Université de Montréal

# Texture volumique multi-échelle pour l'affichage de scènes complexes

par
Karim Ratib

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Décembre 1997

Université de Montréal
Faculté des études supérieures

Ce mémoire de maîtrise intitulé

# Texture volumique multi-échelle pour l'affichage de scènes complexes

présenté par
Karim Ratib

a été évalué par un jury composé des personnes suivantes :

| | |
|---|---|
| Président : | Jean Meunier |
| Directeur de recherche : | Pierre Poulin |
| Membre : | François Major |

Mémoire accepté le : 8 décembre 1997

# Sommaire

Les scènes géométriquement complexes représentent un problème en infographie. Le temps de calcul est élevé à cause du grand nombre de primitives à traiter. Le résultat visuel obtenu est rarement satisfaisant à cause de l'aliassage généré. Les techniques standards de réduction de l'aliassage, telles que le sur-échantillonnage, ne fournissent de bons résultats qu'au prix de temps de calcul excessivement importants. La chevelure est un exemple-type de ce type de scènes. Nous présentons dans cet ouvrage une méthode de représentation multi-échelle des objets qui mène à un rendu plus rapide et de meilleure qualité. Cette représentation se base sur l'information de réflectance au lieu de la géométrie. Les objets géométriques sont analysés au pré-calcul pour créer un volume contenant l'information de réflectance à plusieurs niveaux de résolution. Durant le rendu, le niveau approprié est choisi de sorte à générer le moins d'aliassage possible. De plus, la géométrie complexe est éliminée, résultant en de meilleurs temps de calcul. Nous étudions l'application de cette méthode au rendu de la chevelure, puis nous présentons l'ébauche d'un modèle d'animation adapté à notre représentation.

**Mots-clefs :**
rendu, texture volumique, texel, réflectance, géométrie répétitive, filtrage de géométrie, rendu multi-échelle, rendu multi-résolution, complexité, visibilité, chevelure, cheveux.

*A C., C., C., R.,*
*dans tous les ordres,*
*on se comprend.*

# Remerciements

Beaucoup de monde a contribué à cette maîtrise. D'abord, elle a été possible grâce à la générosité de la Bourse de la Francophonie, envers laquelle je suis profondément reconnaissant.

Pierre Poulin a été mon superviseur, je l'en plains. J'espère qu'il ne regrette pas trop avoir répondu à cet e-mail envoyé d'Egypte, un jour du printemps 95. Pendant ces deux ans, il n'a cessé de m'expliquer, et de me re-expliquer, tout ce que je sais à présent en infographie. Mais j'ose croire que dans la foulée, nous sommes devenus amis.

Pierre a été présent, indispensable, à chaque étape de ce projet. C'est lui qui m'en a suggéré l'idée, et qui m'a dirigé vers une approche prometteuse. Nos interminables discussions m'ont aidé à voir plus clair quand je n'y comprenais plus rien. Il a patiemment attendu que le code prenne forme, alors que les retards se multipliaient. Pour chaque difficulté rencontrée, il s'est mis – presque furieusement – à la tâche afin d'améliorer la qualité de la méthode, tandis que j'étais prêt à abandonner.

Le cours du projet m'a mené à étudier de près une approche proposée par Fabrice Neyret. J'ai eu le privilège de connaître Fabrice, ce qui m'a permis de le harceler de questions lors de ses visites au département, et de le noyer de courrier lorsqu'il arrivait à s'en échapper. Fabrice m'a toujours répondu immédiatement, de bonne volonté, et avec une profusion de détails. Je lui dois beaucoup pour la compréhension des subtilités de sa méthode. Il m'a aussi montré les accords de quelques chansons françaises, j'espère que nous aurons l'occasion de les rejouer ensemble.

Fabrice a trouvé le courage (et le temps) de lire la première version de ce mémoire. Il m'a envoyé une énorme quantité de commentaires, que j'ai pris en considération pour la révision finale. Je remercie par la même occasion les membres du jury dont les commentaires étaient clairs et pertinents.

Marie-Paule Gascuel (une compatriote !) a suivi l'évolution du projet depuis le tout début. Intéressée par le côté animation, elle s'est vue obligée de m'expliquer son modèle par e-mail, à plusieurs reprises, et malgré ma quasi-ignorance de tout ce qui touche à l'animation. Elle attend toujours la réalisation de ce modèle, je souhaite avoir la chance d'y participer un jour.

Je tiens aussi à remercier Arash Habibi, pour son aide à déchiffrer un article, à dessiner un œil, ainsi que pour son excellente cuisine. Les membres du laboratoire d'infographie ont été une source intarissable d'idées, de questions, de réponses, et de très bons CDs. Pierre McKenzie m'a battu au squash un nombre incalculable – oui, même par une machine de Turing – de fois, mais

la revanche est proche ! Enfin, Claire Valois et le personnel administratif du DIRO ont toujours été un modèle d'efficacité et de courtoisie.

# Table des matières

# Table des figures

# Chapitre 1

# Introduction

## 1.1 Objet des travaux

L'un des grands problèmes en infographie est le traitement de scènes très complexes. Ces scènes posent plusieurs défis. D'abord, le grand nombre de détails à conserver requiert une méthode d'accès aux informations efficace et rapide. Ensuite, le temps de calcul nécessaire pour afficher[1] toutes ces informations doit demeurer raisonnable. Enfin, lorsque nous considérons le rendu dans le contexte de la théorie du signal, nous désirons obtenir une bonne approximation (une image) de notre information (les objets de la scène). Une bonne approximation se traduit entre autres par la réduction des hautes fréquences, qui sont collectivement nommées "aliassage" en infographie. Cet aliassage se manifeste par exemple lorsque de nombreux objets lointains sont affichés, résultant en du bruit ou du moiré, dus à la présence de pixels adjacents de différentes couleurs.

Certaines scènes exhibent une grande répétitivité. Par exemple, les prairies, les forêts, la fourrure, la chevelure, peuvent toutes être construites à partir d'un nombre relativement restreint de primitives qui sont répétées et déformées. Ce type de scènes élimine le problème de l'accès rapide aux informations, qui sont typiquement peu nombreuses. Cependant, le problème d'un rendu efficace et exhibant peu d'aliassage demeure entier. L'une des sources de ce problème est que toutes les informations sont traitées identiquement, irrespectivement de leur apport individuel au résultat final. Par exemple, les objets lointains sont traités de la même manière que les objets proches du point de vue. Une solution consiste donc à donner aux informations une importance proportionnelle à leur contribution, d'où l'idée de représenter les mêmes informations à plusieurs échelles.

Cet ouvrage se situe dans le contexte de cette représentation multi-échelle de l'information. Nous y décrivons une méthode de représentation volumique basée sur l'information de réflec-

[1] Dans cet ouvrage, "afficher une image" dénote le processus de calcul et de visualisation d'une image à partir des informations de la scène.

tance, au lieu de la géométrie. Comme exemple-type de l'application de cette méthode à des scènes complexes, nous étudions le cas de la modélisation de la chevelure. Notre modèle exploite la répétitivité de la scène en représentant l'élément répétitif par une texture tri-dimensionnelle à plusieurs niveaux de résolution. Le bénéfice tiré d'une telle représentation est double. D'abord, nous remplaçons la majorité de la géométrie de la scène par des copies de ce volume (puisqu'il représente l'élément répétitif). En supposant que la géométrie répétée est complexe, cet échange représente un gain en mémoire. Ensuite, la représentation multi-résolution permet de choisir le niveau de représentation adapté à l'apport de chaque élément à l'image finale. Nous contribuons ainsi à la réduction de l'aliassage et du temps de calcul (puisque seule l'information strictement nécessaire est traitée).

Afin de construire cette texture volumique, nous devons effectuer une phase de pré-calcul durant laquelle la géométrie explicite de l'élément répétitif est transformée en un volume contenant l'information particulière qui sera utilisée lors du rendu. Cette information est elle-même représentée à plusieurs niveaux de détail afin de réaliser l'approche multi-résolution. L'usager peut alors remplacer la scène originale (contenant la géométrie) par une scène équivalente construite à partir des copies du volume pré-calculé. Durant le rendu, les copies du volume sont affichées.

## 1.2 Organisation de l'ouvrage

Cet ouvrage est divisé en trois grandes parties. La première partie est une introduction à notre méthode : au chapitre 2, nous définissons précisément le problème que nous allons étudier, puis nous présentons les approches existantes à la solution du problème, ainsi qu'une vue d'ensemble de notre méthode. La seconde partie présente les détails de la mise en œuvre, elle est composée des chapitres 3 (construction du volume) et 4 (rendu des copies du volume). Enfin, la troisième partie présente les résultats obtenus, ainsi qu'une revue des extensions possibles, au chapitre 5.

Un mot sur la notation utilisée :

- Les points géométriques sont notés $P, Q, \ldots$ ;
- Les vecteurs sont notés $\mathbf{v}, \mathbf{n}, \ldots$ ;
- Les matrices sont notées $\mathbf{M}, \mathbf{J}, \ldots$ ; $\mathbf{M}^{-1}$ est l'inverse de $\mathbf{M}$, $\mathbf{M}^t$ est sa transposée ;
- La transformation de l'espace-objet à l'espace-monde est notée $\mathcal{T}$ ; $\mathcal{T}^{-1}$ est son inverse (en supposant que cet inverse existe).

# Chapitre 2

# Le problème

*En vue d'extraire les objectifs de ce travail, nous définissons de manière plus précise le problème sur lequel nous nous penchons. Nous présentons ensuite une description des approches existantes concernant le problème ou des problèmes apparentés, puis une introduction à la méthode que nous utilisons.*

## 2.1 Définition du problème

Considérons la scène suivante : un certain nombre d'acteurs virtuels évoluent dans un environnement. Nous désirons effectuer un rendu réaliste de cette scène. Un rendu réaliste implique que les petits détails de la scène devront être pris en considération. Par exemple, les cheveux des acteurs seront affichés.

La chevelure présente un problème ardu en infographie. Une chevelure contient en moyenne 120,000 cheveux individuels, qui sont des cylindres généralisés dont le diamètre varie entre 40 et 120 $\mu$m [Rob88, RCI91]. Un grand nombre de difficultés découlent de cette spécification. D'abord, un cylindre généralisé est discrétisé en une série de polygones. Si les cheveux sont droits, le nombre de polygones générés est un faible multiple du nombre de cheveux, par exemple $5 \times 120,000 = 600,000$ polygones. Mais si la chevelure représentée est ondulée, nous obtenons une quantité de polygones qui dépasse la capacité de traitement de la plupart des algorithmes de rendu, et qui peut même défier les algorithmes de gestion de mémoire des systèmes d'exploitation. Un autre problème est dû à la faible taille des cheveux par rapport aux pixels, qui cause un aliassage important dans les images calculées, à moins d'utiliser des techniques de suréchantillonnage très coûteuses. D'autres problèmes surviennent dans l'animation de la chevelure. Une animation complète nécessite la détection des collisions entre les cheveux et les autres objets de la scène, ainsi qu'entre les cheveux eux-mêmes. Nous obtenons des systèmes d'ordre $n^2$ où $n$ est le nombre d'objets dans la scène, qui sont clairement trop coûteux à résoudre à chaque étape de l'animation. De plus, modéliser les caractéristiques physiques des cheveux, tels que la rigidité, la viscosité (due par exemple à la présence du gel), etc., est important pour obtenir des

résultats réalistes. Cependant, ces calculs alourdissent encore plus les systèmes dynamiques.

La plupart des techniques utilisées jusqu'a présent sont basées sur la simplification des modèles géométriques et dynamiques de la chevelure, afin d'obtenir des systèmes calculables (nous décrivons plus bas les approches que nous avons rencontrées dans la bibliographie). Cependant, ces simplifications nuisent de façon importante au réalisme du résultat final, ce qui remet en question l'utilisation de telles techniques. Notre objectif principal est donc de concevoir et implanter une technique de rendu spécifiquement adaptée à des scènes hautement complexes et qui exhibent une grande répétitivité. Comme exemple-type de ce genre de scène, nous désirons étudier le cas du rendu de la chevelure. Nous désirons aussi prendre en compte l'intégration de cette technique à un modèle d'animation de la chevelure.

Le traitement de la complexité en infographie est un problème ouvert, et nous ne prétendons pas y remédier de façon définitive. Nombreuses sont les options qui se sont présentées à nous aux différentes étapes, c'est pourquoi nous nous sommes efforcés de construire un logiciel extensible et intuitif, qui nous a permis - et qui permettra à d'autres étudiants plus tard - d'expérimenter dans le domaine. Nous ne demandons pas que l'implantation des algorithmes y soit optimale, puisque l'optimisation logicielle est souvent l'antithèse de la généralité. Nous demandons en revanche que l'ordre des algorithmes utilisés soit le plus proche possible de l'optimalité.

Afin de mieux cerner le problème et de choisir une solution, il est nécessaire de revoir les approches existantes au problème du traitement de la chevelure. Nous présentons à la prochaine section un survol du domaine.

## 2.2 Les approches existantes

### 2.2.1 Modélisation

La modélisation de la chevelure a pour but de décrire la configuration de la centaine de milliers de cheveux d'un crâne, en utilisant des descriptions de haut niveau si possible.

#### 2.2.1.1 Géométrie

La méthode la plus couramment utilisée est de représenter chaque cheveu comme une entité géométrique, telle une chaîne de petits cylindres droits, et de modéliser la chevelure à l'aide de cheveux représentatifs, autour desquels des mèches sont créées. Les cheveux de chaque mèche ressemblent plus ou moins au cheveu représentatif, suivant certains paramètres. Par exemple, Watanabe *et al.* [WS92] définissent une mèche comme étant un ensemble de $m$ copies d'un cheveu représentatif, dont la direction varie aléatoirement autour de la direction de celui-ci, suivant un paramètre $r$ contrôlant la variance maximale. Il incombe à l'usager de définir les paramètres de chaque cheveu représentatif. Un tel cheveu est défini par sa longueur $l$, sa grosseur $d$, sa direction $\mathbf{v}$, et le nombre $n$ de segments (des prismes trigonaux) qui forment son corps (voir la figure 2.1). Daldegan *et al.* [DTKT93] utilisent une méthode similaire, sauf que les segments de

cheveu sont représentés par des cylindres droits. La difficulté dans ces deux cas réside dans la spécification des paramètres des cheveux représentatifs afin d'obtenir une coiffure voulue : il est parfois difficile, et long, de traduire notre intention en une simple série de cheveux.

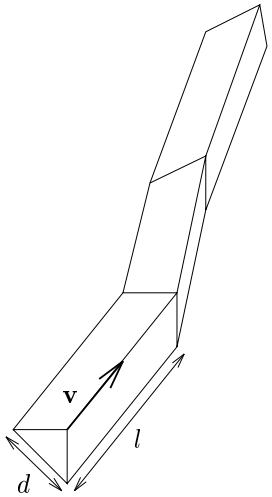


FIG. 2.1: Modèle de prismes trigonaux, Watanabe *et al.* [WS92]

Pour pallier à ce problème, Anjyo *et al.* [AUK92] utilisent un modèle de poutre (*cantilever beam*) fixe à une extrémité et libre à l'autre. Chaque poutre représente un cheveu, elle est divisée en segments d'égale longueur (voir la figure 2.2). Soit une force $F$ appliquée au cheveu, cette force est distribuée également à chaque extrémité des segments du cheveu, causant deux genres de déformation, induites par le moment de courbure (*bending moment*) $M$ et la force de cisaillement (*shearing force*). Cette dernière est négligée dans le modèle, et la déformation est supposée élastique. Nous obtenons alors :

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = -\frac{M}{EI} \tag{2.1}$$

où $E$ est le module de Young et $I$ une propriété géométrique de la poutre, le produit $EI$ représentant la rigidité de celle-ci. A partir de l'équation 2.1, et de la position du premier point $P_0$, il est possible de calculer la position de tous les autres points. Ce modèle permet d'appliquer des forces qui donneront une apparence plus réaliste à la coiffure. La force due à la gravité est appliquée. De plus, d'autres forces externes aident à modéliser une coiffure spécifique. Ceci est accompli en paramétrisant les forces appliquées suivant leur position sur le crâne (qui est modélisé par un ellipsoïde). Un dernier problème à régler ici est de détecter si l'application de ces forces entraîne la pénétration des cheveux dans le crâne. Cette détection est simple dans le cas où le crâne est modélisé par un ellipsoïde : il suffit de vérifier $E(P_i) < 0$ pour chaque point $P_i$ d'un cheveu, où $E(P)$ est la forme quadrique de l'ellipsoïde. En cas d'intersection, Anjyo *et al.* [AUK92] proposent de déplacer directement le point hors de l'ellipsoïde, tandis que Daldegan *et al.* [DTKT93] proposent de calculer la force normale nécessaire à garder le point en dehors,

tel que décrit plus bas. En revanche, le modèle n'inclut pas la détection de collisions entre les cheveux eux-mêmes, ce qui nuit au réalisme du résultat final.

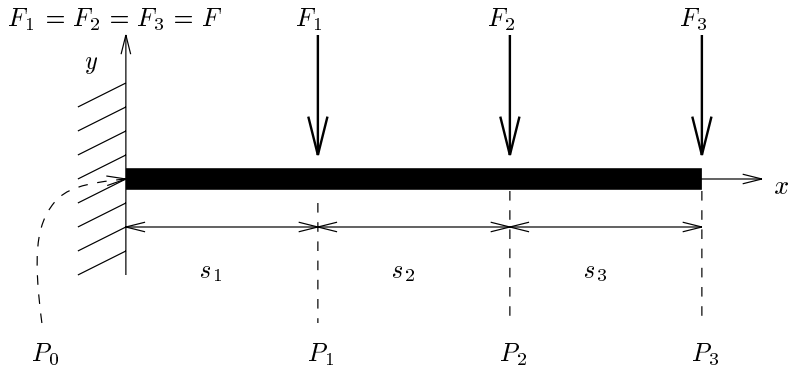


FIG. 2.2: Modèle de poutres, Anjyo *et al.* [AUK92]

#### 2.2.1.2 Sous-géométrie

Une approche totalement différente est proposée par Sourin *et al.* [SPS96] qui utilisent des fonctions implicites pour modéliser des objets complexes. Ils se basent sur l'hypertexture de Perlin *et al.* [PH89] qui définit les objets flous comme une fonction de densité $D(P)$ où $D = 1.0$ à l'intérieur de l'objet, $D = 0.0$ à l'extérieur et $0.0 < D < 1.0$ dans une région intermédiaire. A cette fonction de densité est appliquée une ou plusieurs fonctions de modulation $f_i$ qui servent à contrôler les caractéristiques de l'objet dans la région floue. Perlin *et al.* [PH89] définissent des opérations booléennes sur ces fonctions, opérations qui aident à la modélisation d'objets complexes. Sourin *et al.* [SPS96] définissent la fonction de densité $\mathbf{hair}(P)$ qui est essentiellement une fonction de bruit tri-dimensionnelle projetée sur la surface d'une sphère donnée. Pour un point $P$ donné, le point $P_s$ sur la sphère dans la direction de $P$ est calculé, puis utilisé comme argument pour la fonction **hair**. Une fonction de modulation $\mathbf{hairstyle}(P)$ est ensuite appliquée au résultat pour modéliser la coiffure voulue. Cette représentation de la chevelure est beaucoup plus compacte que la précédente, mais il est difficile de choisir les fonctions de modulation qui résulteront en une coiffure spécifique. Un outil de contrôle important est l'utilisation des opérations booléennes sur ces fonctions.

L'approche de Sourin *et al.* [SPS96] diffère conceptuellement des approches précédentes de par le fait que les cheveux ne sont plus considérés comme des objets géométriques indépendants : la chevelure entière est modélisée par la fonction de densité et les fonctions de modulation décrites plus haut. Nous passons à un niveau sous-géométrique, pour ainsi dire, qui réduit la complexité du niveau géométrique. Cette tendance est retrouvée chez Kajiya *et al.* [KK89], qui introduisent une généralisation des volumes de densité sous la forme de texels, c'est-à-dire des plaquages de texture tri-dimensionnelle qui capturent la micro-géométrie des surfaces et l'approximent par

un modèle d'illumination, prenant en compte l'atténuation due à la diffusion. Il faut noter que cette méthode est une technique de rendu, et non de modélisation : les auteurs décrivent une application de leur méthode à la génération de la fourrure en utilisant des systèmes à particules pour générer les poils de la fourrure. Nous discuterons plus en détail de cette technique de rendu plus bas, mais le but ici est de montrer que la modélisation de la chevelure peut être effectuée à plusieurs niveaux d'une hiérarchie basée sur l'échelle géométrique.

### 2.2.2 Affichage

Diverses approches ont été proposées pour créer des images réalistes de chevelure. En général, les auteurs ont utilisé leur modèle de chevelure tel que décrit plus haut pour effectuer le rendu.

#### 2.2.2.1 Géométrie

Watanabe *et al.* [WS92] utilisent la technique du Z-buffer pour effectuer le rendu de leurs prismes trigonaux. Pour modéliser l'illusion que la surface externe des cheveux brille à la lumière (*backlight*, à cause du passage de la lumière dans un volume de cheveux de faible densité et légèrement réfractif), ils utilisent deux Z-buffers parallèles, l'un devant la tête et le second derrière, pour calculer l'épaisseur de la masse chevelue à chaque pixel (obtenue par la différence entre les valeurs des deux Z-buffers à un pixel donné). Si cette profondeur est faible, alors l'intensité du pixel correspondant est augmentée pour simuler cette brillance. Pour 700,000 prismes et une résolution de $1024 \times 1024$, le temps de rendu est d'à peu près deux minutes sur une machine SGI Iris-4D 210 disposant d'une carte d'accélération graphique GTX capable de Z-buffer. Le résultat de la surbrillance est plaisant, mais l'aliassage du Z-buffer est fortement apparent. Notons aussi que cette technique est complètement *ad hoc* et ne prend pas la densité chevelue en considération. Par exemple, une seule mèche de cheveux concentrés sur une faible épaisseur laissera passer trop de lumière, alors que deux cheveux individuels éloignés mais alignés suffiront à la bloquer. La figure 2.3 illustre ces deux problèmes.

Anjyo *et al.* [AUK92] choisissent comme primitive de rendu des segments de droite tridimensionnels. Ils définissent un modèle de rendu simplifié où la couleur du cheveu est supposée toujours noire, où le coefficient diffus est négligé, et où l'ombrage n'est pas calculé. Puisque les segments de droites ne possèdent pas de volume, ils ne possèdent pas non plus de vecteur normal. Les segments sont alors considérés comme de très fins cylindres, et l'équation d'illumination suivante est dérivée selon le modèle de Blinn [Bli77] :

$$\begin{aligned} I(P) &= I_a k_a + I_s \Phi_s(P) \\ &= I_a k_a + I_s k_s (\mathbf{n} \cdot \mathbf{h})^n \\ &= I_a k_a + I_s k_s (1 - (\mathbf{t} \cdot \mathbf{h})^2)^{\frac{n}{2}} \end{aligned} \tag{2.2}$$

où $I_a$ est l'intensité de la lumière ambiente, $I_s$ l'intensité de la source lumineuse, $k_s$ le coefficient spéculaire, $\mathbf{n}$ le vecteur normal au point $P$, $\mathbf{h}$ le vecteur bisecteur entre la direction de la lumière

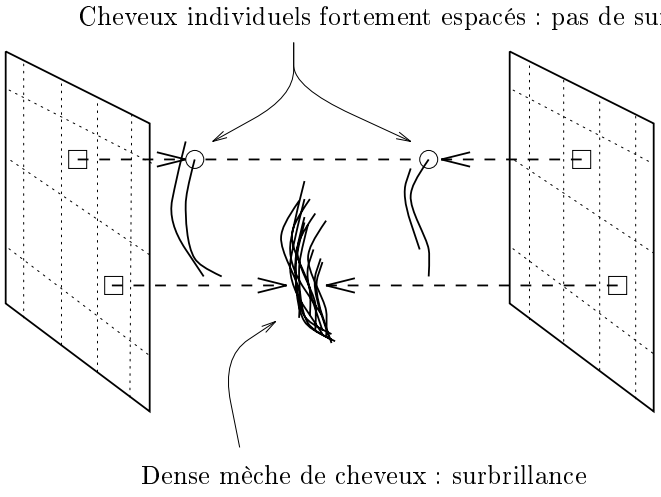


FIG. 2.3: Utilisation de deux Z-buffers parallèles et problèmes rencontrés, Watanabe *et al.* [WS92]

et la direction de vue au point $P$, et $\mathbf{t}$ la tangente au cylindre. Le coefficient de réflexion ambiente $k_a$ joue un rôle important dans la simulation de l'effet d'anisotropie de la chevelure : les auteurs utilisent une distribution normale pour faire varier aléatoirement sa valeur à chaque cheveu. Une fois de plus, le Z-buffer est utilisé comme technique de rendu. Le sur-échantillonnage est appliqué pour réduire l'aliassage. Pour 400,000 segments et une résolution de $1024 \times 1024$, le temps de rendu est d'à peu près 15 secondes sur une machine SGI Iris Power Series disposant d'une carte graphique VGX capable de Z-buffer, d'anti-aliassage et d'interpolation linéaire de couleurs.

Rosenblum *et al.* [RCI91] utilisent aussi un Z-buffer pour afficher les cylindres qui représentent les parties successives d'un cheveu. Ils utilisent le sur-échantillonnage ainsi que le *jittering* pour réduire l'aliassage. Pour calculer correctement l'ombrage, ils utilisent un tampon d'ombrage et évitent les artéfacts sur les contours de l'ombre en créant une pénombre artificielle, à la manière de Reeves *et al.* [RSC87]. Le modèle d'illumination utilisé est celui décrit par Kajiya *et al.* [KK89] pour un cylindre (décrit plus bas, équation 2.6). Sur une station graphique HP 9000/370, avec 16 échantillons par pixel et pour une résolution de $512 \times 512$, leur temps de calcul est de 12 minutes.

Daldegan *et al.* [DTKT93] utilisent le lancer de rayons pour effectuer le rendu de la chevelure. La procédure utilisée est la suivante :

- Un tampon d'ombrage (*shadow buffer*) est calculé pour chaque source lumineuse, pour la scène excluant la chevelure ;
- Un tampon d'ombrage est calculé pour chaque source lumineuse pour la chevelure uniquement ;
- Pour chaque lumière, les deux tampons sont composés en un nouveau tampon ;

- Le Z-buffer de la scène est calculé en utilisant ces nouveaux tampons d'ombrage, excluant la chevelure. La scène est alors affichée sans les cheveux ;
- Les segments de cheveu sont ajoutés à la scène, leur illumination étant calculée selon le modèle suivant :

$$I(P) = I_a k_a + \sum_{i \text{ lumières}} I_i S_i (k_d \sin\theta + k_s \cos^n(\phi + \theta - \pi)) \tag{2.3}$$

  où $S_i$ est un coefficient d'ombrage obtenu en filtrant le pixel au point $P$ avec la valeur du tampon d'ombrage de la source de lumière $i$, $\theta$ l'angle entre la tangente $\mathbf{t}$ au cylindre et la direction de la lumière, et $\phi$ l'angle entre $\mathbf{t}$ et la direction de la caméra.

La performance de cette méthode est rapportée dans [LTT91]. Pour une scène contenant 101,600 cheveux (810,000 segments), et des tampons d'ombrage de résolution $1000 \times 1000$, le temps d'affichage complet est d'à peu près 8 minutes, sur une machine SGI Iris Power Series disposant d'une carte graphique VGX, capable de Z-buffer, d'anti-aliassage et d'interpolation linéaire de couleurs. Le problème d'aliassage reste cependant important, surtout au niveau des ombres, qui apparaissent parfois dans des situations où la densité chevelue est faible.

