Université de Montréal

Rendu interactif de détails de surface par textures 3D semi-transparentes

par

Jean-François Dufort

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la faculté des études supérieures

en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)

en informatique

août 2005

Université de Montréal
Faculté des études supérieures

Ce mémoire de maîtrise intitulé

Rendu interactif de détails de surface par textures 3D semi-transparentes

présenté par
Jean-François Dufort

a été évalué par un jury composé des personnes suivantes :

Président:	Victor Ostromoukhov
Directeur de recherche:	Pierre Poulin
Membre:	Jean Pierre David

Sommaire

Les détails de surface sont essentiels pour améliorer l'apparence de surfaces complexes. C'est pourquoi plusieurs techniques ont été développées pour faire le rendu de tels détails. La plupart des techniques actuelles imposent des limites pour garder un temps de rendu interactif comme l'impossibilité de faire le rendu de détails semi-transparents. Malheureusement, lorsque nous utilisons des représentations hiérarchiques de détails (filtrage) et des effets de lumière importants, ces effets ne peuvent plus être ignorés. Nous proposons une méthode générale pour faire le rendu de détails de surface semi-transparents en utilisant les cartes vidéo actuelles. Ces détails de surface sont stockés dans une texture 3D semi-transparente (RGBα) qui est appliquée sur une coquille formée de tétraèdres qui enveloppe un maillage triangulaire. Les tétraèdres semi-transparents sont d'abord triés par l'application avant d'être rendu dans le bon ordre par la carte vidéo. La couleur résultante de l'atténuation de lumière dans chaque tétraèdre est calculée en espace texture 3D avec un algorithme de traversée de grille de voxels. Le système final est capable de simuler à la fois des détails semi-transparents, mais aussi des détails opaques formés par des textures de déplacement ou des textures de normales appliquées sur de la géométrie procédurale ou créée par un artiste. Peu de précalculs sont faits, alors la méthode permet de déformer la texture 3D et la surface sous-jacente. Plusieurs effets de nuançage sont présentés tels que le masquage visuel, la semi-transparence, le masquage de lumière ainsi que l'émission/absorption de lumière.

Mots clefs :

textures volumiques, texture de déplacement, rendu avec matériel graphique, détails de surface, textures semi-transparentes, rendu interactif.

Abstract

Meso-structure surface details are essential to the appearance of complex surfaces. Therefore, several techniques have been introduced to display these details, but they often face limitations. One common compromise is to avoid altogether costly semi-transparency in interactive contexts. However when dealing with hierarchical representations (important to filtering) and with complex light transferts, semi-transparency must be handled. We propose a general method for rendering semi-transparent meso-structure surface details on current graphics hardware. These surface details are stored in a 3D texture (RGBα) mapped to an outer shell defined by tetrahedra extruded from the surface triangular mesh. The semi-transparent tetrahedra are first sorted on the CPU to then be rendered in correct order. The attenuated color for each ray traversing each tetrahedron is efficiently computed in the 3D texture space with a hardware voxel traversal algorithm. The resulting structure is general enough to simulate from semi-transparent surface details to opaque displacement map, lying on top of procedural or hand-crafted geometry. Because only limited precomputation is required, the method allows for surface and texture deformations, as well as 3D texture animation. Several surface shading effects are demonstrated at interactive frame rates, including visual masking, semi-transparency, self-shadowing, and emission/absorption of light.

Keywords :
volumetric textures, displacement mapping, hardware rendering, meso-structure, semi-transparent textures, interactive rate rendering.

Table des matières

Table des figures

Liste des tableaux

Remerciements

Le temps passe vite et c'est pourquoi il faut s'arrêter un peu pour penser aux gens importants sans qui je ne serais pas rendu où j'en suis.

Tout d'abord, merci à Pierre Poulin pour son soutien constant dans ce projet qui a mis ma patience à l'épreuve plusieurs fois. Sans nos discussions matinales autour d'un croissant, de son café et de mes histoires de bar, ce mémoire n'aurait jamais vu le jour. Merci pour la confiance qu'il m'a donné en fournissant de son temps pour me permettre d'évoluer dans son laboratoire.

Merci aux membres du LIGUM pour créer un environnement où il est si bon de travailler. Grâce à eux, j'ai passé deux années extraordinaires. Je pense particulièrement à Luc avec qui j'ai eu beaucoup de plaisir à discuter de projets d'infographie et programmer et Philippe qui déborde de créativité et de vitalité...

Aussi, merci à mes parents Serge et Monique. Malgré la province qui nous sépare, grâce à ses emails sur la musique, l'informatique, la photographie et la vie en général, j'ai pu recevoir un encouragement à continuer d'une personne importante. Monique, quant à elle, m'a donné un support moral et des encouragements traduits par une curiosité naïve de quelqu'un qui ne sait pas programmé un vidéo.

Je veux souligner le plaisir que j'ai obtenu à partager une passion avec mes amis et collègues du Cat's Corner. Avec eux, j'ai obtenu des moments de détente entouré de musique et danse swing. Merci à Josée-Anne, Gaspard, Ann, Fred, Carl, et toute la gang du studio.

J'aimerais finalement remercier Alla Sheffer et Baining Guo pour m'avoir fourni des modèles 3D et des textures pour tester ce qui a été développé dans ce projet ainsi que le CRSNG pour son soutien financier.

Remarques

Terminologie

La correspondance entre les termes anglais et français se retrouve dans le glossaire à l'annexe A.

Droits d'auteur

Certaines images de ce document ont pu être tirées d articles publiés dans des journaux ou des conférences scientifiques. La provenance des images est clairement indiquée dans les figures et la référence au travail est également fournie.

Voici la note de droits d auteur pour les travaux tirés d une publication d ACM :

Voici la note de droits d'auteur pour les travaux tirés de Graphics Interface :

Voici la note de droits d'auteur pour les travaux tirés de Eurographics :

Chapitre 1

Introduction

Les maîtres sont ceux qui nous montrent ce qui est possible dans l'ordre de l'impossible.

Paul Valéry

En infographie, l'étape finale du passage de la description de la scène (objets, matériaux, lumières et caméra) à l'image s'appelle le rendu. Pour obtenir un rendu qui peut presque tromper l'oeil, un très grand temps de calcul est souvent nécessaire (ordre de minutes à heures). Pourtant, de plus en plus, des applications nécessitant un rendu rapide apparaissent. Le jeu vidéo est une industrie générant plus de profits que les entrées au cinéma et le système de rendu du jeu doit fournir environ 60 images et plus par seconde. Des applications de simulation sont développées pour introduire un individu dans un environnement immersif et tester ses capacités à réagir. L'interaction entre l'usager et l'environnement empêche de tout précalculer. Les médecins ont accès à des systèmes pour explorer des données internes sur le corps de leurs patients, qui doivent être efficacement affichées. Même l'architecture a des besoins dans ce champ d'intérêt puisqu'avec un système de rendu en temps réel, on permet à un usager de modifier la décoration d'une pièce existante et de visiter virtuellement un appartement. Le rendu rapide est primordial pour assurer un sentiment d'immersion. C'est pourquoi une série de techniques et d'algorithmes sont développés dans le but d'accélérer le rendu pour obtenir des images en temps réel ou tout au moins interactif.

Le domaine du rendu en temps réel a considérablement progressé depuis quelques années grâce en partie à l'avancement de la technologie reliée aux cartes vidéo. Il existe

maintenant des techniques efficaces capables de produire des images de bonne qualité en un temps interactif et parfois même en temps réel.

Les détails de surface contribuent de façon importante au rendu photo-réaliste. Dans le passé, les applications en temps réel se contentaient d'un nuanceur [1] de Phong qui donne une apparence lisse et plastique aux objets d'une scène virtuelle. Ajouter des petits détails géométriques à la surface des objets en augmentant le nombre de triangles réduit d'autant la vitesse de rendu. Aujourd'hui, la recherche met le focus sur l'implémentation de méthodes de rendu en temps réel pour représenter des fonctions de réflectance (BRDFs) plus complexes, ainsi que sur l'ajout de détails de surface qui sortent de l'échelle microscopique des BRDFs. Ce niveau de détails crée une représentation intermédiaire des détails visuels à la surface d'un objet virtuel, réduisant du même coup les artefacts de sous-échantillonnage et le nombre de triangles traités par la carte vidéo.

Les représentations courantes de détails de surface incluent les textures 2D, les cartes de déplacement et les textures 3D. Bien que l'utilisation de textures dans une scène virtuelle soit un ajout majeur au réalisme, les textures 2D sont limitées de par leur nature bidimensionnelle et le rendu de détails de profondeur devient impossible (silhouettes, masquage visuel, etc.). Avec un coût de traitement plus grand, ces problèmes peuvent être en partie contournés, mais la géométrie soutenant la texture reste inchangée, ce qui peut réduire le réalisme.

En utilisant les cartes de déplacement, la géométrie de soutien est déplacée pour créer des silhouettes et une apparence plus réaliste. Cependant, cette augmentation de réalisme vient au coût d'ajouter une quantité importante de petits triangles à la scène, ce qui limite les performances et diminue la capacité de faire du rendu en temps réel. De plus, seuls les détails représentables par une carte de hauteur sont reproductibles avec les cartes de déplacement. Certains détails comme la fourrure, les tissus ou les arbres sur une colline ne peuvent pas être représentés par une carte de hauteur.

Ces détails peuvent être simulés par une structure plus générale, d'où un besoin pour des textures 3D. Le rendu de textures 3D opaques est un domaine très à la mode depuis quelques années, mais les textures 3D semi-transparentes ont souvent été délaissées. Le traitement supplémentaire qu'elles requièrent décourage la plupart des chercheurs à

[1] *shader*

développer une méthode pour un rendu rapide d'une telle structure. Pourtant, la simple opération de filtrage d'une texture 3D pour réduire les artefacts de sous-échantillonnage transforme une texture 3D opaque en texture 3D semi-transparente, là où des voxels pleins sont combinés à des voxels vides. Comme les détails de surface sont traités en 3D, cette représentation permet un rendu plus facile de certains effets comme la semi-transparence, les ombrages et le masquage visuel. Il devient donc clair que la possibilité de faire le rendu de détails de surface semi-transparents est une nécessité pour un rendu de qualité de certaines scènes.

Dans ce mémoire, nous présentons une implémentation d'une technique de rendu de textures 3D semi-transparentes sur la carte vidéo. Nous avons utilisé notre technique pour faire le rendu de textures semi-transparentes en incluant des effets tels que l'absorption de lumière, l'émission de lumière, des ombres projetées et un rendu avec filtrage pour éviter les artefacts de sous-échantillonnage. Elle permet aussi de faire le rendu dans le cas particulier où la texture est opaque, reproduisant des résultats similaires à des techniques déjà existantes. L'algorithme est simple, général et peut être étendu à d'autres applications. Par exemple, un module de textures procédurales dynamiques (explosions, fumée, etc.) se grefferait aisément à notre technique.

À partir d'un maillage triangulaire, nous construisons un nouveau maillage composé du maillage de base sur lequel nous posons des prismes. Chaque triangle de base supporte un prisme dans le nouveau modèle. Cette coquille est le volume englobant de la texture 3D apposée sur le maillage. Il existe une bijection entre l'espace dans la coquille et l'espace texture 3D dans le prisme. En utilisant le matériel graphique et sa fonctionnalité de programmation, nous arrivons à effectuer le rendu en temps interactif du nouveau modèle avec la texture 3D. Pour traiter correctement les textures semi-transparentes, il faut ordonner le rendu à l'image. C'est pourquoi nous trions les prismes dans un ordre derrière à devant. La couleur finale d'un pixel de l'image est l'accumulation ordonnée des contributions de tous les prismes visibles dans ce pixel. L'intégration de la couleur avec l'opacité se fait dans un nuanceur de pixels.

Notre approche rend correctement des silhouettes avec relief comme les techniques récentes de cartes de déplacement sur les cartes vidéo (voir chapitre 2), mais permet aussi l'utilisation de textures semi-transparentes. Puisque l'algorithme ne suppose presqu'aucun précalcul, notre méthode permet l'animation de la géométrie de base, la déformation

de la coquille qui contient les détails à la surface des objets, la modification de la paramétrisation de la surface et l'animation de la texture 3D. Ensuite, notre système nous a permis d'explorer certaines directions sur le rendu avec les cartes vidéo. Finalement, notre système offre un environnement de rendu dans lequel le filtrage correct de textures 3D pourrait être inséré.

Ce mémoire est organisé de la façon suivante. Dans le chapitre 2, nous ferons une revue de littérature sur les travaux reliés à notre projet. Ensuite, nous expliquerons l'algorithme permettant d'appliquer la texture 3D sur la géométrie (chapitre 3). Dans le chapitre 4, nous discuterons du tri des primitives graphiques, puis le chapitre 5 sera consacré à l'algorithme permettant d'en faire le rendu en discutant des détails de l'implémentation sur la carte vidéo. Nous présenterons nos résultats dans le chapitre 6 avant de conclure au chapitre 7.

Chapitre 2

Travaux antérieurs

La grande histoire véritable est celle des inventions.

Raymond Queneau

Après la modélisation de la géométrie, l'augmentation du réalisme passe par une série de détails fins à surface des objets. L'objectif du photo-réalisme est d'augmenter la richesse des surfaces dans les images virtuelles dans le but de rendre l'image plus parlante, mais aussi parce que les surfaces naturelles sont complexes et l'usure complexifie les surfaces manufacturées. Nous proposons ici une classification des différents algorithmes de détails de surface en situant notre algorithme dans cette évolution (Figure 2.5).

2.1 2D

Les premières techniques présentées ne modifient pas l'aspect géométrique de la surface, mais en changent son apparence par l'apposition d'une texture qui modifie les propriétés de réflectance de l'objet.

2.1.1 Textures plaquées

Les textures de couleur 2D sont les plus utilisées pour représenter les détails à la surface d'un objet [Hec86]. Une texture est une image composée d'éléments appelés texels. La technique consiste à apposer, comme un collant, une image sur un objet (Figure 2.1). Le premier avantage des textures est qu'elles sont faciles à utiliser et très

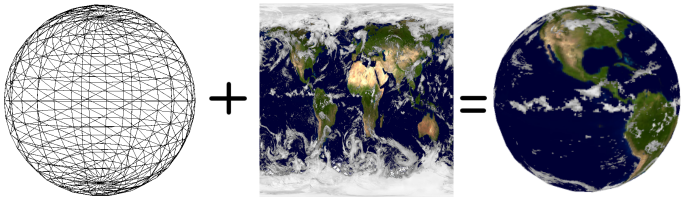

FIG. 2.1 – La texture est plaquée sur la sphère. Image de la terre tirée de la banque d'images du NOAA [Lib].

intuitives. Aussi, les cartes vidéo permettent d'utiliser les textures avec accélération matérielle depuis déjà longtemps. Concrètement, à chaque sommet d'un maillage triangulaire sont associées des coordonnées de texture. Lors de l'étape de la rastérisation, les coordonnées de texture de chaque sommet du triangle sont interpolées pour chaque pixel et la couleur de la texture, le texel, aux coordonnées interpolées est copiée dans le pixel. Même si elles ajoutent de la richesse à très faible coût, ces textures souffrent de leur nature 2D puisqu'elles ne permettent pas les effets dépendants de la direction de la lumière ou du point de vue.

Pour créer une illusion de 3D, des effets de rendu dépendants de la lumière ou du point de vue ont été ajoutés. La technique la plus courante est l'utilisation de placage de reliefs [Bli78]. Chaque pixel d'une texture 2D représente une déviation de la normale à la surface de l'objet. Lors de l'étape du rendu, la normale originale de l'objet à un point est modifiée en fonction de la texture de reliefs avant de faire l'ombrage. Cette technique est simple et donne l'illusion de reliefs à la surface des objets (Figure 2.2) puisque la réflexion de la lumière est modulée en fonction de la carte de reliefs et que l'orientation de la normale est un des facteurs déterminants dans la réflexion. Cependant, le placage de reliefs reste une technique fondamentalement 2D et elle ne permet pas le masquage visuel ni les ombres projetées par les détails sur la surface ou sur d'autres surfaces. De plus, les ombres projetées sur la surface avec reliefs ne sont pas déformées.

Dans la même veine, le placage de reliefs avec parallaxe [Wel04] ajoute l'illusion d'un masquage visuel au placage de reliefs classique. Cette technique s'insère bien dans les autres techniques classiques. En fait, la différence réside dans la lecture de la texture où le texel sélectionné dépend à la fois des coordonnées de texture et du vecteur de vue à la surface de l'objet. En modulant les coordonnées de texture par le vecteur de vue, un effet de parallaxe est ajouté au placage de reliefs (Figure 2.2), augmentant le niveau de réalisme obtenu. Avec l'arrivée des nuanceurs de sommets et de pixels sur

les cartes vidéo, le placage de reliefs avec parallaxe est très simple à ajouter à un engin graphique en temps réel. La Figure 2.2 montre la différence entre un placage de texture conventionnel, un placage de reliefs et l'ajout de l'effet de parallaxe. La modulation de la normale avec une carte de reliefs ajoute des zones plus sombres dans la réflexion de la lumière dans les images du bas de la figure car la surface n'est plus alignée vers la lumière là où la normale est modulée. Il est important de noter que ces effets de reliefs sont simulés. En effet, nous n'observons aucune déformation des silhouettes. L'ajout de la parallaxe par modulation des coordonnées de texture par le vecteur de vue améliore l'effet de relief. Cependant, il s'agit encore d'un effet simulé puisque les détails restent planaires comme le démontrent les images de droite dans la figure. Une limite directe de cette simulation est l'impossibilité de produire des ombres projetées par les détails (les bosses ne projettent pas d'ombres dans la figure). Malgré ces limites, en combinant un placage de reliefs avec un effet de parallaxe, nous obtenons un effet simulé de reliefs amélioré à un faible coût sur les cartes vidéo actuelles.

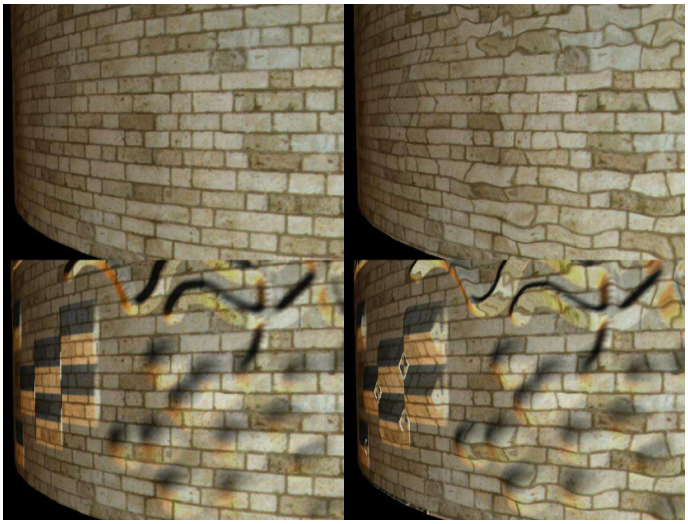

FIG. 2.2 – (gauche haut) placage de texture ; (gauche bas) placage de textures et reliefs ; (droit haut) placage de texture avec parallaxe ; (droit bas) placage de texture avec parallaxe et reliefs. Le placage de reliefs (en bas) assombrit les zones où la normale modulée puisque la nouvelle normale n'est plus alignée avec la source de lumière. La parallaxe (à droite) donne l'impression que les zones de reliefs sont extrudées de la surface. (tiré de [Wel04]).

Même si l'effet de parallaxe est intéressant, il s'agit toujours d'une illusion car il

n'y a toujours pas de véritable masquage visuel, ni d'ombres projetées entre les détails de surface. Les BTFs (*Bidirectional Texture Function*) intègrent ces effets. Une BTF [DvGNK99] représente une fonction de réflectance sur un domaine spatial (un rectangle de texture 2D) pour un ensemble de directions de vue et de lumière. La représentation la plus courante est une collection d'images d'un matériau prises sous différents angles de vue et de lumière (photographies [DvGNK99, MMS+05] ou rendu haute qualité [SBLD03], Figure 2.3). En interpolant correctement entre les différentes images (interpolation trilinéaire $(\theta, \phi)_{\text{entrant, sortant}}$ et $(x, y)_{\text{pixel}}$), on peut estimer l'apparence de la surface pour toutes les directions de lumière et de vue (Figure 2.4). Comme la technique repose sur un échantillonnage dense de l'apparence du matériau, une BTF est souvent très coûteuse en mémoire. De plus, la méthode d'acquisition d'une BTF ne tient pas en compte la courbure de l'objet sur lequel la BTF sera appliquée. Ainsi, l'application est limitée aux modèles avec faible courbure si la cohérence dans les détails veut être conservée. Le filtrage et les représentations multiéchelles deviennent tout aussi difficiles à réaliser malgré quelques essais [MCT+05]. Comme les images d'une BTF sont souvent très redondantes, avec quelques approximations et de la compression, certains travaux récents utilisent avec succès les BTFs pour des applications en temps réel [Sch04, KSS+04].

FIG. 2.3 – Dans une BTF, une série d'images sont prises sous différents angles de vue et de lumière. Image tirée de [MMS+05].

