Dédicace

Avec tout respect et amour je dédie ce modeste travail :

Un grand merci au Dieu qui m'a donnée l'espoir de vivre, et qui m'a donné deux belles perles ma mère et mon père, qui n'ont pas hésité de m'encourager et m'aider.

À mon frère, mes sœurs et à toute la famille.

À mes adorables amis et mes collègues de promotion 2018.

Pour toutes ces personnes et d'autre, je dédie mon mémoire en singe de respecte et de reconnaissance dont je passe toutes mes sincères gratitudes et salutations.

Enfin mes remerciements pour toi qui es en train de lire ces mots.

Nahla

Remerciements

Avant toute chose, je remercie le bon ***Dieu*** *de m'avoir donné toute cette force et le courage de le mener à terme.*

Je remercie, avant toute personne, mon encadreur Monsieur ***BELAICHE Hamza****, pour ses précieux conseils et surtout pour sa disponibilité et sa grande aide.*

Je remercie également les membres du ***jury*** *d'avoir accepté l'évaluation de ce travail.*

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

J'exprime ma profonde reconnaissance à tous ceux qui m'ont aidé à l'élaboration de ce mémoire.

On n'oublie pas mes parents pour leur contribution, leur soutien et leur patience.

Enfin, j'adresse mes plus sincères remerciements à tous mes proches et amis, qui m'ont toujours encouragée au cours de la réalisation de ce mémoire.

Merci à tous et à toutes !

ABDERRAHMANI Nahla

Table des matières

Chapitre 1: Contexte de travail

Chapitre 2: Etat de l'art

Chapitre 3: Conception et mise en œuvre

Chapitre 4: Résultats

Liste des figures

Liste des tableaux

Introduction générale

De nos jours, l'imagerie en trois dimensions est de plus en plus présente dans notre quotidien. Que ce soit de la simple affiche publicitaire au film d'animation, ces images s'insinuent sur nos écrans, dans nos rues et sur nos murs. Pourtant l'informatique graphique est une science moderne dont les débuts datent de moins de 40 ans. L'avènement de nouveaux médias tels que des jeux vidéo ou d'internet a permis de développer ce domaine dont le but, aujourd'hui, est d'obtenir des images photoréalismes afin d'immerger un spectateur dans un monde virtuel. Le réalisme obtenu est grandissant, surprenant, et il n'est pas souvent aisé de distinguer la réalité de la virtualité.

L'image est ainsi un moyen de communication universel dont la richesse du contenu permet aux êtres humains, de tout âge et de toute culture, elle est utilisée, dans tous les domaines, comme un moyen pour exprimer quelques réalités et présenter quelques informations.

Le concept de l'ombre est inséparable de la notion de la lumière, comme vous avez besoin de lumière afin de jeter une ombre. Il existe de nombreuses techniques qui génèrent des ombres.

Le calcul d'ombre dans une application d'infographie est une tâche très coûteuse en temps de calcul. Cela dépend de la méthode de rendu, l'algorithme d'ombre utilisé, le matériel disponible, la qualité de l'ombre, et la complexité de la scène. La génération d'une image ombragée peut être une tâche très longue, elle peut prendre plusieurs millisecondes jusqu'à plusieurs minutes ou même heurs.
Bien que le calcul des ombres soit relié au calcul de visibilité, un problème omniprésent en infographie, il a ses particularités, ce qui a mené au développement de plusieurs algorithmes et structures dédiés à la résolution du problème du calcul d'ombres.

La technique du shadow mapping est décrite par Williams, Le calcul d'ombre consiste à identifier les parties de la scène qui sont cachées de la source de lumière. Cela revient donc intrinsèquement à un calcul de visibilité selon le point de vue de la lumière.

Le principal avantage de cette méthode est sa simplicité, et son universalité. En effet, le fait que l'algorithme soit basé sur une technique image lui permet une totale indépendance vis-à-vis de la scène, quel que soit la richesse de celle-ci ou la complexité des objets qui la composent [41].

Objectif de ce travail :

Ce travail va essayer de répondre aux objectifs suivants :

- Mettre l'accent sur le fondement théorique du calcul d'ombre dans une application d'infographie ;
- Réaliser l'ombrage d'un relief naturel par la technique de Shadow mapping.

Axes du mémoire :

Pour assurer les objectifs de notre travail, nous organisons ce mémoire en quatre chapitres :

- **Chapitre 1 « Contexte de travail» :** dans ce chapitre, nous allons exposer des connaissances sur l'illumination et l'ombrage et la représentation et la modélisation des scènes naturelles ;
- **Chapitre 2 « Etat de l'art » :** ce chapitre va présenter les techniques de génération des ombres en temps réel et les techniques de rendu des terrains ;
- **Chapitre 3 « Conception et mise en œuvre » :** à base des deux premiers chapitres, ce chapitre va présenter la conception de notre application, où nous allons définir l'objectif de l'application et la mise en œuvre de notre système (les variables, structures de données, les méthodes utilisées) ;
- **Chapitre 4 « Résultats» :** dans ce chapitre, nous allons présenter les résultats du notre système.

Ces quatre chapitres seront suivis par une conclusion qui présente le bilan de ce travail et propose ses perspectives potentielles.

Chapitre 1:

Contexte de travail

Introduction

La lumière à travers son meilleur représentant, le soleil, est une source de vie et de création. Mais d'un aspect plus pratique, elle nous est aussi complètement indispensable dans la vie de tous les jours. Sans elle nous sommes aveugles et c'est pourquoi de nombreuses sources de lumière artificielle ont été développées. L'éclairage donne une personnalité à une pièce. La présence de lumières ou d'ombres résultant de l'éclairage permet de mieux distinguer leur forme ou leur donner une apparence plus dramatique. Les ombres jouent un rôle important dans la perception d'une scène 3D. Elles fournissent des informations importantes sur la forme et la position des objets. [1]

Dans ce chapitre, nous introduisons le domaine de l'illumination et des ombres, en donnant des définitions sur les termes les plus utilisés tels que la lumière, l'ombrage et l'ombre et en présentant également une brève description sur les modèles d'éclairement et sur les types de sources lumineuses existantes. Nous détaillons ensuite les techniques pour la génération des ombres, et nous ferons une synthèse sur les travaux de recherche liés à ce domaine.

1. Les sources de lumière

Une source lumineuse est un objet duquel on reçoit un rayonnement lumineux. Cet objet peut soit produire sa propre lumière ou encore refléter le rayonnement d'une autre source [11].

1.1. Les types des sources de lumière

1.1.1. Lumière ponctuelle

Une source de lumière ponctuelle est une source dont les rayons partent tous du centre de la source. La lumière ponctuelle permet de comprendre les différents types de réflexion de lumière d'un objet.

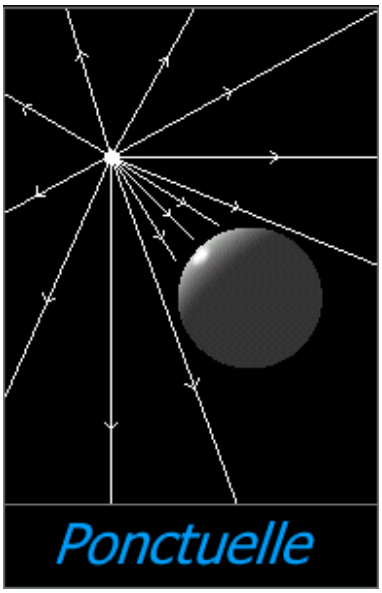


Figure 1: Lumière ponctuelle [6]

Il existe 3 types de réflexion de lumière : la lumière spéculaire, la lumière diffuse et la lumière ambiante. [6]

- ✓ **La lumière spéculaire** est la couleur que l'objet renvoie lorsqu'il est directement frappé par la lumière ;
- ✓ **La lumière diffuse** est la couleur qu'émet graduellement l'objet lorsqu'il frappé par la lumière ;
- ✓ **La lumière ambiante** est la lumière projetée dans tout l'environnement. C'est donc la couleur lorsque qu'il n'y a aucune autre lumière projetée l'objet. [6]

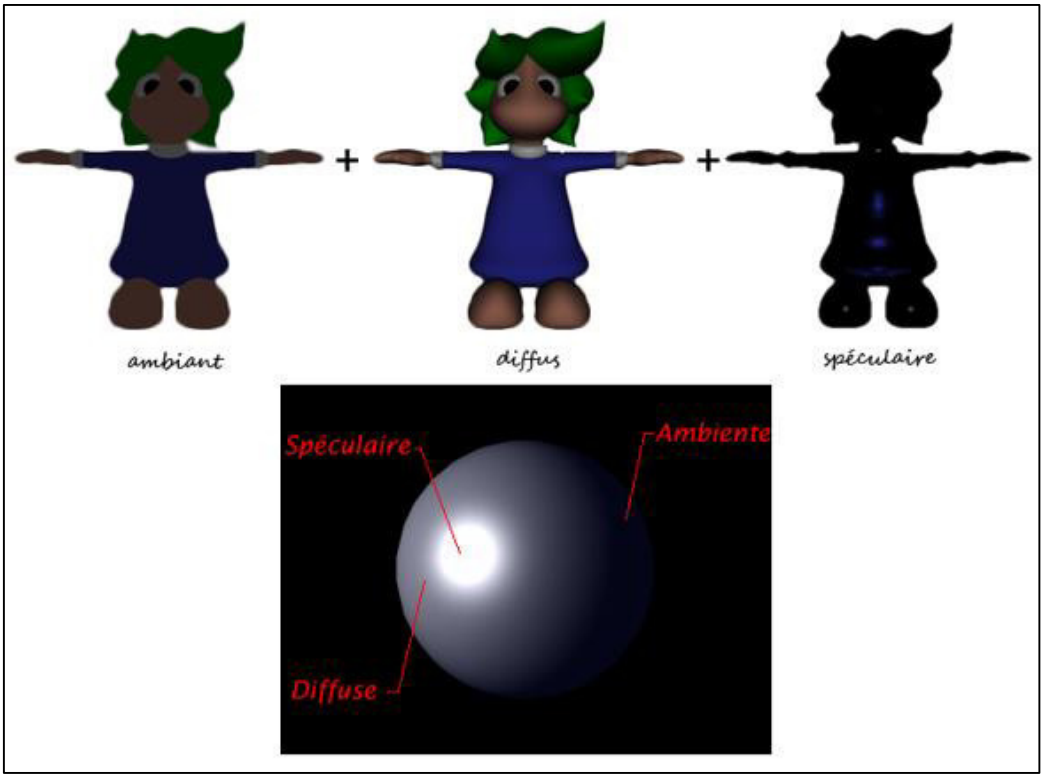


Figure 2: 3 types de réflexion de lumière. [6]

1.1.2. Lumière directionnelle

Une source de lumière directionnelle est une source dont les rayons ont la même direction en tous points de l'espace, par exemple le soleil puisqu'il est tellement loin que tous les rayons de lumières projetés sur la terre sont considérés comme parallèles. [6]

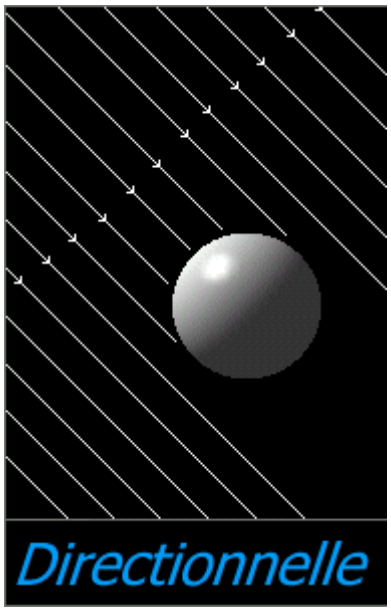


Figure 3: Lumière directionnelle. [6]

2. Illumination

Lors de la formation d'une image, le calcul de la couleur et de l'intensité lumineuse s'effectue généralement en deux étapes. Une première étape consiste à estimer globalement les échanges lumineux dans la scène, afin de connaıtre la quantité de lumière (intensité et couleur) parvenant jusqu'à chaque point de chaque surface. Il s'agit de calculer l'éclairage global dans la scène. Si on veut prendre en compte les échanges lumineux globaux, une estimation peut être pré–calculée ou même déterminée lors du dessin. Une seconde étape va calculer localement comment la surface réagit à la lumière reçue. Ce calcul est effectué par le modèle d'illumination locale. A cette fin, celui–ci utilise les propriétés locales du matériau (couleur, brillance, etc. ...), qui peuvent varier le long de la surface : ce sont précisément les paramètres gères par les modèles d'habillage. [2]

2.1. Modèles d'illumination

Deux types de modèles existent : les modèles d'illumination locale qui ne considèrent que les interactions sources-objets. Et des modèles d'illumination globale lesquels on considère en plus les interactions objets-objets.

2.1.1. Illumination locale

Un modèle d'illumination locale prend en entrée: l'éclairage incident et les propriétés du matériau en un point de la surface. Il modélise la réaction de la surface à la lumière et produit en sortie la lumière renvoyée par l'objet. Ici, le terme lumière regroupe à la fois la couleur et l'intensité lumineuse. [3]

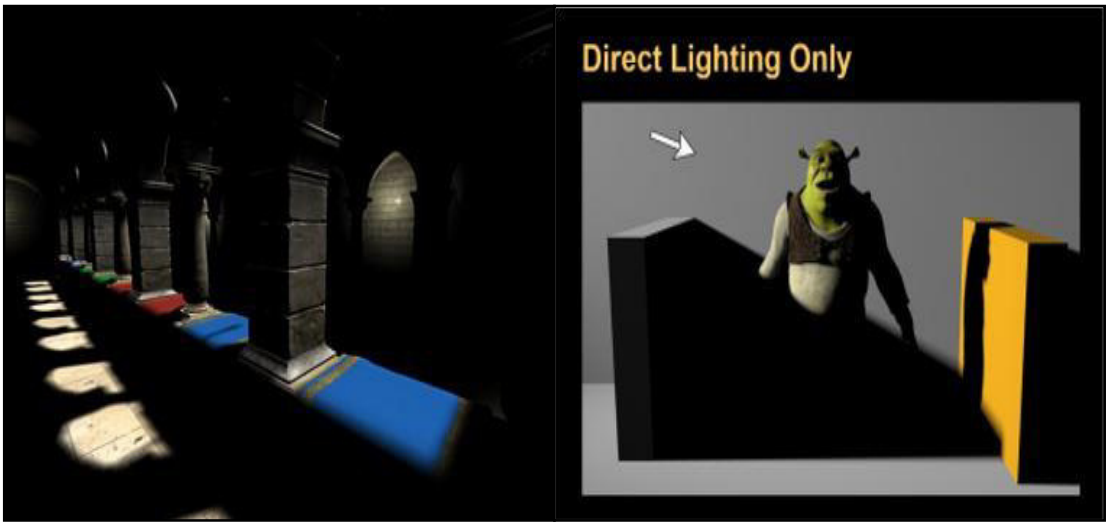


Figure 4: Illumination locale. [5]

2.1.1.1. Modèle d'éclairement de Phong

Il existe de nombreux modèles d'illumination locale, le modèle le plus fumeux est le modèle défini par B.T. PHONG en 1975. Il est un modèle spéculaire classique. Il définit l'intensité lumineuse réfléchie en un point *P* d'une surface en fonction de l'intensité des sources lumineuses. Ce modèle est une généralisation de la loi de Lambert sur la réflexion de la lumière. Cette loi, aussi appelée loi du cosinus, décrit l'intensité diffusée par un point d'une surface mate en fonction de l'intensité et de la direction de la source prise en compte. [1]

