UNIVERSITÉ LIBRE DE BRUXELLES
FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

Les ombres au sein des jeux et des animations

Bruno Rocha Pereira
Pierre Gérard
Antoine Carpentier

Superviseurs : Tom Lenaerts et Jean-Sébastien Lerat

Année académique 2014 - 2015

Résumé

Ce rapport présente les différentes techniques de rendu d'ombre par ordinateurs, les résultats obtenues lors l'implémentation et les perspectives de ces dernières.

Table des matières

Chapitre 1

Introduction

Une ombre est une "zone sombre résultant de l'interception de la lumière ou de l'absence de lumière" [1]. C'est un élément indispensable au réalisme d'une scène d'animation ou de jeu vidéo. En effet ce sont les ombres qui vont apporter l'information quand à la position relative et à la taille des objets qui créent l'ombre. Dans le cas d'objets complexes, elles permettent d'obtenir des informations sur la forme des objets. Dans le monde réel, on est souvent confronté à plusieurs sources lumineuses qui apportent chacune leur lot d'informations en plus.

Nous avons, lors de ce projet d'année, étudié l'impact des ombres sur le réalisme des animations et des jeux vidéos. Cette étude a été réalisée en utilisant et comparant différents algorithmes de génération d'ombre, plus ou moins réalistes.

Nous nous sommes intéressés plus particulièrement aux algorithmes en temps réel car ceux-ci sont plus intéressants et plus attractifs pour une présentation pour le Printemps des Sciences [2].

Deux types d'ombres peuvent être distingués : les *soft shadows* et les *hard shadows*. Les premières ont des bords diffus et les secondes ont des bords nets. Plus la source de lumière est proche d'un objet, plus les bords d'une ombre réaliste sont diffus et inversément. Dans ce projet, nous nous somme intéressé uniquement aux algorithmes de génération des *soft shadows* car ils permettent également de générer des *hard shadows*.

Nous avons testé les différents aspects et effets des ombres dans différents scénarios que nous avons mis en application. Nous avons fait varier les objets, les sources lumineuses, leur nombre et leur mouvement afin de présenter des situations se rapprochant de la réalité.

FIGURE 1.1 – L'importance des ombres en image
Source : http ://maverick.inria.fr/Research/RealTimeShadows/importance.html

1. http ://www.larousse.fr/dictionnaires/francais/ombre/55933
2. http ://www.printempsdessciences.be

Chapitre 2

Etat de l'art

Au fil du temps, différents types d'algorithmes permettant de générer ces ombres ont été présentés dans la littérature scientifique. La littérature scientifique distingue donc deux grandes catégories :

- les *Hard Shadow*,
- les *Soft Shadow*.

Les ombres de type *Hard Shadow* ont été moins considérés ici car elles sont moins réalistes. En effet, ces ombres sont uniformément noires et ne représentent que l'ombre générée par un point lumineux. D'un autre côté, les ombres de type *Soft Shadow* sont beaucoup plus réalistes et sont celles qui sont les plus utilisées. Les algorithmes permettant de faire des ombres de type Soft Shadow seront donc ceux qui seront le plus étudiés ici.

Il existe plusieurs types de *Soft Shadow algorithms*, les principaux sont ceux de :

- *Ray tracing*
- *Shadow mapping*
- *Shadow volume*

Ce sont ces trois types d'algorithmes qui ont été étudiés et mis en applications dans ce projet.

2.1 Ray tracing

L'algorithme de *Ray tracing* a été présenté pour la première fois en 1968 par Arthur Appel[**?**] sous le nom de *Ray casting*. Il a ensuite été continué, sous le nom de *Ray tracing* cette fois, en 1978 par Whitted [**?**], qui y a rajouté la réflexion et la réfraction de la lumière. Cet algorithme consiste à tracer un rayon depuis le point de vue jusque chaque pixel créant une *ray surface*. La surface la plus proche du point de vue sera donc celle qui sera visible. A partir de chaque pixel, il faudra ensuite relier la source lumineuse. Si ce rayon a une intersection avec un quelconque objet, ce pixel sera dans l'ombre. Ceci n'est évidemment pas optimal puisqu'il nécessite un calcul pour chaque pixel de la scène.

2.2 Shadow volume

L'algorithme de *Shadow Volume* a été introduit par Crow [**?**]. Il a ensuite été implémenté grâce à l'accélération matérielle[**?**] mais n'a été que peu utilisé jusqu'à la proposition de Tim Heidmann d'accélérer matériellement cet algorithme sur du matériel accessible au grand public. Cette logique a donné naissance à l'algorithme de *z-pass*[**?**].

