Ombrage et illumination pour le rendu 3D de forêts en temps réel

Florent Cohen

▶ **To cite this version:**

Florent Cohen. Ombrage et illumination pour le rendu 3D de forêts en temps réel. Synthèse d'image et réalité virtuelle [cs.GR]. 2004. inria-00598409

HAL Id: inria-00598409

https://inria.hal.science/inria-00598409

Submitted on 6 Jun 2011

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

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

Institut National Polytechnique de Grenoble
Master Recherche
Imagerie, Vision, Robotique

Ombrage et illumination pour le rendu 3D de forêts en temps réel

Florent Cohen

Juin 2004

Projet effectué sous la responsabilité de
Philippe Decaudin et Fabrice Neyret
dans l'équipe EVASION du laboratoire GRAVIR

EVASION - Laboratoire GRAVIR
INRIA Rhône-Alpes
ZIRST
655 avenue de l'Europe
Montbonnot
38334 Saint Ismier Cedex
http://www-evasion.imag.fr

Table des matières

1 Introduction.. 5
1.1 Présentation de l'équipe.. 5
1.2 Présentation du sujet...6
1.3 Motivation.. 7
1.4 Plan du rapport..7
2 Le rendu par texcells.. 8
2.1 Présentation des texcells..8
2.2 Le mapping apériodique.. 9
1 Les ombres de la forêt : présentation du problème et état de l'art....................................... 10
1.1 Analyse.. 10
Les différents types d'ombres rencontrées en forêt... 10
Contraintes imposées par la méthode de rendu par texcell...11
Évaluation des techniques envisagées... 11
1.2 Présentation des techniques envisagées..12
Le polygone d'ombre (shadow volume)...12
La projection planaire.. 15
La projection sur des surfaces courbes... 16
La depth shadow map.. 17
Les horizons maps...18
Les visibility cube maps..19
Méthodes empiriques de Reeves..20
1.3 Conclusion au sujet de l'ombrage.. 21
2 Illumination de la forêt : présentation du problème et état de l'art.......................................22
Évaluation des techniques envisagées... 22
Le problème du calcul des fonctions d'illumination pour les objets lointains....................... 23
2.1 Présentation de techniques envisagées pour l'illumination des arbres..............................23
Textures de fonctions d'illumination...23
Texture de normales pour la composante diffuse... 24
Évaluation... 24
méthodes de Schlick : approximation par des fonctions rationnelles...................................25
Approximation non linéaire de fonctions d'illuminations (Lafortune)................................. 26
2.2 Conclusion au sujet de l'illumination de la forêt.. 27
3 Ombrage des texcells..28
5.1 Depth shadow map... 28
3.1 Double horizon-maps... 29
Principe...29
Génération des cartes.. 30
Phase de rendu.. 31
Notre implémentation... 33
Analyse... 34
3.2 Auto-ombrage empirique..34
Principe...34
Phase de rendu.. 35
5.4 Résultat final..35
6 Illumination des texcells...37
6.1 Hypothèses préliminaires...37
6.2 Obtention du repère local à chaque éléments des texcells..38

Repère monde → repère local à chacune des faces........ 38
Calcul des cartes de normales associées à chacune des faces........ 39
6.3 Calcul de l'illumination........ 39
6.4 Phase de rendu........ 40
6.5 Mip-mapping........ 41
6.6 Résultats et analyse........ 42
7 Illumination et ombrage........ 44
7.1 Un problème : la carte graphique........ 44
7.2 Une solution dégradée........ 44
8 Bilan et conclusion........ 46
9 Bibliographie 47
10 Annexe A – Listing des vertex et fragment programs........ 50
10.1 Double horizon map et ombrage empirique........ 50
Vertex program........ 50
Fragment program........ 50
10.2 Illumination de Lafortune........ 51
Vertex program........ 51
Fragment program........ 52
10.3 Illumination et ombrage dynamique........ 52
Vertex Program........ 52
Fragment Program........ 53
11 Annexe B - Poster SIGGRAPH 2004........ 55

1 Introduction

Rendre des scènes naturelles de manière réaliste est un sujet d'intérêt majeur pour la recherche en images de synthèse ainsi que pour l'industrie, en particulier pour les simulateurs ou les jeux vidéos. Dans ces applications, les images doivent être calculées et affichées entre 15 et 60 fois par seconde, et ce afin de tromper le cerveau de l'utilisateur et lui donner une sensation de mouvement continu et d'immersion - nous appellerons par la suite des rendus atteignant cette fréquence des rendus temps réél. Ces éléments sont essentiels pour afficher des paysages de manière réaliste, par exemple pour des simulateurs de vol.

Pourtant, rendre en temps réél des scènes comportant de la forêt est à cause de sa grande complexité géométrique une tâche difficile à réaliser. Comment, dans ce contexte, y rajouter un calcul d'éclairage et de l'ombrage dynamique, tout en préservant la vitesse de rendu des images ?

1.1 *Présentation de l'équipe*

L'équipe EVASION du laboratoire GRAVIR effectue des recherches en modélisation, animation, et visualisation d'objets et de phénomènes naturels. Pour cela, deux grand axes de recherche sont privilégiés : d'une part le développement d'outils fondamentaux destinés à spécification de scènes et objets naturels complexes, à la mise au point modèles alternatifs pour la forme, le mouvement et l'apparence ainsi qu'à la conception d'algorithmes reposant sur un niveau de détail adaptatif pour gérer au mieux la complexité ; d'autre part la validation de ces outils sur des scènes naturelles spécifiques, qui vont du monde minéral (océan, ruisseaux, lave, avalanches, nuages) au monde animal (simulation d'organes, visages corps et chevelure d'un personnage, mouvements d'animaux), en passant pas les scènes végétales (morphogénèse de plantes, prairies, arbres). Les axes de recherche sont :

- Développement d'outils fondamentaux
  - Spécification de scènes et objets naturels
  - Modèles alternatifs pour la forme, le mouvement et l'apparence
  - Algorithmes adaptatifs et niveaux de détail
- Étude de scènes naturelles spécifiques et applications
  - Scènes minérales (océan, ruisseaux, lave, avalanches, nuages)
  - Scènes végétales (morphogénèse de plantes, prairies, arbres)
  - Monde animal (simulation d'organes, visages et corps d'un personnage, mouvements d'animaux)

1.2 *Présentation du sujet*

Saint-Laurent-du-Maroni, Guyane

Vue d'une forêt réalisée par ordinateur (David J. Buckley, Craig Ulbricht et Dr. Joseph Berry)

Rendre de manière réaliste une forêt est une tâche difficile, car on peut remarquer que celle-ci est constituée d'une multitude d'arbres, tous différents les uns des autres, eux même construits selon des lois naturelles relativement complexes. En particulier, l'ombrage due aux variations du terrain, aux arbres entre eux, et à la forêt sur sol, est un aspect essentiel du réalisme. De même, l'apparence de la forêt est fonction de la direction du soleil, qui, à cause de fait que le feuillage est translucide, éclaire plus ou moins toutes les feuilles de la forêt.

Illustration 1- Qin et al
Fast Photo-Realistic Rendering of Trees in Daylight

Il est possible d'obtenir des images approchant la réalité en utilisant des techniques de rendu lourdes en calcul, telles que par exemple le lancer de rayon, souvent combinées à des méthodes qui utilisent la géométrie complète des arbres. Le résultat, bien qu'imparfait, peut être décrit comme réaliste. Mais ces techniques nécessitent plusieurs minutes, voire plusieurs heures de calcul par image, et ne sont donc pas adaptées au temps réél.

L'équipe EVASION travaille sur diverses représentations alternatives (i.e. sans maillage polygonal) permettant de visualiser des forêts de façon réaliste et efficace, voire temps réel. Ces représentations exploitent les propriétés de l'objet à représenter : pour une forêt, la similarité permet d'utiliser des textures, et la concentration des détails visuels au voisinage de la surface permet d'envisager (notamment) des techniques volumiques.

Cependant lorsque l'on visualise tout un paysage, comme précédemment indiqué, gérer la forme du moutonnement dû aux arbres n'est pas tout. L'objectif du stage est de trouver des méthodes adaptées d'illumination et d'ombrage en temps réel de forêts rendues pour ce type de représentations.

1.3 Motivation

Illustration 2 - Rendu d'une forêt utilisant la technique des texcells (Philippe Decaudin, Fabrice Neyret)

Nous nous intéressons au survol de forêts en milieu vallonné. De nombreuses recherches ont été réalisées afin d'obtenir un rendu réaliste temps réel, on peut notamment citer les techniques à base de surfels[PFISTER00], ou bien de billboards[MACIEL95], une des méthode les plus prometteuses étant la représentation de la forêt par une couche de texture volumique qui recouvre le terrain, développée à EVASION. Cette méthode est appelée texcells[DN04].

Toutefois, cette méthode utilise un éclairage et un ombrage précalculés, ce qui est gênant car on assiste à une absence d'ombre portée. De plus le changement de position du soleil n'est pas pris en compte.

Nous cherchons à enrichir cette méthode de rendu par de l'ombrage et de l'illumination créés par une source de lumière dynamique en temps réel, en s'appuyant sur la nouvelle génération de cartes graphiques. Les processeurs de celles-ci, appelés GPU (Graphic Processor Unit, par opposition au CPU, Central Processing Unit, le processeur de l'ordinateur), permettent, pour chaque pixel affiché à l'écran, de réaliser des opérations programmables, ce qui jusqu'ici était impossible, accroissant ainsi le réalisme en minimisant le coût.

1.4 Plan du rapport

Nous commencerons par une présentation de la méthode de rendu par texcells dans laquelle nous avons intégré nos travaux au chapitre 2, puis nous ferons un état de l'art pour, d'une part, les techniques d'ombrage dynamique et d'autre part pour les techniques d'illumination dynamique (respectivement aux chapitres 3 et 4).

Nous présenterons ensuite les solutions que nous proposons d'abord pour l'ombrage (chapitre 5) et pour l'illumination (chapitre 6), chacune étant discutée, implémentée et analysée. Nous reprendrons au chapitre 7 ces techniques afin de les coupler et d'obtenir simultanément l'ombrage et l'illumination, puis nous conclurons.

2 Le rendu par texcells

2.1 Présentation des texcells

L'impératif du rendu temps réel a nécessité l'utilisation de techniques d'optimisation disponibles sur les cartes graphiques des ordinateurs personnels ; toutefois, ces cartes ne savent pas rendre l'ombrage d'une scène « directement » d'une part, et d'autre part la méthode d'illumination utilisée est une méthode d'illumination locale, telle que la technique de l'ombrage à plat ou celles décrites par Gouraud [GOURAUD71] ou Phong[PHONG75].

Dans le cadre de ce stage de DEA, nous utiliserons une méthode[MN98] qui permet l'affichage de forêt en temps réel, actuellement avec un ombrage et une illumination précalculée et que nous souhaitons faire évoluer vers un ombrage et une illumination dynamiques.

Cette méthode repose sur les textures volumiques[WILSON94]. Une texture volumique est constituée d'un motif cubique contenant un échantillon de forêt qui sera répété sur le terrain. Comme les cartes graphiques ne savent pas rendre directement les textures volumiques, celle-ci sera découpée en tranches texturées.

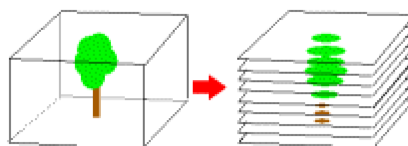


Illustration 3 - découpage en tranches texturées d'un motif (ici un seul arbre)

Ce motif de forêt a été généré par un logiciel de rendu offline (3DSMax de Discreet), et contient en chacun de ses points la transparence (l'alpha) et la couleur (résultante des interactions lumineuses) du bout de voxel considéré.

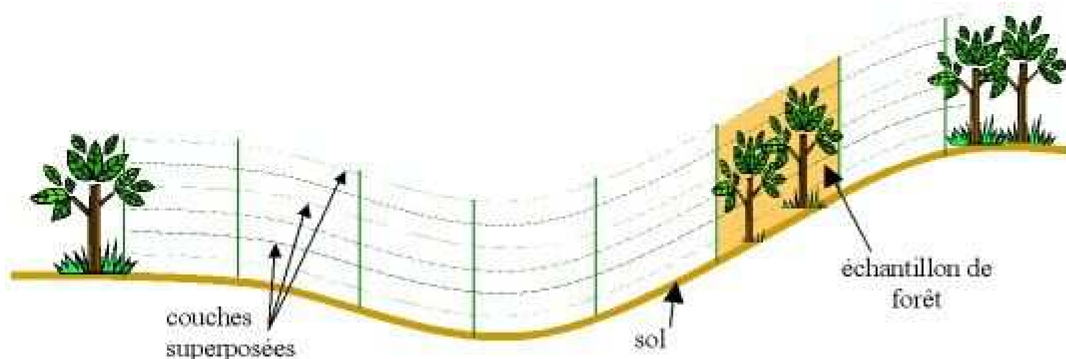
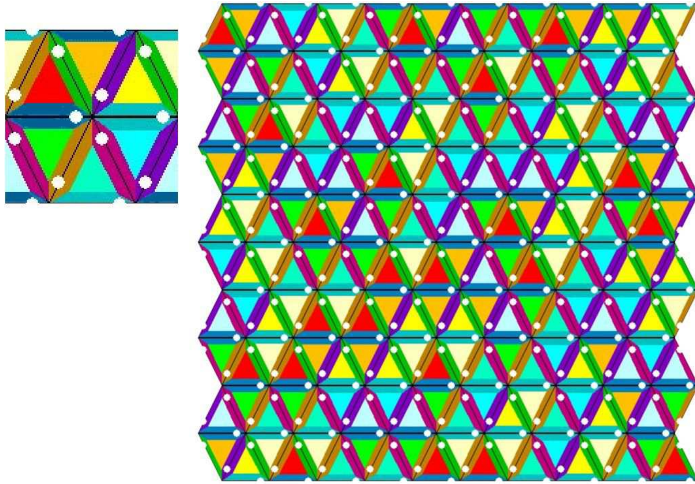


Illustration 4 - Tranche verticale de forêt recomposée

Le rendu de forêts est obtenu en affichant des couches superposées de surfaces texturées parallèles à la surface du sol.

2.2 Le mapping apériodique

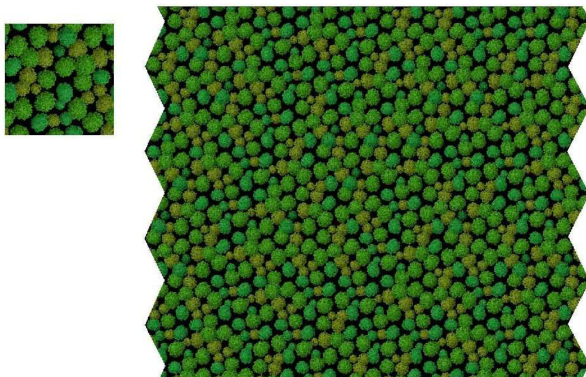
Afin de diminuer l'effet de répétition que l'on obtiendrait si on répétait régulierement le motif (ce qui donne l'impression d'une forêt où tous les arbres ont été soigneusement alignés), nous utilisons une solution[NC99] à base de sets de patterns triangulaires compatibles qui résout à la fois les problèmes de périodicité et de frontières.



Le motif (à gauche) est utilisé pour remplir une zone rectangulaire (à droite) ; pour cela le motif est découpé en triangles suivant les traits noirs, ces triangles sont placés aléatoirement les uns à côté des autres pour remplir la zone, mais en respectant des contraintes sur les arêtes qui permettent de garantir la continuité du dessin ainsi obtenu.

Les couleurs utilisées sur le motif symbolisent les contraintes :

- les arêtes de même couleur doivent être compatibles (i.e. un arbre 'posé' sur une arête d'une certaine couleur doit être dupliqué et posé au même endroit sur les autres arêtes de la même couleur),
- les sommets des triangles ne sont pas couverts (i.e. les arbres ne couvrent pas les sommets des triangles),
- un objet peut couvrir au plus une seule arête (i.e. un arbre peut être contenu complètement dans un triangle ou être à cheval sur deux triangles, mais pas plus),
- le motif est cyclique horizontalement : tout ce qui dépasse sur le bord gauche se retrouve sur le bord droit et inversement,
- le motif est cyclique verticalement mais avec un décalage : nous suivons les contraintes des arêtes donc tout ce qui traverse une arête bleu clair en haut du motif se retrouve en bas du motif décalé sur l'arête bleu claire correspondante et inversement, idem pour l'arête bleu foncée.



L'utilisation d'un tel 'mapping apériodique' casse la répétition et donne l'impression que chaque cellule est unique.

De plus, pour simuler les effets de lumières dûs au relief, on calcule en chaque vertex (sommet géométrique, avec éventuellement des informations supplémentaires) des cellules la valeur de l'ombrage de Gourand ; on multiplie alors l'intensité de la texture par cette valeur.

Les données sont donc des données volumiques, stockées dans des textures.