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

Equation 1 : *Loi de Lambert*

Où: $I(P)_{diffusé}$ = Intensité diffusée au point P

$K_d(P)$= Coefficient de diffusion au point *P*

I_{source} = Intensité de la source

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

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

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

Equation 2 : *Loi de réflexion sur une surface brillante*

Où: $I(P)_{réfléchie}$= Intensité réfléchie au point P

I_{source}= Intensité de la source

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

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

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

n = Exposant de Phong

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

$$I(P) = K_d I_{ambiant} + \sum_{i \leq N} k_d \,.\cos\alpha_i . I_i + \sum_{i \leq N} k_s(\alpha_i).\cos^n \beta_i . I_i$$

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

N = Nombre de sources lumineuses dans la scène

Figure 6: Exemple de modèle de phong

2.1.2. Illumination globale

La lumière provient directement des sources de lumière ainsi que de la lumière distribuée entre les surfaces (lumières secondaires). La recherche d'un modèle représentant l'illumination globale d'une scène passe par la modélisation de toutes les interactions entre les objets [1] : réflexions d'une surface sur une autre, occultation de la lumière par une surface (ombres) de cette scène [12]. Ces interactions sont dépendantes de la réaction locale d'un objet face à une onde lumineuse. [1]

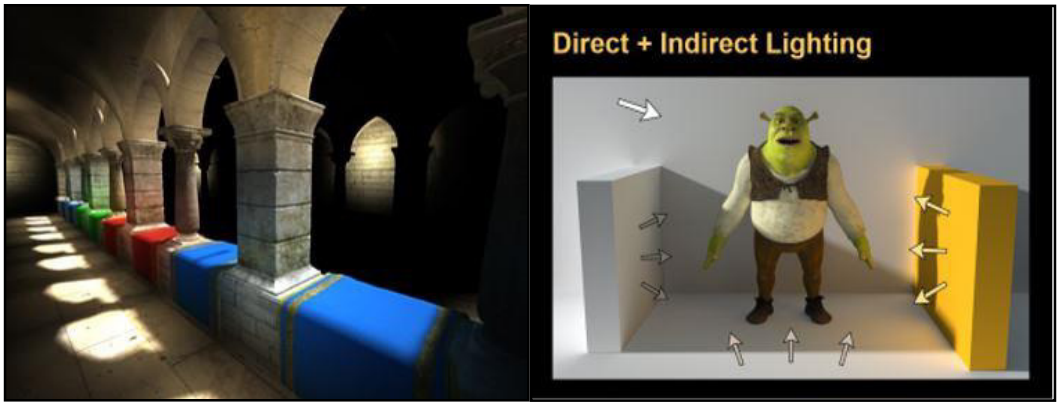


Figure 7: Illumination globale [5]

Deux classes de méthodes existent qui sont basées sur des modèles d'illumination globaux :

a. La première classe concerne les algorithmes de lancer de rayons qui remplacent les illuminations locales ambiantes, diffuses et spéculaires par un modèle à base de réflexion spéculaire et transmission globale ;

b. La deuxième classe concerne les méthodes de radiosité qui modélisent toutes les interactions entre objets par des sources lumineuses dans une étape préliminaire. Une image est ensuite déterminée pour un point de vue donné par des algorithmes classiques d'élimination de parties cachées et d'interpolation d'ombrages. [12]

2.1.2.1. Lancer de rayons

Les algorithmes de lancer de rayons (Eng: ray casting, ray tracing) permettent de déterminer l'illumination ainsi que la visibilité des surfaces en traçant des rayons imaginaires du point d'observation vers la scène. Dans sa version la plus simple, un rayon est lancé pour chaque pixel de l'image, et la couleur du pixel est déterminée à partir de l'illumination du point de la surface intersectée par le rayon qui est la plus proche de l'image. On peut pour cela utiliser les modèles d'illuminations locaux présentés précédemment. [12]

Figure 8 : Image obtenue par l'algorithme de Lancer de rayons [13]

2.1.2.2. Radiosité

Les méthodes de radiosité [1984] apportent une modélisation plus précise des inter-réflexions entre objets. Toutes les énergies lumineuses émises ou réfléchies par chaque surface sont prises en compte. La radiosité caractérise le taux d'énergie quittant une surface et correspond à la somme des taux d'émission de la surface, de réflexion et de transmission par d'autres surfaces. Une différence majeure avec les méthodes de lancer de rayons est que ces méthodes pré-calculent les interactions lumineuses pour un environnement donné indépendamment du point de vue. Les images pour différents points de vues sont générées ensuite. [12]

Figure 9 : Implémentation simple du rendu de radiosité [15]

3. Répartition de la lumière

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

- l'ombrage plat (flat shading) ;
- l'ombrage de Gouraud (Gouraud shading) ;
- l'ombrage de Phong (Phong shading).

3.1. Ombrage plat

La méthode d'ombrage la plus simple pour les facettes polygonales est l'ombrage plat (ou constant) [12]. L'idée est de calculer une seule valeur d'illumination pour l'ensemble de la facette. Par exemple au point milieu de la facette en prenant pour normale à la surface celle du plan contenant la facette. Cette approche est valide par rapport au modèles d'illumination vus précédemment lorsque :

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

3.2. Ombrage de Gouraud

La méthode développée par Gouraud en 1971 élimine les discontinuités d'intensité sur une facette polygonale par interpolation des valeurs d'intensité aux sommets de la facette. Cette méthode est largement utilisée et se retrouve dans la majorité des matériels graphiques existants (librairies, cartes graphiques) [12].

Cette méthode requiert la connaissance de la normale à la surface aux sommets des facettes polygonales. Lorsque les normales sont connues, les intensités aux sommets des facettes polygonales sont calculées. L'interpolation peut ensuite être effectuée à l'aide de l'algorithme de balayage de ligne utilisée pour le remplissage de polygone et le z-buffer [1].

3.3. Ombrage de Phong

L'ombrage de Phong consiste à déterminer la normale en un point d'une facette polygonale par interpolation des normales aux sommets de cette facette.

L'intérêt de cette approche par rapport à l'ombrage de Gouraud réside principalement dans sa capacité à traiter les réflexions spéculaires. Gouraud ne permet pas, en effet, de prendre en compte les réflexions spéculaires lorsque celles-ci sont localisées au centre d'une facette [12].

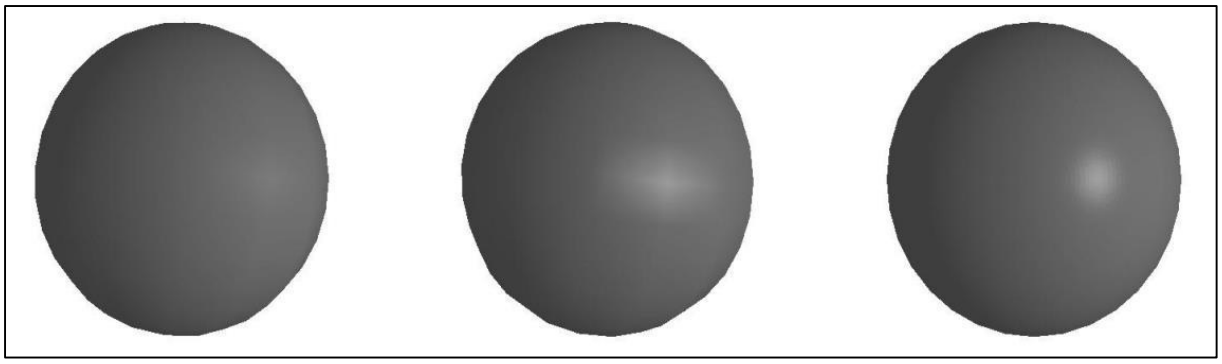


Figure 10 : (a), (b) modèle d'illumination de Phong et ombrage de Gouraud : deux vues différentes ; (c) modèle d'illumination de Phong et ombrage de Phong [12].

4. Les Ombres

Les ombres sont tout simplement nécessaires au réalisme d'une image de synthèse. En effet, la présence des ombres dans une scène fournit des informations essentielles pour la compréhension de celle-ci et notamment sur la position relative des objets entre eux et des sources lumineuses, toute perception de profondeur est impossible sans ombre. [7]

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

Une ombre est une zone 3D où la lumière est bloquée par un objet inséré entre la source de lumière et la surface sur laquelle elle se réfléchit. Les ombres sont utiles à la visualisation d'une image, car elles apportent de l'information sur la forme, la position et les caractéristiques d'un objet, ainsi que sur la position, intensité et taille des lumières. Les ombres se classent tout d'abord en trois catégories bien distinctes :

- **L'ombre propre** : est la zone de l'objet qui ne reçoit pas de lumière. Il s'agit de la partie de l'objet située à l'opposé de la source de lumière ;
- **L'ombre portée** : se situe sur une surface située derrière l'objet (écran, mur, sol etc.) et qui ne reçoit pas de lumière. Cette ombre possède une forme qui reproduit les contours de l'objet éclairé ;
- **Pénombre** : est une zone qui est partiellement éclairée par une source de lumière. [7]

4.2. Types d'ombres

Deux types d'ombres existants:

- **Les ombres directes:** qui correspondent aux zones sombres résultant de l'occultation directe d'une source de lumière par un objet ;
- **Les ombres indirectes:** qui correspondent aux zones sombres résultant de l'occultation de rayonnements indirects par un objet. [7]

4.2.1. Types d'ombres directes

Parmi ces ombres directes, on pourra encore distinguer deux types d'ombres :

- **Les ombres franches « dures » (hard-shadow)** : engendrées par des lumières ponctuelles ou directionnelles. Le principe sur lequel est basée ce rendu d'ombres est simple, il s'agit d'un test binaire : soit la source est visible depuis la surface, soit elle ne l'est pas et dans ce cas on se trouve dans une zone d'ombre. Ce principe laisse apparaître des changements brusques dans la visibilité de la source lumineuse, ce qui a pour conséquence de provoquer des ombres aux contours nets. En réalité, il n'existe pas dans la nature c'est pourquoi les ombres dures manquent de réalisme.
  Exemples des techniques utilisées: Les ombres projectives ou planaires, Les Shadow Maps, Les Shadow Volumes [7] ;

- **Les ombres douces (soft-shadow) :** engendrées quant à elles par les sources de lumière étendues (comme dans la réalité). Dans ce cas, on parlera de zone d'ombre, où la source de lumière est complètement occultée, et de zone de pénombre, où la source ici n'est vu que partiellement. Les ombres douces (taille de la zone de pénombre) dépendent de la taille de la source, de sa distance à l'objet, et de la distance de l'objet à la surface qui reçoit l'ombre. [7]
  Exemples des techniques utilisées: Filtrage, Gestion des sources étendues.

Figure 11 : Ombre dure vs Ombre douce. [7]

5. Génération des ombres

Le calcul d'ombre dans une application d'infographie est une tâche très coûteuse en temps de calcul. Cela dépend de la méthode de rendu, l'algorithme d'ombre utilisé, le matériel disponible, la qualité de l'ombre, et la complexité de la scène. La génération d'une image ombragée peut être une tâche très longue, elle peut prendre plusieurs millisecondes jusqu'à plusieurs minutes ou même heurs. [8]

Bien que le calcul des ombres soit relié au calcul de visibilité [9], un problème omniprésent en infographie, il a ses particularités, ce qui a mené au développement de plusieurs algorithmes et structures dédiés à la résolution du problème du calcul d'ombres. Les techniques de calcul d'ombres peuvent être, dans la majorité des cas, regroupées en 3 catégories : les techniques basées sur le lancer de rayons, les techniques d'illumination globale, ces techniques ne sont pas adaptés au temps réel, car elles sont très coûteuses en temps de calcul. Et les techniques basées sur les volumes d'ombre (basée objet) et finalement, celles basées sur les tampons de profondeur (depth buffers ou shadow maps) (basée image) [8], qui est l'objectif de notre travail, sont adapté en temps réel.

5.1. Volume d'ombre

Cette technique est, de nos jours de plus en plus utilisée surtout avec la sortie des derniers jeux comme Doom3 qui abuse de cette technique. Cette technique est encore en évolution : John Carmack, le concepteur de Doom3 y a apporté son amélioration pour palier à un manque, lorsque l'observateur se situe dans le volume d'ombre. [7]

Figure 12 : Ombre par Shadow Volume dans Doom 3 [18]

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 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. [18]

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

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

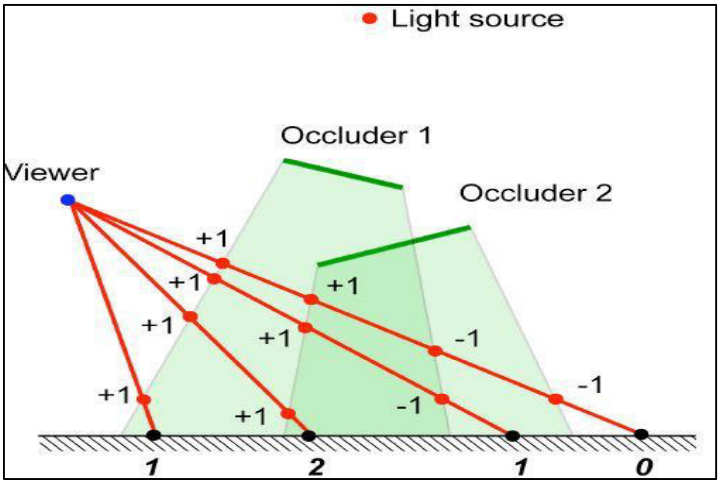


Figure 13 : Technique du shadow volume [18].

5.2. Shadow Mapping

Décrit par Williams, Le calcul d'ombre consiste à identifier les parties de la scène qui sont cachées de la source de lumière. Cela revient donc intrinsèquement à un calcul de visibilité selon le point de vue de la lumière. La technique du shadow mapping exploite ce parallèle. Le principe consiste en l'utilisation d'un Z-Buffer pour déterminer les zones d'ombres [19].

Figure 14 : Shadow mapping. À gauche, vue de la caméra. À droite, tampon de profondeur vu de la lumière, encodé en niveaux de gris. [10]

L'algorithme de shadow mapping

I. Premièrement, la scène est rendue du point de vue de la source de lumière. Les valeurs contenues dans le z-buffer sont alors sauvegardées, elles forment le tampon de profondeur des ombres (shadow map).Durant cette étape, pour chaque fragment, la profondeur A stockée dans la texture projetée est comparée à la distance B entre l'objet et la lumière.
   Si B > A alors la zone est dans l'ombre (figure 13) ;

II. Ensuite, un rendu de la scène est fait du point de vue de la caméra, en utilisant le shadow map pour déterminer les parties éclairées ou dans l'ombre. Pour déterminer si un point est dans l'ombre, on le projette dans le shadow map de la lumière et on compare la profondeur résultant de cette projection à la profondeur contenue dans le shadow map. Si la profondeur du shadow map est plus petite, le point est dans l'ombre, En revanche, si A ≈ B, alors ceci signifie qu'il n'y a pas d'ombre (figure 14).

