comme un BSP
24:   fait ← faux
25:   Entier location ← signe(node.plan, p)
26:   if location > 0 then
27:     if node.pos est une feuille then
28: // La feuille positive est vide, il faut donc remonter vers l'ancêtre intersection
29:       node ← node.ancetre
30:       fait ← vrai
31:     else
32: // on visite le fils positif
33:       node ← node.pos
34:     end if
35:   else
36: // Si la feuille négative est vide, c'est qu'on se trouve dans un volume d'ombre, sinon on continue le
    parcours
37:     if node.neg est une feuille then
38:       Retourner(0)
39:     else
40:       node ← node.neg
41:     end if
42:   end if
43: end while
44: // p est hors de tout les volumes d'ombre
45: Retourner(1)

```

---

### 3.4 Parcours hybride

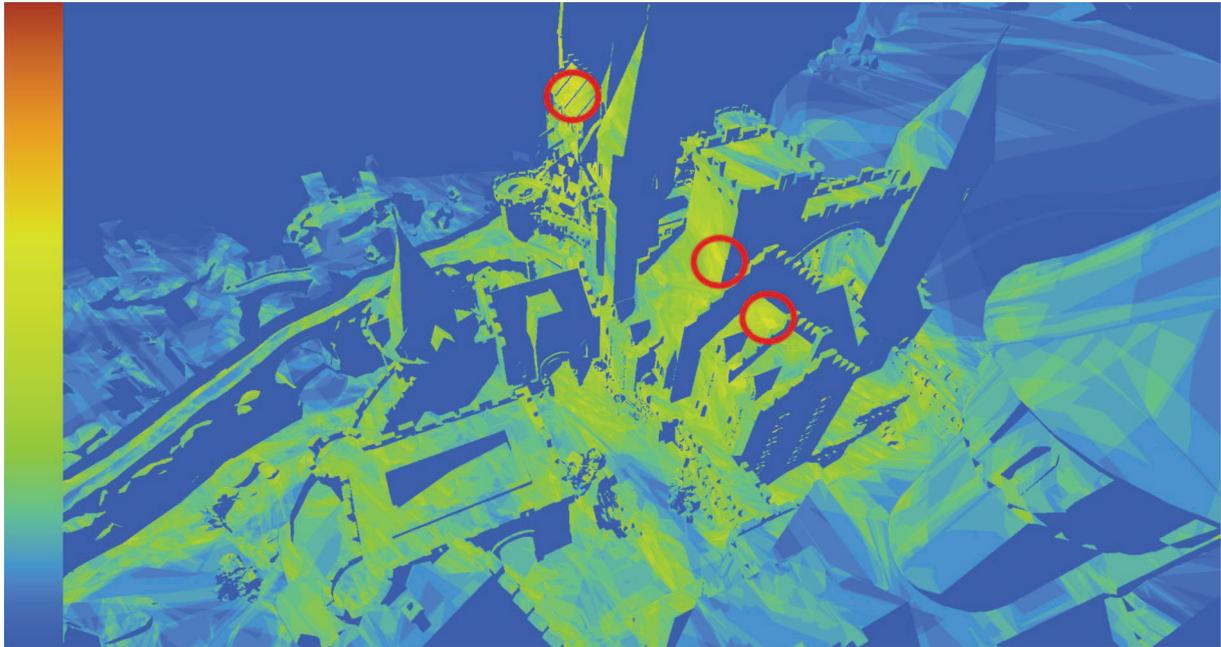


FIGURE 3.10 – Utilisation de la pile

Cette capture montre la profondeur de pile maximale atteinte lors du parcours de la partition de volumes d'ombre sur une échelle de 0 à 32. On peut remarquer qu'il n'y a que 3 petites zones entourées en rouge où la pile est presque entièrement remplie; pour tout le reste de l'image en bleu et vert, la pile est très peu utilisée.

On a vu que le parcours sans pile permet de limiter la pression sur les registres, mais implique de parcourir plus de noeuds que le parcours avec pile. La figure 3.11 montre que pour un parcours avec pile, il n'y a que très peu de points de l'image qui nécessitent de remplir complètement la pile. En partant de ces constatations, on peut envisager un parcours hybride :

- on utilise un parcours avec une petite pile permettant d'alléger la pression sur les registres ;
- lorsque la pile est pleine, on bascule vers un parcours sans pile ;
- quand le parcours sans pile est terminé, on reprend le parcours avec pile là où il s'était arrêté.

Avec un parcours hybride, on fait un compromis entre la pression sur les registres engendrée par la pile et le nombre de noeuds à visiter, et donc le trafic mémoire. Du point de vue programmation sur GPU, cette approche peut sembler peu compatible avec l'architecture parallèle des GPU. En effet pour garantir une exécution parfaitement parallèle, toutes les unités d'exécution d'un bloc doivent suivre les mêmes instructions. Dans le cas d'un parcours hybride, des unités d'exécution d'un même bloc peuvent être amenées à utiliser des algorithmes différents, et donc à réduire l'efficacité du parallélisme en introduisant de la divergence à l'exécution. Dans le cas des partitions des volumes d'ombre, ce problème de divergence est en partie masqué par le fait que les points nécessitant de déclencher un parcours sans pile sont généralement voisins et peu nombreux, comme on peut le voir sur la figure 3.11 : le GPU traitant l'image par groupe de pixels voisins, il n'y a finalement que peu de zones de l'image qui poseront des problèmes de divergence.

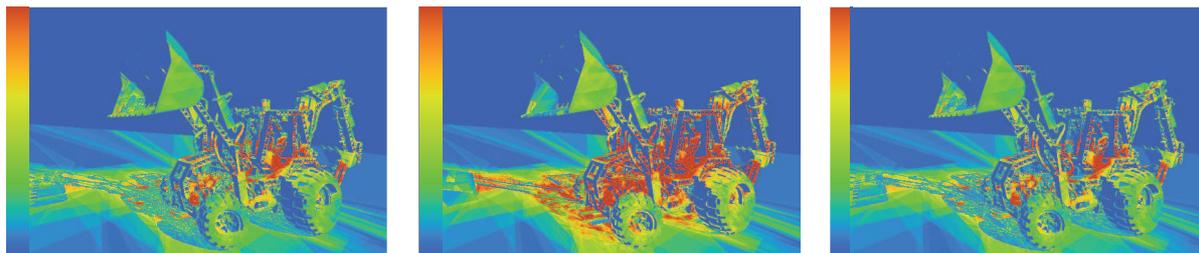


FIGURE 3.11 – Comparaison des coût en noeuds

De gauche à droite, le nombre de noeuds visités pour le parcours avec pile, le parcours sans pile et le parcours hybride avec dans tous les cas les optimisations d’angle et de profondeur activées : le parcours hybride est efficace, le nombre de noeuds visités étant proche du résultat de la version avec une pile tout en évitant le risque de dépassement de pile.

### 3.5 Implémentation

L’implémentation est semblable à celle présentée dans le chapitre 2, avec cependant quelques améliorations apportées à la construction. Les codes sources des différents parcours et constructions associés sont disponibles sur GitHub : <https://github.com/PSVcode/EGSR2016>.

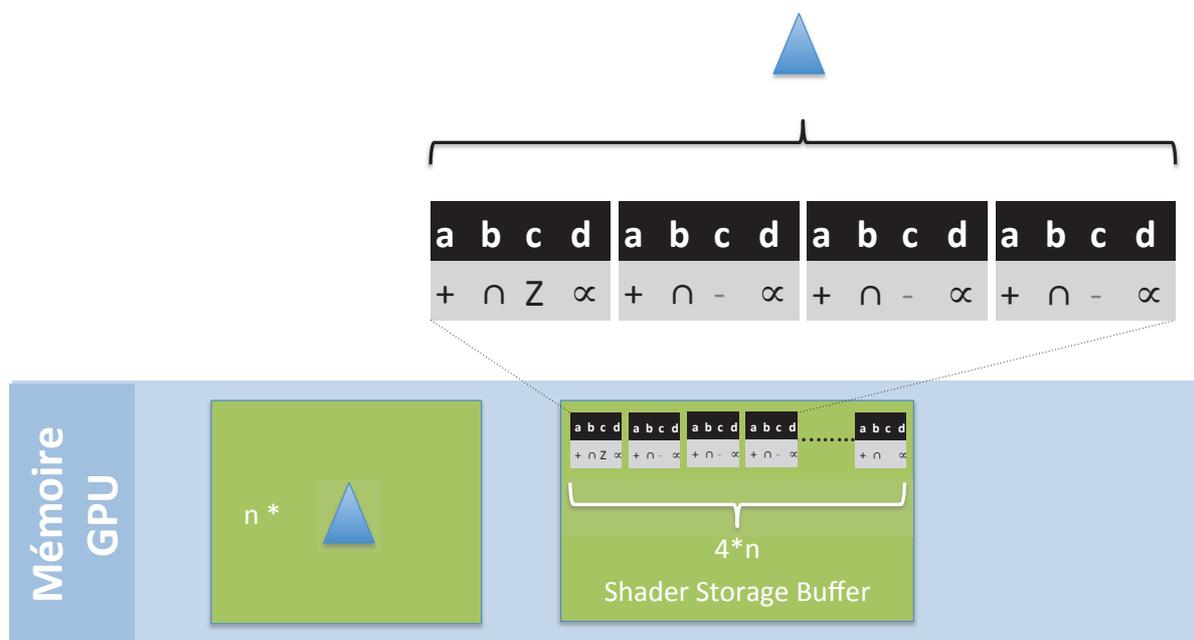


FIGURE 3.12 – Stockage

Chaque noeud contient l’équation du plan d’ombre (a,b,c et d). Le lien vers le fils négatif étant implicite, il n’est plus nécessaire de le stocker : les noeuds d’un même volume d’ombre sont stockés consécutivement dans la mémoire, le fils négatif étant alors systématiquement le noeuds suivant dans le tableau ; la libération d’un entier par noeud permet d’ajouter l’information de profondeur, qui y sera encodée sous la forme d’un entier grâce à la fonction *floatBitsToUint*. Cette distance Z sera stockée dans le premier noeud de chaque volume d’ombre.

