THÈSE DE DOCTORAT DE l'UNIVERSITÉ
PARIS XI
UFR D'INFORMATIQUE
Spécialité: Infographie

présentée par:

Fabrice NEYRET

pour obtenir le titre de **DOCTEUR DE L'UNIVERSITÉ PARIS XI**

TEXTURES VOLUMIQUES pour la Synthèse d'Images

Soutenue le 17 Juin 1996, devant la commission composée de:

M. Claude PUECH	**Rapporteur**
M. Michiel VAN DE PANNE	**Rapporteur**
M. Alain CHESNAIS	**Examinateur**
M. Dominique GOUYOU-BEAUCHAMPS	**Examinateur**
M. Francis SCHMITT	**Examinateur**
Mlle Sabine COQUILLART	**Examinatrice**
M. André GAGALOWICZ	**Examinateur**

A tous les petits princes,
et à toutes les ondines...

Remerciements

Je remercie Messieurs *Claude Puech* et *Michiel Van de Panne* d'avoir accepté d'être rapporteurs de cette thèse, et Messieurs *Alain Chesnais*, *Dominique Gouyou-Beauchamps* et *Francis Schmitt*, pour m'avoir fait l'honneur de bien vouloir faire partie de mon jury.

Je remercie également M. *André Gagalowicz*, mon directeur de recherche, qui m'a accueilli dans son laboratoire à l'INRIA, ainsi que Mlle *Sabine Coquillart*, qui m'a suivi durant ma thèse.

Je tiens à remercier aussi tout le personnel de Syntim, chercheurs et non chercheurs, avec qui j'ai pu avoir des échanges nombreux, et pas uniquement en matière de synthèse d'images ; en particulier, Francis, Philippe et Jos qui ont eu la patience de cohabiter dans le même espace, Xavier pour les longs échanges de mails, Jean-Paul et Laurence incessamment sollicités pour les mille tracas quotidiens.

L'INRIA étant une grande famille, je pense aussi à Evelyne, dont j'ai partagé le bureau un temps et qui m'a fait le plaisir de participer à la relecture de ce manuscrit, à Jean-Baptiste pour nos réflexions sur la communication, et à tous les autres habitants du bâtiment 24, ainsi qu'aux membres de l'audiovisuel sans qui nos images de synthèse ne resteraient que chiffres sur un disque.

Table des matières

Table des figures

Préface

Motivations personnelles

Je vais parler ici des motivations personnelles qui m'ont conduit à choisir ce sujet. Si l'on veut positionner ces travaux au sein de la discipline, on trouvera plus loin une tentative de classification 'sociologique' des acteurs du domaine (en B.1), des valeurs techniques qui les guident, et des grands problèmes qui jalonnent la technique (en B.2).

En tant que synthésiste, mi-scientifique, mi-informaticien, et un peu graphiste, je m'attache à 'faire entrer dans l'ordinateur' ('modéliser' comme on dit dans notre milieu) les objets et phénomènes qui peuplent notre environnement. Ma principale source d'inspiration provient du 'réalisme' des scènes naturelles : un paysage, une forêt, un arbre sous le vent, un vol d'oiseaux, un torrent, le feu, l'océan, un nuage qui bourgeonne...

L'idée que l'on puisse ainsi modéliser l'Univers tient sans doute de la volonté de maîtrise de l'Homme, cet orgueilleux, dont la soif de compréhension est l'un des aspects positifs. Produire (et non reproduire) une image naturelle crédible sur ordinateur à partir d'une poignée de paramètres synthétiques et quelques procédures, c'est avoir compris suffisamment de la physique sous-jacente, c'est avoir saisi un peu de l'essence de la perception, c'est avoir recréé un peu de vie. Car la synthèse ne génère pas simplement des images : elle modélise un monde, juste suffisamment pour qu'on puisse en tirer des vues.

Si l'image est réussie, cela célébrera sans doute la Science des Hommes, au travers des nombreuses disciplines dont les apports progressifs se retrouvent sous la forme des briques de l'usine-à-mondes qu'est une plateforme de synthèse d'images. Mais comme les sciences ne décrivent encore pas tout, ou pas à la bonne échelle, ou pas comme on aimerait, la réussite sera aussi redevable à tout ce mécano informatico-empirique également inclus dans un tel logiciel, qui permet de produire des comportements physicoïdes agréables à partir de 'modèles' dont on trouvera avec peine un lien quelconque avec la physique[1]. Mais comme les

1. On donne en 1.1 un aperçu des critères qui guident les choix de modèles.

simulateurs et autres procédures ne simulent pas tout, ou pas comme on voudrait [2] que ce soit 'réaliste', la réussite rendra finalement hommage à l'art du graphiste, l'*utilisateur*, qui aura su en instillant les bonnes courbes, les bons paramètres, faire transcender toute cette mécanique pour donner le dernier coup de patte qui procure finalement à la scène sa vie.

Mais ceci est un Grâal, car de telles réalisations sont encore hors de portée de la synthèse. Beaucoup d'aspects devront être résolus avant d'y parvenir : simulation physique des phénomènes sous-jacents (l'eau, les effets du vent, l'illumination...), modèles comportementaux des êtres vivants (marche, interactions, effets de groupe...), traitement de descriptions géométriques colossales définies à plusieurs échelles...

Ce dernier point a retenu mon attention le temps de cette thèse pour plusieurs raisons. D'abord, la complexité naturelle est un aspect peu traité de la discipline, alors qu'un objet complexe est autre chose que beaucoup de fois ce qu'on sait faire pour un objet simple, ne serait-ce que pour des raisons de faisabilité technique (temps de modélisation, mémoire disponible, temps de calcul). Ensuite, on sent bien que le traitement des objets complexes dépasse le simple cadre des scènes naturelles : on touche là à un problème de représentation, voire d'unification entre les représentations, où la photométrie n'est peut-être plus si disjointe de la géométrie. C'est en fait l'article fondateur des textures volumiques qui m'a lancé : il contenait tous les germes qui ont inspirés mes travaux, même si Kajiya a peut-être simplement voulu établir un cadre commode lui permettant de produire de la fourrure, pour ne plus jamais y revenir par la suite.

Il se trouve que j'ai été ingénieur à TDI dans une autre vie, avant d'entreprendre cette thèse. J'en ai gardé le point de vue de l'utilisateur : on peut souhaiter qu'un modèle soit joli, crédible, ce qui suppose qu'il soit raisonnablement physique mais sans plus (je m'en justifierai un peu plus loin), mais si l'on souhaite qu'il conduise à des images intéressantes il faut aussi qu'il soit utilisable, c'est à dire contrôlable par le graphiste aux bonnes échelles, et efficace, c'est à dire que l'on puisse en temps fini produire des essais, une image, et même toute une animation.

Puisqu'on parle d'utilisateur, quelles sont les applications qui nécessitent ainsi une figuration informatique des objets naturels ? Il y a bien sûr le plaisir que procure à l'informaticien-chercheur un résultat conforme à ses espérances, mais cela ne saurait justifier le financement de ses travaux. Les débouchés solvables sont essentiellement constitués par le milieu audiovisuel, avec notamment l'industrie du cinéma qui a besoin pour ses effets spéciaux de construire des scènes totalement contrôlables, et par le milieu marketing au sens large, qui

2. Les graphistes ont une idée très précise de ce à quoi doit ressembler une ombre ou un reflet, qui diffère parfois sensiblement de ce qui résulte d'une simulation physique...

de la construction automobile à l'architecture a besoin de présenter un avant-goût d'une réalisation à l'étude. Dans une moindre mesure, l'industrie de la simulation (en y incluant celle des jeux vidéo) souhaite représenter des mondes crédibles, mais ceux-ci doivent être avant tout rapidement affichables donc à l'heure actuelle souvent dépouillés. Nous parlons en B.1.4 des débouchés de la synthèse d'images.

Maintenant, très vite je me justifie quant au 'degré de physique' à introduire dans les représentations quand l'objectif est simplement d'obtenir des images animées destinées à être vues par un spectateur.

Il y a danger à être trop 'physique' : si l'on veut construire le modèle physique d'un objet, comme du tissus, des cheveux, ou un revêtement pigmenté, est-on bien sûr d'avoir trouvé le phénomène prépondérant ? Si c'est le cas, la partie 'structurée' du phénomène explique-t-elle une proportion majoritaire de l'aspect ? A-t-on idée de la variation des paramètres du modèle le long de l'objet ? Les paramètres du modèles sont ils mesurables, et existe-t-il des valeurs numériques connues pour chacun d'eux ? L'échelle de la modélisation est elle compatible avec l'unité de visibilité de l'effet ? Le gain en réalisme par rapport au surcoût est il intéressant ? Le modèle est-il intégrable dans une plateforme logicielle moins 'physique' ? L'utilisateur est-t-il encore maître de l'apparence globale de l'objet généré ? Se contentera-t-il du 'réalisme' ? Est-ce que tout ce qui entoure le modèle, notamment l'éclairage, est également 'physique' ?

Mais il y a aussi danger à n'être pas assez 'physique' : quel est le sens concret des paramètres du modèle ? Peut-on les relier à une unité ? Comment garantir la continuité de transition entre modèles différents ? Comment rendre compte empiriquement des effets corrélés d'une foule de paramètres ? Que se passe-t-il dans les cas limites ?

Je suis donc méfiant vis à vis des modèles comme ceux utilisés pour les textiles, où l'on suppose bien des connaissances quant à la structure pour rendre compte finalement de quelques taches lumineuses. Mais je sais aussi qu'il est difficile d'animer un liquide sans tenir un peu compte de la mécanique des fluides. Le rendering est à la limite, car les grandeurs que l'on manipule sont parfois ambiguës, et certains phénomènes surestimés en compensent en pratique d'autres. Il faut donc pour chaque problème se poser la question de la bonne échelle à laquelle intégrer la physique, en gardant bien à l'esprit le but de la démarche de la synthèse, qui est d'obtenir de belles images qui 'fassent vrai'.

Introduction

Une des principales caractéristiques du monde naturel, vu en tant que scène à modéliser, est sa *complexité*, c'est à dire le fait qu'il soit chargé d'une multitude de détails. De ce point de vue, il y a bien des manières d'être 'complexe' : les détails d'un rocher relèvent d'une organisation évoquant des surfaces fractales, ceux d'une plante semblent répondre à la logique d'une grammaire, d'autres objets complexes comme le feuillage, l'herbe, la fourrure, présentent une similitude plus répétitive... Interviennent entre autres éléments la variété des 'motifs', le hasard de leur variabilité, ainsi que les échelles auxquelles ils s'appliquent.

En fait, un grand nombre d'autres scènes sont susceptibles de receler une telle complexité : fibres textiles, cartes topographiques, images médicales..., c'est pourquoi j'inscris résolument cet aspect parmi les formes du réalisme. Le réalisme est cependant une notion assez plurielle, de plus ce n'est pas le seul critère que l'on puisse se fixer lorsqu'on souhaite modéliser des scènes complexes ; nous évoquerons en section 1.1 différents critères, et en B.2.2 plusieurs acceptions du réalisme.

Si l'on souhaite modéliser une scène où intervient une forme de complexité en employant les outils interactifs génériques habituellement utilisés pour modeler les objets simples, l'utilisateur se trouvera successivement confronté à un certain nombre de problèmes pratiques :

- Tout d'abord au moment de la spécification des formes (phase que l'on nomme *modélisation* en synthèse d'images), il va devoir donner des informations de manière particulièrement répétitive alors qu'à l'évidence on souhaiterait plutôt indiquer une 'loi' générale, informations qui seront particulièrement détaillées alors que les détails des formes seront à peine visible,
- ce qui suppose en sus de disposer de toute cette connaissance fine.
- S'il parvient au terme de cette fastidieuse modélisation, l'utilisateur s'apercevra que la place occupée en mémoire est monstrueuse.
- Si malgré tout le fichier géométrique peut être chargé par le programme de rendu, celui-ci construira l'image à une lenteur désespérante.
- L'image enfin péniblement obtenue, le graphiste constatera avec horreur qu'elle présente un *aliasing*[3] cauchemardesque...

3. L'aliasing est une malédiction qui frappe toute la synthèse d'images, et qui vient de ce qu'on évalue un attribut d'une *surface*, en l'occurrence celle d'un pixel, à partir de la valeur en un seul *point* (cf Annexe C.3). Dans le cas présent, l'image présentera un grouillement de pixels clairs et sombres...

Même si l'on peut adapter la modélisation et optimiser le rendu, il y a à l'évidence un problème de représentation. La solution généralement adoptée en synthèse d'image consiste à réduire le problème en décomposant chaque objet en une partie purement géométrique, la 'forme', et une partie 'habillage' au moyen d'un modèle plus ou moins en trompe-l'œil[4]que l'on nomme *texture*. C'est à l'utilisateur de faire cette séparation dans la scène, selon l'importance qu'il accorde aux différentes composantes vis à vis de ses critères et contraintes. Plus rarement, plutôt que d'effectuer cette décomposition il arrive aussi que l'on utilise des modèles spéciaux décrivant l'ensemble de l'objet complexe à représenter, pour peu que celui-ci appartienne à une famille que l'on sait traiter. Pour tout ce qui n'est pas assumé en tant que géométrie classique, le traitement de la complexité est donc essentiellement assuré par toute une famille de modèles en trompe-l'œil, qui constituent une gamme progressive d''effets 3D'[5] allant de pair avec la progression du coût : le modèle le plus simple applique une image à la surface de l'objet, le plus complexe est capable de couvrir cette surface d'éléments en relief.

C'est à cette dernière catégorie de modèles que l'on va s'intéresser, parce que l'aspect trompe-l'œil finit par y prendre tous les caractères de la géométrie tout en gardant les avantages de la texture (que l'on énumérera en 1.4.1). A la limite elle finit par devenir une représentation concurrente de la géométrie classique qui répond aux situations les plus exigeantes de la complexité.

Objet des travaux

La méthode texturelle la plus aboutie en matière de mimétisme géométrique est celle des *textures volumiques*, introduite en 1989 par Kajiya et Kay dans le but de simuler la fourrure. L'implémentation proposée alors était relativement ad-hoc, peu souple et particulièrement coûteuse en temps de calcul et en mémoire. Mais dans la communauté, quelques personnes ont senti la pertinence de l'approche pour résoudre élégamment le problème de représentation de la complexité, au moins de sa forme répétitive. Curieusement personne ne s'est vraiment lancé, et les textures volumiques auraient pu retomber dans l'oubli. Je me suis employé durant cette thèse à essayer de déployer tout ce que la méthode avait de potentiel, en la rendant plus générale, plus complète et plus rapide. Ceci se traduit par une *nouvelle représentation* des textures volumiques, suffisamment complète pour modéliser les *géométries complexes répétitives*.

4. en ce sens que ces détails n'ont pas d'existence géométrique. Cf C.2.3 pour les différents attributs d'une texture.

5. On évoque à la section 2.1 ce que sont les divers caractères qui procurent l'impression de relief ou 'effet 3D'.

Organisation de l'ouvrage

Après cette introduction, le corps de l'ouvrage se compose de deux parties constituées chacunes de deux chapitres. La première pose le problème du traitement de la complexité, la deuxième expose les travaux qui fondent notre nouvelle représentation des textures volumiques. Le chapitre de conclusion qui suit met en perspective notre modèle dans l'évolution des textures volumiques, récapitule notre représentation, puis évoque les directions à développer.

Le premier chapitre présente les diverses solutions adoptées en synthèse pour traiter les scènes complexes. On y envisage en premier lieu les divers critères qui serviront à en choisir une en fonction du contexte applicatif. Puis après avoir apprécié les améliorations que l'on peut porter aux algorithmes de rendu, et quelques modèles dédiés, on passe en revue les méthodes de trompe-l'œil qui, au stade ultime (les textures volumiques), finissent par ressembler à des représentations géométriques.

Le second chapitre introduit les textures volumiques de façon orthogonale, en déterminant l'information à conserver pour obtenir une représentation ayant tous les effets du 3D. C'est l'occasion de montrer que les textures volumiques de Kajiya, que l'on détaillera en fin de chapitre, sont en bonne place pour répondre au cahier des charges. C'est également l'occasion de citer les travaux de Fournier sur le 'filtrage' géométrique, qui posent le bon cadre du problème. Cela permettra de déterminer ce qu'il faut ajouter à la représentation pour qu'elle puisse se prétendre complète.

Le troisième chapitre expose l'essentiel de la nouvelle représentation [Ney94, Ney95b] : le cœur de la méthode repose sur la modélisation, le filtrage et le rendu de la réflectance, qui est encodée dans chaque voxel du volume. Ces voxels forment l'échantillon de texture volumique, dont les instances répétées et déformées appelées *texels* recouvrent la surface de l'objet à habiller, formant ainsi une couche volumique. La fabrication et le rendu de ces diverses entités sont décrits dans ce chapitre. Après avoir concrétisé ces propos au moyen d'un algorithme et de quelques résultats, on discute des limites de la méthode, et d'un problème commun auquel on se trouve confronté, le traitement de l'incertitude.

Le quatrième chapitre traite des extensions de la méthode, qui visent à l'intégrer aux outils classiques donc à la rendre véritablement utilisable : on y étudie le mapping, la génération de volumes, et le rendu en espace courbe qu'entraîne la déformation des texels. Cela nécessite d'enrichir la physique de la méthode, et ajoute un niveau hiérarchique entre l'échantillon de texture et la couche volumique. Le rendu en espace courbe donne l'occasion de quelques réflexions [Ney95c] quant à la généralisation à d'autres modèles géométriques. On traite également ici de la façon de spécifier le contenu de l'échantillon de texture [Ney96a], de moduler les attributs de ce contenu, et d'animer [Ney95a] les texels.

L'ouvrage se termine par une série d'annexes :

- l'Annexe A énonce nos choix en matière de terminologie, notamment quant au recours aux termes anglais ;
- l'Annexe B propose une présentation sociologique du domaine de la synthèse d'images (origine des acteurs, délimitation du domaine, motivations, débouchés...), ainsi qu'une présentation technique (valeurs et critères, les grands problèmes...) ;
- l'Annexe C donne un aperçu du processus de genèse des images ;
- l'Annexe D rassemble tous les détails techniques volontairement omis du corps de l'ouvrage (algorithmes), ainsi que quelques remarques sur l'optimisation et la parallélisation ;
- l'Annexe E constitue le recueil exhaustif des fonctionnalités de notre implémentation, conclu par l'exemple du script qui a produit la spécification et l'animation de la prairie fleurie qui illustre le chapitre animation.

Contributions

Pour mesurer l'importance d'une contribution à une méthode, une première façon de voir les choses consiste à évaluer les **améliorations techniques apportées** à celle-ci dans son cadre d'origine.

- La nouvelle représentation code la réflectance dans tout le volume, elle permet donc d'y représenter toutes sortes d'objets, et même l'anisotropie. Elle quitte ainsi le domaine du ad-hoc pour devenir plus générale.
- La 'physique' est plus correcte en ce qui concerne l'illumination et l'opacité. On accède également à la couleur.
- L'utilisation d'un octree rend le codage du volume plus compact, et accélère le temps de traversée.
- La représentation est multiéchelle, et le modèle de réflectance s'y prête. Cela permet un calcul du rendu bien plus rapide, et en même temps de meilleure qualité (il y a peu d'aliasing).

On peut également apprécier les apports qui visent à **rendre la méthode plus utilisable**, mieux intégrée aux outils traditionnels.

- Puisque l'on sort des modèles ad-hoc, il faut pouvoir spécifier facilement le contenu de la texture volumique. Nous avons d'abord développé un langage de description géométrique, semblable à ceux utilisés pour les plateformes de synthèse privées de modeleur interactif, puis nous avons entrepris de décrire la conversion en texture volumique des diverses descriptions géométrique classiques. (On pourrait néanmoins imaginer quantité d'outils, interactifs ou non, dédiés à la spécification des échantillons de texture volumique...)
- Le contenu de la texture volumique peut lui-même être 'habillé', en modulant les attributs du matériau comme on le fait pour la surface d'un objet classique.
- La texture est maintenant applicable sur n'importe quel maillage, et se manie quant au *mapping* comme les textures ordinaires.
- L'animation à plusieurs échelles des textures volumiques a également été traitée.

Il arrive aussi que les améliorations qualitatives et quantitatives amènent à **élargir le champ de vision**, au point que la méthode enrichie devienne un outils complètement nouveau, suggérant des pistes pour de grands problèmes en suspens :

- Le faible coût de la méthode transforme un outil pour effets spéciaux prestigieux en modèle de texture épaisse utilisable systématiquement. En particulier, le calcul d'animations complexes n'est plus un luxe à peine imaginable.
- L'adaptativité de la méthode fait que le coût de calcul est désormais lié à ce que l'on voit, et non à ce que l'on connaît. Cette 'moralisation' du coût caractérise la pertinence de la représentation, que l'on retrouve dans la simplicité de l'animation muti-échelle. On sort ainsi de la simple texture, pour accéder à une véritable représentation des objets complexes répétitifs.
- Cela pourrait amener à envisager la problématique des niveaux de détails sous un nouveau jour. Certains travaux récents sur les textures volumiques vont d'ailleurs dans ce sens.
- Il y a peut-être également des recettes à généraliser en ce qui concerne les emprunts aux techniques voisines : les déformations d'objets sans explicitation du résultat sont intéressantes pour bien des représentations délicates à manipuler, de même le rendu volumique pourrait tirer avantage de l'enrichissement du contenu du voxel.

Chapitre 1

La complexité en synthèse d'images

La complexité d'une scène met souvent à mal les systèmes de modélisation et de rendu traditionnels (décrits Annexe B.2.1), parce qu'elle contraint ceux-ci à faire face à de très grandes quantités de primitives, usage auquel ils sont mal adaptés. Il est toutefois possible d'améliorer les performances de ces systèmes, ce qui permet de conserver ces outils traditionnels (donc dotés de nombreuses fonctionnalités), c'est ce que nous verrons en section 1.2.

Certains types de données, souvent liés à une application particulière comme la visualisation de données topographiques ou la visualisation médicale, peuvent cependant faire l'objet de modèles dédiés plus adéquats, comme c'est le cas pour les modèles de terrain et les images volumiques de tomographie, que nous aborderons en section 1.3. La différence de nature de ces données par rapport aux objets habituels de la synthèse et l'absence de notion de scène dans leur application initiale[1] justifient ces modèles dédiés, mais nous verrons que l'on trouve à les réemployer dans le cadre de la synthèse 'classique', c'est à dire s'intéressant à des scènes, éventuellement animées.

La synthèse classique est d'ailleurs riche en modèles dédiés, qui permettent justement de rendre compte d'une certaine complexité par des techniques apparentées au trompe-l'œil (textures), comme nous le verrons en section 1.4. Ces modèles permettent de graduer la fidélité de la représentation, à choisir en fonction de la taille à laquelle apparaissent les détails à représenter.

Le fait de pouvoir graduer la représentation en fonction de l'échelle est d'ailleurs une problématique à part entière connue sous le nom de *niveaux de détails*, que nous aborderons en section 1.5. Cette adaptabilité est cruciale tant pour la performance des calculs que pour la qualité de l'image, et en particulier pour les scènes complexes. Souvent mentionnée, cette problématique commence seulement à être envisagée concrètement pour les applications qui

1. Il s'agit souvent de visualiser un ensemble de données propres à un domaine, données mises en forme via une représentation facilitant la visualisation. Il n'y a donc pas à gérer une scène composée de plusieurs objets, nécessitant une homogénéisation des représentations.

butent sur le temps de traitement (réalité virtuelle, radiosité).

Les textures volumiques font le pont entre tous ces aspects, ce qui en fait un modèle particulièrement adapté aux scènes complexes répétitives. C'est ce que nous montrerons en section 1.6.

1.1 Critères de choix

Les critères qui dictent le choix d'une représentation sont multiples ; ils dépendent des contraintes de l'application et des buts que l'on se fixe.

Le premier critère auquel on pense en synthèse d'image est le *réalisme*. Ce terme générique peut concerner le **réalisme de la géométrie**, le **réalisme du rendu**, ou le **réalisme de l'animation** (on détaille d'autres aspects en Annexe B.2.2). Le réalisme se conçoit bien entendu à des degrés divers, selon qu'on s'intéresse à la crédibilité de l'image ou à l'exacte simulation des phénomènes physiques sous-jacents.

Le second critère auquel on pense est l'*esthétique*, qui dépend du graphiste et donc repose sur la commodité et la puissance des outils mis à sa disposition. En fait, tout modèle qui n'est pas de pure simulation (comme par exemple celle des vagues à la surface de l'océan) nécessite de passer par une phase interactive afin de spécifier un certain nombre de paramètres. Les véritables critères qui entrent en jeu sont alors la **facilité de modélisation** et la **facilité d'animation**, qui sont également liés au degré de **contrôlabilité** du modèle.

Viennent ensuite des critères 'techniques', qui peuvent être en pratique assez décisifs : il s'agit du *coût*, c'est à dire de la **performance** et de la **consommation mémoire**, qu'il faut associer à la **qualité d'image** (absence d'aliasing). Un autre critère est l'**intégrabilité** du modèle à une scène existante. La réalité virtuelle accorde par exemple une importance vitale au temps réel ; la qualité d'image n'est recherchée que dans la mesure ou le premier critère n'est pas remis en cause.

Mais on ne peut pas totalement séparer le choix des modèles géométriques de celui des algorithmes de rendu qui doivent en produire une image (présentés en Annexe C). Les critères choisis peuvent en effet conditionner le type d'algorithme de rendu (d'autant plus que les performances de ces méthodes n'ont pas la même sensibilité à la quantité de primitives), lesquels contraignent le choix des modèles géométriques.

Contraintes de l'algorithme de rendu sur les modèles géométriques

Ainsi, les *rendus projectifs* peuvent être rapides et peu aliasés, mais ne savent gérer que les représentations en facettes, et ne traitent pas le transport de la lumière (reflets, ombres) sauf dans quelques cas simples. La conversion de surfaces gauches en facettes et les astuces pour simuler les ombres peuvent alors considérablement dégrader les performances.

De même la *radiosité* ne connaît aujourd'hui pratiquement que les facettes comme représentation géométrique. Par contre la simulation de l'illumination est très élaborée, ce qui se paye par un temps de rendu important. La jeunesse et l'aspect 'physique' de la méthode limitent considérablement la boîte à outils disponible d'astuces et d'effets sur lesquels l'utilisateur peut s'appuyer pour enrichir la scène.

Le *lancer de rayons* (ray-tracing) a souvent la préférence quand le temps de calcul n'est pas trop contraint ; sa performance est intermédiaire pour une simulation du transport de la lumière intermédiaire. Son principal avantage réside dans sa souplesse : il permet de prendre en compte presque toutes les représentations de données à partir du moment où l'on sait en déduire un point d'intersection et une normale. Il est donc facile d'intégrer un nouveau modèle dans ce cadre. De plus on bénéficie d'une large palette d'outils qui ont étés développés pour ce contexte. La qualité d'image requiert cependant une multiplication du coût, de même si l'on veut approcher le photoréalisme de la radiosité. De plus la méthode est très sensible au nombre d'objets présents dans la scène.

Il faut noter que les grands logiciels offrent maintenant des méthodes mixtes, de manière à utiliser l'algorithme le plus adapté à un effet donné sur un objet donné : après une première passe en projectif, des rayons sont lancés si nécessaires, et certains objets peuvent éventuellement être 'radiants'. Mais ceci vise à intégrer les divers effets photométriques plutôt que que les divers modèles géométriques : dans ce contexte mixte, il faut pouvoir interpréter un même modèle depuis les divers algorithmes de rendu, ce qui pousse à utiliser le modèle le plus simple, la représentation en facettes [2].

Nos critères

L'application type à laquelle je me réfère principalement est la production de séquences d'images pour l'audiovisuel, ce qui fixe quelques-uns de ces critères :

- la qualité d'image est importante même si le degré de réalisme est laissé à l'appréciation du graphiste,
- les modèles doivent être un tant soit peu contrôlables par l'utilisateur, qui doit disposer d'outils de spécification puissants (interactifs ou langages),
- les modèles doivent autant que possible être intégrés en une seule scène,
- toute la mémoire d'une station de travail est utilisable à partir du moment où cela n'altère pas le temps de calcul,
- le temps de calcul doit rester raisonnable, c'est à dire ne pas dépasser environ 20 minutes par image (pour la plupart des productions). A noter que ce dernier point est une contrainte forte pour le lancer de rayons, qui pousse au compromis qualité-réalisme/coût.

2. A noter que le *voxel* (élément de volume) est également un type de donnée traitable par les trois principaux algorithmes de rendu. Il est cependant généralement relégué à la seule représentation des objets vaporeux ou volumiques transparents.

On s'efforcera cependant de conserver ici un point de vue un peu plus large que celui de l'application audiovisuelle.

1.2 Modèles conventionnels

Il s'agit ici de permettre la représentation de scènes complexes en persistant à utiliser les modèles géométriques conventionnels (facettes, patches ou autre). Si l'on met de côté le problème de la construction de ces objets, qui peut par exemple être effectuée de manière procédurale, reste à gérer le comportement de la scène vis à vis du calcul du rendu.

Il se trouve en effet que les différents algorithmes de rendu ne dégénèrent pas de la même façon avec le nombre de primitives, de plus ils n'offrent pas les même possibilités d'adaptation à la complexité de la scène :

• le rendu projectif est une méthode orientée 'objet', qui exploite bien la cohérence entre pixels voisins lors de la fabrication de l'image, y compris pour le traitement de l'aliasing. Mais cette cohérence se dissout quand les primitives sont si nombreuses que leur taille devient inférieure au pixel. De plus, le grand nombre d'occultations entre primitives accroît le coût des algorithmes à base de tri comme ceux à base de Z-buffer. Il n'y a pas grand chose à faire pour améliorer la situation, à part peut-être essayer de traiter des primitives de plus haut niveau donc occupant plus de pixels (cf 1.2.2), ce qui n'est pas facile en projectif. Un avantage cependant de l'orientation 'objet' est que les primitives sont utilisées successivement, et n'ont donc pas besoin d'être conservées simultanément en mémoire.

• le lancer de rayons est une méthode orientée 'image', qui exploite très peu la cohérence entre pixels voisins. De ce fait, le coût par pixel croit linéairement avec le nombre de primitives (dans la méthode générique). Il y a cependant beaucoup à gagner à introduire un peu de cohérence, ce que nous étudions en 1.2.1. De plus, la méthode s'accommode très bien des primitives de haut niveau, exposées en 1.2.2.

• la première passe 'échange d'énergie' de la radiosité devient également coûteuse avec l'accroissement du nombre d'objets. Ce sont des méthodes apparentées aux niveaux de détails qui sont utilisées pour 'factoriser' les échanges entre groupes de surfaces, nous les évoquons en 1.5.

1.2.1 Structurer l'espace

Dès qu'un parcours a lieu au sein d'une base de données géométrique, comme c'est le cas pour le lancer de rayons, on gagne beaucoup à structurer l'espace, ce qui permet de distinguer et d'ordonner ce qui est susceptible de se trouver sur le parcours. Cela se fait par exemple en définissant une *grille 3D* (ou *grid*), et en associant les objets aux voxels qu'ils occupent ; ces voxels peuvent eux-même être regroupés en octree. Le parcours s'effectue alors de voxel en voxel comme pour un tracé de droite en 2D, la discrimination des objets sur le parcours se limitant ainsi à ceux présents dans le voxel courant.

D'un autre côté, on gagne également à pouvoir trancher rapidement quant à la présence d'un objet donné sur un parcours. Cela peut se faire à l'aide de *boîtes englobantes* (ou de forme plus élaborées : ellipsoïde, enveloppe convexe...), pour lesquelles le traitement est moins cher. De manière hiérarchique, on peut ensuite construire les encadrements de groupes d'objets, et ainsi de suite.

Le panachage entre méthode de grilles et méthode de boîtes est bien sûr possible : une grille à faible résolution peut contenir des boîtes dans ses voxels, ou bien une boîte englobante peut contenir une grille de façon à 'trier' la géométrie qu'elle contient.

Quand il ne s'agit que d'ordonner les facettes, ce qui est utile en ray-tracing comme en projectif, il existe d'autres structures comme les BSP-trees [FvDFH90].

1.2.2 Enrichissement des primitives

Une primitive géométrique de haut niveau peut approximer une surface donnée sur une plus grande région qu'une facette, et ainsi limiter la quantité de primitives dans la scène. Le ray-tracing est par exemple capable de traiter directement des descriptions CSG d'objets, composées d'opérations booléennes sur des formes géométriques simples [GN71], ainsi que des surfaces paramétriques ou des isosurfaces. Il est hélas beaucoup plus difficile d'enrichir la famille des primitives reconnaissables par le rendu projectif ou la radiosité, bien que certains travaux aient étés réalisés sur les patches, les quadriques et les surfaces développables.

En matière de scène complexe, on économise de la mémoire en considérant les occurrences d'objets similaires comme des *instances* partageant une même représentation, à une transformation rigide près.

1.3 Modèles dédiés

Certaines applications présentent à la fois des données complexes particulières et des contraintes d'efficacité, qui ont conduit au développement de modèles et d'algorithmes de rendu dédiés. C'est par exemple le cas pour le rendu de terrain, pour lequel les algorithmes d'inverse displacement mapping permettent de visualiser efficacement des cartes topographiques $z(x, y)$, et pour le rendu volumique utilisé notamment pour la visualisation de données tomographiques en imagerie médicale.

Dans ces deux cas l'information géométrique est organisée de façon à totalement structurer l'espace (valeurs au sommets d'une grille), et doit être traitée par un rendu ad-hoc. Un tel modèle peut alors éventuellement être intégré dans le contexte de rendu classique dont dérive l'algorithme de visualisation (i.e. projectif ou lancer de rayons), en tant que nouvelle primitive.

1.3.1 Cartes de terrain

Les cartes d'élévations (displacement mapping [Coo84, MKM89]) ne sont guère qu'un moyen de décrire une géométrie, que l'on peut construire explicitement en facettes si nécessaire. Les méthodes d'inverse displacement [PHL91, Tai92] permettent de manipuler cette représentation sans qu'il soit nécessaire de l'expliciter en facettes, en gérant directement l'intersection entre un rayon et la représentation. Cela permet en outre de le faire efficacement, en procédant de manière hiérarchique. Le principe consiste à organiser les données en quadtree et à précalculer dans chaque case l'altitude maximale. On a alors la garantie que le rayon n'intersecte pas le terrain si ses points d'entrée et de sortie dans la case sont plus hauts que ce maximum, sinon on repose le problème sur les 4 cases filles, et ainsi de suite. Quand on atteint la résolution des données, il faut alors tester l'intersection du rayon et de la surface en la calculant, ce qui n'arrive que dans peu de cases.

On retrouve ces modèles en synthèse classique (dans le cadre du lancer de rayons) pour coder des variations de surface (cf 1.4.2), comme par exemple des bas-reliefs.

1.3.2 Volumes

Les volumes sont codés en voxels, et éventuellement compressés en octrees [Sam90b, Sam90a]. On a d'abord représenté des isosurfaces de la densité, et des volumes translucides pour lesquels la contribution locale d'un voxel se limite à une couleur et une opacité. Lors de la traversée du volume par un rayon, la transparence se multiplie et la couleur s'additionne (après pondération par la transparence cumulée). Puis sont arrivés les modèles plus réalistes pour lesquels on évalue l'illumination locale, en déduisant une normale du gradient de densité. Si l'on souhaite tenir compte des ombres, le coût de calcul augmente considérablement.

Des méthodes projectives fonctionnent aujourd'hui presque en temps réel, en 'factorisant' les distorsions subies par chaque tranche de volume [LL94], au prix de légères approximations. Il existe également des méthodes spectrales, se ramenant à la FFT inverse d'une tranche du spectre du volume [Lev92].

La synthèse classique s'intéresse aux volumes, traités par lancer de rayons, pour la modélisation de fumée, brouillard, etc [AW87]. Ce modèle a également été adapté à la radiosité [RT87].

1.4 Modèles en trompe-l'œil

Dès les premières heures de la synthèse, la pauvreté des modèles géométriques a été compensée par l'habillage des surfaces à l'aide de textures de couleur, multipliant la quantité de détails présents dans l'image afin de rendre compte d'une certaine complexité (d'où le qualificatif de *trompe-l'œil*, puisque ces détails n'ont pas d'existence géométrique). Des méthodes d'habillage plus complexes sont progressivement apparues (textures de transparence, de normales) dans le même esprit de produire un maximum d'illusion de géométrie avec un minimum de coût de calcul et de modélisation (cf 1.4.1), au point que les modèles les plus récents finissent par ressembler à des représentations alternatives de la géométrie (cf 1.4.2).

Ces représentations sont efficaces parce qu'elles sont adaptées à l'échelle où on les utilise. De plus, il est facile de moyenner sur un voisinage des données organisées en carte, plus efficacement que par échantillonnage.

Une approche en trompe-l'œil très différente permet de modéliser certains types de scènes complexes, ce sont les systèmes de particules dont on parle en 1.4.3.

Remarque : on désignera dorénavant par 'géométrie' ou 'représentation géométrique' la description spatiale de la surface des objets (par exemple à l'aide de facettes), par opposition aux trompe-l'œil ou aux représentations volumiques.

1.4.1 Les textures planes

Dans la répartition des rôles en usage en synthèse d'images, la *matière* contrôle l'apparence photométrique des objets, ce qui recouvre tous les paramètres physiques qui régissent l'interaction de la lumière avec la surface : couleur, transparence, modèle de réflectance, état de surface... L'état de surface, notamment, est modélisable par le *bump mapping* [Bli78], qui génère un relief factice en perturbant la directions des normales locales, lesquelles interviennent dans le calcul de la lumière réfléchie.

Une *texture* est un modèle qui permet de spécifier les variations de cette apparence le long de la surface, ce qui revient à spécifier la variation des paramètres intervenant dans le calcul de l'illumination locale (cf C.2.3 et B.2.3). On peut décrire ces variations par le biais d'une simple carte, dupliquée et appliquée selon diverses modalités sur la surface (*mapping*), mais il existe également de nombreux modèles qui construisent directement les paramètres à la surface (textures 3D, modèles stochastiques [GdM85], réaction-diffusion [WK91, Tur91], grammaires et procédures [EMP+94, Miy90, FPdB90]...). Les textures 3D [Per85, Lew89], par exemple, modélisent un matériau fictif (bois, marbre...) en tout point de l'espace à l'aide d'une fonction de bruit, laquelle est évalué à la surface de l'objet[3].

3. Attention à ne pas confondre texture 3D et texture volumique, en dépit de l'apparente interchangeabilité du qualificatif ! Voir aussi [FvDFH90] pour tous ces modèles.

Il faut noter que le problème du mapping, qui s'apparente à celui du tapissier, est loin d'être résolu (bien que curieusement peu de travaux semblent s'intéresser à la question [MYV93, BVI91]), et c'est à l'utilisateur de se débrouiller pour qu'il n'y ait pas trop de distorsions. Sur un autre plan, il commence à apparaître quelques outils pour construire interactivement la texture à la surface [HH90], mais bien d'autres restent à inventer...

Toutes ces méthodes permettent de générer une certaine complexité d'apparence à la surface des objets[4], voire de simuler à des degrés divers un aspect 3D dont la modélisation géométrique serait considérablement plus lourde, au prix de certaines approximations que l'on juge peu importante étant donné l'échelle ou la dynamique des détails.

Avantages des représentations en texture

- Les textures sont une forme de modèle génératif, en ce sens que quelques paramètres servent à construire beaucoup de données. Pour une texture procédurale, les paramètres modulent un motif de base ; pour une texture plaquée, les paramètres sont composés d'un échantillon et de la fonction de mapping. La spécification est donc simplifiée en regard des données brutes générées. Le contrôle de l'aspect par l'utilisateur est relativement global, alors que la modélisation géométrique explicite nécessiterait a priori de spécifier chaque détail.
- Pour la même raison, on peut également gagner en mémoire.
- Les textures permettent de simuler des formes (éventuellement en relief) sans qu'il soit nécessaire de les coder explicitement ; Ceci est dû au fait que quelques indices peuvent suffire à suggérer le 3D (la variation de normale, dans le cas du bump mapping), c'est l'aspect trompe-l'œil. La surface géométrique sous-jacente n'étant pas modifiée par la texture, les calculs d'intersection ou de projection ne sont pas affectés par la complexité prise en charge par la texture, ce qui se traduit par un gain important en performance par rapport à une scène où tout serait codé par la géométrie.

4. D'autres types de textures, dont on n'a pas parlé ici, permettent de simuler le transport de la lumière, de manière à obtenir des ombres (shadow maps [RSC87]) ou des reflets (reflection maps [MH84]) sans lancer de rayons.

- Les textures ont un aspect 'signal bidimensionnel' : disposer de 'champs de valeurs' plutôt que de géométrie permet de moyenner ces valeurs si nécessaire (quand ça a un sens physique, c'est à dire quand l'illumination est une fonction linéaire de ce paramètre). On peut ainsi éviter l'aliasing dû à la petite taille des motifs de la texture sans alourdir le coût du rendu, dans la mesure où l'on peut remplacer le suréchantillonnage par une intégration sur le pixel (cf Annexe C). Cette intégration peut même être formelle ou précalculée (mip-map [Wil83], SAT [Cro84]), ce qui accroît considérablement l'efficacité en comparaison du suréchantillonnage auquel on doit procéder pour le rendu anti-aliasé d'une géométrie.

Les textures ont donc des propriétés coupant court aux inconvénients que l'on rencontre lorsque l'on utilise la représentation géométrique pour encoder une géométrie complexe, situation inconfortable que nous décrivions en introduction.

1.4.2 Vers des représentations géométriques alternatives

Les modèles de texture évoqués précédemment restent cependant trop simples pour simuler un relief important. L'absence de véritable relief se traduit par exemple au niveau des limbes[5] de l'objet, qui restent lisses, et par l'absence d'ombres portées [Tai93]. Il faut donc des modèles plus riches si l'on veut éviter de se ramener à une représentation géométrique, dont on a vu que la complexité de modélisation et le coût de calcul rendent préférables les modèles en trompe-l'œil tant qu'ils restent adaptés à l'échelle des phénomènes dont ils rendent compte.

Le displacement mapping, introduit pour la modélisation des cartes de terrain comme on l'a vu plus haut, répond à ce besoin. Mais si l'on veut générer de véritables petits objets à la surface (fourrure, fibres textiles), un codage des variations de reliefs[6] en $z(x,y)$ est insuffisant, auquel cas on fait appel aux hypertextures [PH89] et aux textures volumiques [KK89]. Ces deux modèles décrivent le matériau dans une couche volumique au voisinage de la surface. Cela est fait formellement pour le premier, à la manière des textures 3D, et au moyen de données volumiques explicites pour le second. Les modèles existants de texture volumique, qui nous concernent au premier chef, seront exposés en 2.4.

Bien entendu l'enrichissement en relief de ces modèles de texture se fait au détriment du coût. Mais pour les détails fins d'une scène complexe (feuillage, herbe, poils...), cela reste très avantageux par rapport à la géométrie, d'autant plus que l'on conserve les avantages

5. Lieux de la surface tangents au regard.

6. Il existe des modèles procéduraux qui fabriquent de la géométrie [FLCB95, PJM94], mais on peut difficilement les classer avec les modèles en trompe-l'œil. Certains autres sont plus ambigus, notamment parce qu'ils adaptent leur apparence au point de vue [WP95].

des textures comme l'échelle du contrôle et la capacité de filtrage. Ces modèles procurent finalement des représentations concurrentes de la géométrie, dans la mesure où elles simulent en définitive la plupart des aspects 3D et s'avèrent plus performantes, du moins pour une certaine famille d'objets comme les scènes complexes répétitives.

1.4.3 Systèmes de particules

Les systèmes de particules [Ree83, RB85], constituent un approche en trompe-l'œil très différente : ce modèle est adapté aux géométries complexes dont les détails sont fins et peuvent être générés par des trajectoires, comme l'herbe, l'eau vive, le feu, etc. Les trajectoires sont produites par des procédures alimentées par quelques paramètres interactifs, et se prêtent donc bien à l'animation par simulation de la dynamique. Outre l'étroitesse de son champ d'application, le modèle est également limité par la spécificité de son algorithme du rendu qui dessine les particules comme des traits de crayon, ce qui diffère totalement de la modélisation classique de la matière et rend assez difficile l'intégration à une scène existante.

1.5 Problématique des niveaux de détails

L'une des entraves générée par les scènes complexes vient de ce que les objets doivent être décrits finement dès lors qu'ils peuvent être vus de près, ce qui engendre surcoût et aliasing chacune des nombreuses fois où ces objets sont vus de loin, puisque chaque pixel doit intégrer toute l'information inutilement détaillée qui s'y projette. Les approches à niveaux de détails s'emploient à adapter la représentation aux diverses échelles où on l'utilise [WP95].

Une première méthode consiste à entretenir plusieurs descriptions géométriques plus ou moins précises de chaque objet, afin d'utiliser la représentation la plus adaptée à l'échelle (une variante plus élaborée se charge d'adapter dynamiquement le maillage). Un premier problème réside dans la construction automatique des représentations allégées, notamment par des méthodes de décimation de polygones [Tur92]. Un second problème concerne la continuité du rendu lorsqu'on s'éloigne ou qu'on se rapproche d'un objet.

Une autre approche est la commutation de modèles [BJ91, BM93, Kaj85], qui revient à passer d'une représentation géométrique à une représentation texturelle, puis photométrique au fur et à mesure que l'on s'éloigne. A nouveau, la continuité de la transition entre deux modèles est un point délicat.

On peut remarquer que les algorithmes de radiosité commencent également à inclure cette problématique, notamment en entretenant une représentation volumique hiérarchique de l'espace [HSD94] destinée à rendre compte aux diverses échelles de l'échange d'énergie entre groupes d'objets.

1.6 Notre représentation

Les textures volumiques telles que nous les exposerons dans ce manuscrit visent à pourvoir à la plupart des critères mentionnés au début de ce chapitre. On ne s'étonnera donc pas que cette représentation ait des points communs avec chacune des méthodes citées, comme on le verra au long des chapitres suivants :

- on modélise un *volume*, au voisinage d'une surface ;
- les fonctions de réflectance qui y sont stockées imitent la géométrie par *trompe-l'œil*;
- ce volume est dupliqué formellement comme le sont les *instances*;
- la couverture continue d'une surface par un motif caractérise une *texture*;
- que ce soit au niveau du mapping ou du volume, le rayon évolue dans un *espace structuré*;
- la construction multiéchelle du volume nous raccorde à la problématique des *niveaux de détails*.

Chapitre 2

Simuler la géométrie

Dans le chapitre précédent, on a vu comment les modèles successifs de texture - qui sont des trompe-l'œil en ceci que les détails représentés n'ont pas d'existence géométrique - rendaient compte de la complexité d'une scène. Le modèle le plus abouti en la matière est celui des textures volumiques, lequel présente de nombreux aspects d'un véritable modèle 3D, tout en gardant les avantages des textures.

Nous allons ici nous poser le problème dans l'autre sens, en partant d'une scène géométrique complexe dont on désire simplifier la représentation sans perdre d'information visible : quels sont les critères que l'on peut se fixer pour obtenir une représentation efficace de la complexité ? Qu'est-il nécessaire de conserver comme information pour rendre compte de l'impression de 3D d'un objet, et plus particulièrement d'un objet complexe ? Nous verrons que cela nous conduit à un modèle proche des textures volumiques, avec un certain nombre d'attributs supplémentaires. Cela nous permettra alors de définir notre représentation, suffisamment à même de rendre compte de la complexité pour remplacer la géométrie, mais encore suffisamment texturelle pour conserver la commodité et l'efficacité des textures.

2.1 Posons le problème

On cherche une bonne représentation de la géométrie complexe répétitive. Cette dernière limitation, la répétitivité, approxime une classe importante et typée de géométries complexes, pour laquelle l'aspect 'instance' des textures est pertinent, c'est-à-dire que la donnée d'un échantillon et d'une loi de répétition suffit à décrire tout l'objet. A cette limitation près, on souhaite obtenir un modèle aussi général que possible, c'est-à-dire capable de représenter toute forme de micro-géométrie, en procurant les mêmes impressions visuelles que les autres modèles 3D disponibles, mais en évitant autant que possible les inconvénients de ces derniers, notamment en matière d'aliasing et de coût de rendu. Les maîtres mots sont donc **modèle général**, **effets 3D**, et **rendu adapté**.

La représentation géométrique est une manière d'encoder avec une certaine précision tout ce que l'on sait de la surface d'un objet (on se passe de l'intérieur, qui n'est généralement pas visible). Ayant ainsi codé tout ce que l'on connaît de l'objet, si l'on ajoute les informations relatives au matériau, il est naturel que l'on puisse en tirer des vues sous divers angles (i.e. calculer un rendu), qui donneront à l'utilisateur l'impression de relief correspondant à l'objet.

Mais toute l'information codée est-elle utile ? Si le seul but de la représentation est de produire des images, alors pour des objets complexes dont les détails sont à peine visible il serait préférable de conserver ce que l'on voit, plutôt que ce que l'on connaît. Les grandes lignes d'une représentation adaptée sont donc : stockage uniquement de l'**information utile**, utilisation des données à la **bonne échelle**, conservation des **effets 3D** au mieux, et comme on l'a fixé plus haut, approche **texturelle**.

Qu'est ce qui est utile pour caractériser visuellement la forme d'un objet, et notamment l'impression de 3D ?

- Sa forme propre c'est à dire son contour se détachant du fond, sa position dans le champ visuel, la parallaxe lors d'un changement de point de vue ;
- les effets de blocage (*blocking effects*), c'est à dire le fait qu'une portion d'un objet peut être occultée par un objet plus proche, soit pour l'observateur (il est caché), soit pour la source lumineuse (il est dans l'ombre) ;
- l'illumination locale, qui donne une information sur l'orientation locale de la surface et ses variations.

Pour un objet de grande dimension, la forme est un attribut très important dans la mesure où il s'étend sur de nombreux pixels. La description de cette forme définit l'orientation locale de la surface, et par conséquent induit l'illumination locale. De ce point de vue, la représentation géométrique se justifie.

Mais pour les petits objets dont la taille est de l'ordre du pixel voire moins, la forme n'est pratiquement pas visible. L'illumination locale reste par contre très importante : dans un brouillard de cristaux de glace ou à la surface de la neige, les reflets sur les facettes ont un effet particulièrement visible alors que ces facettes ne le sont pratiquement pas ! A cette échelle, la photométrie est donc plus importante que la géométrie, et nécessite donc un échantillonnage plus fin. Il serait absurde de raffiner la géométrie dans le seul but de décrire la photométrie : à ce stade, il faut donc séparer le codage de la géométrie de celui de la photométrie. Le premier représente alors la forme à grande échelle, tandis que le second correspond à l'échelle microscopique.

Ebauche de notre modèle

La géométrie complexe occupe l'espace de manière a priori assez arbitraire et peu structurée. Une modélisation en voxels est donc toute indiquée : dans chaque élément d'espace (suffisamment fin pour que la taille de sa projection à l'écran ne dépasse pas celle d'un pixel), on va stocker un élément d'information géométrique signalant l'occupation de cette région, et un élément d'information photométrique encodant la façon dont ce lieu reflète la lumière. L'information de réflectance correspond dans le cas le plus général à une BRDF [1] 4D (voire plus si l'on s'intéresse au comportement en fréquence et en polarisation). Cette information devant a priori être stockée dans chaque voxel, il convient de la coder de la façon la plus compacte possible.

Comme on vient de le suggérer dans le paragraphe précédent, le 'grain d'information' que l'on va utiliser est associé à un volume correspondant à peu près à un pixel de surface projetée sur l'écran : plus fin, on conserverait trop d'information, qu'il faudrait intégrer pour reconstituer la valeur du pixel ; plus gros, on perdrait en résolution donc en information, et on engendrerait un crénelage (la forme des voxels apparaîtrait). Dans la mesure où la scène a vocation à être observée sous plusieurs points de vue donc à distance variable, le modèle doit être défini intrinsèquement. Il doit donc être multiéchelle, c'est-à-dire être capable de fournir des informations adaptées à tous les points de vue possibles, au moins dans un certain intervalle de distance. On obtient alors une représentation optimisant la qualité d'image par rapport au coût de calcul, puisqu'un seul rayon par pixel suffit à 'interroger' le grain d'information (i.e. le voxel) tout en évitant l'aliasing. En se fixant la distance minimale à laquelle on s'approchera d'un objet donné, on obtient la quantité minimale d'informations à fournir pour spécifier l'apparence de l'objet dans le cas le plus exigeant. Les données à échelle plus grossière pourront être construites par pré-intégration (grosso-modo) de la fonction d'occupation et de la fonction de réflectance.

En résumé, la représentation adaptée doit donc être volumique et multiéchelle, le 'grain d'information' correspondant à un voxel et contenant une information sur l'occupation spatiale et une information sur la réflectance. Comme on l'a mentionné au début de cette section, ce volume représente un échantillon de matériau 3D, destiné à être répété et déformé afin de constituer l'objet. On constate que tout en obtenant un modèle capable de représenter la géométrie, on a conservé les avantages des textures : on s'était déjà fixé l'aspect génératif, le codage minimum pousse le trompe-l'œil à l'extrême, et la représentation permet de pré-intégrer les informations.

1. La *Bidirectional Reflectance Distribution Function*, ou fonction de réflectance bidirectionnelle, indique quelle proportion de l'énergie provenant d'une direction donnée est réfléchie dans n'importe quelle autre direction donnée (ce qui donne 4 degrés de liberté).

Restent à définir :

- la nature et le codage de l'information sur l'occupation et sur la réflectance ;
- le moyen de calculer l'apparence de ce 'grain d'information' et plus généralement de produire un rendu de cette représentation ;
- le codage du multiéchelle et la façon de pré-intégrer les données,
- les modalités permettant de passer d'un échantillon à tout l'objet.

On traitera ultérieurement de la manière de spécifier le contenu de l'échantillon.

2.2 Eléments de réponses

La littérature fournit quelques éléments de réponse au cahier des charge que nous venons d'établir.