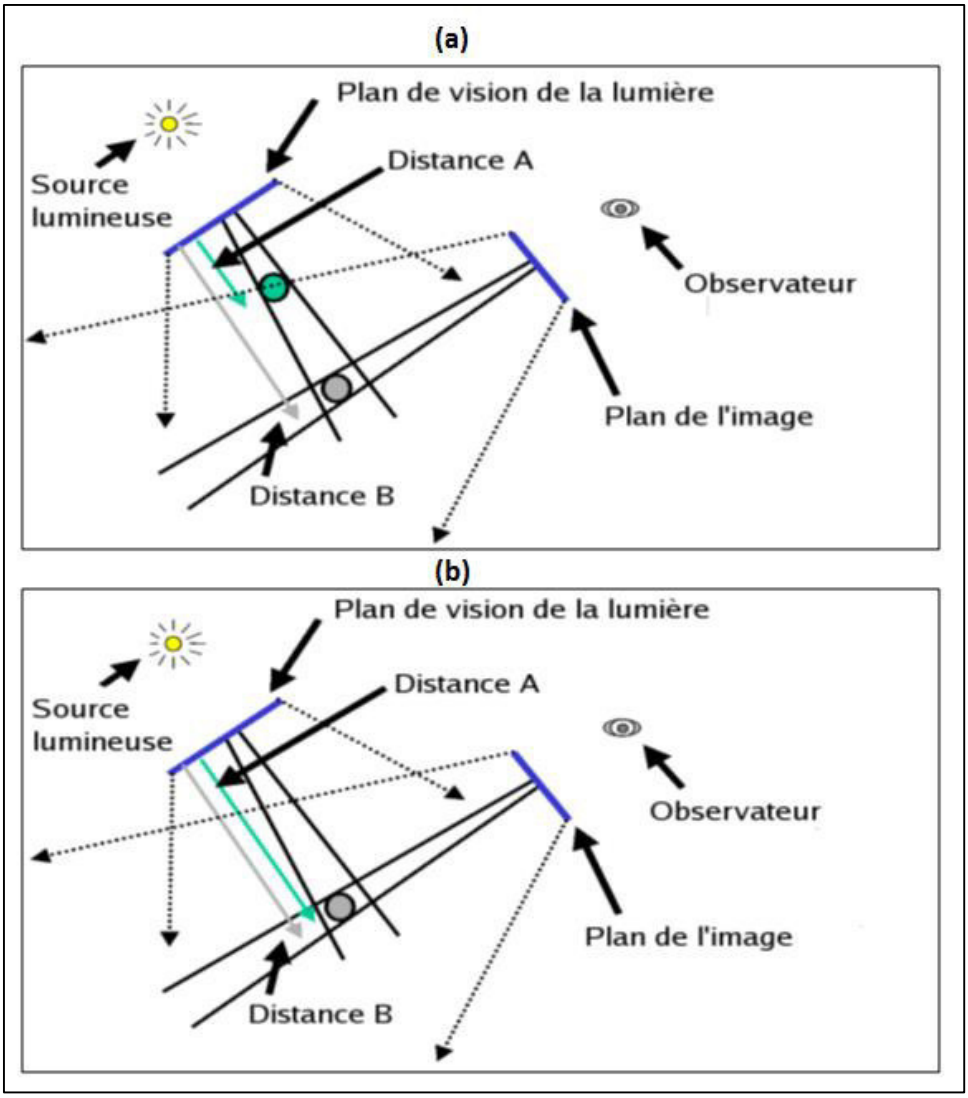


Figure 15 : (a) Cas A < B : la zone est dans l'ombre

(b) Cas A ≈ B : pas d'ombre [4]

6. Bilan

Nous présentons dans ce qui suit une synthèse des travaux présentés dans ce chapitre, en mentionnant leurs catégories communes, leurs avantages et leurs limites dans le tableau suivant :

	Avantages	Inconvénients
Shadow volume	+ Plus grandes précision des ombres. + Traite les cas d'auto-ombrage (self-shadowing). + Positions quelconques lumières/caméra. + Robuste si bien programmé. + Pas d'aliassage.	– Temps de calculs dépend de la complexité des occultants (calculs de la silhouette, extrusion). – Calcul de la silhouette (sur CPU très long). – Des modèles polygonales fermés (Scènes bien modélisées préférables).
Shadow mapping	+ Très simple à implémenter, code compact. + Marche toujours avec n'importe quelle scène (image texture). + Prix indépendant de la complexité de la scène.	– Deux (trois) rendus de la scène. – Problème d'aliassage. – Ne permet que de générer des ombres dures. – Au moins 2 passes sont nécessaires pour le rendu. – Besoin d'extensions OpenGL

Tableau 1.1 : Les avantages et les inconvénients de shadow map et shadow volume [1]

	type de données	**forme du récepteur**	**gestion de l'auto-ombrage**	**vitesse**	**implémentation**	**espace**
Volume d'ombre	géométrique	quelconque	Oui	- Le coloriage de l'ombre est lent (3 passes : 2 pour le calcul du stencil, 1 pour le remplissage). - Déterminer la silhouette est coûteux.	hardware	Objet
Shadow mapping	quelconque	quelconque	oui	correcte (2 passes), indépendante de la complexité de la scène.	hardware	Image

Tableau 1.2 : Synthèse sur les travaux de génération d'ombres [1]

7. Modélisations des scènes naturelles

Le traitement des scènes naturelles en synthèse d'image traite principalement la représentation et le rendu des terrains et des végétaux et leurs interactions à la lumière ainsi que la manière dont ils interagissent entre eux. [16]

7.1. Modélisations et représentations de terrains

La modélisation des terrains est un aspect très important dans plusieurs applications, la géographie, les jeux vidéo, les simulateurs de vols, l'aménagement du territoire etc. La complexité de modélisation et représentation des terrains provient de deux aspects essentiels, le premier aspect concerne les modèles de la génération et l'amplification des données qui définissent le terrain, et le deuxième est les techniques adéquates pour le rendu de ses modèles [16].

Deux grandes familles de méthodes de modélisation existent :

- Les méthodes extrayant des données réelles issues de mesures des reliefs (Modèle Numérique de Terrain), avec pour problèmes la prise de mesures et le stockage de ces grandes quantités de données ;
- Les méthodes générant des reliefs artificiels. [21]

7.1.1. Génération des terrains

Les techniques de génération des terrains peuvent être classées par l'ordre chronologique de leurs apparitions en 3 catégories :

- Techniques à base de plaquage de texture : ces techniques mettent en évidence la technique de rendu et de texturage plutôt que la technique de génération, et elles sont utilisées jusqu'à aujourd'hui pour rendre des paysages générés à base de fractales ou de prises de mesures ;
- Méthodes d'érosion : utilisées pour améliorer le réalisme des paysages synthétisés, et utilisées dans des parties de relief où on veut faire apparaitre le phénomène d'érosion, leur utilisation peut être combinée avec l'utilisation des fractals ou les systèmes de particules ;
- Méthodes fractales : utilisent les fonctions fractales pour la génération des terrains [17].

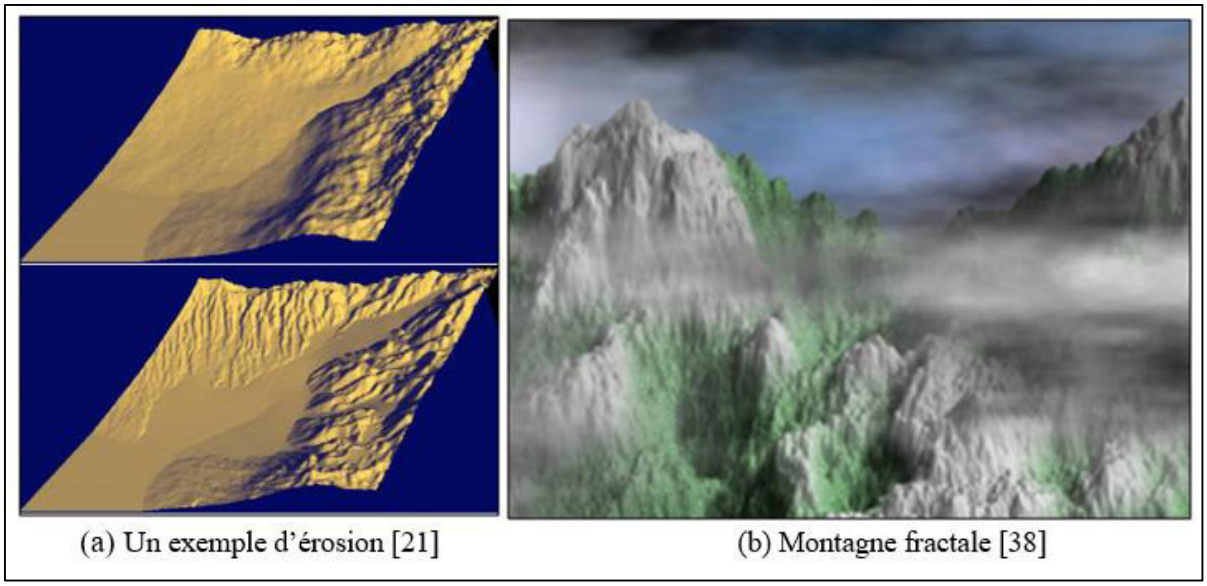


Figure 16 : Exemples des terrains générés [16].

7.1.2. Représentations

Deux structures de données sont essentiellement utilisées pour la représentation des terrains, les cartes d'élévation et les réseaux de polygones. [16]

7.1.2.1. Les cartes d'élévation :

La représentation la plus utilisée pour représenter un terrain est **la carte d'élévation** (height map). Avec cette représentation, un terrain est stocké sous forme d'un tableau à deux dimensions, où chaque case contient la hauteur du terrain. De plus on peut associer une couleur à chaque case, ce qui revient à associer une texture au terrain. La souplesse de cette représentation provient de la possibilité de considérer ces cartes comme des images 2D dont la manipulation est très intuitive : modélisation possible avec des outils de dessin 2D, facilité de génération de niveaux de détails, etc. [21]

7.1.2.2. Les réseaux de polygones :

Dans cette structure les hauteurs d'une carte d'élévation sont représentées par un maillage de triangles, les maillages de triangles sont plus compacts que les grilles de cartes d'élévation, et il existe beaucoup de moteurs de rendu spécialisés dans le rendu rapide des polygones. La difficulté de cette structure est la construction du maillage triangulaire (Triangulated Irregular Network ou TIN), il est important de trouver un compromis entre la précision du maillage (le niveau de détail sur le terrain) et le nombre de triangles. Garland propose une classification pour les techniques de construction de maillage :

- Les grilles uniformes : c'est un échantillonnage régulier des altitudes selon deux axes ;
- Les subdivisions hiérarchiques : application des quadtrees ou des k-d trees ;
- La recherche des singularités : les caractéristiques importantes du terrain (sommet, falaise, vallée ...) sont les nœuds du maillage ;
- Le raffinement du maillage : à partir d'une approximation minimale, de nouveaux points sont créés jusqu'au maillage final ;
- La simplification du maillage : à partir du maillage exhaustif de toutes les latitudes, le réseau est simplifié ;
- Les autres méthodes basées sur les techniques d'optimisations. [16]

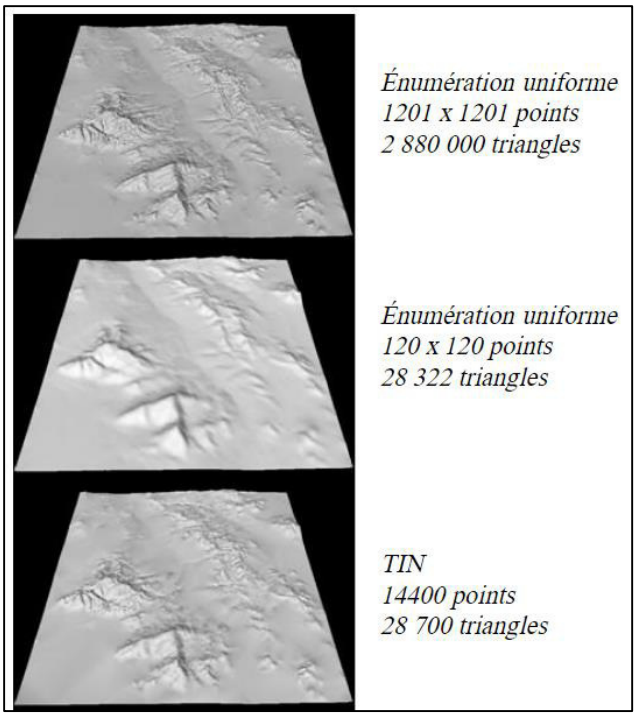


Figure 17 : Un exemple de rendu [16]

Conclusion

Nous avons mis en évidence, dans ce premier chapitre, les notions fondamentales de la lumière, l'ombrage et le domaine de la génération des terrains, en définissant ces notions et en donnant des définitions sur les termes les plus utilisés tels que l'ombrage et l'ombre.

Concernant la modélisation de scènes naturelles, il n'existe pas de technique unique et générale résolvant tous les problèmes.

Dans le prochain chapitre nous faisons une représentation dé taillée sur les cartes d'ombre ainsi que le rendu des terrains.

Chapitre 2:

Etat de l'art

Introduction

Les ombres jouent un rôle très important dans les applications graphiques modernes, car elles augmentent la valeur visuelle globale d'une image rendue. L'algorithme de Shadow Mapping et la technique basée sur les volumes d'ombre sont les techniques les plus populaires pour ajouter des ombres à des scènes 3D.

Dans ce chapitre, nous allons définir les techniques de rendu et les techniques utilisées pour effectuer la visibilité spécialement dans les terrains.

1. Rendu en temps réel

Le rendu en temps réel est une partie du domaine de l'infographie. Tandis que le rendu en synthèses d'images s'occupe à la suite de l'apparition des images produites, le rendu en temps réel se concentre principalement sur la création des images assez rapide pour maintenir l'observateur immergé dans la scène. Pour atteindre ce but, les images doivent être créées à un taux qui ne permet pas à l'observateur de distinguer les différentes images. [5]

2. Rendu

Nous allons décrire maintenant les techniques de rendu permettant de passer de l'espace objet 3D à l'espace image 2D, ainsi que les techniques connexes : calcul des ombres, problèmes de visibilité et optimisations. [21]

2.1. Techniques de rendu

On peut découper le travail de rendu en deux phases : déterminer les surfaces visibles à l'écran, puis calculer la couleur de l'élément de surface (ou la couleur moyenne des éléments) apparaissant en chaque pixel en tenant compte des caractéristiques d'orientation et de matière de celles-ci, ainsi que des conditions d'éclairage. Plusieurs techniques ont été développées dans ce but ; les deux les plus utilisées, à savoir le tampon de profondeur (Z-buffer) et le lancer de rayons (ray-tracing). Nous finirons cette partie par les techniques de rendu spécifiques au terrain.

2.1.1.Z-Buffer et ses extensions

L'algorithme du Z-Buffer est un algorithme d'élimination des parties cachées [20]. Le principe consiste à laisser à une extension du concept de mémoire d'image4 le soin de déterminer la visibilité des surfaces. Le Z-buffer est une mémoire, "un tampon des profondeurs" qui sert à stocker la profondeur de chaque pixel visible à l'écran. Durant le rendu, la profondeur d'un élément candidat à paraître dans le pixel (fragment) est comparée à la valeur de la profondeur déjà stockée pour ce pixel : si cette comparaison indique que le nouveau fragment est devant celui stocké, alors celui-ci remplace le contenu du pixel écrit dans la mémoire image et le Z-buffer est mis à jour. La complexité de cet algorithme est proportionnelle aux nombres de primitives traitées et à leur surface à l'écran. [21]

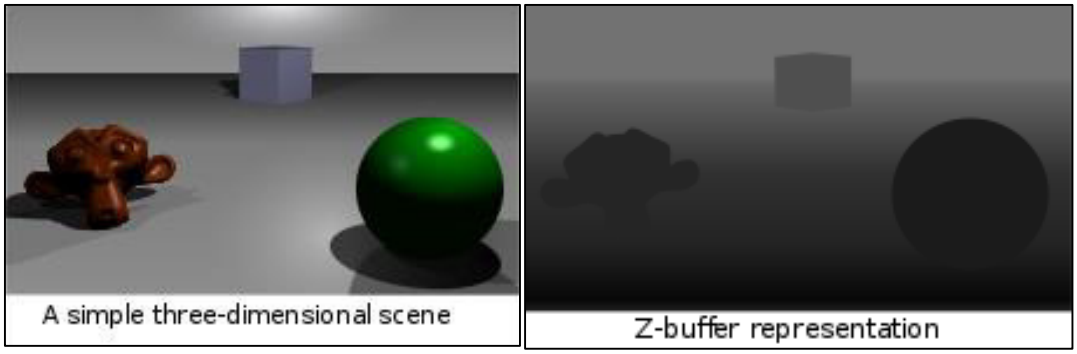


Figure 18 : Représentation d'une simple scène 3D avec Z-Buffer.