## Stockage

La taille des noeuds de la structure est similaire en restant alignée sur des multiples 16 octets. La figure 3.12 détaille l'organisation des données.

## Construction

La construction suit l'algorithme 7, auquel on ajoute les 2 optimisations présentées ci-dessous.

### Frustum adaptatif

Pour la construction de la partition de volumes d'ombre en profondeur, on ajoute une technique communément utilisée avec les volumes d'ombre et les Shadow Maps : le frustum adaptatif, qui vise à conserver uniquement la géométrie située dans le frustum de caméra et ce qui est susceptible de projeter une ombre dans le frustum de vue (cf. figure 3.13). Cette optimisation vise à diminuer le nombre de triangles à traiter, afin de limiter le trafic mémoire et les défauts de mémoire cache.

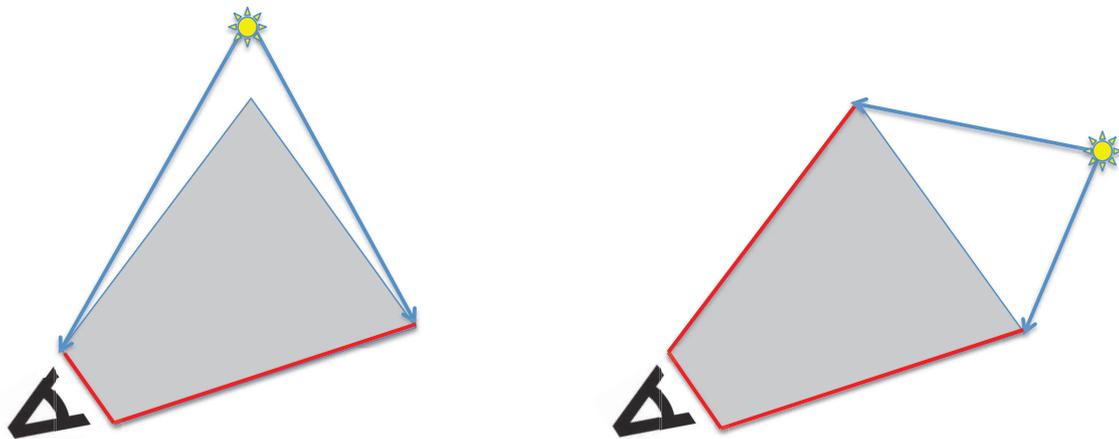


FIGURE 3.13 – Frustum adaptatif

À partir du frustum de vue et de la lumière, on génère un polyèdre contenant le frustum caméra et la lumière. Seule la géométrie qui se trouve dans cette zone délimitée par les segments rouges et bleus sera insérée dans la partition de volumes d'ombre, en évitant ainsi de traiter la géométrie inutile. Pour cette optimisation, les plans délimitant le frustum adaptatif sont calculés une fois par image sur CPU avant d'être copiés sur le GPU. Ensuite, durant la construction de la partition de volumes d'ombre, on vérifiera lors de l'exécution du compute shader sur GPU si un triangle doit ou non être inséré, en comparant sa position par rapport aux différents plans du frustum adaptatif. Cette optimisation n'est efficace que lorsqu'on voit une portion de la scène, mais son coût est négligeable.

### Processus persistants

Les processus persistants ou *persistent threads* (cf. figure 3.14) est une approche de la programmation sur GPU étudiée par Gupta et al. [41], visant à optimiser le traitement de larges plages de données. Dans notre cas, les processus placent un triangle dans la partition de volume, puis en traitent un second et ainsi de suite jusqu'à ce que tous les triangles soient placés.

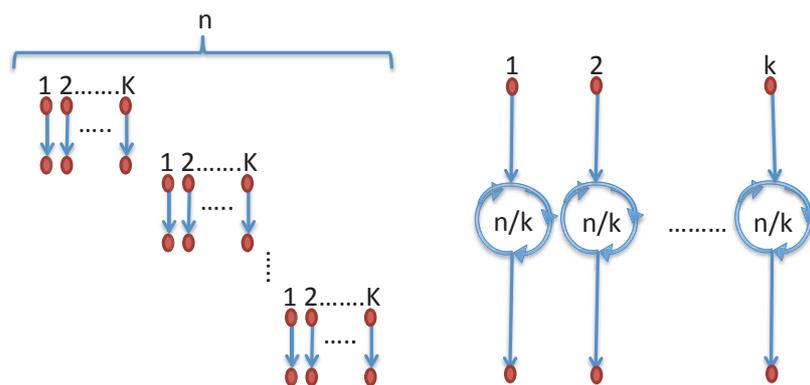


FIGURE 3.14 – Processus persistants

Une approche classique pour traiter  $n$  éléments en parallèle. On lance  $n$  processus comme à gauche. Comme un GPU dispose d'un nombre  $k$  d'unités de calcul, si on lance un traitement où  $n$  est plus grand que  $k$ , alors le GPU va exécuter les  $n$  tâches par blocs successifs. Or le démarrage et la terminaison d'un processus, symbolisés par les cercles rouges, sont des opérations coûteuses sur GPU. Pour éviter de perdre des cycles de calcul, la solution consiste à ce que chaque processus traite non pas un seul élément mais  $n/k$  éléments (dans le cas parfait où l'exécution est parfaitement parallèle). La différence de performance entre les 2 approches est inexistante lorsque  $n$  est petit, mais lorsque  $n$  devient important les processus persistants se montrent plus efficaces.

### 3.6 Résultats

Les mesures ont été faites sur une NVIDIA GTX 980 équipée de 4 Go de mémoire vive avec une résolution de  $1920 \times 1080$  et en utilisant 5 modèles, présentés du plus simple au plus complexe sur les figures 3.15 à 3.19, avec leur nombre de triangles respectif et la consommation mémoire de la partition de volumes d'ombre associée. On utilise une taille de pile de 32 pour le parcours de référence sauf pour ESCAVATOR dont la taille et la complexité géométrique exigent une pile de 48 entiers pour obtenir un résultat correct. Une taille de pile de 12 a été retenue pour le parcours hybride, ce choix sera discuté par la suite. Pour toutes ces mesures réalisées sur des parcours de 2500 images, l'élimination des triangles faces à la lumière a été activée.



FIGURE 3.15 – PLAYGROUND - 131 k triangles (16 Mo)



FIGURE 3.16 – EPIC CITADEL - 394 k triangles issue du SDK de Unreal Engine (48 Mo)

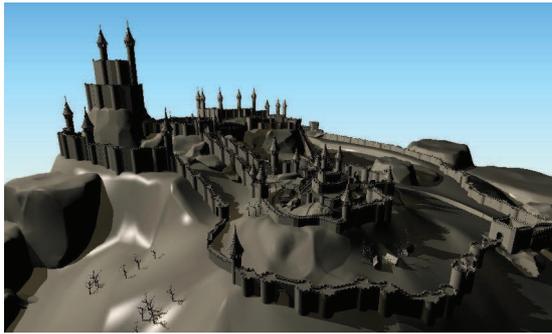


FIGURE 3.17 – CLOSED CITY - 623 k triangles (76 Mo)

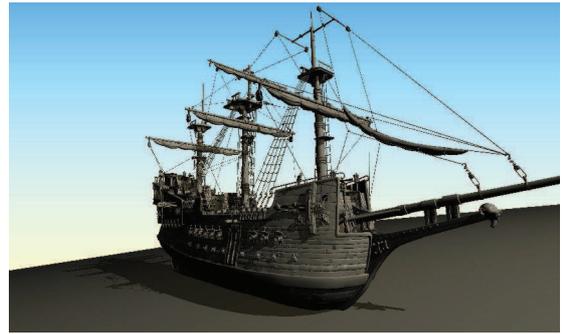


FIGURE 3.18 – SHIP - 956 k triangles (117 Mo)

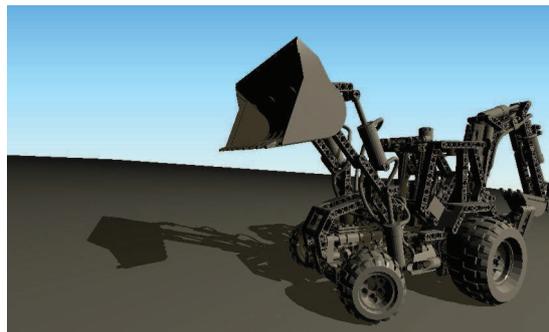


FIGURE 3.19 – ESCAVATOR - 1300 k triangles(138 Mo)

### Efficacité pratique des différents parcours

On sait que le temps de parcours est fonction du nombre de noeuds à visiter : les résultats présentés sur la figure 3.20 illustrent l'efficacité des différents types de parcours, avec de gauche à droite :

- le parcours avec pile présenté dans le chapitre 2 ;
- le parcours avec pile et l'optimisation de profondeur qui permet de diminuer le nombre de noeuds visités par rapport à la version originelle ;
- le parcours sans pile et avec l'optimisation de profondeur ;
- le parcours hybride avec une taille de pile de 12.

