Université Grenoble 1 - Joseph Fourier

Représentations d'arbres réalistes et efficaces pour la synthèse d'images de paysages

Alexandre MEYER

Thèse présentée pour l'obtention du titre de Docteur de l'Université Joseph Fourier
Spécialité Informatique
Arrêté ministériel du 5 juillet 1984 et du 30 mars 1992
Préparée au sein du laboratoire *i*MAGIS-GRAVIR/IMAG-INRIA. UMR CNRS C5527.
Soutenue le 10 décembre 2001

Composition du jury :

Jean-Michel	DISCHLER	Rapporteur
James	STEWART	Rapporteur
Pierre	DINOUARD	Examinateur
Bernard	PEROCHE	Examinateur
Fabrice	NEYRET	Encadrant
Pierre	POULIN	Encadrant
Claude	PUECH	Encadrant

Je dédie cette thèse à mes parents.

Je voudrais remercier en premier lieu Fabrice pour la qualité de son encadrement, toujours présent quand je le demandais, tout en me laissant une liberté que j'ai apprécié. Merci Claude de m'avoir permis de réaliser cette thèse au sein d'iMAGIS. Merci Pierre de m'avoir accueilli aussi chaleureusement au sein de son équipe à Montréal et de m'avoir guider dans mes travaux.

Merci à tous les membres (anciens et actuels) d'iMAGIS et du LIGUM pour ce qui rend le quotidien plus agréable : les discutions, les coups de pouces, la bonne ambiance, les pauses, les pots, les fêtes, les 5 à 7 finissants à pas d'heure, etc.
Merci à mes co-bureaux qui m'ont supportés : Cyrille, Xavier et Franky dont deux d'entres eux on quand même finit par se couper (s'arracher ?) les cheveux... sûrement en signe de détresse.
Merci Sonia pour les parties de raquettes de plages effrénées.
Merci Patricia pour la qualité de ses renseignements et tous les services rendus.
Merci à ceux qui ont partagé mes nuits au labo lors des deadlines : Fabrice, Celine (lors de GI), les siggrapheurs 2001,...
Merci Manu pour cette grap'aventure, pour tous les repas post-bugs (et merci Julie) et pour le mailling blague :-).

Un grand merci à tous mes amis pour toutes les fêtes¸ week-ends, vacances, sorties plongées, skis, ... sans quoi la vie de thésard serait bien plus rude.
Merci Yann, Philou, Cedar, Lio, Nico(s), Martine, Flo, Karim, Stephane(s), Sylvie(s), Annick, Julie(s), Manu, Phil, Guillaume, Zak, Eve, Laurent(s), Lili(s), Marie(s), Dave, Mimie, Renaud, Delph, Eric(s), Pierre(s), Anne, Ninie, Celine(s), Fred(s), Caro, Fanf, Berange, Arnaud, Fabienne, Dan, Cathy, Laurence, Gaëlle(s),... et toutes celles et ceux que j'oublie...

Je garde le meilleur pour la fin en remerciant Mélie dont les contributions à la réalisation de cette thèse seraient trop longues à citer ici, et Paul pour ses superbes fusées en LEGO et autres divertissements.

Merci à tous...

TABLE DES MATIÈRES

TABLE DES FIGURES

Introduction

Du point de vue de la synthèse d'images, la principale caractéristique d'un paysage naturel est sa complexité, dans tous les sens du terme. Un paysage est un vaste terrain de jeux pour l'observateur scientifique où des milliers de phénomènes interagissent entre eux pour donner une richesse qualitative et quantitative de formes, de mouvements, de couleurs, etc. qui engendrent l'émerveillement.

Les débouchés de la synthèse d'images de scènes d'extérieur sont variés. Elles vont des applications cinématographiques, cherchant la qualité à tout prix pour représenter des mondes réels ou imaginaires trop difficiles à recréer et contrôler avec les techniques de trucage, aux jeux vidéo ou simulateurs astreints au temps réel mais cherchant une qualité visuelle de plus en plus riche, en passant par l'aménagement du territoire faisant appel aux outils de visualisation d'un environnement virtuel pour les études d'impacts.

Cette complexité adjointe à cette demande insatisfaite suffit à susciter l'intérêt et à stimuler ce domaine de recherche. L'objet des travaux de cette thèse est le traitement efficace et réaliste de scènes de forêt. Cette finalité nécessite de gérer la multitude de primitives que les arbres engendrent, aussi bien au niveau du temps de calcul que de la qualité des images, et ce, tant dans le cadre du réalisme que du temps réel.

Objet des travaux

La synthèse d'images de paysages nécessite une phase de modélisation[1] des éléments y prenant place, principalement les terrains et les arbres, mais aussi les roches, les ruisseaux, les nuages, l'herbe, etc. Dans cette thèse nous nous concentrerons exclusivement sur le traitement des arbres. Les outils de modélisation d'arbres sont depuis quelques temps passés du monde de la recherche au monde industriel [Bio, LD], ce qui temoigne de leur aboutissement. Il est possible de modéliser de manière réaliste des espèces et des formes variées d'arbres, mais cela se fait au prix de milliers, voire millions de polygones[2]. Le rendu de paysages nécessite souvent des milliers d'arbres

[1] Au sens du terme de modélisation scientifique (représentation), à ne pas confondre avec la "modélisation géométrique", synonyme de "modelage".

[2] Le polygone est la "primitive" de base la plus couramment utilisée en synthèse d'images.

ce qui rend le traitement de ces données très difficile : coût de traitement et aliassage sont les deux principaux problèmes qui en découlent. Les techniques existantes de simplification de maillage en vue de construire des niveaux de détails sont plutôt pensées pour des objets manufacturés, et ne donnent pas de solutions efficaces sur les arbres : les détails trop fins, nombreux et disparates ne peuvent être simplifiés par quelques polygones sans que l'effet des valeurs d'opacité et d'illumination soit altéré. Il faut alors trouver des techniques offrant une alternative à cette utilisation massive de polygones. Notre stratégie consiste à chercher des représentations mieux adaptées au problème. Durant cette thèse nous proposons une philosophie de rendu qui factorise les détails fins (*e.g.* les feuilles) en une représentation plus simple, séparant la forme du comportement vis à vis de la lumière (le modèle d'illumination). Dans ce but nous introduirons deux modèles distincts :

- un modèle d'illumination (*shader*) analytique et multi-échelles représentant le comportement de la lumière,
- un modèle hiérarchique à base d'images (stockant l'illumination) associé à une structure de visibilité pour le calcul de l'ombrage.

Organisation de l'ouvrage

Nous commencerons par une description des phénomènes entrant en jeu dans l'apparence d'un arbre, puis le corps de l'ouvrage se composera de trois parties constituées chacune de deux chapitres. Nous conclurons en récapitulant nos deux représentations, et en envisageant les directions à développer.

Dans la première partie (chapitres 1 et 2), nous exposerons les diverses techniques de modélisation des éléments d'un paysage (terrains, arbres, etc.), ainsi que les techniques classiques de rendu et de visibilité. Nous nous attacherons à montrer la diversité et la complexité des données nécessaires à la modélisation et au rendu d'un paysage. En découlera une description des différentes techniques spécifiques à une classe d'objets (prairie, cheveux, nuages, etc.), mais qui ont pour point commun la gestion de la complexité. Nous montrerons comment les modèles d'illumination intègrent (souvent analytiquement) la complexité d'une micro-géométrie, et comment les modèles dédiés gèrent ces difficultés, en établissant de nouvelles représentations efficaces (*i.e.* n'ayant pas exclusivement recours aux polygones).

Dans la deuxième partie (chapitres 3 et 4), nous présenterons les notions de base qu'il est important d'assimiler pour le développement de modèles hiérarchiques analytiques d'illumination. Nous l'appliquerons ensuite à un exemple concret qui constitue notre première contribution : la réalisation d'une hiérarchie de trois modèles d'illumination analytiques dédiés à la représentation de branches de conifères. Ces modèles sont calculés en effectuant l'intégration analytique d'un modèle d'illumination simple (*e.g.* Phong) sur la distribution de géométrie.

Dans la troisième partie (chapitres 5 et 6), nous développerons une représentation temps réel d'arbres, basée sur l'interpolation d'images pour former des *billboards* et sur l'introduction d'une nouvelle structure de visibilité pour l'ombrage. L'adaptation au point de vue proche et intermédiaire est réalisée en ajoutant une hiérarchie à ce modèle.

L'ouvrage se termine par deux annexes :

- l'Annexe A propose une technique permettant de calculer la moyenne pondérée d'images en utilisant le matériel graphique (ne disposant pas des capacités à traiter plusieurs images simultanément) pour accélérer le calcul d'interpolation.
- l'Annexe B présente les détails de l'implémentation d'un lancer de cônes parallèle sur une grappe de PC expérimentale disposant d'une mémoire partagée distribuée.

Contributions

Nos contributions se situent dans le développement de deux nouvelles représentations :

- Un modèle d'illumination hiérarchique et analytique pour représenter des rameaux de conifères
  - Notre technique est constituée de trois niveaux de modèles d'illumination (*shader*) représentant de la géométrie (aiguilles) dont la taille est visuellement inférieure à celle du pixel.
  - Nos trois modèles d'illumination permettent de représenter à différents niveaux de précision les effets cumulés des petites échelles sans avoir à échantillonner, en tenant compte des ombres et de la visibilité.
  - Ces *shaders* sont conçus par intégration analytique d'un modèle d'illumination simple. Cet aspect analytique offre un rendu efficace en temps de calcul avec une bonne qualité d'images (en particulier avec peu d'aliassage).
  - La continuité de conception de ces trois modèles se traduit par des transitions sans saut lors du rendu.
- Un modèle hiérarchique à base d'images pour le rendu interactif de forêts avec illumination et ombrages
  - Notre représentation basée sur l'association d'une fonction bidirectionnelle de texture (*BTF*) à un *billboard* profite de la capacité des images à contenir l'illumination et l'auto-ombrage. Aucun calcul supplémentaire n'est nécessaire lors du rendu. Le traitement des images comme textures par le matériel graphique permet un temps d'affichage interactif.
  - L'introduction de cubes de visibilité pré-calculés gère l'ombrage sur les objets de la scène et autorise un déplacement interactif de la source de lumière.
  - Nous utilisons cette représentation sous forme hiérarchique (niveaux de détails), ce qui adapte le coût de rendu à la taille de l'objet à l'écran et réduit l'aliassage.
  - Les travaux précédents qui ont introduit le concept de *BTF* l'ont appliqué dans un contexte très restreint, nous avons étendu celui-ci en échantillonnant réellement toutes directions de vue et de lumière, et en séparant l'éclairage ambiant de l'éclairage direct (diffus et spéculaire).
  - Le calcul séparé de l'éclairage ambiant et de l'éclairage direct pour nos *BTF* autorise la prise en compte, au moment du rendu (*i.e.* en temps réel), de l'éclairage du ciel (par exemple pour une scène dont le soleil est caché par les nuages).

Étude de cas

Avant de chercher à reproduire un phénomène, il faut le connaître, et donc commencer par l'observer attentivement. Dans ce cadre, il est également intéressant de se documenter sur la manière avec laquelle d'autres disciplines traitent le même problème. En effet, si la reproduction réaliste du réel est une problématique centrale en synthèse d'images, elle se retrouve aussi en peinture au travers de certains courants artistiques. La connaissance des techniques utilisées par les peintres pourrait donc se révéler riche d'enseignement.

Par conséquent, nous effleurerons à la section 1 le domaine de la peinture en nous appuyant sur deux œuvres classées dans le courant "réalisme" et sur deux œuvres attachées au courant "impressionnisme". En 2, après une brève description du vocabulaire en rapport avec la structure d'un arbre, nous approfondirons nos connaissances sur les arbres, en effectuant une étude de cas détaillée des caractéristiques déterminant leur apparence. En partant de l'expérience du peintre et de l'étude de cas, nous essaierons, à la section 3, de donner une idée intuitive des problèmes que l'on rencontre en synthèse d'images lorsqu'on s'attaque au traitement d'un grand nombre d'arbres.

1 Peindre un arbre

Les travaux de cette thèse visent la réalisation de techniques maîtrisant la complexité des arbres en terme de quantité de données, dans le but de créer des images réalistes. Avec l'intention d'établir un parallèle entre les problèmes que nous rencontreront et les techniques de peinture, nous présentons les œuvres de trois peintres : deux œuvres de Ferdinand G. Waldmüller s'inscrivant dans le courant stylistique du "réalisme", et deux œuvres s'inscrivant dans le courant stylistique de l'"impressionnisme", une de Claude Monet et une de Camille Pissarro. Nous nous intéresserons plus particulièrement à leur manière de peindre les arbres.

La question qui se pose en voyant ces peintures est : "comment fait un peintre pour donner une illusion aussi précise des détails des feuilles alors que son pinceau ne lui permet pas de descendre à des résolutions aussi fines ?" En observant les détails de l'image on s'aperçoit que chaque feuille n'est pas représentée individuellement avec précision, mais que ce sont plutôt des groupes de feuilles qui sont représentés par de petites ellipses, de petites taches colorées de pinceau. L'art du peintre passe par l'élaboration correcte de la couleur de l'ellipse pour donner l'illusion que toutes les feuilles sont présentes (*cf.* figure 1).

FIG. 1 – Deux œuvres du peintre autrichien Ferdinand Georg Waldmüller (1793-1865) s'inscrivant dans le courant stylistique "réaliste", exposées au musée d'art de Viennes.

FIG. 2 – Deux œuvres de style "impressionniste". *À gauche* : "Dame au jardin" de Claude Monet (1840-1926). *À droite* : "Chemin de village" de Camille Pissarro (1830-1903).

En poussant encore plus en avant ce raisonnement que le réalisme ne se trouve pas dans la précision du trait mais dans une utilisation habile de la couleur, le courant artistique "impressionniste" cherche à traduire l'impression ressentie plutôt qu'à représenter objectivement la réalité. Les artistes "impressionniste", avec une analyse quasi scientifique des sensations colorées, substituent au dessin classique la notation des ombres et des reflets. Les objets se diluent dans un courant de lumière et de couleurs : leurs contours perdent toute leur précision. La surface du tableau est formée d'une multitude de petites touches soumises aux changements de la lumière et de la couleur (*cf.* figure 2).

L'idée générale que nous retiendrons des travaux de ces peintres est qu'il n'est pas nécessaire de représenter explicitement tous les détails pour obtenir une image réaliste. Pour un arbre, regrouper les feuilles par paquets peut s'avérer aussi réaliste et bien plus efficace à condition que

la couleur de celui-ci soit correctement élaborée (*i.e.* si elle synthétise le comportement photométrique de toutes les feuilles réunies). C'est l'intuition du peintre qui va le guider dans l'élaboration de cette couleur. Dans le cadre de la synthèse d'images nous ne pouvons pas demander à l'ordinateur d'avoir cette intuition, nous allons donc devoir expliciter les phénomènes impliqués dans l'apparence d'un arbre pour trouver des représentations aptes à intégrer la complexité. Avant de travailler sur l'aspect technique de la conception de ces représentations, il est utile d'effectuer une étude de cas de la structure des arbres, puisque leur apparence en découle directement.

2 Du contenu à l'apparence d'un arbre

Après une brève présentation en 2.1 du vocabulaire lié à la structure d'un arbre, nous ferons en 2.2 une étude de cas des arbres et des phénomènes entrant en jeu dans leur apparence[3].

2.1 Vocabulaire

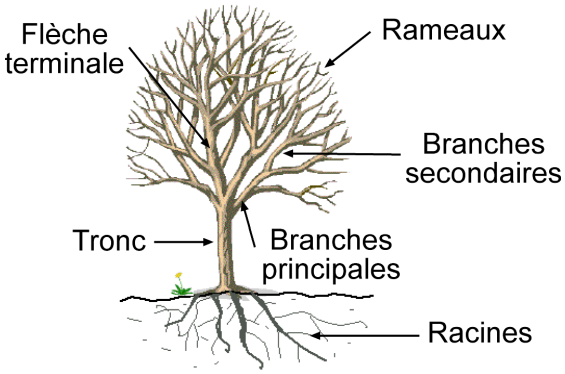

FIG. 3 – Le vocabulaire de base lié à la structure d'un arbre.

Racines : les racines ont trois rôles ; un rôle de stabilisation de la plante, un rôle de transport de la sève et un rôle de stockage de la matière nutritive. Les botanistes en distinguent trois types : la racine principale, les racines pivotantes et les radicelles.
Tronc : le tronc est l'axe principal qui s'est développé et a grossi en circonférence au fur et à mesure des années.
Branches principales ou charpentières : les branches principales partent du tronc, elles se sont formées à cette hauteur dès la première année de leur pousse.
Branches secondaire : les branches secondaires se situent entre les branches principales et les rameaux.
Rameaux et brindilles : la ramure est formée des pousses de l'année. L'ensemble des rameaux forme la couronne de l'arbre. C'est essentiellement cette partie de l'arbre qui est visible de loin.
Flèche terminale ou axe principal : la flèche terminale est un élément majeur du jeune arbre ; c'est la pousse qui détermine la structure du tronc et des branches principales de l'arbre adulte. Si la flèche est coupée, ce sont des pousses latérales qui prendront de la vigueur et la structure de l'arbre se dispersera en deux, trois ou quatre branches principales.

Consulter [Gad] pour plus d'informations.

[3] La plupart des photos d'arbres de cette section proviennent de [bot, Arb].

2.2 Etude de cas

L'objet de cette section est une étude de cas des éléments de la structure d'un arbre responsables de son allure. Notre approche sera celle d'un observateur s'éloignant de l'arbre : nous étudierons en 2.2.1 les feuilles, puis en 2.2.2 les rameaux et nous finirons en 2.2.3 avec les aspects qui font la silhouette générale d'un arbre.

2.2.1 Feuille

FIG. 4 – La texture générale d'un conifère dépend des caractéristiques de couleur des aiguilles. *À gauche* : Mélèze de l'Ouest. Les aiguilles sont mates, courtes et regroupées en faisceaux de 15 à 30 aiguilles. *À droite* : Pin Blanc. Les aiguilles sont regroupées en faisceaux de cinq et de couleur vert bleuté. Leur longueur varie entre 15 et 20 cm.

FIG. 5 – Les aiguilles sont réparties en forme de cône ou dans le plan. *À gauche* : Sapin Baumier. Les aiguilles sont isolées, courtes (1 à 2.5 cm), disposées dans un plan et plates. *À droite* : Épicéa. Les aiguilles sont courtes (1 à 2.5 cm) et disposées en spirale autour du rameau.

On peut distinguer deux sous-familles de feuilles, donnant des aspects complètement différents aux arbres : les aiguilles et les feuilles (au sens commun du terme). La feuille est souvent symétrique par rapport à son axe central, avec une partie gauche et une droite, une face supérieure ou ventrale et une face inférieure ou dorsale. Leur forme est variable : les aiguilles sont plus ou moins cylindriques ; les feuilles peuvent être ovoïdes, dentelées, à lobes arrondis ou pointus, trifoliées, palmées, etc. La dimension, allant de un à quelques dizaines de centimètres, est caractéristique de l'espèce considérée, avec des variations en fonction de la vigueur de pousse du rameau.

Malgré la diversité existante, la forme de la feuille (mise à part la différence feuilles/aiguilles) n'a qu'une influence limitée sur l'apparence globale de l'arbre. Par contre la texture, la couleur (différente pour les deux faces), la transparence, l'aspect mat ou brillant, sont des caractéristiques qui donnent à l'arbre sa texture générale (*cf.* figures 4, 5, 6 et 7). Les feuilles très spéculaires réfléchissent plus de lumière et donnent un aspect brillant à l'arbre (*cf.* figure 7 à droite), alors que des feuilles mates donnent une allure plus foncée (*cf.* figure 4 à gauche).

Contrairement à la majorité des aiguilles la plupart des feuilles ne sont pas persistantes, c'est-à-dire qu'elles tombent pendant les mois d'hiver. L'apparence d'un arbre subit donc des variations au cours de l'année : d'abord par le changement de couleur de ses feuilles, puis par leur chute. En revanche, l'aspect des arbres à feuillage persistant, essentiellement les épineux, n'évolue quasiment pas au cours du temps (*cf.* figures 4 et 5).

FIG. 6 – Les formes des feuilles sont très variées, cependant celles-ci influent peu l'aspect général de l'arbre. *À gauche* : Erable à Grandes Feuilles. Les feuilles sont très grandes et possèdent des sinus profonds. *À droite* : Chêne à gros fruit. Les feuilles possèdent des lobes arrondis.

FIG. 7 – La texture générale d'un arbre dépend en grande partie de la couleur de ses feuilles. *À gauche* : Micocoulier Occidental. Les feuilles sont mates et offrent un aspect différent de chaque côté. Elles sont très pointues au sommet et asymétriques à leur base. *À droite* : Frêne. Les feuilles sont d'un vert vif et très spéculaire. Elles sont composées de cinq à neuf folioles ovales.

2.2.2 Ramure

La ramure est formée des pousses de l'année. La forme (dépendant de l'organisation des feuilles) et la disposition des rameaux influent fortement l'aspect de l'arbre (*cf.* figure 7). Les feuilles sont plus ou moins nombreuses dans un rameau, ce qui est essentiellement déterminé par l'espèce mais aussi par la vivacité de l'arbre. Les rameaux secondaires, au bout des branches, ont une direction particulière : de tombantes à cause de leur poids et de la faiblesse de leur attache (*cf.* figure 8) à parallèles entre elles, presque horizontales, cherchant à s'étaler au maximum pour offrir la surface la plus grande aux rayons du soleil (*cf.* figure 9).

Deux autres aspects contribuant à l'apparence d'un arbre sont la forme et la dispersion de ses rameaux (*cf.* figures 8 et 9). L'aspect de paquet des rameaux se caractérise par leur taille et leur opacité, qui, eux, dépendent du nombre de feuilles et de leur répartition. Par exemple les rameaux du Saule Pleureur (*cf.* figure 8 à droite) ou du Chêne (*cf.* figure 11) sont compacts et denses alors que ceux du Cyprès ou du Peuplier (*cf.* figure 12 à gauche) sont aérés. Leur nombre et leur répartition plus ou moins concentrée donnent un aspect plus ou moins continu et compact à l'arbre (ce qui joue aussi sur la transparence générale comme nous le verrons en 2.2.3). Par exemple le

FIG. 8 – Les rameaux secondaires au bout des branches peuvent avoir une direction pleureuse, donnant un aspect caractéristique à l'espèce. *À gauche* : un Épicéa aux rameaux pleurants. *Au milieu* : le rameau du Saule Pleureur est dense et pleurant. *À droite* : les rameaux dispersés et peu nombreux d'un Saule Pleureur lui donne un aspect aéré.

FIG. 9 – Les feuilles peuvent être parallèles et quasi-horizontales, offrant ainsi une surface maximum au soleil. Les rameaux sont alors moins discernables que pour un arbre à rameaux pleurant. *À gauche* : le rameau d'un Erable de Norvège. *À droite* : trois Erable de Norvège.

Chêne, composé de beaucoup de rameaux a un aspect dense et compact alors que le Saule Pleureur, composé de peu de rameaux a un aspect plus aéré (on distingue ses branches internes).

2.2.3 Forme générale d'un arbre (silhouette)

La forme naturelle d'un arbre dépend de son type biologique et des variations que peut lui faire subir son environnement. On pourra distinguer une forme biologique, issue de la génétique, et une forme réelle de l'individu soumis aux contraintes de son milieu [CIR]. Nous présenterons les deux aspects déterminant la silhouette d'un arbre : la dominance apicale/latérale et la transparence.

Dominante apicale ou latérale

La sève monte : en fonction de l'espèce on observe une réaction plus ou moins marquée par la différence de vigueur entre la croissance verticale et latérale. Ce qui influence la forme générale de l'arbre, aux niveaux des branches principales comme aux niveaux de la ramure.

Les directions et le nombre de branches principales constituent la structure principale de l'arbre et déterminent sa silhouette. Cette structure se décrit en allant des arbres ayant une flèche

dominante (forme globale allongée ou triangulaire) aux arbres ayant plusieurs branches principales (formes plus rondes) (*cf.* figure 10).

FIG. 10 – Le nombre de branches principales détermine l'allure générale de l'arbre. *À gauche* : un Philodendron formé de nombreuses branches principales. *Au milieu* : un Erable à feuilles composées formé de trois ou quatre branches principales. *À droite* : un Copalme d'Amérique, avec une flèche prédominante donnant une forme allongée et une dominance latérale pour ses rameaux, donnant des branches horizontales, presque tombantes.

Cette prédominance pour une croissance verticale ou latérale des branches principales se retrouve aussi dans les branches secondaires et dans la ramure. La dominante apicale engendre des rameaux plutôt verticaux et donne une forme ovoïde, voire allongée, à la silhouette de l'arbre (*cf.* figure 11 à gauche). La dominante latérale produit des rameaux latéraux et donne une forme étalée, voire sphérique, à la silhouette de l'arbre (*cf.* figure 11 à droite).

FIG. 11 – La dominante apicale ou latérale détermine la forme générale de l'arbre, allant d'allongée à étalée. *À gauche* : un Châtaignier. La dominance est apicale pour les branches principales et pour les rameaux, donnant une forme ovoïde. *À droite* : un Chêne. La dominance est latérale pour les branches principales et pour les rameaux, donnant une forme étalée. Le nombre élevé de rameaux lui donne un aspect compact.

Transparence globale

Un aspect important allant de pair avec la silhouette d'un arbre est sa transparence globale. Plus la masse est transparente et plus le détail des contours et contrastes intérieurs prennent de la valeur, et inversement, plus la masse est opaque et plus le contour externe est valorisé.

L'arbre trouve son énergie dans la lumière, les feuilles et les rameaux se développent donc aux endroits où la lumière est la plus présente. Un arbre dense aux rameaux opaques a la majorité

de ses feuilles concentrées sur sa périphérie, l'intérieur étant trop à l'ombre pour y favoriser la pousse massive de nouvelles feuilles (*cf.* 12 à droite). L'auto-ombrage de ce type d'arbre[4] est assez proche de l'auto-ombrage d'une sphère (*cf.* figure 13 à gauche). En revanche, un arbre ayant une transparence plus grande ou une répartition de ses rameaux plus disparate dans son volume (*cf.* figure 12 à gauche) a un auto-ombrage bien plus complexe, la lumière interagissant avec tous les rameaux pour former de nombreuses ombres internes (*cf.* figure 13 à droite).

FIG. 12 – La transparence d'un arbre influence la répartition de ses rameaux et de ses feuilles. *À gauche* : un Tilleul d'Amérique et un *Cladrastis kentukea*, dont l'opacité est forte ont leurs feuilles plutôt réparties sur la périphérie. *À droite* : un Peuplier Baumier et un Cyprès, dont l'opacité est faible, ont leurs feuilles réparties uniformément dans le volume de l'arbre.

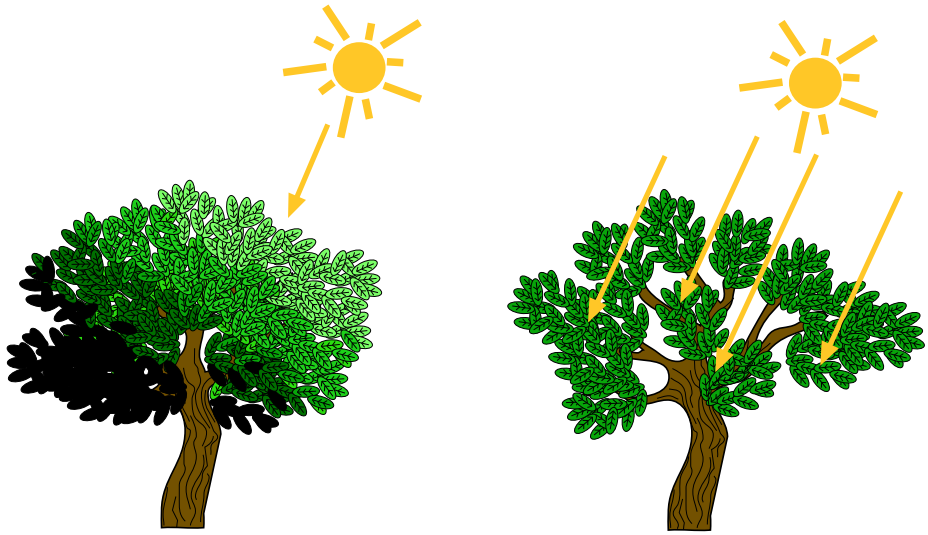

FIG. 13 – *À gauche* : un arbre dense aura un auto-ombrage proche de celui d'une sphère. *À droite* : lorsque la densité et la répartition des rameaux sont aérées, la lumière pénètre dans tout le volume de l'arbre, entraînant un jeu de lumière et d'ombre complexe.

[4] L'auto-ombrage est l'ombre que produit un objet sur lui-même.

2.2.4 En résumé

Pour résumer, on peut dire que l'apparence globale d'un arbre provient de sa texture, de ses rameaux et de sa silhouette.

La texture d'un arbre se caractérise par sa couleur et par sa transparence : sa couleur générale est déterminée par la couleur et l'aspect brillant ou mat des feuilles, et sa transparence par le nombre, la répartition et la taille des feuilles.

La silhouette détaillée d'un arbre est formée par ses rameaux : un petit nombre de rameaux dispersés donne un arbre aéré alors qu'un grand nombre de rameaux sérés lui donne un aspect compact et dense. Leur direction tombante ou non, donne un aspect caractéristique à l'arbre.

La silhouette générale d'un arbre dépend de sa dominance apicale qui engendre une forme ovoïde à allongée ou de sa dominance latérale qui est à l'origine d'une forme étalée à sphérique.

3 Synthèse d'images d'arbres

D'après ce que nous venons de voir, un arbre peut couramment compter plusieurs milliers de feuilles et une scène de paysage contenir plusieurs milliers d'arbres de tous types, ce qui totalise largement plusieurs milliards de primitives (feuilles, aiguilles, éléments de troncs, etc.). Cette complexité est à l'origine de deux problèmes majeurs liés à la reproduction d'arbres en synthèse d'images : la lourdeur du traitement dont nous parlerons en 3.1 et la résolution trop fine des détails que nous traiterons en 3.2.

3.1 Complexité en nombre de primitives

Même si le peintre voulait dessiner toutes les feuilles les unes après les autres sur sa toile, il serait limité dans sa représentation par la résolution de son pinceau et de ses yeux. De ce fait il est contraint de regrouper les feuilles par paquets et de les représenter par une simple touche du pinceau (*i.e.* forme elliptique). En informatique, il est tentant de laisser l'ordinateur calculer les taches répétitives. Il est donc théoriquement possible de laisser la machine traiter toutes les feuilles afin d'obtenir les détails les plus fins pour une image. En pratique ceci n'est pas réalisable car la quantité de données à traiter dépasse largement les capacités d'un ordinateur actuel, de plus de nombreuses applications contraignent les temps de calcul, voire exigent le temps réel (*e.g.* les simulateurs). Une idée pour répondre à ce problème peut être de trouver, à l'instar du peintre, des représentations plus simples à traiter et plus proches de l'effet visible que de l'"atome" structurel. Ce type de représentation simple, intégrant toute la complexité d'un grand nombre de feuilles, devra avoir le même comportement photométrique que l'ensemble des primitives remplacées.

3.2 Idée intuitive de l'aliassage

Le problème de la différence de résolution entre les détails que l'on représente et ce qui est humainement représentable par un pinceau est une bonne introduction au problème d'aliassage que nous exposons maintenant de manière intuitive.

Si on regarde de plus près une image affichée par un ordinateur, on s'aperçoit qu'elle est composée de points (*pixels*). Une solution naïve en synthèse d'images est de calculer la couleur d'un pixel en considérant un échantillon unique, par exemple en appliquant comme couleur celle de l'objet se trouvant en son centre. Si on bouge très légèrement la position de la caméra on s'attend à ce que l'image ne change pratiquement pas. Pourtant du fait de la finesse des détails dans le cas

de scènes de paysages, il est très probable que l'objet se trouvant au centre du pixel change (par exemple d'une feuille verte on peut passer à une branche brune ou au sol). Ce phénomène s'appelle l'aliassage ou *aliasing* (*cf.* figure 14) et se traduit sur les images fixes par une sorte de texture aléatoire peu réaliste ("bruit"), et à l'animation par un effet de grouillement très désagréable.

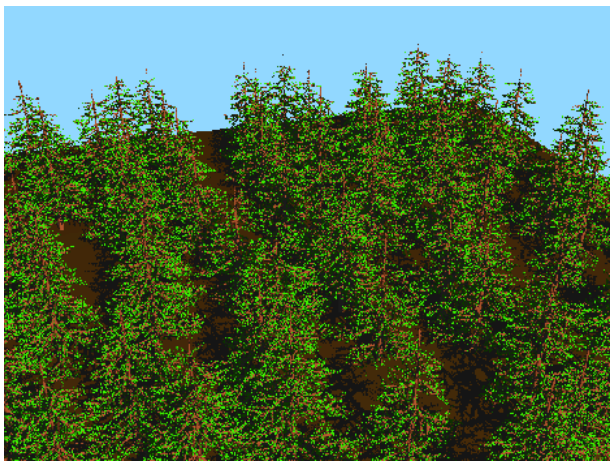

FIG. 14 – Problème d'aliassage : l'image a été calculée en ne traitant qu'un unique objet par pixel choisi aléatoirement.

En regardant la composition de la photographie d'un arbre (*cf.* figure 15), on s'aperçoit que les feuilles ne sont pas distinguables individuellement. En revanche, de par le mécanisme de prise de vue, elles ont bien toutes contribuées à la couleur verte du pixel.

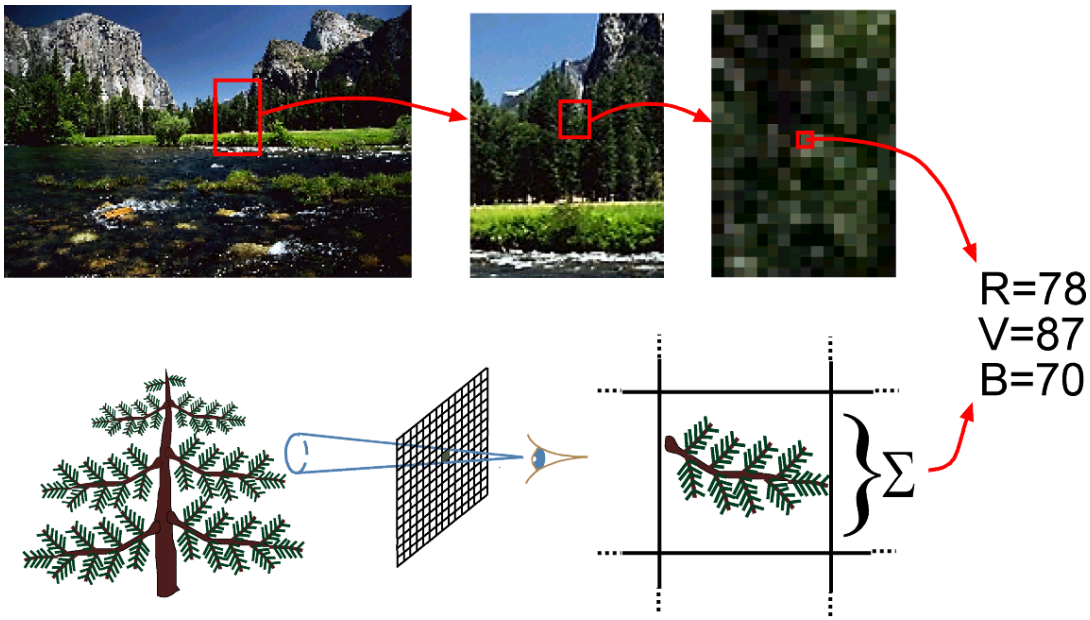

FIG. 15 – La couleur d'un pixel dépend de tous les objets se projetant sur sa surface. Par exemple ce sapin projette de nombreuses données (une partie d'une de ses branches) sur la surface du pixel de la photographie.

Dans le cas d'une photo, tous les objets se trouvant dans un pixel participent (par l'intermédiaire des photons) à la couleur au capteur de l'appareil photo. De la même manière en synthèse

d'images, pour calculer "physiquement" la couleur d'un pixel de l'image il faudrait traiter tous les objets se trouvant dans sa pyramide de vue.

Que se passe-t-il si, dans un soucis de diminution du coût de traitement, on ne considère pas tous les objets apparaissant dans un pixel ? En en négligeant quelques-uns, il est probable que la couleur du pixel obtenue ne soit pas très différente de la couleur exacte. Par contre, si on décide de n'en traiter que quelques-uns choisis de manière aléatoire et uniforme, l'erreur sera plus grande. Même si en moyenne le résultat sera correct, ces erreurs seront visibles par des discontinuités fortes entre des pixels voisins pour une image et par des discontinuités temporelles lors d'une animation (aliassage) : l'œil n'est pas sensible qu'à la moyenne !

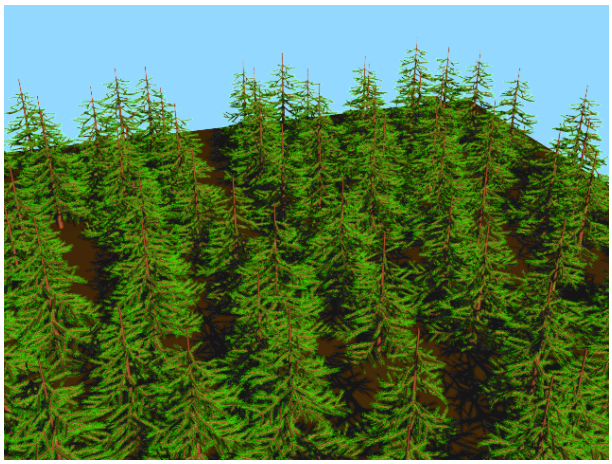

FIG. 16 – Amélioration de l'aliassage : l'image est calculée en traitant au maximum 64 objets par pixel.

4 Conclusion

Ce chapitre permet de mieux nous rendre compte des difficultés qui nous attendent, liées à la représentation des arbres. Ces difficultés sont en grande partie dues à la complexité en nombre de primitives : lourdeur du traitement, finesse des détails, complexité des ombres, des auto-ombrages, problème d'aliassage, etc.

Il ressort de l'étude de cas que les arbres disposent de propriétés hiérarchiques intéressantes : un arbre est constitué d'un axe privilégié (*i.e.* tronc devenant la flèche) puis de branches principales qui sont composées de branches secondaires, elles-mêmes composées de ramures (*i.e.* feuilles ou aiguilles). Ceci ouvre des perspectives de factorisation des traitements. Les rameaux d'une même espèce d'arbres sont assez similaires, et à l'instar du peintre qui représente un groupe de feuilles par une tache de pinceau, il devrait être possible, en synthèse d'images, de regrouper les feuilles d'un arbre (rameau) en une forme simple associée à une technique synthétisant leurs comportements vis à vis de la lumière. Ceci aurait le double avantage de diminuer le coût de traitement et le coût de l'anti-aliassage. La hiérarchie des arbres peut nous aider pour la partie regroupement du travail. En revanche la difficulté consiste à concevoir les techniques transcrivant l'aspect visuel réaliste, tout comme la réalisation de la couleur est délicate pour le peintre. Ce sont à ces deux aspects que se propose de s'attaquer cette thèse.

Première partie

Représenter et rendre les paysages en synthèse d'images : travaux antérieurs et éléments utiles

CHAPITRE 1

Traiter les scènes naturelles

Les scènes naturelles offrent un domaine d'étude extrêmement vaste et complexe, ne serait-ce par la diversité des formes y prenant place : plantes, terrains, fourrures, nuages, feu, fluides, etc. Un des buts premiers de la synthèse d'images étant de reproduire l'apparence de la réalité, nous disposons ici d'un vaste terrain de jeu où quelques jalons ont été posés, mais où il reste encore beaucoup à faire.

On distingue souvent deux aspects dans la complexité d'une scène naturelle : les objets naturels et les phénomènes naturels. Les objets naturels sont concrets et solides, ils interagissent avec la lumière par des modèles matière-lumière, ce sont par exemple les terrains, les végétaux, etc. Les phénomènes naturels ne sont pas tangibles, leur surface n'est pas clairement définie, ce sont par exemple le brouillard, le feu, le vent, etc. Nous avons choisi de travailler sur les objets naturels d'un paysage, et plus précisément sur les arbres : l'objectif de cette thèse est de traiter la complexité introduite par la présence d'arbres. Cette complexité, qui donne la richesse visuelle d'une image se traduit usuellement par un nombre important de primitives, à l'origine de consommation astronomique de mémoire et de temps de calcul.

Si les objets naturels sont tangibles, les mécanismes entrant en jeu dans leur modélisation et leur apparence sont extrêmement complexes. Par exemple, de nombreux facteurs interagissent entre eux à différentes échelles dans la création d'un arbre, comme la richesse en eau du terrain, la quantité de lumière reçue, etc. En 1, nous passerons en revue les techniques existantes de modélisation de terrains et d'arbres, puis nous verrons qu'il est très vite indispensable de modéliser ces objets à différents niveaux de détails si l'on veut pouvoir les utiliser à grande échelle. En 2 nous détaillerons les techniques classiques pour calculer une image à partir de représentations de ces objets naturels (rendu), ainsi que les techniques de visibilité permettant de diminuer le coût de rendu et de faciliter le calcul des ombres.

1 Modélisation de paysages

Nous nous intéresserons essentiellement aux arbres et aux terrains : nous étudierons les modèles de générations de terrains et d'arbres en 1.1 et 1.2, ainsi que les techniques construisant automatiquement leurs niveaux de détails en 1.3.

1.1 Modélisations et représentations de terrains

La modélisation et la visualisation de terrains ont été beaucoup étudiées, d'autant qu'elles intéressent de nombreuses applications comme la géographie, l'aménagement du territoire, l'exploration spatiale, les jeux vidéos, les simulateurs de vol, etc. Pour la synthèse de paysages, qui nous intéresse au premier plan dans cette thèse, les techniques de génération de terrain sont indispensables puisqu'ils représentent le support de quasiment tous les autres éléments (végétation, ruisseaux, etc.). Deux grandes familles de méthodes de modélisation existent :

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

Nous traiterons essentiellement de la génération artificielle de terrains à partir de méthodes procédurales et de leur stockage.

1.1.1 Générations

Les méthodes de génération de terrain sont basées sur la théorie des fractales développée par Manldelbrot [Man78] en 1968, lequel avait remarqué la similitude entre la crête des montagnes et la courbe produite par un mouvement Brownien fractionnaire (*fBm*) à l'origine de sa théorie.

Fournier [FFC82] propose une méthode algorithmique pour le calcul approché du *fBm* basée sur une subdivision récursive du modèle et sur l'introduction d'un facteur aléatoire : pour chaque intervalle non subdivisé, on calcule le point milieu auquel on applique une perturbation aléatoire dont l'amplitude est proportionnelle au niveau de récursivité. Miller [Mil86] proposera une méthode assez similaire en 1986.

Musgrave [MKM89], pour réduire l'apparence uniforme des terrains générés par ces méthodes, apporte deux améliorations :

- l'utilisation des fonctions multi-fractales, qui permettent de prendre en compte les variations hétérogènes d'un même terrain (*e.g.* entre plaines et montagnes) ;
- l'ajout d'un processus d'érosion basé sur un modèle hydraulique : l'eau ruisselle sur le relief en déposant des sédiments ou en érodant la matière. Les temps de calculs sont longs mais les résultats sont très réalistes (*cf.* figure 1.1).

Ces modèles donnent des résultats surprenants de réalisme, mais leur complexité engendre des calculs extrêmement coûteux. Les travaux de Musgrave ont été par la suite, intégré au logiciel *Bryce* [Met] de la société *MetaCreations.*

1.1.2 Représentations

La représentation la plus utilisée pour représenter un terrain est la carte d'élévation[1] (*displacement mapping* [Coo84, MKM89]). Avec cette représentation, un terrain est stocké sous forme

[1] Cette technique est aussi utilisée pour représenter les imperfections d'une surface ou encore les bas reliefs.

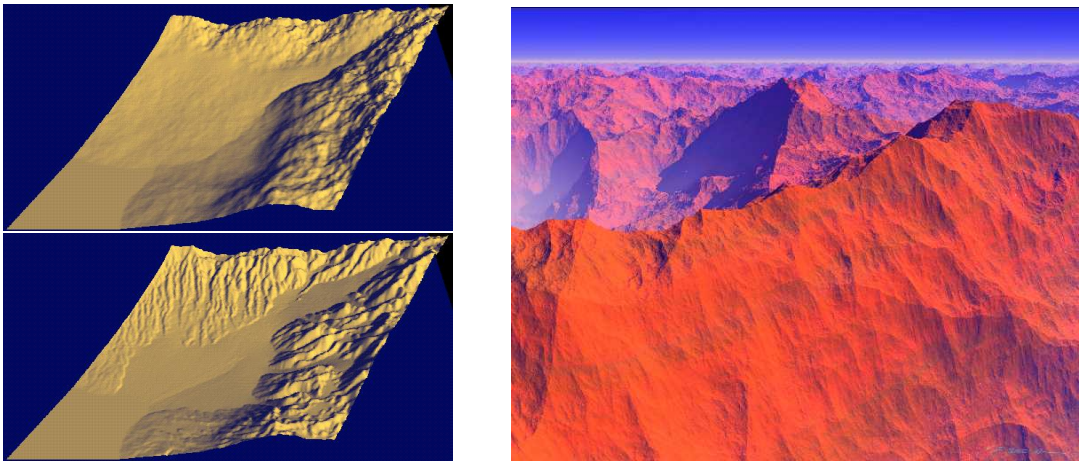

FIG. 1.1 – Le modèle de génération de terrain de Musgrave [MKM89] est basé sur une génération multi-fractales associée à un mécanisme d'érosion. *À gauche* : un exemple d'érosion. *À droite* : un exemple de rendu.

d'un tableau à deux dimensions, où chaque case contient la hauteur du terrain. De plus on peut associer une couleur à chaque case, ce qui revient à associer une texture au terrain. La souplesse de cette représentation provient de la possibilité de considérer ces cartes comme des images 2D dont la manipulation est très intuitive : modélisation possible avec des outils de dessin 2D, facilité de génération de niveaux de détails, etc.

Les travaux présentés ici donnent un bref aperçu des méthodes de modélisation et de représentation des terrains, nous verrons à la section 1.3.1 les techniques de simplification pour construire des niveaux de détails et en 2.1.4 les techniques de rendu et de visibilité qui leur sont spécifiques.

1.2 Modélisations et représentations d'arbres

Après les terrains, les éléments les plus visibles d'un paysage sont les végétaux qui les couvrent, et plus particulièrement les arbres. Il existe aujourd'hui des méthodes capables de construire relativement rapidement, avec plus ou moins de réalisme, la diversité des végétaux d'un paysage. Nous en dénombrons essentiellement trois familles :

- les modèles à base de grammaires ;
- les modèles à base de simulations botaniques ;
- les modèles géométriques.

Ces méthodes construisent dans un premier temps la structure de l'objet, qui est ensuite interprétée pour produire une description géométrique 3D. Nous ne parlerons ici que du processus de modélisation, le rendu étant traité au chapitre 2 (*billboards* 2.3.1, textures volumiques 2.4, système de particules 2.1, images de profondeurs 2.3.4). Nous nous sommes attachés à ces trois familles de modèles parce qu'elles peuvent toutes fournir, plus ou moins facilement, les données de base aux modèles présentés en parties II et III de cette thèse.

Dans cet état de l'art nous ne cherchons pas à être exhaustifs à propos des modèles de génération de végétaux. On peut trouver plus d'informations dans l'état de l'art de la thèse de Chaudy [Cha97], ou dans la catégorisation effectuée par Fournier en 1989 [Fou89].

Les critères auxquels nous porterons attention sont le réalisme du résultat, la diversité des espèces représentables, la nature et la quantité de données nécessaires à la description, les temps de calcul et la prise en compte des influences extérieures (*e.g.* soleil, vent, type de terrain, etc.).

Nous aborderons les modèles à base de grammaires en 1.2.1, suivis des modèles à base de paramètres botaniques en 1.2.2, parce que ce sont les deux familles qui donnent actuellement les meilleurs résultats en terme de diversité et de réalisme. Puis nous verrons un modèle géométrique intéressant car assez simple en 1.2.3 [WP95], qui est un des rares modèles à générer facilement et automatiquement des niveaux de détails.

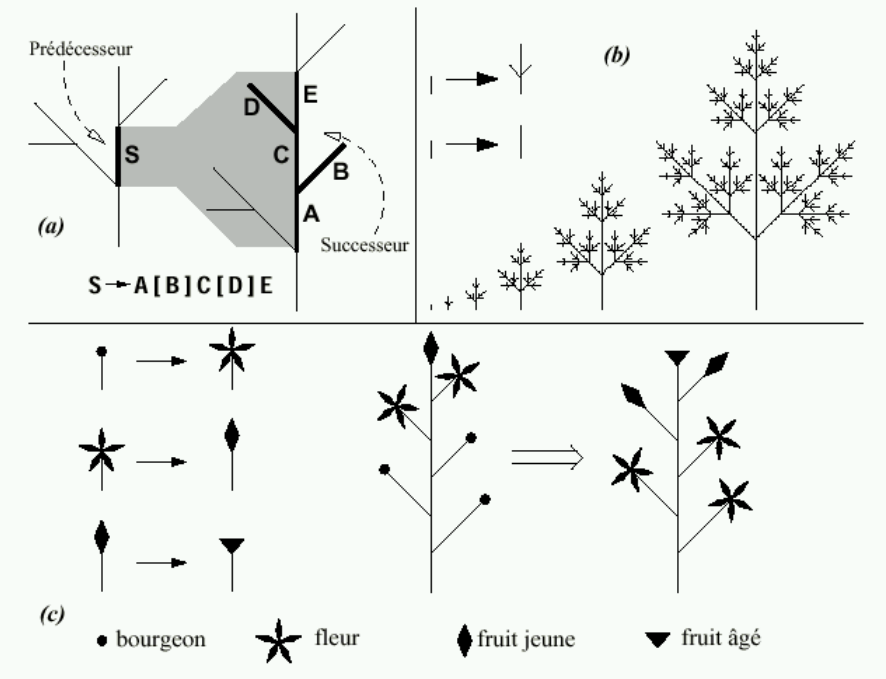

FIG. 1.2 – Principe de réécriture dans les *L-systems* : (a) le motif du prédécesseur est remplacé par le successeur. (b) développement d'une structure ramifiée à partir de deux règles de réécriture. (c) l'évolution des fleurs n'affecte pas la structure sous-jacente. (figure issue de la thèse de Chaudy [Cha97].)

1.2.1 Modèles à base de grammaires

En 1968 Lindenmayer [Lin68] propose un modèle (*L-system*) de développement des structures biologiques basé sur un ensemble de règles de réécriture. Une règle de réécriture est composée de deux membres : le prédécesseur et le successeur. À chaque itération et en parallèle, les occurrences du prédécesseur dans une chaîne sont remplacées par le successeur (*cf.* figure 1.2). Dans le cas des végétaux cette chaîne représente la structure ramifiée de la plante.

Les travaux de Prusinkiewicz [PLH88] ont pris comme base le modèle simple de Lindenmayer et l'ont fortement enrichi. Les processus complexes de croissance de développement sont modélisés grâce à la possibilité des *L-systems* de propager des signaux entre la base et les structures ramifiées. Prusinkiewicz ajoute [PJM94] la prise en compte des facteurs externes comme le voisinage, la recherche de lumière ou l'élagage d'une branche (*cf.* figure 1.3 à gauche). Deussen [DHL$^+$98] utilise le modèle de Prusinkiewicz pour la génération d'un écosystème entier.