La méthode de *z-pass* consiste à premièrement initialiser un *stencil buffer* à zéro et un *depth buffer* avec les valeurs de profondeurs des objets visibles pour ensuite rastérisater les côtés des *shadow volumes*. Pour chaque partie de *shadow volume*, il s'agit ensuite d'incrémenter le pixel du *stencil buffer* correspondant si la face à une normale dans le sens inverse (on rentre alors dans le *shadow volume*) et décrémenté lorsque l'on en ressort. Le *shadow count* représentera alors le niveau d'ombre dans lequel est plongé le

point fixé et l'absence d'ombre si celui-ci est égal à 0.[?] Ce comptage peut être théoriquement réalisé jusqu'à une distance infinie, grâce à une méthode appelée *z-fail* ou *Depth fail*[?, ?].

Les algorithmes de *z-pass* ont néanmoins un défaut dans le cas où l'on place l'observateur dans l'ombre (dans un ou plusieurs shadow volumes). Ce problème a été en partie résolu par HORNUS et autres[?], qui vont proposer d'aligner la vue de la source lumineuse avec celle de l'observateur. Cette technique a été développée en comparant l'algorithme de z-pass, dont ils s'inspirent et qu'ils ont amélioré, avec celui de z-fail. Cette solution n'est pas encore optimale mais à ce jour, aucune autre alternative n'a été proposée.

Dans l'article [?], les auteurs présentent une nouvelle technique qui utilise le *Culling and Clamping (CC)* permettant d'éviter de générer des *shadow volumes* qui sont eux-même dans l'ombre ou qui n'interviennent pas dans l'image finale, ce qui a pour but d'améliorer les performances et donc d'accélérer la génération des ombres dans une scène.

Aila et Akenine-Molle font remarquer en 2004 [?] que les performances de génération des ombres sont inversément proportionnelles à la taille des *shadow volumes*. Pour remédier à cela, ils proposent un nouvel algorithme visant à réduire le temps de rastérisation. Cet algorithme est composé de deux étapes. La première étape consiste à trouver des zones de 8x8 pixels dont les bords sont soit complêtement dans l'ombre soit complêtement illuminés. La seconde étape verra s'effectuer une génération pixel par pixel de l'ombre des pixels se trouvant aux bords de l'ombre.

En 2004, Chan et Durand[?] utilisèrent une technique utilisant à la fois un algorithme de *shadow mapping* et un algorithme de *shadow volume*. Le premier est d'abord utilisé pour créer une *hard shadow* et obtenir la silhouette de l'ombre. L'algorithme de *shadow volume* est ensuite utilisé pour généré une ombre correcte (*soft shadow*) à partir de cette silhouette.

2.3 Shadow mapping

L'algorithme de *Shadow Mapping* a, quant à lui, été introduit par Lance Williams [?].

Le principe des *Shadow Mapping Algorithm* est de dresser dans un premier temps une carte de disparité (*depth map/image*) de la scène, comme vue depuis la source de lumière. Pour chaque *texel*, la profondeur de l'objet le plus proche de la source lumineuse sera stockée. Cet algorithme n'est pas optimal et la technique du *Percentage closer filtering*[?, ?] résoud un problème d'aliasing présent.

Dans le début des années 2000, plusieurs algorithmes utilisant un *filtering* furent présentés. Celui-ci permet d'utiliser une shadow map de basse résolution tout en présentant des résultats convaincants. En 2005, Donelly et Lauritzen [?] proposent un algorithme utilisant la variance de la distribution des profondeurs, visant à réduire fortement l'aliasing habituellement présent dans les algorithme de *shadow mapping* basiques tout en nécessitant peu de stockages et de calculs supplémentaires. En 2008, un nouvel algorithme est présenté[?], proposant une autre méthode pour réduire l'aliasing mais avec une technique encore plus efficace et produisant moins d'artefacts graphiques.

Cependant, l'utilisation de *shadow map* de basse résolution entraine un flou forcé, empêchant la création d'ombres nettes. D'autres algorithmes ont été proposés pour améliorer la précision sans demander de ressources trop énormes.

Les premiers sont les algorithmes appelés *Perspective Shadow Map*[?, ?, ?] utilisent le *warping*, qui permet d'avoir de bons résultats mais dégénerent en *shadow map* ordinaire.