L'optimisation de profondeur se montre efficace sur tous les parcours l'utilisant, même si le parcours sans pile est légèrement plus coûteux en noeuds que les autres à cause des sauts liés au parcours des sous-arbres intersection. Il faut maintenant s'intéresser aux performances mesurées.

### Efficacité pratique et comparaison avec la technique du Z-pass

Les différents algorithmes de partition de volumes d'ombre sont ici comparés à une implémentation des volumes d'ombre avec extraction de silhouette [22] et un rendu de type Z-pass sur GPU, en utilisant les compteurs par pixel. À partir des résultats du tableau 3.1, plusieurs constatations s'imposent :

- le Z-pass est plus lent que toutes les autres méthodes, et présente systématiquement un écart-type important, ce qui témoigne de la saturation plus ou moins rapide de la bande passante mémoire du GPU selon les points de vue. Le Z-pass supporte mal la hausse de la complexité géométrique, et l'écart augmente avec les partitions de volumes d'ombre, ce qui s'explique aisément par le coût logarithmique du parcours d'une partition de volumes d'ombre. Dans le cas de la scène ESCAVATOR,

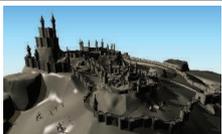
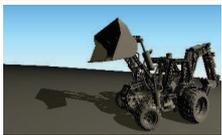
Moyenne du nombre de noeuds visités par pixel (% par rapport au partition de volume d'ombre)				
	Pile	Z-Pile	Z	Z Hybride
 Playground	61	43 (71%)	52 (86%)	43 (71%)
 Epic Citadel	167	104 (62%)	141 (84%)	104 (62%)
 Closed Citadel	85	60 (71%)	72 (85%)	60 (70%)
 Ship	142	100 (71%)	161 (114%)	93 (66%)
 Escavator	116	87 (75%)	125 (106%)	83 (72%)

FIGURE 3.20 – Comparaison des parcours sur 5 scènes

Pour chaque scène, on peut voir le nombre de noeuds visités suivant les différents types de parcours sous forme de carte de chaleur, et le nombre moyen de noeuds visités par image ainsi que le pourcentage de noeuds visités par rapport aux partitions de volumes d'ombre. On peut remarquer l'efficacité de l'optimisation de profondeur, qui permet généralement de diminuer de 30 % le nombre de noeuds. Le parcours sans pile demande de parcourir plus de noeuds que la version avec pile et distance mais l'optimisation de distance rend quand même ce parcours moins coûteux que la méthode originelle ; sauf sur les scènes SHIP et ESCAVATOR, qui sont des modèles à forte densité géométrique et dont les partitions de volumes d'ombre comportent beaucoup de sous-arbres intersection. Le parcours hybride permet d'obtenir, pour chaque scène, le parcours le moins coûteux en noeuds : les résultats sont identiques à ceux du parcours avec pile et distance ; on peut même remarquer des résultats meilleurs pour les scènes les plus complexes comme SHIP et ESCAVATOR.

- le temps de rendu est si important que l'on sort du domaine du rendu temps réel ;
- le parcours avec l'optimisation de distance utilisant une pile se montre plus efficace que le parcours originel, avec des gains moyens d'un facteur variant de 1,13 à 1,48. L'optimisation de distance se montre particulièrement efficace sur la scène ESCAVATOR, avec une diminution importante de l'écart type qui passe de 9,09 ms à 1,51 ms, ce qui démontre l'influence de cette optimisation sur la stabilité de la méthode ;
  - les 3 parcours avec distance présentent des comportements très proches et des performances comparables, le parcours hybride étant le plus efficace sauf sur la scène ESCAVATOR, avec un très léger avantage pour le parcours avec pile et distance ; l'écart-type du parcours hybride se montre systématiquement le plus faible, ce qui prouve la bonne stabilité de la méthode.

Tableau 3.1 – Résultats

Pour chaque scène, on présente le temps moyen de construction et de parcours des différentes méthodes sur 2500 images : *Pile* pour le parcours originel, *Z Pile* pour le parcours avec pile et profondeur, *Z* pour le parcours avec profondeur et sans pile, *Z Hybride* pour le parcours hybride avec profondeur, et, à titre de comparaison le *Z-pass* pour une implémentation classique des Shadow Volumes.

	Temps moyen ( $\sigma$ ) en millisecondes				
	pire/moyen/meilleurs facteur d'accélération par rapport aux partitions de volumes d'ombre				
	Pile	Z Pile	Z	Z Hybride	Z-pass
Playground	5.53 (1.66) -	4.89 (1.04) 0.95 / 1.13 / 1.31	4.88 (1.0) 0.89 / 1.13 / 1.36	4.66 (0.91) 0.93 / 1.19 / 1.43	6.79 (2.8) 0.43 / 0.81 / 2.36
Epic Citadel	9.95 (6.57) -	7.08 (4.40) 0.93 / 1.41 / 2.67	6.67 (3.98) 0.98 / 1.49 / 2.96	6.42 (3.79) 1.06 / 1.55 / 3.05	9.38 (6.93) 0.30 / 1.06 / 2.54
Closed Citade	7.61 (1.79) -	6.60 (1.32) 0.83 / 1.15 / 1.41	6.53 (1.22) 0.81 / 1.17 / 1.48	6.34 (1.20) 0.90 / 1.20 / 1.53	24.6 (8.98) 0.13 / 0.31 / 0.84
Ship	17.58 (6.62) -	13.56 (4.22) 0.81 / 1.30 / 1.81	13.95 (4.40) 0.45 / 1.26 / 1.86	13.92 (3.72) 0.78 / 1.26 / 1.82	25.99 (16.34) 0.13 / 0.68 / 3.19
Escavator	27.87 (9.09) -	18.83 (1.51) 0.99 / 1.48 / 2.15	18.66 (1.57) 1.02 / 1.49 / 2.10	17.52 (1.49) 1.13 / 1.59 / 2.17	57.19 (33.29) 0.13 / 0.49 / 2.64

L'analyse du code assembleur généré pour les différents parcours permet de voir leur besoin en mémoire locale :

- 48 blocs de 32 bits pour le parcours originel ainsi que la version avec la distance ;
- 0 blocs de 32 bits pour le parcours sans pile ;
- 12 blocs de 32 bits pour le parcours hybride.

On peut s'étonner que le surcoût en noeuds visités par le parcours sans pile ne se traduise pas par un temps de rendu supérieur : le surcoût en noeuds visités de la version sans pile est masqué par l'utilisation des 48 blocs de mémoire locale pour la version pile et avec distance.

Les figures 3.21 à 3.25 permettent de voir avec plus de détails le comportement des divers parcours tout au long des 2500 points de vue calculés. On peut remarquer que les courbes du parcours sans pile peuvent présenter des pics, par exemple sur la scène SHIP, tant sur le nombre de noeuds visités que sur le temps d'ombrage : dans certaines configurations, avec des arbres contenant beaucoup de sous-arbres intersection, le coût des sauts vers les différents pères intersection devient élevé. Le parcours hybride permet d'éviter cet écueil, en garantissant une plus grande stabilité sur le nombre de noeuds à visiter et sur le temps de rendu. Sur les scènes EPIC CITADEL, CLOSED CITADEL et SHIP, on peut apercevoir l'effet du frustrum adaptatif : de nettes diminutions du temps de construction sont visibles lorsqu'une petite partie de la scène est visible.