Afin de produire plusieurs spécimens différents d'une même espèce de plantes, Prusinkiewicz introduit un facteur aléatoire dans les *L-systems*, ajoutant une probabilité à chaque règle de production (*cf.* figure 1.3 à droite).

La modélisation géométrique des topologies générées se fait en interprétant les symboles de

FIG. 1.3 – *À gauche* : un exemple de croissance soumis à l'élagage. *À droite* : un exemple de scène modélisée par un *L-system*.

la chaîne par un système de type "tortue graphique"[2]. Chaque symbole de la chaîne va, soit modifier la position courante de la tortue, soit appliquer une rotation à la direction courante, soit empiler/dépiler l'état courant du contexte (position, orientation, etc).

Ce modèle de Lindenmayer et Prusinkiewicz pour la génération de plantes s'applique avec succès à de nombreuses espèces. Il est facilement implémentable et extensible, en tout cas dans sa version de base. Cependant, il est difficile à contrôler et nécessite une certaine connaissance morphologique, bien qu'il existe maintenant divers outils faciles d'utilisation comme par exemple *Xfrog* [LD] de la société *Greenworks.*

1.2.2 Modèles à base de paramètres botaniques

De Reyffye [dREF+88] proposent en 1988 un modèle botanique pour la synthèse de végétaux. Cette approche "biologique" de la modélisation des plantes a pour principe la simulation de l'activité des bourgeons sur une échelle de temps discrète. Des probabilités, caractéristiques de l'espèce et du niveau de développement du végétal considéré, sont affectées à chacun des nœuds d'évolutions possibles du bourgeon. Ces données sont issues des connaissances en botanique et en morphologie des végétaux [CIR]. Des facteurs externes peuvent également être intégrés, comme par exemple, la coupe de certaines branches ou des maladies, l'effet de la gravité sur les branches, la présence d'un obstacle entre la lumière et une partie de la plante, etc.

La description issue du moteur de croissance est interprétée comme un ensemble de primitives géométriques simples (troncs de cônes pour les branches et polygones pour les feuilles) qui peuvent ensuite être traitées comme n'importe quel autre objet de synthèse d'images, ce qui facilite leur utilisation par un modeleur pour leur intégration à un paysage, ainsi que par un moteur de rendu.

Ce modèle très complet et réaliste a abouti à un système de simulation développé par le *CIRAD* (Centre de coopération Internationale en Recherche Agronomique pour le Développement) sous l'appellation *AMAP* [CIR], maintenant commercialisé par la société *Bionatics* [Bio].

[2] Par référence au langage *Logo*.

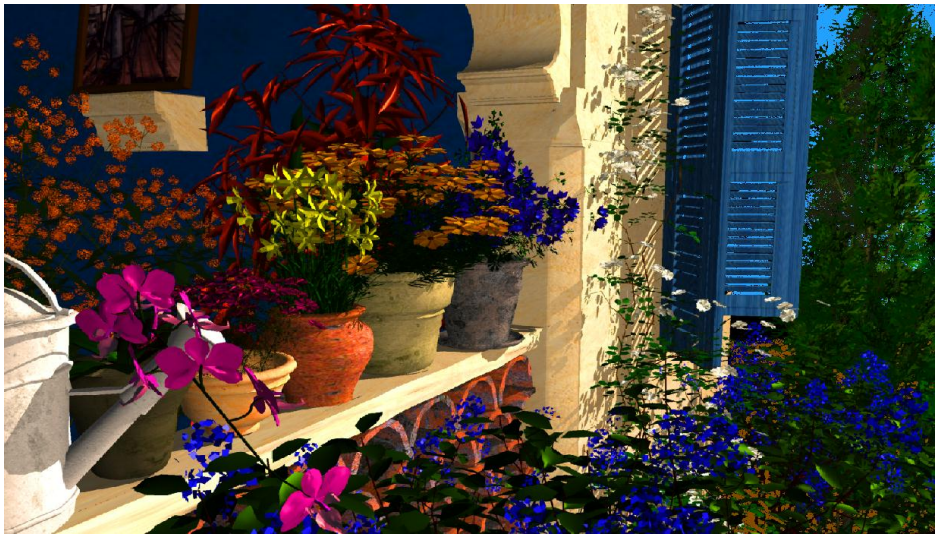

FIG. 1.4 – Exemples de plantes générées par un modèle à base de paramètres botaniques (logiciel *AMAP* issu du CIRAD [CIR], maintenant commercialisé par la société *Bionatics* [Bio]).

1.2.3 Modèles géométriques

En 1995 Weber et Pen [WP95] présentent un modèle intéressant pour la création d'arbres et de leurs niveaux de détails associés. Leur approche est purement géométrique, basée sur un ensemble de paramètres importants qui leur permet d'obtenir des arbres variés et convaincants.

Weber et Pen partent du principe qu'il existe rarement plus de quatre niveaux de récursivité dans la structure ramifiée d'un arbre. Pour chaque niveau de récursivité ils spécifient donc des paramètres géométriques avec une plage de valeurs possibles. Ils prévoient aussi des paramètres pour prendre en compte l'influence du vent, du soleil ou de la taille des branches.

L'aspect intéressant de leur modèle est la possibilité de génération d'une géométrie à complexité variable, ce qui permet d'utiliser leur arbre dans un moteur de rendu à niveaux de détails (*cf.* figure 1.5). Nous reparlerons de niveaux de détails dans le cas des arbres à la section 1.3.1.

1.2.4 Bilan global

Ces modèles de génération de plantes offrent des résultats réalistes grâce à la prise en compte de nombreux facteurs. Les inconvénients propres à l'ensemble de ces modèles sont la nécessité de connaître les paramètres botaniques et morphologiques des plantes (un logiciel comme *AMAP* est fourni avec une large bibliothèque réaliste constituée par le *CIRAD*), ainsi que la lourdeur des descriptions 3D produites qui oblige à utiliser des niveaux de détails pour optimiser la phase de rendu.

1.3 Niveaux de détails

Nous venons de voir les outils de construction importants pour la synthèse de paysages : la génération de terrain et la génération de végétaux. Ces méthodes donnent de bons résultats en terme de réalisme, par contre il est évident que la complexité en nombre de primitives devient vite astronomique si l'on veut représenter tout un paysage : plusieurs centaines de milliers de polygones sont nécessaires pour un arbre, et donc plusieurs milliards pour un paysage entier. Ceci

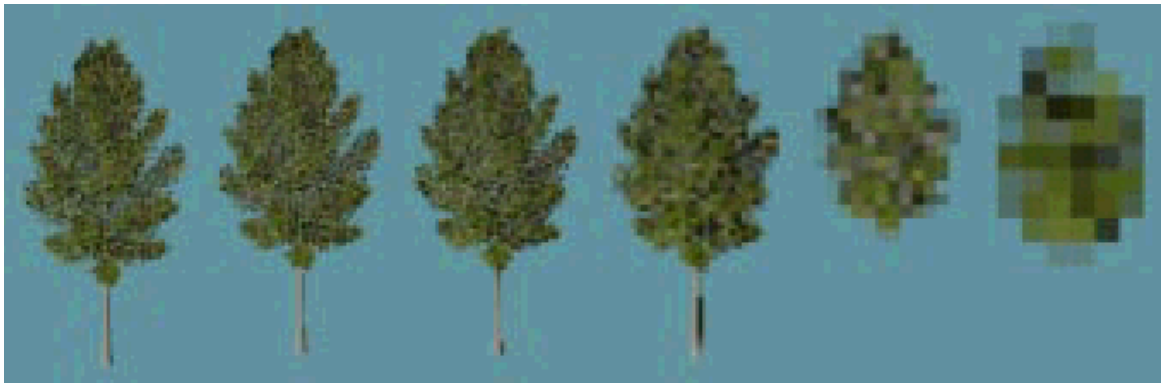

FIG. 1.5 – Exemple de niveaux de détails générés par le modèle géométrique de Weber et Pen [WP95].

augmente énormément le coût de rendu, voire le rend irréalisable. De plus, lorsque des milliers de petites primitives contribuent à la couleur d'un même pixel, les traiter toutes est extrêmement coûteux. Bien souvent, le moteur de rendu se contente de les échantillonner (*cf.* section 2) mais alors, se pose le problème de l'aliassage (*cf.* Étude de cas, section 3.2).

Une solution, devenue standard, est l'utilisation de niveaux de détails : on utilise plusieurs représentations d'un objet, de moins en moins complexes en nombre de primitives au fur et à mesure que l'objet s'éloigne. L'approche classique est de remplacer N polygones suffisamment équivalents par un seul, moins coûteux. Dans le principe, cette notion de niveaux de détails est très efficace. Dans la pratique, la construction des différents niveaux pose de nombreux problèmes : il faut que la transition entre deux niveaux soit continue, qu'aucun artefact ne soit visible à l'œil.

Nous verrons en 1.3.1, que pour les objets polygonaux, il existe de nombreuses techniques de simplifications plus ou moins adaptées à différentes familles d'objets. Nous nous attacherons en particulier au cas des terrains et des arbres, puis nous finirons en 1.3.2 par les techniques de transitions existantes entre différents modèles.

1.3.1 Objets polygonaux à plusieurs niveaux

Pour diminuer le coût de rendu ainsi que l'aliassage, une 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. Un second problème concerne la continuité du rendu lorsqu'on s'éloigne ou qu'on se rapproche de l'objet.

Nous verrons que des techniques existent dans le cas des objets "pleins" (*i.e.* aux surfaces fermées) mais que dans le cas des arbres, d'autres solutions doivent être trouvées.

Cas général

De nombreux travaux portent sur la simplification de maillage d'objets polygonaux ([Hop96, Hop97], etc.). Initialement, leur domaine d'application était la simplification de modèles 3D scannés, qui comportent souvent des millions de polygones (largement redondants). Il devient alors important d'alléger ces représentations pour les rendre utilisables. Ces techniques ont dérivé vers la fabrication automatique de niveaux de détails, souvent avec succès pour certaines classes d'objets (*cf.* figure 1.6).

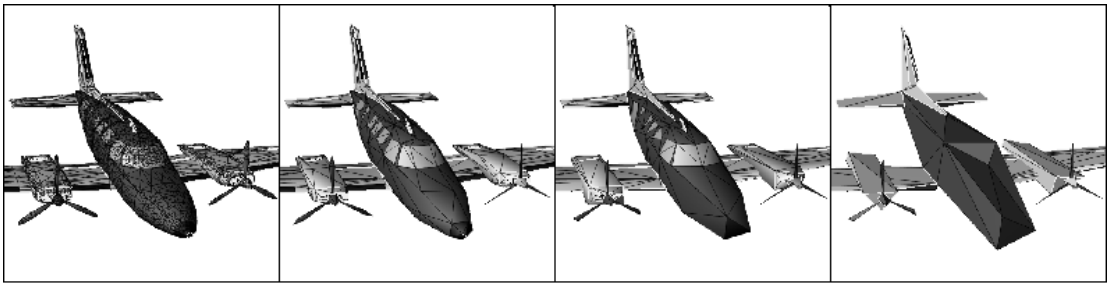

FIG. 1.6 – La simplification de maillage donne de bons résultats avec des objets “pleins”, pour preuve cet avion simplifié par la technique de Hoppe [Hop96].

Cas des terrains

Nous avons vu en 1.1.1 que les terrains sont représentés par des cartes d'élévation pour des facilités de modélisation et de stockage. Pour le rendu, il est nécessaire de convertir la carte d'élévation en série de polygones. La grande quantité de polygones ainsi générés impose de considérer des techniques de simplification pour obtenir un maillage adapté au relief du terrain. En effet, sans simplification, une carte de résolution 256×256 sera représentée par un maillage de $256 \times 256 \times 2 = 131072$ triangles alors que tous les terrains ne nécessitent pas partout une telle précision. Par exemple, une grande zone plate constituée de nombreux triangles pourra avantageusement être remplacée par un polygone unique. Des travaux allant dans ce sens ont été réalisés par Garland [GH95]. Souvent, une deuxième phase de simplification [LKR+96, DWS+97, Hop98] est nécessaire juste avant le rendu pour tenir compte de la position de l'observateur et ainsi satisfaire aux critères de temps réel. De plus, comme la simplification est dynamique, ces modèles garantissent une continuité entre les niveaux.

Cas des arbres

Dans le cas d'un arbre polygonal, il s'agit de remplacer les nombreux polygones représentant des feuilles par quelques uns. Dans le modèle de Weber et Pen, dont nous avons parlé à la section 1.2.3, les auteurs construisent la hiérarchie de niveaux de détails comme ceci : dans l'ordre descendant des niveaux de détails (*i.e.* lorsque l'on s'éloigne progressivement), les branches sont progressivement interprétées comme des lignes et les feuilles comme des points, puis toujours en s'éloignant, certaines branches ou feuilles ne sont plus interprétées (*cf.* figure 1.5). Cela revient à supprimer les branches les moins significatives au fur et à mesure que l'on s'éloigne. Ceci permet une génération automatique et efficace ; cependant, la suppression des feuilles et des branches au fur et à mesure que l'on descend dans la hiérarchie n'est pas très réaliste : il est évident que l'illumination et l'opacité générale de l'arbre va s'en trouver complètement changée et la continuité entre les niveaux ne pourra être garantie.

Bilan

Ces techniques de construction de niveaux de détails par simplification de maillage donnent des résultats efficaces et visuellement continus dans le cas d'objets “pleins” ou dans le cas de terrains, par contre, la nature même de ces techniques comporte une certaine limitation. Par exemple, une tôle ondulée vue de loin sera simplifiée en un plan, ce qui, du point de vue géométrique, peut sembler raisonnable, ne l'est plus du tout du point de vue de la lumière : la manière dont cet objet

réagit à la lumière ne dépend pas de la distance. L'effet visuel moyen des deux modèles sera totalement différent, ce qui se fera sentir lors de la transition par un phénomène de saut dans l'apparence (*poping*). De même une zone dense en feuilles remplacée par son enveloppe géométrique verra sa transparence et son illumination changer. La contribution de nombreuses feuilles ombrées visibles au cœur de la zone disparaîtra.

1.3.2 Transitions entre représentations

Comme nous le disions en introduction un problème essentiel dans l'utilisation des niveaux de détails est la continuité lors de la commutation des modèles. Une solution classique pour forcer cette continuité est de mélanger de manière pondérée les images résultantes de deux niveaux lors de la transition. Cette technique a l'avantage d'être simple et de donner des résultats continus avec toutes les représentations, mais elle est deux fois plus coûteuse puisqu'il faut tracer les deux niveaux à la fois lors de la transition et engendre souvent des artefacts (*e.g.* "écho" de certains détails qui se sont légèrement déplacés entre deux niveaux).

Kajiya dans [Kaj85] introduit la notion de *hiérarchie de modèles*, consistant à passer d'une représentation géométrique à une représentation texturelle, puis photométrique au fur et à mesure que l'on s'éloigne. Nous aborderons durant le chapitre 2 sur les représentations dédiées plusieurs modèles qui se prêtent particulièrement bien à ces transitions : par exemple le modèle de transition entre différentes représentations de détails de Becker (*cf.* section 1.1.8), le modèle des textures volumiques (*cf.* section 2.4), etc. A noter le modèle de Perbet [PC01] destiné à la représentation et à l'animation de prairies en utilisant trois niveaux de détails correspondant à trois représentations : géométrie, représentation 2D $1/2$ et texture. Les résultats sont continus, même lorsque l'herbe se couche sous le vent.

2 Rendu

Nous venons de passer en revu les techniques de générations de terrains et de plantes, ainsi que les techniques associées générant des niveaux de détails, en vue d'alléger les coûts de rendu souvent énormes pour les scènes d'extérieur, et de diminuer les artefacts visuels (l'aliassage) allant de pair avec cette complexité.

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

Nous verrons donc dans cette section consacrée au rendu les techniques classiques de rendu en 2.1 suivies par un aperçu des méthodes de visibilité en 2.2.

2.1 Techniques de rendu

On peut découper le travail de rendu en deux phases : déterminer les surfaces visibles à l'écran, puis calculer la couleur de l'élément de surface (ou la couleur moyenne des éléments) apparaissant en chaque pixel en tenant compte des caractéristiques d'orientation et de matière de celles-ci, ainsi que des conditions d'éclairage. De nombreuses techniques ont été développées dans ce but ; nous ne détaillerons que les deux les plus utilisées, à savoir le tampon de profondeur (*Z-buffer*) en 2.1.1

[3] La détection des faces cachées depuis l'observateur, ainsi que la détermination des ombres, sont deux problèmes de visibilité traités au moment du rendu.

et le lancer de rayons (*ray-tracing*) en 2.1.2. Nous parlerons du rendu volumique en 2.1.3 et nous finirons par les techniques de rendu spécifiques au terrain en 2.1.4.

2.1.1 Le Z-buffer et ses extensions

L'origine du *Z-buffer* est donnée à Catmull [Cat74]. Le principe consiste à laisser à une extension du concept de mémoire d'image[4] le soin de déterminer la visibilité des surfaces. Le *Z-buffer* est une mémoire, "un tampon des profondeurs" qui sert à stocker la profondeur de chaque pixel visible à l'écran. Durant le rendu, la profondeur d'un élément candidat à paraître dans le pixel (fragment) est comparée à la valeur de la profondeur déjà stockée pour ce pixel : si cette comparaison indique que le nouveau fragment est devant celui stocké, alors celui-ci remplace le contenu du pixel écrit dans la mémoire image et le *Z-buffer* est mis à jour. La complexité de cet algorithme est proportionnelle aux nombres de primitives traitées et à leur surface à l'écran ; pour en améliorer encore l'efficacité, Greene propose une version hiérarchique dans [GK93].

De nombreuses extensions à cet algorithme existent, la plus intéressante est peut être le *A-buffer* [Car84] qui conserve pour chaque pixel une liste des fragments candidats à y apparaître et calcule pour chaque fragment la proportion du pixel qu'il recouvre. Ceci permet de traiter les problèmes d'aliassage et d'intégrer des objets semi-transparents.

Une autre amélioration [HA90] est le tampon d'accumulation (*accumulation buffer*) qui permet d'accumuler plusieurs images en déplaçant légèrement la caméra entre chaque rendu, soit pour réduire l'aliassage, soit pour obtenir un effet de flou de mouvement (*motion blur*) ou de mise au point (profondeur de champ).

L'autre solution pour réduire l'aliassage est de subdiviser les pixels. La couleur finale du pixel est obtenue en moyennant les sous-pixels. À noter que cette technique revient à effectuer le rendu à une résolution supérieure de celle de l'image (sur-échantillonnage).

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

Un des avantages du *Z-buffer* est la possibilité de réaliser facilement des implémentations matérielles très efficaces. Aujourd'hui les moteurs graphiques à base de *Z-buffer* permettent le rendu et l'affichage de scènes de plusieurs centaines de milliers de polygones en temps réel. Diverses techniques ont été développées pour obtenir des ombres à l'aide de cartes d'ombre (*cf.* section 2.2.2), ou à l'aide de volumes d'ombre [Cro77, Ber86, Sla92, McC00].

2.1.2 Lancer de rayons

Le principe du lancer de rayons (*ray-tracing*) [Whi80, GAC+89] est de calculer les objets visibles en remontant le chemin de la lumière parvenant à la caméra, *i.e.* en lançant des rayons à travers tous les pixels de l'image (fonctionnement inverse d'un appareil photo). Comme le principe du lancer de rayons est très simple à programmer, on peut lui faire simuler toute l'optique géométrique et ainsi prendre en compte de nombreuses caractéristiques optiques comme la réflexion, la réfraction, l'ombrage, etc. Notamment les reflets de la scène sur une surface lisse sont obtenus en envoyant un rayon secondaire depuis la surface dans la direction miroir à celle d'arrivée par

[4] Une mémoire d'image sert à stocker les attributs (couleur) de chaque pixel de l'espace image.

rapport à la normale. Comme le *Z-buffer*, la deuxième partie du processus consiste alors à calculer la couleur que vont avoir les objets en faisant appel à un modèle d'illumination (*cf.* chapitre 2 section 1.1) pondéré par l'éclairage (*i.e.* l'ombrage). Le principe de calcul des ombres est identique : pour savoir si un objet est éclairé, on lance un rayon d'ombre vers les sources de lumière.

Le coût de cet algorithme est important : pour chaque pixel de l'écran il est necessaire de lancer un rayon, voir plusieurs avec la technique de sur-échantillonage (*cf.* section 2.1.2). Avec un lancer de rayons non optimisé, pour chaque rayon on calcule son intersection avec toutes les primitives de la scène. De plus, pour chaque primitive intersectée d'autres rayons sont lancés (réflexions, ombres, etc.). Pour diminuer le nombre d'intersections à calculer, et donc le coût, il est indispensable de recourir à des techniques d'optimisations (*cf.* section 2.1.2).

Un des problèmes important que doit traiter le moteur de rendu est l'aliassage. Un arbre à lui seul compte plusieurs centaines de branches et plusieurs milliers de feuilles ; on peut donc imaginer la quantité de primitives que compte une forêt et donc le nombre moyen de primitives par pixel (le pire étant à l'horizon). Lors du rendu d'une image il est évident que toutes les primitives ne sont pas visibles explicitement, par contre elles contribuent toutes à l'aspect global de la scène (par exemple les aiguilles d'une forêt de sapins ne sont pas toutes visibles individuellement mais si on les supprime il ne reste que les branches !). Plusieurs centaines de primitives contribuent à la couleur d'un pixel, il faut donc, à défaut de toutes les traiter, en considérer suffisamment pour diminuer au maximum les problèmes d'aliassage.

Sur-échantillonage

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

Lancer de faisceaux

L'autre solution au problème d'aliassage est de calculer l'intégrale de l'ensemble des objets se trouvant dans le pixel : c'est ce que se propose de faire le lancer de faisceaux [HH84] (*beam-tracing*) ou de cônes [Ama84] (*cone-tracing*) en lançant un rayon pyramidal ou conique par pixel. À noter que le lancer de rayons avec sur-échantillonnage, que nous avons vu au paragraphe précédent, est une approximation numérique de cette intégrale.

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

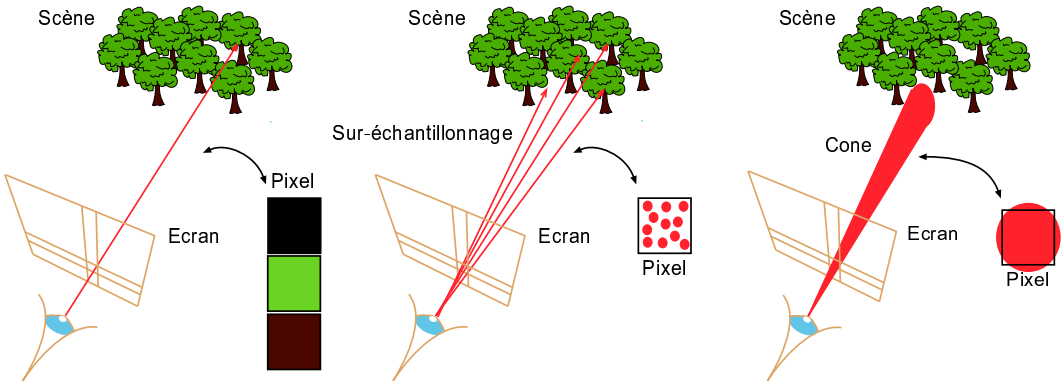

FIG. 1.7 – De nombreuses primitives se projettent dans un pixel. *À droite* : en ne lançant qu'un rayon par pixel la couleur déterminée sera aléatoire (aliassage). *Au milieu* : la solution est d'échantillonner suffisamment le pixel en lançant de nombreux rayons. *À droite* : on lance un unique rayon conique pour considérer tous les objets qu'il contient.

de l'illumination s'arrête à l'intersection de l'objet le plus proche de l'œil, alors que dans le cas du *cone-tracing*, on doit tenir compte de la surface du pixel recouverte par la primitive, et continuer avec les autres primitives se trouvant derrière s'il reste une portion non couverte.

Le sur-échantillonnage est plus simple à implémenter que le *beam-tracing*, mais pour une scène comportant de nombreuses primitives, petites par rapport à la taille d'un pixel une fois projetées, la technique de *beam-tracing* donnera de meilleurs résultats en terme d'aliassage et sûrement même en terme de coût de calcul.

Optimisations

Les techniques d'optimisation du lancer de rayons sont nombreuses. Le principe général est la subdivision de l'espace objet avec pour but la limitation du nombre d'intersections rayon-objet à calculer. Les deux principales techniques sont les volumes englobant (sphères ou boîtes) et les grilles, éventuellement récursives (octree) [SB87, JW89]. De nombreux travaux portent sur ce domaine, pour un éventail de toutes ces techniques, se référer à [GAC+89, Hai].

2.1.3 Rendu volumique par lancer de rayons

Le rendu volumique est destiné à calculer une image 2D d'une structure de données représentée par une grille 3D (chaque case étant appelée un voxel). Un exemple typique de données est une grille où chaque voxel contient une densité (*e.g.* une densité de tissus humains).

La technique classique de rendu est basée sur le principe du lancer de rayons que nous avons vu précédemment. Le rayon parcourt le volume de données en utilisant un algorithme de tracer de droite discrète et accumule au fur et à mesure la transparence, ainsi que la couleur si celle-ci est une donnée de la grille. Pour obtenir la quantité de lumière que reçoit chaque voxel il faut lancer un rayon vers chaque source de lumière (comme dans le cas du lancer de rayons classique).

Nous verrons à la section 2.5.1 du chapitre 2 qu'il existe une technique de rendu volumique plus efficace utilisant un rendu par tranches.

2.1.4 Cas des terrains

Nous avons vu à la section 1.1.2 qu'une représentation commode d'un terrain est la carte d'élévation (*displacement map*), mais de nombreux moteurs de rendu préfèrent traiter des polygones, il

faut alors convertir la carte en géométrie au moment du rendu.

D'autre part, il existe des techniques dites d'*inverse displacement* [PHL91, Tai92], sorte de lancer de rayons adapté, gérant directement l'intersection entre le rayon et la représentation, sans expliciter la carte d'élévation en facettes. Cela permet en outre de le faire efficacement, en procédant de manière hiérarchique : le principe consiste à organiser les données en quadtree et à pré-calculer dans chaque case l'altitude maximale. On a alors la garantie que le rayon ne coupe pas cette portion de terrain si ses points d'entrée et de sortie dans la case sont plus hauts que ce maximum, sinon, on repose le problème sur les quatre cases filles, et ainsi de suite. Quand on atteint la résolution des données, il faut tester l'intersection du rayon et de la surface dans la case en la calculant vraiment, ce qui n'arrive que pour peu de cases.

Une autre technique [PH96], non spécifique au lancer de rayons, basée sur un système de cache, permet de ne pas expliciter toute la géométrie : les polygones sont générés au vol, on ne convertit que les données utiles dans la zone en cours de traitement. On utilise un système de cache pour éviter de générer des polygones à plusieurs reprises.

FIG. 1.8 – *À gauche* : deux images de Lindstrom [LKR+96] illustrant un maillage fin et un maillage grossier que l'on peut utiliser comme niveaux de détails. *À droite* : une image de Premoze [PTS99] illustrant la possibilité d'éclairer un terrain montagneux à différents moments de l'année.

Pour en finir avec le rendu de terrains, nous citons un article de Premoze [PTS99] permettant de synthétiser un paysage de montagne à différents moments de l'année et de la journée. La texture initiale du terrain est extraite de photos. Puis, la méthode permet de simuler la tombée et la fonte de neige ainsi que les différentes illuminations que peut prendre le terrain en fonction de sa composition et de l'éclairage (*cf.* figure 1.8 à droite).

2.2 Visibilité

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

Nous avons vu dans les paragraphes traitant de la modélisation qu'un paysage peut compter des milliards de polygones, ce qui est bien trop pour un moteur de rendu actuel, même câblé par le matériel d'une carte graphique. D'autre part, la majorité des ces primitives n'est pas visible au final car dans le dos de l'observateur, derrière une montagne, derrière un arbre, cachés en arrière plan, etc. Seulement, les supprimer automatiquement est un problème difficile pour un ordinateur. La

visibilité offre donc des algorithmes permettant de n'envoyer au moteur de rendu que les primitives visibles.

Nous proposerons au chapitre 6 une structure pré-calculant l'information de visibilité pour le calcul des ombres. En 2.2.1, nous parlerons donc des techniques de visibilité en général, nous verrons en 2.2.2 la technique des *shadow map* permettant le calcul rapide de la visibilité pour les ombres, puis nous finirons en 2.2.3 avec des techniques de visibilité spécifiques aux terrains.

2.2.1 Généralité

Le but ici n'est pas d'être exhaustif à propos des travaux sur la visibilité, domaine très actif en ce moment. Un état de l'art complet peut être trouvé dans [Dur99].

Historiquement, la suppression des objets cachés est une des grandes motivations des algorithmes de visibilité (*cf.* Jones [Jon71] ou Clark [Cla76]). Les techniques les plus simples sont la suppression des objets se trouvant en dehors du champ de la caméra (*frustum culling*) qui permet de supprimer jusqu'à 75%, voir 80% des primitives, et la suppresion des parties d'un objet se trouvant à l'arrière (*backface culling*). Les techniques plus complexes capables de supprimer les objets cachés par d'autres (*occlusion culling*) sont souvent basées sur un pré-calcul de structures de visibilité (*e.g.* cartes), et permettent de supprimer jusqu'à 95% des objets avec un coût consacré aux tests bien moindre (ceci dépend bien sûr de la configuration de la scène et de la position de l'observateur). Airey [ARB90] et Teller [TS91] furent les premiers à proposer de pré-calculer la visibilité dans des environnements architecturaux intérieurs. Ils exploitent le fait que si l'on est dans une pièce, on ne voit les autres que par les portes ou fenêtres, *i.e.* en construisant une structure de vues (portails) qui prend en considération les zones visibles. Ces techniques pour ce type de scènes sont très efficaces, mais grandes consommatrices de mémoire. Des algorithmes capables de traiter des scènes d'extérieur ont fait leur apparition ces dernières années [COFHZ98], où la notion d'occludeurs est utilisée pour éliminer les zones cachées à la vue. Ces algorithmes sont basés sur la notion de cellules de visibilité : l'espace est découpé en octree et, pour chaque cellule (voxel), on pré-calcule les parties visibles de la scène. Ces techniques fonctionnent correctement lorsque les occludeurs sont de grandes tailles, ce qui n'est pas le cas des arbres où les occludeurs (*i.e.* feuilles) sont petits et nombreux. Durand [DDTP00] et Schaufler [SDDS00] introduisent la notion de fusion de plans d'occlusion, qui permet de grouper les zones contiguës cachées par chaque occludeur (*e.g.* feuille) en des zones plus grandes.

2.2.2 Cartes d'ombres

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

La technique des cartes d'ombres (*shadow map*), introduite par Reeves en 1987 [RSC87], pré-calcule la visibilité depuis la source de lumière dans une carte, en effectuant un rendu de la scène depuis cette source et en conservant le tampon de profondeur. Au moment du rendu d'un objet on calcule la distance entre l'élément vu dans le pixel et la source de lumière. Si cette distance est supérieure à celle stockée dans la *shadow map* alors l'objet est à l'ombre, sinon il est à la lumière.

Lorsque la source de lumière change de position ou lorsqu'un objet bouge, il faut recalculer la carte, ce qui oblige à effectuer deux rendus : un depuis la source de lumière, et un depuis l'œil. Grâce aux performances des cartes graphiques actuelles le temps réel est possible, d'autant que

la plupart des effets peuvent être désactivés pour la construction de la *shadow map* (illumination, texture, etc.) et les niveaux de détails grossiers utilisés. À noter que cette technique ne fonctionne qu'avec des objets opaques, car la carte ne donne que l'objet le plus proche de la source. Un autre inconvénient de cette technique est que la résolution de la texture, ainsi que la dynamique limitée des profondeurs stockées, restreint la précision des ombres. Seules certaines cartes graphiques de haut niveau (Onyx^2 Infinite Reality) disposent de cette fonctionnalité.

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

Des techniques ont été développées pour combiner les aspects positifs des *shadow maps* et des *alpha shadow maps*. C'est-à-dire être capable de gérer des objets transparents ou une densité d'objets très importante par rapport à la résolution de la texture, sans avoir à séparer les occludeurs des objets ombrés. Lokovic [LV00] calcule l'ombrage et l'auto-ombrage d'une chevelure, dont les primitives sont à une résolution trop fine pour la méthode de base. Il découpe l'espace en intervalles de profondeur auxquels il associe une carte. Il appelle l'ensemble *cartes d'ombres en profondeur* (*deep shadow map*). Son implémentation est complètement logicielle et n'est donc plus temps réel. Tae-Yong [KN01] en accélère le calcul en utilisant le matériel graphique, sans pour autant parvenir à atteindre le temps réel. Soler [SS98] propose une technique pour calculer une *alpha shadow map* contenant des ombres douces. Il convolue à l'aide du matériel graphique des images des occludeurs prises depuis différentes positions de la source de lumière.

2.2.3 Cas des terrains

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

Max introduit la notion de carte d'horizon (*horizon map*) dans [Max88] pour ajouter de l'ombrage à une surface dont les normales sont représentées par une carte de perturbations (*bump-map*[5]). L'idée est de calculer la visibilité du ciel à chacun des points échantillonnés de la surface. Max considère huit directions autour de chacun de ces points et détermine l'horizon en utilisant les points échantillonnés (*cf.* figure 1.9 à gauche). Durant le rendu, un point est considéré comme illuminé si le soleil est visible de ce point : pour cela, il utilise la direction échantillonnée la plus proche. La pénombre est calculée à partir de la quantité du disque solaire vu depuis le point. La limitation de cette méthode est le sous-échantillonnage des directions de visibilité qui entraîne des artefacts.

Cabral dans [CMS87] utilise des cartes d'horizon similaires pour calculer la *Bidirectional Reflectance Distribution Function BRDF* (*cf.* section 1.1.6) d'une carte de perturbation des normales (*bump-map*). Il utilise 24 directions de vue et projettent les triangles du terrain à la place des points

[5] Le *bump-mapping* est la perturbation des normales d'une surface sans changer la géométrie (*cf.* chapitre 2 section 1.1.2). Le plus souvent on utilise une carte de normales comme texture de la surface.

lors du calcul de la visibilité, ce qui diminue le sous-échantillonnage mais ne le fait pas disparaître entièrement.

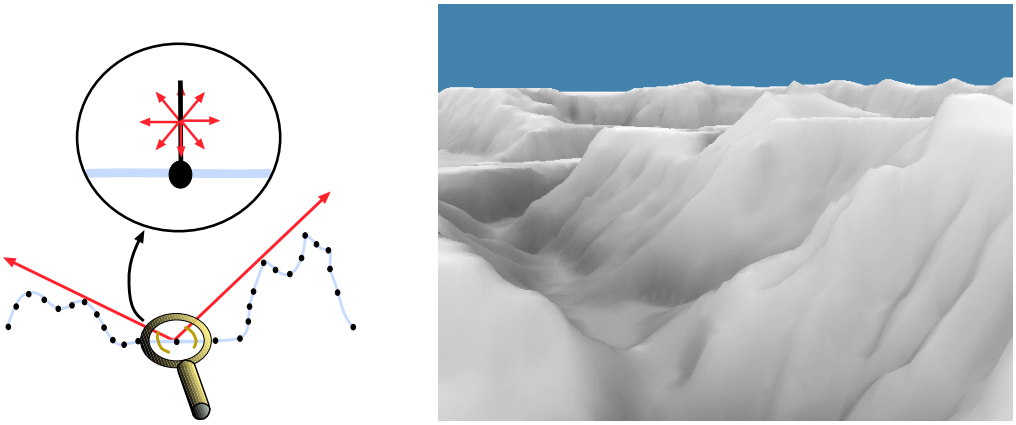

FIG. 1.9 – Cartes d'horizon. *À gauche* : pour chaque point échantillonné de la surface, Max [Max88] calcule dans 8 directions la hauteur de l'horizon. *À droite* : Stewart [Ste98] augmente la précision des cartes d'horizon et les utilise pour calculer l'ombrage et accélérer le rendu en n'envoyant à la carte graphique que les parties visibles du terrain.

Cohen-Or [COS95] utilise une technique d'échantillonnage similaire pour découper le terrain en partie visible et partie invisible dans le but de limiter le nombre de primitives envoyées au moteur de rendu. Stewart dans [Ste97, Ste98] augmente la précision de la méthode en considérant tous les points de la surface pour le calcul de la visibilité (*cf.* figure 1.9 à droite), alors que Max ne considérait que huit directions. En utilisant une hiérarchie il introduit un algorithme rapide lui permettant de calculer les cartes d'horizon rapidement, même pour un très grand nombre d'échantillons.

3 Bilan

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

Les algorithmes de simplification de maillage sont capables de générer efficacement des niveaux de détails pour les terrains, mais leur application aux arbres n'est pas convaincante, notamment à cause de la nature fouillée et dispersée des feuilles. Même si les techniques de visibilité diminuent le nombre de polygones envoyés au moteur de rendu et pré-calculent l'information pour l'affichage des ombres, il semble difficile de traiter rapidement des paysages composés d'à peine quelques dizaines d'arbres.

Il est évident, au regard de ces techniques, qu'il y a avant tout un problème de représentation, au-delà de la simple hiérarchisation qui est déjà poussée à bout dans toutes les techniques actuelles.

CHAPITRE 2

Traiter la complexité

Au cours du chapitre précédent, nous avons traité des représentations dédiées à la synthèse de paysages (*i.e.* terrains et végétaux). Le principal problème qui découle de celles-ci est le traitement des milliards de primitives générées, que le rendu doit traiter. La construction de niveaux de détails par simplification de maillage améliore la situation pour les terrains, par contre, leur utilisation est impossible pour les végétaux du fait de leur structure dispersée. Cette lacune doit être comblée par d'autres chemins, *i.e.* il nous faut trouver d'autres méthodes capables de représenter la complexité de manière plus efficace que les polygones.

L'objectif de ce chapitre est donc de présenter diverses représentations existantes traitant la complexité en synthèse d'images sous toutes sortes de formes (rugosités, fourrures, forêts, nuages, etc). Ce panel de techniques aux approches originales et variées (formes analytiques, octrees, points, textures 3D, etc.) offre des solutions à de nombreux problèmes ; nous verrons cependant qu'il n'existe pas une solution unique traitant tous les problèmes. Bien qu'elles ne soient pas toutes en rapport direct avec la synthèse de paysages, ces techniques présentent un concept ou une philosophie utile à la compréhension des "contributions" que nous présenterons durant cette thèse.

Le plan de ce chapitre est donné par l'échelle des phénomènes à représenter : nous traiterons à la section 1 des complexités microscopiques, puis nous détaillerons à la section 2 les modèles dits *alternatifs*, traitant de complexité à échelle macroscopique.

1 Complexité microscopique

La quête du réalisme fait que l'on cherche à enrichir l'apparence visuelle des surfaces, par exemple en essayant de représenter la rugosité. Le polygone est la primitive de base en synthèse d'images, mais représenter une surface granuleuse par des milliards de micro-polygones n'est pas viable : on a donc cherché à représenter cette complexité à l'aide de représentations mieux adaptées. De nombreux travaux fournissent des formules analytiques ou des tables donnant l'illumina-

tion d'une surface, chacun tenant compte d'une caractéristique nouvelle par rapport au précédent [FvDFH90].

La même philosophie de conception est aussi utilisée pour l'illumination des volumes, par exemple, pour représenter les milieux participatifs comme les nuages, ou les milieux denses en géométrie comme les poils d'une fourrure. Dès que la géométrie est trop complexe au point qu'il ne soit plus raisonnable de la représenter explicitement ou dès que le phénomène n'a pas de surface définie, il devient nécessaire de concevoir des représentations plus efficaces, souvent basées sur une intégration analytique. La conception de ces représentations est le résultat d'une modélisation mathématique poussée, souvent inspirée de techniques que la physique a déjà mises au point et qui fournissent un bon point de départ. Ces méthodes nous intéressent, car cette philosophie de représenter la complexité par une intégration analytique sera aussi la notre, dans la première contribution de cette thèse (*cf.* partie II).

Nous traiterons en 1.1 des modèles d'illumination de surfaces, puis nous décrirons en 1.2 les modèles d'illuminations volumiques.

1.1 Modèles d'illumination d'une surface

Les modèles d'illumination de surface (*shaders*) sont nombreux ; nous ne nous intéresserons qu'à quelques-uns d'entre eux, plus directement reliés aux représentations que nous présenterons dans les contributions de cette thèse. Nous verrons :

- en 1.1.1 le modèle empirique mais classique de Phong qui est souvent le modèle de départ lorsque l'on cherche à intégrer l'illumination totale d'un objet complexe ;
- en 1.1.3 le modèle de Torrance et Sparrow, qui fut un des premiers à intégrer analytiquement les micro-facettes d'une surface ;
- en 1.1.4 divers modèles d'illumination analytiques ;
- en 1.1.5 le modèle anisotropique de Poulin, qui se base sur l'intégration analytique de l'illumination d'une surface couverte de micro-cylindres, problématique proche de ce que nous allons présenter par la suite ;
- en 1.1.6 la fonction de distribution de la réflectance bidirectionnelle (*bidirectionnal réflectance distribution function BRDF*) ;
- en 1.1.7 la notion de fonction bidirectionnelle de texture qui est utilisée pour traiter l'illumination de surfaces à partir de mesures réelles ;
- en 1.3.2 le modèle de Max permettant la transition douce entre différents niveaux de *bump*.

1.1.1 Modèle d'illumination de Phong

Le modèle d'illumination de surface le plus connu, car le plus utilisé en synthèse d'images, est certainement celui de Phong [FvDFH90]. Ce modèle empirique et simple décrit le calcul de la couleur d'un objet en fonction de sa couleur intrinsèque et des sources lumineuses. Plus précisément, il ajoute au modèle diffus de Lambert une contribution spéculaire représentant les reflets de la lampe calculés en fonction de la normale, et des directions de vue et de lumière :

$$\begin{aligned} I &= \mathrm{ambient} + \mathrm{diffus}_{\mathrm{Lambert}} + \mathrm{speculaire}_{\mathrm{Phong}} \\ I &= I_a k_a + I_p (k_d (N.L) \mathbb{1}_{N.L>0} + k_s (R.V) \mathbb{1}^n_{R.V>0}) \end{aligned}$$

avec N la normale à la surface, L la direction de la lumière, V la direction de vue, R le vecteur symétrique de V par rapport à la normale N (*cf.* figure 2.1 à gauche) et n le coefficient de spécularité,

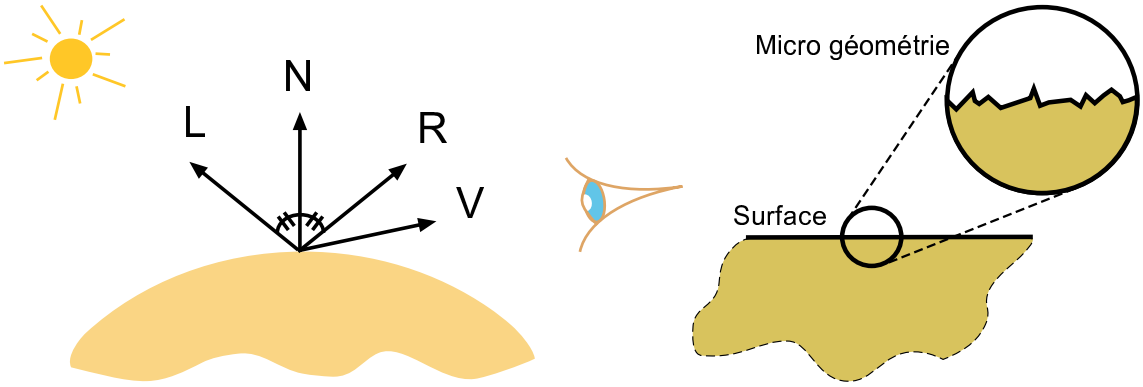

FIG. 2.1 – *À gauche* : les données pour le modèle d'illumination empirique de Phong [FvDFH90]. *À droite* : la micro-géométrie d'une surface.

inversement proportionnel à la rugosité.

Blinn a apporté une formulation alternative de l'illumination de Phong qui utilise le vecteur moyen H du vecteur de vue V et du vecteur de lumière L. (H est aussi appelé le vecteur de direction maximum de spécularité). Le terme de réflexion spéculaire, utilisé en pratique, s'écrit alors $(N.H)^n \mathbb{1}_{N.H>0}$.

Ce modèle a l'avantage d'être simple, facile à implémenter, supporté de nos jours par toutes les cartes graphiques 3D[1], et de donner des résultats visuellement acceptables même si inexacts, ce qui explique son succès.

1.1.2 Cartes et textures de perturbations

Une façon d'augmenter la complexité visuelle tout en minimisant la complexité des traitements est d'utiliser une texture contenant l'image des détails [Cat74, BN76]. Une texture est un tableau 2D que l'on plaque (*mapping*) sur une surface. Dans la version la plus simple, ce tableau contient des couleurs (voir de la transparence) pour former, soit une image unique qui vient s'appliquer sur toute la surface, soit un motif que l'on va répéter pour couvrir celle-ci.

Blinn représente la complexité d'une surface sans la modéliser par des polygones, en introduisant la notion de texture de paramètres de Phong [Bli77] qui permet par exemple, de simuler différents matériaux ou les variations de rugosité d'une plaque de métal tout en utilisant un unique polygone. D'autre part, il introduit l'idée de texture de perturbation de normales (*bump-mapping*) [Bli78], qui revient à perturber les normales pour simuler l'apparence d'une surface non plane sans augmenter le nombre de polygones (*cf.* figure 2.2). Comme aucun relief n'est réellement créé avec cette technique, il n'y a pas d'ombres portées, et la silhouette de l'objet reste lisse.

Un polygone texturé peut se voir soumis à l'aliassage si la résolution de la texture se trouve être plus fine que celle de l'écran. Dans ce cas l'aliassage est plus simple à traiter que pour les polygones (*cf.* chapitre 1 section 2.1.2), car on connaît le voisinage. Williams [Wil83] introduit la technique du *mipmapping*, il pré-calcule la texture à diverses résolutions et, lors du rendu, utilise celle qui est adaptée à la distance, *i.e.* dont le pixel de texture est le plus proche du pixel de l'écran.

[1]Ce modèle est évalué aux sommets des polygones par les cartes graphiques classiques (*i.e.* ne disposant pas de la fonctionnalité d'illumination par pixel). Les valeurs des couleurs en chaque pixel sont alors interpolées lors du remplissage du polygone.

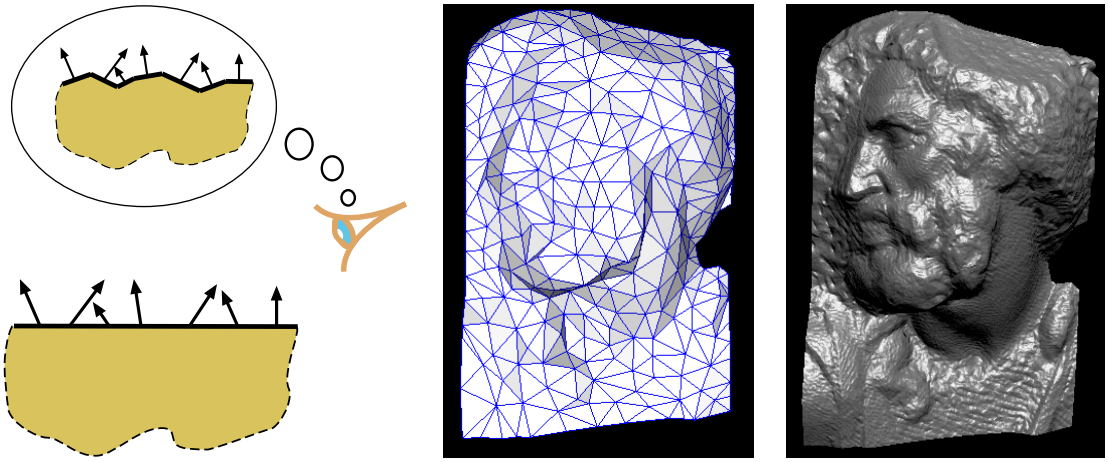

FIG. 2.2 – Carte de perturbations de normales (*Bump-mapping*). *À gauche* : seules les normales à la surface sont perturbées pour donner l'illusion d'un relief authentique. *Au milieu* : le maillage nu. *À droite* : le maillage enrichi par la technique de *bump-mapping*, donnant l'illusion d'un relief fin.

1.1.3 Intégration analytique des micro-facettes d'une surface

Rapidement, des modèles plus complexes et plus proches de la réalité (au sens physique du terme) ont été développés : un de ces premiers modèles, issu de la physique, fut celui de Torrance et Sparrow [TS66, TS67], introduit par Blinn et Cook en synthèse d'images dans [CT82, Bli77]. C'est un modèle analytique basé sur l'idée qu'une surface isotrope est constituée de micro-facettes parfaitement réfléchissantes et distribuées aléatoirement. Torrance et Sparrow estiment statistiquement la réflexion de la lumière dans une direction donnée. Ce type de surface est qualifié d'isotrope car l'intensité de la lumière réfléchie est indépendante de l'orientation de la surface par rapport à la normale, ce qui ne sera pas le cas des situations que nous allons envisager maintenant.

1.1.4 Divers modèles d'illuminations

De nombreux autres modèles d'illumination analytiques existent. Notamment, le modèle anisotrope de Kajiya [Kaj85] basé sur une nouvelle intégration des formules de Beckmann de diffusion d'une surface rugueuse ; le modèle de Lewis [Lew94] qui essaie de rendre les modèles d'illumination classiquement utilisés en synthèse d'images physiquement réalistes en les comparant à des critères ou des données réelles (*BRDF*) ; le modèle de Schlick [Sch94a] dont la paramétrisation forte permet de représenter toutes sortes de caractéristiques (isotrope, anisotrope, homogène, etc.) ; une généralisation anisotrope du modèle de Lambert par Oren qui représente une surface par de nombreuses surfaces lambertiennes [ON94], etc. On trouvera une synthèse de ceci dans l'état de l'art de Schlick [Sch94b].

1.1.5 Modèle d'illumination anisotrope de Poulin

Poulin dans [PF90] propose un modèle analytique anisotrope de réflexion et réfraction basé sur l'utilisation de micro-cylindres. L'anisotropie est représentée par de petits cylindres distribués sur la surface, pour simuler des rayures ou des fils collés. Différents niveaux d'anisotropie peuvent ainsi être obtenus en faisant varier la distance entre deux cylindres ou en plaçant plus ou moins haut la surface par rapport aux axes (*cf.* figure 2.3).

Cette fonction de réflectance anisotrope est évaluée dans [PF90] par le calcul analytique de l'intégrale du modèle d'illumination de Phong sur les cylindres et le plan. Cette approche analytique est calculée de manière exacte pour le terme diffus et avec une approximation pour le terme spéculaire.