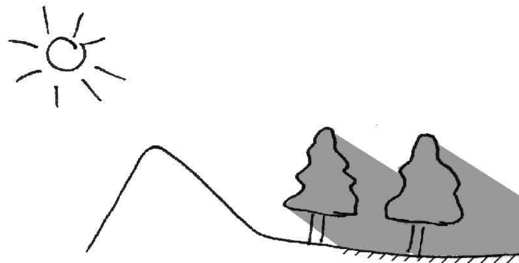
1 Les ombres de la forêt : présentation du problème et état de l'art

1.1 Analyse

Les différents types d'ombres rencontrées en forêt

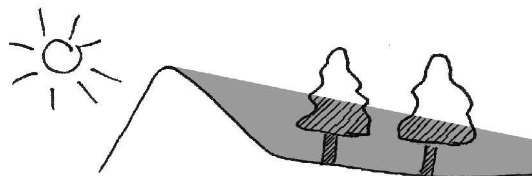
Compte tenu des paramètres de la technique de rendu choisie, les ombres dans une forêt peuvent se décomposer en 4 types :

- Ombrage de la forêt sur le sol



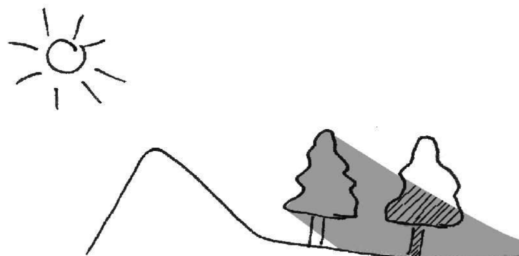
Les arbres, en bloquant les rayons du soleil, agissent sur la luminosité reçue en chaque endroit du sol.

- Ombrage du terrain sur la forêt



Le relief, en faisant obstacle par ses déformations aux rayons provenant de la source lumineuse (le soleil), projette son ombre sur les arbres de la forêt.

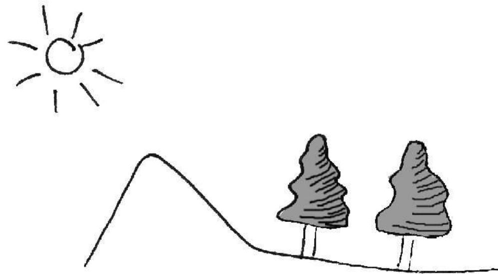
- Ombrage des arbres entre eux



Un arbre dans une forêt n'est pas isolé : des arbres se trouvent entre lui et la source lumineuse et il

est donc susceptible d'être partiellement ombré.

- l'auto-ombrage



Le feuillage et les branches de l'arbre bloquent également la lumière : l'arbre n'est donc pas, de par sa forme, entièrement éclairé. De plus, cette ombre se trouve également à l'intérieur de l'arbre (et pas uniquement sur sa surface), intérieur qui est visible par un observateur (un arbre contient énormément de vide). Cette ombre interne participe à la perception du volume de l'arbre.

Une image de forêt réaliste se doit de recréer ces quatre types d'ombres ; pour cela, il sera nécessaire de sélectionner une/des techniques d'ombrages qui tiendront compte des contraintes imposées par notre solution de rendu.

Contraintes imposées par la méthode de rendu par texcell

La technique de rendu sélectionnée dans le cadre de ce stage (décrite page 8) impose les caractéristiques suivantes :

- Les données concernant les arbres ne sont pas des données géométriques mais volumiques, codées dans plusieurs textures.
- Il faut beaucoup de tranches texturées pour représenter la forêt ; de plus, ces tranches sont souvent « vides » ou faiblement opaques. Traiter ces zones peu porteuses d'informations est coûteux, il serait donc avantageux de restreindre l'application des calculs aux éléments opaques.

Évaluation des techniques envisagées

Afin d'évaluer les techniques du rendu d'ombres, nous nous baserons sur les paramètres suivants :

- **type d'ombres :** ombres dures / ombres douces. Les ombres à profil net sont dites ombres dures, et sont irréalistes. Les ombres douces (soft shadow) ont un contour "flouté" comme dans la réalité. Elles apportent un plus grand réalisme à la scène.
- **adaptée à la transparence :** si le bloqueur, c'est-à-dire l'objet qui va intercepter la lumière et créer une ombre portée, est semi-translucide, l'ombre est-elle aussi semi-translucide?
- **précision du résultat :** certaines méthodes de calculs introduisent (volontairement ou non) des approximations pouvant faire apparaître des « ombres fantômes » ou des trous dans les ombres. Il ne faut pas confondre cette catégorie avec le type d'ombres ; en effet, l'obtention d'une ombre douce est un résultat volontaire!
- **type de données :** géométriques / volumiques.
- **forme du récepteur (plane, quelconque, etc...) :** le terrain n'étant pas plan et le motif étant quelconque, on comprend mieux l'importance de ce paramètre.
- **gestion de l'auto-ombrage :** généralement, les méthodes d'ombrages ne génèrent pas automatiquement l'auto-ombrage.

- **vitesse :** l'algorithme mis en oeuvre est-il long à calculer par le GPU ?
- **implémentation software/hardware:** l'algorithme est bien adapté aux propriétés « cablées » des cartes graphiques (cas hardware) ou bien doit être écrit sous forme de programme (cas software).
- **coût CPU/mémoire**.
- **espace objet/espace image :** L'espace objet est l'espace de description mathématique (ou repère) associé au modèle géométrique de l'objet dans lequel chaque point est représenté par un triplet de coordonnées. Cet espace correspond au monde réel dans lequel se situe l'objet. Les objets y sont décrits avec une précision maximale. A l'opposé, l'espace image est associé à la projection des objets sur le plan image. Cet espace ne correspond qu'à une représentation partielle de l'objet et est fonction de la définition de l'écran.

Le but est de choisir les techniques apportant le meilleur réalisme à moindre coût.

1.2 Présentation des techniques envisagées

Le polygone d'ombre (*shadow volume*)

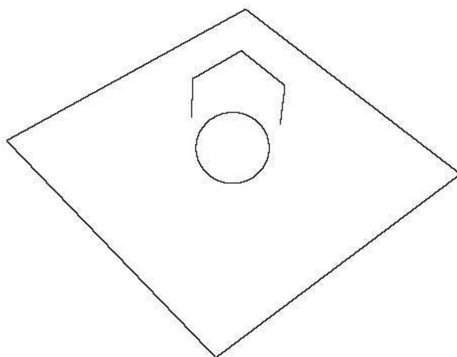
Le principe de cette technique est de déterminer le volume de l'espace qui est dans l'ombre (pour une source donnée) d'un objet, en se basant sur la géométrie de celui-ci[CROW77]. Nous présentons ici une version de l'algorithme adaptée aux propriétés des cartes modernes[HEIDMANN91].

Algorithme

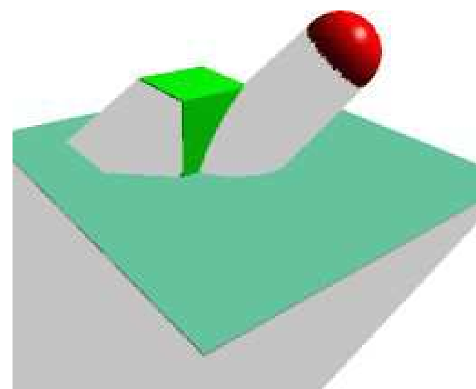
Silhouette

Pour ce faire, une première étape consiste à trouver le contour des objets vues depuis la source, en se basant sur les données géométriques de ceux-ci.

On projette ensuite « à l'infini » dans la direction des rayons lumineux les contours de cette silhouette, ce qui détermine un volume de l'espace ; on recommence ainsi pour tous les objets bloquant la source, puis on réalise l'union de tous ces volumes : on obtient un volume, qu'on appelle *shadow volume*.



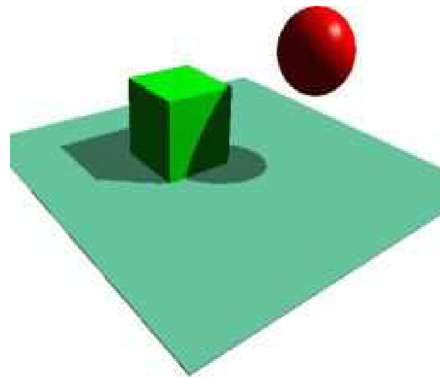
Détermination des contours telles que vus depuis la source lumineuse.



Obtention du shadow volume.

Lors du rendu de la scène, pour chaque fragment, on compare la position de l'espace lui correspondant avec le « shadow volume » déterminé à l'étape précédente : s'il est dedans, on le colorie de manière à ce qu'il apparaisse comme ombré. Sinon, on ne le modifie pas.

On obtient ainsi une scène avec des ombres projetées.



Utilisation du stencil buffer et rendu en deux passes

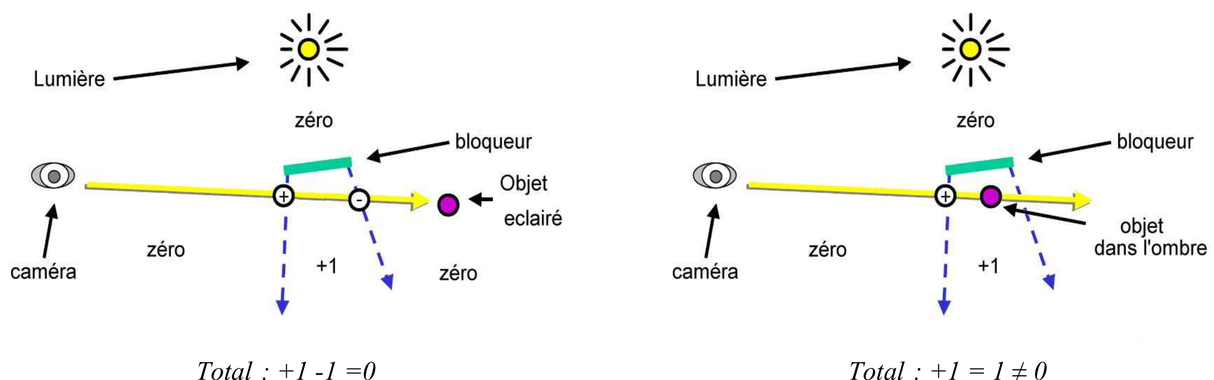
L'opération délicate consiste à la réaliser de l'union des volumes, surtout pour déterminer les nouvelles équations des surfaces. Heureusement, les cartes graphiques possèdent différentes possibilités qui permettent de se passer de cette union. En effet, les cartes possèdent un tapon mémoire qui peut leur servir de pochoir électronique (le *stencil buffer*). L'idée serait de trouver la zone sur l'image de destination qui doit être ombrée, puis d'assombrir toute cette zone sans toucher aux pixels se trouvant en dehors.

La technique consiste alors à rendre la scène sans se soucier des ombres. On initialise les valeurs du stencil buffer à 0.

Ensuite, pour chaque objet, on rend alors les polygones orientés dos à la caméra (en utilisant le culling) et avec le test de profondeur « plus proche ou équidistant » activé. Pour chaque pixel de l'image de destination, au lieu de mettre à jour la couleur pixel, on décrémente la valeur dans le stencil buffer lui correspondant. Ainsi, si un point de l'espace correspondant à un pixel se trouve derrière ce volume, sa valeur dans le stencil buffer diminue. Au contraire, si le point se trouve devant, sa valeur ne bougera pas.

On recommence l'opération, cette fois-ci avec les polygones faisant face à la caméra et en incrémentant la valeur. Ainsi, si un point de l'espace correspondant à un pixel se trouve derrière ce volume, sa valeur dans le stencil buffer augmente. Au contraire, si le point se trouve devant, sa valeur ne bouge pas.

Considérons maintenant tous les cas, pour un bloqueur unique :



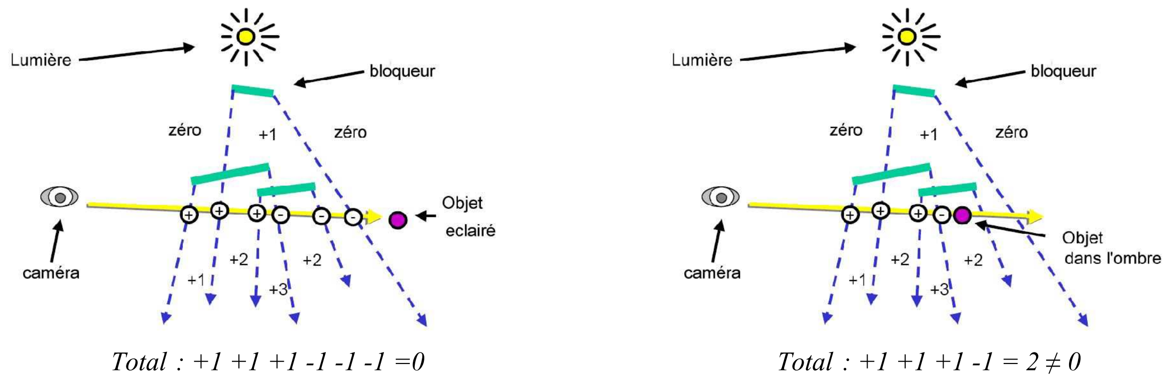
Total : +1 -1 =0 *Total : +1 = 1 ≠ 0*

Un point est entre le shadow volume et la caméra : sa valeur de stencil est 0 (une incrémentation puis une décrémentation)

Un point est dans le shadow volume : sa valeur de stencil est 1 (une incrémentation)

Un point est derrière le shadow volume : sa valeur de stencil est 0 (aucune opération)

On le voit bien, les points dans l'ombre ont leur valeur de stencil non nul. En assombrissant ces zones, on obtient bien l'ombre de l'objet. Cet algorithme se généralise lorsqu'il y a plusieurs bloqueurs.



Total : +1 +1 +1 -1 -1 -1 =0

Total : +1 +1 +1 -1 = 2 ≠ 0

Remarques

Sur des cartes graphiques récentes, on peut spécifier des opérations différentes en fonction du culling des triangles : une passe seulement est alors nécessaire. Certaines cartes peuvent même segmenter l'espace pour ne considérer que l'espace du shadow volume qui participera à la génération de l'image finale, et également adoucir les bords de l'ombre[NVIDIA01].

Un problème se pose si la caméra se trouve dans l'ombre d'un objet : le polygone d'ombre n'a plus de frontières entre la caméra et les objets de la scène, ce qui perturbe le résultat des calculs avec le stencil (la comparaison ne doit plus se faire avec 0). Ce problème a été résolu de manière générique[CARMACK00] et l'algorithme devient :

> Dessiner les faces arrières, ne rien faire si le test de profondeur est validé et incrémenter la valeur du stencil sinon.
>
> Dessiner les faces avant, ne rien faire si le test de profondeur est validé et décrémenter la valeur du stencil sinon.

Évaluation

type d'ombres	ombres dures
adaptée à la transparence	non
précision du résultat	totale
type de données	géométrique
forme du récepteur	quelconque
gestion de l'auto-ombrage	oui
vitesse	le coloriage de l'ombre est lent (3 passes : 2 pour le calcul du stencil, 1 pour le remplissage). déterminer la silhouette est coûteux.
implémentation	hardward
coût CPU/mémoire	calcul de la silhouette (CPU)
espace	objet

La projection planaire

Algorithme

Chaque objet créant de l'ombre dans la scène est rendu deux fois, une fois normalement et une fois sous la forme d'un objet « ombre » plane, qui une fois colorié en noir donnera l'ombre de l'objet.

Pour ce faire, on calcule une nouvelle matrice monde en se basant sur les propriétés (type, position...) de la lumière et sur celles du plan receveur de l'ombre de l'objet[HAINES03] [TESSMAN89].

Considérons un vertex v, « illuminé » par une source lumineuse L située en l, et dont la projection sur le plan d'équation $\pi : \vec{n} \dot{\vec{x}} + d = 0$, se nomme p.

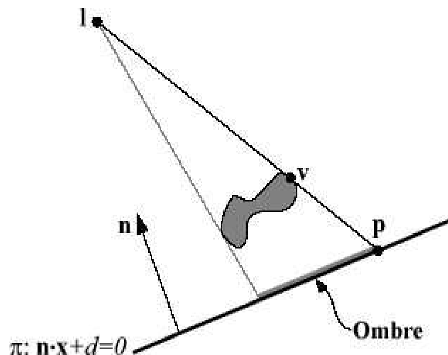


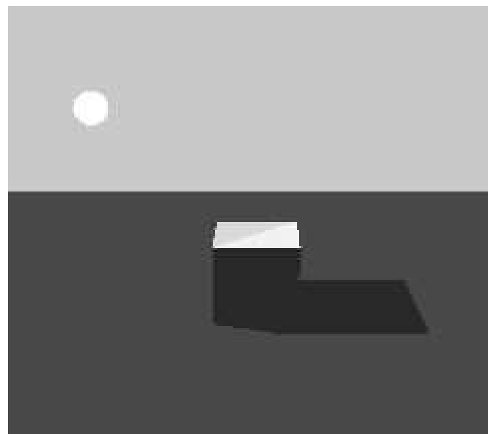
Schéma représentant les différents éléments du problème

La matrice monde à appliquer est :

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

Elle correspond à la projection depuis la source lumineuse L de tout vertex sur le plan π.