Les seconds utilisent le *Partitioning*. Cette approche permet de diviser le frustum de vue et d'utiliser

une *shadow map* pour chaque sous-frustum. Cependant, pour être le plus précis possible, cette technique requiert un grand nombre de subdivisions, ce qui affecte les performances.

Un des seuls algorithmes qui présente une précision au pixel près et qui présente des bonnes performances pour le temps réel est celui qui a été présenté par Sintorn et Assarsson[**?**]. Celui-ci se focalise sur les ombres de la pilosité, qui nécessite de la précision et obtient pourtant des résultats corrects.

L'algorithme GEARS[**?**] rajoute un élément pris en compte, le dynamisme de la scène illuminée ainsi que celle de la lumière tout en gardant des excellentes performances.

Chapitre 3

Méthodes implémentées

Dans ce chapitre, nous allons décrire les différents algorithmes implémentés et l'environnement dans lequel ils ont été testé.

3.1 Choix de l'environnement de test

L'environnement est important. En effet, c'est sur celui ci que les différents algorithmes vont s'appliquer. En choisissant le même environnement pour tous les algorithmes nous pourront comparé leur performance en utilisant différentes *metrics*.

3.1.1 La 3D

OpenGL s'est imposé comme l'API 3D de choix étant donné sa spécification ouverte, ses fonctions bas niveau et sa disponibilité sur un grand nombre de plateformes. Nous avons utilisé OpenGL pour créer des scènes 3D, animer une/des caméra(s) et une/des source(s) de lumière mais également pour générer des ombres avec les algorithmes retenus.

Nous avons utilisé dans la mesure du possible un maximum les fonctions modernes d'OpenGl majoritairement présente dans les jeux et animations d'aujourd'hui. Le code propre a la 3D sera donc des shaders.

Nous avons aussi choisi de ne pas utilisé de moteur de rendu 3D pour car utiliser à la place directement le langage de bas niveau OpenGL permet d'avoir une approche plus fine et de maitriser plus de paramètre. En outre, cela permet de mieux comprendre le fonctionnement de la 3D et cette approche ne masque pas les opérations qui sont faites ce qui facilite a la comparaison des algorithmes.

3.1.2 Langages et bibliothèques utilisés

Les langages utilisés influencent les performances globales des algorithmes et du programme de test. Cependant, si pour chaque algorithme on utilise le même langage, il sera toujours possible de les comparer. Nos choix ont donc été basés sur l'interopérabilité du langage de programmation et de OpenGL. Ils sont les suivants :

- Nous avons utilisé Python 2.7 comme langage de programmation car il permet un développement rapide et possède des bindings vers les bibliothèques OpenGL, OpenCL, numPy etc... écrites en C/C++. Il permet donc d'allier la rapidité d'écriture des langages de scripts à la rapidité d'exécution des langages compilés. Nous avons aussi utilisé des package du Python Package Index pour nous faciliter la tâche.
- Nous avons utilisé pyQt comme librairie qui s'interfacera avec la célèbre bibliothèque Qt 5. Nous avons utilisé cette interface graphique étant donnée sa facilité d'utilisation et son caractère complet. De plus nous avions pour la plupart déjà utilisée cette dernière durant notre cursus.

– Nous comptions utiliser OpenCL pour profiter de puissance de calcul des cartes graphiques modernes. Nous ne l'avons finalement pas utilisé car nous nous sommes redirigés vers la puissance de calcul de numpy associée avec quelques parties d'implémentation en C++
– Nous avons enfin utilisé vispy, librairie de visualisation 3D basée sur OpenGL qui permet d'utiliser les avantages d'OpenGL moderne et d'utiliser des shaders avec une certaine abstraction. Ceci permet d'effectuer les calculs de rendu graphique directement sur le GPU, ce qui accroit la vitesse d'exécution et donc les performances.

3.1.3 Scènes présentées

Plusieurs scènes ont été utilisées pour présenter et mettre en application les différents algorithmes.