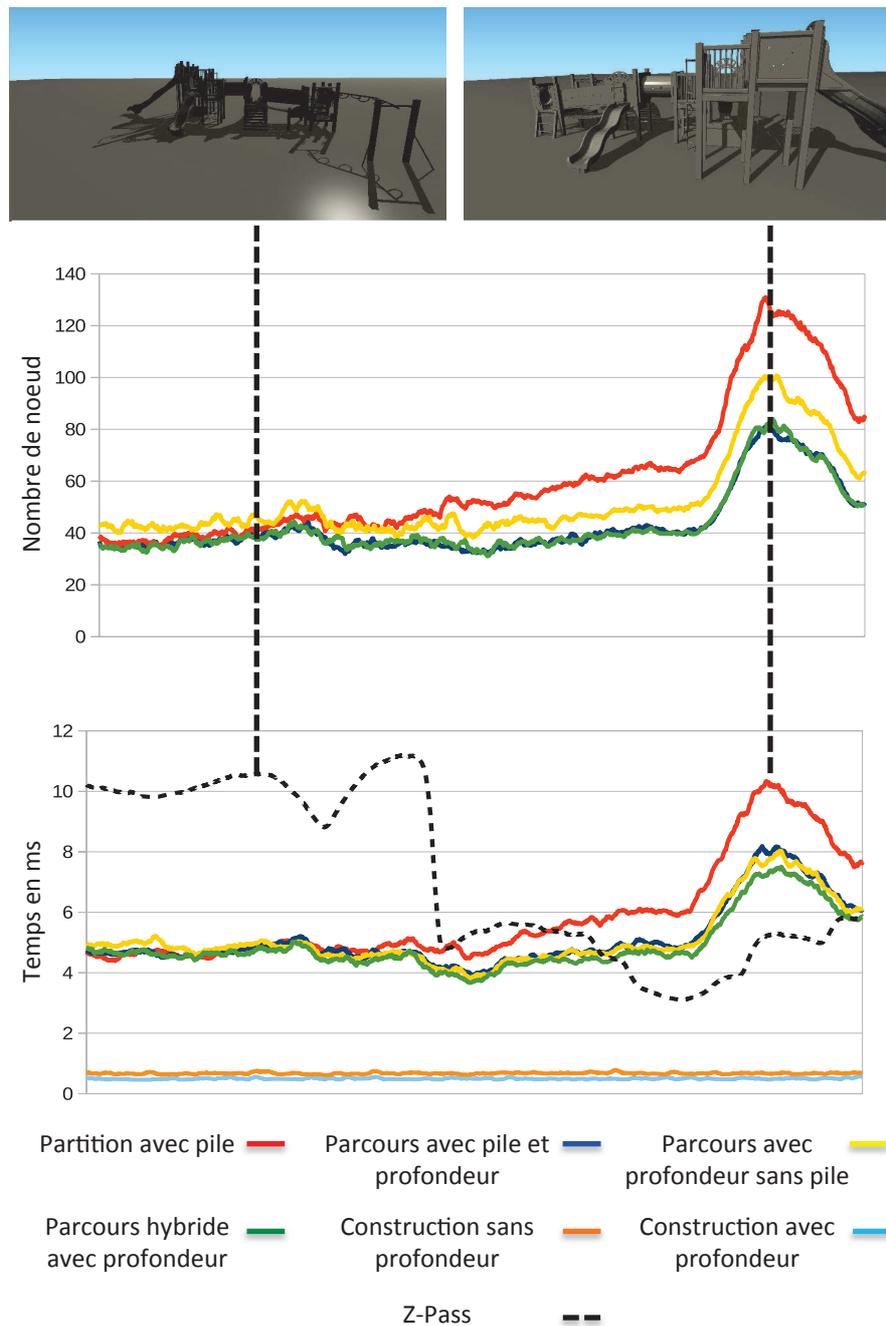


FIGURE 3.21 – Playground - 131K triangles

Sur le point de vue de gauche, les volumes d'ombres se projettent sur une partie importante de l'image, ce qui défavorise le Z-pass. Sur le point de vue de droite, au contraire, les volumes d'ombres sont petits et masqués par la géométrie ; les partitions de volumes d'ombre demandent de visiter beaucoup de noeuds, car ce point de vue génère peu d'ombre, et il est plus long d'être certain qu'un point n'est pas dans l'ombre. L'ajout de la profondeur permet de diminuer d'un tiers le nombre de noeuds à visiter.

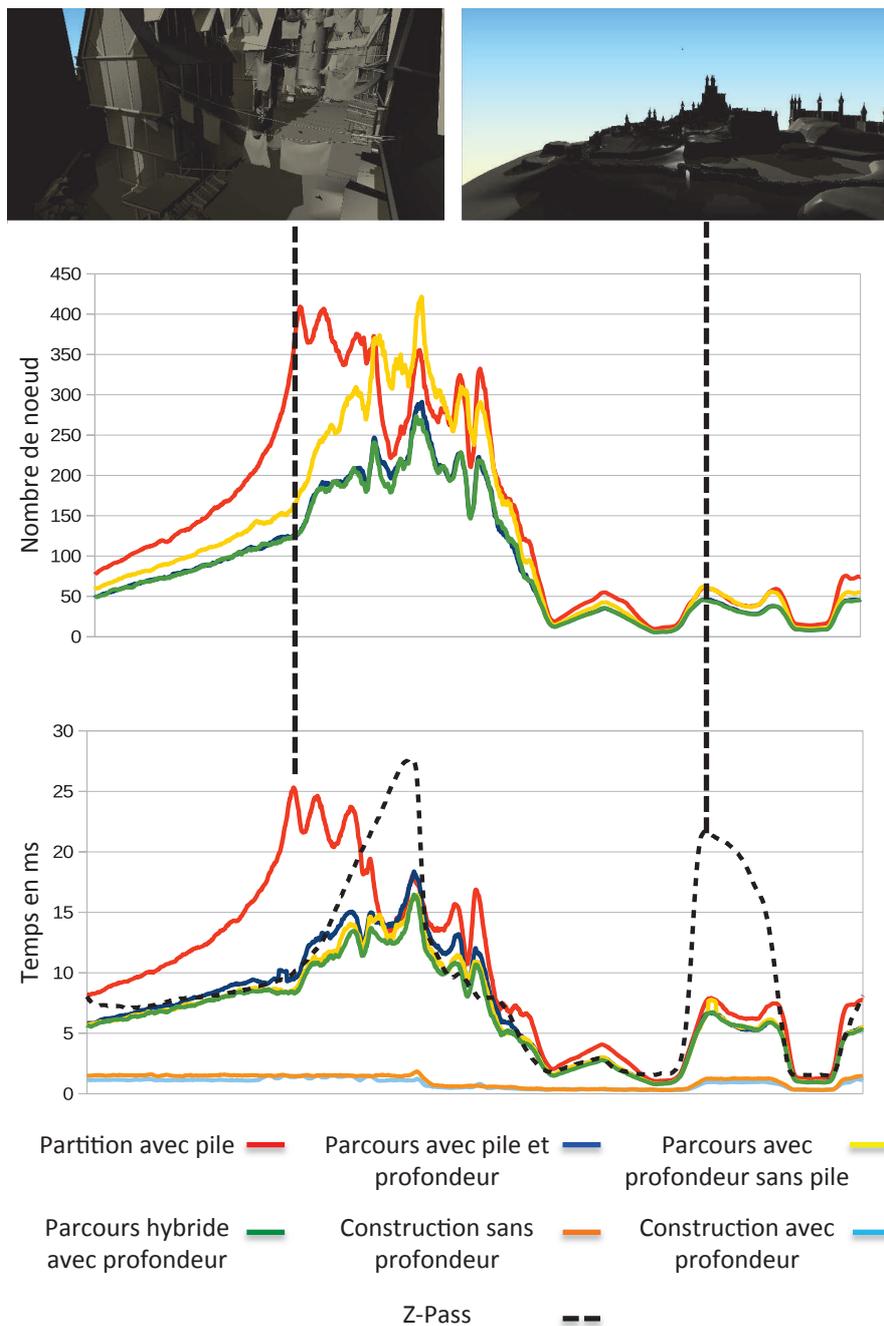


FIGURE 3.22 – Epic Citadel - 393K triangles

Le point de vue de gauche illustre un cas avec une géométrie dense et répartie sur une profondeur importante : l'optimisation de profondeur permet de diviser par 3 le nombre de noeuds à visiter. Le parcours sans pile peut devenir le plus mauvais sur certains de point de vue : s'il y beaucoup de sous-arbres intersection, le surcoût des sauts dans l'arbre devient pénalisant, en masquant même l'effet de l'optimisation de profondeur. Sur l'image de droite, les volumes d'ombre sont étirés et couvrent une bonne portion de l'image, ce qui pénalise le Z-pass.