• Modéliser volumiquement un échantillon de géométrie tout en incluant un modèle photométrique local, ça existe, ce sont les textures volumiques de Kajiya et Kay [KK89], que nous détaillerons au paragraphe 2.4. L'idée consiste à étendre les textures en leur donnant du volume : on spécifie et on stocke un seul échantillon de 'matière géométrique', que l'on répète en le déformant à la surface d'une forme simple, constituant ainsi une 'peau volumique' au voisinage de la surface. L'échantillon volumique est constitué de voxels, lesquels encodent l'information géométrique sous la forme d'une probabilité d'occultation, et l'information photométrique sous la forme d'un modèle de réflectance analytique muni d'un repère. Cette représentation s'applique à de nombreux objets complexes répétitifs, comme la fourrure, l'herbe, le feuillage, etc. Cependant la méthode de Kajiya, qui s'intéresse plus particulièrement à la fourrure, utilise une primitive photométrique unique et identique en tous points du volume, encodée 'en dur' dans le programme et dédiée à la fourrure (cylindres parallèles), elle est donc peu générique. De plus la représentation n'est pas adaptative (i.e. pas multiéchelle), ce qui rend le calcul du rendu particulièrement onéreux. Il faut ainsi une douzaine d'heures pour obtenir la fameuse image du *Teddy Bear*. Enfin il n'y a pas véritablement de mapping, puisque chaque copie du volume repose exactement sur un carreau de la surface (la discrétisation de celle-ci est donc contrainte). Bien qu'elle contienne tous les germes d'une méthode générale puissante, la méthode de Kajiya et Kay telle qu'elle est présentée en 1989 se limite donc à un nouveau modèle exotique pour faire de la fourrure (comme on l'a dit plus haut, les présents travaux visent précisément à exprimer tout ce que la méthode originale avait de potentiel).

• Modéliser la micro-géométrie d'une surface par son comportement photométrique et filtrer celui-ci pour obtenir une représentation multiéchelle, Fournier l'a fait [Fou92] [2]. En maintenant à la fois une BRDF et une carte d'horizon [3], les effets de blocage sont correctement représentés et filtrés. De plus le rendu est efficace, dans la mesure où la représentation est multiéchelle et où la BRDF est représentée par une somme de pics de Phong, primitive largement utilisée en synthèse. Mais le modèle se limite aux surfaces, et la quantité de paramètres est inadaptée à un stockage volumique. En outre l'approximation de la BRDF se fait par minimisation d'erreur, ce qui rend les précalculs relativement lourds.

• Une représentation multiéchelle de la texture, on en utilise couramment, notamment les *mip-map* de Williams [Wil83]. On conserve l'image à plusieurs sous-résolutions, et au rendu on détermine la résolution idéale de la texture telle qu'un pixel de texture se projette en un pixel à l'écran, puis on interpole entre les valeurs du pixel données par les deux échelles disponibles juste au dessus et au dessous de cette résolution idéale. Ce modèle n'est pas parfait, principalement parce qu'il filtre la texture de manière isotrope alors que celle-ci a de bonnes chances d'être comprimée différemment selon les axes, notamment en vue rasante. Mais elle est rapide et simple à implémenter, ce qui fait qu'on la trouve jusque sur les cartes graphiques. Une méthode concurrente, les SAT (summed-area tables) de Crow [Cro84], stocke les primitives de la texture en tout point. La couleur correspondant à un pixel est obtenue par intégration de la texture sur le rectangle englobant la projection inverse du pixel, ce qui revient à combiner les primitives de la texture aux quatre coins.

2.3 Le modèle que nous allons développer

Les éléments de réponse énoncés en 2.2 relatifs au choix d'une représentation de la complexité tracent les grandes lignes que nous allons suivre dans les chapitres suivants, afin de réaliser une représentation efficace des géométries complexes répétitives prenant en compte tous les effets 3D : partant des **textures volumiques** originales de Kajiya et Kay, nous allons rendre la représentation **multiéchelle** à l'aide d'un octree, dans l'esprit des **mip-maps**, et nous allons généraliser la méthode en encodant un modèle de **réflectance** générique [4] en

2. Bien que ces travaux fondamentaux ne semblent pas avoir rencontré l'intérêt qu'ils méritent ; rares sont les papiers qui abordent la simplification de scène autrement que sous l'aspect géométrique.

3. Une carte d'horizon code l'angle solide de l'espace dégagé au dessus de chaque point d'une surface. Cela permet notamment d'évaluer la lumière ambiante, qui doit être plus faible au fond des vallées que sur les crêtes. On peut éventuellement remplacer cet angle solide par un échantillonnage d'azimuts dans les diverses directions.

4. A noter qu'il est de toute façon indispensable dans le cadre du multiéchelle de disposer d'une primitive relativement générique, dans la mesure où le moyennage de plusieurs primitives doit s'exprimer à l'aide d'une primitive de la même famille (i.e. la famille de primitives doit posséder une structure de groupe pour l'addition). Ça n'est assurément pas le cas pour une primitive comme le cylindre.

chaque voxel. Comme la mémoire occupée par un volume est importante, il faut réduire au minimum l'encombrement de chaque voxel, c'est pourquoi nous allons représenter la fonction de réflectance par une distribution de normales, elle-même approximée par une famille de distributions suffisamment riche mais caractérisable par peu de paramètres. Outre un pointeur vers ses voxels fils, chaque voxel de l'octree contiendra donc une probabilité d'occultation (qui *ressemble* à une densité volumique) et une *NDF* (distribution de normales) encodée par six paramètres.

Ceci caractérise l'approximation assumée par le modèle : au delà de la discrétisation de la NDF en six degrés de liberté, il faut noter qu'en représentant une BRDF par une NDF, on perd à petite échelle les effets de blocage (sur une surface non convexe, une même normale peut correspondre à deux sites disjoints qui se masquent l'un l'autre ; on perd donc de l'information en ne conservant que la normale). Cette information consommerait en effet beaucoup de place, alors qu'elle n'intervient qu'en dessous de la taille du pixel. Par conséquent, le filtrage sur un voisinage de plusieurs éléments de micro-géométrie encodés par une NDF sera purement géométrique, sans que les masquages réciproques interviennent.

Ayant désigné les textures volumiques comme modèle de départ à partir duquel nous allons construire notre représentation, il nous faut maintenant détailler cette approche initiale.

2.4 La représentation initiale des textures volumiques

Les texels de Kajiya

Kajiya et Kay ont introduit les texels en 1989 [KK89], principalement dans le but de proposer un modèle de fourrure. Bien que ce papier comprenne déjà les principales idées de base, il faut modifier le programme pour simuler d'autres matériaux, et le rendu est particulièrement lent dans la mesure où il n'y a pas de structure multiéchelle. Quoi qu'il en soit, l'aspect du Teddy Bear obtenu au bout de 12 heures de calcul est tout simplement merveilleux.

Le principe consiste à fabriquer un échantillon de texture volumique, en l'occurrence un échantillon de fourrure, lequel est stocké en un seul exemplaire dans un *volume de référence*, dont les copies déformées appelées *texels* seront mappées sur une surface sous-jacente, constituant ainsi une peau épaisse continue à la surface d'un objet.
L'échantillon est encodé par une structure volumique pleine, dans laquelle sont tracés des cylindres verticaux distribués aléatoirement, représentant les poils de la fourrure. La surface sous-jacente est composée de patches bilinéaires [5] (voir figure 2.1), chaque texel étant posé

5. Un patch bilinéaire est une surface développable portée par un quadrangle gauche.

exactement sur un patch et déformé de telle sorte qu'il colle aux texels voisins afin de constituer une couche continue. Les arêtes verticales (communes avec les texels adjacents) peuvent être orientées par l'utilisateur, ce qui revient à 'coiffer'[6] la texture de fourrure.

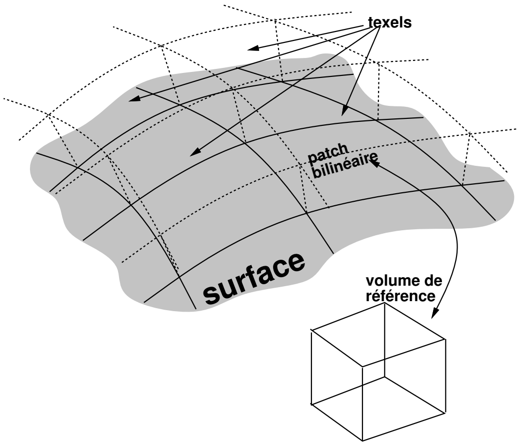


FIG. 2.1 - *Texels sur une surface*

La nature de la structure de donnée et l'algorithme de rendu ressemblent beaucoup à du rendu volumique, sauf que le contenu des voxels n'est pas vraiment une densité mais plutôt une probabilité d'occultation isotrope[7]. De plus, une autre donnée cruciale est stockée dans les voxels : le comportement photométrique local, qui consiste en un repère local et une fonction de réflectance analytique, qui permettent au texel de refléter la lumière comme s'il contenait vraiment un morceau de surface. L'implémentation proposée pour le rendu de fourrure modélise la réflectance d'un poil par celle d'un cylindre, et le repère est limité à la donnée de l'axe de celui-ci. Ces paramètres ne sont en fait pas stockés dans le volume dans la mesure où ils sont invariants dans l'espace de la texture : le volume ne contient que de la fourrure, et tous les cylindres sont verticaux (les variations de la 'coiffure' se traitent au niveau du mapping par déformation des texels), ce qui permet de factoriser ces données. Le volume ne contient ainsi qu'une donnée scalaire, comme pour le rendu volumique, à ceci près qu'il s'agit d'une probabilité d'occultation et non d'une densité[8]. La fonction

6. Un papier [SK92] présente en 1992 un système permettant de le faire.

7. Bien que lors de la construction du volume, les auteurs déterminent cette donnée à partir de l'occupation des voxels.

8. Une densité ne pourrait pas représenter une facette sans épaisseur, laquelle arrête pourtant la lumière. Du point de vue du rendu, c'est bien la quantité de lumière interceptée qui importe, donc la probabilité d'occultation. Kajiya et Kay identifient le problème mais se contentent d'introduire des fonctions de Dirac dans la densité. Nous préférons utiliser directement l'occultation, ce qui évite le problème, sachant que de toutes façons celle-ci peut s'exprimer à partir de la densité ρ ($A = 1 - e^{-\tau\rho . dl}$ pour un voxel traversé

de réflectance correspond à une approximation[9] de l'intégration du modèle d'illumination de Phong sur un cylindre (rappelons qu'elle a pour but de simuler les formes à petite échelle).

La texture volumique de Kajiya est ainsi définie par :

- un volume de référence, dont les voxels contiennent une pseudo-densité (plus la fonction de réflectance si celle-ci n'était pas factorisée),
- la donnée d'un modèle de réflectance valable pour tous les voxels,
- une surface sous-jacente composée de patches bilinéaires,
- un vecteur d'épaisseur (ou *hauteur*) donné en chaque sommet de la surface pour contrôler le 'coiffage' des texels.

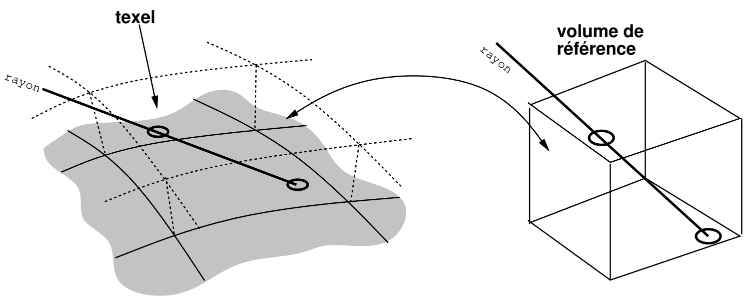


FIG. 2.2 - *Rendu de la texture*

Au rendu, un rayon intersecte la surface sous-jacente et la surface parallèle correspondant au plafond des texels. On se ramène alors dans l'espace de la texture où les texels s'identifient au volume de référence, en considérant comme linéaire la trajectoire du rayon dans cet espace (cf figure 2.2). Comme pour du rendu volumique usuel, le volume est traversé d'avant en arrière, la transparence se multipliant et la luminosité - pondérée par la transparence - se cumulant:

$$I = \sum_{t=near}^{far} (I_{loc}(t). \prod_{t'=near}^{t} e^{-\tau\rho(t').dL}) \quad \rightarrow \quad I = I + I_{loc}(t).T(t) \ , \quad T = T * e^{-\tau\rho(t).dL}$$

Un échantillonage stochastique du rayon est réalisé afin d'alléger les calculs. L'illumination locale est effectuée à l'aide du modèle de réflectance qui simule la présence d'un cylindre, et un rayon est envoyé vers la source de lumière afin de tester l'ombrage du voxel courant.

sur une longueur dl), mais avec l'inconvénient (outre les Diracs) pour Kajiya de devoir définir la constante physique τ liée à l'opacité propre du matériau. Par abus de langage il nous arrivera de parler de densité au lieu de probabilité d'occultation ; remarquons toutefois que pour une faible opacité, $A \sim \tau\rho.dl$. A noter que dans le modèle de Kajiya l'occultation est un scalaire, tandis que dans celui de Shinya [Shi92] et dans le notre [Ney94, Ney95b] c'est une fonction anisotrope (respectivement une valeur dans les trois directions canoniques, et une fonction définie à partir d'un ellipsoïde).

9. L'approximation est poussée : le terme diffus ne dépend pas de l'angle entre l'œil et la lumière !

Les autres représentations des texels

Shinya a proposé quelques améliorations en 1992[10] [Shi92], en stockant l'occultation dans les trois directions canoniques afin d'obtenir une occultation anisotrope (la surface apparente d'un objet varie avec l'angle de vue), et en traversant le volume par des pas successifs dans les trois directions canoniques[11].

Il liste les problèmes non résolus par Kajiya (construction du texel à partir d'une représentation polygonale, préfiltrage des voxels), recommande une approche hiérarchique, et suggère l'usage de données de corrélation entre les contenus des voxels (ce qui ressemble à un vœu pieux, car comment obtenir et tenir compte concrètement de cette information ?).

Restent deux problèmes importants en suspens, la construction du volume et le mapping des texels. Noma [Nom95] a proposé en 1995 une solution au problème de construction des texels à partir d'une représentation particulière, les distributions de petites facettes (obtenues par le logiciel de modélisation de plantes AMAP [dREF+88]).

Dans chaque voxel d'un octree, à tous les niveaux de celui-ci, il échantillonne dans quelques directions (en fait les trois directions canoniques) l'occultation engendrée par les facettes présentes dans le volume. Cela correspond à une projection, qui peut être réalisée par le hardware graphique. La réflectance est obtenue de la manière suivante : pour une direction donnée de la lumière, le terme diffus moyen du matériau inclus dans un voxel peut être obtenu en faisant la moyenne des termes diffus des facettes pondérées par leur surface apparente (laquelle s'obtient également via le hardware). L'auteur échantillonne ce terme pour des directions de la lumière correspondant à la normale moyenne des faces et pour deux directions faisant un angle de $\pi/4$ et $\pi/2$ avec cette normale[12]. Au rendu, la réflectance est obtenue en interpolant entre ces trois valeurs.

Comme nous, Noma utilise donc un modèle multiéchelle dans l'esprit du mip-mapping, encode une opacité anisotrope, et tire sa réflectance d'une représentation existante. Mais il ne tient pas compte du spéculaire, exploite des hypothèses fortes quant à la répartition des données, et ne destine pas ses texels au mapping mais à l'encodage d'objets entiers (en l'occurrence des arbres produits par AMAP).

---

10. Mais nous n'avons pris connaissance de ce papier qu'en cours de thèse, après avoir déjà fait des choix similaires.

11. Il est difficile de comprendre la motivation de ce procédé, dont le seul avantage semble être de ne pas avoir à interpoler les données comme il faudrait le faire pour obtenir l'opacité dans une direction quelconque. On se serait d'ailleurs alors aperçu qu'on perd beaucoup d'information, dans la mesure où l'occultation minimale et maximale n'apparaissent a priori pas dans les directions canoniques (penser a une galette à 45 degrés), lesquelles tendent alors à lisser ces valeurs. Comme on le verra plus tard, notre modèle revient également à coder l'occultation dans 3 directions, mais celles-ci sont données dans un repère propre à chaque voxel. Cela supposera par contre de stocker 6 valeurs plutôt que 3.

12. Bien que ça ne soit pas dit, cela suppose que le matériau soit suffisamment homogène pour qu'il présente le même aspect quelle que soit la direction orthogonale à la normale moyenne que l'on choisisse. Sinon il faudrait d'échantillonner dans tout l'angle solide, bien que cela augmente le coût en stockage et en calcul.

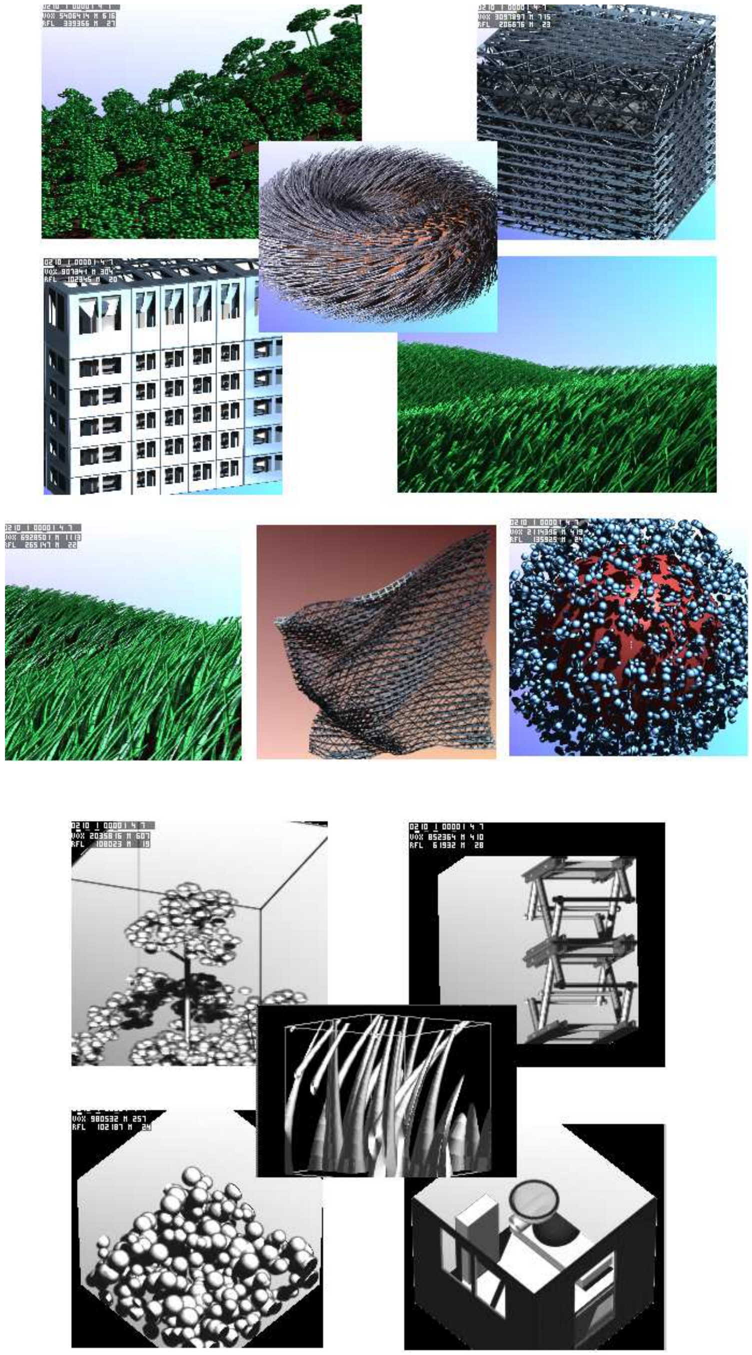


FIG. 2.3 - *Voici le type de scènes complexes que l'on désire représenter (*en haut*) ; les images sont générées avec notre modèle de texture volumique.* en bas *: échantillons de texture (ou 'texels') correspondants.*

Chapitre 3

Un modèle général et multiéchelle de textures volumiques

Conformément au 'cahier des charges' que nous nous sommes fixé au chapitre précédent, les principales caractéristiques dont on veut doter notre représentation sont :

- codage volumique de l'échantillon de texture, au moyen d'un octree (décrit en section 3.2),
- représentation multi-échelle dans l'esprit du mip-map, ce qui suppose de savoir filtrer les données (traitée en section 3.1.2),
- informations géométrique et photométrique séparées, encodées respectivement par une probabilité d'occultation et une approximation de la distribution de normales (définie en section 3.1.1),

Il nous faut également détailler comment se fait le rendu, au niveau local du voxel (cf section 3.1.3) et au niveau du texel (cf section 3.2.2).

Ce chapitre traite du noyau de notre représentation, lequel sera étendu au chapitre suivant. Ainsi, le mapping reste pour l'instant celui utilisé par Kajiya. Outre le volume de référence dont la nature diffère, nous nous donnons donc comme lui :

- une surface faite de patches bilinéaires sur lesquels s'appliquent les texels,
- des vecteurs hauteur en chaque sommet, pour orienter la peau volumique.

Nous suivrons une progression dans la description de l'échelle la plus locale à l'échelle la plus globale. Nous allons donc commencer par décrire en section 3.1 le modèle local, au niveau du voxel, dans la mesure où c'est sur lui que repose le reste de la construction. La réflectance doit simuler la micro-géométrie à l'échelle inférieure au voxel, il nous faut donc expliquer comment la représenter, la filtrer et en calculer le rendu. Nous verrons ensuite en section 3.2 comment traiter le modèle à l'échelle du texel.

3.1 La réflectance

3.1.1 Modéliser la réflectance

La réflectance traduit la façon dont un matériau interagit localement avec la lumière, dans le cas le plus général elle est anisotrope. L'anisotropie se caractérise par la fonction de réflectance bidirectionnelle locale (BRDF), qui indique quelle proportion d'énergie provenant d'une direction donnée se réfléchit dans une autre direction donnée. Plusieurs travaux concernant la modélisation de l'anisotropie [Kaj85, CMS87, HTSG91], ont déjà conduit à diverses représentations de la BRDF : en la tabulant, au moyen de fonctions harmoniques sphériques [SAWG91], à partir de champs de normales (issues d'une bump map) [CMS87], en modélisant des rangées de micro-cylindres [PF90], avec une distribution de pics de Phong [Fou92], à partir d'une nuée de sphères [Bli82]...

Le problème consiste à encoder avec un minimum de paramètres une famille aussi large que possible de BRDFs, et le stockage en volume de ces paramètres dans notre cas rend la contrainte de concision encore plus aiguë. Il se trouve que notre situation ne correspond pas au cas le plus général, pour lequel la BRDF est issue des propriétés optiques de la matière : l'anisotropie que nous cherchons à encoder provient de l'interaction de la micro-géométrie avec la lumière (rayures, granulosité, état de surface), plusieurs des travaux cités ci-dessus font d'ailleurs la même hypothèse. En somme, la fonction de réflectance anisotrope que l'on construit a pour but de 'résumer' la micro-géométrie trop petite pour être visible. Dans un tel cas, on considère souvent que la BRDF est issue de la distribution de normales locale (NDF), ce qui ramène la représentation de 4 à 2 dimensions[1] (la réflectance est alors obtenue en intégrant un modèle simple, typiquement celui de Phong, sur la distribution de normales).

1. Il est important de saisir la différence de nature de ces entités : à une portion de surface on peut associer une distribution de normales sur la sphère de Gauss, les représentations étant équivalentes (à une symétrie près) si la surface est concave ou convexe. Une NDF est donc de nature purement géométrique. La BRDF exprime le transfert d'énergie entre deux directions données, caractérisant l'interaction de la lumière avec un matériau. Elle dépend donc notamment de la physique régissant celui-ci, sachant qu'il existe des matériaux au comportement optique très exotique. Dans les cas usuels, l'expression de l'illumination locale propre au matériau est simple et ne fait intervenir la géométrie qu'à travers la normale (ex: modèles de Gouraud ou de Phong). S'il n'y a pas de masquage d'une partie de la surface par une autre, on obtient alors la BRDF en intégrant ce modèle sur la distribution de normales. Sinon, soit la surface est localement concave ou convexe et il suffit de limiter le champ de l'intégration (toutes les normales correspondant à la surface visible sont dans un même angle solide sur la sphère de Gauss), soit ce n'est pas le cas, ce qui veut dire qu'une même normale peut se référer à deux sites différents (par exemple une crête et une vallée), et donc la NDF ne contient pas suffisamment d'information pour en déduire la BRDF.

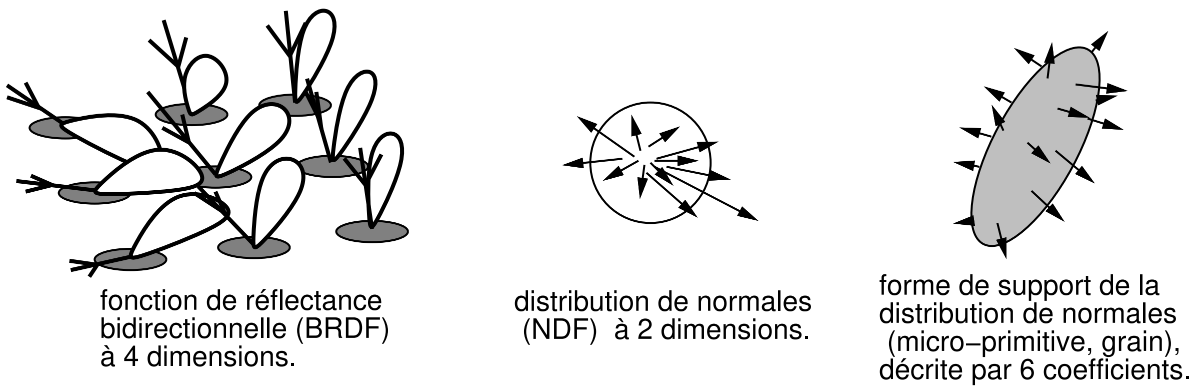


FIG. 3.1 - *BRDF et ses restrictions successives.*

On peut encore diminuer la dimension de la représentation en considérant une famille de formes agissant comme un 'grain' ou une 'cristallisation', qui servira de support à la distribution de normales. La dimension se réduit alors aux quelques paramètres qui caractérisent une telle forme, dont il faut choisir une famille aussi riche que possible.

Même si nous nous étions satisfait d'un modèle de réflectance restreint, nous aurions été obligés d'assouplir le modèle. En effet, par rapport au modèle de Kajiya qui utilise un 'grain' très spécifique, le cylindre, le cadre multiéchelle ajoute une contrainte qui pousse également à la généricité : avec le filtrage, il faut pouvoir recoder avec la représentation choisie le résultat du moyennage de plusieurs instances de cette représentation (i.e. la famille de formes encodant la réflectance doit posséder au moins approximativement une structure de groupe pour l'addition). Avec le modèle de cylindres de Kajiya, par exemple, il est impossible de représenter la moyenne de plusieurs cylindres (sauf bien sûr s'ils sont parallèles). A contrario, la décomposition en pics de Phong de Fournier [Fou92] obtenue par minimisation aux moindres carrés est précise, tout comme l'est la décomposition dans une base de fonctions harmoniques, mais ces représentations demandent beaucoup de paramètres (au moins une vingtaine) qu'il faut stocker localement.

Notre représentation

Nous avons choisi l'ellipsoïde comme primitive de compromis pour encoder la réflectance[2] : l'ellipsoïde permet en effet de simuler des formes communes comme la sphère, le cylindre (en prenant un rayon infini), l'élément de plan (en prenant deux rayons infinis), ainsi que les formes intermédiaires, avec seulement 6 paramètres. Nous verrons à la section suivante que l'on peut raisonnablement approximer la somme d'ellipsoïdes par un ellipsoïde. A noter que cette forme ne représente pas directement une géométrie, mais sert de support

2. L'ellipsoïde ne donne pas la forme de la distribution de normales, mais représente la surface de support de ces normales.

aux normales, elle n'a donc ni dimension ni position, juste une 'forme' (un galbe). Une géométrie non convexe peut donc être représentée par un ellipsoïde si l'on peut en trouver un ayant une distribution de normales proche (cf figure 3.2). La taille de l'ellipsoïde n'a pas

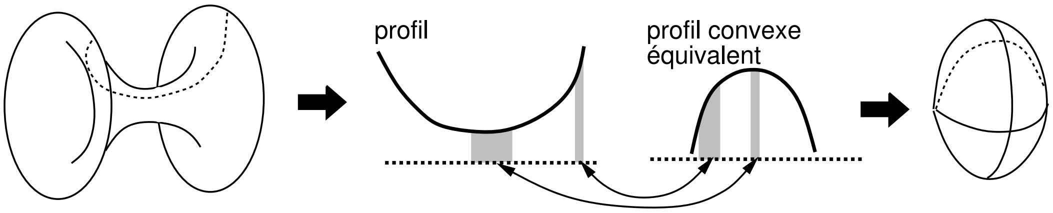


FIG. 3.2 - *Il existe une forme convexe équivalente (du point de vue des normales) pour toute forme 2D, et en 3D au moins pour les convexes 'en creux' et les formes de révolution. Au centre, on montre l'équivalence des distributions de normales pour deux directions particulières.*

de sens mais intervient indirectement quand on intègre le modèle de Phong à sa surface : il faut donc normaliser ces ellipsoïdes afin qu'ils aient le même poids quelle que soit leur forme, durant le filtrage et le rendu (l'information utile est dans le galbe de l'ellipsoïde ; lors de la combinaison d'ellipsoïdes, il n'y a aucune raison que le plus volumineux, par exemple, compte d'avantage). C'est la surface apparente moyenne qui doit servir de normalisation (puisqu'on est dans un contexte projectif), cette moyenne étant approximée en évaluant la surface apparente dans la direction des trois axes.

Il existe plusieurs façon de coder un ellipsoïde. Deux représentations utiles sont une base munie de trois longueurs, et les coefficients d'une forme quadratique. La première forme est plus commode lors de la création, alors que la seconde est préférable lors du filtrage et du rendu. Nous avons préféré coder le repère correspondant à l'ellipsoïde, dans la mesure où il est plus facile de passer d'un repère à une forme quadratique que l'inverse. Le repère est codé sous la forme de deux vecteurs (le troisième axe, le plus long, étant orthogonal et de norme 1[3]).

Afin de ne pas faire de confusion, il faut bien voir que l'on manipule des formes à deux niveaux différents lorsqu'on dessine dans le volume : la forme géométrique à grande échelle qui recouvre plusieurs voxels, et la micro-primitive locale codée dans chaque voxel, qui représente la micro-géométrie (i.e. l'échelle microscopique), cf figure 3.3. Pour les objets usuels, la micro-primitive représente la contribution de la forme *clippée*[4] au voxel, il y a donc une certaine continuité entre les deux échelles : le voxel contient plus ou moins un élément de surface, extrait de la forme globale. Mais avec ce modèle, on peut également représenter les objets anisotropiques, en utilisant une micro-primitive (le 'grain', ou la 'cristallisation')

3. Rappelons que la dimension de l'ellipsoïde n'a pas d'importance, on peut donc fixer un degré de liberté. La normalisation intervient directement lors du calcul du filtrage et du rendu .

4. i.e. la partie de la forme incluse dans le voxel.

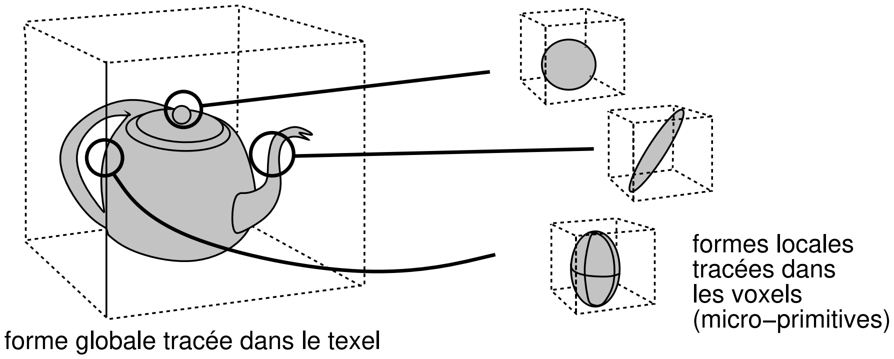


FIG. 3.3 - *Forme globale et formes locales.*

plus ou moins indépendante de la forme globale, comme on le montre en figure 3.4 : pour de l'aluminium brossé, la petite échelle est dominée par les rainures, qui sont invisibles *en tant que formes* à grande échelle. Ces rainures sont donc représentées par les micro-primitives, en utilisant des cylindres ou des ellipsoïdes plus ou moins allongés selon la netteté de ces rainures[5]. De la même façon, on peut modéliser la rugosité en augmentant le galbe des micro-primitives.

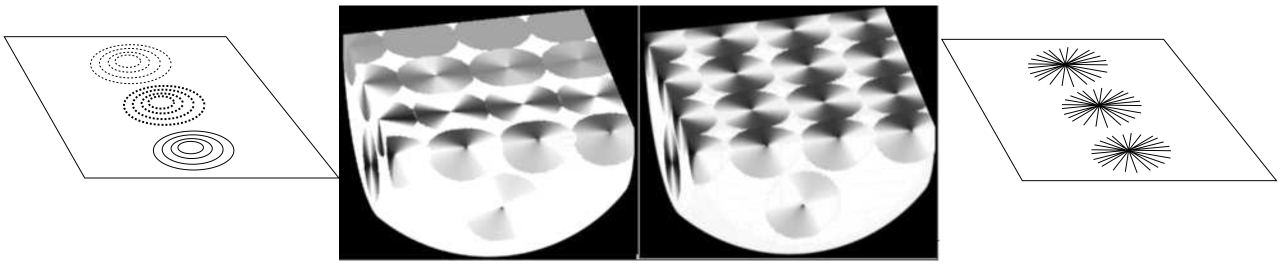


FIG. 3.4 - *Aluminium brossé (un seul texel).* A droite *les primitives locales sont des cylindres radiaux,* à gauche *ce sont des cylindres concentriques progressivement lissés jusqu'à devenir des sphères (i.e. réflectance isotrope). Les 'cylindres' sont modélisés par de longs ellipsoïdes fins.*

3.1.2 Filtrer la réflectance

L'ellipsoïde représente une partie de l'information stockée dans chaque voxel. Mais ces données sont amenées à être combinées, d'une part pour construire la représentation multiéchelle pour laquelle chaque voxel vaut la moyenne de ses huit fils, et d'autre part lors de la modélisation dans la mesure où un voxel peut être occupé par plusieurs objets différents. Il nous faut donc savoir définir la 'somme' d'ellipsoïdes.

5. Ce qui n'est pas sans rappeler les micro-cylindres que Poulin utilise pour représenter l'anisotropie sur une surface [PF90].

Définition intuitive

Les considérations sur les ellipsoïdes doivent se faire dans l'espace dual des distributions de normales plutôt que dans l'espace géométrique, dans la mesure où les primitives locales interviennent par leur réflectance plutôt que par leur géométrie. Il faut se rappeler que les comportements qui semblent à peu près corrects vis-à-vis de la géométrie peuvent s'avérer incorrects dans l'espace des normales.

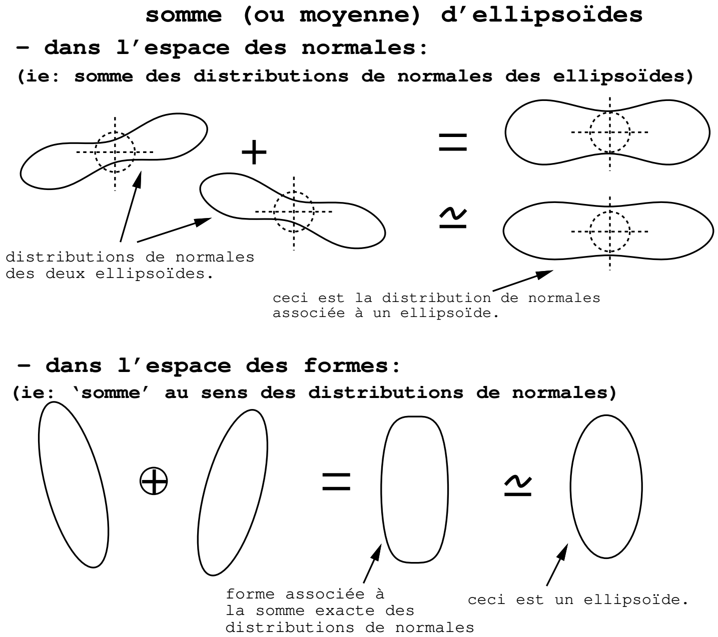


FIG. 3.5 - *'Somme' de deux ellipsoïdes.*

La somme de la distribution de normales de deux ellipsoïdes *ressemble* à la distribution de normales d'un ellipsoïde, l'approximation étant la plus mauvaise pour des ellipsoïdes 'orthogonaux', mais assez bonne pour deux primitives assez semblables. Cela s'illustre bien en 2D (cf figure 3.5).

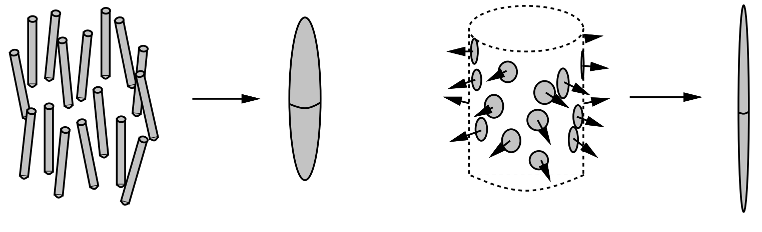


FIG. 3.6 - Filtrage de formes *: Au niveau le plus grossier, une unique primitive résulte des filtrages successifs, et doit représenter la réflectance globale de tout le texel. On peut le vérifier avec deux exemples : une prairie composées de nombreuses tiges, et un large cylindre composé de petits éléments de surface.*

Pour les volumes remplis de manière répétitive, tout se passe bien. Prenons par exemple une prairie modélisée par des tiges cylindriques dont l'orientation fluctue. Le filtrage successif d'un morceau de prairie donne, au niveau le plus grossier (quand tout l'échantillon ne correspond plus qu'à un voxel), un ellipsoïde orienté dans la direction moyenne et d'autant plus épais que la variation des tiges est importante. Autre exemple : un cylindre large est localement pratiquement plat (la courbure est faible à l'intérieur d'un petit élément de volume), il est donc modélisé par des primitives locales aplaties. Le filtrage successif de ces primitives donne au filtrage ultime un quasi-cylindre (cf figure 3.6).

Par contre pour les volumes non répétitifs, l'approximation est réelle : en 2D, la somme de deux ellipses identiques mais orthogonales donne un cercle, correspondant à une réflectance isotrope.

D'autre part, les effets de blocage (i.e. l'occultation et l'ombrage), qui sont perdus avec les méthodes classiques de mapping, sont correctement pris en compte par les textures volumiques dans la mesure où l'information 3D est conservée. Mais les choses se passent plus mal en dessous de la taille du voxel, quand la photométrie remplace la géométrie : lors de l'opération de filtrage la somme est purement géométrique, sans notion d'occultation, ce qui rend les objets cachés partiellement visibles à travers la fonction de réflectance[6]. A nouveau, les choses se passent bien pour les volumes répétitifs, dans la mesure où l'autosimilarité persiste après filtrage (s'il n'y a pas de corrélation dans les positions). Sinon il y a une approximation significative, dans la mesure où les effets de blocage n'apparaissent pas en dessous de la taille du voxel. Ceci est lié au fait qu'une fonction de distribution de normales perd l'information de position, oubliant ainsi qu'une seule normale peut correspondre à deux différentes zones d'un objet non convexe se cachant l'une l'autre.

Un autre point concerne les ombres : au cours des filtrages successifs, la différence de densité entre voxels 'dans' et 'hors' la forme décroît (plus les voxels sont grands, plus il y a de chance qu'ils soient un peu recouverts par l'objet tracé dans le volume). Cette densité tend à devenir uniformément faible, ce qui donne l'impression d'une ombre diffuse dans du brouillard plutôt que l'ombre d'un objet compact.

Bien qu'il faille rester vigilant quant à ces divers effets, dans la pratique les choses se passent relativement bien.

Définition explicite

Comme on l'a suggéré plus haut, *additionner* des ellipsoïdes consiste à choisir la forme dont la distribution de normales est la plus proche de la somme des distributions de normales des ellipsoïdes à moyenner. Afin d'éviter une longue minimisation d'erreur aux moindres carrés, nous avons choisi une façon directe de calculer cette 'somme', ce qui paraît raisonnable

6. L'une des illustrations des effets de blocage concerne les surfaces rugueuses : aux fortes incidences elles deviennent spéculaires, car l'œil ne voit plus que l'alignement des crêtes des bosses qui forme une surface apparente quasiment plane. Le masquage des creux par les bosses transforme donc dans certains cas le comportement en réflectance.

vu la simplicité de la primitive. Néanmoins il n'existe pas de solution analytique quand on intègre à la surface d'un ellipsoïde, il faut donc trouver une approximation.

A partir de la forme quadratique $M^t.Q.M = 1$ associée à un ellipsoïde, il est possible d'obtenir l'expression de la distribution de normales, que l'on peut interpréter comme la densité de probabilité f d'avoir une normale N dans une direction donnée. Elle est égale au jacobien de la bijection entre la surface de l'ellipsoïde et l'espace dual des normales sur la sphère de Gauss : $f_Q(N) = det(Q^{-1})/(N^t.Q^{-1}.N)^2$
Le problème est alors de trouver Q tel que f_Q soit le plus proche possible de $f_{Q_1} + f_{Q_2}$. L'étude de f_Q montre que la fonction est 'presque-additive' en Q^{-1} : en posant $g_{Q^{-1}} = f_Q$, on a $g_{\lambda.Q^{-1}} = \lambda.g_{Q^{-1}}$, donc la somme de primitives identiques est conservative. Le développement limité de $g_{Q_1^{-1}} + g_{Q_2^{-1}}$ pour deux ellipsoïdes identiques tournés de θ l'un par rapport à l'autre est $g_{Q_1^{-1}+Q_2^{-1}} + O(sin^2(\theta))$. Ceci confirme que l'on obtient une bonne approximation pour de faibles angles entre ellipsoïdes.

Nous définissons donc comme 'l'ellipsoïde moyen' celui dont l'inverse de la forme quadratique est la moyenne de l'inverse des formes quadratiques des ellipsoïdes à moyenner. Les ellipsoïdes qu'on additionne sont bien sûr pondérés par la densité ρ_i qui leur est associée dans le voxel :

$$Q^{-1} := \frac{\sum \rho_i.Q_i^{-1}}{\sum \rho_i}$$

La forme quadratique associée à un ellipsoïde s'obtient facilement à partir de la base et des longueurs, qui correspondent respectivement à une matrice R (orthogonale) et L (diagonale) : $Q_i := R_i.L_i^{-2}.R_i^t$. Réciproquement, la base et les longueurs de l'ellipsoïde résultant sont obtenus à partir des valeurs propres λ_i ($r_i = \lambda_i^{-2}$) et des vecteurs propres de Q.

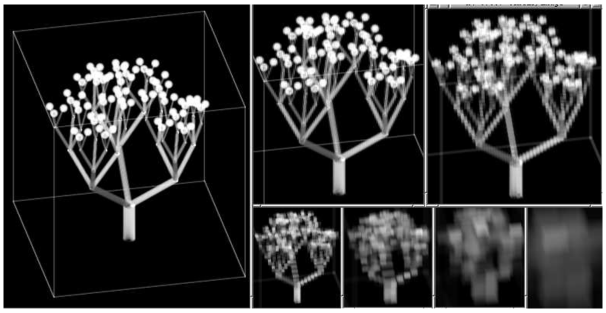


FIG. 3.7 - *Résolutions successives de l'octree de 256^3 à 4^3. Sur une Indigo^2, la construction prend 8 sec, et le rendu 20 sec pour la première image, 12 et 10 pour les deux suivantes, à la résolution 444×444. (Dans leur usage normal, la dimension des images devrait être successivement divisée par deux.)*

La figure 3.7 montre les itérations successives du filtrage des voxels : la série d'images ressemble à des opérations successives de lissage d'image, alors qu'elles sont le rendu du *filtrage géométrique* répété, précalculé au moment du modeling : ce filtrage là ne coûte rien lors du rendu.

3.1.3 Rendre la réflectance

Lors du rendu, il faut cumuler les intensités réfléchies le long du rayon. Au niveau de chaque voxel traversé, il faut calculer la réflectance d'une micro-primitive, c'est à dire la proportion d'énergie d'une source de lumière qui est réfléchie vers l'œil par tout l'ellipsoïde, en accord avec le modèle de réflection locale de Phong.

Les interactions lumineuses avec l'environnement sont résolues par le rendu volumique, qui évalue la quantité de lumière arrivant en un point, et accumule l'illumination et l'opacité le long de chaque rayon. L'illumination locale émise est égale au produit de la lumière incidente et de la fonction de réflectance pondérée par la densité du voxel[7] (albédo). La densité est également modulée par la micro-primitive (anisotropie de l'occultation[8]), en fonction de sa surface apparente dans la direction du rayon.

Evaluer la réflection totale de l'ellipsoïde revient à intégrer le modèle de réflection de Phong sur sa surface apparente. L'intégration sur les coniques donne rarement des solutions analytiques[9], on utilise donc un schéma numérique, en échantillonnant uniformément la surface apparente, cf figure 3.8 (ces calculs sont repris et étendus à la section 4.3.3).

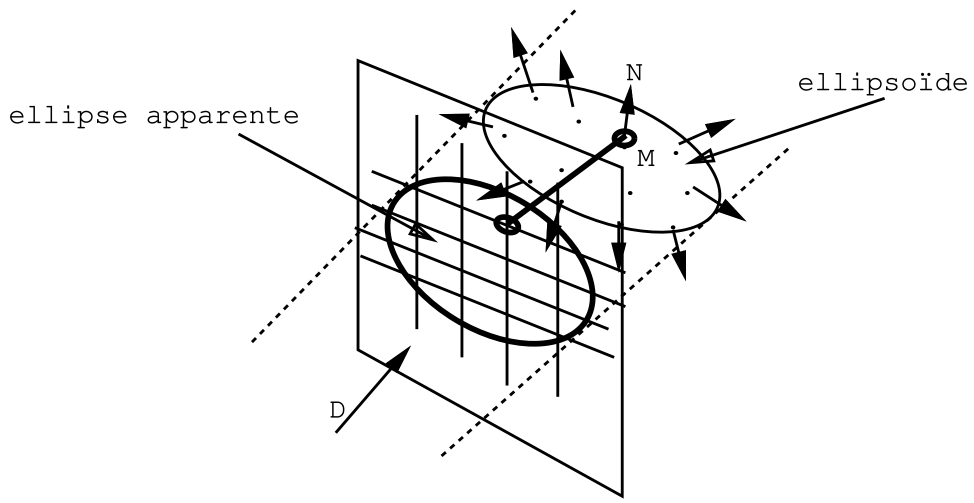


FIG. 3.8 - *Rendu d'un ellipsoïde.*

Pour cela il faut d'abord trouver un encadrement de la zone d'intégration. Le rectangle exinscrit à l'ellipse apparente est tiré de la forme quadratique de l'ellipsoïde : si P est la

7. La densité est prise au prorata de l'épaisseur de voxel traversée par le rayon.

8. La densité - ou plus exactement la probabilité d'occultation - est anisotropique, dans la mesure où l'interception des rayons par un objet irrégulier dépend de l'orientation. Pour un petit objet, l'ellipsoïde utilisé pour encoder la réflectance décrit également l'occultation, via sa surface apparente dans la direction considérée. Cet ellipsoïde est normalisé, et vient pondérer la probabilité d'occultation (i.e. l'occultation moyenne) stockée séparément dans le voxel.

9. Nous avons pourtant longuement essayé...

normale au plan contenant les limbes[10] de l'ellipsoïde ($P = Q.D$,D étant la direction du regard[11]), l'ellipse apparente a pour forme quadratique $Q' = Q - P.P^t/(D.P)$ dont il suffit de chercher les valeurs et vecteurs propres. Pour chaque échantillon sur l'ellipse on sait retrouver le point correspondant sur l'ellipsoïde (l'intersection d'un rayon et d'une conique se traduit par la résolution d'une équation du second degré, dont la solution à retenir est celle qui est la plus proche de l'œil). Remarquons que l'on n'a besoin d'échantillonner que la moitié de la surface de l'ellipse, dans la mesure où les points symétriques par rapport au centre peuvent être obtenus sur l'ellipsoïde en même temps (les termes de l'équation à résoudre sont identiques au signe près).

Pour évaluer la lumière incidente, il faut lancer un rayon d'ombrage vers les sources de lumière afin de tester l'occultation et l'atténuation. Nous nous plaçons dans l'hypothèse de faible albédo[12], nous n'avons donc qu'à prendre en compte l'atténuation sur ce chemin, les réflections secondaire pouvant être négligées.

Il faut remarquer que les formes opaques complexes sont les bienvenues : le calcul est d'autant plus rapide que la forme est occlusive. Ainsi, seule la surface d'un objet compact est visible, donc même un objet compact à fortes circonvolutions apparaît à l'œil comme une surface (discontinue). A l'opposé, les rayons traversent de part en part un volume peu dense, et de nombreux rayons d'ombrage doivent êtres lancés depuis les divers point de la matière diffuse ce qui coûte bien plus cher.

3.2 Le texel

Nous avons pour l'instant spécifié la construction, le filtrage et le rendu d'un voxel isolé. Ces voxels constituent le volume de référence, dont les instances répétées et déformées sont les texels destinés à couvrir les surfaces à texturer. Il nous faut donc à présent spécifier la construction et le rendu du texel.

10. les limbes correspondent aux lieux de l'objet qui sont tangents au regard (i.e. c'est la projection inverse de l'ellipse apparente, qui est une ellipse plane).

11. En effet, M est sur les limbes si la normale en M, qui vaut $Q.M$, est orthogonale à D, soit $(Q.M).D = 0 = M.P$, ce qui signifie que M est sur le plan orthogonal à P.

12. L'hypothèse de faible albédo est souvent faite en rendu volumique, parce qu'elle simplifie énormément les calculs. En réalité, l'inter-réflection entre petits objets est assez importante, c'est elle qui conduit la lumière jusqu'au fond d'un arbre ou jusqu'à la base des poils de la fourrure. Faire cette hypothèse revient donc plus exactement à dire : on ne tient pas compte des réflections secondaires.
A noter que la transparence des petits objets joue un rôle important dans le même sens.

3.2.1 Modéliser le texel

Le volume de référence d'une texture volumique est représenté par un octree, chaque étage correspondant à une résolution d'autant plus grossière qu'on remonte vers la racine. Chaque voxel nœud contient une probabilité d'occultation (en flottant), 6 paramètres pour l'ellipsoïde (chacun en virgule fixe sur 16 bits), et un pointeur sur un bloc de huit fils. Ce pointeur unique permet d'éviter tant la place qu'occuperaient 8 pointeurs séparés que celle qui serait prise par les entêtes cachés du système pour chaque bloc de mémoire alloué. Il peut sembler excessif de coder la densité sur 32 bits ; il se trouve que le compilateur arrondit l'occupation d'une structure aux 32 bits supérieurs, la quantité de mémoire consommée sera donc la même que l'on utilise un octet ou 4 pour la densité. Quant aux paramètres de l'ellipsoïde, ils correspondent aux deux axes les moins longs, l'axe principal étant considéré de longueur 1 et facile à retrouver à partir des deux autres. Les coefficients sont donc tous inférieurs à 1, ce qui permet ce stockage compact sur 16 bits[13].

```
type voxel
      { float densité                         ' probabilité d'occultation
        short r2x,r2y,r2z,  r3x,r3y,r3z       ' ellipsoïde
        voxel[8] *fils                        ' 8 voxels fils
      }
```

Les données sont construites à la résolution maximum indiquée par l'utilisateur, en traçant les formes dans le volume[14], puis une première passe propage le filtrage vers la racine de l'octree, en filtrant les 8 fils pour constituer les valeurs de chaque voxel père. Les réflectances sont moyennées comme indiqué en section 3.1.2, les probabilités d'occultations se filtrent en considérant la probabilité d'occultation moyenne $s = \frac{1}{8}\sum_1^8 \rho_i$, puis en calculant l'opacité d'un bloc qui serait composé de 8 pavés de probabilité d'occultation s : un rayon doit traverser en moyenne deux de ces blocs, la probabilité d'occultation moyenne du bloc est donc $s+(1-s)s$ soit $2s - s^2$.

Une deuxième passe de compaction est faite pour éliminer les redondances inutiles, en l'occurrence les zones de voxels à valeurs presque identiques, qui sont donc également identiques à la valeur du nœud père. Ainsi, les nœuds feuilles ne sont pas tous de la même taille, les voxels restant subdivisés surtout le long des contours (c'était d'ailleurs l'un des buts initiaux des quadtrees que d'offrir une représentation compacte des images), ce qui fait qu'un objet simple nécessite de l'ordre de n^2 voxels pour un volume de taille n^3. En fait, nos octrees sont donc généralement particulièrement creux.