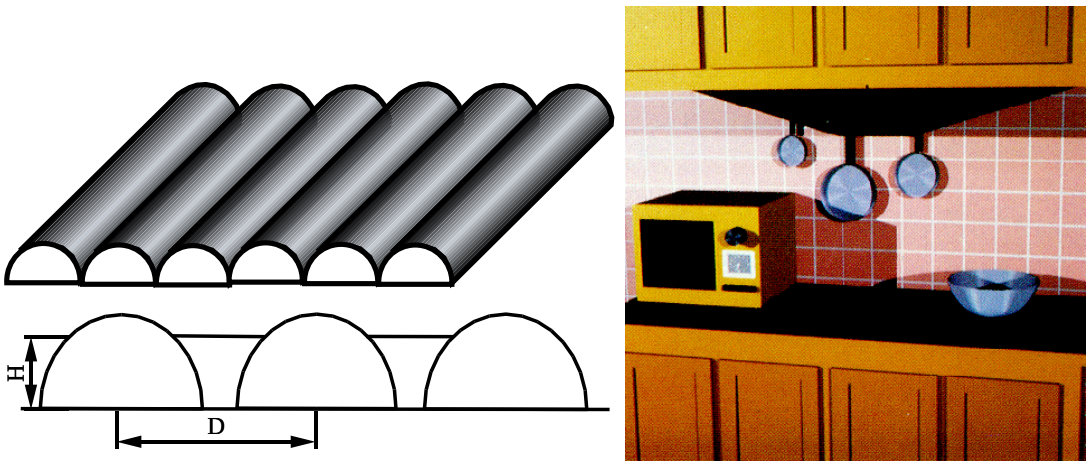

FIG. 2.3 – Modèle d'illumination anisotrope de Poulin [PF90]. *À gauche* : le degré d'anisotropie est contrôlé par H et D. *À droite* : un exemple d'illumination anisotrope (fonds des casseroles).

1.1.6 Fonction de distribution de la réflectance bidirectionnelle (BRDF)

Pour reproduire la complexité d'une surface visible par sa réflectivité, une autre approche consiste à recourir à des mesures réelles du comportement optique d'un matériau, obtenues au moyen d'un gonioréflectomètre. Il y a potentiellement une valeur de réflexion différente pour chaque direction incidente de lumière et chaque direction d'observation, chacune repérée par deux angles. On obtient ainsi une table à quatre dimensions représentant la fonction de distribution de la réflectance bidirectionnelle (*bidirectionnal reflectance distribution function BRDF*). De nombreux travaux portent sur ce domaine, voir [Rus97] pour plus de détails.

De nombreux modèles d'illumination sont basés sur cette technique des *BRDF*, comme le modèle de Gondek [GMN94] dépendant de la longueur d'onde, le modèle de Westin [WAT92] qui approchent une *BRDF* par une méthode simulant la diffusion (*i.e.* Monte Carlo) sur la géométrie, etc.

1.1.7 Fonction bidirectionnelle de texture (BTF)

Dana dans [DvGNK99] propose d'utiliser une base de données [DvGNK] d'images de surfaces réelles photographiées sous toutes les conditions, associée à des techniques de traitement d'images pour en reconstruire la fonction de réflectance. La fonction bidirectionnelle de texture (*Bidirectionnal Texture Function BTF*) qu'introduit Dana associe à un couple (direction de vue, direction de lumière) une image. Cette fonction, qui est formellement 6D, peut être construite en échantillonnant n directions de vue et m directions de lumière (*cf.* figure 2.4).

Lors de notre deuxième contribution (*cf.* partie III) nous réutiliserons l'idée de prendre des images d'un objet depuis plusieurs points de vue, avec pour chacun de ces points de vue différentes positions de lumière.

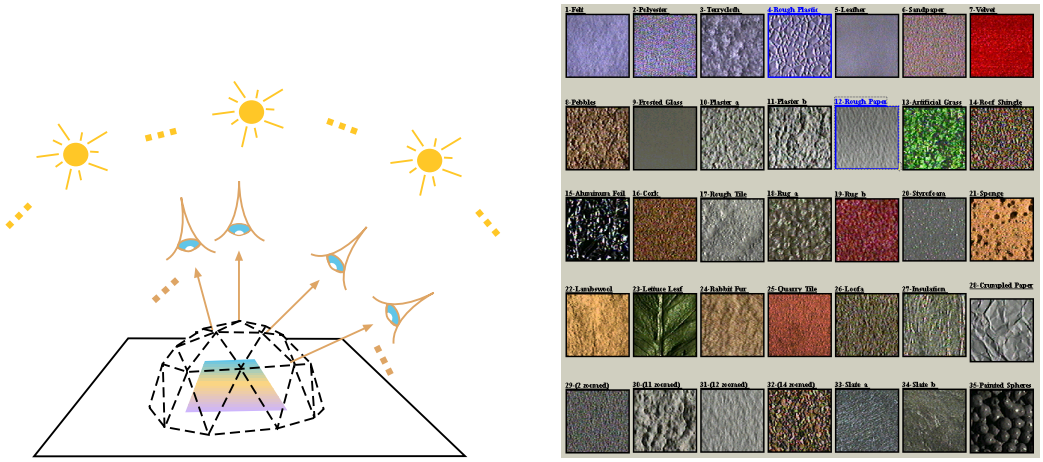

FIG. 2.4 – Fonction bidirectionnelle de texture (*BTF*) [DvGNK99]. *À gauche* : la surface est prise en photo depuis plusieurs points de vue et avec plusieurs directions de lumière. *À droite* : un exemple de surfaces réelles que propose la base de donnée des Universités de Columbia et Utrecht [DvGNK].

1.1.8 Transitions continues entre représentations des détails

En complément de la section 1.3.2 du chapitre 1, qui traitait de la transition entre les représentations, nous abordons ici le cas des transitions entre modèles de *bump*, proposé par Becker et Max dans [BM93].

Cabral et Max [CMS87] utilisent les cartes d'horizon (*cf.* chapitre 1 section 2.2.3) pour apporter l'information de visibilité manquante aux cartes de *bump-mapping* et ainsi tenir compte des interactions du voisinage, même pour des surfaces enrichies par le *bump mapping*.

Les techniques de *BRDF* et de cartes de déplacement (*displacement map*) permettent de tenir compte de l'auto-ombrage, puisque celui-ci est une propriété intrinsèque de ces modèles. Dans [BM93] Becker et Max utilisent ces deux représentations, plus celle du *bump mapping* enrichie de l'auto-ombrage que nous venons de voir, comme différents niveaux de détails. Ils proposent une méthode de transition efficace en tenant compte de la distance à l'œil, de l'angle de vue et de la fréquence de la *bump-map* pour choisir la bonne représentation.

Plus récemment Heidrich [HS99] propose une solution pour implémenter ces techniques de *bump* en temps réel, en utilisant les cartes graphiques standards avec un rendu multi-passes. Toujours en s'aidant du matériel graphique, il apporte une solution temps réel à l'absence d'auto-ombrage du *bump-mapping* [HDKS00].

1.2 Modèles d'illumination volumique

Simuler l'illumination de gaz (nuages, brouillard, etc.) en représentant explicitement les millions de particules qui le composent n'est pas viable, même si une discrétisation grossière peut être utilisée à grande échelle (*e.g.* grille volumique). En revanche, le calcul d'un modèle intégrant analytiquement cette complexité à petite échelle sera efficace. Il en va de même pour de nombreux autres phénomènes complexes comme le ciel, la fourrure, etc.

Le principe d'intégration analytique de l'illumination d'un volume est similaire au cas d'une surface : on intègre un modèle d'illumination simple au sens mathématique du terme sur toute la géométrie du volume en tenant compte de la visibilité et de l'auto-ombrage (qui sont encore plus importants pour un volume que pour une surface).

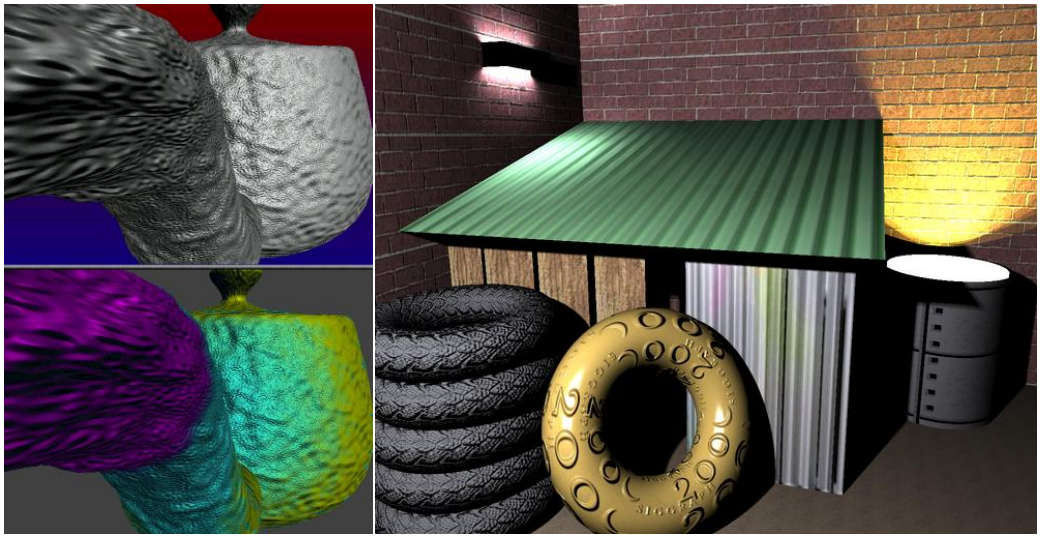

FIG. 2.5 – *À gauche* : transition douce entre les trois modèles : *displacement mapping* en rouge (vue de près), *bump-mapping* en bleu, et *BRDF* en jaune (vue de loin) [BM93]. *À droite* : Heidrich [HDKS00] ajoute les ombres et l'illumination indirecte au *bump-mapping* en temps réel.

Nous présentons en 1.2.1 les modèles d'illumination analytiques basés sur une distribution de particules, suivis en 1.2.2 par un modèle analytique de fourrure.

1.2.1 Modèles d'illumination basés sur une distribution de particules

Blinn dans [Bli82] fut l'un des premiers à proposer un modèle analytique d'illumination en milieu participatif. Son modèle est basé sur une répartition aléatoire dans l'espace de micro-sphères qu'il appelle particules. Il dispose de plusieurs paramètres comme l'épaisseur de la couche de particules, le nombre de sphères par unité de volume, le rayon d'une sphère, qui lui permettent de calculer la probabilité d'intersection entre un rayon et une particule, ainsi que la transparence globale de la couche. La brillance (albédo) et la fonction de phase définissent la transmission de la lumière par une particule en fonction des directions d'observation et d'éclairage. Il calcule analytiquement la quantité de lumière émise dans une direction donnée en ajoutant à la lumière diffusée par les particules, la lumière traversant le milieu (*cf.* figure 2.6).

Les modèles analytiques cherchant à représenter les gaz sont peu nombreux, outre [Bli82] que nous venons de voir, on citera [Sta94] où Stam propose un algorithme de rendu stochastique de gaz (nuages, fumée et feu) représenté par un champ de densité aléatoire. De nombreux autres travaux existent concernant la représentation de nuages mais ce sont pour la plupart des représentations basées sur la simulation. Pour plus d'informations sur les techniques de représentations de nuages, se reporter à [Ebe01, Pro].

Dans le même esprit que le modèle d'illumination analytique de Blinn [Bli82], on citera un modèle de Stam [Sta01] dédié à la représentation de la peau, en prenant en compte les diffusions multiples dans une tranche bornée par deux surfaces rugueuses. Il calcule d'abord une solution discrète aux équations de transfert radiatif, puis par mise en correspondance des courbes (splines) il construit le modèle d'illumination analytique qu'il applique au rendu de peau (*cf.* figure 2.6).

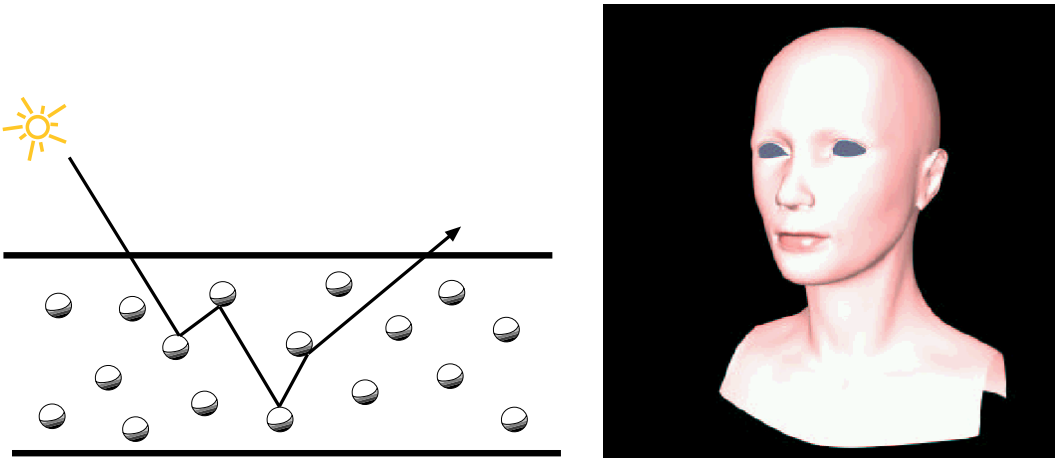

FIG. 2.6 – *À gauche* : Blinn [Bli82] modélise les nuages comme un espace rempli de particules aléatoirement distribuées puis en calcule analytiquement un modèle d'illumination. *À droite* : Stam [Sta01] construit un modèle analytique d'illumination de peau en mettant en correspondance les résultats discrets des équations de transferts radiatifs avec des courbes analytiques.

1.2.2 Modèle de fourrure : illumination d'un cylindre

Miller [Mil88] pré-calcule, dans une table, l'illumination d'un cylindre (à la manière d'une *BRDF*), mis à part qu'il profite de la symétrie axiale pour limiter la quantité de données à stocker. Son but est d'afficher des objets formés d'armatures arrondies semblables à des portions de cylindres. Il applique aussi cette méthode à la fourrure, les poils étant représentés par des cylindres (*cf.* figure 2.7).

Une année après les travaux de Miller, Kajiya et Kay [KK89] intègrent analytiquement l'illumination d'un cylindre, nous détaillerons leur modèle à la section 2.4. Poulin [PF90] avec son modèle d'illumination de surfaces (*cf.* section 1.1.5) utilise aussi l'intégration analytique de l´illumination d'un cylindre pour développer un modèle d'illumination anisotrope.

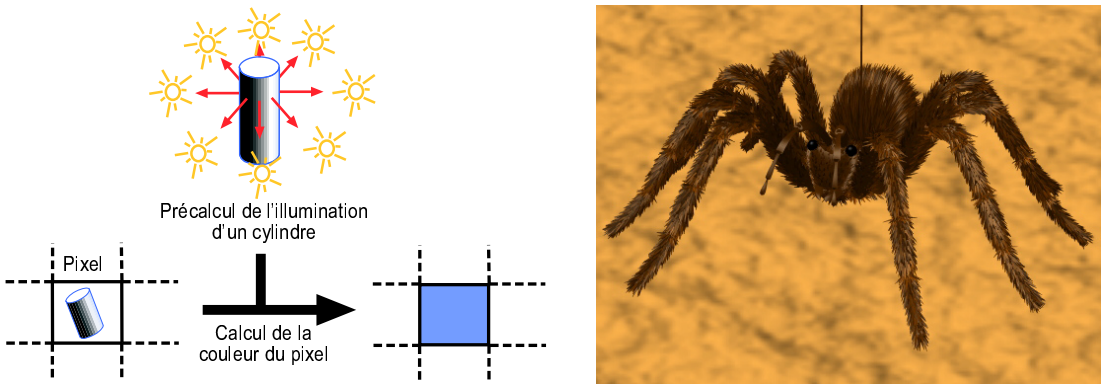

FIG. 2.7 – Illumination d'un cylindre [Mil88]. *À gauche* : Miller pré-calcule, dans une table, l'illumination d'un cylindre à la manière d'une *BRDF*, mis à part qu'il utilise la symétrie axiale d'un cylindre pour stoker un minimum de données. Plus tard Kajiya et Kay [KK89] puis Poulin [PF90] calculerons cette illumination analytiquement. *À droite* : "Borris l'araignée poilue" réalisé à l'aide du modèle de d'illumination d'un cylindre de Miller.

1.3 Bilan des modèles d'illumination

Nous avons vu différents modèles d'illumination surfaciques, puis différents modèles volumiques, lesquels traitent d'une certaine forme de complexité. Le point marquant de cet aperçu est que ces modèles utilisent très souvent une intégration de la complexité d'où découle une formule analytique. L'avantage de ce type d'approche est de fournir une formule prête à être utilisée, facilitant l'implémentation du modèle.

Bien sûr, toute complexité n'est pas toujours facilement intégrable, au sens mathématique du terme : intégrer requière de modéliser la répartition et l'effet des composantes (*e.g.* les particules d'un gaz) sous forme mathématique, ce qui suppose de disposer d'un cadre à priori, de faire des hypothèses, de se ramener éventuellement à une représentation statistique, pour ensuite trouver une forme analytique (éventuellement approchée) de l'intégrale. Le calcul formel, souvent complexe, doit se résoudre sans trop d'approximations, en faisant attention de ne pas faire disparaître ainsi les caractéristiques essentielles de ce que l'on cherche à modéliser. À cause de ces difficultés de modélisation et de réalisation du calcul formel nécessaire à l'élaboration du modèle analytique, on se contente souvent des techniques de discrétisation, plus faciles à mettre en place, mais souvent plus coûteuses en temps de calcul.

Les techniques que nous venons de voir conduisent à une formule ou à une structure de données simple correspondant à l'illumination d'un phénomène ou d'une géométrie. Cependant, il manque l'autre partie nécessaire à la représentation de l'apparence, *i.e.* les formes macroscopiques des objets. Si nous prenons l'exemple des gaz, nous disposons de modèles analytiques d'illumination assez efficaces, mais il nous manque les techniques de modélisation et de stockage de la forme du nuage. Nous allons voir maintenant des représentations que nous qualifions d'alternatives (aux représentations polygonales), qui combinent ces deux aspects pour donner une représentation complète.

2 Représentations alternatives

Les méthodes que nous avons décrites dans les paragraphes précédents avaient toutes pour but d'intégrer les effets de la complexité locale d'une surface ou d'un volume, souvent sous une forme analytique. Mais l'illumination ne suffit pas à représenter le phénomène à grande échelle, il faut l'associer à une forme, à laquelle l'illumination va s'appliquer : pour une surface le modèle d'illumination peut être associé à un polygone, mais pour représenter des phénomènes plus complexes (*e.g.* fourrure, arbres), il faut être capable d'en décrire l'apparence.

On représente les objets par des polygones car leur rendu est simple avec un lancer de rayons ou un *Z-buffer*. Les polygones sont mal adaptés aux objets volumiques (nuages) ou quasi-volumiques, tant la densité de détails est grande (fourrure, feuillage, herbe sur une prairie, etc.). Ils existent une série de représentations plus ou moins spécialisées, que nous qualifierons d'alternatives, qui se proposent de traiter la complexité de manière différente sans recourir nécessairement aux polygones.

Ces modèles se basent sur des représentations très variées, et ils ne traitent pas toujours de paysages. Cependant, ils offrent des solutions élégantes à la prise en charge efficace de la complexité. Cette famille de modèles a été une forte source d'inspiration pour la conception des techniques présentées dans la partie "contributions" de cette thèse ; par conséquent, il est important que nous les détaillions ici.

Nous verrons dans un premier temps l'utilisation de deux primitives très simples, le trait et le point en 2.1 et en 2.2 puis, nous survolerons les techniques utilisant l'image comme support

(*Image Based Rendering IBR*) en 2.3. Nous détaillerons le principe des textures volumiques en 2.4, ainsi que les techniques à base de tranches en 2.5, et nous finirons en 2.6 par une technique alternative représentant de la macro-géométrie par des textures (*e.g.* osier).

2.1 Système de particules

Les systèmes de particules [Ree83, RB85, Cha97], furent parmi les premiers modèles à ne pas utiliser les polygones comme primitives de base : cette technique est adaptée aux géométries complexes dont les détails fins peuvent être représentés par des trajectoires, comme l'eau vive, le feu ou les végétaux.

FIG. 2.8 – Système de particules [Ree83, RB85]. *À gauche* : l'illumination et l'auto-ombrage se calculent à partir des distances Da et Dd. *À droite* : un exemple de paysage généré par cette technique.

L'idée première [Ree83] est de représenter des géométries complexes ou difficilement représentables avec des primitives géométriques classiques, par une primitive très simple mais en très grands nombres : le trait. Les particules évoluent dans l'espace en subissant différentes influences (gravité, lois de croissance, etc.) et forment ainsi des trajectoires. Il n'y a pas d'interaction entre les particules dans le modèle initial.

Reeves et Blau présentent en 1985 [RB85] des applications des systèmes de particules pour la modélisation et le rendu d'éléments naturels : arbres, prairies, etc. L'une des nouveautés est que les particules peuvent interagir entre elles (*e.g.* collisions). Un des aspects intéressant de cet article est la manière dont Reeves et Blau calculent l'illumination et l'auto-ombrage d'un arbre. Une représentation probabiliste de l'arbre est associée à la structure, et est utilisée pour estimer l'ensoleillement d'une particule en fonction de sa position dans l'arbre et de la position du soleil : plus la particule se trouve à l'intérieur de l'arbre, plus elle est à l'ombre (*cf.* figure 2.8 à gauche). Cette approche empirique donne un rendu localement approximatif mais, à cause du très grand nombre de particules l'impression globale est excellente (*cf.* figure 2.8 à droite).

Outre la spécificité de son champ d'application, une des limites de cette technique est son incapacité à modéliser une structure pour une espèce d'arbre donnée. En outre, la spécificité de son algorithme de rendu, qui dessine les particules comme des traits de crayon, diffère totalement de la représentation classique et rend assez difficile son intégration à une scène existante.

2.2 Rendu à base de points

Bien que l'idée d'utiliser le point comme primitive de rendu soit apparue dès 1985 [LW85], ce n'est que très récemment que ce domaine a pris de l'ampleur. En 2000 Pfister [PZvBG00] et Rusinkiewicz [RL00] la revisitent en montrant que le "disque", très facilement géré par une carte graphique, est une primitive bien adaptée à la représentation de nombreux phénomènes ou d'objets complexes.

À la place des polygones, Pfister utilise des points qu'il appelle des *surfels*, compression de *surface elements*. Ces points ne sont pas connexes, ce qui permet d'en adapter le nombre en fonction de la taille de l'objet à l'écran. Le coût d'affichage d'un objet est proportionnel à sa taille à l'écran et non à sa complexité intrinsèque. Ce qui permet de conserver un taux de rafraîchissement de l'image contant, même avec beaucoup d'objets affichés.

Tandis que Pfister convertit les objets polygonaux en *surfels* en pré-traitement, Stamminger [SD01] échantillonne à la volée des objets procéduraux, ce qui offre une plus grande souplesse dans le choix de la précision. Il montre aussi que cette technique est bien adaptée au rendu de phénomènes naturels variés comme une montagne, le mouvement de l'eau ou le rendu d'arbres.

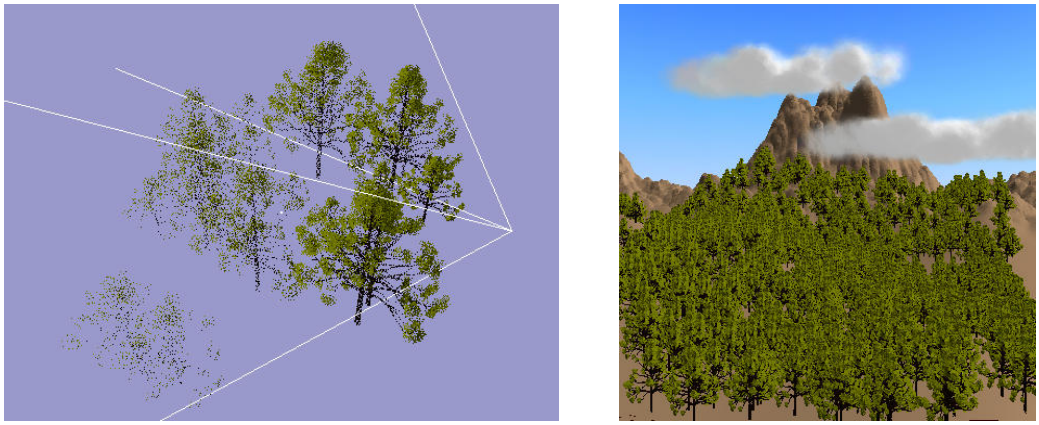

FIG. 2.9 – *À gauche* : des arbres représentés par des points. Plus on s'éloigne de la caméra moins il y a de points affichés. *À droite* : un exemple de paysage rendu avec la technique des points [SD01].

Cette technique est jeune et comporte certains défauts : texturation limitée, objets semi-transparents difficilement représentables, le calcul par moyenne des normales lorsqu'on simplifie le modèle est contestable, etc. Néanmoins l'idée semble très prometteuse, pour preuve la quantité de travaux en cours sur ce sujet [MZG01, PG01].

2.3 Modèles à base d'images

Toujours dans l'idée que certains phénomènes sont trop complexes pour être représentés explicitement en détails (*i.e.* avec beaucoup de polygones) nous survolerons les techniques de rendu à base d'images (*Image Base Rendering IBR*). L'idée est d'utiliser la complexité potentielle que cristallise une image pour représenter un phénomène. La deuxième technique que nous présenterons en partie III de cette thèse est à classer parmi cette famille de méthodes, d'où l'intérêt de présenter les techniques existantes en *IBR*.

Après une étude des techniques de *billboarding* en 2.3.1, nous examinerons en 2.3.2 le concept de fonction plénoptique, en 2.3.3 les techniques utilisant l'image et sa carte de profondeur, en 2.3.4 une méthode de Max spécifique aux arbres et nous finirons en 2.3.5 par un bilan des techniques à base d'images.

2.3.1 Billboards et dérivés

Une représentation très utilisée dans les systèmes temps réel comme les simulateurs ou les jeux vidéo est le *billboard*. Un *billboard* est un polygone recouvert par une texture, représentant par exemple un arbre, que le moteur de rendu oriente toujours vers l'observateur (*cf.* figure 2.10 à gauche et à droite).

L'avantage de cette technique est d'offrir un modèle simple à mettre en œuvre et très efficace ; il est alors possible d'avoir de nombreux arbres pour un faible coût. Le manque de réalisme des arbres lorsque l'observateur se déplace librement dans la scène est souvent caché par le fait que la caméra (*e.g.* un avion ou une voiture) passe à grande vitesse à côté des *billboards*. De plus, le coût mémoire augmente vite si l'on veut avoir une diversité d'apparence des arbres. À noter aussi l'impossibilité de changer l'illumination ou l'ombrage d'un tel modèle parce que celle-ci est fixée dans l'image. Une vue de dessus serait également incorrecte puisque les arbres paraîtraient couchés, une variante consiste alors à contraindre l'orientation verticale, puis à effectuer la rotation en gardant autant que posible l'arbre face à la caméra (*cf.* figure 2.10 à droite).

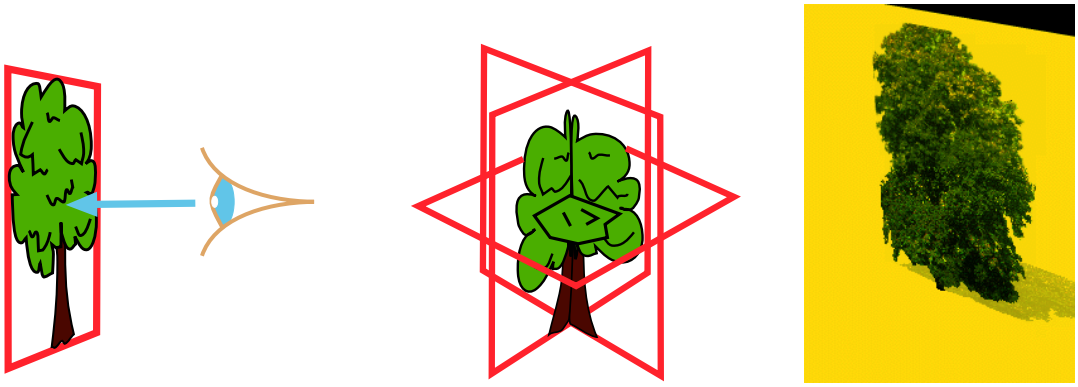

FIG. 2.10 – *À gauche* : *billboard* classique. *Au milieu* : *billboard* croisé. *À droite* : l'aspect plat d'un *billboard*.

Une amélioration possible en terme de réalisme au *billboard* est l'utilisation de trois polygones texturés placés en forme de croix pour représenter un arbre (*cf.* figure 2.10 au milieu), ce qui permet de véritablement tourner autour de l'objet : nous appellerons ceci les *billboards* croisés. Bien qu'améliorant le réalisme des arbres quand on les voit de dessus, cette technique conserve tous les problèmes des *billboards*, comme le peu de réalisme (à cause du manque de relief et de volume), le coût mémoire, avec en plus, le problème technique que pose le mélange des trois images semi-transparente par le moteur de rendu.

Les jeux vidéos utilisent souvent une représentation combinant la géométrie pour le tronc avec des *billboards* pour les branches. Ceci augmente le réalisme des vues proches, mais est plus coûteux en mémoire et en temps de rendu.

Avec le même esprit de mixer géométrie et *billboards* Pulli dans [PCD+97] propose une technique partant d'une série de vues d'un objet dont il dispose d'un maillage grossier. Pour effectuer le rendu depuis un point de vue quelconque, il effectue trois rendus du maillage texturé de l'objet : pour chaque rendu, les textures sont différentes car extraites des séries de vues et seuls les polygones significatifs du maillage sont considérés. Puis, il reconstruit l'image finale en interpolant par pixel les trois images ainsi obtenues.

2.3.2 Fonction plénoptique

En 1995 Bishop [MB95] introduit la fonction *plénoptique* d'Adelson et Bergen dans le but de reconstruire l'image d'un objet depuis n'importe quel point de vue à partir d'une série d'images de cet objet prises lors d'un travelling (*cf.* figure 2.11 à gauche). Cette fonction est à l'origine de deux techniques très connues se basant sur la représentation du champ de lumière, appelé *light field* [LH96] ou *lumigraph* [GGSC96].

Le champ de lumière décrit les caractéristiques complètes de la lumière dans une scène, c'est-à-dire la radiance en un point dans une direction donnée. Ceci est une fonction 5D : position 3D et direction 2D. En obtenant le champ de lumière à partir d'une série d'images, on peut générer de nouvelles images de l'objet sous un nouveau point de vue, simplement en le rééchantillonnant.

À la différence de la plupart des techniques de rendu à base d'images que nous allons voir plus loin, l'image de profondeur n'est pas nécessaire, on peut donc appliquer cette technique à des photographies. Les inconvénients majeurs de ces techniques sont le coût mémoire qu'elles engendrent, la faible possibilité d'accélération du rendu par le matériel graphique standard (à cause de la dimension de la fonction), ainsi que l'impossibilité de changer l'éclairage de la scène puisque celui-ci est stocké implicitement dans les images initiales. En outre, certaines parties cachées de l'objet ne sont pas capturées dans les images, ce qui se traduit lors du rendu par des trous qu'il faut combler en interpolant les valeurs voisines. Il n'est donc pas pensable aujourd'hui d'appliquer une telle technique au rendu de centaines d'objets comme par exemple, une forêt.

Récemment Miller [MRP98] et Wood [WAA+00] ont proposé une technique de compression des *lightfields* couplée à des algorithmes de décompression peu coûteux, limitant ainsi la place mémoire occupée par ces structures de données.

2.3.3 Tranches d'images de profondeur

Shade en 1998 [SGHS98] propose les *Layer Depth Images* (*LDI*), une méthode utilisant plusieurs images et leur tampon de profondeur associé, offrant ainsi une meilleure parallaxe que les techniques vues à la section précédente, et ceci avec moins de données. Cependant cette technique, tout comme les *lumigraphs*, ne permet pas le temps réel sur des centaines d'objets, ni le changement d'éclairage. Les objets semi-opaques ne sont pas représentables, parce que le tampon de profondeur ne stocke qu'une unique valeur de profondeur par pixel. L'année suivante cette technique sera rendue hiérarchique par Bishop dans [CBL99].

Oliveira dans [OBM00] propose une technique dérivée de celle de Shade, mais basée sur la distorsion des textures au moment du rendu pour donner une illusion de perspective.

2.3.4 Un modèle à base d'images de profondeur adapté aux arbres

Max dans [MO95, Max96, MDK99] applique aux arbres les représentations basées sur l'image et son tampon de profondeur. Dans son modèle, un pixel de l'image va contenir la couleur, la profondeur ainsi qu'une unique normale. La représentation est hiérarchique (dans la version la plus récente), c'est-à-dire qu'il part des images d'une feuille (une feuille réelle scannée), avec lesquelles il construit une petite branche dont il tire une image et son tampon de profondeur, lesquels sont utilisés pour construire une branche entière, etc. jusqu'à arriver au niveau de l'arbre. Il pré-calcule une vingtaine de points de vue pour chaque objet, qu'il sélectionne au moment du rendu en fonction de la position de l'œil. Le rendu est effectué avec l'aide du matériel graphique, l'illumination est calculée en post-traitement à l'aide de la normale et de la couleur stockée dans chaque pixel de l'image. L'antialiassage est obtenu par sur-échantillonnage.

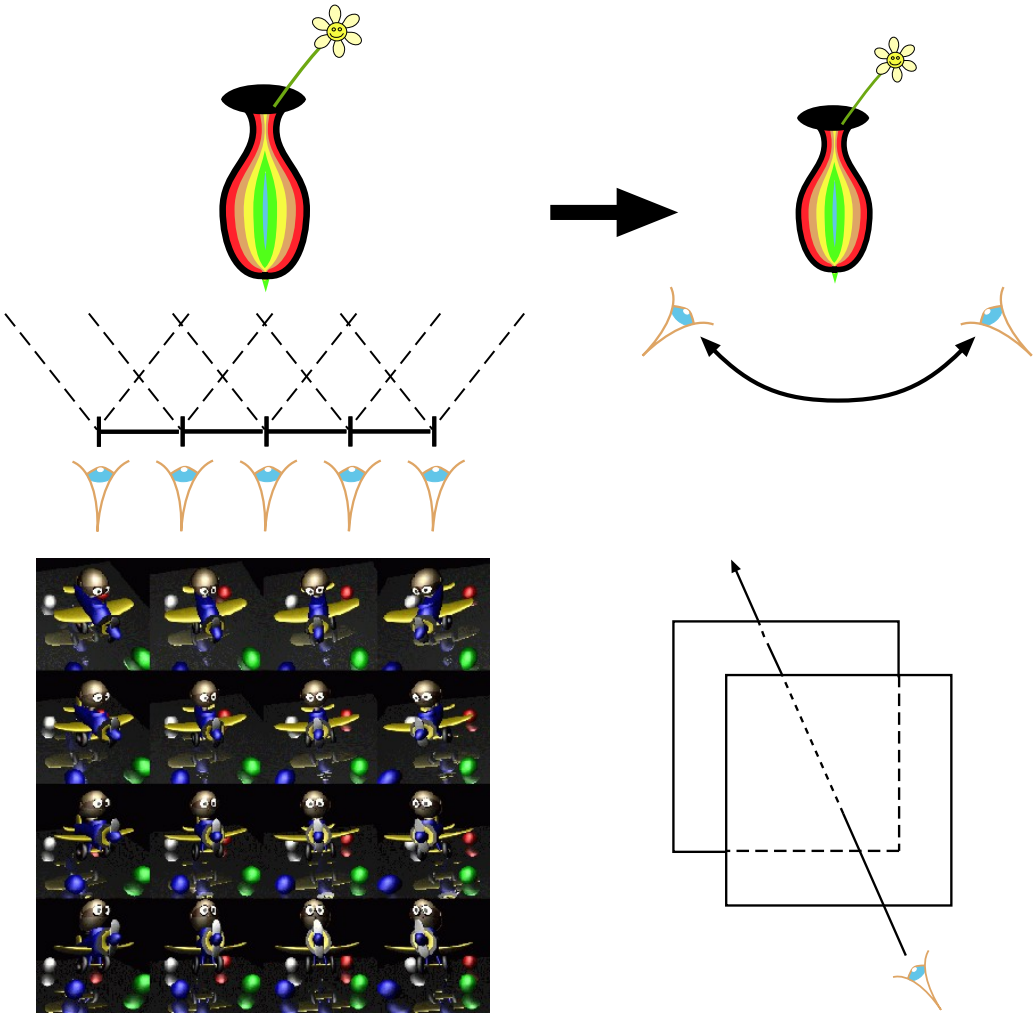

FIG. 2.11 – La technique des *Lightfield* [LH96] ou des *lumigraph* [GGSC96] permet à partir d'une séries d'images (*à gauche*), de reconstruire les points de vue proches d'un objet (*à droite*).

Les inconvénients d'une telle technique sont, comme pour la technique des *LDI* :

- la difficulté d'utiliser pleinement la carte graphique pour accélérer le rendu à cause du traitement final calculant l'illumination,
- l'impossibilité de traiter des objets semi-opaques à cause du tampon de profondeur stockant une unique valeur,
- l'utilisation d'une unique normale par pixel alors que souvent un pixel représente plusieurs feuilles, voir branches.

Max parvient à rendre un très beau verger d'une dizaine d'arbres avec un temps de quelques minutes par images (cependant le matériel actuel devrait permettre d'améliorer nettement ces performances). L'idée de la hiérarchie basée sur les images est très intéressante dans le cas des arbres puisque eux-mêmes sont hiérarchiques. Ce modèle a été une forte source d'inspiration pour celui que nous présenterons dans la partie III de cette thèse.

2.3.5 Bilan des approches à base d'images

Les techniques adaptées au rendu temps réel de grands nombres d'objets, comme les *billboard*, ont le défaut de ne pas offrir un réalisme satisfaisant, de ne pas permettre une grande diversité d'arbres à faible coût mémoire, et de ne pas autoriser la modification dynamique de l'illumination.

La grande qualité des approches *LDI* [SGHS98] ou de celles de Max [MO95, Max96, MDK99] est de pouvoir reconstruire un objet 3D de manière très réaliste à partir de quelques points de vue, par contre l'impossibilité d'utiliser le matériel graphique pour le rendu (ou très peu) est une limitation forte si l'on veut utiliser ces techniques pour rendre une forêt. De plus, il est souvent difficile de modifier l'illumination des objets une fois les images capturées. En revanche, l'idée de Max de calquer une hiérarchie à base d'images sur celle des arbres est très intéressante et doit être retenue.

2.4 Textures volumiques

Le modèle de *textures volumiques* introduit par Kajiya et Kay [KK89] pour le rendu de fourrure puis généralisé par Neyret [Ney95, Ney96, Ney98], est très intéressant sur deux points : tout d'abord parce qu'il constitue une représentation alternative traitant efficacement la complexité de nombreux objets (fil conducteur de ce chapitre), en outre, l'intégrale de l'illumination sur un cylindre effectuée par Kajiya et Kay est le point départ du modèle que présentons au chapitre 4. Nous détaillerons donc le modèle initial en 2.4.1, suivi des améliorations apportées en 2.4.2 et 2.4.3, et nous terminerons par un bilan en 2.4.4.

2.4.1 Le modèle initial [KK89]

Kajiya et Kay développent une approche orientée texture pour représenter des géométries répétitives complexes recouvrant une surface à la manière d'une peau épaisse, comme par exemple la fourrure d'un animal, une forêt sur une colline, etc. Un volume cubique contient un échantillon de référence de cette géométrie encodée sous forme d'un volume de voxels. Les instances de ce volume plaquées sur une surface sont appelées *texels* pour *texture element* (*cf.* figure 2.12). Ce volume de référence est déformé de façon à être plaqué sur la surface, à la manière d'une texture 2D classique.

Le volume cubique de référence est constitué de voxels. Chaque voxel contient formellement trois informations :

- une densité (i.e. une présence) ;
- un ensemble de 3 vecteurs donnant l'orientation locale de la surface (Normale, Tangente, BiNormale) ;
- une fonction indiquant comment la lumière se réfléchit (réflectance).

L'orientation locale de la surface et la réflectance peuvent être regroupées en une unique donnée. Un texel contient donc une information spatiale (densité) et une information sur le comportement local vis à vis de la lumière (réflectance). En réalité, dans leur implémentation, Kajiya et Kay stockent uniquement la densité, car l'orientation et la réflectance sont constantes dans un modèle de fourrure : le volume de référence représente un échantillon constitué de cylindres (les poils), il est ensuite instancié et déformé pour suivre le sens du poil. Ils utilisent comme fonction de réflectance l'intégrale analytique approchée de l'illumination d'un cylindre. L'intégrale du terme spéculaire n'est pas réellement calculée mais est construite de manière *ad hoc*. Plus tard, cette illumination analytique sera améliorée par Goldman dans [Gol97].

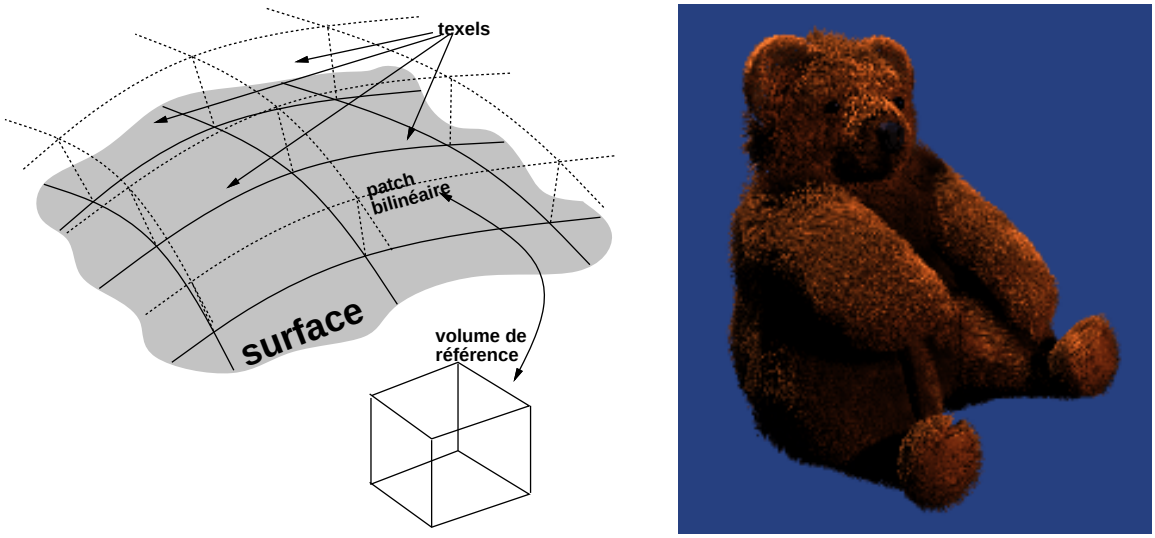

FIG. 2.12 – *À gauche* : le principe des textures volumiques est de plaquer sur la surface un volume de référence pour former une peau épaisse sur l'objet. *À droite* : l'ours de Kajiya et Kay [KK89].

Le volume de référence est destiné à être plaqué de manière répétitive sur toute la surface comme pour une texture surfacique classique. Pour qu'il y ait continuité entre texels voisins il faut que le volume soit déformé : Kajiya et Kay font correspondre les quatre arêtes verticales du volume avec des vecteurs stockés aux nœuds qui sont initialement les normales à la face, mais que l'on peut orienter différemment pour déformer la texture.

Le rendu de Kajiya et Kay utilise un algorithme de lancer de rayons qui devient un peu particulier lorsque celui-ci traverse un texel. Lorsque le rayon traverse un texel on se reporte dans le volume de référence où on applique une technique de rendu volumique : le rayon parcourt les voxels tout en accumulant la transparence et l'illumination locale, qui est évaluée en appliquant la fonction de réflectance, pondérée par l'ombrage obtenu en lançant un rayon entre le voxel et la source de lumière.

Ce modèle a permis à Kajiya et Kay de réaliser de belles images d'ours en peluche, sans aliassage (*cf.* figure 2.12 à droite), mais au prix d'une douzaine d'heures de calcul (à l'époque). Nous allons voir à la section suivante comment cette technique a été étendue et améliorée.

2.4.2 Un modèle plus général

Dans [KK89] les auteurs voulaient représenter une texture bien particulière, la peluche sur un ours. Bien que l'architecture proposée soit générale, le texel et la fonction de réflectance utilisés sont très spécifiques : le texel, qui doit matérialiser un ensemble de poils, contient des cylindres perpendiculaires à la base du volume, et la fonction de réflectance utilisée est cylindrique, n'autorisant que des objets cylindriques dans le volume de référence. Une généralisation de ce modèle a été réalisée par Neyret [Ney95, Ney96, Ney98].

Neyret propose une fonction de réflectance paramétrable (l'ellipsoïde) qui est capable de modéliser de nombreux types de formes. La méthode originale est très lente : malgré l'échantillonnage du rayon, le rendu de Kajiya et Kay considère inutilement beaucoup de voxels dans le cas où

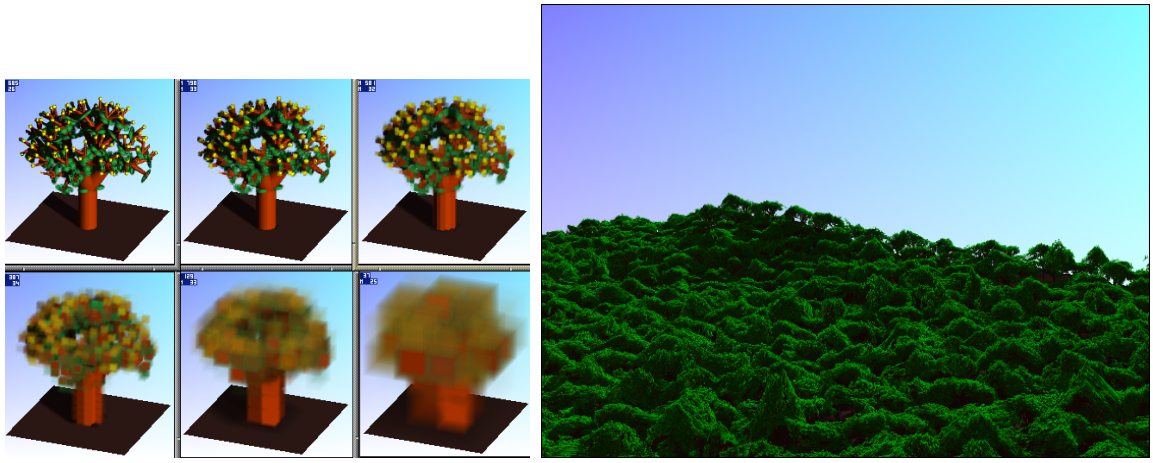

FIG. 2.13 – *À gauche* : un texel représentant un arbre à différents niveaux de détails. *À droite* : une forêt générée par la méthode de Neyret [Ney96], une extension des textures volumiques.

le volume est loin de l'observateur[2]. Neyret introduit donc une approche multi-échelles similaire au *MIP mapping* utilisant les octrees, ce qui n'est possible qu'en introduisant une primitive plus générique ayant une structure de "groupe", *i.e.* qui permette de représenter aussi la combinaison des primitives (*cf.* figure 2.13). Le niveau de détails affiché est ainsi modulé en fonction de la distance qui sépare le texel de l'observateur, ce qui procure un gain de vitesse appréciable, en faisant disparaître les contrastes des détails, mais en gardant la même qualité d'image (i.e. avec très peu d'aliassage). Les octrees apportent aussi un gain en taille non négligeable (le taux de compression est de l'ordre de 95%), en effet un texel non compressé peut vite atteindre une taille importante.

Avec ce modèle enrichi de texture volumique, Neyret arrive à rendre des images de bonne qualité (*cf.* figure 2.13 à droite) dans un temps tout à fait raisonnable, par lancer de rayons (10 à 20 minutes à l'époque) avec un unique rayon par pixel.

2.4.3 Textures volumiques dédiées aux arbres

Noma [Nom95] dérive le principe des textures volumiques de Kajiya pour le rendu spécifique d'arbres. En chaque voxel, il stocke l'opacité de manière discrète en échantillonnant la géométrie depuis une série de directions autour de la sphère, lors du rendu il interpole les valeurs. Pour la fonction d'illumination, il calcule en chaque voxel la moyenne des normales des feuilles et la moyenne des cosinus des angles entre ces normales et les trois axes X,Y,Z. Lors du rendu, il interpole ces valeurs en fonction des angles entre la lumière et les axes X,Y,Z. Il utilise de la géométrie pour les arbres proches qu'il mélange aux texels pour les arbres éloignés.

2.4.4 Bilan des textures volumiques

Les textures volumiques peuvent être vues de deux manières :
- une représentation pour les scènes complexes, facile à contrôler par l'utilisateur ;
- une représentation permettant un rendu très efficace en temps et en qualité, car générique et multi-échelle.

[2] A noter que ce problème existe aussi pour les textures 2D et a été résolu par la technique du *MIP mapping* en pré-calculant la texture à diverses résolutions(*cf.* section 1.1.2).

Cette représentation alternative propose de remplacer une géométrie complexe par un octree où chaque voxel est représenté par une fonction analytique d'illumination. Neyret propose une fonction générique ellipsoïdale. Cette fonction de réflectance générique peut s'avérer inadaptée à certaines géométries : les arêtes vives, ou toutes géométries où la distribution de normales comporte des discontinuités. Pour remédier à ceci l'idée serait de calculer une série de fonctions d'illumination pour des classes d'objets spécifiques : c'est cette idée que nous allons développer durant la partie II.

2.5 Couches d'images

Les textures volumiques donnent des résultats visuels réalistes, et rapidement pour une technique utilisant le lancer de rayons. Cependant, pour des applications exigeant le temps réel, l'obtention d'un gain supplémentaire parait difficilement envisageable en conservant cette technique telle quelle. Des approches adaptées au matériel graphique ont donc été développées. Les moteurs de rendu des cartes graphiques traitant efficacement des polygones, nous présentons ici des techniques basées sur le rendu par couches, où chaque couche est représentée par un polygone texturé. Cette idée était à l'origine destinée au rendu volumique puis elle s'est généralisée à divers domaines du rendu.

Nous verrons en 2.5.1 les techniques de rendu volumique introduisant l'idée de tranches, puis nous détaillerons en 2.5.2 la technique de texture volumique temps réel, ainsi qu'en 2.5.3 les améliorations apportées. Nous finirons en 2.5.4 par un bilan de ces modèles à base de couches.

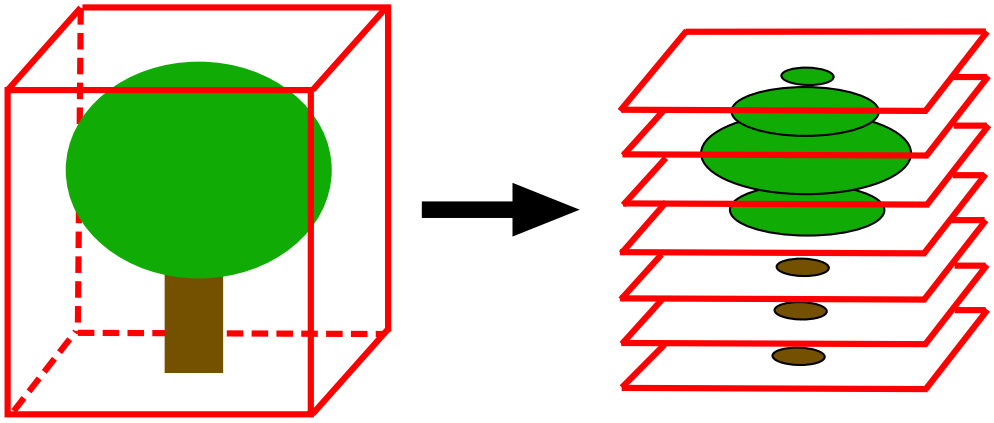

FIG. 2.14 – Le volume de référence des textures volumiques temps réel [MN98] est une superposition de polygones texturés.