2.1.2 Textures précalculées

Les techniques présentées dans la section précédente utilisent des textures 2D de façon simple. Sans précalcul, elles arrivent à simuler un certain nombre d'effets, mais ne peuvent pas en reproduire d'autres à moins d'avoir recours à des techniques coûteuses

FIG. 2.4 – Même si certains effets comme le masquage de lumière sont pris en compte, les BTFs restent 2D comme le montre l'absence de silhouettes détaillées. Image adaptée de [MMS+05].

en mémoire (e.g. BTF). Pour réduire le coût des BTFs, nous pouvons développer un modèle sur l'apparence du matériau. En utilisant les directions de vue et de lumière comme variables, on factorise la couleur de chaque texel en un polynôme [MGW01]. Les coefficients du polynôme sont stockés dans la texture à la place de la BTF elle-même. À l'étape du rendu, on évalue la valeur du polynôme en fonction des directions de vue et de lumière pour retrouver une estimation de l'apparence du matériau au texel voulu. Plus récemment, il a été montré qu'une BTF peut aussi être factorisée dans un modèle non-linéaire en employant des tenseurs [VT04] plutôt qu'une approche du type analyse des composantes principales [1] (PCF). L'approche PCF combine dans une matrice tous les paramètres qui font changer l'apparence d'une BTF. En utilisant un ensemble non-linéaire de fonctions de base, la BTF est séparée en plusieurs facteurs (vecteur de vue, vecteur de lumière). Grâce à cette séparation, une analyse séparée des facteurs est possible.

Une autre stratégie précalcule la visibilité aux différents points des détails de surface. En supposant des détails représentés par une carte de hauteur, en chaque point de ces détails, on détermine l'angle d'élévation nécessaire pour que la lumière soit visible

[1] *principal component factors*

[Max88]. Il est aussi possible d'encoder les directions radiales (azimuth). Après placage de cette texture, appelée carte d'horizon, la comparaison entre l'angle d'élévation de la lumière à l'angle précalculé permet de déterminer si les détails sont dans l'ombre ou non.

On peut précalculer plus d'information dans des textures. En calculant la visibilité entre les points de surface, nous pouvons même simuler des effets de réflexion et diffusion de lumière par les détails de surface [HDKS00]. Dans des textures, l'algorithme stocke en précalcul la surface visible en un ensemble de points pour un ensemble de directions. Avec cette information en main, cette technique simule la diffusion de lumière qui réfléchit plusieurs fois dans les détails. Ainsi, la luminosité globale de l'objet est plus réaliste qu'avec un placage de reliefs ordinaire.

2.1.3 Filtrage de textures

Pour bien comprendre le filtrage de textures, il faut voir une texture comme un signal 2D discret où les texels sont des intensités représentées par des couleurs. Lorsque nous appliquons une texture sur un modèle éloigné de la caméra, il arrive que la résolution de l'image soit trop petite pour bien échantillonner tous les détails de la texture. La fréquence d'échantillonnage de l'algorithme de rendu devient trop faible par rapport aux hautes fréquences de la texture, résultant en du bruit dans l'image. Dans l'exemple simple d'une texture représentant un échiquier noir et blanc plaquée sur un plan éloigné de la caméra, un pixel de l'image sera blanc alors que son voisin pourrait être noir et ce de façon désordonnée. Puisque la règle de Nyquist n'est pas respectée, du bruit apparaît dans l'image.

Idéalement, pour éviter ce bruit, il faudrait que la résolution de l'image soit suffisamment grande pour capturer tous les texels de la texture ce qui serait coûteux en temps de rendu et en mémoire vidéo. Une autre solution consiste à sur-échantillonner la texture pour respecter la loi de Nyquist. L'algorithme parfait consiste donc à calculer la contribution de tous les texels qui sont visibles dans un pixel de l'image pour trouver la couleur finale à afficher. Dans le cas où plusieurs cases de l'échiquier sont visibles dans un seul pixel, la couleur finale serait grise plus ou moins foncée dépendamment du nombre de cases blanches plutôt que d'avoir un pixel blanc ou noir presque aléatoirement.

Évidemment, cet algorithme est impraticable puisqu'il serait beaucoup trop coûteux.

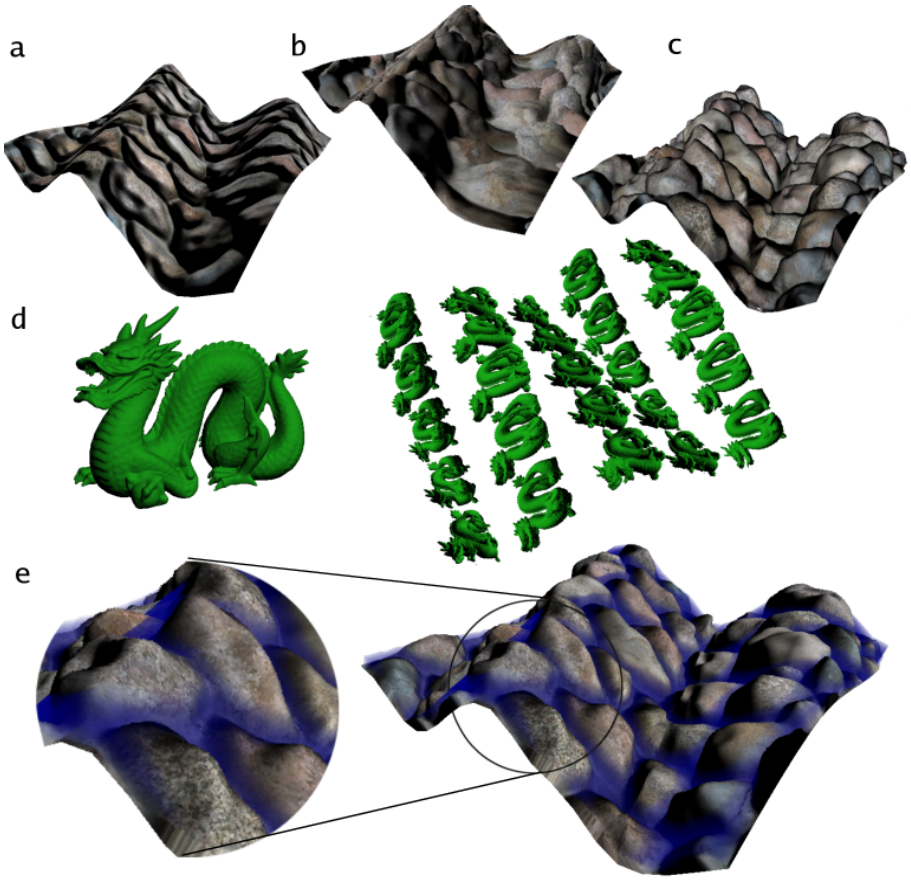

FIG. 2.5 – (a) Une carte de relief et (b) une carte de relief augmentée d'une profondeur ne donnent pas des silhouettes correctes comparativement à (c) une carte de déplacement. Notre technique donne un effet de masquage visuel et des ombres projetées par les détails de surface. N'importe quelle géométrie peut être utilisée pour les détails de surface (d) incluant celle qui ne peut pas être représentée par une carte de hauteur. (e) Des textures semi-transparentes peuvent produire des effets visuels importants tels que des silhouettes détaillées et des combinaisons de couleurs indiquées dans la zone bleue de la texture.

Une autre solution plus praticable consiste à filtrer le signal pour enlever les hautes fréquences d'avance pour éviter le bruit de sous-échantillonnage à la reconstruction du signal plus tard. Le filtrage est une opération mathématique faite normalement en précalcul durant laquelle un filtre est passé en convolution sur la texture pour enlever les hautes fréquences. Plutôt que de calculer la contribution des texels à l'étape du rendu, le filtre combine les texels voisins dans la texture pour trouver leur couleur moyenne (Section 5.3.2). Le poids accordé à chaque texel dépend du filtre choisi. En utilisant un

filtre boîte comme il est souvent fait en rendu par carte vidéo, la limite de cette opération est de faire la moyenne de tous les texels de la texture pour donner une texture à une seule couleur correspondant à la moyenne de toutes les couleurs de la texture.

L'opération de filtrage sur une texture de couleurs est valide puisque la couleur de chaque texel contribue à l'apparence finale du pixel de façon linéaire dans l'étape du rendu. Avec un filtre boîte simple, chaque texel contribue à la couleur finale avec un poids correspondant à la fraction de sa taille sur la taille d'un pixel après projection dans le pixel. Cependant, la même idée ne peut pas s'appliquer directement si d'autres informations sont stockées dans la texture comme des normales pour un placage de reliefs ou des effets dépendants de la vue comme l'apparence stockée dans une BTF.

En effet, les normales ne peuvent pas être combinées de la même façon que les couleurs puisque la normale stockée dans un texel ne contribue pas de façon linéaire à l'apparence d'un pixel. Des fonctions cosinus apparaissent dans l'équation d'illumination. En général, faire la moyenne des normales avant de calculer la réflectance n'est pas équivalent à calculer la réflectance de chaque texel séparément et d'en faire la moyenne après

$$\frac{(\hat{N_1} \cdot \hat{L} + \hat{N_2} \cdot \hat{L})}{2} \neq \frac{0.5 * (\hat{N_1} + \hat{N_2})}{|0.5 * (\hat{N_1} + \hat{N_2})|} \cdot \hat{L}$$

où $\hat{N_1}$ et $\hat{N_2}$ sont des normales dans la texture et $\hat{L}$ est le vecteur de lumière normalisé. Pour filtrer des normales, il faut avoir recours à des distributions de normales [Fou92].

Le même problème arrive dans le filtrage des BTFs. Lors du rendu, l'algorithme sélectionnera à chaque pixel de l'image de la BTF qui correspond le plus à la configuration direction de vue et direction de lumière. Quand l'objet est loin, deux pixels voisins peuvent avoir des directions de vue et de lumière très différentes, alors il est difficile de combiner les images de la BTF en préfiltrage [MCT+05].

Finalement, ce problème survient dans l'utilisation de textures dont les informations ne sont pas combinées de façon linéaire comme dans les cas où l'apparence dépend de la visibilité ou de la direction de lumière.

2.2 Carte de déplacement

Bien que les techniques précédentes offrent des solutions intéressantes au problème des détails de surface en temps réel, elles souffrent toutes de leur nature 2D. Elles limitent les objets à des silhouettes lisses qui ne modélisent pas les reliefs à la surface,

et les effets sont tous limités à la surface de l'objet. Une carte de déplacement [Coo84] est une texture qui stocke à chaque texel une valeur de déplacement (hauteur) du point de surface. Cette technique permet des détails réellement 3D.

2.2.1 Polygonalisation adaptative

Après placage de la carte de déplacement sur la géométrie, chaque point de géométrie est déplacé en fonction de la valeur dans la carte, permettant des effets de masquage de lumière et visuel. Cependant, pour bien reproduire les détails représentés dans la carte, ces points doivent former un maillage suffisamment fin après déplacement sinon certains détails sont perdus. Cet ajout de beaucoup de petits triangles a longtemps limité les cartes de déplacement au rendu de haute qualité tel que dans le standard Renderman [Ups90].

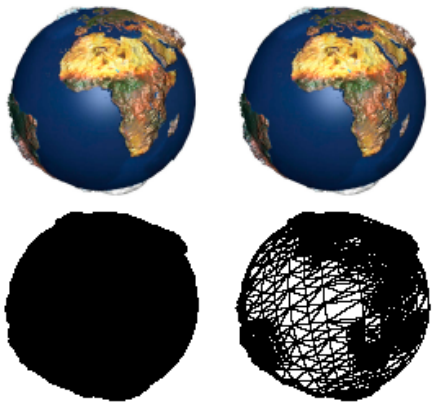

FIG. 2.6 – Comparaison entre une polygonalisation (gauche) uniforme et (droite) adaptative. Des triangles sont ajoutés seulement là où nécessaire. La polygonalisation uniforme est tellement dense pour donner la même qualité de résultats que l'affichage des triangles en fil de fer ne produit qu'une une tache noire. Image tirée de [MM02].

Pour faire le rendu d'un modèle augmenté d'une carte de déplacement en temps réel, une première approche ajoute des triangles à la surface du maillage de façon adaptative [DH00, MM02]. À partir de certaines règles comme la présence d'une silhouette ou la visibilité, des triangles sont ajoutés seulement là où nécessaire (Figure 2.6), allégeant le rendu et offrant une possibilité de rendu sur une carte vidéo.

Finalement, la nouvelle génération de cartes vidéo permet de lire des textures à l'étape de la géométrie dans le pipeline (Section 5.2.1), ce qui permet d'utiliser les

cartes de déplacement de manière efficace en matériel.

2.2.2 Tracer de rayons

Une autre solution consiste à tracer un rayon dans le volume englobant de la carte de déplacement [HEG04]. Des prismes sont érigés à la surface de l'objet pour accueillir la carte de déplacement afin de permettre une généralisation aux surfaces courbes. Ensuite, dans un nuanceur de pixels, une recherche linéaire est faite pour trouver l'intersection entre le rayon de vue et la surface déplacée.

La carte de déplacement est une texture augmentée d'une hauteur [NBM00]. Une recherche binaire peut aussi être effectuée dans un nuanceur de pixels. De plus, en traçant le rayon directement dans la carte de déplacement 2D [YJ04, PNC05], la technique ne nécessite plus de prismes à la surface de l'objet. La méthode suppose que le déplacement de la carte est fait vers l'intérieur du maillage. Dans le nuanceur de pixels, le rayon de vue est projeté sur le plan de la carte de déplacement. En prenant un certain nombre d'échantillons dans cette carte 2D, l'intersection entre le rayon de vue et les détails de surface est trouvée puis la bonne couleur est retournée. De plus, il a été montré qu'on peut faire le rendu d'objets complexes en déformant des cartes de déplacement sur un ensemble de surfaces planaires [NBM00]. Cette approche similaire mais différente de celle présentée ci-haut offre de meilleures performances, mais limite l'application à des surfaces planaires. Le traitement des surfaces courbes ne fonctionne que dans des cas particuliers.

2.2.3 Carte de déplacement dépendante de l'angle de vue

Un VDM (*View-dependent Displacement Map*) [WWT$^+$03] est un ensemble de cartes de déplacement précalculées à partir d'un ensemble de directions de vue. Ceci permet de réduire la quantité de géométrie envoyée à la carte vidéo, mais pour obtenir une bonne interpolation pour toutes les directions de vue, un échantillonnage dense est nécessaire. Au rendu, la direction de vue courante est utilisée pour trouver quelle carte de déplacement est la plus appropriée dans la collection de cartes de déplacement précalculées. Ensuite, les pixels déterminés à l'extérieur de la surface déplacée peuvent être éliminés, ce qui donne des silhouettes en relief. Cette technique produit un masquage visuel, mais certains problèmes peuvent survenir lorsque la surface sur laquelle on applique la carte de

déplacement devient fortement courbe (Figure 2.7).

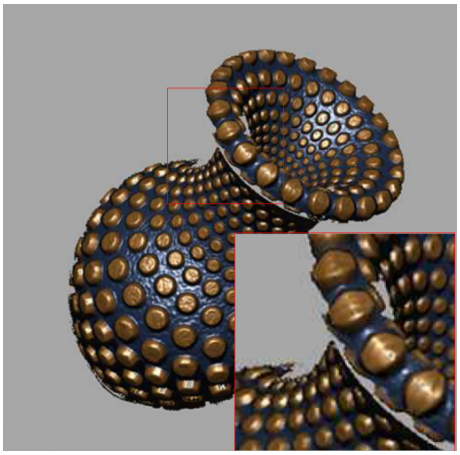

FIG. 2.7 – Problèmes avec objets courbes pour les cartes de déplacement dépendantes de l'angle de vue. Image tirée de [WWT+03].

2.3 Textures 3D

Bien que les techniques de carte de déplacement font le rendu de silhouettes déplacées, toutes les techniques mentionnées précédemment ne fonctionnent qu'avec des détails de surface représentables par une carte de hauteur. Ce sont des techniques limitées à une représentation 2.5D. Cependant, tous les types de détails ne peuvent pas être représentés par une carte de hauteur : fourrure, tissus, gazon, arbres, peau, etc. Les textures 3D peuvent stocker n'importe quelle information dans un domaine totalement 3D.

2.3.1 Textures procédurales

Une contribution majeure des textures 3D a été apportée en introduisant les célèbres fonctions de bruit et de turbulence de Perlin [Per85, Per02]. Ces fonctions simulent un phénomène aléatoire tout en gardant une certaine structure lisse et manipulable pour obtenir une grande variété d'effets. Un grand nombre de matériaux ont été simulés à partir du bruit de Perlin. Bois, marbre, fumée et nuages en sont quelques exemples [EMP+02] (Figure 2.8). Le bruit de Perlin définit une fonction aléatoire dans un domaine volumétrique. Pour appliquer la texture sur un objet, il suffit d'évaluer la fonction aux coordonnées 3D (transformées ou non) des points de surface.

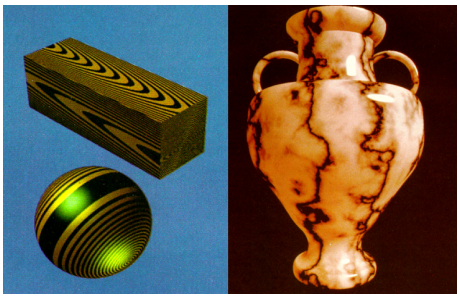

FIG. 2.8 – Textures procédurales. (gauche) Bois. Image tirée de [Pea85]. (droite) Marbre Image tirée de [Per85].

Récemment les possibilités de programmer les processeurs graphiques permettent d'utiliser le bruit de Perlin dans un rendu en temps réel [Per04, Gre05].

2.3.2 Textures volumiques et rendu de qualité

L'idée de plaquer une texture 2D sur un maillage triangulaire peut être généralisée en 3D. À la place de calquer une image sur l'objet, on plaque des unités de volume. Comme en 2D, une texture stocke une image décomposée en texels à plaquer sur un modèle, en 3D, une texture stocke des détails de surface dans des voxels à plaquer. Un exemple serait un modèle simple de stationnement sur lequel on plaque de petites voitures. Les voitures ne sont pas composées de triangles, mais bien stockées dans une texture 3D qui sera plaquée et répétée (Figure 2.9). L'idée de faire un rendu par lancer de rayons à travers une texture 3D a été populaire [KK89, Ney98, PBFJ05]. Le rayon est tracé dans la scène et après une intersection avec la coquille qui englobe la texture 3D sur le modèle, une étape de tracer de rayon est réalisée dans les texels 3D.

Plus récemment, les STFs (*Shell Texture Function*) [CTW+04] ont été introduites. Une simulation d'illumination globale est effectuée sur un échantillon d'une texture 3D. En variant les directions de vue et de lumière et en stockant en précalcul le parcours complexe de la lumière dans l'échantillon de texture, on obtient un échantillon de l'apparence de la texture 3D sous différents angles de vue et de lumière. Pour obtenir un rendu fidèle sous n'importe quel angle de vue ou de lumière, un échantillonnage dense de la fonction de réflectance doit être effectué. Les données non-compressées pour une STF peuvent dépasser 200 Mo d'espace mémoire. À l'étape du rendu, les directions de vue et de lumière sont utilisées pour interroger la structure précalculée et extraire l'apparence du matériau sans resimuler tout le parcours de la lumière dans la texture 3D. Le coût

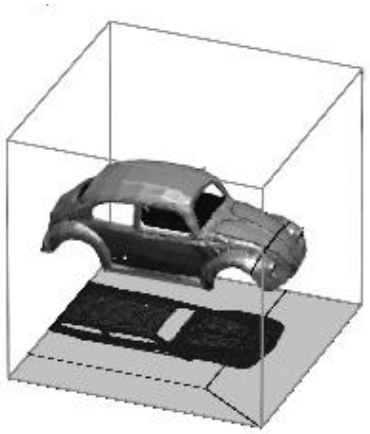

FIG. 2.9 – Exemple de texture 3D de voiture. Image tirée de [Ney96].

en temps de précalcul pour cette technique est très long. En effet, le rendu par lancer de photons utilisé pour la construction de la STF peut nécessiter plusieurs minutes par image. De plus, c'est une technique utilisée en rendu différé de haute qualité.

2.3.3 Textures volumiques et rendu par couches

Pour faire un rendu de textures volumiques en temps réel, Meyer et Neyret [MN98] couvrent la surface d'un maillage triangulaire avec une série de prismes et font le rendu de chaque prisme en les découpant avec des triangles parallèles texturés (Figure 2.10). Chaque triangle correspond à une tranche dans la texture 3D. Pour éviter de voir entre les tranches lorsqu'à angle rasant, le nombre de tranches peut être augmenté. Une autre solution consiste à fournir en précalcul un ensemble de tranches selon quelques orientations et choisir la plus perpendiculaire à l'angle de vue lors du rendu. Malheureusement, ceci peut introduire des effets de discontinuités lorsque le logiciel change d'ensemble de tranches.

Pour éviter ces effets de discontinuités, des tranches alignées avec le point de vue courant sont générées automatiquement à l'étape du rendu [KS01]. La nouvelle géométrie introduite devait alors être transférée de la mémoire principale à la mémoire vidéo à chaque image. Aujourd'hui, ces tranches peuvent être générées directement par la carte vidéo avec un nuanceur de sommets [DN04].