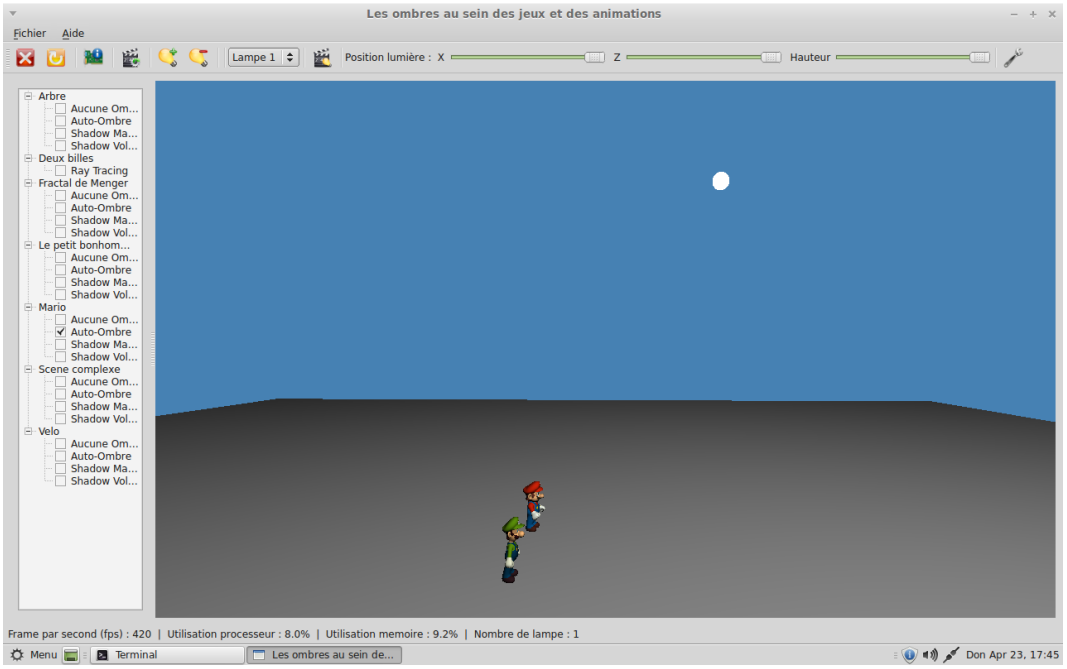

FIGURE 3.1 – Environnement de test

Création des scènes

Il est intéressant de pouvoir placer des modèles en hauteur, à différents endroits. Ceci permet de démontrer que lorsque les ombres sont absentes, on ne peut percevoir les informations relatives ni à la hauteur, ni au relief.

Pour cela nous avons utiliser deux ressources :

1. Des fichiers de configuration de la scène en JSON,
2. Des modèles 3D au format libre OBJ.

Un dossier prédéfinit contient un certain nombre de fichier JSON qui eux contiennent toutes les informations nécessaires a la création d'une scène. Ils contiennent :

1. Le nom de la scène,
2. Une courte description de la scène,
3. Quels modèles 3D sont inclus à la scène,
4. Les algorithmes compatibles avec la scène,
5. La position de ces modèles 3D.

Les modèle 3D au format OBJ proviennent de site internet proposant des modèles libres de droit et sont très divers de manière à pouvoir réaliser beaucoup de test différents : vélo, fractal de menger, mario et luigi, arbres, ...

Lumières

Les lumières ont bien évidemment une importance capitale dans notre environnement de test.

Nous n'avons implémenté que des lumières ponctuelles de couleur blanche. Il est par contre possible d'en mettre autant que souhaité et de changer leur position individuellement. De plus il est possible via un sélectionneur dans la barre d'outils de les déplacer en temps réel. Cela permet de tester les positions critiques des ombres ou des glitch pourrait apparaitre.

Animations

Pour rendre le programme plus interactif et pour pouvoir mieux observé les ombres, nous avons implémenté deux animations différentes.

Une première qui consiste a faire tourner de manière régulière la caméra autour du centre de la scène.

une deuxième qui consiste a faire tourner la lampe sélectionner dans la barre d'outils.

3.1.4 Metrics utilisé

Pour mesurer la performance d'un algorithme nous avons utilisé différentes mesures.

Image par seconde

Le nombre d'image par seconde ou Frame par second (FPS) en anglais est une mesure indiquant le nombre d'image affiché par seconde sur un écran. Plus ce nombre est élevé plus l'animation semble fluide. La méthode de calcul utilisé est la suivante :

$nombredefps = \frac{1seconde}{Tempsncessairepourgnreruneframe}$

Pour obtenir un résultat stable, une moyenne est faite sur un échantillon d'une durée de 1 seconde.

CPU

Une librairie python permet de mesurer le pourcentage d'utilisation du processeur en par le programme courant. Une haute utilisation indique un algorithme peu performant.

Mémoire

Une librairie python permet de mesurer le nombre de megabytes de mémoire vive utilisé par le programme courant. Une haute utilisation indique un algorithme peu performant.