2.5.1 Rendu volumique par couche

Le rendu volumique se calculait usuellement en lançant des rayons à travers l'espace voxelisé, ce qui se traduisait par des temps de calcul très longs. Pour l'accélérer Lacroute et Levoy introduisent le rendu par couches d'images [LL94]. Ils interprètent leurs données volumiques comme des couches, avec l'idée de factoriser les voxels en une texture et de traiter en parallèle ces données. Pour le rendu ils proposent de projeter et composer successivement les couches sur le plan image et ainsi obtenir l'image finale. Chaque couche projetée est combinée avec le résultat précédent en tenant compte de la densité, on peut donc interpréter une couche comme une texture transparente. La projection d'une couche est plus rapide que les calculs de projection pour chaque voxel le long d'un rayon : cette factorisation permet donc un gain de temps important. Lacroute et

Levoy ont imaginé une méthode pour projeter les couches orthogonalement quel que soit l'angle sous lequel on regarde le volume, même avec un rendu en perspective.

Westermann et Ertl [WE98] proposent une adaptation de cette technique en profitant des fonctionnalités des cartes graphiques pour effectuer le rendu en temps réel. Leur implémentation permet même de tenir compte de l'illumination mais à cause de contraintes des cartes graphiques seul un rendu monochrome est possible : ils stockent la normale dans les trois composantes *RGB* du tampon. Le calcul de l'illumination se fait en post-traitement sur tout le tampon à l'aide du matériel graphique. Cette technique est distribuée comme extension *OpenGL* sous le nom de *Volumizer* [SGI].

2.5.2 Texture volumique temps réel

Neyret et moi-même [MN98] (correspondant à mon travail de DEA) avons développé une technique temps réel des textures volumiques (*cf.* section 2.4), en nous inspirant du modèle de Lacroute et Levoy décrit à la section précédente. Nous avons profité de la capacité des cartes graphiques à traiter rapidement des polygones texturés. Le principe est de représenter le texel par une superposition de polygones texturés et transparents (*cf.* figure 2.14).

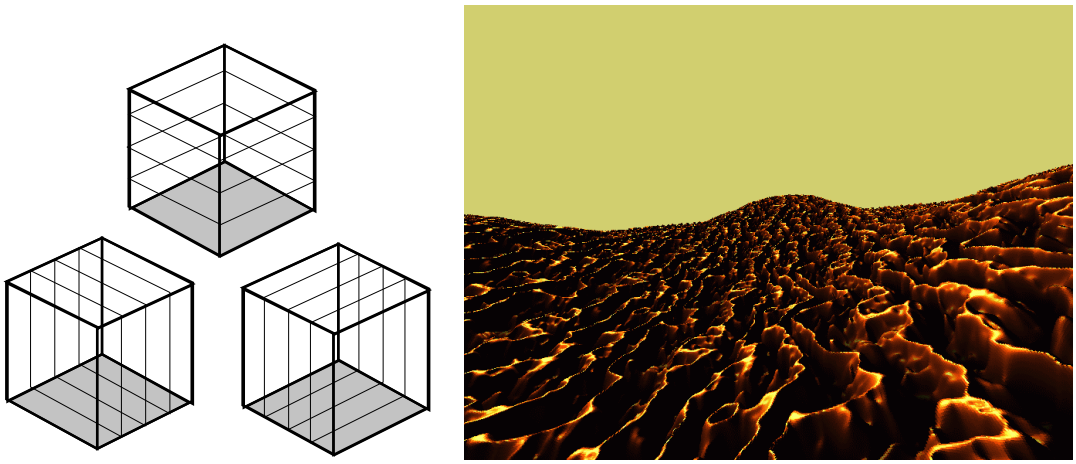

FIG. 2.15 – *À gauche* : trois directions de tranches. *À droite* : exemple de complexité obtenue.

Comme les tranches ne font pas face à l'observateur et n'ont pas d'épaisseur, on risque de voir entre elles. On définie donc trois directions de tranches. Le point de vue de l'observateur détermine laquelle de ces directions va être utilisée (*cf.* figure 2.15). La génération des texels peut se faire de deux manières, soit en convertissant une représentation polygonale, soit à partir d'une texture de hauteur.

Il est à noter que, contrairement à la représentation initiale des textures volumiques, chaque voxel contient directement la couleur de l'objet, c'est-à-dire que l'illumination est inscrite dans le voxel au lieu d'être calculée à chaque rendu. Ceci ne permet donc pas de changements de conditions d'éclairages après la capture des données, mais c'était la concession à faire pour obtenir le temps réel avec le matériel graphique de l'époque.

La même année, la technique de Schaufler [Sch98] utilise aussi des couches d'images pour le rendu accéléré d'objets. Son principe est de regrouper par couches les parties de l'objet ayant la même profondeur, afin de réutiliser cette partie de l'image aux pas de temps suivants (en effet des points se trouvant à la même profondeur se déplacent à la même vitesse quand la caméra se translate).

2.5.3 Autres techniques à base de couches

Modèle dédié aux arbres

Une technique proche de celle des textures volumiques par couches a été proposée dans [Jak00] pour le rendu spécifique d'arbres. Avec leur représentation, seules les feuilles sont représentées dans les tranches de texture, le tronc étant formé par des polygones. L'affichage des trois directions se fait simultanément et non pas en fonction du point de vue comme c'était le cas pour notre modèle. Ils montrent que peu de tranches sont nécessaires au réalisme. Leur modèle peut être rapproché de la technique des *billboards* croisés (*cf.* section 2.3.1).

Modèle dédié à la fourrure

Dans [Len00, LPFH01], Lengyel présente une adaptation intéressante de notre technique pour le rendu temps réel de fourrure. Son volume de référence est constitué de poils. Pour résoudre le problème des vues transversales il n'utilise pas trois directions de tranches comme nous le faisions, mais quatre polygones texturés représentant la silhouette qu'il plaque sur les quatre côtés du texel. Avec cette idée, les comportement des vues aux angles rasants par rapport à la surface deviennent très convaincants pour une approche temps réel (*cf.* figure 2.16 à droite).

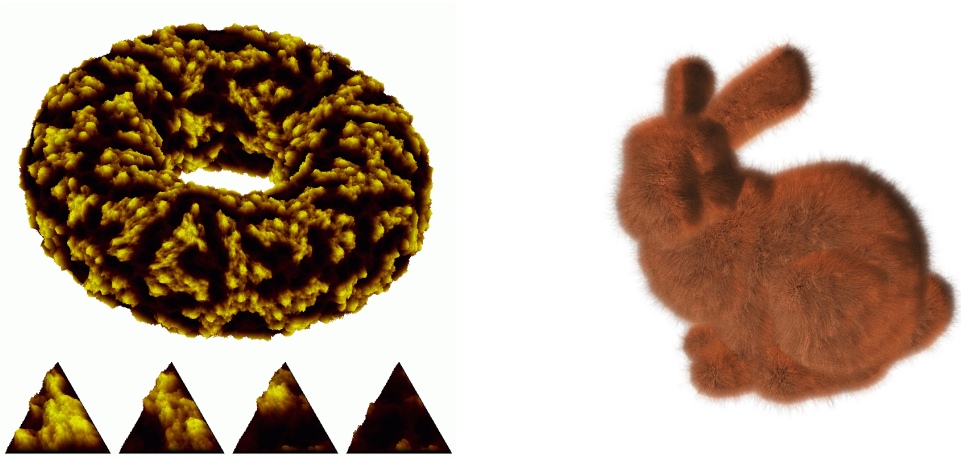

FIG. 2.16 – *À gauche* : un tore recouvert de texels par la méthode de [MN98], mappé sans distorsion apparente ni répétition avec la méthode de [NC99] basée sur des motifs triangulaires complémentaires. *À droite* : un lapin recouvert de texels par la méthode de [LPFH01].

Textures volumiques et illumination

Des travaux très récents de Sénégas [SN01] utilisent les nouvelles fonctionnalités d'illumination par pixel des nouvelles cartes graphiques[3], dans le but d'ajouter un calcul d'illumination aux texels temps réel.

Le matériel graphique des *SGI* que nous avions utilisé en 1998 ne permettaient pas le calcul de l'illumination au niveau des pixels. Celle-ci était évaluée aux sommets des polygones, puis ces

[3]Cartes *GeForce* 2 et 3 de la société *NVidia* et *Radeon* de la société *ATI*

valeurs étaient interpolées lors de la *rasterisation*[4]. Les nouvelles fonctionnalités des cartes graphiques permettent d'évaluer une fonction d'illumination en chaque pixel d'un polygone (*cf.* figure 2.17 à gauche).

Une couche d'un texel est alors composée de deux images : une image de couleur, comme dans notre représentation initiale et une image de normales. Les trois composantes *RGB* de l'image de normales correspondent aux coordonnées de la normale à la surface. Les fonctionnalités dont nous parlions plus haut permettent de micro-programmer une fonction d'illumination en chaque pixel du polygone en tenant compte de la couleur et de la normale. Ceci permet donc de changer interactivement la position de la lumière dans la scène. Il ne manque plus que l'ombrage et l'auto-ombrage pour obtenir un modèle alternatif complet (ce sont d'ailleurs des travaux en cours). Des travaux très récents, proches de ceux-là, ont été développés pour le rendu volumique [EKE01].

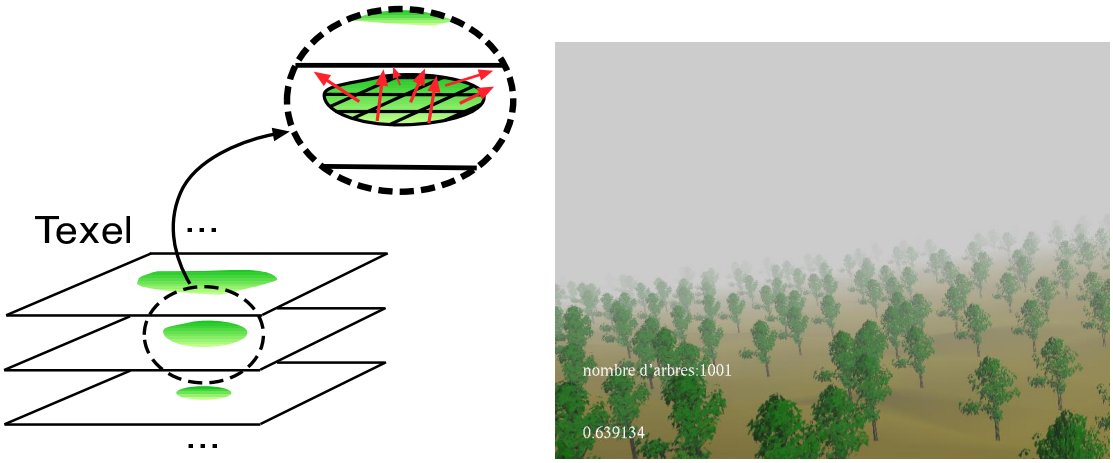

FIG. 2.17 – *À gauche* : avec les nouvelles générations de cartes graphiques, on peut évaluer une fonction d'illumination en chaque pixel du polygone. *À droite* : exemple de rendu temps réel d'une forêt de texel avec l'illumination par pixel [SN01].

2.5.4 Bilan des approches à base de couches d'images

Avec ces approches à base de couches d'images, les coûts de rendu deviennent proportionnels à la complexité de la surface et non à la complexité des objets la recouvrant. Une forêt aura un coût de $n \times m$ polygones où n est le nombre de polygones du terrain et m le nombre de couches pour représenter un arbre (de 64 à 256), ce qui est peu comparativement à sa complexité réelle (représentée visuellement par des milliards de polygones). Pour représenter les micro-géométries, traitées par les cartes de déplacement, il est possible d'obtenir le temps réel avec une technique à base de couches [KS01].

Les fonctionnalités de calcul de l'illumination par pixel des nouvelles cartes graphiques sont en train de lever les limites qu'avaient ces techniques en terme d'illumination. Par contre, la taille mémoire nécessaire à la représentation d'un objet reste toujours une limite à la diversité des objets utilisables simultanément (les performances des cartes graphiques ont explosées, mais pas leur capacité mémoire).

[4]Le processus de *rasterisation* est le fait de remplir le polygone 2D, pixel par pixel, une fois ses sommets projetés sur le plan de l'écran.

2.6 Rendu de macro-géométrie avec une fonction bi-directionnelle de texture

Dischler en 1998 [Dis98] propose une technique de textures de macro-géometrie permettant de traiter simplement des textures en relief comme de l'osier tissé ou des armatures métalliques. Pour cela, il concilie l'idée de lancer de rayons virtuel avec l'utilisation d'une sorte de textures de paramètres de Blinn (*cf.* section 1.1.2).

Le principe de lancer de rayons virtuel est simple : lorsque le rayon arrive sur une surface, on le transpose dans un volume de référence, où le lancer de rayons se poursuit (on retrouve la notion de volume de référence présente aussi dans les textures volumiques).

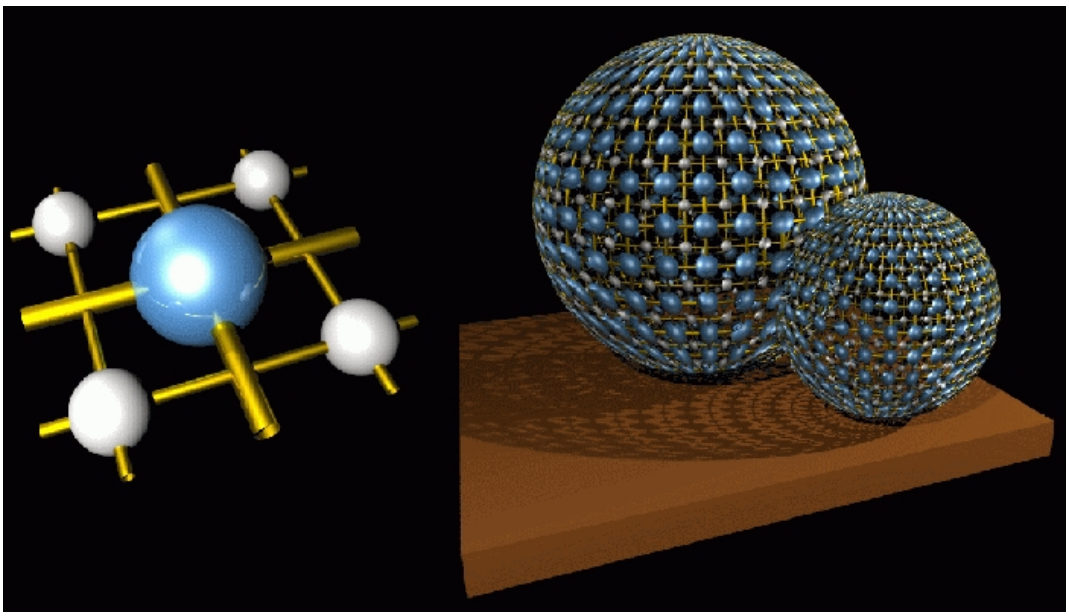

FIG. 2.18 – Les textures de paramètres de Dischler [Dis98]. *À gauche* : le volume de référence. *À droite* : deux sphères formées à partir du volume de référence.

L'idée de Dischler est de stocker dans son volume de référence une fonction de texture bi-directionnelle (*BTF*), qualificatif que Dana avait déjà employé pour ses collections d'images de matériaux prises sous tous les angles (*cf.* section 1.1.7). La texture utilisée est plate, comme une texture ordinaire mais son apparence change avec le point de vue, reproduisant ainsi les effets de parallaxe du relief. Chaque pixel de cette texture stocke des paramètres de normales et de couleurs différents selon la direction d'arrivée du rayon.

Le résultat est très convaincant (*cf.* figure 2.18), d'autant plus que ce type d'objet n'aurait pu être représenté ni par une *bump-map*, ni par une *displacement map*. Il y a cependant des défauts dus à l'absence d'inévitable codage 3D : une surface très distordue présentera des artefacts de parallaxe ; en outre, les rayons ne passent pas d'une case à l'autre, ce qui est pourtant nécessaire dans le cas d'un rayon rasant. Cependant, l'idée de pré-calculer l'information depuis toutes les directions est un principe que nous allons reprendre avec la technique présentée en III.

3 Bilan

À la vue des conclusions partielles que nous avons tirées après la présentation de chaque famille de modèles traitant la complexité géométrique de la nature, il devient clair qu'il n'existe pas

de technique unique et générale résolvant tous les problèmes. L'utilisation de polygones n'étant pas satisfaisante dans de nombreux cas, les chercheurs en synthèse d'images doivent concevoir d'autres représentations plus efficaces, ce qui, comme nous venons de le voir, fonctionne plutôt bien en terme de qualité et de coût. Seulement cette efficacité a un prix : la spécificité. Les modèles alternatifs sont souvent dédiés à une famille d'objets précis, leur domaine d'application est alors plus réduit, ce qui va de pair avec une certaine complexité de mise en œuvre : pour une représentation nouvelle, il faut redéfinir toute la chaîne qui va de la modélisation au rendu en passant par l'animation, et gérer son intégration avec les objets traditionnels[5].

Les paysages, et en particulier les forêts, sont constitués d'objets spécifiques. Cependant l'intérêt en terme d'applications est tel, que le développement d'activités dédiées est justifiable, d'autant que les représentations traditionnelles ne parviennent pas à offrir des solutions acceptables. Ce constat nous a conduit à développer durant cette thèse, des techniques spécifiques aux arbres, s'inspirant de tout ce que nous venons d'exposer ici.

[5] Il existe des projets [Res, Dis, Sof] essayant de regrouper en une application les techniques existantes en matière de synthèse d'images de paysages.

Deuxième partie

Modèles d’illumination hiérarchiques et analytiques [MN00]

CHAPITRE 3

Modèles d'illumination analytiques et hiérarchiques

La complexité des scènes d'extérieur pose de nombreux problèmes en terme de modélisations et de rendu. Cependant, nous venons de voir que les techniques de modélisation donnent aujourd'hui des résultats réalistes et diversifiés, aux prix de nombreux polygones. Les arbres font partie de cette catégorie d'objets dont la surface n'est pas vraiment définie, ce qui rend les parties internes de leur feuillage potentiellement visibles et éclairées. De plus, le rendu de ce type de scène est extrêmement coûteux, et très sujet à l'aliassage. Nous avons montré dans l'état de l'art que l'utilisation des niveaux de détails permet d'améliorer ce type de problème pour d'autres familles d'objets (*e.g.* herbe, terrain, etc.). Pour les arbres, peu de techniques sont capables de calculer automatiquement des niveaux de détails sans changer l'apparence globale. Nous verrons en 1 comment il est possible de s'appuyer sur la hiérarchie naturelle des arbres pour construire des niveaux de détails.

Les détails géométriques comme les feuilles ou les aiguilles sont trop petits pour être visibles individuellement dès que l'observateur se trouve à quelques dizaines de mètres (*cf.* figure 3.1). Les rameaux eux-mêmes se confondent avec la distance, puis les branches, etc. Il est donc intéressant d'essayer de remplacer les données non distinguables visuellement par une primitive ayant le même comportement photométrique que le groupe de géométrie qu'elle représente. Nous montrerons en 2 que cette idée, couplée à la notion de hiérarchie, est une piste pour la construction de niveaux de détails efficaces.

Lors du rendu, il est important d'utiliser un algorithme tirant partie de la connaissance disponible a priori sur le type de données que l'on souhaite traiter pour limiter l'aliassage et diminuer les coûts au maximum. Nous expliquerons donc en 3 les raisons pour lesquelles nous avons choisi le lancer de cônes, et le critère que nous utilisons pour le choix des niveaux de détails.

FIG. 3.1 – Un exemple de la complexité visuelle d'une forêt. L'objectif est de la représenter autrement que par une forte complexité géométrique, très coûteuse en temps de rendu et génératrice d'aliassage.

1 Niveaux de détails pour les arbres

Au fur et à mesure que l'on s'éloigne d'un arbre, on ne discerne plus les différentes parties : on commence par ne plus distinguer les feuilles, les rameaux puis les branches se mélangent, et pour finir, seule la silhouette de l'arbre reste visible. On ne peut pas non plus supprimer simplement les détails fins : la forêt de la figure 3.1 démunie de ses aiguilles aurait une allure très différente, bien que chaque aiguille occupe moins de 1% d'un pixel. Ce constat plaide en faveur des niveaux de détails et de leur continuité.

1.0.1 Simplification de maillage

L'état de l'art montre que peu de modèles sont capables de générer automatiquement et efficacement des niveaux de détails pour les arbres. Le modèle de Weber et Pen (*cf.* chapitre 2 section 1.2.3) simplifie un modèle géométrique d'arbre en supprimant les branches les moins significatives. Ceci allège le coût de rendu mais au détriment de la conservation de l'illumination et de la transparence générale de l'arbre entre les niveaux. Des effets de saut (*poping*) se font sentir lors du passage d'un niveau à l'autre. Une telle technique n'est pas satisfaisante ; plutôt que de décimer, il est sûrement préférable de regrouper.

1.0.2 Utilisation de la hiérarchie

Les arbres possèdent des propriétés hiérarchiques intéressantes : les feuilles se répètent autour d'un tronc pour former une branche secondaire. Plusieurs branches secondaires se répètent pour former une branche principale et plusieurs branches principales forment l'arbre complet (*cf.* figure

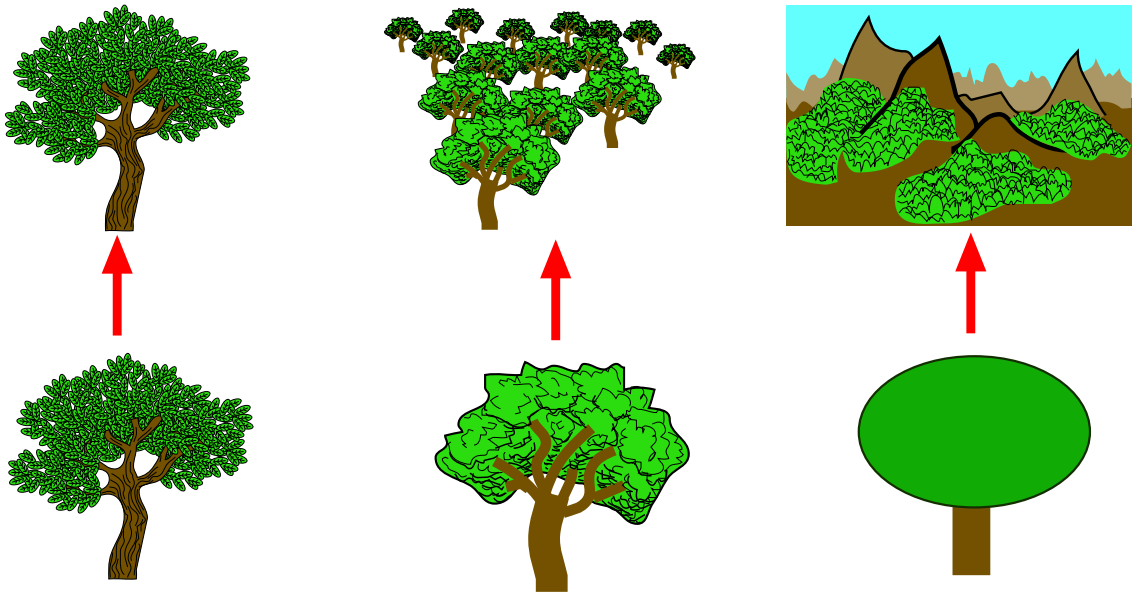

FIG. 3.2 – Différents niveaux de détails.

3.3). De plus, deux branches d'une même famille d'arbres sont assez similaires. Il est même possible d'étendre cette notion de répétition aux paysages puisque deux arbres d'une même famille sont assez semblables.

Ce principe de hiérarchie répétitive peut nous aider pour la construction de niveaux de détails sur deux points :

- la hiérarchie naturelle de l'arbre guidera la hiérarchie de nos niveaux de détails ;
- les structures répétées pourront être factorisées et instanciées, ce qui permet de diminuer le coût mémoire. Pour éviter les effets répétitifs il suffit souvent de changer l'orientation et la taille des objets, voire la palette de couleurs.

Max [MDK99] utilise déjà avec succès ces propriétés hiérarchiques des arbres dans certains de ses modèles (*cf.* chapitre 2 section 2.3.4).

2 Modèles d'illumination analytiques et hiérarchiques

2.1 Forme et illumination

Le détail d'un ensemble de primitives (*e.g.* une branche ou un arbre) n'est pas visible explicitement lorsque l'observateur est loin (*i.e.* lorsque l'ensemble des primitives ne recouvrent que quelques dizaines de pixels de l'image ou moins). Seule la forme et la couleur d'ensemble sont alors distinguées. L'idée que nous présentons ici est de représenter ce groupe de primitives par sa forme, associée à une formule analytique décrivant son comportement photométrique et son opacité globale (*cf.* figure 3.4), c'est-à-dire par un modèle d'illumination (*shader*). Ce *shader* est obtenu en intégrant un modèle d'illumination simple[1](*e.g.* celui de Phong) sur l'ensemble des primitives du groupe en tenant compte de la visibilité[2], et en calculant leur opacité moyenne.

[1]Les calculs mathématiques sur lesquels repose cette intégrale ne sont toutefois réalisables analytiquement que si la fonction d'illumination est simple.

[2]Ceci se traduit par une restriction du domaine d'intégration.

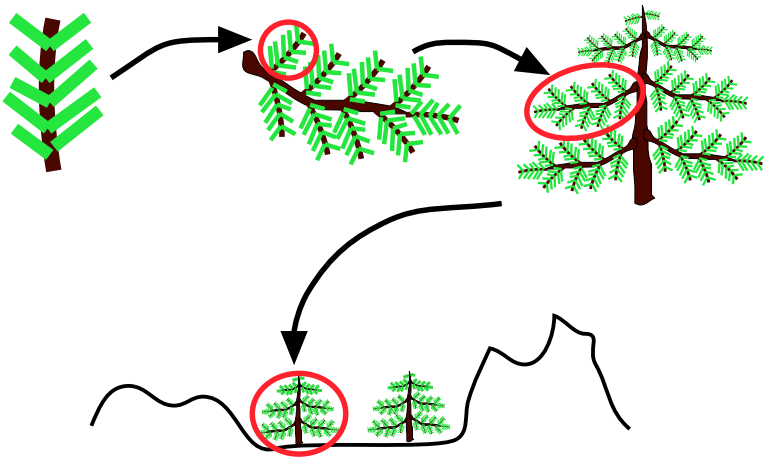

FIG. 3.3 – Les arbres sont hiérarchiques : plusieurs branches secondaires forment une branche principale, et plusieurs branches principales forment l'arbre. Toutes les branches se ressemblent, il est donc possible de les instancier. De même, un arbre peut être instancié plusieurs fois dans une scène. Il suffit alors de modifier son orientation et sa taille pour éviter l'impression d'uniformité.

Cette idée est présente dans beaucoup de modèles d'illumination de surface ou par exemple dans la représentation de fourrure de Kajiya puisqu'il intègre le modèle d'illumination de Phong sur un cylindre (*cf.* chapitre 2 section 1.2.2).

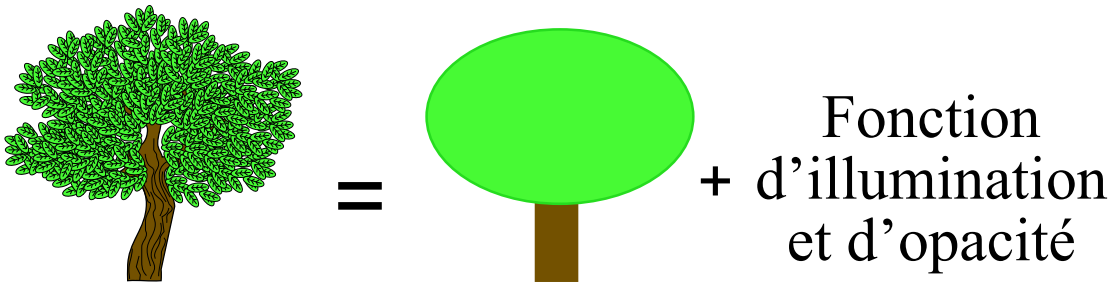

FIG. 3.4 – Un objet vu de loin peut être représenté par sa forme et son modèle d'illumination (*shader*). Le résultat de son rendu sera de meilleure qualité (*i.e.* avec peu d'aliassage), avec un temps de calcul plus petit que si on utilise la représentation où toute la géométrie est utilisée.

2.2 Hiérarchie

Le calcul de l'intégrale du modèle d'illumination sur les primitives n'est réalisable que si une connaissance a priori forte existe sur l'agencement des primitives. L'aspect hiérarchique des arbres dont nous parlions à la section 1, peut nous servir de base au calcul. Si nous essayons de calculer l'intégrale analytique de la fonction de Phong sur un arbre entier il est probable que le calcul ne soit pas réalisable (la géométrie trop complexe d'un arbre ne pourra se traduire par une modélisation mathématique raisonnable pour les calculs). En revanche, nous pouvons procéder par étapes, tout en construisant les niveaux de détails. Nous commençons par calculer le *shader* $\mathrm{I}_{\mathtt{branche}}$ d'une branche en intégrant le modèle illumination de Phong sur l'ensemble de ses primitives (*e.g.* en fixant une géométrie simple des feuilles). En utilisant ce résultat, nous calculons le *shader* $\mathrm{I}_{\mathtt{arbre}}$ d'un arbre entier, en calculant l'intégrale de $\mathrm{I}_{\mathtt{branche}}$ sur l'ensemble des branches (*cf.* figure 3.5).

De plus, un terme d'opacité moyenne doit être calculé pour chaque niveau de la hiérarchie. La description que nous venons de faire se limite à deux niveaux dans un but pédagogique ; il est bien sûr préférable d'en avoir plus.

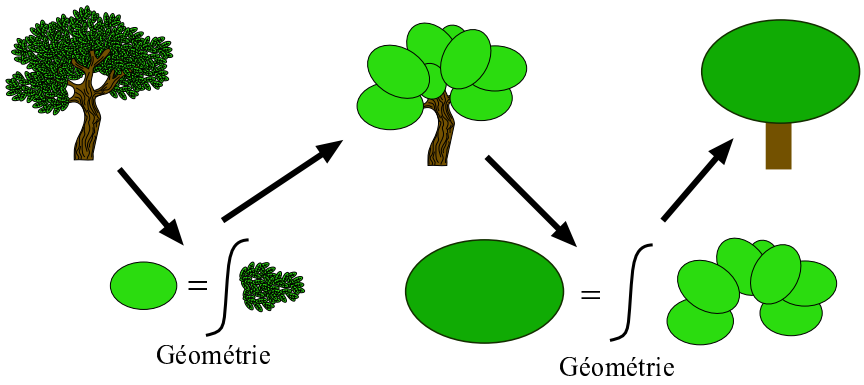

FIG. 3.5 – Principe d'une hiérarchie de *shaders* analytiques dans le cas d'un arbre. L'illumination analytique d'une branche est calculée en intégrant la fonction d'illumination de base (*e.g.* le modèle de Phong) sur l'ensemble des feuilles d'une branche (une connaissance a priori de la disposition des feuilles est nécessaire pour simplifier la modélisation mathématique). À partir de cette fonction d'illumination (et toujours de la connaissance a priori de la répartition de la matière) on calcule finalement l'illumination de l'arbre entier. De plus, un terme d'opacité moyenne doit être calculé pour chaque niveau de la hiérarchie.

Nous venons de présenté une idée générale pour la construction de niveaux de détails d'arbres fondée sur l'intégration analytique du modèle d'illumination et sur leur hiérarchie naturelle, il nous reste maintenant à mettre en pratique ce concept pour prouver sa pertinence, ce que nous ferons au chapitre suivant. Avant ça, je tiens à préciser quelques points en rapport avec l'algorithme de rendu que nous avons utilisé.

3 Considérations liées au rendu

Les niveaux de détails, même basés sur une illumination analytique, doivent être utilisés à bon escient lors de la phase de rendu. Nous ferons quelques précisions concernant le lancer de cônes en 3.1, et introduirons en 3.2 un critère permettant le choix du "bon" niveau de détails.

3.1 Lancer de cônes

L'ombrage jouant un rôle important dans le réalisme des images, nous avons choisi d'utiliser un rendu par lancer de rayons. L'état de l'art nous indique que l'algorithme de *beam-tracing* permet de diminuer l'aliassage plus efficacement que le sur-échantillonnage dans le cas d'objets à hautes fréquences, comme les arbres (*cf.* chapitre 1 section 2.1.2). Du fait de la complexité du calcul exact d'intersection de l'algorithme de *beam-tracing*, nous avons choisi d'utiliser une adaptation de cet algorithme, le *cone-tracing* : pour chaque objet intersecté par notre rayon conique nous calculons son pourcentage a de recouvrement de la section du rayon. Le calcul global de la couleur d'un pixel est :

$$C_{pixel} = a_A.\alpha_A.C_A + (1 - a_A.\alpha_A) \times a_B.\alpha_B.C_B + ...$$

avec a_A le pourcentage de recouvrement du pixel par l'objet A et α_A l'opacité de l'objet A, etc. Le calcul du pourcentage de recouvrement ne tient pas compte de la répartition géométrique

des objets dans le pixel. Néanmoins cette approximation est valide si les objets sont distribués uniformément dans l'espace, du moins sans corrélation (ce qui est le cas des arbres).

Ce schéma de calcul constitue un bon compromis temps de calcul/aliassage. Nous pouvons ainsi lancer un seul rayon par pixel, sans pour autant surcharger le calcul d'une précision superflue (pour les arbres) qu'apporterait le *beam-tracing*. Remarque : ce schéma de calcul est similaire à celui de l'algorithme du *A-buffer* (qu'il aurait été possible d'utiliser ici) qui calcule de la même manière un pourcentage de recouvrement du pixel.

3.2 Choix du niveau de détails

Durant la phase de rendu il faut choisir le niveau de détails à utiliser. Celui-ci doit avoir une résolution légèrement inférieure à la taille d'un pixel une fois qu'il est projeté à l'écran (*cf.* figure 3.6) :

- une résolution trop fine conserve les problèmes d'une représentation sans niveaux de détails, à savoir un traitement de la visibilité complexe et un nombre de primitives à traiter important : le coût reste élevé et l'aliassage toujours présent (*cf.* figure 3.6 à gauche).
- une résolution trop grossière est immédiatement visible par l'utilisateur car l'ensemble de l'objet est uniforme et les fréquences sont gommées (*cf.* figure 3.6 à droite).

Le choix du bon niveau de détails est donc fonction de la taille de la primitive à l'écran, qui elle dépend de la distance entre l'objet et l'observateur.
Remarque : en acceptant un léger compromis sur la qualité, l'utilisateur peut forcer l'utilisation d'un niveau de détail légèrement au delà de sa limite de validité, diminuant ainsi la charge de calcul. L'utilisation d'un niveau de détails grossier hors de sa limite de validité tend à donner des images floues, ce qui peut être un avantage pour les ombres, ainsi que pour simuler la profondeur de champ.

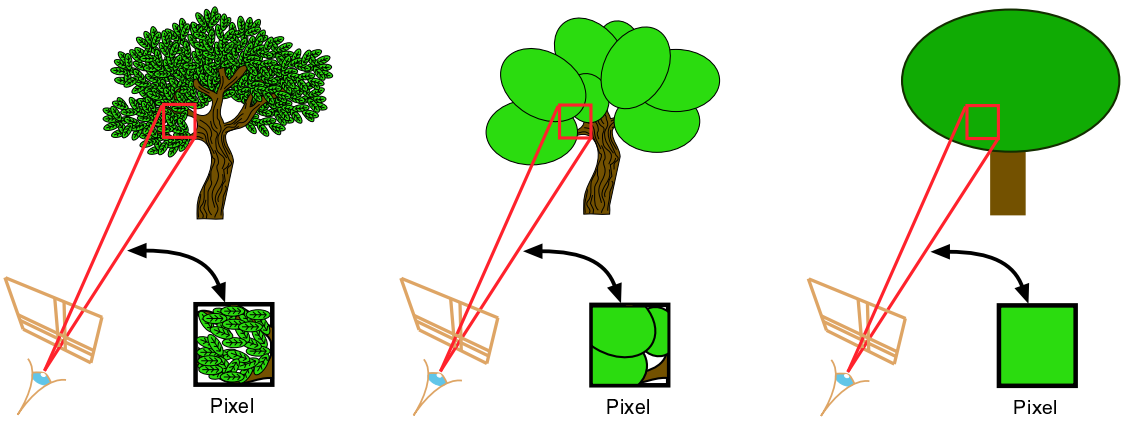

FIG. 3.6 – Le bon niveau de détails est choisi en fonction de l'éloignement de l'objet à l'œil mais aussi de la taille de ses primitives. *À gauche* : de nombreuses feuilles se trouvent dans le pixel ; on retombe dans les problèmes liés au grand nombre de détails (visibilité complexe, aliassage, coût élevé). *Au milieu* : une poignée de branches se trouvent dans le pixel ; le traitement sera rapide et ne fera pas apparaître de zone uniforme. Ce niveau de détails est le bon choix pour ce pixel. *À droite* : l'ellipse représentant l'arbre entier recouvre plusieurs pixels, ce qui donnera un aspect uniforme non réaliste à l'arbre.

CHAPITRE 4

Modèle analytique et hiérarchique d'illumination pour les conifères

La complexité d'un sapin est grande, plusieurs centaines de milliers d'aiguilles. Cependant, le détail de chaque aiguille n'est pas visible explicitement dès que l'observateur est éloigné de l'arbre. Dans ce chapitre nous appliquons l'idée de remplacer les données non visibles explicitement par une primitive "floue" reproduisant le même comportement photométrique que les géométries qu'elle représente. On dispose d'une connaissance a priori sur la géométrie d'un rameau de conifères, puisqu'une aiguille a une forme proche du cylindre et que leurs répartitions sur une branche est assez simple à caractériser (*cf.* Étude de cas).

Nous introduisons plusieurs primitives analytiques pour différents niveaux de détails. En 1 nous poserons les hypothèses et présenterons les concepts de notre modèle, puis nous présenterons les trois niveaux du modèle : en 2 le *shader* représentant une aiguille, en 3 le *shader* représentant une révolution d'aiguilles (cône d'aiguilles) et en 4 le *shader* représentant un rameau complet. Enfin, nous finirons en 5 par les résultats et en 6 par la conclusion.

1 Shader dédiés aux conifères

Nous posons ici les hypothèses et les bases de notre modèle avec, en 1.1 notre modèle d'arbre, en 1.2 une description des trois modèles d'illumination ou *shaders* formant la hiérarchie, et en 1.3 le détail de ce que nous devons calculer.

1.1 Notre modèle d'arbres (*cf.* figure 4.1)

- Un arbre est un ensemble de branches et d'aiguilles que nous construisons en utilisant un *L-system*.

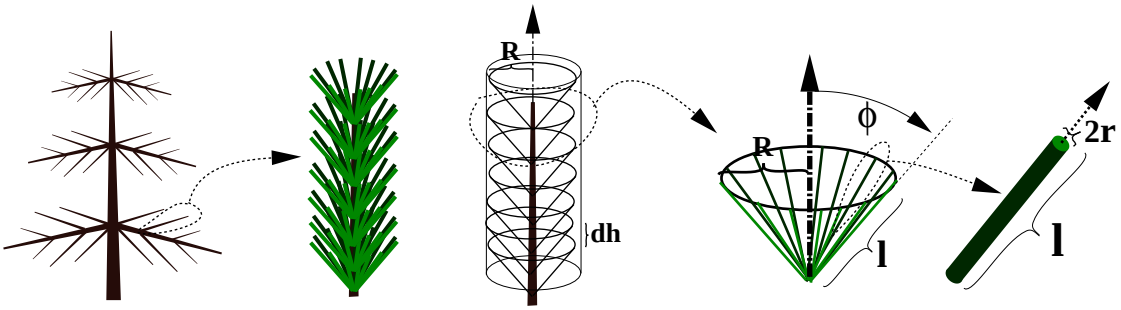

FIG. 4.1 – Notre description hiérarchique d'un arbre.

- Les branches sont représentées par de la géométrie classique.
- Les aiguilles sont des cylindres faisant un angle ϕ avec la branche, de longueur l, de rayon r et de densité (*i.e.* de distribution) ρ. Les paramètres changent peu le long de la branche, on peut les considérer comme constants.
- La projection d'une aiguille sur le plan perpendiculaire à la branche est de $R = l.\sin(\phi)$.
- Nous supposons que les aiguilles sont distribuées en cônes, avec N aiguilles par cône. La distance entre deux cônes le long de la branche est dh. L'espacement entre deux aiguilles est $\frac{2\pi R}{N}$, comme l'espacement entre deux cônes est dh, il est raisonnable de choisir $dh = \frac{2\pi R}{N} = \sqrt{\rho}$. Nous avons donc la relation $dh\frac{2\pi R}{N} = \rho$.

1.2 Rendu multi-échelle

En fonction de la distance, la plus petite primitive que nous utilisons à l'affichage est l'aiguille (niveau un), le cône (niveau deux) ou la branche (niveau trois). Nous rendons la scène en utilisant un lancer de cônes : le rayon conique est utilisé pour estimer la taille apparente de la primitive et pour calculer le recouvrement α du pixel. Nous utilisons aussi des rayons coniques pour l'ombrage, en supposant que la source de lumière est ponctuelle.

Le principal problème est de calculer la réflectance globale et l'opacité des primitives considérées en incluant les ombres internes. Puisque nous utilisons des rayons coniques, le sur-échantillonnage[1] est inutile (*i.e.* un unique rayon par pixel est lancé).

La contribution essentielle que nous apportons est la représentation multi-échelles, qui sera détaillée dans les sections suivantes, les trois *shaders* alors obtenus par intégration et la méthode que nous utilisons pour résoudre ces intégrales, en particulier l'interprétation géométrique de la visibilité et de l'ombrage effectuée pour le calcul du niveau 3.

1.3 Qu'avons-nous à calculer ?

Dans cette section nous estimons le travail à effectuer pour le calcul analytique des *shaders*. Le résultat et le détail des intégrations successives se trouvent dans les trois prochaines sections. Les vecteurs $\vec{L}$ et $\vec{V}$ sont considérés comme constants pour l'objet car la source de lumière et le point de vue sont éloignés.

[1] Le sur-échantillonnage peut être vu comme une approximation numérique du lancer de cônes (*cf.* chapitre 1 section 2.1.2).

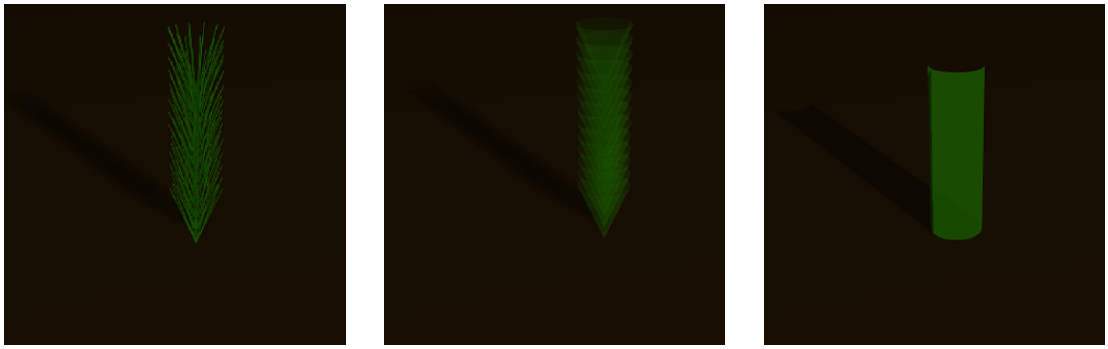

FIG. 4.2 – Les trois niveaux de modèles d'illumination ou *shaders*. *À droite* : la primitive est le cylindre représentant les aiguilles. *Au milieu* : la primitive est le cône représentant une révolution d'aiguilles. *À droite* : la primitive est le cylindre représentant un rameau d'aiguilles.

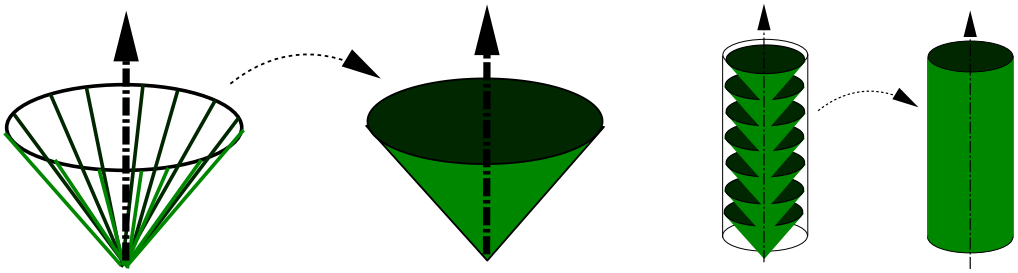

FIG. 4.3 – *À gauche* : le modèle de cône continu. *À droite* : le modèle de rameau continu.

1.3.1 Niveau un (aiguilles)

Pour calculer l'illumination d'une aiguille, nous devons intégrer la lumière diffuse I_d et spéculaire I_s reflétées par un cylindre (*cf.* chapitre 2 section 1.2.2). Nous ne calculons jamais explicitement l'intersection entre une aiguille et le rayon conique, mais l'intersection du rayon avec le cône d'aiguilles. Nous considérons que les aiguilles visibles sont sur la partie avant du cône, puis nous sommons les illuminations.

1.3.2 Niveau 2 (cônes)

En appliquant la loi des grands nombres, nous considérons que l'illumination d'un cône d'aiguilles est équivalente à l'illumination d'un cône semi-opaque continu où chaque point reflète la lumière comme une aiguille entière le ferait (*cf.* figure 4.3). L'opacité A est la moyenne de la surface du cône couverte par les aiguilles, ce qui donne $A = \frac{2N_r}{\pi R}$. L'illumination totale réfléchie est A multiplié par l'intégrale dans l'espace des pixels de l'illumination d'un cylindre sur la partie visible du cône. Les parties avant et arrière du cône sont considérées séparément, et seule une portion de cette partie peut être visible dans un pixel. L'intégration analytique n'est pas triviale, et nécessite certaines approximations.

1.3.3 Niveau 3 (rameaux)

Nous considérons que le *shader* d'un rameau d'aiguilles est équivalent à un cylindre volumique, anisotrope et semi-opaque, fait d'une imbrication de cônes (*cf.* figure 4.3). L'illumination et l'opacité de la partie avant et arrière du cône correspondent à celles du deuxième niveau que

nous avons vu précédemment. En appliquant la loi des grands nombres, le volume est considéré comme continu et anisotrope : l'opacité doit reproduire le même effet que celui produit par tous les cônes traversés par un rayon si le rendu se faisait avec le niveau 2, ce qui dépend fortement de l'angle du rayon (*cf.* figure 4.10). La partie difficile est l'intégration volumique analytique, en tenant compte de la visibilité et de l'ombrage. En supposant que l'on puisse utiliser une approximation linéaire de la loi de composition des opacités, *i.e.* $(1-A)^n \approx (1-nA)$ valide aux opacités faibles, nous transformerons cette intégrale en une forme géométrique.

2 Shader analytique d'une aiguille

Dans notre modèle, une aiguille est représentée par un cylindre (*cf.* chapitre 2 section 1.2.2), ayant comme *shader* celui de Phong. Nous devons donc intégrer les composantes diffuses et spéculaires d'un cylindre dans l'espace écran (*i.e.* nous devons sommer cette contribution dans le pixel en tenant compte de l'opacité).

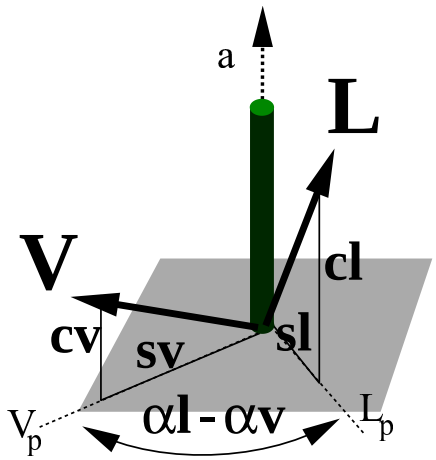

FIG. 4.4 – Une aiguille.

2.1 Illumination diffuse

La composante diffuse depuis le point de vue est

$$\begin{aligned} R_d^{cyl} &= \frac{\int_{pixel} (N.L)\mathbb{1}_{(N.L>0)} dS_{pix}}{\int_{pixel} 1 dS_{pix}} \\ &= \frac{\int_{cylindre} (N.L)\mathbb{1}_{(N.L>0)}.(N.V)\mathbb{1}_{(N.V>0)} dS}{\int_{cylindre} (N.V)\mathbb{1}_{(N.V>0)} dS} \end{aligned}$$

Posons c_v et c_l les projections de V et L sur l'axe du cylindre $\vec{a}$, *i.e.* $c_v = (\vec{a}.V)$ et $c_l = (\vec{a}.L)$ (*cf.* figure 4.4). Posons V_p et L_p les projections de V et L sur le plan perpendiculaire au cylindre et s_v et s_l leur norme. Ce qui nous donne :

$$R_d^{cyl} = \frac{\int_{\alpha=\alpha_0}^{\alpha_l} s_l \cos(\alpha - \alpha_L) s_v \cos(\alpha - \alpha_V) d\alpha}{\int_{\alpha=\alpha_V-\frac{\pi}{2}}^{\alpha_V+\frac{\pi}{2}} s_v \cos(\alpha - \alpha_V) d\alpha}$$

avec α_V et α_L l'angle entre V_p et L_p dans le plan. Les bornes de visibilité α_0 et α_1 sont $\alpha_V - \frac{\pi}{2}$ et $\alpha_L + \frac{\pi}{2}$ si $L \times V$ a la même direction que $\vec{a}$, sinon ces bornes sont $\alpha_0 = \alpha_L - \frac{\pi}{2}$ et $\alpha_1 = \alpha_V + \frac{\pi}{2}$. Nous introduisons $\Delta\alpha = |\alpha_V - \alpha_L|$, et obtenons alors :

$$R_d^{cyl} = \frac{s_l s_v/2(\sin(\Delta\alpha) + (\pi - \Delta\alpha)\cos(\Delta\alpha))}{2s_v}$$

$$\boxed{R_d^{cyl} = \frac{s_l}{4}(\sin(\Delta\alpha) + (\pi - \Delta\alpha)\cos(\Delta\alpha))} \tag{4.1}$$

2.2 Illumination spéculaire

L'illumination spéculaire depuis le point de vue est :

$$\begin{aligned} R_s^{cyl} &= \frac{\int_{pixel}(N.H)^n \mathbb{1}_{(N.H>0)}.dS_{pix}}{\int_{pixel} 1.dS_{pix}} \\ &= \frac{\int_{cylindre}(N.H)^n \mathbb{1}_{(N.H>0)}.(N.V)\mathbb{1}_{(N.V>0)}.dS}{\int_{cylindre}(N.V)\mathbb{1}_{(N.V>0)}.dS} \end{aligned}$$

avec le demi-vecteur $H = \frac{V+L}{|V+L|}$ et n l'exposant spéculaire.
Posons H_p, c_h, s_h et α_V défini comme pour L et V. Alors :

$$R_s^{cyl} = \frac{\int_{\alpha=\alpha_0}^{\alpha_1} s_h^n \cos^n(\alpha - \alpha_H) s_v \cos(\alpha - \alpha_V) d\alpha}{\int_{\alpha=\alpha_V-\frac{\pi}{2}}^{\alpha_V+\frac{\pi}{2}} s_v \cos(\alpha - \alpha_V) d\alpha}$$

Il est connu que $\cos^n(x)$ est similaire à $e^{-\frac{n}{2}x^2}$ pour n grand (ce qui est la cas ici). De plus, la densité de la fonction est concentrée sur $x = 0$ (l'écart type vaut $1/\sqrt{n}$, et n est généralement plus grand que 100), alors nous avons $\cos^n(x - x_0)f(x) \approx \cos^n(x - x_0)f(x_0)$. Finalement nous obtenons :

$$R_s^{cyl} \approx (s_h^n s_v \cos(\alpha_H - \alpha_V) \int_{\alpha=\alpha_0}^{\alpha_1} e^{-\frac{n}{2}(\alpha-\alpha_H)^2} d\alpha)/2s_v$$

Puisque $\int_{-\infty}^{\infty} e^{-\frac{1}{2}(\frac{x}{\sigma})^2} = \sqrt{2\pi}\sigma$, l'intégrale précédente devient $\sqrt{\frac{2\pi}{n}}$ si $\alpha_H \in [\alpha_0, \alpha_1]$ (ce qui est toujours le cas). Donc :

$$\boxed{R_s^{cyl} \approx \frac{1}{2} s_h^n \cos(\alpha_H - \alpha_V)\sqrt{\frac{2\pi}{n}}} \tag{4.2}$$

2.3 Opacité

L'opacité est la proportion du rectangle de l'aiguille apparent qui se projette dans le pixel. Si l'aiguille est totalement couverte par le pixel alors :

$$\boxed{alpha^{cyl} = \frac{2r.s_v l}{S_{pix}}} \tag{4.3}$$

où S_{pix} représente la surface de la section du rayon conique à la distance de la primitive. L'illumination diffuse et spéculaire valent donc $I_d = alpha R_d$ et $I_s = alpha R_s$.