À la place de générer les tranches prisme par prisme, Lensch *et al.* [LDS02] génèrent des plans qui coupent tout l'objet (ou la scène) dans un ordre de derrière à devant. Un nuanceur de sommets calcule l'intersection entre les plans de coupe et les prismes plaqués

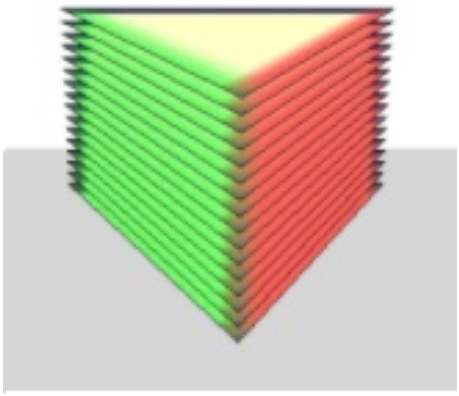

FIG. 2.10 – Chaque prisme est découpé en un ensemble de polygones parallèles. Image tirée de [DN04].

sur le maillage de base. Le rendu dans l'ordre permet des textures semi-transparentes. Le nombre de plans dépend de la complexité de l'objet rendu (nombre de prismes) et un très grand nombre de plans peut être nécessaire pour faire un rendu de tous les détails contenus dans la texture 3D. Avec 1000 plans, pour le rendu d'une scène de terrain, ils arrivent à un taux de 4 images par seconde sur une GeForce4. Une des raisons principales d'un bon taux de rafraîchissement malgré un grand nombre de plans et de prismes est l'utilisation de faux sommets. Tous les prismes envoyés à la carte vidéo ont 6 sommets et un nuanceur de sommets déplace les sommets sur les points d'intersection avec les plans de coupe.

2.3.4 Textures volumiques et approches récentes

Les cartes de déplacement généralisées [WTL+04] diffèrent des stratégies de découpage pour le rendu. Dans une étape de précalcul, une fonction à cinq dimensions est calculée en lancer de rayons dans la géométrie représentée par la texture 3D. Cette fonction reconstruit la visibilité des points dans la texture 3D pour un ensemble de directions. Le modèle sur lequel la texture 3D est appliquée est couvert de prismes. À l'étape du rendu, l'intersection entre le rayon de vue et le prisme sert de point d'entrée pour le rayon équivalent en espace texture. La fonction 5D précalculée détermine quel point de la texture 3D ce rayon intersecte en premier. Si rien n'est intersecté, le segment de rayon dans ce prisme est laissé transparent. Même si la structure de visibilité est compressée

(2-4 Mo for une texture typique $64 \times 64 \times 32$) et qu'elle offre des performances en temps réel (46 images par seconde pour un modèle de 11 000 polygones), la technique reste valide uniquement pour les textures 3D opaques et statiques.

Plus récemment, une technique similaire à la carte de relief (Section 2.2.2) a été présentée où le parcours en espace texture est accéléré par un parcours de voxels qui saute les voxels vides [Don05]. Cette technique est limitée à des modèles planaires et des cartes de déplacement, mais l'auteur stipule qu'elle est généralisable aux textures 3D et aux objets courbes. Cependant, comme aucune intégration d'absorption de lumière n'est effectuée sur le rayon, la technique ne s'applique pas aux textures semi-transparentes.

2.4 Rendu volumétrique

Le rendu de données volumétriques (médicales, atmosphériques, simulation scientifiques, etc.) a inspiré beaucoup de travaux dans le domaine des textures 3D. Trois approches sont fréquemment utilisées pour faire le rendu de données volumétriques en temps interactif. La première consiste à découper avec des plans le volume englobant des données [EKE01, KPHE02]. La deuxième consiste à lancer des rayons dans le volume englobant des données et utiliser un algorithme de traversée de grilles 3D pour accumuler l'opacité de chaque rayon [Lev90, AW87, KW03]. Finalement, la projection de tétraèdres [ST90] subdivise les données volumétriques en tétraèdres. À l'étape du rendu, chaque tétraèdre est divisé en triangles dépendamment de sa forme après projection dans l'image. Les couleurs à chaque sommet de ces triangles sont retrouvées en fonction d'une table d'absorption précalculée qui dépend de l'opacité pré-intégrée des données et de l'épaisseur des tétraèdres. La couleur est interpolée entre chaque sommet des triangles pour donner l'image finale. Cet algorithme a été implanté correctement avec les cartes vidéo [WMFC02, WKE02, KQE04]. Pour avoir des performances en temps interactif (1 à 3 images par seconde pour 200 000 tétraèdres), la projection des tétraèdres est faite dans un nuanceur de sommets pour remplacer le traitement en mémoire principale. Aussi, pour éviter le tri des tétraèdres, un modèle optique simplifié d'accumulation de couleur est utilisé et une table d'absorption est précalculée. Cette méthode est bien adaptée aux données volumétriques uniformes, mais avec beaucoup de petits détails de surface, la subdivision en tétraèdres devient coûteuse. De plus, la répétition des détails à la surface de l'objet devient difficile comparativement à une stratégie de placage de

textures 3D.

2.5 Récapitulation

Dans la grande majorité des travaux sur le rendu de détails de surface, l'aspect semi-transparent a été négligé. Pourtant, nous savons que la semi-transparence est nécessaire pour un bon nombre d'effets de rendu tels que l'absorption, le mélange de couleurs et le filtrage de textures opaques. La plupart des modèles de détails de surface présentés font des simplifications en supposant les détails opaques, représentables par une carte de déplacement ou en précalculant la visibilité, limitant les modifications apportables à la scène après plaquage des détails de surface. D'un autre côté, les travaux en rendu de données volumétriques servent à développer des techniques de rendu avec semi-transparence pour des données à haute résolution, baissant les performances.

Nous croyons que les avancées technologiques dans le rendu avec les cartes vidéo, surtout au niveau des nuanceurs de pixels et sommets, permettent d'explorer des solutions pour le plaquage de détails de surface combinant les avantages réunis des textures semi-transparentes plaquées sur un modèle polygonisé et la généralité des textures 3D. Dans cette optique, ce mémoire présente un travail qui nous a été inspiré par les algorithmes de rendu de données volumétriques et qui répond au problème du rendu de détails de surface semi-transparents avec des textures 3D. Notre technique reproduit des résultats déjà obtenus par des modèles simplifiés de détails de surface avec des effets comme le masquage visuel ou les ombres projetées, en plus d'autres effets qui ne sont pas possibles avec les autres techniques comme l'absorption de lumière, le mélange de couleurs par semi-transparence et les ombres volumétriques. De plus, elle offre un cadre de travail permettant le rendu d'une texture 3D quelconque avec filtrage.

Chapitre 3

Coquille

J'ai rencontré Isocèle. Il a une idée pour un nouveau triangle.

Woody Allen

La première étape de notre méthode consiste à construire une coquille autour du modèle 3D pour accueillir la texture 3D. Dans ce chapitre, nous décrivons la structure qui offre le support de cette texture. À la surface du maillage de base, les détails de surface seront répétés si nécessaire par un pavage de la texture dans la coquille enrobant le modèle 3D.

3.1 Extrusion de la coquille

Pour construire la coquille, chaque sommet de la surface est copié puis cette copie est déplacée. De façon similaire au travail de Wang *et al.* [WTL+04], chaque triangle donne naissance à un triangle homologue défini par les trois sommets ainsi déplacés. Le déplacement de base d'une longueur donnée est le long de la normale de chaque sommet. Cependant, comme la méthode ne requiert pas de précalcul supplémentaire, nous pouvons modifier la longueur du déplacement en temps réel dans le système, changer la direction de déplacement, appliquer des opérateurs de voisinage sur les nouveaux sommets et même animer le maillage sous-jacent (Section 5.3.3). Dans ce travail, nous nous sommes concentrés sur le rendu des textures 3D et non sur la modélisation de l'extrusion. Nous invitons le lecteur à consulter le travail de Peng *et al.* [PKZ04] pour une

méthode efficace de modélisation de coquilles permettant des modifications interactives en plus d'assurer la non-intersection entre les prismes.

Partant de l'observation qu'une coquille s'éloigne d'une surface, le déplacement du dessus de la coquille est géré par le gradient de la fonction de distance à la surface. Lorsqu'il est défini, le gradient de la fonction de distance pointe en direction de la normale. Le déplacement de base dans notre implémentation se fait donc en direction du gradient de la fonction de distance. Cependant, Peng *et al.* définissent une forme de gradient étendu de la fonction de distance à la surface du maillage et introduisent une fonction de distance moyenne à la surface. En combinant ces deux fonctions dans une intégrale sur les points de la surface qui sont dans le voisinage du point courant, ils déterminent une distance maximale que le déplacement de la coquille peut faire en ce point. Ainsi, la coquille déplacée ne s'auto-intersectera pas. Dans notre approche simplifiée ou dans la méthode de Peng *et al.*, le résultat de cette opération est une couche qui englobe le maillage original. En ajoutant des triangles entre les sommets originaux et les nouveaux sommets, nous fermons les prismes (Figures 3.1 et 3.3).

Pour faciliter le tri des prismes à l'étape du rendu, nous subdivisons chaque prisme en trois tétraèdres. Le tri des tétraèdres est nécessaire au rendu puisque la semi-transparence des textures nous oblige à faire un rendu de derrière à devant (Section 4.1). Cette subdivision nous assure que chaque morceau de prisme est convexe. Pour assurer une connectivité dans la coquille, il est important que deux prismes voisins soient subdivisés de manière cohérente en tétraèdres. Une règle globale qui ne dépend pas de la connectivité des tétraèdres est établie. Nous assignons à chaque sommet un indice unique dans le maillage. Si on assigne les indices (A, B, C) aux sommets de la base du prisme et les indices (A', B', C') aux sommets du triangle extrudé, nous pouvons subdiviser le prisme en trois tétraèdres (A, B, C, C'), (A, B, B', C') et (A, A', B', C'). La condition supplémentaire que $A < B < C$ nous assure d'obtenir une subdivision cohérente dans tout le maillage [HEG04]. Il suffit d'interchanger les sommets qui ne respectent pas cette condition, ainsi que leurs sommets extrudés pour finaliser la coquille.

Chaque sommet a les propriétés suivantes. D'abord, chaque sommet possède des coordonnées de texture 3D (u, v, w). Pour les sommets originaux, la coordonnée est fournie dans la paramétrisation du modèle pour laquelle nous assignons une valeur 0 à la coordonnée w. Le sommet copié et déplacé a les mêmes coordonnées (u,v), mais sa

coordonnée w est 1. La position monde (x, y, z) de chaque sommet est connue. Chaque sommet possède également un système d'axes local (tangente, normale, binormale) utile pour l'ombrage de la surface.

Nous pouvons trouver un système d'axes local de la façon suivante [FK03]. Soient les positions des trois sommets d'un triangle

$$v_0 = (x_0, y_0, z_0)$$
$$v_1 = (x_1, y_1, z_1)$$
$$v_2 = (x_2, y_2, z_2)$$

et les coordonnées de texture 2D des sommets

$$t_0 = (u_0, v_0)$$
$$t_1 = (u_1, v_1)$$
$$t_2 = (u_2, v_2).$$

Comme les trois sommets forment un plan et que leurs coordonnées aussi, nous pouvons écrire des équations de plans en fonction de u et v

$$A_0 x + B_0 u + C_0 v + D_0 = 0$$
$$A_1 y + B_1 u + C_1 v + D_1 = 0$$
$$A_2 z + B_2 u + C_2 v + D_2 = 0.$$

Grâce aux coordonnées connues dans le maillage, il nous est possible de trouver les valeurs des inconnues de ces équations et définir les coordonnées objet en fonction des coordonnées de texture sur le triangle

$$x = \frac{-B_0 u - C_0 v - D_0}{A_0}$$
$$y = \frac{-B_1 u - C_1 v - D_1}{A_1}$$
$$z = \frac{-B_2 u - C_2 v - D_2}{A_2}.$$

Ce qui nous intéresse est de trouver un vecteur de variation de la position en fonction de la coordonnée de texture. Nous utilisons la dérivée partielle suivante comme tangente

$$T = \left(\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u}\right) = \left(\frac{-B_0}{A_0}, \frac{-B_1}{A_1}, \frac{-B_2}{A_2}\right).$$

Avec la tangente $\hat{T}$ calculée et la normale $\hat{N}$ fournie dans le modèle, la binormale $\hat{B}$ est calculée avec

$$\hat{T} = \frac{T}{\|T\|}$$
$$\hat{B} = \hat{N} \times \hat{T}.$$

Finalement, nous assignons aussi à chaque tétraèdre un ensemble de propriétés d'extrusion. Premièrement, nous trouvons l'équation du plan (A^p, B^p, C^p, D^p) de chaque triangle contenu dans le tétraèdre. Le plan soutenant le triangle peut s'écrire avec l'équation

$$A^p x + B^p y + C^p z + D^p = 0.$$

Soient les sommets v_1, v_2, v_3 définis plus haut, la normale est

$$\hat{N}_{plan} = (A^p, B^p, C^p) = \frac{(v_3 - v_1) \times (v_2 - v_1)}{\|(v_3 - v_1) \times (v_2 - v_1)\|}.$$

Par convention, nous voulons que la normale pointe vers l'extérieur du tétraèdre qui contient le triangle. Si elle ne pointe pas vers l'extérieur, on la multiplie par -1. Trouver D^p revient à résoudre

$$D^p = -A^p x_0 - B^p y_0 - C^p z_0.$$

Deuxièmement, chaque prisme stocke une matrice qui correspond à la fonction bijective entre l'espace monde et l'espace texture. Ces données supplémentaires servent au rendu des tétraèdres dans le nuanceur de sommets.

Une extrusion de coquille telle que décrite satisfait pour nos applications. Malgré tout, dans certains modèles très concaves, la coquille peut s'auto-intersecter. Cependant, l'approche de la coquille pour plaquer une texture 3D peut couvrir un bon nombre de modèles planaires ou à différentes courbures convexes et concaves (Figure 3.2).

3.2 Espace texture 3D

Comme son nom l'indique, une texture 3D est un signal sur un domaine à trois dimensions. La façon intuitive de comprendre une texture 3D est d'avoir une grille régulière de dimensions (x, y, z). Chaque case de cette grille est appelée un voxel, une unité de volume. Une grande résolution de la grille permet de représenter des détails plus fins. Chaque voxel contient une information quelconque (couleur, normale, etc.).

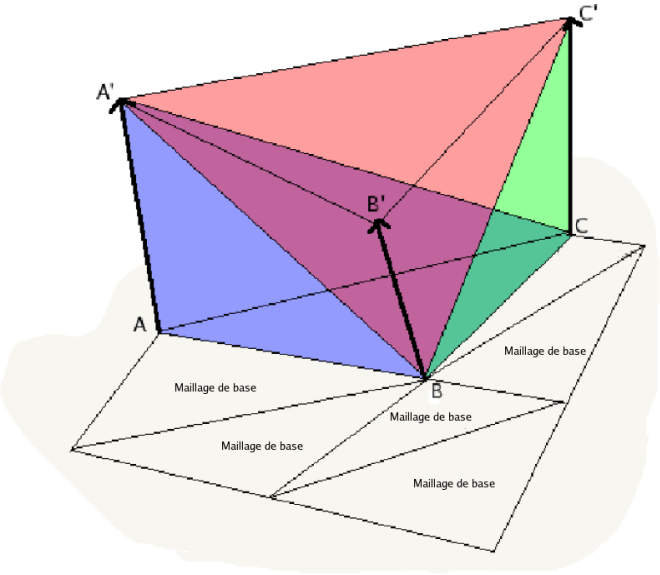

FIG. 3.1 – Chaque triangle du maillage de base est extrudé en trois tétraèdres. La décomposition en tétraèdres permet un tri efficace lors du rendu (chapitre 4).

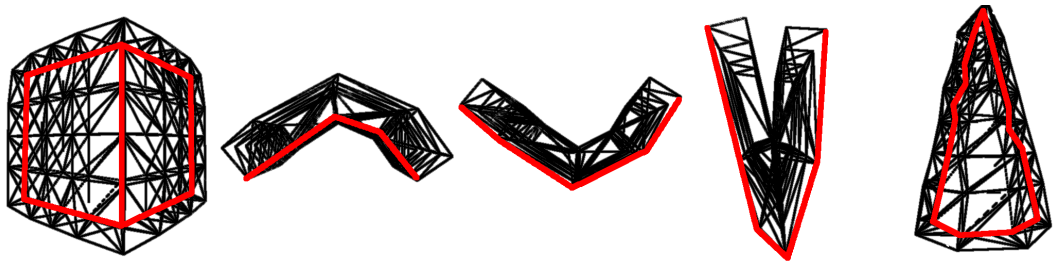

FIG. 3.2 – Exemples de coquilles sur maillages de différentes courbures. De gauche à droite, coquille sur cube avec des coins ; coquille convexe [1] ; coquille concave avec faible courbure ; problème d'auto-intersection qui peut survenir sur les maillages concaves de trop forte courbure ; coquille sur un cône déformé présentant des concavités et des convexités. La coquille peut s'adapter à un bon nombre de modèles géométriques.

La technique présentée dans ce mémoire n'impose pas de limite sur la résolution des textures. Cependant, les cartes vidéo ont une limite mémoire physique ainsi que des contraintes sur la taille des textures. En pratique, une résolution de $512 \times 512 \times 32$ a été suffisante pour la plupart de nos résultats. Une texture de cette résolution avec quatre canaux de couleur occupe sur la carte vidéo 32 Mo d'espace mémoire.

[1] Le mot convexe est utilisé dans ce contexte si pour tout segment de droite reliant deux sommets extrudés de la coquille, le segment ne passe par par l'extérieur de la coquille sans intersecter la surface de base représentée par une ligne rouge dans cette figure. Inversement pour concave.

Notre implémentation qui construit les textures 3D prend en entrée un modèle géométrique. Elle découpe le modèle en z tranches projetées de façon orthographique[1] sur une caméra placée au zénith. Les voxels sur la frontière du modèle 3D ont la couleur du polygone qu'ils contiennent. Ensuite, une analyse est faite pour remplir l'intérieur du modèle voxelisé dans la texture. En bouclant sur chaque colonne de voxels, un drapeau est initialisé à *extérieur*, puis lorsque l'itérateur rencontre un voxel plein (coloré), le drapeau est mis à *intérieur*. Chaque voxel *intérieur* est rempli. Les normales servent aussi à traiter les auto-intersections dans le modèle original. L'itérateur entre à l'intérieur du modèle s'il est à l'extérieur et qu'il rencontre un voxel plein dont la normale pointe vers le haut. Cet algorithme simple fonctionne, mais nécessite quand même des manipulations manuelles de la texture finale puisque dans certains cas pathologiques, des voxels sont remplis alors qu'ils devraient être vides. Les voxels intérieurs sont remplis de la couleur moyenne de leurs voisins.

L'espace texture 3D est défini sur le domaine $[0, 1]$ dans les trois dimensions et juxtapose les voxels. Les coordonnées de texture 3D de la forme (u, v, w) font la correspondance entre les sommets sur le maillage et les voxels dans la grille. Pour chaque sommet, les coordonnées de texture indiquent quel voxel doit être utilisé pour faire le rendu. La paramétrisation du modèle consiste à attribuer à chaque sommet ses coordonnées de texture. Un artiste peut faire le travail manuellement, mais des techniques de paramétrisation automatiques existent pour limiter la déformation entre l'espace texture et l'espace géométrique (par exemple [GGS03]). Nous avons opté pour la paramétrisation manuelle puisqu'elle permet un grand contrôle du placage de la texture.

3.3 Fonction bijective

Dans la grande majorité des correspondances entre maillage et texture 3D, chaque tétraèdre possède une fonction bijective entre l'espace objet et l'espace texture. Dans ce mémoire, les exposants ^{o} et ^{t} représentent les espaces objet et texture respectivement. Nous construisons la matrice $\mathbf{M}^{o \to t}$ transformant un point en espace 3D objet vers l'espace 3D texture. Cette matrice définit une transformation affine. La correspondance des coordonnées spatiales 3D vers les coordonnées texture 2D utilise souvent une

[1] Une projection orthographique dont le plan de projection est en $z = 0$ est une transformation linéaire qui envoie le vecteur (x, y, z) en $(x, y, 0)$.

transformation affine [Hec86]. Une transformation affine peut inclure une rotation, un changement d'échelle, un cisaillement et une translation [FvDKH96]. Dans notre situation, nous estimons la correspondance entre un espace 3D objet à un espace 3D texture par la transformation affine qui s'exprime sous la forme suivante

$$\mathbf{M}^{o\to t} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}.$$

Ainsi, soit un point $p^o = (x^o, y^o, z^o, 1)^T$ dans le tétraèdre, ses coordonnées texture 3D correspondante $p^t = (u^t, v^t, z^t)$ s'estime par $p^t = \mathbf{M}^{o\to t} p^o$. La matrice précédente donne l'ensemble d'équations

$$u^t = ax^o + by^o + cz^o + d$$
$$v^t = ex^o + fy^o + gz^o + h$$
$$w^t = ix^o + jy^o + kz^o + l$$

pour chaque sommet du tétraèdre. Les coefficients de la matrice $\mathbf{M}^{o\to t}$ sont trouvés par lissage linéaire au sens des moindres carrés.