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



Résultat du rendu de l'ombre d'une boite par projection planaire

Les points éventuellement délicats sont :

- les ombres semi-transparentes, qui nécessitent la non-superposition des éléments constituants l'ombre. Une des solutions est de transformer la représentation de l'objet en une représentation convexe, ce qui assure, si le culling est activé, qu'un seul polygone couvre chaque pixel se trouvant dans l'ombre.
- le problème du rendu des ombres sortant de l'objet plane receveur de l'ombre : des tests supplémentaires (tests de profondeur ou du stencil buffer) doivent être effectués afin de déterminer

quelles parties de l'ombre doivent effectivement être affichées.

A cause de la précision limitée du z-buffer, le test de profondeur peut ne pas discriminer avec exactitude qui de l'ombre ou du sol se trouve le plus proche de l'observateur ; dans certains cas, l'ombre peut passer « sous » le sol, et ce de manière totalement arbitraire. Au pire, on peut observer un phénomène de « flipping » dans une séquence d'images.

Évaluation

type d'ombres	ombres dures
adaptée à la transparence	oui
précision du résultat	totale
type de données	géométrique
forme du récepteur	plan
gestion de l'auto-ombrage	non
vitesse	très rapide : 1 passe
implémentation	hardware
coût CPU/mémoire	nulle
espace	objet

La projection sur des surfaces courbes

Algorithme

Nous l'avons vu, une des limitations de la technique de la projection planaire est qu'une ombre ne se plaque que sur une surface plane. Une extension de cette méthode a été proposée[NGUYEN99] afin de pouvoir projeter une ombre sur tout type de surface.

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

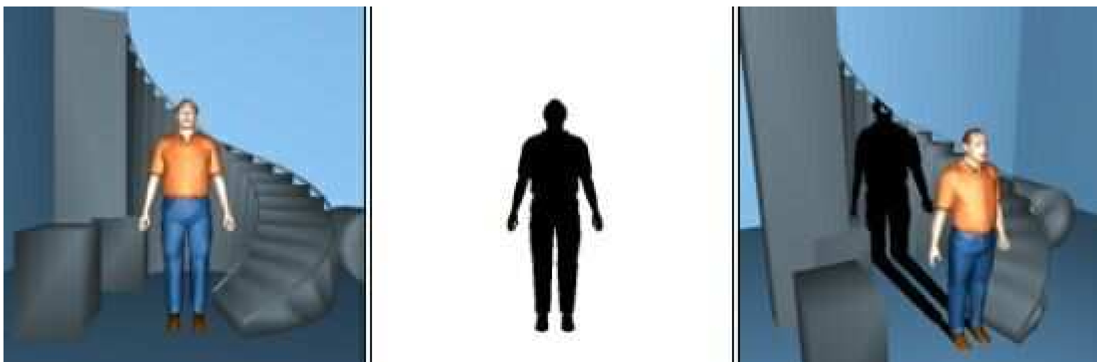


Illustration 5 - les 3 étapes de la projection sur des surfaces courbes

On rend ensuite les objets recevant l'ombre en plaquant dessus la texture précédemment obtenue, projetée en utilisant un mode de projection et une matrice de texture calculés à partir des propriétés de la lumière source.

Enfin, on rend dans un dernier temps le bloqueur pour compléter la scène.

Évaluation

type d'ombres	ombres dures
adaptée à la transparence	oui
précision du résultat	En fonction de la résolution de la texture, des artefacts peuvent apparaître sur les bords des ombres.
type de données	quelconque
forme du récepteur	quelconque
gestion de l'auto-ombrage	non
vitesse	correcte : il faut rendre 2 fois les bloqueurs
implémentation	hardware
coût CPU/mémoire	Stockage de la texture.
espace	image

Une des difficultés est qu'il faut déterminer qui est le bloqueur et qui est receveur de l'ombre.

La depth shadow map

Présentation de la technique

Il équivalent de dire qu'un point est dans l'ombre d'un objet par rapport à une source lumineuse que de dire que le point n'est pas visible par cette même source[CROW77] : le problème de l'ombrage se ramène à un problème de visibilité. La « *depth shadow map* » utilise cette propriété.

Son principe[WILLIAMS78], adapté pour le rendu sur les cartes graphiques actuelles, en est le suivant : on rend la scène (ou mieux uniquement sa profondeur) depuis la lumière et l'on transforme le tampon des profondeurs en une texture (appelée texture de profondeur).

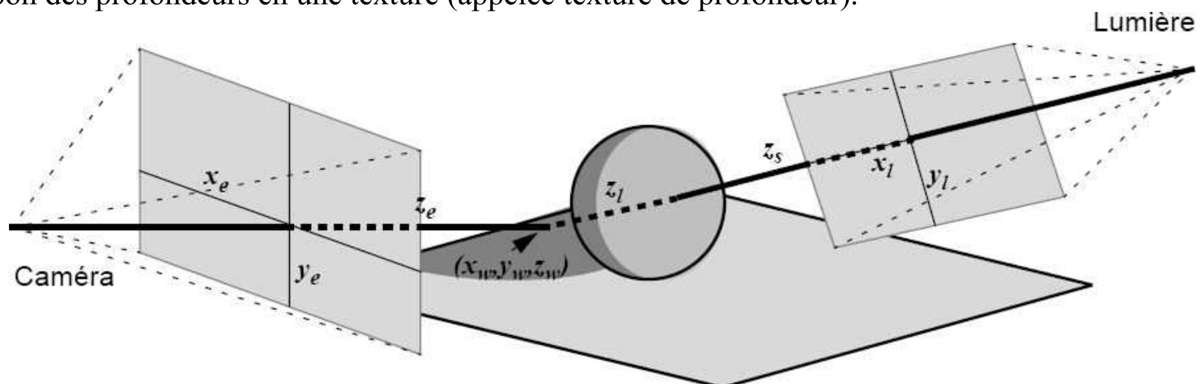


Illustration 6 - Notations utilisées pour le calcul des ombres avec une shadow map

On rend ensuite normalement la scène, mais en projetant la texture de profondeur sur les objets de la scène depuis la lumière (en utilisant la matrice de texture) ; on réalise alors en chacun des points dessinés l'opération suivante :

- si la profondeur du point, vu depuis la lumière, est égale à la profondeur contenue dans le texel associé au point, alors le point est visible depuis la source de lumière et il est donc éclairé : on ne fait aucune opération sur le fragment.
- par contre si la profondeur du point, vu depuis la lumière, est plus grande (comme sur le schéma : ici la profondeur du point est égale à $z_1 + z_s$), il doit y avoir un bloqueur entre le point et la lumière : on diminue l'intensité du fragment afin de simuler de l'ombre.

Évaluation

type d'ombres	ombres dures
adaptée à la transparence	non
précision du résultat	En fonction de la résolution de la texture, des artefacts peuvent apparaître sur les bords des ombres (correction possible en software [FERNANDO01]). De plus, la précision des mesures de profondeur est limitée.
type de données	quelconque
forme du récepteur	quelconque
gestion de l'auto-ombrage	oui
vitesse	correcte (2 passes), indépendante de la complexité de la scène.
implémentation	hardware
coût CPU/mémoire	nulle
espace	Image

Remarque

Il existe une méthode hybride permettant de determiner le shadow volume à partir d'une shadow map [MCCOOL00]; nous n'en parlerons pas ici, car cette méthode serait longue et coûteuse si elle était appliquée au rendu par texcell.

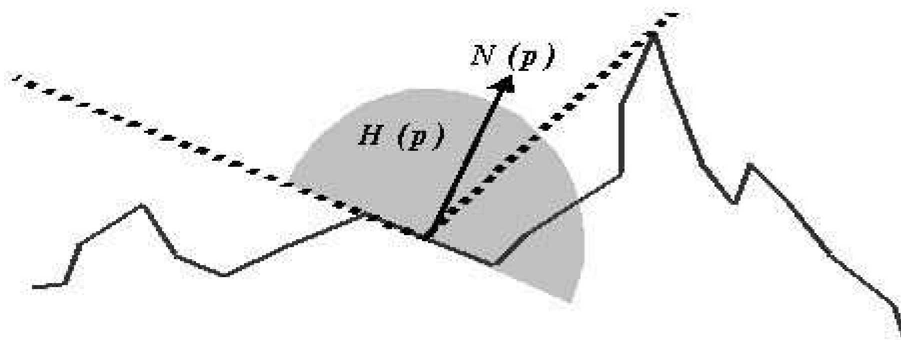
Les horizons maps

Présentation de la technique

Cette technique[MAX88][SLOAN2000] utilise une carte précalculée permettant de connaître pour tout élément d'une surface, l'angle minimal à partir duquel celui-ci est éclairé par une source lumineuse située dans une direction donnée.

Précalcul

Soit p un élément de la surface d'un objet. On découpe l'hémisphère des directions sortantes centré sur la normale de la surface en plusieurs ensembles de direction (typiquement 4 ou 8).



Ensuite pour chaque ensemble de direction E, on recherche pour toutes les directions $\vec{d}$ appartenant à E l'angle minimale à partir duquel la source lumineuse peut être visible depuis p dans la direction $\vec{d}$; on intègre ensuite sur E le résultat obtenu.

On obtient ainsi une table (horizon map), qui est un table à 3 paramètres (p_x, p_y, E) et qui

contient l'angle minimale à partir duquel on peut voir la source lumineuse depuis *p* dans une des directions appartenant à E. Cet angle est forcément positif, la lumière ne pouvant provenir de dessous le sol.

Utilisation

Au moment du rendu, pour chaque élément de la surface, en fonction de sa direction par rapport à la source lumineuse, on va piocher l'angle minimal dans l'horizon map ; on compare celui-ci avec l'angle effectif de la source lumineuse. Si celui-ci est plus grand, alors l'élément de surface est éclairé, sinon il est ombré. En général, on interpole les résultats entre des ensembles de directions voisines, afin que les ombres glissent « doucement » lorsque la position de la lumière change.

Évaluation

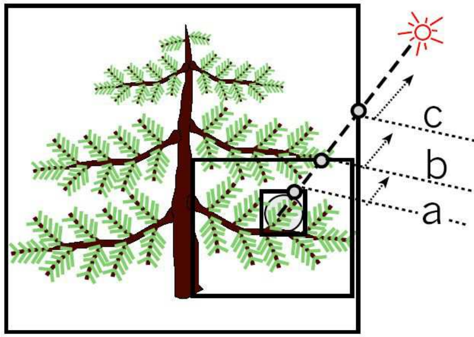
type d'ombres	ombres douces (flou autour du contour) : on ne peut l'utiliser donc que pour des sources lumineuses lointaines (typiquement le soleil), ce qui nous intéresse ici.
adaptée à la transparence	non
précision du résultat	Les zones d'ombres sont approximatives.
type de données	quelconque
forme du récepteur	quelconque
gestion de l'auto-ombrage	oui
vitesse	correcte
implémentation	hardware
coût CPU/mémoire	coût mémoire d'une texture, car il faut stocker la carte – généralement très acceptable.
espace	image

Les visibility cube maps

Présentation de la technique

Cette technique[MFP01] a été développée dans le contexte du rendu temps réel d'arbres avec ombrage dynamique.

Son principe est basé sur les cubes maps - une cube map est un ensemble de six textures centrées sur un repère. Il est donc possible de stocker dans une cube map l'environnement entourant un point. Pour chaque direction, on précalcule la quantité de lumière reçue par ce point provenant de cette direction, et on stocke cette information dans le pixel de texture correspond dans la cube map, sous la forme d'un réel compris entre 0 (absence de lumière) et 1 (pleine lumière).



Dessin 1 - L'ombrage de la branche est la composition successive des ombrages des cube map a, b et c (extrait de [MFP01])

L'idée consiste à créer une hiérarchie de telles cubes maps ; celles-ci ne comportent alors pas la quantité de lumière reçue pour l'ensemble de la scène, mais juste la contribution du niveau supérieur à chacune. On peut alors obtenir la quantité de lumière en un point en combinant l'ensemble des informations des niveaux supérieurs, lorsqu'on a besoin d'une cube map de visibilité à un point non échantillonné, une interpolation entre les cubes maps environnantes est réalisée.

Si les objets de la scène sont identiques, on peut alors en plaçant judicieusement la hiérarchie de cube maps faire l'économie d'un grand nombre de cube maps – en effet, seule celle englobant la totalité de chaque objet, et qui contient donc des informations globales reflétant l'influence de la scène sur les objets, nécessite d'être stockée pour chaque objet.

Évaluation

type d'ombres	ombres douces
adaptée à la transparence	oui
précision du résultat	les zones d'ombres sont approximatives.
type de données	quelconque
forme du récepteur	quelconque
gestion de l'auto-ombrage	non
vitesse	correcte
implémentation	hardware
coût CPU/mémoire	coût mémoire important, car il faut toutes les hiérarchies de cube maps.
espace	image

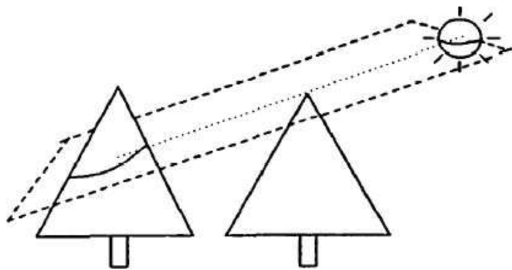
Remarque

Afin de minimiser le nombre de cube maps, il faut réduire le nombre d'objets différents contenues dans la scène, ce qui n'est pas toujours possible.

Méthodes empiriques de Reeves

Présentation

Toutefois, Reeves et Blau ont utilisé une technique empirique intéressante dans le cas du rendu d'arbres par particules[Reev85] : remarquant que les branches ou les feuilles sont très éclairées lorsqu'elles sont directement exposées au soleil, et que cette condition a plus de chance de se réaliser à la périphérie de l'arbre, ils ont diminué exponentiellement la luminosité des particules en fonction de leur distance par rapport à la source lumineuse, et par rapport à leur hauteur du sol. L'auto-ombrage est ainsi simulé.



Dessin 2 - Plan au dessus duquel les particules des arbres sont pleinement éclairées

Illustration 7 - "André's forest" (1985), résultats des heuristiques mises en oeuvre par Reeves et Blau

Pour simuler l'ombre des arbres entre eux, la technique suivante est utilisée : pour un arbre particulier, la hauteur et la position de ses voisins, ainsi que la position de la lumière, définissent un plan qui frôle la cime des arbres les plus proches et qui passe par la source lumineuse. Les particules de l'arbre situées au dessus de ce plan sont pleinement éclairées, tandis que les particules situées en dessous d'une certaine distance du plan sont à l'ombre. Entre, les particules éclairées sont choisies au hasard.

Évaluation

Il est difficile d'évaluer une telle méthode avec la grille précédemment utilisée ; toutefois, on peut noter que ces méthodes empiriques peuvent être implémentée dans le GPU, mais qu'elles nécessitent de pouvoir différencier les arbres entre eux.

1.3 Conclusion au sujet de l'ombrage

Malheureusement, aucune des techniques précitées ne peut répondre exactement à notre problématique : certaines ne sont pas adaptées à la façon dont sont stockées nos données soit parce qu'elles nécessitent de la géométrie(Shadow volume), soit parce qu'elles ont besoin de la possibilité de différencier les arbres entre eux (visibility cube maps, méthodes heuristiques de Reeves). D'autres ne permettent pas de reproduire l'ombrage à cause de la géométrie de notre scène (projection planaire) ou du fait qu'il y ait énormément de bloqueurs (projection sur des surfaces courbes).

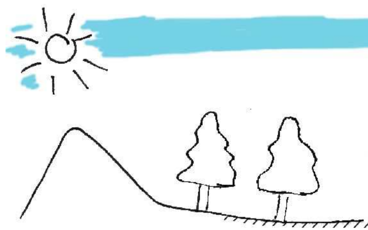
Néanmoins, les shadow maps et les horizons maps apparaissent comme les solutions aux meilleurs compromis, car elles ne sont pas dépendantes du types de données, ni de la forme du terrain. En plus, elles sont indépendantes de la complexité des données, ce qui est très important ici, et donc se font en temps relativement constant. Les premières semblent très prometteuses, car elles sont beaucoup plus précises que les horizons maps d'une part, et d'autres part ces dernières ne peuvent gérer l'ombrage de données volumiques.

Nous implémenterons au chapitre 5 ces méthodes, en essayant de palier leurs inconvénients (notamment en proposant une extension de la méthode des horizon maps, compatible avec des données volumiques), et analyserons alors les résultats obtenus.

2 Illumination de la forêt : présentation du problème et état de l'art

Un modèle d'illumination est une méthode informatique permettant de calculer l'énergie émise en un point d'une scène. Cette énergie est fonction de l'énergie arrivant sur ce point, et des matériaux constituants ce point ; de plus, elle n'est pas émise de la même façon suivant toutes les longueurs d'ondes, ce qui visuellement se traduit par l'émission d'une couleur et d'une intensité dans une certaine direction de l'espace.

On peut décomposer les modèles d'illumination en 2 catégories : les modèles locaux, qui sont des approximations ne prenant pas en compte les objets environnant celui considéré, et les modèles globaux qui déterminent le jeu des échanges d'énergies entre les différents constituants de la scène.



Dans le cas qui nous intéresse, nous simplifions l'éclairage, en prenant en compte uniquement de l'effet du soleil, source située à l'infini, et éventuellement quand cela est possible, de l'effet du ciel, source lumineuse étendue : quand le soleil est bas dans le ciel, c'est ce dernier qui tient un rôle prédominant dans l'éclairage.