Miller [Mil88] décrit un modèle d'illumination anisotropique qui peut être appliqué aux modèles géométriques de chevelure décrits plus haut. Son modèle est basé sur des surfaces dont l'anisotropie est fortement uni-directionnelle, c'est pourquoi des cylindres parallèles disposés sur la surface peuvent capturer cette anisotropie. Pour accélérer le rendu, une table de réflectance, indexée par la direction de la tangente à la surface, est créée pour calculer le coefficient de réflexion pour une direction donnée. La table est créée en calculant ce coefficient pour toutes les valeurs possibles de la normale à la surface. L'auteur étend l'algorithme du A-buffer [Car84] avec son modèle de cylindres pour afficher des objets en fil-de-fer correctement illuminés. Cette méthode est aussi appliquée au rendu de la fourrure.

#### 2.2.2.2 Sous-géométrie

Sourin *et al.* [SPS96] ne décrivent pas la technique de rendu qu'ils utilisent. Cependant, leur approche pouvant être considérée comme une extension de l'hypertexture de Perlin *et al.* [PH89], nous décrivons ici l'approche de ces derniers. Les auteurs utilisent le rendu volumique pour afficher leur hypertexture. Cette technique est $O(n^3)$ pour une résolution d'image de $n \times n$, et les auteurs suggèrent une implantation parallèle ou distribuée, qui sera optimale (d'après eux) car les fonctions de modulation sont indépendantes à chaque point d'échantillonnage. La méthode de *ray marching* est utilisée, elle procède comme suit :

- Pour chaque pixel, un rayon est lancé dans la scène ;
- Le rayon est tronqué contre le volume englobant de l'hypertexture. En cas d'intersection, les paramètres $t_0$ et $t_1$ d'entrée et de sortie du rayon sont calculés ;

- L'hypertexture est échantillonnée régulièrement sur le rayon entre $t_0$ et $t_1$ en calculant $f(t)$ à chaque échantillon, $f$ étant la fonction de modulation et $\Delta t$ le pas ;
- Si $0 < f(t) < 1$, alors $\nabla f(t)$ est calculé, puis normalisé pour obtenir la valeur de la normale utilisée dans le calcul de l'illumination ;
- A chaque échantillon $k$, l'opacité $\alpha$ est accumulée de la manière suivante :

$$\begin{aligned} \alpha_k &\leftarrow 1 - (1 - \rho_k)^{c\Delta t} \\ \alpha &\leftarrow \alpha + \alpha_k(1 - \alpha) \end{aligned}$$

  où $c$ est une constante de normalisation, $\alpha_k$ est l'opacité à l'échantillon et $\rho_k$ la densité du volume à l'échantillon. Si l'opacité excède 1, alors l'itération est terminée pour ce rayon.

L'aliassage est traité en imposant un seuil à la fréquence de l'hypertexture générée en fonction du pas $\Delta t$ utilisé. Les auteurs forcent le minimum de la période de $f(t)$ à demeurer égale à $\Delta t$. Pour une résolution de $512 \times 512$, le temps de rendu est de 3-15 heures sur une machine Sun 4-260, dépendamment de la complexité de la fonction de modulation utilisée.

Tel que nous l'avons mentionné plus haut, la notion de texels de Kajiya *et al.* [KK89] est une généralisation des volumes de densité. Chaque texel est un cube contenant l'information nécessaire pour calculer l'illumination sur le trajet d'un rayon le traversant. Plus précisément, le texel est défini comme un triplet $(\rho, \mathbf{B}, \Psi)$, où $\rho(x, y, z)$ est un champ de densités représentant l'aire relative projetée sur la paroi du texel dans la direction d'un rayon (prenant en considération l'occlusion de surfaces), $\mathbf{B} = [\mathbf{n}(x, y, z), \mathbf{t}(x, y, z), \mathbf{b}(x, y, z)]$ un champ de référentiels représentant l'orientation locale de la surface dans le texel, et $\Psi(x, y, z, \theta, \phi, \psi)$ un champ de fonctions de réflectance bidirectionnelle (BRDF). Les texels sont déformés (par une déformation trilinéaire) pour adhérer à la surface courbe d'un objet choisi. L'illumination est calculée par un algorithme de lancer de rayons modifié comme suit :

- Pour chaque pixel, un rayon est lancé dans la scène ;
- Le rayon est tronqué dans le texel. En cas d'intersection, les paramètres $t_0$ et $t_1$ d'entrée et de sortie du rayon sont calculés ;
- Le texel est échantillonné sur le trajet du rayon, en divisant le rayon en intervalles égaux et en choisissant aléatoirement un point sur chaque intervalle.
- L'atténuation $T$ à l'intérieur du texel est calculée comme suit :

$$T = e^{-\tau \sum_{s=t_0}^{t_1} \rho(x(s), y(s), z(s))} \tag{2.4}$$

  où $\tau$ sert à convertir la densité en une atténuation.

– La brillance $B$ émanant du texel tel que vue par le rayon est calculée comme suit :

$$\begin{aligned} B &= \sum_{t=t_0}^{t_1} [e^{-\tau \sum_{s=t_0}^{t} \rho(x(s),y(s),z(s))} \\ &\quad \times \sum_{i \text{ lumières}} I_i(x(t),y(t),z(t)) \Psi(x(t),y(t),z(t),\theta,\phi,\rho) \\ &\quad \times \rho(x(t),y(t),z(t))]. \end{aligned} \tag{2.5}$$

L'intensité $I_i$ pour chaque lumière est calculée récursivement en lançant un rayon de l'échantillon courant vers la lumière $i$. La brillance ainsi obtenue est multipliée par l'atténuation pour obtenir l'intensité.

– Le modèle d'illumination locale $\Psi$ est calculé comme suit :

$$\Psi = I_d k_d \sin(\mathbf{t},\mathbf{l}) + I_s k_s (\cos(\mathbf{t},\mathbf{l})\cos(\mathbf{t},\mathbf{d}) + \sin(\mathbf{t},\mathbf{l})\sin(\mathbf{t},\mathbf{d}))^n \tag{2.6}$$

où $\mathbf{t}$ est la tangente au cylindre, $\mathbf{l}$ la direction de la lumière, et $\mathbf{d}$ la direction du point de vue.

Pour effectuer le rendu de la chevelure, plusieurs simplifications peuvent être introduites. D'abord, le cheveu étant modélisé comme un cylindre infiniment fin, le système de référence (*frame*) est réduit à la tangente $\mathbf{t}(x,y,z)$. De plus, chaque cheveu possède la même orientation par rapport à la surface de l'objet, cette tangente est donc constante pour tout le volume. Le modèle d'illumination $\Psi$ est lui aussi constant pour l'ensemble de la chevelure. Chaque texel ne conserve donc que la densité $\rho$, qui équivaut à une occultation isotrope. Les auteurs modélisent en fait deux couches de fourrure, l'une plus dense que l'autre, pour obtenir un réalisme plus convaincant. Ces deux couches sont modélisées par deux distributions de Poisson sur la topologie du tore. Le temps de calcul pour une résolution de $1280 \times 1024$ est de 2 heures sur un réseau de 16 processeurs (12 IBM 3090 et 4 IBM 3081), la tâche recevant 30%-50% des processeurs, soit environ 12 heures CPU.

La texture volumique de Kajiya *et al.* [KK89] a généré de l'intérêt, grâce au fait qu'elle capture la complexité des surfaces en réduisant l'aliassage de façon importante. Shinya [Shi92] présente une extension importante à cette approche, en introduisant la notion de hiérarchie, de manière à représenter la réflectance des objets à plusieurs résolutions afin de réduire davantage l'aliassage. L'auteur décrit de façon sommaire les étapes nécessaires à effectuer la construction d'une texture volumique multi-échelle à partir d'une géométrie arbitraire, ainsi que son affichage. Plusieurs détails ne sont pas résolus cependant. D'abord, les termes d'atténuation $T$ (équation 2.4) et de brillance $B$ (équation 2.5) sont de la forme $e^{-\tau\rho}$, indiquant un modèle stochastique basé sur une distribution de Poisson. Rappelons-nous que ces termes estiment l'illumination selon la projection des surfaces incluses dans un texel sur une paroi du texel. Ce modèle stochastique, quoique satisfaisant pour des texels lointains, ne fonctionne plus correctement lorsqu'on s'en rapproche. De plus, il rend impossible de réunir géométrie et texture volumique en

un même objet. Noma [Nom95] propose une solution à ce problème en introduisant des termes d'atténuation et de brillance obtenus en échantillonnant les surfaces à l'intérieur du texel. Il utilise un genre de *MIP map* [1] [Wil83] 3D qui permet d'échantillonner les surfaces à plusieurs résolutions, dans un espace régulièrement subdivisé en cellules (un octree [Sam90b, Sam90a]). Il définit la transparence $T$ comme suit :

$$T = \prod_{k \text{ cellules}} (1 - \rho(x(t_k), y(t_k), z(t_k), \theta, \phi, d_k)) \tag{2.7}$$

où la densité $\rho$ dépend du point $(x, y, z)$, de la direction de vision $(\theta, \phi)$ et du diamètre d'échantillonnage $d_k$. En général, ce diamètre doit être proportionnel à $t_{k+1} - t_k$ pour capturer les informations dans la cellule $k$. La densité est conservée dans un *MIP map* 3D, et la valeur désirée est obtenue par interpolation trilinéaire. La brillance $B$ devient alors :

$$\begin{aligned} B \quad = \quad & \sum_{k} [\prod_{j=1}^{k-1} (1 - \rho(x(t_j), y(t_j), z(t_j), \theta, \phi, d_j)) \\ & \times \sum_{i \text{ lumières}} I_i(x(t_k), y(t_k), z(t_k)) \Psi(x(t_k), y(t_k), z(t_k), \theta, \phi, \psi) \\ & \times \rho(x(t_k), y(t_k), z(t_k), \theta, \phi, d_k)]. \end{aligned} \tag{2.8}$$

Notons que cette approche, comme celle de Kajiya *et al.* [KK89], nécessite la modification du modèle d'illumination $\Psi$ et de la densité $\rho$ en fonction du type d'objet et de la scène que nous désirons afficher. Notons aussi que la densité est synonyme d'opacité dans ce modèle, ce qui revient à dire que le matériau est considéré totalement opaque. L'auteur ne rapporte pas la performance de sa méthode.

Une autre approche hiérarchique pour le rendu de la texture volumique, bien plus générale celle-ci, est proposée par Neyret [Ney95b]. Son approche se base sur le filtrage de la réflectance, afin d'approximer la distribution de normales à l'intérieur d'un texel à plusieurs niveaux. En fait, la méthode décrite n'utilise qu'un seul texel de référence, texel qui est copié virtuellement[2] et déformé sur toute la surface de l'objet désiré. Le texel est encodé par un octree, dont chaque voxel conserve la densité locale et une micro-primitive représentant le modèle de réflectance. La surface complexe à modéliser est échantillonnée au niveau le plus fin de l'octree. Les niveaux supérieurs sont générés en filtrant les niveaux inférieurs, c'est-à-dire en créant des micro-primitives dont la distribution de normales est une moyenne des niveaux inférieurs. L'auteur choisit l'ellipsoïde comme micro-primitive. Son choix est guidé par le fait que l'ellipsoïde possède six degrés de liberté, et que la somme des distributions de normales de deux ellipsoïdes peut être approximée

[1] MIP : *multum in parvo*, latin signifiant "beaucoup dans peu". Le *MIP map* répresente une texture 2D à plusieurs résolutions à l'aide d'une pyramide où chaque niveau $i$ conserve la texture à une résolution deux fois plus basse que le niveau $i + 1$.

[2] Pour éviter de dupliquer inutilement les informations, chaque copie ne conserve qu'un pointeur vers le texel de référence.

(quoique grossièrement) par la distribution de normales d'une autre ellipsoïde. L'algorithme de rendu de Kajiya *et al.* [KK89] est alors modifié pour accommoder la nouvelle structure des texels : le lancer de cônes est utilisé pour choisir le niveau approprié de l'octree (à la manière d'un *MIP map* 3D), suivant la projection inverse du pixel sur la surface intersectée. L'auteur obtient ainsi un modèle de réflectance général qui est adapté aux textures répétitives, qui offre une anisotropie simple (modélisée par la direction des ellipsoïdes), et dont la performance est bonne : pour une résolution de texels de $256^3$, contenant chacun 2000 sphères, et $50 \times 500$ texels affichés (c'est-à-dire l'équivalent de 50 millions de sphères), le temps de calcul est de 14 minutes sur une machine SGI Indigo, et l'image résultante exhibe très peu d'aliassage. Neyret poursuit son approche dans plusieurs autres rapports [Ney95a] [Ney95c] [Ney96a], où il améliore son modèle d'illumination et introduit les concepts pertinents à l'animation de texels, que nous verrons plus bas. Cependant, la grande efficacité de cette méthode provient du fait que seul un volume de référence est créé, les texels étant des copies déformées de ce volume.

### 2.2.3 Animation

Seules les approches purement géométriques décrivent une méthode générale pour animer la chevelure. Watanabe *et al.* [WS92] utilisent leur modèle de prismes trigonaux pour approximer le mouvement de chaque cheveu en fonction du vent ou du mouvement de la tête. Ils décrivent très brièvement leur méthode qui utilise des vecteurs d'accélération (incluant la gravité) et une trajectoire parabolique.

Anjyo *et al.* [AUK92] et Daldegan *et al.* [DTKT93] tentent de se rapprocher de la physique. Ils introduisent un modèle d'équations différentielles de moments angulaires uni-dimensionnels. Soit un cheveu modélisé par une série de $k$ segments droits $s_i, (1 \leq i \leq k)$ où $P_i$ sont les extrémités des segments et $P_0$ la position du cheveu sur le scalp. Seuls les angles et les moments de torsion entre les segments sont considérés. Un système de coordonnées polaires est utilisé. Le système obtenu est le suivant :

$$\begin{aligned} I_i \frac{\mathrm{d}^2\theta_i}{\mathrm{d}t^2} + \gamma_i \frac{\mathrm{d}\theta_i}{\mathrm{d}t} &= M_\theta \\ I_i \frac{\mathrm{d}^2\phi_i}{\mathrm{d}t^2} + \gamma_i \frac{\mathrm{d}\phi_i}{\mathrm{d}t} &= M_\phi \end{aligned} \tag{2.9}$$

où $I_i$ est le moment d'inertie du segment $s_i$, $\gamma_i$ le coefficient d'amortissement (*damping*), $M_\theta$ et $M_\phi$ les composantes du moment de torsion. Ces dernières sont définies comme suit :

$$\begin{aligned} M_\theta &= -k_\theta(\theta - \theta_0) + M_{\theta,\text{externe}} \\ M_\phi &= -k_\phi(\phi - \phi_0) + M_{\phi,\text{externe}} \end{aligned} \tag{2.10}$$

où le premier terme dans chaque équation représente le moment dû à l'effet de ressort, et le second terme représente le moment dû aux forces externes telles que la gravité, l'inertie et le vent.

Un problème à régler ici est celui de la détection d'inter-collisions entre les cheveux, et des collisions avec le corps humain. Les inter-collisions sont très coûteuses à calculer, c'est pourquoi toutes les approches revues ici n'en traitent pas, et se concentrent plutôt sur les collisions avec le corps. Anjyo *et al.* [AUK92] utilisent l'heuristique suivante durant l'application des forces aux segments d'un cheveu : Soit une force externe $\mathbf{F}$ et un point $P_i$ calculé durant la résolution des équations 2.9. La direction $\mathbf{d}_i$ du segment $s_i$ est calculée par rapport à l'ellipsoïde du crâne, en utilisant les dérivées partielles de la forme quadrique de l'ellipsoïde :

$$\mathbf{d}_i = (E_x^{'}(P_i), E_y^{'}(P_i), E_z^{'}(P_i)).$$

Si $\mathbf{d}_i \cdot \mathbf{F} < \alpha \|\mathbf{d}_i\| \|\mathbf{F}\|$, où $|\alpha| < 1$, et que $P_i$ soit près du crâne, alors la force $\mathbf{F}$ est atténuée par des facteurs pré-déterminés $0.0 < \epsilon_i < 1.0$, ces facteurs approchant 0 pour des segments connectés au crâne, et approchant 1 pour les extrémités libres. Les auteurs considèrent la nouvelle force $\epsilon_i \mathbf{F}$ comme la résultante de $\mathbf{F}$ avec une force répulsive.

Rosenblum *et al.* [RCI91] créent un modèle d'animation simplifié qui ne traite que des cheveux qui sont originellement droits. Chaque segment de cheveu (un cylindre droit) est représenté par deux masses ponctuelles reliées par un ressort rigide (les cheveux étant considérés incompressibles). De plus, une charnière (*hinge*) est placée entre chaque segment de cheveu, tel que l'illustre la figure 2.4. La force du ressort $r$ est calculée comme suit :

$$F_r = k_r x - D_r(k_r x) \tag{2.11}$$

où $k_r$ est la constante du ressort, $D_r$ est une constante de dissipation et $x$ le déplacement dans la direction du segment. La force de la charnière $c$ est calculée comme suit :

$$F_c = k_c \theta - D_c(k_c \theta) \tag{2.12}$$

où $\theta$ est le déplacement angulaire de la charnière entre les deux segments. Cette force est appliquée aux masses externes des segments reliés de manière à les aligner. De plus, la force de gravité $F_g = mg$ et une force de traînée globale $F_d = vD_d$ sont appliquées à chaque masse $m$ de vitesse $v$, où $g$ est l'accélération due à la gravité et $D_d$ une constante de traînée. A chaque étape de l'animation, l'intervalle de temps correspondant est subdivisé, les forces sont calculées pour chaque sous-intervalle, et les positions des masses ponctuelles (c'est-à-dire des segments) sont modifiées en conséquence. Ce modèle introduit plusieurs simplifications, notamment au niveau de la charnière $c$, où les positions des masses externes n'entrent pas dans le calcul de $F_c$, et au niveau des constantes de dissipation $D_r$ et $D_c$, dont les valeurs sont arbitraires.

Daldegan *et al.* [DTKT93] traitent la collision avec le corps entier, ayant créé une représentation cylindrique discrétisée du corps au préalable. La détection de collision procède comme suit : Chaque point $P_i$ est transformé dans le système de coordonnées cylindriques du corps pour obtenir $(r_{P_i}, \theta_{P_i}, y_{P_i})$. Un point $Q$ du corps est trouvé, tel que $P_i$ et $Q$ correspondent par l'azimuth $\theta$ et la hauteur $y$. Une fois le rayon $r_Q$ trouvé par interpolation linéaire, les deux

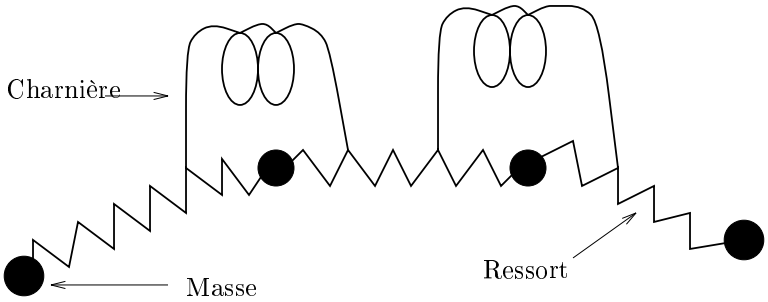


FIG. 2.4: Modèle dynamique d'un cheveu, Rosenblum *et al.* [RCI91]

rayons sont comparés pour déterminer si $P_i$ est à l'intérieur du corps ou non. Si oui, le point $T$ le plus proche de $P_i$ sur la surface du corps est calculé par interpolation linéaire, et une force répulsive peut être alors calculée pour bouger $P_i$ à $T$. Soit une force externe $\mathbf{F}$, cette force est décomposée en deux composantes, l'une tangente au corps au point $T$ et l'autre normale à ce point. La première ne joue aucun rôle dans la répulsion, elle est obtenue ainsi :

$$\mathbf{F}_{\text{tan}} = \mathbf{F} - (\mathbf{F} \cdot \mathbf{n})\mathbf{n} \tag{2.13}$$

où $\mathbf{n}$ est la normale au corps au point $T$. La force répulsive dans la direction de la normale est obtenue ainsi :

$$\mathbf{F}_{\text{rep}} = -(kP_i \cdot T + c\mathbf{v}_i \cdot \mathbf{n})\mathbf{n} \tag{2.14}$$

où $\mathbf{v}_i$ est la vitesse du point $P_i$, $k$ est une constante représentant l'importance de la contrainte, et $c$ le coefficient d'amortissement (*damping*). La force répulsive totale est obtenue par l'addition des deux forces $\mathbf{F}_{\text{tan}}$ et $\mathbf{F}_{\text{rep}}$.

Neyret [Ney95a] décrit un modèle d'animation simple pour ses texels. Rappelons que les texels sont utilisés pour "coiffer" une surface, ce modèle d'animation est donc fortement lié à la notion de peau volumique. Dans le cas où la texture est localement immobile, l'animation de la surface sous-jacente est suffisante pour donner l'illusion du mouvement de la texture. Sinon, il est possible d'animer les hauteurs des texels sous l'effet de champs de vitesse ou de force, par exemple pour animer des herbes dans le vent. Dans ce cas, les informations contenues dans le texel ne sont pas affectées, seule la déformation du texel est modifiée en fonction de l'orientation de ses hauteurs. Enfin, dans le cas où la géométrie contenue dans le texel est modifiée, il est possible, quoique très coûteux, de recalculer les texels représentant les étapes successives de l'animation.

## 2.3 Description générale de notre méthode

Notre objectif essentiel est de pouvoir afficher une scène complexe. Le rendu en temps réel n'est pas un objectif mais nous désirons tout de même maintenir une performance raisonnable, afin notamment de permettre la production de séquences d'images. Nous avons choisi de suivre l'approche de Neyret [Ney95a, Ney95b, Ney95c, Ney96a, Ney96b] parce qu'elle offre plusieurs avantages par rapport aux autres approches revues ici.

D'abord, elle utilise comme base la texture volumique introduite par Kajiya *et al.* [KK89]. Comme nous l'avons décrit plus haut, cette texture volumique ne conserve que les informations de réflectance et d'opacité là où les détails géométriques deviennent très complexes et donc coûteux à afficher, d'une part à cause des nombreuses primitives à traiter, et d'autre part à cause du sur-échantillonnage agressif nécessaire pour réduire suffisamment l'aliassage. Cette représentation permet d'effectuer un rendu volumique de la texture, c'est-à-dire en évaluant l'intégrale de l'illumination sur le trajet d'un rayon parcourant la texture. L'idée ici est de pré-calculer l'information de réflectance afin que le rendu puisse être effectué avec le minimum possible de sur-échantillonnage. De plus, une fois la texture pré-calculée, le coût de la traverser sera à peu près indépendant de la complexité de la géométrie originale.

L'approche de Neyret améliore celle de Kajiya *et al.* [KK89] en introduisant l'idée de texture volumique multi-échelle, qui permet de choisir la résolution appropriée durant le rendu, afin de réduire encore plus l'aliassage. En fait, Neyret n'utilise qu'un seul rayon par pixel pour afficher sa texture volumique. De plus, la fonction d'occultation $\rho$ utilisée par Neyret est anisotropique tandis que celle de Kajiya est équivalente à la densité du voxel (et donc isotropique).

L'approche multi-résolution est aussi proposée par Noma [Nom95], mais celui-ci crée un modèle de réflectance spécifique au type de scène qu'il modélise. Cette spécialisation pose de sérieuses limitations quant à l'applicabilité de son modèle. En effet, pour chaque configuration géométrique différente, le modèle de réflectance diffèrera. Noma analyse chaque type de scène pour dériver une approximation statistique de la réflectance. Cette approche est en contradiction avec l'un de nos objectifs qui est de maintenir un certain degré de généralité dans notre méthode, ce qui implique la possibilité de représenter un grand nombre de scènes en utilisant la même technique. Bien sûr, nous voulons appliquer notre technique à la simulation de la chevelure, qui est un domaine bien spécifique. Cependant, nous pensons qu'une analyse statistique globale de la chevelure serait insuffisante pour représenter tous les types de cheveux et toutes les configurations géométriques possibles. C'est bien pourquoi les deux scènes décrites par Noma consistent simplement de sphères, et de feuilles d'arbres.

Notre approche est donc globalement similaire à celle de Neyret. Cependant, la mise en œuvre sera différente, nous le verrons en détail dans les prochains chapitres. Pour le moment, contentons-nous de décrire de manière générale les étapes de notre méthode (voir figure 2.5).

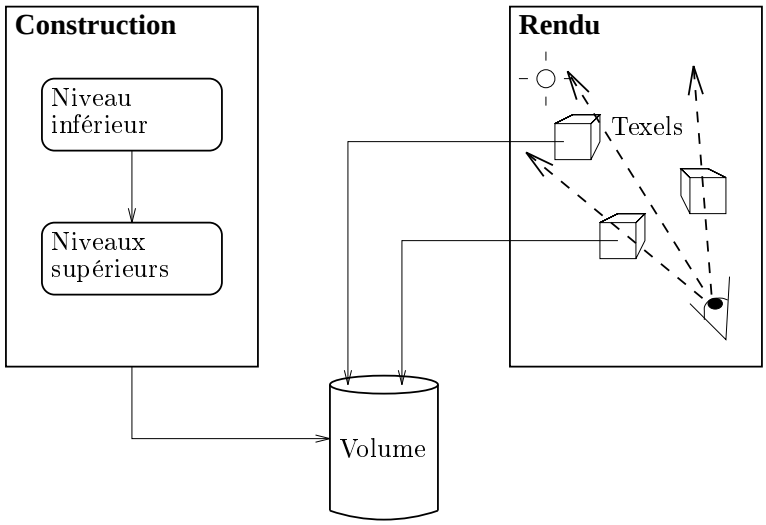


FIG. 2.5: Etapes de la méthode.

### 2.3.1 Construction des texels

L'objectif ici est de créer un *volume de référence* à partir d'une liste d'objets. Cette étape s'effectue au pré-calcul, elle permet de représenter ces objets comme une texture volumique qui sera parcourue durant le rendu. Comme Neyret, nous créons un octree dont les *voxels*-feuilles contiennent l'information voulue des objets donnés. Nous avons choisi l'octree [Sam90b, Sam90a] comme structure de subdivision spatiale hiérarchique. Cette structure est légèrement moins performante que d'autres, telles que le BSP-tree [FKN80, FAG83], mais elle se distingue par sa simplicité de conception et d'implantation.

Les niveaux supérieurs de l'octree contiennent des versions plus grossières de l'information calculée aux voxels-feuilles, obtenues par filtrage. Enfin, le volume est sauvegardé sur fichier, pour être utilisé durant le rendu.

Il faut déterminer les informations importantes à conserver dans chaque voxel. Nous en discutons ici, mais nous ne traitons pas des moyens de représentation ou d'approximation de ces informations, nous citons uniquement celles qui sont essentielles pour effectuer le rendu volumique.

**La réflectance des objets** Pour chaque point sur la surface des objets contenus dans un voxel, une fonction de distribution de réflectance bidirectionnelle (BRDF [TS67]) est définie. Cette fonction permet de décider de l'intensité lumineuse réfléchie dans chaque direction, pour chaque direction incidente.

**La visibilité dans le voxel** Pour une certaine direction de vue, certaines surfaces en cacheront d'autres dans le voxel, et nous voulons ne prendre en compte que celles qui sont visibles pour cette direction.