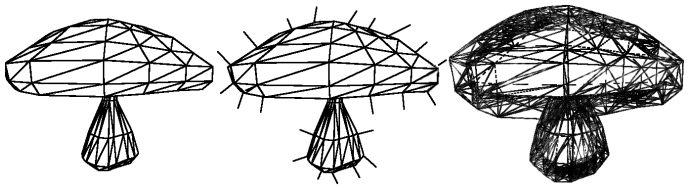

FIG. 3.3 – Les sommets d'un maillage sont copiés, puis déplacés en fonction d'un vecteur de déplacement. Ensuite, une coquille de tétraèdres est créée pour accueillir la texture 3D.

Chapitre 4

Rendu avec tri

Ne rien livrer au hasard, c'est économiser du travail.

Antoine Albalat

En rendu sur carte vidéo, la semi-transparence crée un problème supplémentaire. L'ordre dans lequel on rend les primitives à l'image est important puisque l'accumulation de la couleur et de l'opacité se fait par mélange de valeur alpha. Le mélange de valeur alpha est une technique pour approximer l'intégration de l'opacité (stockée dans le canal alpha) et de la couleur dans un volume. Les couleurs et l'opacité s'accumulent dans le tampon avec un opérateur de mélange pour simuler de la semi-transparence. Dans ce chapitre, nous expliquons le mélange de valeur alpha qui est à la base de notre algorithme ainsi que les deux méthodes utilisées pour trier les primitives (tétraèdres) lors du rendu : le découpage en couches de profondeur [Eve99] et l'algorithme SXMPVO [CMSW04].

4.1 Mélange de valeur alpha

Dans un milieu participatif uniforme, la lumière est absorbée de façon exponentielle [HRS04]

$$L(x_0, \vec{w}) = e^{-\kappa x} L(x, \vec{w})$$

où $L(x_0, \vec{w})$ est la radiance sortant du médium à la position x_0 et en direction $\vec{w}$, $L(x, \vec{w})$ est la radiance entrant dans le médium au point x en direction $\vec{w}$ et κ est le coefficient d'absorption du médium. Dans le cas d'un médium non-uniforme, le coefficient constant et la radiance deviennent dépendants de la position dans le médium. L'absorption totale

et la couleur doivent être intégrées

$$L(x_0, \vec{w}) = \int_0^{\infty} L(x, \vec{w}) e^{-\int_0^x \kappa(u)du} dx. \tag{4.1}$$

Cette intégrale peut être résolue numériquement en échantillonnant le médium le long d'un rayon et en accumulant l'opacité et la couleur. Cette opération s'appelle la marche par étape sur un rayon. Cependant, évaluer cette intégrale, même par cette technique, est coûteux en matériel (fonction exponentielle). C'est pourquoi on a plutôt recours à une estimation de cette intégrale en introduisant l'opacité d'une tranche de volume (valeur alpha) où Δt est l'épaisseur de cette tranche et i est l'indice de la tranche si on découpe le rayon en N tranches

$$A_i = 1 - e^{-\kappa(i\Delta t)\Delta t}.$$

Alors, l'intégrale de l'équation 4.1 peut être estimée par

$$L(x_0, \vec{w}) \approx \sum_{i=0}^{N} L_i \prod_{j=0}^{i-1} (1 - A_j)$$

évaluée de façon itérative. En partant de l'échantillon le plus éloigné le long du rayon dans le médium, on accumule l'opacité et la couleur dans un tampon en effectuant une opération de multiplication et d'addition entre l'échantillon courant et les anciens. L'accumulation peut aussi se faire en commençant par le devant du médium. Dans les deux cas, l'ordre des échantillons de couleur retournés est important (Figure 4.1). C'est pourquoi on doit utiliser un algorithme de tri dans notre système.

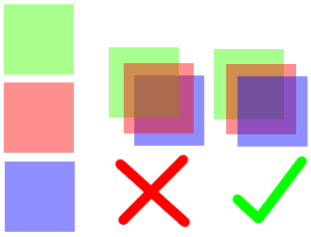

FIG. 4.1 – (gauche) Trois fragments vert (derrière), rouge (milieu) et bleu (devant). (centre) Les trois fragments sont rendus dans le mauvais ordre donnant un mélange de couleurs bizarre. (droite) L'ordre de rendu de derrière à devant donne le résultat attendu.

En OpenGL, ce processus de mélange de valeur alpha s'active dans le tampon de couleur. L'opération se commande en spécifiant une opération à effectuer entre le fragment de triangle courant et la couleur déjà présente à ce pixel dans le tampon (résultant

d'un rendu précédent). Si le fragment courant est indicé $_{src}$ et le fragment précédent est $_{dst}$, OpenGL effectue l'opération suivante sur chacun des quatre canaux (RGBA) pour faire du mélange de valeur alpha dans le tampon de couleur

$$C_{finale} = w_{src}C_{src} + w_{dst}C_{dst}$$

où les valeurs w_{src} et w_{dst} sont des poids de mélange spécifiés par l'application. Ces poids peuvent être constants pour tous les pixels ou dépendre de la valeur alpha du fragment source ou destination.

4.2 Découpage en couches de profondeur

L'algorithme de découpage en couches de profondeur [Eve99] permet de rendre les primitives semi-transparentes en effectuant plusieurs passes de rendu de la même scène. Lorsque le nombre de polygones projetés dans un pixel est grand, le nombre de passes requis est grand, ce qui rend l'algorithme impraticable. L'algorithme est assez simple et général ; il peut rendre n'importe quelle forme de primitives et n'importe quelle complexité de scène. Ainsi, comme il est coûteux pour des scènes complexes, nous l'avons implanté, mais utilisons un autre algorithme de tri dans la version finale (Section 4.3).

L'algorithme procède comme suit. Premièrement nous initialisons une texture de profondeur au plan de découpage éloigné. Ensuite, chaque passe de rendu simule deux tests de profondeur sur une carte avec un seul tampon de profondeur. L'intuition de cette méthode est qu'à chaque passe de rendu, nous voulons garder seulement la couche visible la plus proche qui n'a pas été déjà rendue. Ainsi, la première passe retourne les premiers tétraèdres visibles, la deuxième passe retourne les deuxièmes tétraèdres visibles et ainsi de suite (Figure 4.2). Le processus est similaire à un épluchage de la scène couche par couche. Le coût provient qu'à chaque passe, tous les polygones doivent être traités, même s'ils seront finalement abandonnés dans l'étape des pixels. Chaque passe consiste à

1. Vider le tampon de profondeur.
2. Mettre le test de tampon de profondeur à *plus petit que* et rendre les tétraèdres et leurs triangles en les mettant opaques sans toutefois écrire dans le tampon de couleur de la carte. Dans cette opération, nous comparons la profondeur des fragments résultants à la profondeur de la passe précédente pour ne garder que

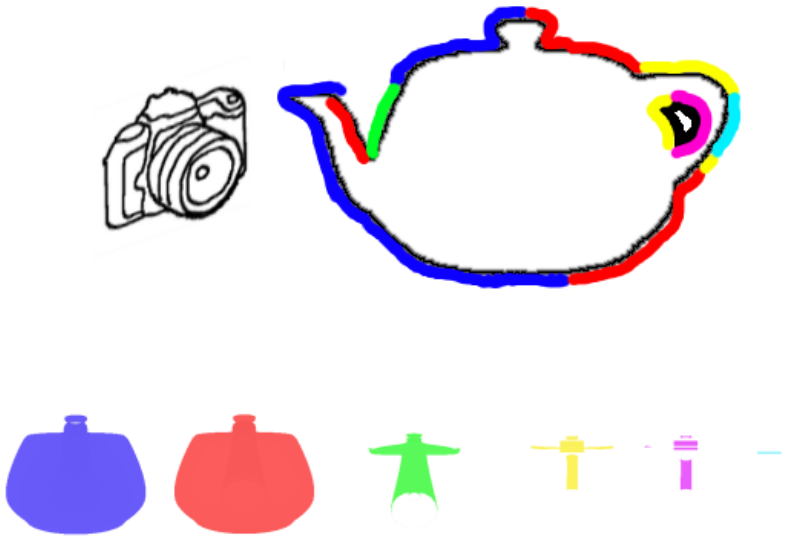

FIG. 4.2 – Les fragments rendus par découpage en couches de profondeur sont dans l'ordre : 1-bleus, 2-rouges, 3-verts, 4-jaunes, 5-violets et 6-cyans. Cette scène nécessite donc six passes de rendu.

les fragments qui sont les plus proches de l'oeil, mais en même temps, plus loins que ceux rendus dans la passe précédente.

3. Copier le tampon de profondeur courant dans la texture de profondeur pour être utilisé dans la prochaine passe de rendu.

La couleur retournée par chaque tétraèdre visible est mélangée par valeur alpha à la couleur courante du tampon de couleur avec poids $(1 - \alpha_{dst}, 1)$.

À cause de la complexité de certaines de nos scènes test (environ 40 triangles dans un pixel), 20 secondes ont été nécessaires pour faire le rendu d'un vase (Figure 4.7) avec textures volumétriques en utilisant le découpage en couches de profondeur, comparativement à 0.5 secondes avec notre stratégie de tri dans la mémoire principale.

4.2.1 Analyse et performances

L'algorithme de découpage en couches de profondeur permet de faire un rendu trié complètement sur la carte vidéo. Comme il fonctionne entièrement en espace image, l'algorithme assure que le tri sera correct pour la résolution de l'image puisque le tri

se fait au niveau des pixels et non des prismes. La performance d'un tel algorithme dépend surtout de la complexité en profondeur de la scène. Plus le nombre de triangles projetant dans un pixel est grand, plus l'application doit découper la scène en un grand nombre de tranches. Chaque tranche nécessite un rendu ainsi qu'un mélange par valeur alpha avec les tranches précédentes.

Soit T le nombre de triangles dans le modèle (nombre de tétraèdres $\times 4$) disposés en P couches de profondeur et une résolution d'image $x \times y$. Il faut P rendus de la scène pour obtenir un rendu trié. Premièrement, chaque rendu nécessite le traitement de $T \times 3$ sommets dans le nuanceur de sommets et une passe pour vider le tampon de profondeur de la carte vidéo en réinitialisant $x \times y \times 3$ octets de mémoire. Cette étape est coûteuse au niveau du transfert des données des sommets. Deuxièmement, les pixels couverts par les triangles de la couche actuelle passeront dans un nuanceur de pixels. Le nuanceur exclut les pixels cachés, mais fait l'intégration de couleur pour les pixels visibles. Finalement, la carte doit faire plusieurs opérations de mélange de couleur alpha pour accumuler la couleur de toutes les couches de profondeur dans le tampon de couleur, ce qui est coûteux en opération mémoire. Le temps de rendu dépend donc du nombre de triangles du modèle, du nombre de couches de profondeur et du nombre de pixels couverts par le modèle. Les deux premiers facteurs sont liés, alors que le nombre de pixels couverts dépend de la résolution de l'image et de la grosseur de l'objet projeté dans l'image.

Dans la Figure 4.3, le nombre de triangles est fixé à 512 et le nombre de pixels couverts reste le même. Nous faisons varier le nombre de couches de profondeur et notons le temps de rendu nécessaire. Les résultats de la même expérience avec un autre modèle sont exposés dans la Figure 4.4. Le nombre de triangles dans cette figure est de 17 136. Le temps de rendu croît linéairement en fonction du nombre de couches de profondeur. Le grand coût nécessaire au rendu avec découpage en couches de profondeur indique qu'il ne s'agit pas de l'algorithme idéal pour faire le rendu trié dans notre technique. Lorsque la scène est constituée de maillages triangulaires ordinaires, le découpage en couches de profondeur peut être satisfaisant, mais l'utilisation de tétraèdres pour faire une coquille accueillant la texture 3D ajoute une complexité de profondeur supplémentaire qui empêche cette méthode d'avoir des performances supérieures. Même pour un modèle simple avec 512 triangles, coquilles incluses, la technique descend en bas de 10 images

par seconde avec une complexité de 8 couches de profondeur.

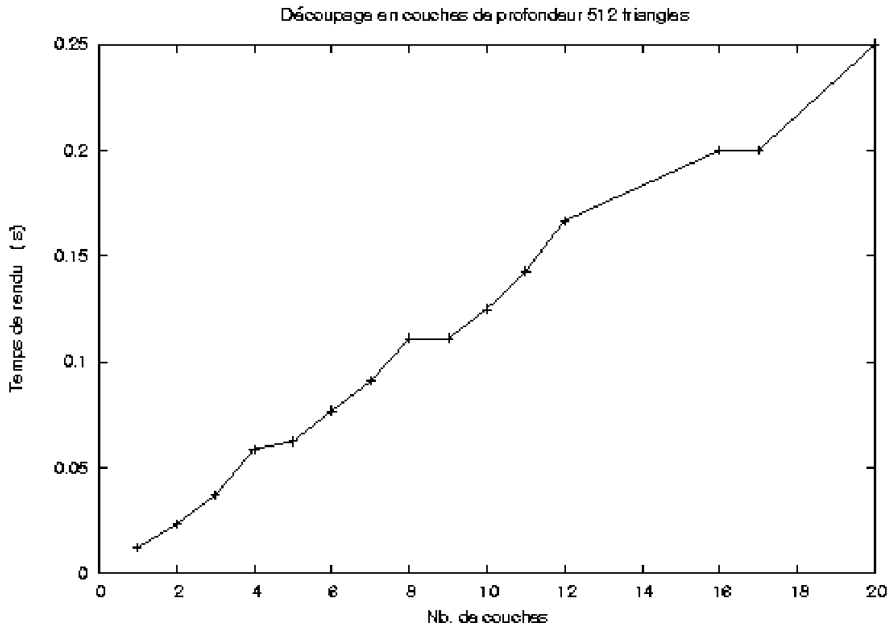

FIG. 4.3 – Temps de rendu pour un modèle de 512 triangles avec le découpage en couches de profondeur. Le nombre de pixels couverts reste fixe.

4.3 SXMPVO

Une autre approche consiste à trier les primitives avant d'en faire le rendu sur la carte vidéo. Nous avons implanté un algorithme en espace image appelé SXMPVO [CMSW04] qui trie des cellules convexes. Soit un ensemble de tétraèdres et une projection, SXMPVO retourne un ordre de visibilité correct s'il n'existe pas de cycles dans l'ensemble. Par construction de la coquille extrudée, les tétraèdres ne s'intersectent pas, donc les configurations qui pourraient introduire des cycles de visibilité sont rares dans la plupart des modèles 3D. Les cycles sont cependant détectés et ils peuvent être traités par un algorithme de découpage de tétraèdres.

Soit S l'ensemble de tous les tétraèdres du modèle. Chaque tétraèdre possède une liste de tétraèdres qui le cachent. Appelons cette relation "derrière" ($\propto$). Un tétraèdre est marqué derrière un autre s'il doit être rendu avant celui-ci pour conserver l'ordre derrière à devant. Par exemple, soient $A, B \in S$, A est derrière B ($A \propto B$) si pour un pixel donné dans l'image, un fragment de A est derrière un fragment de B. Dans ce cas précis, A doit être rendu avant B si nous voulons faire un rendu de derrière à

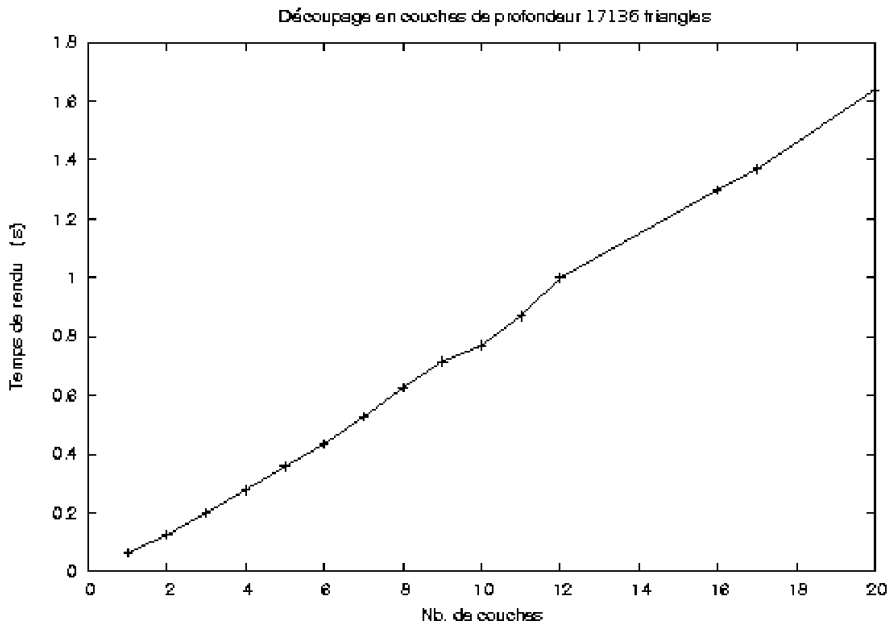

FIG. 4.4 – Temps de rendu pour un modèle de 17136 triangles avec le découpage en couches de profondeur. Le nombre de pixels couverts reste fixe.

devant car certains fragments sont derrières. Notons que cette propriété est transitive. Cette propriété est suffisante pour qu'on puisse établir un ordre de rendu entre les tétraèdres s'il n'existe pas de cycles. De façon équivalente, pour une suite de tétraèdres $S = t_0 \propto t_1 \propto t_2 \propto ... \propto t_n$ avec $t_0, t_1, ..., t_n \in S$, chaque t_i pigé dans S doit y apparaître une seule fois (si $t_i = t_j$, alors $i = j$). Pour faire un rendu trié, il suffit de faire le rendu des tétraèdres dans le sens de cette suite.

L'algorithme SXMPVO fonctionne en deux étapes. Premièrement, chaque paire de tétraèdres qui partagent une face (triangle) est triée en fonction du point de vue (i.e. vecteur positionné à un sommet du triangle en direction de la caméra). Pour ce faire, nous faisons le produit scalaire entre la normale N du triangle avec le vecteur de vue V. Si $N \cdot V < 0$, le triangle fait dos à la caméra. Par convention, la normale d'un triangle pointe à l'extérieur du tétraèdre dont il fait partie. Alors, nous savons que le triangle testé faisant dos à la caméra fait partie du tétraèdre qui cache son voisin. Nous pouvons ajouter le tétraèdre "bloqueur" à la liste des relations "derrière" du tétraèdre bloqué. Cette première étape permet d'établir un ordre partiel entre les tétraèdres qui partagent une face. Par exemple, A et B partagent un triangle F. Dans A, la normale de F pointe vers l'oeil et dans B, la normale de F pointe en direction opposée (Figure 4.5. Alors,

$A \propto B$.

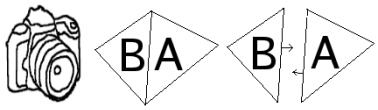

FIG. 4.5 – Exemple équivalent en 2D. Deux triangles voisins partagent une face. La normale pointe à l'extérieur du triangle. Le triangle A possédant le côté qui fait face à la caméra est derrière le triangle B possédant le côté faisant dos à la caméra.

La suite n'est pas complétée à ce stade. Il faut établir des relations entre les tétraèdres qui ne partagent pas de face, mais qui sont bel et bien dans une relation "derrière". Par exemple, deux tétraèdres ne partageant pas de face, peuvent se projeter dans un même pixel. Les triangles qui ne sont pas partagés entre plusieurs tétraèdres sont projetés dans une version simplifiée d'un *A-buffer* [Car84]. La version simplifiée du *A-buffer* stocke pour chaque pixel une liste de tous les triangles qui y sont projetés. Chaque liste de triangles de chaque pixel du *A-buffer* est triée selon la profondeur du fragment. Placés dans cet ordre, les triangles permettent d'établir d'autres relations "derrière" entre les tétraèdres. Un triangle devant un autre dans la liste (s'ils n'appartiennent pas au même tétraèdre) cache nécessairement son suivant dans la liste. Ainsi, on peut ajouter une relation "derrière" entre les deux tétraèdres qui contiennent ces triangles.

Toutes ces relations "derrière" établissent un graphe. Les tétraèdres qui n'ont pas de tétraèdres dans leur liste de bloqueurs ne sont bloqués par aucun tétraèdre et doivent être donc rendus en dernier. Ils forment un ensemble de points de départ pour une traversée de ce graphe. L'ordre de visibilité correct est donné par une traversée en profondeur d'abord du graphe des relations de "derrière".

Pour des raisons d'efficacité, le *A-buffer* peut être rendu à une résolution plus faible tant que tous les tétraèdres soient projetés et visibles. Dans le cas contraire, les tétraèdres manquant résultent dans des trous visibles dans le rendu final. Dans ce mémoire pour éviter ce problème, nous avons utilisé un *A-buffer* de la même résolution que le tampon de couleur.

4.3.1 Analyse et performances

Contrairement à l'algorithme par découpage de couches de profondeur, SXMPVO tient compte de la connectivité entre les triangles. Les triangles faisant partie d'un même prisme sont triés séparément avant la projection dans le *A-buffer*. Le désavantage majeur est l'impossibilité de trier tel quel un maillage qui contient un cycle de visibilité. De plus, cet algorithme utilise les ressources du processeur central.