Comme le souligne [JACQ2001], les modèles d'illumination de la canopée présupposent que les feuilles ont un comportement Lambertien, c'est-à-dire qu'elles se comportent comme des objets parfaitement lisses, ce qui n'est pas le cas – des recherches sont d'ailleurs encore actuellement menées pour essayer de déterminer les liens entre la fonction d'illumination et la physiologie d'une feuille.

Évaluation des techniques envisagées

Afin d'évaluer les techniques d'illumination, nous nous baserons sur les paramètres suivants :

- **type d'illumination :** locale ou globale.
- **Réalisme du résultat.**
- **précalcul :** la technique nécessite le calcul de données à l'initialisation du programme, données qui seront alors stockées et utilisées lors du rendu, évitant par la-même de reproduire ces calculs, et permettant donc de gagner du temps – par contre, ces données prendront de la place en mémoire...
- **vitesse :** toute opération a un coup en temps. On essaye ici de déterminer la vitesse d'un algorithme, c'est à dire le temps qu'il lui faut pour se réaliser sur tous les pixels d'une image.
- **implémentation software/hardware :** l'algorithme est bien adapté aux propriétés « cablées » des cartes graphiques (cas hardware) ou bien doit être écrit sous forme de programme (cas software).
- **coût CPU/mémoire :** on basera notre analyse sur un motif de texcells d'une taille de 256x256 pixels, le motif comprenant 32 couches de textures, et sans mip-mapping (avec celui-ci, il faut multiplier les résultats par 4) : Le mipmapping est le fait d'afficher des textures de taille différente selon la distance à laquelle on la regarde. De près on utilise une grande texture détaillée et de loin une petite avec très peu de détails, afin de limiter l'usage de la mémoire de texture de la carte vidéo.

On recherchera de manière préférentielle les techniques applicables en temps réel et apportant le meilleur réalisme, et ce à moindre coût (c'est pour cela qu'on ne parlera pas de solutions à base de ray-tracing ou de transfert d'énergie).

Le problème du calcul des fonctions d'illumination pour les objets lointains

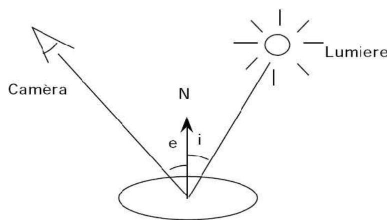
Lorsque l'on regarde un objet de près, chacun de ces éléments ont une fonction d'illumination ; lorsque l'objet s'éloigne, tout se passe pour l'observateur comme-ci ces fonctions d'illumination fusionnent pour en donner une nouvelle.

Dans le cas du rendu par ordinateur, plusieurs éléments géométriques viennent se projeter dans chaque pixel, et chacun apporte sa contribution à celui-ci ; ainsi l'illumination d'un pixel est le résultat de la composition des fonctions d'illumination des éléments se projetant dans celui-ci.

2.1 Présentation de techniques envisagées pour l'illumination des arbres

Textures de fonctions d'illumination

Présentation de la technique



Comme rappelé en préambule, l'illumination d'une surface est fonction de plusieurs paramètres. Une simplification consiste à ne tenir compte que de l'angle de vue de l'objet et de l'angle que fait sa normale par rapport à la lumière. On obtient ainsi une fonction de 2 paramètres, fonction qui n'est généralement pas directement calculable.

Aussi, une des solutions afin d'avoir une illumination correcte consiste à stocker dans une table à deux entrées (une texture bidimensionnelle) cette fonction d'illumination, et ce pour chaque point de l'objet. Au moment du rendu, un calcul simple permet alors de retrouver la couleur en un point.

Évaluation

type d'illumination	locale
réalisme du résultat	total, des photos pouvant servir de brtf. Une limitation est la quantification des angles.
précalcul	non
vitesse	calcul + accès mémoire (faible)
implémentation software/hardware	hardware
vitesse	bonne
<u>estimation du coût CPU/mémoire</u> Pour une fonction d'illumination de 32x32 angles, cela donne : • poids d'une fonction d'illumination : 32*32*4 (valeur stockée sur 4 octets RGBA) = 4096 octets • poids pour le motif des texcells : 268435456 octets = **268 méga-octets.** Le coût CPU est nul, le calcul et l'accès mémoire se faisant dans le GPU.	

Texture de normales pour la composante diffuse

Présentation de la technique

Une autre simplification consiste à dire que la couleur obtenue ne dépend que de la position de la source lumineuse par rapport à la normale de chaque point de l'objet.

On peut donc réaliser une table contenant, pour chaque point du texcell, sa normale, la couleur diffuse et ambiante en ce point[BLINN78]. Le stockage des normales se fait à travers une texture, en associant à chaque triplet (normé) de coordonnées (x,y,z) un triplet de composantes chromatiques (r_n, g_n, b_n), qui, si la texture ne peut contenir des valeurs signées, seront liées par la relation :

$$(r_n, g_n, b_n) = (\frac{x+1}{2}, \frac{y+1}{2}, \frac{z+1}{2})$$

Au moment du rendu d'un fragment, un produit scalaire suivi d'une multiplication sont effectués pour obtenir la contribution diffuse en ce point. Si (l_x, l_y, l_z) est le vecteur (normé) de la direction lumineuse par rapport au point, et si on pose $k = max(0, \begin{pmatrix} r_n*2-1 \\ g_n*2-1 \\ b_n*2-1 \end{pmatrix} \cdot \begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix})$, alors :

$$\begin{pmatrix} d_r \\ d_v \\ d_b \end{pmatrix} = k \times \begin{pmatrix} r \\ g \\ b \end{pmatrix}$$

et l'illumination en ce point s'obtient donc par :

$$\begin{pmatrix} R \\ V \\ B \end{pmatrix} = \begin{pmatrix} d_r \\ d_v \\ d_b \end{pmatrix} + \begin{pmatrix} a_r \\ a_v \\ a_b \end{pmatrix} ,$$

où (a_r, a_v, a_b) contient la couleur ambiante en ce fragment.

L'impression suggestive est celle d'une géométrie complexe là où il n'y a en fait qu'un simple polygone. Un inconvénient fort est qu'à grande distance, l'illumination n'est plus cohérente [FOURNIER92], car l'illumination d'un ensemble de points dont l'illumination est obtenue par ces calculs n'est pas le résultat d'un tel calcul!

Évaluation

type d'illumination	locale
réalisme du résultat	donne un aspect plastique aux choses (éclairage lambertien).
précalcul	non
vitesse	correcte
implémentation software/hardware	hardware
<u>estimation du coût CPU/mémoire</u> • par point : 3 octets (diffus) + 3 octets (normale) + 3 octets (ambiant) + 1 octet (alpha) = 10 octets • poids pour le motif des texcells :20971520 octets = **20 méga-octets** Le coût CPU est nul, les calculs se faisant dans le GPU.	

Méthode de Schlick : approximation par des fonctions rationnelles

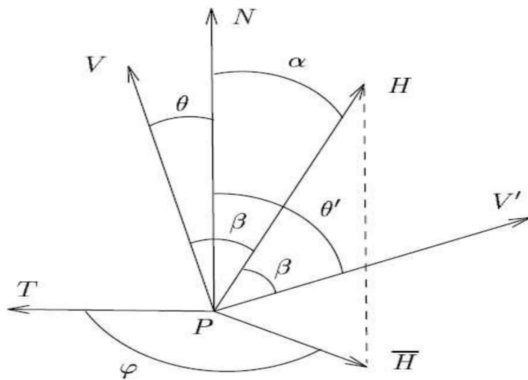
Présentation de la technique

Un des principes sur lequel s'appuie ce modèle[NOE99][SCHLICK94] est le suivant : on approxime les fonctions relativement complexes apparaissant dans les modèles d'éclairement par des fractions rationnelles. L'approximation des fonctions est obtenue à partir de conditions pouvant être n'importe quelle caractéristique intrinsèque de la fonction : valeur en un point donné de la fonction ou d'une de ses dérivées, équation intégrale ou différentielle à laquelle la fonction doit satisfaire, ...

Pour définir son modèle, Schlick définit deux types de matériaux :

- les matériaux simples, ayant des propriétés optiques homogènes (métal, verre, papier, etc.),
- les matériaux doubles, ayant des propriétés optiques hétérogènes (plastique, peau, surface peinte ou vernie, etc.) et qui sont constitués d'une couche plus ou moins translucide reposant sur une couche opaque, chaque couche correspondant à un matériau simple.

Les notations sont les suivantes :



V: direction de la lumière réflechie
V': direction de la lumière incidente
N: normale à la surfaces
T: tangente à la surfaces
H: vecteur support de la bissectrice de V et V'
$\bar{H}$: projection de H orthogonalement à N

$t=H.N\,;u=H.V\,;v=(V.N)\,;v'=V'.N\,;w=T.\bar{H}$

Schlick propose, comme BRDF(*Bidirectional Reflectance Distribution Function)*, c'est-à-dire comme fonction indiquant comment un matériau réfléchie la lumière, les équations suivantes :

- pour un matériau simple :

$$f_{schlick}(t,u,v,v',w)=S_{\lambda}(u)D(t,v,v',w)$$

- pour un matériau double :

$$f_{schlick}(t,u,v,v',w)=S_{\lambda}(u)D(t,v,v',w)+[1-S_{\lambda}(u)]S'_{\lambda}(u)D'(t,v,v',w)$$

où S_{λ} est un terme spectral et D un terme directionnel.

Pour S_{λ} , Schlick propose :

- soit de prendre une valeur constante C_{λ} ,
- soit d'utiliser une approximation rationnelle du terme de Fresnel : $S_{\lambda}(u)=C_{\lambda}+(1-C_{\lambda})(1-u)^5$

Pour le terme directionnel, Schlick propose la formule :

$$D(t,v,v',w)=\frac{a}{\pi}+\frac{b}{4\pi v\,v'}B(t,v,v',w)+\frac{c}{v\,dV'}\Delta \quad \text{avec a + b + c = 1.}$$

Δ une fonction de Dirac valant 1 dans dV' (voisinage de la direction de V') et 0 ailleurs, et :

$$B(t,v,v',w)=\frac{G(v)G(v')}{4\pi v\, v'}Z(t)A(w)+\frac{1-G(v)G(v')}{4\pi v\, v'}$$

$G(v)=\frac{v}{r-rv+v}$ est une approximation du facteur d'atténuation géométrique de Smith (r est la rugosité de la surface, entre 0, qui signifie purement spéculaire et 1, qui signifie parfaitement diffus). On a aussi :

$$Z(t)=\frac{r}{1+rt^2-t^2} \text{ et } A(w)=\sqrt{\left(\frac{w}{p^2-p^2w^2+w^2}\right)}$$

Avec p le facteur d'anisotropie, entre 0, qui signifie une complète anisotropie, et 1 qui signifie une parfaite isotropie. Ce modèle est assez intéressant, par le fait qu'il a peu de paramètres (p, r, a, et b) et qu'il permet de représenter pas mal de comportements différents.

Toutefois, on peut lui reprocher son grand nombre de calculs. Comme les paramètres apparaissent dans chacune des fonctions, il est impossible de les précalculer. Cette BRDF a été proposée car elle était peu coûteuse à calculer ... par les CPUs. Toutefois, ces calculs s' adaptent mal aux GPUs.

Évaluation

type d'illumination	locale
réalisme du résultat	très bon
précalcul	non
vitesse	très lent
implémentation software/hardware	software
estimation du coût CPU/mémoire • par point : 4 octets (coefficients p, r, a et b) + 3 octets (la normale) = 7 octets • poids pour le motif des texcells, 14680064 octets = 14 **méga-octets** Le coût CPU est nul, les calculs se faisant dans le GPU.	

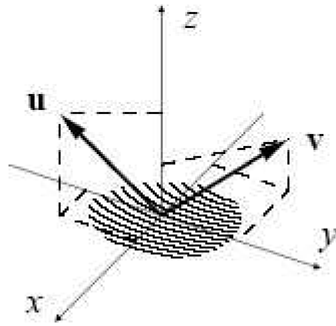
Approximation non linéaire de fonctions d'illuminations (Lafortune)

Présentation de la technique

l'idée consiste à approximer la BRDF par un modèle de lobes de cosinus[LAF97] :

$$\begin{pmatrix}R\\V\\B\end{pmatrix}=\begin{pmatrix}a_r\\a_v\\a_b\end{pmatrix}+\left(\sum_j\left[C_x^j u_x v_x+C_y^j u_y v_y+C_z^j u_z v_z\right]^{n_j}\right)*\begin{pmatrix}d_r\\d_v\\d_b\end{pmatrix}*max(0,u_z) \text{, où :}$$

- (a_r, a_v, a_b) contient la couleur ambiante de ce fragment
- (d_r, d_v, d_b) contient la couleur diffuse de ce fragment



- $\vec{u}$ est le vecteur (normé) pointant vers la source lumineuse
- $\vec{v}$ est le vecteur (normé) pointant vers l'observateur
- j est le nombre de lobes
- C_x^j, C_y^j, C_z^j, n_j contiennent l'albedo et la forme du lobe (ces coefficients sont fonction du matériau). Les C_a^j ont des valeurs comprises entre -1.5 et 1.5, quand aux n_j, ils varient entre 0 et 10000.

Par convention, toutes les puissances d'un terme négatif sont nulles.

Tous les vecteurs sont exprimés dans le repère centré sur le point à illuminer, et tel que l'axe z est confondu avec la normale, et les axes x et y sont les principales directions d'anisotropie. Les paramètres C_x^j et C_y^j règlent l'anisotropie, lorsqu'ils sont différents. De plus, lorsqu'ils sont négatifs, le lobe est plutôt un lobe dans la direction miroir, alors que, quand ils sont positifs, le lobe est du côté de la rétro-réflexion.

Ce modèle permet d'obtenir une large gamme de BRDFs, avec des résultats visuels satisfaisants et très proches de la réalité. Il est relativement simple et cette méthode a déjà été implémentée dans des programmes utilisant les cartes graphiques actuelles[MCALLISTER02].

Évaluation

type d'illumination	locale
réalisme du résultat	très bon
précalcul	non
vitesse	lent
implémentation software/hardware	software
estimation du coût CPU/mémoire • par point : 7 octets (diffus, ambiant, alpha) + (4 octets x j (coefficients) + 1)= 8 + 4j octets • poids pour le motif des texcells, avec j = 1 : 25165824 octets = **25 méga-octets** Le coût CPU est nul, les calculs se faisant dans le GPU.	

2.2 Conclusion au sujet de l'illumination de la forêt

Pour un effet de réalisme accru, nous ne pouvons nous contenter d'un simple produit scalaire entre un vecteur directeur pointant vers la source lumineuse et la normale en chaque point des texels, et pour des raisons de place mémoire, nous ne pouvons utiliser les textures d'illumination.

La méthode de Schlick, quoi que pouvant donner des illuminations complexes, est trop gourmande en calcul. La méthode de Lafortune cumule plusieurs avantages : elle est bien adaptée au GPU, et donne des fonctions d'illuminations complexes – nous avons donc décider d'implémenter cette méthode, comme nous le montrons au chapitre 6.

3 Ombrage des texcells

Notre étude préliminaire (réalisée au chapitre 10) nous a permi de nous intéresser aux méthodes des shadow maps et des horizons maps. Nous les avons implémentées, et présentons ici nos résultats et analyses.

5.1 Depth shadow map

Le rendu d'un ombrage en utilisant la méthode des depth shadow maps se fait en deux temps; nous devons tout d'abord rendre la scène vue depuis la lumière, et stocker la distance par rapport à celle-ci des bloqueurs, distance lue dans le z-buffer.

Sans intervention de notre part, les couches de textures semi-transparentes écrivent dans le z-buffer, quelle que soient la transparence des éléments traités. Or, si certaines éléments sont totalement transparents, cela signifie que l'espace qu'ils occupent est vide - il ne faut pas donc pas que ces éléments puissent écrire dans le z-buffer ; aussi nous activons l'alpha-test, qui permet de fixer le seuil alpha à partir duquel la carte ne traite plus les fragments.

Nous obtenons ainsi dans le z-buffer la distance aux objets contenues dans les texcells, et non plus la distance aux texcells. Nous avons utilisé un alpha-test dont le seuil est 0.3, afin de s'assurer que les ombres soient bien à l'intérieur des arbres dans les cas limite.

On pourrait penser qu'une depth shadow map serait suffisante pour reproduire complètement l'ombrage ; toutefois, si l'ombrage d'une image est très bon, cette technique souffre de son manque de précision, ce qui fait qu'un léger déplacement de la source lumineuse introduit un décalage de l'ombre sur les feuilles et on obtient alors un effet de scintillement très notable.

Illustration 8 - Scène sans ombrage

Illustration 9 - Scène avec depth shadow map

Nous avions pensé diminuer cet effet de scintillement en implémentant la méthode dite des « perspective shadow maps »[SD02], qui travaille dans l'espace ayant subi la projection de la caméra, ce qui est sensé diminuer les erreurs d'approximation.