**Les matériaux des objets** Calculer l'illumination en un point du voxel revient à évaluer un modèle d'illumination locale en ce point, modèle qui utilise typiquement les propriétés de réflectance du matériau en ce point, telles que la couleur, la rugosité, la transparence, etc.

### 2.3.2 Rendu des texels

Nous effectuons le rendu de notre texture volumique dans le contexte d'un algorithme de lancer de rayons classique. Etant donné un volume de référence pré-calculé $V$, nous plaçons des *texels* dans la scène, texels qui sont des copies virtuelles déformées de ce volume. Ainsi, un seul volume de référence peut donner naissance à plusieurs formes différentes selon la déformation des texels. Un texel est donc représenté par la structure suivante :

```
struct Texel
{
  Volume* V ;
  Déformation D ;
} ;
```

Quand un rayon intersecte un texel, un algorithme de rendu volumique est appliqué à ce texel. Le rayon est alors considéré comme un cylindre, dont le diamètre varie inversement avec la distance du point de vue, et sur le trajet duquel nous voulons intégrer l'illumination, à la manière de Kajiya *et al.* [KK89]. Le diamètre du rayon nous permettra de choisir quel niveau de l'octree utiliser pour le calcul, dans l'esprit d'un *MIP map* 3D. Nous traversons alors l'octree de part en part, accumulant l'illumination calculée dans chaque voxel traversé, et nous arrêtant une fois que l'opacité cumulée des voxels devient totale.

A l'intérieur d'un voxel, l'illumination est calculée par intégration du modèle d'illumination locale (tel que celui de Phong [Pho75]) sur la distribution des normales à l'intérieur du voxel. Le résultat est ensuite pondéré par l'opacité accumulée jusqu'ici.

## 2.4 Application à la modélisation de la chevelure

Nous désirons appliquer notre texture volumique à la modélisation de la chevelure. Nous proposons au chapitre 5 un modèle de chevelure basé sur le concept de *mèche*, une mèche étant formée d'une chaîne de texels déformés. Ce modèle permet de représenter économiquement une importante géométrie (l'ensemble des cheveux de la mèche) en utilisant un nombre limité de texels. Nous verrons aussi l'ébauche d'une méthode d'animation adaptée à ce modèle, et nous discuterons des simplifications que nous pouvons apporter à notre méthode pour traiter spécifiquement de la chevelure. Ces simplifications se situent surtout au niveau de la construction du volume de référence, où les informations spécifiques à la chevelure peuvent être plus concises que les informations générales décrites plus haut.

# Chapitre 3

# Construction des texels

*Nous décrivons dans ce chapitre la méthode de construction des texels, introduite au chapitre précédent. Nous mettons en valeur les différentes options qui se sont présentées à chaque étape, et nous justifions les choix effectués.*

## 3.1 Algorithme

Etant donnée une liste d'objets, cette phase construit le volume de référence et le sauvegarde dans un fichier. Cette construction se fait en deux étapes : la création des informations à haute résolution, c'est-à-dire dans les voxels-feuilles, et la propagation de ces informations aux niveaux supérieurs de l'octree.

Nous construisons un octree $O$ contenant les objets donnés dans une liste $L$. Nous commençons par trouver la boîte englobante $B$ des objets de $L$. Ayant choisi un niveau maximum de subdivision pour l'octree, nous créons le voxel racine $V$ et nous appelons la procédure de construction récursive `AjouterObjetsAuxVoxels(` $V$`,` $L$`,` $B$`, Niveau` $n$`=0 )`.

```
AjouterObjetsAuxVoxels( Voxel V, ListeObjets L, Boîte B, Niveau n )
{
  ListeObjets NL ;
  pour ( chaque Objet O dans L )
  {
    si ( PositionBoîte( O, B ) n'est pas à l'extérieur )
    {
      Ajouter( O, NL ) ;
    }
  }
  si ( NL est vide ) retour ;
  si ( n est maximum )
```

```
  {
    ConstruireInformation( V, NL ) ;
  }
  sinon
  {
    de ( i = 1 à 8 )
    {
      Voxel V_i = nouvel enfant de V ;
      Boîte B_i = CalculerBoîte( B, i ) ;
      AjouterObjetsAuxVoxels( V_i, NL, B_i, n + 1 ) ;
    }
    si ( V_1,...,V_8 sont vides )
    {
      détruire V_1,...,V_8 ;
      marquer V = Vide ;
      retour ;
    }
    si ( V_1,...,V_8 sont opaques )
    {
      marquer V = Opaque ;
      retour ;
    }
    MoyennerInformationEnfants( V ) ;
  }
}
```

## 3.2 Intersection des objets avec les voxels

Notons que l'algorithme `AjouterObjetsAuxVoxels` ne spécifie pas comment les objets sont intersectés avec la boîte englobante d'un voxel. Nous faisons la distinction entre les objets surfaciques (plan, surface de Bézier, etc.) et les objets volumiques (sphère, cube, etc.). Les premiers peuvent seulement intersecter la boîte, alors que les seconds peuvent aussi l'englober. L'intersection est la responsibilité de l'objet lui-même, qui fournit une fonction pour calculer la position d'une boîte englobante par rapport à sa surface (soit la boîte est à l'extérieur de la surface, soit elle est complètement à l'intérieur, soit elle contient une partie ou la totalité de la surface, voir figure 3.1). Par exemple, le processus d'intersection entre une sphère et une boîte cubique est simplement le suivant :

```
fonction PositionBoîte( Sphère S, Boîte B ) : Position
{
  Vecteur v = Centre( B ) - Centre( S );
  Float d = ||v|| - Rayon( S );
  Float s = Arête( B );
  Float r = sqrt(3s²/4);
  si ( d > r ) retour Vide;
  si ( d < 0 et -d > r ) retour Opaque;
  retour Frontière;
}
```

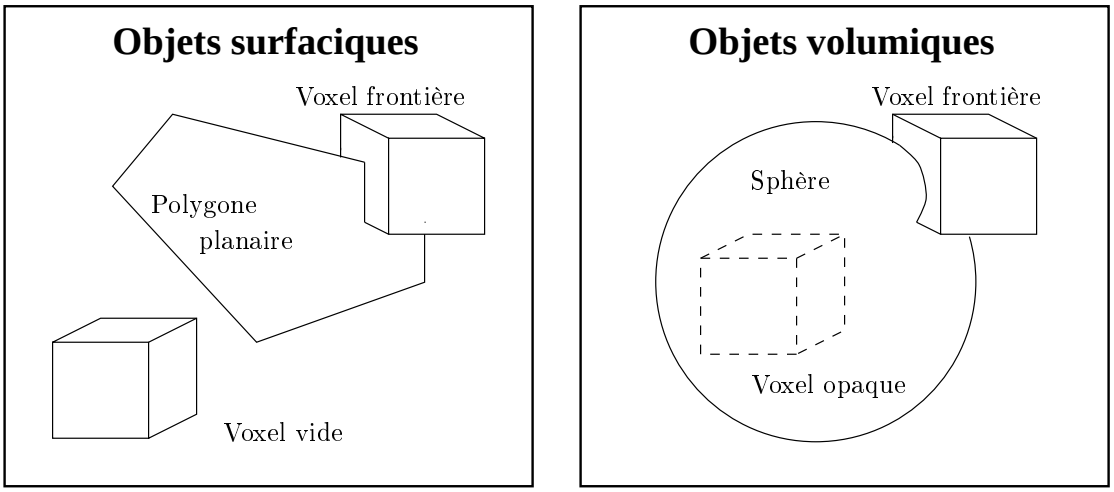


FIG. 3.1: Intersection des voxels avec les objets.

Notons que la fonction `PositionBoîte` ne doit pas nécessairement donner la position exacte de la boîte englobante par rapport à l'objet. En effet, cette position est parfois très difficile à calculer ; la fonction peut toujours répondre, de manière prudente, `Frontière`, afin de forcer l'algorithme à la récursion. L'octree résultant demeurera cependant correct, car nous vérifions après chaque récursion que les enfants ne sont pas tous vides (auquel cas ils peuvent être détruits) ou opaques (auquel cas le voxel courant est marqué opaque aussi). Le seul inconvénient sera un temps de calcul plus important.

## 3.3 Informations calculées dans les voxels

L'algorithme ne spécifie pas non plus quelles sont les informations construites dans chaque voxel. Les fonctions `ConstruireInformation` et `MoyennerInformationEnfants` se chargent de cela. La structure du voxel est représentée de la manière suivante :

```
struct Voxel
{
  NDF n(θ, φ) ;
  Vecteur v ;
  Matériau M ;
} ;
```

### 3.3.1 Fonction de distribution des normales $\mathbf{n}(\theta, \phi)$

Nous allons approximer la distribution de normales du voxel $V$ par une NDF (*normal distribution function* : fonction de distribution de normales) positionnée en son centre. Cette NDF est construite à partir d'un échantillonnage du voxel. Pour chaque paire de faces opposées, nous lançons un certain nombre de rayons aléatoirement générés d'une face à l'autre. Pour chaque rayon $R$, nous déterminons si un objet $O$ de $NL$ intersecte $R$. Dans ce cas, nous trouvons les points d'intersection ainsi que la normale $\mathbf{n}$ à l'objet $O$ aux premier et dernier points. Une fois tous les rayons lancés, nous obtenons une série de normales qui approxime notre NDF. Il s'agit alors de l'encoder d'une manière compacte. La figure 3.2 illustre ce processus de construction en 2D.

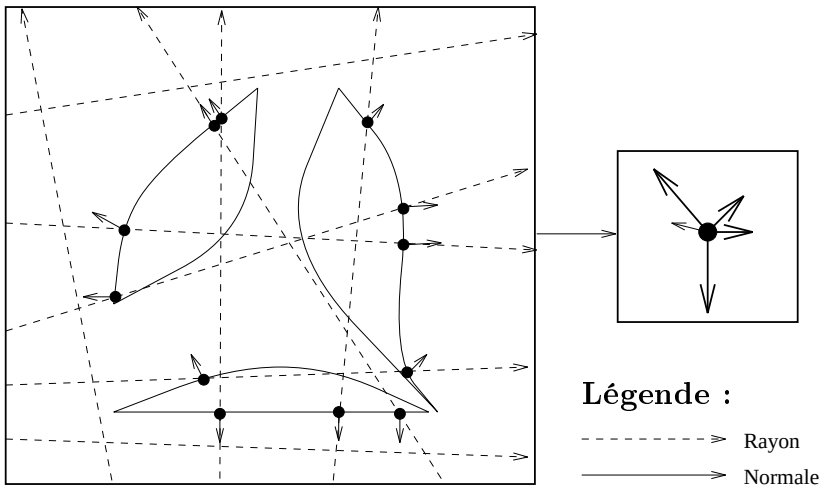


FIG. 3.2: Construction de la NDF.

Notons que si un voxel est complètement à l'intérieur d'un objet, nous ne pouvons plus parler de réflectance pour ce voxel, car la normale à l'objet n'est pas définie à l'intérieur de sa surface. Il sera donc nécessaire de marquer ces voxels comme opaques ; leur méthode de rendu sera différente, comme nous le verrons au chapitre suivant.

Nous décrivons maintenant le mode de représentation de la NDF. Nous voulons souligner que cette représentation est plus ou moins indépendante de la construction, mais qu'elle doit répondre aux critères suivants :

1. Nous désirons une représentation compacte : la NDF devra être conservée dans tous les voxels remplis de l'octree, soit un maximum de $2^{3n+3} - 1/2^3 - 1$ pour un octree de dimensions $n \times n \times n$.
2. Nous désirons une représentation assez puissante pour comporter des normales dans toutes les directions, puisque ce cas pourrait facilement se présenter dans un voxel arbitraire. De même, la représentation ne devrait pas créer des directions qui n'existent pas dans la NDF originale.
3. Nous désirons une représentation pour laquelle l'opérateur d'addition est défini, ce qui nous permettra d'additionner les fonctions de NDF pendant la construction, et de les filtrer pour créer les niveaux supérieurs de l'octree.
4. Nous désirons une représentation que nous puissions facilement manipuler ; les opérations que nous effectuerons seront typiquement l'évaluation de l'intégrale de modèles d'illumination locale sur la NDF.

La représentation de la NDF et de la BRDF a déjà été étudiée en infographie. Par exemple, Fournier [Fou92] représente une BRDF complexe comme une combinaison linéaire de fonctions de NDF simples et de fonctions de réflectances simples, considérées comme des bases dans l'espace de la BRDF. Fournier choisit comme bases des fonctions de la forme $\cos^n \theta$ (cette fonction est utilisée pour représenter le terme spéculaire dans le modèle de Phong [Pho75], d'où la désignation "pics de Phong"). La NDF peut aussi être représentée de la même manière :

$$\mathbf{n}(\theta, \phi) = \sum_i c_i \times (\mathbf{N}(\theta_i, \phi_i) \cdot \mathbf{H}(\theta_i, \phi_i, \theta, \phi))^{n_i} \tag{3.1}$$

où $\mathbf{N}(\theta_i, \phi_i) \cdot \mathbf{H}(\theta_i, \phi_i, \theta, \phi)$ représentent les bases sinusoïdales. Pour trouver les coefficients $c_i$, Fournier propose un processus de minimisation aux moindres carrés, sans en donner les détails. Un inconvénient de cette représentation dans notre cas est que l'opérateur d'addition n'est pas défini pour ces fonctions, ce qui rend plus difficile le processus de filtrage des informations.

Une représentation hiérarchique de fonctions sphériques est proposée par Schröder et Sweldens [SS95]. Les auteurs construisent une nouvelle famille d'ondelettes adaptée à la géométrie de la sphère. Ces ondelettes permettent d'effectuer une analyse multi-résolution de fonctions sur la sphère ; une des applications décrites par les auteurs est la représentation de la BRDF. Cependant, un très grand nombre de coefficients (2000-6000) est nécessaire pour obtenir de bons résultats. Il est à noter que représenter une NDF est beaucoup moins coûteux qu'une BRDF, puisque nous passons de quatre dimensions à deux.

Un autre outil d'analyse spécifiquement adapté à la sphère est le développement en harmoniques sphériques. Cette méthode est utilisée par Sillion et Puech [SP94] pour représenter la BRDF $\rho(\theta_i, \phi_i, \theta_o, \phi_o)$ dans le contexte de l'illumination globale. En particulier, les auteurs décrivent le processus d'approximation d'une BRDF donnée à l'aide d'harmoniques sphériques, puis de son utilisation dans le calcul de la radiance $L$ à une patche $k$ dans une direction donnée

$(\theta_0, \phi_0)$ :

$$L_k(\theta_0, \phi_0) = \sum_{i=1}^{n} \Phi_i \rho_k(\theta_0, \phi_0, \theta_i, \phi_i) \tag{3.2}$$

où $\Phi_i$ est l'intensité lumineuse de la source $i$. Notre cas est similaire. Il serait possible d'approximer notre NDF par des harmoniques sphériques ; nous devrions ensuite calculer l'intégrale de cette NDF sur toutes les directions $(\theta, \phi)$, un peu à la manière du calcul de la radiance. Les harmoniques sphériques souffrent cependant d'un support global, ainsi que d'un coût important de calcul. Cependant, elles permettent facilement le filtrage parce que l'opérateur d'addition est défini pour ces fonctions.

Neyret [Ney95b] propose une représentation très grossière de la NDF en utilisant un ellipsoïde comme support des normales, c'est-à-dire que la NDF est représentée par les normales à la surface de l'ellipsoïde. L'avantage de cette représentation est qu'elle est très compacte (un ellipsoïde peut être considéré comme une sphère canonique transformée par un facteur d'échelle suivi d'une rotation arbitraire ; trois vecteurs suffisent donc à définir l'ellipsoïde). De plus, des normales de toutes directions peuvent être représentées. C'est pourquoi notre premier choix pour la représentation de la NDF s'est porté sur l'ellipsoïde. Cependant, notre approche diffère de celle de Neyret dans plusieurs aspects.

D'abord, nous avons choisi de considérer l'ellipsoïde comme la NDF elle-même et non pas comme support de la NDF, c'est-à-dire que la distance entre chaque point $P$ à la surface de l'ellipsoïde et le centre $C$ représente la densité de normales dans la direction $P-C$ (voir figure 3.3 pour une illustration en 2D). Cette représentation a l'avantage d'être beaucoup plus simple que celle choisie par Neyret, autant pour la construction de la NDF que pour sa manipulation durant le rendu.

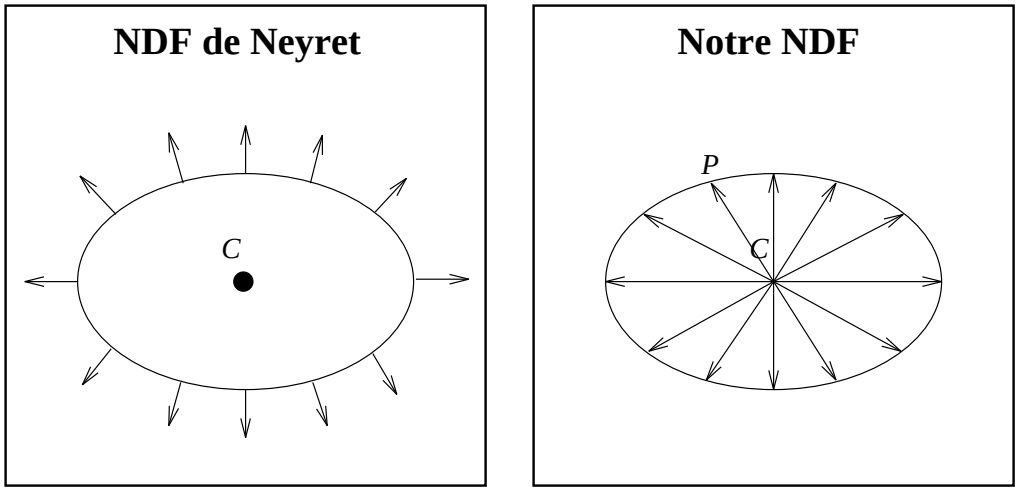


FIG. 3.3: Différence de représentation entre la NDF de Neyret et la nôtre.

Neyret [Ney95b] utilise une heuristique très simple pour construire un ellipsoïde représentant

la NDF. A l'intérieur d'un voxel, le gradient de la fonction de distance à l'objet fournit une normale qui représente l'axe principal de l'ellipsoïde. Les deux autres axes sont définis par l'utilisateur qui spécifie un "galbe". Cette méthode de construction ne fournit de résultats corrects que si la plupart des surfaces à l'intérieur du voxel pointent dans la même direction ; sinon l'approximation sera excessivement grossière. De plus, les normales cachées par d'autres surfaces contribueront quand même à la NDF finale.

Nous avons choisi d'échantillonner le voxel pour obtenir non pas une seule normale, mais un ensemble de normales, et uniquement celles qui sont visibles de l'extérieur du voxel. De plus, nous voulons spécifier l'ellipsoïde de façon entièrement automatique. Plusieurs solutions ont été étudiées. Une minimisation aux moindres carrés pour ajuster les normales à la surface de l'ellipsoïde [Pra87] fournit la solution la plus exacte. La fonction à minimiser est la forme quadratique de l'ellipsoïde centré autour de l'origine :

$$E(x, y, z) := ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz = 0 \tag{3.3}$$

Une fois les coefficients $(a, b, c, d, e, f)$ trouvés par la méthode des moindres carrés, les axes de l'ellipsoïde peuvent être obtenus en multipliant les valeurs propres $\lambda_i$ par les vecteurs propres unitaires $\mathbf{v}_i$ de la matrice $\mathbf{Q}$ associée à la forme quadratique

$$\mathbf{Q} = \begin{pmatrix} a & d & f \\ d & b & e \\ f & e & c \end{pmatrix} \tag{3.4}$$

Nous utilisons pour le moment une approximation heuristique de cette minimisation, heuristique qui est cependant moins grossière que celle employée par Neyret. Nous discrétisons les directions $(\theta, \phi)$ en $n \times n$ cases (*bins*), puis nous ajoutons chaque normale échantillonnée à la case qui lui correspond. La case qui contient le plus grand nombre de normales est choisie comme axe principal de l'ellipsoïde. Ensuite, la case de cardinalité maximum et qui est orthogonale au premier axe est choisie comme second axe. Enfin, le troisième axe est le produit vectoriel des deux premiers. L'ellipsoïde ainsi obtenu représente la densité des normales dans toutes les directions : la distance entre chaque point $P$ de la surface et le centre $C$ de l'ellipsoïde définit la densité des normales dans la direction $P - C$.

L'ellipsoïde présente toutefois une limitation apparemment sévère : il est symétrique autour de son centre. Cela signifie que toute normale dans une direction $\mathbf{d}$ a la même importance que la normale dans la direction $-\mathbf{d}$. Par conséquent, une NDF ellipsoïdale ne peut pas représenter la normale d'un plan par exemple. Dans la figure 3.4 de gauche, deux polygones planaires sont parallèles au plan de vue, et leurs normales pointent dans des directions opposées. Le polygone de gauche est illuminé car sa normale pointe dans la direction du point de vue, alors que celui de droite ne l'est pas. La figure 3.4 de droite contient le texel construit à partir de ces deux polygones, du même point de vue. Nous voyons que les deux plans sont illuminés, à cause de

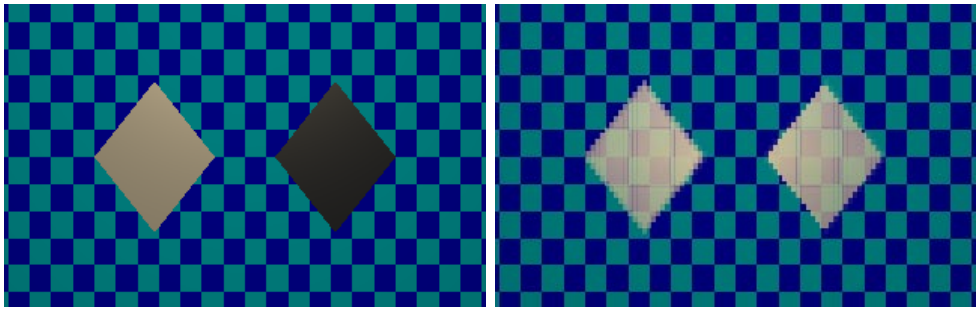

FIG. 3.4: Le texel construit à partir des polygones exhibe une erreur due à la symétrie de la représentation de la NDF.

la symétrie des normales de la NDF. Il pourrait sembler que cette limitation rende invalide l'utilisation de l'ellipsoïde comme NDF, et cependant ce n'est pas le cas. En effet, dans le cas des objets solides, les normales "complémentaires" aux normales véritables des objets pointeront vers l'intérieur du volume, qui est toujours invisible puisque les objets sont considérés pleins. En pratique donc, ce défaut apparaît surtout pour les objets surfaciques et dans les régions où l'opacité des voxels est faible, comme sur les bords des objets volumiques.

Nous avons aussi besoin de pouvoir filtrer notre NDF. Le filtrage s'effectue d'une manière similaire au processus d'ajustement décrit plus haut. Etant donnée une liste de fonctions de NDF ellipsoïdales, nous discrétisons à nouveau les directions $(\theta, \phi)$ en $n \times n$ cases, puis nous échantillonnons chaque ellipsoïde de manière à ajouter chaque échantillon à la case qui correspond à sa direction. Nous répétons le processus d'ajustement sur la discrétisation obtenue. Le résultat nous fournit alors un ellipsoïde approximant la somme des ellipsoïdes d'entrée.

### 3.3.2 Visibilité v

Nous conservons une information sur la visibilité axiale, c'est-à-dire la visibilité dans les trois axes principaux. Cette information est obtenue grâce à notre échantillonnage. En effet, chaque voxel étant lui-même aligné sur les axes (puisque la boîte englobante de l'octree est alignée), les rayons lancés à partir de chaque face du voxel sont associés à un axe principal. Pour chaque face, le rapport entre les rayons ayant intersecté un quelconque objet et le nombre total de rayons lancés approxime la visibilité pour cette face. Par exemple, dans la figure 3.2, le vecteur visibilité (en 2D) est égal à $(3/5, 3/4)$. Notons que les visibilités depuis deux faces opposées sont identiques, puisque la visibilité est une information purement géométrique qui ignore la transparence du matériau.

Notons aussi que cette construction ne prend pas en considération la visibilité pour des directions non parallèles aux axes. Une approximation plus complète consisterait à échantillonner la visibilité pour toutes les directions, qui serait bien plus coûteuse à conserver. Nous discutons de cette possibilité comme extension au chapitre 5.

### 3.3.3 Matériau $M$

Plusieurs objets peuvent être inclus dans le même voxel. Il s'agit alors de représenter les propriétés des matériaux de ces objets dans chaque voxel. Une bonne approximation consisterait à associer à chaque direction de la NDF le matériau visible dans cette direction. Nous avons choisi une approximation plus simple, qui consiste à filtrer toutes les propriétés des matériaux du voxel (transparence $\tau$, couleur $C$, coefficients diffus $k_d$ et spéculaire $k_s$, rugosité $n$, etc.), afin de créer un nouveau matériau "moyen". Cette moyenne est strictement incorrecte, puisque la moyenne de deux rugosités par exemple n'est pas leur valeur moyenne. En effet, le modèle de Phong [Pho75] utilise un terme spéculaire de la forme :

$$I_s = \sum_{j \text{ lumières}} I_j k_s \cos^n \theta_j \tag{3.5}$$

La moyenne de deux valeurs de $n$ est donc celle qui correspond à une valeur de $\cos^n \theta$ moyenne. Cependant, pour les autres propriétés du matériau telle que la couleur, cette approximation est acceptable.

### 3.3.4 Création des niveaux supérieurs

Une fois l'information créée dans les voxels-feuilles, cette information peut être propagée aux niveaux supérieurs de l'octree pour fournir des versions de plus basses résolutions. Ceci équivaut à filtrer l'information de huit voxels-enfants et de conserver le résultat dans leur voxel-parent. La procédure `MoyennerInformationEnfants` se charge de cette tâche, en utilisant les propriétés de filtrage des informations que nous avons vues à la section précédente. Notons que ce filtrage n'est pas essentiel à la construction des voxels. En effet, il est possible, quoique plus coûteux, de recalculer l'information à *chaque* niveau de l'octree. Ceci serait accompli tout simplement en appelant la procédure `ConstruireInformation` au lieu de `MoyennerInformationEnfants` dans l'algorithme `AjouterObjetsAuxVoxels`. De plus, les informations ainsi obtenues seraient plus précises que celles obtenues par filtrage, puisque le filtrage dissipe l'information de par sa nature. Cependant, le passage d'un niveau à l'autre dans l'octree pourrait perdre de la continuité, au cas où l'absence de filtrage générerait de l'aliassage. Pour pouvoir comparer ces deux approches, notre système permet soit de filtrer, soit de recalculer les informations à chaque niveau supérieur.

*Nous avons décrit dans ce chapitre notre méthode de construction du volume de référence, étant donnée une liste d'objets géométriques. Une fois le volume créé, il est sauvegardé sur fichier, pour être utilisé comme référence par les texels de la scène lors du rendu. Le chapitre suivant décrit la méthode utilisée pour effectuer le rendu de ces texels.*

# Chapitre 4

# Rendu des texels

*Nous décrivons dans ce chapitre la méthode de rendu des texels. Nous commençons par décrire l'algorithme utilisé, puis nous élaborons les détails techniques et les choix effectués.*

## 4.1 Algorithme

Le rendu des texels s'effectue dans le contexte d'un algorithme de lancer de rayons classique [Whi80, Gla89]. Nous avons choisi cette technique de rendu car les texels s'y intégrent naturellement. Les texels sont cependant indépendants de l'algorithme de rendu utilisé. Nous présenterons au chapitre 5 quelques idées sur leur intégration à d'autres méthodes telles que le rendu projectif.