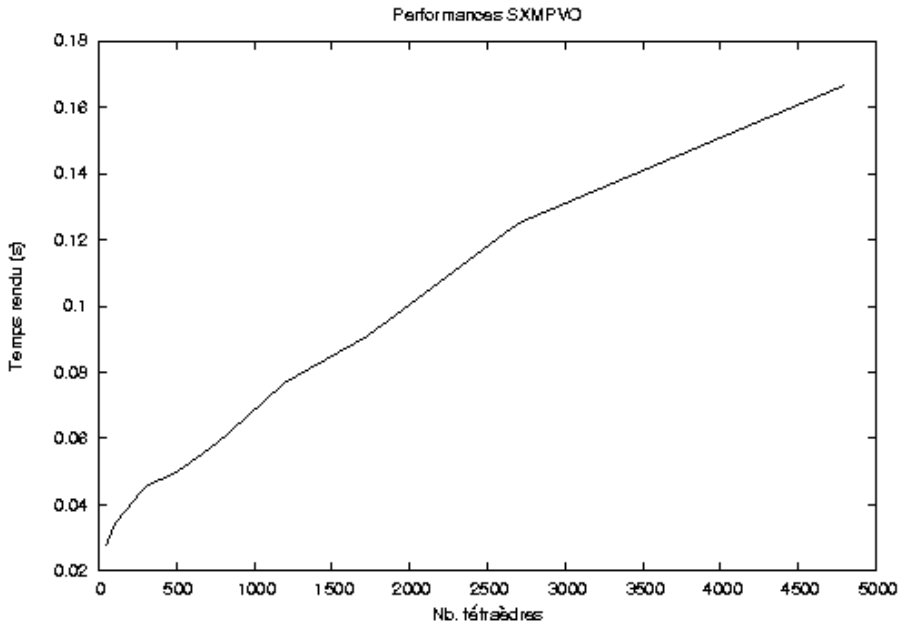

FIG. 4.6 – Graphique qui démontre que malgré un nombre grandissant de tétraèdres, SXMVPO continue d'offrir de meilleures performances dans notre technique. Le modèle choisi était une sphère creuse dont la subdivision augmente pour chaque test.

Les facteurs majeurs pour les performances de SXMPVO sont le nombre de tétraèdres et le nombre de trous dans le modèle. Les tétraèdres qui ne sont pas connectés sont projetés dans un *A-buffer* en mémoire principale. Cette opération de rastérisation logicielle [1] est coûteuse. Cependant, cette opération n'est effectuée qu'une fois par image. Une fois complétée, les triangles sont envoyés dans le bon ordre à la carte vidéo, allégeant son travail.

La Figure 4.6 affiche le résultat d'une expérience dans laquelle une sphère de subdivision grandissante est rendue avec notre technique. Le rendu trié utilise SXMPVO. La première observation qu'on note est qu'il existe un coût de base à SXMPVO. La

[1] *software*

première mesure effectuée compte 48 tétraèdres et le temps de rendu est d'environ 0.3 seconde. Le tri en logiciel limite déjà les performances de la technique à 30 images par seconde avec un modèle simple. Cependant, quand le nombre de tétraèdres croît, les performances ne dégradent pas aussi rapidement qu'avec le découpage en couches de profondeur (Figures 4.3 et 4.4). Même avec 19 000 triangles dans le modèle, le rendu se fait à 6 images par seconde. Les graphiques de SXMPVO et du découpage en couches de profondeur ont été faits à la même résolution d'image, la même texture 3D et le même nombre d'échantillons par segment de rayon.

Pour garder un taux de rafraîchissement interactif de 10 images par seconde, le modèle avec coquille peut contenir environ 2000 tétraèdres.

L'utilisation d'un *A-buffer* logiciel à la place du *Z-buffer* en matériel est nécessaire puisque SXMPVO trie une liste des tétraèdres visibles pour chaque pixel de l'image. Le tampon de profondeur de la carte vidéo peut contenir une seule valeur. Récemment, la possibilité de faire du rendu dans plusieurs tampons en même temps [2] permet d'implanter une structure de données similaire à un *A-buffer* avec profondeur maximale entièrement sur la carte vidéo appelée, *K-buffer* [CC05]. Partant d'une liste de tétraèdres presque triée, cet algorithme utilise une technique de fenêtrage de la liste de tétraèdres stockée dans des textures et utilise des nuanceurs pour écrire les tétraèdres dans le tampon de couleur dans le bon ordre. Ce nouvel algorithme suggère qu'une cohérence temporelle pourrait être exploitée si dans une première image, on procède au tri total avec SXMVPO, mais pour des images subséquentes, avec des mouvements de caméra légers, on utilise le *K-buffer* pour retrier les tétraèdres.

[2] MRT

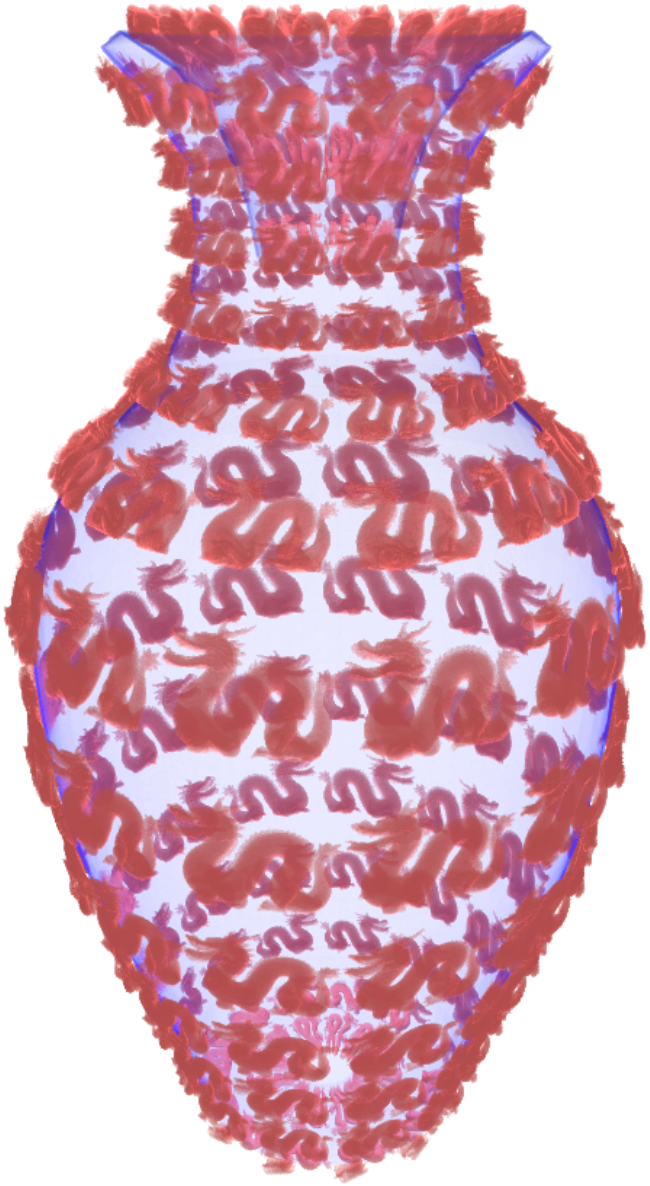

FIG. 4.7 – Vase avec une texture de dragons semi-transparente. Le terme de correction de l'opacité assure une intégration de couleur valide par mélange de valeur alpha. Les dragons sont plus opaques que le reste du vase puisqu'ils ont une valeur alpha plus grande servant à simuler un matériau plus absorbant. La résolution de la texture est $128 \times 128 \times 32$ et dix échantillons sont pris dans chaque segment de rayon.

Chapitre 5

Rendu

Et si tout n'était qu'illusion et que rien n'existait ? Dans ce cas, j'aurais vraiment payé mon tapis beaucoup trop cher.

Woody Allen

Dans ce chapitre, nous expliquons notre algorithme de rendu. Le rendu est fait sur la carte vidéo en utilisant des nuanceurs de sommets et de pixels. Après une explication intuitive de l'algorithme, nous présentons les détails de notre implémentation.

5.1 Algorithme

L'algorithme de rendu est composé de trois étapes (Figure 5.14) :

(a) Pour les modèles qui contiennent des textures semi-transparentes, les tétraèdres visibles sont triés dans un ordre derrière à devant en utilisant une de nos deux stratégies (chapitre 4).

(b) Pour chaque tétraèdre, nous trouvons le point d'entrée et le point de sortie du rayon de vue. Nous transformons ces points de l'espace monde 3D à l'espace texture 3D.

(c) Une intégration de l'opacité et de la couleur est faite dans le tétraèdre sur ce segment de rayon dans la texture 3D en faisant une sommation par mélange d'opacité des voxels traversés dans la texture 3D.

5.1.1 Calcul d'intersection entre le rayon et les tétraèdres

Chaque tétraèdre est rendu comme une primitive de base. Nous devons trouver les points d'entrée et de sortie du rayon de vue dans le tétraèdre courant et transformer ces points en espace texture 3D (Figure 5.14 b et c). Pour des raisons d'efficacité, seuls les triangles faisant face à la caméra sont rendus par la carte vidéo. Un nuanceur de sommets définit le rayon sur lequel nous faisons l'accumulation de l'opacité dans le nuanceur de pixels. Chaque sommet du tétraèdre est traité par le nuanceur de sommets. Parmi ses attributs, on retrouve ses coordonnées de texture 3D et l'équation du plan qui lui est opposé dans le tétraèdre. Soit le triangle courant, nous savons que le point d'entrée est dans ce triangle et que le point de sortie est quelque part situé sur un des trois autres triangles du tétraèdre.

Dans le nuanceur de sommets, le point d'entrée du rayon de vue dans le tétraèdre en espace monde p_i^o correspond à la position du sommet courant en espace monde. Pour trouver le point de sortie du rayon, nous calculons l'intersection entre le rayon de vue et les trois autres plans du tétraèdre. Puisque le sommet courant est partagé par deux des trois autres triangles, deux des trois intersections correspondent au sommet lui-même et n'ont pas besoin d'être calculées.

Soient le plan $P = Ax + By + Cz + D = 0$ et le rayon $R = R^o + \vec{R^d}t$. À l'intersection $p = (a, b, c)$, les deux équations doivent être satisfaites. Substituons la position du point dans P par l'équation du rayon et résolvons pour t.

$$\begin{aligned}
Ax + By + Cz + D &= 0 \\
A(R_x^o + R_x^d t) + B(R_y^o + R_y^d t) + C(R_z^o + R_z^d t) + D &= 0 \\
AR_x^o + AR_x^d t + BR_y^o + BR_y^d t + CR_z^o + CR_z^d t + D &= 0 \\
AR_x^d t + BR_y^d t + CR_z^d t &= -\left(AR_x^o + BR_y^o + CR_z^o + D\right) \\
t &= \frac{-\left(AR_x^o + BR_y^o + CR_z^o + D\right)}{AR_x^d + BR_y^d + CR_z^d}.
\end{aligned}$$

De cette façon, nous calculons le point d'intersection entre le rayon et les trois autres plans du tétraèdre et aussi la distance du point d'origine du rayon aux autres plans.

Soit le point d'intersection trouvé p_o^o, ses coordonnées de texture correspondantes p_o^t sont trouvées en le multipliant par la matrice $\mathbf{M}^{o \to t}$ définie précédemment (Section 3.3). Ainsi, le rastériseur interpole le point d'entrée p_i^o et le point de sortie p_o^o dans le tétraèdre en espace monde. Les coordonnées de texture correspondantes (u, v, w) pour les deux

points (p_i^t et p_o^t) sont aussi interpolées par le rastériseur. Comme les coordonnées de texture du point de sortie ne sont pas des propriétés affines en espace image, l'interpolation doit faire appel à un terme de correction de perspective pour être valide (en supplément de celui fourni par la carte vidéo). En effet, l'interpolation linéaire en espace image de ces coordonnées ne respecte pas la nature projective de ce calcul ; sans terme de correction, une erreur s'ajouterait au calcul (Figure 5.1). Même si cette opération d'intersection est similaire à celle retrouvée dans le travail de Wang *et al.* [WTL+04], il est plus simple puisque nous utilisons des tétraèdres (convexes) plutôt que des prismes de forme arbitraire. Finalement, un nuanceur de pixels accumule l'opacité et la couleur de ce segment de rayon pour le combiner à la couleur des autres tétraèdres.

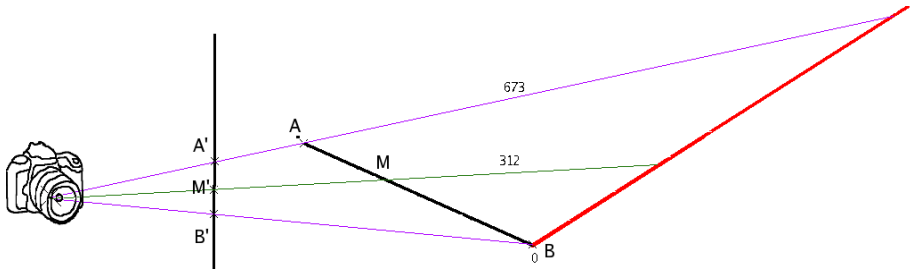

FIG. 5.1 – Interpolation linéaire (2D) erronée en espace image. Les points A, B et M projettent respectivement sur l'image en A', B', M'. La distance réelle entre le point A du plan noir et le plan rouge est de 673 pixels. La distance entre le point B et le plan rouge est de 0 pixel. M' est situé à mi-chemin entre A' et B' dans l'image, alors par interpolation linéaire en espace image, on pourrait déduire que la distance entre M et le plan rouge est de 336.5 pixels. Cependant, la distance réelle est de 312 pixels. C'est pourquoi un terme de correction perspectif est nécessaire avant l'interpolation.

5.1.2 Intégration d'opacité et couleur

Les points d'intersection calculés précédemment permettent de définir le segment de rayon dans le tétraèdre en espace texture (et son équivalent R^o en espace monde)

$$R^t = p_i^t + \lambda(p_o^t - p_i^t)$$

où $\lambda \in [0, 1]$.

Le rayon R^t permet d'échantillonner la texture 3D dans son espace uniforme. La texture 3D représente des détails de surface dans une grille 3D avec axes $(u, v, w) \in [0, 1]$.

La résolution de la texture découpe en cubes (voxels) la texture. Plusieurs algorithmes efficaces peuvent être utilisés pour échantillonner le segment de rayon dans cet espace régulier.

La première option consiste à utiliser un algorithme de traversée de grille uniforme en 3D [AW87]. L'algorithme initialise des variables d'incrément sur la position du rayon par rapport au paramètre λ. Ces variables incluent une variation de λ par incrément de un voxel selon chacun des trois axes. À chaque itération de l'algorithme, un test est effectué pour déterminer quel voxel est le plus proche de la position actuelle sur le rayon. Chaque avancement de λ sur le rayon incrémente les compteurs du bon taux de variation assurant qu'aucun voxel n'est manqué dans la marche sur le rayon (Figure 5.2). Malgré cet avantage, l'algorithme reste coûteux puisque beaucoup de lectures de la texture sont nécessaires, surtout dans le cas de textures à très haute résolution.

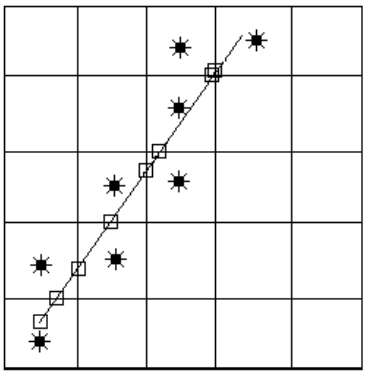

FIG. 5.2 – Échantillonnage DDA : les carrés indiquent les incréments sur le rayon et les étoiles indiquent les voxels échantillonnés.

Une autre option est la marche par pas uniforme. Dans ce cas, la marche prend un premier échantillon à l'origine p_i^t du rayon correspondant à $\lambda = 0$. Si on prend N échantillons, l'algorithme incrémente ensuite chaque échantillon λ de $\frac{\|p_o^t - p_i^t\|}{N}$. Cette deuxième approche peut manquer certains détails (voxels) dans la texture si N est trop petit par rapport à la longueur du segment en voxels (Figure 5.3). De plus, un échantillonnage uniforme sera souvent superflu dans les zones de textures vides ou pour les très petits segments de rayon qui ne traversent pas beaucoup de voxels. Cependant, l'algorithme est plus simple et ces compromis sont acceptables dans un contexte de rendu avec carte vidéo où la cohérence et le parallélisme est important. Le critère de performance devient donc le nombre d'échantillons pris.

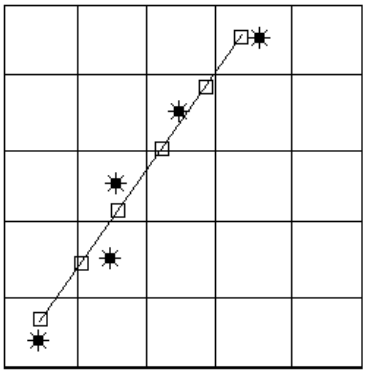

FIG. 5.3 – Échantillonnage uniforme : les carrés indiquent les incréments sur le rayon et les étoiles indiquent les voxels échantillonnés.

La couleur finale retournée par le nuanceur de pixels est le résultat d'une accumulation d'opacité et de couleur par mélange de valeur alpha des voxels traversés par R^t dans la texture 3D. La qualité de la reconstruction dépend donc de la résolution de la texture 3D et de l'algorithme choisi pour échantillonner les voxels sur le segment de rayon. Nous utilisons cette équation pour le mélange de valeur alpha [Lev90] :

$$C'_i = C'_{i-1} + (1 - \alpha'_{i-1})C_i \tag{5.1}$$

$$\alpha'_i = \alpha'_{i-1} + (1 - \alpha'_{i-1})\alpha_i. \tag{5.2}$$

Une fois l'échantillonnage du segment de rayon à l'intérieur du tétraèdre terminé, la couleur résultante est mélangée par valeur alpha à la couleur déjà présente dans le tampon de couleur avec les poids $(1 - \alpha_{dst}, 1)$.

Dépendamment du schéma d'intégration choisi, la distance entre les échantillons peut ne pas être uniforme et la valeur alpha des voxels évalue une opacité sur une distance constante qui n'est sûrement pas identique à la distance entre les échantillons dans l'intégration. C'est pourquoi nous devons corriger la valeur d'opacité d'un échantillon en fonction de la distance parcourue par la lumière dans ce bout de rayon ($\Delta t \|R^o\|$). Nous utilisons ce terme de correction pour la valeur alpha d'un échantillon [HRS04] :

$$\alpha_i = 1 - (1 - \alpha_i^t)^{\Delta t \|R^o\|}$$

où α_i^t est la valeur alpha stockée dans la texture 3D à la position de l'échantillon courant et α_i est la valeur corrigée.

À chaque échantillon sur le segment de rayon, il serait possible d'échantillonner aussi un rayon en direction de la lumière pour ajouter de la diffusion à notre modèle pour

ajouter des éclats de lumière dans les milieux participatifs. En effet, la diffusion de lumière dans la texture correspond à intégrer la quantité de lumière émise d'une source de lumière et qui est propagée dans la texture 3D. Cette intégrale peut être évaluée avec d'autres lancers de rayons vers la lumière et vers les voxels voisins pour connaître leurs propriétés de diffusion et d'absorption. Cependant, tout comme dans le travail de Wang *et al.* [WTL+04], nous n'avons aucune information sur les tétraèdres qui séparent le tétraèdre courant de la lumière, et donc nous devons ignorer tous les effets de diffusions dans la texture 3D. Nous pourrions utiliser une structure comme la carte d'ombrage avec profondeur [LV00, KN01] à la place d'une carte d'ombrage régulière. Une telle carte stocke dans une texture une fonction d'absorption de lumière en fonction de la distance à la source. Pour la construire, il faudrait faire un rendu similaire au rendu fait pour obtenir l'image du point de vue de la source de lumière, mais cette structure serait trop coûteuse à recalculer à chaque déplacement de lumière ou déformation de l'objet.

5.2 Implémentation

Nous avons implanté l'algorithme en trois modules. Le premier est une application exécutée sur le processeur central qui s'occupe de lancer les appels à la carte vidéo pour le rendu et envoyer les données du modèle géométrique 3D. Le deuxième est un nuanceur de sommets qui est en charge du calcul d'intersection entre le rayon de vue et le tétraèdre. Finalement, le nuanceur de pixels est responsable de l'accumulation de la couleur et de l'opacité.

5.2.1 Pipeline graphique

Les premières générations de cartes vidéo offraient des opérations de base sur les sommets et les pixels. Ces opérations étaient fixes dans le pipeline. Les nouvelles générations de cartes vidéo permettent de changer ces opérations fixes avec des programmes appelés nuanceurs. Un nuanceur remplace totalement les opérations fixes et permet de fournir des instructions au processeur graphique pour transformer les sommets ou les pixels selon les désirs du programmeur.