3.2 Algorithmes implémentés

3.2.1 Shadow volume

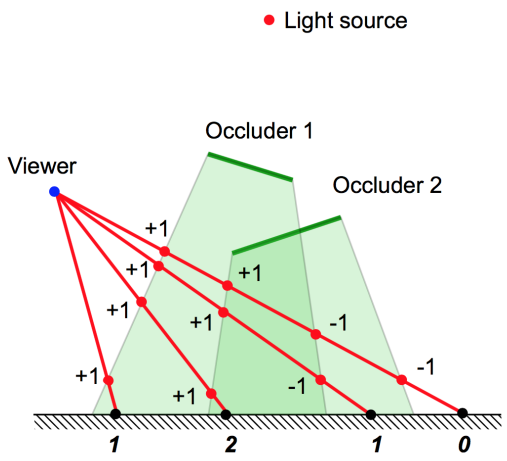

FIGURE 3.2 – Shadown volume
Source : https ://hal.inria.fr/inria-00281388

L'algorithme de shadow volume peut être divisé en deux étapes.

Tout d'abord, il s'agit de trouver les contours de chaque objet chargé dans la scène depuis le point de vue de la lumière. Puisque le point de vue se trouve à la position de la lumière, les contours à trouver sont toutes les arêtes communes à un triangle se trouvant dans la lumière et un triangle se trouvant dans l'ombre. L'algorithme fonctionne en ajoutant les arêtes des triangles situés dans l'ombre dans une liste et en les enlevant de la liste si ils sont déjà dedans. De cette manière, seules les arêtes qui ne sont pas communes à deux triangles dans l'ombre (donc qui sont communes à un triangle dans l'ombre et un dans la lumière) sont conservées. Pour déterminer si un triangle est dans l'ombre ou la lumière, on calcule le vecteur normal de ce triangle et on fait le produit scalaire de ce vecteur avec le vecteur de direction de la lumière. Si le résultat est positif, le triangle est dirigé dans le même sens que la lumière et donc se trouve sur le coté opposé à la lumière. Une fois que les arêtes définissant les contours sont déterminées, on les projette à l'infini dans la direction opposée à la lumière ce qui crée une homothétie du contour. Le contour de l'objet et son homothétie délimitent alors un volume, appelé shadow volume ou volume d'ombre. Nous avons commencé par implémenter cette étape en Python avant de nous rendre compte que la puissance de calcul nécessaire était trop grande. Cette étape est donc implémentée dans une fonction en C++ appelée depuis le code Python.

Une fois que ces volumes sont générés, il suffit de déterminer quels objets ou parties d'objets se situent dans un ou plusieurs volumes, et de ce fait, dans l'ombre. Pour ce faire, le rendu de la scène est effectué une première fois sans la lumière pour dessiner la scène dans l'ombre. Ensuite le rendu de la scène est effectué deux fois en activant le stencil test dans OpenGL et en désactivant l'écriture sur le color buffer et le depth buffer, de manière à ne pas modifier l'image affichée. La première fois, on incrémente la valeur du stencil buffer chaque fois que l'on rentre dans un shadow volume (en activant le back-face culling). La deuxième fois on le décrémente chaque fois que l'on en sort (avec le front-face culling). Pour chaque pixel de la scène, on obtiendra alors une valeur valant soit zéro pour un point situé dans la lumière soit une valeur plus grande représentant l'intensité de l'ombre. Un nouveau rendu est alors effectué avec la lumière uniquement aux endroits où le stencil buffer est égal à zéro.

3.2.2 Shadow mapping

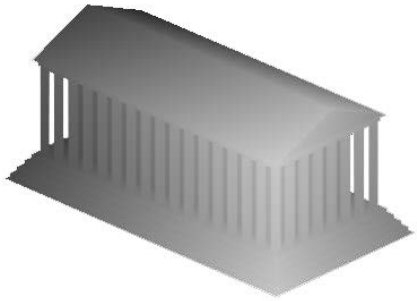

FIGURE 3.3 – Shadown mapping
Source : wikipedia.org

L'algorithme de shadow mapping se déroule également en deux étapes.

La première étape consiste à générer une shadow map d'une résolution choisie. Une shadow map est une représentation depuis le point de vue de lumière de la profondeur de chaque point de la scène. Pour ce faire, OpenGL offre la structure d'un depth buffer, permettant de calculer la profondeur de chaque pixel visible depuis la lumière en effectuant le rendu.