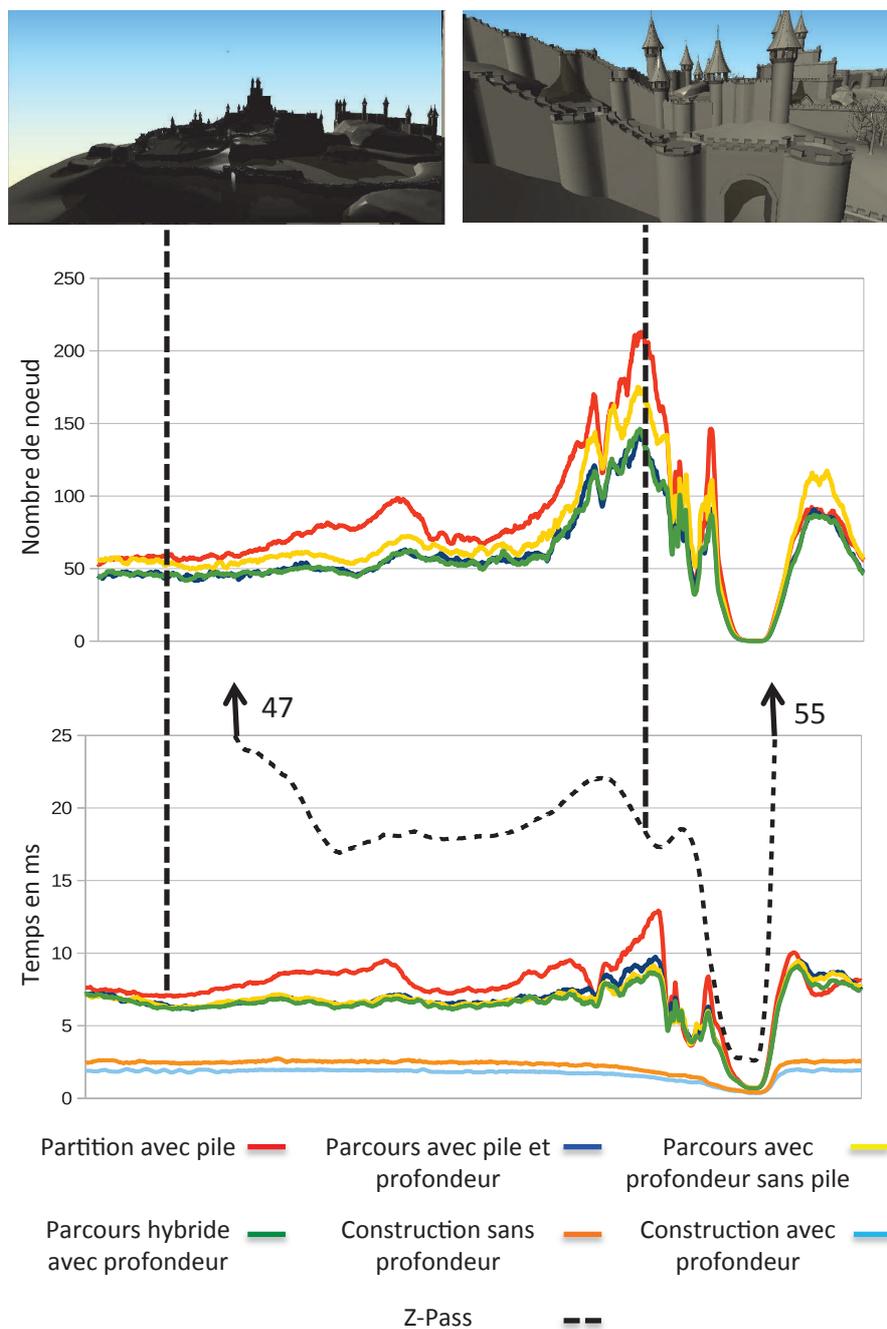


FIGURE 3.23 – Closed Citadel - 623K triangles

Sur l'image de gauche, l'ombre de silhouettes complexes qui s'étalent sur toute la scène montre les limites du Z-pass, dont les temps de rendu sont à la limite du temps réel sur ce point de vue. L'image de droite montre le point de vue qui demande de visiter le plus de noeuds pour les partitions de volumes d'ombre ; cependant, l'optimisation de profondeur permet d'économiser plus d'une cinquantaine de noeuds par requête. Même sur ce point de vue coûteux en noeuds, les partitions de volumes d'ombre se montrent plus rapide que le Z-pass quelque soit le type de parcours.

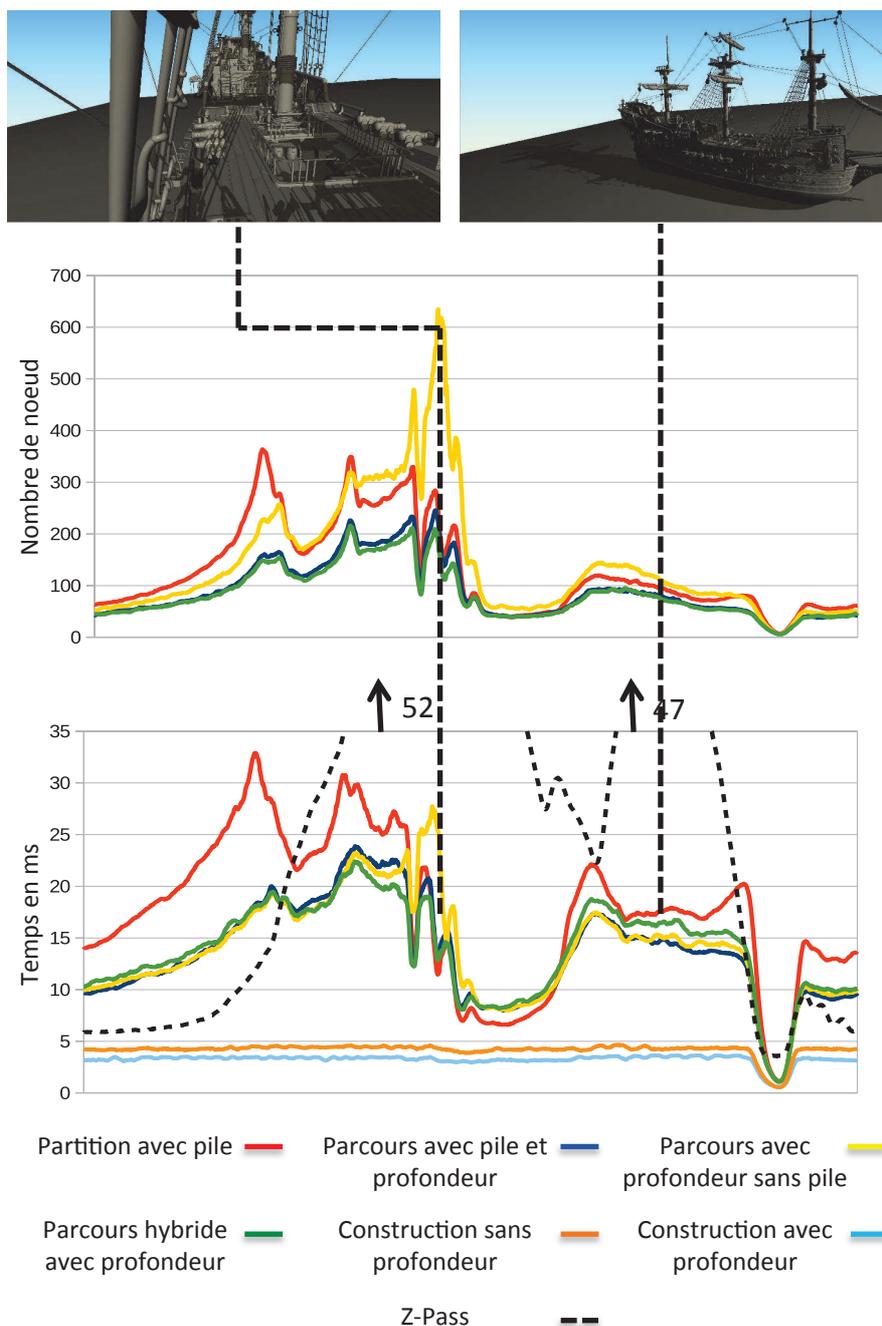


FIGURE 3.24 – Ship - 956K triangles

L'image de gauche illustre un point de vue qui génère une partition de volumes d'ombre avec beaucoup de sous-arbres intersection : les aller-retours dans les sous-arbres intersection défavorisent le parcours sans pile, en triplant le nombre de noeuds à visiter par rapport au parcours hybride qui est ici le plus efficace. C'est à cause de ce genre de cas pathologiques qui nuisent à la stabilité du parcours sans pile qu'il faut lui préférer le parcours hybride. Sur l'image de droite, le Z-pass est mis à rude épreuve avec une ombre de silhouette complexe. Ici, le parcours montre particulièrement l'instabilité du Z-pass dont les temps de rendu varient d'un facteur 10 selon les points de vue.