Seulement, non seulement les erreurs étaient amplifiées sur les éléments lointains (ce qui est un des défauts connus de la méthode), mais cette fois-ci la scène « vibrait » à chaque mouvement de caméra... Nous trouvons le résultat obtenu par de simple depth shadow map plus convaincant.

Nous pouvons par contre remarquer que cette technique donne des résultats très intéressants dans le

cas du rendu d'images fixes – pour s'en convaincre, on peut comparer les images obtenues avec la prise de vue ci-dessous :

Illustration 10 - Photographie d'une montagne ensoleillé (La Tronche, près de Grenoble)

Nous proposons ici une méthode, adaptée au rendu de texcells, qui doit permettre d'une part de combler cet handicap et d'autre part de réaliser efficacement l'ombrage de la forêt.

3.1 Double horizon-maps

Principe

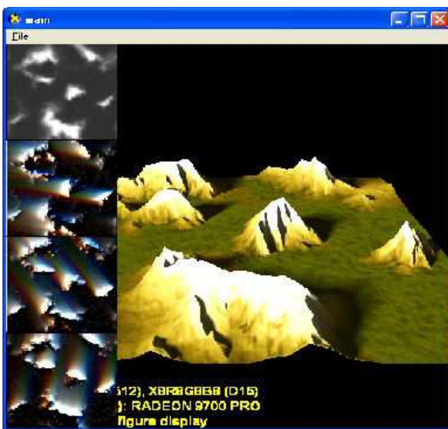


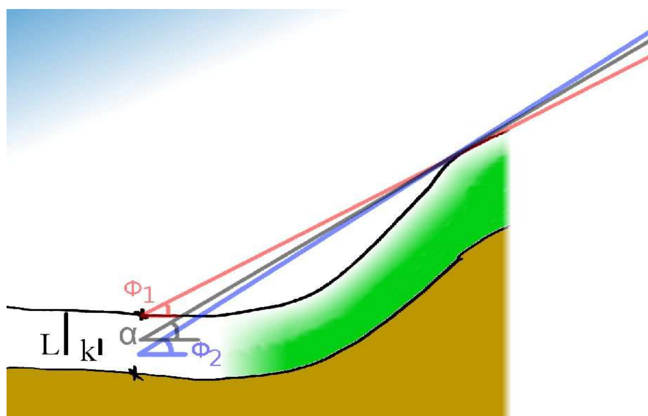
Illustration 11 - Ombrage obtenue par horizon map

Comme expliqué au paragraphe 3.2 (page 17), une horizon map est une carte qui comporte pour chaque point l'azimut minimal à partir duquel, dans une direction donnée, la source lumineuse est visible. Cette carte, précalculée, est ensuite analysée en temps réel au moment du rendu.

Cette technique donne des résultats remarquables pour les terrains : en effet, la principale source lumineuse considérée est alors le soleil.

Malheureusement, nous travaillons sur un terrain recouvert d'une forêt – nous devons donc adapter les horizons maps aux données volumiques.

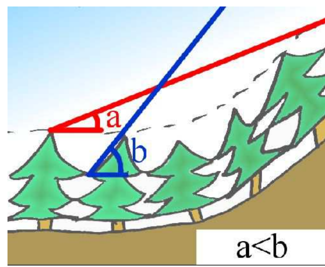
On pourrait associer à chaque couche de textures contenues dans les texcells une horizon map ; toutefois, il convient de stocker une très grande quantité de données, ce qui n'est pas justifié par égard au réalisme apporté à la scène.



Dessin 3 - l'angle α, située sur une couche intermédiaire, est obtenue par la formule : $\alpha=(\frac{k}{L})*\phi_1+(1-\frac{k}{L})*\phi_2$

Notre solution a donc été d'introduire une interpolation entre deux horizon maps : une calculée au niveau des cimes des arbres, et une calculée à une niveau intermédiaire entre la canopée et le sol.

l'interpolation se fera sur les azimuts minimaux : ainsi, on obtient de manière plus précise les frontières lumière/ombres ; cela peut aussi s'interpréter comme si la carte des angles d'une couche de texture intermédiaire était une carte intermédiaire entre les cartes des couches extrêmes.

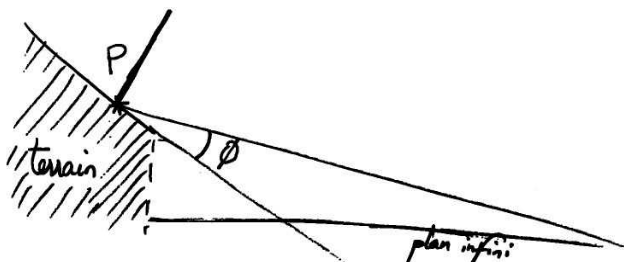


Dessin 4 - Les azimuts obtenus sont plus grands au milieu des branches

Les azimuts minimaux obtenus seront bien sûr plus grands au milieu des branches, compte tenu de la plus faible visibilité du ciel (d'ailleurs, une horizon map calculée depuis le sol donne une carte avec des angles maximaux (90°), ce qui se traduira par une ombre constante quelle que soit la position de la source lumineuse).

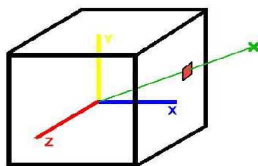
Génération des cartes

Les cartes sont générées par le même afficheur que celui effectuant le rendu en temps réel. Au démarrage de celui-ci, les textures sont chargées en mémoire, et noircies afin de n'obtenir à l'image que des zones blanches ou noires.



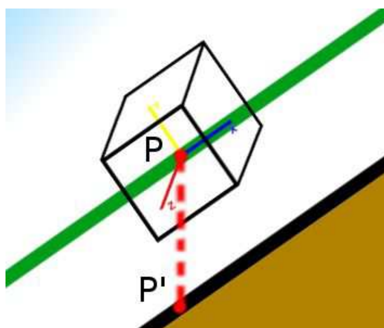
Dessin 5 - Sans le plan infini, l'angle vaut zéro et la zone peut être illuminée si la lumière est située sous le terrain...

De même, on ajoute à la scène un plan infini noir, afin de simuler le fait que le terrain est infini : sans ce plan, comme on ne possède qu'un extrait de terrain, les bords de celui-ci sont longés par des gouffres énormes, ce qui modifie la valeur des azimuts minimaux.


Dessin 6 - Contrairement à l'adressage classique de textures à base de coordonnées u et v, une cube map est adressée grâce à un vecteur.

Nous allons calculer les angles limites en analysant des cube maps ; une cube map est un ensemble de six textures centrées sur un repère. Il est donc possible de stocker dans une cube map l'environnement entourant un point.

L'intérêt d'une cube map est qu'elle associe à chaque direction un pixel de textures – on peut donc stocker une information par direction.



Dessin 7 - la cube map est orientée parallèlement au sol

Nous allons échantillonner le terrain en posant une grille régulière dessus ; chaque point P' de cette grille est la projection, suivant la verticale, sur le terrain du centre P des cubes-maps qui vont être calculées.

La cube map est tournée de telle façon que la face supérieure soit contenue dans le plan orthogonal à la normale N au sol au point P ; en effet, l' illumination n'est linéaire que dans H(P), l'hémisphère des directions sortantes centré sur la normale de la surface - en dehors, la contribution de la lumière est nulle : la lumière ne pouvant venir de par le sol, on peut donc éliminer la face inférieure de la cube map, qui n'en contient donc plus que 5.

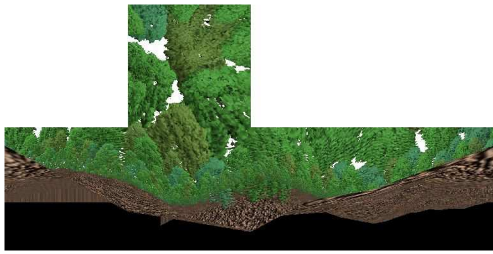


Illustration 12 - Une cube map (couleur) obtenue en un point de la grille...

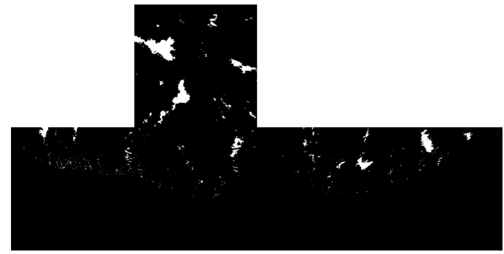
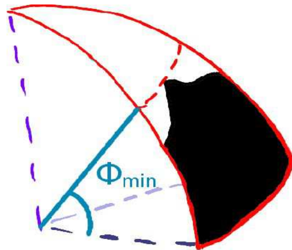


Illustration 13 - ... et la même en noir et blanc, qui servira au précalcul de l'horizon-map



Dessin 8 - recherche de l'angle minimale

On divise ensuite H(p) en un certain nombre X de secteurs identiques (dans notre implémentation, $X=4$). On parcourt alors ensuite chaque secteur S à la recherche de l'azimut minimal, c'est à dire l'angle $\phi_{min}(S)$ à partir duquel toutes les directions situées au dessus renvoient une couleur blanche dans la cube map.

On stocke alors dans un tableau pour chaque point P, le vecteur V contenant l'ensemble des azimuts minimaux : c'est l'horizon map.

Phase de rendu

Une fois cette phase de précalcul effectuée, nous pouvons alors calculer chacune des images. Pour ce faire, nous devons, afin de maximiser les performances, programmer la carte graphique pour qu'elle réalise nos opérations. Nous avons choisi pour la programmer d'utiliser GLSlang, qui est le langage de programmation de carte graphique haut niveau de l'API graphique OpenGL ; il permet de spécifier les opérations que la carte doit réaliser pour chaque pixel qu'elle doit afficher ou chaque vertex qui lui est envoyé.

Nous avons réalisé un programme en utilisant ce langage ; ce programme se décompose en deux parties : le vertex program, et le fragment program, qui chacun ont leur importance dans le rendu des ombres par horizon maps.

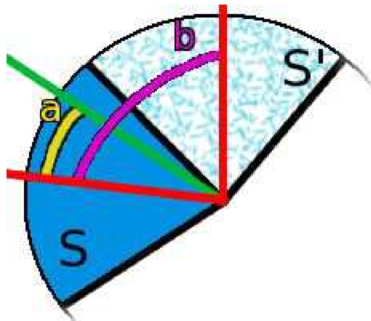
Le vertex program

Pour chaque vertex envoyé à la carte, ce programme va calculer, dans l'espace des directions sortantes, la direction de la lumière. Connaissant cette direction, on peut en déduire l'azimut de la lumière par rapport au vertex. Ces données (le vecteur directeur et l'angle) sont envoyés en paramètre au fragment program.

Le fragment program

C'est ici que se passe le plus gros du travail : l'analyse du vecteur directeur $\vec{l}$ va donner le secteur S dans lequel se situe la source lumineuse. On ne s'occupe pour le moment que d'une seule horizon map, notée *x* : on va alors lire dans sa table précalculée l'azimut minimal $\phi^{x}_{min}(S)$ de ce secteur.

Si on pose $\Gamma(\vec{l})=\phi^{x}_{min}(S)$, comme nous n'avons découpé H(P) qu'en X secteurs différents, nous n'aurons que X ombres différentes, à azimut de la source lumineuse constant. Pour palier cet inconvénient, nous allons interpoler la valeur des azimuts des secteurs voisins de S : on obtiendra alors ainsi une valeur de $\Gamma(\vec{l})$ qui est plus proche de la valeur qu'il devrait avoir.



Dessin 9 - Groupe des directions et direction de la source lumineuse

Pour ce faire, on va supposer que pour tout secteur, ϕ_{min}^{x} est la valeur correcte pour la *direction principale* du secteur angulaire (qui est la bissectrice de ce secteur, et est représentée par les demi-droites rouges sur le schéma ci-contre, la demi-droite verte représentant la direction de la source lumineuse).

On obtient donc que :

$$\Gamma_x(\vec{l})=(\frac{a}{b})*\phi_{min}^{x}(S')+(1-\frac{a}{b})*\phi_{min}^{x}(S)$$

Ainsi, les transitions entre secteurs se font de manière plus harmonieuse, ce qui renforce la crédibilité de l'ombrage. Nous devons toutefois encore réaliser une interpolation entre les angles $\Gamma_1(\vec{l})$, obtenu grâce à la carte précalculée au niveau de la canope, et $\Gamma_2(\vec{l})$, précalculée à un niveau intermédiaire dans la forêt. Finalement, comme rappelé en introduction,

$$\Gamma(\vec{l})=(\frac{k}{L})*\Gamma_1(\vec{l})+(1-\frac{k}{L})*\Gamma_2(\vec{l})$$

On compare ensuite l'azimut nouvellement obtenu, $\Gamma(\vec{l})$, avec l'azimut ϕ de la lumière.

- Si celui-ci est plus petit que $\Gamma(\vec{l})$, alors le fragment est dans l'ombre : on l'assombrit, en mettant son coefficient d'illumination C à 0.
- Si celui-ci est plus grand, alors le fragment est éclairé : on l'éclaire, C vaut 1.

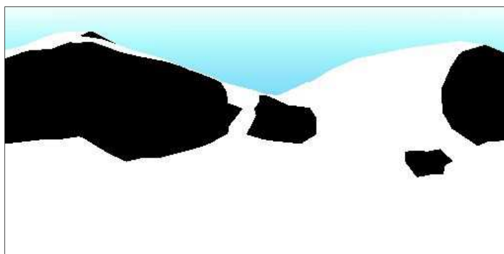
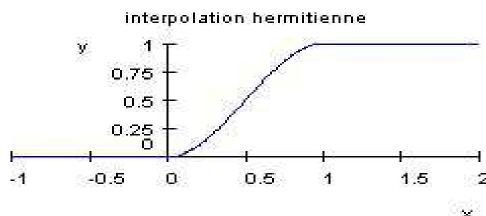


Illustration 14 - Ombres dures obtenue par horizon map

Si on procède de la sorte, on obtient alors des ombres dures, aux frontières marquées.

Afin de simuler une source étendue, et donc obtenir des ombres douces, on se donne un réel positif ε tel que :

- si $\phi < \Gamma(\vec{l})-\epsilon$, C=0 : le fragment est totalement dans l'ombre.
- si $\phi > \Gamma(\vec{l})+\epsilon$, C=1 : le fragment est totalement dans la lumière.
- Si $\phi\in[\Gamma(\vec{l})-\epsilon;\Gamma(\vec{l})+\epsilon]$, alors on réalise une interpolation hermitienne entre les valeurs 0 et 1 ; cela est équivalent à :



$$t=\frac{(\phi-\Gamma(\vec{l}))}{(2\epsilon)};C=t*t*(3-2*t)$$

Cette opération est rapidement réalisée par les cartes graphiques actuelles.

On obtient alors une transition douce entre les zones à l'ombre et les zones illuminées.

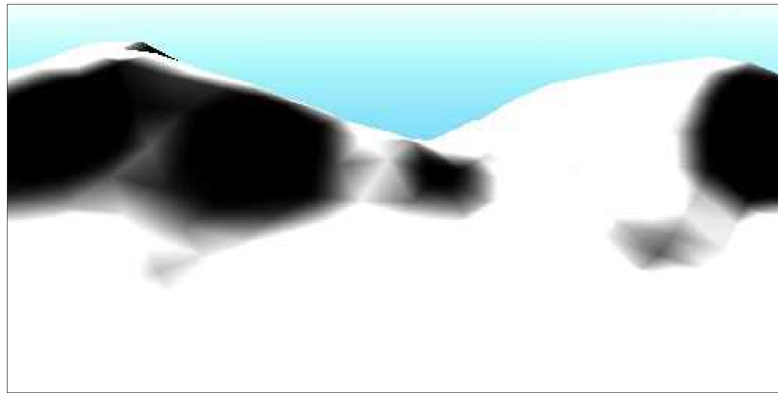


Illustration 15- Ombres douces obtenue par horizon map

Enfin finalement, nous multiplions C par la valeur obtenue en effectuant un le calcul de l'illumination; on peut voir si dessous le résultat si l'éclairage est un éclairage de gouraud classique.

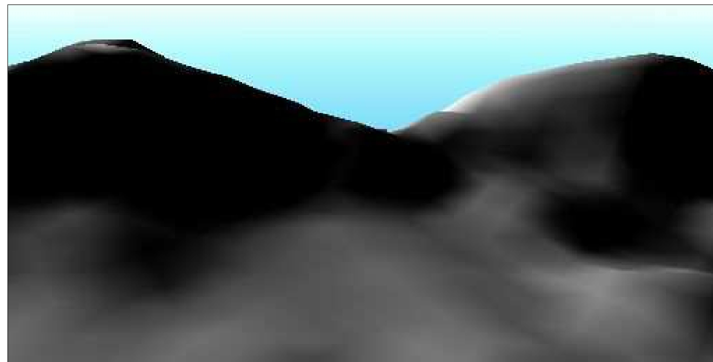


Illustration 16 - Résultat final d'un ombrage mélangeant gourand et horizon map

Notre implémentation

Nous avons décidé que, pour notre implémentation, X vaudrait 4, car on peut stocker au maximum 4 composants par texture. Nous avons donc réalisé une carte des azimuts que nous avons sauvé sous la forme d'un fichier image, les composantes à valeur forte représentant des angles forts :

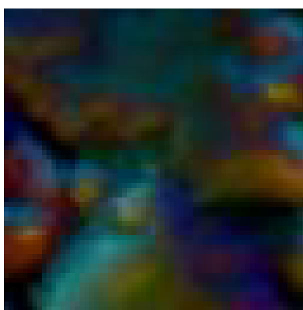
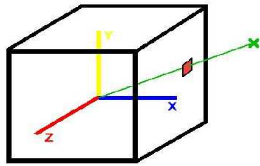


Illustration 17 - Horizon map de taille 64x64 ; chaque composante représente une direction.