Dans la seconde étape, on effectue le rendu de la scène depuis la caméra. On considère pour chaque pixel sa distance à la lumière par rapport à la profondeur au même endroit dans la shadow map. Si la première est plus grande que la seconde, alors ce pixel est dans l'ombre et sa couleur est assombrie. On simule également la réflexion spéculaire en prenant en compte le vecteur normal de chaque pixel par rapport au vecteur direction de la lumière. Plus un pixel est tourné vers la lumière, plus sa couleur est illuminée.

Si la scène considérée contient plusieurs sources lumineuses, le principe d'implémentation reste identique. Il suffit alors de réaliser différentes shadow map, chacune avec le point de vue d'une lumière. Ces différentes ombres sont superposées en assombrissant un pixel proportionnellement au nombre d'ombres dans lequel il se trouve.

Un apport a été effectué pour approcher d'une soft shadow en utilisant une méthode de dispersion des points d'ombre à l'aide d'un disque de Poisson, ce qui floute les contours de l'ombre et ajoute au réalisme.

3.2.3 Ray tracing

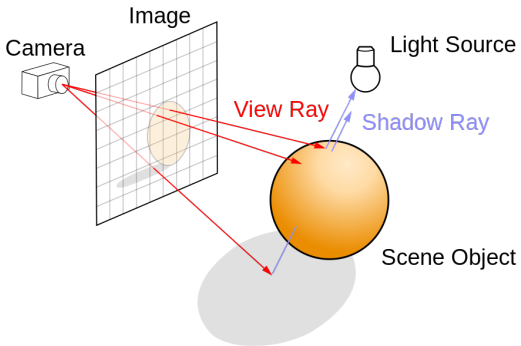

FIGURE 3.4 – Ray Tracing
Source : wikipedia.org

Le ray tracing consiste à tracer un segment de droite depuis le point de vue de l'observateur jusque chaque pixel de la scène. Ce segment de droite simule un rayon de lumière en sens inverse. A partir de ce pixel, on lance un rayon vers l'ensemble des sources lumineuses de manière à déterminer sa luminosité. En effet si un rayon rencontre un autre objet avant de rencontrer une source lumineuse, cela signifie que cet objet est dans l'ombre par rapport à cette source lumineuse.

On peut noter que cette technique est fonctionne a rebours de la réalité physique ou le rayon lumineux par de la lumière pour arriver à l'oeil de l'observateur. Elle permet donc une très bonne simulation du comportement de la lumière mais n'est pas applicable en temps réel. C'est pourquoi la scène où nous avons implémenté le ray tracing n'est pas interactive.

Cependant, l'environnement de test est trop lourd et complexe pour du temps réel avec cet algorithme et nous avons donc créé une scène spéciale non interactive avec deux billes qui tournent en rond. Ces deux billes ont la propriétés d'être très simple et n'être composé que de très peu de primitive (triangles). Nous nous sommes aussi limité aux ombres des objets sur le sol. L'ombre de la première bille n'est donc pas visible sur la deuxième et inversement.

Chapitre 4

Résultats expérimentaux

4.1 Les ombres sont indispensables

Dans notre environnement de test, nous avions implémenté deux mises en scènes, une sans ombre et une avec ombre.

Il a été remarqué que l'absence d'ombres privait non seulement de réalisme les scènes considérées mais également d'informations quant au relief et à la position des objets et des personnages que présente la scène. En effet si on prend par exemple la scène ou l'on voit un personnage le fait de ne pas avoir d'ombre semble montré au spectateur que le personnage est un dessin 2D sans relief peu esthétique.

Après avoir ajouté uniquement l'ombre que les objets et personnages créent sur eux-même, l'information quant au relief est apportée mais pas celle de position. Grâce a cela, on a pu confirmer que c'est donc bien l'ombre d'un élément de la scène sur l'autre et non sur lui-même qui permet à l'observateur de déterminer sa position relative.

4.2 Les différents types d'ombre

D'après nos résultats subjectif, les algorithmes hard shadow

ICI ICI ICI. NON PAS LA. ICI!!