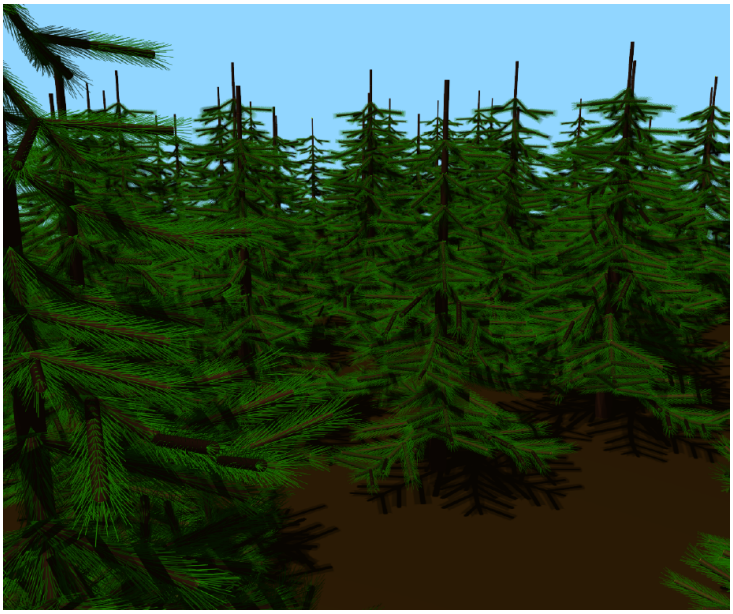

FIG. 4.5 – Notre *shader* de cylindre permet de représenter une forêt de sapins dont les aiguilles sont considérées comme cylindriques sans avoir à les trianguler ou à les échantillonner.

3 Shader analytique d'un cône d'aiguilles

Comme nous l'avons vu à la section 1.3.2, nous considérons à cette échelle qu'un cône d'aiguilles est une surface continue semi-opaque d'opacité A où chaque point de la surface a une réflexion identique à celle d'un cylindre. Nous devons donc intégrer l'illumination d'un cylindre sur un cône d'ouverture ϕ pour tous les axes $\vec{a_\theta}$ d'aiguilles valides. Dans le système de coordonnées polaires associé au cône nous avons $L = (\theta_L, \phi_L)$, où ϕ_L est l'angle entre L et l'axe du cône. De même nous avons $V = (\theta_V, \phi_V)$.

3.1 Illumination diffuse

L'illumination diffuse est donnée par :

$$\begin{aligned} I_d^{cone} &= A \int_{\theta=\theta_V-\frac{\pi}{2}}^{\theta_V+\frac{\pi}{2}} l.s_v R_d^{cyl}(\vec{a_\theta}) \\ &= \frac{l.A}{4} \int_{\theta=\theta_V-\frac{\pi}{2}}^{\theta_V+\frac{\pi}{2}} s_l s_v \left(\sin(\Delta\alpha) + (\pi - \Delta\alpha)\cos(\Delta\alpha)\right) \end{aligned}$$

où ls_v est la longueur apparente d'une aiguille. Cette formule n'est pas intégrable analytiquement. Nous approchons alors $s_l s_v(\sin(\Delta\alpha) + (\pi - \Delta\alpha)\cos(\Delta\alpha))$ en utilisant la fonction[2]

$$F = s_l s_v . \left(\frac{1+\cos(\Delta\alpha)}{2}\right)(2 + (\pi - 2)\cos(\Delta\alpha))$$

qui a la même valeur et la même dérivée en 0, $\frac{\pi}{2}$ et π. De plus, l'erreur maximale entre les deux fonctions est plus petite que 1%[3].

[2]Nous avons trouvé les différentes formules approximantes par tâtonnement, en cherchant des formes intégrables et en contrôlant les différences sous *Maple*.

[3]L'approximation de F par $\pi\left(\frac{1+\cos(\Delta\alpha)}{2}\right)^{1.6515}$ est aussi très bonne : l'erreur est plus petite que 1.5%. Il est plus

Si

$$\cos(\Delta\alpha) = \frac{(L_p.V_p)}{(|L_p|.|V_p|)} = \frac{(L.V) - c_l c_v}{s_l s_v}$$

alors

$$\int F = (L.V + s_l s_v - c_l c_v)\,.(2 + (\pi - 2)(L.V - c_l c_v)/s_l s_v)$$

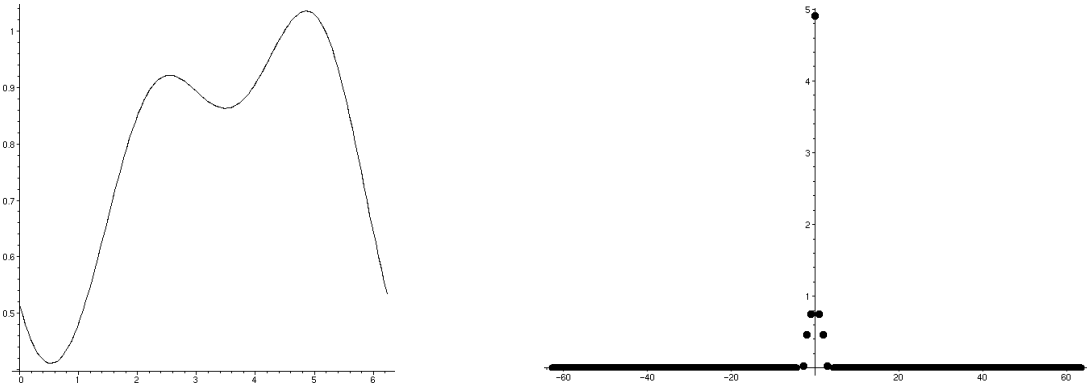

FIG. 4.6 – *À gauche* un exemple de la courbe F, pour $L = (0, 1.2)$, $V = (1, 1.5)$ et $\phi = 0.5$. Cette courbe est plutôt lisse malgré l'aspect complexe de ses facteurs. *À droite* : la FFT de cette courbe. Notez que l'énergie est clairement concentrée dans les fréquence 0, 1 et 2, la motivation d'approcher F avec une combinaison linéaire de 1, $\cos(\theta - \theta_A)$, $\cos(2(\theta - \theta_B))$ est donc justifiée. NB : les valeurs à droite sont dûes à la symétrie de la FFT.

Si nous traçons cette fonction avec *Maple* pour plusieurs valeurs des paramètres L, V et ϕ, il apparaît que la courbe est très lisse (*cf.* figure 4.6 à gauche) et ressemble à une combinaison linéaire de 1, $\cos(\theta - \theta_A)$ et $\cos(2(\theta - \theta_B))$. L'évaluation de la FFT sur des courbes discrétisées montre qu'il n'y a pratiquement pas d'énergie en dehors des fréquences 0, 1 et 2 (*cf.* figure 4.6 à droite). Nous essayons donc d'approcher la courbe F en utilisant des positions et des valeurs des extrema. Le premier facteur est responsable d'un maximum de variation de F et est facile à calculer. De ce fait, nous approchons la courbe F par $(L.V) + s_l s_v - c_l c_v$ dont les extrema correspondent aux mêmes valeurs de θ que F.

Le terme $c_l c_v - s_l s_v$ est égal à $\cos(\widehat{AL} + \widehat{AV})$ avec $\widehat{AL}$ l'angle entre les vecteurs $\vec{a}$ et L, et $\widehat{AV}$ l'angle entre les vecteurs $\vec{a}$ et V. Ces angles varient peu entre leur minimum et leur maximum si le vecteur $\vec{a}$ tourne autour du cône, nous représentons donc les variations de $\widehat{AL}$ par la forme $A_L + B_L \cos(\theta - \theta_L)$ avec $A_L = \max(\phi_L, \phi)$, $B_L = \min(\phi_L, \phi)$. Nous faisons de même pour $\widehat{AV}$.

Si nous développons $\widehat{AL} + \widehat{AV}$ avec cette approximation nous obtenons l'expression :

$$A_\Sigma + B_\Sigma \cos(\theta - \theta_\Sigma)$$

avec

$$\begin{aligned} A_\Sigma &= A_L + A_V \\ B_\Sigma^2 &= B_L^2 + B_V^2 + 2B_L B_V \cos(\theta_L - \theta_V) \\ \cos(\theta_\Sigma) &= (B_L \cos(\theta_L) + B_V \cos(\theta_V))/B_\Sigma \\ \sin(\theta_\Sigma) &= (B_L \sin(\theta_L) + B_V \sin(\theta_V))/B_\Sigma \end{aligned}$$

simple d'utiliser une puissance de 1.5, 2 ou 1 (avec des erreurs respectives de 4%, 7%, 18% à la place de $\frac{\log(\pi)}{\log(2)}$), pour résoudre l'intégrale analytiquement.

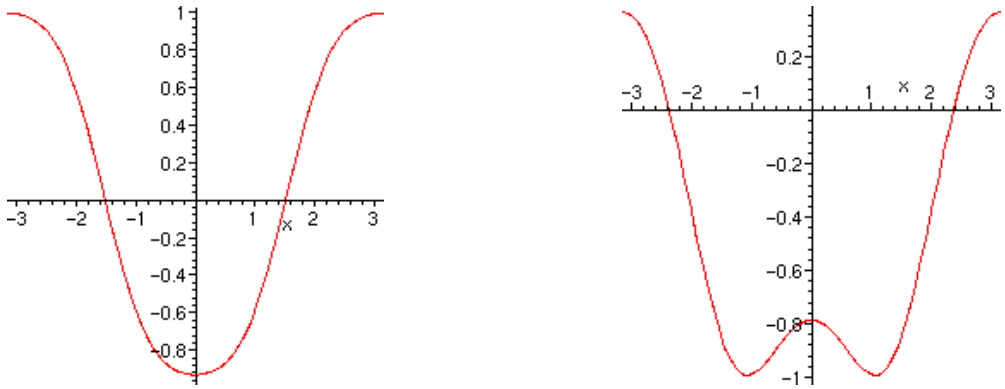

FIG. 4.7 – Les deux types d'aspects de la courbe $\cos(A_\Sigma + B_\Sigma * \cos(\theta - \theta_\Sigma))$, dépendant de ce que $A_\Sigma + B_\Sigma * \cos(\theta - \theta_\Sigma)$ croise π (*à droite*) ou non (*à gauche*).

Notre but étant d'approcher F, nous cherchons les extrema de $F \approx (L.V) - \cos(\widehat{AL} + \widehat{AV})$. Ils correspondent aux extrema de $\widehat{AL} + \widehat{AV}$, soit à la valeur pour laquelle $\widehat{AL} + \widehat{AV}$ croise π. Si $\widehat{AL} + \widehat{AV}$ ne croise pas π, F est similaire à une fonction cosinus. Si elle le croise, F a un "chapeau" et ressemble à la combinaison d'un cosinus et d'un cosinus à fréquence double (*cf.* figure 4.7). La similitude est grande si ϕ_L et ϕ_V ne sont pas proches de ϕ. Alors, nous pouvons obtenir explicitement les extrema de la courbe.
Comme nous essayons précisément d'approcher F sous la forme

$$(L.V) - (\lambda_0 + \lambda_1 \cos(\theta - \theta_m) + \lambda_2 \cos(2(\theta - \theta_m)))$$

nous pouvons utiliser les paramètres issus de ces extremum.
Soit $M = \cos(A_\Sigma - B_\Sigma)$ et $m = \cos(A_\Sigma + B_\Sigma)$, alors
$\theta_m = \theta_\Sigma$,
$\lambda_1 = \frac{(m-M)}{2}$,
$\lambda_0 = \frac{(m+M)}{2} - \lambda_2$, avec $\lambda_2 = 0$ dans le cas où aucun croisement avec π n'apparaît (*i.e.* $A_\Sigma + B_\Sigma$ et $A_\Sigma - B_\Sigma$ sont entre $[0, \pi]$) ;
$\lambda_2 = \frac{\lambda_1 B_\Sigma}{4(2\pi - A_\Sigma)}$ dans le cas d'un croisement avec π ($A_\Sigma + B_\Sigma > \pi > A_\Sigma - B_\Sigma$).
Maintenant, nous pouvons facilement obtenir l'intégrale de F :

$$I_d^{cone} = \frac{lA}{4} \int_{\theta=\theta_V-\frac{\pi}{2}}^{\theta_V+\frac{\pi}{2}} F$$

$$\boxed{I_d^{cone} = \frac{l.A}{4} (\pi(LV - \lambda_0) - 2\lambda_1 \cos(\theta_V - \theta_\Sigma))} \quad (4.4)$$

où

$$\cos(\theta_V - \theta_\Sigma) = \frac{B_L \cos(\Delta\theta) + B_V}{\sqrt{B_L^2 + B_V^2 + 2B_L B_V \cos(\Delta\theta)}} \text{ et } \Delta\theta = \theta_L - \theta_V.$$

3.2 Illumination spéculaire

L'illumination spéculaire est donnée par :

$$I_s^{cone} = A \int_{\theta=\theta_V-\frac{\pi}{2}}^{\theta_V+\frac{\pi}{2}} l.s_\nu R_s^{cyl}(\vec{a_\theta})$$

$$= \frac{l.A}{2}\sqrt{\frac{2\pi}{n}}\int_{\theta=\theta_V-\frac{\pi}{2}}^{\theta_V+\frac{\pi}{2}} s_l s_h^n \cos(\alpha_H - \alpha_V)$$

avec ls_v la longueur apparente d'une aiguille. Encore une fois, s_h^n est une fonction dont la densité est concentrée quand $s_h = 1$, ce qui arrive quand $c_h = 0$, c'est-à-dire quand H est orthogonal à l'axe d'une aiguille $\vec{a}$. Donc la valeur $\theta_H^\perp$ existe seulement si $\phi_H \in [\frac{\pi}{2} - \phi, \frac{\pi}{2} + \phi]$, sinon $I_s^{cone} = 0$. Si $\theta_H^\perp$ existe, nous avons à nouveau $s_h^n f(\theta) \approx s_h^n f(\theta_H^\perp)$.
Comme $s_l s_h \cos(\alpha_H - \alpha_V) = (V.H) - c_h c_v$, nous obtenons finalement :

$$\boxed{I_s^{cone} \approx \frac{l.A}{2}\frac{2\pi}{n}(V.H)\epsilon} \tag{4.5}$$

où $\epsilon = 1$ si $\phi_H \in [\frac{\pi}{2} - \phi, \frac{\pi}{2} + \phi]$, sinon $\epsilon = 0$.

Remarque : quand les deux valeurs où H est orthogonal à $\vec{a}$ apparaissent sur la même face (avant ou arrière), $\epsilon = 2$.

3.3 Opacité

L'opacité est donnée par :

$$alpha^{cone} = A\int_{\theta=\theta_V-\frac{\pi}{2}}^{\theta_V+\frac{\pi}{2}} ls_v$$

Comme $s_v = \sin(\widehat{AV})$, nous approchons $\widehat{AV}$ by $A_V + B_V\cos(\theta - \theta_V)$ par le même moyen que pour la composante diffuse. C'est-à-dire :

$$alpha^{cone} = lA(\lambda_0^V\pi + 2\lambda_1^L\cos(\theta_V - \theta_V)) \ , \ i.e$$

$$alpha^{cone} = l.A(\pi\cos(\phi)\cos(\phi_V) - 2\sin(\phi)\sin(\phi_V)) \tag{4.6}$$

4 Modèle d'illumination analytique d'un rameau d'aiguilles

Nous considérons qu'un rameau d'aiguilles est un volume ayant une forme de cylindre et une opacité anisotrope (*cf.* figure 4.9). Pour calculer le *shader* analytique d'un tel objet nous devons calculer analytiquement un rendu volumique du cylindre.

Comme l'opacité A n'est pas constante le long du rayon et du rayon d'ombrage, nous obtenons :

$$I = \frac{1}{S_{pix}}\int_{(x,y)\in pixel}\int_{z=near}^{far} AI^{cyl}e^{-\int_0^{l_z}\sigma}e^{-\int_0^{l_{shad}}\sigma} \tag{4.7}$$

avec $e^{-\sigma} = T = (1 - A)$ la transparence anisotrope, l_z la longueur du rayon dans le volume et l_{shad} la longueur du rayon d'ombrage dans la volume.

Nous devons maintenant exprimer l'opacité et calculer l'intégrale, ce qui requière quelques approximations.

4.1 Traversée d'un rameau d'aiguilles en 2D

Nous nous plaçons dans le cas d'une branche infinie avec un empilement d'aiguilles ayant une direction ϕ relativement à l'axe de la branche (*cf.* figure 4.8 à gauche). Posons R le rayon de la branche et dh la distance verticale entre deux cônes d'aiguilles. Soit un rayon traversant la branche, faisant un angle ϕ_r avec l'axe de celle-ci. La longueur du rayon à l'intérieur de la branche est $R/sin(\phi_r)$.
Les distances entre les intersections sont $\delta = dh\frac{\sin(\phi)}{\sin|\phi_r-\phi|}$
Le nombre moyen d'intersections est $\frac{R}{dh}\frac{\sin|\phi_r-\phi|}{\sin(\phi)\sin(\phi_r)}$
Notons $k(\phi_r,\phi)$ la quantité

$$k(\phi_r,\phi) = \frac{\sin|\phi_r-\phi|}{\sin(\phi)\sin(\phi_r)} = |\frac{1}{\tan(\phi)} - \frac{1}{\tan(\phi_r)}|$$

L'opacité de la branche le long du rayon est $1 - T^{\frac{R}{dh}k(\phi_r,\phi)}$

Afin de raccourcir les notations, notons $k_r = k(\phi_r,\phi)$ et $\overline{k}_r = k(\phi_r,\pi-\phi)$. $\overline{k}_r$ correspond à la traversée d'une branche ayant une symétrie axiale.

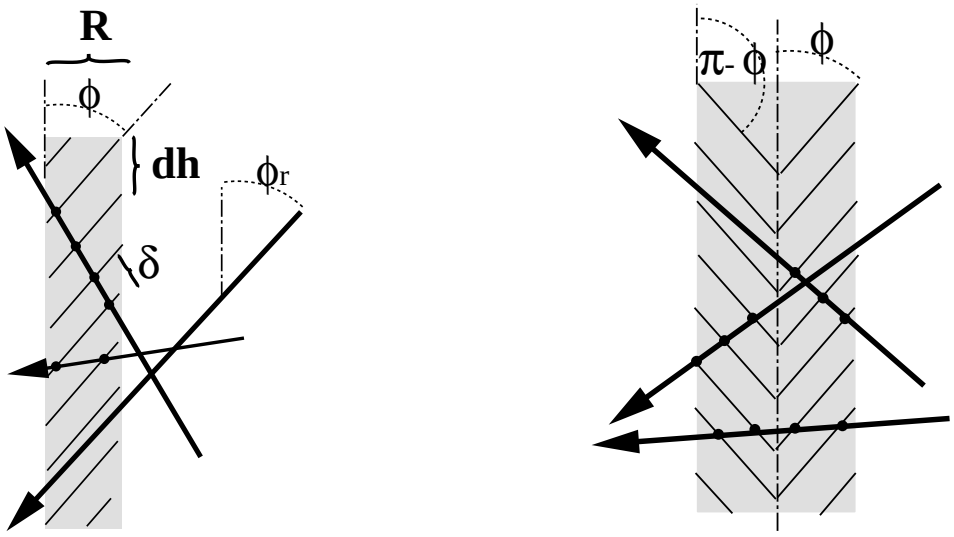

FIG. 4.8 – *À gauche* : champ 2D d'aiguilles parallèles. *À droite* : rameau d'aiguilles 2D. Notez la variation de l'opacité en fonction de la direction du rayon (surtout visible à gauche).

Un rameau d'aiguilles 2D est composé de deux champs d'aiguilles différents, celui de droite ayant pour orientation d'aiguilles ϕ et celui de gauche $\pi - \phi$ (*cf.* figure 4.8 à gauche). Le nombre total d'intersections le long du rayon est donc :

$$\frac{R}{dh}(k_r + \overline{k}_r) = \frac{R}{dh}\frac{\sin|\phi_r-\phi| + \sin|\phi_r+\phi|}{\sin(\phi)\sin(\phi_r)} = \frac{R}{dh}\frac{2}{\tan(\min(\phi,\phi_r))}$$

Ce qui signifie que pour un rayon restant dans l'ouverture du cône d'aiguilles, l'opacité totale reste constant et ce, quelle que soit la partie du cône qui est devant et qui est derrière. Ceci est également vrai pour le rayon d'ombre. Si le rayon passe par l'ouverture du cône (par en haut ou par en bas) l'opacité passe à 100% quand ϕ_r = 0 ou π.

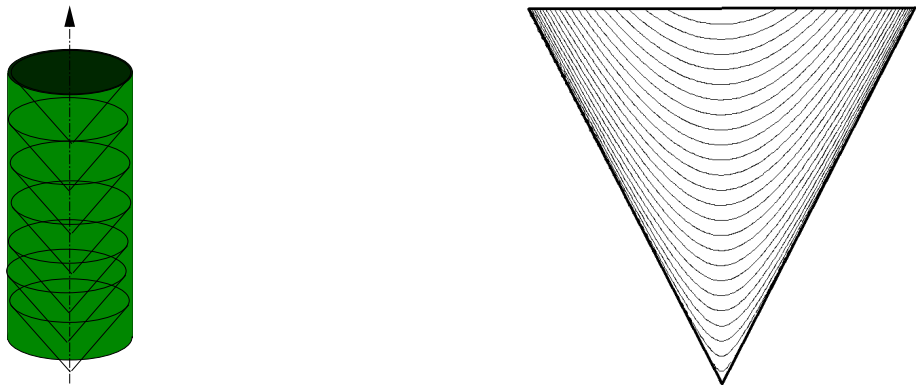

FIG. 4.9 – *À gauche* : nous modélisons un rameau d'aiguilles par un cylindre volumétrique semi-opaque. Cette opacité est anisotrope et reproduit la variation du nombre d'intersections entre le rayon et les cônes sous-jacents. *À droite* : intersection d'un plan P_x avec un cône. Nous approchons les hyperboles par leur asymptote.

4.2 Extension à la 3D

Revenons à notre rameau d'aiguilles en 3D. En 3D, si un rayon traverse l'axe de la branche, la situation est équivalente au cas 2D vu précédemment. Mais généralement, un rayon ne traverse pas exactement l'axe. Plaçons nous dans le plan parallèle à l'axe du cône contenant le rayon. Soit x la distance à l'axe et P_x ce plan. L'intersection du volume du rameau d'aiguilles (constitué de cônes) avec le plan donne un ensemble d'hyperboles. Nous approchons raisonnablement ces hyperboles par leurs deux asymptotes (*cf.* figure 4.9 à droite). Le plan contient les aiguilles ayant la même orientation ϕ et le même décalage dh qu'en 3D, dans une branche d'épaisseur $2R_x$ avec $R_x = \sqrt{R^2 - x^2}$. Nous pouvons donc calculer le nombre d'intersections en utilisant la formule 2D. Pour estimer la quantité de lumière atteignant un point du rayon, nous considérons le rayon d'ombre partant de ce point. De manière similaire, nous introduisons le plan parallèle à l'axe du cône et contenant ce rayon d'ombrage (*cf.* figure 4.10). Le nombre d'intersections peut être obtenu comme pour le rayon principal.

4.3 Traversée d'un rameau d'aiguilles 3D

Nous pouvons maintenant revenir à l'intégrale volumique (4.7). Nous choisissons la paramétrisation cartésienne de la surface de manière à ce que l'axe $\vec{x}$ soit orthogonal au cylindre. Donc le plan P_x est indexé par x (*i.e.* x est cohérent avec la section précédente). En conséquence, il n'est pas nécessaire d'intégrer le long de l'axe $\vec{y}$, puisque le cylindre est homogène dans cette direction. Remarquez que l'albédo A de l'équation doit être corrigée en A/δ, puisque aucune énergie n'est présente dans l'espace entre deux cônes. De manière similaire pour la longueur dl, l'opacité est $e^{-\sigma dl} = T^{dl/\delta}$. Effectuons le changement de variable de (x, z) vers (x, z') dans le plan orthogonal au cylindre. Cela signifie que nous indexons un point sur le rayon par sa projection sur le plan orthogonal. Le jacobien de la transformation est $\frac{1}{\sin(\phi)}$. L'opacité associée à un élément de longueur dl' sur le plan est $T^{\frac{dl'}{\sin(\phi)\delta}} = T^{dl'\frac{k()}{dh}}$.

4.4 Division de l'intégrale en régions

Nous tirons comme information du cas 2D que l'opacité le long du rayon est constante pour la partie avant et pour la partie arrière (ces deux parties correspondent aux deux orientations des

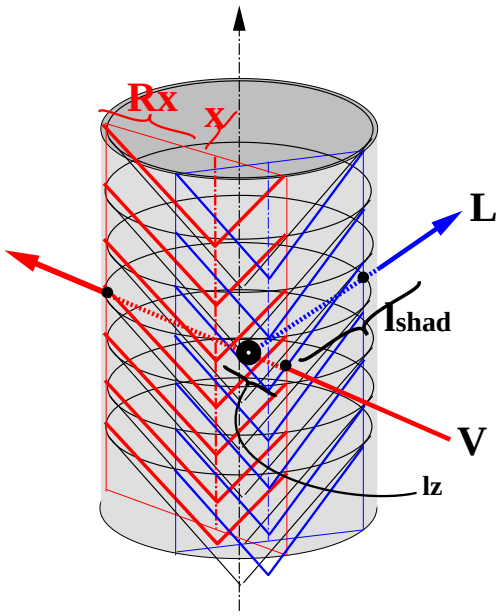

FIG. 4.10 – Le volume intersecté par le plan vertical contenant le rayon ressemble au cas 2D.

aiguilles dans le plan P_x). Nous avons donc :

$$I = \frac{1}{2R}\int_{x=-R}^{R}\left(\int_{z=-R_x}^{0} A\frac{k_v}{dh}.I^{cyl}.T^{\frac{k_v}{dh}(R_x+z)}.T^{\int_0^{l_{shad}}\frac{k()}{dh}} + \int_{z=0}^{Rx} A\frac{\overline{k}_v}{dh}.I^{cyl}.T^{(\frac{k_v}{dh}R_x+\frac{\overline{k}_v}{dh}z)}.T^{\int_0^{l_{shad}}\frac{k()}{dh}}\right)$$

Nous divisons le disque (de la section du rameau d'aiguilles) en deux régions F_V et R_V (avant et arrière) relativement à V (*cf.* figure 4.11), tout comme à la section 3, nous avions divisé les cônes en une face avant et une face arrière, afin d'évaluer l'illumination. Sur chaque région $k()$ est constante. Supposons que I^{cyl} soit constante pour chacune des deux régions du volume, et approchons la par la valeur I^{cyl}_{front} et I^{cyl}_{rear}. L'intégrale devient

$$I = \frac{A}{2Rdh}\int_{x=-R}^{R}\left(k_v I^{cyl}_{front}\int_{z=-R_x}^{0} T^{\frac{k_v}{dh}(R_x+z)}T^{\int_0^{l_{shad}}\frac{k()}{dh}} + \overline{k}_v I^{cyl}_{rear}\int_{z=0}^{Rx} T^{(\frac{k_v}{dh}R_x+\frac{\overline{k}_v}{dh}z)}T^{\int_0^{l_{shad}}\frac{k()}{dh}}\right)$$

Pour résoudre l'intégrale, nous allons diviser à nouveau le disque dans le but de séparer la partie avant et arrière F_L et R_L relativement à L. La longueur du rayon d'ombre dépend de z d'une manière complexe, ce qui rend l'exponentiel difficile à intégrer analytiquement.

Pour rendre cette intégrale réalisable analytiquement, nous utilisons une approximation linéaire de composition des opacités : $(1-A)^n \approx (1-nA)$ qui est valide si $nA \ll 1$ (*i.e.* si le rameau d'aiguilles n'est pas trop dense). Alors $(1-A)^{n_1}(1-A)^{n_2} \approx 1-n_1A-n_2A$, ce qui assure la séparation des facteurs. L'intégrale se définit donc par :

$$I = I_{F_V} + I_{R_V} = \frac{A}{2Rdh}\left(I^{cyl}_{front}k_v I_{F'_V} + I^{cyl}_{rear}\overline{k}_v I_{R'_V}\right)$$

avec

$$I_{F'_V} = \int_{F_V} 1 - A\int_{F_V}\frac{k_v}{dh}(R_x+z) - A\int_{F_V\times R_L} z_{shad}\frac{\overline{k}_l}{dh} - A\int_{F_V\times F_L} z_{shad}\frac{k_l}{dh}$$

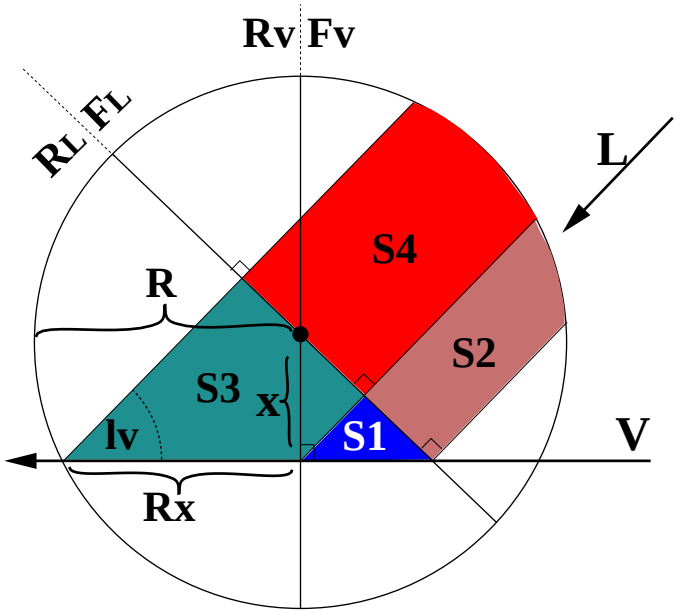

FIG. 4.11 – Le volume de la branche d'aiguille vu par une section orthogonale. Les surfaces des quatre régions $S_1 = F_V \times R_L$, $S_2 = F_V \times F_L$, $S_3 = R_V \times R_L$, $S_4 = R_V \times F_L$ sont proportionnelles à l'intégrale de la longueur du rayon d'ombre, avec l'origine placée sur le rayon (seul le cas générique figure ici). Nous devons intégrer cette surface pour tous les x.

$$I_{R'_V} = \int_{R_V} 1 - A \int_{R_V} (\frac{k_\nu}{dh} R_x + \frac{\overline{k}_\nu}{dh} z) - A \int_{R_V \times R_L} z_{shad} \frac{\overline{k}_l}{dh} - A \int_{R_V \times F_L} z_{shad} \frac{k_l}{dh}$$

avec $F_V \times R_L$ la région de R_L couverte par les rayons d'ombres dont l'origine est dans F_V et ainsi de suite pour les autres régions (*cf.* figure 4.10).

4.5 Intégration géométrique

Nous arrangeons ceci ainsi :

$$I_{F'_V} = \pi \frac{R^3}{2} - A \frac{k_\nu}{dh} \frac{2}{3} R^3 - A \frac{\overline{k}_l}{dh} \int_{F_V \times R_L} z_{shad} - A \frac{k_l}{dh} \int_{F_V \times F_L} z_{shad}$$

$$I_{R'_V} = \pi \frac{R^2}{2} - A \frac{k_\nu}{dh} \frac{4}{3} R^3 - A \frac{\overline{k}_\nu}{dh} \frac{2}{3} R^3 - A \frac{\overline{k}_l}{dh} \int_{R_V \times R_L} z_{shad} - A \frac{k_l}{dh} \int_{R_V \times F_L} z_{shad}$$

Les quatre intégrales restant dans les formules ci-dessus somment la longueur des rayons d'ombrage qui commencent sur chaque point du rayon et sont inscrits dans la région indiquée en indice, pour chaque rayon incident. Pour le moment, considérons seulement l'intégrale le long du rayon. Le rayon d'ombre balaie une région puisque son origine suit le rayon. L'intégrale de sa longueur le long du rayon principal ressemble fortement à la surface de cette région : elle est proportionnelle à $\frac{1}{\sin(l_\nu)}$ où l_ν est l'angle entre la projection L_P et V_P de L et V dans le plan orthogonal. La "preuve" de ceci est que si P_L est orthogonal à V_P alors l'intégrale de la longueur est l'aire d'une surface. Autrement, on peut se replacer dans ce cas avec un changement de variables ayant pour jacobien $\frac{1}{\sin(l_\nu)}$. Afin de calculer l'intégrale le long du rayon, nous devons mesurer l'aire de chaque région balayée S_1, S_2, S_3, S_4 en utilisant des relations géométriques et trigonométriques. Nous devons ensuite intégrer le résultat pour chaque rayon. Après de longs et fastidieux

calculs nous obtenons un résultat simple et symétrique (sans approximation) :

$$\begin{aligned}\int S_1 &= (1+\cos(l_v))\frac{R^3}{3}\sin(l_v)\\ \int S_2 &= (1-\cos(l_v))\frac{R^3}{3}\sin(l_v)\\ \int S_3 &= (1+\cos(l_v)/3)R^3\sin(l_v)\\ \int S_4 &= (1-\cos(l_v)/3)R^3\sin(l_v)\end{aligned} \tag{4.8}$$

Le facteur $\sin(l_v)$ disparaît quand on multiplie par le jacobien.

4.6 Résultat de l'illumination d'un rameau d'aiguilles

L'opacité se déduit facilement :

$$1-\alpha_{F_V} = \frac{1}{2R}\int_{x=-R}^{R} A^{R_x \frac{k_v}{dh}} \approx 1-\frac{AR}{dh}\frac{\pi}{4}k_v$$

i.e.

$$\boxed{\alpha_{F_V} = ak_v\ ,\ \ \alpha_{R_V} = a\overline{k}_v} \tag{4.9}$$

Nous introduisons de manière similaire l'opacité dans la direction de la lumière : $\alpha_{F_L} = ak_l$, $\alpha_{R_L} = a\overline{k}_l$ et nous avons finalement $I = I_{F_V} + I_{R_V}$ avec

$$I_{F_V} = I^{cyl}_{front}\alpha_{F_V}\left(1-\frac{8}{3\pi^2}\left(2\alpha_{F_v}+(1-\cos(lv))\alpha_{R_L}+(3-\cos(lv))\alpha F_L\right)\right)$$

$$\boxed{I_{R_V} = I^{cyl}_{rear}\alpha_{R_V}\left(1-\frac{8}{3\pi^2}\left(4\alpha_{F_v}+2\alpha_{R_v}+(1+\cos(lv))\alpha_{R_L}+(3+\cos(lv))\alpha F_L\right)\right)} \tag{4.10}$$

5 Résultats

Une propriété majeure de ce modèle est l'évolution de son coût en fonction du nombre d'aiguilles, *i.e.* la complexité en fonction du nombre d'aiguilles N par cône et du nombre $\frac{l}{dh}$ de cônes sur une branche par unité de longueur (ces deux nombres sont proportionnels à la racine carrée de la densité d'aiguilles). Si N est multiplié par deux, le nombre d'intersections pour le niveau un et le nombre d'échantillons par pixel qu'un lancer de rayons doit traiter est multiplié par deux, alors que les niveaux deux et trois ne sont pas affectés. Ce raisonnement s'applique si dh est divisé par deux. Le coût des rayons d'ombrage évolue de la même manière. Alors qu'un lancer de rayons classique lance un rayon d'ombre pour chaque échantillon, notre modèle factorise le rayon pour la partie extérieure à la branche.

Nous avons comparé l'efficacité de notre modèle avec un lancer de rayons classique, *Rayshade*, utilisant le sur-échantillonnage pour diminuer l'aliassage. Il est important de savoir que le nombre maximum de rayons lancé par pixel de *Raushade* est de 64. Donc, quand un arbre est loin (c'est-à-dire moins de 100 pixels de haut), *Rayshade* ne lance pas assez de rayons pour éviter l'aliassage.

Il semble efficace en temps de calcul mais c'est au dépend de la qualité. Sur une image fixe comportant beaucoup de hautes fréquences, comme peut l'être une image de forêt, l'aliassage n'est pas toujours visible parce qu'il est difficile de distinguer le bruit de l'information. Par contre, dès que l'on calcule une animation, l'aliassage apparaît immédiatement (grouillement).

Notre scène de test comporte 80 sapins, qui représentent à peu près 127 pixels de haut pour les plus proches et 64 pour les plus lointains (*cf.* figure 4.12).

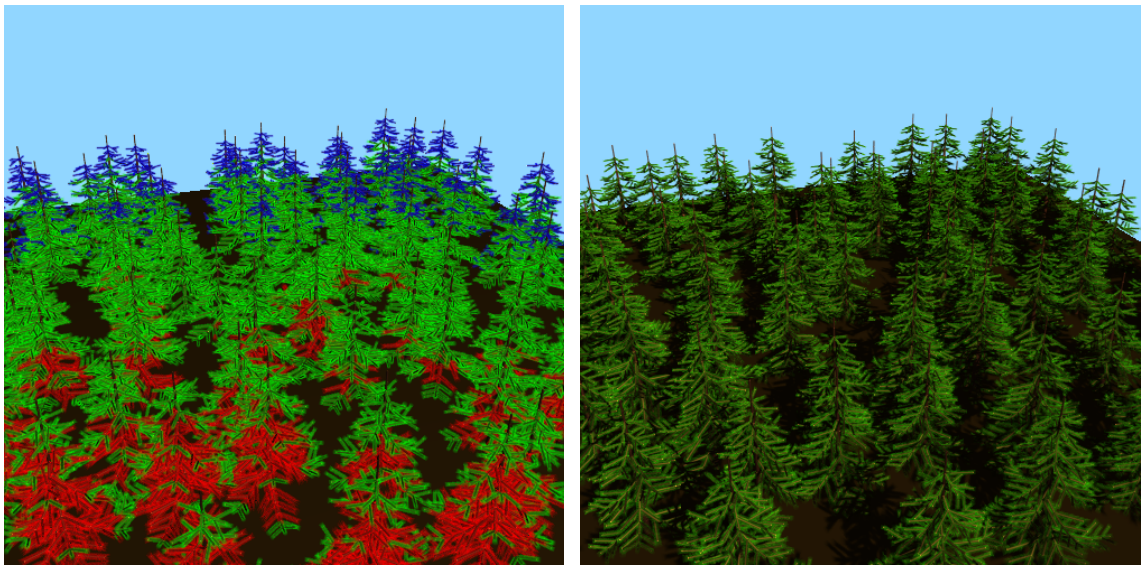

FIG. 4.12 – Notre scène de test. *À gauche* : Les trois niveaux de détails (niveau un en rouge, niveau deux en vert et niveau trois en bleu). *À droite* : 80 sapins.

Les sapins utilisés pour nos tests contiennent généralement 300 branches et à peu près 30000 aiguilles, la scène contient donc environ 2 millions d'aiguilles. Concernant une branche, un cône a 3.94 cm de haut et à un rayon de 1.6 cm, une ouverture de $\pi/8$ et le pas entre les cônes est de 0.9 cm. Il y a 12 aiguilles par cône pour ces arbres, qui ont un rayon de 0.05 cm et une longueur de 4.25 cm. En moyenne, 4.4 cônes sont imbriqués, par conséquent un rayon passant à travers l'axe orthogonalement à la branche traversera en moyenne 8.8 couches.

Considérons maintenant les temps de rendu. Les tests de comparaison entre notre modèle et le modèle classique[4] qu'utilise *Rayshade* ont été effectués sur une Onyx^2 Infinite Reality en exploitant un seul processeur. *Rayshade* a effectué le rendu de l'image 4.12 en 65.3 minutes avec une optimisation à base de grilles. Notre modèle[5] calcule l'image en 8.1 minutes. Avec une implémentation largement améliorable notre méthode est 8.1 fois plus rapide que le modèle classique représenté par *Rayshade*. Pour un grand paysage où les arbres lointains sont très petits, *Rayshade* ne peut empêcher un fort aliassage du fait de son seuil maximum de 64 rayons par pixel. Si nous pouvions augmenter cette limite, le gain en notre faveur serait largement plus important.

[4] Le système *Rayshade* utilise le sur-échantillonnage pour diminuer l'aliassage, et toutes les aiguilles sont représentées par des cylindres.

[5] Notre implémentation utilise le lancer de cônes (un unique rayon par pixel) pour diminuer l'aliassage et nous utilisons les 3 niveaux de détails.

5.1 Parallèlisation de l'algorithme

Malgré l'accélération due à notre modèle, une scène complexe calculée par lancer de rayons devient vite très coûteuse en temps de calcul. Diminuer ces temps de calcul est intéressant dans le cadre d'applications de mise au point ou de rendu interactif : pour un artiste il est commode de disposer d'un aperçu du résultat rapidement. Pour ceci, une des possibilités à étudier est le calcul parallèle.

Nous avons parallélisé [MC01] notre lancer de cônes (*cf.* annexe B) suivant un schéma de parallélisation classique, *i.e.* en découpant une animation ou une image en sous-parties de manière à distribuer les tâches. Notre première machine de test a été une Onyx^2 Infinite Reality avec 6 processeurs (*MIPS R12000* à 400MHz), puis nous avons effectué le portage de l'application sur une grappe expérimentale de PC[6]. Les résultats de cette parallélisation sont quasi-linéaires sur les deux plates-formes : avec 6 processeurs une animation est calculée 6 fois plus vite et une image 5.9 fois plus vite. En performance absolue la grappe de 12 PC est environ 30% plus rapide que l'*Onyx* avec 6 processeurs. La différence se situe donc au niveau du coût financier des deux machines : l'*Onyx* coûte à peu près quatre fois le prix d'une telle grappe. On en conclue que pour une application comme la notre, où l'algorithme est fortement parallélisable, l'investissement dans une grappe de PC est un choix à envisager. Consulter l'annexe B pour plus de détails.

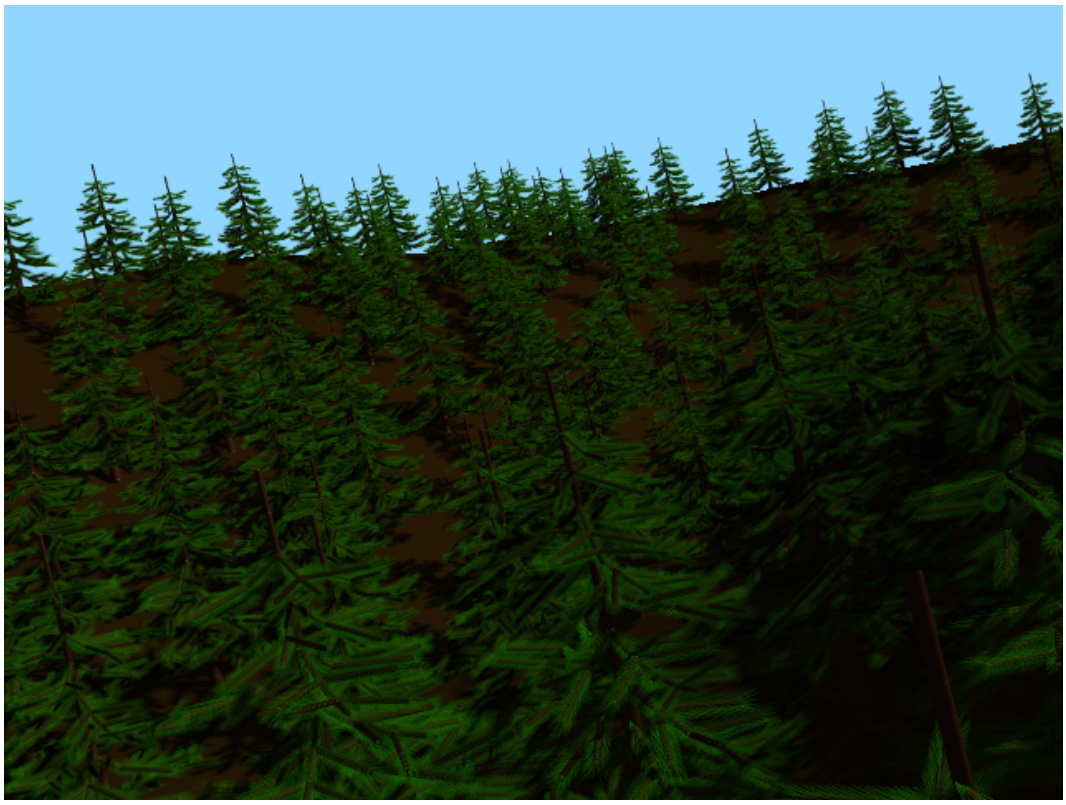

FIG. 4.13 – Une scène de 1000 arbres.

[6]Nous avons utilisé la grappe de l'équipe *SIRAC* composée de 12 PC Pentium II (450MHz) sous Linux, disposant du module expérimentale *SciFS* (developpé par Emmanuel Cecchet) offrant une mémoire partagée distribuée facilitant la gestion de la mémoire pour des applications distribuées.

6 Conclusion et perspectives

Nous avons introduit un ensemble de trois *shaders* capables de représenter à différents niveaux (aiguilles, cônes et rameaux) les effets cumulés des niveaux inférieurs sans avoir à les échantillonner, en tenant compte de l'auto-ombrage interne ainsi que de la visibilité. Comme toutes nos intégrations sont analytiques, nous pouvons produire des images de qualité (en particulier avec peu d'aliassage) et ce, rapidement. D'un point de vue théorique, nous aimerions améliorer les approximations faites. Il serait, par exemple, intéressant de lever l'hypothèse d'un albédo faible en substituant une loi polynomiale à l'approximation linéaire.

Les paramètres des *shaders* nous permettent de simuler différents types de conifères (pins, sapins, etc.) et de moduler les caractéristiques d'un arbre (par exemple pour simuler l'effet du vent). Nous avons été capables d'intégrer ces *shaders* parce les objets de notre étude sont très structurés. Réciproquement, l'usage massif de cette connaissance a priori, fait que ces trois *shaders* ne peuvent simuler que des branches composées d'aiguilles. Dans la nature, de nombreux objets sont composés d'une unique sorte de structure, ou présentent des similarités, il est donc possible d'intégrer analytiquement leur *shader*. La prochaine étape pour nous serait de simuler d'autres types d'arbres, pour lesquels la structure est plus stochastique (concernant la distribution et l'orientation des feuilles). Il serait aussi intéressant de gérer des structures plus grandes en nombres de primitives, comme un ensemble de rameaux, voire un arbre entier.

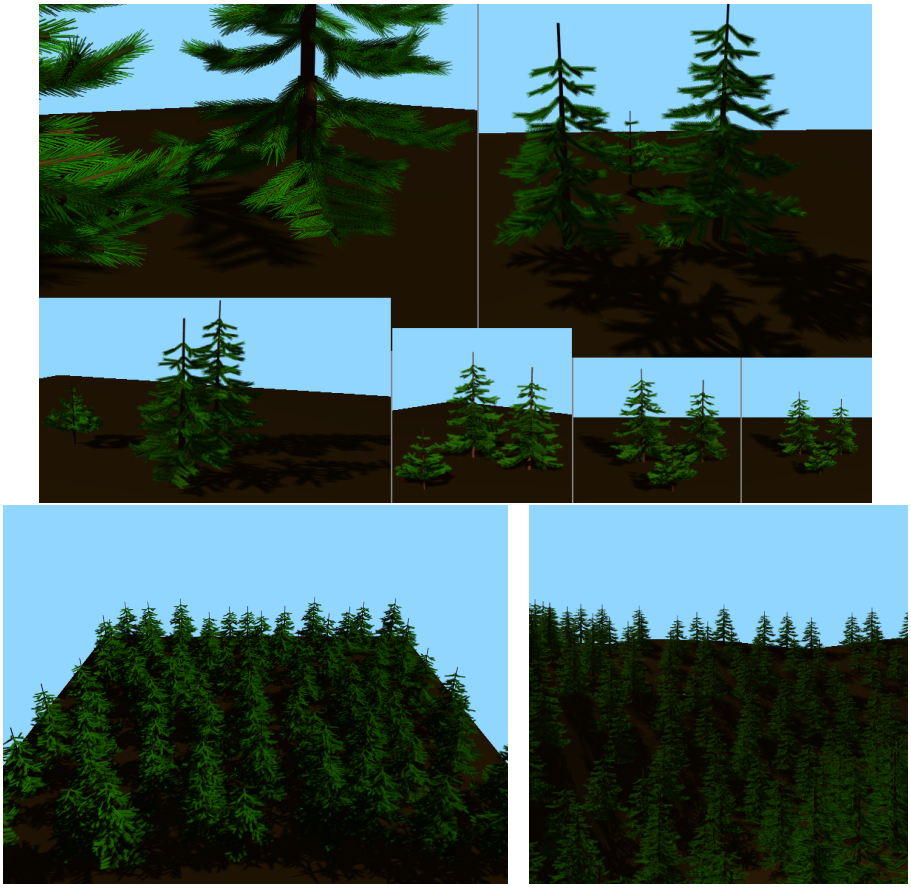

FIG. 4.14 – *En haut* : trois arbres, depuis un point de vue proche jusqu'à un point de vue lointain. *À gauche* : 100 arbres sur un carré. *À droite* : une scène de 1000 arbres.