De nombreuses extensions à cet algorithme existent:

- Abuffer (anti-aliased), la plus intéressante, qui conserve pour chaque pixel une liste des fragments candidats à y apparaître et calcule pour chaque fragment la proportion du pixel qu'il recouvre. Ceci permet de traiter les problèmes d'aliassage et d'intégrer des objets semi-transparents [33] ;
- Une autre amélioration est le tampon d'accumulation (accumulation buffer) qui permet d'accumuler plusieurs images en déplaçant légèrement la caméra entre chaque rendu, soit pour réduire l'aliassage, soit pour obtenir un effet de flou de mouvement (motion blur) ou de mise au point (profondeur de champ) [34] ;
- L'autre solution pour réduire l'aliassage est de subdiviser les pixels. La couleur finale du pixel est obtenue en moyennant les sous-pixels. À noter que cette technique revient à effectuer le rendu à une résolution supérieure de celle de l'image (sur-échantillonnage). [21]

Cet algorithme ne traite pas les ombres, ni les réflexions et le traitement de la transparence nécessite un tri des polygones avant de les envoyer à la carte graphique. Un fragment de transparence α et de couleur C_f ayant réussi le test de profondeur sera composé de la manière suivante avec la couleur C_p déjà stockée dans le pixel :
$C_p = \alpha \times C_f + (1 - \alpha) \times C_p$, ce qui correspond bien à un calcul de transparence si tous les fragments se trouvant derrière celui-ci ont déjà été tracés (d'où le tri des fragments en prétraitement).

Un des avantages du Z-buffer est la possibilité de réaliser facilement des implémentations matérielles très efficaces. Aujourd'hui les moteurs graphiques à base de Z-buffer permettent le rendu et l'affichage de scènes de plusieurs centaines de milliers de polygones en temps réel [21]. Diverses techniques ont été développées pour obtenir des ombres à l'aide de cartes d'ombre, ou à l'aide de volumes d'ombre [35] [36] [37].

2.1.2.Lancer de rayons

Le principe du lancer de rayons (ray-tracing) est de calculer les objets visibles en remontant le chemin de la lumière parvenant à la caméra, i.e. en lançant des rayons à travers tous les pixels de l'image (fonctionnement inverse d'un appareil photo). Comme le principe du lancer de rayons est très simple à programmer, on peut lui faire simuler toute l'optique géométrique et ainsi prendre en compte de nombreuses caractéristiques optiques comme la réflexion, la réfraction, l'ombrage, etc. Notamment les reflets de la scène sur une surface lisse sont obtenus en envoyant un rayon secondaire depuis la surface dans la direction miroir à celle d'arrivée par rapport à la normale. Comme le Z-buffer, la deuxième partie du processus consiste alors à calculer la couleur que vont avoir les objets en faisant appel à un modèle d'illumination pondéré par l'éclairage (i.e. l'ombrage). Le principe de calcul des ombres est identique : pour savoir si un objet est éclairé, on lance un rayon d'ombre vers les sources de lumière.

Problème d'aliassage en lancer de rayons:

- Sur-échantillonnage : lancer plusieurs rayons par pixel, distribués aléatoirement, puis à moyenner les résultats pour donner la couleur finale. Le nombre de rayons lancés peut-être adaptatif, c'est-à-dire varié d'un pixel à l'autre en fonction du nombre d'objets présents dans le pixel : ce nombre n'étant pas connu à l'avance, on décide, après avoir lancé plusieurs rayons, de poursuivre tant que l'écart type des couleurs est supérieur à un certain seuil paramétrable. Cette méthode statistique donne un bon rapport efficacité/coût pour des scènes d'intérieur ou des scènes peu fouillées, c'est-à-dire lorsque le nombre de rayons lancés par pixel n'excède pas dix ou vingt. Cependant, dès que la complexité de la scène est trop importante son efficacité est limitée : soit on lance un nombre de rayons suffisant, mais les temps de calcul explosent, soit l'aliassage se fera rapidement sentir, surtout lors du calcul d'animation.

- Lancer de faisceaux : calculer l'intégrale de l'ensemble des objets se trouvant dans le pixel: c'est ce que se propose de faire le lancer de faisceaux [29] (beam-tracing) ou de cônes [28] (cone-tracing) en lançant un rayon pyramidal ou conique par pixel. À noter que le lancer de rayons avec sur-échantillonnage, que nous avons vu au paragraphe précédent, est une approximation numérique de cette intégrale [21].

L'avantage de cette technique de beam-tracing est de capturer tous les objets d'un pixel en lançant un unique rayon, l'inconvénient est qu'elle augmente la complexité du calcul pour chaque primitive. En effet, l'intersection entre une pyramide (ou un cône) et une primitive de la scène est plus compliquée à calculer que l'intersection entre une droite et cette primitive, comme c'est le cas avec le lancer de rayons classique. [21]

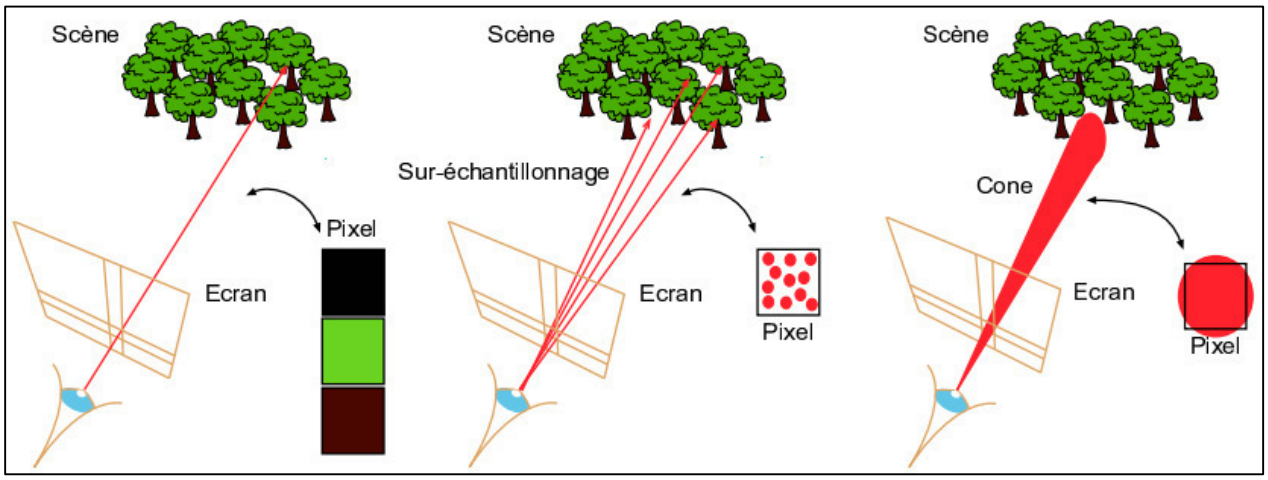


Figure 19 : De nombreuses primitives se projettent dans un pixel. À droite : en ne lançant qu'un rayon par pixel la couleur déterminée sera aléatoire (aliassage). Au milieu : la solution est d'échantillonner suffisamment le pixel en lançant de nombreux rayons. À droite : on lance un unique rayon conique pour considérer tous les objets qu'il contient. [21]

2.1.2.1. Optimisations

Les techniques d'optimisation du lancer de rayons sont nombreuses. Le principe général est la subdivision de l'espace objet avec pour but la limitation du nombre d'intersections rayon-objet à calculer [22]. Exemples des techniques: les volumes englobants et Division spatiale.

a) Volumes englobants

L'idée est d'utiliser des objets qui permettent des calculs d'intersection simples pour englober des objets dont les calculs d'intersection sont complexes. Si un rayon intercepte un volume englobant, alors on calculera précisément son intersection avec l'objet contenu dans le volume. Sinon, on regarde un autre volume englobant, etc. L'efficacité d la technique dépend de l'adéquation du volume englobant avec l'objet [39].

La complexité de l'algorithme ne change pas : elle est linéaire à chaque niveau de l'arbre. L'idée est d'utiliser des volumes englobants hiérarchisés: le principe est d'englober récursivement un groupe de volumes englobants par un volume englobant plus gros. La complexité théorique ainsi obtenue est logarithmique. Les données sont alors organisées dans un arbre où chaque nœud est un volume pointant sur les objets qu'il contient.

Figure 20 : Un modèle 3D avec sa boîte englobante dessinée en pointillés.

b) Division spatiale

Les volumes englobants prennent en compte les objets dans la scène. La division spatiale pour le lancer de rayons est une adaptation de la division spatiale classique (décomposition en voxels et représentation par octree) On cherche à tenir compte de la densité d'objets dans la scène. Chaque voxel pointe sur une liste d'objets dont les surfaces intérsectent le volume. Les objets d'un voxel donné sont testés par rapport aux 6 faces du voxel. S'il y a intersection, l'objet est ajouté à la liste d'intersections.

2.1.3.Rendu volumique par lancer de rayons

Le rendu volumique est destiné à calculer une image 2D d'une structure de données représentée par une grille 3D (chaque case étant appelée un voxel). Un exemple typique de données est une grille où chaque voxel contient une densité (*e.g.* une densité de tissus humains). La technique classique de rendu est basée sur le principe du lancer de rayons que nous avons vu précédemment. Le rayon parcourt le volume de données en utilisant un algorithme de tracer de droite discrète et accumule au fur et à mesure la transparence, ainsi que la couleur si celle-ci est une donnée de la grille. Pour obtenir la quantité de lumière que reçoit chaque voxel il faut lancer un rayon vers chaque source de lumière (comme dans le cas du lancer de rayons classique).

2.1.4.Cas des terrains

Nous avons vu qu'une représentation commode d'un terrain est la carte d'élévation (*height map*), mais de nombreux moteurs de rendu préfèrent traiter des polygones, il faut alors convertir la carte en géométrie au moment du rendu.

Il existe des techniques dites d'*inverse displacement* , sorte de lancer de rayons adapté, gérant directement l'intersection entre le rayon et la représentation, sans expliciter la carte d'élévation en facettes. Cela permet en outre de le faire efficacement, en procédant de manière hiérarchique : le principe consiste à organiser les données en quadtree et à pré-calculer dans chaque case l'altitude maximale. On a alors la garantie que le rayon ne coupe pas cette portion de terrain si ses points d'entrée et de sortie dans la case sont plus hauts que ce maximum, sinon, on repose le problème sur les quatre cases filles, et ainsi de suite. Quand on atteint la résolution

des données, il faut tester l'intersection du rayon et de la surface dans la case en la calculant vraiment, ce qui n'arrive que pour peu de cases [40].

Une autre technique, non spécifique au lancer de rayons, basée sur un système de cache, permet de ne pas expliciter toute la géométrie : les polygones sont générés au vol, on ne convertit que les données utiles dans la zone en cours de traitement. On utilise un système de cache pour éviter de générer des polygones à plusieurs reprises. [21]

2.2. Visibilité

La visibilité tient une part importante dans l'algorithmique de la synthèse d'images. Ses applications sont, entre autres, la suppression des parties cachées pour la détermination de l'ombrage (optimisation).

2.2.1.Généralité

La suppression des objets cachés est une des grandes motivations des algorithmes de visibilité. Les techniques les plus simples sont la suppression des objets se trouvant en dehors du champ de la caméra (*frustum culling*) qui permet de supprimer jusqu'à 75%, voir 80% des primitives, et la suppression des parties d'un objet se trouvant à l'arrière (*backface culling*). Les techniques plus complexes capables de supprimer les objets cachés par d'autres (*occlusion culling*) sont souvent basées sur un pré-calcul de structures de visibilité (cartes), et permettent de supprimer jusqu'à 95% des objets avec un coût consacré aux tests bien moindre (ceci dépend bien sûr de la configuration de la scène et de la position de l'observateur). Airey et Teller furent les premiers à proposer de pré-calculer la visibilité dans des environnements architecturaux intérieurs. Ils exploitent le fait que si l'on est dans une pièce, on ne voit les autres que par les portes ou fenêtres, i.e. en construisant une structure de vues (portails) qui prend en considération les zones visibles. Ces techniques pour ce type de scènes sont très efficaces, mais grandes consommatrices de mémoire. Des algorithmes capables de traiter des scènes d'extérieur ont fait leur apparition ces dernières années, où la notion d'occludeurs est utilisée pour éliminer les zones cachées à la vue. Ces algorithmes sont basés sur la notion de cellules de visibilité : l'espace est découpé en outrée et, pour chaque cellule (voxel), on pré-calcule les parties visibles de la scène. Ces techniques fonctionnent correctement lorsque les occludeurs sont de grandes tailles, ce qui n'est pas le cas des arbres où les occludeurs (i.e. feuilles) sont petits et nombreux.

Durand et Schaufler introduisent la notion de fusion de plans d'occlusion, qui permet de grouper les zones contiguës cachées par chaque occludeur (feuille) en des zones plus grandes. [21]

2.2.2.Cartes d'ombres

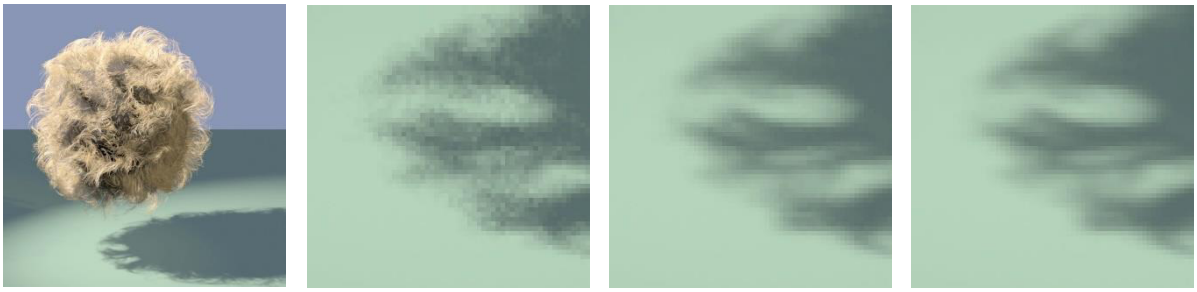
Nous venons de passer en revue les techniques de pré-calcul de la visibilité dans le but de ne traiter que les objets visibles. Le calcul des ombres est un problème assez similaire, puisqu'il revient à déterminer les objets visibles depuis les sources de lumière, avec pour avantage que celles-ci sont souvent fixes dans le temps.

La technique des cartes d'ombres (*Shadow Map*), introduite par Reeves en 1987, pré-calcule la visibilité depuis la source de lumière dans une carte, en effectuant un rendu de la scène depuis cette source et en conservant le tampon de profondeur. Au moment du rendu d'un objet on calcule la distance entre l'élément vu dans le pixel et la source de lumière. Si cette distance est supérieure à celle stockée dans la shadow map alors l'objet est à l'ombre, sinon il est à la lumière.

Lorsque la source de lumière change de position ou lorsqu'un objet bouge, il faut recalculer la carte, ce qui oblige à effectuer deux rendus : un depuis la source de lumière, et un depuis l'œil [30].