4.3 Résultats de chaque algorithme

4.3.1 Benchmarking

	FPS	CPU	Mémoire
Aucune ombre	400	20%	3,2%
Auto ombre	225	28%	3,2%
Ray tracing	?	?	?
Shadow mapping	250	30%	3,2%
Shadow volume	120	20%	3,2%

4.3.2 Shadow volume

Cet algorithme a été difficile à implémenter et ne fonctionne toujours pas entièrement. La partie la plus demandeuse en temps de calcul et en mémoire, à savoir la création des shadows volume à été réalisée mais les ombres ne s'affichent pas correctement. La difficulté réside dans le fait que cet algorithme utilise extensivement les possibilités de la carte graphique et donc nécessite des connaissances en OpenGL que nous ne maîtrisons pas encore bien. Nous pouvons donc évaluer les performances de cet algorithme mais pas son réalisme. L'étape la plus lente est de créer les shadows volumes mais, ne dépendant pas de la

position de la caméra, cette étape est réalisée uniquement quand le nombre ou la position des lumières est modifiée. Ceci permet d'obtenir un nombre de *fps* inférieur au shadow mapping mais bien suffisant pour que l'image soit fluide. La deuxième étape prend un temps négligeable mais n'est pas encore au point pour le moment. Néanmoins, l'algorithme montre déjà des self-shadows nettes et la forme de l'ombre au sol mais pas l'ombre correcte.

4.3.3 Shadow mapping

Comme prévu, cet algorithme est le plus efficace des trois algorithmes considérés. Le résultat obtenu obtient un nombre de *fps* plus que correct. Le réalisme offert dépend en grande partie de la résolution imposée de la shadow map. Une grande shadow map implique une grande précision mais également un plus grand nombre de calculs à réaliser. Notre implémentation montre donc une ombre correcte. La résolution de la shadow map, influençant la précision de l'ombre peut toutefois être changée, pour offrir une ombre plus ou moins pixelisée. Puisque la shadow map est calculée du point de vue de la lumière, plus celle-ci est loin de la scène, plus la résolution nécessaire pour obtenir une bonne précision est grande. Les ombres sont crénelées à cause de la résolution limitée de la shadow map. Pour résoudre ce problème, on peut ajouter un biais lors du sampling dans la shadow map mais si ce biais devient trop grand, les ombres "flottent". On peut également ajouter du réalisme en modifiant la répartition du disque de Poisson de manière à flouter les contours mais en diminuant trop la répartition, on voit apparaître des nuages de points dans l'ombre. Quand on augmente trop la répartition, le floutage disparaît. Cet algorithme doit donc etre adapté au type de scène (objets sphériques, parallélipipèdiques,...) et de lumière.

4.3.4 Ray tracing

Les ombres obtenues par cette algorithme sont très réalistes, étant donné le fait qu'il considère chaque pixel de la scène indépendamment et qu'OpenGL effectue automatiquement une interpolation des pixels pour réduire le crénelage. L'ombre est donc appliquée sur chaque pixel un à un, et cet algorithme offre donc une précision au pixel près. Le ray tracing permet un excellent rendu d'ombre mais on ne peut pas en comparer correctement ses performances avec les autres algorithme car il est le seul à avoir sa propre scène. Cependant, on peut quand même remarqué que l'algorithme n'a pas de problème de performance pour une scène simple. Les ombres affichés au sol sont précises et dénués de glitchs.

Chapitre 5

Discussion

5.1 Comparaison des algorithmes

	FPS	CPU	Mémoire
Aucune ombre	400	20%	3,2%
Auto ombre	225	28%	3,2%
Ray tracing	?	?	?
Shadow mapping	250	30%	3,2%
Shadow volume	120	20%	3,2%

Il ne nous est pas possible de comparer le Ray Tracing car nous n'avons pas réussi à faire en sorte qu'il crée l'ombre en temps réel dans notre environnement de test. Ceci n'est pas étonnant sachant qu'il fait un très grand nombre de calcul. Pour rappel, il lance un rayon de chaque point observé vers chaque point lumineux.

Cependant, il permet donc d'obtenir les meilleurs résultats lorsque les performances ne sont pas un facteur limitant, comme par exemple pour des animations qui ne sont pas en temps réels, ou même des images fixes. La technique du ray tracing peut reproduit de manières exact des phénomènes physique comme la refraction et la reflexion et des phénomènes optiques tels que les caustiques, l'illumination globale ou encore la dispersion lumineuse.

Sans surprise, la scène sans aucune ombre offre les plus grandes performances. Néanmoins, le niveau de réalisme est bas. Quand on compare la même scène sans ombres et avec des auto-ombres, on ne remarque pas qu'il s'agit des mêmes objets. La première semble en deux dimensions alors que la deuxième montre clairement le relief. En comparant avec l'algorithme de shadow mapping ou de shadow volume, la scène prend une toute autre dimension. On voit clairement le relief et la position de chaque objet. Les scènes sans ombres peuvent eêtre utilisées dans le cas où le matériel n'est pas du tout performant mais actuellement la puissance de calcul est suffisante pour afficher un minimum d'ombres.