Quand un rayon de la scène intersecte un texel (la méthode d'intersection est détaillée plus bas), la procédure `IlluminerTexel( Texel` $T$`, Rayon` $R$`, Point` $P$ `)` est appelée pour calculer l'intensité du rayon $R$ traversant ce texel. Un rendu volumique est effectué à l'intérieur de cette procédure. Selon l'équation 2.5 proposée par Kajiya *et al.* [KK89], l'intensité est obtenue par intégration de la fonction de réflectance sur le trajet du rayon à l'intérieur du volume pour chaque source lumineuse de la scène, modulée par l'atténuation cumulée sur le rayon, ainsi que par l'atténuation due au blocage des source lumineuses. Dans notre cas, cette équation se traduit par la somme des illuminations locales à chaque voxel traversé par le rayon, en cumulant aussi l'atténuation à chaque voxel.

La procédure initialise l'intensité (c'est-à-dire la couleur) à zéro, et le facteur d'atténuation au minimum. Ensuite, la traversée du volume de ce texel commence. Pour chaque voxel traversé, l'illumination locale au centre du voxel est calculée. Elle est pondérée par l'atténuation globale actuelle et par l'occlusion due à l'ombrage, puis elle est cumulée dans la couleur globale. L'atténuation locale est aussi cumulée dans l'atténuation globale. La figure 4.1 illustre le calcul d'illumination de la couleur $C$ et de l'atténuation $A$ pour les deux premiers voxels $V_1, V_2$ sur le trajet du rayon. Le chemin du rayon est sauvegardé pour permettre une seconde passe au niveau supérieur de l'octree, dans l'esprit du *MIP map*. Cette approche multi-résolution nécessite que

nous considérions le rayon comme un cône à l'intérieur du texel. Nous transformons le rayon en cône en calculant son ouverture à l'entrée du texel.

Nous donnons d'abord une vue d'ensemble de cette procédure, puis nous ajouterons les détails dans les sections suivantes.

```
fonction IlluminerTexel( Texel T, Rayon R, Point P ) : Couleur
{
  Couleur C = Noir;
  Float Att = 1;
  Float d = OuvertureRayon( R, P );
  Rayon R_T = TransformerRayon( R, d, Déformation( T ) );

  // Première passe du calcul : niveau inférieur de l'octree.
  ItérateurOctree it( Volume( T ), R_T );
  tant que ( Valide( it ) et Att > 0 )
  {
    Voxel V = VoxelCourant( it );
    [Couleur C_V, Float Att_V] = IlluminerVoxel( V, T, R_T );
    C += Att * C_V;
    Att *= Att_V;

    Avance( it );
  }

  // Si le rayon a trouvé des voxels non-vides,
  // effectuer une seconde passe du calcul : niveau supérieur.
  si ( Att < 1 )
  {
    Couleur C' = Noir;
    Float Att' = 1;
    ItérateurListe il( CheminParent( it ) );
    tant que ( Valide( il ) et Att' > 0 )
    {
      Voxel V = VoxelCourant( il );
      [Couleur C_V, Float Att_V] = IlluminerVoxel( V, T, R_T );
      C' += Att' * C_V;
      Att' *= Att_V;
    }
```

```
    Float f = CalculerFacteurMipmap( it ) ;
    C = f * C + (1 − f) * C′ ;
    Att = f * Att + (1 − f) * Att′ ;
  }

  // Si l'atténuation n'est pas maximale, poursuivre le rayon dans la scène.
  si ( Att > 0 )
  {
    Point X = PointSortie( it ) ;
    C += Att * LancerRayonDansScène( R, X ) ;
  }
  retour C ;
}
```

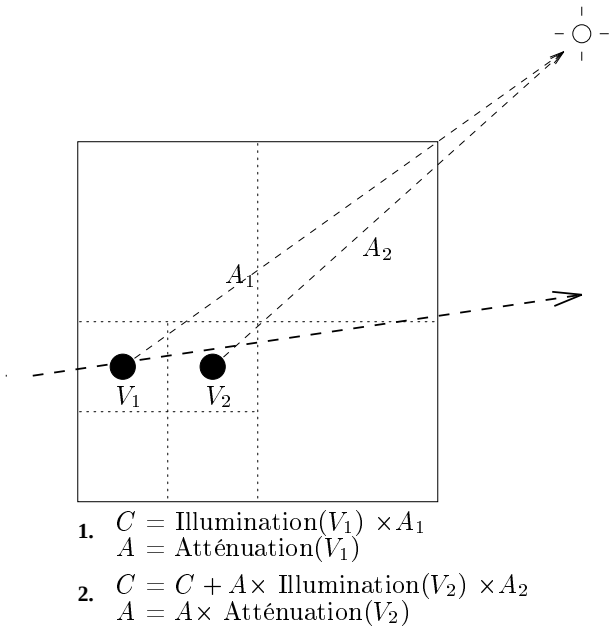


FIG. 4.1: Calcul de l'illumination $C$ et de l'atténuation $A$ du texel. $A_1$ et $A_2$ représentent l'atténuation d'un rayon d'ombrage lancé à partir de $V_1$ et $V_2$ respectivement.

## 4.2 Opérations sur le rayon d'entrée $R$

Notons d'abord que `IlluminerTexel` accepte un rayon $R$ dans l'espace-scène. Il nous faut transformer le rayon $R$ en cône dans espace-texel, puisque le texel est une copie virtuelle déformée du volume de référence. Le passage du rayon en espace-texel pose un problème. En effet, à moins que nous ne traitions que de déformations affines, le rayon transforme ne sera plus linéaire.

Pour correctement traverser le volume de référence en espace-texel, il faudrait connaître la représentation non-linéaire du rayon après déformation et intersecter les voxels avec ce rayon généralisé. Cependant, le coût du calcul serait prohibitif, et il n'existe pas de solution analytique pour tous les genres de déformation. Plusieurs approximations sont alors possibles. Nous pouvons discrétiser le rayon au niveau des voxels, c'est-à-dire que nous considérons le rayon comme étant linéaire à l'intérieur d'un seul voxel, mais non-linéaire au niveau du texel. Ainsi, d'un voxel à l'autre, il serait nécessaire de calculer la nouvelle position et direction du segment de rayon. Une autre approximation encore plus grossière est de considérer le rayon comme étant linéaire à l'intérieur du texel entier, mais de calculer analytiquement les points d'entrée et de sortie. Ainsi, la traversée de l'octree se fait sans aucune modification. C'est cette dernière approche que nous utilisons pour le moment. Il est à noter que dans le cas des fortes déformations, comme une FFD (*free-form deformation* [SP86]) qui plierait le texel en forme de "U", cette approximation serait tout-à-fait incorrecte (tel que l'illustre la figure 4.2).

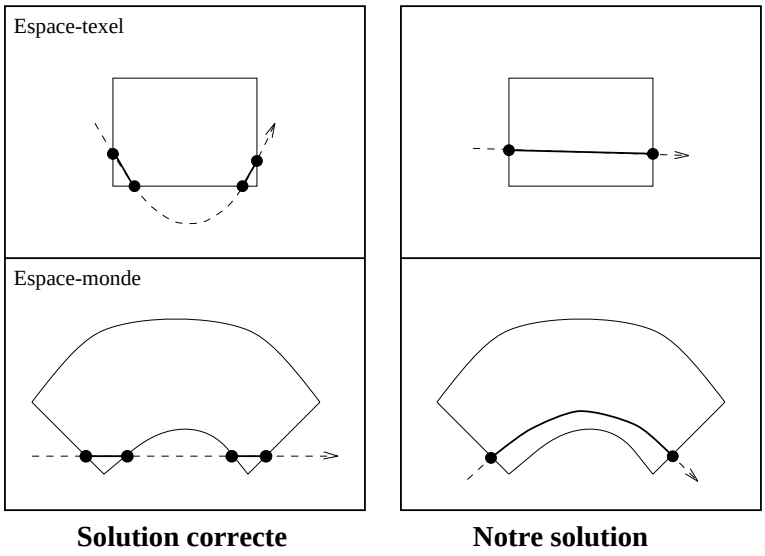


FIG. 4.2: Traversée incorrecte du voxel dans le cas de fortes déformations.

Il faut ensuite calculer l'ouverture de ce rayon, ouverture qui nous permettra de choisir la résolution adéquate à l'intérieur de l'octree, comme nous le verrons plus tard, de manière à minimiser l'aliassage. Plus l'ouverture est importante, plus basse sera la résolution utilisée, et donc plus important sera le filtrage – et le flou résultant – dans l'image. Nous considérons cette ouverture égale à l'entrée et à la sortie du texel, le cône est donc techniquement un cylindre qui traverse l'octree. L'ouverture correspond à la projection inverse d'un pixel de l'écran sur un plan parallèle à l'écran. Plus on s'éloigne de l'écran, plus grande sera la région de la scène projetée sur un seul pixel. Nous considérons le pixel comme un cercle, l'ouverture du rayon est donc une mesure de cette région – circulaire – de la scène ; elle est obtenue par la règle des triangles semblables (voir figure 4.3). Nous avons choisi de considérer le pixel comme un

cercle pour faciliter le processus d'intersection des rayons avec les texels. L'utilisation d'un pixel rectangulaire aurait donné naissance à des rayons pyramidaux qui sont plus coûteux à traiter. Nous perdons cependant de l'information de cette manière à cause des régions vides entre les pixels circulaires, mais nous stipulons que l'effet de cette approximation est négligeable.

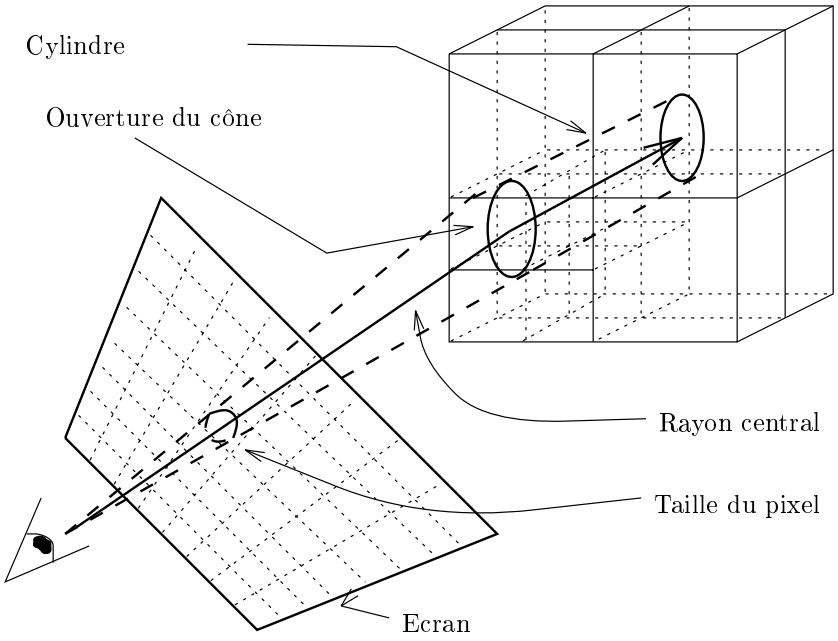


FIG. 4.3: Traversée du texel.

Ayant calculé l'ouverture du cône en espace-monde, il faut maintenant la transformer en espace-texel. Considérons le jacobien de la déformation en un point donné $P$. Ce jacobien fournit le référentiel local de la déformation en $P$, ainsi que le facteur d'échelle de chaque axe du référentiel. Transformer un segment de droite en $P$ dans l'espace-texel revient donc à exprimer ce segment dans un référentiel local, lequel est obtenu par le jacobien de l'inverse de la déformation[1] (puisque nous passons de l'espace-monde à l'espace-texel.) Nous approximons ce résultat en choisissant l'axe le plus long comme facteur d'échelle sur l'ouverture du cône, ce qui correspond à une approche prudente qui favorise le filtrage.

## 4.3 Traversée du rayon $R_T$ dans le texel $T$

Une fois le rayon $R$ transformé, nous obtenons un cône $R_T$ en espace-texel. Nous pouvons alors calculer les voxels de $T$ traversés par ce cône. L'idée de la traversée est de trouver les points d'intersection de $R_T$ avec les parois des voxels, et de retourner chaque voxel intersecté dont la taille correspond à l'ouverture de $R_T$. L'itérateur `ItérateurOctree` se charge de cette traversée de manière récursive. L'intersection du cône avec le voxel-racine, c'est-à-dire avec les parois du

[1] Etant donnée une déformation inversible $\mathcal{T}$, $J_{\mathcal{T}^{-1}} = J_{\mathcal{T}}^{-1}$.

texel, est calculée. Le segment défini par les points d'entrée-sortie est empilé, puis la fonction `ProchainVoxel` est appelée. Cette fonction dépile un segment, puis vérifie si la taille du voxel correspondant à ce segment est inférieure ou égale à l'ouverture du cône. Si c'est le cas, ce voxel est retourné. Sinon, il faut subdiviser ce voxel pour trouver lesquels de ses enfants sont traversés par le cône. Si le voxel n'a pas d'enfant, il est retourné, en indiquant qu'il ne répond pas au critère. Sinon, on calcule l'intersection du rayon avec les plans médians du voxel (c'est-à-dire ses parois internes). La position des points d'intersection nous permet de déduire lesquels des enfants sont traversés. Les segments obtenus sont triés, puis empilés pour une nouvelle itération. La traversée du voxel est terminée une fois que la pile de segments est vide.

```
struct Segment
{
  Point Entree ;
  Float t_Entree ;
  Point Sortie ;
  Float t_Sortie ;
  Voxel V ;
} ;

function ProchainVoxel( ItérateurOctree it, Rayon R_T ) : Voxel
{
  tant que ( Pile( it ) n'est pas vide )
  {
    Segment S = Dépile( it ) ;
    Voxel V = Voxel( S ) ;
    si ( Taille( Boîte( V ) ) ≤ Ouverture( R_T ) )
    {
      retour V ;
    }

    si ( V n'a pas d'enfant )
    {
      Indicateur( V ) ;
      retour V ;
    }

    Point C = Centre( Boîte( V ) ) ;
    Liste<Float> L ;
    Float t_1 = Intersection( R_T, C.x ) ;
    si ( t_Entree < t_1 < t_Sortie ) Ajoute( t_1, L ) ;
```

```
    Float t2 = Intersection( R_T, C.y );
    si ( t_Entree < t2 < t_Sortie ) Ajoute( t2, L );
    Float t3 = Intersection( R_T, C.z );
    si ( t_Entree < t3 < t_Sortie ) Ajoute( t3, L );
    Ajoute( t_Entree, L );
    Ajoute( t_Sortie, L );
    Trie( L );

    pour toute paire ( t_i, t_{i+1} dans L )
    {
      Voxel V_i = TrouverEnfant( V, t_i, C );
      Empiler( it, Segment( Extrap( R_T, t_i ), t_i,
                            Extrap( R_T, t_{i+1} ), t_{i+1},
                            V_i, Boîte( V_i ) ) );
    }
  }
  retour NULL;
}
```

Souvenons-nous que dans l'espace-texel, la boîte englobante du texel (et de tout voxel) est alignée sur les axes du référentiel. C'est pourquoi l'intersection du rayon avec les faces des voxels n'est pas coûteuse. Les points d'intersection trouvés ne sont retenus que s'ils se trouvent entre les points d'entrée-sortie du segment. Enfin, pour trouver le voxel-enfant étant donné le point d'intersection, il suffit de comparer le point d'intersection avec le centre du voxel. Pour chaque axe, il existe deux possibilités (au-dessus ou au-dessous). Chacune des $2^3 = 8$ possibilités correspond donc à l'un des huit voxels-enfants.

## 4.4 Calcul de l'illumination d'un voxel $V$

A chaque voxel $V$ traversé par le cône $R_T$, la procédure `IlluminerVoxel` calcule l'illumination locale. Nous évaluons l'intégrale de la NDF sur un modèle d'illumination (tel que celui de Phong [Pho75]) au centre du voxel, pour obtenir la valeur de l'intensité locale $C_V$. Cette intensité est atténuée par l'atténuation globale du cône cumulée jusqu'à $V$. L'atténuation locale $Att_V$ est aussi calculée, elle vient augmenter l'atténuation globale. $Att_V$ est obtenue en combinant la visibilité dans la direction du cône et la transparence du matériau du voxel, en pondérant par la distance traversée à l'intérieur du voxel. Il faut aussi traiter le cas particulier où le voxel est à l'intérieur d'un objet, en utilisant la valeur de la couleur calculée sur le voxel frontalier le long de ce rayon.

```
fonction IlluminerVoxel( Voxel V, Texel T, Rayon R_T ) : [Couleur, Float]
{
  Point P_o = Centre( Boîte( V ) );
  Point P = TransformerPoint( P_o, Déformation( T ) );
  Float Att_V = 1 - ( ( 1 - CalculerVisibilité( V, R_T ) ) *
                      CalculerFacteurDistance( V, R_T ) *
                      (1 - Transparence( Matériau( V ) ) ) );
  Couleur C_V ;
  si ( V est Intérieur )
  {
    C_V = DernièreCouleur ;
  }
  sinon
  {
    pour chaque ( Lumière L dans la scène )
    {
      Vecteur l = Position( L ) - P ;
      Atténuation( L ) = AtténuerTexel( T, Rayon( P, l ) ;
    }
    C_V = IntégrerNDF( V, T, R_T ) ;
    C_V += V.M.k_a * I_a ;
    DernièreCouleur = C_V ;
  }

  C_V *= 1 - Att_V ;
  retour [C_V, Att_V] ;
}
```

La fonction `CalculerVisibilité` calcule le facteur de visibilité du voxel dans la direction $\mathbf{d}$ du rayon $R_T$. Comme nous l'avons vu plus haut, nous représentons la visibilité par trois pourcentages d'occlusion, pour les trois axes du référentiel, dans un vecteur $\mathbf{v}$. Etant donnée une direction $\mathbf{d}$, nous estimons la visibilité dans cette direction par le produit scalaire $\mathbf{d} \cdot \mathbf{v}$, en prenant soin d'utiliser la valeur absolue des trois composantes de $\mathbf{d}$, puisque la visibilité est toujours non-négative, et qu'elle est égale pour $\mathbf{d}$ et $-\mathbf{d}$. Nous devons aussi appliquer un seuil de 1 à la visibilité résultante.

La fonction `CalculerFacteurDistance` vient pondérer l'atténuation par la longueur du segment de rayon traversant $V$. Un rayon traversant le voxel sur une très faible distance sera faiblement atténué dans ce voxel. Rappelons-nous que durant la construction du voxel, nous

estimons l'occlusion en lançant des rayons à partir de chaque face du voxel vers la face opposée. Nous obtenons ainsi une approximation de la valeur de l'occlusion, uniquement pour les cas où le rayon traverse le voxel de part en part. Cependant, le rayon peut aussi pénétrer un voxel et en sortir par des faces adjacentes. Dans ce cas, la distance parcourue par le rayon à l'intérieur du voxel peut être plus faible que l'arête du voxel, situation que la construction ne traite pas. Nous pourrions modifier la méthode de construction en échantillonnant le voxel sur toutes les directions, ce qui se révélerait très coûteux. Nous avons choisi de pondérer la valeur de la visibilité obtenue pendant la construction par un facteur égal à 1 si la longueur du segment de rayon $dl$ traversant le voxel est supérieure ou égale à l'arête du voxel $s$, et égal à $dl/s$ sinon. Nous nous assurons ainsi que nous ne surestimons pas l'occlusion due au voxel.

Nous lançons un cône d'ombrage vers chaque source lumineuse dans la scène. Ce cône doit d'abord traverser le texel courant avant de continuer son trajet dans la scène. Calculer l'opacité d'un texel dans `AtténuerTexel` est tout-à-fait similaire au calcul d'illumination de l'algorithme `IlluminerTexel`, la seule différence étant que la fonction `IlluminerVoxel` n'est pas appelée, seule l'atténuation $Att$ étant calculée.

## 4.5 Intégration du modèle d'illumination locale sur la NDF

Une fois l'atténuation pour chaque source lumineuse obtenue, la fonction `IntégrerNDF` est appelée pour intégrer le modèle d'illumination locale sur la NDF du voxel. Dans le cas de notre NDF représentée par un ellipsoïde, nous ne désirons intégrer le modèle d'illumination locale que sur la partie visible de l'ellipsoïde, qui correspond aux normales visibles à partir de notre point de vue. Nous pourrions tenter de trouver cette partie visible analytiquement, mais nous nous sommes restreints pour le moment à échantillonner aléatoirement la totalité de l'ellipsoïde en coordonnées polaires, puis à rejeter les normales qui ne sont pas visibles. Cet échantillonnage nous fournit aussi l'importance de chaque normale dans la NDF. Nous multiplions alors la valeur de l'illumination locale pour chaque normale par son importance, puis nous normalisons le résultat final. L'algorithme `IntégrerNDF` illustre ce processus.

```
fonction IntégrerNDF( Voxel V, Texel T, Rayon R_T ) : Couleur
{
  Point P_o = Centre( Boîte( V ) );
  Point P = TransformerPoint( P_o, Déformation( T ) );
  Vecteur v = PosOeil - P;
  Matrice J = Jac( Déformation( T ), P )^-1t;
  Float c = 0;
  Couleur C = Noir;
  tant que ( pas assez d'échantillons )
  {
```

```
    Point M = GénérerPoint() ;
    Vecteur n = J * M ;
    Float n = n · v * ||M|| ;
    si ( n > 0 )
    {
      c += n ;
      C += n * IlluminationLocale( V, P, n, v ) ;
    }
  }
  retour C / c ;
}
```

Une subtilité est à noter ici. Lorsque nous effectuons un échantillonnage du voxel pour obtenir notre distribution de normales, nous obtenons une distribution d'égale importance pour les trois directions axiales, c'est-à-dire que la distribution est biaisée sur les trois axes (il serait préférable d'échantillonner la visibilité sur une sphère englobant le voxel, nous discutons de cette possibilité comme extension au chapitre 5). Durant le rendu, lorsqu'un rayon atteint l'ellipsoïde dans une certaine direction, nous désirons obtenir l'importance des normales de l'ellipsoïde dans la direction de ce rayon (en plus de l'importance de chaque normale dans sa propre direction). Autrement dit, nous désirons trouver la projection orthographique de chaque normale sur un plan orienté selon la direction du rayon (voir la figure 4.4). Le terme $\mathbf{n} \cdot \mathbf{v}$ se charge de calculer cette projection, et vient pondérer l'importance de la normale pour le calcul de l'illumination.

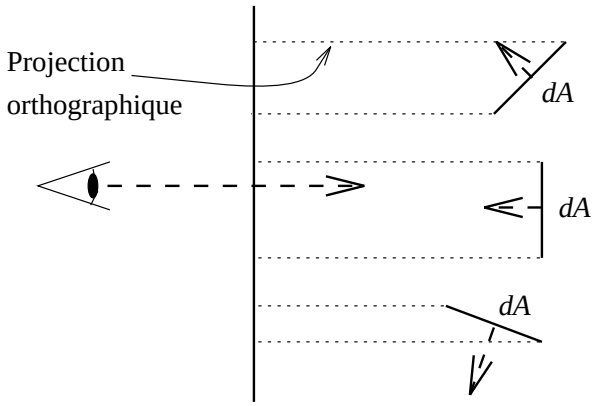


FIG. 4.4: Prise en compte du facteur de projection des normales dans la direction du rayon.

### 4.5.1 Calcul des normales

Notons que la NDF est originellement positionnée dans l'espace-texel ; il est nécessaire de la transformer en espace-monde pour calculer l'illumination. Nous effectuons cette étape en transformant chaque normale obtenue de l'ellipsoïde par la transposée du jacobien de la déformation inverse du texel. Soit $\mathcal{T}$ une transformation inversible de l'espace-objet à l'espace-monde, $\mathbf{v}_o$ un vecteur dans l'espace-objet, et $\mathbf{n}_o$ la normale à une surface dans l'espace-objet, on a :

$$\begin{aligned} \mathbf{n}_o \cdot \mathbf{v}_o &= \mathbf{n}_o \cdot \mathbf{J}_{\mathcal{T}}^{-1} \mathbf{J}_{\mathcal{T}} \mathbf{v}_o \\ &= \mathbf{J}_{\mathcal{T}}^{-1\mathrm{t}} \mathbf{n}_o \cdot \mathbf{J}_{\mathcal{T}} \mathbf{v}_o \\ &= \mathbf{J}_{\mathcal{T}}^{-1\mathrm{t}} \mathbf{n}_o \cdot \mathbf{v} \end{aligned} \tag{4.1}$$

Donc la normale en espace-monde $\mathbf{n}$ est égale à $\mathbf{J}_{\mathcal{T}}^{-1\mathrm{t}} \mathbf{n}_o$. Cette normale est ensuite pondérée par son importance, obtenue par évaluation de la longueur de la normale en espace-texel.

### 4.5.2 Echantillonnage de l'ellipsoïde

La fonction `GénérerPoint` génère une paire $(\theta, \phi)$ aléatoire, puis convertit cette paire en un point sur l'ellipsoïde. Le passage de coordonnées polaires aux coordonnées cartésiennes sur une sphère canonique s'effectue ainsi :

$$\begin{aligned} x &= \sin\theta \cos\phi, \\ y &= \sin\theta \sin\phi, \\ z &= \cos\theta. \end{aligned} \tag{4.2}$$

Il suffit alors de passer d'un point sur la sphère canonique à un point sur l'ellipsoïde, ce qui revient à transformer le point par la matrice formée par les trois vecteurs caractéristiques de l'ellipsoïde. Notons que bien que la déformation de notre texel soit arbitraire, les normales ainsi échantillonnées seront valides, même si l'ellipsoïde lui-même est fortement déformé. Ceci est dû au fait que nous échantillonnons les normales en espace-objet *puis* nous les transformons individuellement en espace-monde.

Quoique cette approche converge vers la solution correcte, il serait plus efficace d'utiliser la direction du rayon pour déterminer la région de l'ellipsoïde à échantillonner. Malheureusement, ceci demanderait le passage de l'espace-monde à l'espace-objet, soit une transformation inverse qui est coûteuse à évaluer, et pour laquelle il n'existe pas en général de solution analytique.

### 4.5.3 Calcul de l'illumination locale

La fonction `IlluminationLocale` calcule le modèle d'illumination locale en un point $P$, dont la normale est $\mathbf{n}$, étant donnée la direction de vue $\mathbf{v}$. Cette fonction applique simplement le modèle de Phong [Pho75], de la manière suivante :

```
fonction IlluminationLocale( Voxel V, Point P, Vecteur n, Vecteur v ) : Couleur
{
  Couleur C = Noir ;
  pour chaque ( Lumière L dans la scène )
  {
    Couleur I = Noir ;
    Float d = || Position( L ) - P || ;
    Vecteur l = Normalise( Position( L ) - P ) ;
    si ( n · l > 0 )
    {
      I += k_d * (n · l) ;
      Vecteur h = Normalise( v + l ) ;
      si ( n · h > 0 ) I += k_s * (n · h)^n ;
    }
    C += Atténuation( L ) * I_L * I / d^2 ;
  }
  retour C ;
}
```

## 4.6 Calcul multi-résolution

En général, l'ouverture d'un cône ne correspond jamais exactement à la taille d'un voxel, elle tombe plutôt entre les tailles de deux niveaux adjacents. Il faut donc calculer l'illumination à ces deux niveaux, puis interpoler la valeur finale selon la distance entre l'ouverture du cône et la taille des niveaux concernés. Lors de la traversée de l'octree, l'itérateur `ItérateurOctree` sauvegarde le chemin du rayon. Ainsi, le même chemin peut être retracé à plus faible résolution. Le facteur d'interpolation est calculé au prorata de la différence de taille entre l'ouverture $r$ du cône et les arêtes des voxels des deux niveaux. Soient $s_i$ la taille du voxel au niveau $i$, $s_{i+1}$ la taille du voxel au niveau $i+1$, et $C_i$ et $C_{i+1}$ les valeurs de la couleur calculée à ces deux niveaux. La valeur finale $C$ est simplement une interpolation linéaire entre ces deux valeurs, $C = fC_i + (1-f)C_{i+1}$, où $f = (r - s_{i+1})/(s_i - s_{i+1})$.