Le rendu par carte vidéo se base sur la philosophie de l'architecture avec pipeline. Partant des données géométriques et de textures stockées en mémoire principale, le chemin qui mène à l'image finale est divisé en plusieurs étages d'un pipeline (Figure 5.4).

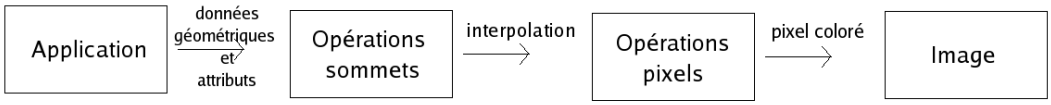

FIG. 5.4 – Pipeline du rendu par carte vidéo.

1. La première étape consiste à envoyer les données à la carte vidéo à partir de l'application principale. Ces données sont sous forme de sommets avec des attributs tels que la couleur, les coordonnées de texture et la normale.
2. La deuxième étape gère les sommets. Chaque sommet subit un ensemble de transformations qui le repositionne dans l'espace de l'image (projection) et ses attributs peuvent aussi être changés par un nuanceur de sommets.
3. La troisième étape est l'interpolation par un rastériseur. Cette unité s'occupe de transformer chaque primitive triangle en un ensemble de fragments. Les propriétés de chaque fragment sont interpolées entre les trois sommets du triangle correspondant.
4. La quatrième étape transforme chaque fragment en un pixel dans l'image. Un nuanceur de pixels peut transformer la couleur finale du pixel en fonction des propriétés du fragment reçu en entrée et en échantillonnant des textures.

Cette architecture en pipeline permet un rendu très efficace car pendant que certains pixels sont en train d'être modifiés dans le nuanceur de pixels, d'autres sommets peuvent entrer dans le pipeline, favorisant le parallélisme. Aussi, plusieurs pipelines de sommets ou de pixels peuvent être installés sur la carte vidéo pour pouvoir transformer plusieurs sommets ou pixels en même temps.

5.2.2 Application

Notre système utilise OpenGL comme API de rendu [Ope]. Au chargement, l'application principale extrude la coquille du maillage de base et calcule les informations sur les plans, les matrices de transformation espace monde à espace texture. Ces informations sont stockées dans une liste d'affichage OpenGL [WNDS03] en utilisant les registres d'attributs des sommets (Tableau 5.1).

L'application s'occupe aussi de la gestion des tampons de rendu. Si l'algorithme de découpage en couches de profondeur est utilisé, la stratégie de ping-pong entre les

Registre matériel	Attribut associé au sommet
POSITION	position monde
NORMAL	normale
TEXCOORD0	coordonnées texture 3D
TEXCOORD1	équation du premier plan
TEXCOORD2	équation du deuxième plan
TEXCOORD3	équation du troisième plan
TEXCOORD4	première rangée de la matrice $\mathbf{M}^{o \to t}$ du tétraèdre
TEXCOORD5	deuxième rangée de la matrice $\mathbf{M}^{o \to t}$ du tétraèdre
TEXCOORD6	troisième rangée de la matrice $\mathbf{M}^{o \to t}$ du tétraèdre
TEXCOORD7	tangente

TAB. 5.1 – Attributs d'un sommet

tampons de profondeur est gérée par copie d'un objet de texture à un autre. De plus, l'application gère un tampon de couleur en point flottant [ARB]. Plusieurs tétraèdres peuvent contribuer à la couleur d'un seul pixel dépendamment de l'opacité des voxels de la texture. Tel que noté par Krauss *et al.* [KQE04], des artefacts de mélange peuvent apparaître à cause d'un manque de précision dans un tampon de couleur traditionnel (8 bits en nombre entier par canal de couleur). En utilisant un tampon de couleur à point flottant (16 bits en point flottant par canal de couleur), nous réduisons fortement ces artefacts (Figure 5.5). Le coût en mémoire et en temps de calcul d'un tampon de couleur en point flottant est plus élevé qu'un tampon traditionnel et un tel tampon n'est pas toujours nécessaire (*e.g.* les hautes fréquences d'une texture masquent ces artefacts). Nous avons opté pour laisser cette fonctionnalité au choix de l'utilisateur.

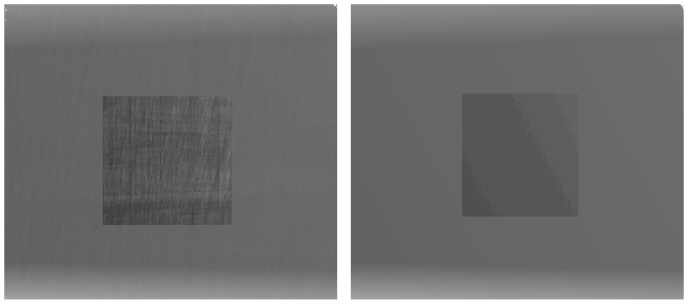

FIG. 5.5 – (gauche) Le mélange de couleur dans le tampon de couleur à nombres entiers amène des artefacts tandis que (droite) l'ajout du mélange dans un tampon en point flottant à notre technique réduit ces artefacts. Les zones centrales ont été retouchées pour augmenter le contraste à des fins de visualisation.

5.2.3 Nuanceur de sommets

Le nuanceur de sommets calcule l'intersection entre le rayon de vue et chacun des plans qui forment le tétraèdre (sauf le plan d'entrée). Premièrement, chaque intersection est calculée tel qu'expliqué plus haut (Section 5.1.1) en espace monde. Ensuite, ces trois intersections sont transformées en espace texture 3D pour interpolation par le rastériseur. La profondeur de l'intersection est utilisée pour déterminer laquelle de ces trois intersections est la plus proche (donc le véritable point de sortie du tétraèdre) dans le nuanceur de pixels.

Une subtilité importante subsiste. La distance d'intersection avec les autres plans n'est pas une propriété affine du triangle courant. C'est une propriété projective (Figure 5.1). Soit une propriété du triangle quelconque (position spatiale, coordonnées de texture, couleur, etc.), si r est la valeur de cette propriété et que p est la position du point courant sur le triangle, alors $r = \mathbf{M}p$ où $\mathbf{M}$ est une matrice de projection. La carte vidéo utilise un terme correctif pour les propriétés affines du triangle (*e.g.* coordonnées de texture) pour qu'après projection perspective, l'interpolation en espace image soit valide. L'interpolation en espace image de la profondeur d'intersection avec les autres plans donnerait un effet de glissement. Pour éviter ce problème, il faut multiplier les intersections trouvées par un facteur correctif de perspective supplémentaire à celui fourni par la carte. Nous avons multiplié les distances d'intersection par le rapport des profondeurs entre la position du sommet courant et la position de l'intersection. Soient $p = (p_x, p_y, p_z)$ la position du point sur le triangle en espace caméra (la caméra centrée à l'origine) et $i = (i_x, i_y, i_z)$ la position de l'intersection avec le plan opposé en espace caméra, le facteur correctif est p_z/i_z. Ce rapport est un terme valide pour l'interpolation de la coordonnée homogène résultant en une interpolation en espace image valide après projection perspective [HM91].

5.2.4 Nuanceur de pixels

Le nuanceur de pixels accumule la couleur et l'opacité pour un segment de rayon donné. Comme il est exécuté pour chaque tétraèdre projeté dans chaque pixel, la traversée des voxels dans la texture 3D doit être très efficace. Nous avons implanté plusieurs méthodes tout en effectuant des tests de performance. Premièrement, nous avons essayé un algorithme de traversée de grille 3D [AW87], puisque cet algorithme garan-

tit qu'aucun voxel n'est manqué par l'échantillonnage. Cependant, à cause de la haute résolution de certaines textures 3D (notre technique n'est pas limitée par la résolution de la texture), cet algorithme nécessite beaucoup de lectures de texture et offre donc de faibles performances (Figure 5.6). Le graphique démontre que l'échantillonnage sur une grille régulière donne des performances équivalentes avec de très petites textures seulement. L'échantillonnage uniforme donne des résultats similaires (jugé à l'oeil) pour de meilleures performances. Il est intéressant de noter que l'échantillonnage par traversée de grille dépend beaucoup du contenu de la texture. En effet, la texture *clôture* et la texture *fourrure* ont la même résolution, mais la texture *clôture* est beaucoup plus dense que la texture *fourrure*. La traversée de grille s'arrête plus rapidement dans ce cas et les performances sont meilleures. Dans le cas d'un échantillonnage uniforme, le contenu de la texture ne semble pas avoir autant d'impact. Il faut noter que la faible résolution de la texture *jello* avantage largement l'échantillonnage par traversée de grille. Aussi, la valeur pour un échantillonnage uniforme de 10 échantillons pour la texture *arbre* a été mise pour comparaison, mais dans ce cas précis, la qualité du résultat visuel était largement en deçà des autres méthodes.

Nous avons aussi essayé un échantillonnage adaptatif qui prend en considération la longueur du segment de rayon dans l'espace volumétrique de la texture et la résolution de la texture combinées à une approche pour sauter par dessus les voxels vides [Don05]. La première approche nécessite un nombre variable d'itérations dans la boucle principale et la deuxième approche requiert beaucoup de lectures de texture dont les coordonnées dépendent d'autres lectures de texture. C'est pourquoi dans notre implémentation finale, nous fixons le nombre d'échantillons pour un objet. Le nombre d'échantillons est un compromis entre la vitesse de rendu et la qualité et peut être choisi à l'exécution du programme. Notons aussi que la technique présentée dans ce mémoire, contrairement à la technique de Donelly [Don05], utilise des textures 3D. L'utilisation des textures 3D dans les cartes vidéo est plus lente que l'utilisation des textures 2D. Le système de cache n'est pas aussi performant. Cela explique peut-être pourquoi une technique adaptative fonctionne bien en 2D, mais moins bien dans notre cas 3D.

Dans notre implémentation, chaque voxel de la texture 3D a une normale associée stockée dans une autre texture 3D (une carte de normales 3D). Les normales peuvent être définies par un artiste où aussi échantillonnées lors de la construction de la texture

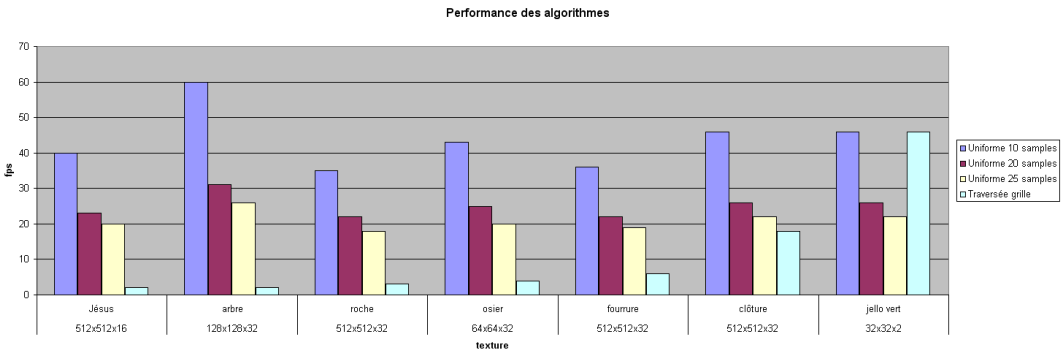

FIG. 5.6 – Comparaison des algorithmes d'échantillonnage des voxels.

3D. Lors de l'étape du rendu, un système d'axes local est interpolé entre les sommets des triangles pour l'illumination. En transformant la normale stockée dans la texture et le vecteur vers la lumière dans le système d'axes local, une fonction de réflectance isotropique peut être évaluée par pixel.

5.3 Applications

5.3.1 Effets de rendu

Tel que mentionné dans la Section 5.2.4, le premier effet est la réflectance par pixel. Chaque voxel a une normale associée dans une autre texture 3D. Lors du rendu, cette normale est utilisée pour calculer la réflectance des détails de surface par pixel. Les sommets du modèle 3D ont un système d'axes (tangente, normale, binormale) ($\hat{T} = (T_x, T_y, T_z), \hat{B} = (B_x, B_y, B_z), \hat{N} = (N_x, N_y, N_z)$). Avec une transformation matricielle, la direction de la lumière $\hat{L}$ est transformée dans ce système d'axes local avec l'équation suivante :

$$\hat{L}_{local} = \begin{pmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{pmatrix} \hat{L}.$$

Le rastériseur interpole la direction de lumière par pixel. Dans notre implémentation, nous utilisons la BRDF constante lambertienne. L'illumination lambertienne s'exprime par l'équation

$$I_{\text{lumière}} = \max(0, \hat{N} \cdot \hat{L}).$$

Dans notre nuanceur de pixels, l'opération suivante est effectuée pour calculer la lumière réfléchie

$$I_{\text{lumière}} = \max(0, (0, 0, 1) \cdot \hat{L}_{local}).$$

Pour obtenir des ombres, nous avons utilisé une carte d'ombrage. La position monde du point d'entrée du rayon dans le tétraèdre et du point de sortie sont connues. Dans la phase d'échantillonnage sur le rayon, nous utilisons une position monde interpolée entre ces deux valeurs pour estimer la position monde de l'échantillon. Cette position estimée sert de valeur pour tester la présence de lumière dans la carte d'ombrage. Cette méthode nous permet de faire des ombres projetées par les détails de surface, sur les détails de surface ou sur d'autres objets de la scène. De plus, nous pouvons faire des ombres volumétriques puisque les échantillons sont pris dans une texture semi-transparente. Finalement, l'interpolation de la position des échantillons nous permet d'utiliser le tampon de profondeur de la carte vidéo comme algorithme de visibilité entre les primitives pour produire du masquage visuel.

D'autres effets peuvent être rendus comme l'émission de lumière ou l'absorption puisque l'accumulation de l'opacité et de la couleur se fait dans l'espace volumétrique de la texture 5.7. Le canal alpha de la texture 3D nous permet de définir des textures où l'absorption de lumière n'est pas uniforme, résultant en des régions de texture où l'opacité varie. Les canaux de couleur peuvent être utilisés pour marquer des voxels spéciaux qui ajoutent de la lumière à l'accumulation pour donner un effet de lueur.

5.3.2 Filtrage

Le filtrage de détails de surface a souvent été négligé dans les travaux reliés. Cependant, l'absence de filtrage résulte en des artefacts de sous-échantillonnage très apparents (Figure 5.11). Le filtrage permet de réduire les hautes fréquences d'un signal, une texture dans notre cas, pour qu'un faible échantillonnage de celle-ci dans le rendu soit suffisant. S'il reste des hautes fréquences dans la texture et que nous n'échantillons pas suffisamment cette dernière, les fréquences se traduisent par du bruit à l'image (Figure 5.8). En placage de texture, l'idéal est d'avoir un ratio de un pour un entre les pixels de l'image et les texels de la texture. En 2D, on peut combiner les texels et en faire la moyenne pour avoir une représentation à plusieurs échelles. En 3D, le filtrage valide des détails est souvent très difficile à calculer à cause de la dépendance de l'angle de vue et le masquage dans la texture volumétrique. Nous ignorons cette dépendance et faisons un filtrage basé sur le *mipmap* similaire au travail de Décaudin et Neyret [DN04].

Le *mipmap* est une structure pyramidale qui stocke une texture 2D filtrée à plusieurs

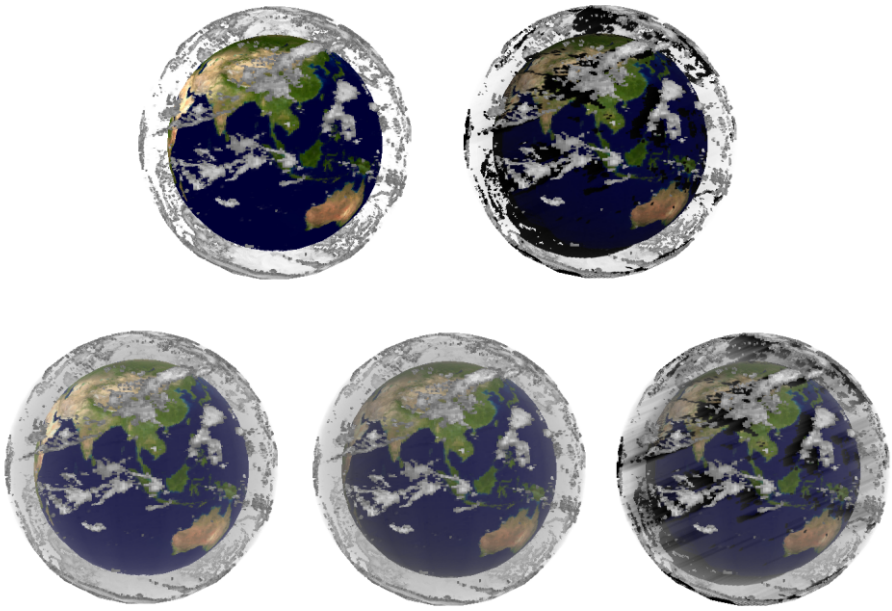

FIG. 5.7 – Différents effets rendus sur la terre. Première rangée, de gauche à droite : simple rendu de la texture avec masquage visuel ; ajout des ombres projetées par les nuages sur la terre. Deuxième rangée, de gauche à droite : ajout de nuages semi-transparents avec absorption de lumière ; modification de la réflectance de la terre, les zones moins éclairées deviennent plus sombres ; ombres volumétriques projetées dans les nuages semi-transparents.

niveaux de détails. Le premier niveau de *mipmap* contient la texture originale et les niveaux supérieurs contiennent des versions filtrés de la texture (Figure 5.9). À l'étape du rendu, la lecture de texture se fait au niveau du *mipmap* qui conserve le ratio grosseur de pixel/grosseur de texel le plus près de 1. Cette façon de filtrer réduit plusieurs artefacts de sous-échantillonnage dus au placage de texture. Même si le *mipmap* est valide dans une seule direction de vue (orthographique et perpendiculaire à la texture) et suppose un filtre boîte, il reste une méthode de filtrage efficace accélérée en matériel graphique et facile à utiliser.

En commençant par le plus bas niveau (le plus détaillé) de la pyramide de filtrage, nous combinons les couches de la texture 3D en utilisant l'équation d'opacité (Équation 5.1) pour filtrer dans la direction de la dimension z (hauteur) de la texture. Nous faisons aussi la moyenne des couleurs des voxels que nous combinons dans les deux autres dimensions (horizontal) comme il est fait dans un *mipmap* ordinaire. Même

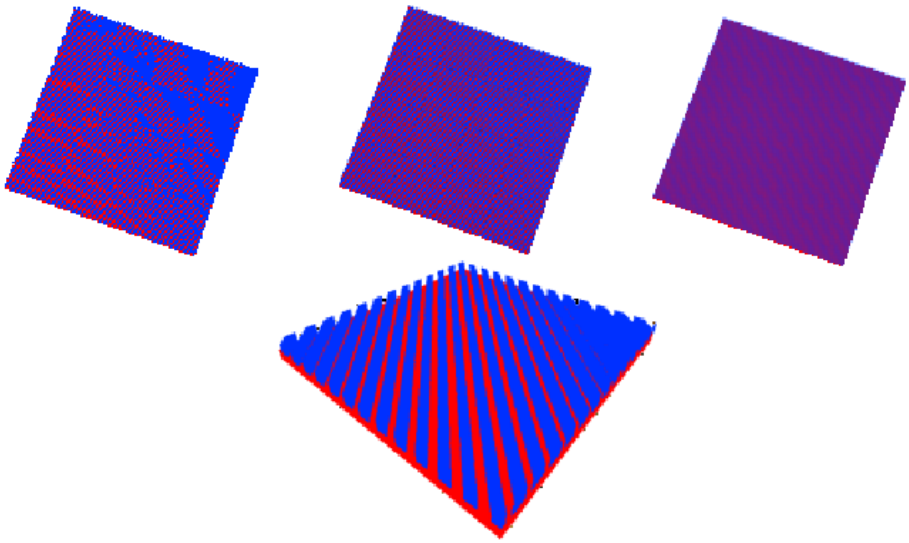

FIG. 5.8 – De gauche à droite, trois niveaux de filtrage. Le matériau est une série de bandes diagonales bleues sur fond rouge.

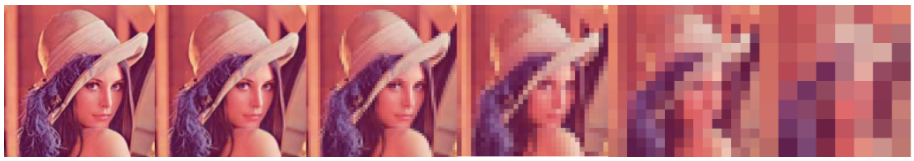

FIG. 5.9 – *Mipmap*.

sans la dépendance à l'angle de vue, ce type de filtrage donne des résultats satisfaisants qui réduisent la plupart des artefacts de sous-échantillonnage. Nous laissons en travaux futurs la recherche pour trouver une méthode qui donnera un meilleur filtrage.

Combiner des voxels vides et des voxels pleins résulte en des voxels semi-transparents. Dans les techniques classiques qui ne traitent pas la semi-transparence, un seuil est établi et les voxels dont la valeur alpha est en dessous du seuil sont considérés vides. Le problème est que tranquillement, la texture devient de plus en plus transparente et donc disparaît de l'image au fur et à mesure qu'elle s'éloigne. Dans notre cas, les voxels semi-transparents sont correctement traités par notre algorithme tel qu'illustré dans la Figure 5.10 en ajoutant leur contribution de couleur et d'opacité au pixel final.