Troisième partie

Modèle hiérarchique à base d'images pour le rendu interactif de forêts avec illumination et ombrages [MNP01]

CHAPITRE 5

Modèle à base d'images avec illumination et ombrage

Le rendu des nombreux arbres d'un paysage est particulièrement coûteux. Le nombre important de détails visibles, la complexité de l'illumination (micro-ombrage d'une branche sur une autre, illumination du ciel, etc.) et de la situation en terme de visibilité (les bloqueurs sont très nombreux et de petites tailles) font que bien souvent le modèle simpliste de *billboard* est utilisé pour des applications temps réel (*cf.* chapitre 2 section 2.3.1). Malheureusement ce modèle ne permet pas un réalisme géométrique et photométrique poussé.

Les deux aspects entrant en compte dans le calcul de la couleur d'un objet sont l'illumination ("réaction" de l'objet à la lumière), et l'ombrage (quantité de lumière arrivant sur l'objet sans être bloquée) qui correspond à un problème de visibilité. La nouvelle représentation que nous introduisons dans ce chapitre est inspirée par l'idée des *billboards* à laquelle nous associons une *fonction bidirectionnelle de texture* (*cf.* chapitre 2 section 1.1.7) afin de gérer l'illumination et d'améliorer l'effet 3D. Pour les problèmes de visibilité liés à l'ombrage, nous introduisons une structure de données inspirée des cartes d'horizon (*cf.* chapitre 1 section 2.2.3) et des cellules de visibilité (*cf.* chapitre 1 section 2.2.1). Notre but est d'obtenir une impression visuelle plus réaliste, avec une gestion de l'illumination et de l'ombrage dans des temps de calcul permettant l'interactivité, y compris pour la prise en compte du changement d'éclairage.

Ce chapitre est découpé en quatre parties : en section 1, nous traiterons de notre représentation à base de *billboards* et de *BTF* , nous détaillerons à la section 2 notre structure de visibilité pour l'ombrage, puis nous proposerons en 3 une technique pour traiter les ombres au sol et nous finirons en 4 par les résultats.

1 Billboard et fonction bidirectionnelle de texture

La nouvelle représentation que nous introduisons dans cette section est dérivée de l'association d'une fonction bidirectionnelle de texture (*Bidirectional Texture Function*, ou *BTF*) avec une représentation basée sur l'image inspirée des *billboards*. Nous voulons améliorer le concept de *billboard* en remédiant à l'absence de relief et de parallaxe qui le caractérise et en introduisant la prise en compte de l'illumination.

Nous présentons en 1.1 la technique de construction d'une *BTF*, puis nous verrons en 1.2 comment utiliser ces données au moment du rendu.

1.1 Construction d'une BTF

1.1.1 Principe

Le principe d'une fonction bidirectionnelle de texture (*BTF*) est d'associer une image de l'objet à chaque couple direction de vue plus direction de lumière, tout comme une *BRDF* associe une couleur à un couple de directions de vue et de lumière (*cf.* chapitre 2 section 1.1.6).

Pour le calcul, nous considérons n directions de vue, et pour chacune de ces directions de vue nous considérons m directions de lumière (*cf.* figure 5.1). À chacun de ces couples (directions de vue, directions de lumière), nous associons une image de l'objet, composée de la couleur (*RGB*) et de l'opacité (*alpha*).

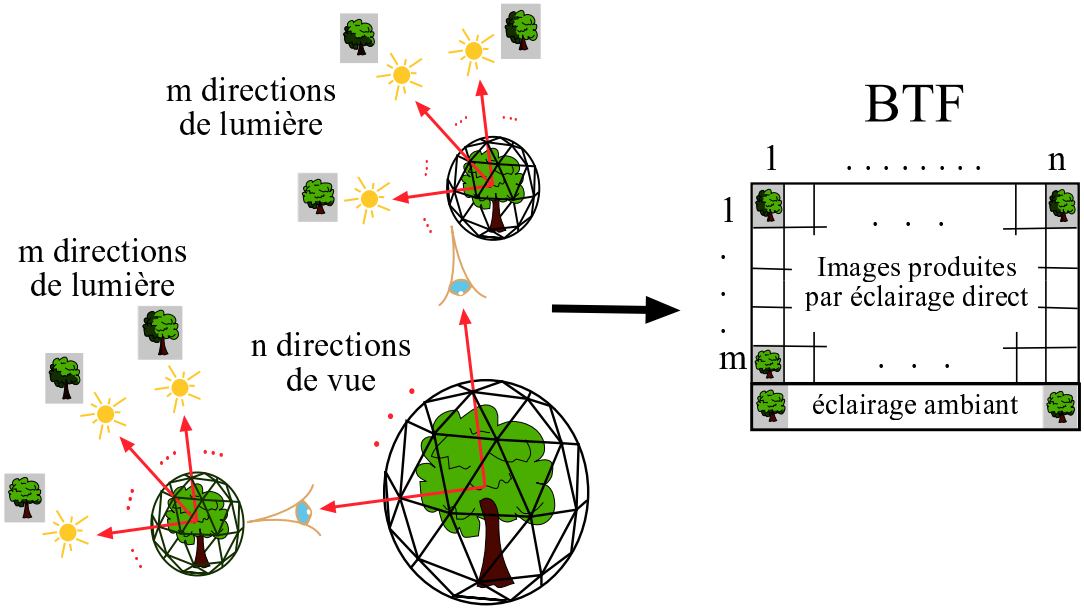

FIG. 5.1 – Pour construire une fonction bidirectionnelle de texture (*BTF*), on considère n directions de vue, autour de l'objet. Pour chacune de ces directions de vue on considère m directions de lumière. On calcule ainsi $n \times m$ images contenant l'objet éclairé par une lumière directe (*i.e.* éclairage diffuse et spéculaire) et n images contenant l'objet éclairé par une lumière ambiante.

Si nous nous limitons à des objets dont les reflets ne varient pas brusquement, un petit nombre de directions de lumière suffit. Nous destinons ces images à remplacer la géométrie 3D d'un objet apparaissant petit à l'écran, ce qui nous autorise à stocker des images de petites résolutions (*e.g.* 32×32 ou 64×64), et ainsi limiter l'occupation mémoire (*cf.* figure 5.2). Il y a un compromis

à choisir entre qualité et coût mémoire : plus la résolution de nos images est grande (donnant des objets précis) et moins on dispose de mémoire pour calculer des directions de vue différentes.

Afin de disposer d'un véritable espace vectoriel permettant de représenter des conditions d'éclairage variées, il faut séparer les fonctions de bases. Nous choisissons de distinguer une base pour l'éclairage direct et une pour l'éclairage ambiant, conformément à l'usage en synthèse d'images. On spécifie l'éclairage par un ensemble de sources directes et une source ambiante, l'intensité et la couleur de chacune étant librement choisies. Ceci nous conduit à calculer deux séries d'images : une série de n images contenant l'objet éclairé par une lumière ambiante (*i.e.* non directionnelle) et une série de $n \times m$ images contenant l'objet éclairé par une lumière directe (*i.e.* diffuse et spéculaire) (*cf.* figure 5.1). La séparation de l'ambiant augmente légèrement la place mémoire occupée (quelques Ko), mais elle permet d'effectuer des rendus où l'ambiant joue un rôle important. Par exemple si la majorité de l'éclairage provient du ciel car le soleil est opacifié par les nuages ou est déjà couché (*e.g.* effet de ciel orangé)), ou encore des rendus de scènes comportant plusieurs sources de lumière. À noter que cette séparation de l'ambiant et de la lumière directe est optionnelle dans notre implémentation : l'utilisateur choisit si ce léger surcoût mémoire est nécessaire ou non à son application.

FIG. 5.2 – Un exemple de *BTF* : un pin avec 18 directions de vue et 6 directions de lumière. Les images sont de taille 64×64 ce qui donne une occupation mémoire de $(64 \times 64 \times 4) \times 18 \times 6 = 1.7\text{Mo}$, compatible avec la mémoire texture d'une station (32 Mo ou 64 Mo sur une *GeForce*).

Grâce à la *BTF*, l'illumination et l'auto-ombrage sont encodés directement dans l'image : aucun autre calcul est nécessaire lors de la phase de rendu, ce qui permet d'obtenir l'interactivité comme nous le verrons plus loin. La construction d'une *BTF* en pré-calcul est effectuée par un moteur de rendu donnant les meilleurs résultats visuels possibles en terme de réalisme (*i.e.* illumination et ombrage), ce qui augmente d'autant l'aspect réaliste du rendu final.

1.1.2 Discrétisation de la sphère

Nous choisissons les directions de vue et de lumière autour de la sphère de manière régulière. Pour cela nous utilisons un schéma de discrétisation récursif basé sur une subdivision des triangles équilatéraux. Nous partons d'une sphère discrétisée régulièrement par six points formant huit triangles équilatéraux (*cf.* figure 5.3). Ensuite, nous divisons chacun de ces triangles en quatre nouveaux triangles en prenant le milieu de chaque segment, que nous reprojetons sur la sphère. Nous obtenons la discrétisation suivante en nombre de sommets et donc en nombre de directions de vue ou de lumière : 6, 18, 66, 257, etc. Ce schéma à l'avantage d'être simple et régulier.

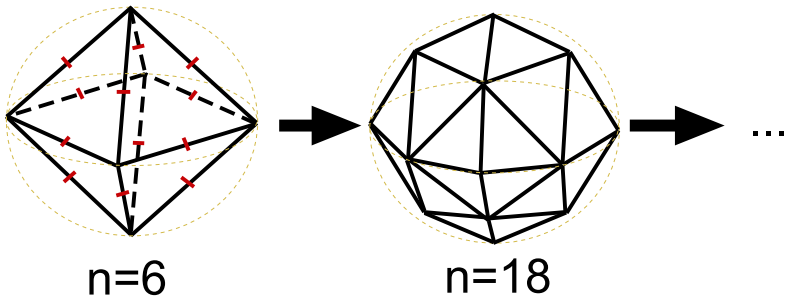

FIG. 5.3 – La discrétisation de la sphère que nous utilisons pour choisir les directions de vue et de lumière lors de la construction d'une *BTF*. La sphère initiale comporte six points formant huit triangles équilatéraux. La subdivision consiste à former, à partir de chaque triangle, quatre nouveaux triangles équilatéraux en prenant le milieu de chaque segment, que nous reprojetons sur la sphère.

Pour la construction des directions de vue et de lumière de nos *BTF* il est possible d'utiliser n'importe quels autres schémas de discrétisation. En particulier nous pourrions regrouper l'échantillonnage des directions de vue et des directions de lumière et ainsi effectuer l'échantillonnage dans un espace 4D (en représentant les directions par leurs coordonnées polaires, nous aurions 2D pour les directions de vue et 2D pour les directions de lumière). Cette technique est classiquement utilisée pour la construction de *BRDF*.

1.2 Rendu

Nous venons de décrire la structure et la construction d'une *BTF*, nous présentons maintenant la méthode de rendu de notre représentation, ainsi que les améliorations apportées. Notre objectif étant le temps réel pour le rendu d'un grand nombre d'objets, l'idée est d'étendre la technique des *billboards* en se servant au maximum des informations contenues dans une *BTF*.

Nous décrirons en 1.2.1 l'idée de base de notre rendu, suivie en 1.2.2 par les deux schémas de composition d'images possibles pour former un *billboard*, et nous finirons en 1.2.3 par proposer une technique simple augmentant les performances de rendu.

1.2.1 Principe

Lors du rendu, la direction de l'œil et la direction de la source de lumière nous permettent d'extraire les images les plus proches stockées dans la *BTF*. Notre représentation est fondée sur la combinaison de ces images pré-calculées dans le but de former un *billboard*.

Le premier problème est de trouver les directions échantillonnées stockées dans la *BTF* les plus proches des directions de vue et de lumière. Nous tabulons dans des cartes pré-calculées (*cf.* figure 5.5) le numéro des trois échantillons les plus proches d'une direction donnée. Dans ces tables, nous pré-calculons aussi les pondérations associées à ces trois directions. Nous extrayons les trois directions de vue les plus proches stockées dans la *BTF* et, de même, les trois directions de lumière[1]. Ce qui nous donne un total de 12 images (*cf.* figure 5.4), chaque direction de vue donne quatre images : une image représentant l'objet éclairé à la lumière ambiante et trois images représentant l'objet éclairé par la lumière directe depuis les trois directions de lumière. Si le traitement séparé de l'ambiant n'est pas activé nous obtenons neuf images.

[1]Dans le cas où, dans la *BTF*, le nombre de directions de vue est différent du nombre de directions de lumière nous pré-calculons deux tables.

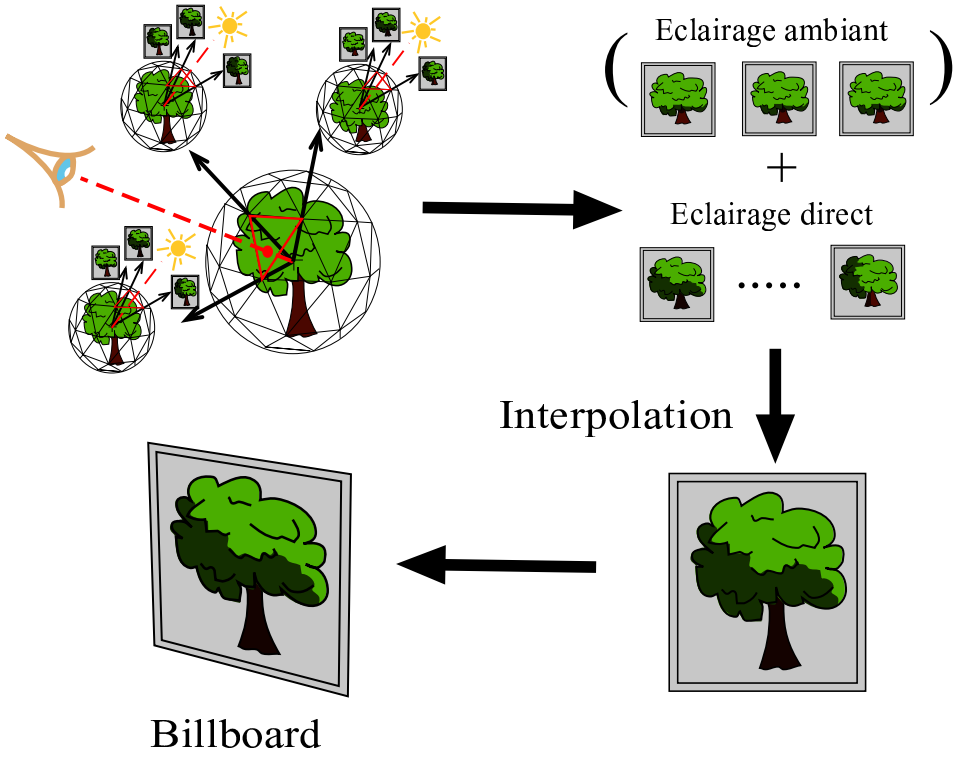

FIG. 5.4 – A partir de la direction de vue et de la direction de lumière on extrait les images de la *BTF*. Ces images sont interpolées de manière pondérée afin d'obtenir une représentation proche du *billboard* avec *OpenGL*.

En nous plaçant dans un espace 4D pour l'échantillonnage des directions de vue et de lumière, comme nous l'avons suggéré à la section précédente, nous n'aurions plus neuf images pour les directions de vue et de lumière mais cinq images, plus les trois images pour l'éclairage ambiant (si il est activée), soit un total de huit images.

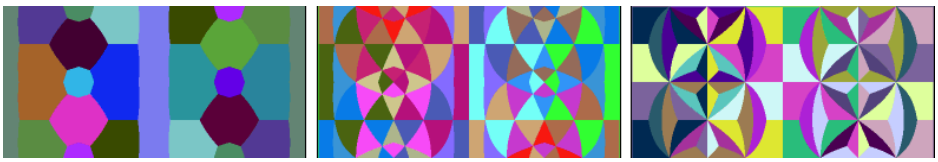

FIG. 5.5 – Nous utilisons une table pré-calculée pour déterminer les trois directions les plus proches. Les deux axes correspondent aux coordonnées polaires de la direction et les trois composantes *RGB* aux trois numéros des directions les plus proches (le coefficient de pondération n'est pas visible sur ces images).

1.2.2 Deux techniques pour les billboards

Nous disposons d'images de l'objet et de leur pondération. Il nous reste à composer ces images le plus adéquatement possible. L'idée générale est d'interpoler ces images pour les utiliser comme *billboard*, mais deux questions se posent alors :

– Faut-il plaquer ces images dans le plan perpendiculaire à la direction de l'œil, ou faut-il utiliser leur orientation d'origine, *i.e.* les trois plans perpendiculaires aux directions de vue (*cf.* figure 5.6) ? On peut rapprocher ces deux modes de rendu aux deux techniques

de *billboard* existantes : les *billboards* plats, toujours orientés vers l'œil, et les *billboards* croisés formés de trois plans perpendiculaires (*cf.* chapitre 2 section 2.3.1).

– Comment moyenner de manière pondérée plusieurs images en utilisant le matériel graphique ? Ceci n'est pas une simple composition des transparences (*alpha blending*) comme on pourrait le penser au premier abord. Il faut donc trouver un moyen d'utiliser astucieusement le matériel pour résoudre ce problème (*cf.* annexe A), d'autant plus qu'au moment de ces travaux le *multi-texturing* n'existait pas et le matériel *SGI* utilisé ne permettait pas de traiter plusieurs images simultanément.

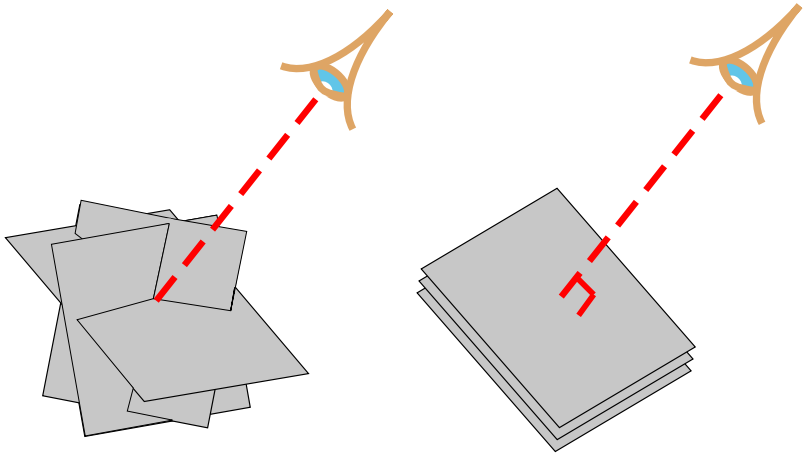

FIG. 5.6 – Pour nos *billboards* deux possibilités de rendu sont possibles. *À gauche* : les images sont perpendiculaires à la direction de vue encodée dans la *BTF*. *À droite* : les images sont toutes dans le plan perpendiculaire à la direction de l'œil.

Nous avons testé ces deux techniques, *i.e.* avec un plan unique ou avec trois plans. Notre critère étant l'apparence visuelle, nos tests ont montré que le choix de l'une ou l'autre technique dépend du nombre de directions de vue pré-calculées dans la *BTF*. Lorsque ce nombre de vues est très grand (257 ou plus), la technique utilisant trois plans tend vers la technique utilisant le plan perpendiculaire à l'œil. En effet, plus on a de directions encodées dans la *BTF*, plus ces directions sont proches les unes des autres, et donc, plus les plans perpendiculaires correspondants sont proches. Dans ce cas là, les deux techniques engendrent des résultats similaires. Lorsque le nombre de directions de vue encodées dans la *BTF* est de six, il vaut mieux utiliser trois plans perpendiculaires et ainsi obtenir un *billboard* croisé qui offre un meilleur effet de perspective et de volume qu'un *billboard* plat (*cf.* chapitre 2 figure 2.10). Pour des nombres de directions de vues intermédiaires (18 ou 66), il vaut mieux utiliser un plan unique dont la continuité est plus douce.

Que nous utilisions l'une ou l'autre de ces techniques, nous devons effectuer, lors du rendu, une combinaison pondérée des images : $\sum_i \mathtt{Poids}_i \times \mathtt{Image}_i$. Pour obtenir un temps de rendu proche du temps réel il est impossible d'effectuer ces calculs par logiciel et la composition des transparences effectuée par le matériel graphique ne donne pas les résultats attendus. Nous proposons en annexe A une solution utilisant le matériel graphique, adaptée de la composition des transparences. Pour une compréhension du coût de rendu nous dirons succinctement que cette technique fonctionne en séparant le traitement de la composante *alpha*, des composantes de couleur *RGB*. Pour la mise en place de la composante *alpha* nous effectuons trois passes (une pour chaque direction de vue) sans mettre à jour la couleur, puis nous réalisons les passes pour les composantes *RGB* sans mettre à jour la composante *alpha*. Cette technique ajoute donc trois passes de rendu, ce qui entraîne un total de 15 passes par objet (12 passes quand la séparation de l'ambiant

n'est pas activée). Pour le cas d'un échantillonnage dans un espace 4D la mise à jour de l'*alpha* coûterait cinq passes, ce qui donnerait un rendu en 13 passes (10 passes quand la séparation de l'ambiant n'est pas activée).

Aux vues des performances actuelles des cartes graphiques, 15 passes par objet permet un rendu temps réel de bon nombre d'objets. Si ce nombre de passes peut paraître élevé, il faut se rappeler que ces textures servent à représenter un arbre ou une branche qui sont des milliers de fois plus complexes en nombre de polygones. De plus, ces passes peuvent être factorisées en utilisant les capacités de multi-textures dont disposent les nouvelles générations de cartes graphiques. Avec le *multi-texturing*, les passes préliminaires de mise à jour de la composante *alpha* ne sont plus utiles : avec quatre textures simultanées[2], on peut factoriser les images de l'objet sous les différentes directions de lumière en une passe, et obtenir ainsi un rendu en trois passes au total (soit des performances cinq fois supérieures à ce que nous obtenons actuellement sans cela).

1.2.3 Optimisation

Le rendu d'un *billboard* demande plusieurs passes de rendu par objet (entre 8 et 15). Nous avons vu que, lors du rendu, chaque image est pondérée. L'optimisation proposée ici consiste à ne tenir compte que des images dont le poids est supérieur à un seuil paramétrable par l'utilisateur. Si une image n'est pas affichée, les poids des autres images sont normalisés. Nous verrons dans la partie résultat que cette simple technique permet de diminuer de moitié en moyenne le nombre de passes, tout en gardant un aspect visuel acceptable. En revanche, un seuil trop élevé introduit un effet de clignotement (*poping*) immédiatement visible par un observateur averti.

2 Cube de visibilité (VCM)

Nous venons de voir que l'utilisation d'une *BTF* associée à une adaptation des *billboards* permet d'afficher un objet, par exemple un arbre, en temps réel avec une illumination et un auto-ombrage correct, même avec une source de lumière en mouvement. À ce stade, il nous manque encore l'information concernant l'ombrage d'un objet sur un autre.

Le matériel graphique n'est pas prévu pour calculer les ombres, et une scène de forêt est trop complexe pour pouvoir évaluer celles-ci à la volée par lancer de rayons. Une carte d'ombrage globale (*cf.* chapitre 1 section 2.2.2) demanderait une résolution trop importante, et ne pourrait gérer les objets partiellement transparents qu'autorise notre représentation. La solution que nous proposons ici passe par un pré-calcul de l'information de visibilité entre les objets et la lumière.

Bien que nous pré-calculions l'information, nous aimerions pouvoir changer la direction de la lumière interactivement. Cependant, il parait difficile de traiter cet ombrage au niveau de chaque pixel du *billboard*, puisque celui-ci est traité comme un polygone par la carte graphique, laquelle ne permet pas de calcul très élaboré au niveau du pixel[3].

Nous introduirons à la section 2.1 le principe des cubes de visibilité, puis nous expliquerons à la section 2.2 comment utiliser un tel cube de visibilité avec les *billboards* que nous venons de voir, et nous finirons à la section 4 par un court aperçu des possibilités et des problèmes de ces techniques.

[2] Ce qui est supporté par une *GeForce3*.

[3] La fonctionnalité permettant un calcul par pixel des couleurs n'est apparue que très récemment parmi les cartes graphiques, et ne permet pas encore d'effectuer n'importe quel traitement. Sous *OpenGL* 1.1, que l'on trouve notamment sur les *SGI* utilisées pendant ma thèse, l'éclairage est évalué aux sommets des faces, et la couleur résultante est interpolée à l'intérieur.

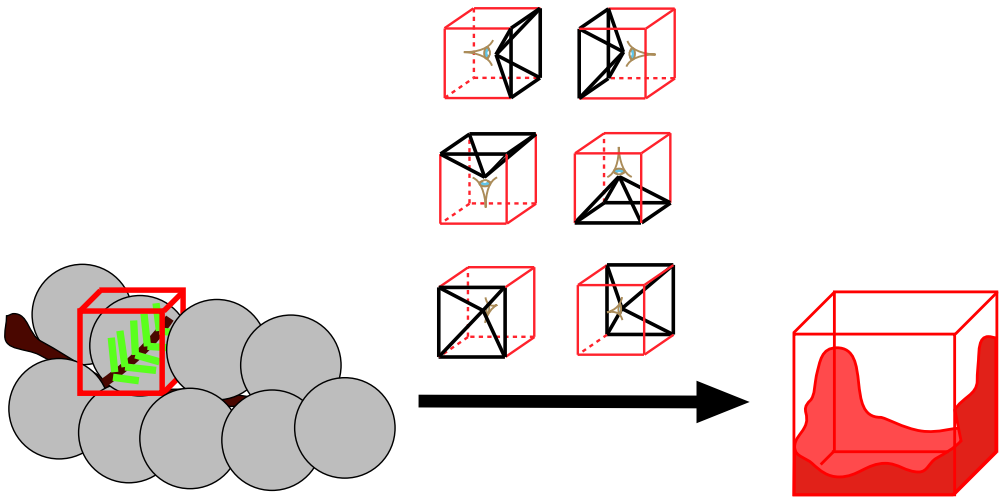

FIG. 5.7 – Pour construire un cube de visibilité (*VCM*), nous effectuons le rendu de toute la scène sur ses six faces en ne considérant que l'information d'opacité (*alpha*). Ces six images contiennent alors la visibilité depuis ce point dans toutes les directions. Ce pré-calcul permettra de savoir instantanément si la lumière d'une source d'une direction donnée peut atteindre le point central.

2.1 Principe et construction

Le but des cubes de visibilité (*Visibility Cube Maps, ou VCM*) est d'offrir une structure de données pré-calculée permettant de traiter l'ombrage d'une scène et d'augmenter ainsi le réalisme de nos *billboards*. Le principe est de stocker la visibilité d'un point donné dans toutes les directions en utilisant six cartes de visibilité formant un cube. L'idée est inspirée des cartes d'horizons[4] (*cf.* chapitre 1 section 2.2.3) et des cellules de visibilité (*cf.* chapitre 1 section 2.2.1).

Pour construire un *VCM*, nous effectuons le rendu de la scène complète sur chacune des six faces du cube, la caméra étant au centre du cube, et en ne considérant que les valeurs d'opacité (*cf.* figure 5.7). Ce rendu est effectué par le matériel graphique avec l'algorithme de rendu vu à la section précédente puis, par logiciel, on extrait la composante alpha de l'image. Un *VCM* peut être vu comme le dual d'une carte d'ombre (*cf.* chapitre 1 section 2.2.2), qui encode la visibilité de toute la scène depuis un point unique : la source de lumière. Pour chaque *VCM* on calcule la valeur d'opacité moyenne, permettant de pondérer l'illumination ambiante (*i.e.* éclairage dû au ciel). Elle est obtenue en moyennant toutes les valeurs d'opacité des faces.

Grâce aux six cartes de visibilité, nous disposons pour chaque direction de l'accessibilité de la lumière vers ce point. L'accessibilité (équivalente à une transparence) de la lumière dans une direction considérée vaut 1 moins la valeur d'occlusion. Chaque pixel des six cartes de visibilité contient une valeur d'occlusion comprise entre 0 et 1 : 0 quand la source de lumière est complètement occultée dans cette direction, 1 quand elle est complètement visible.

[4] Excepté que Max [Max88] utilise des données 1D alors que nous avons besoin de donnée 2D.

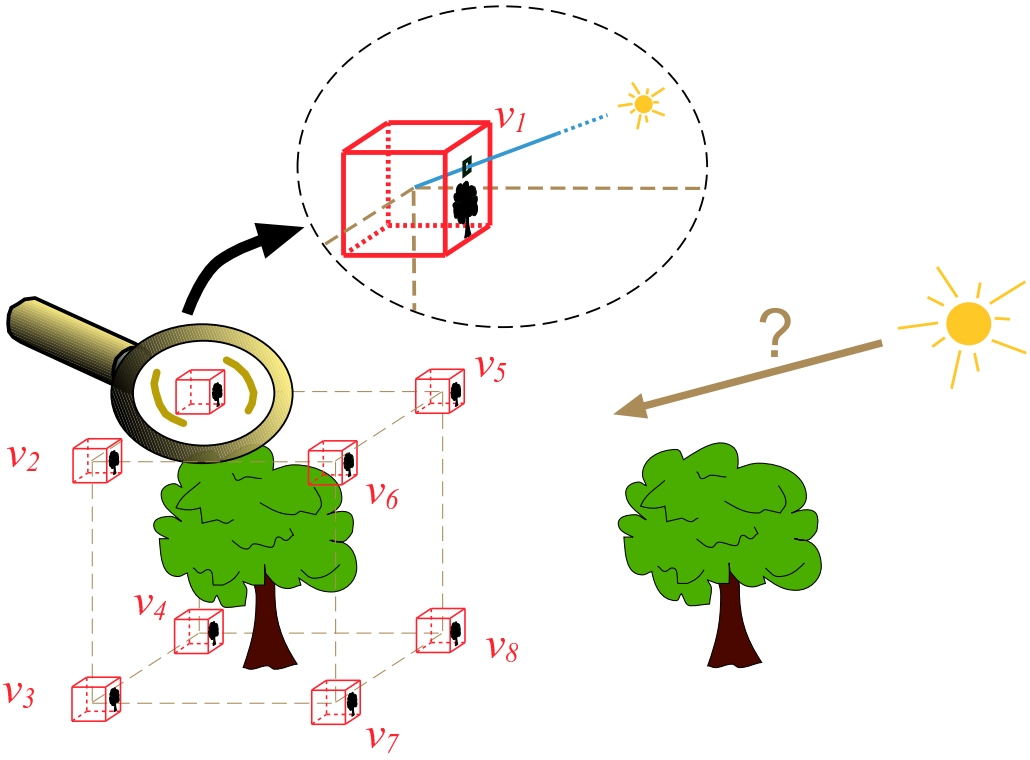

FIG. 5.8 – On place n cubes de visibilité autour de l'objet. Dans notre implémentation, nous en plaçons en général huit aux sommets de la boîte englobante.

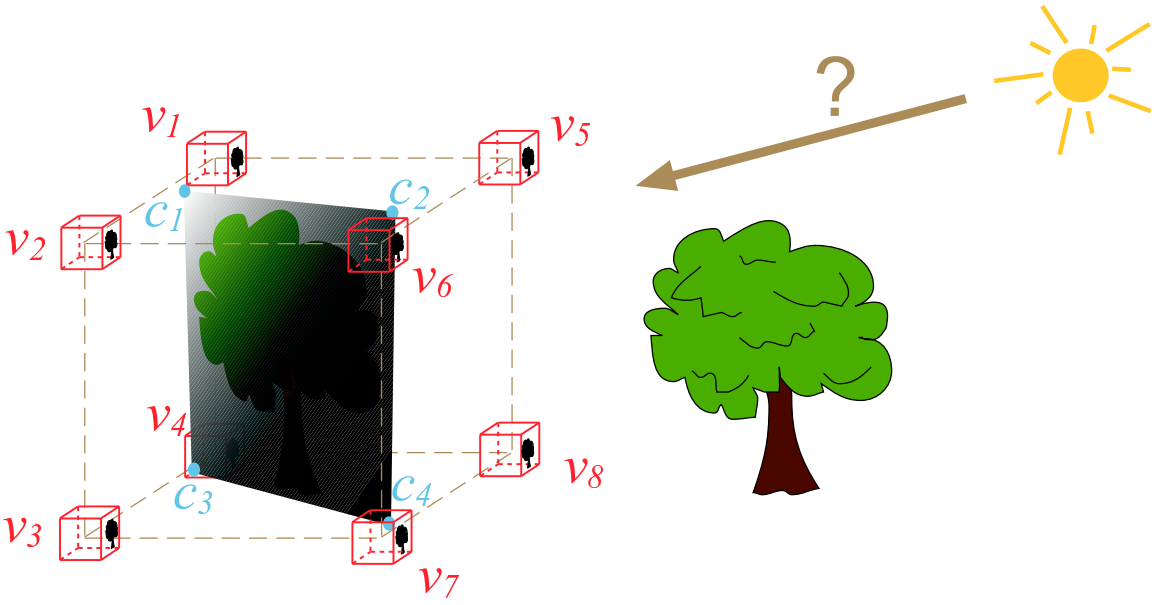

FIG. 5.9 – Soit une direction de lumière. Avec les huit valeurs de visibilité données par les cubes de visibilité qui se trouvent autour de l'objet, nous calculons par interpolation les quatre coefficients d'assombrissement aux sommets du *billboard*. Ces quatre coefficients sont utilisés par *OpenGL* comme couleurs aux sommets du polygone, le matériel graphique se chargeant de l'interpolation lors du remplissage du polygone. Cette couleur multipliera en chaque pixel le dessin du *billboard*.

2.2 Cube de visibilité et rendu de billboard

Un *VCM* fournit l'information de visibilité d'un point dans toutes les directions. Cependant, celle-ci varie d'un point à un autre. Pour représenter la visibilité d'un objet complet, nous échantillonnons le "champs de *VCM*" dans le volume de l'objet[5]. Au moment du rendu, la visibilité en un point précis du volume de l'objet est évaluée par interpolation entre les valeurs données par les *VCM* aux noeuds voisins. Comme l'objet est représenté par un *billboard*, nous calculons une valeur d'ombrage aux quatre sommets de celui-ci (par logiciel), puis nous confions au matériel graphique le soin de les interpoler lors de la *rasterisation.* Le matériel multiplie, en chaque pixel, ces valeurs d'ombres avec la couleur de la texture pour obtenir (assombrir) la couleur finale du polygone (*cf.* figure 5.9). Par la suite, lorsque nous ferons référence à une valeur de visibilité associée à une direction, il s'agira de la valeur calculée par interpolation entre les valeurs de visibilité des *VCM* aux nœuds comme expliqué dans cette section.

3 Ombres au sol

Pour obtenir l'information de visibilité d'un terrain, Max [Max88] et Stewart [Ste98] échantillonnent la surface et y place des cartes d'horizons (*cf.* chapitre 1 section 2.2.3). Avec le même esprit, nous pourrions placer des *VCM* sur la surface échantillonnée, mais cette solution serait très coûteuse et peu précise, d'autant plus qu'au sol, on s'attend à bien discerner le contour des ombres portées. En effet, pour échantillonner suffisamment finement une surface et ainsi obtenir des ombres détaillées, il faudrait des millions de *VCM* , avec ce que cela implique en temps de précalcul et en mémoire. La solution que nous proposons est une adaptation des techniques classiques de cartes d'ombre (*cf.* chapitre 1 section 2.2.2).

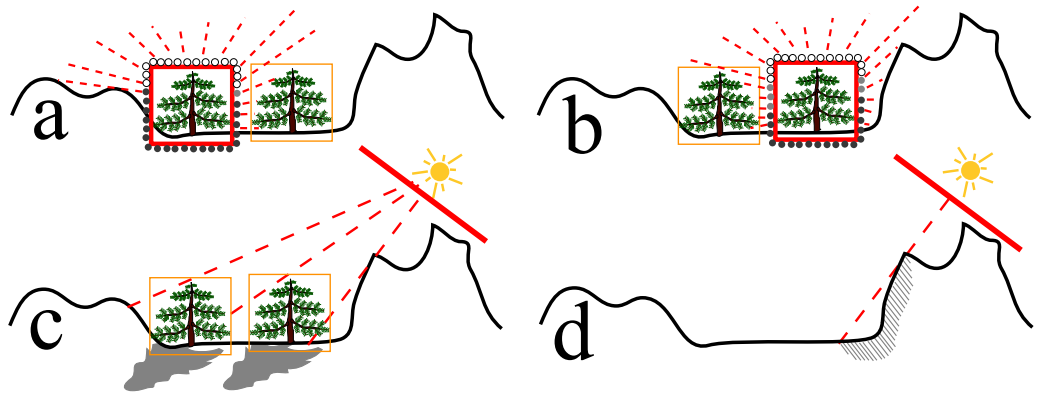

FIG. 5.10 – Les quatre cas de figure d'interaction entre les arbres et le terrain. *a,b* : Les cubes de visibilité de chaque arbre prennent en compte la présence de la montagne et des autres arbres (l'auto-ombrage étant inclus dans les *BTF* et par les *VCM* pour les niveaux inférieurs de la hiérarchie). *c* : L'*alpha shadow map* nous donne les ombres douces des arbres sur le sol. *d* : La *Z shadow map* (ou *depth map*) nous donne l'auto-ombrage de la montagne.

Nous utilisons deux sortes de *shadow map* combinées, dont les avantages et les inconvénients sont complémentaires (*cf.* chapitre 1 section 2.2.2) : une *alpha shadow map* qui ne gère pas l'auto-ombrage, est utilisée pour les objets semi-transparents se projetant sur le terrain (*e.g.* les arbres),

[5]Dans notre implémentation nous plaçons généralement huit cubes de visibilité aux huit sommets de la boîte englobante de l'objet (*cf.* figure 5.8). Si ce choix offre un bon compromis entre coût mémoire et qualité, il n'est pas une contrainte du modèle.

et une *Z shadow map* qui ne gère pas les objets semi-transparents, pour l'auto-ombrage du terrain (*cf.* figure 5.10 en bas). Ces *shadow maps* sont construites en effectuant un rendu (simplifié) de la scène depuis le point de vue de la source de lumière[6].

Pour l'*alpha shadow map*, seuls les arbres sont dessinés, en ne considérant que les valeurs de transparence (*alpha*) des textures, utilisées comme niveaux de gris (*cf.* figure 5.11). Le terrain est dessiné "en invisible" *i.e.* en blanc : il ne doit pas produire d'ombre, mais il doit quand même cacher les arbres qui sont de l'autre côté. Pour la *Z shadow map* qui prend en compte les auto-ombrages du terrain (*e.g.* l'ombre d'une montagne sur une vallée), une carte de profondeur est utilisée (*depth map*[7]). À noter qu'à la place d'une *Z shadow map* n'importe quelle technique de calcul des ombres de terrains peut être utilisée (*cf.* chapitre 1 section 2.2.3).

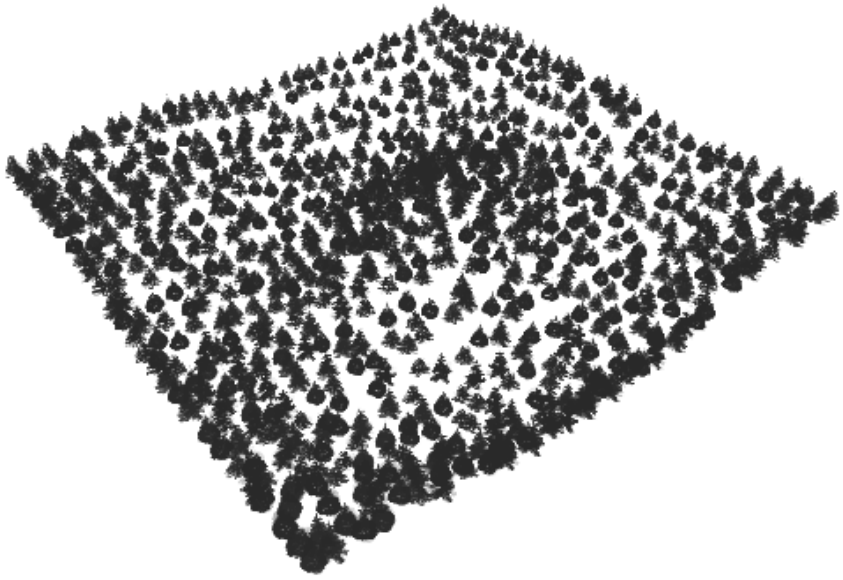

FIG. 5.11 – Un exemple d'*alpha shadow map* prise depuis la source de lumière où seule l'opacité des arbres est représentée. Le sol est tracé en blanc pour cacher les objets qui sont éventuellement derrière lui et pour ne pas générer d'ombre. Cette texture est ensuite utilisée comme texture d'ombre plaquée sur le terrain, et est recalculée à chaque changement de position de la lumière.

4 Résultats et conclusions

Cette extension des *billboards* par l'utilisation de *BTF* et des cubes de visibilité nous permet d'afficher en temps interactif (5 à 20 images par seconde sur une Onyx^2 Infinite Reality, en utilisant un algorithme de suppression des objets hors du champ de la caméra) des scènes complexes comportant des centaines d'arbres (*cf.* figure 5.13) avec illumination, auto-ombrage (inclus dans les images) et ombrage (grâce aux cubes de visibilité). Cette structure de données autorise des objets semi-transparents et, pour cette occasion, l'antialiassage (*i.e.* de petits détails dans un pixel

[6]Malheureusement la construction dynamique de ces deux *shadow map* prend un temps non négligeable, même en utilisant le matériel. Quand la direction de la lumière change nous devons mettre à jour ces cartes et le taux de rafraîchissement s'en trouve diminué.

[7]L'ombrage par *depth map* doit être supporté par le matériel graphique, ce qui est le cas de l'*Infinite Reality* de *SGI* que nous avons utilisée, mais aussi de la carte *Geforce 3* de *Nvidia*.

correspondent à une fraction d'opacité).

Cependant, la faible résolution des images utilisées (en général 64×64) et l'interpolation d'images introduisent un effet de flou qui diminue le réalisme en masquant les détails pour les objets au premier plan (*cf.* figure 5.13 à droite). Un arbre suffisamment éloigné de la caméra ne souffre pas de ces artefacts. La représentation de l'objet doit avoir une taille à l'écran inférieure à la résolution des images de la *BTF* : si un pixel de la texture (de la *BTF*) recouvre plusieurs pixels de l'image on observera un manque de finesse dans les détails. Dans le cas contraire, l'image paraîtra détaillée (*cf.* chapitre 3 section 3.2). Comment adapter cette représentation pour traiter un arbre proche de la caméra ? Augmenter la résolution des images n'est pas raisonnable, car le coût en mémoire deviendrait vite rédhibitoire. Cette représentation fonctionne bien pour des objets apparaissant petits à l'écran.

Notre solution consiste à décomposer un arbre en sous-parties : une branche peut être représentée par un de nos *billboard* . Un arbre entier et réaliste, même pour des vues proches, peut être affiché par un ensemble de branches de ce type. Nous introduisons ainsi des niveaux de détails, mais ceci revient à augmenter l'ordre de grandeur du nombre de primitives présentes. Calculer et stocker un cube de visibilité différent pour chaque instance d'objet de la scène devient alors excessivement coûteux, étant donné le grand nombre de branches composant une forêt. Nous avons donc mis au point une structure de données hiérarchique basée sur l'instanciation qui fait l'objet du chapitre suivant.

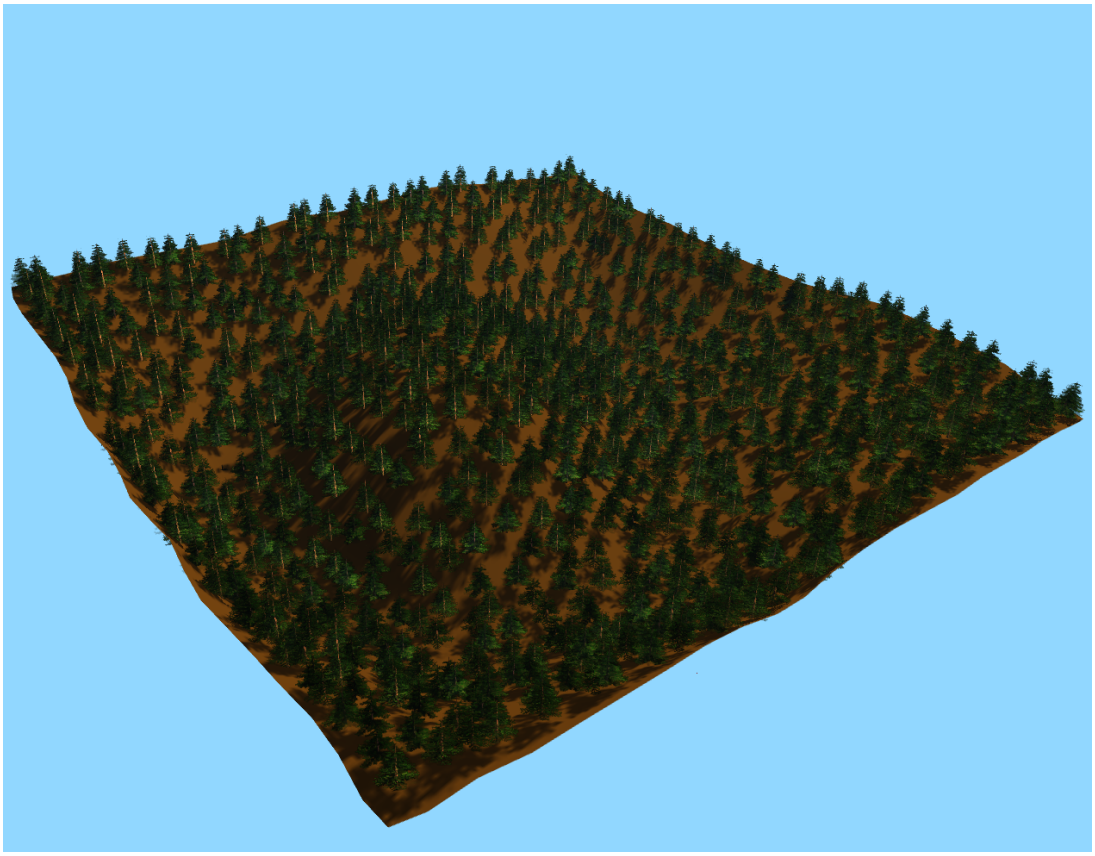

FIG. 5.12 – Une vue globale de la scène.

FIG. 5.13 – une scène de 1000 arbres rendue et ombrée en temps interactif grâce à nos extensions des *billboards* et des cubes de visibilité. *À droite* : notre extension des *billboards* mélange plusieurs vues d'objet ce qui donne un effet de flou quand l'objet est rapproché.

CHAPITRE 6

Hiérarchie de BTF

Au chapitre précédent nous avons exposé une nouvelle représentation permettant de traiter efficacement des objets (*e.g.* un arbre ou une branche) ayant une taille réduite à l'écran (moins de 64×64 pixels) avec illumination et ombrage. En exploitant les propriétés pertinentes des arbres, nous proposons dans ce chapitre de construire une mouture hiérarchie de ces représentations. La complexité de l'apparence des arbres tient essentiellement à la redondance de leurs éléments et à la structure naturellement récursive de leur construction. Les arbres sont composés d'un tronc, de branches principales, elles-mêmes composées de branches secondaires. Les rameaux sont constitués de feuilles ou d'aiguilles. D'autre part, les feuilles, les aiguilles, les branches et les arbres se ressemblent entre eux (*cf.* Étude de cas). Cette propriété permet des simplifications de modélisation : les éléments d'une famille peuvent être représentés par des instances d'un modèle générique, et un élément complexe par un ensemble d'objets similaires, plus simples (*cf.* figure 6.3 colonne de gauche).

Notre but est d'obtenir l'interactivité dans des scènes de forêts, même avec des arbres proches, tout en gérant l'illumination, l'auto-ombrage et l'ombrage. Pour atteindre ce niveau de qualité, nous introduisons une hiérarchie de textures bidirectionnelles : un arbre est encodé en utilisant plusieurs niveaux de détails calqués sur la hiérarchie naturelle évoquée précédemment. Ceci permet un rendu adapté à la taille de l'objet à l'écran, et donc au nombre de détails explicitement visibles. Nous utilisons la même organisation hiérarchique pour le calcul de la visibilité vis à vis de la source de lumière. En combinant les occlusions de cet élément à tous les niveaux de la hiérarchie nous pouvons recalculer en temps interactif les ombrages au fur et à mesure que la lumière bouge.

Nous décrirons à la section 1 le principe de notre hiérarchie, nous ferons à la section 2 une synthèse de la construction des données, nous verrons à la section 3 comment s'effectue le rendu, et nous finirons par les résultats et la conclusion en 4 et 5.

1 Hiérarchie

Nous profitons de la structure fortement hiérarchique des arbres pour construire les niveaux de détails, composés par des *billboards* associés à une fonction bidirectionnelle de texture (*BTF*), par des *VCM* et, éventuellement, par de la géométrie (*e.g.* pour les troncs). La ressemblance entre deux éléments d'arbre nous permet d'utiliser l'instanciation et ainsi de gagner en mémoire. Nous verrons en 1.1 la hiérarchie de *BTF*, en 1.2 la hiérarchie de *VCM* associée et nous finirons en 1.3 par une synthèse expliquant nos niveaux de détails.

1.1 Hiérachie de BTF

Notre construction d'arbre est hiérarchique, nous considérons qu'un arbre est constitué de branches principales, puis qu'une branche principale est constituée de branches secondaires (*cf.* figure 6.3 colonne de gauche). Les branches principales sont construites à partir d'instances de la branche secondaire plus le tronc. Sur le même principe, les arbres sont élaborés à partir d'instances des branches principales (et éventuellement d'instances de branches secondaires lorsque celles-ci ne font pas partie d'une branche principale) et du tronc. Nous profitons de des possibilités d'instanciation que nous offrent les arbres pour ne stocker qu'un nombre limité de formes de branches secondaires et de formes de branches principales.

Cette hiérarchie est conçue pour y calquer les *BTF* et profiter des instanciations pour ne pas surcharger la mémoire. Nous avons trois niveaux de *BTF* entièrement calqués sur cette hiérarchie : un niveau de *BTF* représentant la branche secondaire, un niveau représentant les branches principales et un niveau représentant l'arbre (*cf.* figure 6.1).

Dans notre implémentation, nous utilisons des *L-systems* (*cf.* chapitre 1 section 1.2.1) pour la construction des arbres. Nous avons fait ce choix à cause de la simplicité de l'implémentation, mais n'importe quelle technique fournissant l'information de hiérarchie est utilisable.

Pour l'ombrage il nous faut maintenant associer des *VCM* à cette hiérarchie, ce que nous détaillerons à la section suivante.