13. On a même d'abord stocké sur 8 bits, mais le filtrage récursif nécessite plus de précision.

14. Pour l'instant on ne s'intéresse pas à la façon de définir ces formes ; cela sera l'objet de la section 4.1.

3.2.2 Rendre le texel

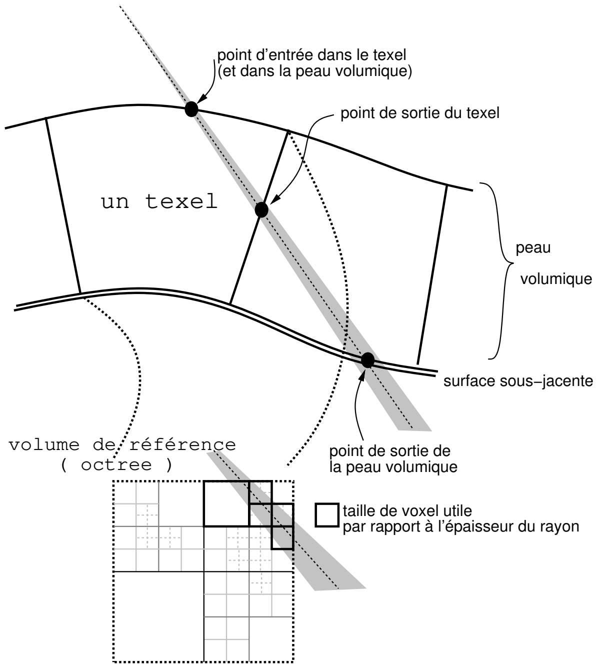


FIG. 3.9 - *Parcours du rayon*

On ramène le rendu d'un texel à celui du volume de référence, en reportant les points d'entrée et de sortie du rayon et en considérant que celui-ci se propage toujours en ligne droite, cf figure 3.9 (dans le volume de référence, il devrait en toute rigueur être courbe[15]). On connaît l'ouverture du rayon, c'est à dire la taille de la projection inverse d'un pixel sur le texel. Le principe du mip-mapping consiste à utiliser la taille de pixel de texture (ici un voxel de l'octree) la plus proche du diamètre du rayon (ou plus exactement, à interpoler entre les deux tailles les plus proches). On va alors utiliser l'étage de l'octree correspondant à cette résolution optimale[16] (ce qui revient à supposer que le cône du rayon est localement un cylindre au niveau du texel), puis traverser alors le volume comme un rendu volumique classique le ferait en cumulant l'intensité pondérée par l'opacité et en multipliant la transparence le long du rayon. On peut ainsi considérer que l'on fait du cône-tracing plutôt que du ray-tracing, dans la mesure où l'on va rendre le volume en intégrant sur toute l'épaisseur du rayon.
La structure de l'octree nous invite à le traverser de façon récursive, selon l'algorithme ci-dessous, en initiant le processus par `cross(racine,1,ouv, point1,t1, point2,t2)` :

15. Le traitement de la courbure des rayons fait l'objet de la section 4.3.3.
16. En fait on fait le calcul pour les résolutions justes inférieure et supérieure, et on interpole le résultat.

```
cross (voxel, r,                     ' noeud courant et son diamètre (racine=1)
       ouv,                          ' ouverture du rayon par rapport au texel
       {ui,vi,wi},ti,                ' entrée et sortie du voxel  (coordonnées
       {uo,vo,wo},to                 '        locales et ordonnée sur le rayon)
      )
  si T opaque alors fin
  if (r<=ouv ou pas de fils)               ' résolution à utiliser
    rendu_local(voxel,r, T,L) -> (T,L)     ' Transparence et Lumière
    fin
  ' on n'a pas encore atteint la bonne échelle, il faut subdiviser
  empile({ui,vi,wi},ti)
  ' intersection du rayon avec les 3 plans médians
  intersection avec u=.5 ->  (v,w, t)     ' points d'intersection avec
  si existe alors empile ({.5,v,w},t)     ' la grille à la résolution
  intersection avec v=.5 ->  (u,w, t)     ' supérieure.
  si existe alors empile ({u,.5,w},t)
  intersection avec w=.5 ->  (u,v, t)
  si existe alors empile ({u,v,.5},t)
  empile ({uo,vo,wo},to)
  trier liste par t croissant
  point1 = liste[1], t1=point1.t
  pour tout point de la liste au delà de 1
    point2 = liste[i] , t2 = point2.t
    du = (point1.u>.5) ou (point1.u=.5 et uo>ui)  ' l'octant du segment P1-P2
    dv = (point1.v>.5) ou (point1.v=.5 et uo>ui)  ' est défini par les 3 bits
    dw = (point1.w>.5) ou (point1.w=.5 et uo>ui)  ' [dw|dv|du]
    D  = (du,dv,dw)
    cross (voxel.fils[du+2.dv+4.dw],r/2,ouv, {2*point1-D},t1, {2*point2-D},t2)
    point1 = point2, t1 = t2
```

Cet algorithme se prête à de nombreuses optimisations. D'abord on peut facilement le dé-récursiver, en gérant nous même la pile des paramètres d'appel. Cela simplifie notamment l'arrêt de la traversée dès que le rayon est totalement opacifié. Ensuite il se trouve qu'on utilise une arithmétique très simple, qui se prête bien au calcul en nombre entier. L'opération la plus coûteuse est la règle de 3 du calcul d'intersection qui comprend une division : l'intersection avec le plan $(u = .5)$ est $(P2_i - P1_i) * (.5 - ui)/(uo - ui) + P1_i$ (à noter que toutes les autres multiplications et divisions se ramènent à des décalages de bits dans la mesure où le second facteur est 2). Nous traitons le cas complet (notamment avec l'interpolation entre les deux résolutions les plus proches) en Annexe D.

3.3 De la caméra au texel

Dans les sections précédentes, nous avons d'abord décrit la construction et le rendu au niveau d'un voxel (en 3.1), puis au niveau d'un texel (en 3.2). Continuons à prendre du recul et abordons les deux échelles suivantes, la texture et la scène. (On trouvera les détails techniques en Annexe D.)

Modéliser la surface

Nous utilisons des surfaces produites avec d'autres outils, bien que nous ayons développé quelques routines permettant de construire les plus simples d'entre elles (plans, sphères, tores, cylindres, boîtes, surfaces paramétrées). Nous disposons également d'un mode particulier où l'on simule un plan infini.

La méthode de base telle que nous l'avons exposée dans ce chapitre contraint à l'utilisation de surfaces composées de quadrangles, puisque pour l'instant nous utilisons le même procédé de mapping que Kajiya, un texel correspondant exactement à un patch bilinéaire. Nous supprimerons cette contrainte au chapitre suivant.

Assez classiquement, nous entretenons une liste de surfaces, une liste de faces et une liste de sommets. Les adjacences entre faces sont pré-traitées de sorte que chaque face pointe vers ses voisines. Une sphère englobante (tenant compte du texel associé à la face) est également précalculée. La structure liée à un point contient ses coordonnées spatiales, ses coordonnées texture (en prévision de la suite), et un vecteur hauteur (déterminant une arête verticale de texel). On associe également une sphère englobante à la surface, mais surtout une grille 3D (ou *grid*), subdivisant en voxels l'espace de la boîte englobante afin d'y repérer les facettes en fonction de leur voisinage.

Rendre la surface

Un rayon partant de l'œil et passant successivement à travers chaque pixel de l'écran interroge la scène. De façon très classique, et en utilisant toutes les optimisations permises par les sphères englobantes et la grille 3D des objets[17], on obtient le numéro de la face intersectée la plus proche, ainsi que les coordonnées barycentriques (vis à vis de la face et plus généralement du texel qui la prolonge) du point d'impact.

17. Il faut bien se rendre compte qu'en ray-tracing, en dépit de toutes ces optimisations, on est quand même amené à tester l'intersection d'une multitude de facettes inutilement (c'est l'obtention de coordonnées barycentriques hors de [0,1] qui indique qu'une facette n'est pas bonne). Toutes les méthodes d'optimisation visent à diminuer le nombre de tests inutiles, mais en même temps ajoutent un coût de traitement. D'autre part, il faut garder à l'esprit que dans notre cas les facettes sont des patches bilinéaires, dont le coût d'intersection est plus élevé que pour une facette plane.

Plus que le point d'impact, on détermine également la taille du rayon conique engendré par le pixel, afin de choisir la taille de voxel appropriée. Cette 'ouverture' du rayon est égale à $\frac{\sqrt{3}}{H}.\frac{d}{f}.(\vec{D}.\vec{a})$ avec H la résolution verticale de l'image, d la distance du point d'impact et $\vec{D}$ sa direction, f la 'distance focale'[18] de la caméra et $\vec{a}$ la direction pointée par la caméra. $\frac{1}{H}.\frac{d}{f}$ est une simple règle de trois qui donne la largeur d'un faisceau d'un pixel d'épaisseur au bout d'une distance d, le produit scalaire corrige le fait que le pixel en question est pris sur un plan et non une sphère, et le terme $\sqrt{3}$ prend en compte l'étalement moyen de la tâche selon l'inclinaison de la surface sur laquelle le cône tombe (c'est approximatif, mais cette grandeur unique qu'est l'ouverture doit s'appliquer à toutes les faces du voxel). Il faut ensuite ramener cette ouverture dans l'espace du texel, en la divisant par le min ou par le max des côtés du texel selon que l'on préfère un peu d'aliasing ou un peu de flou. L'ouverture du rayon est alors directement comparable à la taille d'un voxel (elle vaut 1 quand le rayon conique embrasse tout le texel), et peut donc être utilisée pour le mip-mapping.

Cette façon de procéder n'est pas parfaite, d'autant plus qu'elle suppose que l'ouverture est la même pour tout le texel, c'est à dire que le cône du rayon est localement un cylindre (mais cette hypothèse est raisonnable puisque les texels sont censés être vus de loin). Dans la version étendue du modèle, nous affinons le réglage en utilisant le min ou le max du jacobien de la déformation subie par le texel, mais de toutes façons l'usage d'une valeur unique alors que la compression apparente de la texture n'est pas la même selon les 3 axes du texel est forcément approximatif. Rappelons toutefois que ce défaut est inhérent au mip-mapping et existe également en 2D.

Parcourir la texture volumique

Il faut maintenant parcourir la peau volumique jusqu'à ce qu'on en sorte ou que le pixel devienne totalement opaque. Nous connaissons le point d'entrée dans le texel puisqu'il est donné par l'intersection du rayon et de la surface. On cherche alors le point de sortie du texel (qui ne fait pas forcément sortir de la texture, notamment si la vue est rasante). A partir de là on peut se ramener au rendu volumique dans le volume de référence comme expliqué plus haut (cf 3.2.2). Après quoi, si le rayon n'est pas opacifié, il faut continuer : à l'aide de la liste d'adjacence contenue dans la face, on trouve la facette voisine concernée par la poursuite du rayon, on convertit les coordonnées barycentriques du point de sortie du texel pour en faire un point d'entrée du texel voisin, et ainsi de suite.

18. En synthèse d'images, c'est le terme impropre choisi pour désigner la distance de l'œil à l'écran.

3.4 En résumé

- Outre une pseudo-densité (la probabilité d'occultation), chaque voxel contient une fonction de réflectance représentée par la distribution de normales d'un ellipsoïde, qui permet de simuler l'interaction de la lumière avec la géométrie inférieure à la taille du voxel.
- Ces voxels sont organisés en octree, ce qui permet à la fois de comprimer fortement le volume et de représenter les données à plusieurs échelles. Celui-ci constitue le volume de référence de la texture.
- Cet octree est utilisé lors du rendu exactement comme le mip-mapping en 2D, pour fournir des données préfiltrées lorsqu'un rayon atteint la surface, de manière à éviter tant l'aliasing que le sur-échantillonnage.
- Les instances de cet octree, i.e. les copies *formelles* (l'octree n'est stocké qu'une fois) déformées, recouvrent une surface en se collant les unes contre les autres de manière à former une couche continue. Ces copies sont les *texels*[19].

3.4.1 Algorithme

Voici l'algorithme qui résume l'ensemble du processus de rendu, et notamment du parcours dans les diverses structures. Se reporter à l'Annexe D pour les détails.

Remarque : on a fait apparaître dans cet algorithme une structure que l'on introduit au chapitre suivant. Les *boîtes* correspondent aux texels de Kajiya, on verra que dans notre implémentation finale celles-ci contiennent plusieurs *texels* (copies du volume de référence).

```
j = hauteur de l'image
démultiplier le programme sur N threads

tant qu'il reste des lignes à calculer,
: s'occuper de la ligne j et décrémenter j :
: |
: | i balaye la ligne j de l'image
: | : lancer le rayon(i,j) :
: | : |
```

19. Il nous est néanmoins arrivé de parler par abus de langage *du* texel, pour qualifier le volume de référence.

```
: | : |  couleur pixel Ctot=0, transparence Ttot=1
: | : |  trouver la face f intersectée par le rayon en (U,V,W) :
: | : |  |
: | : |  | trier les objets dont la sphère englobante est traversée
: | : |  | : parcourir leur grille
: | : |  | : : trier les faces (sto dans le voxel du grid) dont la sphère est traversée
: | : |  | : : : calculer l'intersection avec leur surfaces libres
: | : |  | : : : conserver celle à distance min
: | : |
: | : |  tant qu'on n'est pas ressorti de la peau volumique
: | : |  :  chercher le point de sorti de la boîte (U',V',W') :
: | : |  :  |
: | : |  :  | intersection du rayon avec les 6 côtés (patches bilinéaires)
: | : |  :  | trier et garder celui à distance positive min sur le rayon
: | : |  :
: | : |  :  traverser la boîte :
: | : |  :  |
: | : |  :  | (u,v,w) = trilin(U,V,W)
: | : |  :  | tant qu'on n'est pas sortie de la boîte
: | : |  :  | :  chercher le point de sortie du texel (u',v',w')
: | : |  :  | :  traverser le texel :
: | : |  :  | :  |
: | : |  :  | :  | empiler le segment de rayon (u,v,w)-(u',v',w') taille 1
: | : |  :  | :  | tant qu'il y a des segments empiles,
: | : |  :  | :  | :  dépiler un segment
: | : |  :  | :  | :  est-ce le voxel optimal ? (taille<ouverture ou taille=min)
: | : |  :  | :  | :    si oui, rendre le voxel :
: | : |  :  | :  | :       |
: | : |  :  | :  | :       | si le voxel n'est pas vide
: | : |  :  | :  | :       |   calculer la réflectance Id,Is et l'occultation A :
: | : |  :  | :  | :       |   |
: | : |  :  | :  | :       |   | déterminer l'ellipse apparente
: | : |  :  | :  | :       |   | calculer la surface apparente A
: | : |  :  | :  | :       |   | Id=0 Is=0 n=0
: | : |  :  | :  | :       |   | échantillonner sa surface -> x,y
: | : |  :  | :  | :       |   |   déterminer le point z(x,y) sur l'ellipsoïde
: | : |  :  | :  | :       |   |   en déduire la normale N
: | : |  :  | :  | :       |   |   calculer diffus d et spéculaire s
: | : |  :  | :  | :       |   |   Id=Id+d, Is=Is+s, n=n+1
: | : |  :  | :  | :       |   | Id=Id/n, Is=Is/n
: | : |  :  | :  | :       |
: | : |  :  | :  | :       |   envoyer rayon d'ombrage depuis trilin_inv(u,v,w) vers lum
: | : |  :  | :  | :       |   -> luminosité O
: | : |  :  | :  | :       |   en déduire couleur du voxel en fn de la matière Ca,Cd,Cs :
: | : |  :  | :  | :       |    | C = A(Ca+O(Cd.Id+Cs.Is))
: | : |  :  | :  | :       |   Ctot = Ctot + Ttot. C
: | : |  :  | :  | :       |   Ttot = Ttot * (1-A)
: | : |  :  | :  | :       |   si le rayon est opaque, fin du rayon
: | : |  :  | :  | :
: | : |  :  | :  | :    si non,
: | : |  :  | :  | :      subdiviser le segment selon les voxels fils
: | : |  :  | :  | :      trier les sous-segments
: | : |  :  | :  | :      : les empiler (ui,vi,wi)-(uo,vo,wo) taille/2
: | : |  :  | :
: | : |  :  | :(u,v,w) = (u',v',w')
: | : |  :
: | : |  :(U,V,W) = (U',V',W')
: | : |
: | : | poursuivre le rayon dans la scène depuis (U,V,W)
```

3.4.2 Résultats

Afin d'illustrer l''illusion géométrique' engendrée par les texels (et plus particulièrement par leurs fonctions de réflectance), nous montrons d'abord en figure 3.11.1 un exemple de données complexes stockées dans un seul grand texel (un échantillon ordinaire de texture n'est pas supposé être si complexe !).

L'étrange image de buissons et de sphères de la figure 3.11.2 montre le filtrage progressif des texels avec l'éloignement : un texel à résolution 256^3 contient un buisson de 2000 feuilles en forme de galette ; 50×500 texels sont mappés sur une surface plane. Cela représente l'équivalent d'une base de donnée géométrique de 50 millions de galettes, soit au moins 400 millions de facettes triangulaires. Bien que l'on n'utilise qu'un rayon par pixel, il n'y a pratiquement pas d'aliasing. Les calculs prennent 14 minutes sur une *Indigo*^{2} à la résolution 444×444.

Avec l'image du velours 3.13.1, on peut observer les effets anisotropiques engendrés par la répétition de petits motifs en relief : chaque poil obéit localement au modèle de Phong, mais en fonction de l'orientation on peut voir l'accumulation du haut illuminé des poils (en haut de la bosse), ou l'ombre au pied des poils (à droite de la bosse), en plus des effets d'auto-occultation pour chaque poil (équivalents aux phases de la lune, voir figure 3.10). Sur la gauche on observe l'ombre 'classique' de la bosse, la lumière venant de droite.

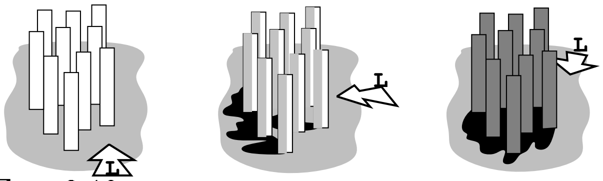


FIG. 3.10 - *Jeux d'ombres sur les poils du velours.*

Les trois paires d'images 3.13.2 à 3.13.4 montrent divers types de texel à résolution 128^3, ainsi que leur mapping sur des surfaces composées de 100 à 1000 patches bilinéaires (quand on voit les texels de trop près, les voxels sont parfois visibles individuellement). A la taille vidéo 768×576, les calculs prennent entre 5 et 20 minutes, une large part de ce temps étant consacré au calcul de l'intersection entre les rayons et les patches de la géométrie.

Avec l'image de la forêt 3.13.2, on constate que la représentation de l'échantillon peut être grossière : la quantité de détails à peindre dans le texel doit juste correspondre à la distance minimale que l'utilisateur souhaite garantir au cours de son animation. Sur la scène des échafaudages 3.13.4, le volume de référence montre que l'information de réflectance peut produire l'illusion d'une résolution élevée (cependant, les détails plus fins que la taille d'un voxel deviennent légèrement transparents). Le tore à fourrure de l'image 3.13.3 illustre le cas des texels cycliques (comme pour la prairie de l'image 3.13.1). Sur l'image texturée, il reste un peu d'aliasing dû au fait que les texels sont fortement déformés lors du 'coiffage' de la fourrure.

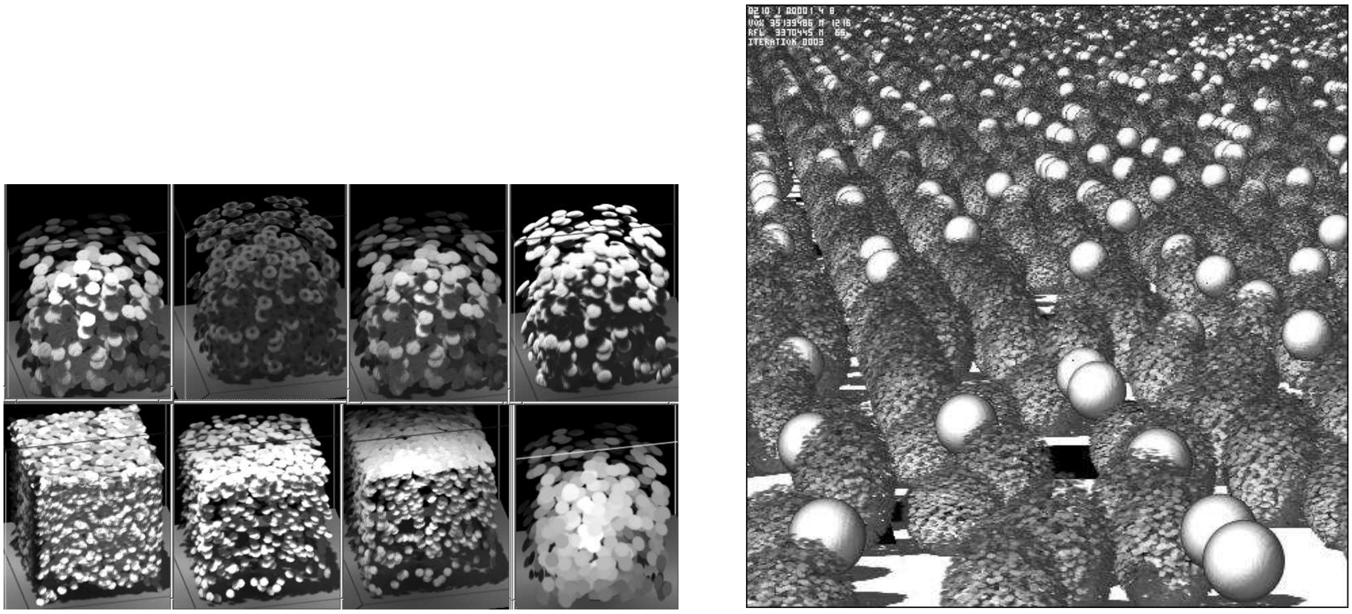


FIG. 3.11 - A gauche *: buissons taillés, avec différentes sortes de micro-primitives pour l'anisotropie. (Chaque texel est entouré d'un cadre dont on voit l'ombre sur le sol, ce dernier étant lui-même légèrement anisotropique).* A droite *: paysage de* 50×500 *buissons et sphères (14 minutes). Noter la continuité du filtrage entre l'avant-plan et l'arrière-plan.*

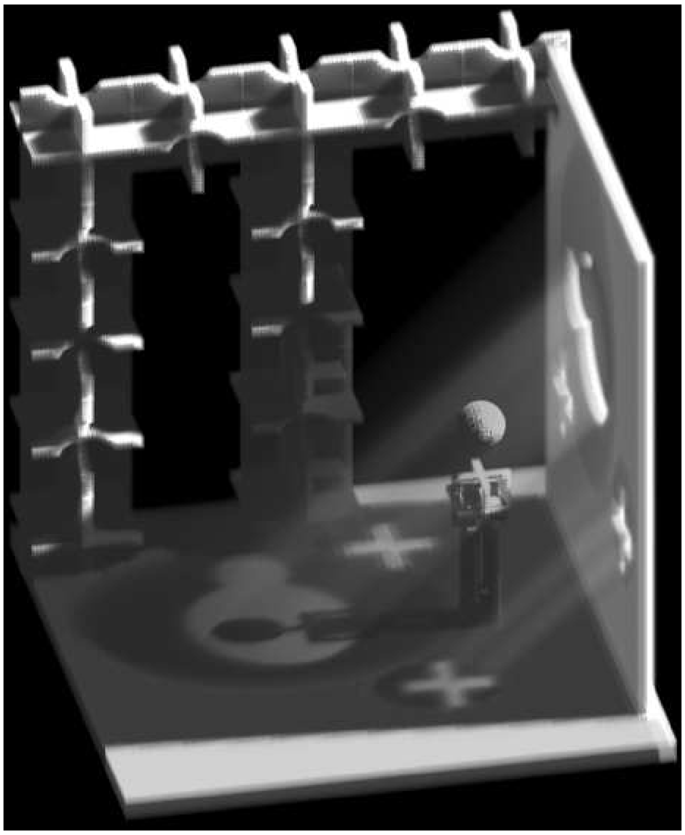
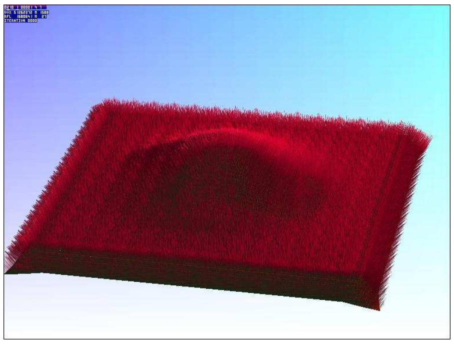
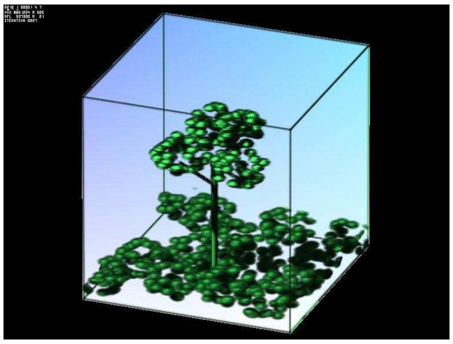


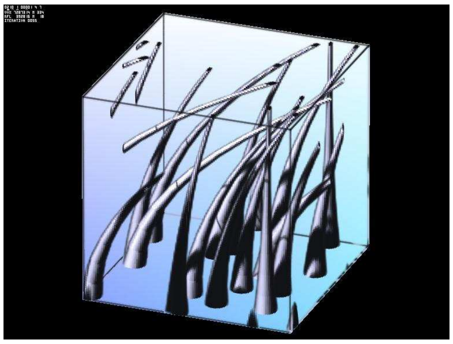
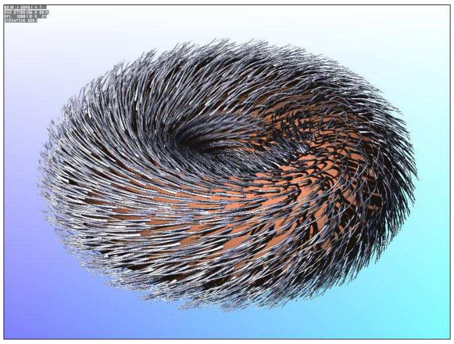
FIG. 3.12 - *Remarque en aparté : à l'intérieur d'un texel, le calcul ressemble beaucoup à du rendu volumique. La preuve, l'image ci-dessus est un texel unique...*

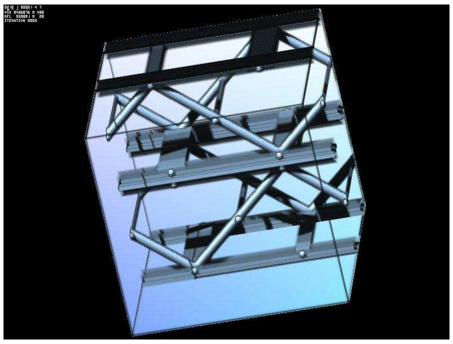
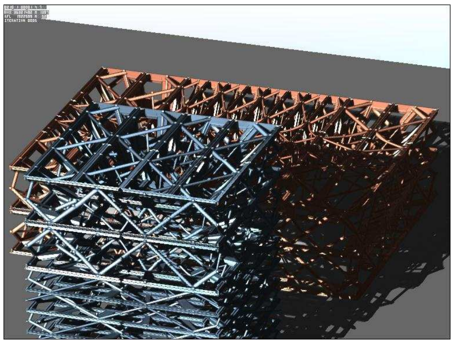
3.13.1: *A gauche* : une prairie de 1400 patches bilinéaires. Le texel contient 16 brins d'herbe qui ont une section en'V' ; sa résolution est de 128^3 (compression 91%). *A droite* : velours. La distribution des cylindres perpendiculaires à la surface engendre une anisotropie à grande échelle.




3.13.2: *A gauche* : un texel seul, de résolution 128^3 (compression 92%), conçu pour être vu de loin. *A droite* : mapping sur une colline composée de 578 patches bilinéaires (23 minutes de rendu, dont 12 pour le calcul d'intersection avec les patches).

3.13.3: *A gauche* : poils tracés cycliquement dans un volume 128^3 (comprimé à 93%). Le rendu du texel isolé prend 3.5 minutes. *A droite* : mapping sur un tore composé de 240 patches (12 minutes).

3.13.4: *A gauche* : texel représentant des matériaux de construction, à la résolution 128^3 et comprimé à 92% *A droite* : mapping de l'échafaudage sur des formes cubiques (81 patches bilinéaires, 14 minutes).

FIG. 3.13 - *Illustration avec des scènes complexes du modèle de texels.*

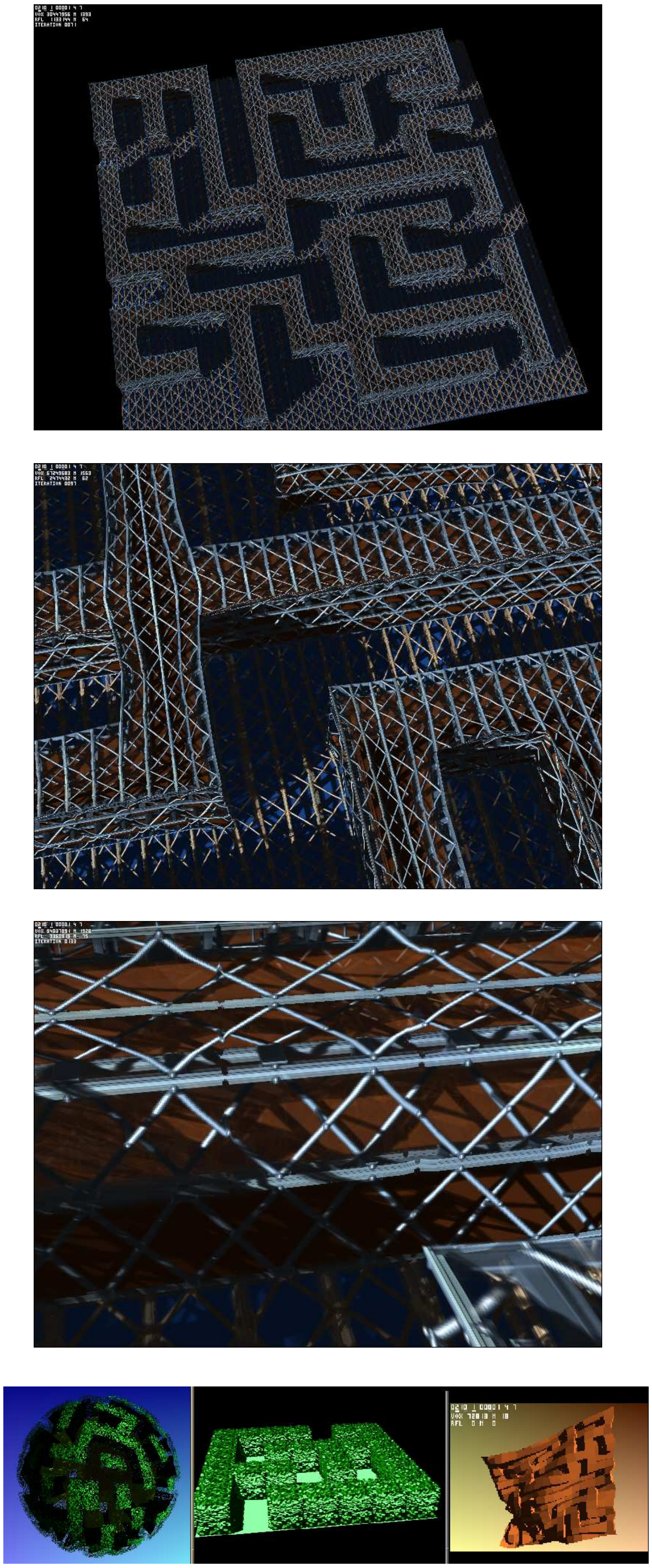


FIG. 3.14 - *Trois extraits d'une animation présentant un zoom continu sur un labyrinthe, partant d'une distance telle qu'un texel se projette en un pixel, jusqu'à arriver assez près pour qu'un voxel se projette en un pixel.* En bas : *variantes sur le thème du labyrinthe.*

A titre de comparaison, la fameuse image du Teddy Bear qui illustre le papier de Kajiya et Kay nécessite l'équivalent d'une douzaines d'heures CPU sur un IBM 3090, à résolution 1280×1024. Une autre méthode pour simuler la géométrie répétitive, dont il faut également comparer les performances, est celle des systèmes de particules. Le réalisme induit par la complexité géométrique est également impressionnant, mais le rendu de ces images prend plusieurs heures, l'illumination et la forme des objets sont spécifiés par programmation, et le rendu est délicat à intégrer dans une scène ray-tracée ordinaire (laquelle doit prendre en compte les interactions lumineuses comme les ombres et les reflets).

3.5 Discussion

3.5.1 Limites de la méthode

L'approche des textures volumiques telle que nous l'avons exposée à ce stade a bien sûr des limites et quelques inconvénients :

- en tant qu'approche texturelle, les textures volumiques s'appliquent à la modélisation d'objets présentant un motif répétitif, et aux surfaces dont le mapping peut être effectué avec des déformations raisonnables.
- il semblerait naturel de pouvoir imbriquer deux 'textures géométriques', ou de poser un objet réel (i.e. polyédrique) et une texture creuse au même endroit, mais cela s'avère en fait très difficile à implémenter [20].
- la texturation suppose qu'il n'y ait pas de mouvement dans le motif lui-même, faute de quoi le motif de référence devra être recalculé à chaque image.
- il n'y a jamais de préfiltrage de données exact, comme dans le cas du mip-mapping en 2D : il faut faire un choix entre le flou et l'aliasing, ou bien accepter de sur-échantillonner un peu dans les mauvais cas. Quoiqu'il en soit, les choses se passent infiniment mieux qu'avec une scène réellement composée de millions de petits objets polyédriques répétitifs !
- la représentation de la fonction de réflectance choisie (ellipsoïde) est relativement simple et par conséquent approximative, on perd donc de l'information au filtrage.
- le filtrage ne prend pas parfaitement en compte les effets de blocage, notamment en dessous de la taille du voxel, ce qui peut limiter l'usage des texels dans certains cas.
- il faudrait savoir estimer finement la taille de voxel à utiliser dans le cas des fortes déformations afin de supprimer l'aliasing résiduel sans flouter les données (on répondra en

20. car le rendu ne se fait pas au même niveau : les objets polyédriques sont intersectés par le ray-tracing ordinaire, tandis que la texture est traversée par notre rendu spécifique. Il faudrait donc mixer les deux, ce qui a probablement un coût...
On traite en section 5.3 de quelques extensions souhaitables du modèle.

partie à ce point en section 4.3.3).

- de plus, le mapping se réfère pour l'instant à une surface faite de patches bilinéaires, ce qui est plutôt limitatif. Il faudrait adapter la paramétrisation des textures utilisée en 2D. Nous résolvons ce point au chapitre suivant en 4.2.
- le contrôle de la couleur n'est pas prévu de manière précise ; le texel décrit une forme à laquelle on associe globalement une couleur. Ce point est délicat à gérer, nous l'aborderons en 4.4.
- la probabilité d'occultation intervient un peu aussi comme une densité (probabilité de présence). Peut-être faudrait-il traiter séparément la densité, l'occultation, et la transparence (cette dernière pouvant être colorée, d'ailleurs).
- et, bien sûr, pour que l'outil soit réellement utilisable et productif, il faut compléter l'implémentation par plusieurs méthodes de construction de texels, afin de pouvoir spécifier les objets avec les outils usuels (image tomographique 3D, systèmes de particules, surfaces polyédriques, fonctions de bruit, L-systèmes...). Nous traitons cet aspect au chapitre suivant, section 4.1.

3.5.2 L'incertitude en synthèse d'images

Comme on aura pu le constater, le calcul du parcours est complexe, et passe par plusieurs étapes au fur et à mesure du passage de la scène à la texture, de la texture au texel, puis au voxel (et nous allons encore ajouter un étage au chapitre suivant). Il est clair qu'une erreur de calcul en cours de route se propage, et conduit au calcul erroné d'un pixel. Les calculs que l'on fait sont eux-même un peu compliqués : intersection d'un rayon et d'un patch, extraction des coordonnées texture, calcul de valeurs et de vecteurs propres.

Non seulement ces calculs ne sont pas toujours stables, mais il faut en plus souvent savoir séparer parmi les solutions trouvées (pour un calcul d'intersection avec une grille, par exemple) celle qu'il ne faut pas choisir parce qu'elle correspond à la position actuelle. Avec l'imprécision des calculs, un point situé tout près derrière le point courant peut très bien se retrouver légèrement devant, et donc retenu comme point d'intersection suivant ! Et calculer en double plutôt qu'en float ne fait que repousser les problèmes un peu plus loin.

Cela dit obtenir *la* bonne solution n'est pas forcément indispensable. Il faut au moins garantir la cohérence, c'est à dire qu'à une même question on obtienne toujours la même réponse quelle que soit la voie par laquelle on l'aborde (par exemple, choix de la facette atteinte quand un rayon tombe juste à la limite de deux facettes[21]). Il y a en tout cas un écueil à éviter par dessus tout : le blocage du programme en un point du parcours parce que

21. Si l'on s'y prend mal, il se peut en effet que l'ordinateur décide que les deux facettes sont atteintes, ou aucune, ou l'une ou l'autre selon la façon de poser la question.

le point courant est élu comme point suivant.

Il existe finalement peu de solutions pour traiter ces problèmes récurrents en synthèse d'images :

- faire attention à l'ordre des opérations afin de regrouper les ordres de grandeur identiques, normaliser les grandeurs quand c'est possible ;
- effectuer les opération géométriques toujours dans le même ordre afin de garantir au moins la cohérence (par exemple un segment commun à deux facettes doit être toujours parcouru dans le même sens) ;
- faire quelques tests déterminant la position d'une solution (par exemple par rapport à un plan) avant d'en effectuer le calcul ;
- certains préconisent le recours à l'arithmétique exacte, au moins dans les cas désespérés [BJM+93]. Le domaine est en pleine effervescence et ces méthodes semblent prometteuses, mais elles ne sont pas encore totalement adaptées aux situations rencontrées (et posent des contraintes lourdes de programmation).

Alors que faire ? Les trois premières solutions sont raisonnables mais insuffisantes, nous avons bien sûr essayé de tenir compte de leurs recommandations. Quant aux cas non réglés, nous avons eu recours à la mauvaise solution souvent employée, l'emploi d'ε à chaque fois qu'il fallait être sûr de ne pas rester bloqué en un point. Ainsi par exemple, quand il faut prolonger le rayon après sa première intersection avec la paroi d'un texel, on avance un peu le point de départ pour être certain que la procédure d'intersection trouvera bien la suivante et non pas à nouveau le point courant. De même, on s'accorde une tolérance de ε lors des calculs d'intersection, c'est à dire que l'on accepte des coordonnées barycentriques entre $-\varepsilon$ et $1+\varepsilon$, mais que l'on marque la facette comme douteuse afin d'accorder la préférence à sa voisine si celle-ci a une intersection plus 'franche' avec le rayon.

Chapitre 4

Extension du modèle

Le modèle décrit au chapitre précédent constitue le noyau de la représentation, dont on a abordé le fonctionnement et l'implémentation. Mais il reste de nombreux aspects pratiques à traiter avant d'obtenir une méthode réellement utilisable, reliée aux outils existants. Ainsi, il faut maintenant décrire :

- comment **spécifier le contenu** des texels, ou comment convertir des données existantes en texels (cf section 4.1),
- comment **mapper la texture** sur une surface, indépendamment de la forme et de la taille des faces (cf section 4.2), et tenir compte de cette déformation au rendu (cf section 4.3),
- comment **'habiller' le contenu** des texels, de la même façon que l'on texture une surface géométrique (cf section 4.4),
- comment **animer** les textures volumiques (cf section 4.5),
- comment **intégrer** l'ensemble dans une plateforme de lancer de rayons (cf section 4.2.2).

4.1 Construction de l'échantillon de texture

Il existe déjà de nombreuses représentations et outils de modelage pour spécifier la forme des objets, tant génériques que spécifiques, que les utilisateurs maîtrisent bien et souhaitent pouvoir utiliser. Une façon commode de spécifier le contenu du volume de référence consiste donc à convertir les représentations existantes en volumes (même si des éditeurs dédiés[1] peuvent ajouter quelques 'plus'). Des méthodes de voxelisation[2] adaptées à certaines re-

1. Le premier réflexe face à une nouvelle représentation est souvent de lui associer un éditeur, interactif ou déclaratif. Nous avons ainsi tout d'abord implémenté un petit langage de description de formes, permettant soit de composer des objets à partir de primitives, soit de charger une image scanner 3D.

2. La scan-conversion 3D consiste à balayer la scène à la manière d'un scan-line (et peut comme lui se faire de façon incrémentale), en remplissant les voxels situés entre le point d'entrée et le point de sortie de la ligne dans un objet [TL88, Kau87].

présentations existent déjà, mais ces méthodes parcourent systématiquement la totalité du volume, et sont assez coûteuses. De plus, nous n'avons pas seulement besoin de connaître le quota d'occupation des voxels, mais également la réflectance locale, ce qui nécessite de toutes façons d'étendre ces méthodes.

Les représentations existantes

La CSG [GN71, FvDFH90], les maillages polyédriques, les fonctions implicites [WMW86], les systèmes de particules [Ree83, RB85], les L-systèmes et apparentées [Smi84, PLH88, dREF+88] sont des techniques **à base de primitives** : ces constructions spécifient une forme en combinant des formes simples (les primitives étant respectivement un solide, une facette, un élément de squelette, un segment de trajectoire, un symbole terminal). La simplicité de ces primitives autorise leur conversion de manière plus efficace et précise que par scan-conversion brutale : ce genre de primitives peut être défini par une fonction de distance, ou plus commodément par la distance à sa surface, que l'on considère comme négative si le point à tester est à l'intérieur de la forme. La surface correspond ainsi à l'isovaleur zéro d'une fonction, et le volume intérieur correspond aux valeurs négatives. Les normales sont données par le gradient de la fonction de distance. Nous traitons de la conversion de ce type de modèles en section 4.1.1.

Au contraire, les données scanner et les hypertextures [PH89] sont des représentations intrinsèquement **volumiques**, qui ont une valeur en tout point de l'espace. L'orientation locale et la densité ne peuvent donc être déterminées qu'en balayant tout le volume. Nous détaillons ces modèles en section 4.1.2.

Modalités de la construction

Notons qu'en utilisant un stockage volumique, construire une forme s'apparente à peindre un dessin en 3D : il faut remplir les cases du volume en fonction de leur occupation, ce qui correspond à une opacité, et y encoder la distribution de normales des objets pour chaque portion d'espace, qui peuvent se voir comme une 'cristallisation'. Ce procédé de 'tracé' peut être modifié de manière à permettre toutes sortes d'opérations lors de la spécification d'une forme : la fonction de tracé qui d'ordinaire se contente de fixer une valeur dans un voxel peut être modifiée afin de fournir une variété de **modes de tracé**, qui sont autant de façons de combiner les formes (i.e. la forme tracée et celle qui est déjà dans le volume). Quelques modes de tracé commodes sont l'ajout d'une valeur au contenu précédent du volume (ce qu'on peut considérer comme l'union de formes), la soustraction, la multiplication, le positionnement de la NDF mais pas de la probabilité d'occultation (ce qu'on peut comparer au bump-mapping d'une surface), ou vice-versa, le seuillage de la valeur précédente en fonction de la nouvelle, etc. On utilise notamment ce mécanisme (décrit section 4.1.3) pour implémenter les opérateurs CSG, comme nous le verrons plus loin.

Comme on l'a vu au chapitre précédent, la construction correspond à la détermination de l'échelle la plus fine de l'octree, après quoi il reste à effectuer une passe de propagation vers les étages supérieurs afin de précalculer les données filtrées à toutes les résolutions, puis une passe de compression pour supprimer les données redondantes (une zone quasi-constante ne nécessitant d'être représentée qu'à une échelle grossière).

4.1.1 Modèles à base de primitives

Les structures de données multiéchelles comme les octrees sont bien adaptées à la construction récursive [Sam90b] : on peut désigner efficacement le volume occupé par un objet en parcourant récursivement l'octree, dans l'esprit de l'algorithme de Warnock en 2D [FvDFH90], à partir du moment où l'on est capable de déterminer en toute région de l'espace si l'on est dedans, dehors, ou sur la frontière de l'objet. Le nombre de voxels visités est alors de l'ordre de la surface de l'objet, et non de son volume. Cela entraîne d'ailleurs que les octrees sont souvent assez creux. La fonction qui donne la distance entre un point à tester et la surface de la forme à construire, distance considérée par commodité comme négative à l'intérieur de la forme, fournit un moyen de déterminer le statut d'un voxel vis à vis de l'objet (voir figure 4.1) :

- si la distance depuis le centre du voxel courant (initialement la racine de l'octree) est plus grande que le rayon de la sphère exinscrite au voxel, on est sûr que ce voxel est hors de la forme.
- si cette distance est négative, sa valeur absolue étant plus grande que le rayon, on est sûr que ce voxel est dans l'objet.
- sinon le voxel est probablement sur la frontière, donc ambigu. Il faut scinder le voxel en huit fils et appliquer le critère à chacun d'eux. Quand on atteint la taille minimale (correspondant à la résolution choisie par l'utilisateur), on évalue et on stocke l'information géométrique[3] correspondant à cette zone de l'espace.

Il est facile d'obtenir la fonction de distance d'une primitive simple comme une sphère ou un cylindre, et on obtient directement les normales en calculant le gradient de cette fonction. Remarquons que l'on n'est pas obligé d'utiliser la distance euclidienne : n'importe quelle fonction nulle à la surface de l'objet et ne croissant pas plus vite que la distance euclidienne peut

3. Comme on l'a vu au chapitre précédent, l'information géométrique comprend une probabilité d'occultation (ayant le même rôle que le plan alpha pour le compositing, cf [FvDFH90]), et six paramètres caractérisant l'ellipsoïde qui approxime la NDF locale (l'ellipsoïde étant le support de ces normales, et non la forme de la distribution elle-même).

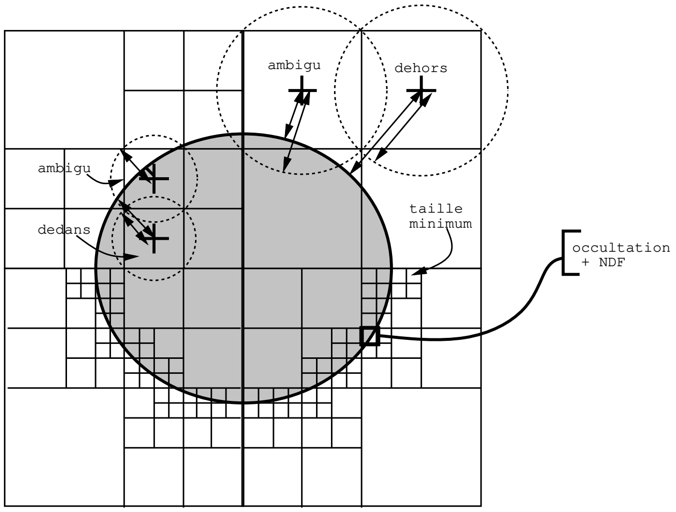


FIG. 4.1 - *Construction volumique récursive d'une forme (représentée en 2D).*

convenir (cette restriction empêche un voxel ambigu d'être faussement déclaré extérieur).

Nous avons implémenté un certain nombre de primitives, dont l'utilisateur dispose pour décrire la 'scène' qu'il veut mettre dans le volume au moyen d'un script déclaratif, et dont on va voir qu'elles permettent aussi de convertir les représentations usuelles : ligne à section cylindrique ou rectangulaire, plan, demi-espace, sphère, ellipsoïde, parallélépipède, cylindre, cône tronqué, et quelques primitives spécifiques comme un segment de brin d'herbe à section en 'V', et quelques autres.

Par exemple, une sphère $\{C, r\}$ est obtenue avec la fonction de distance $d(M, sphere) = |\overrightarrow{MC}| - r$, dont découle le critère $\frac{1}{2}(1 - \frac{d}{r_{voxel}})$ qui est supérieur à 1 si le voxel est hors de la sphère, négatif s'il est à l'intérieur, et entre 0 et 1 s'il est sur la frontière. La normale en tout point est simplement $\vec{MC}/|\vec{MC}|$. Pour un parallélépipède $\{C, (r_1, r_2, r_3)\}$, la fonction de distance est $d(M, rect) = max_{i=1..3}(|\vec{MC_i} - r_i|)$, et la normale $\vec{N} = (\delta_{dist=|\vec{MC_i}-r_i|})_{i=1..3}$. Les autres primitives s'obtiennent de la même façon, avec des formules légèrement plus compliquées.

Conversion des représentations usuelles

Comme on l'a dit plus haut, nombre de représentations usuelles peuvent s'exprimer à l'aide de primitives (les autres font l'objet de la section suivante).

• La première représentation découlant de nos primitives est la CSG, combinant des formes simples avec des opérations booléennes. Les opérateurs CSG comme l'union ou l'intersection sont faciles à implémenter : on combine une nouvelle forme avec la forme précedemment construite dans le volume en utilisant un mode de tracé particulier, qui ajoute la nouvelle valeur à la valeur précédente au lieu de la redéfinir (cf section 4.1.3). L'évaluation d'un arbre CSG est plus compliquée, dans la mesure où chaque sous-arbre doit être tracé dans un volume différent, ce qui peut devenir coûteux en mémoire (en effet $(A \setminus B) \cup (C \setminus D)$ doit d'abord construire indépendamment $(A \setminus B)$ et $(C \setminus D)$, même s'ils occupent partiellement le même espace). Un opérateur CSG aura juste à opérer soit entre un volume déjà constitué et une primitive, soit entre deux volumes.

• Les primitives sont également utilisables pour les représentations à base de 'squelettes', comme les L-systèmes et les systèmes de particules : les segments successifs sont tracés dans le volume de référence en utilisant la primitive cylindre ou cône, voire un modèle cône-sphères [Max90, Ney90] négociant les arrondis.

• On peut représenter les maillages triangulaires selon le même principe en utilisant la primitive triangle : la distance euclidienne d'un point à un triangle est relativement facile à obtenir, et la distribution de normales est constante et égale à la normale au triangle. Cette primitive peut sembler un peu particulière en ce qu'elle n'a pas d'épaisseur : on trace la surface du maillage et non le volume délimité (car les maillages peuvent constituer des surfaces non fermées). Dans la pratique, l'algorithme de tracé récursif subdivise les voxels incluant le triangle et s'arrêtent à la résolution du volume, ce qui fait qu'il y a toujours une épaisseur résiduelle.

FIG. 4.2 - à gauche : *maillage polyédrique.* à droite : *sa forme texelisée.*

• Nous avons également implémenté les fonctions implicites dans le même esprit : on peut dériver une fonction de distance approximative à partir du potentiel, dans la mesure où les seules propriétés utiles pour cette fonction sont de valoir zéro sur la surface, et de ne pas croître plus vite que la distance euclidienne. Notre approximation de cette distance est $\frac{1}{\sqrt{pot}} - 1$, où $pot(M) = \sum \pi_i / d_i^2$, π_i et $d_i(M)$ étant respectivement le poids et la distance euclidienne à l'élément du squelette qui peut être un point, un segment ou une face triangulaire.

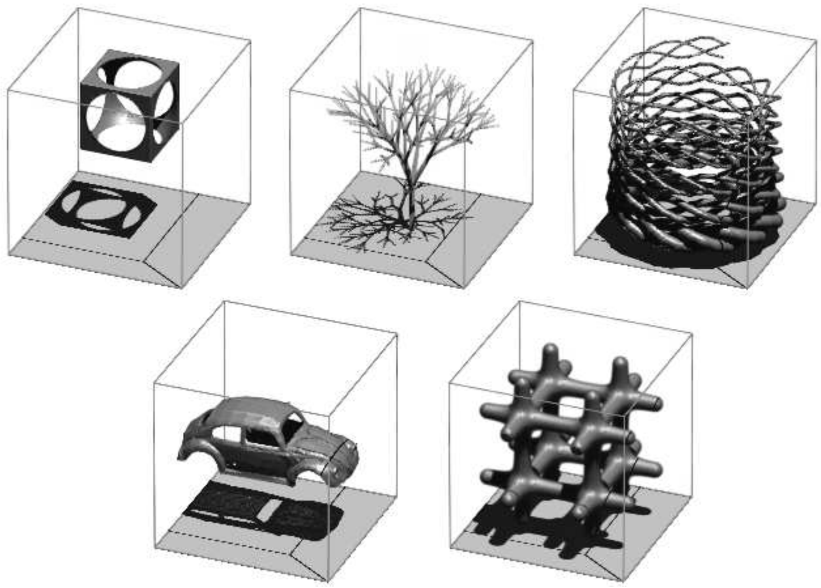


FIG. 4.3 - *De gauche à droite et de haut en bas, conversion en texel de modèles : CSG, L-système, système de particules, maillage, surface implicite.*

4.1.2 Conversion des représentations volumiques

Les données volumiques comme les images tomographiques 3D (champ de densité explicite), ou les hypertextures obtenues à partir de la fonction de bruit de Perlin [Per85] (champ procédural) constituent un type bien différent de représentation. On ne peut pas facilement dériver une fonction de distance du champ 3D, dans la mesure où les données ne représentent pas vraiment une surface. On doit donc recourir à une voxelisation systématique : on échantillonne les données à la résolution la plus fine, et on crée la hiérarchie de voxels 'au vol' , là où on trouve des zones non-vides de l'espace.

Avec un bruit de Perlin, les normales s'obtiennent en calculant analytiquement[4] le gradient de la fonction, alors que pour les images de densité comme les images scanner 3D, il n'y a guère d'autre solution que d'estimer le gradient à partir de la variation de densité dans un voisinage (pour éviter de parcourir l'octree, on ne considère qu'un voisinage 2x2). Cela donne des résultats moins bons dans le second cas, ce qui montre que la connaissance directe

4. La fonction de bruit élémentaire $b()$ est une interpolation de degré 4 entre des valeurs données sur une grille 3D. Elle est continue dérivable, et a pour pseudo-période le pas de la grille. On en dérive une fonction de 'turbulence' fractale $t(x) = \sum \frac{1}{2^i}.b(2^i.x)$, la fonction de bruit de Perlin, que l'on peut scaler et translater afin de régler la taille et la phase de la texture générée. Nous ajoutons la possibilité de multiplier cette fonction par un terme de répulsion des bords de degré 3, au choix de l'utilisateur. Les valeurs du bruit et du gradient s'obtiennent donc sans difficulté.

de la NDF est plus importante à cette échelle que la distribution de densité (i.e. il faudrait une résolution bien plus fine dans le second cas pour obtenir la qualité du premier cas).