Dans le cas où l'ouverture du cône est inférieure à la taille du dernier niveau de l'octree, nous ne calculons l'illumination que pour le dernier niveau. Ce cas se présente lorsque le texel est très près du point de vue. Il faudrait alors calculer l'illumination en utilisant la véritable géométrie des objets du texel, ce que nous n'avons pas implanté – nous en discutons au chapitre

suivant.

*Dans ce chapitre, nous avons décrit la méthode de rendu des texels dans une scène. Ces texels sont des copies virtuelles déformées d'un volume de référence, construit pendant une phase de pré-calcul, décrite au chapitre précédent. Ainsi, nous avons établi les fondations de notre méthode de texture volumique multi-résolution. Les résultats obtenus laissent augurer des extensions intéressantes, et un grand nombre d'améliorations demeurent inexplorées. Nous présentons au chapitre suivant ces résultats, sous forme d'images et de statistiques, puis nous discutons des extensions à la méthode de base, afin de donner une vue d'ensemble concrète des nombreuses possibilités de la texture volumique multi-résolution.*

# Chapitre 5

# Résultats et extensions

*Nous présentons ici les résultats de notre approche et ses contributions. Nous proposons ensuite une liste d'extensions possibles – et souhaitables – dans les domaines du rendu, de la modélisation, et de l'animation des texels, en notant les simplifications que nous pouvons apporter à notre méthode pour traiter spécifiquement de la chevelure.*

## 5.1 Résultats et limitations

Nous présentons ici les images créées par notre système, ainsi que les statistiques concernant les temps de pré-calcul, les temps d'affichage, et les tailles de fichiers générés. Nous discutons ensuite des contributions de cet ouvrage.

Les figures 5.1, 5.2 et 5.3 représentent les différentes informations calculées dans le texel. Toutes ces images sont créées à partir d'une sphère géométrique, illustree dans la figure 5.1 de gauche, qui est transformée en un volume de résolution $64^3$. L'arrière-plan y est parallèle au plan image et une texture de damier y est apposée [1]. La source lumineuse est un point placé à 45°, vers le haut. La figure 5.1 de droite illustre le détail de la structure de subdivision spatiale utilisée, les voxels non-vides y ont une couleur constante. La figure 5.2 de gauche représente l'information d'atténuation qui vient pondérer la couleur constante des voxels. Nous pouvons remarquer que les bords de la sphère y sont flous, à cause de la faible opacité des voxels sur les bords de la sphère.

La figure 5.2 de droite représente l'illumination du texel. Si nous comparons cette image au rendu géométrique de la sphère (figure 5.1 gauche), nous pouvons remarquer que l'illumination est globalement correcte, mais que le bas de la sphère texélisée exhibe l'erreur due à la symétrie de la NDF ellipsoïdale. Le *highlight* est aussi plus étendu que sur la sphère originale, toujours à cause de cette symétrie. Les figures 5.3 illustrent le processus du rendu à une résolution de $16^3$ et de $8^3$ respectivement. Les coûts en mémoire et en temps de calcul sont rapportés dans la table

[1] La texture de damier ne fait pas partie du texel.

| Résolution | Précalcul | Mémoire |
|---|---|---|
| $8^3$ | 4 sec. | 0.073 Mo |
| $16^3$ | 16 sec. | 0.344 Mo |
| $32^3$ | 70 sec. | 1.550 Mo |
| $64^3$ | 295 sec. | 6.360 Mo |

TAB. 5.1: Construction d'un texel à diverses résolutions.

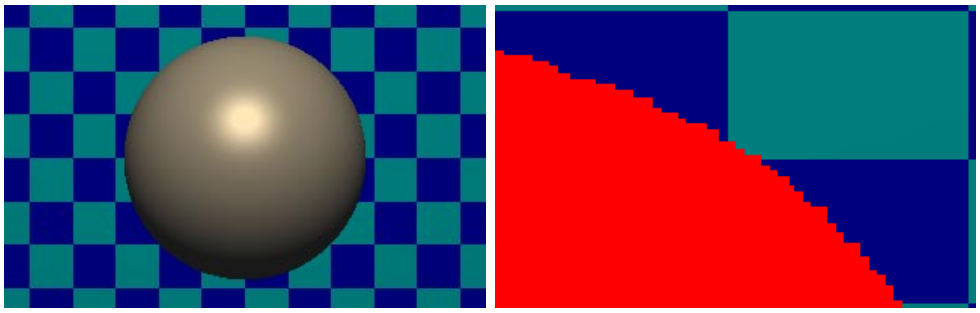

FIG. 5.1: Une sphère géométrique (gauche) et détail du texel correspondant (droite) où les voxels non-vides sont uniformément colorés.

ci-dessous[2]. Le graphe 5.4 du haut représente la mémoire utilisée (ordonnées) en fonction de la résolution du texel (abscisses), tandis que le graphe du bas représente le temps de pré-calcul (ordonnées) en fonction de la résolution (abscisses). Nous remarquons que les deux mesures exhibent une relation $O(n^2)$ avec la résolution $n \times n \times n$. Ce résulat est consistant avec le fait que seuls les voxels à la surface de la sphère sont analysés et conservés, puisque ces voxels occupent généralement $O(n^2)$ du texel. Les voxels considérés opaques (à l'intérieur de la sphère) représentent une partie négligeable du calcul et de la mémoire puisqu'ils ne sont traités qu'à la résolution la plus basse.

Nous pouvons illustrer la précision de notre méthode pour la représentation de la réflectance par les figures suivantes. La figure 5.5 de gauche représente 15 sphères géométriques dont le coefficient spéculaire $k_s$ varie de 0.5 à 0.1 de haut en bas, et dont le terme de rugosité spéculaire $n$ de 200 à 3 de gauche à droite. La figure 5.5 de droite représente un texel construit à partir de ces 15 sphères. Nous pouvons y remarquer la position correcte, et la taille supérieure, des *highlights* spéculaires. Notons aussi la haute définition de l'ombre créée par les voxels sur l'arrière-plan, ainsi que les bords plus flous de cette ombre. Le volume de référence est calculé à une résolution de $128^3$ en 363 sec., il occupe environ 7.69Mo de mémoire.

[2]Tous les calculs sont effectués sur une machine SGI Indy R5000. Pour le rendu, la résolution de l'image est de $320 \times 200$, et un seul rayon est lancé par pixel.

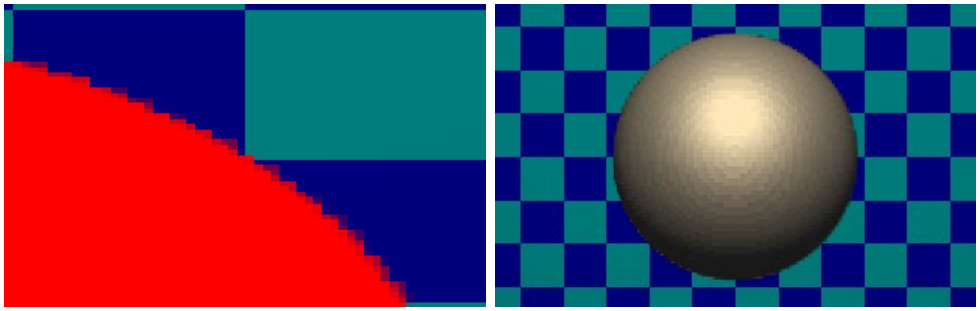

FIG. 5.2: Détail du texel (gauche) prenant en considération l'opacité des voxels sans calcul d'illumination, et texel illuminé (droite).

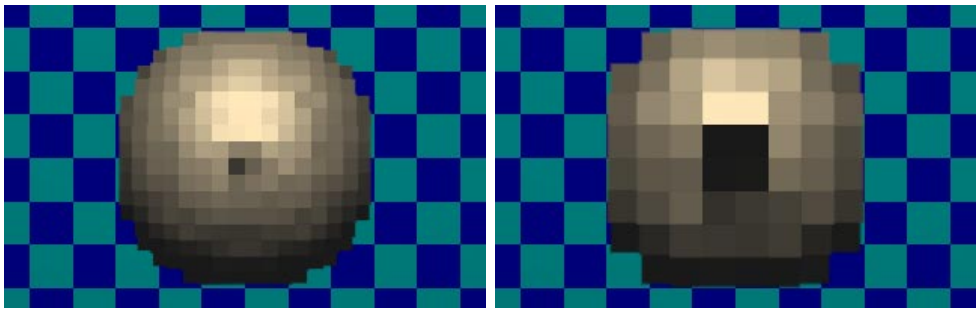

FIG. 5.3: Texel à une résolution de $16^3$ (gauche) et de $8^3$ (droite).

Nous étudions maintenant la performance de notre méthode pour des scènes relativement complexes. Nous créons quatre échantillons de "poils", dont la géométrie est formée respectivement de 25, 100, 900 et 3600 ellipsoïdes très fins et allongés. Tous les volumes sont créés à une résolution de $64^3$. Les temps de pré-calcul et les quantités de mémoire utilisées sont rapportées ci-dessous. Les figures 5.7 et 5.8 illustrent la géométrie des échantillons (gauche) et les texels résultants (droite) pour 25 et 100 ellipsoïdes respectivement[3]. La figure 5.9 illustre le texel contenant 100 ellipsoïdes à une résolution de $16^3$ (gauche) et de $8^3$ (droite). Le graphe 5.6 du haut représente la mémoire utilisée (ordonnées) en fonction du nombre d'ellipsoïdes (abscisses), tandis que le graphe du bas représente le temps de pré-calcul (ordonnées) en fonction du nombre d'ellipsoïdes (abscisses). Nous remarquons que ces deux mesures exhibent une relation $O(\log n)$ avec le nombre d'ellipsoïdes $n$. Ce comportement est lié à la configuration géométrique des primitives que nous échantillonnons. En effet, des ellipsoïdes peu nombreux et aléatoirement positionnés ont tendance à occuper des voxels vides, ce qui explique la croissance rapide du coût en mémoire et en temps de pré-calcul. Une fois une certaine densité atteinte, toutefois, les ellipsoïdes voisins commencent à occuper les mêmes voxels, et le coût se stabilise alors en une

[3]Les lignes horizontales qui apparaissent sur les figures sont dues à l'impression PostScript.

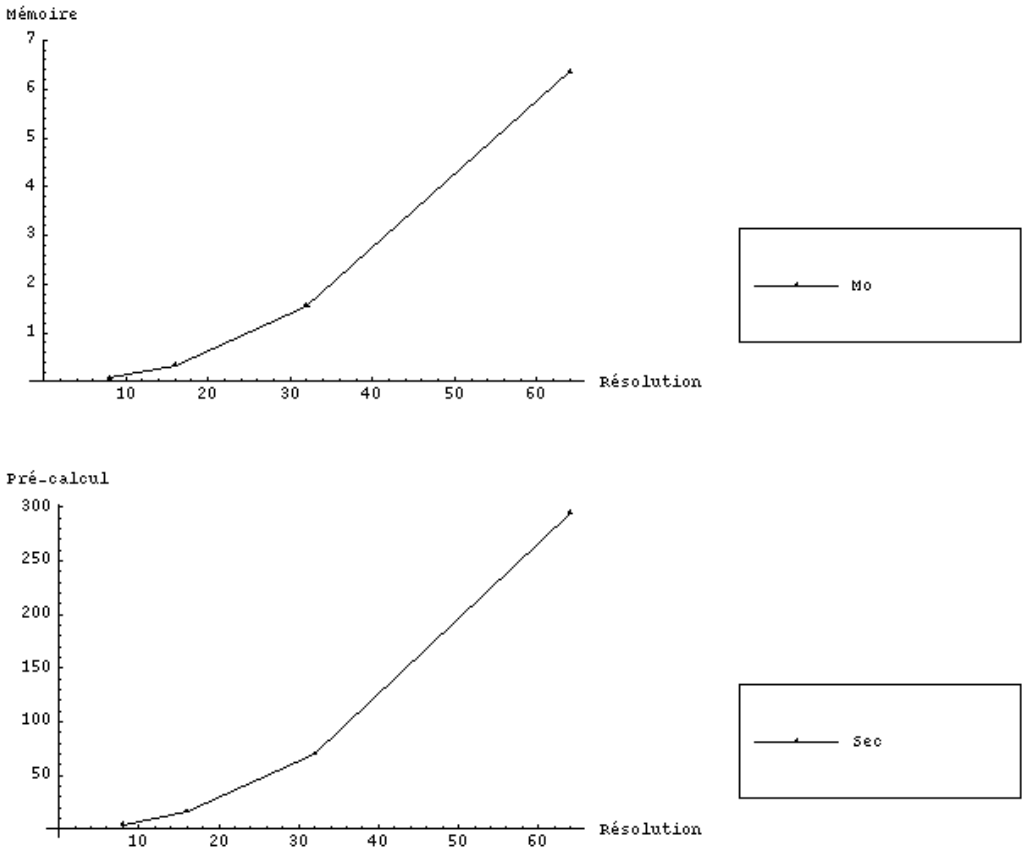


FIG. 5.4: Relation entre résolution et mémoire utilisée (haut) et entre résolution et temps de pré-calcul (bas).

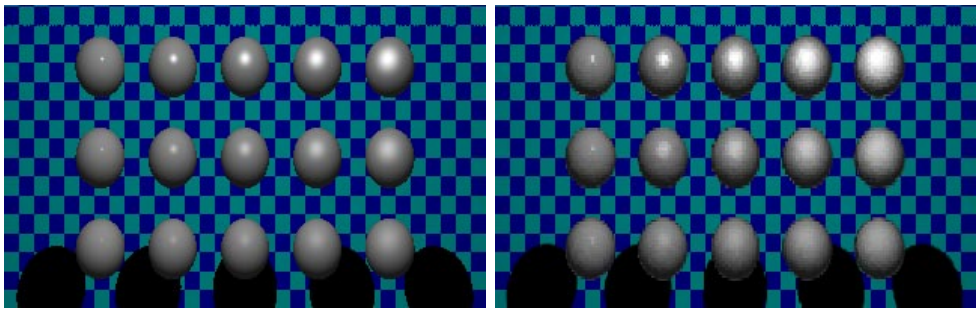

FIG. 5.5: 15 sphères géométriques de spécularités différentes (gauche) et le texel correspondant (droite).

| Ellipsoïdes | Précalcul | Mémoire |
|---|---|---|
| 25 | 220 sec. | 2.78 Mo |
| 100 | 751 sec. | 10.59 Mo |
| 900 | 1874 sec. | 29.10 Mo |
| 3600 | 3913 sec. | 55.23 Mo |

TAB. 5.2: Texel de "poils" de diverses géométries.

| Nombre d'objets | Géométrie sec. | Texels sec. |
|---|---|---|
| 100×1 | 142 | 294 |
| 100×16 | 289 | 1170 |
| 100×25 | 392 | 1406 |
| 100×100 | 883 | 1379 |
| 100×400 | 1920 | 1328 |
| 100×625 | 2576 | 1332 |
| 100×900 | — | 1489 |
| 100×2500 | — | 2068 |
| 100×3600 | — | 2497 |

TAB. 5.3: Scènes de complexité croissante et temps de rendu associés.

croissance faiblement linéaire.

Il nous faut enfin vérifier l'efficacité de notre méthode durant le rendu. A cet effet, nous utilisons un texel contenant 100 poils, et nous créons une série de scènes de complexité croissante. Pour chaque scène, nous créons une représentation purement géométrique, et une autre basée sur notre texel, puis nous comparons les temps de rendu des deux. Les dimensions des scènes et les temps de calcul pour chaque représentation sont rapportés dans la table 5.3, et le graphe 5.10 illustre la relation entre les trois mesures. Le temps de calcul pour une représentation à base de texels croît de façon linéaire avec le nombre de texels visibles, alors que le temps de calcul pour une représentation géométrique exhibe une relation au moins $O(n \log n)$. Cette constatation est consistante avec le fait que plus les texels sont éloignés du point de vue, moins leur rendu est coûteux, ce qui n'est pas le cas avec la géométrie.

La figure 5.11 illustre les deux représentations d'une scène contenant 62,500 poils géométriques (gauche), soit 625 texels (droite). Nous pouvons remarquer que la figure de gauche exhibe plus d'aliassage que celle de droite. Cependant, nous remarquons aussi la présence d'une ligne de démarcation (indiquée par une flèche) sur les poils de la figure de droite. Cette ligne correspond au passage d'un niveau de résolution à un autre dans l'octree du texel. Cet effet est d'autant

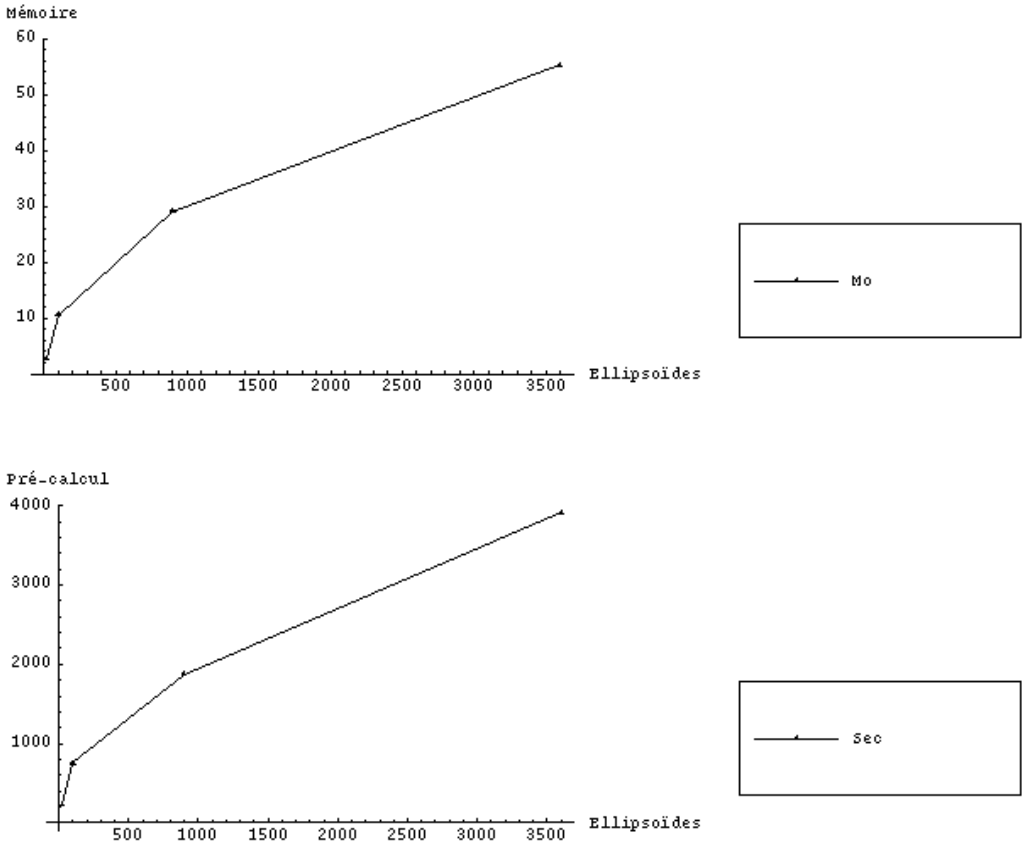


FIG. 5.6: Relation entre nombre de primitives et mémoire utilisée (haut) et entre nombre de primitives et temps de pré-calcul (bas).

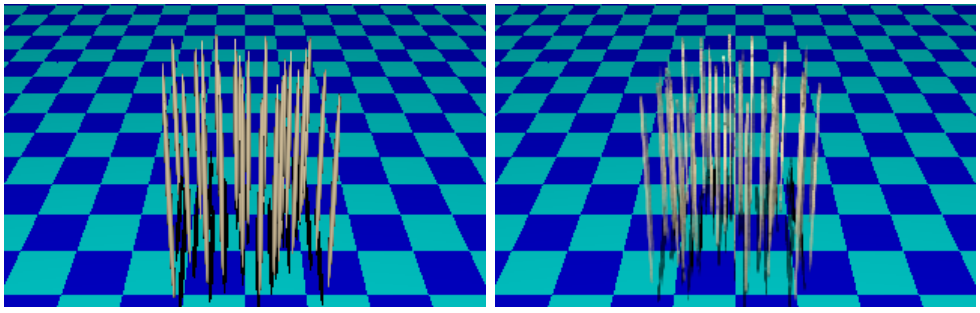

FIG. 5.7: 25 poils géométriques (gauche) et le texel correspondant (droite).

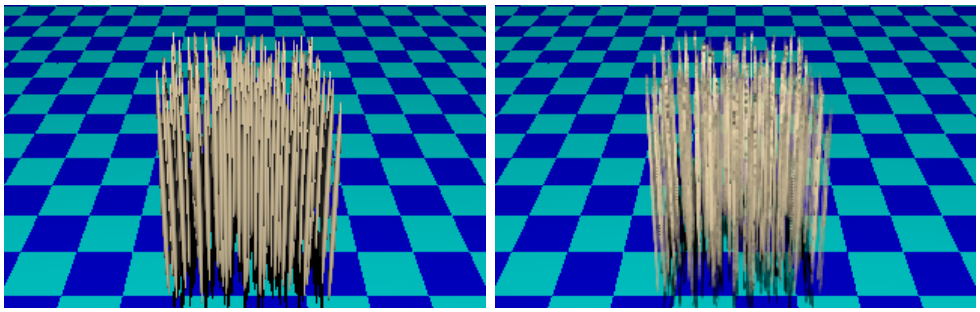

FIG. 5.8: 100 poils géométriques (gauche) et le texel correspondant (droite).

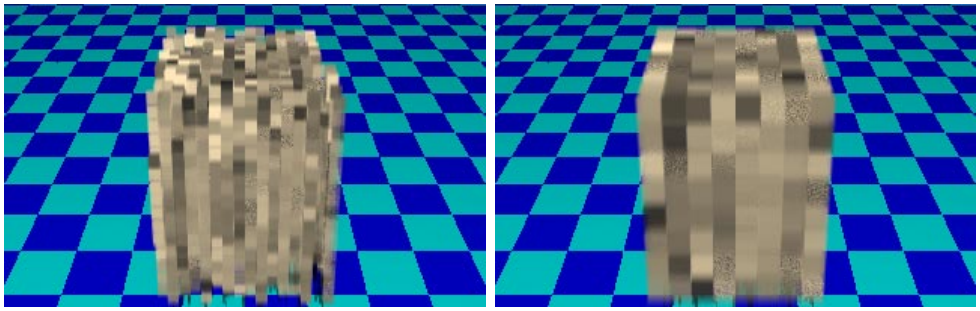

FIG. 5.9: Texel de 100 poils à une résolution de $16^3$ (gauche) et de $8^3$ (droite).

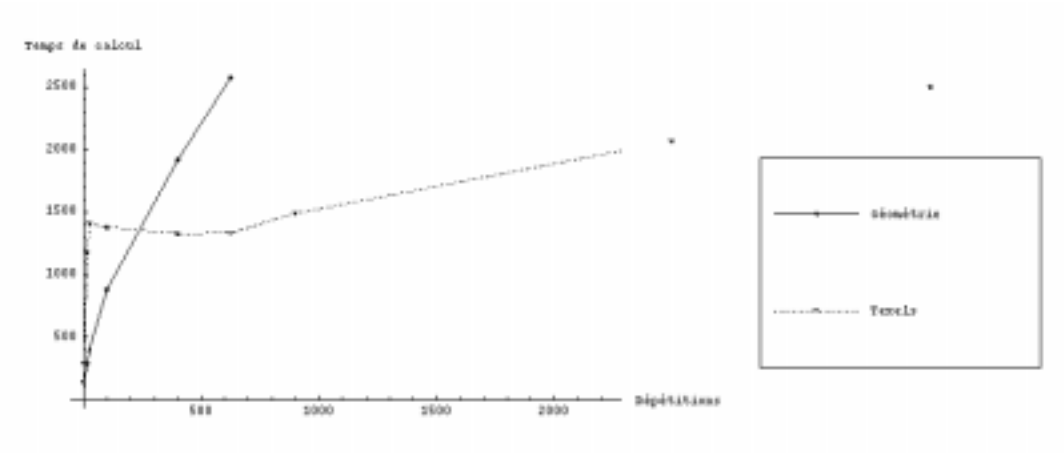


FIG. 5.10: Relation entre les temps de rendu et les scènes de complexité croissante.

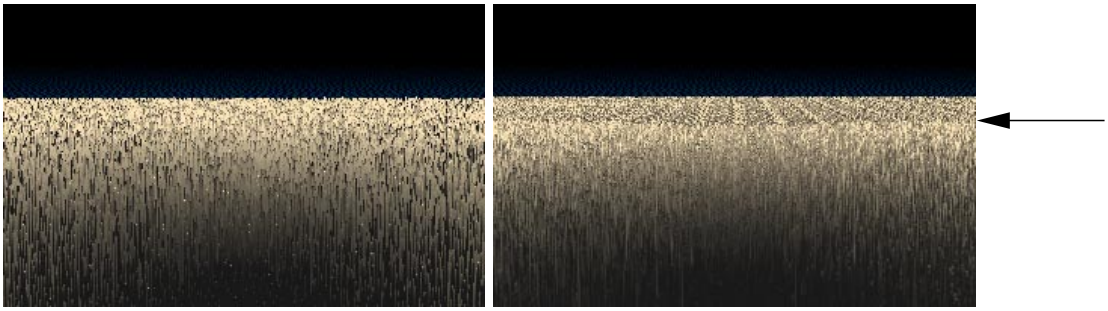

FIG. 5.11: $100 \times 625$ poils et texels associés.

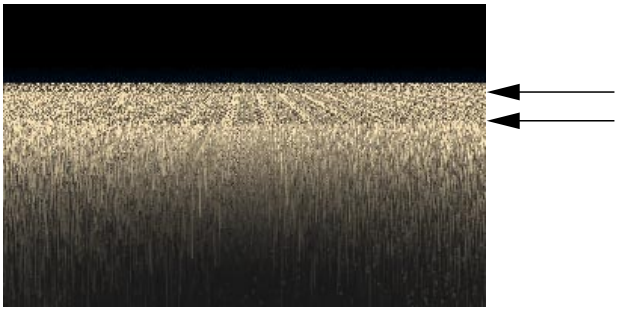

FIG. 5.12: 3600 texels et trois niveaux de résolution distincts.

plus marqué que tous les texels sont alignés dans notre scène. La figure 5.12 (3600 texels) illustre ce phénomène encore plus clairement : ici, nous pouvons voir deux lignes de démarcation qui signalent trois niveaux de résolution distincts.

Les statistiques et les résultats décrits ici révèlent bien l'utilité et les limitations des texels. Leur usage est intéressant lorsque la complexité des détails répétitifs franchit un certain seuil. En-deçà de ce seuil, la géométrie l'emporte tant par le temps de calcul que par le résultat visuel. Cependant, au-delà du seuil, les texels fournissent un temps de rendu presque constant, ainsi que des résultats visuels satisfaisants grâce au filtrage qui leur est intrinsèque. Le prix à payer pour ces bénéfices est une phase de pré-calcul modeste, qui permet par ailleurs de contrôler le niveau d'information désiré.