Illustration 18 - Table des angles dans la quatrième direction

Nous avons également essayé de minimiser les calculs réalisés dans le fragment program, compte tenu du grand nombre de pixels traités. Aussi, nous proposons une méthode qui fait que l'analyse du vecteur directeur et la transition douce entre les secteurs, autrement dit le calcul de $\Gamma_x(\vec{l})$, se réalisent en 2 opérations très simples.



Dessin 10 - Contrairement à l'adressage classique de textures à base de coordonnées u et v une cube map est adressée grâce à un vecteur.

Tout d'abord, il faut savoir que les cartes graphiques actuelles permettent d'accéder à des textures de cube maps, c'est à dire que si l'on donne un vecteur à la carte graphique, celle-ci va nous retourner un vecteur chromatique qui correspond à la valeur du pixel sur le cube de texture pointé par le vecteur directeur.

Ainsi, on peut construire une cube map, qui contient dans ses textures le nom du secteur qui correspond à chaque direction. En donnant en entrée le vecteur directeur $\vec{l}$, on obtiendrait en retour le secteur dans lequel lire la valeur de l'azimut.

Mais il y a mieux : comme le vecteur chromatique renvoyé contient 4 composantes, et qu'il y a 4 secteurs, on peut stocker dans chacune de celles-ci la contribution de chacun de ces secteurs.

Ainsi, supposons par exemple que la source lumineuse soit dans la direction principale du premier secteur ; en regardant dans la cube map, on obtiendrait (1,0,0,0) ; de même si la source est dans la direction pricipale du second secteur, on aurait (0,1,0,0), et si elle est à mi-angle, (0.5,0.5,0,0).

Il suffit donc alors de faire un produit scalaire entre la valeur lue dans la cube map, et le vecteur des valeurs lues pour obtenir l'azimut limite.

Analyse

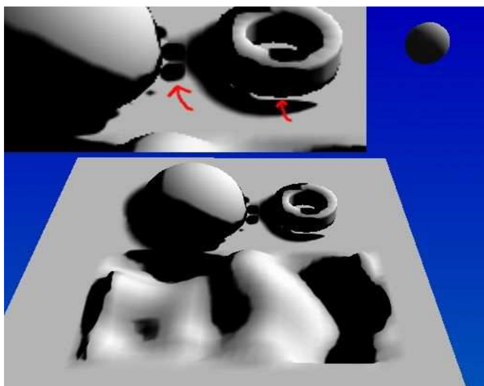


Illustration 19 - Les problèmes rencontrées

Cette technique n'est pas parfaite : en effet, les transitions entre les secteurs apparaissent parfois comme « artificielles », et des problèmes se posent pour les objets à angles très vifs, comme le montre l'illustration ci-contre (la source lumineuse est représentée par la boule noire située en haut à droite de l'image, et les ombres crées grâce aux horizon maps ont été amplifiées).

En effet, des taches d'ombres et des trous de lumière apparaissent au niveau de ces angles vifs. Une des solutions consiste à augmenter la résolution de la carte des angles, ce qui va diminuer la surface des ces artefacts.

On pourra toutefois remarquer que sur les terrains, les changements d'angles se font graduellement, et que sur le terrain situé en bas de l'image, les ombres sont correctes.

Nous pouvons donc s'attendre que cette technique donnera des bons résultats sur les terrains.

3.2 Auto-ombrage empirique

Principe

Nous pouvons observer que les sous-bois sont beaucoup plus ombrés que la cime, et celle-ci est très éclairée. En assombrissant les couches les plus proches du sol, et en éclaircissant la cime par rapport au milieu des arbres, nous ré-introduisons cette propriété.

Connaissant la position de la couche par rapport au sol, une interpolation linéaire permet alors d'ombrer ou d'illuminer celle-ci. Cette ombre n'est, toutefois, absolument pas dépendante de la position de la lumière.

Phase de rendu

Illustration 20 - Sans l'ombrage empirique

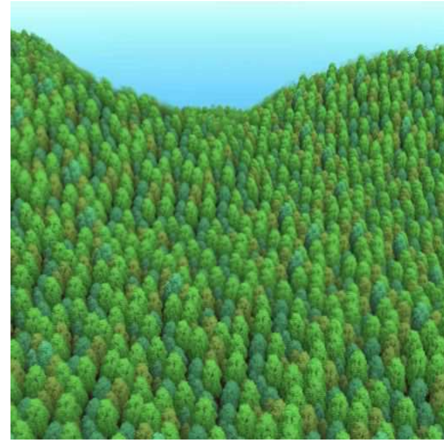


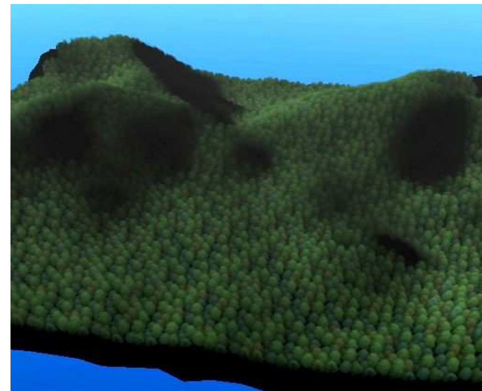
Illustration 21 - Avec l'ombrage empirique

Le coefficient d'illumination, C, vaut 0.5 en bas des arbres, et 1,5 en haut de ceux-ci; entre, C est fonction de la hauteur h suivant la loi : $C(h)=0.5+h$. On multiplie ensuite C par la couleur du motif.

On pourrait essayer d'adapter les double horizon maps afin d'imiter ce genre de comportement, en portant la borne supérieur de C, le coefficient d'illumination, de 1 à 1.5 par exemple, et en utilisant sur la couche la plus basse une horizon map avec des angles minimaux à 90° - ce qui a pour effet de toujours assombrir les couches les plus basses de la forêt - toutefois, le calcul d'une horizon map est quelque chose de beaucoup plus coûteux qu'une simple interpolation linéaire!

5.4 Résultat final

En cumulant les deux techniques précédemment évoqués, nous obtenons un ombrage du terrain sur les arbres très bon, ainsi qu'un ombrage des arbres entre eux plutôt cohérent. La méthode est, contrairement au shadow map, robuste au changement de position de la source lumineuse, et le fait que l'ombrage empirique ne soit pas fonction de la lumière ne gène pas.



On pourra remarquer toutefois plusieurs points négatifs de cette technique:

- une horizon map de très grande résolution (au moins 512x512) est nécessaire afin d'obtenir des ombres de qualités comparable à celles obtenues par la shadow map.
- il existe un certain nombre de facteurs à régler afin d'obtenir une image correcte ; en effet, il faut régler l'influence de chacune des sortes d'ombrages entre elles, ce qui donne beaucoup de coefficients à régler empiriquement : la force de l'ombre des horizons maps et de celle de l'ombrage heuristique, ainsi que l'importance de l'ombrage de Gouraud. On pourra voir dans ses paramètres l'influence des lumières émises par le ciel et le soleil – la quantité de lumière fournie par chacune ainsi que le rapport de ces quantités jouant sur les forces de chacun des types d'ombre.

Le rendu, effectué sur un pentium IV a 2 Ghz équipé d'une carte graphique GeForce FX 5800 s'effectue au rythme d'environ 10 images par seconde, pour des scènes rendu avec une résolution de 512x512 pixels. Les goulots d'étranglement des performances sont le grand nombre de pixels à traiter pour afficher une image, et surtout le fait que le GLSlang n'est pas complètement supporté par la carte sur laquelle nous faisons nos tests : au dessus d'une certaine taille de programme, la charge du CPU monte anormalement, ce qui prouve que tous les calculs ne sont pas réalisés par le GPU – le transfert des données entre le CPU et le GPU étant généralement très lent. Nous pensons que l'amélioration du GLSlang (qui est toujours en développement) ainsi que les possibilités offertes par les futures générations de cartes graphiques pourront résoudre ce problème et permettre d'atteindre des fréquences élevées.

6 Illumination des texcells

Comme nous l'avons vu lors du rappel de l'état de l'art, il est nécessaire d'avoir une fonction d'illumination plus complexe que la fonction lambertienne, sous peine d'avoir une forêt aux aspects « plastiques ». Nous proposons ici une implémentation des lobes de Lafortune, adaptée au rendu de forêts et aux cartes graphiques modernes.

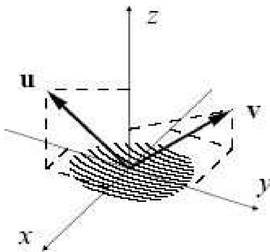
6.1 *Hypothèses préliminaires*

l'illumination de Lafortune a déjà été adaptée aux cartes graphiques modernes par McAllister et al. [MCALLISTER02] ; toutefois, cette implémentation n'est pas adaptée à nos besoins. En effet, elle s'effectue en plusieurs passes (une par matériau différent) d'une part, et d'autre part, elle est, ici, inutilement trop précise.

En effet, la répartition des orientations des feuilles est globalement homogène, et celles-ci ont une forte fréquence spatiale. Ces propriétés ont pour conséquence d'atténuer l'influence des reflets des feuilles. Dans ces conditions, un seul lobe est suffisant dans le calcul de la fonction d'illumination – l'implémentation de McAllister utilise entre 1 et 2 lobes, suivant les matériaux des scènes, ce qui compte tenu du grand nombre de fragments que nous avons à calculer, est trop cher.

Afin de simplifier les calculs et le modèle nous avons introduit les simplifications suivantes :

- La fonction d'illumination des feuilles est toujours isotrope : en effet, même si elle ne l'est pas localement, le fait que leur orientation soit quelconque moyenner l'anisotropie et donc rend l'illumination globalement isotrope. Cela introduit les simplifications suivantes :



- les directions *x* et *y* peuvent être quelconques (du moment qu'elle forme un repère orthogonale avec l'axe *z*...)
- Pour chaque lobe *j*, $C_x^j = C_y^j = C_{xy}^j$

- Comme dit précédemment, nous n'utiliserons qu'un seul lobe par fonction d'illumination : j=1.

Pour chaque fragment, le fonction d'illumination s'obtient donc par :

$$\begin{pmatrix} R \\ V \\ B \end{pmatrix} = \begin{pmatrix} a_r \\ a_v \\ a_b \end{pmatrix} + \left(\left[C_{xy}(u_x v_x + u_y v_y) + C_z u_z v_z \right]^n \right) * \begin{pmatrix} d_r \\ d_v \\ d_b \end{pmatrix} * max(u_z, 0)$$

où :

- $\vec{u}$ est le vecteur (normé) pointant vers la source lumineuse,

- $\vec{v}$ est le vecteur (normé) pointant vers l'observateur.

Ces vecteurs étant exprimés dans le repère centrée sur le point à illuminer, et tel que l'axe *z* est confondu avec la normale, et les axes *x* et *y* sont les principales directions d'anisotropie.

Afin de calculer l'illumination en un pixel de la forêt, nous devons obtenir tout d'abord son repère

locale, c'est un dire un repère tel que la normale soit colinéaire à l'axe z.

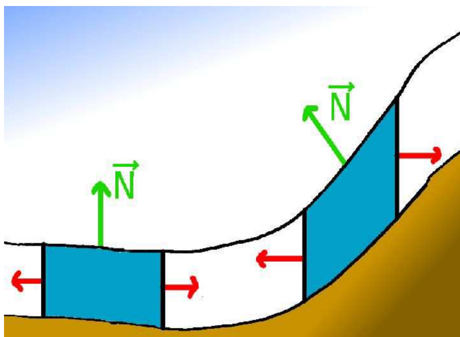
6.2 Obtention du repère local à chaque éléments des texcells

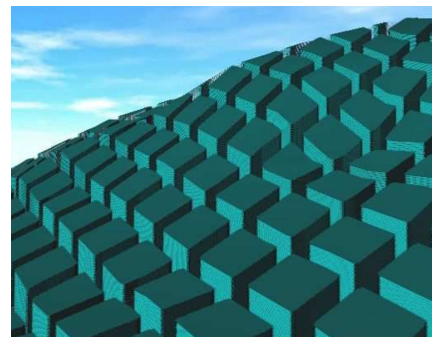
Pour trouver ce repère, nous procéderons en deux étapes : l'obtention de la matrice de passage du repère monde au repère local à chacune des faces, et l'obtention de la normale à chaque pixel de texture, exprimée dans ce denier repère.

Par la suite, nous considérerons que l'axe z du repère monde pointe vers le ciel, colinéairement au vecteur gravitation.

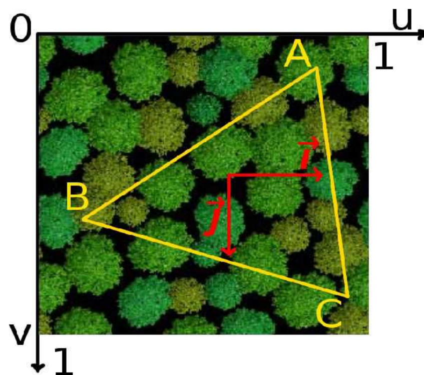
Repère monde → repère local à chacune des faces

Il faut, dans une première étape, déterminer le repère locale à la face contenant une couche de la texture ; toutefois, à cause de la manière avec laquelle sont construits les texcells, ce repère n'est pas orthogonal!





Le motif est ici constitué de boîtes : on peut voir sur l'image de droite que les faces orthogonales au sol restent parallèles entre elles.



Dessin 11 - Le repère des couches et celui de l'espace texture sont colinéaires.

Il est toutefois construit de tels manières que les vecteurs $\vec{i}$ et $\vec{j}$, supports des axes x et y du repère monde, soient parallèles aux axes du repère des textures.

On obtient un tel repère de la manière suivante :

- on recherche tout d'abord l'expression d'un vecteur w colinéaire à u en fonction des vecteurs $\overrightarrow{AB}$ et $\overrightarrow{AC}$, et des coordonnées de textures des points, $\begin{pmatrix} u_A \\ v_A \end{pmatrix}, \begin{pmatrix} u_B \\ v_B \end{pmatrix}, \begin{pmatrix} u_C \\ v_C \end{pmatrix}$:

$$\vec{w} = \frac{\overrightarrow{w'}}{\|\overrightarrow{w'}\|} \text{ , avec } \overrightarrow{w'} = c_1 \overrightarrow{AB} + c2 \overrightarrow{AC} \text{ où } c_1 = \frac{-v_C}{(u_{C.x} v_B - u_B v_C)} , c_2 = \frac{v_B}{(u_C v_B - u_B v_C)}$$

- Par construction les coordonnées de texture en B et en C sont différentes, on a donc toujours une solution. On obtient le repère local à la face $(\vec{T}, \vec{B}, \vec{N})$ par les calculs suivants : posons $\vec{Z} = \vec{w} \wedge \vec{N}$. Alors $\vec{B} = \vec{Z} \wedge \vec{N} ; \vec{T} = \vec{N} \wedge \vec{B}$ (On peut vérifier que $(\vec{T}, \vec{B}, \vec{N})$ est bien un repère orthonormé).
- Ensuite, de part la construction des texcells, on projette sur le plan horizontal du repère monde les vecteurs $\vec{T}$ et $\vec{B}$: après normalisation de ces projections, on obtient alors les vecteurs $\vec{t}$ et $\vec{b}$, qui sont parallèles aux vecteurs $\vec{i}$ *et* $\vec{j}$ et qui ont conservé l'orientation du repère $(\vec{T}, \vec{B}, \vec{N})$ (il est à noter que cette méthode de construction exclue les faces totalement verticale : de toutes façons, la végétation a du mal à se fixer sur les pentes très inclinées).

La matrice de passage recherchée est alors $M_1 = [\vec{t}\,\vec{b}\,\vec{N}]^{-1}$

Calcul des cartes de normales associées à chacune des faces

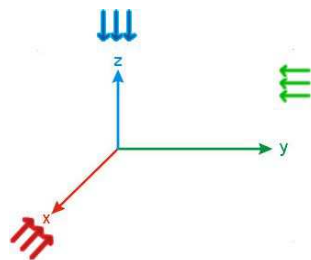
Illustration 22 - Une carte des normales. Elle encode les coordonnées X,Y,Z des vecteurs dans les composantes RVB.

Une fois dans le repère des faces, on aimerait connaître, pour chaque fragment, la matrice de passage vers un repère où l'axe des *z* est colinéaire à la normale en ce fragment. Pour se faire, nous aimerions associer à chaque tranche texturée des texcells la carte des normales correspondante.

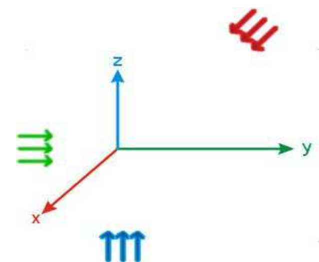
Afin d'obtenir ces cartes de normales, nous utilisons le modèle géométrique de forêt ayant servi à créer le motif de textures. Pour se faire, comme expliqué dans le chapitre sur l'illumination, nous allons associer à chaque composante chromatique un axe du repère, et essayer d'obtenir une image codant ces informations.