Une technique similaire à celle-ci est l'*alpha shadow map* qui n'utilise que des cartes de transparence, et s'applique quand il y a une séparation entre les objets projetant leur ombre (les occludeurs) et les objets ombrés. Lors du calcul de la carte, seuls les objets occludeurs sont rendus. Le tampon est ensuite utilisé comme texture multiplicative sur les objets ombrés (i.e. la couleur est pondérée par l'opacité stockée dans la carte). Contrairement à la technique des *shadow maps*, les *alpha shadow maps* autorisent des occludeurs semi-opaques, ne demandent pas de fonctionnalité spécifique du matériel graphique et ne posent pas le problème de la dynamique des profondeurs. Par contre, elles demandent de bien séparer (et donc de connaître) les occludeurs des objets ombrés (ce qui est par exemple le cas d'arbres projetant leur ombre sur un terrain). [21]

Des techniques ont été développées pour combiner les aspects positifs des shadow maps et des alpha shadow maps. C'est-à-dire être capable de gérer des objets transparents ou une densité d'objets très importante par rapport à la résolution de la texture, sans avoir à séparer les occludeurs des objets ombrés. Lokovic calcule l'ombrage et l'auto-ombrage d'une chevelure (figure 21), dont les primitives sont à une résolution trop fine pour la méthode de base. Il découpe l'espace en intervalles de profondeur auxquels il associe une carte. Il appelle l'ensemble cartes d'ombres en profondeur (*deep shadow map*). Son implémentation est complètement logicielle et n'est donc plus temps réel [31]. Tae-Yong en accélère le calcul en utilisant le matériel graphique, sans pour autant parvenir à atteindre le temps réel. Soler propose une technique pour calculer un alpha shadow map contenant des ombres douces. Il convolue à l'aide du matériel graphique des images des occludeurs prises depuis différentes positions de la source de lumière. [32]



(a) Balle avec 50 000 chevelures *(b) 512512 Normal shadow map* *(c) 4k×4k Normal shadow map* *(d) 512×512 Deep shadow map*

Figure 21 : Démonstration du bruit à partir des diverses méthodes d'ombre. [23]

Figure 22 : Une théière poilue [32]

2.2.2.1. Problèmes de shadow map

Le problème majeur de l'algorithme Shadow map est la possibilité qu'un aliassage très visible apparaisse au niveau des contours des ombres [1].
Plusieurs techniques ont donc été proposées pour atténuer ce problème:

- **Percentage Closer Filtering (PCF):** proposé par Reeves et al. Cet algorithme de filtrage effectue le test du z sur plusieurs pixels voisins du shadow map, de façon à obtenir des tons de gris aux frontières de l'ombre. Le PCF diminue de beaucoup l'aliassage, mais pour des tailles de filtre assez coûteuses. Le matériel graphique moderne implante une version simplifiée de cet algorithme, sous la forme de filtrage bilinéaire des textures de profondeur.
- **Light-space perspective shadow maps:** Les LiSPSM ajustent la matrice de projection de la lumière afin d'obtenir une plus grande précision pour les objets proches de la caméra. C'est très important dans les cas de « duelling frustra » : tu regardes dans une direction, mais la lumière spot « semble » dans la direction opposée. Vous avez une grande précision pour la carte d'ombres près de la lumière, soit loin de toi et une faible résolution proche de la caméra, là où tu en as le plus besoin. Par contre, les LiSPSM sont délicates à implémenter.
- **Shadow map en cacade:** Les CSM gèrent le même problème que les LiSPSM mais d'une manière différente. Elles utilisent simplement plusieurs (2-4) cartes d'ombres standards pour les différentes parties de la zone vue. La première gère les premiers mètres, donc tu vas obtenir une bonne résolution pour une petite zone. La prochaine carte d'ombres gère les objets plus loin. La dernière carte d'ombres gère la grosse partie de la scène, mais à cause de la perspective, elle ne sera pas aussi importante visuellement que la zone la plus proche.

2.2.3.Cas des terrains

Nous présentons ici des méthodes traitant plus spécifiquement de visibilité de scènes de terrains dans le but d'en calculer l'auto-ombrage pour bien montrer le relief. Ces techniques sont également utilisées pour limiter le nombre de polygones envoyés au moteur de rendu.

- Max introduit la notion de carte d'horizon (*horizon map*) pour ajouter de l'ombrage à une surface dont les normales sont représentées par une carte de perturbations (*bumpmap*). L'idée est de calculer la visibilité du ciel à chacun des points échantillonnés de la surface. Max considère huit directions autour de chacun de ces points et détermine l'horizon en utilisant les points échantillonnés (figure 23 à gauche). Durant le rendu, un point est considéré comme illuminé si le soleil est visible de ce point : pour cela, il utilise la direction échantillonnée la plus proche. La pénombre est calculée à partir de la quantité du disque solaire vu depuis le point. La limitation de cette méthode est le sous échantillonnage des directions de visibilité qui entraîne des artefacts. [24]
- Cabral utilise des cartes d'horizon similaires pour calculer la *Bidirectional Reflectance Distribution Function BRDF* d'une carte de perturbation des normales (*bump-map*). Il utilise 24 directions de vue et projettent les triangles du terrain à la place des points lors du calcul de la visibilité, ce qui diminue le sous-échantillonnage mais ne le fait pas disparaître entièrement. [25]
- Cohen-Or utilise une technique d'échantillonnage similaire pour découper le terrain en partie visible et partie invisible dans le but de limiter le nombre de primitives envoyées au moteur de rendu [26]. Stewart augmente la précision de la méthode en considérant tous les points de la surface pour le calcul de la visibilité (figure 23 à droite), alors que Max ne considérait que huit directions. En utilisant une hiérarchie il introduit un algorithme rapide lui permettant de calculer les cartes d'horizon rapidement, même pour un très grand nombre d'échantillons [27].

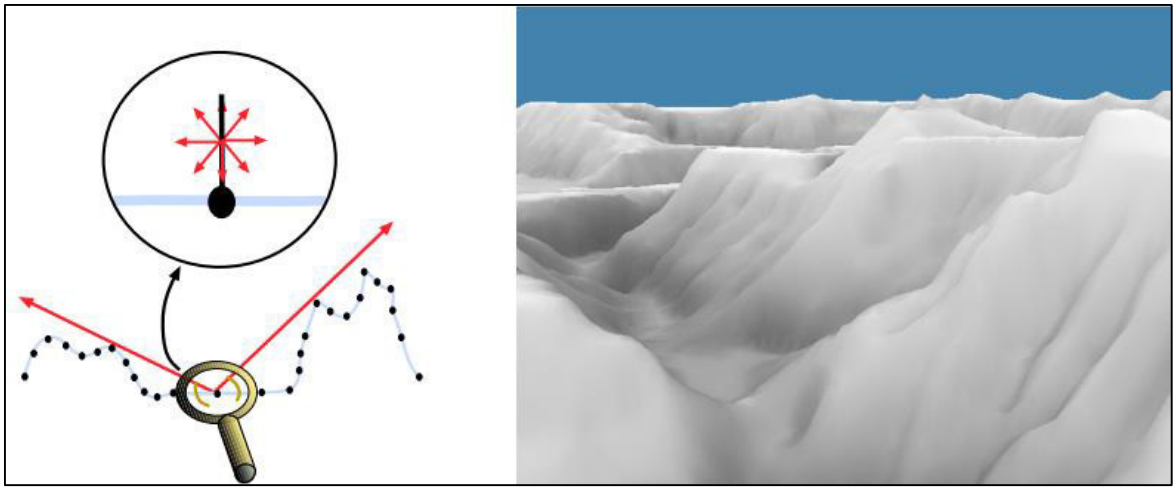


Figure 23 : Cartes d'horizon. À gauche : pour chaque point échantillonné de la surface, on calcule dans 8 directions la hauteur de l'horizon [24]. À droite : Stewart [27] augmente la précision des cartes d'horizon et les utilise pour calculer l'ombrage et accélérer le rendu en n'envoyant à la carte graphique que les parties visibles du terrain.

3. Bilan

Les techniques de modélisation de terrains et d'arbres donnent des résultats remarquables en terme de réalisme, même s'il reste encore des travaux à effectuer, notamment pour les vues rapprochées (écorces, terre, poussière, sable, etc.). Cependant, les modèles générés sont extrêmement lourds de polygones. Les techniques de rendu sont bien incapables de traiter convenablement (*i.e.* sans coût de calcul prohibitif et sans aliassage) les milliards de primitives qu'engendrent les arbres.

Les algorithmes de simplification de maillage sont capables de générer efficacement des niveaux de détails pour les terrains, mais leur application aux arbres n'est pas convaincante, notamment à cause de la nature fouillée et dispersée des feuilles. Même si les techniques de visibilité diminuent le nombre de polygones envoyés au moteur de rendu et pré-calculent l'information pour l'affichage des ombres, il semble difficile de traiter rapidement des paysages composés d'à peine quelques dizaines d'arbres. Il est évident, au regard de ces techniques, qu'il y a avant tout un problème de représentation, au-delà de la simple hiérarchisation qui est déjà poussée à bout dans toutes les techniques actuelles.

Conclusion

Dans ce chapitre, on a essayé de mettre en évidence les techniques de rendu et les techniques utilisées pour effectuer la visibilité spécialement dans les terrains. On a commencé par les techniques de rendu utilisées dans les objets 3D et les terrains, après on a expliqué la notion de la visibilité et ces techniques et on a terminé par un bilan des tous les techniques.

A l'issus de ce chapitre nous avons peut conclure notre objectif principal et d'avoir une idée précise sur les techniques de génération des ombres en temps réel et les techniques de rendu des terrains.

Chapitre 3:

Conception et mise en œuvre

Introduction et objectifs

Dans ce chapitre, nous allons présenter l'essentiel de notre travail, pour cela nous allons commencer par définir les objectifs à atteindre, puis nous exposons les étapes de conception et de réalisation de notre projet.

La phase de conception est la phase la plus importante pour pouvoir développer une bonne application, elle a pour objectif de définir les différents composants de l'application qui coopèrent ensembles à la réalisation des fonctionnalités souhaitées.

La phase de mise en œuvre a pour objectif de décrire les étapes d'implémentation de notre application, elle va vous permettre de détailler les différentes structures de données utilisées et les algorithmes nécessaires pour la réalisation de notre application.

Notre projet a pour but de faire un rendu d'un terrain et le calcul rapide de shadow map pour ce terrain. Les principaux objectifs auxquels doit répondre notre application sont:

- Rendu du terrain;
- Le rendu des cartes d'ombre;
- L'affichage final.

I. La phase de conception

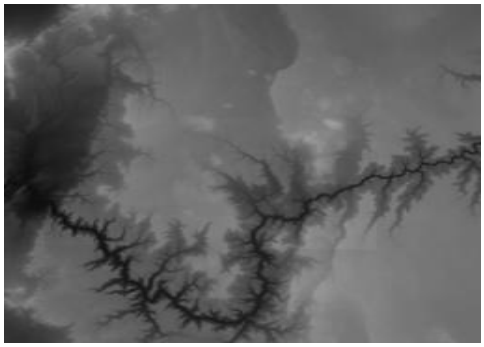
1. Rendu des terrains et Shadow map

1.1. Rendu des terrains

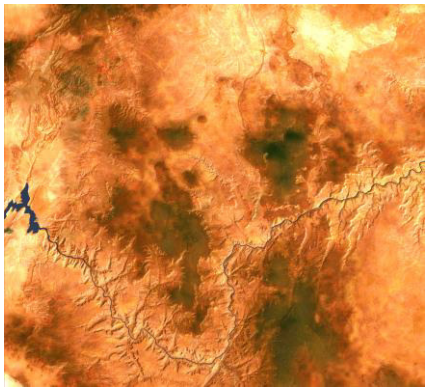
1.1.1. Construire le terrain

Dans notre implémentation nous avons utilisé un terrain construit à partir de deux images: la première est une carte des hauteurs (heightsmap), sous forme d'une image à niveaux de gris (Figure 24 (a)).

Les normales sont obtenues par le produit vectoriel entre les vecteurs générés par les deux voisins sur l'axe $\overrightarrow{ox}$ et $\overrightarrow{oz}$ à partir du sommet au quel on veut calculer la normale. Pour colorier les différentes zones du terrain on utilise la deuxième image colorée (Figure 24 (b)).



(a)- Une image d'hauteurs « heightsmap »



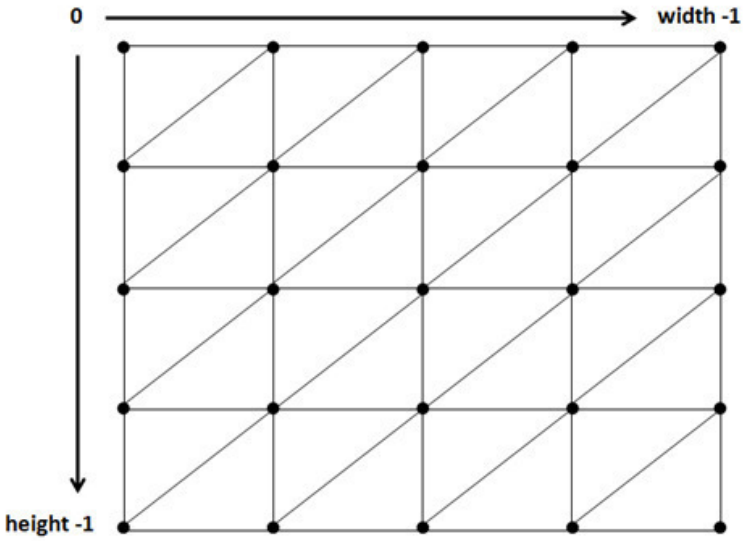
(b)- Une texture du terrain.

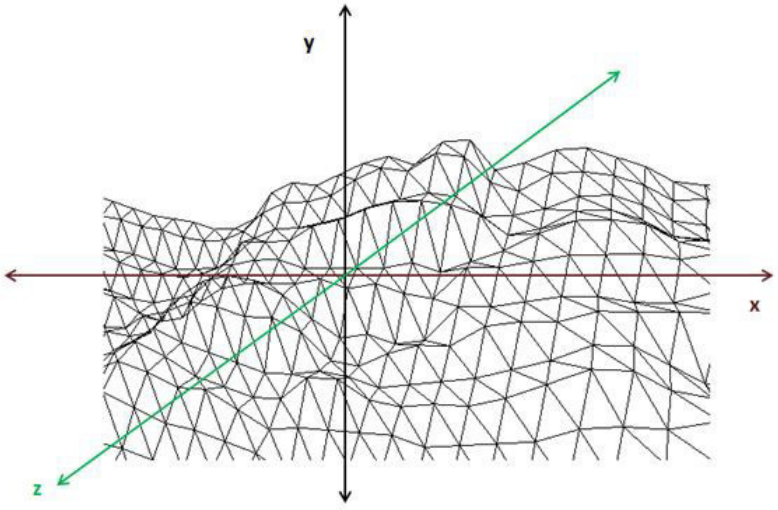
Figure 24: Les images utilisées dans l'application.