L'algorithme d'auto-ombre ne diminue pas beaucoup les performances par rapport à l'algorithme sans ombre car les calculs sont simples et réalisés par la carte graphique. Il peut donc servir de base pour apporter du réalisme sur des plate-formes peu puissantes.

Le plus performant mais également le moins précis des trois algorithmes que nous avons ici considéré est le shadow mapping. Celui-ci est toutefois le seul a offrir plusieurs niveaux de précision en fonction de la résolution qui est choisie pour la shadow map. Sa performance est également très bonne et nous permet d'envisager de créer des scènes complexes sans soucis de fluidité.

Shadow volume

5.2 Cas d'utilisation des algorithmes

5.2.1 Ray Tracing

Le ray tracing est utilisé dans des cas ou une excellente qualité de représentation d'ombre est nécessaire. En effet, il offre une précision au pixel près. Cependant cette précision a un coût, celui des performances. Il faudra en effet un matériel pouvant soutenir la quantité de calculs nécessaire pour ce type d'algorithme. de plus, la scène pour laquelle les ombres doivent être ajoutées doit être d'une complexité limitée.

5.2.2 Shadow mapping

5.2.3 Shadow volume

5.3 Alternatives

D'après la littérature, il n'existe pas d'alternative à ces trois algorithmes. En effet, chaque algorithme existant se base sur un des trois utilisé ci dessus et tente de l'améliorer ou de modifier certaines propriétés en fonction des désidérata de l'utilisateur.

5.4 Approches utilisées dans l'industrie

Nous allons illustré cette section par deux exemples :

5.4.1 La précision du Ray Tracing

La technique du ray tracing peut reproduit de manières exactes des phénomènes physique et optique. Il est donc beaucoup utilisé pour ce qui nécessite un rendu de haute qualité comme par exemple pour l'édition d'image du logiciel Adobe Photoshop. Le nom de leur logiciel est Adobe Ray Tracer.

5.4.2 Le compromis du Shadow volume

Moins précis que le ray-tracing mais également plus performant et plus adapté aux représentations en temps réel, l'algorithme de Shadow Volume offre quant à lui une excellent rapport entre performances et précision. C'est pour cette raison que cet algorithme est utilisé dans des logiciels d'animation nécessitant du temps réel, comme le logiciel utilisé chez Pixar, PhotoRealistic RenderMan.

Chapitre 6

Conclusion et perspectives

Conformément aux informations qui avaient été extraites de la littérature scientifique, les ombres apportent des informations quant à la position des différents éléments d'une scène 3D, ainsi qu'un apport de réalisme nécessaire aux animations et aux jeux vidéo. Sans ombre, il n'y a pas non plus de perception de relief pour une scène 3D. Le relief est pourtant crucial dans une animation ou un jeu vidéo car sans lui, la 3D ne sera pas perçue.

Les deux types d'ombres existant ont été considérés dans cette étude. Les hard shadows comme les soft shadows. Les premières sont les ombres qui sont celles dues à une seule source lumineuse ponctuelle. Elles sont donc quasi inexistantes dans la nature. En comparaison avec celles-ci, les soft shadows, ombres diffuses, sont abondamment plus réalistes. En effet, elles sont le résultat d'éléments placés dans un environnement illuminé de force sources lumineuses ponctuelles, comme retrouvés dans la nature. Au fil du temps, les sociétés d'animation et de développement de jeux vidéos se sont tournés des hard shadows vers les soft shadows lorsque le matériel le permettait, dans le but d'obtenir une hausse de réalisme.

Nous avons ici implémenté et comparé 3 différents algorithmes d'implémentation d'ombres de manière informatisée. Le premier type d'algorithme est le ray tracing. Celui-ci est le plus intuitif car il consiste à réaliser des calculs relatifs aux ombres pour chaque pixel vu par la caméra. Le second type d'algorithme considéré ici est celui du shadow volume. Celui-ci approche le problème différemment. En effet, il crée, en extrudant les contours des éléments de la scène, des volumes dans lesquels tout point est alors considéré dans l'ombre, alors que tout point n'étant pas dans un volume est considéré comme illuminé. Enfin, nous avons étudié ici le shadow mapping. Ce dernier est constitué de deux principales étapes. Lors de la première étape, une shadow map est créée. Celle-ci est en fait une carte de profondeur, sur laquelle est indiquée la profondeur de chaque partie de la scène depuis la lumière. Avec cette carte de profondeur, la profondeur des différentes parties de la scène peut alors être comparée et il est alors possible de déterminer la position relative de ces parties.

Bibliographie