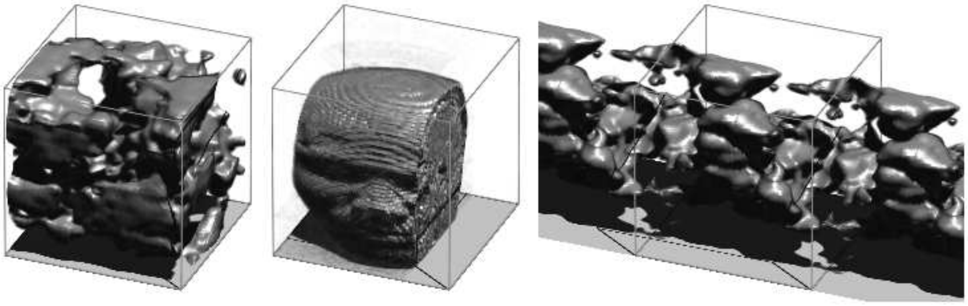


FIG. 4.4 - *Conversion en texel d'une hypertexture (*A gauche*) et d'une image tomographique (*au centre*).* A droite : *motif cyclique d'hypertexture.*

4.1.3 Modes de tracé

Nous avons implémenté une douzaine de *modes de tracé* agissant lors de la construction des objets, afin de prendre en compte diverses combinaisons intéressantes entre la forme déjà présente dans le volume et la forme tracée. Ceci se fait au niveau des voxels, en opérant sur la densité et sur la distribution de normales, entre les valeurs déjà présentes dans le voxel et celles soumises à la fonction de tracé (i.e. provenant de la construction d'une nouvelle forme). Pour ce qui concerne la densité, il est pertinent d'imposer, d'additionner, de soustraire, de multiplier, de prendre le min ou le max des valeurs. Quant à la distribution de normales (NDF), il est intéressant soit de l'imposer, soit de l'additionner au prorata des densités, soit encore de l'additionner au prorata des densités permutées comme nous allons le voir.
Voici les modes de tracé que nous avons implémentés :

- **SET** : impose la nouvelle densité et la nouvelle NDF. C'est le mode normal de tracé, sans gestion particulière des intersections avec la forme précédente.
- **DRAW** : impose la densité. Permet de dessiner avec un trait isotrope (brouillard), ou de faire varier après coup la densité d'une forme par exemple pour simuler de la transparence.
- **ANISO** : impose la NDF. Permet de faire varier les reflets indépendamment de la forme locale, afin de simuler l'anisotropie due aux microstructures (ex : aluminium brossé). Comme pour le bump-mapping en 2D [Bli78], des variations de surface sont ainsi simulées en modifiant la normale.
- **SEUIL** : seuille la forme déjà stockée : remplace la densité par 0 ou 1 selon qu'elle est au dessus ou au dessous de la valeur fournie en guise de seuil.

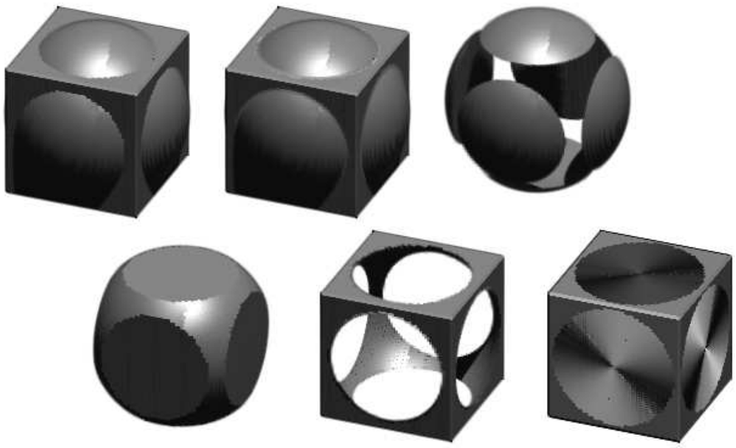


FIG. 4.5 - *Quelques modes de tracé :* en haut : *cube* $\cup$ *sphère (OR), cube + sphère (ADD ; noter le raccord plus lisse), sphère* $\setminus$ *cube (SUB),* en bas : *cube* $\cap$ *sphère (AND), cube* $\setminus$ *sphère (SUB), cube avec réflectances de 6 sphères (ANISO).*

- **COL** : il s'agit d'un mode très particulier utilisé pour la visualisation scientifique et incompatible avec le filtrage, où le poids faible de la densité code une couleur : ce mode conserve la densité originale et applique la valeur fournie en guise de couleur, permettant ainsi de peindre sur (et dans) les formes.
- **ADD, ERASE, AND, OR** : combine les densités (en prenant le min pour AND et le max pour OR), ainsi que les distributions de normales. OR correspond à l'union et AND à l'intersection de formes. Remarque pour ADD : ρ étant considéré comme une probabitité d'occultation plutôt que comme une densité au sens strict, les ρ se masquent en s'ajoutant (ρ_1 étant l'ancienne valeur et ρ_2 la nouvelle, on prend $\rho\ := \rho_1 + (1-\rho_1)\rho_2$) , et ne peuvent donc pas dépasser 1, contrairement à ce qu'il se passerait pour de vraies densités qui, elles, s'additionnent simplement.
  Attention : ERASE, et ADD avec une densité négative, ne correspondent pas à la soustraction logique. Ces modes sont utiles notamment sans NDF pour opacifier ou grignoter les formes sans toucher aux normales.
- **SUB** : soustrait les densités et impose la NDF (soustraction logique de formes).
- **MUL** : multiplie les densités (utile pour estomper).
- **MUL2** : multiplie les densités et combine les NDF au prorata des densités permutées. Ce mode est utilisé quand on veut forger une forme qui est le produit de deux champs de densité, par exemple une hypertexture et une fonction de répulsion des parois. La NDF correspondant à la distribution des gradients de ces champs, on a $\nabla(\rho_1.\rho_2) = \rho 2.\nabla\rho_1 + \rho 1.\nabla\rho 2$ soit $NDF\ := \rho 2.NDF_1 + \rho 1.NDF_2$, ρ_1 étant l'ancienne valeur et ρ_2 la nouvelle.

4.1.4 Discussion du modèle

• La voxelisation d'une forme apporte bien sûr certaines approximations : vues de près, les formes semblent faites de cubes et sont un peu floues. Pour éviter ce genre d'effet, la résolution du volume doit être adaptée au point de vue le plus proche prévu par l'utilisateur. Cela peut consommer beaucoup de mémoire, mais il faut garder à l'esprit que les texels sont faits pour représenter des petits détails dans une scène complexe, ils ne sont pas supposés gérer les gros plans. Dans ce genre de situation, il faut effectuer une transition[5] vers la représentation polyédrique [Nom95].
• Si la texture est répétée sans la moindre variation, elle aura une apparence très répétitive. Nous traitons ce problème en section 4.2.4.
• La représentation de la couleur n'est pas prise en compte par le modèle volumique, qui est une représentation purement géométrique (en fait, il n'est pas simple de filtrer de la géométrie colorée[6]). Les divers auteurs traitant des texels [KK89, Shi92, Ney95b, Nom95] associent une couleur de matière à tout un motif de texture. Nous nous accomodons de ce problème de deux façons (exposées en section 4.4) : en autorisant la fusion des texels disjoints dotés de matières différentes, ce qui est limité mais facile à implémenter en rendu volumique, et en implémentant les textures classiques (images ou procédurales) afin de les utiliser *à l'intérieur* des texels.

4.2 Le mapping

Le propre d'une texture plaquée, qu'elle soit définie par un échantillon ou par un modèle procédural[7], est d'être caractérisée par un motif défini a priori, lequel est ensuite mappé

5. c'est bien l'esprit de la commutation de modèles dans la problématique des niveaux de détails [Kaj85, BM93].

6. On ne peut pas se contenter de stocker une couleur dans les voxels en plus de la densité et de la fonction de réflectance, car la couleur apparente est intrinsèquement liée aux effets de blocage : penser aux tissus dont la couleur dépend de l'angle sous lequel on les regarde[Tai93], parce que les fils d'une couleur sont amenés à cacher les fils d'une autre couleur. Il faudrait donc non seulement stocker une fonction de réflectance pour chacunes des trois fréquences R,G et B, mais aussi tenir compte lors du filtrage des interactions entre ces trois fonctions. Il est clair que notre approche purement géométrique du filtrage, qui ne respecte pas les effets de blocage en dessous de la taille du pixel (ce qui tend à sous-évaluer l'ombrage dû aux enfractuosités), ne pourra a fortiori pas traiter correctement les effets dus aux masquages entre couleurs.

7. A l'exception des textures 3D de Perlin, qui dans leur usage habituel sont définies en tout point de l'espace et échantillonnées sur le lieu correspondant à la surface de l'objet, comme si celui-ci avait été taillé dans la masse du matériau représenté par la texture. Cela dit, on peut considérer que les paramètres réglant la position et l'échelle de la texture représentent un 'mapping' entre l'espace propre de la texture et l'espace de la scène (d'autant plus que l'on pourrait très bien effectuer une déformation plus complexe que cette transformation affine).

sur une surface (cf Annexe C.2.3). La fonction de plongement de l'espace de la texture vers l'espace 3D de la scène est loin d'être triviale, comme s'en sera rendu compte toute personne qui aura un jour tenté de tapisser une pièce un peu compliquée (et encore, l'architecture occidentale emploie essentiellement des surfaces développables). Bien que ce problème suscite sommes toutes assez peu de recherches, il est ardu, pas très bien posé, et loin d'être résolu [BVI91, MYV93]. Il se trouve hélas que les utilisateurs vivent avec cette carence, dont ils s'accommodent avec ingéniosité. Mais ne pouvant pas nous attaquer à tout, nous supposerons qu'une telle fonction de plongement peut être fournie, sous la forme de coordonnées textures attachées aux points de la surface à habiller, comme on le fait en 2D. Il faut cependant décrire ce qu'il advient du degré de liberté supplémentaire, l'épaisseur de la texture volumique, tant au niveau de la spécification mapping, qui s'étend assez facilement comme on le voit en section 4.2.1, que du rendu qui lui diffère fortement de celui des textures 2D[8], traité en section 4.2.2. Il faut également décrire comment se prolonge au 3D l'analogie avec le mip-mapping [Wil83] pour le filtrage de textures (il n'y a pas de gros problème, comme nous le verrons en 4.2.2).

4.2.1 Du mapping dans le mapping

Le mapping utilisé par Kajiya est assez rudimentaire ; il nécessite que la surface sous-jacente soit constituée de patches bilinéaires, sur lesquels reposent exactement les texels (la base du texel est confondue avec le patch). Les texels sont déformés pour coller les uns aux autres, afin de former une couche volumique continue. Cela contraint fortement le maillage de la surface, puisque l'orientation et la taille des patches dictent celles des texels. Cependant, le maillage est également contraint en tant que discrétisation d'une surface continue (i.e. il doit être suffisamment raffiné), et ces deux contraintes ne sont pas forcément compatibles. De plus, il n'est pas toujours facile de mailler une surface avec des quadrangles (surtout si l'orientation des faces est contrainte par l'effet visuel recherché pour la texture).

8. En effet, dans le cas des textures classiques, le rayon et la texture sont séparément plongés depuis leur espace respectif 1D et 2D dans l'espace 3D de la scène, et ne coïncident donc qu'en un point. Le calcul du rendu se divise alors en deux parties bien séparées, la phase 'géométrique' qui détermine le point de la surface vu par le rayon, et la détermination de la couleur apparente de ce point, laquelle dépend de la couleur intrinsèque définie par la texture, de la réflectance locale, et de la lumière incidente (i.e. la triade matière - illumination locale - illumination globale).

Par contre les textures volumiques sont 3D, leur coïncidence avec le rayon se fait donc sur tout un segment de rayon à traver l'épaisseur de la texture. De ce fait, la partie 'rendu de texture' comporte maintenant elle-même une phase 'géométrique' (parcours sur le segment de rayon) et une phase 'photométrique' (couleur apparente d'un voxel).

Ce type de mapping est donc incomplet et peu satisfaisant, aussi nous sommes nous efforcés de prolonger les approches qui ont cours en matière de texture 2D.

- Il est souhaitable de pouvoir utiliser des maillages quelconques[9], produits par des outils génériques ou issus de données pré-existantes.
- Il vaut mieux laisser les seuls critères géométriques décider de la subdivision du maillage, notamment pour des raisons d'économie.
- Le mapping de la texture est entièrement défini par la donnée de coordonnées texture au niveau des sommets du maillage.

Notre représentation

Pour définir le mapping, on se donne donc en chaque sommet de la surface :

- un vecteur $\vec{u} = (u, v, w)$ indiquant les coordonnées texture du sommet,
- un vecteur $\vec{H}$ appelé *hauteur*, qui contrôle l'orientation verticale de la texture (permettant ainsi de 'coiffer' les texels),
- un scalaire dw tel que le point au sommet de $\vec{H}$ ait comme coordonnées texture $(u, v, w + dw)$. En général on a $w = 0$ et $dw = 1$.[10]

Avec ce formalisme, le mapping type Kajiya correspond à une surface faite de patches bilinéaires réguliers, dont les sommets ont pour u et v des valeurs entières successives d'une face à l'autre : les limites des texels coïncident avec les limites des faces.

Puisque dans notre cas le texel ne correspond plus directement à une face, il nous faut introduire une nouvelle entité : on appelle *boîte* le volume au dessus de la face, délimité par les hauteurs issues des quatres sommets. Dans le cas de Kajiya, boîte et texel sont confondus.

On manipule alors trois espaces, l'absolu $\mathcal{S}$ dans lequel s'expriment les coordonnées géométriques notées (x, y, z), l'espace $\mathcal{F}$ lié à la face (et à sa boîte) correspondant aux coordonnées barycentriques, et l'espace $\mathcal{T}$ attaché à la texture dont les coordonnées sont notées (u, v, w) (voir figure 4.6). Si l'on considère la texture volumique comme une peau épaisse mappée sur une surface, les texels sont mappés *dans* cette peau.

$\mathcal{F}$ sert de base à la paramétrisation de la boîte, on y définit les coordonnées barycentriques $\vec{U} = (U, V, W)$ (U et V correspondant à celles de la face et W à l'épaisseur). Les bords de la face correspondent à l'isovaleur 0 ou 1 de U ou V, le sol (i.e. la face) correspond à l'isovaleur $W = 1$ tandis que la surface décalée (portée par les extrémités des hauteurs) est associée à l'isovaleur $W = 0$.[11]

9. Dans notre implémentation nous utilisons des maillages triangulaires ou quadrangulaires (les triangles étant gérés comme des quadrangles dégénérés), ce qui correspond pratiquement au cas général puisque les polygones plus complexes peuvent se décomposer en ceux-ci.

10. $dw > 1$ signifie que le motif revient plusieur fois dans l'épaisseur de la peau volumique, ce qui est moins naturel que sa répétition sur la largeur de la face.

11. Cette curieuse notation est héritée de nos premières implémentations pour lesquelles la peau volumique se situait *sous* la surface. Elle peut aujourd'hui arbitrairement se situer au dessus ou au dessous, selon l'orientation des hauteurs.

Le passage de $\mathcal{F}$ à $\mathcal{T}$ et de $\mathcal{F}$ à $\mathcal{S}$ se fait par interpolation trilinéaire, puisque les coordonnées spatiales et texture sont obtenues en tout point à partir des valeurs aux 8 sommets de la boîte (les 4 sommets de la face plus les 4 extrémités des hauteurs).

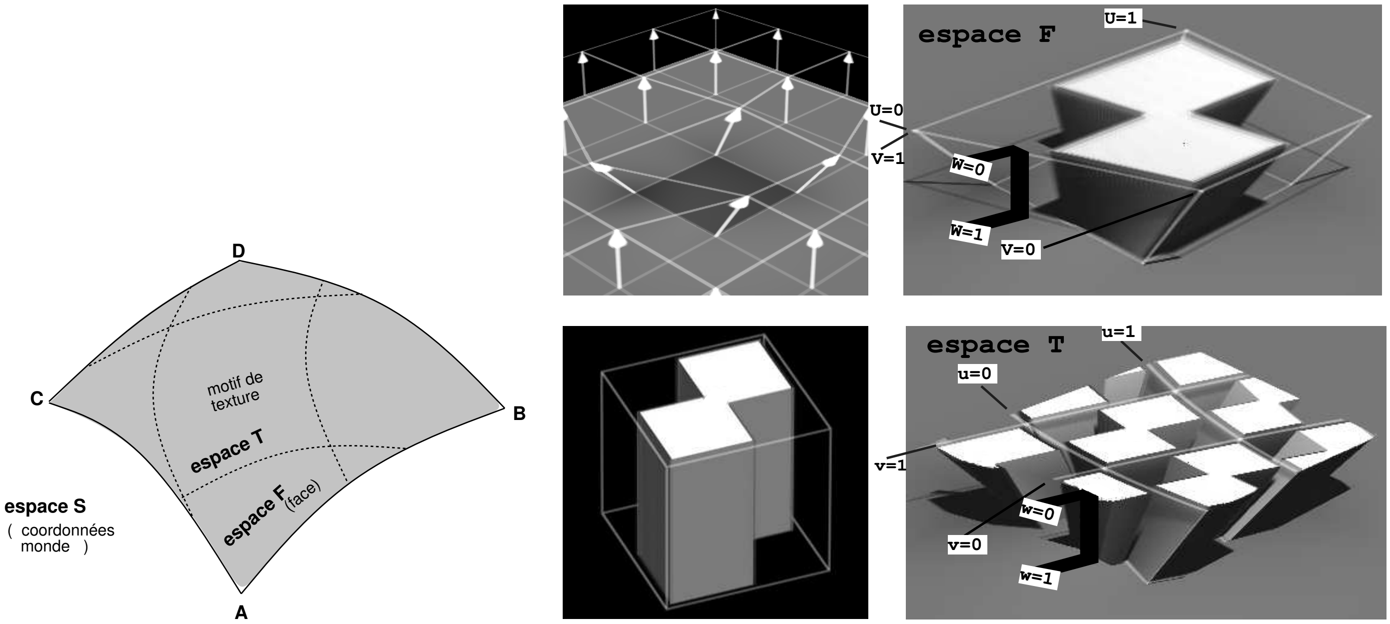


FIG. 4.6 - *Systèmes de coordonnées. $\mathcal{F}$ est la paramétrisation canonique de la face (i.e. barycentrique). Les paramètrisations $\mathcal{S}$ et $\mathcal{T}$ sont interpolées à partir de $\mathcal{F}$ à l'aide des valeurs aux sommets.* Croquis *: illustration pour une texture plane.* En haut à gauche *: les boîtes correspondant aux faces.* En bas à gauche *: volume de référence.* En haut à droite *: mapping type Kajiya : un texel correspond exactement à une face (pour la clarté, on ne mappe que la boîte centrale).* En bas à droite *: mapping quelconque défini par les (u, v, w) aux 8 sommets (on a isolé ici la boîte centrale).*

Un pointeur de matière est également associé à chaque face, lequelle indique le numéro du volume de référence à utiliser à cet endroit, et la description classique de matériau de type Phong (couleurs ambiante, diffuse et spéculaire, rugosité). En fait la spécification de la matière est un peu plus compliquée dans notre implémentation, dans la mesure où nous autorisons la fusion de deux texels dans un même volume, la superposition de couches de texels, et certaines opérations comme le scaling, la translation et les permutations d'orientation (nous en parlerons en section 4.2.4). Plus encore, les paramètres de Phong peuvent être eux-mêmes construits à partir de fonctions comme le mapping d'image ou les bruits 3D de Perlin, dont les paramètres peuvent eux-mêmes être des fonctions... Cela se traduit par l'ajout de quelques champs et par le chaînage des divers descripteurs de matière associés à une même face (cf 4.4).

4.2.2 Parcours de la texture volumique

On peut remarquer que contrairement aux textures 2D, où le calcul de la couleur locale de la texture est indépendant du parcours géométrique de la scène, on doit dans le cas des textures volumiques traverser le volume. Le rendu 'local' s'apparente ainsi davantage à du rendu volumique qu'à du calcul de texture, ce qui est somme toute normal puisque les textures 3D visent à imiter la géométrie 3D. (On requalifie alors de 'local' ce qu'il se passe au niveau d'un voxel.)

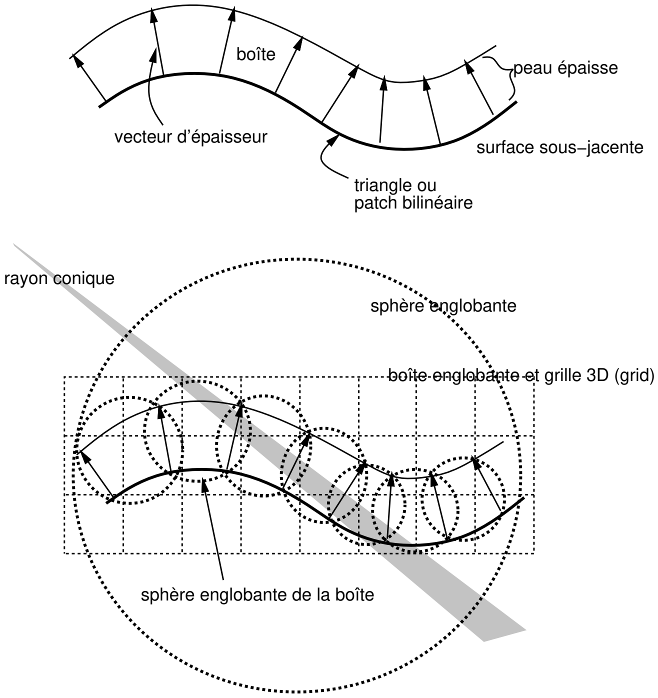


FIG. 4.7 - *Grille et boîtes englobantes*

Dans l'espace de la scène $\mathcal{S}$

On précalcule une sphère englobante et une grille[12] autour de la surface (y compris la sur-épaisseur due à la couche volumique), et une sphère englobante pour chaque boîte associée à sa face (voir figure 4.7). De manière assez classique en ray-tracing, cela permet de gagner en efficacité dans la recherche de la première face intersectée par le rayon. Cela est d'autant plus vital que dans notre cas quand une face doit vraiment être testée, il faut calculer l'intersection avec les six faces bilinéaires des boîtes. Cependant les texels sont collés

12. Une grille 3D (ou *grid*) correspond à une 'voxelisation' de l'espace à l'intérieur de la boîte englobante d'un objet, les voxels contenant la liste des facettes présentes dans leur espace. Ceci accélère fortement le lancer de rayons, dans la mesure où seules les facettes contenues dans les voxels qui sont sur la trajectoire du rayon doivent être testées. Mieux encore, le rayon peut parcourir pas à pas cet espace structuré, comme on le fait pour le tracé de droite en 2D.

les uns aux autres, de telle sorte que dans la plupart des cas seules les deux facettes sous et sur la peau volumique ont besoin d'être testées lors de la recherche de l'intersection avec le rayon.

Une fois que le rayon est entré dans la peau volumique, on quitte le ray-tracing classique. La traversée de la peau exploite la cohérence du voisinage des faces, de telle sorte qu'il n'y a qu'à calculer le point par où le rayon quitte une boîte pour savoir quelle sera la boîte traversée suivante et les coordonnées du point d'entrée dans celle-ci. Cette partie de la traversée est décrite dans le chapitre précédent. On notera qu'en dépit de la déformation trilinéaire des texels, nous n'avons à calculer que des intersections entre le rayon et des patches bilinéaires, ce qui donne un simple système $Q1$ 2×2 à résoudre[13].

Dans l'espace de la peau $\mathcal{F}$

A cette étape on commute dans l'espace $\mathcal{F}$ associé aux boîtes (c'est ce qui est fait quand on cherche les coordonnées barycentriques du point d'entrée et du point de sortie). Dans le cas de Kajiya $\mathcal{T}$ et $\mathcal{F}$ sont confondus, mais dans le cas général il faut traverser également l'espace $\mathcal{F}$ avant d'atteindre la zone correspondant à un seul texel (voir figure 4.8). Il faut noter que dans l'espace $\mathcal{F}$ les rayons ne se propagent plus en ligne droite, mais selon la transformée trilinéaire inverse d'une droite. Dans la mesure où les déformations de la texture ne sont pas très vives, on peut soit utiliser un schéma itératif simple donnant les intersections du rayon courbe avec la grille (u, v, w) liée à l'espace $\mathcal{T}$ (cf section 4.3), afin de calculer les points d'entrée corrects dans les texels, soit considérer carrément que le rayon est linéaire (c'est l'hypothèse que fait Kajiya sans le dire), ce qui est en général suffisant. Le trajet de boîte en boîte, lui, est juste de toutes façons, dans la mesure où il est déterminé dans l'espace $\mathcal{S}$ où les rayons sont rectilignes.

13. Sauf si la caméra est *dans* la boîte, auquel cas il faut résoudre un système $Q1$ 3×3 pour convertir le point de départ du rayon en coordonnées boîte, ce qui conduit à des équations de degrés 3. C'est faisable, mais nous ne l'avons pas implémenté dans la mesure où les texels ne sont pas fait pour être vu d'aussi près.
Un système $Q1$ est de degré un en chacune des variables, i.e. le terme de plus haut degré est $U.V.W$.

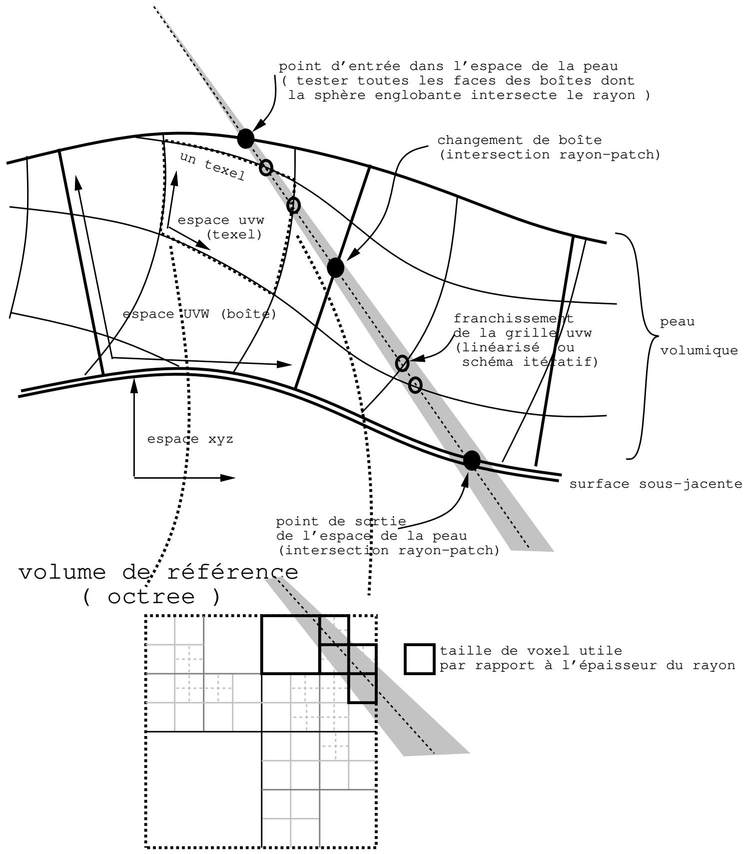


FIG. 4.8 - *Parcours du rayon*

Dans l'espace des texels $\mathcal{T}$

Entre deux tranches de la grille (u, v, w) nous sommes dans la région d'un texel, on poursuit donc le parcours en commutant dans le volume de référence (i.e. l'espace $\mathcal{T}$) que l'on traverse linéairement (on a vu au chapitre précédent que cela se faisait efficacement par dichotomie, il serait assez coûteux et peu utile de considérer des rayons courbes à ce niveau). Cette partie du ray-tracing ressemble alors beaucoup à un rendu volumique classique, à deux points près : la structure de données multi-échelle, et la présence de fonctions de réflectance dans les voxels. Le multi-échelle intervient avec le fait que notre rendu adaptatif s'apparente à une sorte de cône-tracing : connaissant l'épaisseur du rayon à cet endroit, on estime la taille de voxel qui correspond le mieux à cette ouverture[14], ce qui est exactement du mip-mapping [Wil83]. Tout comme pour le mip-mapping que nous généralisons ici au 3D, ce

14. Plus exactement on trouve un encadrement de la taille idéale, on devra donc calculer la couleur et l'opacité à ces deux échelles et interpoler le résultat. La traversée récursive de l'octree permet de faire ce calcul en une seule passe.

filtrage est une approximation dans la mesure où la 'compression apparente' de la texture varie selon les directions [15] (la compression fait intervenir la distance de la caméra, sa focale, et le jacobien de la déformation au point d'entrée que l'on suppose représentatif de tout le segment de texel traversé par ce rayon). On choisira donc l'ouverture correspondant à la plus petite dimension de la compression si l'on préfère une image nette avec un peu d'aliasing, ou à la plus grande dimension si l'on préfère un léger flou.

On a déjà décrit ce rendu volumique dans le chapitre précédent: on fait une accumulation de l'avant vers l'arrière le long du rayon, cumulant l'illumination et multipliant la transparence, à la résolution évaluée d'après l'ouverture du rayon. L'originalité tient dans le calcul de l'illumination locale selon le modèle de Phong, qui prend en compte la distribution locale des normales approximée par un ellipsoïde (l'ellipsoïde est également utilisé pour moduler l'occultation moyenne stockée dans le voxel).

Rayons secondaires

Des rayons d'ombrage sont lancés vers les sources si la densité locale est significative (ce qui nécessite hélas d'exprimer son point de départ en coordonnées boîte, impliquant une transformation trilinéaire inverse [16]). Enfin, un rayon est lancé à travers la scène afin de prolonger le premier si par hasard il est parvenu à traverser toute la couche volumique (il faut également prendre en compte le matériau de la surface sous-jacente elle-même, si elle n'est pas transparente).

4.2.3 Rendu de la texture volumique

Le mapping des texels dans la peau volumique recouvrant une surface revient à dupliquer et déformer *formellement* le volume de référence. La copie est formelle en ce sens qu'on gère la déformation lors du rendu, en ramenant les rayons dans l'espace de la texture ; il n'y a donc pas de transformation de données à effectuer. Le parcours que l'on a décrit à la section précédente effectue précisément cette déformation formelle de la forme géométrique inclue dans la texture.

Mais en fait, cette description de la déformation géométrique n'est pas suffisante. En effet, il faut également déterminer ce qu'il advient de la photométrie : celle-ci est pilotée par les normales à l'objet voxelisé, or il se trouve que les normales d'un objet déformé *ne sont*

15. On pourrait envisager l'utilisation de SAT [Cro84] si notre technique d'addition des réflectances n'était pas approximative, auquel cas on pourrait évaluer à partir d'un volume de primitives l'intégrale du volume sur un parallélépipède plus adapté à la compression apparente.

16. Grâce à notre définition particulière du mapping, où u et v sont constants le long des hauteurs et dw constant, ce calcul se ramène heureusement à une transformation bilinéaire inverse.

pas les normales initiales transformées par la déformation. Ainsi, une tôle ondulée étirée dans le sens de ses oscillations ne reflétera plus la lumière de la même manière (tant à cause de l'illumination locale que de l'ombrage), et peut à la limite devenir pratiquement plane[17].

Ces remarques sont valables pour toute déformation, et dépassent donc le cadre des texels. Nous traitons dans le détail la façon de tenir compte de cet effet sur l'illumination locale en section 4.3. Nous verrons que cela nécessite de connaître le jacobien de la déformation en tout point. Il n'est bien sûr pas question d'évaluer celui-ci continuellement, on confie donc au traitement du parcours le soin d'évaluer le jacobien de temps en temps, à chaque fois que le rayon pénètre dans une nouvelle boîte (bien que ceci puisse devenir approximatif dans le cas des boîtes subissant une déformation importante).

4.2.4 Manipulations continues, manipulations discrètes

Un simple mapping engendre une texture à l'aspect très régulier, ce qui n'est pas toujours souhaitable. On peut améliorer cet aspect en perturbant les divers paramètres.

Dans le cas où la texture représente un continuum sans frontière, le mapping doit être continu faute de quoi des lignes de brisure apparaîtraient. En outre, cela contraint le motif à être cyclique (topologie de tore), c'est à dire que ce qui déborde d'un côté apparaît de l'autre. L'aspect répétitif du mapping ne peut alors être estompé qu'en jitterant :

- les coordonnées texture,
- l'épaisseur des texels,
- les vecteurs qui définissent l'orientation de la peau volumique.

Un bruit de Perlin [Per85] convient très bien à cet effet.

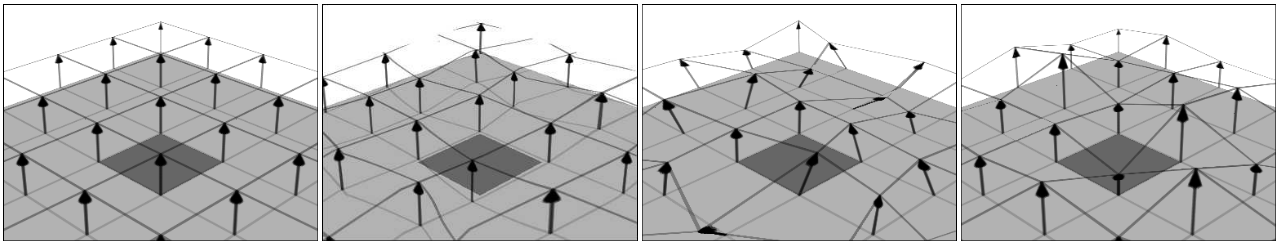


FIG. 4.9 - *Perturbations continues : mapping non perturbé ; perturbation des coordonnées texture, de l'orientation des vecteurs, de leur amplitude.*

17. Il faut cependant distinguer en matière de déformation celles qui simulent réellement la déformation d'un objet physique, comme ici, et celles qui servent à spécifier une forme, laquelle 'acquiert' une matière une fois ce processus terminé.

Mais dans le cas où la texture représente un motif isolé, la contrainte de continuité tombe (à condition toutefois que l'on puisse constituer le motif sans traverser le bord d'un texel). On dispose alors de nouvelles manipulations possibles pour chaque texel :

- changement du volume de référence et de la matière associée,
- scaling du contenu,
- translation du contenu,
- symétries et rotations de $\pi/2$ (i.e. toute permutation des axes).

Ces opérations sont faites dans l'espace $\mathcal{F}$ des boîtes, les perturbations discrètes étant spécifiées en même temps que la matière, associée aux faces.

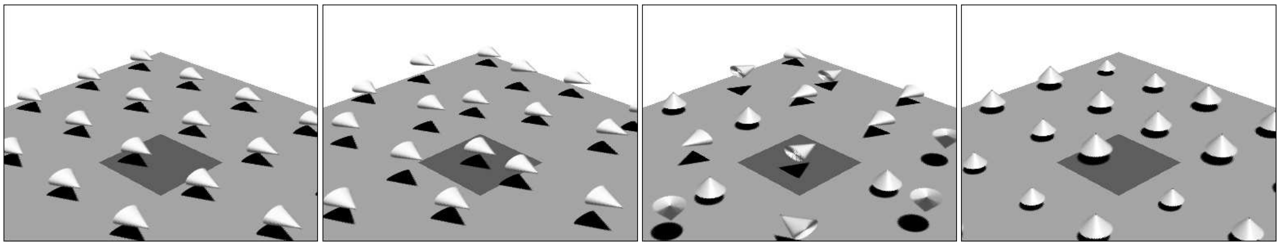


FIG. 4.10 - *Perturbations discrètes : mapping non perturbé ; perturbation par translation, rotation, scaling.*

4.2.5 Résultats

Nous avons appliqué notre méthode pour générer divers paysages, les scènes naturelles étant généralement riches en détails. Ces images ont été calculées sur une SGI Indy dotée d'un processeur R4400 à 200Mhz. Le rendu est comme toujours effectué avec un seul rayon par pixel, et demande en moyenne 20 minutes pour des images à résolution vidéo (768×576).

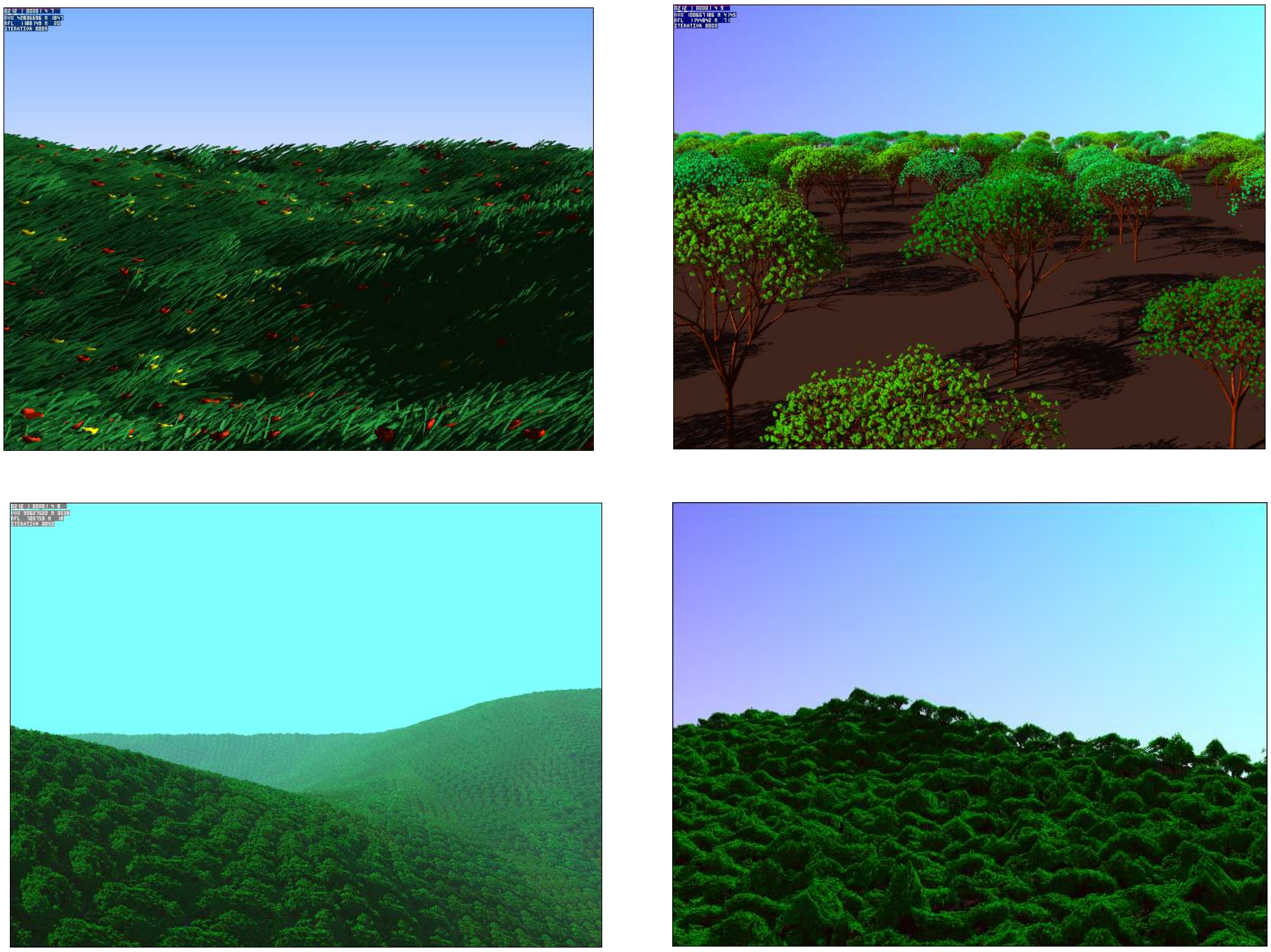


FIG. 4.11 - *Illustration du modèle étendu, avec des scènes naturelles.*

La première scène est une prairie, couvrant une colline composée de 1400 patches bilinéaires. Le mapping est perturbé en altérant les vecteurs hauteur. Chaque texel contient 16 brins d'herbe et parfois une fleur. Au total, 22000 brins d'herbe et 700 fleurs sont présents dans la scène. Les brins sont générés à l'aide d'une trajectoire parabolique, comme on le fait avec les systèmes de particules, et ont une section en forme de 'V'. Le texel est à résolution 128^3.

La seconde scène représente 512 arbres dispersés sur un terrain plat composé de 1024 patches bilinéaires. Les arbres sont construits par 6 itérations d'un L-système, produisant 2154 branches et 6336 feuilles. Le volume de référence contient un arbre isolé. Dans la mesure où dans l'animation que nous avons tirée de cette scène il arrive que la caméra frôle les arbres, nous avons pris une résolution d'octree de 512^3 (le volume est compressé à plus de 99.9%). On utilise un seul modèle d'arbre, que l'on modifie selon les texels en changeant la taille, l'orientation, la position et la matière. L'image comme l'animation permettent de constater la transition continue entre les arbres d'arrière-plan et d'avant-plan (i.e. les arbres du fond 'expriment' toutes leurs données, sans que cela ne coûte cher ni ne génère d'aliasing, puisqu'on peut utiliser les données pré-intégrées).

Le troisième exemple est une forêt couvrant une montagne, il contient 25000 arbres mappés sur une surface de 1404 patches bilinéaires. La texture est continue, ce qui signifie que le contenu du volume de référence est cyclique (topologie de tore), et consiste en deux arbres qui dépassent du bord du texel. Les deux arbres sont générés avec le même L-système que précédemment, avec des paramètres différents. Les coordonnées texture et les hauteurs sont perturbées. Les arbres sont vus de loin la plupart du temps, mais la caméra s'approche assez près, on utilise donc une résolution de 256^3. On notera que la scène représente environ 200 millions de primitives (branches et feuilles), qui produisent des ombres détaillées quand on est assez près pour les voir, que lors des zooms la transition entre les échelles est invisible, et que la scène présente très peu d'aliasing bien qu'on n'utilise qu'un seul rayon par pixel.

La quatrième image est faite sur le même principe, avec une surface de 12210 faces triangulaires sur laquelle reposent 253 arbres. Pour l'image précédente un patch était couvert par environ 18 arbres, tandis qu'ici un arbre repose sur 48 faces. Les perturbations étant alors définies à une petite échelle, on voit qu'elles permettent de générer une texture non monotone bien plus naturelle.

4.2.6 Discussion : le problème du mapping

Nous avons expliqué plus haut comment calculer le rendu d'une texture mappée sur une surface. Cela suppose d'avoir d'abord été capable de la mapper, c'est à dire d'associer à chaque sommet les coordonnées décrivant le plongement de la texture dans la scène. Le problème paraît simple au premier abord, il est vrai que pour une surface quasiment plane comme l'est un paysage, la longitude et la latitude des sommets suffit amplement. Mais quiconque a essayé un jour de tapisser une salle un peu complexe sait que la tâche peut parfois s'avérer insoluble. Sur une sphère par exemple, on risque fort d'avoir des plis (qui pour une texture se traduiront par un resserrement latéral). Et si l'on veut carrément recouvrir de fourrure la surface géométrique d'un animal à l'aide d'une texture, ça devient complètement impossible.

De quels moyens dispose-t-on pour définir un mapping ?
- mapping intrinsèque (latitude et longitude héritées de la construction du maillage, quand celui-ci est à base de quadrangles).
- projection plane dans une direction, projection cylindrique ou sphérique.
- optimisation de maillage, à l'aide d'atlas de textures [MYV93, Mai92] (c'est une méthode qui minimise les déformations à condition de lui indiquer où elle a le droit de craquer), ou en suivant les géodésiques [BVI91]. Mais le maillage optimum n'est pas très satisfaisant pour les objets compliqués.

Faute de mieux, nous avons utilisé les deux premières méthodes, qui sont pratiquement les seules que l'on rencontre dans les logiciels du commerce, mais aucune ne donne de résultat satisfaisant pour texturer la peau d'un animal. Sans doute est-ce l'idée même d'utiliser des coordonnées texture qui ne va pas : recouvrir une surface de carreaux de taille similaire qui se recollent bien les uns les autres est sans doute un problème mal posé. Comme nous le disons en Annexe C.2.3, on peut concevoir d'autres méthodes, qui s'appuient directement sur la topologie des objets. Il est clair que dans ce cadre, les texels n'ont plus leur place...

Cependant, quand on utilise une texture non continue (composée d'objets isolés), il reste toujours la solution de poser les texels individuellement, centrés sur une distribution poissonnienne de points sur une surface (épines) ou un volume (touffe d'épines de pin).

4.3 Le rendu en espace courbe

Nous avons vu lors de la section précédente que le suivi des rayons dans l'espace des textures revenait à traiter du lancer de rayons en espace courbe. Ce problème dépasse largement celui des textures volumiques : il concerne les déformations géométriques spatiales en général, qui sont utilisées soit pour modeler les objets[SP86, Bec94, Coq90], soit pour les déformer au cours du temps [CJ91].

En effet, il existe deux façons de rendre les objets ayant subi une déformation spatiale : soit calculer explicitement la nouvelle forme de l'objet et la soumettre au lancer de rayons ordinaire, soit ramener dans l'espace de départ de l'objet les rayons traversant l'espace déformé, ce qui revient à les courber.

Il se trouve qu'en pratique on se ramène toujours à la première solution, la seconde semblant susciter peu de recherches. Pourtant, il existe de nombreux cas où la seconde serait plus avantageuse :

- reconstruire un objet déformé avec la même représentation n'est pas toujours possible : comment par exemple ré-exprimer à l'aide de fonctions implicites à base de squelette (ex: métaballs) la déformation spatiale d'un objet de ce type ? La CSG pose le même problème, et les splines ne se conservent pour certaines déformations qu'à condition d'augmenter leur degré.

- on se ramène généralement à une triangulation de l'objet que l'on déforme, ce qui à la fois appauvrit l'information et augmente la consommation de mémoire. Il est en outre difficile d'obtenir une triangulation régulière du maillage final.
- les normales sont généralement recalculées à partir de la triangulation finale, ce qui est approximatif et surtout détruit toute information qui aurait put être préalablement stockée dans les normales (bump mapping).
- le rendu réaliste, qui peut faire intervenir une fonction de réflectance locale dépendant des directions (BRDF), a besoin d'information sur le repère local : la granulosité par exemple change avec l'étirement (dans le cas où la déformation ne sert pas à spécifier un objet mais bien à simuler une déformation réelle). Cette information est elle aussi perdue, bien que l'on pourrait concevoir que les modèles de déformation traitent du devenir de ce genre d'information locale[18].
- les objets complexes ont parfois plus de composantes (facettes ou autres) que de pixels visibles (ex : les L-systèmes). Le calcul explicite de l'objet déformé peut donc s'avérer inutilement coûteux, notamment si la déformation est animée et doit donc être recalculée à chaque image.

Si l'on considère le mapping de texel comme une déformation spatiale simple proche des FFD, on peut se permettre de généraliser un peu notre propos. Nous allons donc traiter de la détermination du parcours des rayons courbes, dans le cadre relativement stable des faibles déformations qui nous concernent, et de l'incidence de la courbure sur l'illumination locale.

4.3.1 Principe du parcours

Le calcul de l'intersection de rayons non linéaires avec la géométrie est en soi relativement complexe. Par contre, lorsque les données sont spatialement structurées (ex : volume en voxels) ou peuvent le devenir (ex : grille 3D triant la géométrie), le rayon peut être suivi pas à pas comme les algorithmes de tracé de droites le font pour déterminer les pixels couverts.

Soit une déformation spatiale T qui transforme un point M_o en $M_s = T(M_o)$, les indices 'o' et 's' désignant l'espace initial dans lequel est défini l'objet, et l'espace de la scène. La transformation inverse est en général difficile à obtenir explicitement (ex : déformation tricubique inverse), mais on peut s'en passer si l'on sait définir une boîte englobante autour de l'objet dans laquelle la transformation est bijective : dans ce cas la transformée inverse du rayon est connexe et sans boucle, le rendu peut alors se faire en calculant l'intersection

18. Il me semble que ce genre de lacune résulte typiquement du cloisonnement entre sous-disciplines de la synthèse d'images, ce qui fait que les préoccupations des chercheurs en modélisation et en rendu sont très disjointes.

du rayon avec la boîte déformée dans l'espace de la scène (cf [Kaj82]), puis en ramenant ces points d'intersection dans l'espace de l'objet, afin de suivre le rayon dans cet espace. Il suffit donc de décrire comment se fait le parcours entre ces deux points.

Le jacobien J_T de la transformation peut être calculé en tout point, et le jacobien $J_{T^{-1}}$ de la transformation inverse est l'inverse de la première matrice. Un vecteur V_o au point M_o est transformé par T en $V_s = J_T(M_o).V_o$.

Résolution numérique

Cela nous permet de résoudre numériquement le problème de l'inversion de la transformation T, c'est à dire de trouver M_o connaissant M_s (respectivement nommés U et X dans le pseudo-code suivant, U s'exprimant en coordonnées barycentriques de la boîte englobante). L'approximation $T^{-1}(M + dM) \simeq T^{-1}(M) + J_T^{-1}(M)dM$ permet en itérant d'obtenir un algorithme simple pour *invT(M)*:

```
U = (.5 .5 .5)
jusqu'à convergence
   X0 = T(U)
   dX = X-X0
   si |dX|<eps alors fin
   invJ = inv(J(U))
   dU = invJ.dX
   U  = U+dU
retourne U
```

On pourrait alors construire un algorithme itératif type Newton pour trouver l'intersection du rayon courbe avec une surface, mais il faudrait l'utiliser autant de fois que l'on teste les intersections du rayon avec toutes les faces candidates, ce qui entraînerait un coût prohibitif. La structuration que l'on a exigée des données permet de suivre le rayon cellule par cellule, le problème étant alors juste de trouver le point de sortie de la cellule courante. A l'intérieur de la case on peut supposer le rayon localement linéaire, et se ramener à la méthode de rendu classique. Ceci nous ramène au problème du tracé de droite, c'est à dire à la détermination des intersections entre un rayon et une grille régulière, sauf que dans notre cas la grille et le rayon ne sont pas droits dans le même espace.

Le rayon est défini par son origine X0 et sa direction D. Il nous faut trouver son intersection avec la grille, c'est à dire avec l'une des tranches parallèles aux axes, définie par l'isovaleur $u_i = c$. Cela se fait en approximant le rayon par sa tangente pour calculer l'intersection avec la grille (u_1, u_2, u_3), dont on mesure la distance pour avancer d'autant le long du rayon rectiligne dans l'espace scène (cf figure 4.12), puis en itérant. Les déformations des

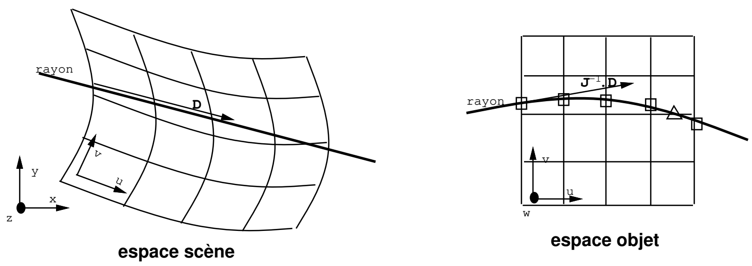


FIG. 4.12 - *Résolution numérique de l'intersection rayon courbe - grille.*

texels étant simples et peu amples, la convergence se fait toujours en quelques itérations. Cela donne l'algorithme *atteint_tranche(i,c,X0,D)*:

```
X = X0
jusqu'à convergence
   U = invT(X)
   si |U[i]-c|<eps alors fin
   invJ = inv(J(U))
   dU = invJ.D
   d = si dU[i]<>0  alors  (c-U[i])/dU[i]  sinon  1
   X = X+d.D
retourne X
```

Optimisations

- On peut améliorer la rapidité de convergence en initialisant U à (.5 .5 .5) au début de *atteint_tranche* puis en ne l'initialisant plus dans *invT*. Le point de départ est ainsi la valeur de U calculée à l'itération précédente.
- Cela peut également être fait pour l'initialisation de *atteint_tranche*, afin d'exploiter la cohérence des rayons (le U de départ obtenu pour le rayon précédent aura des chances d'être proche de la solution).
- Une autre optimisation consiste à ne pas ré-évaluer (ni ré-inverser) J dans *invT*, en utilisant la dernière valeur calculée dans *atteint_tranche*. On gagne ainsi en temps de calcul bien que la perte de précision entraîne quelques itérations supplémentaires.
- On garantit la convergence contre les cas pathologiques en clippant U afin qu'il reste dans la boîte englobante (en dehors, la transformée n'est pas forcément bijective).
- Pour économiser le nombre d'intersections de tranches, on peut utiliser une structuration hiérarchique de l'espace, afin de ne subdiviser les cellules que là où il y a de la matière, et de traverser rapidement les zones vides. L'octree utilisé pour coder les texels constitue une telle structuration hiérarchique de l'espace [Sam90b, Sam90a], qui permet de ne subdiviser récursivement que les voxels pleins.

Pour obtenir l'algorithme de tracé complet allant d'un point à un autre en traquant les intersections successives avec la grille, il faut déterminer dans chaque cellule quelle est la prochaine tranche à atteindre. On peut obtenir une estimation en approximant le rayon par sa tangente : si le point courant est U et la direction du rayon est $Du = invJ.D$, alors la distance le long de la droite entre U et les 6 tranches environnantes est donnée par $h.\frac{frac(U_i/h)}{Du_i}$ et $h.\frac{1-frac(U_i/h)}{Du_i}$ (pour une grille de pas h). La plus petite valeur positive donne une estimation du bon point d'intersection et de la tranche à considérer. (Mais si la courbure est forte, il n'y a guère d'autre solution que de calculer précisement les 6 intersections et de comparer les distances.)

4.3.2 Incidence sur l'illumination

Les reflets d'une lentille sont très différents de ceux d'une sphère, de même une surface granuleuse devient plus lisse quand on l'étire ce qui se voit dans sa façon de refléter la lumière. Les déformations transforment donc fortement l'illumination locale.

Il est difficile de spécifier ce qu'il advient d'une BRDF[19] au cours d'une déformation, dans la mesure où a priori cette fonction ne découle pas directement de la géométrie. Mais quand l'anisotropie est due à la micro-géométrie (comme pour l'aluminium brossé), la BRDF peut être représentée par une distribution de normales [PF90, CMS87, Fou92], forme sous laquelle on sait reporter les transformations géométriques.

On procède au calcul de l'illumination locale lorsque le rayon a atteint un point de l'objet, ce qui ne peut se faire directement dans l'espace objet : l'illumination dépend des normales, or la transformation de la normale N_o de l'objet au point M_o n'est pas normale à l'objet déformé : $normal_{T(obj)}(T(M_o)) \neq J_T.normal_{obj}(M_o)$. Il en découle que le modèle d'illumination de Phong qui utilise les produits scalaires entre la normale locale et les vecteurs d et L dans l'espace scène ne peut pas être évalué directement dans l'espace objet, comme on pourrait être tenté de le faire en utilisant la normale locale obtenue dans cet espace et en y ramenant les vecteurs d et L. Pour calculer le modèle de Phong avec les données disponibles dans l'espace objet, il faut soit réévaluer la normale à partir de la transformation du voisinage de M_o, soit modifier le produit scalaire de telle sorte qu'il donne le même résultat que le produit scalaire canonique dans l'espace de la scène. Le modèle de Phong pourra alors être évalué, de même pour les modèles de réflectance anisotropique (lesquels opèrent sur une distribution de normales).