1.1.2.Rendu

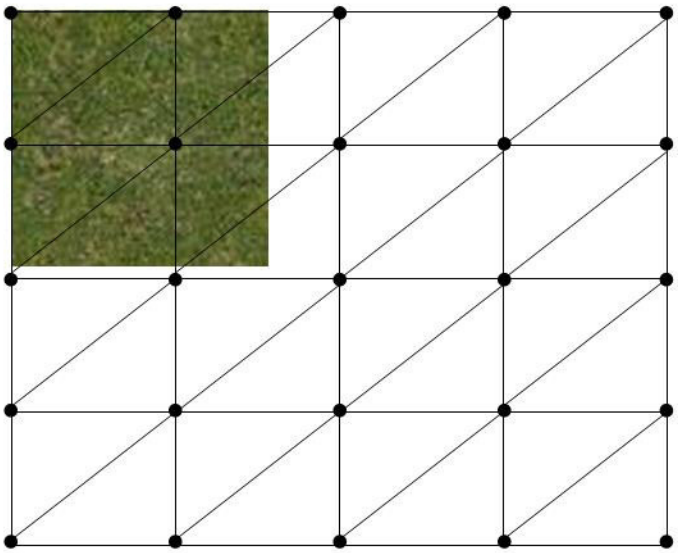
Pour optimiser le rendu du programme, on mettra tous les points dans un tableau à 1 dimension qu'on parcourra lors de l'affichage:

Le terrain est défini par un ensemble de points qui forment un maillage en forme de cases carrées. Pour le dessiner, on peut dessiner directement les cases avec des GL_TRIANGLE_STRIP pour obtenir un rendu optimisé. Voici comment dessiner les triangles de chaque case sans les textures :





Concernant la texture:



1.2. Shadow Map

Pour produire les ombres sur le terrain, nous avons utilisé une carte d'ombre « Shadow map », calculé par GPU, en positionnant la caméra dans la position de la source lumineuse, et en faisant le rendu dans un Frame Buffer Object "FBO", qu'on attache au résultat du rendu. Nous spécifions le type du stockage par « GL_DEPTH_COMPONENT24 », que nous l'attachons au buffer de profondeur par la fonction « glFramebufferRenderbuffer », puis nous définissons la texture sur laquelle le résultat va être sauvegardé, par la fonction « glFramebufferTexture », enfin on désactive le rendu pour les buffers autre que le depth buffer par un appel « glDrawBuffer(GL_NONE) », et on fait un premier rendu de la scène avec l'illumination désactivée.

1.2.1.Problème d'aliasing

Comme on a vu dans le chapitre précédent, l'un des problèmes de shadow map est l'effet de l'aliasing. Et pour cela, on a proposé la méthode de « Shadow map en cascade » pour voir un résultat plus efficace. Donc au lieu d'avoir 2 passes, comme dans le cas de shadow map classique, on va voir 3 passes du rendu.

1.2.2.Première passe du rendu

Premièrement, la scène est rendue du point de vue de la source de lumière. Les valeurs contenues dans le z-buffer sont alors sauvegardées, elles forment le tampon de profondeur des ombres comme illustré dans la figure 25.

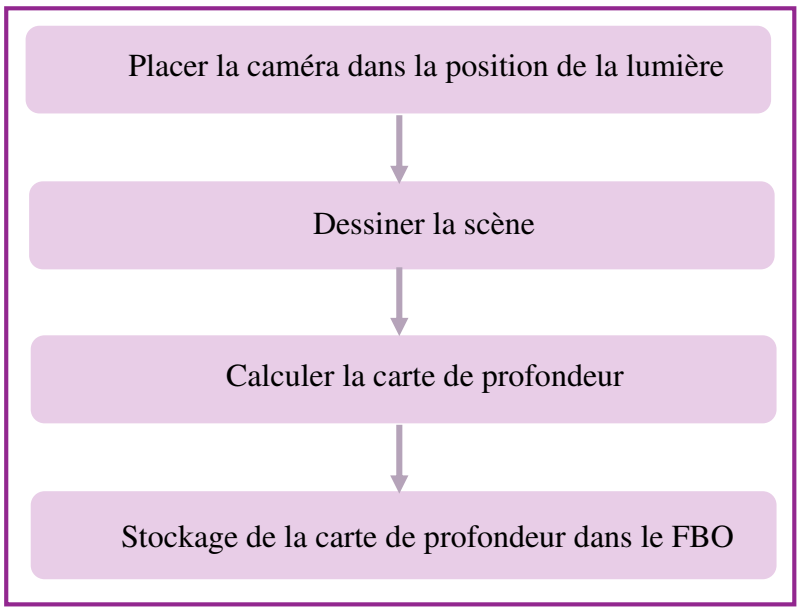


Figure 25: Première passe du rendu.

1.2.3.La deuxième et la troisième passe du rendu

Après avoir fait le rendu de la source lumineuse, on va faire le rendu du point de vue de la camera: Attacher les depth maps. Le dernier rendu sera le calcul de l'ombrage et l'affichage final en utilisant la matrice Bias (Figure 26).

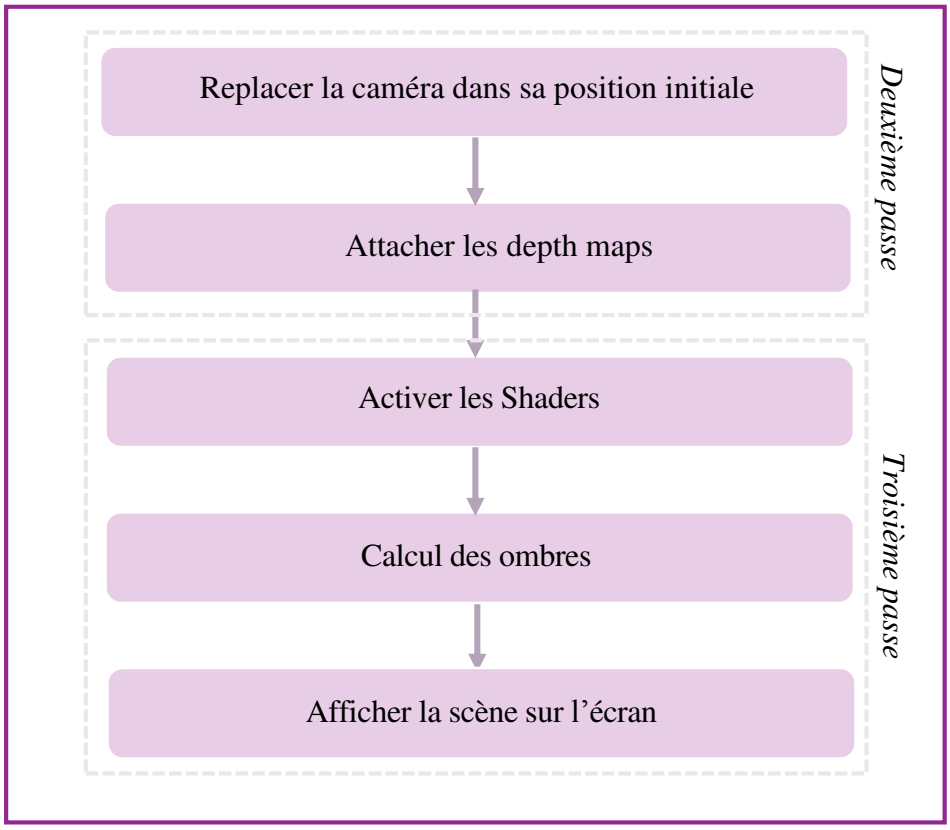


Figure 26: Deuxième et troisième passe du rendu.

2. Algorithme de rendu de l'application

Le schéma ci-dessus représente la conception générale de notre application, qui consiste à faire le rendu des terrains et Shadow map en temps réel.

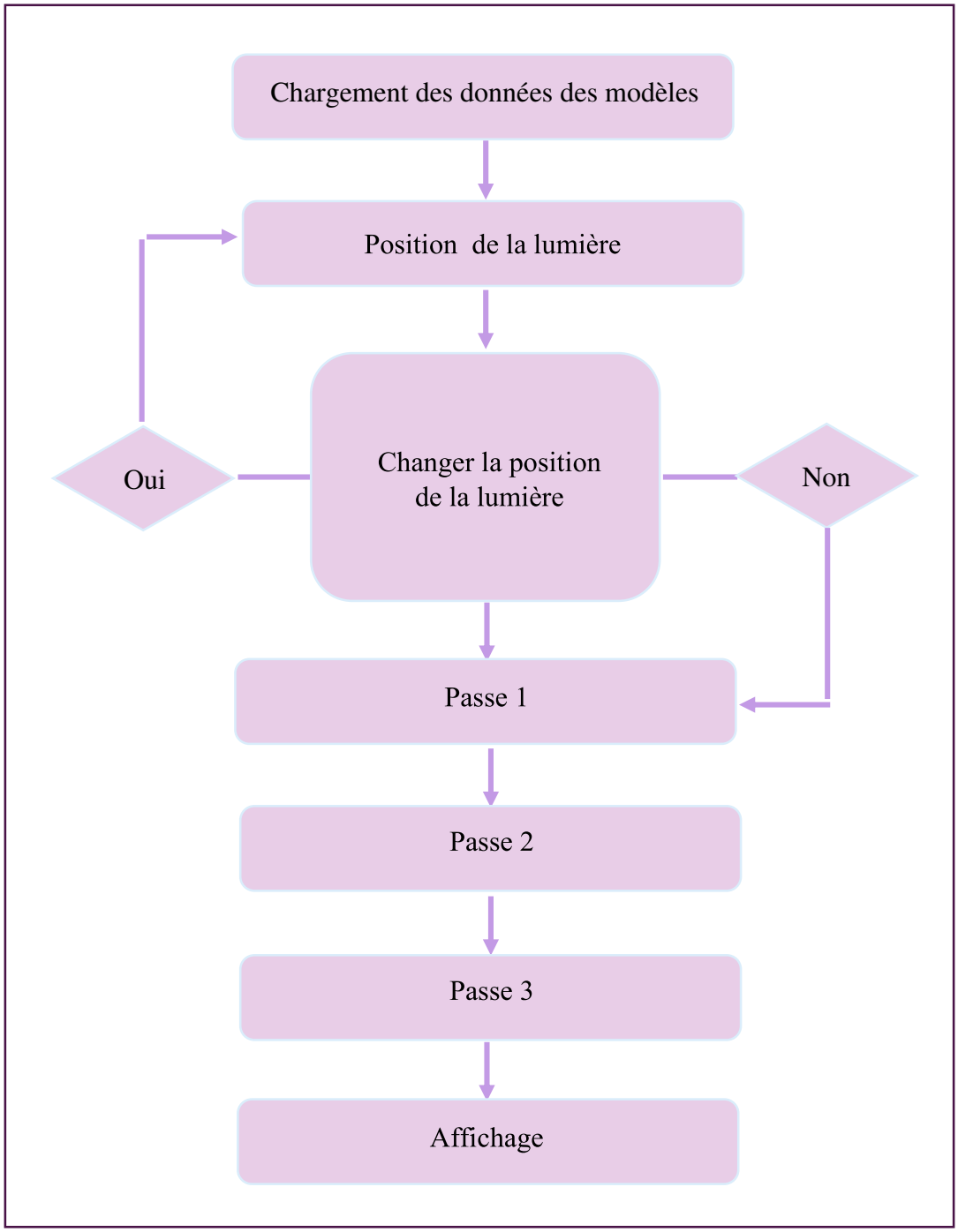


Figure 27: Schéma général.

II. La phase de mise en œuvre

La mise en œuvre de notre application nécessite tout d'abord la spécification des outils et plateforme adéquats, ce que nous allons présenter en premier lieu. Dans un deuxième lieu, nous donnons une description des structures de données et des méthodes nous avons travaillé.

1. Environnement et langages de programmation utilisés

Dans notre application, nous avons utilisé Visual C++ 2010, est nous avons programmé notre application en utilise les deux langages, C++ et GLSL, et quelques bibliothèques:

1.1. Environnement

1.1.1. Visuel Studio

Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications web ASP.NET, des services web XML, des applications bureautiques et des applications mobiles. Visual Basic, Visual C++, Visual C# utilisent tous le même environnement de développement intégré (IDE), qui leur permet de partager des outils et facilite la création de solutions faisant appel à plusieurs langages. Par ailleurs, ces langages permettent de mieux tirer parti des fonctionnalités du Framework .NET, qui fournit un accès à des technologies clés simplifiant le développement d'applications web ASP et de services web XML grâce à Visual Web Developer.

1.2. Langages de programmation et bibliothèques

1.2.1. C++

C++ est un langage de programmation d'usage universel. C'est un langage statique soutenant la programmation procédurale, l'abstraction de données, la programmation orientée objet, et la programmation générique. Notre choix de Visual C++ est motivé par les faits suivants:

- ✓ Le langage de programmation C++ est un langage très puissant.
- ✓ Il offre la possibilité de programmer avec les classes et les objets (orienté objet).
- ✓ L'environnement Visual C++ donne des bons résultats au sens qualitatif et Facile d'intégrer les bibliothèques OpenGL dans l'environnement Visual C++.

1.2.2.OpenGL « Open Graphics Library »

Une interface de programmation (API) graphique pour le rendu 3D projectif. Initialement conçue par Silicon Graphics Inc (IRIS-GL 1990).

- Est un système graphique qui permet de visualiser une scène 3D (et aussi 2D), capable de communiquer avec le matériel graphique. Il s'agit d'une bibliothèque d'outils de traitement graphique indépendante du matériel où l'on peut créer des programmes interactifs produisant des images réalistes ou abstraites en couleurs d'objets 3D.Le Principe de programmation OpenGL et élimination des parties cachées.

1.2.3.Shader

Un Shader est un programme informatique, utilisé en image de synthèse, pour paramétrer une partie du processus de rendu réalisé par une carte graphique ou un moteur de rendu logiciel. Ils peuvent permettre de décrire l'absorption et la diffusion de la lumière, la texture à utiliser, les réflexions et réfractions, l'ombrage, le déplacement de primitives. Il existe également des langages de haut niveau spécifiques comme HLSL, Cg, GLSL. Dans notre travail nous allons utiliser le langage GLSL car il permet la programmation GPU avec OpenGL.
Il en existe trois types: Fragment Shader, Geometry Shader et Vertex Shader

1.2.3.1. Fragment Shader (Pixel Shader)

C'est l'étape qui va définir la couleur de chaque pixel de la forme délimitée par les vertices.

1.2.3.2. Geometry Shader

Opération sur les primitives: ajout / retrait de sommets et primitives.

1.2.3.3. Vertex Shader

Opération sur les sommets. Cette étape prend un vertex à part pour travailler dessus. S'il y a 3 vertices (pour un triangle) alors le vertex shader sera exécuté 3 fois.

1.2.4.GLSL « OpenGL Shading Language »

Est un langage permettant la programmation GPU de scènes OpenGL. Le langage glsl est basé sur le langage C, mais comporte plusieurs restrictions :

- La taille du code est relativement limitée.
- Le nombre d'itérations dans les boucle for limité.

Pour utiliser GLSL, il vous faut une carte graphique ATI ou nVidia, de préférence récente…

1.2.4.1. Les variables de communication en GLSL

Les variables Uniform : Variable globale constante partagée par tous les instances d'un shader en cours d'exécution. La valeur d'une variable uniform est déterminée par le programme client. Communes au Vertex et Fragment Shader

Les variables Varying : Variable globale dont chaque shader possède sa propre copie. Ces variables servent à faire passer des valeurs du VS au FS par interpolation. Servent à passer des valeurs (couleurs, vecteurs, position, etc.) du VS au PS. Le VS retourne des valeurs par sommets et le FS reçoit des valeurs par pixel.

1.2.5.Les Frame buffer objets (FBO)

Partie de la mémoire vidéo allouée pour contenir l'information d'une image. Il s'agit d'un **écran caché** qui utilise les **3** types de buffers, les FBOs sont utilisé pour éviter le transfert GPU/CPU. Les FBOs ont l'avantage de permettre de conserver et d'utiliser les informations calculées par le GPU dans la mémoire du GPU (VRAM). En effet, avec les anciennes méthodes, il fallait faire des allers et retours entre le programme OpenGL et le GPU. Maintenant, les données restent sur le GPU, les programmes ne subissent plus des latences à cause du bus de données.