Les contributions de cet ouvrage se situent à deux niveaux. D'abord, nous proposons un grand nombre de généralisations au modèle de texture volumique multi-résolution de Neyret [Ney96b]. Revoyons rapidement les principaux aspects de cette généralisation :

- Notre modèle de texture volumique est totalement tri-dimensionnel, les texels sont conçus pour être positionnés arbitrairement dans l'espace. Le modèle de Neyret est limité à des texels placés sur une surface, formant une "peau volumique".
- Notre mode de construction du voxel se base sur l'échantillonnage des objets à représenter, et non sur une seule évaluation de la fonction de distance entre le voxel et l'objet géo-

métrique, telle qu'effectuée par Neyret. Ce dernier utilise le gradient de cette fonction de distance pour déterminer la valeur de la normale de l'objet dans le voxel. Par conséquent, si plusieurs objets occupent le même voxel et qu'un objet en cache un autre, la normale de l'objet caché sera quand même prise en compte dans la construction. En revanche, notre échantillonnage garantit que seules les surfaces visibles de l'extérieur du voxel seront traitées. De plus, Neyret ne calcule qu'une seule normale à l'objet dans un voxel alors que notre échantillonnage offre une meilleure approximation de la distribution des normales.

- Notre mode de construction de la NDF des voxels est totalement automatique, alors que celui de Neyret requiert la spécification manuelle d'un "galbe" s'appliquant à tous les ellipsoïdes créés.
- La texture de volumique de Neyret n'inclue pas d'information sur les matériaux représentés : le matériau de la surface sous-jacente est utilisé partout. Notre approche inclue cette information au niveau du voxel, permettant de représenter des matériaux divers dans un seul volume.

La seconde contribution de cet ouvrage est la création d'un logiciel orienté-objet, stable et extensible, permettant de facilement modifier les algorithmes existants et d'en écrire de nouveaux. Nous avons tenté de bien modulariser le système, même au détriment de l'efficacité, et d'abstraire tous les concepts que nous utilisons. Par exemple, les concepts de NDF, de déformation et de modèle d'illumination locale ne sont utilisés qu'à travers une interface, permettant de modifier leur représentation interne à tout moment, sans avoir à modifier la méthode globale. L'idée-clé ici a été de donner à chaque participant une responsabilité correspondant à la définition de son rôle dans la méthode. Par exemple, la NDF est responsable de calculer l'intégrale d'un modèle d'illumination locale (dont elle ne connaît que l'interface). Une description complète de l'architecture du système est donnée à l'appendice A. Ce système sera utile pour tester de nouvelles approches rapidement sans avoir à ré-implanter les services de base ou le processus global.

## 5.2 Extensions

L'idée de base de la texture volumique multi-résolution est simple, mais un grand nombre de détails entrent en jeu dans la mise en œuvre de la méthode. Nous n'avons pas pu explorer les différentes options qui se présentaient à chaque étape, c'est pourquoi nous discutons ici des nombreuses extensions envisageables à notre approche de base.

### 5.2.1 Construction

Nous n'avons pas étudié la compression du volume de référence. Cette compression est nécessaire car beaucoup d'informations sont conservées dans chaque voxel, ce qui nous coûte cher

en mémoire. Nous présentons ici quelques idées pour la compression.

Notre NDF ellipsoïdale peut être encodée par deux vecteurs. Souvenons-nous que les trois axes de l'ellipsoïde $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ sont orthogonaux : $k\mathbf{e}_3 = \mathbf{e}_1 \times \mathbf{e}_2$. Nous pouvons normaliser les vecteurs par rapport à $\mathbf{e}_3$, les vecteurs normalisés définiront un ellipsoïde représentant une NDF identique à la NDF originale. Il suffit donc de conserver $\mathbf{e}'_1 = \mathbf{e}_1/\|\mathbf{e}_3\|$ et $\mathbf{e}'_2 = \mathbf{e}_2/\|\mathbf{e}_3\|$, le troisième axe pourra être généré durant le rendu en évaluant $\mathbf{e}'_1 \times \mathbf{e}'_2$.

Le volume peut aussi être compressé en éliminant la redondance entre les voxels-enfants et leur parent. Nous définissons une fonction d'égalité qui compare les informations de deux voxels. Si les informations des huit voxels-enfants sont jugées égales à celles du voxel-parent, nous pouvons détruire les voxels-enfants. Durant le rendu, quand nous avons besoin d'accéder à leurs informations, nous pouvons simplement lire celles du parent. La fonction d'égalité utilisée doit comparer l'égalité de tous les types d'informations contenues dans le voxel : NDF, visibilité et matériau. Il s'agit alors de définir l'égalité pour ces types d'informations. Comparer deux vecteurs de visibilités est simple, comparer les propriétés des matériaux l'est aussi, mais comparer deux ellipsoïdes est plus ardu puisque vérifier l'égalité des axes ne suffit pas.

### 5.2.2 Rendu

Un grand nombre d'améliorations restent à être explorées pour le rendu des texels. Citons-en quelques-unes d'abord, avant d'en discuter en détail :

- Modèle de réflectance du voxel ;
- Tracé du rayon dans l'espace-objet ;
- Superposition de plusieurs texels ;
- Généralisation du critère de subdivision pendant la traversée du texel ;
- Intégration de la géométrie ;
- Meilleure représentation de la visiblité ;
- Filtrage de texels entiers ;
- Autres méthodes de rendu ;

#### 5.2.2.1 Modèle de réflectance du voxel

Nous nous sommes limités à l'utilisation d'une NDF ellipsoïdale, similaire à celle de Neyret. La construction de cette NDF laisse à désirer : notre heuristique se base sur l'hypothèse qu'une certaine direction est plus importante que les autres, et qu'une seconde direction orthogonale à la première existe. Cette heuristique produit de bons résultats si un faible nombre de surfaces (de faible courbure) est contenu dans chaque voxel, ce qui est généralement le cas pour un octree

de haute résolution. Une meilleure solution consisterait, comme nous l'avons déjà mentionné, à ajuster les normales à la surface d'un ellipsoïde en utilisant la minimisation aux moindres carrés par exemple. Pratt [Pra87] décrit en détail la technique nécessaire pour obtenir de bons résultats.

Nous pourrions aussi utiliser d'autres types de représentation de la NDF. Nous avons brièvement décrit en section 3.3.1 les différentes représentations de la BRDF qui ont déjà été proposées, il serait nécessaire de les adapter à la NDF dans notre cas. Contentons-nous ici de mentionner ces différentes approches :

- Les courbes sinusoïdales (pics de Phong) [Fou92] ;
- Les harmoniques sphériques [SP94] ;
- Les ondelettes sphériques [SS95].

Enfin, nous pourrions choisir des fonctions de NDF adaptées aux types de scènes ou de géométries que nous désirons représenter. Nous perdrions ainsi en généralité, pour gagner en efficacité et en précision. Par exemple, Noma [Nom95] représente la réflectance d'un très grand nombre de sphères à plusieurs résolutions, en utilisant des texels contenant une sphère parfaitement diffuse dans chaque voxel de plus bas niveau. La réflectance qu'il utilise dans ce cas est très simple, elle est donnée par :

$$\Psi = \frac{2}{3\pi} k_d (\sin\theta + (\pi - \theta)\cos\theta) \tag{5.1}$$

où $\theta$ est l'angle entre la direction de la source lumineuse et la direction du point de vue. Bien sûr, représenter la réflectance analytiquement pour une géométrie arbitraire est impossible, mais si cette géométrie était connue à l'avance, il serait possible d'obtenir une telle représentation.

Nous pouvons penser à appliquer l'approche de Noma à notre cas. Nous considérons le texel comme un segment de mèche de cheveux. Ces segments, superposés et placés sur le crâne, forment une chevelure. Nous verrons plus loin les conditions géométriques nécessaires à obtenir de bons résultats, nous nous contentons ici de traiter la réflectance. Fixons la géométrie du texel à priori, choisissons par exemple de placer un cylindre $x^2 + y^2 = r^2$ dans chaque voxel de plus bas niveau, tel que l'illustre la figure 5.13. Nous pouvons ainsi déterminer la réflectance des voxels analytiquement. Au plus bas niveau, elle correspond au modèle d'illumination d'un cylindre, décrit par Kajiya *et al.* [KK89] :

$$I = I_d k_d \sin(\mathbf{t}, \mathbf{l}) + I_s k_s (\cos(\mathbf{t}, \mathbf{l})\cos(\mathbf{t}, \mathbf{d}) + \sin(\mathbf{t}, \mathbf{l})\sin(\mathbf{t}, \mathbf{d}))^n \tag{5.2}$$

où $\mathbf{t}$ est la tangente au cylindre, $\mathbf{l}$ la direction de la source lumineuse, et $\mathbf{d}$ la direction du point de vue. Aux niveaux supérieurs, il s'agit de trouver une approximation analytique de la réflectance moyenne de celle des voxels-enfants. Cette approche éliminerait le coût de construction et de conservation du volume de référence, et ramènerait le calcul du rendu à l'évaluation

du modèle d'illumination ci-dessus pour les voxels traversés par les rayons. Notons cependant qu'une telle spécialisation pourrait difficilement être utilisée pour modéliser des cheveux un peu plus complexes (tels que des cheveux boucles). Dans ces cas, il sera nécessaire de revenir à notre méthode de construction générale.

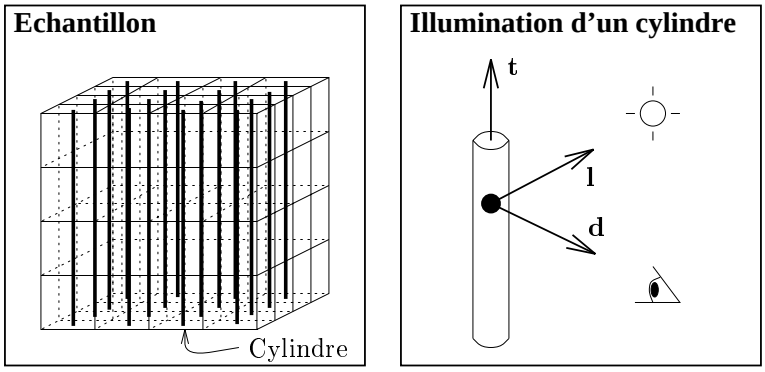


FIG. 5.13: Echantillon de mèche et modèle d'illumination d'un cylindre individuel.

#### 5.2.2.2 Tracé du rayon dans l'espace-objet

L'algorithme `ProchainVoxel` (section 4.3) effectue le tracé d'un rayon linéaire dans l'octree, en espace-objet. Cependant, le rayon n'est pas nécessairement linéaire dans cet espace, sa nature dépend de la déformation du texel. Comme nous l'avons vu, l'approximation linéaire du rayon peut être totalement incorrecte, et générer de fortes distortions dans l'image résultante.

La solution analytique consisterait à dériver, pour chaque type de déformation, la nature du rayon déformé, puis d'intersecter les faces des voxels avec ce rayon généralisé. Non seulement cette solution serait très complexe, mais en plus son calcul serait très coûteux. Une approximation de cette solution consiste à considérer le rayon linéaire à l'intérieur de chaque voxel individuel, mais de calculer le changement de direction du rayon à la sortie de chaque voxel en utilisant le jacobien de la déformation. Adapter l'algorithme `ProchainVoxel` à cette modification ne serait cependant pas simple, car cet algorithme suppose implicitement que le rayon demeure linéaire à l'intérieur du texel entier, ce qui permet de suivre l'approche de subdivision récursive. Une solution simple serait d'utiliser l'algorithme de Glassner [Gla84] pour la traversée d'un octree, algorithme moins efficace mais qui ne présuppose pas un rayon linéaire. Une solution plus complexe consisterait à modifier la méthode de subdivision du voxel en huit octants dans `ProchainVoxel` afin de prendre en compte la courbure du rayon.

Une autre limitation de notre méthode est que nous effectuons le tracé d'un rayon, et non d'un cône. Souvenons-nous que notre *MIP map* 3D utilise un cône pour décider du niveau de l'octree à traiter. Dans l'algorithme actuel `ProchainVoxel`, nous ne prenons en considération que

la direction du cône, et nous utilisons son ouverture à l'entrée du texel comme mesure du niveau de l'octree à choisir pour le rendu. Cependant, le cône peut intersecter plusieurs voxels simultanement pendant la traversée du texel (voir figure 4.3). Pour correctement calculer l'illumination du cône, il nous faudrait détecter tous ces voxels, calculer leur illumination individuelle, puis pondérer chacune par le pourcentage du voxel intersecté par le cône. L'algorithme de traversée serait cependant très complexe à concevoir, et sûrement très coûteux à éxécuter, surtout si nous désirons traiter aussi le cas de cônes non-linéaires.

#### 5.2.2.3 Recouvrement de plusieurs texels

Notre algorithme de rendu ne traite pas le cas de texels qui se recouvrent dans l'espace : un seul texel (le premier intersecté par le rayon) serait affiché. Il serait désirable de traiter le recouvrement de plusieurs texels pendant le rendu : l'animation et la modélisation en bénéficieraient, comme nous le verrons plus bas. Fusionner deux texels revient à détecter les intervalles qui leur sont communs sur le trajet du rayon. Sur ces intervalles, un problème se pose : chaque texel cache une partie de l'autre et affecte donc l'illumination de l'autre, tel que l'illustre la figure 5.14. Nous pouvons simplifier le problème en négligeant l'effet d'un texel sur l'autre au niveau des voxels individuels. Ainsi, l'algorithme de rendu pour les voxels individuels resterait le même, seule la traversée de l'octree serait modifiée de manière à détecter les intersections avec d'autres texels.

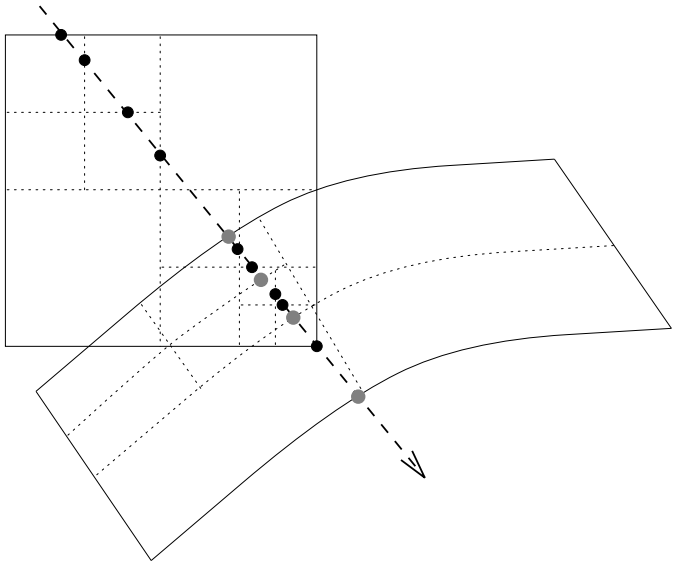

FIG. 5.14: Recouvrement de deux texels.

L'algorithme de traversée devra être modifié de façon importante pour traiter correctement le recouvrement de texels. D'abord, cette traversée devra être effectuée en espace-scène et non en espace-objet puisque chaque texel est déformé différemment. Il sera nécessaire, à chaque

intersection, de retransformer le rayon dans l'espace-objet correspondant au texel intersecté. En général, l'algorithme aura la forme suivante :

1. Pour chaque rayon lancé, les intersections avec tous les texels sont calculées.
2. S'il n'y a pas de chevauchement dans les intersections, nous revenons à notre algorithme de traversée `ProchainVoxel`.
3. Sinon, nous créons une pile de segments de rayon en espace-monde, contenant les points d'intersection avec les texels qui se chevauchent. Cette pile contient initialement les points d'entrée et de sortie du rayon pour chaque texel intersecté. Ces points sont triés selon leur distance de l'origine du rayon.
4. Nous subdivisons chaque texel en huit octants à la manière de `ProchainVoxel`. Cependant, les nouveaux segments ne correspondent plus nécessairement à des voxels entiers, comme l'illustre la figure 5.14. De plus, chaque segment réfère à tous les voxels des différents texels qu'il traverse.
5. Lorsque un segment correspond à un voxel de plus bas niveau, nous calculons l'illumination et l'opacité pour tous les voxels traversés par ce segment, et nous accumulons ces valeurs jusqu'à ce que l'occlusion soit complète.

#### 5.2.2.4 Généralisation du critère de subdivision pendant la traversée du texel

Lorsqu'un cône $R$ traverse le texel, nous choisissons le niveau de l'octree à traiter en comparant l'ouverture du cône avec la taille des voxels de chaque niveau. Nous pouvons considérer cette comparaison comme une fonction $N(R)$ qui détermine le niveau de l'octree à traiter. Cette formulation nous permet d'étudier d'autres critères de choix qui pourraient améliorer le rendu. Par exemple, nous pourrions déterminer le niveau désiré non seulement en fonction de la distance du point de vue, mais aussi en fonction de la distance des sources lumineuses. Ainsi, plus une source serait proche, plus il serait nécessaire de descendre à une résolution plus fine. De plus, nous pourrions séparer le calcul de l'illumination locale au voxel du calcul d'ombrage. Pour une source lointaine, nous pourrions effectuer le calcul d'ombrage à basse résolution, et effectuer le calcul d'illumination locale à plus haute résolution (sans avoir à recalculer l'ombrage à chaque voxel).

#### 5.2.2.5 Intégration de la géométrie

Une autre amélioration consiste à intégrer la géométrie à l'algorithme de rendu des texels, au cas où le point de vue serait très proche. Ce cas est détecté par le fait que l'ouverture du rayon au point d'intersection avec le texel est inférieure à la taille des voxels de plus bas niveau. A ce moment, nous pourrions intersecter le rayon avec les primitives qui constituent le voxel, afin de calculer l'illumination sur ces primitives directement. Un traitement *MIP map* serait nécessaire ici aussi pour assurer une continuité entre le plus bas niveau du texel et la géométrie ;

une interpolation linéaire basée sur l'ouverture du rayon serait possible, telle que décrite à la section 4.6.

Le véritable problème qui se pose ici est dû au fait que ces primitives seront déformées par la déformation agissant sur le texel : si cette déformation est affine, une simple transformation inverse du rayon sera suffisante. Cependant, si nous utilisons des déformations plus fortes, il faudra effectuer un coûteux lancer de rayon dans un espace déformé (voir par exemple [Bar86]).

#### 5.2.2.6 Meilleure représentation de la visibilité

Rappelons que notre approximation de la visibilité du voxel consiste en un pourcentage d'occlusion pour chaque axe $\{X, Y, Z\}$ du repère-monde. Pour évaluer l'occlusion dans une direction $\mathbf{d}$ donnée, nous évaluons le produit scalaire $\mathbf{d} \cdot \mathbf{v}$, où $\mathbf{v}$ est le vecteur contenant les pourcentages d'occlusion axiaux. Cette approximation est basée sur l'hypothèse que la visibilité varie de façon continue entre les axes, ce qui est en général incorrect. Nous pouvons mieux approximer la fonction d'occlusion du voxel par une fonction sur une sphère $v(\theta, \phi)$ au centre du voxel, ce qui nous permettrait de l'approximer de la même manière que nous approximons la NDF, en échantillonnant la visibilité dans toutes les directions.

Nous pouvons aussi améliorer la représentation des matériaux du voxel, en les reliant à la visibilité. L'information que nous conservons actuellement est un matériau "moyen" obtenu par filtrage des matériaux contenus dans le voxel. Nous pouvons penser à associer à chaque direction de visibilité le matériau qui est visible dans cette direction. Ainsi, nous pourrions effectuer un rendu plus précis. La fonction de visibilité $v(\theta, \phi)$ retournerait alors une paire $(\mathbf{v}, M)$ pour toute direction $(\theta, \phi)$[4].

Une optimisation possible se situe au niveau de l'illumination du voxel (Algorithme `IlluminerVoxel`, section 4.4). Pendant ce calcul, nous lançons un cône d'ombrage à partir du centre du voxel vers chaque source lumineuse. Ce cône doit d'abord accumuler l'opacité dans le texel courant avant d'être lancé dans la scène. Nous pourrions créer une table d'opacités pour toutes les directions, dans chaque voxel situé sur les bords du texel, comme l'illustre la figure 5.15 (a) (pour un seul voxel). Cette table serait créée par échantillonnage du volume de référence : à chaque voxel du bord, nous lançons des rayons dans toutes les directions de l'hémisphère inférieure du voxel (la moitié de ces échantillons seront redondants puisque l'atténuation est commutative) afin d'accumuler les atténuations des voxels dans ces directions, puis nous conservons les atténuations ainsi calculées dans une fonction $Op(\theta, \phi)$. Lors du rendu, nous désirons lancer un cône d'ombrage à partir d'un voxel dans la direction d'une source lumineuse donnée. Nous trouvons le voxel du bord touché par ce cône, puis nous consultons sa

[4] En fait, l'implantation logicielle de notre méthode définit déjà une telle fonction $v(\theta, \phi)$, qui retourne toujours le matériau moyen pour le moment.

table d'opacités pour trouver celle qui correspond à la direction du cône. Cette valeur représente l'opacité accumulée par un rayon traversant le texel de part en part, il faut donc la pondérer par la distance entre le voxel de départ et le bord du texel (la figure 5.15 (b) illustre ce processus). Cette optimisation ne serait valable que pour une distribution uniforme de l'opacité à l'intérieur du texel. Elle pourrait cependant accélérer le rendu de façon importante, puisque le nombre de rayons d'ombrage lancés est proportionnel au produit du nombre de voxels traversés par le nombre de sources lumineuses dans la scène. Nous pourrions aussi conserver l'opacité accumulée dans toutes les directions pour tous les voxels du volume, afin d'éliminer la restriction d'uniformité, mais le coût en mémoire serait encore plus important.

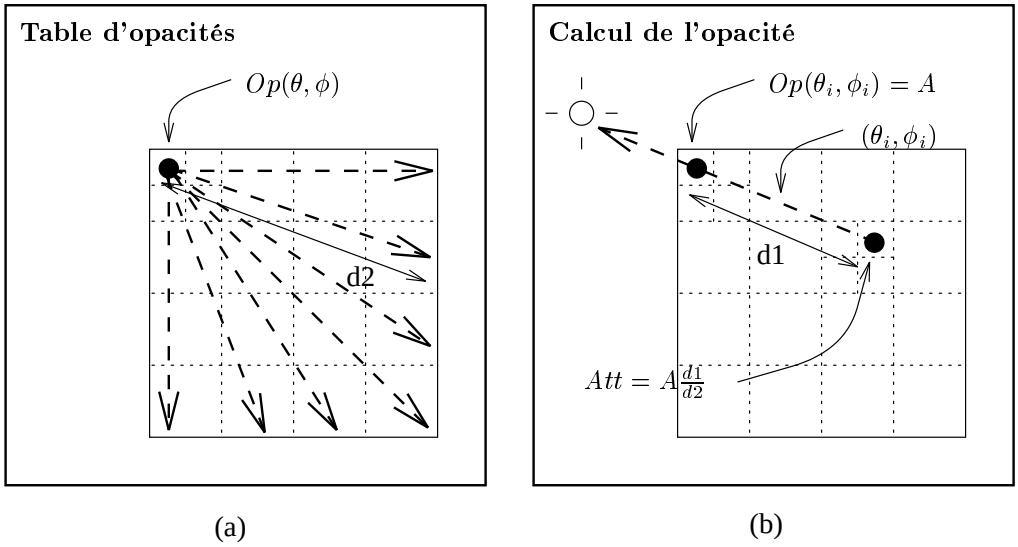


FIG. 5.15: Calcul et utilisation de la table d'opacités.

#### 5.2.2.7 Filtrage de texels entiers

Une limitation de notre méthode apparaît lorsque les texels sont placés très loin du point de vue. Dans ce cas, plusieurs texels entiers seront projetés dans un même pixel, causant à nouveau de l'aliassage. Nous pouvons surmonter ce problème dans notre contexte de lancer de cônes. Si nous calculons le pourcentage de l'aire du cône occupée par chaque texel, nous pouvons calculer l'illumination pour chacun des texels, puis produire la moyenne des illuminations pondérées par les pourcentages respectifs. Nous effectuons ainsi une sorte de filtrage sur les texels comme s'ils étaient des objets standards. Toutes les techniques de filtrage sont alors envisageables [Hec89].

Cependant, cette solution ne peut être utilisée dans le cas où un grand nombre de texels sont projetés dans le même pixel, car le coût du calcul serait excessif. Nous pouvons penser à réutiliser notre concept de texture multi-résolution pour résoudre ce problème d'une manière plus générale. Si nous créons un texel "virtuel" dont les feuilles contiennent eux-mêmes des

texels, nous pouvons choisir le niveau de résolution adapté au rendu, comme nous le faisons actuellement. La seule difficulté ici est de créer les niveaux supérieurs de ce texel virtuel, car il faut pour cela pouvoir filtrer les texels entiers en pré-calcul, afin de ne pas avoir à descendre aux feuilles durant le rendu (ce qui équivaudrait à la première solution). Filtrer deux texels ne pose conceptuellement pas de problème, il suffit de moyenner les voxels qui correspondent à la même position dans les deux octrees respectifs. Cependant, nous créerions ainsi un nouveau volume de référence, ce qui alourdirait le coût de la méthode en mémoire. Il faudrait alors s'assurer de la répétition de ce nouveau texel, pour qu'il puisse être réutilisé et ainsi justifier ce coût additionnel.

#### 5.2.2.8 Autres méthodes de rendu

La texture volumique peut être adaptée à d'autres méthodes de rendu que le lancer de rayons. L'idée principale est que, indépendamment de l'algorithme de rendu utilisé, la texture volumique doit être traversée par un cône pour calculer l'illumination d'un point de vue donné. Nous allons brièvement discuter ici de l'intégration de la texture volumique aux méthodes de rendu suivantes : le Z-buffer et le A-buffer, les arbres BSP, et le balayage de lignes (*scan-line conversion*).

La technique du Z-buffer [Cat74] est l'une des plus simples méthodes de rendu par détermination des surfaces visibles. Pour chaque pixel de l'écran, nous conservons la profondeur $z$ de l'objet le plus proche de l'écran. La couleur du pixel est donc déterminée en comparant la distance entre chaque objet de la scène et ce pixel, la couleur finale étant celle de l'objet le plus proche.

Dans notre cas, le texel est un objet de la scène, nous pouvons donc déterminer la distance entre le texel et l'écran. S'il est au premier plan, nous pouvons alors le traverser par un cône lancé à partir du pixel, tel que nous le faisons actuellement, et donner au pixel la couleur ainsi calculée. Le calcul d'ombrage à chaque voxel peut être effectué en créant un Z-buffer pour chaque source lumineuse [Wil78]. Ce nouveau Z-buffer sert à déterminer si un objet bloque l'illumination du voxel. Si ce n'est pas le cas, l'atténuation d'ombrage due au texel lui-même peut être calculée en traversant le texel avec un cône ou en utilisant la table d'opacités décrite plus haut.

Les arbres BSP [FKN80, FAG83] servent à partionner l'espace-scène de façon irrégulière, pour des scènes statiques. Chaque polygone de la scène définit un plan qui divise l'espace en deux demi-espaces, et qui peut potentiellement couper tous les autres polygones de la scène. Chaque polygone des demi-espaces définit à son tour un autre plan de subdivision, et ainsi de suite. Nous obtenons un arbre qui peut être utilisé pour déterminer les polygones visibles d'un point de vue arbitraire en temps linéaire.