5.3.3 Animation

Un des avantages de notre algorithme de rendu est de permettre l'animation du modèle géométrique de base ainsi que de la texture.

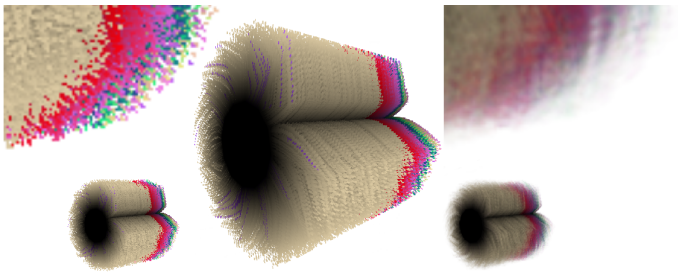

FIG. 5.10 – Un cylindre duquel émerge de la fourrure de couleur avec une fente. Notre algorithme de filtrage partiel réduit la plupart des artefacts (gauche) de sous-échantillonnage (droite).

FIG. 5.11 – Démonstration de filtrage sur un champignon avec une texture de bois déformé. Au centre, le champignon est près de la caméra alors l'échantillonnage de la texture est grand. En éloignant le champignon de la caméra, l'échantillonnage de la texture devient plus faible, résultant en du bruit (gauche). En petit, le champignon tel que rendu dans l'image et en grand, gros plan sur le bruit. Notre algorithme filtre la plupart de ces artefacts (droite).

La déformation du modèle de base modifie les propriétés géométriques de l'extrusion et change l'apparence de la texture 3D plaquée (Figure 5.12). Dans un système complet, les tétraèdres qui sont modifiés par une déformation du modèle doivent être mis-à-jour. Comme les tétraèdres sont déjà créés, il suffit de lever un drapeau pour chaque tétraèdre qui a été modifié. Dans une seconde passe, nous recalculons, pour chaque tétraèdre modifié, les équations des plans des triangles qui le constituent ainsi que la matrice de transformation $\mathbf{M}^{o \to t}$. Dans le cas d'un système d'animation par images clés, ces informations peuvent être pré-calculées pour chaque image clé dans une séquence d'animation et interpolées lors du rendu pour les images intermédiaires. Dans le cas d'un système interactif et dynamique, il faut bien identifier les tétraèdres qui sont modifiés pour ne pas mettre à jour tous les tétraèdres de la scène. Aussi, après déformation, un

algorithme peut s'assurer en temps interactif qu'aucun tétraèdre est intersecté par un tétraèdre voisin [PKZ04].

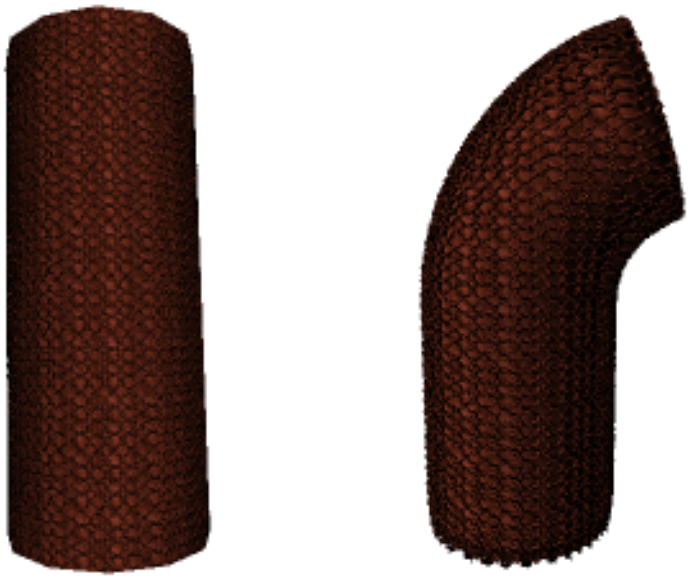

FIG. 5.12 – Le tube subit une déformation géométrique.

Un autre type d'animation que notre méthode permet est la transformation de la texture. Les éléments d'une texture peuvent être animés soit en fournissant à la carte vidéo une texture différente pour chaque image de la texture animée ou en changeant la fonction de correspondance entre les coordonnées objet et coordonnées texture. Une telle transformation permet des déformations de la paramétrisation de la texture et simule un déplacement des éléments de texture sur l'objet. Finalement, l'opacité des voxels peut aussi être modifiée permettant de mettre en valeur certaines parties de la texture (Figure 5.13).

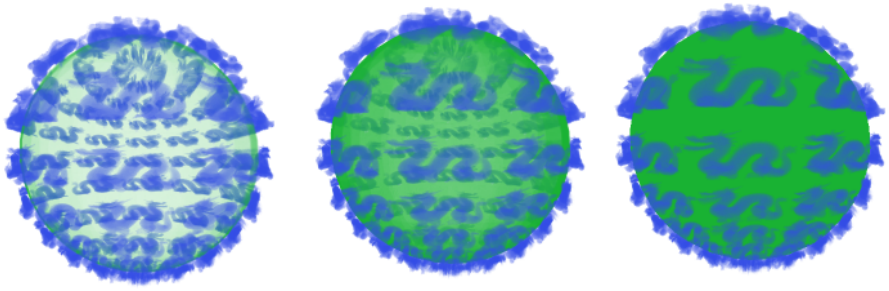

FIG. 5.13 – La partie verte du matériau devient plus en plus opaque dans l'animation.

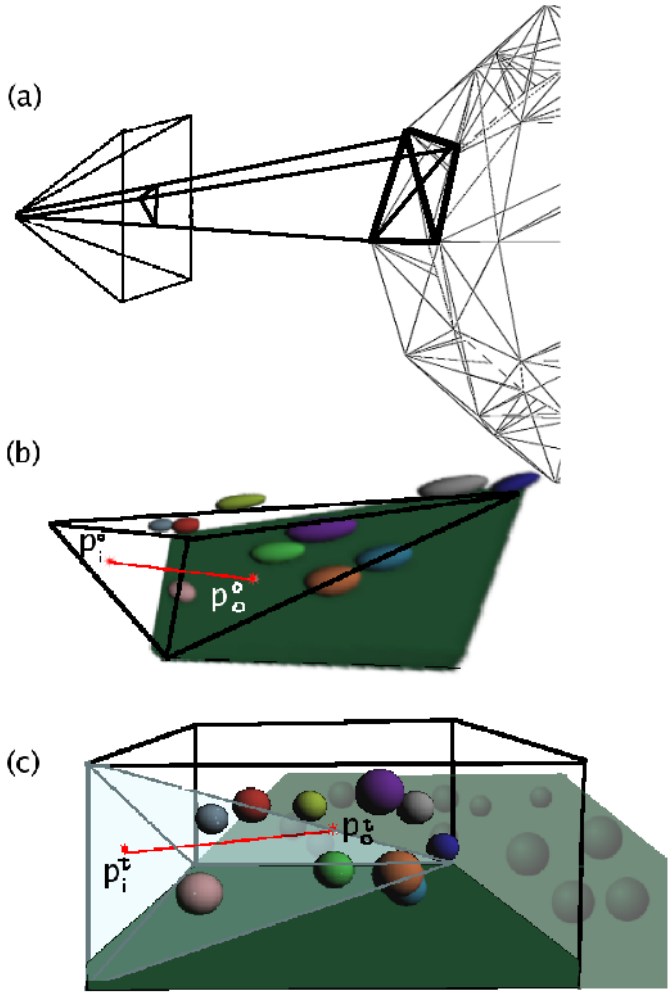

FIG. 5.14 – Aperçu de l'algorithme. (a) Pour les textures semi-transparentes, les tétraèdres sont triés. Un tétraèdre projeté dans un pixel définit un point d'entrée dans le tétraèdre p_i^o. (b) Le point de sortie p_o^o est déterminé en calculant l'intersection du rayon de vue avec les autres plans du tétraèdre. (c) Le segment p_o^o - p_i^o est transformé en espace texture 3D. Finalement, la texture 3D est échantillonnée le long de ce segment pour accumuler la couleur et l'opacité avant d'ajouter la contribution de ce tétraèdre dans le pixel par mélange de valeur alpha.

Chapitre 6

Résultats

La plus grande victoire est la victoire sur soi.

Platon

6.1 Résultats

Toutes nos images ont été rendues en résolution 512×512 à partir de notre implémentation avec une carte vidéo nVidia 6800 GT possédant 256 Mo de mémoire vidéo. La machine principale roulait Linux sur un microprocesseur Athlon64 3500+.

Notre approche a un certain nombre d'avantages sur les approches précédentes dans le rendu de détails de surface. Premièrement, nous pouvons faire le rendu de textures semi-transparentes (Figure 6.2), ce qui est nécessaire lorsque nous faisons du filtrage de voxels dans une représentation multiéchelles. Cette généralisation des détails de surface aux textures semi-transparentes se réalise au coût supplémentaire d'un tri lors du rendu. Cette étape critique est la plus coûteuse de notre système et les performances en souffrent comparativement aux techniques se limitant aux textures opaques. Ces dernières ne calculent pas d'absorption de lumière et peuvent donc précalculer la visibilité. De plus, notre technique ne se limite pas aux détails de surface représentables par une carte de déplacement. Elle permet d'utiliser n'importe quelle texture 3D. Les résultats montrent clairement les silhouettes détaillées que notre technique rend pour toutes les textures 3D (Figure 6.3). Finalement, comme notre méthode ne nécessite presque pas de précalcul, le modèle de base ainsi que la texture peuvent être animés (Figure 6.5).

Le coût en mémoire du système final est réparti entre les données du maillage extrudé

(les tétraèdres) et la texture 3D. Tel que mentionné dans la Section 3.3, chaque triangle du maillage de base est extrudé en trois tétraèdres. Ainsi, chaque sommet est dupliqué et le nombre de triangles est multiplié par 12 (les triangles partagés entre les tétraèdres sont comptés deux fois). Ce coût en mémoire (linéaire du nombre de triangles de base) est comparable aux autres méthodes de détails de surface qui se basent sur une extrusion du maillage de base [WTL+04]. Cependant, le coût pour stocker les données de la texture est moindre avec notre méthode puisque nous ne stockons pas de fonction de visibilité. Nous pouvons utiliser des textures à haute résolution sans stocker aucune autre information que la texture elle-même ($x \times y \times z \times RGB\alpha$) et son équivalent pour les normales.

La Figure 6.4 (gauche) montre une application de notre méthode sur un modèle très polygonisé (8640 triangles modèle de base, 25920 tetraèdres en tout) sur lequel nous avons plaqué une texture d'arbre (résolution : $128 \times 128 \times 32$). Le rendu avec la carte vidéo se fait au rythme de 2 images par seconde. La polygonisation permet d'utiliser notre méthode avec des objets courbes. Dans ce cas précis, la texture est opaque donc l'étape du tri n'est pas nécessaire. Pour augmenter les performances, nous avons essayé une stratégie multi-passes. La première passe sert à rendre le maillage de base (opaque) et la seconde passe utilise le test de rejet de fragments précoce sur la profondeur z du fragment pour rejeter les fragments cachés. Les gains de performance sont négligeables.

La même texture a été appliquée sur une modèle plus simple, mais cette fois, en ajoutant de la semi-transparence sous forme de brouillard. Le brouillard ajoute de l'atmosphère à la scène (Figure 6.4 (droite)). Le maillage extrudé contient 96 tétraèdres. À chaque image, les tétraèdres sont triés et l'algorithme d'échantillonnage prend 15 échantillons dans la texture le long d'un segment de rayon. Dans cet exemple, 22% du temps est passé au tri et le reste (88%) est utilisé pour le rendu. Le rythme de rendu est de 16 images par seconde. Cet exemple démontre des effets de rendu tels que le masquage visuel, l'absorption de lumière par le medium semi-transparent et aussi des ombres volumétriques dans le brouillard.

Finalement, bien que notre technique permette le rendu de textures semi-transparentes, elle reproduit aussi des résultats similaires aux méthodes déjà publiées dans le cas de textures opaques (Figure 6.5).

Finalement, les Figures 6.5 et 6.6 montre des images tirées d'une animation. Dans le

Scène	Figure	# tri.	Rés. texture	# échan.	img. sec.
Théière (Beethoven)	Figure 6.2	25704	$128 \times 128 \times 32$	10	4
Théière (osier)	Figure 6.3	25704	$64 \times 64 \times 32$	20	5
Vase (arbres)	Figure 6.4	103680	$128 \times 128 \times 32$	10	2
Vase (dragons)	Figure 4.7	103680	$128 \times 128 \times 32$	10	2
Plan (arbres)	Figure 6.4	384	$128 \times 128 \times 32$	15	16
Tube (roches)	Figure 6.5	1440	$512 \times 512 \times 32$	10	30

TAB. 6.1 – Résumé de nos résultats. Le nombre de triangles est le total des triangles envoyés à la carte vidéo : nombre de triangles par maillage de base $\times$ 3 tétraèdres par triangle de base $\times$ 4 triangles par tétraèdre.

premier cas, la texture est animée en changeant l'opacité des voxels de fourrure. Dans le deuxième cas, le maillage de base est modifié en appliquant une transformation de bruit sur ses sommets. Une séquence vidéo montrant une séance de l'utilisation de la technique avec des animations de textures et de modèles est disponible sur le site associé à ce mémoire http ://www.iro.umontreal.ca/labs/infographie/theses/dufortjf.

Le Tableau 6.1 résume nos résultats et la Figure 6.1 montre un rendu de différents matériaux pour illustrer la généralité de la technique.

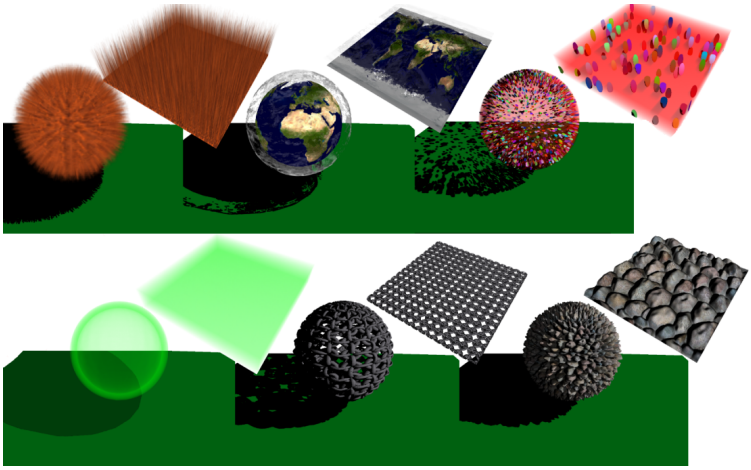

FIG. 6.1 – Sphères avec différentes textures 3D rendues en temps interactif. Première rangée, de gauche à droite : fourrure semi-transparente ; nuages semi-transparents projetant des ombres sur la terre ; boules dans un matériau rouge semi-transparent avec des ombres volumétriques. Deuxième rangée, de gauche à droite : matériau uniforme vert absorbant ; chaîne opaque ; roches.

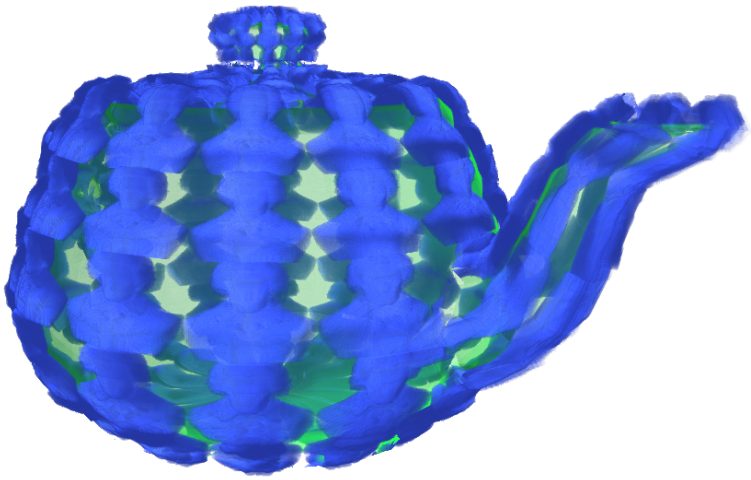

FIG. 6.2 – Modèle de théière (6426 tétraèdres) avec une texture semi-transparente de Beethoven ($128 \times 128 \times 32$). Remarquez les silhouettes détaillées et le mélange de couleur dû à la semi-transparence.
4 images par seconde avec 10 échantillons par segment de rayon.

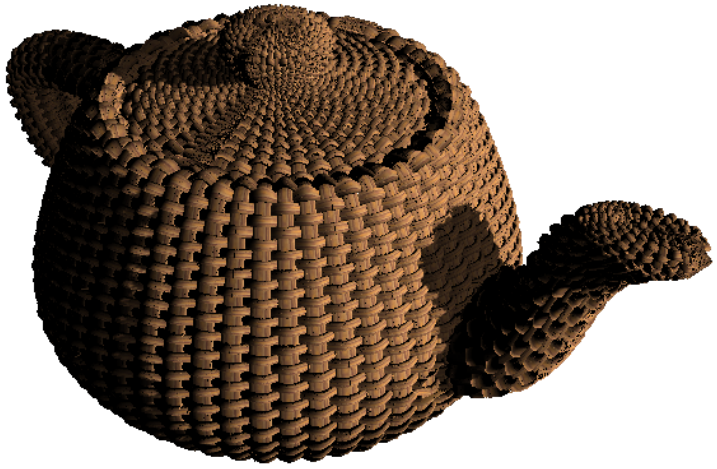

FIG. 6.3 – Modèle de théière avec un point de vue différent et une texture opaque seulement ($64 \times 64 \times 32$). Les ombres ont été ajoutées à l'effet de masquage visuel.
5 images par seconde avec 20 échantillons de texture par segment de rayon.

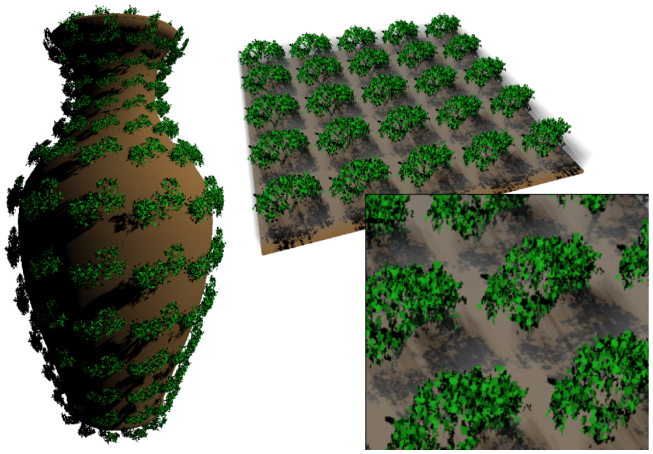

FIG. 6.4 – Modèle du vase (25920 tétraèdres) avec une texture opaque répétée d'un arbre ($128 \times 128 \times 32$). Même sur les objets courbes, notre algorithme rend les ombres correctement. Ceci serait impossible avec une méthode qui estime une planarité locale d'un objet. Aussi, puisque les arbres croissent à l'extérieur du vase, la condition de planarité locale devient de plus en plus fausse avec l'augmentation de l'extrusion ce qui produirait des ombres avec une mauvaise déformation. Notre méthode traite les ombres dans un espace projectif et le résultat est cohérent avec la courbure de l'objet.

FIG. 6.5 – Extrême gauche : Des détails de roches appliqués sur un cylindre avec notre technique. Rendu à 30 images par seconde en prenant 10 échantillons de texture par segment de rayon. Les trois images suivantes sont tirées d'une animation où le tube subit une déformation de bruit sur ses sommets.

FIG. 6.6 – Une sphère avec de la fourrure qui pousse et qui change de densité. Les textures dynamiques sont automatiquement traitées par notre méthode.

Chapitre 7

Conclusion

L'avenir contient de grandes occasions. Il révèle aussi des pièges. Le problème sera d'éviter les pièges, de saisir les occasions et de rentrer chez soi pour six heures.

Woody Allen

Le rendu de détails de surface contribue dans une image virtuelle qui à ajouter beaucoup de réalisme aux objets. Plusieurs algorithmes existent déjà pour le rendu de tels détails. Les travaux récents essaient de transporter les algorithmes coûteux pour le rendu haute qualité vers un rendu en temps réel.

Ce mémoire décrit une technique de rendu de détails de surface. Contrairement aux autres techniques similaires, l'algorithme décrit ne suppose rien sur les détails qui peuvent représenter n'importe quelle méso-structure dans une texture 3D, incluant les détails semi-transparents. Au coût de perdre le temps réel (notre implémentation offre des temps plutôt interactif), la technique offre une généralité nécessaire au rendu de détails de surface. En effet, la semi-transparence a souvent été ignorée dans les algorithmes de rendu de détails de surface. Nous savons pourtant que l'opération de filtrage de méso-structures introduit de la semi-transparence et l'ignorer produit nécessairement une image *erronée*. Finalement, la représentation choisie pour les détails est la texture 3D qui offre plus de flexibilité puisqu'elle peut rendre des détails non représentable par une carte de hauteur. Notre technique peut donc reproduire les résultats des autres techniques tout en faisant le rendu de détails non traitables par les autres algorithmes.