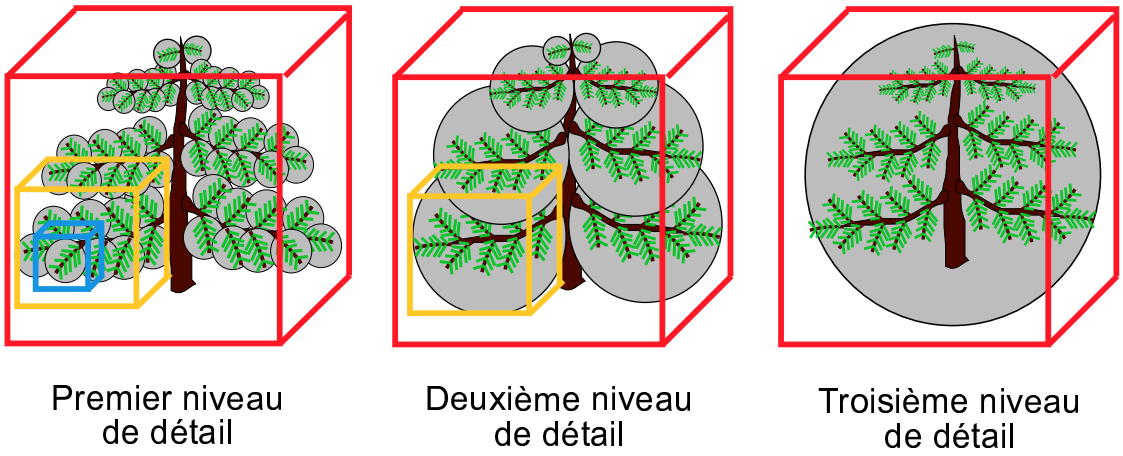

FIG. 6.1 – Nos trois niveaux de détails sont calqués sur la hiérarchie naturelle des arbres. Les objets entourés d'un cercle représentent une instance de *BTF*. Les cubes représentent la hiérarchie de *VCM* : en rouge le *VCM* de l'arbre, en jaune le *VCM* d'une branche principale et en bleu le *VCM* d'une branche secondaire.

1.2 Hiérarchie de VCM

Pour traiter les ombres, chaque instance de branches doit être munie à priori d'une structure de visibilité. Cependant, associer un cube de visibilité à chacune serait extrêmement coûteux : une forêt de 1000 arbres formés de 10 branches principales, chacune constituée de 15 branches secondaires demanderait $1000 \times 10 \times 15 = 150000$ *VCM*, ce qui n'est acceptable ni pour le temps de pré-calcul, ni pour le stockage. Nous introduisons une hiérarchie de *VCM* s'appuyant sur celle des arbres que nous avons vue à la section précédente. Grâce à cette possibilité d'instanciation, le concept de *VCM* devient utilisable à une échelle plus grande.

Pour un niveau de hiérarchie donnée, les *VCM* ne vont contenir que l'information de visibilité vis à vis des autres objets du même niveau. Au niveau le plus haut (l'arbre pour notre implémentation) un *VCM* (en rouge sur la figure 6.1) est stocké pour chaque instance d'objet et contient l'information de visibilité entre l'arbre et le reste de la scène (*i.e.* les autres arbres et le terrain). Pour les niveaux inférieurs, un *VCM* contient l'information de visibilité vis à vis des autres objets de son niveau (et non par rapport à toute la scène). Prenons l'exemple d'une branche principale : elle est composée de branches secondaires dont les *VCM* (en bleu sur la figure 6.1) stockent l'information de visibilité entre elles. Autrement dit le *VCM* d'une branche secondaire ne contient que l'information de visibilité relative à ses congénères, qui se trouvent dans la même branche principale qu'elle.

Avec ce principe, un objet est entièrement "autonome" vis à vis de son auto-ombrage : les *VCM* des sous-objets dont il est composé fournissent l'information d'ombrage interne. Ceci permet d'instancier entièrement cet objet : ses *BTF* tout comme ses *VCM*. L'utilisation massive de l'instanciation est alors possible, donc l'occupation mémoire reste raisonnable : quelques dizaines de Mo pour une scène de 1000 arbres (*cf.* section 4). Si nécessaire, il serait possible d'ajouter des niveaux supplémentaires à la hiérarchie, correspondant par exemple à un groupe d'arbres, ce qui diminuerait d'autant le coût mémoire (mais contraindrait la disposition et l'orientation des arbres).

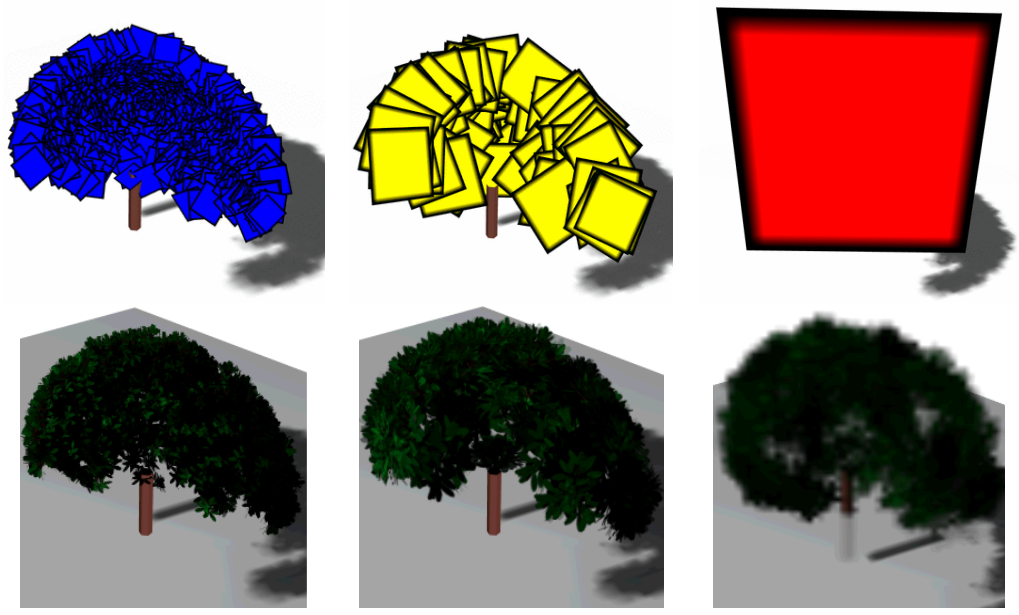

FIG. 6.2 – Les trois niveaux de détails. *À gauche* : les *billboards* (en bleu) représentent les branches secondaires. *Au milieu* : les *billboards* (en jaune) représentent les branches principales. *À droite* : le *billboard* (en rouge) représente tout l'arbre.

1.3 Niveaux de détails

Les trois niveaux de notre hiérarchie constituent aussi les niveaux de détails pour le rendu (*cf.* figures 6.1 et 6.2). Pour ce qui suit, on suppose que nous avons un unique type d'arbre, un unique type de branche principale et un unique type de branche secondaire.
Le premier niveau de détails (le plus précis) est composé :

- des instances de la *BTF* de la branche secondaire,
- des instances des *VCM* donnant la visibilité entre les branches secondaires d'une même branche principale,
- des instances des *VCM* donnant la visibilité entre les branches principales de l'arbre,
- du *VCM* donnant la visibilité entre l'arbre et le reste de la scène (*i.e.* les autres arbres et le terrain).

Le deuxième niveau de détails (intermédiaire) est composé :

- des instances de la *BTF* de la branche principale,
- des instances des *VCM* donnant la visibilité entre les branches principales d'un même arbre,
- du *VCM* donnant la visibilité entre l'arbre et le reste de la scène.

Le troisième niveau de détails (le plus grossier) est composé :

- d'une instance de la *BTF* d'un arbre complet,
- du *VCM* donnant la visibilité entre l'arbre et le reste de la scène.

Nous verrons à la section 3.2 comment les valeurs de visibilité des différents niveaux de hiérarchie sont combinées lors du rendu. L'ordre et la manière de construire les données sont très importants pour la compréhension de cette hiérarchie, nous allons donc les détailler dans la section suivante.

2 Constructions des données

Nous détaillons maintenant le parcourt à suivre lors de la construction des données d'une scène complète. Nos scènes sont principalement composées du terrain, et de nombreuses instances d'un ou plusieurs modèles d'arbres. Pour fabriquer les données, nous avons besoin d'encoder dans notre représentation un modèle d'arbre existant, de manière aussi transparente que possible pour l'utilisateur. Deux structures hiérarchiques imbriquées sont à construire : les *BTF* encodant l'apparence de chaque arbre et branche, et les *VCM* encodant la visibilité d'un objet dans toutes les directions (*i.e.* visibilité avec la lumière). Cette construction s'effectue récursivement et est illustrée par la figure 6.3.

Nous partons d'une représentation géométrique d'une branche secondaire, dont nous calculons la *BTF* par lancer de rayons[1]. Le lancer de rayons offre de nombreuses fonctionnalités comme l'auto-ombrage, l'anti-aliassage (en utilisant par exemple le lancer de cônes déjà utilisé au chapitre 3), la transparence, et l'utilisation de modèles d'illumination évolués de manière à obtenir des images les plus réalistes possibles. Les éléments de la hiérarchie sont composés de *BTF* et d'objets géométriques quelconques (*e.g.* des polygones pour le tronc), la seule contrainte étant que leur rendu soit rapide pour ne pas pénaliser le rendu de toute la scène.

Les *BTF* des branches secondaires sont instanciées pour construire une branche principale, puis nous calculons leurs *VCM* associés pour l'ombrage. Avec ceci, les *BTF* des branches prin-

[1] Les rendus pour les autres niveaux et le rendu final se font avec notre moteur de rendu accéléré par le matériel graphique.

cipales sont calculées par notre moteur de rendu[2]. Pour le calcul de la *BTF* de l'arbre complet, on instancie les *BTF* des branches principales et des branches secondaires auxquelles on associe des *VCM*. Cette *BTF* d'un arbre complet est ensuite instanciée pour former la scène, pour finir le *VCM* associé à chaque arbre donnant la visibilité entre l'arbre et le reste de la scène est calculé. À chaque étape de la construction, nos objets disposent de l'illumination, l'auto-ombrage et l'ombrage, notre rendu est donc complet et permet de construire des *BTF* réalistes.

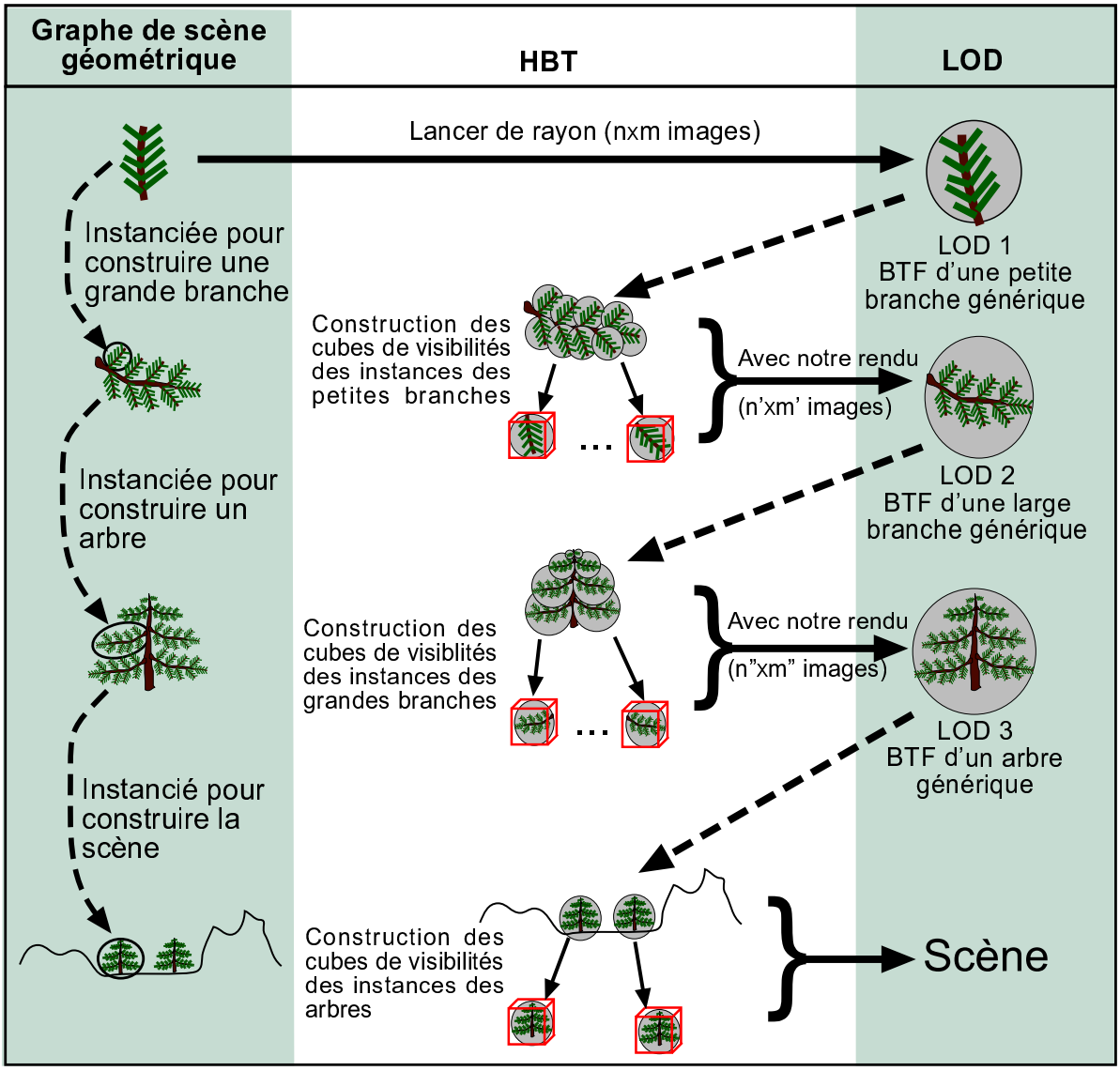

FIG. 6.3 – Schéma global de construction de la hiérarchie.

[2]Notre implémentation utilise *OpenGL* et l'accélération matérielle des cartes graphiques (Onyx^2 Infinite Reality dans notre cas).

3 Rendu

Nous venons de décrire la structure de notre représentation hiérarchique ainsi que sa construction. Cette partie est consacrée à la description de la phase de rendu. Nous traiterons en 3.1 la question du choix du niveau de détails, puis nous présenterons en 3.2 le calcul de l'ombrage d'un objet de la hiérarchie, et nous finirons en 3.3 par notre algorithme de rendu en indiquant les parties accélérées par le matériel graphique.

3.1 Choix du niveau de détails

Lors du rendu il faut déterminer le "bon" niveau de détails, celui qui sera le moins coûteux, tout en réduisant l'aliassage au maximum, et en limitant l'effet de clignotement (*poping*) lors des transitions.

En utilisant des textures, le critère que nous avons présenté à la section 3.2 du chapitre 3 se traduit par : la résolution de l'image utilisée comme texture doit être, une fois projetée, plus fine que la résolution de l'écran. Un polygone de taille l, se trouvant à distance z de la caméra aura une taille de $\frac{l}{z}N.V$ dans le repère écran, avec N la normale du polygone et V le vecteur allant de l'œil vers l'objet. Soit r la résolution de la texture se trouvant plaquée sur le polygone[3], la taille d'un pixel de texture $t_{texture}$ dans le repère écran vaut $t_{texture} = \frac{l}{z.r}N.V$. Les polygones de nos *billboards* sont quasiment toujours parallèles à l'écran, nous supposons donc que $N.V \approx 1$. Notre critère indique que la taille d'un pixel de texture doit être inférieure à la taille d'un pixel écran, ce qui se traduit par $t_{texture} = \frac{l}{z.r} < 1$.

Pour déterminer si un niveau de détails donné est valide il faut considérer les valeurs de l et r de ses *billboards* dont le rapport $G = l/r$ est le plus grand, car ceux sont eux qui sont susceptibles de ne pas respecter le critère. Si $\frac{G_{max}}{z} < 1$ est vérifié alors, ce niveau de détails est valide pour la distance z.
Remarque : cette formule nous donne les plages de distances pour lesquelles le niveau de détails est valide.

3.2 Calcul de l'ombrage dans la hiérarchie

Nous avons vu à la section 2.2 du chapitre 5, que pour habiller d'ombres un *billboard*, nous calculions quatre coefficients d'assombrissement pour les quatre sommets du polygone sous-jacent. Nous obtenons chacune de ces quatre valeurs par interpolation tri-linéaire en fonction des valeurs de visibilité données par les *VCM* placés autour de l'objet. Avec la version hiérarchique de notre représentation il nous faut tenir compte aussi des *VCM* placés dans tous les niveaux supérieurs de la hiérarchie. Avec les trois niveaux de détails que nous utilisons pour les arbres, l'ombrage se calcule ainsi (*cf.* figure 6.4) :

- L'ombrage d'un arbre rendu avec le premier niveau de détails (*i.e.* le *billboard* représente l'arbre complet) est similaire à ce que nous avons vu à la section 2.2 du chapitre 5. L'ombrage d de l'arbre par rapport à la scène est calculé par logiciel aux quatre sommets du polygone, et sera multiplié avec l'auto-ombrage a inclus dans l'image de la *BTF* par le matériel graphique au moment de la rasterisation.
- L'ombrage d'une branche principale d'un arbre rendu avec le deuxième niveau de détails est calculé par logiciel aux quatre sommets du polygone. Chacune de ces quatre valeurs est calculée par $c \times d$, où c est la valeur de visibilité du sommet par rapport aux autres branches

[3]Dans notre implémentation la résolution des images des *BTF* est 32×32 ou 64×64.

principales de l'arbre (donnée par les *VCM* jaunes), et d est la valeur de visibilité du sommet par rapport à la scène (donnée par les *VCM* rouges).

– L'ombrage d'une branche secondaire d'un arbre rendu avec le troisième niveau de détails est calculé par logiciel aux quatre sommets du polygone. Chacune de ces valeurs est calculée par $b \times c \times d$, où b est la valeur de visibilité du sommet par rapport aux autres branches secondaires de sa branche principale (donnée par les *VCM* bleus), c est la valeur de visibilité du sommet par rapport aux branches principales autres que celle à qui elle appartient (donnée par les *VCM* jaunes), et d est la valeur de visibilité du sommet par rapport à la scène (donnée par les *VCM* rouges).

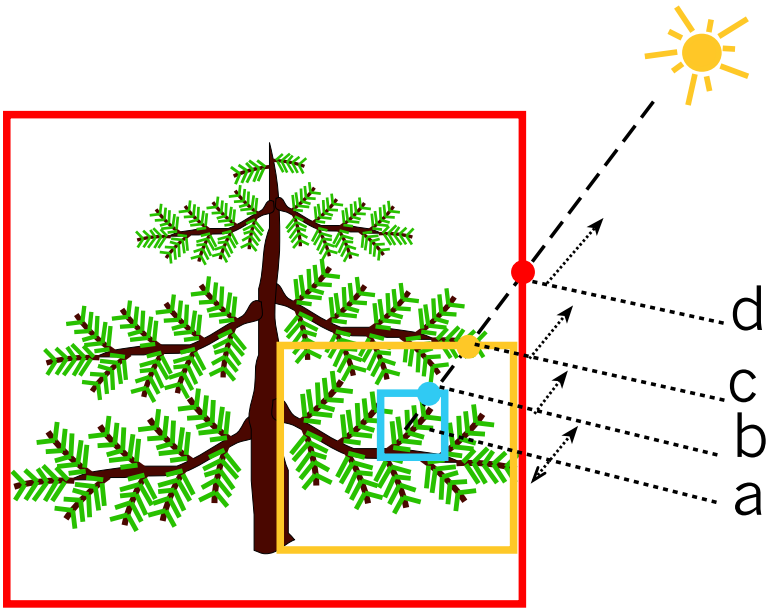

FIG. 6.4 – L'ombrage dans la hiérarchie de visibilité est donné par la multiplication de : a l'auto-ombrage inclus dans les images de la *BTF*, b l'ombrage des branches secondaires entre elles, c l'ombrage des branches principales entre elles, d l'ombrage de l'arbre par rapport à la scène.

En résumer, le statut des ombres suit le schéma suivant :

– l'auto-ombrage a est encodé dans les images de la *BTF* (*cf.* figure 6.4).
– les *VCM* d'un objet contiennent l'information de blocage de la lumière avec les autres objets du même niveau de hiérarchie.
– les cubes de visibilité au plus haut niveau de la hiérarchie encodent l'information de blocage entre leur objet associé (*i.e.* un arbre) et le reste de la scène (*i.e.* les autres arbres et le terrain).

3.3 Algorithme de rendu

Notre algorithme de rendu est le suivant (*cf.* figure 6.1) : nous n'affichons que les arbres qui se trouvent dans la pyramide de vue, nous les trions de l'arrière vers l'avant (pour gérer correctement la transparence), puis nous choisissons le niveau de détails de chaque arbre en fonction de sa distance à l'œil. Nous traitons tous les éléments géométriques éventuellement existants (*e.g.* tronc) puis, toutes les *BTF* (en les représentant par des *billboards*), en déterminant les coefficients d'ombrage à l'aide de la hiérarchie de visibilité.

Algorithme 6.1 Algorithme de rendu

Afficher_terrain(*depthshadowmap, alphashadowmap*)
Affiche le terrain avec ses ombres (matériel).

Arbres_pyramide ← ***Selectionne_Arbres_Visibles***(Arbres)
Sélectionne les arbres se trouvant dans la pyramide de vue (logiciel).

Arbres_triés ← ***Trie_Arbres***(Arbres_pyramide)
Trie les arbres visible de l'arrière vers l'avant (logiciel).

pour *A* ∈ *Arbres_triés* **faire**

Alod ← ***Determine_LOD***(A)

Affiche_ElementGéométrique(*Alod*)
*Affiche les éventuelles éléments géométrique (*e.g. *tronc) du niveau de détails,*
i.e. *ceux qui ne sont pas des BTF (matériel).*

pour *BTF* ∈ *Alod* **faire**

VCM ← ***Donne_VCM***(*BTF, Alod, A*)
Détermine les VCM de la hiérarchie nécessaire au calcul des ombres qui suit.

Coeff_Ombre ← ***Calcul_Ombre***(*BTF, VCM*)
Calcule les coefficients d'ombre du billboard en multipliant
les différentes valeurs d'ombrage données par la hiérarchie de VCM (logiciel).

pour *Image* ∈ *BTF* **faire**

Pimage ← ***Donne_Pondération***(*Image, Table*)
Détermine la pondération de l'image grâce à une table pré-calculée (logiciel).

Affiche_Billboard(*Image, Pimage, Coeff_Ombre*)
OpenGL *traite le billboard (matériel).*

fin pour

fin pour

fin pour

FIG. 6.5 – *De gauche à droite et de haut en bas* : un pin avec ombres, sans ombres, valeur de l'ombrage. Le pin avec seulement l'ambiant, coefficient de visibilité ambiante (*i.e.* près du tronc moins de lumière arrive du ciel), les *billboards* utilisés au niveau de détail le plus fin.

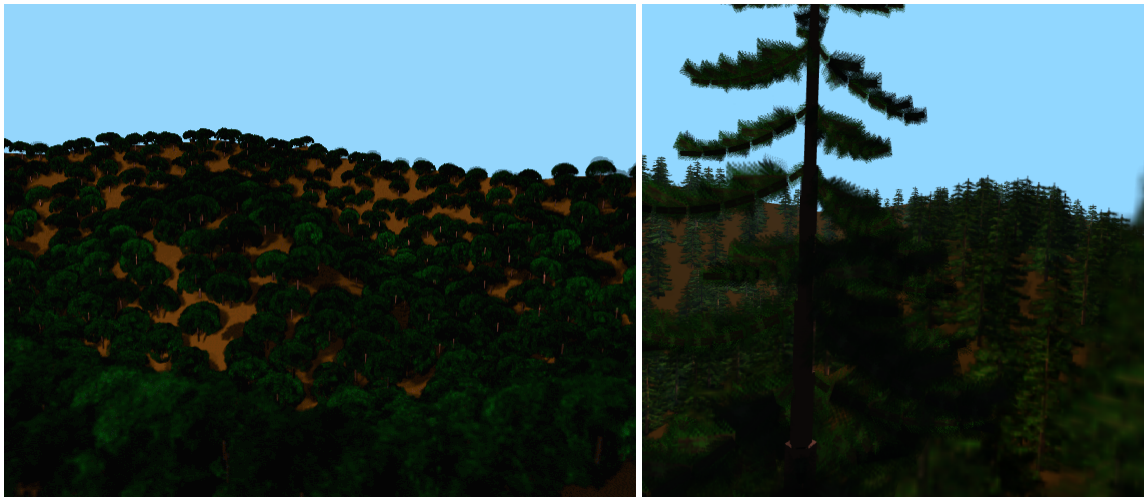

FIG. 6.6 – *cf.* figure 6.7

4 Résultats

Dans notre implémentation, nous utilisons des images de tailles 32×32 ou 64×64 en *RGBA*. Pour l'observateur et pour la lumière, notre discrétisation de la sphère est de 6 ou 18 échantillons, ce qui offre un bon compromis entre coût mémoire et qualité visuelle. Ces *BTF* coûtent en mémoire texture lors du rendu $32 \times 32 \times 4 \times 6 \times 6$ (144 Kb), $32 \times 32 \times 4 \times 18 \times 6$ (432 Kb), ou $32 \times 32 \times 4 \times 18 \times 18$ (1.3 Mb), plus la partie ambiante $32 \times 32 \times 4 \times 6$ (24 Kb) ou $32 \times 32 \times 4 \times 18$ (72 Kb).

Nos modèles d'arbres ont été générés par *L-system*. Nous avons utilisé deux sortes d'épineux et une sorte de feuillus pour nos tests, chacun représenté par trois niveaux de la hiérarchie décrite à la section 1 : les branches secondaires comportent des feuilles ou des aiguilles, les branches principales sont constituées à partir d'instances de branches secondaires, et les arbres sont constitués d'instances de branches principales et de branches secondaires. Les sapins comptent environ 30 branches principales, 300 branches secondaires et 40000 aiguilles. Un arbre suit le même graphe de scène que l'arbre géométrique (ce qui ne pose aucun problème grâce au *L-system*).

Pour les *VCM*, nous avons calculé une résolution de 32×32 soit 6 Ko par *VCM*. La scène de test (*cf.* figure 6.7) contient environ 1000 arbres, et grâce à l'instanciation $8 \times (1000 + 30 + 10)$ *VCM* seulement : 1000 pour les arbres, 30 pour les branches principales formant l'arbre de référence et 10 pour les branches secondaires formant la branche principale de référence. La structure de visibilité coûte donc $8 \times 6 \times 1000$ Ko = 48 Mo pour une scène comportant 1000 instances d'un arbre (elle est stockée en mémoire centrale sans jamais être chargée sur la carte graphique). Le temps de pré-calcul de toutes les données de la scène est d'environ 75 minutes en utilisant une Onyx^2 Infinite Reality, dont 2/3 pour la visibilité et 1/3 pour la hiérarchie de *BTF*.

Durant le rendu, nous tenons compte du soleil comme source directionnelle de lumière, et de l'illumination du ciel comme source ambiante (*i.e.* une source de lumière plus l'ambiant). Dans le pire des cas, notre algorithme nécessite l'affichage de quinze polygones texturés par objet : trois pour traiter la composante *alpha*, neuf pour l'illumination directe et trois pour l'illumination ambiante (*i.e.* le ciel). L'optimisation que nous avons énoncée à la section 1.2.3 du chapitre 5 pour afficher le *billboard* permet de ne considérer que cinq images en moyenne au lieu de neuf, ce qui multiplie quasiment par deux le taux de rafraîchissement.

La carte graphique *Infinite Reality* que nous avons utilisée ne dispose pas de la fonctionnalité de *multitexturing*[4], alors que les cartes graphiques récentes permettent plusieurs textures par polygones : ce qui diviserait d'autant le coût de rendu.

Nous avons produit des animations de 640×480 (*cf.* figure 6.7), calculées sur Onyx^2 Infinite Reality, montrant un survol d'une petite forêt de 1000 arbres avec un taux de rafraîchissement de 7 à 20 images par seconde (nous avons un gain de 20% par rapport à ces chiffres en utilisant le même programme avec la carte *Nvidia GeForce 2* de *Nvidia*, *i.e.* sans profiter du *multitexturing*).

Un sapin géométrique classique est représenté par à peu près 120000 polygones. Avec le niveau de détails le plus précis nous avons un taux de rafraîchissement huit fois supérieur au rendu avec tous les polygones, avec le niveau intermédiaire un gain de 18 et avec le niveau le plus grossier nous avons un gain de 30.

[4] Le *multitexturing* permet d'appliquer plusieurs textures à un polygone en une seule passe de rendu.

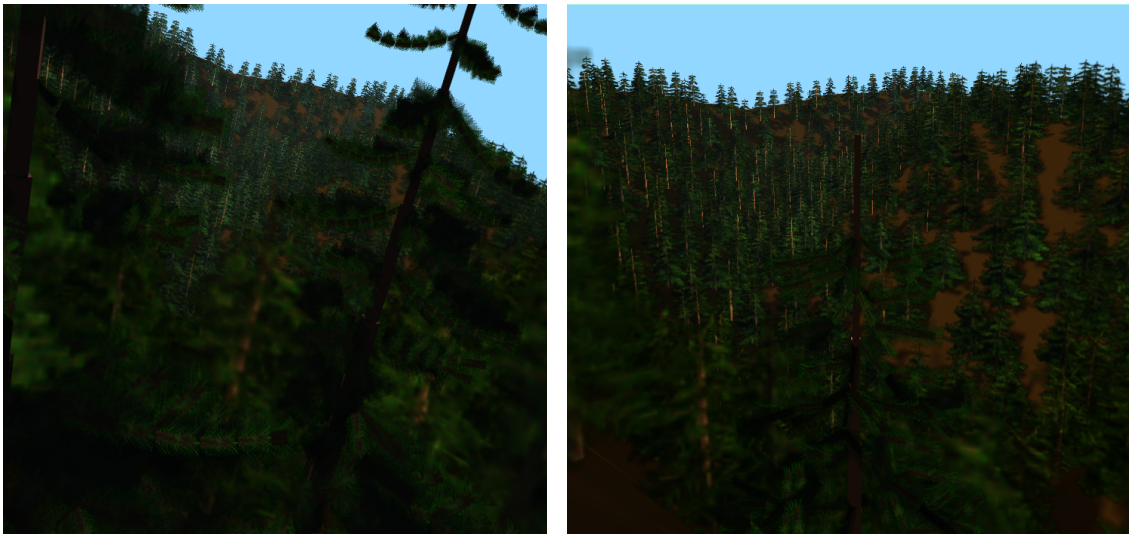

FIG. 6.7 – Quatre vues d'une forêts (1000 arbres sur un terrain avec éclairage et ombres). Le rendu s'effectue entre 7 et 20 images par seconde sur notre machine de test ($Onyx^2$ Infinite Reality). En utilisant les fonctionnalités des nouvelles cartes graphiques, le nombre d'images par seconde peut largement être amélioré. Remarquez les arbres détaillés au premier plan. Une animation est disponible à l'adresse : http ://www-imagis.imag.fr/ Alexandre.Meyer/research/MNP01/index.html

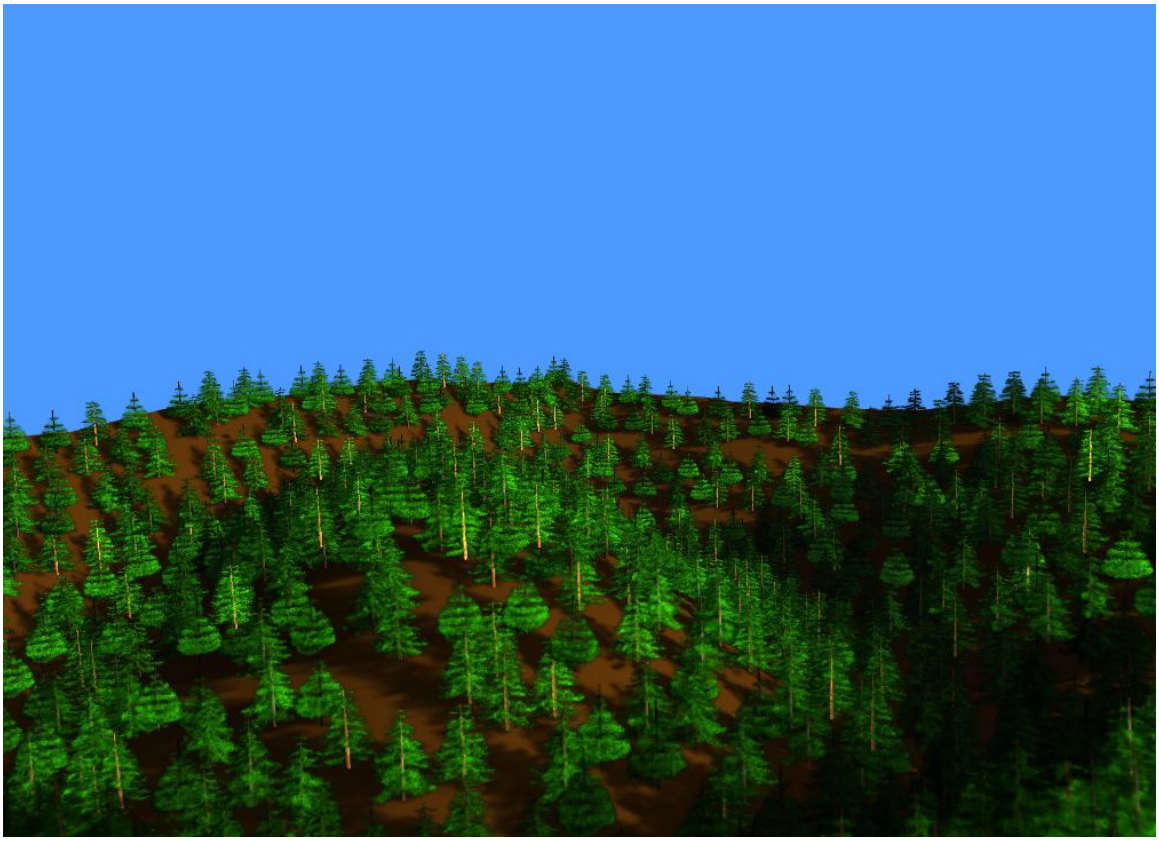

FIG. 6.8 – *"La forêt du Père Noël"* (Paul, 3 ans)

5 Conclusion et perspectives

Nous avons introduit une nouvelle représentation à base d'images, associée à une structure de visibilité, destinée au rendu d'arbres. Celle-ci donne des images de qualité avec des effets complexes comme l'illumination, l'auto-ombrage et l'ombrage, l'illumination du ciel, en tenant compte du mouvement du soleil. Notre implémentation, naïve, permet un taux de rafraîchissement de 7 à 20 images par seconde sur une Onyx^2 Infinite Reality avec une scène de 1000 arbres. L'usage des nouvelles générations de cartes devrait permettre de gagner encore un ordre de grandeur.

Notre hiérarchie est constituée de *BTF* associées à des *billboards* sur laquelle se calque une hiérarchie de *VCM* pour l'ombrage. Malgré les six degrés de liberté d'une *BTF*, la mémoire consommée pour la scène complète est seulement de quelques dizaines de mégaoctets. Notre représentation est efficace en terme de mémoire si les données se prêtent à une hiérarchisation et à l'instanciation, ce qui est le cas des arbres. Elle pourrait être appliqué sur d'autres types de données (*e.g.* ville).

Spécularités, *BRDF* et transparence sont gérées par notre méthode, mais la précision en est limitée par la densité de l'échantillonnage des directions de vue et de lumière. L'échantillonnage des directions et des positions des *VCM* ont les mêmes défauts, et l'interpolation tri-linéaire ne permet pas une reconstruction parfaite des données. Ce problème disparaît quand l'échantillonnage est très fin, au prix du coût mémoire. Il y a donc un équilibre à trouver entre qualité visuelle et quantité de mémoire utilisée. Néanmoins, les résultats montrent qu'un survol de qualité au-dessus d'un terrain peut être produit avec un coût mémoire raisonnable, et avec plusieurs types d'arbres différents.

Des améliorations à notre algorithme de rendu peuvent être apportées, comme l'introduction d'une grille pour limiter le nombre d'objets à rendre ce qui permettrait d'augmenter la taille de la scène calculée. Différents autres choix d'implémentation (par exemple avec une sphère 4D) peuvent être envisagés pour notre structure de *BTF* et de *VCM* pour la construction et le rendu.

Afin d'améliorer le réalisme, il peut être envisager d'utiliser comme point de départ de notre hiérarchie des photos réelles de branches secondaires (comme toute la hiérarchie est basée sur la première *BTF*, en améliorant le réalisme de celle-ci, on améliore le réalisme global). Des tests sur des données réelles d'arbres sont aussi à envisager : tester la réaction de notre hiérarchie avec des arbres dont la structure est plus proche du réel que nos arbres générés par *L-system*.

Il serait aussi intéressant de voir comment les nouvelles et futures fonctionnalités des cartes graphiques permettraient d'améliorer cette technique. Le *multitexturing* permettrait certainement de diminuer rapidement le nombre de passes et donc d'augmenter le nombre d'images par seconde. Le calcul d'illumination par pixel pourrait être appliqué à l'ombrage des *billboards*, en vue de le rendre plus précis. Les texture 3D font leur apparition sur les cartes de type *Geforce*, on peut imaginer des textures 4D, qui permettraient de traiter les *BTF* directement par le matériel graphique.

Conclusions et perspectives

Deux nouvelles représentations

Cette thèse, située dans le cadre de la synthèse d'images de paysages, a été consacrée plus spécifiquement à la représentation des arbres pour le rendu. Les techniques de modélisation d'arbres donnent à ce jour de bons résultats en termes de diversité d'espèces et de formes représentables, mais au prix de la génération de milliers de polygones par arbre, et par conséquent, de milliards pour une forêt. En plus du coût exorbitant de traitement, cette multitude de polygones représente des détails très fins qui, une fois projetés à l'écran, ont une taille souvent inférieure à celle d'un pixel, et posent de gros problèmes d'aliassage. Pour résoudre ces difficultés, l'approche couramment envisagée est l'utilisation de niveaux de détails. Les techniques de simplification de maillage visent à remplacer un ensemble de polygones par un polygone unique. Cependant, l'aspect disparate et non continu de la répartition des feuilles d'un arbre ne permet pas d'appliquer ces méthodes sur un arbre sans en modifier l'opacité et l'illumination globale.

L'idée directrice développée au cours de cette thèse a été de représenter un ensemble de primitives (les feuilles ou les branches) par paquet, tout comme le peintre les factorise par des taches de pinceau (*cf.* Étude de cas). Ceci conduit à des représentations conservant l'aspect visuel, *i.e.* ayant le même comportement vis à vis de la lumière que l'ensemble des primitives remplacées.

En suivant cette idée, nous avons proposé deux approches hiérarchiques. La première, destinée au rendu haute-qualité, est basée sur le calcul analytique du modèle d'illumination d'une géométrie représentant un rameau de conifère. La seconde, destinée au rendu temps réel, est basée sur des ensembles hiérarchiques d'images, pré-calculant toutes les configurations d'illumination et d'auto-ombrage.

Un modèle hiérarchique de shaders analytiques

Nous avons introduit (*cf.* partie II) une hiérarchie de trois *shaders* capables de représenter à une échelle donnée les effets cumulés des niveaux plus fins, sans avoir à les échantillonner, et en prenant en compte l'auto-ombrage et la visibilité. Nous nous sommes servis de la connaissance disponible a priori concernant la distribution géométrique des aiguilles pour calculer analytique-

ment ces caractéristiques visuelles :

- Le premier *shader* est basée sur l'intégration du modèle d'illumination (classique) de Phong sur un cylindre représentant une aiguille.
- Le deuxième *shader* correspond à l'illumination d'un cône d'aiguilles que nous avons modélisé par une distribution continue semi-transparente d'aiguilles du type précédent.
- Le troisième *shader* correspond à l'illumination d'un rameau d'aiguilles que nous avons modélisé par une série de cônes d'aiguilles empilés.

L'aspect analytique de nos *shaders* permet dans le même temps d'accélérer les calculs (notre implémentation, facilement optimisable, est environ 8 fois plus rapide que le système de lancer de rayons *Rayshade* utilisant le sur-échantillonnage pour diminuer l'aliassage) et d'obtenir des images de qualité (en particulier avec peu d'aliassage).

Un modèle hiérarchique à base d'images pour la visualisation temps réel d'arbres

Nous avons introduit (*cf.* partie III) une représentation à base d'images dédiées aux arbres, permettant un rendu de qualité avec des effets complexes comme l'illumination, l'ombrage, la prise en compte de l'illumination du ciel et le mouvement de la source de lumière. Notre implémentation, largement optimisable (notamment avec la nouvelle génération de cartes graphiques grand public), tourne de 7 à 20 images par seconde sur une Onyx^2 Infinite Reality avec une scène comptant 1000 arbres.

Notre représentation se compose d'une hiérarchie de *BTF* (une fonction qui associe une image à chaque couple direction de vue-direction de lumière), que nous affichons à l'aide de *billboards* en interpolant les images. En association à cette hiérarchie, nous construisons une structure précalculée, basée sur des *cubes de visibilité* (*VCM*) qui nous permet de traiter l'auto-ombrage et l'ombrage en temps interactif. Du fait de la hiérarchie et de l'instanciation des données, la consommation mémoire reste raisonnable : quelques dizaines de Mo en mémoire texture pour plusieurs espèces d'arbres et quelques dizaines de Mo en mémoire centrale pour les *VCM* pré-calculés. Il y a un compromis à faire entre quantité de mémoire occupée et qualité visuelle : comme les *BTF* et les *VCM* sont des structures discrètes, plus l'échantillonnage des directions est fin, et plus la qualité du rendu sera élevée, mais plus la quantité de mémoire sera importante. Néanmoins, avec quelques instances d'arbres différents et une quantité mémoire raisonnable nous avons pu produire une scène de 1000 arbres avec illumination, auto-ombrage, ombrage, et avec possibilité de déplacer interactivement la caméra et la source de lumière.

Bilan

Nous avons introduit deux nouvelles représentations traitant la complexité des arbres, en remplaçant un ensemble de primitives (*i.e.* feuilles et branches) par une représentation plus efficace. Ces deux représentations permettent un rendu d'arbres efficace (*i.e.* réaliste et sans aliassage) et rapide : l'une dans le cadre du rendu de qualité et l'autre dans le cadre du temps réel. Ces techniques améliorent fortement la complexité et la qualité visuelle présentes dans les scènes complexes de forêts (*e.g.* auparavant aucun moteur interactif de rendu ne permettait de modifier dynamiquement la lumière).

Perspectives

Augmenter le réalisme des arbres

Il serait intéressant de voir comment l'hypothèse de forte instanciation présente dans notre hiérarchie de *BTF* (nécessaire à la faible consommation mémoire) se comporte avec une grande diversité d'arbres. Il faudrait étudier comment traiter l'instanciation de branches avec des logiciels de modélisation d'arbres plus élaborés [Bio, LD] que notre implémentation des *L-systems*, ce qui permettrait de tester les capacités de notre modèle à grande échelle.

Augmenter le réalisme de nos modèles

Dans notre modèle hiérarchique à base d'images (*cf.* chapitre 6) l'apparence de toute la hiérarchie est basée sur celle de la *BTF* initiale : une *BTF* d'un niveau donné est calculée à partir des *BTF* des niveaux inférieurs. Pour améliorer le réalisme, il serait possible d'utiliser des images réelles de branches à la manière de Max (*cf.* chapitre 2 section 2.3.4). Debevec [DHT+00] utilise un appareillage sophistiqué pour capturer des images de visages humains avec différentes directions de lumière. Cette technique pourrait également être utilisée pour construire la *BTF* initiale à partir de photos, ce qui enrichirait toute la hiérarchie.

En allant plus loin, il serait intéressant d'essayer de reconstruire un *shader* analytique en mettant en correspondance les données d'illumination extraites de ces images et des courbes paramétriques, à la manière de Stam [Sta01] pour son modèle de rendu de peau (*cf.* chapitre 2 section 1.2.1).

Shaders temps réel

Avec l'arrivée des techniques de calcul d'illumination par pixel (*per pixel shading*) sur les nouvelles générations de cartes, il devient envisageable de transférer l'évaluation des *shaders* complexes au matériel de la carte graphique. Dans ce but, l'arrivée des micro-langages de *shader* [PMTH01] (inspirés de celui de *Renderman*) nous aiderait certainement.

L'ombrage de nos *billboards* est calculé aux quatre sommets du polygone, puis interpolé par le matériel graphique lors de la phase de *rasterisation.* Toujours grâce au calcul par pixel, il est sûrement possible d'améliorer cet ombrage, en calculant pour chaque pixel du *billboard* son ombrage exact à partir des cartes de visibilité chargées sur la carte sous forme de texture.

Animation d'arbres

La structure de visibilité que nous avons introduite au chapitre 6 pour l'ombrage est relativement générale, et pourrait être utilisée pour d'autres types de scènes. En revanche, elle ne permet pas l'animation d'objets sans avoir à recalculer les cartes de visibilité. Certains types d'animations doivent être réalisés à moindre coût. Par exemple, il serait intéressant d'essayer d'animer avec peu d'amplitude un arbre au vent sans recalculer les cartes : le résultat est alors inexact, mais il est probable que, visuellement, cette approximation soit acceptable.

Une approche plus correcte serait de ne reconstruire que les cartes qui ont été modifiées par le mouvement de l'objet. Pour ceci, on pourrait s'inspirer de techniques d'animation en radiosité hiérarchique capables de recalculer uniquement les liens modifiés entre les différentes parties d'une scène (*cluster*) lors du mouvement d'une scène.

Familles de shaders dédiées

La nature offre de nombreuses familles d'objets présentant une structure relativement régulière et comportant des similarités. Il devrait par conséquent être possible pour chacune d'entre elles de calculer analytiquement leur modèle d'illumination, comme nous l'avons fait pour les rameaux de conifères.

Pour les feuillus, dont la structure est plus stochastique (concernant la distribution et l'orientation des feuilles), il faudrait envisager un modèle statistique. Dans ce cas, la connaissance a priori prend une forme plus probabiliste, dont l'intégrale est similaire (il faudrait pour cela, obtenir ce type de données auprès de botanistes). Avec ces données, il doit être possible de construire un modèle paramétrable, convenant donc à beaucoup de familles de feuillus.

Représentation temps réel basée sur le point

Au chapitre consacré à l'étude de cas, nous avons vu une technique présente chez les peintres "réalistes", puis développée par les "impressionnistes", consistant à représenter un paquet de feuilles par un "point", en réalité une tache de pinceau.

Durant l'état de l'art, nous avons évoqué une technique temps réel de rendu par points, les *surfels* (*cf.* chapitre 2 section 2.2) qu'il serait intéressant d'appliquer au rendu d'arbres en s'inspirant de la technique des peintres. Un paquet de feuilles peut être représenté par un "point" (un disque à l'écran), que le matériel graphique sait traiter rapidement. Il faut alors trouver une manière de calculer la couleur de ce disque. Il est raisonnablement envisageable d'utiliser un *shader* similaire à celui que nous avons développé au cours de cette thèse pour les rameaux d'aiguilles. Pour être plus générique, il doit être possible de construire un *shader* dans l'esprit de la fonction de réflectance introduite par Neyret [Ney96] pour les textures volumiques (*cf.* chapitre 2 section 2.4) ou celle introduite par Fournier pour les surfaces complexes [Fou92], et d'essayer de l'enrichir en ajoutant une information (éventuellement statistique) d'auto-ombrage.

Peinture évolutive

Avec notre modèle de *shaders* nous disposons d'un outil capable de représenter le comportement photométrique d'un rameau de conifère, et ceci de manière indépendante de l'objet sous-jacent. Il serait alors possible d'utiliser ces *shaders* comme outil de peinture en semi-relief : un coup de pinceau ordinaire, avec comme matériau "aiguilles", permettrait de faire "vivre" le tableau, en modifiant, soient les paramètres locaux (orientation, longueur, densité), soient les paramètres globaux (orientation de la lumière). En outre, il serait envisageable de s'appuyer sur une peinture existante : en recouvrant virtuellement les différentes zones de la toile par un *shader* approprié, il deviendrait possible d'en modifier l'aspect général en changeant les conditions d'illumination. Par exemple, on pourrait simuler l'aspect d'une toile de Waldmüller (*cf.* Étude de cas) au couché du soleil.

ANNEXE A

Moyenne pondérée d'images avec le matériel graphique

Nous voulons effectuer une moyenne pondérée de n images en profitant de l'accélération qu'offre le matériel graphique par rapport aux traitement de cette opération par logiciel. La composition des transparences (*alpha blending*), opération classique du matériel graphique ne donne pas le résultat attendu. De plus notre but est de composer le résultat de cette moyenne avec les données qui sont déjà stocké dans le tampon de couleur et de profondeur.

Nous verrons en 2 que la composition des transparences ne permet pas de calculer la moyenne d'images, puis en 3 et en 4 nous proposerons deux solutions à ce problème. Enfin, en 5 nous conclurons par une considération à propos du coût.

1 Moyenne pondérée d'images

L'objet de cette annexe est de calculer la moyenne d'images $I1, I2, etc.$ dont les pondérations respectives sont $P1, P2, etc.$:

$$M_{rgba} = P1.I1_{rgba} + P2.I2_{rgba} + ... \tag{A.1}$$

Puis d'appliquer le résultat de cette moyenne M_{rgba} au tampon T_{rgba} :

$$T^{final}_{rgba} = (P1 + P2 + ...).M_{alpha}.M_{rgba} + (1 - (P1 + P2 + ...).M_{alpha}).T^{0}_{rgba}$$

La somme des pondérations vaut 1 *i.e.* $P1 + P2 + ... = 1$, donc :

$$T^{final}_{rgba} = M_{alpha}.M_{rgba} + (1 - M_{alpha}).T^{0}_{rgba} \tag{A.2}$$

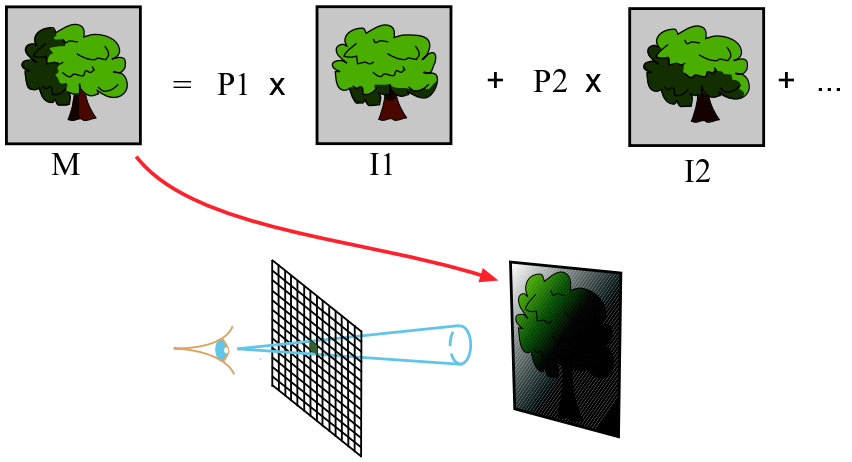

FIG. A.1 – Nous voulons effectuer une moyenne pondérée d'images en profitant de l'accélération qu'offre le matériel graphique.

2 Composition des transparences

Dans cette section, nous calculerons le résultat de la composition des transparences, au sens classique, et nous verrons que ce n'est pas le résultat escompté. A des fins pédagogiques, nous considérons uniquement deux images I1 et I2, de poids respectifs P1 et P2. Le raisonnement effectué ici se généralise facilement. Nous noterons T^k_{rgba} et T^k_{alpha} les composantes RGBA et la composante *alpha* du pixel du tampon après la kème passe.