2. Des structures de données et méthodes utilisées

Pour implémenter les différents modules de notre application, nous avons utilisé les structures et les méthodes de données suivantes :

2.1. Les structures

- Concernant la position du camera et la position de la lumière

```
float cam_pos[3] = {-8.f, 32.f, -2.f};
float cam_view[3] = {0.84f, 0.03f, 0.54f};
GLfloat light_dir[4] = {-0.05f, 0.85f, 0.25f, 0.0f};
```

- Concernant le scale height

```
#define SCALE    0.2f
```

- Concernant les valeurs de hauteurs et de normales

```
float *heights;
 float *normals;
```

- Concernant la matrice bias

```
const float bias[16] = { 0.5f, 0.0f, 0.0f, 0.0f,
                         0.0f, 0.5f, 0.0f, 0.0f,
                         0.0f, 0.0f, 0.5f, 0.0f,
                         0.5f, 0.5f, 0.5f, 1.0f     };
```

- Concernant shadow map FBO

```
GLuint shadowMapFb;
```

2.2. Les méthodes utilisées

Les méthodes suivantes sont des principales méthodes:

1) **Charger la scène:** `void Load ();`

   On va charger les textures de base et les objets (arbres,..).

2) **Dessiner le terrain:** `void MakeTerrain();`

   On dessine le terrain, comme on a déjà dit, à l'aide de `GL_TRIANGLE_STRIP`.

3) **Initialiser l'environnement:** `void initEnvironment(int terrainDim);`

   A l'aide de cette méthode, on va lire des shaders et générer FBOs pour le shadow map.

```
void initEnvironment(int terrainDim)
{
	…
	singleShader = createShaders( "shadow_vertex.glsl",
"shadow_single_fragment.glsl");

	….
	//FBOs for the shadow map
	glGenFramebuffersEXT(1, &shadowMapFb);
	glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, shadowMapFb);
	glDrawBuffer(GL_NONE);
	glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

	glGenTextures(1, &shadowMap);
	glBindTexture(GL_TEXTURE_2D_ARRAY_EXT, shadowMap);
	glTexImage3D(GL_TEXTURE_2D_ARRAY_EXT, 0, GL_DEPTH_COMPONENT24, mapSize, mapSize,
hfdhdsfj……4ksdf..jdMAX_SPLITS, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
	glTexParameteri(GL_TEXTURE_2D_ARRAY_EXT, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D_ARRAY_EXT, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D_ARRAY_EXT, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
	glTexParameteri(GL_TEXTURE_2D_ARRAY_EXT, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D_ARRAY_EXT, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
}
```

4) **La 1ère passe du rendu:** `void updateShadowMap();`

```
void updateShadowMap()
{
	float shad_modelview[16];

	//1st pass : From light's view
	glDisable(GL_TEXTURE_2D);
	glMatrixMode(GL_PROJECTION);
	glPushMatrix();

	glMatrixMode(GL_MODELVIEW);
	glPushMatrix();
	glLoadIdentity();
```

```
        gluLookAt(0, 0, 0, -light_dir[0], -light_dir[1], -light_dir[2], -1.0f, 0.0f,
0.0f);
        glGetFloatv(GL_MODELVIEW_MATRIX, shad_modelview);

        //Store the screen viewport and render only to the shadow map
        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, shadowMapFb);
        glPushAttrib(GL_VIEWPORT_BIT);
        glViewport(0, 0, mapSize, mapSize);
…
```

5) La 2ème et la 3ème passe du rendu `void renderShadowMap();`

```
        //2nd pass : From camera's point of view
        cameraInverse(cam_inverse_modelview, cam_modelview);

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(FOV, WINDOW_RATIO, frusts[0].neard, frusts[mapSplit-1].fard);
        glGetFloatv(GL_PROJECTION_MATRIX, cam_proj);
        …
        //3rd pass : Draw with bright light, using Bias matrix
        for(int i = 0; i < mapSplit; i++)
        {
                float light_m[16];
                far_bound[i] = 0.5f * (-frusts[i].fard * cam_proj[10] + cam_proj[14]) /
frusts[i].fard + 0.5f;

                glActiveTexture(GL_TEXTURE0 + (GLenum)i);
                glMatrixMode(GL_TEXTURE);
                glLoadMatrixf(bias);
                glMultMatrixf(shadowCpm[i]);
        …
```

6) `void updateShadowMap(),inline GLuint CompileGLSLShader( GLenum target, const char* shader)`: Fonctions de chargement et de compilation du fichier shader.

7) `inline GLuint LinkGLSLProgram( GLuint vertexShader, GLuint fragmentShader)` : Créer un programme composé de Vertex et Fragment shader.

8) `void onMouseMotion(int x, int y)`: le principe de cette fonction et de changé la postion de la source de lumière lor de la mouvement de la sourie avec le clé "Shift" .

9) `void onSpecialKeyDown(int key, int x, int y)`: le principe de cette fonction et de changer la postion du camera.

10) Génère le rendu final du terrain:

```
void onRender()
{
	//Update
	fps->Update();
	updateShadowMap();
	cameraLookAt();

	//Draw the scene
	renderShadowMap();
	terrain->Draw();

	//Draw text
#ifdef ENABLE_TRACE
	sprintf(cameraString, "Camera at (%.2f, %.2f, %.2f) Look (%.2f, %.2f, %.2f)",
		cam_pos[0], cam_pos[1], cam_pos[2], cam_view[0], cam_view[1],
cam_view[2]);
	sprintf(lightString, "Light Source at (%.2f, %.2f, %.2f)",
		light_dir[0], light_dir[1], light_dir[2]);
	sprintf(fpsString, "Current FPS : %.2f", fps->GetFps());
#endif
	glDisable(GL_LIGHTING);
	glColor3f(0.0f, 0.0f, 0.0f);
	setOrthographicProjection();
	glPushMatrix();
	glLoadIdentity();
	drawString("ESC          -- Exit the game");
	drawString("Direction    -- Move your position");
	drawString("Drag         -- Alter your viewport");
	drawString("Drag (Shift) -- Alter the light source");
#ifdef ENABLE_TRACE
	drawString("-----------------------------------------------------------------");
	drawString(cameraString);
	drawString(lightString);
	drawString(fpsString);
	drawString("-----------------------------------------------------------------");
#endif
	glPopMatrix();
	resetPerspectiveProjection();
	glEnable(GL_LIGHTING);

	glUseProgram(0);
	glFlush ();
    glutSwapBuffers();
}
```

```
void resetPerspectiveProjection()
{
	glMatrixMode(GL_PROJECTION);
	glPopMatrix();
	glMatrixMode(GL_MODELVIEW);
}
```

Conclusion

Nous avons vu dans cette partie, la description de notre application, où nous avons défini l'objectif de l'application, la partie de conception et on termine par la mise en œuvre de notre projet (les variables, structures de données, les méthodes utilisées).

La partie suivante est consacrée pour présenter quelques résultats de rendu obtenus par notre projet.

Chapitre 4:

Résultats

Introduction

Ce chapitre a pour objectif de présenter les différents résultats obtenus durant l'étape de réalisation de notre application.

Les spécifications matérielles qui permettent d'obtenir ces résultats sont :

- Type de processeur : Intel(R) Core(TM) i5-241M CPU @ 2.30GHz 2.30 GHz ;
- Capacité de la RAM : 4.00Go ;
- La carte graphique : NVIDIA GeForce 315M ;
- Système d'exploitation : Windows 8.1 Professionnel 64 bits ;
- Environnement: Visual Basic C++ ;
- Version OpenGL Core: 3.3 et version GLSL: 3.30.

1. Exemples de résultats de rendu

Gestion de clavier:

- ESC : Quitter le jeu
- Direction du clavier : Déplacez la position de la caméra.
- Glisser la sourie: Changez votre viewport.
- Glisser la sourie + Shift: Modifier la source de lumière

Figure 28 : Rendu initial du terrain.

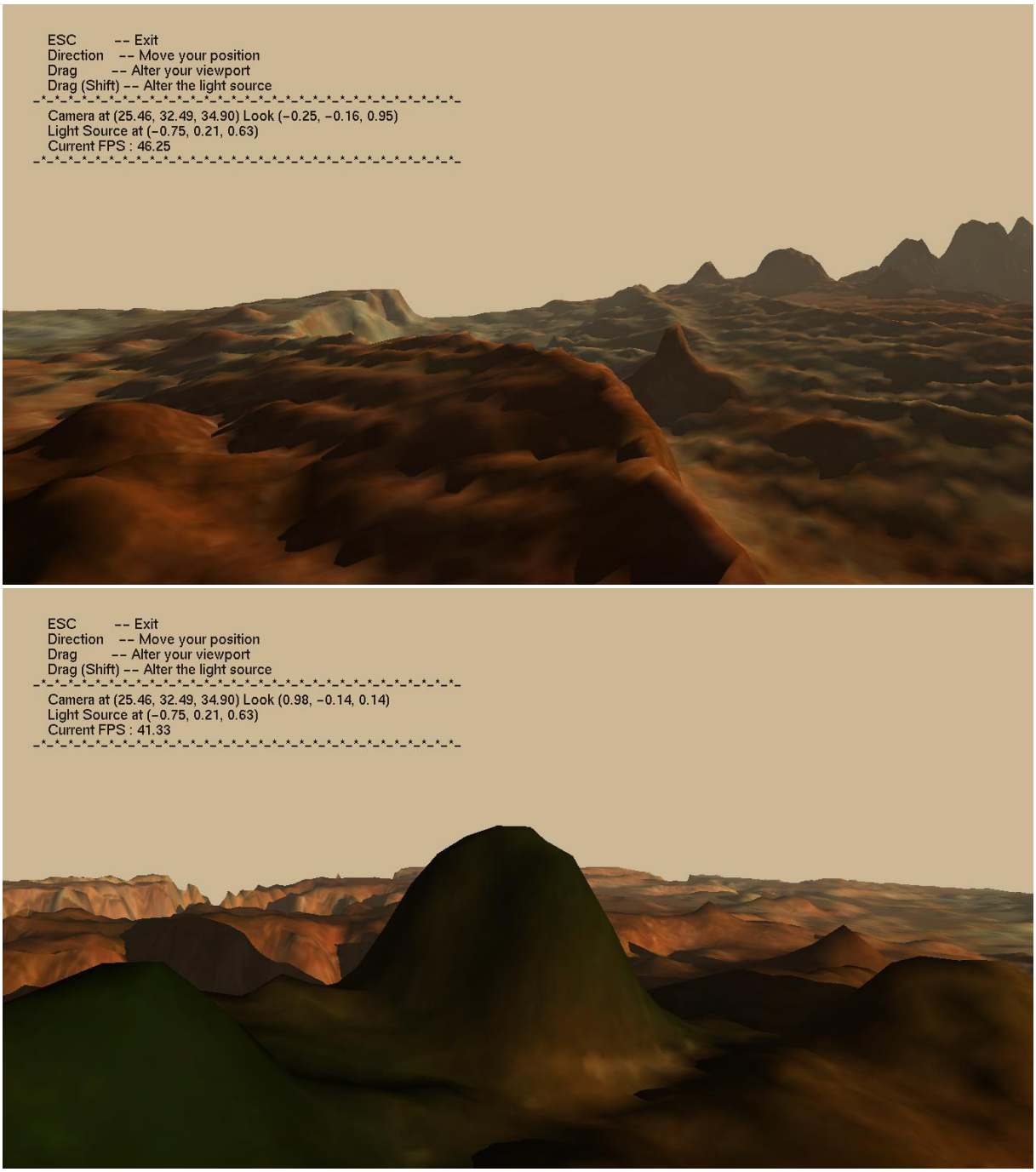


Figure 29: La scène après le changement de viewport

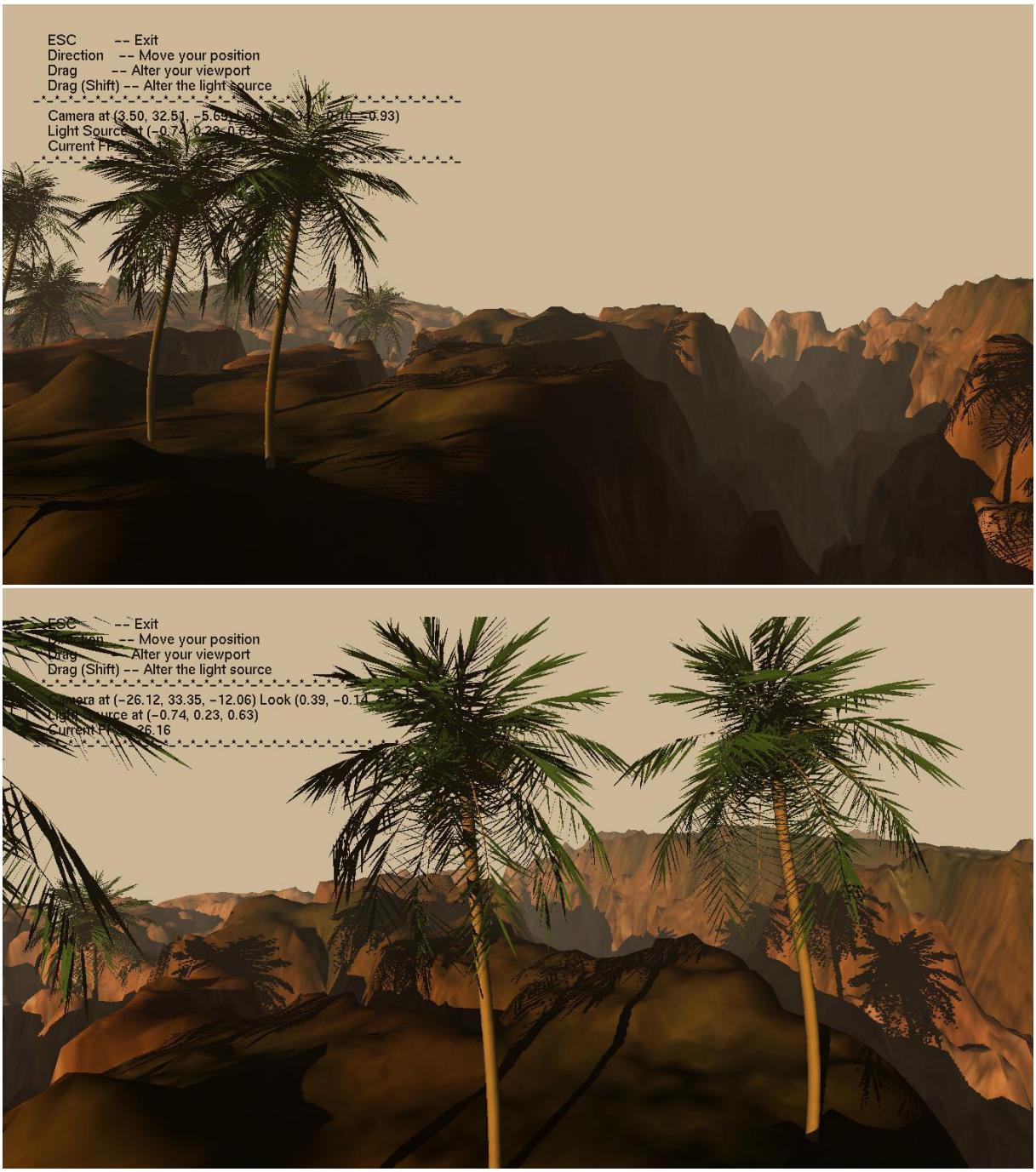
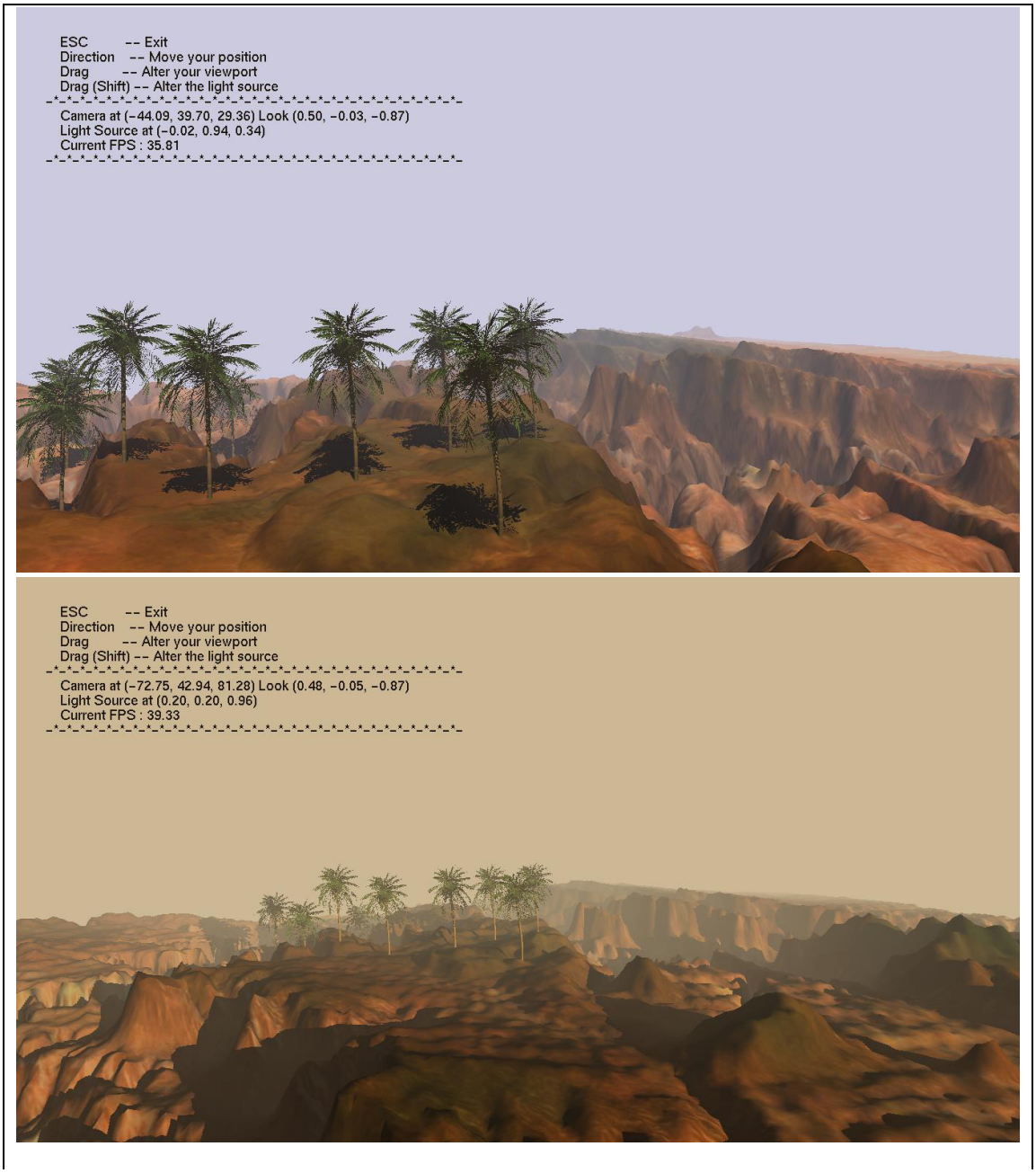