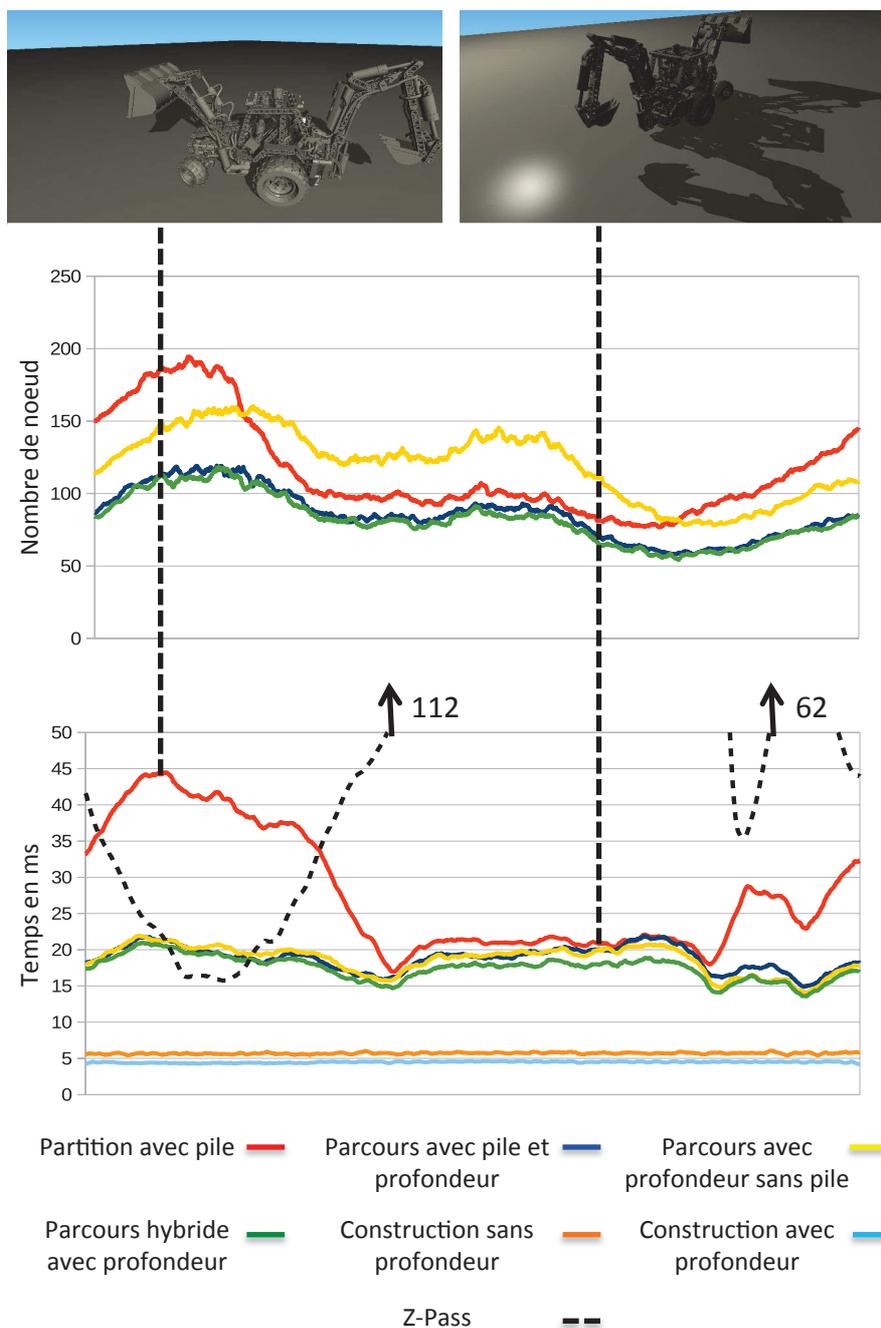


FIGURE 3.25 – Escavator - 1300K triangles

L'image de gauche représente un cas défavorable pour les partitions de volumes d'ombre originelles : il y a peu de points dans l'ombre, et la densité de géométrie entre la lumière et ces points est importante. L'optimisation de distance se montre très efficace en divisant par plus de 2 le temps de rendu. Ce genre de point de vue est favorable au Z-pass, les volumes d'ombre étant petits. En revanche, sur la capture de droite, le Z-pass ne permet pas de faire du rendu temps réel, les volumes d'ombre sont trop nombreux et trop grands.

## 3.7 Limitations

### Le choix de la taille de la pile

Pour le parcours hybride, la pile utilisée est d'une taille de 12 dans toutes les scènes, afin de permettre les comparaisons. La figure 3.26 montre l'évolution du temps moyen de rendu en fonction de la taille de la pile et de la scène : on peut voir que sur les 3 scènes les plus simples, PLAYGROUND, EPIC CITADEL et CLOSED CITADEL, une taille de pile de 12 est quasiment optimale. Alors que sur les modèles plus complexes une taille plus importante est nécessaire pour obtenir les meilleurs résultats : respectivement 16 pour la scène SHIP et 20 pour EXCAVATOR. Pour la cohérence des comparaisons, nous avons pris une même taille de pile pour tous les modèles, mais ce choix, d'une pile de 12 noeuds, n'est pas optimal pour toutes les scènes :

- avec une pile de 12 le rendu de SHIP prend 13.92 ms, alors qu'avec une pile de 16 le temps de rendu chute de presque 1 ms à 12.99 ms ;
- sur EXCAVATOR, le passage d'une pile de 12 à 20 noeuds permet de gagner presque 1 ms sur le temps de rendu.

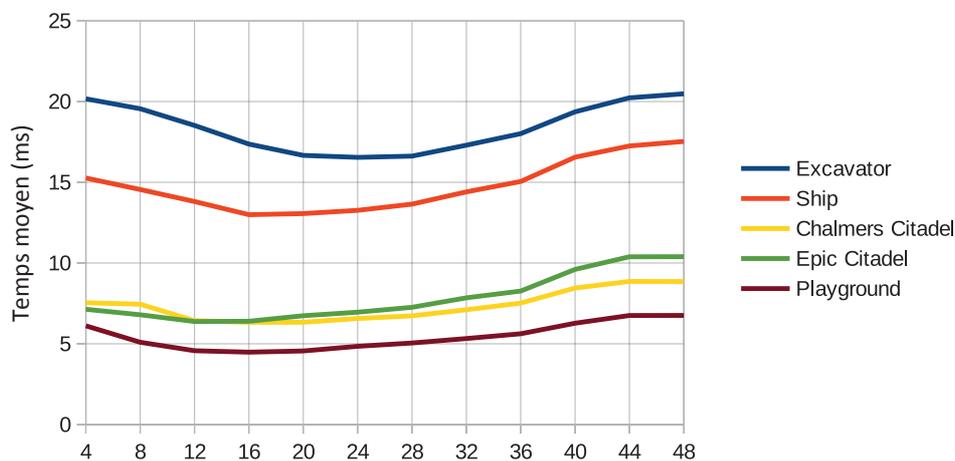


FIGURE 3.26 – Impact de la taille de la pile pour un parcours hybride

Pour chaque scène, on mesure le temps moyen sur une animation avec différentes tailles de pile.

Sur des scènes complexes, une petite taille de pile va provoquer un déclenchement rapide du parcours sans pile qui demande de parcourir plus de noeuds, et perdre en efficacité à cause d'une divergence plus importante à l'exécution. Tandis qu'avec une taille de pile importante, la pression sur les registres augmente : on se rapproche du parcours avec pile et distance. Même si le choix d'une pile de 12 noeuds ne garantit pas les meilleures performances dans la totalité des cas, c'est globalement le compromis le plus efficace, et il évite tout risque de dépassement de pile.

### La construction

L'optimisation de la distance mériterait une meilleure utilisation, par exemple en construisant une partition de volumes d'ombres qui insérerait les triangles par ordre croissant de distance par rapport à la lumière. Malheureusement cette approche est incompatible avec l'introduction d'aléas dans l'ordre d'insertion des triangles, qui est nécessaire pour obtenir un arbre relativement équilibré. Le parcours des partitions de volumes d'ombres ont l'avantage d'avoir un parcours en  $O(\log(n))$  pour  $n$  triangles mais cela

ne doit pas faire oublier le coût de construction qui est  $O(n \log(n))$ . Lorsque les modèles géométriques deviennent très complexes, comme sur la figure 3.27, le temps de la construction de la partition de volumes d'ombre devient significatif, voire prédominant.

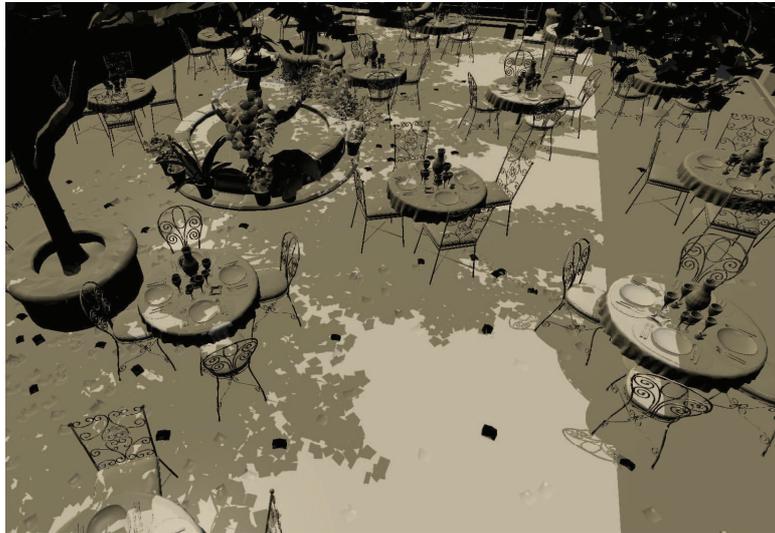


FIGURE 3.27 – SAN MIGEL avec ses 7 millions de triangles

Même sur NVIDIA GTX 980, le temps de construction de 45 ms est conséquent, alors même que le parcours de la partition reste rapide, à 17 ms, compte tenu de la complexité géométrique de la scène.

### La parcours

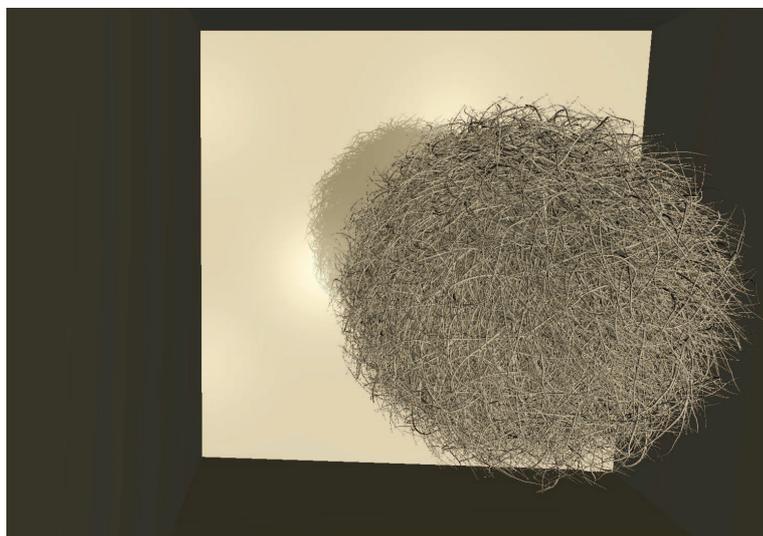


FIGURE 3.28 – HAIR BALL

Sur ce modèle dense en géométrie avec ses 2 millions de triangles, le temps de construction est de 13 ms sur une NVIDIA GTX 980, mais le temps de parcours devient important : 40 ms.