19. fonction de réflectance bidirectionnelle

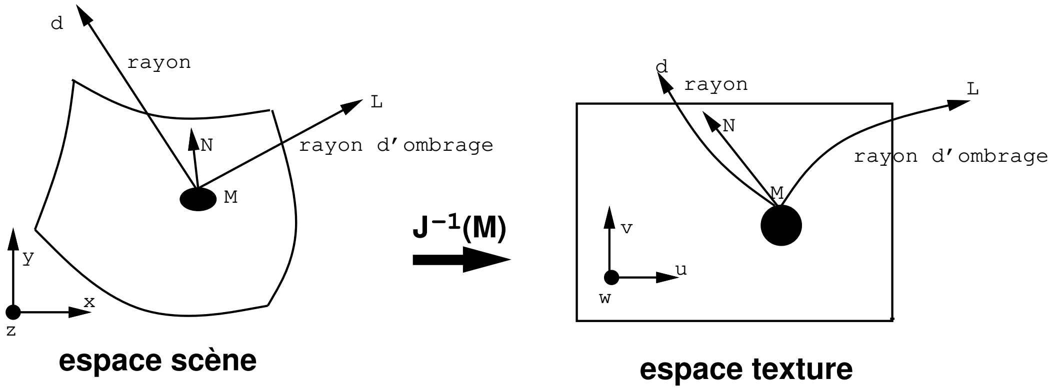


FIG. 4.13 - *Transformation du voisinage et des vecteurs.*

Calcul de l'illumination

Notons $< v_1, v_2 >$ le produit scalaire canonique de v_1 et v_2, et $\|v\|$ la norme euclidienne de v. Etant donné un vecteur V_o et la normale N_o en M_o dans l'espace objet, on a $< N_o, V_o >=< N_o, J^{-1}.J.V_o >=< J^{-1t}.N_o, V_s >$. Cela donne la normale dans l'espace de la scène[20] $N'_s = J^{-1t}.N_o/\|J^{-1t}.N_o\|$. Donc $< N'_s, V_s >=< N_o/\|J^{-1t}.N_o\|, V_o >$, N_o et N'_s étant la normale à l'objet avant et après déformation, ce qui définit le produit scalaire modifié $< N_o, V_o >_T$. Une autre façon de dire la même chose est de considérer que l'on convertit la normale locale N_o en N'_s au lieu de $J.N_o$ avant d'effectuer le produit scalaire canonique de N avec un vecteur V dans l'espace scène.

Le modèle d'illumination de Phong[21] $C_{diff} < N, L > +C_{spec}(< N, H >)^r$ au point M s'obtient maintenant à partir des coefficients de visibilité $vis_d :=< N, d >_*$ et $vis_L :=< N, L >_*$ relatifs à la direction de la lumière et de l'observateur L et d, l'étoile en indice désignant soit le produit scalaire canonique, soit le modifié $< .,. >_T$ selon l'espace où l'on prend N :

$$Illu = C_{diff}.vis_L(N).1_{(vis>0)} + C_{spec}.(\frac{vis_d(N) + vis_L(N)}{\|d + L\|)})^r.1_{(vis>0)}$$

(la fonction $1_{(cond)}$ vaut 1 si *cond* est vraie et 0 sinon).

20. Si V_o est dans le plan tangent dans l'espace objet, on a $< N_o, V_o >= 0$, alors $V_s = J.V_o$ est dans le plan tangent dans l'espace de la scène. Comme $< J^{-1t}.N_o, V_s >=< N_o, V_o >= 0$, $J^{-1t}.N_o$ est orthogonal à V_s, c'est donc la normale dans l'espace de la scène.

21. $H = \frac{d+L}{\|d+L\|}$. r est l'inverse de la rugosité. C_{diff} et C_{spec} sont les coefficients de réflection diffuse et spéculaire.

Cas de la réflectance anisotropique

L'illumination locale en un point en fonction de la distribution de normales $\mathcal{N}$ est évaluée en intégrant le modèle de Phong sur la distribution de normales, sur la sphère de Gauss :

$$Illu = \frac{\int_{\mathcal{G}auss} Phong(N,d,L).vis_d(N).1_{(vis>0)}.d\mathcal{N}(N)}{\int_{\mathcal{G}auss} vis_d(N).1_{(vis>0)}.d\mathcal{N}(N)}$$

(Une normale est associée à un élément de surface ds ayant une surface apparente $ds.vis_d$ ou 0 si la normale n'est pas orientée vers l'observateur, et a une distribution $d\mathcal{N}(N)$, tout ceci pondérant la réflectance locale de Phong. L'intégrale est normalisée de telle sorte que la surface apparente soit 1.)

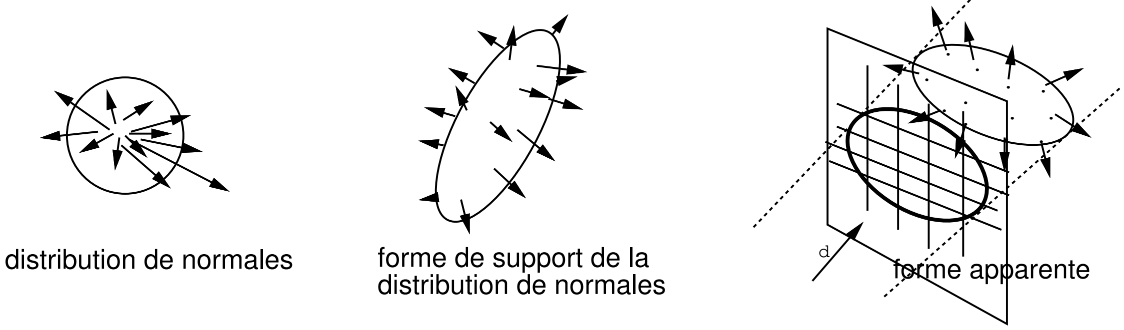


FIG. 4.14 - *Evaluation de l'illumination produite par la distribution de normales.*

On peut éventuellement considérer que la distribution de normales $\mathcal{N}$ correspond à celle d'une forme, qui peut être vue comme un 'grain' ou une 'cristallisation'. L'illumination peut alors s'écrire

$$Illu = \frac{\int_{shape} Phong(N(M),d,L).vis_d(N(M)).1_{(vis>0)}.dS}{\int_{shape} vis_d(N(M)).1_{(vis>0)}.dS}$$

Cette intégrale peut s'écrire directement sur la surface 2D apparente :

$$Illu = \frac{\int_{2Dsurf} Phong(N(M_{(x,y)}),d,L).ds}{\int_{2Dsurf} ds}$$

que l'on peut approximer numériquement par $\frac{1}{L.H}.\sum^L \sum^H Phong(N(M_{(x,y)}),d,L)$

Le schéma de calcul de l'illumination est alors :
- évaluer la surface apparente de la forme qui modélise la réflectance,
- pour chaque échantillon sur la surface,
  - calculer la normale N associée au point correspondant sur la forme,
  - évaluer vis_d et vis_L dans l'espace objet ou scène,
  - en déduire la valeur de l'illumination locale selon Phong,
  - que l'on cumule pour obtenir finalement $Illu$.

4.3.3 Cas des texels

Pour les textures volumiques, la déformation T correspond au mapping sur la surface sous-jacente, ou plus précisément au mapping des boîtes (qui sont confondues avec les texels dans le modèle de Kajiya). Comme on l'a décrit au chapitre précédent, le modèle de réflectance locale encode la distribution de normales $\mathcal{N}$ à l'aide d'un ellipsoïde, stocké dans chaque voxel du volume de référence en plus de la probabilité d'occultation. Si l'on suppose que la déformation est linéaire au voisinage du centre O_c du voxel (la déformation locale étant déterminée par le jacobien $J_T(O_c)$), la réflectance peut toujours être encodée par un ellipsoïde dans l'espace scène [22].

Pour calculer l'illumination locale, nous avons vu dans la section précédente que l'on pouvait soit opérer dans l'espace objet, à condition d'utiliser un produit scalaire modifié, soit opérer dans l'espace scène en y transformant le voisinage (i.e. la distribution de normales). C'est cette dernière solution que nous avons choisie [23], le problème est donc de convertir dans l'espace scène l'ellipsoïde défini dans l'espace objet, afin d'en extraire ses normales et l'ellipse apparente à échantillonner. En outre, dans le cas des textures volumiques ces ellipsoïdes doivent subir une normalisation supplémentaire : deux ellipsoïdes de galbes différents doivent avoir le même poids, c'est à dire la même surface apparente moyenne, puisqu'ils n'ont pas pour rôle de coder la densité d'occupation. Le rapport entre la surface apparente dans une direction et la surface apparente moyenne correspond à la part variable de l'occultation anisotropique.

Calcul de l'illumination

Un ellipsoïde se caractérise par une base R et trois longueurs r_i, la forme quadratique associée $M_o^t.Q_o.M_o = 1$ a pour matrice $Q_o = R^t.D^{-2}.R$ dans l'espace initial, avec D diagonale telle que $D_{i,i} = r_i$, R normale, et M_o un point sur l'ellipsoïde (l'origine étant prise au centre de l'ellipsoïde). Convertir l'ellipsoïde dans l'espace scène est équivalent à ramener dans l'espace objet un point M_s de l'ellipsoïde résultant :
$M_s^t.Q.M_s = 1$ (avec Q positive) $\Leftrightarrow \|D^{-1}.R.J_{T^{-1}}.M_s\| = 1 \rightarrow Q = J_{T^{-1}}^t.R^t.D^{-2}.R.J_{T^{-1}}$

Comme indiqué plus haut, on doit extraire la surface apparente et la normale correspondant à un point échantillonné sur cette surface. L'ellipse apparente est la projection de l'ellipsoïde parallèlement à la direction d de l'observateur. La matrice de sa forme quadratique est $Q' = Q - Qdd^tQ/d^tQd$. Les valeurs propres l et h et les vecteurs propres $\vec{l}$ et $\vec{h}$ en

22. Puisque la transformée affine d'un ellipsoïde est un ellipsoïde.

23. Pour des raisons de commodité, notamment pour ne pas avoir à transformer toutes les sources de lumière dans l'espace objet. En fait nous avions d'abord implémenté l'autre solution, quand nous ne gérions qu'une source.

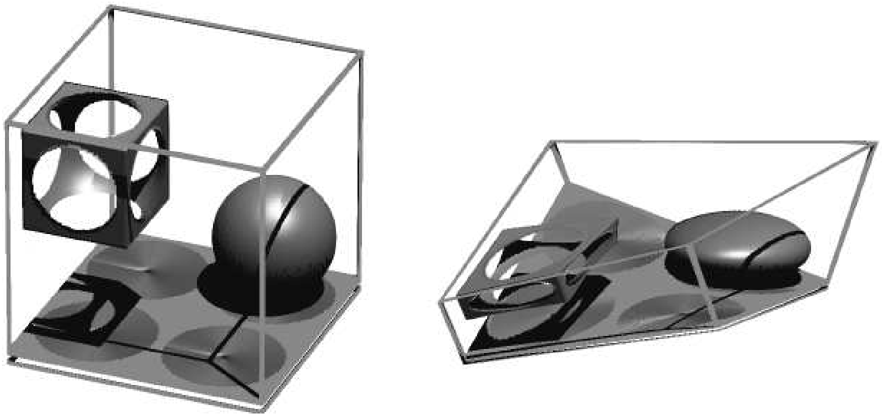


FIG. 4.15 - *Un texel de texture volumique avant et après la distorsion trilinéaire due au mapping. Le sol est légèrement anisotropique. On peut constater sur la sphère que la transformation des effets lumineux (spéculaire, diffus et ombres) ne suit pas la déformation géométrique comme cela aurait été le cas s'ils avaient été 'peints' dans la texture.*

donnent la boîte englobante. Etant donné (x, y) sur la surface de l'ellipse apparente, z tel que $M = (x, y, z)$ soit sur l'ellipsoïde est obtenu à partir de l'équation du second degré

$$z^2.d^tQd + 2z.(x.d^tQl + y.d^tQh) + (x^2.l^tQl + y^2.h^tQh + 2xy.h^tQl - 1) = 0$$

$z(-x, -y)$ peut être calculé simultanément, il suffit donc d'échantillonner la moitié de l'ellipse. Tous les termes $v_1^tQv_2$ sont bien sûr précalculés. Lors de l'échantillonnage de la demi-ellipse apparente, la normale à l'ellipsoïde en M s'obtient par $N = \frac{Q_s.M}{||Q_s.M||}$, qui conduit à l'illumination en ce point par le modèle de Phong. Sans calculer de produit matrice-vecteur et en utilisant les termes précalculé $v_1^tQv_2$, on extrait directement $QM = x.Ql + y.Qh + z.Qd$ d'où l'on tire la normale, puis l'illumination locale. Le schéma de calcul de l'illumination détaillé plus haut peut alors être évalué.

Comme on l'a dit, il faut procéder à une normalisation du résultat pour traiter avec le même poids les différentes formes, normalisation qui correspond à la surface apparente moyenne de l'ellipsoïde. On l'approxime à partir de la surface apparente dans la direction des trois axes :
$Smoy \simeq \pi/3.(r_1r_2 + r_2r_3 + r_1r_3)$.
Il n'est bien sûr pas question de calculer les valeurs propres pour trouver les r_i, ce qui conduirait à résoudre une équation de degré 3 (rappelons que tous ces calculs se font pour chaque voxel traversé). En définissant la matrice $K = J_{T^{-1}}^t.R^t.D^{-1}.R.J_{T^{-1}}$, qui a $1/r_i$ pour valeurs propres quand $J_T = Id$, on a $Smoy \simeq \pi/3.|trace(K)/det(K)|$ (ce n'est exact que lorsque la déformation est un mouvement rigide).

En sortie de calcul, on dispose alors d'une contribution diffuse, une contribution spéculaire, et une surface apparente normalisée c'est à dire l'occultation effective, qui correspond à la fois à l'albédo (on reflète d'autant plus de lumière que le milieu est dense) et à l'opacité (on voit d'autant moins à travers que le milieu est dense). Selon les couleurs ambiante, diffuse et spéculaire du matériau et en fonction de la lumière incidente évaluée par les rayons d'ombrage, en tenant compte de l'opacité cumulée avant d'atteindre le point courant du volume, on obtient en définitive la contribution du voxel à la couleur présente du pixel d'où part le rayon. L'opacité du pixel augmente de la valeur de celle du voxel, et on poursuit la traversée du rayon tant que les voxels peuvent apporter une contribution au pixel, c'est à dire jusqu'à ce que l'opacité soit totale (ou la texture traversée de part en part).

4.4 La matière

Les textures volumiques représentent un matériau plus qu'une matière, en ce sens qu'elles décrivent la répartition de la matière au voisinage d'une surface plutôt que sa couleur. La réflectance elle même ne varie que pour simuler une micro-géométrie. Comme pour la géométrie classique, l'utilisateur a besoin de définir les variations de couleur au sein de l'espace occupé par les texels. Plus généralement, on veut pouvoir piloter le long de la forme les paramètres du modèle d'illumination, ce qui est exactement le rôle d'une texture classique. Il nous faut donc définir comment les textures volumiques peuvent hériter de l'habillage dont on dote usuellement les surfaces.

Dans notre implémentation, nous attachons comme c'est classique une structure *matière* aux faces. Cette structure contient les coefficients du modèle de Phong, et le type de texel à appliquer à cet endroit. Elle contient également une transparence, et une permutation éventuelle des orientations du texel (une autre structure attachée aux faces contient en outre les informations de perturbation discrètes vues en 4.2.4, translation, rotations, scaling). Deux autres catégories d'information importantes s'y trouvent : un pointeur sur une procédure de calcul, qui va nous servir à gérer les textures, et un chaînage sur une autre matière, qui va nous permettre de superposer[24] les texels dans un même espace.

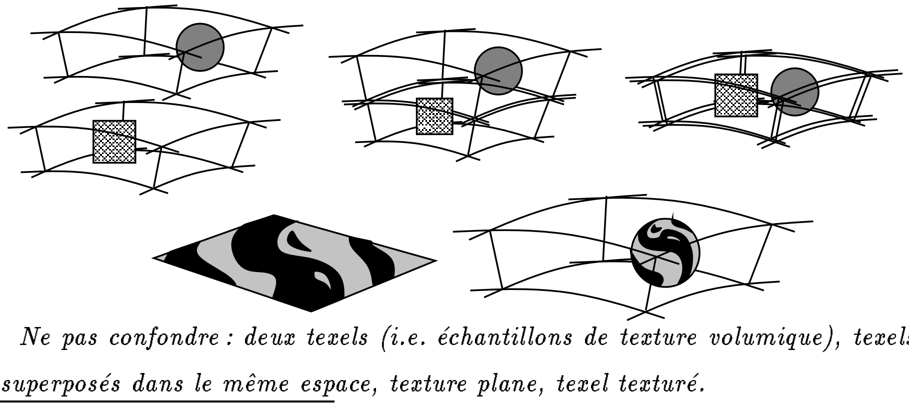


FIG. 4.16 - *Ne pas confondre : deux texels (i.e. échantillons de texture volumique), texels superposés en strates, texels superposés dans le même espace, texture plane, texel texturé.*

24. Il s'agit ici de superposition au sens de 'fusion', et non de strates comme il en était question plus haut ; cf figure 4.16.

4.4.1 Des textures dans la texture

Une texture (classique) correspond à une fonction qui détermine les paramètres de la matière en chaque point d'un objet. Elle obtient ces valeurs soit par calcul, soit au moyen d'une carte. Dans notre implémentation, elle dispose de paramètres qui sont eux-mêmes des matières (afin par exemple d'interpoler entre deux couleurs en fonction de la valeur d'un bruit de Perlin).

Ces paramètres peuvent eux-mêmes être obtenus par voie procédurale, le mécanisme ressemble donc fort à un arbre de shading [NF87], et suppose un mécanisme de mise à jour évolué si l'on veut éviter de recalculer plusieurs fois les mêmes valeurs : chaque matière correspond à un nœud de cet arbre (qui peut être un graphe), ce nœud contenant :

- un indicateur de validité signalant si le nœud a déjà été évalué ;
- un indicateur d'échantillonnage signalant si le nœud doit être évalué une fois par face ou en tout point ;
- un indicateur signalant l'espace auquel s'applique la texture (scène, face ou texel) ;
- jusqu'à quatre paramètres matières ;
- une méthode d'invalidation marquant le nœud non valide (sauf s'il ne doit être évalué qu'une fois par face et que celle-ci n'a pas changé), et propage l'invalidation à ses éventuels paramètres matières procédurales ;
- une méthode de calcul, qui provoque d'abord le calcul de ses paramètres matières s'ils sont eux-mêmes procéduraux et invalide, puis détermine les valeurs de la matière en fonction des paramètres et de la position du point courant ;
- une méthode de libération, appelée lorsque l'utilisateur veut détruire ou redéfinir une matière.

De surcroît, nous disposons de trois systèmes de coordonnées possibles pour exprimer le paramétrage de la texture, l'espace $\mathcal{S}$ de la scène, l'espace $\mathcal{F}$ de la surface (et des boîtes), l'espace $\mathcal{T}$ de la texture, qu'on utilise le plus souvent. Cela donne l'algorithme suivant pour la fonction chargée de calculer la valeur de la matière en un point :

```
si (mati->texture)
  récupère_position X                          ' coordonnées texel  (u,v,w)
  récupère_face face
  si (mati->echantillonnage=FACE et face inchangée) alors retour
  si (mati->espace = BOITE ou SCENE) alors X = texture_vers_boite(X)
  si (mati->espace = SCENE) alors X = boite_vers_scene(X)
  mati->invalide(mati)
  mati->eval(mati,X,face)
```

La méthode d'invalidation d'une matière qui utilise les paramètres matières 2 et 3 sera :

```
si (mati->valide=0) alors retour              ' déjà invalidée
si (mati->echantillonnage=FACE et face inchangée) alors retour
mati->valide = 0                              ' invalide, et propage l'invalidation
si (mati->mati2 existe et est procédurale) alors mati2->invalide(mati2)
si (mati->mati3 existe et est procédurale) alors mati3->invalide(mati3)
```

La méthode de calcul **eval**(mati,X,face) est construite sur le schéma suivant :
(noter que ses paramètres sont stockés dans une structure matière, en utilisant des variables prévues pour un autre usage)

```
D = mati->mati1->speculaire                   ' Direction
S = mati->mati1->diffus                       ' Scaling
T = mati->mati1->ambiant                      ' Translation
k = mati->mati1->rugos                        ' un paramètre optionnel
u = texture(X, D,S,T,k)                       ' coef d'interpolation
si (mati->mati2 est procédurale et invalide) eval(mati2)
si (mati->mati3 est procédurale et invalide) eval(mati3)
mati->speculaire = interpol(u, mati2->speculaire,mati3->speculaire)
mati->diffus     = interpol(u, mati2->diffus    ,mati3->diffus    )
mati->ambiant    = interpol(u, mati2->ambiant   ,mati3->ambiant   )
```

4.4.2 Mélange de texels

De même qu'en 2D on peut superposer des textures qui sont par endroit transparentes, Il est intéressant de pouvoir superposer (i.e. fusionner) des texels. Dans la mesure où un texel ne remplit qu'une partie de l'espace disponible, on peut ainsi composer une zone de la peau volumique avec plusieurs instances de textures volumiques.

Cette superposition est spécifiée en chaînant les matières les unes aux autres [25]. Dans notre implémentation c'est le seul moyen - avec les textures - de disposer de plusieurs couleurs dans le même volume. C'est également un moyen commode de rendre la texture moins répétitive, dans la mesure où l'on peut superposer diverses combinaisons de texels, et que l'on peut opérer n'importe quelle symétrie ou rotation de $\pi/2$ (i.e. toute permutation des axes) à chacun des texels que l'on superpose.

Il se trouve que la structure en octree et son mode de parcours se prêtent bien à la traversée en parallèle de structures identiques : les octrees sont en effet superposables (même avec des

25. Ce qui était initialement prévu pour donner une couleur différente au texel et au sol.

rotations de $pi/2$), seule diffère la profondeur de leurs ramifications (il serait considérablement plus difficile de traiter deux texels s'intersectant de manière quelconque). On se contente donc, dans la fonction de traversée de l'octree détaillée au chapitre précédent (section 3.2.2), de maintenir une liste de voxels au lieu d'un seul, et de calculer les illuminations locales à la suite les unes des autres quand la résolution adéquate est atteinte. Cela semble constituer une approximation dans la mesure où les voxels occupant un même espace sont rendus dans un ordre donné, alors que l'opacité se cumule entre chaque (i.e. le texel chaîné en premier apparaît plus). Ceci ne se produit cependant que lorsqu'un même voxel est effectivement occupé par plusieurs texels, ce qui est rare.

A noter que les divers texels n'ont pas forcément la même résolution. Il faut donc compter et marquer les voxels dont on a atteint la taille ultime afin de ne poursuivre la subdivision que pour les autres, tant qu'il y en a.

4.4.3 Résultats

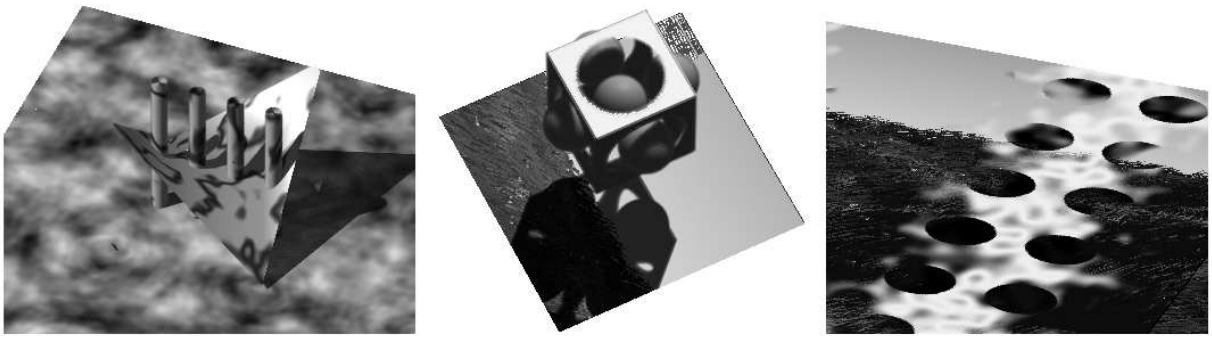


FIG. 4.17 - A gauche : *texel en marbre, surface en nuage (deux bruits de Perlin).* Au milieu : *deux texels superposés (cube et sphères), image plaquée sur la surface.* A droite : *aspect d'un arbre à 3 nœuds, constitué d'une texture de Perlin ayant pour paramètres de couleur 2 textures images.*

4.5 L'animation

L'animation de scènes complexes est tout aussi cruciale pour le réalisme que leur modélisation. Penser par exemple aux effets du vent dans un champ ou dans un feuillage, aux ondulations de la fourrure d'un animal qui court... Elle pose d'ailleurs le même genre de problèmes : le mouvement se décline en énormément de composantes qui présentent cependant une certaine similarité, ce qui entraîne le problème de la quantité d'information que l'utilisateur doit spécifier - donc connaître dans le détail - , et de l'échelle à laquelle il faut donner les spécifications du mouvement.

Comme pour la modélisation d'objets complexes avec les textures volumiques, il est commode et efficace de séparer la description du mouvement local à petite échelle du mouvement d'ensemble à grande échelle. Il faut noter que la plupart des représentations de scènes complexes sont d'abord conçues pour les scènes statiques. Elles visent soit à spécifier la géométrie (ex : L-systèmes), soit à la hiérarchiser pour accélérer son rendu (ex : hiérarchie de boîtes englobantes). Deux types d'approches ont été développés pour créer des animations crédibles dans le cadre du 'réalisme naturel' (*natural look*) : les modèles physiques ou pseudo-physiques (ex : vagues sur l'océan [FR86]), et les systèmes de particules (ex : chutes d'eau[Ree83]). Leurs résultats sont spectaculaires, mais ne peuvent pas facilement être réutilisés : les deux méthodes sont monolithiques, et il n'y a pas de séparation entre le modèle et son animation (et même son rendu, pour les systèmes de particules). Il est alors difficile d'animer une forme préexistante donnée de cette façon. De plus, les deux méthodes prennent le contrôle de la 'simulation', ce qui fait qu'il est difficile pour l'utilisateur de spécifier précisément et interactivement le mouvement. Un problème particulier des systèmes de particules réside dans leur spécificité de rendu : il est difficile de les intégrer parmi d'autres objets dans une scène autrement que par compositing [26], ce qui limite les interactions lumineuses de type ombrage ou reflet.

Dans ce contexte, les textures volumiques offrent un certain nombre d'avantages :

- la disposition des objets incorporés dans les texels de la peau volumique est déterminée par l'orientation de celle-ci, elle même guidée par la surface sous-jacente et les vecteurs aux points. Il n'y a donc pas besoin de 'squelette' pour gérer le mouvement, celui-ci provient essentiellement de déformations spatiales, distinctes de l'objet lui-même.
- les tests de collision entre objets sont très simples à gérer, il suffit de s'assurer que les texels ne s'intersectent pas.
- il n'y a pas d'autocollision à l'intérieur des texels tant que ceux-ci ne sont pas dégénérés (il faut que le rayon de courbure soit supérieur à l'épaisseur).
- la spécification de l'animation peut se faire avec peu de paramètres.
- le rendu des textures volumiques étant rapide, le calcul de séquences n'est plus un luxe, même pour de simples tests de mise au point.

26. Mélange d'images en tenant compte du plan alpha (opacité de l'image).

4.5.1 Un modèle multiéchelle d'animation

L'animation de la texture volumique se fait en jouant sur les paramètres de mapping, principalement en contrôlant les vecteurs qui orientent la peau volumique (il sont initialement orthogonaux à la surface) : leur mouvement définit la déformation des boîtes, qui entraîne la déformation de leur contenu.

- Pour une surface déformable, comme du tissu, la peau volumique qui la recouvre suit naturellement les déformations comme le ferait un matériau continu (cf 4.5.2).
- Sur une surface rigide, le mouvement peut être directement généré par un champ de force agissant sur les vecteurs aux points, déformant ainsi les texels (cf 4.5.3).
- Un troisième mode d'animation correspond au dessin animé : les étapes successives d'un mouvement simple (comme l'oscillation des feuilles dans le feuillage) peuvent être conservées par quelques volumes, que l'on utilise alternativement au cours du temps (cf 4.5.4).

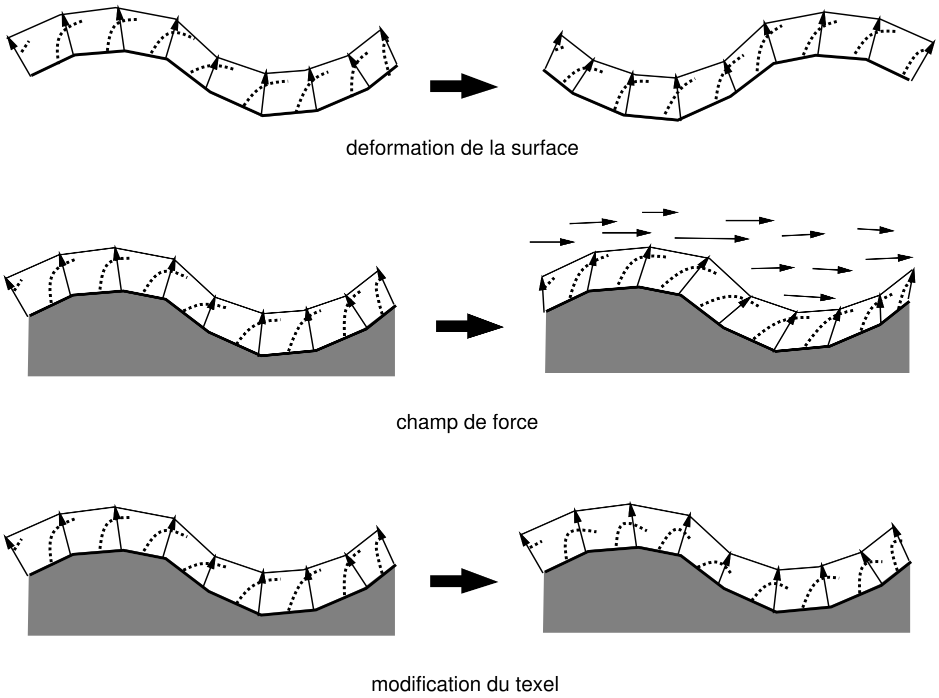


FIG. 4.18 - *Les trois manières d'animer les texels.*

Pour une animation réaliste, on peut fusionner les trois méthodes afin de gérer les différentes échelles du mouvement : la fourrure sur la peau d'un animal en train de courir suit les déformations du corps de l'animal, ondule sous l'effet de l'accélération et de l'inertie, et sans doute les poils oscillent localement. De même pour un arbre sous le vent, les branches et leur ramure ploient, le feuillage ondule dans la ramure, et les feuilles oscillent dans le feuillage.

4.5.2 Animer la surface

La peau volumique est liée à la surface, et les vecteurs aux sommets donnent son orientation, ce qui permet de 'coiffer' les texels en penchant ces vecteurs par rapport aux normales à la surface. L'animation des texels découle de la simple prise en compte de la nouvelle surface et de ses nouvelles normales à chaque pas de temps. Il faut juste prendre garde au maximum de courbure de la surface, qui peut conduire à des dégénérescences quand le rayon de courbure est proche ou inférieur à l'épaisseur de la couche. En figure 4.21, un volume de référence représentant un morceau d'échafaudage est mappé sur un drapeau animé par un modèle masse-ressort non linéaire [27].

4.5.3 Animer les déformations

La déformation des texels peut être explicitement contrôlée par le mouvement des arêtes verticales. Ce mouvement peut être généré par un champ de force (ex : un champ de Laplace [WH91], un flux stochastique[SF92]) agissant sur les arêtes, ou par un schéma dynamique (ex : masse-ressort, modèle d'élasticité [TF88]) liant le haut des arêtes verticales.

La figure 4.20 représente une prairie sous le vent. Un texel contient 16 brins d'herbe. Le mouvement est généré par un champ de force animé agissant sur les arêtes verticales. Ce champ $\vec{F}(M,t)$ modélise une rafale de vent combinée à un bruit aléatoire, codés par deux champs séparés. Etant donné la direction de propagation $\vec{F}/\|\vec{F}\|$, l'intensité du vent au point M est obtenue par l'expression de la propagation des ondes $f(2\pi\frac{\vec{F}.\vec{M}}{\lambda} - \omega.t)$ (l'origine est arbitraire).

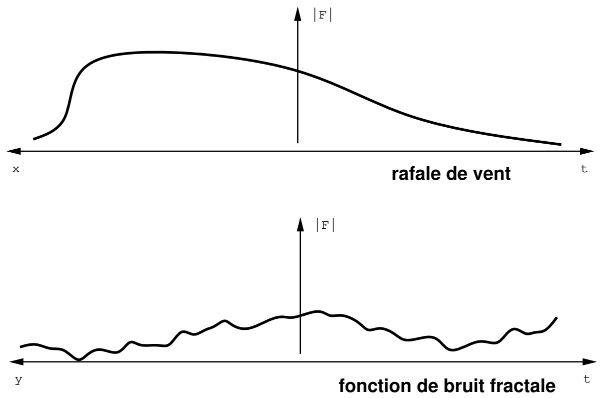


FIG. 4.19 - *Intensité du vent dans les directions horizontales x et y.*

---

27. le modèle de tissu est dû à Xavier Provot [Pro95].

Pour la composante pure du vent, $f()$ est une fonction continue périodique, dont le motif que l'on a choisi a une attaque brusque et une chute lente (voir figure 4.19) de manière à imiter une rafale. La composante bruit est construite dans la direction orthogonale du vent, avec un bruit 3D fractal comme intensité $f()$ dans l'expression de propagation. Le *bruit 3D*, ordinairement utilisé en synthèse pour les textures [Per85], fournit un signal qui est simultanément continu, dérivable et pseudo aléatoire. La pseudo-fréquence de la fonction de bruit peut être contrôlée, et est utilisée pour définir une fonction fractale appelée *turbulence*, plus réaliste à la fois pour les textures et les mouvements aléatoires. Les vagues d'ombre et de luminosité appuient l'effet de mouvement.

4.5.4 Animer le volume

L'animation type dessin animé est basée sur le cyclage d'étapes du mouvement. Ceci peut également se faire en 3D pour les mouvements simples ou rapides comme l'oscillation de parties de l'objet codé dans le texel : quelques états sont codés dans différents volumes, que l'on utilise successivement au cours du temps (on pourrait également modifier un unique volume à chaque pas de temps). Cependant cela accroît soit le coût de stockage soit le nombre de volumes à recalculer. Cette méthode doit donc être réservée aux effets spécifiques présentant des mouvements très simples, comme les oscillations des feuilles dans le feuillage. Cela permet également de briser la régularité du mapping, la phase de la boucle d'animation pouvant être distribuée aléatoirement sur la surface.

4.5.5 Résultats

Avec un peu d'imagination à partir des commentaires fait plus haut, et avec un petit effort pour entendre le bruit du vent, il doit être possible de voir bouger ces images conformément aux animations que nous avons calculées...

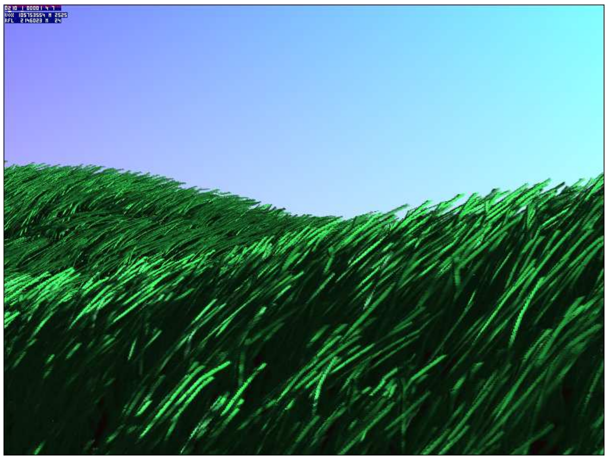


FIG. 4.20 - *Prairie sous le vent.*

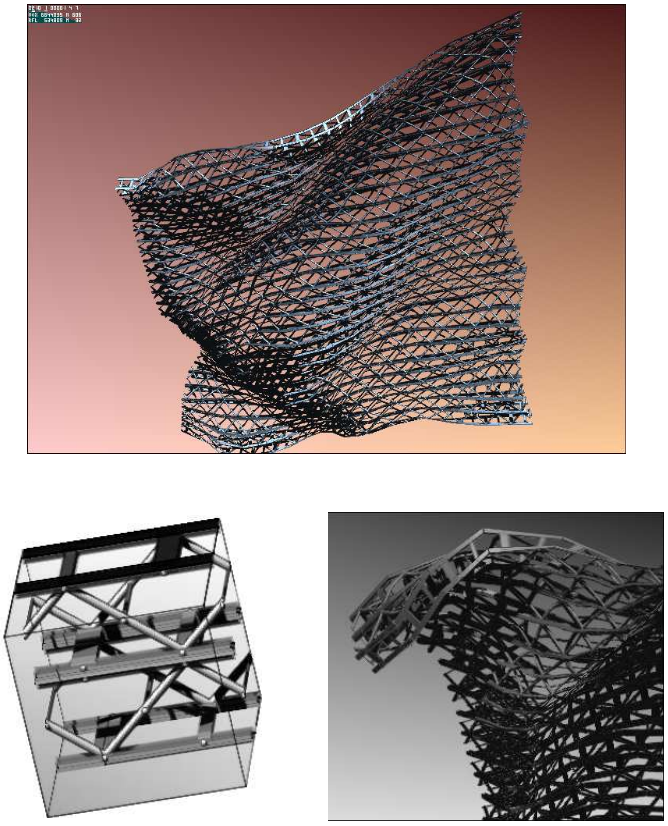


FIG. 4.21 - En haut *: un drapeau en échafaudage.* A gauche *: un texel seul.* A droite *: gros plan sur un coin.*

4.6 En résumé

Ce chapitre avait pour but d'étendre le noyau de textures volumiques dont nous avons décrit le modèle au chapitre précédent, dans le but d'en faire une représentation utilisable. A ce stade, on peut maintenant :

- créer les texels à partir d'objets préexistants de différentes natures ;
- mapper ces texels sur les surfaces aussi librement qu'on peut le faire en 2D, avec un degré de liberté en plus, le vecteur qui en tout sommet donne l'orientation de l'épaisseur ;
- doter ces texels d'une matière, qui revient à mettre de la texture ordinaire sur les formes contenues dans les texels, ou combiner des texels afin de composer un texel complexe qui soit une véritable petite scène avec plusieurs objets de plusieurs matières.
- animer les texels à diverses échelles, afin de leur donner vie.

Aux limites près de la méthode indiquées à la fin du chapitre précédent, qui proviennent pour la plupart de la notion même de texture, on dispose donc maintenant d'un véritable outil au service des graphistes, qui devrait permettre facilement et à moindre coût de faire grouiller les scènes de petits détails. Le modèle n'est bien sûr pas pour autant définitivement abouti, nous envisageons quelques uns des développements possibles au chapitre 5.3.

Chapitre 5

Conclusions

5.1 Émergence d'une nouvelle approche

Au fur et à mesure de l'enrichissement de la représentation des textures volumiques, ce qui n'était au départ qu'un modèle dédié à la fourrure se transforme peu à peu en représentation concurrente de la géométrie, non seulement avantageuse dans le cas des objets complexes répétitifs, mais également à même de proposer des débuts de solution à plusieurs grands problèmes de la synthèse d'images.

5.1.1 A l'origine, un modèle de fourrure

Dans l'article fondateur de Kajiya et Kay en 1989, les textures volumiques sont introduites pour réaliser des modèles 'exotiques', notamment la fourrure. L'article débouche sur la fameuse image de l'ours en peluche.

La construction est facile, les résultats sont convainquants, mais le modèle est très limité et fort coûteux.

5.1.2 Une nouvelle race de textures, avec une épaisseur

Même si l'implémentation était limitée, les bases d'une approche assez générale étaient présentes dans l'article fondateur.

L'idée plus générale consiste à introduire une nouvelle sorte de texture, qui donne des effets plus 'puissants' et plus '3D' que le bump mapping ou le displacement mapping.

Ceci est obtenu en utilisant comme échantillon de texture un cube volumique à la place de l'image traditionnelle, et en intégrant des données de réflectances dans la texture. Cette réflectance joue le rôle des normales dans le bump mapping 2D, mais l'idée d'une 'texture de réflectance' n'est pas sans rappeler les articles de Fournier et Poulin [Fou92, PF90] visant à mapper l'anisotropie. Cela fait également écho aux réflexions de Kajiya concernant la hiérarchie de représentation, qui devrait être alternativement géométrique, texturelle ou photométrique selon l'échelle (i.e. la taille apparente).

5.1.3 Un modèle de scènes complexes

Au chapitre 3 [1], j'essaie de rendre effectif tout ce que les texels ont de puissance potentielle depuis l'origine. Le modèle est rendu plus général en codant réellement la réflectance dans le volume, le stockage devient plus économe grâce aux octrees, et surtout la méthode devient multiéchelle ce qui la rend bon marché alors qu'elle était initialement ultra-coûteuse. Ce dernier point constitue l'équivalent du mip-mapping pour les textures 2D (l'idée d'utiliser la réflectance pour 'résumer' la géométrie qu'on voudrait représenter localement peut être itérée, en 'résumant' les réflectances d'une zone par une seule fonction de réflectance).

Les textures volumiques deviennent alors une représentation concurrente de la géométrie, plus efficace quand il s'agit d'objets complexes. On obtient ainsi un modèle général pour les scènes répétitives complexes. A la limite, on pourrait aussi les utiliser pour un objet non répétitif très complexe, où une représentation en facette est hors de prix.

Animation

Dans la section 4.5 [2], je montre que la représentation en texels se prête particulièrement bien à l'animation, en permettant de séparer les échelles : à grande échelle, c'est la surface de support qui se déforme (ramure d'un arbre sous le vent, peau d'un animal qui court) ; à moyenne échelle, la matière - le texel - se déforme (ondulations sous l'effet du vent dans le feuillage, ou sous l'effet de l'inertie dans la fourrure) ; à petite échelle, les objets s'agitent localement (bruissement des feuilles) - cyclage entre quelques volumes de référence -.

Encore plus général

Après avoir traité l'animation, j'ai rapproché le mapping des texels des méthodes classiques de mapping de texture (auparavant la surface devait être subdivisée en patches bilinéaires, servant chacun de base à un texel), et j'ai étudié comment mettre de la texture dans la texture volumique (solid texturing à la Perlin pour la couleur ou la transparence), cf chapitre 4 et [Ney96a]. J'ai également visé à rendre plus 'physiques' les effets des déformations sur le contenu du texel (rayons courbes, déformation de l'illumination, cf 4.3 et [Ney95c]).

Il reste de nombreuses tâches à réaliser, et il y a de nombreuses voies à explorer :

1. Dont l'essentiel a été publié à GI95 [Ney95b].
2. Qui a fait l'objet d'une publication à EWAS95 [Ney95a].

il faut développer tout ce qui permet de rendre la texture moins répétitive, tout ce qui permet aux texels de ressembler encore plus à de vrais objets 3D (interpénétration de deux texels ou d'un texel et d'un objet, blocage à petite échelle), et tout ce qui permet de construire les texels (outils interactifs, extraction depuis des images réelles...), outre la conversion depuis une BDD existante traitée au chapitre 4.

Une voie consiste à enrichir la représentation, mais on peut aussi a contrario chercher à décliner la méthode vers l'allégement : si l'on veut coder tout objet d'une scène par un texel il faut réduire le stockage, si l'on veut essayer de s'adapter à la réalité virtuelle il faut gagner 3 ordres de grandeur sur le temps de calcul (ce qui nécessite de passer au rendu projectif, en s'inspirant des nouvelles méthodes temps réel de visualisation volumique).

Un enrichissement intéressant serait de pousser la hiérarchie de l'octree en prévoyant de stocker un texel dans un voxel : si l'on veut coder toute une ramure dans un texel, on peut vouloir y faire figurer toutes les feuilles, qui se ressemblent donc qui pourraient être codées par un seul texel. De la même façon, on pourrait passer de la ramure à l'arbre et de l'arbre à la forêt, celle-ci couvrant le paysage. On pourrait alors procéder à des zooms poussés explorant un modèle unique...

5.1.4 Conséquence : une nouvelle approche pour les niveaux de détails

Comme le disent les remarques de Kajiya sur la hiérarchie de représentation, confortées par les travaux de Fournier, la décimation des facettes ne **peut pas** être la bonne façon de traiter les niveaux de détails, si l'objectif est le réalisme : vue de loin, une tôle ondulée est géométriquement proche d'une tôle plate, mais son illumination est complètement différente puisque les normales restent 'sinusoïdales', même de loin. La démarche de Fournier en 2D comme celle des texels en 3D permet précisément de séparer la géométrie et la 'réflectance' (codant les variations locales de la géométrie). Les texels sont donc potentiellement des bons concurrents des représentations surfaciques explicites (ex : facettes), au moins pour ce qui concerne les niveaux de détails. Reste cependant à alléger le stockage si l'on en vient à utiliser un texel par objet.

5.1.5 Conséquence : manipulation facile de l'anisotropie

Comme la réflectance et la 'géométrie' sont codées séparément, il est très facile de donner une réflectance arbitraire à une surface (cf à la fin de 3.1.1). Le modèle simplifié de réflectance que nous utilisons rend l'opération légère quant au nombre de paramètres à gérer (grosso-modo une rugosité différente selon 3 axes, et un repère) et au surcoût de rendu (nul). On peut même utiliser les filtrages successifs du modèle multiéchelle pour obtenir en filtrage ultime le modèle de réflectance anisotropique qui correspond à une micro-géométrie. Bien sûr, il s'agit plus d'un modèle commode pour disposer d'effets d'anisotropie que d'un modèle physique, mais il est suffisant pour la plupart des cas rencontrés en production d'image (aluminium brossé, fond de casserole...).

5.1.6 Conséquence : du nouveau en volumique

Ayant introduit un modèle photométrique dans chaque voxel, il n'est plus utile de garder une résolution très fine de la géométrie dans le seul but de restaurer une normale utilisable pour la visualisation. D'autre part, la représentation apporte un moyen de faire entrer une surface 'dans' le volume, plutôt que d'essayer d'intercaler représentation en volume et en facettes.

5.1.7 Conséquence : une autre façon de gérer les déformations spatiales

Les texels ne sont rien d'autre que des déformations spatiales d'un objet de référence codé sous forme de volume. Comme les texels sont nombreux dans une image, petits par rapport à la quantité d'information qu'ils contiennent, et parfois déformés au cours du temps, il serait extrêmement coûteux de construire explicitement les objets résultants des déformations spatiales. De plus, on ne sait généralement pas construire la déformation spatiale de voxels, comme pour beaucoup d'autres représentations (fonctions implicites, CSG...), à moins de changer de modèle (par facettisation, typiquement).

La démarche suivie pour les texels, consistant à déformer les rayons pour se ramener à l'objet de référence plutôt que de construire la transformation de l'objet, est donc avantageuse pour traiter la déformation d'objets relativement complexes et la déformation d'objets représentés autrement que par une surface explicite (cf 4.3). L'article [Ney95c] introduisait cette démarche dans le cadre général, qui pose cependant le problème des rayons courbes (que l'on peut soit linéariser, soit approximer par grillage du volume pour les déformations simples).

5.2 Le nouveau modèle de textures volumiques

5.2.1 Représentation

Voici les diverses structures définissant notre représentation. Nous avons omis les informations que l'on trouve dans tous les modèles de synthèse d'images (notamment en matière de description géométrique et d'optimisation de rendu).

• La surface sous-jacente :

- le maillage est à base de faces triangulaires ou quadrangulaires ;
- données aux sommets :
  - vecteurs hauteur, contrôlant l'orientation verticale de la texture (permettant ainsi de 'coiffer' les texels) ;
  - coordonnées textures (u, v, w), ainsi que l'offset de texture dw (l'extrémité du vecteur hauteur a pour coordonnées textures $(u, v, w + dw)$;
- données aux faces :
  - matière à utiliser (laquelle contient le type du texel) ;
  - permutations et symétries éventuelles à appliquer aux texels ;

• La matière :

- paramètres classiques de Phong (couleurs ambiante, diffuse et spéculaire, rugosité) ;
- fonction de calcul éventuelle de la couleur et de la transparence (texture classique à appliquer au texel), plus 4 paramètres de matière éventuels, plus quelques flags et fonctions nécessaires au bon fonctionnement de l'arbre de matières ;
- numéro du type de texel à utiliser ;
- permutations et symétries éventuelles à appliquer à ce texel ;
- chaînage éventuel à une autre matière, pour fusionner à un autre texel, ou pour indiquer la matière du 'sol'.

• L'échantillon de texture :

- volume représenté par un octree, dont les étages correspondent aux diverses résolutions de la texture volumique ;
- les voxels contiennent :
  - une probabilité d'occultation ;
  - 6 paramètres d'un ellipsoïde représentant la réflectance locale ;
  - un pointeur vers un bloc de 8 voxels fils.

5.2.2 Fonctionnalité

L'utilisateur n'a pas à se soucier des détails techniques de la représentation. Pour lui, les paramètres utiles sont :

- une surface sous-jacente à base de facettes, faisant partie d'une scène 3D ;
- une méthode de mapping (de type classique si on prend $w = 0$ et $dw = 1$) ;
- des outils de perturbation des vecteurs hauteur, qui peuvent être animés ;
- des outils de construction d'échantillons de texture à partir de représentations classiques (ex : maillages facettes, L-systèmes, systèmes de particules, hypertexture, etc) ;
- des paramètres de matière relativement classiques, qui incorporent le numéro de texture volumique à employer.

5.2.3 Bilan

Nous avons implémenté le modèle sous la forme d'une plateforme de synthèse d'images, laquelle a déjà produit de nombreuses animations de scènes complexes pour un coût de rendu allant de 10 à 40 minutes selon les images (20 minutes en général), à comparer aux 12 heures pour l'image de l'ours en peluche de Kajiya. Non seulement on peut maintenant calculer des scènes très complexes, mais leur animation est accessible (ce qui représentait jusqu'à présent un luxe inimaginable), de même pour le calcul de bouts d'essais. Les modèles sont *texelisés* à partir de maillages existants, de L-systèmes ou de scripts, l'animation peut se faire à l'aide de modèles de vent, le rendu est intégré avec le ray-tracing usuel (à la différence des systèmes de particules, notamment), on bénéficie donc autant que possible des outils auxquels l'utilisateur est habitué.

L'objectif d'obtenir un système utilisable semble atteint, mais l'aventure n'est pas terminée pour autant. Au contraire, maintenant que la faisabilité est bien établie, sans doute faudrait-il d'une part étendre le modèle local, d'autre part ajouter de nombreux outils ! Il faudrait par exemple étudier plus profondément la nature des données à stocker et leur mode de filtrage, gérer les problèmes d'imprécision numérique, etc. D'autre part, il faudrait disposer d'outils de façonnage des texels, et d'outils de mapping sur les surfaces ou dans l'espace...

5.3 Horizons

Notre objectif était de concrétiser les potentialités des textures volumiques, en établissant tous les traits qui caractérisent un outil véritablement utilisable, à la différence d'un modèle exotique et isolé pour lequel on devrait produire des paramètres 'à la main'. Ainsi, nous avons fait le pont entre les texels et les représentations géométriques usuelles, les méthodes d'animation, les techniques de mapping et le rendu de scène par lancer de rayons.

Bien des voies s'ouvrent maintenant pour les textures volumiques. les unes pour perfectionner la représentation, les autres pour enrichir l'environnement d'outils de spécification, d'autres encore pour mener les textures volumiques au delà de son territoire actuel.

5.3.1 Intersections entre texels et avec la géométrie

Les textures volumiques ne se comportent pas encore totalement comme de la géométrie, ce qui est d'autant plus choquant qu'elles en ont toute l'apparence. Ainsi on ne peut pas encore mettre le point de vue **dans** la texture (ce qui ne demanderait pas de gros développements), et on ne peut pas 'poser' un véritable objet géométrique dans l'espace occupé par un texel, fût-il vide à l'endroit visé. De même, deux texels ne peuvent pas s'intersecter alors qu'ils ne constituent qu'une boîte englobante des objets représentés. Ce problème a déjà été abordé pour l'intégration de données surfaciques et volumiques[Fru91] : plutôt que de cumuler l'illumination au fur et à mesure de la traversée par le rayon, il faut établir une liste des sites traversés, triée par ordre d'éloignement, avant de calculer l'illumination. Rappelons que nous traitons pour l'instant le cas de la fusion de texels (recouvrement total), ce qui permet par combinaisons de varier les motifs de la texture.

5.3.2 Texels libres - texels multicouches

La géométrie complexe n'apparaît pas uniquement sous la forme de peau surfacique, il serait alors intéressant de disposer directement les texels dans l'espace (par exemple pour du feuillage). Nous gérons une première méthode, qui consiste à superposer[3] plusieurs couches de texels afin de former une grille spatiale. Une autre méthode consisterait à répartir directement les texels a priori non alignés dans l'espace, notamment pour le feuillage, ce qui suppose auparavant d'avoir résolu les problèmes d'intersection entre texels.

3. 'Superposer' signifie ici 'l'un *sur* l'autre' et non pas 'l'un *dans* l'autre'.

5.3.3 Distorsions dues aux approximations linéaires

La linéarisation des rayons dans l'espace des textures où ils devraient être courbes pose quelques problèmes de distorsion. Nous avons déjà vu plus haut qu'il était possible de gérer la courbure des rayons dans l'espace des boîtes (peau volumique). Mais il serait notablement plus coûteux d'opérer de même dans l'espace de la texture, qui est actuellement parcouru par simple dichotomie le long de la trajectoire[4]. Cependant quand c'est vraiment nécessaire, on pourrait partiellement courber les rayons en subdivisant les volumes : entre les volumes on est dans l'espace des boîtes, où l'on sait gérer la courbure (puisque cet espace est parcouru progressivement, et non récursivement). A noter qu'en 2D, on doit pour les mêmes raisons subdiviser le maillage lorsque la distorsion des coordonnées texture est trop forte.