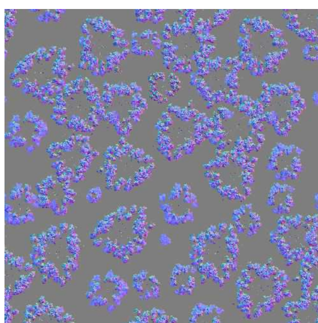
Dans Renderman® de Pixar, nous aurions écrit un shader nous permettant d'obtenir directement l'image codant les normales. Toutefois, sous la version de 3DSMax à laquelle nous avions accès, cela n'est pas possible.

Nous avons donc tout d'abord éliminer les composantes ambiantes et spéculaires des matériaux : ceux-ci sont alors complètement diffus – nous leur donnons comme couleur diffuse la couleur blanche.



Nous avons placé alors dans la scène 3 couples de lumières directionnelles, chacun ayant ses directions lumineuses colinéaires à une axe du repère monde, mais à l'intérieur de chaque couple, celles-ci sont opposées. Chaque couple a pour couleur lumineuse une couleur primaire pure : le couple parallèle au x a la couleur rouge, celui suivant y la couleur verte et enfin celui suivant z la couleur bleue.



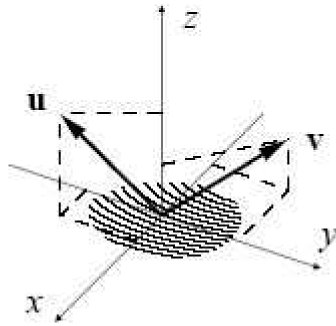


En effectuant deux rendus, à chaque fois avec une seule des lumières de chaque couple active, et en recombinant les images obtenues, on arrive à obtenir la carte des normales de chaque couche des texcells.

Nous connaissons ainsi pour chaque élément du texel la direction de sa normale.

6.3 Calcul de l'illumination

Rappelons la formule donnée par Lafortune et al. dans le cas de nos simplifications. Pour chaque fragment, elle s'obtient par :



$$\begin{pmatrix} R \\ V \\ B \end{pmatrix} = \begin{pmatrix} a_r \\ a_v \\ a_b \end{pmatrix} + \left(\left[C_{xy}(u_x v_x + u_y v_y) + C_z u_z v_z \right]^n \right) * \begin{pmatrix} d_r \\ d_v \\ d_b \end{pmatrix} * max(u_z, 0)$$

où :

- $\vec{u}$ est le vecteur (normé) pointant vers la source lumineuse,
- $\vec{v}$ est le vecteur (normé) pointant vers l'observateur,

Ces vecteurs étant exprimés dans le repère centrée sur le point à illuminer, et tel que l'axe z est confondu avec la normale, et les axes x et y sont les principales directions d'anisotropie.

Posons $\alpha = \left[C_{xy} * (u_x v_x + u_y v_y) + C_z * u_z v_z \right]^n$, et intéressons nous un peu plus à ce terme. Celui-ci peut se réécrire :

$$\alpha = \left((\vec{v})^T \begin{bmatrix} C_{xy} & 0 & 0 \\ 0 & C_{xy} & 0 \\ 0 & 0 & C_z \end{bmatrix} \vec{u} \right)^n = \left((\vec{v})^T C \vec{u} \right)^n \text{ avec } C = \begin{bmatrix} C_{xy} & 0 & 0 \\ 0 & C_{xy} & 0 \\ 0 & 0 & C_z \end{bmatrix}$$

Si l'on exprime les vecteurs $\vec{u}$ et $\vec{v}$ dans le repère monde, l'expression devient :

$$\alpha = \left((M_2\ M_1\ \vec{u})^T C (M_2\ M_1\ \vec{v}) \right)^n = \left((M_1\ \vec{u})^T \left[M_2^T C M_2 \right] (M_1\ \vec{v}) \right)^n$$

Le terme $M = M_2^T C M_2$ correspond, comme M_2 est une matrice de passage dans une base orthonormée et C diagonale, a une double dilatation de l'espace : une suivant l'axe des z du repère locale au fragment, et une autre dans l'espace vectoriel complémentaire ; si on appelle $\vec{N}$ le vecteur normal au fragment, $M \vec{u}$ peut alors s'écrire :

$$M \vec{u} = C_{xy}(\vec{u} - (\vec{u} . \vec{N}) \vec{N}) + C_z (\vec{u} . \vec{N}) \vec{N}$$

$$M \vec{u} = C_{xy} \vec{u} + (C_z - C_{xy})(\vec{u} . \vec{N}) \vec{N}$$

Donc, si on pose $D = max(\vec{u} . \vec{N}, 0)$ et $\tilde{C} = (C_z - C_{xy})$,

$$\begin{pmatrix} R \\ V \\ B \end{pmatrix} = \begin{pmatrix} a_r \\ a_v \\ a_b \end{pmatrix} + D \left(\left[(C_{xy}\ \vec{u} + \tilde{C} D \vec{N}) . \vec{v} \right]^n \right) \begin{pmatrix} d_r \\ d_v \\ d_b \end{pmatrix}$$

l'avantage de cette formule est que $\vec{u}, \vec{v}, \vec{N}$ peuvent être exprimés dans le repère local à la face.

Nous avons donc à stocker 6 coefficients par fragment : les trois coefficients du lobe, et l'expression de la normale dans le repère local. Compte tenu du fait que l'on ne peut stocker au maximum que 4 éléments par texture, cela revient donc à utiliser 2 textures.

Il faut noter que dans le cas des textures dont les composants sont des réels, ceux-ci sont ramenés à l'intervalle [0 ;1]. Une renormalisation est donc nécessaire.

6.4 Phase de rendu

A l'initialisation de l'application, pour chaque vertex de chaque face des supports des texcells, on a précalculé la matrice M_1 permettant de passer du repère monde au repère local. Celle-ci sera transmise au vertex program en tant qu'attribut du vertex.

On a également précalculé et stocké dans les textures les 6 coefficients permettant de calculer le

lobe étendu.

Le vertex program

Pour chaque vertex envoyé à la carte, ce programme va calculer le vecteur $\vec{v}$ donnant la direction de la caméra ainsi que celui donnant la direction lumineuse $\vec{u}$, le tout exprimé dans l'espace locale à la face. Il va ensuite passer ces paramètres au fragment program.

Le valeur de $\vec{v}$ s'obtient en faisant $\vec{v}=\frac{\vec{v}'}{\|\vec{v}'\|}$, avec $\vec{v}'=M_1\left[\frac{1}{Camera_w}\begin{pmatrix}Camera_x\\Camera_y\\Camera_z\end{pmatrix}-\frac{1}{Vertex_w}\begin{pmatrix}Vertex_x\\Vertex_y\\Vertex_z\end{pmatrix}\right]$, où M_1 est la matrice de passage du repère monde vers le repère locale à la face (passée comme attribut du vertex), $(Camera_x, Camera_y, Camera_z, Camera_w)$ les coordonnées homogènes de la caméra dans le repère monde, et $(Vertex_x, Vertex_y, Vertex_z, Vertex_w)$ celles du vertex en cours de traitement. De même, $\vec{u}$ se calcule par : $\vec{u}=\frac{\vec{u}'}{\|\vec{u}'\|}$, avec $\vec{u}'=M_1\begin{pmatrix}Lumière_x\\Lumière_y\\Lumière_z\end{pmatrix}$, où $(Lumière_x, Lumière_y, Lumière_z, 0)$ est le vecteur position, dans le repère monde, de la lumière en coordonnées homogènes.

Le fragment program

Pour chaque fragment, la carte graphique a réalisé une interpolation sur les vecteurs $\vec{v_1}, \vec{v_2}, \vec{v_3}$ donnant la direction de la caméra calculé en chaque vertex du triangle – nous appellerons par la suite $\vec{v}$ cette interpolation (elle réalise la même interpolation sur les vecteurs $\vec{u_1}, \vec{u_2}, \vec{u_3}$, mais comme ceux-ci sont identiques par face, nous obtenons toujours le même vecteur $\vec{u}$).

Le fragment program va alors lire les coefficients du lobe C_1, C_2, n ainsi que l'expression de la normale $\vec{N}$, et effectuer le calcul de l'illumination :

$$\begin{pmatrix}R\\V\\B\end{pmatrix}=\begin{pmatrix}a_r\\a_v\\a_b\end{pmatrix}+D\left(\left[(C_1\,\vec{u}+C_2\;D\vec{N}).\vec{v}\right]^n\right)\begin{pmatrix}d_r\\d_v\\d_b\end{pmatrix}\quad \text{avec}\quad D=max(\vec{u}.\vec{N},0)$$

6.5 Mip-mapping

Un traitement particulier doit être effectuer lors du mip-mapping, afin d'assurer la cohérence des fonctions d'illumination. Comme rappelé au chapitre 4 sur l'illumination, chaque pixel est le résultat de la projection d'un grand nombre de données géométriques, chacune apportant sa contribution au niveau de l'éclairage de ce pixel – il convient donc de calculer la fonction d'illumination

L'idéal serait que l'interpolation de m BRDFs écrites sous la forme d'un lobe de cosinus généralisé puisse s'écrire sous la forme d'un lobe de cosinus généralisé :

$$\exists\left(C_{xy},\tilde{C},\vec{N},n\right),\forall(\vec{u},\vec{v}),$$

$$\left[(C_{xy}\,\vec{u}+\tilde{C}(\vec{u}.\vec{N})\vec{N}).\vec{v}\right]^n=\frac{1}{m}\sum_{j=1}^{m}\mu_j\left[(C_{xy}^j\,\vec{u}+\tilde{C}_j(\vec{u}.\vec{N}_j)\vec{N}_j).\vec{v}\right]^{n_j}$$

Malheureusement, ce n'est à priori pas le cas. Une des solutions possibles est de calculer la BRDF

résultante et d'essayer de trouver le lobe de cosinus généralisé qui approche le mieux cette BRDF.

On peut prendre en première approximation $\vec{N} = \frac{\sum_{j=1}^{m} \mu_i \vec{N}_j}{\left\| \sum_{j=1}^{m} \mu_i \vec{N}_j \right\|}$, et calculer dans cet espace l'expression de la BRDF approchée via des méthodes dérivées de Levenberg-Marquardt, une méthode qui est fiable et converge généralement après quelques itérations vers une solution précise. La méthode dite d'« Adaptive simulated annealing »[INGBER] donnant des résultats intéressants dans le cas de recherche d'approximation de BRDFs en BRDF de Lafortune[Backer03].

L'approximation d'une BRDF est un calcul itératif assez long, et qui doit donc être précalculé.

6.6 *Résultats et analyse*

Nous n'avons pas eu le temps de finir d'implémenter le mip-mapping des coefficients.

Voici quelques résultats que nous pouvons obtenir en éliminant toute forme de mipmapping. La vitesse de calcul est de l'ordre de 4 fps, pour des scènes de 512x512 pixels.

Tout comme pour l'ombrage dynamique, une des grandes limitations est la réalisation par le CPU de certaines tâches dédiées au GPU.

Nous avons la possibilité d'augmenter grandement le réalisme ajoutant à l'équation d'illumination les composantes chromatiques des différentes sources lumineuses ; en effet, le ciel n'émet pas une lumière blanche, tout comme le soleil. En jouant sur ces paramètres, nous pouvons reproduire des couchers de soleil ou un ciel nuageux, au prix malheureusement de calculs plus importants. Toutefois, à cause de la perte de puissance due au calcul du CPU, cette augmentation de calcul n'a aucune influence sur la vitesse du rendu...

Ainsi, les images ci-dessous sont toujours obtenues à 4 fps.

Afin d'utiliser au mieux les capacités de réalisme du modèle de Lafortune, il faudrait pouvoir obtenir les coefficients de celui-ci pour certains types d'arbres (nous avons pris de manière empirique

$C_{xy}=0.86$, $C_z=0.77$, et n=0.5), et finir d'implémenter le mipmapping pour les objets lointains, afin d'améliorer la qualité de l'affichage. Néanmoins, ces résultats sont très encourageants.

7 Illumination et ombrage

Nous avons vu aux chapitres précédents des méthodes d'ombrage et d'illumination adaptées au rendu par texcells de la forêt, et qui nous ont permis d'obtenir des résultats convaincants: les doubles horizon map ainsi qu'une méthode empirique nous ont permi d'ombrer la forêt, mais avec une illumination précalculée, tandis que l'illumination calculée par le modèle de Lafortune nous permet d'obtenir un comportement lumineux réaliste, et ce sans ombrage.

Nous avons donc cherché à coupler les effets précédemment réalisés : nous présentons ici les difficultés que nous avons rencontrées.

7.1 Un problème : la carte graphique

Afin d'implémenter l'ombrage des doubles horizon maps, nous avons besoin de 4 textures : deux pour les horizons maps, une pour la motif, et enfin une cube map.

Pour implémenter Lafortune, nous avons besoin de 4 textures : deux contenants les coefficients et la normale, et deux contenants le motif ambiant et le motif diffus.

La combinaison devrait donc comporter 7 textures : celles des motifs ambiant et diffus, celles des coefficients, celles des horizon maps et enfin celle de la cube map.

Or, la carte graphique sur laquelle a été effectué le développement ne peut acceder simultanément qu'à 4 textures par fragment – On peut néanmoins réduire le nombre de texture utilisé, en remplaçant la cube map pour une fonction jouant un rôle équivalent dans les fragments program.

Nous nous retrouvons donc avec 6 textures à accéder par fragment, ce qui reste bien trop grand pour notre carte graphique (une Geforce FX 5800); les cartes récentes telles que l' ATI X800 peuvent, elles, traiter jusqu'à 16 textures simultanément.

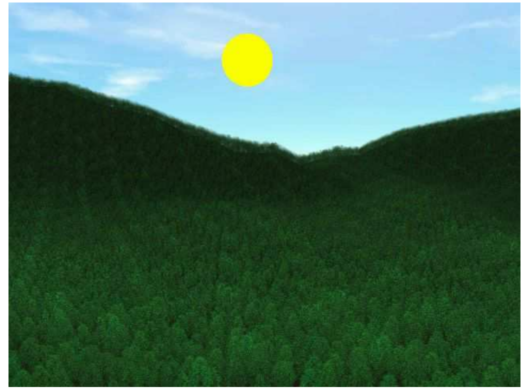
7.2 Une solution dégradée

Nous avons donc opté, afin d'essayer de coupler ces deux effets, pour une solution intermédiaire : nous n'utiliserions qu'une seule horizon map, et nous abandonnons le modèle de Lafortune pour une illumination par pixel de type lambertien : la fonction d'illumination est alors simplement $\vec{N}.\vec{l}$, avec $\vec{N}$ le vecteur unitaire normal à chaque fragment contenu dans une normal map, et $\vec{l}$ le vecteur pointant vers la source lumineuse.

Nous utilisons alors les 4 textures accessibles simultanément.

Là aussi, pour des raisons théoriques, nous ne pouvons appliquer le filtrage des textures de normal. Aussi, nous avons désactivé tous les filtrages au moment de faire le rendu.

Voici quelques exemples de résultats, obtenus à des fréquences variant entre 3 et 4 fps , sachant que certains points du code auraient pu être grandement optimisés:



Le couplage des méthodes s'est réalisé en une journée, preuve de la forte modularité des différents éléments du programme.

L'un des points positifs est qu'il n'y a pas eu de problèmes de fusion entre les différentes techniques employés : celle-ci n'a pas créé d'artefacts. L'illumination utilisée n'est pas celui de Lafortune, mais ces résultats laissent supposer que si celle-ci était implémentée, cela ne poserait également pas de problèmes d'incohérence : nous espérons pouvoir avoir à notre disposition des cartes permettant d'accéder à plus de textures.

Malgré la simplification du modèle d'éclairage, les résultats sont toutefois convaincants.

8 Bilan et conclusion

Nous avons présenté ici de nouveaux moyens d'obtenir de l'ombrage pour des scènes où des forêts sont rendues par la méthode des texcells, en interpolant plusieurs horizon maps entre elles, ainsi qu'une implémentation de l'illumination de Lafortune, le tout adapté aux nouvelles capacités des cartes graphiques.

Là où les méthodes déjà existantes ne permettaient qu'un éclairage et un ombrage précalculé, nous arrivons a obtenir des résultats satisfaisants et très encourageants, tant au niveau de la qualité visuelle que de la fréquence des images générées, qui permettent un usage interactif des ces méthodes. De plus, elles sont aisées à programmer (les programmes envoyés à la carte graphique ne font qu'une vingtaine de lignes, et l'adaptation d'un code déjà existant s'est réalisé en moins de deux semaines), ce qui peut leur assurer une large diffusion.

Il est à noter que ces techniques d'ombrage et d'illumination peuvent convenir à tous types de scènes naturelles complexes dont les éléments sont réparties dans une couche posée sur une surface, telles que des poils, une coupe des cellules de la peau, etc... Notre apport a déjà fait sur ce point l'objet d'un poster à la conférence SIGGRAPH 2004[CDN04].