Dans notre cas, nous pouvons représenter les parois externes du texel par des polygones, qui,

en plus de se comporter comme les autres polygones de la scène, pointent aussi vers le texel. Ainsi, lorsque l'algorithme de BSP demande à afficher un polygone appartenant au texel, notre méthode de rendu peut être activée, sans grandes modifications à la méthode actuelle. Le calcul d'ombrage à chaque voxel peut être effectué en considérant la source lumineuse comme le point de vue, ce qui permet de déterminer si des polygones bloquent l'illumination du voxel [CF89].

La technique de balayage de lignes [BK70, Bou70, Wat70] affiche les polygones une ligne à la fois, en exploitant la cohérence entre les lignes successives et entre les arêtes des polygones. Une liste d'arêtes triée selon leurs coordonnées $y$ (après projection sur l'écran) sert à déterminer rapidement les polygones à afficher à chaque ligne. Une liste d'arêtes "actives" pour chaque ligne est triée selon leurs coordonnées $x$ pour déterminer le polygone à afficher à chaque pixel.

Ici aussi, nous pouvons créer des polygones représentant les faces des texels. Ces polygones participent à l'algorithme de balayage de lignes sans modification, à l'exception de l'affichage, où le rendu volumique du texel détermine la couleur à chaque pixel. Nous pouvons exploiter la cohérence pour rapidement déterminer la traversée du texel par un cône, d'un pixel au pixel voisin. Nous pouvons aussi exploiter la cohérence pour la détermination du niveau de l'octree à afficher. Le calcul d'ombrage à chaque voxel peut être effectué en considérant la source lumineuse comme un autre point de vue, où nous appliquons le balayage de lignes à nouveau [NTN92]. Ce balayage d'ombrage aurait cependant un coût très élevé.

### 5.2.3 Modélisation

Nous proposons ici et à la section suivante un modèle de chevelure utilisant la texture volumique comme fondation. Nous avons vu que la chevelure représente un défi en infographie, à cause du très grand nombre de primitives géométriques nécessaires pour modéliser une chevelure réaliste. Notre modèle se base sur l'idée, déjà exploitée, de traiter non pas des cheveux individuels mais de mèches entières. Watanabe *et al.* [WS92] définissent la mèche comme étant un ensemble de $m$ copies d'un cheveu directeur géométrique. Notre approche est différente : nous allons créer un *échantillon* de mèche, représentant typiquement une section de la mèche. Cet échantillon est ensuite transformé en texture volumique, soit par construction tel que décrit au chapitre 3, soit analytiquement à la façon de Noma [Nom95]. Nous définissons une mèche comme étant une arborescence des copies de cet échantillon (les texels). La mèche débute par un segment sur le crâne, puis elle peut se subdiviser en deux segments, qui peuvent eux-mêmes se subdiviser plus loin. Cette structure arborescente offre plus de flexibilité que si nous n'utilisions qu'une seule chaîne pour toute la mèche, surtout qu'il est possible de désirer plus de détails (et donc plus de segments) sur la partie externe de la chevelure qu'à l'intérieur. Chaque segment est représenté par une chaîne de texels. La direction d'un segment est définie par une trajectoire passant par le centre des texels. Sur un segment, chaque paire de texels adjacents se partagent une face

commune, et la trajectoire du segment définit la déformation de chaque texel individuel, tel que l'illustre la figure 5.16.

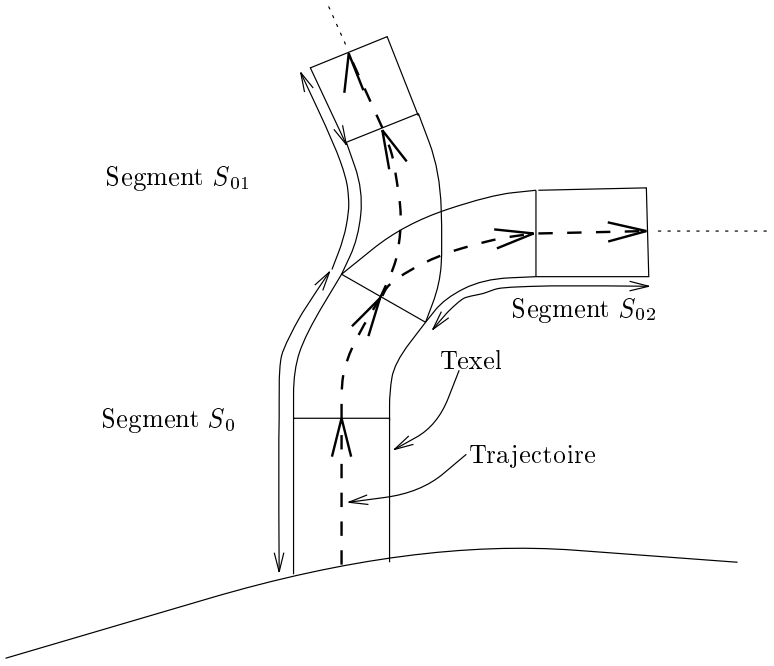


FIG. 5.16: Modèle de mèches.

Plusieurs points sont à prendre en considération ici. D'abord, une chevelure composée de mèches provenant d'un seul échantillon se révèlera sans doute contraignant : nous avons besoin de plus de flexibilité dans la modélisation. Plusieurs solutions s'offrent à nous. Nous pouvons créer plusieurs échantillons, et les utiliser comme autant de "blocs de construction" pour modéliser la chevelure. Cette approche peut être coûteuse en mémoire si la chevelure désirée exhibe une grande variété de configurations, ce qui nécessitera un grand nombre d'échantillons différents. Une seconde solution plus complexe consisterait à fusionner plusieurs échantillons afin de créer l'illusion de la diversité. Nous avons présenté à la section précédente l'esquisse d'un algorithme de rendu qui accomoderait les texels fusionnés.

Anjyo *et al.* [AUK92] proposent une approche intéressante pour faciliter la création de coiffures complexes. Nous avons décrit leur méthode à la section 2.2.3, rappelons ici ses points principaux. Les auteurs utilisent un modèle de forces statiques agissant sur une position de départ des cheveux, et simulent le système obtenu jusqu'à l'équilibre, qui définit la coiffure désirée. Nous pouvons utiliser la même approche, qui s'intégre naturellement au modèle d'animation dont nous présentons l'ébauche à la section suivante.

Notons aussi l'importance du choix de la famille de déformations agissant sur les texels. En effet, la trajectoire d'un segment de mèche sera approximée par la déformation des texels. Plus cette déformation est flexible, plus l'approximation sera proche de la trajectoire désirée.

Cependant, nous avons vu que la déformation du texel influe sur le rendu, et que de fortes déformations peuvent causer des difficultés à ce niveau. Nous avons présenté à la section précédente une modification de l'algorithme de rendu qui accomoderait de telles déformations.

Un autre point important est la continuité de la géométrie contenue dans les texels. Dans le cas des cheveux, nous désirons que chaque cheveu individuel dans une mèche exhibe une continuité $C^1$ au moins, pour assurer un résultat visuel plaisant. Plusieurs facteurs entrent en jeu ici. D'abord, la géométrie contenue dans le texel (c'est-à-dire les cheveux individuels) doit être cyclique sur les faces qui seront communes dans une mèche. Cette condition garantit la continuité $C^0$. La continuité $C^1$ est plus problématique. Prenons l'exemple d'une déformation affine par exemple : entre chaque paire de texels adjacents, la géométrie exhibera une discontinuité du premier degré. Il s'agit alors de trouver une famille de déformations qui permet de conserver cette continuité. Le point-clé ici est que la déformation d'un texel individuel ne *peut pas* suffire à assurer une continuité entre deux texels adjacents, il faut aussi que les déformations de ces deux texels s'accordent à garantir cette continuité. Nous pouvons penser à utiliser un modèle de splines cubiques ou quadratiques sur la trajectoire de la mèche pour obtenir le résultat désiré, tel que l'illustre la figure 5.17. Cependant, cette approche nécessite le développement d'une déformation spécialement adaptée, qui soit respectivement cubique ou quadratique dans la direction de la trajectoire, et au moins linéaire dans les deux autres dimensions. Une déformation FFD [SP86] pourrait potentiellement remplir cette condition.

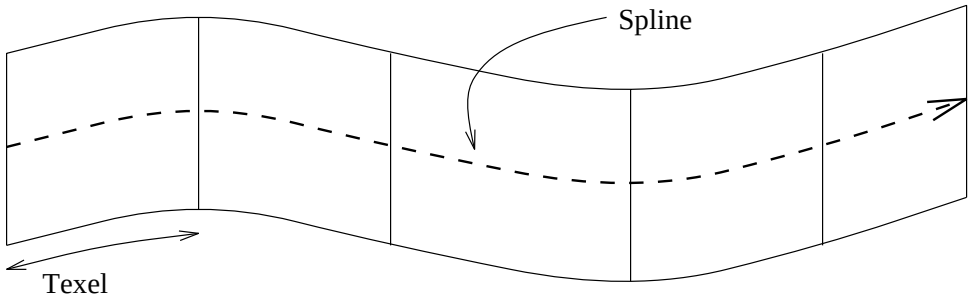


FIG. 5.17: Une spline sur la trajectoire de la mèche définit la déformation des texels.

Enfin, le processus de modélisation de la chevelure ne peut être effectué de façon pratique sans logiciel de modélisation dédié. Ce logiciel doit offrir les options suivantes :

**L'importation d'un modèle de crâne** sur lequel la chevelure sera placée ;

**La spécification de la position des mèches sur le crâne.** Cette spécification pourrait être effectuée manuellement (par l'usager), automatiquement (selon un ensemble de règles déterminant le positionnement des mèches), ou par une combinaison de ces deux méthodes ;

**La spécification de la trajectoire des mèches.** Une fois de plus, cette spécification pourrait être effectuée selon un processus manuel et/ou automatique ;

**La création d'échantillons de mèches.** L'utilisateur peut créer une géométrie (à l'aide d'outils de description de scène standards) qui servira d'échantillon pour la chevelure.

**La création des mèches à partir des échantillons.** Une fois tous les paramètres des mèches et des échantillons spécifiés, le logiciel doit créer la chevelure. Pour chaque mèche, la déformation des texels est ajustée de manière à obtenir la meilleure approximation de la trajectoire spécifiée.

### 5.2.4 Animation

Nous présentons ici l'ébauche d'un modèle d'animation de la chevelure spécialement adapté aux texels[5]. Plus précisément, ce modèle considère la mèche de cheveux comme une arborescence spatiale de cubes déformés. Examinons les objectifs de la simulation :

- La modélisation d'une coiffure complexe au repos et dynamiquement ;
- La détection de collisions entre la chevelure et le corps ;
- Le traitement de la collision et du frottement entre mèches.

#### 5.2.4.1 Description du modèle

Le modèle proposé est le suivant : nous animons une particule orientée (c'est-à-dire une masse munie d'un repère local) sur chaque face raccordant deux texels. L'un des axes du repère est aligné avec la trajectoire du segment de la mèche, les deux autres axes définissent la torsion du texel. Les particules sont reliées par des ressorts assez rigides le long de la mèche. D'autres ressorts placés sur les faces communes contrôlent la torsion des mèches, ces ressorts relient les valeurs des angles de rotation des texels consécutifs. La figure 5.18 illustre ce modèle.

Nous pouvons utiliser ce modèle pour créer des coiffures complexes de deux manières différentes. Nous pouvons appliquer des forces de départ sur les particules afin d'obtenir une position de repos qui définit la coiffure, à la manière d'Anjyo *et al.* [AUK92]. Nous pouvons aussi manipuler directement les repères locaux des particules orientées ainsi que la trajectoire globale des segments des mèches afin d'avoir un plus grand contrôle sur le résultat final. En fait, notre outil de modélisation décrit à la section précédente devrait présenter ces deux options à l'utilisateur. En un premier temps, celui-ci spécifierait les forces agissant sur la chevelure, afin d'obtenir une coiffure préliminaire. Ensuite, il ajusterait manuellement les mèches pour obtenir le résultat final.

Le traitement des collisions entre les mèches et avec le corps peut être effectué simplement en définissant une zone (sphérique ou cylindrique) où sont appliquées des forces répulsives autour de chaque texel. Les forces de répulsion sont calculées en fonction de la distance entre la mèche et l'objet voisin (soit une autre mèche soit le corps). Nous procédons de même pour le traitement

[5]Ce modèle a été développé conjointement avec Marie-Paule Cani-Gascuel, de l'équipe iMAGIS, laboratoire GRAVIR/IMAG, INRIA, France.

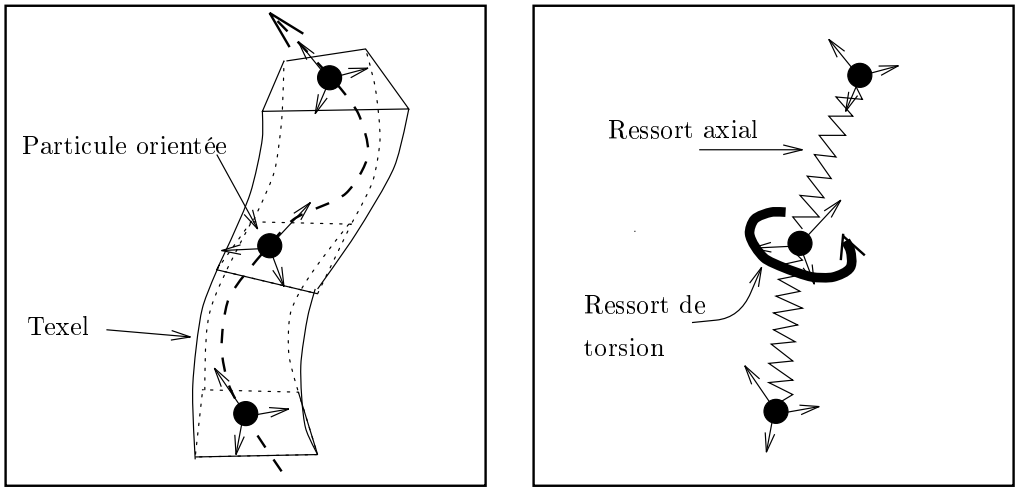


FIG. 5.18: Modèle d'animation.

du frottement entre deux mèches. Le frottement résulte de la différence de vitesse entre les deux mèches ; soient $v_1, v_2$ leurs vitesses et $c_1, c_2$ leurs coefficients de frottement respectivement, alors la force de frottement $F_1 = c_1 c_2 (v_2 - v_1)$ et $F_2 = -F_1$. Nous créons une zone de frottement autour de chaque mèche où ces forces agissent, cette zone doit englober la zone de collision pour eviter que les mèches ne se repoussent.

#### 5.2.4.2 Intégration au modèle de texels

La partie critique ici est l'intégration de ce modèle d'animation avec le rendu des texels. En effet, le rendu traite de texels déformés par une famille spécifique de déformations spatiales. D'autre part, chaque étape de l'animation produit une nouvelle position et orientation de particules orientées. Il faut alors traduire cette information en une déformation appartenant à la famille spécifiée. Nous pouvons décomposer ce problème en deux parties. D'abord, étant données la position des particules et leur orientation dans l'axe de la mèche, il faut trouver la trajectoire de la mèche. Dans notre modèle de splines décrit plus haut, cette trajectoire pourrait correspondre à une courbe d'energie minimale satisfaisant les contraintes de position et de direction (voir par exemple [Hor83, BN90, MS93, VW95]). Ensuite, étant données cette trajectoire et l'information sur la torsion des texels individuels (définie par le repère des particules orientées), il faut trouver une déformation spatiale pour chaque texel, qui approxime la trajectoire et la torsion. La figure 5.19 illustre ce processus.

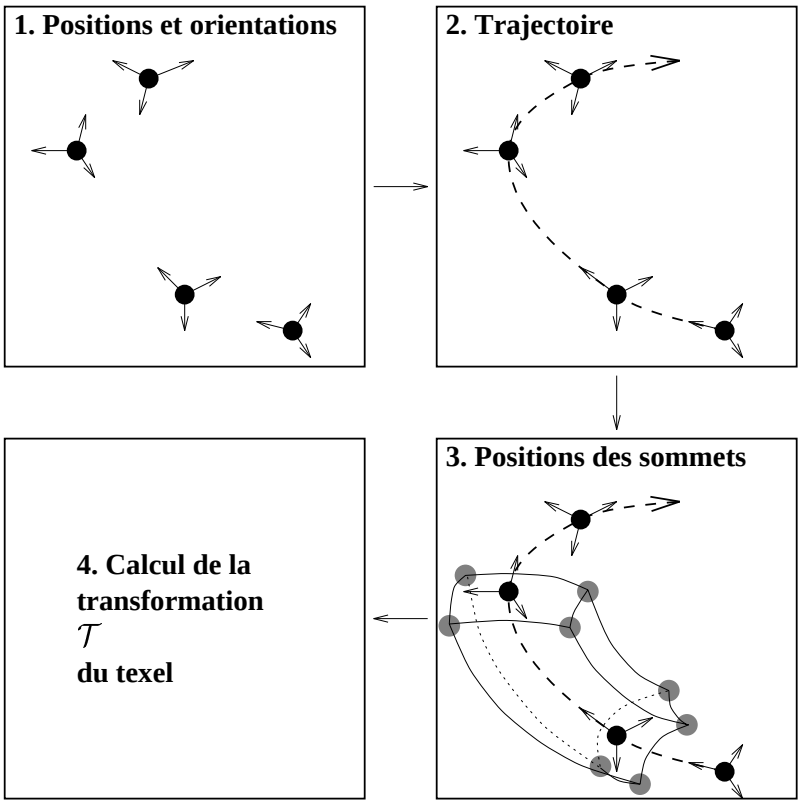


FIG. 5.19: Conversion des positions et orientations des particules en déformations de texels.

# Chapitre 6

# Conclusion

*Nous terminons ici par un résumé de notre méthode, puis nous la replaçons dans le contexte global du problème de la complexité en infographie.*

Nous avons présenté une méthode multi-résolution pour l'affichage efficace et visuellement acceptable de scènes complexes exhibant une grande répétitivité. Cette méthode a pour but d'éliminer l'élément géométrique de répétitivité de ces scènes, afin de le remplacer par un volume contenant l'information de réflectance de cette géométrie à plusieurs niveaux de résolution. Ce volume consiste en une structure de subdivision spatiale hiérarchique, l'octree, dont les feuilles contiennent l'information de réflectance obtenue par échantillonnage de la géométrie. Les niveaux supérieurs de l'octree sont obtenus par filtrage successifs des niveaux inférieurs.

Une scène composée de copies virtuelles de ce volume, les texels, peut alors être créée pour remplacer la scène originale qui contient les objets géométriques. Les texels sont affichés par un rendu volumique, durant lequel des cônes traversent le volume de référence afin d'accumuler l'illumination dans les cellules de ce volume. Selon la distance entre le texel et le point de vue, le niveau approprié de l'octree est choisi pour le calcul de l'illumination. Si le texel est trop proche du point de vue, le résultat visuel pourrait en souffrir, c'est pourquoi la géométrie originale peut être affichée dans ce cas.

Cette approche présente une solution au problème de la complexité géométrique des scènes. Le volume de référence est une représentation approximative de l'élément de répétitivité, mais le degré de cette approximation peut être contrôlé de façon à obtenir les résultats désirés. Le gain en temps de calcul est particulièrement important lorsque l'élément de répétitivité est un objet géométrique complexe. En effet, le coût d'affichage du volume est borné par sa résolution maximale, alors que le coût d'affichage de la géométrie est borné par sa complexité. Nous imposons ainsi une borne supérieure à l'affichage de la scène, sans pour autant perdre de l'information. De plus, cette approche présente une solution au problème de l'aliassage dû à

une géométrie microscopique. Cette géométrie est remplacée par l'information de la réflectance qu'elle génère, et cette réflectance est représentée à plusieurs résolutions pour choisir le degré de précision qui minimise l'aliassage durant le rendu.

Notre approche s'inscrit au sein de la méthodologie de la représentation multi-échelle des objets. Cette méthodologie repose sur le principe qu'un objet peut être représenté à plusieurs niveaux, en fonction de la quantité de détails désirés. La micro-structure de l'objet est une représentation qui ignore les détails microscopiques. Cette représentation consiste généralement de maillages polygonaux ou de fonctions algébriques représentant des objets géométriques. A un niveau inférieur, la micro-structure de l'objet capture les détails de celui-ci. Ces détails peuvent être représentés de plusieurs façons. Crow (cite dans [Kaj85]) propose l'utilisation de modèles géométriques locaux au-dessus de la surface macroscopique de l'objet. Cette approche est cependant coûteuse, et génère beaucoup d'aliassage. Un nombre d'approximations sont donc proposées. Le *displacement-map* [Coo84, CCC87] définit l'elevation locale de chaque point au-dessus de la surface macroscopique. Le *bump-map* [Bli77] définit la perturbation de chaque normale à la surface. La BRDF [TS67] définit l'intensité lumineuse réfléchie dans chaque direction pour chaque direction incidente. La BRDF est une fonction coûteuse à conserver, et un grand nombre de modèles sont proposés pour représenter la BRDF de certaines configurations géométriques spécifiques. Par exemple, Poulin *et al.* [PF90] proposent un modèle hiérarchique de cylindres invisibles qui génèrent une BRDF anisotropique.

Chaque représentation citée ici représente un niveau de détails spécifique dans la représentation hiérarchique des objets. Cependant, le modèle hiérarchique global ainsi obtenu n'est pas continu, car chaque représentation est indépendante des autres. Becker *et al.* [BM93] proposent un modèle continu qui unifie le *displacement-map*, *bump-map*, et la BRDF. Cependant, leur modèle n'est pas complet, l'ombrage en étant notamment absent. Notre approche propose une autre vue de cette représentation hiérarchique. Le volume de référence que nous utilisons est l'incarnation de toute la hiérarchie de détails. Prenons par exemple le cas de la chevelure : le maillage qui forme le crâne représente un modèle géométrique macroscopique, et les mèches de cheveux constituent les détails. Ces détails sont représentés à plusieurs résolutions dans les différents niveaux du volume de référence. De plus, la représentation géométrique des détails eux-mêmes peut être utilisée lorsque le point de vue est proche. Nous obtenons ici une nouvelle sorte de hiérarchie récursive qui illustre la flexibilité de notre approche. La texture volumique représente les détails répétitifs d'un objet. Dans l'octree du volume, ces détails sont conservés sous forme de modèles de réflectance multi-échelle. La géométrie des détails est aussi disponible, elle contient à son tour d'autres détails répétitifs, qui sont représentés par un autre volume de référence, et ainsi de suite. La hiérarchie obtenue ainsi permet de représenter les objets à un niveau de détail arbitraire, à l'opposé des méthodes vues plus haut, où chacune est limitée à un niveau spécifique. De plus, notre hiérarchie assure un passage continu d'un niveau à l'autre,

grâce au filtrage que nous effectuons.

En conclusion, la texture volumique est un outil puissant pour modéliser les scènes répétitives complexes qui comportent un grand nombre de détails. Dans cet ouvrage, nous avons présente un modèle de base de la texture volumique, puis nous avons présente un grand nombre d'extensions qui seraient nécessaires pour obtenir de meilleurs résultats. Nous avons aussi étudié l'utilisation de la texture volumique dans le rendu, la modélisation et l'animation de la chevelure, qui est un domaine où la plupart des techniques classiques d'infographie produisent des résultats visuellement insatisfaisants. Enfin, nous avons construit un logiciel de rendu flexible et général pour permettre une investigation plus profonde des nombreuses possibilités de cette approche.

# Bibliographie


[AUK92] Ken Anjyo, Yoshiaki Usami et Tsuneya Kurihara. « A simple method for extracting the natural beauty of hair ». *Computer Graphics*, volume 26, numéro 2, pages 111–120, juillet 1992.

[Bar86] Alan H. Barr. « Ray Tracing Deformed Surfaces ». In David C. Evans et Russell J. Athay, éditeurs. *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 287–296, août 1986.

[BK70] W. J. Bouknight et K. C. Kelly. « An algorithm for producing half-tone computer graphics presentations with shadows and movable light sources ». In *Proc. AFIPS JSCC*, volume 36, pages 1–10, 1970.

[Bli77] James F. Blinn. « Models of Light Reflection For Computer Synthesized Pictures ». In James George, éditeur. *Computer Graphics (SIGGRAPH '77 Proceedings)*, volume 11, pages 192–198, juillet 1977.

[BM93] Barry G. Becker et Nelson L. Max. « Smooth Transitions between Bump Rendering Algorithms ». In James T. Kajiya, éditeur. *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 183–190, août 1993.

[BN90] Alfred M. Bruckstein et Arun N. Netravali. « On minimal energy trajectories ». *Computer Vision, Graphics, and Image Processing*, volume 49, numéro 3, pages 283–296, mars 1990.

[Bou70] W. Jack Bouknight. « A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Presentations ». *Communications of the ACM*, septembre 1970.

[Car84] Loren Carpenter. « The A-buffer, an Antialiased Hidden Surface Method ». In Hank Christiansen, éditeur. *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 103–108, juillet 1984.

[Cat74] Edwin E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.D. thesis, Dept. of CS, U. of Utah, décembre 1974. bicubic patches, 1st texture mapping.

[CCC87] Robert L. Cook, Loren Carpenter et Edwin Catmull. « The Reyes Image Rendering Architecture ». In Maureen C. Stone, éditeur. *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 95–102, juillet 1987.

[CF89] Norman Chin et Steven Feiner. « Near Real-Time Shadow Generation Using BSP Trees ». In Jeffrey Lane, éditeur. *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 99–106, juillet 1989.

[Coo84] Robert L. Cook. « Shade trees ». In Hank Christiansen, éditeur. *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 223–231, juillet 1984.

[DTKT93] Agnes Daldegan, Nadia Magnenat Thalmann, Tsuneya Kurihara et Daniel Thalmann. « An Integrated System for Modeling, Animating and Rendering Hair ». In R. J. Hubbold et R. Juan, éditeurs. *Eurographics '93*, pages 211–221, Oxford, UK, 1993. Eurographics, Blackwell Publishers.

[FAG83] H. Fuchs, G. D. Abram et E. D. Grant. « Near real-time shaded display of rigid objects ». In *Computer Graphics (SIGGRAPH '83 Proceedings)*, volume 17, pages 65–72, juillet 1983.

[FKN80] H. Fuchs, Z. M. Kedem et B. F. Naylor. « On Visible Surface Generation by a Priori Tree Structures ». In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14, pages 124–133, juillet 1980.

[Fou92] Alain Fournier. « Normal distribution functions and multiple surfaces ». In *Graphics Interface '92 Workshop on Local Illumination*, pages 45–52, mai 1992.

[Gla84] Andrew S. Glassner. « Space Subdivision For Fast Ray Tracing ». *IEEE Computer Graphics and Applications*, volume 4, numéro 10, pages 15–22, octobre 1984.

[Gla89] Andrew (editor) Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989.

[Hec89] Paul S. Heckbert. « Fundamentals of Texture Mapping and Image Warping ». M.sc. thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, juin 1989.

[Hor83] B. Horn. « The Curve of Least Energy ». *ACM Transactions on Mathematical Software*, volume 9, numéro 4, 1983.

[Kaj85] James T. Kajiya. « Anisotropic Reflection Models ». In B. A. Barsky, éditeur. *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 15–21, juillet 1985.

[KK89] James T. Kajiya et Timothy L. Kay. « Rendering Fur with Three Dimensional Textures ». *Computer Graphics*, volume 23, numéro 3, pages 271–280, juillet 1989.