Figure 30: La scène après le changement de déplacement de la position du camera.


ESC -- Exit
Direction -- Move your position
Drag -- Alter your viewport
Drag (Shift) -- Alter the light source
Camera at (-44.09, 39.70, 29.36) Look (0.50, -0.03, -0.87)
Light Source at (-0.02, 0.94, 0.34)
Current FPS : 35.81
ESC -- Exit
Direction -- Move your position
Drag -- Alter your viewport
Drag (Shift) -- Alter the light source
Camera at (-72.75, 42.94, 81.28) Look (0.48, -0.05, -0.87)
Light Source at (0.20, 0.20, 0.96)
Current FPS : 39.33

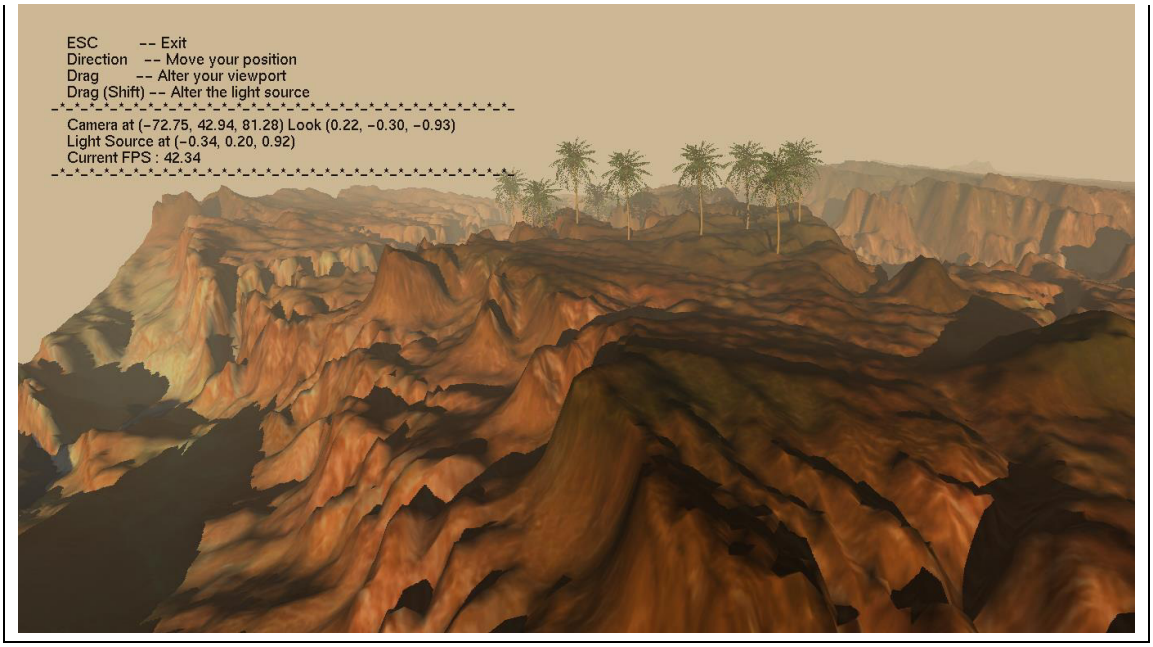


Figure 31: La scène après des changements de la position de la source.

2. Discussions

Les résultats obtenus et qui sont illustré dans la section précédente montre une représentation des cartes d'ombre dans un terrain 3D.

Les travaux présentés dans ce chapitre montrent que le calcul des cartes d'ombre peut être effectué en tout point de la scène sans sacrifier la vitesse du rendu. L'efficacité de notre approche repose d'une part sur la vitesse du GPU. Contrairement aux autres méthodes, où le calcul de la carte d'ombre exige un calcul sur le CPU.

Conclusion et perspectives

Dans ce chapitre on a décrit les outils de réalisation du notre projet, en présentant l'environnement matériel et logiciel exploité, dont le choix s'est porté sur la Shadow Map dans un terrain, puis nous avons montré quelques résultats obtenus par notre application.

Pour les perspectives nous proposant :

1. Utiliser une interface MFC au lieu d'une interface Glut pour faciliter l'interaction avec l'application, et de charger facilement nos modèles OBJ.

2. Ajouter une ou des sources lumineuses avec plusieurs méthodes et essayer de rendu la scène et leurs objets dans la nuit avec la lune comme une source de lumière.

3. Changer les modèles OBJ par d'autres modèles 3D comme par exemple les modèles MD2, 3DSMAX, MD3… etc.

4. Faire le rendu d'une forêt.

5. Proposer une méthode hybride en utilisant les deux techniques du rendu des ombres (volume d'ombre et la carte d'ombre).

Conclusion générale

A travers ce travail, nous avons présenté le rendu des terrains qui sont considérés comme des objets naturels et la technique d'ombrage « shadow map » pour les terrains en exploitant les performances des processeurs graphiques récents.

En effet, Les ombres jouent un rôle fondamental dans le réalisme d'une scène 3D. Les domaines d'application du rendu des cartes d'ombre sont très variés et intéressent plusieurs disciplines comme les jeux vidéo, et les films d'animations 3D.

Nous avons ensuite effectué une étude théorique sur les terrains et les cartes d'ombre en synthèse d'images et spécifiquement dans le domaine du rendu des terrains, cette étude nous a permis de proposer et de réaliser une application sur le rendu des terrains et shadow map.

En réalisant ce travail, nous avons pu acquérir :

1. Des connaissances sur le rendu des terrains et l'application des ombres et en particulier le rendu des cartes d'ombre dans les terrains.
2. Quelques techniques d'illumination et d'ombrage.
3. L'utilisation de la programmation GPU.

L'utilisation des bibliothèques graphiques et des shaders dans les algorithmes de synthèse permet d'optimiser ces algorithmes que ce soit dans le temps de calcul ou dans notre système pour éclairer une scène de synthèse avec les ombres.

Pour montrer le fruit de notre travail, nous avons consacré, tout un chapitre (résultats et expérimentation), pour bien présenter les résultats obtenus qui étaient satisfaisants et encourageants.

Bibliographie

[1] ZERARI A.El Mouméne (2011). Volume d'ombre en rendu temps réel. Thèse Magister: Informatique option Synthèse d'Image et Vie Artificielle. Biskra: Université de Biskra.

[2] Sylvain Lefebvre. Modèles d'habillage de surface pour la synthèse d'images.. Interface homme-machine [cs.HC]. Université Joseph-Fourier - Grenoble I, 2005.

[3] HAL. Modèles d'habillage de surface pour la synthèse d'images [En ligne]. https://hal.archives-ouvertes.fr/tel-00012154/document. Consulté le 03/02/2018.

[4] ACHOUR Nihad (2015). Shadow Map via les shaders Et les GPUs. Mémoire Licence: Informatique. Biskra : Université de Biskra.

[5] Cours de Synthèse d'image Master 1: Modèles d'illumination locale, ZERARI A.El Mouméne, Maître assistant "A" en informatique. Université de Biskra, 2017.

[6] CEGEP de Sainte- Foy. Principe de lumière [En ligne]. http://www2.cegep-ste-foy.qc.ca/departements/freesite-informatique/ProjetsRechercheH2005/A04-620-Equipe15/lumiere.htm. Consulté le 03/08/2017.

[7] Dupuis Emmanuel (2005). Gestion des ombres en temps réel. Rapport de stage: Master I STIC-Informatique: Université de Reims.

[8] = Michael schwàrzler. 'Accurate soft shadows in Real-Time Application'.Février2009

[9] = COHEN-OR D, CHRYSANTHOUY, SILVA C.T, DURAND F: A survey of visibility for walkthrough applications. IEEE Transactions on Visualization and Computer Graphics 9, 1(2003), 3–15

[10] = Cours**:** Chapitre 2.4 - Illumination en temps réel, Olivier Godin, enseignant en informatique. Université Sherbrooke. 2014.

[11] = ALLOPROF. Les sources lumineuses [En ligne]. http://test.alloprof.qc.ca/physique/generalites-en-physique/les-phenomenes-lumineux/les-sources-lumineuses.aspx . Consulté le 18/09/2017.

[12] = Edmond Boyer. Rendu réaliste. Cours Master informatique – Synthèse d'images. INRIA Centre de Recherche Grenoble-Rhônes-Alpes, 2009

[13] = Regards sur sciences. Le lancer de rayons [En ligne]. http://regards.sur.sciences.free.fr/images/pov/le_lancer_de_rayons.htm . Consulté le 18/09/2017.

[15] = MATHR. Adventures with radiosity [En ligne].https://mathr.co.uk/blog/2015-08-03_adventures_with_radiosity.html . Consulté le 22/10/2017.

[16] = Henni Nadir (2013). Illumination et ombre pour les forêts pour un rendu temps réel. Thèse de Magister: Informatique, Option : Synthèse d'Image et Vie Artificielle **:** Université de Biskra.

[17] = N.Janey : Modélisation et synthèse d'image d'arbres et bassins fluviaux combinant méthode combinatoires et plongement automatique d'arbres et cartes planaires. Thèse de doctorat à l'université de FRANCHE-COMTE .1992

[18] = Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch et François Sillion : "A survey of real-time soft shadows algorithms". In Computer Graphics Forum, volume 22(4), pp. 753 –774, 2003.

[18] = Julien Gerhards (2017): Partition de volumes d'ombre : une alternative pour le rendu d'ombres en temps réel. Thèse de doctorat: Informatique et Application. UNIVERSITÉ DE LIMOGES.

[19] = Lance Williams: Casting curved shadows on curved surfaces. In Proc. SIGGRAPH 78, volume 12, août 1978.

[20] = Raphaello. Le Z-Buffer [En ligne]. http://raphaello.univ-fcomte.fr/Ig/ZBuffer/ZBuffer.htm . Consulté le 12/02/2018.

[21] = Alexandre MEYER (2001). Représentations d'arbres réalistes et efficaces pour la synthèse d'images de paysages. Thèse de doctorat: informatique. Université Joseph Fourier.

[22] = J.M Snyder et A.H.Barr. Ray tracing complex models containing surface tessellations. July 1987. Pages 119 - 128

[23] = Tom Lokovic et Eric Veach : Deep Shadow Maps, 2000.

[24] = N.Max. Horizon mapping :shadows for bump-mapped surfaces. 1988

[25] = B.Cabral, N.Max et R.Springmeyer. Biderecrional reflection functions from surface bump maps. 1987

[26] = D.Cohen-Or et A.Shaked. Visibility and dead-zones in digital terrain maps. 1995

[27] = J.Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. 1998

[28] = J.Amanatides. Ray tracing with cones. 1984

[29] = P.S.Heckbert et P.Hanrahan. Beam tracing polygonal objects. 1984

[30] = W.T.Reeves, V.H.Salesion et R.L.Cook. Rendering antialiased shadow with deepth maps. 1987

[31] = T.Lokovic et E.Veach. Deep shadow maps. 2000

[32] = T-Y.Kim et U.Neumann. Opacity shadow maps. 2001

[33] = L.Carpenter. The a-buffer, an antialiased hidden surface method. 1984

[34] = P.Haeberli et k.Akeley. The accumulation buffer: Hardware support for high-quality rendering. 1990

[35] = F.C.Crow. Shadow algorithms for computer graphics. 1977

[36] = P.Bergeron. A general version of Crow's shadow volumes. 1986

[37] = M.Slater. A comparqison of three shadow volume algorithms. 1992

[38] = spirit-science. IMAGES FRACTALES, Une introduction pour les curieux et les non-connaisseurs [En ligne]. http://www.spirit-science.fr/Matiere/Fractales.html . Consulté le 11/03/2018

[39] = WHITTED (T.). -"An improved illumination model for shaded display.". - Communication of ACM, 23(6), juin 1980.

[40] = J.W.Patterson, S.G.Hoggar et J.R.Logie. Inverse displacement mapping. 1991

[41] = Les Ombres en synthèse 3D, http://joss16.free.fr/Les%20Ombres%20en%20synth%E8se%203D.pdf