Il existe des types de modèles très chargés en géométrie, comme HAIR BALL (cf.,figure 3.28), très dense en géométrie et générant une partition de volumes d'ombre dense en sous-arbres intersection, donc

le parcours peut devenir très lent et les trajets dans la structure de pixels voisins peuvent devenir très différents, ce qui fait perdre l'effet bénéfique de la mémoire cache.

### 3.8 Conclusion

Nous avons mis en évidence la moindre efficacité des partitions de volumes d'ombre sur certain point de vue dans les scènes comportant de la géométrie répartie en profondeur, puis l'importance de l'usage de la mémoire du GPU. En réponse, l'introduction de l'optimisation de distance permet de diminuer de manière non négligeable le nombre de noeuds à visiter et donc de limiter les accès mémoires. D'autre part, la modification du parcours de la structure et l'adoption du parcours hybride permettent de diminuer la pression sur les registres, tout en supprimant le risque d'erreurs liées à la taille de la pile de la méthode originelle. Avec cette évolution des partitions de volumes d'ombre, la méthode gagne en stabilité et rapidité. Il reste le paramètre de la taille de la pile pour le parcours hybride : même si la taille de 12 entiers choisie n'est pas systématiquement la meilleure, elle reste efficace pour des scènes allant jusqu'au million de triangles.

La méthode montre ses limites lorsque la quantité de triangles est importante ou quand la géométrie est très dense ; dans le premier cas c'est la construction qui devient pénalisante alors que dans le second cas c'est le parcours qui le devient. Pour améliorer la méthode et pouvoir traiter des scènes dépassant le million de triangles il faudrait envisager un mode de construction différent. De même, il pourrait être intéressant d'adopter un parcours différent en s'inspirant de la méthode d'Erik Sintorn [30], pour faire traverser la partition de volumes d'ombre par une structure hiérarchique des pixels de l'image pour pouvoir envisager le rendu dans de très hautes résolutions.

# Bibliographie

- [1] J. M. Hasenfratz, M. Lapierre, N. Holzschuch, F. Sillion, and A. GRAVIR/IMAG-INRIA, “A Survey of Real-time Soft Shadows Algorithms,” *Computer Graphics Forum*, 2003.
- [2] T. Nishita and E. Nakamae, “Continuous tone representation of three-dimensional objects taking account of shadows and interreflection,” *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 23–30, Jul. 1985. [Online]. Available : <http://doi.acm.org/10.1145/325165.325169>
- [3] E. Heitz, J. Dupuy, S. Hill, and D. Neubelt, “Real-time polygonal-light shading with linearly transformed cosines,” *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 41, 2016.
- [4] L. Williams, “Casting curved shadows on curved surfaces,” *Computer Graphics (SIGGRAPH ’78 Proceedings)*, vol. 12, no. 3, pp. 270–274, Aug. 1978.
- [5] F. C. Crow, “Shadow algorithms for computer graphics,” *Computer Graphics (SIGGRAPH ’77 Proceedings)*, vol. 11, no. 2, Summer 1977.
- [6] E. E. Catmull, “A subdivision algorithm for computer display of curved surfaces.” Ph.D. dissertation, 1974, aAI7504786.
- [7] M. Wimmer, “Hard shadows aliasing and remedies,” 2013, cours SIGGRAPH 2013.
- [8] Y. Wang and S. Molnar, “Second-depth shadow mapping,” Chapel Hill, NC, USA, Tech. Rep., 1994.
- [9] L. Scandolo, P. Bauszat, and E. Eisemann, “Compressed multiresolution hierarchies for high-quality precomputed shadows,” *Computer Graphics Forum (Proc. Eurographics)*, vol. 35, no. 2, May 2016. [Online]. Available : <http://graphics.tudelft.nl/Publications-new/2016/SBE16>
- [10] E. Eisemann, M. Schwarz, U. Assarsson, and M. Wimmer, *Real-time shadows*. CRC Press, 2011.
- [11] S. Brabec, T. Annen, and H.-P. Seidel, “Shadow mapping for hemispherical and omnidirectional light sources,” in *Advances in Modelling, Animation and Rendering*. Springer, 2002, pp. 397–407.
- [12] J. Hourcade and A. Nicolas, “Algorithms for antialiased cast shadows,” *Computers And Graphics*, vol. 9, no. 3, pp. 259 – 265, 1985. [Online]. Available : <http://www.sciencedirect.com/science/article/pii/0097849385900524>
- [13] S. Brabec, T. Annen, and H.-P. Seidel, “Practical shadow mapping,” *Journal of Graphics Tools*, vol. 7, no. 4, pp. 9–18, 2002. [Online]. Available : <http://dx.doi.org/10.1080/10867651.2002.10487567>
- [14] M. Stamminger and G. Drettakis, “Perspective shadow maps,” *ACM Trans. Graph.*, vol. 21, no. 3, pp. 557–562, Jul. 2002. [Online]. Available : <http://doi.acm.org/10.1145/566654.566616>
- [15] R. Dimitrov, “Cascaded shadow maps,” *Developer Documentation, NVIDIA Corp*, 2007.
- [16] N. Kasyan, “Playing with real-time shadows,” 2013, cours SIGGRAPH 2013.
- [17] E. Sintorn, V. Kämpe, O. Olsson, and U. Assarsson, “Compact precomputed voxelized shadows,” *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 150, 2014.
- [18] N. K. M. S. M. W. Elmar Eisemann, Ulf Assarsson, “Shadow maps and shadow volumes,” 2013, cours SIGGRAPH 2013.
- [19] T. Heidmann, “Real shadows, real time,” *Iris Universe*, vol. 18, pp. 28–31, 1991, silicon Graphics, Inc.
- [20] W. Bilodeau and M. Songy, “Method for rendering shadows using a shadow volume and a stencil buffer,” May 7 2002, uS Patent 6,384,822. [Online]. Available : <https://www.google.com/patents/US6384822>
- [21] J. Carmack, early 2000.

- [22] T. Akenine-Möller and U. Assarsson, “On the degree of vertices in a shadow volume silhouette,” *Journal of Graphics Tools*, vol. 8, no. 4, pp. 21–24, 2003. [Online]. Available : <http://dx.doi.org/10.1080/10867651.2003.10487591>
- [23] P. Bergeron, “A general version of crow’s shadow volumes,” *IEEE Computer Graphics and applications*, vol. 6, no. 9, pp. 17–28, 1986.
- [24] D. B. Lloyd, J. Wendt, N. K. Govindaraju, and D. Manocha, “CC Shadow Volumes,” in *Eurographics Workshop on Rendering*, A. Keller and H. W. Jensen, Eds. The Eurographics Association, 2004.
- [25] M. Stich, C. Wächter, and A. Keller, “Efficient and robust shadow volumes using hierarchical occlusion culling and geometry shaders,” *GPU Gems*, vol. 3, pp. 239–256, 2007.
- [26] B. Lloyd, J. Wendt, N. Govindaraju, and D. Manocha, “Cc shadow volumes,” in *ACM SIGGRAPH 2004 Sketches*. ACM, 2004, p. 146.
- [27] E. Eisemann and X. Décoret, “Fast scene voxelization and applications,” in *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. ACM, 2006, pp. 71–78.
- [28] T. Aila and S. Laine, “Alias-free shadow maps.” *Rendering techniques*, vol. 2004, p. 15th, 2004.
- [29] E. Sintorn, V. Kämpe, O. Olsson, and U. Assarsson, “Per-triangle shadow volumes using a view-sample cluster hierarchy,” in *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 2014, pp. 111–118.
- [30] E. Sintorn, O. Olsson, and U. Assarsson, “An efficient alias-free shadow algorithm for opaque and transparent objects using per-triangle shadow volumes,” in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 6. ACM, 2011, p. 153.
- [31] C. Wyman, R. Hoetzlein, and A. Lefohn, “Frustum-traced raster shadows : Revisiting irregular z-buffers,” in *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*. ACM, 2015, pp. 15–23.
- [32] N. Chin and S. Feiner, “Near real-time shadow generation using bsp trees,” *ACM SIGGRAPH Computer Graphics*, vol. 23, no. 3, pp. 99–106, 1989.
- [33] B. Naylor, J. Amanatides, and W. Thibault, “Merging bsp trees yields polyhedral set operations,” *ACM Siggraph Computer Graphics*, vol. 24, no. 4, pp. 115–124, 1990.