[LTT91] A.M. LeBlanc, R. Turner et D. Thalmann. « Rendering Hair using Pixel Blending and Shadow Buffers ». *The Journal of Visualization and Computer Animation*, volume 2, numéro 3, pages 92–97, 1991.

[Mil88] Gavin S. P. Miller. « From wire-frames to furry animals ». In *Proceedings of Graphics Interface '88*, pages 138–145, juin 1988.

[MS93] H. P. Moreton et C. H. Sequin. « Scale-invariant minimum-cost curves : fair and robust design implements ». *Computer Graphics Forum*, volume 12, numéro 3, pages C473–C484, septembre 1993.

[Ney95a] Fabrice Neyret. « Animated texels ». In *Eurographics Workshop On Animation And Simulation '95*, pages 97–103, septembre 1995.

[Ney95b] Fabrice Neyret. « A General and Multiscale Method for Volumetric Textures ». In *Graphics Interface '95 Proceedings*, pages 83–91, juin 1995.

[Ney95c] Fabrice Neyret. « Local Illumination in Deformed Space ». Rapport technique 2856, INRIA, avril 1995.

[Ney96a] Fabrice Neyret. « Synthesizing Verdant Landscapes using Volumetric Textures ». In *Eurographics Workshop on Rendering '96*, juin 1996.

[Ney96b] Fabrice Neyret. *Textures volumiques pour la synthese d'images.* Thèse de doctorat, Universite Paris XI, U.F.R. d'Informatique, 1996.

[Nom95] Tsukasa Noma. « Bridging Between Surface Rendering And Volume Rendering For Multi-Resolution Display ». In *6th Eurographics Workshop On Rendering*, juin 1995.

[NTN92] T. Nishita, S. Takita et E. Nakamae. « A Shading Model of Parallel Cylindrical Light Sources ». In *Visual Computing (Proceedings of CG International '92).* Springer-Verlag, 1992.

[PF90] Pierre Poulin et Alain Fournier. « A Model for Anisotropic Reflection ». *Computer Graphics*, volume 24, numéro 4, pages 273–282, août 1990.

[PH89] Ken Perlin et Eric M. Hoffert. « Hypertexture ». *Computer Graphics*, volume 23, numéro 3, pages 253–262, juillet 1989.

[Pho75] Bui-T. Phong. « Illumination for Computer Generated Pictures ». *Communications of the ACM*, volume 18, numéro 6, pages 311–317, juin 1975.

[Pra87] Vaughan Pratt. « Direct Least-Squares Fitting of Algebraic Surfaces ». *Computer Graphics*, volume 21, numéro 4, pages 145–152, juillet 1987.

[RCI91] R.E. Rosenblum, W. E. Carlson et E. Tripp III. « Simulating the Structure and Dynamics of Human Hair : Modelling, Rendering and Animation ». *The Journal of Visualization and Computer Animation*, volume 2, numéro 4, pages 141–148, 1991.

[Rob88] Clarence R. Robbins. *Chemical and Physical Behavior of Human Hair.* Springer-Verlag Inc., New York, second édition, 1988.

[RSC87] William T. Reeves, David H. Salesin et Robert L. Cook. « Rendering Antialiased Shadows with Depth Maps ». In Maureen C. Stone, éditeur. *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 283–291, juillet 1987.

[Sam90a] Hanan Samet. *Applications of Spatial Data Structures : Computer Graphics, Image Processing, and GIS.* Addison-Wesley, 1990.

[Sam90b] Hanan Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[Shi92] Mikio Shinya. « Hierarchical 3D texture ». In *Graphics Interface '92 Workshop on Local Illumination*, pages 61–67, mai 1992.

[SP86] Thomas W. Sederberg et Scott R. Parry. « Free-Form Deformation of Solid Geometric Models ». In David C. Evans et Russell J. Athay, éditeurs. *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 151–160, août 1986.

[SP94] François Sillion et Claude Puech. *Radiosity and Global Illumination.* Morgan Kaufmann, San Francisco, 1994. excellent coverage of radiosity and global illumination algorithms.

[SPS96] A. Sourin, A. Pasko et V. Savchenko. « Using Real Functions with Application to Hair Modelling ». *Computers and Graphics*, volume 20, numéro 1, pages 11–19, 1996.

[SS95] P. Schroeder et W. Sweldens. « Spherical Wavelets : Efficiently Representing Functions on the Sphere ». *Computer Graphics*, volume 29, numéro Annual Conference Series, pages 161–172, 1995.

[TS67] K. E. Torrance et E. M. Sparrow. « Theory for Off-Specular Reflection from Roughened Surfaces ». *Journal of Optical Society of America*, volume 57, numéro 9, 1967.

[VW95] R. C. Veltkamp et W. Wesselink. « Modeling 3D Curves of Minimal Energy ». *Computer Graphics Forum*, volume 14, numéro 3, pages C/97–C/110, septembre 1995.

[Wam] Bruce E. Wampler. « V – A C++ GUI Framework ». Disponible sur l'Internet à `http ://www.cs.unm.edu/ wampler/doc/vwebref/vwebref.html`.

[Wat70] G. S. Watkins. « A Real-Time Visible Surface Algorithm ». Rapport technique UTECH-CSc-70-101, University of Utah, Salt Lake City, Utah, 1970.

[Whi80] Turner Whitted. « An Improved Illumination Model for Shaded Display ». *CACM*, volume 23, numéro 6, pages 343–9, juin 1980. also in Tutorial : Computer Graphics : Image Synthesis, Computer Society Press, Washington, 1988. (Abstract) in Computer Graphics, no. 2 (SIGGRAPH '79 Proceedings), p. 14.

[Wil78] Lance Williams. « Casting Curved Shadows on Curved Surfaces ». *Computer Graphics*, volume 12, numéro 3, pages 270–274, août 1978.

[Wil83] Lance Williams. « Pyramidal Parametrics ». In *Computer Graphics (SIGGRAPH '83 Proceedings)*, volume 17, pages 1–11, juillet 1983.

[Wil94] N. Wilt. *Object-Oriented Ray Tracing in C++.* John Wiley and Sons, 1994.

[WS92] Yasuhiko Watanabe et Yasuhito Suenaga. « A trigonal prism-based method for hair image generation ». *IEEE Computer Graphics and Applications*, volume 12, numéro 1, pages 47–53, janvier 1992.

# Annexe A

# Implantation

*Nous décrivons l'implantation logicielle dans ce chapitre, en mettant l'emphase sur l'architecture extensible du système. Ce chapitre peut être utilisé comme référence de programmation pour le logiciel.*

Nous avons choisi d'utiliser un logiciel de lancer de rayons existant, afin de nous concentrer sur la problématique. Le logiciel OORT [Wil94], distribué sur l'Internet[1], est écrit en C++, et son architecture est suffisamment flexible pour permettre l'ajout de nouvelles primitives. Les classes des différentes primitives implantent l'interface `Object3D`, qui interagit avec les autres classes du lanceur de rayons comme l'illustre la figure A.1.

Le texel est lui-même une primitive, il implante lui aussi l'interface `Object3D`. L'objet `Texel` est donc le "pont" entre le lanceur de rayons et notre algorithme. A quelques exceptions près, il serait possible d'adapter notre système à un autre logiciel de lancer de rayons, uniquement en modifiant l'interface de `Texel`. Internement, `Texel` met en œuvre tout le mécanisme de rendu de notre algorithme. Le mécanisme de construction du volume de référence étant indépendant, nous décrirons plus bas son implantation.

## A.1 Rendu

Notre objectif est de créer un système de rendu général et flexible, de sorte que modifier les détails de l'implantation soit relativement aisé. Pour atteindre cet objectif, il est indispensable de trouver les classes de notre domaine, et de répartir notre algorithme sur ces classes afin que chacune ait une responsabilité précise qui corresponde à son identité.

D'après la discussion des chapitres précédents, nous pouvons identifier les classes suivantes dans notre domaine : le texel bien sûr, le volume de référence (qui est aussi un octree), le

[1] Disponible à `http ://wuarchive.wustl.edu/graphics/graphics/books/Object-Oriented-Ray-Tracing/`

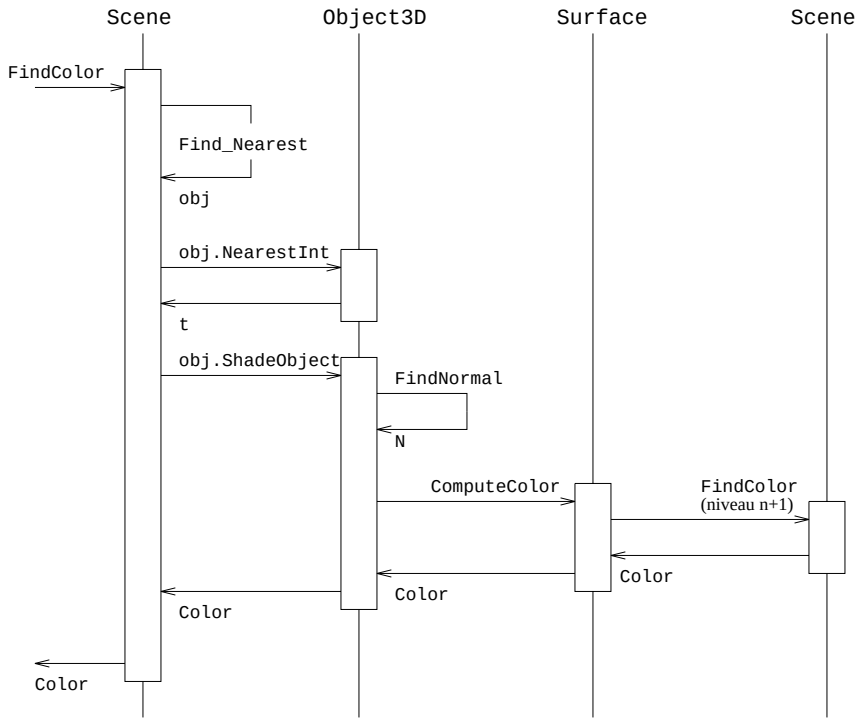


FIG. A.1: Interaction de `Object3D` avec le système `OORT`.

voxel (qui est aussi une cellule de l'octree), la déformation spatiale, la NDF, et le matériau. L'algorithme de rendu sera donc réparti sur toutes ces classes. Les figures A.3 et A.4 illustrent les relations entre ces classes. Le déroulement du rendu, illustré par la figure A.2 s'effectue de la manière suivante :

- OORT lance un rayon dans la scène. La méthode `Object3D::NearestInt` de chaque objet est appelée pour déterminer si ce rayon intersecte l'objet. Si cet objet est un `Texel`, il s'agit d'intersecter un rayon avec une boîte englobante (la boîte englobante du volume de référence) déformée par la déformation spatiale du texel. La méthode `ITransform::IntersectBoxRay` est appelée pour vérifier si une intersection existe. Cette méthode est la responsabilité de la déformation car pour le même rayon linéaire et la même boîte englobante, seule la déformation affecte la position et la forme de la boîte.
- Si une intersection est détectée, OORT appelle `Object3D::ShadeObject` pour calculer la couleur du pixel. `Texel::ShadeObject` implante l'algorithme `IlluminerTexel` de la section 4.1. Elle instancie un itérateur `RayOctreeIterator` qui se charge de traverser le volume. Cet itérateur implante l'algorithme `ProchainVoxel` de la section 4.3. A chaque voxel intersecte, la méthode `VolumeVoxel::Shade` est appelée pour calculer l'illumination et l'opacité à ce voxel. Ce calcul est orchestré par `VolumeVoxel`, mais en réalité un grand nombre de classes y participe.

- `VolumeVoxel::Shade` implante l'algorithme `IlluminerVoxel` de la section 4.4. Elle commence par calculer l'atténuation pour chaque source lumineuse, et conserve cette information dans une structure dynamique `LightInfo` qui sera utilisée plus bas. Ensuite, `VolumeVoxel::Shade` invoque `INDF::Shade` qui se charge d'intégrer le modèle d'illumination locale sur la NDF du voxel. Dans le cas de notre NDF ellipsoïdale, la méthode `EllipRefl::Shade` est appelée, elle implante l'algorithme `IntégrerNDF` de la section 4.5.
- `EllipRefl::Shade` échantillonne des normales sur la surface de l'ellipsoïde. Pour chaque normale visible à partir du point de vue, `ITexelSurface::LocalShading` est appelée pour calculer l'illumination locale pour cette normale. Le modèle d'illumination locale est implanté par le matériau lui-même, car les propriétés du matériau sont indissociables du modèle d'illumination locale choisi. Dans notre cas, le modèle de Phong [Pho75] est implanté par la classe `SimpleSurface` dérivée de `ITexelSurface`.

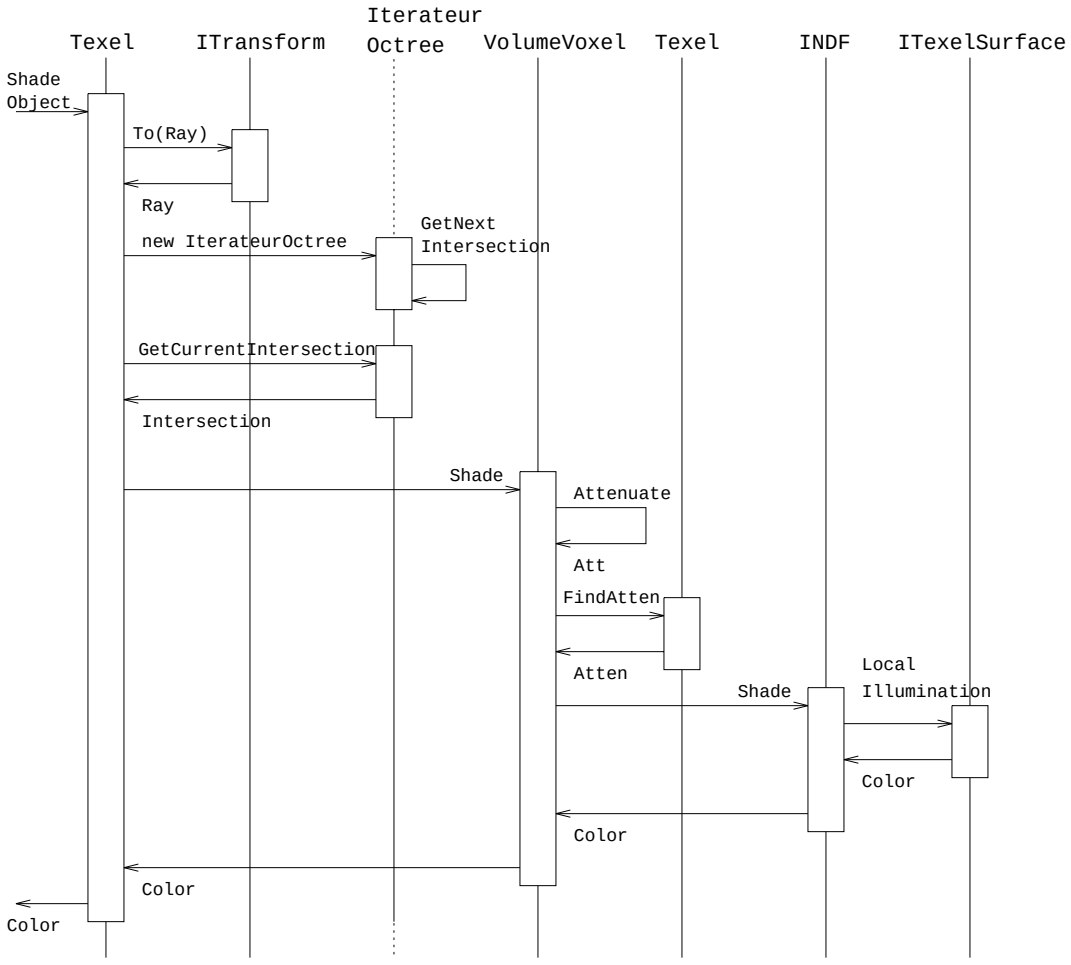


FIG. A.2: Intéraction entre les classes durant le rendu.

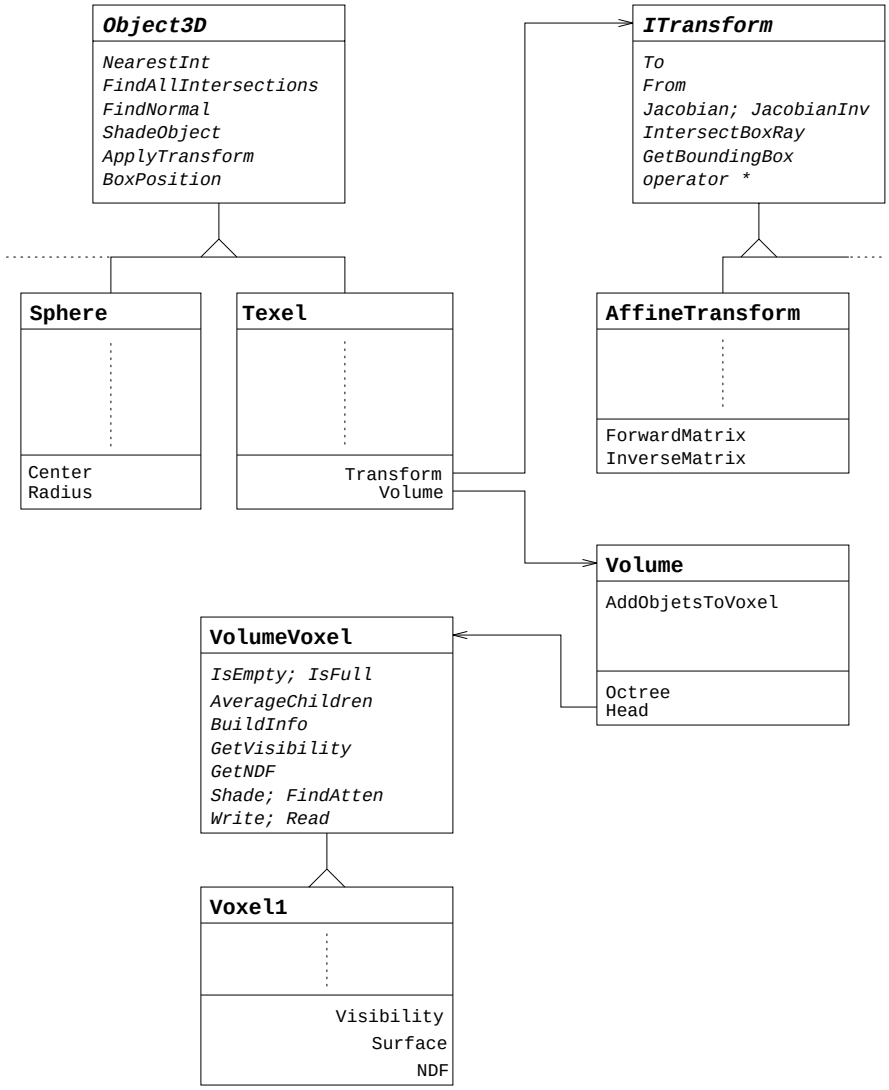


FIG. A.3: Hiérarchie des classes du système, A.

## A.2 Construction

La construction du volume de référence met en jeu les mêmes classes que le rendu, à l'exception de `Texel` qui ne joue aucun rôle ici. La figure A.5 illustre l'interaction entre les différentes classes durant la construction ; le déroulement s'effectue de la manière suivante :

– Lorsque le constructeur de `Volume` est appelé avec une liste de primitives, la construction du volume commence. La boîte englobante du volume est calculée comme l'union des boîtes englobantes des primitives. L'usine de voxels `IVoxelFactory::CreateVoxel` est appelée pour créer la racine de l'octree, puis la méthode récursive `Volume::AddObjectsToVoxels` est appelée pour remplir chaque voxel de l'octree.

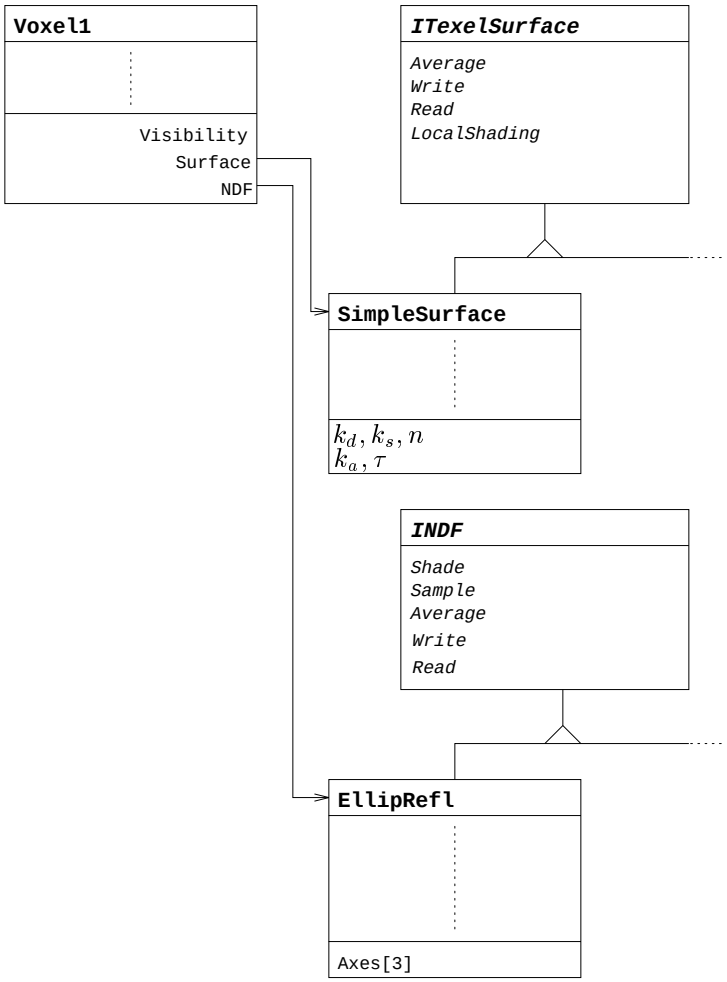


FIG. A.4: Hiérarchie des classes du système, B.

- `Volume::AddObjectsToVoxels` implante la fonction `AjouterObjetsAuxVoxels` de la section 3.1. Pour chaque primitive dans la liste, la méthode `Object3D::BoxPosition` est appelée pour décider si la boîte englobante du voxel est à l'intérieur, à l'extérieur ou sur la frontière de l'objet (voir section 3.2). Cette méthode n'existait pas dans l'interface originelle d'`Object3D`, il nous a fallu la rajouter.
- Si le voxel courant est un voxel-feuille, `Volume::AddObjectsToVoxels` appelle ensuite la méthode `VolumeVoxel::BuildInfo` qui construit les informations du voxel. Notre voxel (`Voxel1`) contient une NDF, un matériau et une visibilité : ce sont ces informations qui sont donc construites ici.
  - `Voxel1::BuildInfo` (voir section 3.3) commence par générer un certain nombre de rayons aléatoires sur les faces du voxel, puis ces rayons sont intersectés avec les primitives de la liste. Pour chaque intersection, la normale à la surface de l'objet intersecté est calculée par `Object3D::FindNormal`. Le nombre d'intersections servira

plus tard au calcul de la visibilité.

- `Voxel1::BuildInfo` appelle ensuite `INDF::SampleNDF` qui se charge de créer une NDF étant donnée une liste de normales. Notre NDF ellipsoïdale crée la NDF selon la méthode décrite à la section 3.3.1.
- `Voxel1::BuildInfo` appelle enfin `ITexelSurface::Average` pour créer le matériau moyen à partir du matériau des objets contenus dans le voxel, selon la discussion de la section 3.3.3.

– Si le voxel courant n'est pas de plus bas niveau, `Volume::AddObjectsToVoxels` entre en récursion, puis appelle `VolumeVoxel::AverageChildren` qui moyenne les informations de ses enfants. Cette méthode invoque `INDF::Average` pour la NDF des enfants et `ITexelSurface::Average` pour les matériaux des enfants.

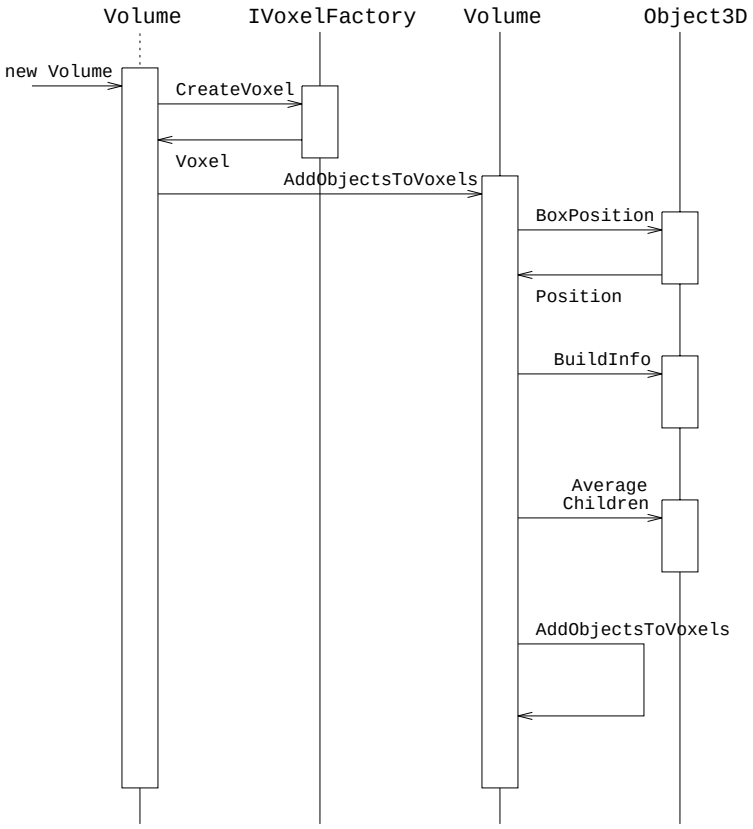


FIG. A.5: Intéraction entre les classes durant la construction.

## A.3 Détails divers

### A.3.1 Interface utilisateur

Nous utilisons un *application framework* nomme `V` [Wam], distribué sur l'Internet[2]. Ce *framework* peut être utilisé sur des plateformes UNIX, MS-Windows ou Macintosh. Nous avons écrit le code nécessaire pour intégrer le lanceur de rayons `OORT` à `V` : la classe `Scène` implante une horloge `vTimer` de `V` et lance une série de rayons à chaque coup d'horloge.

### A.3.2 Méthodes virtuelles

Nous utilisons le mécanisme d'heritage de `C++` pour réaliser le polymorphisme. Ce n'est pas le seul choix possible ; en fait nous l'avons choisi pour sa simplicité d'implantation, malgré la pénalité due à l'utilisation des fonctions virtuelles. Un autre choix, plus efficace, serait d'utiliser les *templates* au lieu des diverses interfaces. Prenons par exemple l'interface `INDF` : au lieu de dériver la classe `EllipRefl` de `INDF`, nous pourrions déclarer `EllipRefl` avec les mêmes méthodes que `INDF`, et modifier la classe `VolumeVoxel` de la manière suivante :

```
template <class TNDF> class VolumeVoxel
{
  ...
  TNDF m_NDF ;
} ;
```

Cette méthode permettrait d'instancier des voxels contenant `EllipRefl`, sans avoir à utiliser des méthodes virtuelles qui ralentissent l'exécution. En revanche, échanger dynamiquement le type de NDF utilisée s'avèrerait considérablement plus difficile.

[2]Disponible à `http ://www.cs.unm.edu/~wampler/vgui/vgui.html`.