Toutefois, nous pensons pouvoir aller encore plus loin, notamment en optimisant l'ensemble et en profitant des nouvelles capacités des dernières cartes graphiques - celles-ci nous permettrons sûrement d'implémenter simultanément les techniques d'ombrages par double horizon maps et l'illumination de Lafortune, ainsi que de tirer pleinement partie de la puissance des GPU.

Afin d'améliorer le réalisme de la forêt, notamment pour des vues plus proches des arbres, on peut coupler la méthode des texcells avec des méthodes de rendu géométriques (billboard et polygones pour les arbres proches, surfels pour les arbres lointains) – il serait alors intéressant de trouver un moyen d'étendre les méthodes que nous avons proposées à ces rendus, tout en s'assurant que le passage d'un mode de rendu à un autre se fasse de manière graduelle et cohérente.

9 Bibliographie

[Backer03] BACKER J., KIMBERLEY K., AND SKRYPNYK I., 2003. Stochastic Local Search for BRDF Parameterization.

[BLINN78] BLINN F. 1978. Simulation of wrinkled surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ACM Press, 286-292.

[CARMACK00] CARMACK J. 2000. Carmack On Shadow Volumes, developer.nvidia.com/attach/5628.

[CDN04] COHEN, F., DECAUDIN, P., AND NEYRET, F. 2004. GPU-Based Lighting and Shadowing of Complex Natural Scenes. *SIGGRAPH 2004 Poster (conf. Select CD-ROM and conf. DVD-ROM).* Los Angeles, USA (2004). www-imagis.imag.fr/Publications/2004/CDN04/.

[CROW77] CROW, F. C. 1977. Shadow algorithms for computer graphics. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, ACM Press, 242-248.

[DN04] DECAUDIN, P., AND NEYRET, F. 2004. Rendering forest scenes in real-time. In *Eurographics Symposium on Rendering*, A. K. H. W. Jensen, Ed.

[FERNANDO01] FERNANDO, R., FERNANDEZ, S., BALA, K., AND GREENBERG, D. P. 2001. Adaptive shadow maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, 387-390.

[FOURNIER92] FOURNIER, A. 1992. Filtering normal maps and creating multiple surfaces. Tech. rep.

[GOURAUD71] GOURAUD, H. 1971. Continuous Shading of Curved Surfaces. *IEEE Transactions on Computers C-20*, 6, 623-629.

[HAINES03] HAINES E., MÖLLER T., AND AKENINE-MOLLER T. 2003. Real-Time Rendering, A. K. Peters, Ltd., ISBN 1568811829.

[HEIDMANN91] HOEIDMANN, T. 1991. *Real shadows, real time. Iris Universe*, 18:28-31. Silicon Graphics, Inc.

[INGBER] INGBER L. Adaptive Simulated Annealing for Nonlinear Systems, www.ingber.com/#ASA.

[JACQ01] JACQUEMOUD S., AND USTIN S.L. 2001. *Leaf optical properties: A state of the art*, in *Proc. 8th Int. Symp. Physical Measurements & Signatures in Remote Sensing*, Aussois (France), 8-12 January 2001, CNES, pages 223-232.

[LAFORTUNE97] LAFORTUNE, E. P. F., FOO, S.-C., TORRANCE, K. E., AND GREENBERG, D. P. 1997. Non-linear approximation of reflectance functions. In *Proc. of SIGGRAPH 97*.

[MACIEL95] MACIEL, P. W. C., AND SHIRLEY, P. 1995. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM Press, 95-ff.

[MAX88] MAX, N. L. 1988. Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer 4*, 2 (July), 109-117.

[MCALLISTER02] MCALLISTER, D. K., LASTRA, A., AND HEIDRICH, W. 2002. Efficient rendering of spatial bi-directional reflectance distribution functions. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, 79-88.

[MCCOOL00] MCCOOL, M. D. 2000. Shadow volume reconstruction from depth maps. *ACM Trans. Graph. 19*, 1, 1-26.

[MFP01] MEYER A., NEYRET F., AND POULIN, P. 2001. Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering 2001*, -.

[MN98] MEYER, A., AND NEYRET, F. 1998. Interactive volumetric textures. In *Eurographics Rendering Workshop 1998*, Springer Wein, New York City, NY, G. Drettakis and N. Max, Eds., Eurographics, 157-168. ISBN 3-211-83213-0.

[NC99] NEYRET, F., AND CANI, M.-P. 1999. Pattern-based texturing revisited. *Computer Graphics* (Aug), 235-242. SIGGRAPH'99 Conference Proceedings.

[NGUYEN99] NGUYEN, H. 1999. *Casting Shadows on Volumes*. Game Developer 6 (3): 44-53

[NOE99] NOE N. 1999. *Etude de fonctions de distribution de la réflectance bidirectionnelle*, Thèse ENSM.SE et Université Jean Monnet.

[NVIDIA01] NVIDIA, 2003. Technical Brief: UltraShadow Technology, www.nvidia.com/object/LO_20030508_6927.html

[PFISTER00] PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. In *Proceedings of SIGGRAPH'00*, ACM, 335-342.

[PHONG75] PHONG, B. T. 1975. Illumination for computer generated pictures. *Commun. ACM 18*, 6, 311-317.

[Reev85] REEVES, W. T., AND BLAU, R. 1985. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM Press, 313-322.

[SCHLICK94] SCHLICK, C. 1994. An inexpensive BRDF model for physically-based rendering. *Computer Graphics Forum 13*, 3, 233-246.

[SLOAN2000] SLOAN, P., AND COHEN, M. F. 2000. Hardware accelerated horizon mapping. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, 281-286.

[SD02] STAMMINGER, M., AND DRETTAKIS, G. 2002. Perspective shadow maps. In *Proceedings of ACM SIGGRAPH 2002*, ACM Press/ ACM SIGGRAPH, J. Hughes, Ed.

[TESSMAN89] TESSMAN, T. 1989. Casting shadows on flat surfaces. *Iris Universe* (Winter), 16-19. Silicon Graphics, Inc.

[WILLIAMS78] WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings) 12*, 3 (Aug.), 270-274.

[WILSON94] WILSON, O., VANGELDER, A., AND WILHELMS, J. 1994. Direct volume rendering via 3d textures. Tech. rep.

10 Annexe A – Listing des vertex et fragment programs

10.1 Double horizon map et ombrage empirique

Vertex program

```
/* Données passées par l'afficheur */
uniform vec3 LightPosition;  // Direction de la lumière dans le repère monde

/* données pour le fragment program */
varying vec3 lightDir;    // interpolated surface local coordinate light direction
varying float angle;      // angle de la lumière par rapport à la face

void main()
{
      // On calcule les coordonnées de texture du motif
      gl_TexCoord[0] = gl_TextureMatrix[0]*gl_MultiTexCoord0;
      // On plaque les horizons maps sur le terrain
      gl_TexCoord[1].xy = (gl_Vertex.xz+0.79)/(2*0.79);
      // On traite le vertex (clipping, etc)
      gl_Position= ftransform();

      // Calcul du repère local à la face

      const vec3 n = normalize(gl_Normal);
      vec3 b = cross(n,vec3(1,0,0));
      const vec3 t = cross(n,b);
      b = cross(n,t);

      // Transforme la direction de la lumière dans le repere locale
      const mat3 world_locale=mat3(
                  t.x,b.x,n.x,
                  t.y,b.y,n.y,
                  t.z,b.z,n.z
                  );
      const vec3 v=world_locale * LightPosition;

      // Calcul de l'azimut de la lumière
      angle=asin(v.z)/(3.1415927/2.0f);
      lightDir = v;
}
```

Fragment program

```
/* Données passées par l'afficheur */
uniform sampler2D texture;    // motif
uniform sampler2D horizonMap_bas;   // horizon map du bas
uniform sampler2D horizonMap_haut;  // horizon map du haut
uniform samplerCube conversionMap;  // texture cubique permettant de réaliser les
                                    // interpolation
uniform float volumicSlice;         // hauteur de la couche (0: niveau du sol,
                                          1 hauteur de la cime)

/* données récupérées du vertex program */
varying vec3 lightDir;    // interpolated surface local coordinate light direction
varying float angle;      // angle de la lumière par rapport à la face
```

```
void main()
{
	// On elimine les fragments qui sont en dehors des horizon maps
	if ( (gl_TexCoord[1].x<=0) || (gl_TexCoord[1].x>=1)
			|| (gl_TexCoord[1].y<=0) || (gl_TexCoord[1].y>=1) ) {
		discard;
	}

	// On récupère les informations stockées dans les textures
	const vec2 textureCoords=gl_TexCoord[0].xy;
	const vec2 horizonCoords=gl_TexCoord[1].xy;
	const vec4 textureValue = texture2D(texture, textureCoords);
	const vec4 horizonValue = texture2D(horizonMap_bas, horizonCoords);
	const vec4 horizon2Value = texture2D(horizonMap_haut, horizonCoords);
	const vec4 conversionValue = textureCube(conversionMap, lightDir.xzy);

	// Ici se réalise l'interpolation des directions
	const float angleLimit = dot(horizonValue,conversionValue);
	const float angleLimit2 = dot(horizon2Value,conversionValue);

	// On interpole les azimuts limites
	const float limit=mix(angleLimit,angleLimit2,volumicSlice);

	gl_FragColor.xyz=
		((smoothstep(limit-0.1,limit+0.1,angle))*lightDir.z*0.7+0.3)*
		(volumicSlice+0.5)*textureValue.xyz;
	gl_FragColor.w=textureValue.w;
}
```

10.2 Illumination de Lafortune

Vertex program

```
/* Données passées par l'afficheur */
uniform vec3 LightPosition;   // Direction de la lumière dans le repère monde
uniform vec3 CameraPosition;  // Position de la lumière dans le repère monde
uniform float volumicSlice2;  // Decalage par rapport au sol, dans le répère monde,
                              // de la couche de texture à laquelle
                              // appartient le vertex

/* données pour le fragment program */
varying vec3 lightDir;    // interpolated surface local coordinate light direction
varying vec3 viewDir;     // interpolated surface local coordinate view direction

void main()
{
	gl_TexCoord[0] = gl_TextureMatrix[0]*gl_MultiTexCoord0;
	gl_Position= ftransform();

	// On récupère la matrice M1

	const vec3 t = gl_Color.xyz;
	const vec3 b = gl_SecondaryColor.xyz;
	const vec3 n = gl_MultiTexCoord7.xyz;

	// On calcule la direction de la lumière dans le repère locale
	const mat3 world_tangente=mat3(
			t.x,t.y,t.z,
			b.x,b.y,b.z,
			n.x,n.y,n.z
			);
	{
		const vec3 v=world_tangente * LightPosition;
```

```
		lightDir = normalize(v);
	}

	// On calcule la direction de la caméra dans le repère local
	{
		const vec3 pos3 = (gl_Vertex.xyz)/gl_Vertex.w;
		const vec3 v = ( world_tangente *
				( CameraPosition.xyz -
					vec3(pos3.x,pos3.y+volumicSlice2,pos3.z)));

		viewDir = normalize(v);
	}

}
```

Fragment program

```
/* Données passées par l'afficheur */
uniform sampler2D ambiantMap;       // motif d'ambiant
uniform sampler2D diffuseMap;       // motid de diffus
uniform sampler2D componentMap1;    // Facteurs de Lafortune
uniform sampler2D componentMap2;    //

/* données récupérées du vertex program */
varying vec3 lightDir;    // interpolated surface local coordinate light direction
varying vec3 viewDir;     // interpolated surface local coordinate view direction

void main()
{
	const vec2 coords=gl_TexCoord[0].xy;
	const vec4 ambiantValue = texture2D(ambiantMap, coords);
	const vec4 diffuseValue = texture2D(diffuseMap, coords);
	const vec4 componentValue1 = texture2D(componentMap1, coords);
	const vec3 componentValue2 = texture2D(componentMap2, coords);

	// resultat

	const vec3 l=normalize(lightDir);
	const vec3 v=normalize(viewDir);
	const vec3 N=(componentValue1.xyz*2-1)*ambiantValue.a;
	const float D=dot(N,l);

	const float dotp2 = dot(	(componentValue2.x*4-2)*l+
					(componentValue2.y*6-3)*D*N,v);
	const float dotp = max(pow(clamp(dotp2,0,1),0.5)*D,0);

	const vec3 resultat = ((ambiantValue.xyz*vec3(0.3,0.5,0.6) + dotp	*
(vec3(diffuseValue.xyz)) ))/(1.5f);

	gl_FragColor.xyz = resultat;

	gl_FragColor.a = ambiantValue.a;
}
```

10.3 Illumination et ombrage dynamique

Vertex Program

```
/* Données passées par l'afficheur */
uniform vec3 LightPosition; // Direction de la lumière dans le repère monde

/* données pour le fragment program */
varying float angle;     // azimut de la lumière dans le repere locale à la face
```

```
varying vec3 lightDirPPL;// position de la lumière dans le repère locale à la face
varying vec3 lightDirHM; // position de la lumière dans le repère orthogonal
                         // lié à la face

void main()
{
      //  coordonnées de texture
      gl_TexCoord[0] = gl_TextureMatrix[0]*gl_MultiTexCoord0;
      gl_TexCoord[1].xy = (gl_Vertex.xz+0.79)/(2*0.79);
      gl_Position= ftransform();

    // position de la lumière pour l'éclairage (repere locale à la face,
    // avec normales transformées)

    {
        const vec3 n = gl_MultiTexCoord7.xyz;
            const vec3 t = gl_Color.xyz;
            const vec3 b = gl_SecondaryColor.xyz;

            // Transform light position into surface local coordinates
            const mat3 world_tangente=mat3(
                                                t.x,t.y,t.z,
                                                b.x,b.y,b.z,
                                                n.x,n.y,n.z
                                                );

          const vec3 v=world_tangente * LightPosition;

            lightDirPPL = normalize(v);
    }

    // position de la lumière pour les horizons maps, dans le repere
    // orthogonal lié à la face

      {
          const vec3 n = gl_Normal;
            vec3 b = cross(n,vec3(1,0,0));
            const vec3 t = cross(n,b);
            b = cross(n,t);

            // Transform light position into surface local coordinates
            const mat3 world_tangente=mat3(
                                                      t.x,b.x,n.x,
                                                      t.y,b.y,n.y,
                                                      t.z,b.z,n.z
                                                      );
            const vec3 v=world_tangente * LightPosition;

            angle=asin(v.z)/(3.1415927/2.0f);
            lightDirHM = v;
    }
}
```

Fragment Program

```
uniform sampler2D horizonMap_bas;
uniform sampler2D horizonMap_haut;
uniform samplerCube conversionMap;
uniform sampler2D ambiantMap;
uniform sampler2D diffuseMap;
uniform sampler2D normalMap;
uniform float volumicSlice;

varying vec3 lightDirPPL;       // interpolated surface local coordinate light
direction
```

```
varying vec3 lightDirHM;        // interpolated surface local coordinate light
direction
varying float angle;

void main()
{
	if ( (gl_TexCoord[1].x<=0) || (gl_TexCoord[1].x>=1)
		|| (gl_TexCoord[1].y<=0) || (gl_TexCoord[1].y>=1) ) {
		discard;
	}

	const vec2 coords=gl_TexCoord[0].xy;
	const vec4 ambiantValue = texture2D(ambiantMap, coords);
	const vec4 normaleValue = vec4(texture2DLod(normalMap, coords,0).xyz,1);
	const vec4 diffuseValue = texture2D(diffuseMap, coords);

	const vec2 horizonCoords=gl_TexCoord[1].xy;
	const vec4 horizonValue = texture2D(horizonMap_bas, horizonCoords);
	const vec4 horizon2Value = texture2D(horizonMap_haut, horizonCoords);
	const vec4 _conversionValue = textureCube(conversionMap, lightDirHM.xzy);

	// Nous n'avons plus de cube map: nous faisons les calculs à la main
	const mat2 m = mat2( cos(3.1415927/4),-sin(3.1415927/4),sin(3.1415927/4),cos
(3.1415927/4));
	const vec2 nlight= max(min(m* normalize(lightDirHM.xy),1),-1);
	vec4 conversionValue=max(asin(vec4(nlight.yx,-nlight.yx))/(3.1415927/2.0),
vec4(0));
	if (lightDirHM.z<0) conversionValue=0;

	const float angleLimit = dot(horizonValue,conversionValue);
	//

	// illumination
	const vec3 l=normalize(lightDirPPL);
	const float dotp = min(max(dot(l.xyz,
				(normaleValue.xyz*2-1)
				),0.0),1.0f);

	const vec3 resultat = ((ambiantValue.a*0.15f*vec3(0.7,1,0.8)
+ambiantValue.xyz*vec3(0.3,0.5,0.6)*2.3) + ((smoothstep(angleLimit-
0.1,angleLimit+0.1,angle)*0.7+0.3)*dotp)  * (vec3(0,diffuseValue.yz))*0.65)/(1.5f);

	gl_FragColor.xyz=(volumicSlice*0.5+0.5)*resultat.xyz;

	gl_FragColor.a=ambiantValue.a;
}
```

11 Annexe B - Poster SIGGRAPH 2004

Voici ci-joint le poster tel qu'il a été soumis à la conférence SIGGRAPH 2004.