La composition des transparences classique s'utilise en *OpenGL* avec pour équation de mélange *ADD* et pour coefficients `(SRC_ALPHA,1-SRC_ALPHA)`. Le poids d'une image (P1 ou P2) est transmis à *OpenGL* comme étant la couleur du polygone texturé (*glcolor3f(P1,P1,P1)*). *OpenGL* permet d'effectuer une opération de mélange entre la couleur du polygone et sa texture, grâce à la fonction *glTexEnvf* : nous la configurons à *GL_MODULATE*, ce qui correspond à une multiplication des composantes entre elles.

Après le rendu de la première image les pixels du tampon valent :

$$T^1_{rgba} = (P1.I1_{alpha}).I1_{rgba} + (1 - P1.I1_{alpha}).T^0_{rgba}$$

Puis après le rendu de la deuxième image, nous obtenons :

$$\begin{aligned} T^2_{rgba} &= (P2.I2_{alpha}).I2_{rgba} + (1 - P2.I2_{alpha}).T^1_{rgba} \\ &= (P2.I2_{alpha}).I2_{rgba} + (1 - P2.I2_{alpha}).(P1.I1_{alpha}).I1_{rgba} + \\ &\quad (1 - P2.I2_{alpha}).(1 - P1.I1_{alpha}).T^0_{rgba} \end{aligned} \tag{A.3}$$

En utilisant la composition des transparence (*cf.* équation A.3), nous n'obtenons pas le résultat recherché (*cf.* équation A.2). Nous venons d'effectuer le calcul pour deux images, il est trivial que pour n images le résultat ne sera toujours pas celui recherché. Il faut alors trouver une autre solution.

3 Utilisation d'un tampon annexe

Pour moyenner des images de manière pondérées à l'aide du matériel graphique nous proposons d'effectuer le rendu des images dans un tampon annexe TA (*P-Buffer*) en utilisant une formule

de composition que nous allons décrire maintenant, différente de la composition des transparences. Nous utilisons ensuite ce tampon comme texture d'un polygone représentant le *billboard* . Dans le tampon annexe TA nous calculons $TA_{rgba} = P1.I1_{rgba} + P2.I2_{rgba}$, en utilisant comme équation de mélange *ADD* avec pour coefficients `(SRC_ALPHA,0)`.

Puis nous convertissons l'image résultat de ce calcul en texture, ce qui demande un traitement relativement coûteux au matériel graphique. Cette solution nous offre le résultat attendu, mais est coûteuse, à cause de la conversion du tampon en texture. Dans la section suivante, nous proposons une solution qui n'utilise pas de tampon annexe.

4 Rendu direct

Afin d'éviter l'utilisation d'un tampon annexe, nous proposons une solution adaptée au cas particulier de nos *billboards* et basée sur la séparation de la somme A.2 en deux parties. Nous supposons que toutes les images à moyenner représentent la même géométrie (*i.e.* les valeurs de la composantes *alpha* est identiques sur toutes les images). Ceci est bien le cas de nos *billboards* (*cf.* chapitre 5 section 1.2) lorsque nous composons les images d'illuminations (prises depuis le même point de vue mais avec différentes directions de lumière). Les images prises depuis des points de vue différents que nous composons sont toujours assez similaires (car par hypothèse les trois points de vue sont choisis pour être les plus proches possibles du point de vue de la caméra), leurs valeurs d'*alpha* peuvent donc être considérées comme équivalentes. Ceci se traduit par $M_{alpha} = I1_{alpha} = I2_{alpha} = ...$

Nous séparons donc la somme A.2 en deux parties :

- la partie traitant de ce qui se trouve dans le tampon : $(1 - M_{alpha}).T^0_{rgba}$
  Comme nous avons supposez que toutes les images représentent la même géométrie (*i.e.* les valeurs de la composantes *alpha* est identiques sur toutes les images), cette partie peut être effectuée en une passe en utilisant n'importe laquelle des images à moyenner car $M_{alpha} = I1_{alpha} = I2_{alpha} = ...$
  Pour réaliser cette opération, nous utilisons l'équation de mélange *ADD* en *OpenGL* avec pour coefficients `(0,1-SRC_ALPHA)` et effectuons le rendu d'une image des images (par exemple la première) avec pour coefficient de pondération 1.
- La partie ajoutant les images au tampon :

$$\begin{aligned} M_{alpha}.M_{rgba} &= M_{alpha}.(P1.I1_{rgba} + P2.I2_{rgba} + ...) \\ &= M_{alpha}.P1.I1_{rgba} + M_{alpha}.P2.I2_{rgba} + ... \end{aligned}$$

  Comme $M_{alpha} = I1_{alpha} = I2_{alpha} = ...$, cette partie consiste simplement à ajouter les n images au tampon :

$$M_{alpha}.M_{rgba} = I1_{alpha}.P1.I1_{rgba} + I2_{alpha}.P2.I2_{rgba} + ...$$

  Pour ceci nous utilisons *(SRC_ALPHA,1)* comme coefficients de mélange et nous rendons les n images comme texture d'un polygone ayant pour valeur d'*alpha* la pondération de l'image, comme dans le cas de l'utilisation d'un tampon annexe.

Pour éliminer les interactions dues au test de profondeur, nous le désactivons pour toutes les passes de rendu sauf pour la dernière.

Algorithme A.1 Algorithme

```
Dessine_scène( arrière_vers_avant)
Désactive( écriture_Z)
BlendCoeff( 0 , 1-SRC_ALPHA)
AlphaPolygone( 1 )
Dessine( images[1] )
BlendCoeff( SRC_ALPHA , 1 )
 pour  i ∈ 1..Nimages faire
       | si ( i==Nimages ) alors
       |      | Active(écriture_Z)
       | fin si
       | AlphaPolygone( poids[i])
       | Dessine( images[i])
 fin pour
```

5 Considération de coût et conclusions

La solution utilisant un tampon annexe (*P-buffer*) requière n passes pour construire l'image, la conversion de ce tampon en texture, plus une passe de rendu du *billboard* dans la scène en le texturant de l'image précédemment calculée. Ce qui totalise $n + 1$ rendus, plus surtout une conversion du tampon en texture, ce qui est relativement coûteux.

En revanche, notre solution alternative ne coûte que $n + 1$ passes : une passe pour traiter la composante *alpha* et n passes pour le traitement classique des images. Nous évitons donc la conversion du tampon en texture.

Remarque : L'objet de cette annexe n'est pas nécessaire dans le cas de l'utilisation d'une carte capable de traiter plusieurs textures en une passe (*multi-texturing*[1]).

[1]La fonctionnalité de *multi-texturing* n'est pas présente sur $Onyx^2$ Infinite Reality, mais est disponible sur *Nvidia GeForce* 2 et 3.

ANNEXE B

Lancer de cônes parallèle utilisant une mémoire partagée distribuée sur grappe de PC SCI [MC01]

Comme nous l'avons vu dans l'état de l'art, le lancer de rayons (et donc le lancer de cônes) reste un algorithme coûteux. Les nombreuses optimisations existantes sont poussées à bout, pour diminuer encore les temps de calcul il est intéressant de paralléliser. D'autant plus, que l'algorithme de lancer de rayons se parallélise très simplement.

Le but de cette annexe n'est pas de présenter un nouveau schéma de parallélisation du lancer de rayons, de nombreux travaux portent sur ce domaine et de nombreux algorithmes efficaces existent [WDP99, Hai]. Notre but est de comparer deux types d'architectures, en utilisant l'algorithme de lancer de rayons comme repère : les architectures parallèles et les grappes d'ordinateur.

La machine cible sur laquelle nous avons initialement développé notre lancer de cônes (variante du lancer de rayons) est une machine parallèle : Onyx^2 Infinite Reality disposant de 6 processeurs et 4 Go de mémoire. Cependant, les machines parallèles coûtent cher (l'*Onyx* n'échappe pas à cette règle), plus qu'un réseau de machines disposant au total du même nombre de CPU et de la même quantité de mémoire, donc théoriquement de la même puissance de calcul. L'avantage des machines parallèles est de partager directement les ressources (mémoires, disques, etc.), alors que les réseaux de machines partagent leurs ressources en utilisant le réseau. Cependant, avec l'augmentation des débits des réseaux et la création de nouvelles technologies accélérant les communications [SIR, HEB+01], la question de l'utilisation de grappes de machines en remplacement des machines parallèles se pose.

Afin, d'effectuer cette comparaison entre machine parallèle et grappe de machine, l'équipe *SIRAC* a mis à notre disposition une grappe expérimentale de 12 machines fonctionnant sous le système d'exploitation *Linux*. Le portage de notre lancer de cônes parallèle sur cette grappe a été,

du même coup, l'occasion pour eux de tester en grandeur nature la pertinence de leur choix de réseau et de leur implémentation des outils le partage des ressources.

Nous verrons rapidement en 1 le schéma de parallélisation classique que nous utilisons, nous décrirons en 2 les particularités de cette grappe expérimentale de PC/Linux, puis nous donnerons en 3 les détails utiles (issus de notre expérience) au développement d'applications sur ce type de grappe, et nous conclurons en 4 par des résultats comparatifs entre les deux architectures.

1 Lancer de cônes parallèle

Notre parallélisation de l'algorithme de lancer de rayons suit un schéma classique [CPC98], qui consiste à découper l'image ou l'animation en sous-parties qui sont distribuées entre les processeurs ou les machines. Nous disposons de deux schémas de parallélisation :

- Nous décomposons une animation en images, le calcul de chaque image est distribué sur les différents processeurs. Les images résultats de chaque processus sont sauvées sur un disque partagé par *NFS* (*cf.* figure B.1 à gauche).
- Nous décomposons une image en sous-images, le calcul de chaque partie est distribuée sur les différent processeurs. Un processus particulier s'occupe de collecter les sous-parties de l'image, puis sauve le résultat final sur le disque (*cf.* figure B.1 à droite).

Dans les deux cas tous les processus partagent les données de la scène, seuls les objets se déplaçant sont dupliqués (*i.e.* la caméra).

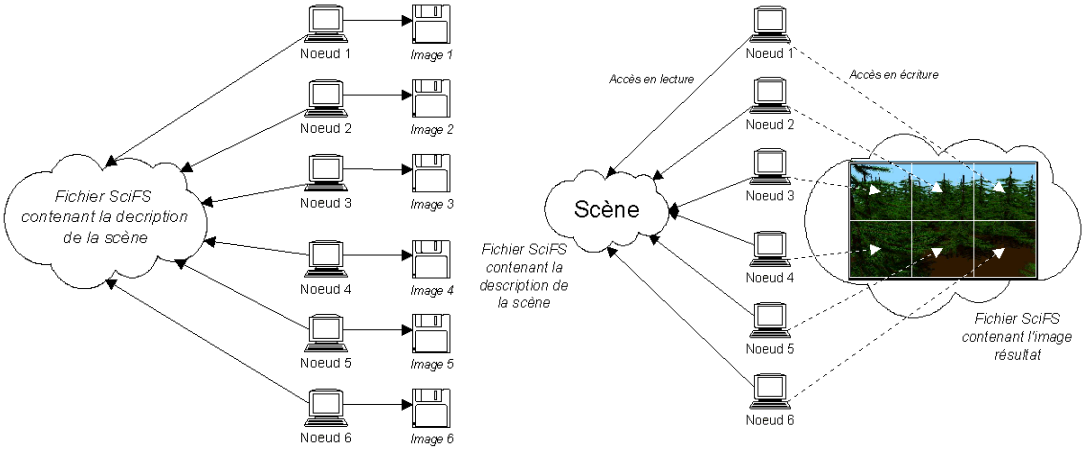

FIG. B.1 – Les deux schémas de parallélisation.

Ces deux schémas de parallélisation sont efficaces : peu de données sont partagées en écriture, ce qui limite le nombre de verrous[1] à utiliser et donc les attentes potentielles. La scène est partagée par tous les processus et est accédée uniquement en lecture (à part lors du chargement initiale), nous n'utilisons aucun verrou pour cette partie des données. Pour le calcul parallèle d'une animation, notre implémentation ne partage aucun segment de mémoire (les images étant indépendantes), donc aucun interblocage n'est possible : cas idéal. En revanche, pour le calcul distribué d'une image, la zone de données lui correspondant est partagée en écriture, ce qui impose l'utilisation d'un verrou.

[1] Un verrou permet d'autoriser l'accès à une ressource à un nombre limité de tâches. Les tâches demandants une ressource non disponible sont mises en attente.

Dans un but de comparaison coût/performance entre une machine parallèle ($Onyx^2$ Infinite Reality) et une grappe de PC, nous avons testé ces deux schémas de parallélisation sur les deux architectures. Avant d'exposer les résultats de cette comparaison, nous présentons les spécificités de la grappe de PC utilisée.

2 Grappe de PC et SciFS

La grappe utilisée dispose de 12 Pentium II (450 MHz) équipée du réseau *SCI* facilitant le partage de la mémoire. Apres un survol des capacités qu'offre le réseau *SCI* en 2.1, nous verrons en 2.2 le principe de fonctionnement de la mémoire partagée distribuée logicielle *SciFS* développé par les membres de l'équipe *SIRAC*.

2.1 La technologie SCI

La technologie réseau *Scalable Coherent Interface* (*SCI*) permet de lire ou d'écrire dans la mémoire de n'importe quelle autre machine du réseau sans interrompre le processeur de la machine distante et ceci par deux moyens :

- Le processeur demandant effectue sa requête à sa carte *SCI* par entrée/sortie programmées, puis attend la réponse.
- Le processeur demandant effectue la requête à sa carte *SCI* par émission d'une requête DMA, puis reprend la main jusqu'à ce que la carte *SCI* émette une interruption l'informant du résultat.

Dans les deux cas la cartes *SCI* de la machine demandante transmet la requête à la carte *SCI* de la machine distante, qui interroge la mémoire (en passant par le bus) sans interrompre son processeur, puis transmet le résultat.

Cette technologie permet un partage puissant des ressources mémoire au sein d'une grappe de PC. Un accès à la mémoire d'une autre machine à travers la technologie *SCI* est plus lent qu'un accès direct à la mémoire locale, mais considérablement plus rapide que si cette requête était traitée par un système de *socket*. Cette technologie a conduit des membres de l'équipe *SIRAC* à développer une mémoire partagée distribuée (*DSM*) logicielle : *SciFS*.

2.2 Une mémoire partagée distribuée logicielle basée sur un réseau SCI

La mémoire partagée distribuée (*Distributed Shared Memory DSM*) logicielle, baptisé *SciFS*, développé par l'équipe *SIRAC*, et plus particulièrement par Emannuel Cecchet durant sa thèse [Cec01, KHCR99, SIR] se présente sous la forme d'une extension (module) au noyau de *Linux*. Elle permet aux développeurs de "haut niveau" (en l'occurrence nous) de profiter pleinement des capacités de la technologie *SCI* sans avoir à se plonger dans les couches bases du système (*cf.* figure B.2).

Concrètement, cette *DSM* permet de se servir des $n \times 256Mo$ de mémoire des n machines de la grappe comme si il s'agissait d'une mémoire unique, et ce de manière transparente. Le module de *DSM* se charge de répartir les données sur les différentes machines et éventuellement de les déplacer ou de les dupliquer en fonction des statistiques d'accès aux données.

Dans la pratique, cette mémoire partagée s'utilise comme un fichier : une fois crée, le fichier mémoire peut être ouvert depuis n'importe quelle machine du réseau, et permet ainsi, à n'importe quel processus d'accéder aux données en lecture comme en écriture. Un système de verrous est mis à disposition des développeurs pour gérer les accès concurrents aux données.

Pour limiter les accès à distance qui, bien que rapides, sont plus coûteux que les accès locaux, *SciFS* implémente plusieurs techniques [KCR98]. Nous citerons les trois couramment utilisées :

- *Fixed* : la machine qui crée le fichier de mémoire partagée le garde en local jusqu'à sa libération.
- *First touch* : la première machine à accéder aux données du fichier de mémoire partagée le garde en local jusqu'à sa libération.
- *Global* : les données de la mémoire partagée sont dupliquées sur toutes les machines y faisant des accès nombreux en lecture. Un système de jeton est utilisé pour l'écriture : la machine disposant du jeton possède les droits en écriture, cela signifie que les écritures effectuées par les autres machines lui sont transmises, puis les données dupliquées sur les autres machines sont mises à jour. En suivant les statistiques, la machine effectuant le plus d'écritures dispose du jeton.

Le développeur de haut niveau fournit en options une de ces trois techniques lors de la création du fichier de mémoire partagée, puis ne s'en occupe plus.

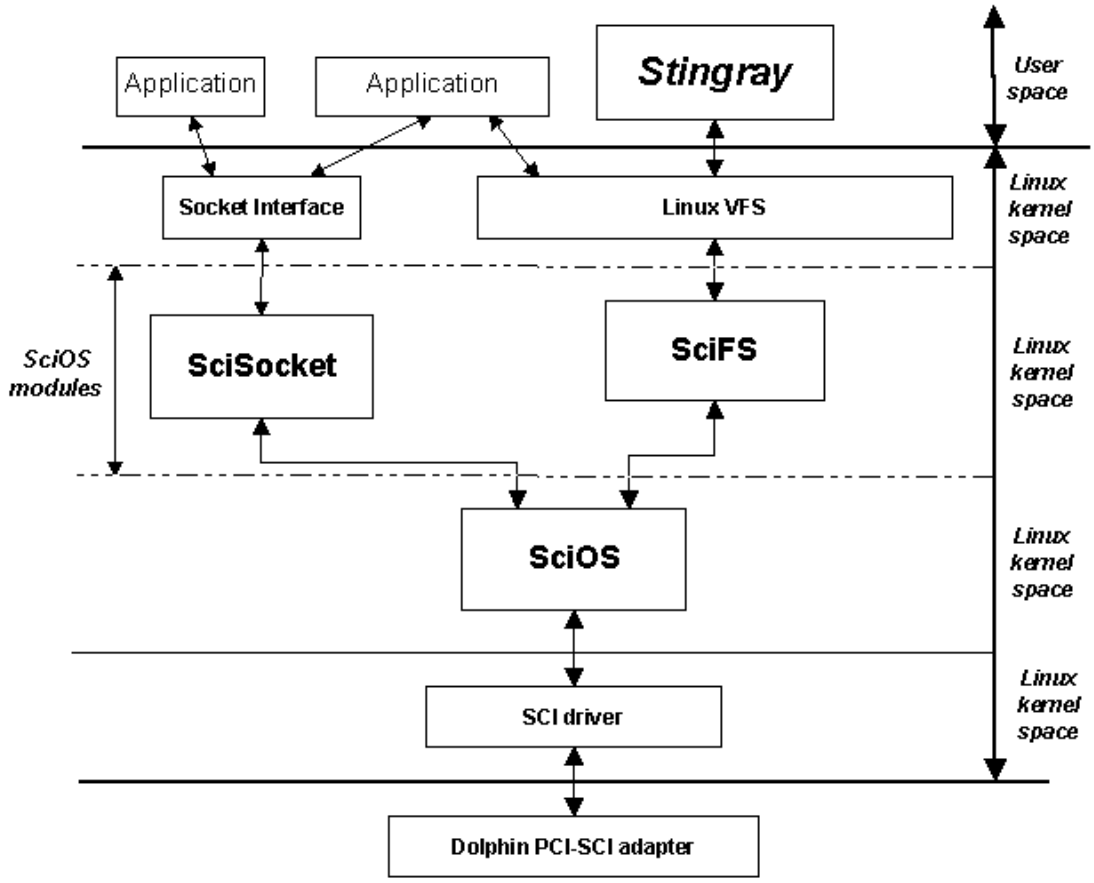

FIG. B.2 – Notre lancer de cônes parallèle est implémenté sur une grappe de PC équipée d'un noyau *Linux* étendu par le module *SciFS*.

3 Développement d'une application sur grappe SciFS

L'objectif de cette section est de discuter de l'utilisation de la *DSM SciFS* avec notre lancer de cônes parallèle et de donner des informations techniques aux futures développeurs amener à l'utiliser.

3.1 Répartition des données

Dans nos deux schémas de parallélisation (*cf.* section 1), nous utilisons l'option *Global* pour la zone mémoire contenant la scène et, l'option *Fixed* pour l'image. Avec la technique *Global*, le système *SciFS* duplique les données de la scène sur les machines accèdent en lecture aux segments partagés[2]. Le calcul se fera alors, sans pratiquement aucun surcoût dû à la répartition des données, puisque chaque processus accède à des données qui sont dupliquées en locale. Nous fixons le segment de l'image sur la machine du processus collecteur de données. Afin de ne pas surcharger le réseau en requête de petite taille, nos processus de calcul stockent leurs résultats dans un tampon et l'envoient au processus collecteur une fois qu'il est plein[3].

3.2 Implémentation

L'implémentation de notre lancer de cônes sur *SciFS* a été rapide (environ 2 semaines), aucun changement n'a été nécessaire au niveau des schémas de parallélisation. Par contre des problèmes techniques non triviaux ont été à résoudre. Cette section a pour but de faire gagner du temps aux futures développeurs (*C++*) amener à utiliser *SciFS* :

- Gestion mémoire : la fonction *new* (C++) doit être surchargée pour allouer un segment partagé (*i.e.* ouvrir un fichier sur le disque). Une nouvelle classe *allocateur* pour *STL* doit être écrite.
- Processus légers et processus lourds : l'utilisation de processus légers (*thread*) ne peut plus se faire puisque chaque processus tourne sur des machines différentes. Nous les avons remplacés par des appels systèmes créant un processus lourd (*fork*) exécutant une commande à distance de type *rsh*.
- Initialisation des données : la table de symboles utilisée par un programme écrit en C++ (*VTBL*) ne se trouve pas dans un segment de données partagées. Il faut que chaque processus chargent la scène dans le segment de mémoire partagée, ce qui est redondant, mais qui permet d'initialiser correctement la *VTBL* de tous les processus. Ce chargement est rapide et n'est effectué qu'une fois au début du programme.

De plus amples informations technique à propos de *SciFS* peuvent être trouvées dans [SIR].

4 Performances et conclusions

Nos deux plates-formes de tests ont été une Onyx^2 Infinite Reality équipé de 6 processeurs R12000 à 400 MHz avec 4 Go de mémoire, et une grappe de 12 PC Penium II à 450 MHz avec 512Mo de mémoire chacun (totalisant 6 Go de mémoire partagée). La parallélisation utilisée offre, sur les deux plates-formes, des performances linéaires dans le cas d'une animation et quasi-linéaire

[2] Les tailles de nos données sont toujours inférieures à la mémoire de chaque machine, donc cette duplication ne pose pas de problème au système.

[3] La taille de paquets donnant un résultat optimal est de 4Ko, la taille de nos tampons est donc identique, mais ceci est paramétrable.

dans le cas d'une image (sur la grappe le temps de calcul est 5.9 fois plus rapide avec 6 processus qu'avec un). Ces résultats sont dus au faible partage des ressources, donc à l'utilisation de peu de verrous, qui sont les principales sources d'attente et de ralentissement. En performances absolues, l'*Onyx* avec 6 processus et environ 30% plus rapide que 6 processus répartis sur six PC de la grappe[4]. En revanche, en profitant des capacités maximales de la grappe (soit 12 machines), les temps lui deviennent favorables (30% plus rapide que l'*Onyx* à 6 processeurs). Avec 12 processeurs, il est évident que l'*Onyx* repasserait devant, mais ceci se ferait au prix d'un coût financier élevés. Une *Onyx* avec 6 processeurs et 4 Go de mémoire (sans carte graphique) coûte environ 1MF alors qu'une grappe de 12 PC équipés de cartes *SCI* (sans écran) coûte environ 250KF, soit quatre fois moins.

Dans le cas de notre application particulièrement bien profilée à l'utilisation d'une grappe (peu de partage de données), il est financièrement plus avantageux d'utiliser une grappe de PC qu'une machine parallèle, et ce pour des performances équivalentes. Il est évident que cette affirmation n'est pas généralisable, et dépend du type d'application développé mais les résultats de cette annexe prouve qu'il est utile d'investiguer la cette direction de grappe de machines.

[4]Un processeur R12000 à 400MHz est plus rapide qu'un Pentium II à 450MHz au niveau des opérations flottantes, ce qui explique, en grande partie, que l'*Onyx* est plus rapide que la grappe en performance absolue.

BIBLIOGRAPHIE

[Ama84] J. Amanatides. Ray tracing with cones. In *SIGGRAPH 1984, Computer Graphics Proceedings*, pages 129–135, 1984. pages 43

[Arb] Les arbres d'amérique du nord. http ://www.domtar.com/arbre/photot.htm. pages 21

[ARB90] J. M. Airey, J. H. Rohlf, and F. Brooks Jr. Towards image realism with interactive update rates in complex virtual building enviroments. In *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 1990. pages 46

[Ber86] P. Bergeron. A general version of Crow's shadow volumes. *IEEE Computer Graphics and Applications*, pages 17–28, 1986. pages 42

[Bio] Bionatics. http ://www.bionatics.com. pages 10, 15, 37, 38, 129

[Bli77] J. F. Blinn. Models of light reflection for computer synthesized pictures. In *SIGGRAPH 1977, Computer Graphics Proceedings*, pages 192–198, July 1977. pages 51, 52

[Bli78] J. F. Blinn. Simulation of wrinkled surfaces. In *SIGGRAPH 1978, Computer Graphics Proceedings*, pages 286–292, August 1978. pages 51

[Bli82] J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *SIGGRAPH 1982, Computer Graphics Proceedings*, pages 21–29, July 1982. pages 11, 55, 56

[BM93] B. G. Becker and N. L. Max. Smooth transitions between bump rendering algorithms. In *SIGGRAPH 1993, Computer Graphics Proceedings*, pages 183–190, August 1993. pages 11, 54, 55

[BN76] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, pages 542–547, October 1976. pages 51

[bot] Plants of university of delaware botanic garden. http ://blue-hen.ags.udel.edu/udbg/intro_udbg.html. pages 21

[Car84] L. Carpenter. The a-buffer, an antialiased hidden surface method. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, pages 103–108, 1984. pages 42

[Cat74] E. E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* Ph.d. thesis, University of Utah, December 1974. pages 42, 51

[CBL99] C.-F. Chang, G. Bishop, and A. Lastra. LDI tree : A hierarchical representation for image-based rendering. In *SIGGRAPH 1999, Computer Graphics Proceedings*, pages 291–298, Los Angeles, 1999. pages 61

[Cec01] E. Cecchet. *Apport des réseaux à capacité d'adressage pour des grappes à mémoire partagée distribuée logicielle - Conception et applications.* PhD thesis, Thèse de doctorat de l'Institut National Polytechnique de Grenoble, 2001. pages 137

[Cha97] C. Chaudy. *Modélisation et rendu d'images réalistes de paysages naturels*. PhD thesis, Université Joseph Fourier, 1997. pages 10, 35, 36, 58

[CIR] CIRAD. Centre de coopération Internationale en Recherche Agronomique pour le Développement, AMAP. http ://www.cirad.fr/produits/amap/amap.html. pages 10, 24, 37, 38

[Cla76] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, October 1976. pages 46

[CMS87] B. Cabral, N. Max, and R. Springmeyer. Bidirectional reflection functions from surface bump maps. In *SIGGRAPH 1987, Computer Graphics Proceedings*, pages 273–281, July 1987. pages 47, 54

[COFHZ98] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. In *Computer Graphics Forum*, pages 243–253, 1998. pages 46

[Coo84] R. L. Cook. Shade trees. In *SIGGRAPH 1984, Computer Graphics Proceedings*, pages 223–231, July 1984. pages 34

[COS95] D. Cohen-Or and A. Shaked. Visibility and dead-zones in digital terrain maps. *Computer Graphics Forum*, pages C/171–C/180, September 1995. pages 48

[CPC98] R. Cook, T. Porter, and L. Carpenter. Distributed raytracing. In *Significant Seminal Papers of Computer Graphics : Pioneering Efforts that shaped the Field*, pages 77–86, 1998. pages 136

[Cro77] F. C. Crow. Shadow algorithms for computer graphics. In *SIGGRAPH 1977, Computer Graphics Proceedings*, pages 242–248, July 1977. pages 42

[CT82] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, pages 7–24, January 1982. pages 52

[DDTP00] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 239–248, 2000. pages 46

[DHL+98] O. Deussen, P. Hanrahan, B. Lintermann, R. Mech, M. Pharr, and P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *SIGGRAPH 1998, Computer Graphics Proceedings*, pages 275–286, July 1998. pages 36

[DHT+00] P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, and M. Sagar. Acquiring the reflectance field of a human face. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 145–156, 2000. pages 129

[Dis] B. Discoe. Virtual terrain project. http ://www.vterrain.org/index.html. pages 71

[Dis98] J.-M. Dischler. Efficiently rendering macrogeometric surface structures using bi-directional texture functions. In *Eurographics Workshop on Rendering 98*, pages 169–180, 1998. pages 11, 70

[dREF+88] P. de Reffye, C. Edelin, J. Françon, M. Jaeger, and C. Puech. Plant models faithful to botanical structure and development. In *SIGGRAPH 1988, Computer Graphics Proceedings*, pages 151–158, August 1988. pages 37

[Dur99] F. Durand. *3D Visibility : Analytical Study and Applications.* PhD thesis, Universite Joseph Fourier, Grenoble, France, July 1999. pages 46

[DvGNK] K. Dana, B. van Ginneken, S. Nayar, and J. Koenderink. Columbia-utrecht reflectance and texture database. http ://www.cs.columbia.edu/CAVE/curet/.index.html. pages 11, 53, 54

[DvGNK99] K.J. Dana, B. van Ginneken, S.K. Nayar, and J.J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics*, 18(1) :1–34, January 1999. pages 11, 53, 54

[DWS+97] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing terrain : Real-time optimally adapting meshes. In *IEEE Visualization '97*, 1997. pages 40

[Ebe01] David S. Ebert. Simulating nature : Realistic and interactive techniques, 2001. pages 55

[EKE01] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Workshop on Graphics Hardware 2001*. Siggraph/Eurographics, August 2001. pages 69

[FFC82] A. Fournier, D. Fussell, and L. Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, pages 371–384, June 1982. pages 34

[Fou89] A. Fournier. The modelling of natural phenomena. In *Proceedings of Graphics Interface '89*, pages 191–202, June 1989. pages 35

[Fou92] A. Fournier. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination*, pages 45–52, May 1992. pages 130

[FvDFH90] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics : Principles and Practices (2nd Edition).* Addison Wesley, 1990. pages 11, 50, 51

[GAC+89] A. S. Glassner, J. Arvo, R. L. Cook, E. Haines, P. Hanrahan, P. Heckbert, and D. B. Kirk. *An Introduction to Ray Tracing.* Academic Press, London, 1989. pages 42, 44

[Gad] B. Gadrat. Végétaux et architecture de paysage. http ://www.designvegetal.com/gadrat. pages 21

[GGSC96] S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen. The lumigraph. In *SIGGRAPH 1996, Computer Graphics Proceedings*, pages 43–54, August 1996. pages 11, 61, 62

[GH95] M. Garland and P. S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical report, CS Dept., Carnegie Mellon U., September 1995. pages 40

[GK93] N. Greene and M. Kass. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240, 1993. pages 42

[GMN94] J. S. Gondek, G. W. Meyer, and J. G. Newman. Wavelength dependent reflectance functions. In *SIGGRAPH 1994, Computer Graphics Proceedings*, pages 213–220, July 1994. pages 53

[Gol97] D. B. Goldman. Fake fur rendering. In *SIGGRAPH 1997, Computer Graphics Proceedings*, pages 127–134, 1997. pages 63

[HA90] P. Haeberli and K. Akeley. The accumulation buffer : Hardware support for high-quality rendering. In *SIGGRAPH 1990, Computer Graphics Proceedings*, pages 309–318, August 1990. pages 42

[Hai] E. Haines. Ray tracing news guide. pages 44, 135

[HDKS00] W. Heidrich, K. Daubert, J. Kautz, and H.-P. Seidel. Illuminating micro geometry based on precomputed visibility. In *Proceedings of the Computer Graphics Conference 2000*, pages 455–464, July 2000. pages 11, 54, 55

[HEB+01] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordon Stoll, Matthew Everett, and Pat Hanrahan. Wiregl : A scalable graphics system for clusters. In *SIGGRAPH 1990, Computer Graphics Proceedings*, 2001. pages 135

[HH84] P. S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. In *SIGGRAPH 1984, Computer Graphics Proceedings*, pages 119–127, 1984. pages 43

[Hop96] H. Hoppe. Progressive meshes. In *SIGGRAPH 1996, Computer Graphics Proceedings*, pages 99–108, 1996. pages 10, 39, 40

[Hop97] H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 1997, Computer Graphics Proceedings*, pages 189–198, August 1997. pages 39

[Hop98] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings IEEE Visualization'98*, pages 35–42, 1998. pages 40

[HS99] W. Heidrich and H.-P. Seidel. Realistic, hardware-accelerated shading and lighting. In *SIGGRAPH 1999, Computer Graphics Proceedings*, pages 171–178, 1999. pages 54

[Jak00] A. Jakulin. Interactive vegetation rendering with slicing and blending. In *Proc. Eurographics 2000 (Short Presentations)*. Eurographics, August 2000. pages 68

[Jon71] C. B. Jones. A new approach to the 'hidden line' problem. *Computer Journal*, pages 232–237, August 1971. pages 46

[JW89] D. Jevans and B. Wyvill. Adaptive voxel subdivision for ray tracing. In *Proceedings of Graphics Interface '89*, pages 164–172, June 1989. pages 44

[Kaj85] J. T. Kajiya. Anisotropic reflection models. In *SIGGRAPH 1985, Computer Graphics Proceedings*, pages 15–21, July 1985. pages 41, 52

[KCR98] P. T. Koch, E. Cecchet, and X. Ronsset de Pina. Global management of coherent shared memory on an SCI cluster. In *Proc. of the European Multimedia, Multiprocessor Systems and Electronic Commerce Conference (SCI Europe'98)*, September 1998. pages 138

[KHCR99] P. T. Koch, J. S. Hansen, E. Cecchet, and X. Ronsset de Pina. SciOS : An SCI-based software distributed shared memory. In *Proc. of the 1st Workshop on Software Distributed Shared Memory (WSDSM'99)*, June 1999. pages 137

[KK89] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. In *SIGGRAPH 1989, Computer Graphics Proceedings*, pages 271–280, July 1989. pages 11, 56, 63, 64

[KN01] T.-Y. Kim and U. Neumann. Opacity shadow maps. In *Eurographics Workshop on Rendering 2001*, July 2001. pages 47

[KS01] J. Kautz and H.-P. Seidel. Hardware accelerated displacement mapping for image based rendering. In *Graphics Interface*, May 2001. pages 69

[LD] Lintermann and Deussen. Xfrog. `http ://www.xfrogdownloads.com/`. pages 15, 37, 129

[Len00] J. E. Lengyel. Real-time fur. In *Eurographics Workshop on Rendering 2000*, June 2000. pages 68

[Lew94] R. R. Lewis. Making shaders more physically plausible. *Computer Graphics Forum*, pages 109–120, June 1994. pages 52

[LH96] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH 1996, Computer Graphics Proceedings*, pages 31–42, August 1996. pages 11, 61, 62

[Lin68] A. Lindenmayer. Mathematical models for cellular interactions in development, I & II. *Journal of Theoretical Biology*, pages 280–315, 1968.
Lindenmayer's original articles on L-Systems. pages 36

[LKR+96] P. Lindstrom, D. Koller, W. Ribarsky, L.Hodges, N. Faust, and G. Turner. Real-time continuous level of detail rendering of height fields. In *SIGGRAPH 1996, Computer Graphics Proceedings*, pages 109–118, 1996. pages 10, 40, 45

[LL94] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH 1994, Computer Graphics Proceedings*, pages 451–458, July 1994. pages 66

[LPFH01] J. Lengyel, E. Praun, A. Finkelstein, and H. Hoppe. Real-time fur over arbitrary surfaces. In *ACM 2001 Symposium on Interactive 3D Graphics*, USA, Mar 2001. pages 11, 68

[LV00] T. Lokovic and E. Veach. Deep shadow maps. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 385–392, 2000. pages 47

[LW85] M. Levoy and T. Whitted. The use of points as a display primitive. Technical report, Computer Science Department, University of North Carolina at Chapel Hill, 1985. pages 59

[Man78] B. Mandelbrot. *Fractals : Form, chance, and dimension.* W. H. Freeman, New York, 1978. pages 34

[Max88] N.L. Max. Horizon mapping : shadows for bump-mapped surfaces. *The Visual Computer*, pages 109–117, July 1988. pages 10, 47, 48, 108, 110

[Max96] N. Max. Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In *Eurographics Rendering Workshop 1996*, pages 165–174. Eurographics, June 1996. ISBN 3-211-82883-4. pages 61, 63

[MB95] L. McMillan and G. Bishop. Plenoptic modeling : An image-based rendering system. In *SIGGRAPH 1995, Computer Graphics Proceedings*, pages 39–46, August 1995. pages 61

[MC01] A. Meyer and E. Cecchet. Stingray : Cone tracing using a software dsm for sci clusters. In *Third IEEE International Conference on Cluster Computing,*, 2001. pages 96, 135

[McC00] M. D. McCool. Shadow volume reconstruction from depth maps. In *ACM Transactions on Graphics*, pages 1–26, 2000. pages 42

[MDK99] N. Max, O. Deussen, and B. Keating. Hierarchical image-based rendering using texture mapping hardware. In *Eurographics Workshop on Rendering 99*, pages 57–62, 1999. pages 61, 63, 77

[Met] Metacreations. Bryce. http ://www.metacreations.com/products/bryce4/. pages 34

[Mil86] G. S. P. Miller. The definition and rendering of terrain maps. In *SIGGRAPH 1986, Computer Graphics Proceedings*, pages 39–48, August 1986. pages 34

[Mil88] G.S.P. Miller. From wire-frames to furry animals. In *Proceedings of Graphics Interface '88*, pages 138–145, June 1988. pages 11, 56

[MKM89] F. Kenton Musgrave, Craig E. Kolb, and Robert S. Mace. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH 1989, Computer Graphics Proceedings*, pages 41–50, July 1989. pages 10, 34, 35

[MN98] A. Meyer and F. Neyret. Interactive volumetric textures. In *Eurographics Workshop on Rendering 1998*, pages 157–168, June 1998. pages 11, 66, 67, 68

[MN00] A. Meyer and F. Neyret. Multiscale shaders for the efficient realistic rendering of pine-trees. In *Graphics Interface*, May 2000. pages 6, 73

[MNP01] A. Meyer, F. Neyret, and P. Poulin. Interactive rendering of trees with shading and shadows. In *Eurographics Workshop on Rendering*, Jul 2001. pages 7, 99

[MO95] N. Max and K. Ohsaki. Rendering trees from precomputed Z-buffer views. In *Eurographics Workshop on Rendering 1995*, June 1995. pages 61, 63

[MRP98] G.S.P. Miller, S. Rubin, and D. Ponceleon. Lazy decompression of surface light fields for precomputed global illumination. *Eurographics Workshop on Rendering 1998*, pages 281–292, June 1998. pages 61

[MZG01] J. van Baar M. Zwicker, H. Pfister and M. Gross. Surface splatting. In *Computer Graphics (SIGGRAPH '01 Proceedings)*, August 2001. pages 59

[NC99] F. Neyret and M.P. Cani. Pattern-based texturing revisited. In *SIGGRAPH 1999, Computer Graphics Proceedings*, pages 235–242, August 1999. pages 11, 68

[Ney95] F. Neyret. A general and multiscale method for volumetric textures. In *Graphics Interface'95 Proceedings*, pages 83–91, May 1995. pages 63, 64

[Ney96] F. Neyret. Synthesizing verdant landscapes using volumetric textures. In *Eurographics Workshop on Rendering'96*, pages 215–224, June 1996. pages 11, 63, 64, 65, 130

[Ney98] F. Neyret. Modeling animating and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, January 1998. pages 63, 64

[Nom95] T. Noma. Bridging between surface rendering and volume rendering for multiresolution display. In *Eurographics Workshop on Rendering 1995*, pages 31–40, June 1995. pages 65

[OBM00] M.M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *SIGGRAPH 2000, Computer Graphics Proceedings*, Computer Graphics Proceedings, July 2000. pages 61

[ON94] M. Oren and S. K. Nayar. Generalization of lambert's reflectance model. In *SIGGRAPH 1994, Computer Graphics Proceedings*, pages 239–246, July 1994. pages 52

[PC01] Franky Perbet and M.-P. Cani. Animating prairies in real-time. In *ACM Symposium on Interactive 3D Graphics*, USA, Mar 2001. pages 41

[PCD+97] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle. View-based rendering : Visualizing real objects from scanned range and color data. In *Eurographics Workshop on Rendering 97*, pages 23–34, June 1997. pages 60

[PF90] P. Poulin and A. Fournier. A model for anisotropic reflection. In *SIGGRAPH 1990, Computer Graphics Proceedings*, pages 273–282, August 1990. pages 11, 52, 53, 56

[PG01] M. Pauly and M. Gross. Spectral processing of point-sampled geometry. In *SIGGRAPH 2001, Computer Graphics Proceedings*, August 2001. pages 59

[PH96] M. Pharr and P. Hanrahan. Geometry caching for ray-tracing displacement maps. In *Eurographics Rendering Workshop 1996*, pages 31–40, 1996. pages 45

[PHL91] J. W. Patterson, S. G. Hoggar, and J. R. Logie. Inverse displacement mapping. *Computer Graphics Forum*, 10(2) :129–139, June 1991. pages 45

[PJM94] P. Prusinkiewicz, M. James, and R. Měch. Synthetic topiary. In *SIGGRAPH 1994, Computer Graphics Proceedings*, pages 351–358, July 1994. pages 36

[PLH88] P. Prusinkiewicz, A. Lindenmayer, and J. Hanan. Developmental models of herbaceous plants for computer imagery purposes. In *SIGGRAPH 1988, Computer Graphics Proceedings*, pages 141–150, August 1988. pages 36

[PMTH01] K. Proudfoot, W. R. Mark, S. Tzvetkov, and P. Hanrahan. A real-time procedural shading system for programmable graphics hardware. In *SIGGRAPH 2001, Computer Graphics Proceedings*, August 2001. pages 129

[Pro] Virtual Terrain Project. Clouds. http ://www.vterrain.org/Atmosphere/clouds.html. pages 55

[PTS99] S. Premoze, W. Thompson, and P. Shirley. Geospecific rendering of alpine terrain. In *10th Eurographics Workshop on Rendering*, June 1999. pages 10, 45

[PZvBG00] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels : Surface elements as rendering primitives. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 335–342, 2000. pages 59

[RB85] W. T. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *SIGGRAPH 1985, Computer Graphics Proceedings*, pages 313–322, July 1985. pages 11, 58

[Ree83] W.T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, pages 91–108, April 1983. pages 11, 58

[Res] Pacific Meridian Resources. Virtual forest. http ://www.innovativegis.com/products/vforest. pages 71

[RL00] S. Rusinkiewicz and M. Levoy. QSplat : A multiresolution point rendering system for large meshes. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 343–352, 2000. pages 59

[RSC87] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering antialiased shadows with depth maps. In *SIGGRAPH 1987, Computer Graphics Proceedings*, pages 283–291, July 1987. pages 46

[Rus97] S. Rusinkiewicz. A survey of brdf representation for computer graphics, 1997. pages 53

[SB87] J. M. Snyder and A. H. Barr. Ray tracing complex models containing surface tessellations. In *SIGGRAPH 1987, Computer Graphics Proceedings*, pages 119–128, July 1987. pages 44

[Sch94a] C. Schlick. An inexpensive BRDF model for physically-based rendering. *Computer Graphics Forum*, pages C/233–C/246, 1994. pages 52

[Sch94b] C. Schlick. A survey of shading and reflectance models. *Computer Graphics Forum*, 13(2) :121–131, June 1994. pages 52

[Sch98] G. Schaufler. Per-object image warping with layered impostors. In *Eurographics Workshop on Rendering 98*, pages 145–156, 1998. pages 67

[SD01] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 01)*. Eurographics, 2001. pages 11, 59

[SDDS00] G. Schaufler, J. Dorsey, X. Decoret, and F.X. Sillion. Conservative volumetric visibility with occluder fusion. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 229–238, 2000. pages 46

[SGHS98] J. Shade, S.J. Gortler, L. He, and R. Szeliski. Layered depth images. In *SIGGRAPH 1998, Computer Graphics Proceedings*, pages 231–242, July 1998. pages 61, 63

[SGI] SGI. Volumizer. http ://www.sgi.com/software/volumizer/. pages 67

[SIR] SIRAC. Scios/scifs. http ://sci-serv.inrialpes.fr/. pages 135, 137, 139

[Sla92] M. Slater. A comparison of three shadow volume algorithms. *The Visual Computer*, pages 25–38, 1992. pages 42

[SN01] F. Senegas and F. Neyret. Visualisation haute-qualité de forêts en temps-réel à l'aide de représentations alternatives. Master's thesis, INPG, Grenoble, Juin 2001. pages 11, 68, 69

[Sof] Planetside Software. Terragen. http ://www.planetside.co.uk/. pages 71

[SS98] C. Soler and F. Sillion. Fast calculation of soft shadow textures using convolution. In *SIGGRAPH 1998, Computer Graphics Proceedings*, pages 321–332, July 1998. pages 47

[Sta94] J. Stam. Stochastic rendering of density fields. In *Proceedings of Graphics Interface '94*, pages 51–58, Banff, Alberta, Canada, May 1994. pages 55

[Sta01] J. Stam. Model for a skin layer bounded by rough surfaces. In *Eurographics Workshop on Rendering 2001*, June 2001. pages 11, 55, 56, 129

[Ste97] J. Stewart. Hierarchical visibility in terrains. In *Eurographics Rendering Workshop*, June 1997. pages 48

[Ste98] J. Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. *IEEE Transactions on Visualization and Computer Graphics*, 4(1) :82–93, March 1998. pages 10, 48, 110

[Tai92] F. Taillefer. Fast inverse displacement mapping and shading in shadow. In *Graphics Interface '92 Workshop on Local Illumination*, pages 53–60, May 1992. pages 45

[TS66] K. E. Torrance and E. M. Sparrow. Polarization, directional distribution, and off-specular peak phenomena in light reflected from roughened surfaces. *Journal of Optical Society of America*, 1966. pages 52

[TS67] K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of Optical Society of America*, 1967. pages 52

[TS91] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walktroughs. In *SIGGRAPH 1991, Computer Graphics Proceedings*, 1991. pages 46

[WAA+00] D.N. Wood, D.I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D.H. Salesin, and W. Stuetzle. Surface light fields for 3D photography. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 287–296, 2000. pages 61

[WAT92] S. H. Westin, J. R. Arvo, and K. E. Torrance. Predicting reflectance functions from complex surfaces. In *SIGGRAPH 1992, Computer Graphics Proceedings*, pages 255–264, 1992. pages 53

[WDP99] B. Walter, G. Drettakis, and S. Parker. Interactive rendering using the render cache. In *Rendering techniques '99 (Proceedings of the 10th Eurographics Workshop on Rendering)*, pages 235–246, June 1999. pages 135

[WE98] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH 1998, Computer Graphics Proceedings*, August 1998. pages 66

[Whi80] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, pages 343–349, 1980. pages 42

[Wil83] L. Williams. Pyramidal parametrics. In *SIGGRAPH 1983, Computer Graphics Proceedings*, pages 1–11, July 1983. pages 51

[WP95] J. Weber and J. Penn. Creation and rendering of realistic trees. In *SIGGRAPH 1995, Computer Graphics Proceedings*, pages 119–128, August 1995. pages 10, 36, 38, 39

Thèse de doctorat en sciences de l'Université Joseph Fourier

Représentations d'arbres réalistes et efficaces pour la synthèse d'images de paysages

Alexandre Meyer

Résumé : Cette thèse, située dans le cadre de la synthèse d'images de paysages, est consacrée à des représentations d'arbres adaptées soit au rendu de haute-qualité, soit au rendu temps réel. Les techniques de modélisation d'arbres donnent à ce jour de bons résultats en termes de diversité d'espèces et de formes représentables. Cependant, leur représentation géométrique nécessite une multitude de polygones représentant des détails fins, source d'un coût de calcul important et de gros problèmes d'aliassage lors du rendu de l'image. Pourtant, la construction de niveaux de détails par simplification de maillage ne peut s'appliquer à un arbre sans en modifier l'opacité et l'illumination globale, à cause du caractère disparate et non continu du feuillage. En suivant l'idée de représenter un ensemble de primitives (feuilles ou branches) par paquet intégrant les aspects géométriques et photométriques, nous avons conçu deux nouvelles représentations.

La première, destinée au rendu haute-qualité, est basée sur le calcul analytique du modèle d'illumination global d'une géométrie représentant un rameau d'aiguilles de conifère. En nous servant des connaissances a priori concernant la distribution géométrique des aiguilles, nous avons mis au point une hiérarchie de trois *shaders* capables de représenter à une échelle donnée les effets cumulés des niveaux plus fins, sans avoir à les échantillonner, et en tenant compte de l'auto-ombrage et de la visibilité. Le caractère analytique de ces *shaders* permet à la fois d'accélérer considérablement les temps de calcul et d'obtenir des images de qualité, en particulier avec très peu d'aliassage.

La deuxième, destinée au rendu temps réel, se compose d'une hiérarchie d'images correspondant à l'échantillonnage des directions de vue et de lumière, que nous affichons à l'aide de *billboards* en interpolant les images. Nous y associons une structure de visibilité pré-calculée, basée sur des *cubes de visibilité*, pour traiter l'auto-ombrage et l'ombrage en temps interactif. Notre implémentation permet l'affichage interactif d'une forêt de 1000 arbres avec illumination, auto-ombrage, ombrage, et avec possibilité de déplacer interactivement la source de lumière.

Mots-Clés : Synthèse d'Images, Scènes Naturelles, Modèles d'Illumination, Lancer de Rayons, Niveaux de Détails, Rendu Temps Réel, Réalisme, Visibilité, Ombrage.

Abstract : This PhD thesis is dedicated to the representation of trees for high quality or real time landscapes rendering. Techniques of tree modeling give good results in terms of diversity of species and shapes. However, the multitude of polygons of their geometrical representations generates a high computing cost and important problems of aliasing at rendering. Mesh simplification in order to build levels of details is not adapted to trees since it alters global illumination and opacity, because of the discontinuous repartition of the leaves. With the idea of representing groups of primitives (leaves or branches) using packets combining geometric and photometric appearance, we have introduced two new representations.

The first, intended to high quality rendering, is based on the analytic computation of the global illumination model of a geometric branch of needles. By using the knowledge of the geometric distribution of needles, we computed analytically a hierarchy of three *shaders* able to represent effect of lower levels, without sampling and with self-shadowing and visibility. The analytic aspect of these *shaders* allows in the same time the acceleration of computing time and good image quality, particularly with little aliasing.

The second, intended to real time rendering, is based on hierarchical sets of images corresponding to the sampling of view and light directions, which we render using *billboards* by interpoling images. We associate a precomputed visibility structure, based on visibility cubes maps, for self-shadowing and shadows. Our implementation allows the interactive rendering of a forest of 1000 trees with shading, self-shadowing, shadowing and the capability of moving light position interactively.

Key words : Computer Graphics, Natural Scenes, Shaders, Ray-tracing, Levels of Details, Real-Time Rendering, Realism, Visibility, Shadowing.