- [34] Y. Chrysanthou and M. Slater, “Shadow volume bsp trees for computation of shadows in dynamic scenes,” in *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, ser. I3D ’95. New York, NY, USA : ACM, 1995, pp. 45–50. [Online]. Available : <http://doi.acm.org/10.1145/199404.199412>
- [35] L. Carpenter, “The a -buffer, an antialiased hidden surface method,” *SIGGRAPH Comput. Graph.*, vol. 18, no. 3, pp. 103–108, Jan. 1984. [Online]. Available : <http://doi.acm.org/10.1145/964965.808585>
- [36] T. Nishita and E. Nakamae, “Continuous tone representation of three-dimensional objects taking account of shadows and interreflection,” in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’85. New York, NY, USA : ACM, 1985, pp. 23–30. [Online]. Available : <http://doi.acm.org/10.1145/325334.325169>
- [37] S. Popov, J. Günther, H.-P. Seidel, and P. Slusallek, “Stackless kd-tree traversal for high performance gpu ray tracing,” in *Computer Graphics Forum*, vol. 26, no. 3. Wiley Online Library, 2007, pp. 415–424.
- [38] R. Barringer and T. Akenine-Möller, “Dynamic stackless binary tree traversal,” *Journal of Computer Graphics Techniques*, vol. 2, no. 1, pp. 38–49, 2013.

- [39] M. Hapala, T. Davidovič, I. Wald, V. Havran, and P. Slusallek, “Efficient stack-less bvh traversal for ray tracing,” in *Proceedings of the 27th Spring Conference on Computer Graphics*. ACM, 2011, pp. 7–12.
- [40] S. Laine, “Restart trail for stackless bvh traversal,” in *Proceedings of the Conference on High Performance Graphics*. Eurographics Association, 2010, pp. 107–111.
- [41] K. Gupta, J. A. Stuart, and J. D. Owens, “A study of persistent threads style gpu programming for gpgpu workloads,” in *Innovative Parallel Computing (InPar), 2012*. IEEE, 2012, pp. 1–14.
- [42] J. Gerhards, F. Mora, L. Aveneau, and D. Ghazanfarpour, “Partitioned Shadow Volumes,” *Computer Graphics Forum, Proceedings of Eurographics 2015*, 2015.
- [43] F. Mora, J. Gerhards, L. Aveneau, and D. Ghazanfarpour, “Deep partitioned shadow volumes using stackless and hybrid traversals,” in *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*, E. Eisemann and E. Fiume, Eds. The Eurographics Association, 2016.

# Conclusion

Cette conclusion résume le contexte de cette thèse ainsi que les contributions apportées en rapport avec le but initial. Par la suite, on évoquera les limitations de la méthode ainsi que les possibles améliorations.

Le premier chapitre présente les 2 classes de méthodes prédominantes. D'un côté, les Shadow Maps, une méthode image rapide et stable dont les nombreuses itérations visent à diminuer autant que possible ses problèmes d'aliassages et d'artefacts tout en rendant la technique plus complexe et dépendante de nombreux paramètres. De l'autre, la technique des volumes d'ombre qui bénéficie de l'exactitude de son approche géométrique parfois au prix de la saturation de la bande passante mémoire du GPU rendant la méthode particulièrement instable malgré les améliorations qui y ont été apportées. Ces deux types d'approches ont été explorées en profondeur par la communauté scientifique et laissent peu de place à des évolutions significatives, d'où la publication récente de techniques hybrides. Enfin, est abordée la technique des SVBSP dont l'idée servira de base à la technique des partitions de volumes d'ombre.

Dans le second chapitre, on commence par remarquer que les opérations CSG de la méthode des SVBSP sont les sources de faiblesse de la technique : ces opérations mènent à des duplications d'informations dans la structure qui ne permettent pas d'avoir une empreinte mémoire prédictible, ce qui est particulièrement préjudiciable sur les GPU qui ne possèdent pas de mécanisme de mémoire virtuelle. A partir de ces constatations, une nouvelle structure de données est proposée sous la forme d'un arbre ternaire construit sur les plans délimitant les volumes d'ombre de la scène. Un des avantages de cette structure est que son empreinte mémoire est fixe pour un nombre de triangles donnés, avantage lié à la structure ternaire de l'arbre qui combiné à un parcours optimisé délivre des performances intéressantes.

Le chapitre 3 s'intéresse à l'analyse du parcours des partitions de volume d'ombres : c'est la partie la plus coûteuse de la méthode pour des scènes dont la géométrie n'excède pas le million de triangles. Un premier point à noter est une faiblesse de la structure lorsque la géométrie s'étale en profondeur du point de vue de la lumière, dans ce cas, le parcours est coûteux. Ce problème est résolu par l'ajout d'une information de profondeur dans la partition de volumes d'ombre. Le second point est l'usage d'une pile dans le parcours d'une partition de volumes d'ombre : si la pile est trop grande, elle diminue les performances en augmentant la pression sur les registres du GPU ; si elle est trop petite, des erreurs ou blocages peuvent survenir. Une première idée est d'envisager un parcours sans pile qui a l'avantage de diminuer la pression sur les registres mais qui se traduit par un surcoût en nombre de nœuds qui peut être désavantageux suivant le point de vue et la position de la lumière. Observant que peu de pixels utilisent la pile entièrement, un parcours hybride est proposé : il utilise un parcours avec une pile de petite taille, suffisante pour la majorité des pixels à ombrer, et lorsque la pile est remplie, le parcours bascule en mode sans pile.

La technique des partitions de volumes d'ombre propose une solution originale et simple de mise en œuvre qui permet d'avoir un résultat exact par pixel comme les techniques basées volumes d'ombre tout en ayant un comportement très différent : le parcours des partitions de volumes d'ombre face à la complexité géométrique est en  $O(\log(n))$  permettant ainsi de supporter des modèles géométriques complexes que les volumes d'ombre pourraient difficilement gérer en temps réel. Même si cette technique n'est pas la plus rapide dans le calcul d'ombre dure exacte, elle reste rapide et montre qu'il était justifié de remettre au goût du jour un type de méthode pourtant délaissée.

Il reste malgré tout des limitations, lorsque la géométrie dépasse le million de triangles, le temps de construction en  $O(n \log(n))$  devient prépondérant dans le temps de rendu. En dépit des tentatives multiples, la correction de cette limite est un véritable problème dont la résolution ne semble pas aisée. De même, avec des configurations géométriques denses du type boule de cheveux, le parcours perd en efficacité, ceci est un point commun aux méthodes basées géométrie. Une amélioration intéressante consisterait à modifier la méthode pour ne plus faire d'interrogation de la partition de volumes d'ombre

par pixel mais de s'orienter vers une interrogation par groupe de pixels grâce à une structure accélératrice construite sur le tampon de profondeur. Cette amélioration permettrait d'envisager de supporter la tendance nette à la hausse de résolution des écrans. Il faut aussi remarquer qu'on peut envisager d'étendre le champ d'utilisation des arbres ternaires à d'autres problématique de rendu, comme pour du lancer de rayons ou du calcul d'occlusion ambiante ou même à la gestion de collisions. L'empreinte mémoire fixe des arbres ternaires est une propriété intéressante pour du calcul sur GPU.

L'objectif de cette thèse était de proposer une méthode alternative aux grands classiques que sont les Shadow Maps et les volumes d'ombre. Les partitions de volumes d'ombre se démarquent clairement de cet existant en abordant différemment le problème et en présentant des propriétés algorithmiques intéressantes. Comme toute méthode, des limitations ont été soulignées. Mais pour une approche nouvelle, les performances obtenues sont déjà très compétitives, surtout à l'aune des 40 années de recherche dont bénéficient les Shadow Maps et les volumes d'ombre. En comparaison, les partitions de volumes d'ombre restent à explorer et appellent certainement de futures évolutions qui, nous l'espérons, seront facilitées par la diffusion des implémentations nos algorithmes (<https://github.com/PSVcode/EG2015> et <https://github.com/PSVcode/EGSR2016>).



## Partition de volumes d'ombre : une alternative pour le rendu d'ombres en temps réel

**Résumé :** Cette thèse aborde la problématique du calcul d'ombre exact par pixel en temps réel. Ce mémoire propose une nouvelle méthode de rendu d'ombre dure avec une partition de volumes d'ombre : un arbre ternaire basés sur les plans d'ombre des volumes d'ombre de la scène est construit dans un premier temps, avant de l'interroger pour déterminer l'ombrage des pixels de l'image. Une des propriétés intéressantes de cette structure est la prédictibilité de son empreinte mémoire ; contrairement aux méthodes de calcul d'ombre, cette méthode supporte des scènes géométriquement complexes de l'ordre du million de triangles grâce au comportement logarithmique du parcours de la structure vis à vis de la complexité géométrique.

**Mots clés :** Ombre dure, volume d'ombre, visibilité, structure de données.

## Partitioned Shadow Volumes : an alternative for real-time shadow rendering

**Abstract :** This thesis focuses on exact per pixel hard shadow computation. We propose a new method for rendering hard shadows using a partition of shadow volumes : first, a ternary tree based on the shadow planes of the shadow volumes, and then, it is traversed to determine the shading of each pixel in the image. An interesting property of this structure is the predictability of its memory footprint ; Unlike other geometric shadow methods, our approach supports complex scenes up to a million triangles thanks to the logarithmic behavior of the structure traversal with respect to the geometric complexity.

**Keywords :** Hard shadows, shadow volume, visibility, data structure.

**XLIM - UMR CNRS n° 7252**  
123, avenue Albert Thomas - 87060 LIMOGES