5.3.4 Autres modèles de déformations

Les déformations trilinéaires ne sont pas $\mathcal{G}^1$, il serait intéressant d'en utiliser de plus souples comme les splines. Le coût d'intersection deviendrait cependant plus élevé, d'autant plus qu'on a besoin de permuter entre les divers espaces (absolu, boîtes et texture) et de calculer les Jacobiens. Il serait sans doute plus simple de raffiner le maillage de la surface sous-jacente sans toucher aux coordonnées texture.

5.3.5 Transitions : de la géométrie dans les texels

Bien que nous ne l'ayons pas montré explicitement, la transition entre texels et géométrie authentique ne doit pas poser de problème dans la mesure où les deux modèles sont 3D. Il faut cependant noter qu'a priori, cette transition ne se pose pas entre un volume tout seul et un objet, mais bien entre des texels à base de voxels et des 'texels à base de géométrie' : on pourrait en effet appliquer une bonne partie de l'approche à un volume de référence qui contiendrait une liste de facettes plutôt qu'une grille de voxels. Le texel serait alors une boîte de déformation, et chacun d'entre eux une instance généralisée (car déformée) du motif de base. Cette déformation ne serait que formelle comme on l'a déjà vu, en ramenant les rayons dans l'espace de référence plutôt que de transformer des millier de facettes, voire des modèles non aisément déformables (fonctions implicites, CSG...).

4. A noter que l'on pourrait à la rigueur dichotomer le long d'une spline, puisqu'il en existe une construction récursive.

5.3.6 Construction automatique - sources de données

Nous avons déjà évoqué la conversion automatique de représentations existantes en volumes. Reste que modéliser un échantillon de matière comporte quelques spécificités comme la cyclicité, et qu'il serait commode de bénéficier de divers outils de manipulation et de spécification. Notamment, il serait intéressant de pouvoir construire un volume à partir de données réelles, par exemple pour du feuillage.

5.3.7 Effets de blocage (blocking effects)

La fonction de réflectance que nous utilisons est relativement simple. Cela rend approximatif le filtrage de données hétérogènes, un cube étant par exemple représenté comme une sphère quand il est englobé dans un seul voxel (les angles sont lissés par la fonction). De plus l'absence des effets de blocage à l'échelle du voxel, due tant au filtrage 'géométrique'[5] qu'à l'absence de tables d'horizon[6], tend à sous-estimer l'auto-ombrage. Par contre, enrichir le modèle revient aussi à l'alourdir... Il y a donc des choix à faire selon les objectifs visés, mais il serait sans doute intéressant de disposer de modèles alternatifs pour la réflectance.

5.3.8 Des texels en temps réel ?

Le cadre 'lancer de rayons' a été imposé dès le début, l'objectif annoncé étant la qualité d'image. Mais on pourrait se poser le même problème de représenter efficacement les objets complexes dans un cadre projectif, ce qui serait fort utile notamment en réalité virtuelle où le manque de complexité des scènes est patent. De plus les problèmes de niveaux de détails s'y posent de façon particulièrement saillante. Il faudrait bien sûr reprendre l'élaboration depuis le début, mais les nouvelles méthodes de rendu volumique en temps réel de Levoy [LL94] par superposition de tranches laissent peut-être augurer des solutions raisonnablement rapides.

5.3.9 Multiéchelle au carré : des texels dans les texels

Notre modèle fonctionne pour le moment pour une gamme d'échelles donnée, la plus petite étant atteinte quand chaque voxel se projette en un pixel, et la plus grande quand tout le texel se projette en un pixel. Mais il est clair que la nature présente des autosimilarités à d'autres échelles : quand on modélise un arbre en un texel, on rencontre une certaine complexité répétitive dans les ramures, qui elles-mêmes contiennent un feuillage relativement répétitif,

5. un filtrage exact devrait prendre en compte les positions respectives des primitives à filtrer de manière à tenir compte des masquages réciproques, ce qui n'est assurément pas simple puisque cela dépend de l'angle de vue.

6. Une table d'horizon réglerait au moins le masquage de la lumière, c'est à dire l'ombrage.

qui lui-même se compose de nervures ou bien d'aiguilles... Il est alors très tentant (et il faut se laisser tenter !) d'ajouter un nouveau pas au multiéchelle, en laissant chaque voxel terminal contenir soit des données brutes soit ... un texel, muni d'un repère local. On pourrait alors concevoir un zoom plongeant du survol de la forêt amazonienne couvrant les collines, jusqu'aux nervures d'une feuille, avec un seul modèle...

Annexe A

Questions de langage

Le choix de l'origine linguistique du vocabulaire qu'on utilise pour les sciences et techniques, notamment en synthèse d'images et en informatique, semble susciter de telles batailles qu'il me faut justifier mes propres choix :

Il me paraît clair que dans la mesure du possible, l'auteur doit faire l'effort d'utiliser les expressions qui existent de toute éternité - ou du moins depuis longtemps - dans sa langue, même si la pratique de l'anglais donne la tentation d'utiliser des formes plus concises. Plus perverse est l'influence anglo-saxonne sur la tournure des phrases, notamment l'usage intensif du passif (je suis déjà contaminé !), et de certaines expressions faussement proches comme 'tels que' ou 'sophistiqué'. On peut noter que nombre de mots techniques sont suffisamment anciens pour qu'on les emploie sans avoir à réfléchir, comme les noms des principaux périphériques (clavier, souris, écran, tablette), et que certains autres sont si directement traduisibles que le mot français est préféré sans problème, comme c'est le cas pour une partie des éléments d'interaction graphique (fenêtre, menu, icône), la traduction étant même parfois un peu trop directe ('librairie').

Cependant une forte partie du vocabulaire récent s'est constitué dans la langue des pays les plus actifs dans la technologie donnée, et force est de constater que ça n'est pas la notre. Ces mots ont dans ce contexte un sens très étroit, parfois relativement éloigné du sens courant qui n'est plus guère que mnémotechnique (scroll, scan, buffer, swap, shell, keyframe, sweep, trimmed curve...). Mieux encore, les connotations étant pour nous relativement vierges, le mot peut même être employé en français et en anglais avec un sens différent (mail et courrier). Il serait d'ailleurs presque tentant d'employer dans le langage courant des concepts techniques qui se trouvent être des idées avant d'être des mots anglais, bien que on puisse faire l'effort de s'en sortir à l'aide d'une périphrase. Mais quant au simple vocabulaire technique, faut-il vraiment inventer a posteriori de mauvaises francisations quand le terme anglais est adopté par la communauté, employé et à employer dans les publications ? Serait-il vraiment raisonnable de concevoir que les étudiants apprennent en cours un vocabulaire inemployé dans la plus grande partie de la communauté ?

Un autre problème est celui du juste mot : à quoi sert d'imposer un équivalent français s'il n'a que peu de rapport au sens courant, voire paraisse parfois inventé pour l'occasion ('crénelage', 'tampon de profondeur'). De plus, nombre de substantifs se traduisent par de lourdes périphrases (lancer - de - rayons).

J'ai donc choisi autant que possible d'écrire en français, bien sûr, tout en conservant le vocabulaire anglais à chaque fois qu'il me semblait plus éclairant. Afin de ménager les susceptibilités, je propose ici quelques traductions dont l'on pourrait convenir pour les principaux d'entre eux :

- *aliasing*: selon les circonstances moiré, crénelage, pointillé, clignotement... renvoie à la fois à la théorie du signal et aux divers phénomènes concrets, à la différence d'un mot comme 'anticrénelage'.
- *batch*: calculé en différé.
- *(motion) blur*: flou (de bougé)
- *bounding*: englobant (sphère, boîte).
- *bump (map), bumper[v]*: (carte, texture de) froissement.
- *bug*: erreur due à un logiciel ou un script (donc plus précis que 'erreur').
- *cache*: cache, par facilité (bloc note ?).
- *clipper[v]*: couper ce qui dépasse.
- *compositing*: montage, mélange d'images 2D.
- *flag*: indicateur ; surtout pas 'drapeau' !
- *grid*: grille 3D de 'caches' spatiaux.
- *hard*: en dur, câblé.
- *jittering, jitterer[v]*: agiter, perturber aléatoirement.
- *label*: label, par facilité (attribut, étiquette ?).
- *lense flare*: reflet sur l'optique.
- *map*: carte, table, texture selon le contexte (un peu plus précis).
- *mapping,X-map,mapper[v]*: recouvrement, tapissage. 'Projection' n'a pas vraiment la même connotation (plutôt parallèle), et 'placage' semble un peu rigide.
- *modeling*: spécification géométrique, modelage, plutôt que modélisation qui est l'activité de toute discipline scientifique.
- *proceedings*: comptes rendus, actes de conférence.
- *ray-tracing, ray-tracer*: lancer de rayons, en substantif. Mais comment distingue-t-on en

français de ray-casting ?

- *rendering, render*: rendu, mais trop connoté peinture (ou désagréablement dans le sens courant !) ce qui contraint à la lourde expression 'algorithme de rendu'.
- *mask*: masque, par facilité.
- *morphing*: transformation géométrique (plus vague), métamorphose.
- *parsing, parser[v], parser*: balayer, parcourir systématiquement (une structure linéaire).
- *patch*: morceau de surface (souvent à 4 côtés).
- *pattern*: exactement synonyme de 'motif' qui est tout aussi concis, c'est donc un cas (le seul de cette liste) dans lequel il n'y a aucune raison d'utiliser le mot anglais.
- *sampling, sample*: pour une fois, je lui préfère échantillonnage (resp. échantillon) qui a à la fois un sens technique et commun juste.
- *scaling*: changement de taille, agrandissement/réduction, zoom (!)... il n'y a pas d'expression concise ayant le même sens (homothétie est trop mathématique, connote l'isotropie, et induit en erreur pour une texture).
- *scan, scanner[v], scanner*: balayer, parcourir systématiquement (une structure surfacique, ou par extension, volumique).
- *script*: fichier de commandes.
- *shading*: à la rigueur illumination locale ou dégradé, mais surtout pas ombrage !
- *shell*: intraduisible autrement que par 'fenêtre d'où on lance des commandes interactivement'.
- *smooth*: pas anguleux (lisse, doux ou mou ne sont pas très éclairants).
- *soft*: selon le contexte, 'fait logiciellement' ou 'doux' (soft shadows).
- *spline*: courbe ou fonction paramétrique de degrés 3.
- *split*: subdiviser, scinder.
- *swapper[v]*: ralentir (à cause d'un suremploi de mémoire).
- *texturing*: habillage (moins précis).
- *timing*: séquençage (moins clair).
- *(quad)tree, (oc)tree*: intraduisible dans le contexte image ('arbre octal' connote peu l'aspect pavage).
- *Z-buffer*: intraduisible, et sûrement pas littéralement par 'tampon de profondeur' ! A la rigueur moyennant généralisation, 'rendu projectif'. Même problème pour A-buffer.
- *(pi)xel, (vo)xel, (te)xel*: intraduisibles sauf par périphrase.

Annexe B

Le contexte de la discipline ‘Synthèse d’images’

B.1 Présentation sociologique

B.1.1 Variété d’aspects

On trouve de la synthèse d’images dans beaucoup de domaines, mais sous des formes très différentes. Selon les domaines où on la rencontre, la synthèse est attachée à différentes connotations :

- Aspect artistique : trucage de cinéma, publicité, habillage de chaîne, ‘flying logo’ (vidéo d’entreprise) ;
- Aspect ludique : jeux 3D, jeux sur CD-ROM (séquences précalculées), consoles de jeux, réalité virtuelle (telle qu’elle est pour l’instant) ;
- Aspect visualisation : CAO, visualisation scientifique, architecture, imagerie médicale.
- Aspect simulation : simulateurs de vol, de conduite, etc.

Délimitation du domaine

Il s’avère en fait assez difficile de fixer la limite entre application classique et synthèse d’image, qu’il s’agisse de qualifier une application (CAO) ou une étape d’un processus (spécification de l’éclairage). Cela tient notamment aux transversalités de la discipline, dont on parlera un peu plus loin : la synthèse emprunte à divers domaines, mais elle est également utilisée dans ces domaines en tant qu’outil. Ça serait simple si par synthèse on entendait uniquement l’aspect post-traitement qu’est le calcul du rendu (la limite est ainsi claire en visualisation scientifique : les données proviennent d’une discipline, leur visualisation est effectuée à l’aide d’une autre). Mais la synthèse, notamment dans le domaine audiovisuel, se compose d'autres aspects que le rendu, et comprend notamment le modelage et l'animation, qui empiètent sur le domaine des autres applications.

La distinction est particulièrement difficile à faire entre simulation et synthèse. Pour une production audiovisuelle on peut avoir besoin de simuler le comportement dynamique d'une voiture, la répartition de la lumière dans une cathédrale, la croissance d'un arbre, la formation d'un nuage ou la propagation d'une flamme, etc. Réciproquement, les physiciens simulent et visualisent la déformation des structures lors d'un crash-test, étudient l'écoulement d'un fluide afin de déterminer la traînée aérodynamique ou l'évacuation de la chaleur, les architectes - ou les metteurs en scène - évaluent l'éclairage dans un immeuble ou un décor en cours d'élaboration, etc.

Pour qualifier si une étude donnée relève de la synthèse, c'est plus la finalité que les techniques utilisées qui entrent en jeu : le calcul de l'éclairement ou du rayonnement thermique est il destiné à produire une jolie image, ou sert il a pronostiquer les conséquences des choix concernant le gros-œuvre d'un futur bâtiment ? Le modèle mécanique a-t-il pour but de faire se déplacer une voiture de façon réaliste, ou d'éprouver la qualité du système réel en cours de simulation ? Le fluide est-il animé pour évoquer un écoulement, ou est-on directement concerné par le champ de vitesses et le gradient de température ?

Comme on le voit, une même technique, ou plutôt un même sous-domaine (car le choix des techniques dépend des buts[1]) peut donner lieu à des développements dans des buts très différents, ce qui n'empêche pas que de temps à autre, les chercheurs d'un domaine applicatif s'approprient les techniques développées dans l'autre[2].

1. Ou du moins elle devraient, ça n'est hélas pas toujours le cas ; il semble parfois en effet difficile pour un chercheur de viser un objectif *puis* de choisir la méthode qui lui convient le mieux.

2. Cela n'est curieusement pas si fréquent, cette remarque étant valable pour beaucoup de disciplines où le cloisonnement se fait non seulement au niveau des domaines, mais même des sous-sous-domaines. De façon sporadique un 'pont' se fait transitoirement (entre la biologie et l'électronique, la neurologie et l'analyse d'image, la thermique et la synthèse...), parfois même par le biais d'un seul papier (dont on se demande comment il a pu résister à la sélection puis être remarqué, puisqu'il ne suit pas les canons de sa discipline). Le canal entre disciplines se ferme alors et un bouillonnement apparaît dans la nouvelle terre d'accueil de la technique importée, qui conduit souvent à bien des précipitations et des erreurs suivies de re-découvertes qui avaient pourtant été analysées un siècle plutôt dans la discipline initiale.

J'ai ainsi vécu les débuts de la radiosité en synthèse [Ney91], alors que j'avais travaillé sur la même méthode à EDF dans le cadre de l'évaluation d'échanges radiatifs au cours d'un incendie [Ney89]. Importée par Goral en 1984, la radiosité a rapidement explosé au point de phagocyter de larges portions des conférences de synthèse entre 1989 et 1993 (détournant alors l'attention des autres problèmes dans la sous-discipline du rendu), alors que les papiers que nous utilisions en thermique dataient des années 50-60, et s'étaient déjà posés le problème de la robustesse des intégrations que l'on a redécouvert en 88-89 en synthèse...

Le grand public

La synthèse d'images est fortement perçue par le grand public, mais principalement selon l'aspect ludique et artistique, d'où un préjugé de 'domaine pas très sérieux', pas 'scientifique', qui prévaut souvent même au sein de la communauté scientifique. Il faut dire que la seule observation des résultats de la synthèse d'images laisse difficilement percevoir la nature des techniques mises en œuvre pour l'obtenir, la technique étant d'autant plus transparente que le résultat semble naturel. Il faut vraiment un éclair de perspicacité pour se dire 'eh, la zone tamisée de cette pénombre est le résultat d'un calcul !', ou 'mais comment ont-ils modélisé les plis sur ce vêtement ?'. La technique qu'on peut 'lire' en premier en regardant une production, c'est celle des modélistes, des animateurs, des éclairagistes, des cadreurs, en un mot celle des utilisateurs.

Le grand public (via les média) attache la notion de puissance de l'informatique aux images de synthèse, domaine illustratif par excellence, à travers des applications qui sont en fait souvent de la simulation (crash-tests). Les média (et les futurologues[3]) brouillent totalement la notion du temps de calcul, entretenant la confusion entre réalité virtuelle et synthèse d'images réaliste, ainsi que du travail humain à produire (on pourrait croire que le monde de synthèse, parallèle au nôtre, vit tout seul à l'intérieur des ordinateurs, et qu'on l'observe à travers les écrans des stations !).

B.1.2 Motivation des acteurs

Quelles sont les motivations qui animent les chercheurs des différents sous-domaines de la synthèse d'image ? Il se trouve qu'en tant que discipline applicative elle est assez transversale (et ce doublement, par les techniques utilisées et par les applications), et recrute dans les autres branches scientifiques. Ainsi la synthèse - et les gens qui la font - proviennent des domaines de l'informatique, des mathématiques, de l'art, et de la physique, et cette origine demeure (il y a peu de 'synthésistes moyens' résultant d'une hypothétique fusion).

• En fonction de leur origine, les chercheurs s'intéressent prioritairement à l'esthétique du modelage ou de l'aspect, au réalisme de l'apparence ou au réalisme physique (photoréalisme, simulation mécanique), à l''esthétique mathématique' (NURBS), aux trucages et bidouillages informatiques (le plus d'effets pour le moins de complexité possible)...

Chacune des sous-disciplines est donc séparée en fonction des valeurs associées à l'origine des chercheurs, avec parfois des prédominances. Ainsi le modeling s'intéresse aux surfaces paramétriques propices aux équations et aux outils interactifs de modelage intuitifs, l'animation simule la dynamique mais comprend aussi le key-framing et le morphing, le rendu local traite des BRDF comme des textures de bruits de Perlin, le rendu global s'attache soit à la vitesse d'affichage soit au réalisme de l'image.

3. Tant les amoureux des objets nomades que ceux qui annoncent l'invasion des clones virtuels... C'est eux hélas qui font cependant office d'experts vis à vis de la presse et même des pouvoirs publics...

• Selon leur centre d'intérêt, leur origine et leur tempérament, les acteurs s'attaquent au photoréalisme (global ou local), à de nouveaux modèles géométriques (mathématiques ou exotiques), au temps réel (interactif, réalité virtuelle), à la simulation (dynamique ou mécanique), aux applications industrielles (CAO, éclairage, réalité virtuelle, production audiovisuelle, simulation), à l'efficacité (hardware, parallélisme), à l'interface utilisateur, aux grands problèmes dont nous parlons plus loin...

B.1.3 Transversalités

Comme on l'a dit, la synthèse procède de la rencontre (qui n'est pas une fusion) de l'informatique, des mathématiques, de l'art et de la physique. Les liens restent forts avec ces domaines, ainsi qu'avec le développement du hardware et l'industrie (CAO, la future réalité virtuelle[4], les utilisateurs supposés).

En matière d'informatique, on trouve des liens avec l'analyse numérique, le développement des langages, l'algorithmique, l'ergonomie des interfaces. Certaines disciplines algorithmiques sont particulièrement proches : les bases de données, l'infographie, le traitement d'image, les langages, la cognition. Ces liens se traduisent ou s'expliquent par l'origine des acteurs, par les compétences nécessaires au domaine, et par les objectifs applicatifs visés.

4. si elle réussit à émerger hors de son cadre ludique actuel. On peut remarquer que le succès applicatif d'un domaine scientifique a un rôle parfois néfaste sur son développement :

- L'injection de capitaux dans un domaine encore en développement accroît sa démographie plus vite que son évolution scientifique, ce qui produit l'effet du trempage vis à vis du recuit (en métallurgie) : le domaine se segmente en sous-domaines sous la pression du flux de travaux et de publications, avant que soit établie la pertinence de ce découpage (typiquement en analyse d'image entre segmentation, reconstruction et reconnaissance), et empêchant la persistance d'une vue globale. Cela commence à être le cas pour la réalité virtuelle, où tout l'effort semble porter sur le débit en polygones des hardwares graphiques, avant qu'ait été vraiment établie la pertinence des modèles polyédriques pour cet usage.
- Quand un domaine applicatif qui surgit d'une discipline scientifique explose économiquement, comme ça commence à être le cas de la réalité virtuelle, et notamment de sa tendance ludique (premier domaine abordé), le développement dû à l'apport de capitaux et à la demande industrielle réoriente le pôle d'intérêt de toute la discipline, par la démographie des chercheurs d'abord, mais aussi par la phagocytose des moyens de publication et par la nature des sources de financement (noter ainsi l'engouement soudain des laboratoires pour l'écologie et l'étude de la terre, lesquels sont souvent ceux qui vivaient des financements de l'armée peu auparavant).
- Les promesses applicatives prématurées peuvent engendrer une déception suivie d'un désinvestissement qui peut conduire à l'extinction de la branche : ça a été le cas pour les réseaux de neurones dans l'après-guerre qui ont connu une seconde naissance ces dernières années, c'est encore le cas pour l'IA et la vision artificielle.

B.1.4 Débouchés - importance économique

En réalité, l'importance économique est extrêmement faible en valeur aujourd'hui, et plus encore en Europe, par rapport à la place qu'occupe la discipline dans le discours des média, et même par rapport aux effectifs de la recherche. Depuis les origines de la discipline, on observe une foi déraisonnée en l'avenir, impression soutenue par la pression médiatique qui promet des lendemains riches en applications (comme on l'a remarqué plus haut, le fait qu'*une* application comme la réalité virtuelle décolle actuellement n'est pourtant pas forcément de bonne augure pour la discipline). L'idée de synthèse intéresse, les applications sont *potentiellement* intéressantes, mais il n'y a que très peu de nécessité industrielle réelle, donc peu de demande réelle de produits commerciaux.
Plusieurs leurres ont ainsi frappé l'industrie de la synthèse :

- Le marché de la visualisation scientifique, pour laquelle il y a une réelle demande, sauf que celle-ci est non-solvable (laboratoires).
- Le marché de la CAO[5] pour lequel la visualisation est une nécessité mais uniquement en fin de chaîne ou pour les besoins marketing (les dessinateurs 'voient' bien mieux avec des plans, même s'ils se déclarent séduits par le gadget de la prévisualisation réaliste).
- Le marché de l'architecture, pour lequel il est vrai que la production de maquettes est vitale au vu des mécanismes d'attribution de marchés, mais qui est essentiellement composé - du moins en France - par de très petits cabinets avec peu de moyens et peu de compétences informatiques.

En réalité l'avenir économique de la synthèse est incertain (hors du ludique, et de quelques niches), ou alors il se fera dans sa phagocytose par quelques applications spécifiques (réalité virtuelle, web...), ce qui entraînera un désinvestissement de larges pans de la variété de ses techniques.

Les débouchés actuels sont pour l'instant l'audiovisuel (cinéma, publicité, flying logo, jeux), et dans une moindre mesure la CAO, l'architecture, le design, la visualisation médicale, la simulation (vol, conduite, réalité virtuelle, jeux), et la visualisation scientifique.

Au niveau collaborations industrielles, la recherche en synthèse d'images est par nature très appliquée, et l'on y raisonne souvent en terme d'utilisateur et de plateforme logicielle. Il est d'ailleurs remarquable que la grand messe annuelle de la synthèse, la conférence-exposition Siggraph, est à la fois le passage obligé des chercheurs et des industriels. Mais réciproquement le très petit monde industriel de la synthèse a une culture assez scientifique (il accède notamment aux proceedings des conférences), et se passe souvent de collaboration directe : il y a couramment transfert de connaissance et de personnel entre la recherche et l'industrie, mais souvent sans retour financier pour les laboratoires !

5. Certaines sociétés avaient fait le pari dans les années 80 de l'inversion d'importance entre le débouché audiovisuel et celui de la CAO, imaginant dans un futur proche un poste de visualisation par poste de CAO...

B.2 Présentation technique

On se place ici essentiellement dans le cadre des applications audiovisuelles (mais pas exclusivement).

B.2.1 Composantes d'un système

Une chaîne de synthèse d'images comprend habituellement cinq phases :
- la modélisation, dont le but est de modeler la forme des objets qui composeront la scène et de placer les sources de lumière,
- l'animation, qui sert à spécifier le mouvement ou le comportement des objets,
- l'habillage, qui permet de décrire l'aspect de la matière de la surface des objets,
- le rendu, qui effectue en batch le calcul du rendu à partir des données spécifiées dans les trois phases précédentes (cf Annexe C),
- la postproduction, qui s'occupe du montage, des effets spéciaux, et de l'incrustation de certains objets exotiques traités en dehors de la chaîne classique.

B.2.2 Les valeurs

Bien que s'attaquant à des problèmes généraux et abstraits (c'est à dire pas directement des techniques de bas niveau), la synthèse est une discipline scientifique intrinsèquement appliquée, où la plupart des acteurs pensent en terme d'utilisateur ou d'utilisation. La taille des problèmes fait que l'on peut souvent dans le temps d'une thèse traiter un sujet, des spécifications à la réalisation effective. D'où un sens concret, une perception relativement globale, et pas mal de réalisme par rapport aux contraintes industrielles[6] (ce qui est plutôt rare en matière de recherche !). Mais il y a parfois des dérives, quand une équipe est trop ciblée et son personnel trop monolithique. Les dérives portent par exemple sur le traitement d'un seul aspect des objets manipulés[7], ou au détriment de l'intuitivité des paramètres de contrôle ou de la généricité des modèles (morphing de volumes), ou quant au temps de calcul (modèles photoréalistes sans soucis d'efficacité), etc.

6. Dès lors, quel dommage que la demande industrielle effective soit si faible !

7. Par exemple le seul aspect mathématique des surfaces, au détriment des autres informations qui y sont attachées dans la suite du traitement, comme les coordonnées texture. Les surfaces implicites ont par exemple connu un grand succès parmi les chercheurs, alors qu'il est très difficile d'y attacher des informations (à moins de trianguler). Le problème ne semble pas susciter un grand intérêt pour l'instant [Ped95].

Voici quelques unes des principales valeurs qui guident les acteurs du domaine :

Interactivité

Cela concerne le modelage et l'animation, mais aussi l'habillage.

L'utilisateur est omniprésent en synthèse (sauf dans les dérives vues plus haut). Cette valeur concerne le choix des paramètres manipulables, la rapidité, l'ergonomie de l'interface utilisateur, et la conception des outils interactifs de manipulation : pâte a modeler numérique, précision CAO, modèles physique, construction mécano (CSG, surfaces raccord), modélisation sous contraintes, spécification sémantique, acquisition de la forme d'objets réels...

Réalisme

On a abordé ce sujet dans l'introduction et le premier chapitre (1.1). On a vu que ce terme est en fait très vague, et qu'il faudrait plutôt parler *des* réalismes. Ceux-ci comprennent le photoréalisme (image capable de passer pour du réel), lequel comporte un aspect local (physique des matériaux) et global (physique du transport radiatif), la vraisemblance du mouvement, la course aux phénomènes physiques pour le rendu (irisation, arc en ciel...), et aux modèles dynamiques et comportementaux pour l'animation.

On inclut difficilement la fidélité de la modélisation dans le réalisme (à part peut-être pour des modèles spécifiques comme pour le visage), mais il me semble important de prendre en compte le réalisme que confère la complexité géométrique (après tout, c'est bien l'objet de ce mémoire !).

On peut donc dégager quatre aspects principaux :

- le réalisme photométrique,
- le réalisme par l'animation,
- le réalisme physique[8],
- le réalisme par la complexité.

Temps réel

Cette branche de la recherche est complètement séparée du rendu tel qu'il est traité en production de séquences, et concerne plutôt la réalité virtuelle et les simulateurs. (A une nuance près : les interactifs de modélisation et d'animation utilisent également des visualisations, qui doivent tourner en 'temps interactif').

8. i.e. dont les résultats sont comparables point à point à des données réelles. Cette précision est utile car le terme de 'modèle physique' est fortement galvaudé dans la discipline, au point de signifier parfois 'simulation par interactions locales' (particules, liaisons masse-ressort) par opposition à 'solution globale analytique'. A noter que ces comportements locaux sont souvent réglés... par une approche empirique !

Par le lien avec son domaine applicatif, il est remarquable que la synthèse d'images est une discipline où l'on a en général une notion réaliste des implications des choix techniques (les algorithmes temps réel ne sont pas les algorithmes lents actuels tournant sur les machines de demain !).

La contrainte de temps réel conduit à des modèles différents (Z-buffer et ses extensions pour obtenir ombres et reflets, systèmes de particules), suggère une hiérarchisation des constructions par importance (niveaux de détails, stratégie de calcul en radiosité), suppose de gérer la synchronisation et le timing (visualisation sans double buffer, réactivité), et favorise le développement de solutions en hardware.

Efficacité d'implémentation

Cela concerne avant tout le rendu.
En synthèse, on porte une attention particulière au temps de calcul, qui est même l'un des critères d'évaluation d'un algorithme, car il réside l'idée que les travaux doivent conduire à un produit final utilisable.

Ceci se traduit par un goût certain pour la programmation à bas niveau (au moins en rendu projectif et en ray-tracing) : choix du langage C aisément optimisable, gestion de la mémoire, écriture d'algorithmes compacts, relativement peu d'usage des techniques objet [9], optimisations poussées, parallélisme.

Cet état d'esprit a cependant des inconvénients qui ont mis en péril un jour ou l'autre les diverses entreprises du domaine : les choix sont liés à une technique qui évolue, ils sont souvent adaptés à une architecture particulière, ils nuisent à la maintenabilité des logiciels [10] et à la réutilisabilité des développements. De plus, cette forte représentation des contingences informatiques peut occasionner une autre dérive, la prise en compte des aspects informatiques avant la dimension physique ou conceptuelle.

La conception d'un logiciel de rendu pose un problème de structuration des données : une application de synthèse d'images, c'est peut-être surtout [11] une structure de donnée, avant

9. On peut difficilement opter pour l'usage des entiers plutôt que des flottants et des pointeurs plutôt que des offsets, et en même temps programmer en orienté-objet par envoi de messages ! Il faut cependant noter qu'avec le progrès des processeurs (notamment des FPU et des caches), l'arbitrage est en train de changer de côté quant à ces choix.

10. Songer que dans ces entreprises, il faut simultanément concevoir une future version novatrice *(n+1).0*, développer la prochaine version *n.(m+1)*, débugguer l'actuelle version *n.m*, et maintenir la ou les versions précédentes *n.0* à *n.(m-1)* ! Dès lors, la pérennité logicielle est essentielle, alors que ces sociétés jeunes et à la croissance trop rapide développent dans l'urgence au péril de leur vie (vers 1990, plusieurs ont frôlé la débâcle quand il leur a fallu consacrer trop d'énergie à produire une *(n+1).0*).

11. Au moins pour ce qui concerne le rendu. Pour le modeling et l'animation, on a vu que l'interactif était également très important.

d'être une liste de fonctionnalités : le modeling et l'animation ont pour but de constituer cette BDD de façon aussi intuitive que possible, le rendu a pour fonction de produire la séquence d'image qui découle de l'interaction de ces données. Comme de plus cette structure doit avoir des propriétés différentes selon la phase de la chaîne de production, on peut se demander s'il n'est pas fatal que les développeurs doivent attaquer le problème à un niveau assez bas.

B.2.3 Les grands problèmes

Il existe dans le champ de la synthèse d'images de nombreux grands problèmes ouverts, théoriques ou concrets. Hélas, cette discipline connaît une grande sensibilité aux modes : par exemple le rendu est actuellement essentiellement focalisé sur la radiosité, l'animation a longtemps collé au keyframe... Nous allons citer quelques-uns de ces grands problèmes, dont plusieurs ne suscitent que peu de recherches[12].

Placage des textures

Ce n'est plus à la mode, bien que le problème général ne soit toujours pas résolu[13]. On ne sait toujours pas bien habiller une surface à partir d'échantillons plans [BVI91, MYV93, Mai92]. Cela pose des problèmes de distorsion, de raccords, de délimitation [Ney96b], de glissement à la surface lors l'animation, d'adaptation automatique au contexte (contraintes sémantique ou topologiques).

Peu de solutions alternatives réalistes sont proposées quant à la génération 'sur site', directement sur la surface des objets. Pourtant dans le monde réel, le placage de textures n'intervient que dans certains cas très particuliers, sur des surfaces par construction développables, essentiellement le tissus et la tapisserie.

- Les textures 3D [Per85, Lew89] sont pratiquement la seule méthode alternative répandue, mais ne concernent que les matériaux dont l'aspect de surface est dû au motif présent dans la masse (marbre, bois).
- Des textures biologiques comme la peau, les zébrures ou la fourrure se produisent par interactions chimiques [WK91, Tur91, FLCB95] de proximité lors du développent, et se passent de coordonnées textures !
- On peut également concevoir de générer une texture comme si on la peignait sur un objet existant, sans placage [HH90].
- Enfin, on peut décrire localement la texture, par exemple par ses moments statistiques [GdM85, Mai92], afin de couvrir la surface de proche en proche.

12. Mais après tout, c'est peut-être pour ça qu'on peut les qualifier de problèmes !

13. Il est d'ailleurs mal posé, mais pourtant indispensable au point qu'on peut se demander comment font les graphistes pour s'en sortir.

Niveaux de détails

Comme on l'a vu au premier chapitre, ce domaine commence à être traité, d'autant plus que l'enjeu pour la réalité virtuelle est grand puisque le problème peut devenir bloquant pour les BDD colossales issues de la CAO. Mais pour l'instant, la seule voie pratiquement semble être l'approche décimation polygonale (couplée avec la course à la puissance du hardware).

Comme on le verra plus loin dans le cadre de l'unification des modèles, et comme on l'a dit au premier chapitre, il faudrait pouvoir passer continûment d'une représentation à une autre [Kaj85], pour utiliser la plus adaptée à l'échelle courante (et nos travaux se situent, entre autres, dans ce contexte).

Les réalismes

On a cité le réalisme parmi les valeurs des synthésistes, les diverses formes concernant le photoréalisme, le comportement et mouvement, la complexité, la fidélité physique (calibration des valeurs numériques par rapport aux données réelles). Les deux premières modalités sont à la mode, les deux autres relativement négligées.

Unifications

Des modèles différents prétendent représenter le même monde selon des contextes différents. Le changement des conditions étant a priori progressif, il faut pouvoir passer continûment d'une représentation à l'autre afin d'utiliser le plus efficace sans transition visible. De plus, chaque modèle possède des spécificités propres qui le rendent particulièrement commode dans certaines circonstances et inutilisable dans d'autres, il serait donc avantageux de coupler les méthodes.

On gagnerait donc à unifier les modèles et représentations dans chaque domaine, et à créer des synergies entre domaines mitoyens : le keyframe et la dynamique, la modélisation et l'animation, la modélisation interactive et déclarative (et toutes les autres modes de modélisation), l'animation et la simulation, les trois grands algorithmes rendus...
Cela concerne également la problématique des niveaux de détail (entre représentations), la simulation comportementale des foules, le mélange de la synthèse et des images réelles [JNP+95, JMN+96]...

Performances et fiabilité

Lors du développement logiciel, on est confronté à une dialectique efficacité/pérennité. Il faut donc choisir des compromis pour les diverses alternatives :

modularité vs performances, pérennité vs ouverture, hardware vs parallélisme, contrôle continu pour l'utilisateur du coût en temps et en mémoire, respect de la chaîne de la couleur...

Annexe C

Le rendu, genèse des images

Il n'est pas question ici d'aborder le rendu de manière exhaustive, mais juste de re-situer les divers éléments les uns par rapport aux autres.

C.1 Généralités

L'existence de la synthèse d'image a le mérite de susciter l'attention sur la physique de la lumière ainsi que sur la nature et la mesure de la vision humaine : Que *voit*-on ? à travers notre capteur, l'œil, qui floute, intègre, échantillonne, trie, interprète, on perçoit la lumière (i.e. une gamme de radiations électro-magnétiques) que nous réfléchit la surface des objets (en gros), en fonction de la propriété des sources, du milieu où baigne la scène, des matériaux, des états de surface.

Le problème de la synthèse semble alors simple : il s'agit de simuler l'optique physique tout en exploitant la relative myopie de l'œil[1], ce qui semble être un problème bien posé. Cependant on s'apercevra assez vite que le parallélisme de la nature, par exemple quand il s'agit d'échanges énergétiques, pose quelques problèmes calculatoires. La limite des connaissances et du matériel utilisé pour la visualisation est également bien réelle. Malgré tout, cela pose un premier aspect rencontré en synthèse, le **principe de simulation**. Il faut cepen-

1. Rappelons ces limites de notre capteur : perception des couleurs sur un octave de fréquences, capteurs de fréquences à 3 degrés de liberté alors que la plage est continue, perception inégale des couleurs (le rouge est perçu 2 fois mieux que le vert et 6 fois plus que le bleu), dynamique bien plus faible pour la chrominance que pour la luminance, et de toute façon insignifiante par rapport à la perception du son (qui tolère une plage de 10^5), optique réticulée et imprécise, alimentation sanguine de la rétine du côté exposé à la lumière (on se demande à quoi pensait le concepteur), tâche aveugle, résolution concentrée sur un faible angle solide (la taille d'une orange à bout de bras), persistance rétinienne...

(Heureusement qu'un système nerveux redresse tout ça au point que l'on ait peu conscience de ces limites, ce qui offre au moins l'avantage de prouver que la vision est affaire de logiciel plus que de capteur.)

dant relativiser : si l'on considère que le but de la synthèse est juste de produire des images crédibles, et encore, alors il faut plutôt se baser sur des critères qualitatifs ou esthétiques moins contraignants. Cela donne une première dialectique **art/réalisme**.

Un autre problème vient de ce qu'en synthèse, ce n'est pas le monde qui se projette sur le capteur mais le 'capteur' algorithmique qui interroge le monde (la scène). C'est le **principe d'échantillonnage**, ramenant au domaine du traitement du signal, et qui engendre les problèmes d'**aliasing** (cf C.3) que l'on trouve à de nombreux niveaux (bords et lignes en escalier, moiré des textures, clignotements à l'animation, éclairement en plaid en radiosité...).

Contrairement à ce qu'on a suggéré ici, la première tâche à laquelle s'est consacrée la synthèse n'est pas le réalisme mais la simple visualisation, tâche qui reste bien sûr d'actualité. Et pour chacun des deux termes de la dialectique **photoréalisme/visualisation**, les diverses phases modélisation, animation, habillage et rendu se comprennent différemment.

Dans ce dernier cas on peut toujours rétorquer a posteriori que la visualisation est une approximation du photoréalisme. Il y a cependant un peu de vrai dans cette remarque, auquel cas cela signifie que les notions physiques ont d'abord subi une **approximation informatique** aveugle avant que l'on comprenne les grandeur et simplifie en connaissance de cause. Ce problème est fréquemment rencontré, et relève de la dialectique **informatique/science** (en un mot la bidouille vs la rigueur). Cela a notamment joué en matière de modèles d'illumination (Phong), propices a l'empirisme informatique.

C.2 Principes

Comme on l'a dit au début de la section précédente, on *voit* la lumière des sources lumineuses réfléchie par les objets qui nous entourent. Il y a donc trois grandes modalités qui interviennent dans le processus :

- le transport de la lumière à travers l'espace, des sources aux objets, entre les objets, puis des objets à l'œil, c'est l'**illumination globale**,
- l'interaction de la lumière et du matériau à la surface des objets, c'est l'**illumination locale**, (on peut y ajouter l'interaction de la lumière avec le milieu ambiant),
- la constitution de ce matériau et de ses variations, définie par la **matière**, dont les variations sont codées par les **textures**.

C.2.1 Illumination globale

Cette modalité régule le transport énergétique, dont l'un des effets est prosaïquement la détermination des parties cachées. Dans les méthodes les plus simples, c'est directement la fonctionnalité que l'on vise, en faisant l'impasse sur le transport des sources aux objets qui pourrait pourtant être occlus ; dans les autres c'est une conséquence de la simulation.

Rendus projectifs

Ces méthodes simples ont pour premier objet la visualisation, aussi rapide que possible, basée sur la projection des facettes sur l'écran en conjonction avec la détermination des parties cachées. Ce sont les algorithmes du peintre et du Z-buffer (lequel se programme très bien en hard, ce qui permet d'atteindre le temps réel, soit 25 images par seconde). L'approche scan-line du tracé des facettes permet de rendre le calcul incrémental.

Le A-buffer, plus raffiné, tend à poser le rendu projectif comme solution acceptable pour le réalisme, en traitant la transparence et l'anti-aliasing. A noter que des extensions permettent d'obtenir au moins approximativement des ombres et des reflets.

Rendus par échantillonnage

Il s'agit du célèbre ray-tracing, dont le principe simplissime est d'échantillonner et de remonter les rayons qui arrivent à l'œil en appliquant les lois de Descartes, et des extensions au ray-tracing :

- le cône-tracing, qui règle le problème de l'aliasing auquel le ray-tracing est très sensible,
- le ray-tracing distribué, qui simule des distributions de rayons donc des échanges non strictement spéculaires.

Le calcul de l'ombrage est effectué en lançant un rayon de l'objet vers les sources, ce qui a l'inconvénient de ne pas tolérer de phénomène optique autre que l'occlusion pour la lumière incidente (alors que les rayons principaux peuvent subir réfraction et diffusion). La signature de cette méthode est un 'réalisme rutilant', présentant typiquement des reflets et des réflections multiples sur une carrosserie.

Un avantage de la méthode, outre la simplicité de programmation (qui se paie par une moindre efficacité), est la grande richesse géométrique qu'elle autorise : tout modèle est gérable à partir du moment où l'on sait déterminer l'intersection avec un rayon et la normale en un point. Notons que la plupart des logiciels du commerce utilisent aujourd'hui des rendus intégrés, où la première passe se fait en A-buffer, et où on ne lance en seconde passe des rayons secondaires (directs ou d'ombrage) que pour certains objets désignés par l'utilisateur.

Rendus par équilibre énergétique

L'équilibrage énergétique concerne avant tout les scènes d'intérieur, et produit donc un 'réalisme feutré' difficile à obtenir avec les méthodes précédentes. Une approche partielle à base de ray-tracing a néanmoins été proposée dans ce sens, émettant des rayons 'directs' depuis les sources lumineuses en plus des rayons remontés depuis l'œil (elle avait au moins l'avantage d'être intégrable aux autres rendus).

Mais la simulation de l'équilibre radiatif se fait essentiellement par les méthodes de radiosité, pour lesquelles une première passe se charge de 'faire rayonner' chaque facette éclairée vers les autres jusqu'à atteindre l'équilibre, puis une seconde passe de rendu classique se charge de 'lire' l'énergie à la surface des facettes comme si c'était une couleur. Le coût de la première passe fait que cette méthode est aujourd'hui encore très chère.

C.2.2 Illumination locale

L'interaction locale de la lumière et de la matière fait intervenir le comportement optique des matériaux, lequel incombe en partie aux détails microscopiques de la surface. Le modèle le plus simple, celui de Gouraud, simule une surface lambertienne (la lumière incidente est reçue de façon homogène au prorata de la surface apparente, et elle est réémise de façon isotrope). Le modèle de Phong ajoute un terme spéculaire, simulant la tache spéculaire floue qui entoure le reflet de la source lumineuse à la surface. Des modèles plus précis (Cook-Torrance[CT82] et suivants [Sch94, HTSG91]), plus spécifiques (filtrage de bump [Fou92]), ou plus dédiés (pigments[GMN94], état de surface[ON94]) sont ensuite venus compléter la palette.

Ces modèles de matière comportent un certain nombre de paramètres, que l'utilisateur détermine en fonction de l'effet recherché. Mais ces paramètres sont souvent amenés à varier le long de la surface, ce qu'on appelle alors une **texture**. Cela est rendu gérable à l'utilisateur en passant généralement par une carte (des valeurs du paramètre) considérée comme un motif de base, et une fonction de mapping se chargeant d'appliquer les copies de la carte sur les surfaces.

C.2.3 Les textures

Comme on l'a vu dans les premiers chapitres, celles-ci permettent de simuler un matériau, un phénomène physique, un état de surface, une micro-géométrie (elles participent donc à la problématique des niveaux de détails), en habillant la surface avec des valeurs explicites (via un modèle d'illumination local). Le paramètre le plus simple auquel on pense est la couleur du matériau, ou sa transparence, mais tous peuvent varier par voie de texturage. Les

textures poursuivent donc un triple objectif de **simplicité** pour l'utilisateur, de **réalisme** via l'enrichissement de l'image, et d'**efficacité** dans le calcul du rendu.
La spécification des textures comporte trois aspects :

- choix de la **variable** texturée :
  couleur, transparence, paramètres de matière (ambiant, diffus, spéculaire, rugosité), ombre, reflets, normales, élévation, commutation entre textures...
- méthode de **plaquage** :
  soit en utilisant le paramétrage de la surface, soit via une projection plane, cylindrique ou sphérique, soit par une optimisation de la déformation (atlas), soit par une définition intrinsèque (textures 3D, textures topologiques, cartes de réflection). Pour isoler la définition du placage du rendu de la texture, on associe des coordonnées texture à un point de la surface, qui seront déterminées par les méthodes de placage. Ceci permet en outre de déformer une surface *après* texturation.
- **nature** de la texture :
  la valeur des paramètres de la matière peut être indiquée par une image, des moments statistiques, ou une procédure (bruit contrôlé, grammaires...).

Un aspect intéressant des textures est qu'elles permettent facilement de lutter contre l'aliasing. Lutter contre l'aliasing se fait en **intégrant** la lumière sur toute la surface du pixel plutôt que de se contenter d'échantillonner au centre. On peut ramener à la surface de l'objet les limites du pixel, et intégrer à ce niveau (ce qui économise le coût de lancer d'un rayon). Mieux encore, quand les paramètres de la texture apparaissent linéairement dans la loi de l'illumination locale, comme c'est le cas pour la couleur, on peut factoriser et ainsi se ramener à la seule intégrale du paramètre, laquelle peut être précalculée une fois pour toutes. Les méthodes mip-map[Wil83] et SAT[Cro84] mettent en place de telles tabulations, la première précalculant des zones carrées de diverses tailles (qui correspondent à l'image aux diverses résolutions), la seconde précalculant la primitive de la texture (intégrale sur un rectangle qui va du coin à la position courante). Notons que le mip-map est programmé en hard sur les cartes graphiques puissantes, ce qui offre la possibilité d'une certaine richesse en temps réel.

C.3 L'aliasing

L'aliasing est un phénomène qui apparaît à plusieurs stades du processus de synthèse :
- 'dents de scie' sur une ligne de crêtes discrétisée par le maillage,
- phénomène des 'roues de chariot' qui semblent tourner à l'envers à l'animation,
- clignotement des petits objets à l'animation,
- limites en dents de scie des ombres propres,
- lignes et bords des objets en escalier,
- moirés dans la texture,
- éclairement 'en plaid' d'une surface en radiosité.

Il provient généralement de ce qu'on évalue un attribut d'un *continuum*, par exemple une surface, à partir de la valeur en un seul *point*. La surface peut être selon le cas celle d'un pixel à l'écran, d'un pixel de texture, d'un pixel d'hémicube (en radiosité), ou un élément de surface sur un objet.

Il s'agit donc d'un problème d'échantillonnage (en considérant que l'algorithme de rendu est un capteur qui 'interroge' la scène), dont l'étude relève du traitement du signal. Cette discipline nous apprend à travers le théorème de Shannon que l'échantillonnage ne fait perdre de l'information que si la fréquence d'échantillonnage vaut moins du double de la fréquence maximum présente dans le signal. Si cette condition n'est pas respectée, les hautes fréquences se 'replient' sur les basses fréquences et font apparaître l'aliasing comme l'illustre la figure C.1.

L'une des méthodes d'anti-aliasing consiste précisément à prendre plusieurs échantillons pour chaque pixel et à en faire la moyenne, ce qui revient à augmenter la fréquence d'échantillonnage. On peut aussi se conformer au critère de Shannon en filtrant les données de manière à faire disparaître ses hautes fréquences (e.g. préfiltrage du mip-mapping). Un signal correctement échantillonné peut alors restituer le signal continu dont il est issu en appliquant un filtre de reconstruction.

Une autre façon de procéder consiste à abandonner l'échantillonnage pour tenir compte de toutes les valeurs : c'est ce qui est fait en A-buffer ou en cône-tracing, où l'on intègre tout ce qui est visible dans l'angle solide d'un pixel.

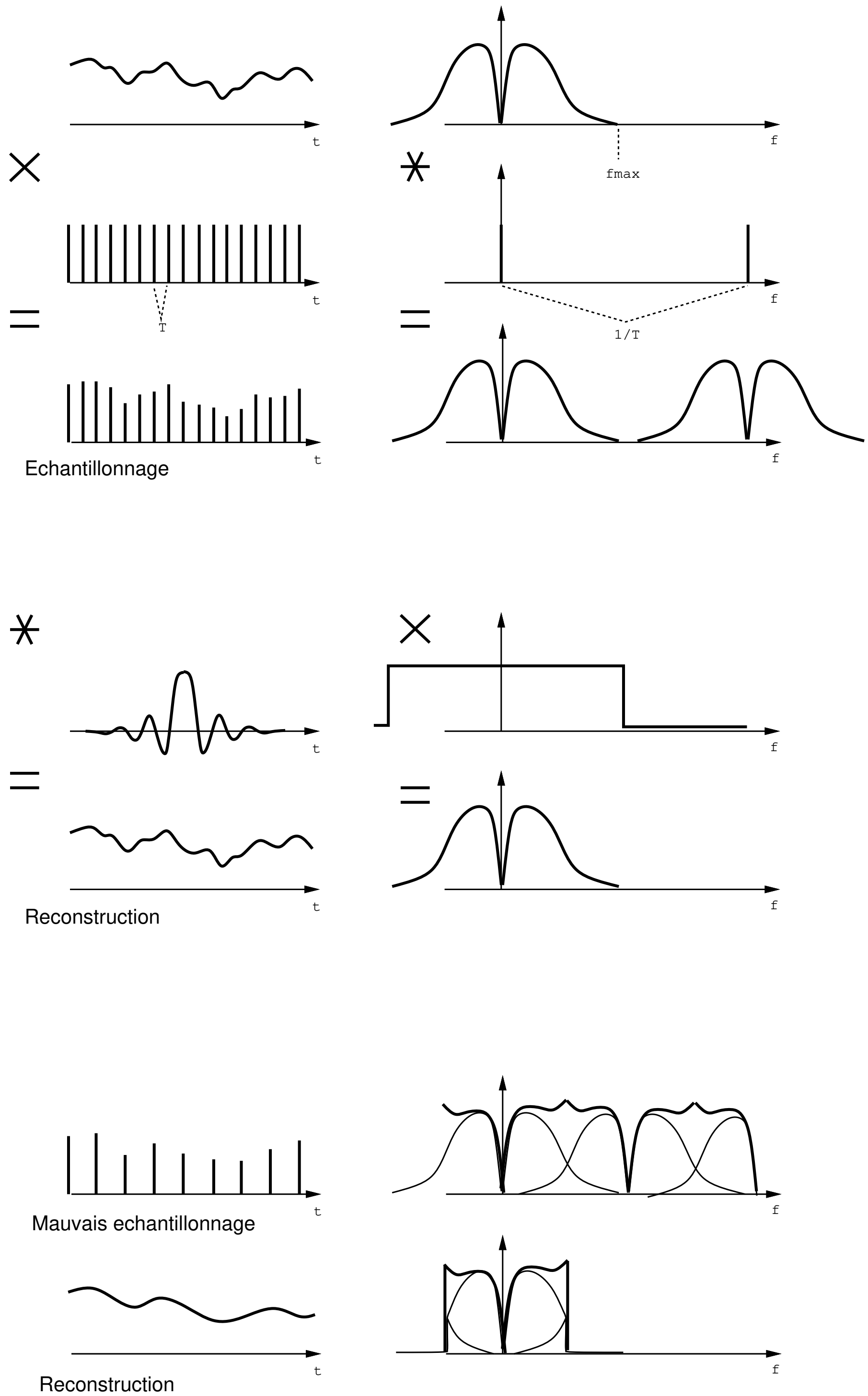


FIG. C.1 - *Echantillonnage et reconstruction d'un signal, vu dans l'espace canonique et dans l'espace de Fourier. Le signal échantillonné provient de la multiplication du signal continu par un peigne de Dirac. Le signal est reconstruit en appliquant un filtre passe-bas. Les deux graphiques en bas illustrent le repliement de spectre qui apparaît quand la fréquence d'échantillonnage n'est pas adaptée au signal.*

Annexe D

Détails techniques

Nous détaillons ici les algorithmes des principaux modules logiciels. Le programme compte au total 50 fichiers qui représentent plus de 19000 lignes de C, 700 fonctions en source et 200 fonctions en macro. Il n'est donc pas question ici d'être exhaustif, nous nous contenterons de donner les grandes lignes de fonctionnalités représentatives concernant la construction des texels et leur rendu. On pourra se faire une idée de l'ensemble en consultant le mode d'emploi en Annexe E.

la procédure **main** analyse les paramètres de la ligne de commande et lance la lecture du script (en fait, un premier script fixant les valeurs par défaut est lancé avant le script fourni par l'utilisateur).
script analyse le script qui décrit les texels, la scène, les matières, et les paramètres de rendu. Par le jeux des `INCLUDE`s, la fonction script peut se lancer elle même récursivement.