Dans ce projet de recherche, nos contributions ont été

1. La création d'un système de rendu de détails de surface opaques en utilisant des techniques courantes et récentes (cartes de déplacement, carte de reliefs, etc.), rapides et efficaces implantées dans des nuanceurs pour carte vidéo.
2. L'extension des techniques de rendu courantes aux textures 3D permettant de sortir du moule des cartes de hauteur.
3. Un sous-système de traitement des textures 3D semi-transparentes qui comprend
   (a) L'implémentation de deux algorithmes de tri de primitives graphiques pour le rendu ordonné des primitives semi-transparentes.
   (b) Un schéma d'intégration de l'opacité dans le nuanceur de pixels.
4. L'ajout d'effets de rendu au système tels que de l'illumination locale et des ombres projetées.
5. Une démonstration des possibilités offertes par un tel système à travers des applications importantes de l'infographie
   (a) filtrage de méso-structure
   (b) animation.
6. La transformation de modèles 3D en textures 3D pouvant être plaquées sur un maillage.
7. La publication de la méthode et des résultats obtenus suite à nos expérimentations [DLP05].

La nécessité de rendre les détails semi-transparents a déjà été montrées dans le passé. Le rendu rapide de ces détails devient un problème intéressant auquel il est impératif que nous trouvions une solution. L'assemblage de ces composantes dans notre système final donne des résultats satisfaisants en temps interactif. Le rendu de haute qualité offre des résultats supérieurs en terme de qualité visuelle, mais plus coûteux en temps de plusieurs ordres de grandeur. Une autre approche a été développée de façon concurrente [PBFJ05] dans laquelle une stratégie similaire d'intégration de prismes a été proposée. Malgré les différences au niveau de l'intégration de l'opacité, la différence majeure entre notre algorithme et celui présenté par Porumbescu *et al.* est que ce dernier vise le rendu haute qualité et tout le rendu est fait dans un logiciel pour le processeur central. Dans notre cas, le rendu est fait sur la carte vidéo pour de meilleures performances d'au moins deux ordres de grandeur (moins d'une seconde par image plutôt que minutes).

7.1 Perspectives futures

D'autres améliorations sont à faire dans des travaux futurs.

D'abord, l'ombrage peut être amélioré en utilisant un meilleur traitement des normales dans le rendu. Nous croyons qu'une distribution de normales [Ney95] serait nécessaire pour éviter des artefacts d'ombrage quand la normale stockée dans la texture est utilisée pour faire l'illumination locale des détails. Cette distribution s'insère parfaitement dans la recherche d'un autre algorithme de filtrage. Dans ce mémoire, nous nous sommes concentrés sur le rendu des détails et nous avons mis de côté la création des détails. Aussi, l'algorithme de filtrage que nous avons utilisé affiche l'intérêt de notre méthode pour cette application, mais reste invalide en soi. Nous croyons que le développement d'un algorithme de filtrage valide pour les détails de surface est un projet qui mérite beaucoup d'attention.

Pour créer nos résultats, certaines textures étaient construites à la main et d'autres étaient créées à partir de modèles 3D. La voxelisation de modèles 3D est un domaine qui pourrait contribuer de façon significative à notre projet puisque la création de textures 3D deviendrait plus intuitive pour un artiste.

Malgré les avancées que nous amenons dans ce mémoire, il serait intéressant de pousser d'avantage l'analyse des performances. Bien que nous offrons des performances en temps interactif, notre algorithme ne peut pas être utilisé actuellement dans une application en temps réel. De façon contre-intuitive, nos tests de performance sur des algorithmes plus efficaces sur un processeur central pour l'intégration de l'opacité se sont avérés négatifs sur la carte vidéo. Nous comprenons que l'architecture différente des deux processeurs nous oblige à repenser à notre mentalité de programmation. Aussi, en choisissant d'utiliser les textures 3D, nous savions que nous utilisions une fonctionnalité moins performante que les textures 2D sur les cartes vidéo. Finalement, une optimisation de nos nuanceurs pourrait encore augmenter les performances.

Le concept de texture 3D pour remplacer les détails de surface géométriques est connu. En plus de réduire la quantité d'information envoyée à la carte vidéo, la texture offre la capacité de préfiltrer les détails de texture. Cependant, nous aimerions pouvoir insérer notre projet dans un cadre de rendu avec niveaux de détails. Comme notre algorithme est coûteux, il serait intéressant de passer à un autre algorithme quand ce n'est plus nécessaire d'avoir des détails semi-transparents aussi précis. Une étude sur

les niveaux de détails hiérarchiques serait intéressante et ce projet s'insère dans cette optique.

En conclusion, nous croyons que le rendu de détails de surface semi-transparents est important. Notre avons présenté une technique pour rendre ces détails en temps interactif. Nos résultats démontrent qu'il est possible d'obtenir des résultats satisfaisants sans faire de précalcul de visibilité, offrant la possibilité de faire de l'animation et d'avoir des textures dynamiques. Malgré le coût associé augmenté des textures semi-transparentes dans notre technique, nous croyons qu'un système de rendu de détails semi-transparents sera nécessaire dans les applications de demain. Déjà, plusieurs travaux sont présentés pour le rendu de détails opaques en temps réel. La semi-transparence arrivera sous peu et nous offrons déjà une solution partielle à ce problème.

Annexe A

Correspondance termes français et anglais

- ***Apprendre un modèle*** : model fitting
- ***Analyse des composantes principales*** : principal component factors
- ***Carte de déplacement*** : displacement mapping
- ***Carte de relief*** : height field
- ***Découpage en couches de profondeur*** : depth peeling
- ***Image clé*** : keyframe
- ***Lissage linéaire au sens des moindres carrés*** : least square linear fitting
- ***Marche sur le rayon*** : ray marching
- ***Mélange alpha*** : alpha blending
- ***Nuanceur*** : shader
- ***Nuanceur de fragments*** : fragment shader
- ***Nuanceur de sommets*** : vertex shader
- ***Ombrage*** : shading
- ***Placage de reliefs*** : bump mapping
- ***Placage de reliefs avec parallaxe*** : parallax mapping
- ***Placage de textures*** : texture mappign
- ***Rastérisation*** : rasterization
- ***Sur-échantillonner*** : to oversample
- ***Tracer de rayons*** : raytracing

Bibliographie

[ARB] ARB_color_buffer_float. oss.sgi.com/projects/ogl-sample/registry/ARB/color_buffer_float.txt.

[AW87] John Amanatides et Andrew Woo. « A fast voxel traversal algorithm for ray tracing ». Dans *Eurographics '87*, pages 3–10, août 1987.

[Bli78] James F. Blinn. « Simulation of Wrinkled Surfaces ». Dans *Proceedings of SIGGRAPH 1978*, pages 286–292. ACM, ACM Press / ACM SIGGRAPH, 1978.

[Car84] Loren Carpenter. « The A-buffer, an antialiased hidden surface method ». Dans *SIGGRAPH '84 : Proceedings of the 11th annual Conference on Computer graphics and interactive techniques*, pages 103–108, New York, NY, USA, 1984. ACM Press.

[CC05] Steven P. Callahan et Joao L. D. Comba. « Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering ». *IEEE Transactions on Visualization and Computer Graphics*, volume 11, numéro 3, pages 285–295, 2005.

[CMSW04] Richard Cook, Nelson Max, Claáudio T. Silva et Peter L. Williams. « Image-space visibility ordering for cell projection volume rendering of unstructured data ». *IEEE Transactions on Visualization and Computer Graphics*, volume 10, numéro 6, pages 695–707, 2004.

[Coo84] Robert L. Cook. « Shade Trees ». Dans *Proceedings of SIGGRAPH 1984*, pages 223–231. ACM, ACM Press / ACM SIGGRAPH, 1984.

[CTW+04] Yanyun Chen, Xin Tong, Jiaping Wang, Stephen Lin, Baining Guo et Heung-Yeung Shum. « Shell Texture Functions ». Dans *Proceedings of*

SIGGRAPH 2004, pages 343–353. ACM, ACM Press / ACM SIGGRAPH, 2004.

[DH00] Michael Doggett et Johannes Hirche. « Adaptive view dependent tessellation of displacement maps ». Dans *Proceedings of SIGGRAPH 2000*, pages 59–66. ACM, ACM Press / ACM SIGGRAPH, 2000.

[DLP05] Jean-François Dufort, Luc Leblanc et Pierre Poulin. « Interactive Rendering of Meso-structure Surface Details using Semi-transparent 3D Textures ». Dans *Proceedings Vision, Modeling, and Visualization 2005*, pages x–x, novembre 2005. À paraître.

[DN04] Philippe Decaudin et Fabrice Neyret. « Rendering Forest Scenes in Real-Time ». Dans H. W. Jensen et A. Keller, éditeurs. *Rendering Techniques '04 (Eurographics Symposium on Rendering)*, pages 93–102, juin 2004.

[Don05] William Donelly. *GPU Gems 2 : Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapitre 8. Addison Wesley, 2005.

[DvGNK99] Kristin J. Dana, Bram van Ginneken, Shree K. Nayar et Jan J. Koenderink. « Reflectance and texture of real-world surfaces ». *ACM Transactions on Graphics*, volume 18, numéro 1, pages 1–34, 1999.

[EKE01] Klaus Engel, Martin Kraus et Thomas Ertl. « High-quality pre-integrated volume rendering using hardware-accelerated pixel shading ». Dans *HWWS '01 : Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 9–16, New York, NY, USA, 2001. ACM Press.

[EMP+02] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin et Steven Worley. *Texturing & Modeling : A Procedural Approach*. Morgan Kaufmann, troisième édition, 2002.

[Eve99] Cass Everitt. « Interactive Order-Independant Transparency ». NVIDIA Corporation, White Paper, 1999.

[FK03] Randima Fernando et Mark J. Kilgard. *The Cg Tutorial : The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, 2003.

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

[FvDKH96] J. D. Foley, A. van Dam, Feiner S. K. et J. F. Hughes. *Computer Graphics, Principles and Practice Second Edition in C.* Addison-Wesley, deuxième édition, 1996.

[GGS03] Craig Gotsman, Xianfeng Gu et Alla Sheffer. « Fundamentals of spherical parameterization for 3D meshes ». *ACM Trans. Graph.*, volume 22, numéro 3, pages 358–363, 2003.

[Gre05] Simon Green. *GPU Gems 2 : Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapitre 26. Addison Wesley, 2005.

[HDKS00] Wolfgang Heidrich, Katja Daubert, Jan Kautz et Hans-Peter Seidel. « Illuminating micro geometry based on precomputed visibility ». Dans *SIGGRAPH '00 : Proceedings of the 27th annual Conference on Computer graphics and interactive techniques*, pages 455–464, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Hec86] Paul S. Heckbert. « Survey of Texture Mapping ». *IEEE Computer Graphics & Applications*, volume 6, numéro 11, pages 56–67, novembre 1986.

[HEG04] Johannes Hirche, Alexander Ehlert et Stefan Guthe. « Hardware Accelerated Per-Pixel Displacement Mapping ». Dans *Proceedings of Graphics Interface 2004*, pages 153–160. Canadian Information Processing Society, Canadian Human-Computer Communications Society / A.K. Peters LTD., 2004.

[HM91] Paul Heckbert et Henry Moreton. « Interpolation for Polygon Texture Mapping and Shading ». Dans *State of the Art in Computer Graphics Summer Institute*, pages 101–111. Springer-Verlag, 1991.

[HRS04] Markus Hadwiger et Christof Rezk-Salama. *Real-Time Volume Graphics.* SIGGRAPH 2004 Notes de cours #28, 2004.

[KK89] James T. Kajiya et Timothy L. Kay. « Rendering Fur with Three Dimensional Textures ». Dans *Proceedings of SIGGRAPH 1989*, pages 271–280. ACM, ACM Press / ACM SIGGRAPH, 1989.

[KN01] Tae-Yong Kim et Ulrich Neumann. « Opacity Shadow Maps ». Dans *Proceedings of Eurographics Workshop on Rendering Techniques*, pages 177–182, London, UK, 2001. Springer-Verlag.

[KPHE02] Joe Kniss, Simon Premoze, Charles Hansen et David Ebert. « Interactive translucent volume rendering and procedural modeling ». Dans *VIS '02 : Proceedings of the Conference on Visualization '02*, pages 109–116, Washington, DC, USA, 2002. IEEE Computer Society.

[KQE04] Martin Kraus, Wei Qiao et David S. Ebert. « Projecting tetrahedra without rendering artifacts ». Dans *IEEE Visualization 2004*, pages 27–34. IEEE, 2004.

[KS01] Jan Kautz et Hans-Peter Seidel. « Hardware accelerated displacement mapping for image based rendering ». Dans *Proceedings of Graphics Interface 2001*, pages 61–70. Canadian Information Processing Society, Canadian Human-Computer Communications Society / A.K. Peters LTD., 2001.

[KSS+04] Jan Kautz, Mirko Sattler, Ralf Sarlette, Reinhard Klein et Hans-Peter Seidel. « Decoupling BRDFs from surface mesostructures ». Dans *Proceedings of the 2004 Conference on Graphics Interface*, pages 177–182. Canadian Human-Computer Communications Society, 2004.

[KW03] Jens Krueger et Ruediger Westermann. « Acceleration Techniques for GPU-based Volume Rendering ». Dans *VIS '03 : Proceedings of the Conference on Visualization '03*. IEEE Computer Society, 2003.

[LDS02] Hendrik P.A. Lensch, Katja Daubert et Hans-Peter Seidel. « Interactive Semi-Transparent Volumetric Textures ». Dans *Proceedings of Vision, Modeling, and Visualization VMV 2002*, pages 505–512, 2002.

[Lev90] Marc Levoy. « Efficient Ray Tracing of Volume Data ». Dans *Proceedings of SIGGRAPH 1990*, pages 245–261. ACM, ACM Press / ACM SIGGRAPH, 1990.

[Lib] NOAA Photo Library. http ://www.photolib.noaa.gov.

[LV00] Tom Lokovic et Eric Veach. « Deep shadow maps ». Dans *SIGGRAPH 2000*, pages 385–392, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Max88] Nelson L. Max. « Horizon mapping : shadows for bump-mapped surfaces ». *The Visual Computer*, volume 4, pages 109–117, juillet 1988.

[MCT+05] Wan-Chun Ma, Sung-Hsiang Chao, Yu-Ting Tseng, Yung-Yu Chuang, Chun-Fa Chang, Bing-Yu Chen et Ming Ouhyoung. « Level-of-detail representation of bidirectional texture functions for real-time rendering ». Dans *SI3D '05 : Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, pages 187–194, New York, NY, USA, 2005. ACM Press.

[MGW01] Tom Malzbender, Dan Gelb et Hans Wolters. « Polynomial Texture Maps ». Dans *Proceedings of SIGGRAPH 2001*, pages 519–528. ACM, ACM Press / ACM SIGGRAPH, 2001.

[MM02] Kevin Moule et Michael McCool. « Efficient Bounded Adaptive Tessellation of Displacement Maps ». Dans *Proceedings of Graphics Interface 2002*. Canadian Information Processing Society, Canadian Human-Computer Communications Society / A.K. Peters LTD., 2002.

[MMS+05] G. Müller, J. Meseth, M. Sattler, R. Sarlette et R. Klein. « Acquisition, Synthesis and Rendering of Bidirectional Texture Functions ». *Computer Graphics Forum*, volume 24, numéro 1, pages 83–109, Mar 2005.

[MN98] Alexandre Meyer et Fabrice Neyret. « Interactive Volumetric Textures ». Dans *Eurographics Rendering Workshop 1998*, pages 157–168. Eurographics, Springer Wein, 1998.

[NBM00] Manuel M. Oliveira Neto, Gary Bishop et David McAllister. « Relief texture mapping ». Dans *Proceedings of SIGGRAPH 2000*, pages 359–368. ACM, ACM Press / ACM SIGGRAPH, 2000.

[Ney95] Fabrice Neyret. « A General and Multiscale Model for Volumetric Textures ». Dans *Proceedings of Graphics Interface 1995*, pages 83–91. Canadian Information Processing Society, Canadian Human-Computer Communications Society, 1995.

[Ney96] Fabrice Neyret. « Synthesizing Verdant Landscapes using Volumetric Textures ». Dans Xavier Pueyo et Peter Schröder, éditeurs. *Eurographics Rendering Workshop 1996*, pages 215–224, New York City, NY, juin 1996. Eurographics, Springer Wein. ISBN 3-211-82883-4.

[Ney98] Fabrice Neyret. « Modeling, Animating, and Rendering Complex Scenes using Volumetric Textures ». *IEEE Transactions on Visualization and Computer Graphics*, volume 4, numéro 1, pages 55–70, 1998.

[Ope] OpenGL. http ://www.opengl.org.

[PBFJ05] Serban D. Porumbescu, Brian Budge, Louis Feng et Kenneth I. Joy. « Shell Maps ». Dans *Proceedings of SIGGRAPH 2005*, pages 626–633. ACM, ACM Press / ACM SIGGRAPH, 2005.

[Pea85] Darwyn R. Peachey. « Solid texturing of complex surfaces ». Dans *SIGGRAPH '85 : Proceedings of the 12th annual Conference on Computer graphics and interactive techniques*, pages 279–286, New York, NY, USA, 1985. ACM Press.

[Per85] Ken Perlin. « An image synthesizer ». Dans *SIGGRAPH '85 : Proceedings of the 12th annual Conference on Computer graphics and interactive techniques*, pages 287–296, New York, NY, USA, 1985. ACM Press.

[Per02] Ken Perlin. « Improving noise ». Dans *SIGGRAPH '02 : Proceedings of the 29th annual Conference on Computer graphics and interactive techniques*, pages 681–682, New York, NY, USA, 2002. ACM Press.

[Per04] Ken Perlin. *GPU Gems 1 : Programming Techniques, Tips, and Tricks for Real-Time Graphics*, chapitre 5. Addison Wesley, 2004.

[PKZ04] Jianbo Peng, Daniel Kristjansson et Denis Zorin. « Interactive modeling of topologically complex geometric detail ». Dans *Proceedings of SIGGRAPH 2004*, pages 635–643. ACM, ACM Press / ACM SIGGRAPH, 2004.

[PNC05] Fabio Policarpo, Manuel M. Oliveira Neto et Joao Comba. « Real-Time Relief Mapping on Arbitrary Polygonal Surfaces ». Dans *Symposium on Interactive 3D Graphics and Games 2005*, pages 155–162, avril 2005.

[SBLD03] Frank Suykens, Karl Berge, Ares Lagae et Philip Dutré. « Interactive Rendering with Bidirectional Texture Functions. ». *Computer Graphics Forum*, volume 22, numéro 3, pages 463–472, 2003.

[Sch04] M. Schneider. « Real-Time BTF Rendering ». Dans *The 8th Central European Seminar on Computer Graphics*, pages 79–86, avril 2004.

[ST90] Peter Shirley et Allan Tuchman. « A polygonal approximation to direct scalar volume rendering ». Dans *Proceedings of Workshop on Volume Visualization 1990*, pages 63–70. ACM, 1990.

[Ups90] Steve Upstill. *The RenderMan Companion : A Programmer's Guide to Realistic Computer Graphics*. Addison-Wesley Professional, 1990.

[VT04] M. Alex O. Vasilescu et Demetri Terzopoulos. « TensorTextures : multilinear image-based rendering ». *ACM Trans. Graph.*, volume 23, numéro 3, pages 336–342, 2004.

[Wel04] Terry Welsh. « Parallax Mapping with Offset Limiting : A PerPixel Approximation of Uneven Surfaces ». Infiscape Corporation, 2004.

[WKE02] Manfred Weiler, Martin Kraus et Thomas Ertl. « Hardware-based view-independent cell projection ». Dans *VVS '02 : Proceedings of the 2002 IEEE symposium on Volume Visualization and Graphics*, pages 13–22, Piscataway, NJ, USA, 2002. IEEE Press.

[WMFC02] Brian Wylie, Kenneth Moreland, Lee Ann Fisk et Patricia Crossno. « Tetrahedral Projection using Vertex Shaders ». Dans *Proceedings of IEEE Volume Visualization and Graphics Symposium 2002*, pages 7–12, 2002.

[WNDS03] Mason Woo, Jackie Neider, Tom Davis et Dave Shreiner. *OpenGL Programming Guide : the Official Guide to Learning OpenGL, Version 1.4*. Addison-Wesley, 2003.

[WTL+04] Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo et Heung-Yeung Shum. « Generalized Displacement Maps ». Dans H. W. Jensen et A. Keller, éditeurs. *Rendering Techniques '04 (Eurographics Symposium on Rendering)*, juin 2004.

[WWT+03] Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo et Heung-Yeung Shum. « View-dependent displacement mapping ». Dans *Proceedings of SIGGRAPH 2003*, pages 334–339. ACM, ACM Press / ACM SIGGRAPH, 2003.

[YJ04] Keith Yerex et Martin Jagersand. « Displacement Mapping with Raycasting in Hardware ». SIGGRAPH 2004 Sketch, 2004.