D.1 Analyse de script

Cette partie fait office d'interaction avec l'utilisateur, soit par le biais d'un fichier contenant les commandes, soit en prenant les commandes directement au clavier. La réalisation en est relativement classique. Chaque ligne brute est transformée successivement en commandes séparées :

- une première passe filtre les commentaires, concatène les extensions de lignes (une ligne terminant par '\' se poursuit à la ligne suivante), isole les commandes séparées par ':' ;
- une seconde passe remplace par leur valeur les variables signalées par préfixe, les expressions en polonais inversé signalées par '{' et '}' , et les caractères spéciaux (préfixés par \),
- une troisième passe reconnaît les variables non préfixées et élimine les caractères spéciaux qui ne servent qu'à la lisibilité.

A noter que quelques commandes utilisent le résultat de la seconde passe, notamment parce qu'elles ont besoin de paramètres comprenant des caractères alphabétiques ou spéciaux (noms de fichier, typiquement) qui seraient mal interprétés par la troisième passe. De plus un nom de variable doit être préservé s'il est en premier argument, pour les instructions d'affectations (`SET,ADD`...). On signale les commandes concernées en commençant leur nom par une majuscule.

Les commandes sont alors analysées par une grande série de comparaisons testant chaque mot du langage, et effectuant la tâche requise une fois le mot reconnu. Ces 276 commandes se distinguent en 5 catégories :

- les commandes de programmation, qui comprennent les instructions de contrôle (boucles, tests, includes...) et de calcul (arithmétique, hasard, splines...),
- les commandes décrivant la scène, c'est à dire l'ensemble des objets géométriques à la surface desquels on va pouvoir mapper les textures volumiques,
- les commandes servant à la construction des volumes de référence des textures volumiques,
- les commandes spécifiant les paramètres de rendu (caméra, lumières, matières, résolutions, options de calcul...)
- les commandes provoquant le calcul du rendu.

D.2 Construction géométrique

Les formes géométriques servant à construire un échantillon de texture volumique se déclinent en 3 catégories :

- les primitives géométriques simples (cube, sphère, cône, cylindre...) qui sont construites à partir de leur fonction de distance comme vu en section 4.1.1,
- les formes composées : groupes de primitives, fonctions implicites, L-systèmes, maillages,
- les fonctions volumiques vues en section 4.1.2 : hypertextures, images scanner.

D.2.1 apply_primit(voxel,r,n, dens,OBJ,O,A,P1,P2,dist(),norm())

C'est le cœur de la construction des primitives dans le volume de référence. On lui passe en paramètre le voxel-nœud auquel s'applique la construction récurrente, la taille r de ce voxel, le degré n de récursion maximum restant à faire, la densité à appliquer au tracé, le mode de réflectance OBJ, le centre de la primitive et les divers paramètres associés (axe, rayons), et les fonctions de distance et de gradient de distance (fournissant la normale locale).

Le calcul est fait en normalisant les données par rapport au voxel courant (le centre de ce nœud de l'octree s'exprime toujours (.5.5.5)). Le mode de réflectance OBJ détermine l'ellipsoïde qui encode la réflectance locale : si une fonction de normale est fournie on l'utilise,

sinon la forme de l'ellipsoïde est codée dans OBJ, ainsi que son orientation qui se fait soit radialement (OBJ>0), soit le long de l'axe (OBJ<0).
L'algorithme est le suivant :

```
invalide  voxel                ' pour indiquer que cette branche de
fibre  = (.5 .5 .5) - O        '      l'octree a été modifiée.
t = dist(fibre,r,A,P1,P2)
si (t<0) alors fin             ' voxel DANS l'objet
si (t>1) alors fin             ' voxel HORS de l'objet
' le voxel est sur la frontière, il faut subdiviser
si (n=0) alors                 ' on est au bout de la récursion
  si (OBJ<0) alors
    fibre = N
   sinon si (norm() existe) alors
    fibre = norm(fibre,r,A,P1,P2)
   sinon
    normalise fibre
  permutations éventuelles sur fibre    ' si une rotation est à effectuer
  fabrique_ellipsoïde (fibre,ABS(OBJ)) -> refl
  remplit_voxel (voxel, dens*t, refl)
  fin
sinon                              ' on n'est pas au bout de la récursion
  si (voxel n'a pas de fils) alors
    créer 8 fils à voxel
    invalide voxel
  pour chaque fils de voxel          ' on récure.
    O = 2* ( O - centre(fils) ) + (.5 .5 .5)
    apply_primit (fils,2*r,n-1, dens,OBJ,O,A,P1,P2,dist(),norm())
```

D.2.2 implicit

Les fonctions implicites correspondent à une surface isopotentielle, obtenue à partir d'un squelette composé de primitives point, segment, face triangulaire valuées. Comme dit en section 4.1.1, ce cas est traité comme les primitives simples en se ramenant à une fonction de distance et son gradient, avec une distance non euclidienne fonction du potentiel.

Une structure `implicit` contient une liste de types valant `POINT`, `SEGMENT` ou `FACE`, une liste de poids et une liste de points (qu'il faut considérer à l'unité, par paire ou par triplets selon le type).

La fonction *dist_implicit* fournie à *apply_primit* est alors :

```
pot = 0
pour chaque élément
  si (type=POINT) alors
    '  d2 = distance au point au carré
    P1 = (1/r)*fibre - point(1)           ' vect P1M
    d2 = norme(P1)^2
  si (type=SEGMENT) alors
    '  d2 = distance au segment au carré
    P1 = (1/r)*fibre - point(1)           ' vect P1M
    P2 = (1/r)*fibre - point(2)           ' vect P2M
    V = P2-P1
    si (P2.V1<=0) alors d2 = norme(P1)^2
    si (P2.V1>=0) alors d2 = norme(P2)^2
    sinon d2 = norme(P1)^2 - (P1.V1)^2/norme(V1)^2
  si (type=FACE) alors
    '  d2 = distance au triangle au carré
    ...
  pot = pot + poids^2/d2
  dist = 1/rac(pot) ou INFINI si pot<EPS              ' distance au squelette
  critère = 1/2 - (isopotentielle-dist) / (rac(3)/2)  ' 0 dehors, 1 dedans
```

D.2.3 hypertexture

L'hypertexture utilisée ici correspond à un bruit de Perlin dans tout l'espace, de profondeur de récursion donnée, de pseudo-fréquences sx, sy et sz, générant éventuellement un texel cyclique. Cette fonction incorpore en outre un terme de répulsion des parois, de champ d'action donné et concernant les parois demandées. La répulsion agit sur une distance correspondant au champ d'action, avec une intensité fonction du troisième degrés de la distance de telle sorte que la répulsion passe de 0 à 1 dans le champ d'action avec une dérivée nulle aux deux extrêmes (c'est donc $d^2(3-2d)$, de dérivée $6d(1-d)$).

L'algorithme de tracé de l'hypertexture dans le volume est le suivant :

```
pour tout point i,j,k du volume
  rep = répulsion des bords dans les 3 directions
  r_grad = gradient de la répulsion
```

```
bruit_Perlin(i*sx,j*sy,k*sz,options) -> (br, b_grad)
b_grad = b_grad * (sx,sy,sz)          ' grad( bruit(s.x) )
v = br*rep
grad = rep*b_grad+br*r_grad           ' grad(br*rep)
v = plateau (v,seuil0,seuil1)         ' seuillage haut/bas
si (v>0) alors                        ' il y a de la matière
  normalise grad
  permutations éventuelles sur grad   ' si une rotation est à effectuer
  fabrique_ellipsoïde (grad,ABS(OBJ)) -> refl
  remplit_voxel (voxel(i,j,k), v, refl)
```

D.3 Filtrage

Le filtrage s'effectue en partant des feuilles de l'octree et en remontant jusqu'à la racine. Lors du tracé, on a préalablement marqué les voxels parents, afin de limiter les traitements aux seules zones modifiées depuis la dernière passe filtrage-compression. A noter que le filtrage de 8 fils est un moyennage, similaire à l'addition d'ellipsoïdes décrite en 3.1.2.

ellip_filtre(voxel)

```
densT = somme des densités des 8 fils de voxel.
si densT=0 fin
dtot = 0 ,  Qtot = matrice nulle
pour chacun des 8 fils non vides de voxel
   dens = dens(fils[i])/densT
   extrait l'ellipsoïde du fils -> vecteurs R[1..3], rayons r[1..3]
   normalise les rayons  ' normalisation = PI.(r0r1+r1r2+r0r2)/3 ,  pour que
   ' tous les ellipsoïdes aient le même gabarit, défini par surf apparente moyenne
   construit la matrice Q de la forme quadratique associée
   Qtot = Qtot + dens*inv(Q)     ' ( en fait on construit directement Q^-1 )
   dtot = dtot + dens

trouve les valeurs  propres de Qtot -> r[1..3]
trouve les vecteurs propres de Qtot -> R[1..3]
r[i] = rac(r[i])
stocke_primitive(R1,r1/r0, R2,r2/r0)  ' stockage compact de l'ellipsoïde:
  ' on ne conserve que 2 vecteurs, le 3ieme etant orthogonal et de norme 1.
```

D.4 Rendu

la commande **rendu** du script appelle **saveview** qui lance le rendu ligne par ligne, éventuellement en parallèle.
scanray lance les rayons en chaque pixel de la ligne.
Le tracé d'un rayon est effectué par **intervox** que nous détaillons plus bas.
Quand une surface texturée est atteinte, la peau volumique est traversée par **crosstex** jusqu'à atteindre un texel.
Ce texel est traversé par **crossvox** que nous détaillons plus loin.
Le calcul de l'illumination locale est effectuée par **rayvox**, qui se charge d'envoyer un rayon d'ombrage avec intervox muni de la fonction de calcul local **shadevox**.
Le calcul de la réflectance est assuré par **ellip_refl** (traité plus bas).

D.4.1 intervox(O,D,lo,ouv,rendulocal(),L,T)→(L,T)

Il y a quatre type d'appel de la fonction de tracé de rayon :

- rayon 'couleur' (i.e. pas le rayon d'ombrage) venant d'en dehors de la peau volumique.
  Reconnu par le fait que l'indicateur *cur_face* soit négatif. Il faut alors intersecter la géométrie, ce qui est fait par **hitgeom**, de manière à obtenir un numéro de face et des coordonnées dans l'espace boîte (i.e. face + volume associé). On en tire également une abscisse l sur le rayon (distance parcourue).
- rayon couleur venant de l'intérieur de la peau volumique.
  Il n'y a jamais d'appel de ce type dans la mesure où toute la peau est traversée par une boucle dans **intervox**.
- rayon d'ombrage venant d'en dehors de la peau volumique.
  hitgeom fournit les informations face, l, coordonnées.
- rayon d'ombrage venant de l'intérieur de la peau volumique.
  Reconnu par le fait que l'indicateur *cur_face* contient le numéro de la face. L'indicateur *from_surf* signale si le rayon provient du sol ou du volume. La position O est exprimée en coordonnées boîte, avec des manipulations discrètes (ex: permutation des axes) qu'il faut décompter quand le rayon provient du volume. On convertit O en coordonnées scène.

Au sortir du prétraitement effectué en fonction du type d'appel, on se trouve avec O et D définissant le rayon en coordonnées scène, une distance l qui sert à calculer l'ouverture du rayon, un numéro de face, et des coordonnées boîte $\vec{u}$ sur celle-ci. L'algorithme est alors le suivant :

```
si point_d'impact existe
  point_de_vue = -D
  from_surf = 0
  si (position = sol ou couvercle) alors
    si (rayon couleur) alors
      convertit_voisinage           ' calcule jacobiens
      calcule l'ouverture
    face_refl()                     ' illumination de la surface
    si (opaque) alors out=-1        ' fin
    from_surf = 0
  si out=0 alors
   boucle
     out = outcube() -> (u0,l0)     ' point de sortie du texel
     si (sortie) alors fin boucle   ' sortie de la peau
     manipulations discrètes
     clip dans le texel
     si (vide) alors suite boucle   ' on est dans la marge
     boîte_to_texture(u,u0) -> (U,U0)
     si (rayon couleur)
       mémorise_position            ' U,U0,l,f
       convertit_voisinage          ' calcule jacobiens
       manipulations discrètes sur jacobiens
     cur_face = f
     calcule l'ouverture
     mati = matière(f)
     crosstex(U,U0,rendulocal())    ' traversée du texel
     si (opaque) alors fin boucle
     u0=u, l0=l, boîte_suivante -> (u,l,f)
   tant que (dans la peau)
   u=u0
   si (position = sol ou couvercle) alors
     face_refl()                    ' illumination de la surface
     si (opaque) alors out=-1       ' fin
     from_surf = 0
  restaure mati et point de vue     ' à la fin d'un rayon d'ombrage
  si (sortie) alors
    boîte_to_scene(u) -> O
```

```
  cur_face = -1
  intervox(O,D,l+l0)              ' si on sort de la peau, rayon suite
  cur_face = face
```

D.4.2 hitgeom(O,D)→(f,u,l)

Parmi les surfaces valides dont la sphère englobante intersecte le rayon, on effectue une traversée du *grid* subdivisant leur boîte englobante (cf section 4.2.2). Chaque cellule du *grid* contient les faces dont la boîte est au moins en partie dans le volume de la cellule (voir figure 4.7). On considère celles dont la sphère englobante intersecte le rayon. Parmi celles-ci, **hitface** calcule les vraies intersections du rayon avec les patches bilinéaires, en les parcourant dans l'ordre croissant des distances, et on garde le point d'intersection le plus proche (noter qu'il n'y a besoin de tester que les patches qui sont en contact avec l'extérieur, soit généralement deux par boîte).

D.4.3 hitface(O,D,M1,M2,M3,M4)→(u,l)

On teste si le rayon intersecte la sphère englobante de la face, et si le rayon croise les segments de la face à l'intérieur de celle-ci (pour un rayon (O,D) et un segment AB, le produit mixte $< OA, D, AB >$ est positif si le rayon passe 'à gauche' de AB ; on répète le test sur tout le contour décrit dans le sens trigonométrique). Si oui il faut vraiment calculer le point d'intersection. Si la face est un parallélogramme le problème est linéaire, sinon on a affaire à un système Q1 de deux équations à deux inconnues, qui se ramène à une équation du second degrés. Il faut en outre traiter les nombreux cas de dégénérescence.

D.4.4 face_refl(f,u,D,l0,ouv,L,T)→(L,T)

Il s'agit de calculer l'illumination locale sur une surface, ce qui donne l'algorithme :

```
mati = matière_sol(f)
eval_mati(f,u)                  ' calcul de la matière
si (rayon ombrage) alors
  T = T * transp
  restaure mati
  fin
eval autoombrage et shading
si (lumière visible) alors
  si (vers volume) alors
```

```
     O = u
     from_surf = 1
    sinon
     O = boîte_to_scène (u)
   intervox(O,lum,l0,shadevox)     ' rayon d'ombrage
 L = L + illumination
 T = T * transparence
 restaure mati
```

D.4.5 crosstex et crossvox

Crosstex traverse l'espace des texels entre les coordonnées de départ et d'arrivée fournies, de manière similaire à un tracé de droite : pour chaque cellule (correspondant à un texel) il faut déterminer le point ou le rayon sort de la cellule. On peut utiliser à ce niveau le tracé de rayon courbe vu section 4.3. **crossvox** est alors appelé pour effectuer la traversée de chaque texel.

crossvox est la procédure qui traverse l'octree récursivement, et appelle à chaque voxel de taille correspondant à l'ouverture du rayon la fonction de calcul du rendu local (**rayvox** ou **shadevox** selon que l'on traite un rayon couleur ou un rayon d'ombrage). Elle est relativement complexe dans la mesure où les traitements sont dérécursivés, et traitent de l'interpolation mip-map ainsi que des texels superposés en une seule passe. Par contre les calculs sont normalisés à la taille du voxel courant, ce qui permet en outre de travailler en virgule fixe. Voici le schéma du parcours dichotomique :

```
empiler  voxel=racine de l'octree, point d'entrée, point de sortie
dépiler un segment tant qu'il y en a
  si (niveau fin) alors  ' soit résolution utile atteinte, soit voxel feuille
    rendulocal() -> (L,T)               ' rendulocal() = rayvox ou shadevox
    si opaque alors fin boucle
  calcule les intersections du rayon avec les plans u=.5, v=.5, w=.5
  fait une liste triée par ordre de profondeur avec
  ces points, plus le point d'entrée et de sortie du voxel courant
  empile les segments correspondants dans l'ordre inverse
  segment suivant
```

La dérécursivation se fait en empilant les contextes manuellement ; le traitement de la superposition fait intervenir un vecteur de voxels ; l'interpolation mip-map utilise un système à états :

l'idée du mip-map est d'évaluer la texture au niveau plus grossier que la résolution idéale (*état 1*) et au niveau plus fin (*état 2*), et d'interpoler entre les deux valeurs (*état 3*). Dans notre parcours récursif, cela implique de différer les calculs, le stockage et le retour d'où la nécessité d'un système à états. La variable d'état `eval_outil` vaut 1 quand on doit passer à l'évaluation fine (*état 2*), -1 quand on doit clore un calcul (rayon stoppé), sinon 0 (*état 0* ou *état 1*). L'évaluation grossière est faite quand la taille du voxel courant est juste supérieure à l'ouverture du rayon ; on 'programme' alors l'évaluation fine en mettant `eval_outil` à 1 et en évaluant le coefficient d'interpolation `eval_a`, ainsi que la profondeur `eval_r` de la résolution fine juste inférieure à l'ouverture (moitié de la grossière). Aux tours de boucle suivants on sera donc à la résolution fine ($r =$ `eval_r`), puis quand les segments à résolution fine seront épuisés, la remontée de la récurrence s'amorcera avec $r >$ `eval_r`, ce qui signalera qu'on a terminé le calcul fin et qu'on peut faire l'interpolation (*état 3*).

Cela donne l'algorithme suivant (sans le traitement des texels superposés) :

```
empiler  voxel=racine de l'octree, point d'entrée, point de sortie
dépiler un segment tant qu'il y en a
  si (eval_outil non nul  et r>eval_r) alors
    ' on passe en ETAT3
    interpoler (eval_a, L,T,L0,T0)->(L,T)
    si (eval_outil=-1 et  calcul grossier opaque) alors fin boucle
    eval_outil = 0       ' retour à ETAT0

  si (niveau fin) alors  ' soit résolution fine atteinte, soit voxel feuille
    si (eval_outil=-1) segment suivant  ' on cloture ce segment
    calcul fin -> (L,T)                 ' rendulocal() = rayvox ou shadevox
    si opaque alors
      si (ETAT2) alors
        eval_outil=-1, segment suivant  ' on va passer en ETAT3 puis fin
       sinon
        fin boucle
    ' le prochain tour de boucle sera soit en ETAT0 (si on y est déjà),
    ' soit ETAT2 ou ETAT3 selon qu'il reste ou non des segments fins en pile.
  si (niveau juste grossier) alors   ' i.e. le suivant est < ouverture
    ' on passe d'ETAT0 à ETAT1
    eval_outil=1,  eval_r=r/2
    évalue eval_a                    ' pour l'interpolation à l'ETAT3
    calcul grossier -> (L0,T0)       ' rendulocal(). result mémorisé pour ETAT3
    , on passe en ETAT2
```

```
  calcule les intersections du rayon avec les plans u=.5, v=.5, w=.5
  fait une liste triée par ordre de profondeur avec
  ces points, plus le point d'entrée et de sortie du voxel courant
  empile les segments correspondants dans l'ordre inverse

  segment suivant

si (eval_outil=1 et r>eval_r) alors
  ' on entre en ETAT3
  interpoler
```

D.4.6 rendulocal()

Il s'agit de calculer l'illumination locale. **rayvox** est utilisé pour un rayon couleur, et **shadevox** pour un rayon d'ombrage.

Il faut tout d'abord évaluer la matière si elle est procédurale, et calculer la réflectance par **ellip_refl**. On doit également le faire pour un rayon d'ombrage si l'on accepte les matières procédurales agissant sur la transparence, ce qui est puissant mais coûteux, et si l'on souhaite tenir compte de l'anisotropie de la densité pour le calcul l'opacité.

Pour un rayon couleur, ellip_refl fournit l'énergie diffuse et spéculaire ainsi que l'opacité effective, reste à évaluer la présence de lumière en lançant un rayon d'ombrage par **intervox** muni de l'outil de rendu local **shadevox**. Il faut hélas pour cela récupérer la position en coordonnées boîte, ce qui nécessite de stocker préalablement les coordonnées texel, et de coûteusement les convertir (la conversion se fait par une méthode itérative). Reste alors à cumuler l'illumination pondérée par l'opacité, et à multiplier la transparence. La fin de la traversée est signalée quand un seuil d'opacité du rayon est atteint.

Cela conduit à l'algorithme suivant pour le calcul de l'illumination locale en un voxel :

rayvox(voxel,l,dl,Ltot,Ttot)→Ltot,Ttot

(*l*: distance sur le rayon. *dl*: portion du voxel traversée par le rayon.)

```
si (dens(voxel)<EPS) retourne 0       ' le voxel est vide: pas de contribution
trsp =  1-transparence(dens(voxel)*dl)' transparence(dens*dl) := 1-dens*dl
eval_mati()                           ' calcul de la matière courante MATI
trsp = 1-(1-MATI->transp)*(1-trsp)    ' module trsp en fonction de la matière
vis  = (1-trsp)*Ttot                  ' visibilité de la réflectance = Ttot*albedo
```

```
evalue_refl(voxel) -> L,occlus        ' ellip_refl() -> réflectance,occultation
' L tient compte de la réflectance diffuse et spéculaire (donc utilise MATI)
trsp = 1-occlus*(1-trsp)              ' module trsp en fonction de l'ellipsoïde
vis  = vis*occlus                     ' idem pour vis
recup_contexte() -> x,y,z,t,mati,ouv,face ' car le rayon d'ombrage va être émis
texel_vers_boite(face,x,y,z) -> x,y,z ' en coord boîte: il faut savoir où on est
omb=0
pour chaque source de lumière
  T=1                                 ' lancement d'un rayon d'ombrage
  intervox({x,y,z}, lum[i], 1,ouv*2,shadevox(),T,nil) -> T
  omb = omb + T*lum_intensite[i]      ' contribution à la lumière incidente
L = L + MATI->ambiant      ' la réflectance contenait diff et spec, on ajoute amb
Ltot = Ltot + L*vis        ' on cumule la contribution à la couleur du pixel
Ttot = Ttot * trsp         ' et on multiplie la transparence.
si (Ttot<1-MAX_OPACITE) fin du rayon
```

D.4.7 ellip_refl(voxel)

Il s'agit ici de calculer l'occultation et la réflectance du voxel, selon la procédure expliquée en 4.3.3.

```
extrait les 3 axes de l'ellipsoïde contenu dans le voxel.
r1,r2,r3  = longueurs de ces axes
P = matrice de passage du repère voxel au repère de l'ellipsoïde
P = P.Jxu     ' intègre le passage du repère monde au repère voxel
S = matrice diagonale (1/r1,1/r2,1/r3)
B = S.P       ' passage au repère où l'ellipsoïde est une sphère
Q = Bt.B      ' forme quadratique de l'ellipsoïde M.Q.M=1

  ' Calcul de la surface apparente moyenne:
  ' si v1,v2,v3 sont les trois rayons de l'ellipsoïde dans le repère monde,
  ' on a  surface apparente moyenne = 1/3 (pi.v1.v2 + pi.v1.v3 + pi.v2.v3)
  ' On pourrait les obtenir via les valeurs propres de Q
  ' qui sont  1/v1^2, 1/v2^2, 1/v3^2, mais le calcul est trop cher.
  ' -> on construit une matrice où l'expression apparaît directement.
S = matrice diagonale (1/rac(r1),1/rac(r2),1/rac(r3))
B = S.P
K = Bt.B
```

```
Smoy = trace(K)/det(K)              ' 3/pi * surface apparente moyenne

  ' calcul de la surface apparente:
D = direction camera
QD = Q.D, DQD = D.QD
fabrique un repere (D,I,J) orthonormé
  '  On cherche l'équation de l'ellipse apparente :
  '  P=QD, Q' = Q-P.tP/DQD
extrait les valeurs propres l,h et les vecteurs propres L,H  de Q'
ratio = 3.l.h./Smoy                 ' Sapparente/Smoyenne

si (rayon d'ombrage) retourne ratio ' ratio = l'occultation (1 pour une sphère)

  ' calcul de la réflectance:

  ' M = {x,y,z} = x.L + y.H + z.D
  ' Pour chaque échantillon {x,y} sur l'ellipse on va chercher z sur l'ellipsoïde.
  ' Puis on tirera N = Q.M, avec lequel on calculera l'illumination locale.

  ' MQM=1 -> équation du seconddegré en z: a.z2 + 2b.z + c = 0
  ' a  = DQD          ( en réalité on divise tout par a                          )
  ' b  = x*LQD+y*HQD  ( -> faire comme si a=1 et retrancher 1/a à c au lieu de 1 )
  ' c  = x*x*LQL+y*y*HQH+2.*x*y*LQH -1
  ' mais on va évaluer ces coefficients de façon incrémentale

  ' on precalcule:
L.QD, Q.L, L.QL,   H.QD, Q.H, H.QH,  LQH

PAS = .25  ' pour 16 echantillons sur la surface de l'ellipse
initialise incréments db/dx et d2c/dx2

pour y de PAS à 1 par pas de 2.PAS
   initialise b,c, incrément dc/dx
   yQH = y.QH

   pour x de -1+PAS à 1 par pas de 2.PAS
     calcule par incrément b,c, dc/dx
```

```
    delta = b*b-a*c
    si (delta>0)    ' on est sur l'ellipse apparente
      surf++        ' comptabilise
      z= (-b+rac(delta))/a   ' on est obligé de calculer explicitement N,
      N = x.QL + yQH + z.QD  ' car on a besoin de sa norme.
      ps0 = M.D

      pour chaque source de lumière
        ps1 = N.lum[i]              ' N.L
        ps2 = (ps0+ps1)/normH[i]    ' N.H  ( norme(H) est précalculé )
        si les contributions sont positives,
           n = norm(N)
           diff += lum_intensite[i]* ps1/n
           spec += lum_intensite[i]*(ps2/n)^MATI->rugos
        ' la contrib du point apparent symétrique s'obtient
        ' facilement au passage car z' = (b+rac(delta))/a

coul = (diff/surf)*MATI->diff + (spec/surf)*MATI->SPEC

retourne coul,ratio
```

D.5 Optimisations des calculs et de la mémoire

D.5.1 Optimisations

Le programme que nous avons implémenté a été un rendu volumique avant de devenir un rendu de texel. La boucle de voxels de **crossvox** était alors la plus interne, nous avons donc cherché à l'optimiser en se ramenant à du calcul en virgule fixe, qui revient à traiter des nombres entiers. Nous avons donc écrit une bibliothèque de macros permettant de manipuler les nombres à virgule fixe. La procédure **crossvox** s'y prête bien dans la mesure où en normalisant les calculs à chaque voxel, on se trouve à faire essentiellement des additions, des soustractions, des multiplications par 2 et des divisions par 2 pour parcourir le volume. Il faut cependant 3 divisions pour la règle de trois permettant de trouver l'intersection du rayon avec les plans $u = 1, v = 1, w = 1$.

Nous nous sommes alors aperçus que si les trois opérations +,-,* sont 3 fois plus rapides en entier qu'en flottant, la division est, elle, 3 fois plus lente ! La prochaine génération de processeurs des Silicon Graphics étant réputée améliorer les calculs flottants, il n'est plus évident qu'il soit intéressant de se ramener ainsi à des calculs entiers. Dans la même veine, nous nous sommes aperçus que les optimiseurs se débrouillent mieux avec des boucles utilisant des indices de tableau que des boucles incrémentant des pointeurs. Il semblerait donc que ces catégories d'optimisations soient dorénavant à oublier, alors que la façon dont le code conserve la structure du cache semble autrement plus importante (mais hélas assez difficile à prévoir).

Par contre l'étude du 'profil' de l'exécutable (étude du coût de chaque fonction avec la commande `prof`) est toujours riche d'enseignements : nous nous sommes ainsi aperçus que la fonction racine carrée occupait le tiers du temps de calcul, nous l'avons donc immédiatement tabulée (ainsi que les fonctions cos, $\sqrt{1-x^2}$ et arccos). Il faut retenir que ce ne sont pas forcément les parties les plus complexes qui sont les plus coûteuses... Un profilage du code est donc indispensable dans le choix des parties à optimiser.

On gagne également beaucoup à installer des caches pour toutes les fonctions d'interpolation, de conversion de systèmes de coordonnée et de calcul de jacobiens, dans la mesure où une partie des calculs ne dépend que de la face (on stocke ainsi tout ce qui est précalculable avec cette seule information). Le gain provient de ce que la localité est relativement bien conservée lors du balayage de l'image. Peut-être pourrait-on même introduire deux systèmes de caches, séparants le traitement des rayons couleur de celui des rayon d'ombrage.

Ces considérations n'ont l'air de rien, mais ce sont elles qui permettent de gagner facilement un facteur 10 en temps de calcul.

D.5.2 Calcul incrémental

Comme nous l'avons vu à la fin de la section 4.3.3, il est parfois possible de factoriser à l'extrême le calcul d'une expression dont les paramètres varient de façon régulière en ne calculant que les incréments d'une itération à l'autre.

Les calculs de forme quadratiques se prêtent au même genre d'optimisation : tous nos termes s'écrivent sous la forme $v_1^t.Q.v_2$, v_1 et v_2 étant le fruit d'un calcul linéaire (on cherche un point de l'ellipsoïde $M^t.Q.M = 1$ avec $M := x.\vec{l} + y.\vec{h} + z.\vec{d}$, (x, y) balayant un plan, ou on évalue la valeur de $N := Q.M/\|Q.M\|$). En précalculant les termes constants $c_1^t.Q.c_2$, on se ramène ainsi à une expression scalaire qui peut elle-même s'écrire de façon incrémentale.

D.5.3 Gestion mémoire

Lors de la compression de l'octree, des blocs de mémoire vont être libérés mais resteront isolés au milieu des blocs utiles, donc difficiles à réutiliser. Nous avons donc commencé à écrire une bibliothèque de fonctions d'allocation-libération-purge de la mémoire, afin soit de réutiliser prioritairement soit de faire migrer en fin de mémoire les blocs libres jusqu'à constituer une page de mémoire libérable. Il se trouve hélas qu'Unix ne SAIT PAS rendre au système une page de mémoire entièrement libre, les blocs libres continuent alors bêtement à occuper plusieurs pages qui font bien défaut aux autres applications (c'est d'ailleurs la raison pour laquelle il faut relancer X11 de temps en temps sur les stations qui n'ont pas résolu le problème). L'optimisation devenant virtuelle, nous avons laissé ces travaux en plan.

D.5.4 Parallélisation

L'INRIA dispose d'une machine parallèle 'all-cache' KSR1 à 72 processeurs, disposant de la bibliothèque *presto* qui facilite l'écriture de programmes multi-threads (i.e. se démultipliant à un moment donné de leur déroulement). La machine prend à sa charge la gestion de la mémoire (cohérence de la représentation dans la mémoire de chaque processeur), on dispose cependant du qualificatif **_private** pour signaler les variables globales qui doivent rester locale à chaque thread (et seront donc dupliquées).

Une équipe de threads est constituée à l'appel de `tid = pr_create_team(NB_PROC)`, elle sera détruite par `pr_destroy_team(tid)`. Une fonction est lancée en parallèle sur l'équipe de threads par `pr_pcall(tid,fonction,copyargs(arguments))` ; un thread peut connaitre son numéro par `pr_mid()` et la taille de l'équipe par `pr_tsize()`.

La façon la plus simple de paralléliser le rendu consiste à faire évaluer chaque ligne par un processeur différent, c'est donc la fonction **scanray** qui sera donnée en pâture à l'équipe de threads. Le compteur de ligne vaut initialement le nombre de ligne, **scanray** se contente de décrémenter ce compteur, puis d'allouer et de calculer la ligne correspondante tant qu'on n'a pas atteint 0. Dans le cas monotâche, la fonction épuisera donc les lignes une à une, tandis que dans le cas multitâche les threads vont se partager les premières lignes, puis ensuite viendront consommer les numéros de lignes restants au fur et à mesure qu'ils auront fini leur travail.

Les processeurs de KSR étant approximativement de la puissance d'un SUN2, le gain de temps permis par l'emploi de 70 processeurs est réel mais pas mirifique, la KSR1 étant alors environ 3 fois plus puissante qu'une Silicon Graphics indigo^2 à 200 Mhz. (En revanche elle sera bien plus lente à la construction de l'octree, non parallélisée).

Annexe E

Mode d'emploi

E.1 Fonctionnalités

La plateforme développée constitue un ray-tracer conçu pour illustrer la technique des textures volumiques, ce qui justifie ses nombreuses lacunes. Il est commandé au moyen d'un script éventuellement conversationnel, dont les commandes se distinguent en plusieurs catégories :

– **commandes de programmation**
  - contrôle (boucles, if, include, ...)
  - calcul (arithmétique, hasard, splines, ...)

– **spécification du rendu**
  - variables système (seuil d'opacité, taille de grid, type ombrage, filtrage, ...)
  - spécification de l'image (taille, avant et arrière plan, ...)
  - caméra et lumières
  - matières classiques (couleurs, textures plaquées, textures de Perlin)

– **description des objets géométriques, pouvant recevoir les texels**
  - mode de rendu :
    . `MAP 0` : visualisation du volume de référence courant
    . `MAP 1` : plan infini de texels non déformés
    . `MAP 2` : rendu de la scène (i.e. en considérant les objets)
  - objets géométriques (chargement, construction, déformation,...)
  - manipulations sur les faces et les normales (parcours, bump, vent,...)
  - application des matières sur les faces, et orientation des texels
  - mapping des texels sur la surface (par défaut un texel par face)

– **construction des texels, i.e. du pattern volumique**

- paramètres de tracé dans le volume (cyclique, CSG, réflectance...)
- primitives géométriques (cylindre, sphère, cube,...)
- formes composées: groupes, fonctions implicites sur squelette, L-systems
- modèles volumiques: scanner, hypertextures, maillage

– **calcul de l'image**

- rendu normal ou spécifique

remarque: l'animation est faite simplement en utilisant une boucle.

E.1.1 Généralités

Les principes généraux du langage et les commandes importantes sont détaillées ici. On trouvera la liste des commandes à la section suivante.

paramètres

- il n'y a pas de test sur le nombre de paramètres fournis aux commandes, par contre il existe souvent des valeurs par défaut.
- certaines fonctions à grand nombre de paramètres (`defcam, map_fonc, traj`) acceptent '.' pour désigner une valeur par défaut. ex: `1 . 2`

variables

- elles sont désignées par un seul caractère: a..z, A..Z, 0..9.
- on accède à leur valeur par `$v` (flottant) ou `%v` (entier).
- on peut presque toujours se passer du préfixe $, sauf pour le premier paramètre d'une fonction en majuscule (car la variable désigne une destination) et pour les fonctions prenant une chaîne en paramètre (car il y aurait ambiguïté).
- donc ATTENTION: avec `LOOP, WHILE, IF` et les variables systèmes, il faut préfixer par $ ou % les variables passées en paramètre.
- `SET i v` est une affectation simple, équivalente à `SET i $v` ou `SET i = v`.
  `SET $i v` est une indirection ! (variables 0..9 vues comme un tableau) → si $i=0 et $0=5, alors `SET $$i` affecte la variable 5

caractères spéciaux

- `\` : la commande continue à ligne suivante (tout ce qui est à droite est ignoré)
  ATTENTION à ne pas dépasser 255 caractères au total.
- `#` : au début ou dans la ligne : le reste de la ligne est un commentaire.

- `/*` : commentaire comme en C, autour d'un bloc de lignes.
- `:` : sépare plusieurs commandes mises sur une même ligne.
- `$i` : valeur de la variable i (flottant).
- `%i` : valeur formatée (entier sur 4 chiffres, typiquement utilisée en suffixe de nom de fichier).
- `^i` : valeur du compteur de la $i^{\text{ième}}$ boucle (0 = la plus imbriquée).
- `@` : retour à la ligne dans une chaîne (commande `LABEL`).
- `+*-/()[],=` : pour clarifier les scripts, l'utilisateur peut utiliser ces symboles entre les opérateurs. ATTENTION : ils sont simplement ignorés, SAUF pour certaines fonctions où il faut les éviter (notamment les structures de contrôle et les commandes prenant une chaîne en paramètre).
- `\car` : empêche le script d'interpréter le caractère (`#:%$^`... → `\# \: \$` ...)
- `{exp}` : désigne une expression en polonais inversé, voir `EVAL`.

noms de fichier

- Pour les scripts, '-' désigne STDIN.
- les images sont en RGB si l'extension est `.rgb`, sinon le format est inrimage ; le plan alpha est pris en compte.
- ATTENTION : à cause d'un défaut du parsing, il faut éviter les noms contenant des lettres isolées entourées de chiffres, sinon la lettre est prise pour un nom de variable (ex: `truc/I2image`).

  En principe, les couacs NFS sont gérés...

labels

- ils sont constitués d'un nombre entre 1 et 99 au début de la ligne, et sont utilisé par `GOTO` et `GOSUB`.
- il y a un nombre limité de labels simultanés, par contre on peut les recycler.

  ATTENTION : un label donné indexe soit la ligne la plus récente où il est défini, soit la plus proche dans la suite du script.

  `GOSUB` mémorise la ligne pour y retourner au prochain `RET`. On peut sortir de plusieurs `GOSUB` imbriqués par `RET n`.
- ATTENTION : il faut 'gommer' les `if` et les boucles en cours par `POPIF` et `POPLOOP` si on quitte un `if` par un `GOTO`.

  `ON i GOTO lab1,lab2,..labn` est simulé par `INDEX l i lab1 ... : GOTO %l`

scripts interactifs

- `SYNC` permet de tenir compte des modifications du script en cours d'exécution.
- `WAIT` attend que le paramètre passe à 1 dans le script (par modification du fichier).
- `ESCAPE -` (ou nom de script '-') passe la main à l'utilisateur.

- `PRESSKEY` attend que l'utilisateur appuie sur 'ENTER', et lui permet de prendre la main en tapant '-'.
- `ctrl-Z` arrête le calcul du rendu (mais sauve quand même l'image).
- `SCRIBE` enregistre les commandes, par exemple pour garder un historique (copie les commandes à la fin du fichier indiqué).
  remarque : on peut empiler les `SCRIBE`, ou l'annuler temporairement.
  → historique : (ssi on a fait `SCRIBE` avant)
  . `h` montre les commandes précédentes.
  . `! n` ré-exécute la $N^{\text{ième}}$ commande.
  . `!!` ré-exécute la dernière commande.
  quand un script donne la main à l'utilisateur, celui-ci peut toujours faire des `GOTO/GOSUB` vers le script d'appel, ou appeler un autre script par `INCLUDE` ou `ESCAPE`, mais il ne peut utiliser les structures (`IF`, boucles...).

structures de contrôle
- labels: cf + haut.
- `INCLUDE et RETURN` : pour exécuter un script secondaire puis en revenir. Les paramètres sont constitués d'une ligne de script précédée par ':', mais les expressions sont évaluées avant l'envoi (idem au retour). Donc ATTENTION : ce mécanisme ne reconnaît pas les variables intermédiaires, ex : `RETURN : SET i 2 : SET j $i` se trompe pour j. Exception : ça marche pour les paramètres donnés depuis le shell.
- Les variables sont initialisées à -1, ce qui permet de tester les paramètres manquants lors d'un appel.
- `GOTO/GOSUB` : voir plus haut.
- `LOOP/DO/WHILE` : doit être la seule commande de la ligne. On accède à la valeur de la $i^{\text{ième}}$ boucle imbriquée par `^i` (`^0` désigne la boucle courante). Ne pas oublier le '%' si le paramètre est une variable.
- `IF` : en cas de `GOTO`, il faut 'gommer' les `IF` shuntés, par `POPIF n`. Ne pas oublier le '$' si le paramètre est une variable.

fonctions et expressions
- cf remarques plus haut concernant les variables.
- `PUSH/POP` : on dispose d'une pile, utile notamment pour passer des arguments lors d'un `GOSUB`.
- `INDEX` : permet de choisir une valeur dans une liste (simule un tableau).
- `RNDMODE mode` : contrôle de la séquence aléatoire, permettant d'enregistrer les tirages et de les rejouer. Concerne presque tous les nombres aléatoires utilisés directement ou non

(script, wind, etc mais pas lsysteme).

mode = `simple`	cas habituel.
mode = `store`	les tirages de nombres sont tous enregistrés.
mode = `read`	les tirages suivent les valeurs précédemment enregistrées.
mode = `reread`	on reprend la lecture de l'enregistrement après un arrêt temporaire.
mode = `free`	efface l'enregistrement.

. si on ne fait pas `free` entre deux `store`s, la séquence est prolongée.
. on peut écarter un tirage: `store`, `simple`, tirage isolé, puis re-`store`.
. après un `free`, on repasse automatiquement en mode `simple`.
. en mode `read`, la séquence est lue depuis le début, et cycle en cas de dépassement.

- `EVENT` : choisi un événement selon des probabilités données.
- `GEOMETRIC/RAISON/PERIOD` : permet de calculer les paramètres d'une suite géométrique afin d'aller d'une valeur à une autre. ($v = vo * q^n \rightarrow$ trouver n ou q...)
- `defspline/SPLINE` : définition et utilisation d'une spline.
- `EVAL` : utilise l'évaluateur d'expression du L-système pour calculer une expression en polonais inversé, retourne la dernière valeur de la pile.
  ATTENTION : il faut remplacer \$ et #, déjà interprétés par le script, par ! et " (ou `\$ \#`). On ne peut bien sûr pas accéder aux variables du pattern (réservé aux règles de `lsysteme`), par contre on peut oublier # devant les nombres s'il n'y a pas d'ambiguïté. Les variables globales `?0..?9` sont indépendantes de celles du script, on y accède avant et après `EVAL` par `SETvars` et `GETvars`. Par contre on peut mixer: si i vaut 3, `\#$i` est compris `#3` par l'évaluateur.
- Evaluateur d'expression: (en polonais inversé)
  op: arithmétiques: `+, -, *, /`, logiques: `<, >, =, &, |, x,`
  fonc: `r`(nd) `c`(os) `s`(in) `t`(an) `e`(xp) `l`(og) `^` `xyzTn` (tex3d) `xabP` (plateau)
  vect: `n`(norm) `N`(normalize) `#f` = const, `n` = variable du pattern (0..9)
  . = pop, `_n` = duplique le $n^{\text{ième}}$ dernier élément de la pile `\n` = le remet
  `?n, @n` = lire ou écrire dans la variable globale (0..9)
  ex: `r#.2*#.9+` $= (rnd * .2) + .9$ génère un nombre aléatoire entre .9 et 1.1
  On peut également utiliser des subroutines `$nom` définies par `$nom:exp[:cond]` via la commande `regle` ou lu par `lsysteme`. Dans le cas de `EVAL`, on peut donc construire des fonction prédéfinies.
- fichier: `open/close/READ/write` fourni un système de lecture/enregistrement de nombres: ex: pour utiliser une trajectoire de caméra définie hors programme (ou pour la sauver), pour caler 2 séries d'objets sur les mêmes points, etc.

variables système

- ATTENTION: certaines variables système sont locales à un niveau d'include : `DENS, EPAIS, OBJ, type_refl, curmat, cursurf` (mais pas `curtex`)... cf le qualificatif (global) ou (local) dans la liste des instruction.
- si on positionne `GLOBAL` à 1, `DENS, EPAIS, OBJ` sont exportées (c'est à dire que l'on fixe la valeur par défaut).
- `PROF` : à définir en principe au début du script. Elle est utilisé pour
  - le tracé des primitives dans le texel (la résolution du volume est $2^{prof} \times 2^{prof} \times 2^{prof}$).
  - le rendu (dimension par défaut, définition de l'ouverture quand `FILTRE`=0).
  - la sauvegarde au format scanner.
  - les particules (obsolète).

  → à définir avant le tracé.

  → si on change `PROF` selon les texels, le remettre avant chaque modification ou sauvegarde.

  ATTENTION : loadscan fixe `PROF` selon les données lues.

  remarques : plusieurs texels, même superposés, peuvent avoir des résolutions différentes. Si l'on doit changer `PROF` pour un texel, c'est plutôt pour l'augmenter (ex: on lit une image scan, puis on ajoute un tracé fin).

paramètres du rendu

- `OMBRE` : 0 : pas d'ombre, 1 : n'utilise que la densité, 2 et 3 : anisotrope (3 est le plus cher). `RESO_OMBRE coef` donne le gain d'ouverture pour les rayons d'ombrage.
- `FILTRE` déclenche le rendu adaptatif, `FILTRE_LIM` filtre au minimum (aliase mais évite le flou pour une forte perspective, en se basant sur la plus faible contraction plutôt que la plus forte), `SMOOTH` permet des transitions douces dans le rendu adaptatif.
- `RGBmaterial Cspec Cdiff Camb rugos [sol trsp]`
  - sol: 0 si sol seul (pas de texel, que la géométrie)
    -1 si texel seul (pas de sol, par défaut)
    numéro de la matière du sol sinon (un texel plus un sol). Elle doit déjà exister. remarque: on peut jouer du `cree_mat/mat_num` pour créer puis définir plus tard. éventuellement on peut mettre le numéro de matière courant (même matière pour texel et sol).
  - trsp: transparence, utile uniquement pour une matière de sol.

  remarques :
  - le texel, `GEOM_OPAQUE` et `GEOM_INTERP` sont pris en compte à ce moment. Il suffit donc de faire `TEXEL n` avant `RGBmaterial` si l'on a précédemment défini tous les texels.
  - on peut superposer (i.e. mélanger) des texels de couleurs différentes en chaînant les matières les unes les autres via le paramètre sol. Avec les textures, c'est la seule façon de faire varier la couleur dans un volume.

- `colormaterial`: pour modifier après coup la couleur d'une matière. les 3 coefficients supplémentaires règlent la luminosité des trois teintes.
  remarque : `RGBmaterial` remet à zéro toutes les autres informations.
- `texelmaterial t [pu pv pw]` : à faire après `RGBmaterial` pour associer après coup un texel à la matière (mais on peut s'en passer car `RGBmaterial` prend le texel courant).
  t=numéro de texel
  pu pv pw = permutations des 3 axes du volume, p $\in$ {1,2,3,-1,-2,-3}
- `mat_proc` : `typeC typeT mat1 mat2 mat3 mat4` matières procédurales (textures 3D type Perlin: marbre, bozo...).
  . type = `pois_U`,`pois_M`,`marbre_U`,`marbre_M`,`bozo_U`,`bozo_M`... pour la couleur puis la transparence, sinon `solid`. utilise les coordonnées espace si le suffixe est `_M`, texel si `_U`, patch si `_u` .
  . mat : pour le paramétrage de la texture, on utilise des pseudo-matières. en général:
  . on range dans mat1 les vecteurs D(direction) dans Cs, S(scaling) dans Cd, T(paramètres) dans Ca et prof (pour text3d) dans rugos.
  . on range les deux matières à interpoler dans mat2 et mat3.
  . mat2 et 3 peuvent elles-mêmes être des matières procédurales ou image.
  . `mat_img` : `image mat1 [mat2] [mat3] [0/1]` matière texture plaquée (image)
  . nom = fichier image
  . mat1 = (D,S,T) où seuls les deux premières coordonnées sont utilisées. Dx,Dy = numéro des directions définissant le cadre de l'image, D $\in$ {1,2,3,-1,-2,-3}
  . couleur = *interp*(image*mat2, mat3) en fonction de la transparence de mat3.
  . mat2 et 3 peuvent elles-mêmes êtres des matières procédurales ou image.
  . on peut éventuellement se passer de mat3, et de mat2.
  . rep = 0 pour projeter selon l'espace M et 1 pour l'espace U
- `imagefond`: en cas de bug localisé dans l'image, on peut mettre l'image incorrecte en fond (en retirant la marge noire), puis recalculer un rectangle: `resol` permet de spécifier les coordonnées à recalculer.
- allocation dynamique des texels:
  après le calcul d'une image, on peut savoir quelle a été l'importance visible d'une matière (en tenant compte ou non des ombres) (`GETmat_mark i flag`, flag=1 pour visibilité directe, =0 pour compter les ombre), et si une matière est vue pour la première fois ou vient de disparaître (`GETmat_stat i flag`). $\rightarrow$ on peut par exemple réutiliser plusieurs fois la matière 1 pour des faces qui ne sont jamais vues ensembles, et durant l'animation modifier le texel associé au bon moment (on a donc un seul texel à la fois en mémoire).

contrôle des objets et du mapping géométrique le cas `MAP=2` correspond au rendu normal de la scène, considérant le mapping des texels à la surface des objets, par opposition à `MAP=1` (plan infini de texels) et `MAP=0` (un texel seul).

- placage sur surf :
  les texels sont plaqués sur la surface, leurs côtés verticaux suivent les normales aux points (mais en principe vers l'intérieur). Le 'haut' des texels (w=1) est toujours du côté de la 'vraie' surface. Il y a plusieurs façons d'orienter ces normales:
  . scale> 0 dans `load/renorm`: les texels sont sous la surface, sinon au dessus.
  . `GEOM_OPAQUE=1` la vraie surface est visible -1: c'est la surface décalée. C'est la matière superposée la plus 'externe' qui défini ce flag.
  . un scale<0 dans `movegeom` inverse intérieur et extérieur.
  . faire `HAUTBAS` pour inverser le texel à la création (dessus-dessous, ce qui est plus naturel s'il est destiné à être au dessus de la surface).
  . `EPAIS_CST` évite les déformations arrondissant les coins de la géométrie et maintient une épaisseur constante en allongeant les normales obliques. A faire avant un `load` ou un `renorm`.
  . scale=0 dans `load/renorm` rend les texels approximativement cubiques et tournés vers l'extérieur.
- `modifs geom`:
  . Au `load` et au `renorm`, scale est orienté vers l'intérieur. scale=0 rend les texels approximativement cubiques, tournés vers l'extérieur.
  . les modifications portent sur la dernière géométrie chargée, sinon changer avec `geom_num`.
  . Il FAUT faire un `renormgeom` après les modifications (`extrude, rotface`).
- `multicouche` : plusieurs couches de texels.
  . à ne faire qu'une fois ! Manipulations limitées, pas au point.
  . offset entre les indexes des couches: `nbpt_tot/nbcouche`.
  . si on donne un numéro de couche dans `GETnb/first`, on obtient le résultat pour une couche.
- `interpgeom`:
  pour le morphing entre surface SIMILAIRES (même maillage).
- `GETvois,DIR`, etc :
  on dispose de toutes les fonctions du type 'LOGO' pour se déplacer parmi les faces, y laisser un marqueur, et les modifier (extrusion, matière, etc). cf `GETlabel`.
- `windgeom: x y z [0/1 f0]  [T phi amp] [p op k]`
  déforme le texel par un champ de force.
  xyz = direction de la force
  flag : 0 = mouvement rigide 1 = élongation permise

force $= f0 + amp * fonc^p(2pi.dist/T - phi)$, $dist = \vec{F}.\vec{M}$
fonc() dépend de op: 0: sin 1: sin aplati 3: rafale(k) 2: texture3D

primitives géométriques (tracé volumique dans le texel)

- types d'objets: (fixé par `OBJ`)
  Cela correspond aux primitives microscopiques ('cristallisation','fibre',réflectance locale) réparties dans le volume des objets, responsables du comportement en réflectance (et d'une partie de la transparence).
  . Le chiffre des centaines correspond au type de primitive: 0 = aiguille, 1 = galette (avec un décalage : 001 est le premier et 100 le dernier objet de type 0).
  . Les deux autres chiffres (01..100) codent un paramètre associé (en pourcent).
  . Un signe moins indique que le fibrage est constant et orienté par la normale fournie, faute de quoi il est soit radial (par rapport au centre fourni), soit donné par une fonction de normale précisée avec `REFL`.
  . valeurs de `OBJ`:
  `0` : volume isotrope (pas de primitive).
  `1..100` : fibrage cylindrique plus (1) ou moins (100) prononcé.
  `101..200` : feuilletage plus (1) ou moins (100) prononcé.
  Théoriquement, 0,100,200,>200 donnent le même effet (isotrope)...
  Mieux vaut éviter les valeurs trop marquées (< 5), à cause du pas d'intégration numérique.
  . `REFL primitive` utilise les normales correspondant à la primitive indiquée
    - primitive = `line, plan, demiespace, sphere, galette, ellipse, cube, rect, rectzone, rectline, barre, cross, cylindre, cone`
    - on peut ainsi théoriquement mettre des normales de cube sur une sphère...
    - intérêt pour les prolongements: line/rectline avec `REFL cylindre/barre` s'abstient de mettre des normales sur la tranche.
    - ATTENTION, tout n'est pas compatible. sont universels: sphère,cube, cylindre si axe.
  `REFL [RADIAL]` annule la fonction de normale (on repasse en fibrage radial).
  `REFL AUTO` fait associer les normales correspondant aux primitives.
  ATTENTION: `REFL` est pris en compte ssi OBJ>0, sinon le fibrage est constant.
- `DRAWMODE mode` détermine l'effet des commandes de tracé sur le texel :
  . `ADD` (normal) cumule densités et primitives,
  . `SET` impose densités et primitives,
  . `ANISO` impose la primitive seule,
  . `DRAW` impose la densité seule,
  . `OR` cumule les primitives mais prend le OR (=MAX) des densités,

. `AND` cumule les primitives mais prend le AND(=MIN) des densités,
. `ERASE` cumule les primitives mais retire la densité,
. `SUB` impose la primitive mais retire la densité,
. `MUL` multiplie le voxel par la densité (= atténuation du contenu),
. `MUL2` idem, mais combine les primitives,
. `COL` coloriage (cf `rendulabel` et `COL`),
. `SEUIL` sépare les densités selon un seuil,
. `STOP` plus d'effets sur le texel.

- les primitives elles-mêmes:
  leurs paramètres ont souvent des valeurs par défaut, mais les incohérences ne sont pas testées.
  On utilise souvent la valeur de `EPAIS`, par exemple pour la galette et l'ellipsoïde. ATTENTION: il correspond au plus petit des rayons.
- `STOPRIMIT/ADDPRIMIT/playprimit` :
  permet de constituer un groupe de primitives que l'on peut répéter plusieurs fois dans le texel en changeant le scaling et la position.
- `implicit`:
  permet de définir une fonction implicite à partir d'un squelette à base de points, segments, faces pondérés et d'un potentiel. Il y a parfois des problèmes avec les pondérations négatifs.
- coloriage: (uniquement avec `rendulabel`)
  il s'agit d'un mode très particulier d'utilisation des volumes utile pour la visualisation scientifique, il y a peu d'intérêt à l'utiliser pour construire des texels. Le rendu se fait avec `rendulabel`.
  Dans ce mode, la densité code une couleur: 0..255 = [dens,R,G,B]. On dispose alors de la commande `COL r g b` en remplacement de `DENS`. (cependant, l'effet se cumule toujours avec celui des matières.) remarque: `DENS 1` et `COL 1 1 1` sont équivalents.
  Dès lors, on a intérêt a supprimer tout lissage des densités, avec `LISSAGE 0`.
  Pour manipuler un volume de densités variées (ex: scan), il faut donc le seuiller:
  `DRAWMODE seuil : cube (.5 .5 .5) .5` On peut ensuite repeindre des parties du volume avec `DRAWMODE col`. remarque: comme les couleurs sont en poids faible, la commande `rendu` donne des résultats cohérents (mais sans couleurs !). La 'couleur' mentionnée ci-dessus a peu de degrés de liberté (3x2bits) puisqu'elle est codée dans DENS. Mais on peut aussi la considérer comme un index 0..63 dans une table de vraies teintes:
  `COLTAB i  r g b` connecte la colormap et défini la teinte pour l'index i.
  `COLTAB -1` revient à la visu directe des indexes comme couleurs simplifiées.

manips globales sur l'octree (remplissage du texel)

- compression:
  les primitives sont dessinées pré-compressées dans le volume. Mais on peut gagner encore beaucoup en packant les zones à faibles variations, les trous, et le déchet des opérations CSG. `save` le fait automatiquement, mais mieux vaut faire `compress` après une primitive qui laisse beaucoup de miettes (CSG...), ou quand on défini plusieurs texels à la suite.
- permutations :
  symétries et rotations d'un texel (ou une instance, ou une couche). On indique ce que deviennent les directions 1,2,3, ex: `2,-1,3`, le signe '-' indiquant l'inversion du sens.
  . lors de la construction du texel: `PERMUT` → toutes les instances sont affectées.
  . lors du rendu
    . liée a la facette: `setomap` (MAP 2), `omap`(MAP 1)
    . liée a la matière: `texelmaterial pu,pv,pw`
- `loadscan: nom %sx %sy %sz [c0 c1]` chargement d'une image scanner.
  On peut préciser un scaling (il faut que $s = 2^{-n}$ pour l'instant car on ne fait pas d'interpolation) et un seuillage entre c0 et c1.
- `hypertex: n [c0 c1][sx sy sz][msk e][px py pz]`
  hypertexture à base de textures 3D en volume (ce n'est pas un pléonasme !).
  n: profondeur de la récursion.
  c0 c1: seuillage. Flou si c0 et c1 ne sont pas proches. Espace encombré si petit.
  sx,sy,sz: échelle (= pseudo fréquence pour la profondeur 1)
  msk: répulsion des bords: 1:u0 2:u1 4:v0 8:v1 16:w0 32:w1 e=force(<.5)
  px py pz: périodicité (vaut mieux si le texel est mappé) =sx,sy,0 par défaut. 0 = pas périodique.
- `geomtex`: lit une BDD .obj (format standart, en ASCII) et la dessine dans le texel.
- L-systèmes:
  on peut soit exécuter une chaîne toute prête (par `loadlsys`, mais ça suppose de disposer d'un générateur externe, et ça supporte une taille limitée), soit charger/modifier/construire un L-système qu'on pourra tracer.
  xyz: centre s: scaling N: prof de récurrence flag= élément visible 1:feuilles 2:tronc et branches 4:fleurs
  . constitution d'une chaîne:
    (les paramètres ont des valeurs par défauts, les valeurs sont indiquées par 'v' dans la chaîne puis données en vrac après la chaîne).
    On ne reconnaît que les commandes suivantes:
    . `B(e)`: base de l'arbre, épaisseur.
    . `T(l,e)`: morceau de tronc ou de tige, de longueur et d'épaisseur donnés.

. `F(e)`: feuille.
. `M(l)`: déplacement.
. `V(xyze)`: tige donnée par un vecteur.
. `A`: fleur.
. `+ = \ &`: rotations inclinant l'axe (angle fourni) .
. `# %`: rotations autour de l'axe.
. `@`: nouvel axe principal donné par un vecteur.
. `[souschaine]`: mémorise le repère avant d'exécuter la sous-chaîne.
remarque: si les règles sont données depuis le script, comme celui-ci interprète certain caractères spéciaux, il faut remplacer `\ # %` par `\\`, `\# ou !`, `\%`.

- constitution d'un L-système (chargé ou construit dans le script):
suite de règles type `pattern:substitution` , le pattern `init` correspondant à l'arbre initial.
Règle conditionnelle: `pattern:subst:cond` .
On peut décomposer la chaîne `subst` en plusieurs modules `$nom`, les sous-chaînes étant définies par `$nom:definition[:cond]` (dans le script, utiliser `"` et `,` (ou `\$ \:`) au lieu de $ et :).
Variables et expressions:
Les mots peuvent être suivis de paramètres. Ils sont désignés par 'v' dans le pattern, et par une expression entre parenthèses dans la définition.
expressions: cf EVAL.
On crée les règles par `regles` ou par le prompt `L>`, on peut en charger par `creelsys`. (la première rencontrée est prioritaire, mais peut avoir une condition).
Le L-système est alors exécuté (change en chaîne) et tracé par `execlsys`.
`creelsys: nom N [flag]`
on peut avoir déjà crée des règles avant l'appel (elles auront priorité), voire faire jouer plusieurs L-systèmes successifs sur une même chaîne en évolution (par exemple pour lisser en démultipliant les tiges, ou transformer les terminaisons).
nom='.' si on crée la chaîne avec les seules règles déjà en mémoire
N= niveau de récursion
flag= 1 s'il faut RAZ la chaîne (pour en refaire une avec les même règles) 2 s'il faut RAZ les règles (changer de L-système en gardant la chaîne) 3=1+2 par défaut (si une chaîne existait, on efface tout).
remarques: on prolonge l'exécution d'un pas avec `contlsys`.
On construit une nouvelle instance avec `creelsys . N 1` .
Faire `creelsys . 0` pour remettre à zéro la mémoire.
On peut visualiser la chaîne courante avec `afflsys`, et les règles avec `affregles`.

- Particule-systèmes:
  - . `partic` est obsolète.
  - . il est facile de simuler avec des boucles et `cone, barre, feuilleseg`.
  - . `traj      N V0xyz e0 de expr_pol_inv`

    simule une trajectoire sur N pas de temps.

    Initialisation: Vitesse0 (en fait $V * dt$) et épaisseur0

    Evolution: de/dt, et règle définissant la force (en fait $F * dt2/M$).

    En fait, cela crée un L-système: on le trace avec `execlsys`.

    Donc on peut libérer la mémoire par `creelsys . 0`, et on peut continuer de N pas par `contlsys`, etc.

    remarque: la position courante est donnée par `?0 ?1 ?2`, on peut donc définir un champ de force variable. On peut contrôler directement la vitesse par `del_regle $dv` auquel cas v=F, directement.

E.1.2 Liste des commandes

Une étoile indique que la fonction est détaillée dans la section précédente.
OLD désigne les fonctions obsolètes.

—– structures de contrôle —–

(attention: comme rappelé par %, les variables doivent ici être préfixées)
label = numéro au début de la ligne (nombre limité, mais on peut recycler)

INCLUDE file [: paras]	*	exécute le fichier donné, passe les paramètres
RETURN [: paras]	*	arrête, et retourne les paramètres éventuels
ESCAPE file		exécute *file* ds le même contexte, jusqu'à END
GOTO/GOSUB %i		continue au label i
RET [%n]		revient après le GOSUB (remonte n fois)
LOOP %n ... NEXT	*	boucle n fois (toujours au moins 1)
DO ... WHILE %c		boucle tant que (toujours au moins 1)
CONTINUE [%n]		va à la fin de la boucle n (1 = intérieure)
BREAK [%n]		sort de la boucle n
POPLOOP [%n]		oublie n niveau de boucle (ex:sortie par IF)
IF %c ... ELSE ... ENDIF	*	blocs conditionnels
IFEQ/IFNEQ/IFINF/IFSUP %x %v..		raccourcit pour cond x=v, x#v, x<v, x>v
IFIN/IFOUT %x %inf %sup		IN:inclus dans OUT:exclus de
POPIF [%n]		oublie n niveau de IF (ex:sortie par GOTO)
END		arrête le script
EXIT		sort du programme
SYNC		remet à jour le fichier courant (→ interaction)
WAIT %c		variante: attend que c devienne vrai (")
PRESSKEY [flag_escape]		attend 'ENTER', laisse évent l'utilisateur prendre la main
SYSTEM %commande		exécute une commande shell
SCRIBE %nom / ENDSCRIBE		enregistre la session
PRINT %texte		affichage de texte
LABEL/ADDLABEL %texte		incrustation de texte dans l'image. @ = return
NOLABEL [%0/1]		supprime les labels internes au programme

—– variables et expressions —–

variables = 0..9,A..Z,a..z , valeur de i = $i

<table>
<tr><td>SET i val</td><td></td><td>affecte une valeur à une variable i</td></tr>
<tr><td>INPUT i texte</td><td></td><td>lit i au clavier</td></tr>
<tr><td>PUSH i</td><td>(local)</td><td>empile / dépile i</td></tr>
<tr><td>POP i</td><td></td><td></td></tr>
<tr><td>INDEX v i v1 ... [v10]</td><td></td><td>$v = v[i]</td></tr>
<tr><td>ADD/SUB/MUL/DIV/INV/MOD i val</td><td></td><td>opère sur la variable i:</td></tr>
<tr><td>ABS/ENT/FRAC</td><td>i val</td><td></td></tr>
<tr><td>SIN/COS/C2S/TAN</td><td>i val [s]</td><td>$i := $i OP2 val ou $i := OP1 val</td></tr>
<tr><td>SQRT/LOG/EXP</td><td>i val</td><td></td></tr>
<tr><td>SUP/INF/SGN/EQ/NEQ</td><td>i val</td><td>$i := op_bool(i,v)</td></tr>
<tr><td>AND/OR/XOR/NAND/NOR</td><td>i val</td><td></td></tr>
<tr><td>NOT</td><td>i</td><td></td></tr>
<tr><td>DEG2RAD/RAD2DEG</td><td>i [s]</td><td>conversion deg ↔ rad</td></tr>
<tr><td>ANG i x y [z]</td><td></td><td>conversion cartésien</td></tr>
<tr><td>PHI i x y z</td><td></td><td>conversion en polaire</td></tr>
<tr><td>DIST i x y [z]</td><td></td><td>(2D ou 3D)</td></tr>
<tr><td>POL2X/Y/Z i ang phi [dist]</td><td></td><td></td></tr>
<tr><td>MAX/MIN i x y [z]</td><td></td><td>max/min des 2 ou 3 valeurs</td></tr>
<tr><td>AMAX/AMIN i x y [z]</td><td></td><td>en valeurs absolues. amax=distance L1</td></tr>
<tr><td>INVMAX/INVMIN/INVAMAX/INVAMIN</td><td></td><td>numéro de la variable extrême (1,2 ou 3)</td></tr>
<tr><td>RNDMODE</td><td>* (global)</td><td>contrôle de la séquence aléatoire.</td></tr>
<tr><td>RND i [val] [ofs]</td><td></td><td>$i := $RND * val + ofs$ RND= 0..1</td></tr>
<tr><td>SRND i [val] [ofs]</td><td></td><td>$i := $RND * val + ofs$ RND=-1..1</td></tr>
<tr><td>EVENT i p1 ... [p10]</td><td></td><td>tire i entier selon les probas indiquées</td></tr>
<tr><td>AFFINE</td><td>i a x b</td><td>$i := $ax + b$ (= suite arithmétique)</td></tr>
<tr><td>NORMAL</td><td>i x xo L</td><td>$i := $(x - x0)/L$</td></tr>
<tr><td>PLATEAU</td><td>i x xo x1</td><td>fonction χ: 0 si x<x0, 1 si x>x1, lin entre</td></tr>
<tr><td>P2</td><td>i x a b c</td><td>$i := $a.x^2 + b.x + c$</td></tr>
<tr><td>NBRACP2</td><td>i a b c</td><td></td></tr>
<tr><td>RAC1P2</td><td>i a b c</td><td>racines de P2</td></tr>
<tr><td>RAC2P2</td><td>i a b c</td><td></td></tr>
<tr><td>GEOMETRIC</td><td>i q n v0</td><td>$i := $v0 * q^n$ (suite géométrique)</td></tr>
<tr><td>RAISON</td><td>q n v1 v0</td><td>trouve q tq $v0.q^n = v1$;</td></tr>
<tr><td>PERIODE</td><td>n q v1 v0</td><td>trouve n tq $v0.q^n = v1$;</td></tr>
<tr><td>EVAL</td><td>i expr_pol_inv *</td><td>op: +*-/ <>=&|x rcstel^TP .\ _n ?n @n !cst</td></tr>
</table>

```
SETvars    [i] [n]                   met/prend les n vars à partir de i
GETvars    [i] [n]                     dans le contexte de l'évaluateur d'expr.
TEX3D     i  x y [z] [N] [lx ly lz] l=période, sinon non périodique
defspline i  a0 a1 [da0 da1 T]      définition d'une spline P3[i](t) NB: da=d*T
SPLINE  v  i t [t0 T]  (local)      v = val de spline[i](t), spline sur [t0, t0+T]
open     file          (local)      ouverture d'un fichier de nombres
close    [n]                        fermeture
file      n                         fixe le fichier courant
popfile  [n]                        fichier d'avant
pushfile [n]                        fichier d'après
READ      i                         lit une valeur
write     v                         écrit une valeur
writeln   v                         écrit une valeur et passe à la ligne
```

—– variables systeme —–

```
PROF          prof   *     ^        fixe la profondeur (à mettre en 1ère ligne)
SHOWPROF      prof         |        fixe la profondeur visible
KSR/NBPROC    nb_proc      |        pour la parallélisation. (0: fin de //)
                           |          ( à utiliser juste avant section // )
TRACE    0/1/2             |        mode trace, pour le débug
SORTIE   0/1               |        sortie = lumière(1) ou transparence(0)
AFFICHE  0/1/2             |        affichage en cours de calcul (2: pas close)
COULEUR [0/1]          (global)     mode couleur ou noir et blanc
COMPR variation_max        |        fixe l'écart max à la moyenne pour comprimer
OPAQUE seuil               |        opacité maxi que le rayon ne franchi pas
GEOM_OPAQUE  0/1/-1        | *      la géométrie elle-même (haut/fond) est visible
GEOM_INTERP  0/1           |        on interpole les normales sur la géométrie
GEOM_GRID    i j k [n]     |        grid 3d assoc aux géoms pour opt ray-tracing
EPAIS_CST    0/1           | *      1 : épaisseur texel pas affectée par déformation
LISSAGE      [a]           |        taille du flou autour des voxels. normal=1.
MAP          0/1/2         |        volume isolé / plan de texels / scène complète
TEXEL  n                   v        édite le texel n (à mentionner dans nmap)
EPAIS  rayon          (local)       fixe la taille des trait (1/2 épaisseur)
DENS dens             (local)       fixe la densité des tracés
COL  r g b            (local)       fixe la couleur (cf rendulabel )
COLTAB c  r g b       (global)      transforme les 'couleurs' en index vers une teinte
OBJ  num              (local)       fixe le type des primitives (cf plus bas)
```

..... paramètres du rendu (global) .....

`resol L H [ym yM xm xM]`	résolution. (par défaut en fonction de PROF)
`defcam xc yc zc [xv yv zv] [rol][couv]`	défini caméra: position, visée, etc
`movecam xc yc zc [rol] [couv]`	déplace la caméra à telle position
`avancecam d`	" " dans la direction de visée
`camera dist [couv]`	distance caméra. couv = taille monde visible
`focale angle`	angle d'ouverture de la caméra
`GETfocale i`	'focale'. à diviser pour zoomer
`OMBRE 0/1/2/3 *`	prise en compte ou non des ombres
`PERS 1/0`	mode perspective
`FILTRE 1/0`	filtrage auto en fonction taille (mip-map)
`FILTRE_LIM 1/0`	limite la poussée du filtrage
`SMOOTH 1/0`	interp entre deux niveaux de profondeur
`TAU tau`	opacité $= 1 - e^{-\tau.\rho.dl}$ ou $\tau.\rho.dl$ selon le modèle
`DBX i j k l`	4 paramètres pour debug (usuellement x,y)
`material spec diff amb rugos OLD`	matière des objets
`RGBmaterial Cspec Cdiff Camb rugos [sol trsp]`	* Couls = R, G, B
`colormaterial Cspec Cdiff Camb rugos [cs cd ca]`	* Couls = R, G, B
`texelmaterial t [pu pv pw]`	* texel associé à matière
`mat_img image mat1 mat2 mat3 [0/1] *`	matière texture plaquée
`mat_proc typeC typeT mat1 mat2 mat3 mat4 *`	matière procédurale
`fond C00 C01 C10 C11`	dégradé du fond, 3 composantes par coin
`imagefond nom [marge]`	charge une image de fond, enlève une marge
`imageavant nom [marge]`	charge une image de fond, enlève une marge
`lum dx dy dz [intens][i]`	modifie la source i
`addlum dx dy dz [intens]`	ajoute une source
`dellum`	détruit la dernière source
`Ciso seuil`	seuil pour isovaleurs

—– contrôle du mapping plan (MAP=1) —–

`texture u0 v0 u1 v1 [z0 z1]`	taille de la zone de texture pour MAP 1
`TCOUCHE  k`	couche de texture à éditer
`nmap  i j  num`	map de numéro de tex: 0=vide, 1=texel_1, ...
`dmap  i j  dx dy dz`	map de déplacement
`smap  i j  sx sy sz`	map de scaling
`omap  i j  ix iy iz`	map d'orientation: 1=u 2=v 3=w, signe=sens
`nmapfill  num`	remplissage d'une couche
`dmapfill  dx dy dz`	
`smapfill  sx sy sz`	

—– contrôle du mapping sur géométrie (MAP=2) —–

..... contrôle des surfaces géométriques (objets de la scène) ....

`RAZ_ALL_GEOM`	détruit toutes les géométries et matières
`loadgeom  nom [%scaleN]  *`	lit une géométrie Action3D pour mapping
`loadgeomPOV nom objet [%scaleN]`	lit la géométrie d'un objet d'un fichier POV
`boitegeom  sx sy [sz] [scaleN]`	crée une boîte (sans toit) ou un plan (sz=0)
`toregeom  R r N n [scaleN]`	crée un tore
`foncgeom  nx ny h scaleN exp_pol_inv`	crée une surface $z = expression(x, y)$
`savegeom  nom`	sauve la géométrie Action3D courante
`geom_num  n`	édite la surface n
`GETnbgeom  i`	nombre de surfaces
`GETcurgeom i`	numéro de la surface courante
`renormgeom  [scaleN]  *`	à faire après avoir maltraité une géométrie
`delgeom  [n]`	détruit la surface en mémoire [LA DERNIERE]
`movegeom  x y z [scale]`	déplace et zoom la géométrie
`centregeom  [scale]`	centre la géométrie autour de 0
`dupgeom  [0/1]`	duplique la géom. nouv invisible si 1 (ex : mémo)
`cutgeom  f`	isole la face f de la surface courante
`explodegeom [z]`	découpe la surface en faces indépendantes
`cachegeom  [n] [1/0]`	rend invisible la surface
`copygeom  modele`	copie la géom depuis un modèle SEMBLABLE
`interpgeom mod0 mod1 t`	interpole entre deux modèles SEMBLABLES
`interpP  mod0 mod1 t`	n'interpole que les points
`interpN  mod0 mod1 t`	n'interpole que les normales

```
scalegeom   sx sy sz [0/1]        3 scalings, si opt=1 retourne intérieur/extérieur
symgeom     sx sy sz [0/1]   OLD  3 symétries, si opt=1 retourne (identique scale!)
rotgeom     a  vx vy vz           rotation de la géométrie
geomX/Y/Zsur/sous v               cale la géométrie dans une des 6 directions
splitgeom                         divise les faces en 4
GETgeomX       x [n]              centre de gravité de l'objet
GETgeomY       y [n]
GETgeomZ       z [n]
GETgeomR       r [n]              rayon de la sphère englobante

..... manips sur les faces ........
GETnbface/GETnbpoint i [z]        donne le nombre de faces / de points
GETfirstface/GETfirstpoint i [z] donne l'indice de départ, ev z=num_couche
GETnbcouche                    *  donne le nombre de couches
ISvalid   i f                     indique si f est une face valide
GETnord/GETsud/GETest/GETwest     f donne la face voisine de f
GEThaut/GETbas f' f               idem pour multicouche
GETvois  f' d f                   donne la face voisine de f ds la dir d
DIRvois  d  f f'                  direction de f' par rapport à f
DIRkeep  d' d f                   convertit d pour continuer tout droit après f'!
DIRturn  d  i                     pivote de i quarts (+ = sens trigo)
extrude  z  f                  *  pousse la face de z (selon les normales)
multicouche +-nb               *  couches de texel, vers normale si +
rotface      a f                  tourne la face de a quarts.
rotfacefill  a [z]                tourne les faces (ev juste couche [z])
windgeom   x y z [0/1 f0]  [T phi amp] [p op k]      force (ev sin) sur texels
windface     x y z f  [0/1 1/0 applique une force aux normales
windfacefill x y z [0/1 1/0] ] (flag1= rep local, flag2= mvt rigide)
rndface      f x [y z] [1/0]      perturbation rnd des sommets de la face
rndfacefill  x [y z] [1/0] [z] flag = 1 pour mvt rigide (défaut)
scaleface    s f                  agrandit la hauteur le texel
scalefacefill s [z]               agrandit la hauteur de tous les texels
TSTfacedir     t (nx ny nz) f     t = 1 si f est orientée comme indiqué
GETfaceX       x f                centre du texel
GETfaceY       y f                (à faire après un renorm)
GETfaceZ       z f
GETfaceR       r f
```

..... attributs des faces ....

. . . . . . matières . . . .

`cree_mat`	crée une nouvelle matière (material+n.texel)
`dup_mat modele`	" " depuis le modèle
`mat_num i`	reprend matière i (pour modif coul ,mais pas texel)
`GETcurmat i`	numéro de la matière courante
`GETnbmat i`	nombre de matières
`GETmat_mark i [1/0]` *	importance visible de la matière (0: &ombres)
`GETmat_stat i [1/0]`	0:pas vu 1:apparaît 2:disparaît 3:toujours vu
`GETmati i f`	lit le numéro de matière de la face f
`setmati i f`	définit la matière de f
`defmati i`	mat par def pour les nouvelles faces et MAP0
`setmatifill i [z]`	définit la matière de tout l'objet
`setdmap x y z f`	définit un déplacement dans le texel
`setsmap x y z f`	définit un scaling dans le texel
`setomap i j k f`	définit une permutation ds le texel
`setdmapfill x y z [z]`	idem pour tout l'objet (ev juste couche [z])
`setsmapfill x y z [z]`	" " " "
`setomapfill i j k [z]`	" " " "

. . . . . labels . . . . . .

`GETlabel1/GETlabel2 i f`	lit le label
`setlabel1/setlabel2 i f`	attache un label à la face (bloc note)
`deflabel1/deflabel2 i`	labels par défaut pour les nouvelles faces
`inc/dec label1/2 f`	raccourcit pour +1, -1

. . . . . mapping . . . . .

`map_local`	mapping intrinsèque (un texel=une face)
`map_diapo ux uy uz vx vy vz [x y z]`	projection diapo en M, axes U,V
`map_sphere su sv [x y z]`	proj sphérique centre M, scale texture su,sv
`map_fonc M [I J K] exp_pol_inv`	uvw calculés par exp.
`map_Zsin Tx Ty [phi phi A zo]`	agite le z de la texture en $A.sin(u).sin(v) + zo$
`map_Z dw [w0 1/0]`	épaisseur et décalage. flag = &décalage.
`rndmap f x [y z]`	perturbation rnd des coord texture
`rndmapfill x [y z] [z]`	idem pour tout l'objet (ev juste couche [z])

—– primitives géométriques (pour tracé volumique dans le texel) —–

`REFL`	`[primit]`	`* (local)`	précise la fonction à utiliser pour reflets
`DRAWMODE`	`[mode]`	`* (global)`	précise le mode de tracé dans le volume
`MODULO`	`[%0/1]`	`(global)`	dessine de façon cyclique dans le texel
`HAUTBAS`	`[%0/1]`	`(global)`	dessine dans le texel la tête en bas
`HORIZVERT`	`[%0/1]`	`(global)`	couche le texel
`PERMUT`	`pu pv pw`	`(global)`	permutations (1,2,3,-1,-2,-3)
`STOPRIMIT`		`(global)`	mémorise les primitives au lieu de les tracer
`ADDPRIMIT`			continue le stockage après un `PLAY`
`FREEPRIMIT`			libère les primitives stockées
`playprimit`	`[x y z][s]`		dessine les primitives stockées
`line`	`u v w  u v w`		ligne entre les deux points
`rectline`	`u v w  u v w  rh rv`		poutre entre deux points
`plan`	`u v w  nx ny nz`		plan donné par un point et la normale
`demiespace`	`u v w  nx ny nz`		espace au dessus du plan
`sphere`	`u v w  r`		sphère donnée par le centre et le rayon
`galette`	`u v w  r nx ny nz`		galette donnée par centre, rayon et axe
`ellipse`	`u v w rx ry Axez Axex`		comme pour galette, épaisseur = `EPAIS`
`cube`	`u v w  r`		cube donné par le centre et le demi-côté
`rect`	`u v w  rx ry rz`		parallélépipède donné par centre et rayons
`rectzone`	`u v w  u v w`		parallélépipède donné par 2 points extrêmes
`cross`	`u v w  ro r`		croix 3D, rayon r, épaisseur ro
`cylindre`	`u v [w r nx ny nz]`		cylindre donné par point, rayon et axe
`cone`	`u v w  u v w  r1 r2`		cône donné par 2 points et 2 sections
`barre`	`u v w  nx ny nz  r1 r2`		barre infinie ” ” ” (éviter `HORIZVERT`)
`feuilleseg`	`u v w  u v w  r1 r2 a`		segment conique de feuille en v (ouv a)
`sinusoide`	`u v w  r nx ny nz A`		sinusoïde, pt origine, T/2, plan, amplitude
`impl_reset`			remet à zéro la fonction implicite courante
`impl_point`	`x y z  pot`		ajoute un point avec un poids pot
`impl_line`	`x y z  x y z  pot`		ajoute une ligne
`impl_seg`	`x y z  x y z  pot`		ajoute un segment (ligne sauf dernier point)
`impl_face`	`x y z  x y z  x y z pot`		ajoute une facette triangulaire
`implicit`	`epais  [x y z][s]`		lance le tracé de la fonction implicite.
`partic`	`n`	`OLD`	trace la trajectoire N. faire grid avant.

—– manips sur l'octree (remplissage du texel) —–

`raz`	`[0/1]`		remet à zéro tout le texel 1:free
`grid`			crée le texel entier (à mettre ev, au début)
`compress`		*	propag + surcompress (auto avec save)
`savetexel`	`nom`		sauve le texel
`loadtexel`	`nom`		lit le texel
`savescan`	`nom`		sauve le texel sous forme de données scanner
`loadscan`	`nom %sx %sy %sz [c0 c1]`		lit le texel à partir de données scan
`hypertex`	`n [c0 c1][sx sy sz][msk e][px py pz]`		= prof seuils scaling amort périod
`geomtex`	`[x y z] [s] [geom]`		met la géométrie .obj dans le texel
`loadlsys`	`nom [x y z] [s] [flag]`		L-système: flag = quels elem visibles
`lsysteme`	`nom N [x y z] [s] [flag]`		`loadlsys`:précalc `lsysteme`:en direct
`creelsys`	`nom N [free_flag]`	`(global)`	`lsysteme = creelsys+execlsys+free`
`contlsys`	`[N]`		prolonge la chaîne de N pas
`afflsys`	`[flag]`		affiche la chaîne (que la chaîne si 0)
`affregles`			affiche les règles du L-système
`regle`	`pattern,subst[,cond]`	`(global)`	ajoute une règle (de pref avant `creelsys`)
`del_regle`	`pattern`		recherche et détruit une règle
`SCALElsys`	`v [x y z]`		trouve l'échelle optimale
`execlsys`	`[x y z] [s] [flag]`	`(local)`	exécution (répétable) du tracé dans le texel
`traj`	`N V0xyz e0 de expr_pol_inv`		exp=force. ex: !0!0!-.1 remarque: @012=pos

—— calcul du rendu ——

`rendu`	`nom [zoom] [alpha]`		calcule une vue et sauve (NB: zoom/PROF)
`rendulabel`	`" " "`		mode spécial (dens = [dens,R,G,B])
`save`	`nom theta phi rol zoom iso`	`OLD`	calcule une vue selon l'orientation donnée
`save3`	`nom`	`OLD`	calcule et sauve les vues des 3 côtés
`view`	`nom [flag_xv]`		affiche le fichier image nom (1 : lance xv)
`closeview`	`i`		ferme la fenêtre i

E.2 Exemple

```
# ********** prairie sous le vent **********

#  ---- construction des texels

PROF  7          # résolution 128^3

TEXEL 1 # ===== herbe

SET p=0           # 0 pour herbe, 1 pour poil

IF $p
  EPAIS .1
ELSE
  EPAIS .01       # herbe
ENDIF
DENS 1 : OBJ 110
HAUTBAS 1
MODULO  1         # texel cyclique
REFL auto : DRAWMODE set

SET j .1          # 4 x 4 brin d'herbes
LOOP 4
  SET i -.5
  LOOP 4
    SRND u .12 : ADD u i    # perturbe la position
    SRND v .12 : ADD v j
    IF $p
      SET r=.05
    ELSE
      SET r=.1              # diamètre
    ENDIF
    RND V .15 : ADD V .05   # rectitude
    SRND k .2               # orientation = (1,k,V)
    SET d=.1
    AFFINE x = k*d+v        # position
    AFFINE y = 1*d+u        #
    SET z=0                 # hauteur
    RND h -.02 : ADD h .1
    LOOP 10       # segments en trajectoire parabolique
      SET X=x : SET Y=y : SET Z=z : SET R=r
      ADD z h
      AFFINE d = V*z+d      # orientation du segment
      AFFINE x = k*d+v
      AFFINE y = 1*d+u
      MUL r .8              # décroissance de l'épaisseur
      IF $p
        cone  (X Y Z) (x y z) R r
      ELSE
        feuilleseg  (X Y Z) (x y z) R r -.5  # segment d'herbe
      ENDIF
    NEXT
    ADD i .25
  NEXT
  ADD j .25
NEXT

TEXEL 2 # ===== 2 coquelicots

DENS 1
sphere (.5 .5 .9) .075
sphere (.3 .9 .8) .075
DENS -1
sphere (.5 .5 .975) .075
sphere (.3 .9 .875) .075
DENS 1
```

```
# ----- geometrie et matiere

EPAIS_CST 0
GEOM_GRID (16 16 16)  2
loadgeom objets/colline2.obj -.25  # chargement de la surface
centregeom 4    # positionnement de la surface
scalegeom  (2 1 1)
geomZsur 0

GEOM_INTERP 1
GEOM_OPAQUE 1
RGBmaterial \    # 1: sol marron
  (0 0 0) (.45 .27 .2) (.06 .03 .03)  50    0  0
# speculaire  diffus      ambiant   rugos link trsp
TEXEL 1
cree_mat
RGBmaterial \    # 2: herbe verte + sol
  (.3 .3 .3) (.15 .6 .23) (.02 .075 .03)  50 1
TEXEL 2
cree_mat
RGBmaterial \    # 3: fleurs rouges + herbe ( + sol )
  (.6 .6 .6) (1.2 .2 .0) (.12 .02 .0)  30 2
cree_mat
RGBmaterial \    # 4: fleurs jaunes + herbe ( + sol )
  (.6 .6 .6) (1 1 .0) (.1 .1 .0)  30 2

setmatifill 2                 # remplit la surface d'herbe
rndfacefill ( 1 1 1 )         # perturbe les hauteurs

# on distribue les fleurs aleatoirement

LOOP 466                      # fleurs rouges
  RND i 1404 + 1
  setmati 3 $i
NEXT
LOOP 233                      # fleurs jaunes
  RND i 1404 + 1
  setmati 4 $i
NEXT

dupgeom 1                     # memorise l'etat de la BDD

# ----- animation et rendu

resol 768 576                 # resolution video
MAP 2 : SMOOTH 1 : PERS 1 : FILTRE 1 : OMBRE 1
fond (1.4 1.2 1) (1.4 1.2 1) (.5 .7 1) (.5 .7 1) # ciel

camera 4  .8
lum    (-3 4 1.2)             # lumiere
defcam (-9 -5 2) ( 7 1.5 0)   # camera

SET i=0
LOOP  30                      # animation de 30 images
  PRINT ---- iteration $i
  LABEL itération %i
  copygeom 2                  # restaure la BDD
  AFFINE p = .2094*i          # phase du vent
  windgeom ( 0 1 0) 1  0  (4 p .6) .5 3 .8  # vent en rafale en Y
  windgeom ( 1 0 0) 1 -.3 (1 p .6) 0  2     # vent random en X
  rendu storage/prairie.%i   # calcul du rendu
  ADD i 1
NEXT
```

Bibliographie

[AW87] Amanatides (John) et Woo (Andrew). – A fast voxel traversal algorithm for ray tracing. *In : Eurographics '87*, éd. par Marechal (G.). pp. 3–10. – North-Holland.

[Bec94] Bechmann (D.). – Space deformation models survey. *Computers & Graphics*, vol. 18, n° 4, 1994, pp. 571–586.

[BJ91] Beigbeder (Michel) et Jahami (Ghassan). – Managing levels of detail with textured polygons. *In : COMPUGRAPHICS '91*, pp. 479–489.

[BJM+93] Benouamer (M.O.), Jaillon (P.), Michelucci (D.), et Moreau (J-M.). – A lazy solution to imprecision in computational geometry. *In : Proceedings of the 5th Canadian Conference on Computational Geometry*, pp. 73–78.

[Bli78] Blinn (James F.). – Simulation of wrinkled surfaces. *In : Computer Graphics (SIGGRAPH '78 Proceedings)*, pp. 286–292.

[Bli82] Blinn (J. F.). – Light reflection functions for simulation of clouds and dusty surfaces. *In : Computer Graphics (SIGGRAPH '82 Proceedings)*, pp. 21–29.

[BM93] Becker (Barry G.) et Max (Nelson L.). – Smooth transitions between bump rendering algorithms. *In : Computer Graphics (SIGGRAPH '93 Proceedings)*, éd. par Kajiya (James T.), pp. 183–190.

[BVI91] Bennis (Chakib), Vézien (Jean-Marc) et Iglésias (Gérard). – Piecewise surface flattening for non-distorted texture mapping. *In : Computer Graphics (SIGGRAPH '91 Proceedings)*, éd. par Sederberg (Thomas W.), pp. 237–246.

[CJ91] Coquillart (Sabine) et Jancène (Pierre). – Animated free-form deformation: An interactive animation technique. *In : Computer Graphics (SIGGRAPH '91 Proceedings)*, éd. par Sederberg (Thomas W.), pp. 23–26.

[CMS87] Cabral (Brian), Max (Nelson) et Springmeyer (Rebecca). – Bidirectional reflection functions from surface bump maps. *In : Computer Graphics (SIGGRAPH '87 Proceedings)*, éd. par Stone (Maureen C.), pp. 273–281.

[Coo84] Cook (Robert L.). – Shade trees. *In : Computer Graphics (SIGGRAPH '84 Proceedings)*, éd. par Christiansen (Hank), pp. 223–231.

[Coq90] Coquillart (Sabine). – Extended free-form deformation: A sculpturing tool for 3D geometric modeling. *In : Computer Graphics (SIGGRAPH '90 Proceedings)*, éd. par Baskett (Forest), pp. 187–196.

[Cro84] Crow (Franklin C.). – Summed-area tables for texture mapping. *In : Computer Graphics (SIGGRAPH '84 Proceedings)*, éd. par Christiansen (Hank), pp. 207–212.

[CT82] Cook (R. L.) et Torrance (K. E.). – A reflectance model for computer graphics. *ACM Transactions on Graphics*, vol. 1, n° 1, janvier 1982, pp. 7–24.

[dREF+88] de Reffye (Phillippe), Edelin (Claude), Françon (Jean), Jaeger (Marc) et Puech (Claude). – Plant models faithful to botanical structure and development. *In : Computer Graphics (SIGGRAPH '88 Proceedings)*, éd. par Dill (John), pp. 151–158.

[EMP+94] Ebert (David), Musgrave (Kent), Peachey (Darwyn), Perlin (Ken) et Worley. – *Texturing and Modeling: A Procedural Approach.* – Academic Press, octobre 1994. ISBN 0-12-228760-6.

[FLCB95] Fleischer (K. W.), Laidlaw (D. H.), Currin (B. L.) et Barr (A. H.). – Cellular texture generation. *In : Computer Graphics (SIGGRAPH '95 Proceedings)*, pp. 239–248. – Robert Cook, 1995.

[Fou92] Fournier (Alain). – Normal distribution functions and multiple surfaces. *In : Graphics Interface '92 Workshop on Local Illumination*, pp. 45–52.

[FPdB90] Fracchia (F. David), Prusinkiewicz (Przemyslaw) et de Boer (Martin J. M.). – Visualization of the development of multicellular structures. *In : Proceedings of Graphics Interface '90*, pp. 267–277.

[FR86] Fournier (Alain) et Reeves (William T.). – A simple model of ocean waves. *In : Computer Graphics (SIGGRAPH '86 Proceedings)*, éd. par Evans (David C.) et Athay (Russell J.), pp. 75–84.

[Fru91] Fruhauf (M.). – Combining volume rendering with line and surface rendering. *In : Eurographics '91*, éd. par Purgathofer (Werner). pp. 21–32. – North-Holland.

[FvDFH90] Foley (J. D.), van Dam (A.), Feiner (S. K.) et Hughes (J. F.). – *Computer Graphics: Principles and Practices (2nd Edition)*. – Addison Wesley, 1990.

[GdM85] Gagalowicz (Andre) et de Ma (Song). – Model driven synthesis of natural textures for 3-D scenes. *In : Eurographics '85*, pp. 91–108. – 1985.

[GMN94] Gondek (Jay S.), Meyer (Gary W.) et Newman (Jonathan G.). – Wavelength dependent reflectance functions. *In : Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, éd. par Glassner (Andrew). ACM SIGGRAPH, pp. 213–220. – ACM Press. ISBN 0-89791-667-0.

[GN71] Goldstein (Robert A.) et Nagel (Roger). – 3-D visual simulation. *Simulation*, vol. 16, n° 1, janvier 1971, pp. 25–31.

[HH90] Hanrahan (Pat) et Haeberli (Paul E.). – Direct WYSIWYG painting and texturing on 3D shapes. *In : Computer Graphics (SIGGRAPH '90 Proceedings)*, éd. par Baskett (Forest), pp. 215–223.

[HSD94] Holzschuch (Nicolas), Sillion (Francois) et Dretakkis (George). – An efficient progressive refinement strategy for hierarchical radiosity. *In : Fifth Eurographics Workshop on Rendering*, pp. 343–357. – Darmstadt, Germany, juin 1994.

[HTSG91] He (Xiao D.), Torrance (Kenneth E.), Sillion (Francois X.) et Greenberg (Donald P.). – A comprehensive physical model for light reflection. *In : Computer Graphics (SIGGRAPH '91 Proceedings)*, éd. par Sederberg (Thomas W.), pp. 175–186.

[JMN+96] Jancène (P.), Meilhac (C.), Neyret (F.), Provot (X.), Tarel (J.-P.), Vézien (J.-M.) et Vérroust (A.). – Réalité enrichie par synthèse. *In : 10ème congrès AFCET, Reconnaissance des Formes et Intelligence Artificielle*. – Rennes, France, 1996.

[JNP+95] Jancène (Pierre), Neyret (Fabrice), Provot (Xavier), Tarel (Jean-Philippe), Vézien (Jean-Marc), Meilhac (Christophe) et Vérroust (Anne). – Res: computing the interactions between real and virtual objects in video sequences. *In : Second IEEE Workshop on Networked Realities*. – Boston, Massachusetts (USA), octobre 1995. http://www-rocq.inria.fr/syntim/textes/nr95-fra.html.

[Kaj82] Kajiya (James T.). – Ray tracing parametric patches. *In : Computer Graphics (SIGGRAPH '82 Proceedings)*, pp. 245–254.

[Kaj85] Kajiya (James T.). – Anisotropic reflection models. *In : Computer Graphics (SIGGRAPH '85 Proceedings)*, éd. par Barsky (B. A.), pp. 15–21.

[Kau87] Kaufman (Arie). – Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes. *In : Computer Graphics (SIGGRAPH '87 Proceedings)*, éd. par Stone (Maureen C.), pp. 171–179.

[KK89] Kajiya (James T.) et Kay (Timothy L.). – Rendering fur with three dimensional textures. *In : Computer Graphics (SIGGRAPH '89 Proceedings)*, éd. par Lane (Jeffrey), pp. 271–280.

[Lev92] Levoy (Marc). – Volume rendering using the fourier projection-slice theorem. *In : Proceedings of Graphics Interface '92*, pp. 61–69.

[Lew89] Lewis (John-Peter). – Algorithms for solid noise synthesis. *In : Computer Graphics (SIGGRAPH '89 Proceedings)*, éd. par Lane (Jeffrey), pp. 263–270.

[LL94] Lacroute (Philippe) et Levoy (Marc). – Fast volume rendering using a shear–warp factorization of the viewing transformation. *In : Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, éd. par Glassner (Andrew). ACM SIGGRAPH, pp. 451–458. – ACM Press. ISBN 0-89791-667-0.

[Mai92] Maillot (J.). – *Trois approches du placage de texture sur un objet tridimensionnel.* – Thèse de PhD, Université Paris-VI, 1992.

[Max90] Max (Nelson L.). – Cone-spheres. *In : Computer Graphics (SIGGRAPH '90 Proceedings)*, éd. par Baskett (Forest), pp. 59–62.

[MH84] Miller (Gene S.) et Hoffman (C. Robert). – Illumination and reflection maps: Simulated objects in simulated and real environments. *In : SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes.* – juillet 1984.

[Mil88] Miller (Gavin S. P.). – From wire-frames to furry animals. *In : Proceedings of Graphics Interface '88*, pp. 138–145.

[Miy90] Miyata (Kazunori). – A method of generating stone wall patterns. *In : Computer Graphics (SIGGRAPH '90 Proceedings)*, éd. par Baskett (Forest), pp. 387–394.

[MKM89] Musgrave (F. Kenton), Kolb (Craig E.) et Mace (Robert S.). – The synthesis and rendering of eroded fractal terrains. *In : Computer Graphics (SIGGRAPH '89 Proceedings)*, éd. par Lane (Jeffrey), pp. 41–50.

[MYV93] Maillot (Jérôme), Yahia (Hussein) et Verroust (Anne). – Interactive texture mapping. *In : Computer Graphics (SIGGRAPH '93 Proceedings)*, éd. par Kajiya (James T.), pp. 27–34.

[Ney89] Neyret (Fabrice). – *MAGIC-M: étude des échanges radiatifs au sein d'un local complexe.* – Rapport technique, EDF/DER - stage DESS Ingenierie Mathématique, 1989.

[Ney90] Neyret (Fabrice). – *Représentation des tubes pour la polygonalisation des systèmes de particules.* – Rapport technique, TDI - stage ENST, 1990.

[Ney91] Neyret (Fabrice). – *Etat de l'art en radiosité.* – Rapport technique, mémoire de fin d'études ENST, 1991.

[Ney94] Neyret (Fabrice). – Un modèle général et multiéchelle de textures volumiques. *In : AFIG'94 Proceedings*, pp. 211–222.

[Ney95a] Neyret (Fabrice). – Animated texels. *In : Eurographics Workshop on Animation and Simulation'95*, pp. 97–103.

[Ney95b] Neyret (Fabrice). – A general and multiscale method for volumetric textures. *In : Graphics Interface'95 Proceedings*, pp. 83–91.

[Ney95c] Neyret (Fabrice). – *Local Illumination in Deformed Space.* – Rapport technique, RR-2856, INRIA, 1995.

[Ney96a] Neyret (Fabrice). – Synthesizing verdant landscapes using volumetric textures. *In : Eurographics Workshop on Rendering'96*, pp. RR–2846.

[Ney96b] Neyret (Fabrice). – *Trimmed Textures.* – Rapport technique, RR-2857, INRIA, 1996.

[NF87] Nadas (Tom) et Fournier (Alain). – GRAPE: An environment to build display processes. *In : Computer Graphics (SIGGRAPH '87 Proceedings)*, éd. par Stone (Maureen C.), pp. 75–84.

[Nom95] Noma (Tsukasa). – Bridging between surface rendering and volume rendering for multi-resolution display. *In : 6th Eurographics Workshop on Rendering.*

[ON94] Oren (Michael) et Nayar (Shree K.). – Generalization of lambert's reflectance model. *In : Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, éd. par Glassner (Andrew). ACM SIGGRAPH, pp. 239–246. – ACM Press. ISBN 0-89791-667-0.

[Ped95] Pedersen (Hans Kohling). – Decorating implicit surfaces. *In : Computer Graphics (SIGGRAPH '95 Proceedings)*, éd. par Cook (Robert), pp. 291–300.

[Per85] Perlin (Ken). – An image synthesizer. *In : Computer Graphics (SIGGRAPH '85 Proceedings)*, éd. par Barsky (B. A.), pp. 287–296.

[PF90] Poulin (Pierre) et Fournier (Alain). – A model for anisotropic reflection. *In : Computer Graphics (SIGGRAPH '90 Proceedings)*, éd. par Baskett (Forest), pp. 273–282.

[PH89] Perlin (Ken) et Hoffert (Eric M.). – Hypertexture. *In : Computer Graphics (SIGGRAPH '89 Proceedings)*, éd. par Lane (Jeffrey), pp. 253–262.

[PHL91] Patterson (J. W.), Hoggar (S. G.) et Logie (J. R.). – Inverse displacement mapping. *Computer Graphics Forum*, vol. 10, n° 2, juin 1991, pp. 129–139.

[PJM94] Prusinkiewicz (Przemyslaw), James (Mark) et Měch (Radomír). – Synthetic topiary. *In : Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, éd. par Glassner (Andrew). ACM SIGGRAPH, pp. 351–358. – ACM Press. ISBN 0-89791-667-0.

[PLH88] Prusinkiewicz (Przemyslaw), Lindenmayer (Aristid) et Hanan (James). – Developmental models of herbaceous plants for computer imagery purposes. *In : Computer Graphics (SIGGRAPH '88 Proceedings)*, éd. par Dill (John), pp. 141–150.

[Pro95] Provot (Xavier). – Deformation constraints in a mass-spring model to describe rigid cloth behavior. *In : Graphics Interface'95 Proceedings*, pp. 147–154.

[RB85] Reeves (William T.) et Blau (Ricki). – Approximate and probabilistic algorithms for shading and rendering structured particle systems. *In : Computer Graphics (SIGGRAPH '85 Proceedings)*, éd. par Barsky (B. A.), pp. 313–322.

[Ree83] Reeves (W. T.). – Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, vol. 2, avril 1983, pp. 91–108.

[RSC87] Reeves (William T.), Salesin (David H.) et Cook (Robert L.). – Rendering antialiased shadows with depth maps. *In : Computer Graphics (SIGGRAPH '87 Proceedings)*, éd. par Stone (Maureen C.), pp. 283–291.

[RT87] Rushmeier (Holly E.) et Torrance (Kenneth E.). – The zonal method for calculating light intensities in the presence of a participating medium. *In : Computer*

Graphics (SIGGRAPH '87 Proceedings), éd. par Stone (Maureen C.), pp. 293–302.

[Sam90a] Samet (Hanan). – *Applications of Spatial Data Structures.* – Reading, Massachusetts, Addison-Wesley, 1990.

[Sam90b] Samet (Hanan). – *Design and Analysis of Spatial Data Structures.* – Reading, Massachusetts, Addison-Wesley, 1990.

[SAWG91] Sillion (Francois X.), Arvo (James R.), Westin (Stephen H.) et Greenberg (Donald P.). – A global illumination solution for general reflectance distributions. *In : Computer Graphics (SIGGRAPH '91 Proceedings)*, éd. par Sederberg (Thomas W.), pp. 187–196.

[Sch94] Schlick (Christophe). – A survey of shading and reflectance models. *Computer Graphics Forum*, vol. 13, n° 2, juin 1994, pp. 121–131.

[SF92] Shinya (Mikio) et Fournier (Alain). – Stochastic motion - motion under the influence of wind. *In : Computer Graphics Forum (Eurographics '92)*, éd. par Kilgour (A.C.) et Kjelldahl (L.), pp. 119–128.

[Shi92] Shinya (Mikio). – Hierarchical 3D texture. *In : Graphics Interface '92 Workshop on Local Illumination*, pp. 61–67.

[SK92] Snyder (John M.) et Kajiya (James T.). – Generative modeling: A symbolic system for geometric modeling. *In : Computer Graphics (SIGGRAPH '92 Proceedings)*, éd. par Catmull (Edwin E.), pp. 369–378.

[Smi84] Smith (Alvy Ray). – Plants, fractals and formal languages. *In : Computer Graphics (SIGGRAPH '84 Proceedings)*, éd. par Christiansen (Hank), pp. 1–10.

[SP86] Sederberg (Thomas W.) et Parry (Scott R.). – Free-form deformation of solid geometric models. *In : Computer Graphics (SIGGRAPH '86 Proceedings)*, éd. par Evans (David C.) et Athay (Russell J.), pp. 151–160.

[Tai92] Taillefer (Frédéric). – Fast inverse displacement mapping and shading in shadow. *In : Graphics Interface '92 Workshop on Local Illumination*, pp. 53–60.

[Tai93] Taillefer (F.). – *Modélisation du rendu et du comportement des surfaces tissées.* – Thèse de PhD, Université Paris-VII, 1993.

[TF88] Terzopoulos (Demetri) et Fleischer (Kurt). – Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *In : Computer Graphics (SIGGRAPH '88 Proceedings)*, éd. par Dill (John), pp. 269–278.

[TL88] Tang (Zesheng) et Lu (Shengkai). – A new algorithm for converting boundary representation to octree. *In : Eurographics '88*, éd. par Duce (D. A.) et Jancene (P.). pp. 105–116. – North-Holland.

[Tur91] Turk (Greg). – Generating textures for arbitrary surfaces using reaction-diffusion. *In : Computer Graphics (SIGGRAPH '91 Proceedings)*, éd. par Sederberg (Thomas W.), pp. 289–298.

[Tur92] Turk (Greg). – Re-tiling polygonal surfaces. *In : Computer Graphics (SIGGRAPH '92 Proceedings)*, éd. par Catmull (Edwin E.), pp. 55–64.

[WH91] Wejchert (Jakub) et Haumann (David). – Animation aerodynamics. *In : Computer Graphics (SIGGRAPH '91 Proceedings)*, éd. par Sederberg (Thomas W.), pp. 19–22.

[Wil83] Williams (Lance). – Pyramidal parametrics. *In : Computer Graphics (SIGGRAPH '83 Proceedings)*, pp. 1–11.

[WK91] Witkin (Andrew) et Kass (Michael). – Reaction-diffusion textures. *In : Computer Graphics (SIGGRAPH '91 Proceedings)*, éd. par Sederberg (Thomas W.), pp. 299–308.

[WMW86] Wyvill (Geoff), McPheeters (Craig) et Wyvill (Brian). – Soft objects. *In : Advanced Computer Graphics (Proceedings of Computer Graphics Tokyo '86)*, éd. par Kunii (Tsiyasu L.). pp. 113–128. – Springer-Verlag.

[WP95] Weber (Jason) et Penn (Joseph). – Creation and rendering of realistic trees. *In : Computer Graphics (SIGGRAPH '95 Proceedings)*, éd. par Cook (Robert), pp. 119–128.

Thèse de doctorat ès sciences de l'université Paris XI Orsay

Textures Volumiques
pour la synthèse d'images

Fabrice Neyret

Résumé

Les textures volumiques, introduites par Kajiya et Kay en 1989, permettent de représenter des géométries complexes en modélisant une 'peau volumique' au voisinage d'une surface, cette peau étant composée d'un motif de référence dupliqué et déformé à la manière d'une texture classique. Nous avons transformé cette méthode lente et destinée au cas particulier de la fourrure en une méthode générale, commode et rapide pour modéliser, animer et rendre les géométries répétitives complexes comme le feuillage, l'herbe, la forêt, la fourrure, etc... Notre représentation est volumique, mutiéchelle, et code une fonction de réflectance en chaque voxel, ce qui permet de ne recourir qu'à l'information strictement nécessaire pour obtenir l'image, qui est ainsi calculée rapidement et sans aliasing en avec un seul rayon par pixel.

Nous montrons également comment convertir les représentations existantes en volumes, comment mapper les textures volumiques, et comment intégrer le rendu au lancer de rayons usuel.

Mots-Clés : Synthèse d'Images, Réalisme, Textures, Lancer de rayons, Complexité, Texels, Multiéchelle, Niveaux de détails.

Summary

Volumetric textures, introduced by Kajiya and Kay in 1989, aims at modeling complex geometries by creating a 'volumetric skin' in the neighborhood of a surface. This skin is composed of a repeated and deformed pattern, as it is for a classical texture. We have transformed this expensive method designed to model fur into a general, user-friendly and efficient tool, that is able to model, animate and render complex repetitive geometries such as foliage, grass, forest, fur, etc... Our representation is volumetric, multiscale, and encodes a reflectance function in each voxel. This allows us to use only the information that is necessary to obtain an image. This one is thus computed quickly and without aliasing, using a single ray per pixel.

We show also how to convert existing representations into volumes, how to map volumetric textures, and how to integrate the rendering into the standart ray-tracing algorithm.

Keywords : Image Synthesis, Realism, Textures, Ray tracing, Complexity, Texels, Multiscaling, Level of details.